ITERATIVE LDPC CDMA RECEIVER WITH EM ESTIMATION

A Thesis by

Avinash Mathur

B.E., Pune University, 2004

Submitted to the Department of Electrical and Computer Engineering
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

July 2007
© Copyright 2007 by Avinash Mathur
All Rights Reserved
ITERATIVE LDPC CDMA RECEIVER WITH EM ESTIMATION

I have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Electrical Engineering.

__________________________________
Hyuck M. Kwon, Committee Chair

We have read this thesis and recommend its acceptance:

__________________________________
John Watkins Committee Member

__________________________________
Nancy Bereman, Committee Member
DEDICATION

To my family and friends

You have made this possible
ACKNOWLEDGEMENTS

I would like to thank the faculty and staff of the Department of Electrical and Computer Engineering at Wichita State University. I would also like to thank my fellow classmates for the help and encouragement they have given me, and Dr. Kwon M. Hyuck for the knowledge and guidance he has given throughout the process. Finally I would like to thank my friends and family, who have been my greatest support and encouragement.
This thesis proposed a scheme of obtaining an estimate of channel coefficient and noise power spectral density (PSD), using iterative expectation-maximization (EM) channel estimation, based on a low-density parity-check (LDPC) code-division multiple-access receiver. At the receiver, an initial estimate was obtained with the aid of pilot symbols. Pilot bits were distributed among subframes followed by spreading and binary phase-shift keying. Subsequent values of channel coefficient and noise PSD both were updated iteratively by the soft feedback from the LDPC decoder. The updated channel coefficient and noise PSD were iteratively passed to the LDPC decoder, which resulted in improved decoding accuracy. The algorithm was tested on both a single user for constant noise PSD and a more realistic multiuser environment for a time-varying interference-plus-noise PSD estimation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. SYSTEM MODEL</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Source Information</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Encoding and Modulation</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Channel Model</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Iterative Receiver Structure</td>
<td>7</td>
</tr>
<tr>
<td>3. ITERATIVE LDPC CDMA RECEIVER</td>
<td>10</td>
</tr>
<tr>
<td>3.1 EM Algorithm</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Initial Receiver Iteration</td>
<td>15</td>
</tr>
<tr>
<td>4. SIMULATION RESULTS</td>
<td>18</td>
</tr>
<tr>
<td>5. CONCLUSION</td>
<td>27</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>28</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>30</td>
</tr>
<tr>
<td>Computer Programs</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Transmitter</td>
<td>4</td>
</tr>
<tr>
<td>2.</td>
<td>BPSK and QPSK constellation</td>
<td>6</td>
</tr>
<tr>
<td>3.</td>
<td>Iterative receiver</td>
<td>7</td>
</tr>
<tr>
<td>4.</td>
<td>Change in estimated fade values versus actual fading value</td>
<td>19</td>
</tr>
<tr>
<td>5.</td>
<td>Comparison of different pilot overhead and number of subframes</td>
<td>20</td>
</tr>
<tr>
<td>6.</td>
<td>BER versus $E_b/N_0$ for BPSK with constant $N_0$ for correlated Jakes fading</td>
<td>21</td>
</tr>
<tr>
<td>7.</td>
<td>BER versus $E_b/N_0$ for BPSK with constant $N_0$ for uncorrelated Rayleigh fading</td>
<td>22</td>
</tr>
<tr>
<td>8.</td>
<td>BER versus $E_b/N_0$ for QPSK with constant $N_0$ for correlated Jakes fading</td>
<td>23</td>
</tr>
<tr>
<td>9.</td>
<td>BER versus $E_b/N_0$ for QPSK with constant $N_0$ for uncorrelated Rayleigh fading</td>
<td>23</td>
</tr>
<tr>
<td>10.</td>
<td>BER versus $E_b/I_0$ for BPSK with adaptive $I_0$ for uncorrelated Rayleigh fading</td>
<td>24</td>
</tr>
<tr>
<td>11.</td>
<td>BER versus $E_b/I_0$ for BPSK with non-adaptive $I_0$ for uncorrelated Rayleigh fading</td>
<td>25</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>BPSK</td>
<td>Binary Phase-Shift Keying</td>
<td></td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access</td>
<td></td>
</tr>
<tr>
<td>EM</td>
<td>Expectation Maximization</td>
<td></td>
</tr>
<tr>
<td>LDPC</td>
<td>Low Density Parity Check</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>Maximum a Posteriori</td>
<td></td>
</tr>
<tr>
<td>NFSK</td>
<td>Noncoherent Frequency-Shift Keying</td>
<td></td>
</tr>
<tr>
<td>PSD</td>
<td>Power Spectral Density</td>
<td></td>
</tr>
<tr>
<td>QPSK</td>
<td>Quadrature Phase-Shift Keying</td>
<td></td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-Noise Ratio</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS

$\alpha$  \hspace{1cm} Alpha
$\beta$  \hspace{1cm} Beta
$\gamma$  \hspace{1cm} Gamma
$\phi$  \hspace{1cm} Phi
$\theta$  \hspace{1cm} Theta
CHAPTER 1
INTRODUCTION

1.1 Conventional Method

Conventional receivers attempt to ascertain signal-to-noise ratio (SNR) by measuring received signal strength at the receiver front end and assuming a known constant noise power spectral density (PSD). This thesis attempted to eliminate the need for such measurements and assumptions by the use of pilot symbols and iterative expectation-maximization (EM) channel estimations. In practice, noise, including multiple access and partial-band jamming, PSD is time varying. Hence, the model estimating the time-varying PSD was expected to show significant gains over a conventional design of constant PSD.

Studies have already been done using this kind of iterative receiver concept in [1, 2], where both channel coefficient and noise PSD were iteratively estimated by feeding back the decoder soft outputs. Iterative EM estimation and turbo coding were studied assuming non-coherent frequency-shift keying modulation and demodulation [1]. An iterative EM estimation and turbo coding were also studied [2], but coherent quadrature phase-shift keying modulation and demodulation were considered. In general, turbo coding requires a long channel interleaver and a deinterleaver that can cause severe latency and large memory space.

1.2 Proposed Method

Here, the iterative receiver extended the concepts of Torrieri et al. [1, 2] further by employing low-density parity-check (LDPC) codes instead of turbo codes, because LDPC codes do not require any channel interleaving and deinterleaving [3]. LDPC codes have interleaving effects internally when their code words are long enough, compared to memory length in a channel. The proposed system was tested under a block-fading environment using either a pure
Rayleigh or a Jakes model, whereas the system used by Torrieri et al. [2] was tested under Jakes fading.

Further, the major objective of feeding back LDPC decoder soft outputs into the iterative EM-based channel estimator was to reduce the pilot overhead. This thesis also studied the pilot overload reduction. At first, only a constant PSD model [6] was tested for the simplicity of using the proposed noise PSD estimation scheme, and then it was extended for a more realistic time-varying PSD model [7].

Use of this new model will produce several practical advantages for the business sector compared with the current system. First, due to the estimation of fading and noise coefficients at the receiver end, the decoder can more accurately decode the received message and reduce the number of errors, thus giving the received message a closer representation of the sent message providing better quality assurance. Second, service providers spend a considerable amount of money on wireless channels. If a packet message can be decoded the first time with no error, it would not have to be re-sent multiple times. Thus, the given channel is free to send further packet messages, which produces a higher degree of channel usage and decreases latency. Third, currently expensive power amplifiers must be used at the back end of the transmitter to amplify the signal before transmitting. With the new model the same bit error rate (BER) can be obtained at a lower signal-to-noise ratio ($E_b/N_0$). This reduces the load on power amplifiers, and thus a cost savings can be obtained by replacing higher-grade amplifiers with less-expensive ones.
1.3 Overview

This thesis is organized as follows: Chapter 2 describes the system model including the channel. Chapter 3 presents the proposed EM-based iterative LDPC receiver, including code-division multiple-access (CDMA) spreading. Chapter 4 shows simulations results, and Chapter 5 draws conclusions.
CHAPTER 2
SYSTEM MODEL

2.1 Source Information

Figure 1 shows a block diagram of a transmitter consisting of an LDPC encoder, spreading, and modulation. Both binary phase-shift keying (BPSK) modulation and quadrature phase-shift keying (QPSK) modulation were considered separately. User (A) was the desired user and User (B) was the interference. Input to an encoder in Figure 1 was a binary independent, identically distributed data source from a user. A block of length \( m \), i.e., \( \mathbf{v}^{(A)} = [v_1^{(A)}, ..., v_m^{(A)}] \), was taken by the User (A)’s encoder. Similarly, a block of length \( m \), \( \mathbf{v}^{(B)} = [v_1^{(B)}, ..., v_m^{(B)}] \) with \( v_m^{(B)} \in [0,1] \), was taken by the User (B)’s encoder. There were only two states for the users’ transmission activities, i.e., “ON” and “OFF.” In the first test system, User (A) was always in the “ON” state and User (B) was always turned “OFF.” In the second test system, User (A) was always in the “ON” state, whereas User (B) changed its state periodically with a duty cycle of 10 percent.

Figure 1. Transmitter.
2.2 Encoding and Modulation

Each \((1 \times m)\) message vector \(\mathbf{v}^{(A)}\) and \(\mathbf{v}^{(B)}\) if present was encoded into a \((1 \times n)\) codeword \(\mathbf{u}^{(A)} = [u_1^{(A)}, ..., u_n^{(A)}]\) and \(\mathbf{u}^{(B)} = [u_1^{(B)}, ..., u_n^{(B)}]\) with \(u_i^{(A)} \in [1, 0]\) and \(u_i^{(B)} \in [1, 0]\) using a regular LDPC code whose generator and parity-check matrices were denoted as a \((m \times n)\) systematic matrix \(G\) and \(((n-m) \times n)\) matrix \(H\), respectively [3]. The code words for both users were generated as

\[
\mathbf{u}^{(A)} = \mathbf{v}^{(A)}G \\
\mathbf{u}^{(B)} = \mathbf{v}^{(B)}G
\]  

(1)

The \(m\) and \(n\) were chosen to be 1,000 and 2,000, respectively, for the simulation. The Hamming weights of the columns and rows in \(H\) were set equal to 3 and 6, respectively, to consider a regular LDPC coding. The encoded symbols were spread with a spreading factor of \(g = 8\) (chips/code symbol). Each chip in a spread signal vector \(x\) was then mapped into a modulation symbol. Each modulation mapped \(m\)-encoded bits into a modulation symbol. In this paper both BPSK and QPSK modulation was carried out. For example, BPSK mapped \(m = 1\) encoded bits and, QPSK mapped \(m = 2\) encoded bits into a modulation symbol. Both BPSK and QPSK constellations are shown in Figure 2 [8]. The modulated vectors were \(\mathbf{x}^{(A)}\) and \(\mathbf{x}^{(B)}\). Spreading was used to mitigate the effects of any interfering signals. Interfering signals were not considered in the first simulation but were considered in the later part of this thesis.

The size of a subframe, which was set equal to 40 code symbols for the simulation was denoted by \(n_{SF}\). Each codeword of \(n = 2,000\) code symbols is split into \(n/n_{SF} = 50\) subframes. The subframe size was assumed to be smaller than a fading block size over which a fading coefficient was constant, and \(n_p = 4\) pilot symbols were multiplexed in the middle position of each subframe for the initial channel estimation. The overhead due to using the pilot symbols
was only 9 percent in this thesis, whereas it is typically about 16 percent or higher in a conventional system such as the Third Generation Partnership Project (3GPP).

![BPSK and QPSK constellations](image)

Figure 2. BPSK and QPSK constellations.

### 2.3 Channel Model

The channel was modeled as a complex additive white Gaussian noise (AWGN) channel with PSD of $N_0/2$ W/Hz per dimension. Each subframe of $n_{SF} + n_p$ symbols was multiplied by a uncorrelated or correlated fading coefficient that was generated in every fading block of $n_{FB}$ symbols from either a pure Rayleigh or a Jakes fading model [4]. In the simulation, $n_{FB} = n_{SF} + n_p = 44$. Other combinations can be considered. When a Jakes fading coefficient was used, it corresponded to a mobile velocity equal to 300 km/hr and carrier frequency equal to 2 GHz. The data rate $R_b$ was set to 100 kbps.
2.4 Iterative Receiver Structure

Figure 3 shows a block diagram of the proposed iterative receiver. The received signal was downconverted and passed through a chip-matched filter.

\[ y_k = C_{(\gamma_k/\gamma_{max})}^{(d)} x_k^{(d)} + C_{(\gamma_k/\gamma_{max})}^{(b)} x_k^{(b)} + n_k, \quad 1 \leq k \leq ng \]  \hspace{1cm} (2)

where superscripts (A) and (B) denote Users (A) and (B), respectively

\[ y_k = \text{received data at the } k^{\text{th}} \text{ chip in a code word}, \quad C_{(\gamma_k/\gamma_{max})}^{(d)} = \sqrt{E_s} \alpha_{(k/\gamma_{max})}^{(d)} e^{i\phi_{(k/\gamma_{max})}} \text{ and} \]

\[ C_{(\gamma_k/\gamma_{max})}^{(b)} = \sqrt{E_s} \alpha_{(k/\gamma_{max})}^{(b)} e^{i\phi_{(k/\gamma_{max})}} \] are independent complex block fading coefficients for User (A) and (B), respectively, whose magnitudes and phases were constant over the \( k/(n_{chip}g) \)-th fading block

\( \lfloor \gamma \rfloor \) was the largest integer less than or equal to \( \gamma \)

\( x_k^{(d)} = p_k^{(d)} u_k^{(d)} \) and \( x_k^{(b)} = p_k^{(b)} u_k^{(b)} \) were code symbols for User (A) and (B), respectively at the \( k^{\text{th}} \) chip index
\[ \beta = \lceil (k-1)/g \rceil + 1, \beta = 1, \ldots, n \]

\[ p_k = \pm 1, \text{the } k\text{-th chip of spreading sequence} \]

\[ u_\beta = \pm 1, \text{the } \beta\text{-th code symbol} \]

\[ n_k = \text{a complex AWGN sample at the } k\text{-th chip with } E[|n_k|^2] = N_0 \]

\[ k = \text{chip index in a code word} \]

\[ n = \text{the number of code symbols in a code word} \]

\[ g = \text{spreading factor or number of chips per code symbol} \]

\[ n_{FB} = \text{the number of code symbols in a fading block over which the fading coefficient was constant} \]

\[ E_s = \text{average code-symbol energy} \]

\[ \alpha = \text{a complex fading amplitude, which was constant over } n_{FB} \text{ code symbols and had } E[|\alpha|^2] = 1 \]

\[ \phi = \text{unknown fading phase which was constant over } n_{FB} \text{ code symbols} \]

Three iteration indices were used throughout this thesis:

- \( i \) denoted the index for the internal \textit{EM iteration} in the channel estimator of Figure 2, \( i = 1, \ldots, i_{\text{max}} \) where \( i_{\text{max}} \) was set to 10, which was found to be sufficient for any receiver iteration \( j \) through simulation.

- \( j \) denoted the index for the closed-loop \textit{receiver iteration} in Figure 2.

- \( l \) denoted the index for the internal \textit{LDPC decoding iteration} in Figure 2, \( l = 1, \ldots, l_{\text{max}}, \) where \( l_{\text{max}} \) was set to 20, which was found to be sufficient for any receiver iteration \( j \) through simulation.
Hence, a receiver iteration was defined as $l_{\text{max}} = 20$ decoder iterations followed by $i_{\text{max}} = 10$ EM internal iterations. In the simulation experiments, $j = 0, 1, \ldots, 9$ receiver iterations were tested because the improvement for $j$ beyond 9 was minimal.

Let $\hat{\theta}^{(i)} = \left( \hat{\mathbf{C}}^{(i)}, \hat{\mathbf{N}}^{(i)} \right)$ represent the estimates of the channel coefficient and noise PSD parameters at the $i$th EM internal iteration for the $j$th closed-loop receiver. No EM iterations were performed for the initial estimate. The initial estimate $\hat{\theta}^{\text{initial}} = \left( \hat{\mathbf{C}}^{\text{initial}}, \hat{\mathbf{N}}^{\text{initial}} \right)$ was obtained while the switch in Figure 3 was set to position 1 by taking the average of the received pilot intervals every subframe after multiplying by the known pilot symbols. The subsequent receiver iterations were performed while the switch in Figure 3 was set to position 2 in order to refine the initial estimates with the aid of soft feedback from the LDPC decoder. The refined estimate was further used to improve LDPC outputs, and so on, as shown in Figure 3.
CHAPTER 3
ITERATIVE LDPC CDMA RECEIVER

3.1 EM Algorithm

Theoretically, ML estimation $\hat{\theta}$ can be obtained using a received data vector $y = [y_1, y_2, ..., y_m]$ referred to as incomplete data, by maximizing the conditional log-likelihood function as

$$\hat{\theta} = \arg\max_{\theta} \ln f(y | \theta)$$ \hspace{1cm} (3)

However, the computation of this equation is almost prohibitive in practice. Hence, the expectation of the conditional log-likelihood of $z = (y, x)$, known as complete data, i.e.,

$$E_{\phi, \theta}[\ln f(z | \theta)],$$

was iteratively maximized with respect to $\theta$, where expectation was taken with respect to $x$ given $y$ and $\theta$. This is why the algorithm was called an expectation maximization (EM) algorithm.

The conditional probability density function (p.d.f.) of $z$ can be written as

$$f(z | \theta) = f(x, y | \theta) = f(y | x, \theta) f(x | \theta)$$ \hspace{1cm} (4)

$$f(z | \theta) = f(y | x, \theta) f(x)$$ \hspace{1cm} (5)

where the last equality is from the independence between the transmitted signal vector $x$ and parameter $\theta$. Taking the log on both sides leads to

$$\ln f(z | \theta) = \ln f(y | x, \theta) + \ln f(x)$$ \hspace{1cm} (6)

Using the chip independence and the probability density function (pdf) of AWGN noise, the conditional pdf $f(y | x, \theta)$ can be written as
\[
f(y | \theta, x) = \frac{1}{(\pi I_o)^{n}} \cdot \exp \left(-\frac{\sum_{k=1}^{m} \left|y_k - Cx_k\right|^2}{I_o}\right) \tag{7}
\]

The first test case produced \(N_0\) instead of \(I_0\) as the interference signal, i.e., User(B) was “OFF.” On further simplifying we get

\[
\ln f(y | \theta, x) = -ng \ln(\pi I_o) - \frac{1}{I_o} \sum_{k=1}^{m} \left[|y_k|^2 + |Cx_k|^2 - 2 \Re(y_k^* Cx_k)\right] \tag{8}
\]

where the superscript * denotes the conjugate. Since the second term in the right-hand side of equation (5) was unrelated to the maximization with respect to \(\theta\), equation (8) was used to estimate the parameter as

\[
\hat{\theta} = \arg \max_{\theta} E_{x,y,\theta} \left[\ln f(y | x, \theta)\right] \tag{9}
\]

Hence, the EM algorithm can be summarized in the following four steps:

**Step 1**: Find initial estimate: \(\hat{\theta}^{(0)}\) using the pilot symbols.

**Step 2** (E-Step): Compute an objective function, i.e., a conditional expectation of \(\ln f(z | \theta)\) for a given parameter \(\theta\) using a conditional density \(f(x | y, \hat{\theta}^{(0)}_{(i)})\). In view of equation (9), the objective function can be taken as

\[
\chi(\theta | \hat{\theta}^{(i)}_{(i)}) = E_{x,y,\theta} \left[\ln f(y | x, \theta)\right] = \int \ln f(y | x, \theta) f(x | y, \hat{\theta}^{(i)}_{(i)}) dx \tag{10}
\]

Substituting equation (8) into equation (10) leads to

\[
\chi(\theta | \hat{\theta}^{(i)}_{(i)}) = -ng \ln(\pi I_o) - \frac{1}{I_o} \sum_{k=1}^{m} \left[|y_k|^2 + |C|^2\right] + \frac{2}{I_o} \sum_{k=1}^{m} \left|\text{Re}(y_k^* C\hat{\theta}^{(i)}_{(i)})\right| \tag{11}
\]
where

\[ I_0 = \begin{cases} 0 & \text{if User (B) is OFF} \\ N_0 + E_s \cdot T_c / T_s & \text{if User (B) is ON} \end{cases} \] (12)

\( T_c = \) chip interval, and \( T_s = \) code symbol interval.

\[
\bar{u}_{\mu, j}^{(i)} = E_{y_j, \hat{y}_j^{(i)}} \left[u_{\mu} \right] = E_{y_j, \hat{y}_j^{(i)}} \left[u_{\mu} \right] = E_{y_j, \hat{y}_j^{(i)}} \left[u_{\mu} \right]
\] (13)

\[
y_{\mu} = \left(y_{\beta, k+1}, \ldots, y_{\beta, K} \right)^{T}
\] (14)

In equations (11)-(14)

\[
\chi(\theta | \hat{\theta}_{(j)}^{(i)}) = \int \ln f(y | x, \theta) \prod_{\beta=1}^{K} f(x_{\beta} | y, \hat{\theta}_{(j)}^{(i)}) dx
\] (15)

**Step 3 (M-step):** Obtain optimum parameter for the \((i+1)\)st EM iteration by maximizing the conditional expectation in equations (10)-(11) as

\[
\hat{\theta}_{(j)}^{(i+1)} = \arg \max_{\theta} \chi(\theta | \hat{\theta}_{(j)}^{(i)})
\] (16)

Taking the derivatives of the objective function in equation (11) with respect to the real and imaginary parts of the parameter \(C\), and then setting the results equal to zero yields the estimate of the channel coefficient

\[
\hat{C}_{(j)}^{(i+1)} = \frac{1}{n} \sum_{k=1}^{n} \frac{y_{k}^{*} p_{s} \bar{u}_{\mu, j}^{(i)}}{g}
\] (17)

where

\( \hat{C}_{(j)}^{(i+1)} = \) User (A)’s fading channel coefficient estimate at the \((i+1)\)-st EM and \(j\)-th receiver iteration, and

\( \bar{u}_{\mu, j}^{(i)} = \) expectation of User (A)’s \(\beta\)-th code symbol at the \(i\)-th EM and \(j\)-th receiver iteration.
Even in the second test setup where User (B)’s signal causes interference, it was not necessary to estimate the fading channel coefficients because interference cancellation was not required in the proposed system. From the derivative of the objective function with respect to parameter \( I_0 \) and then setting the result equal to zero, the interference plus noise PSD was estimated as

\[
\hat{I}_{s(i)}^{(i)} = \frac{1}{n g} \sum_{i=1}^{n} \left[ |y_i|^2 + |\hat{C}_{(i)}^{(i)}|^2 - 2 \sum_{k=1}^{n} \text{Re}(y^* P \hat{C}_{(i)}^{(i)} y^* C_{(i)}^{(i)}) \right]
\]

(18)

where

\[
I_0 = \begin{cases} 
N_0 & \text{if User (B) is OFF} \\
N_0 + E_s T_s / T_s & \text{if User (B) is ON}
\end{cases}
\]

(19)

**Step 4:** If either \( \| \hat{\theta}_{(j)}^{(i)} - \hat{\theta}_{(j)}^{(i)} \| < \varepsilon \) or \( i = i_{\text{max}} \), stop.

The channel estimate \( \hat{\theta}_{(j)}^{(i)} = \left( \hat{C}_{(i)}^{(i)} \right) \) was fed into the LDPC decoder for the \( j \)-th receiver iteration. Then, the LDPC decoder with this channel estimate decoded the demodulated code symbols iteratively up to \( l_{\text{max}} = 20 \) times for each receiver iteration \( j \) using an MAP rule. If the syndrome vector was zero before reaching the maximum number of decoding iterations \( l_{\text{max}} \), then the LDPC decoding stopped for a given receiver iteration \( j \). The probability values of \( \text{Pr} \left[ u_\beta = -1 \mid y \right] \) were stored for each code symbol \( \beta \) at the final LDPC decoding iteration and fed back into the EM channel estimator.

For BPSK modulation, the expectation of \( u_\beta, \beta = 1, ..., n \) at the \( i \)-th EM and \( j \)-th receiver iteration was found from equation (13) as

\[
\bar{u}_{\beta(i)}^{(i)} = \text{Pr}(u_\beta = 1 \mid \hat{\theta}_{(j)}^{(i)}) f(y \mid u_\beta = 1, \hat{\theta}_{(j)}^{(i)}) f(y \mid \hat{\theta}_{(j)}^{(i)})
\]

(20)

\[
- \text{Pr}(u_\beta = -1 \mid \hat{\theta}_{(j)}^{(i)}) f(y \mid u_\beta = -1, \hat{\theta}_{(j)}^{(i)}) f(y \mid \hat{\theta}_{(j)}^{(i)})
\]

For notation, let
\[ s_{\beta(j)} = \Pr(u_\beta = -1| \hat{\theta}_{(j)}) \] (21)

The \( s_{\beta(j)} \) values were the LDPC decoder outputs and were fed back to the EM parameter estimator where the LDPC decoder took the previous parameter estimation \( \hat{\theta}_{(j)} = \left( \hat{C}_{(j)}, \hat{I}_{(j)} \right) \) to generate \( s_{\beta(j)} \). Then, the \( \bar{u}_{\beta(j)}^{(i)} \) at the \( i \)-th EM and \( j \)-th receiver iteration in equation (20) can be rewritten as

\[
\bar{u}_{\beta(j)}^{(i)} = \frac{1 - s_{\beta(j)} - s_{\beta(j)}}{1 - s_{\beta(j)} + s_{\beta(j)}} \frac{f(y_\beta | u_\beta = -1, \hat{\theta}_{(j)}^{(i)})}{f(y_\beta | u_\beta = +1, \hat{\theta}_{(j)}^{(i)})}
\] (22)

Define parameter \( R_{\beta(j)}^{(i)} \) as

\[
R_{\beta(j)}^{(i)} = \frac{f(y_\beta | u_\beta = -1, \hat{\theta}_{(j)}^{(i)})}{f(y_\beta | u_\beta = +1, \hat{\theta}_{(j)}^{(i)})}
\] (23)

Using equation (7) and the estimates gives

\[
R_{\beta(j)}^{(i)} = \exp \left[ \sum_{k \neq \beta} \frac{4 p_k \text{Re} \left( y_k \hat{C}_{(j)}^{(i)} \right)}{I_{\beta(j)}^{(j)}} \right]
\] (24)

Therefore, \( \bar{u}_{\beta(j)}^{(i)} \) can be computed as

\[
\bar{u}_{\beta(j)}^{(i)} = \frac{1 - s_{\beta(j)} - s_{\beta(j)} R_{\beta(j)}^{(i)}}{1 - s_{\beta(j)} + s_{\beta(j)} R_{\beta(j)}^{(i)}}
\] (25)

For QPSK modulation, the following symbol probabilities for \( u_\beta, \beta = 1, ..., 2n \) were defined as

\[
s_1 = \Pr(u_\beta = 1), \ s_2 = \Pr(u_\beta = j), \ s_3 = \Pr(u_\beta = -1), \ s_4 = \Pr(u_\beta = -j)
\] (26)

For QPSK modulation, the expectation of \( u_\beta \) at the \( i \)-th EM and \( j \)-th receiver iteration was found from equation (13) as
\[ \bar{u}_\beta = \frac{s_i R_{i,j} + j s_2 R_{2,j} - s_3 R_{3,j} - j s_4 R_{4,j}}{\sum_{n=1}^{4} s_n R_{n,j}} \]  

(27)

Parameter \( R_{(i,j)}^{(\beta)} \) was defined as

\[
R_{1,j} = \exp \left( \frac{2 \hat{C}_i}{\hat{N}_{0,j}} y_{kR} \right), \quad R_{2,j} = \exp \left( \frac{2 \hat{C}_i}{\hat{N}_{0,j}} y_{kI} \right)
\]

\[
R_{3,j} = \exp \left( -\frac{2 \hat{C}_i}{\hat{N}_{0,j}} y_{kR} \right), \quad R_{4,j} = \exp \left( -\frac{2 \hat{C}_i}{\hat{N}_{0,j}} y_{kI} \right)
\]

(28)

where \( y_{kR} \) is the real part of the \( k \)th-received symbol after \( \pi/4 \) rotation and \( y_{kI} \) is the imaginary part of the \( k \)th-received symbol \( \pi/4 \) rotation.

For a given receiver iteration \( j \), \( \bar{u}_{(j)}^{(\beta)} \) and \( R_{(j)}^{(\beta)} \) were updated \( i_{\text{max}} \) times using soft values of \( s_{(j)}^{(\beta)} \), which were sent from the LDPC decoder. When a pilot symbol was processed, \( s_{(j)}^{\beta} \) was set equal to zero for that symbol in the channel estimator. Now the receiver iteration increased to \( j+1 \). Using \( \bar{u}_{(j)}^{(i)} \) and \( R_{(j)}^{(i)} \) from the previous receiver iteration and using equations (17) and (18), \( \hat{C}_{(j+1)}^{(i)} \) and \( \hat{I}_{(j+1)}^{(i)} \) were calculated, which in turn were passed to equations (24) and (25) for BPSK and equations (27) and (28) for QPSK to provide better values of \( R_{(j+1)}^{(i)} \) and \( \bar{u}_{(j+1)}^{(i)} \). The improved values of \( \bar{u}_{(j+1)}^{(i)} \) and \( R_{(j+1)}^{(i)} \) were again used to calculate refined \( \hat{C}_{(j+1)}^{(i)} \) and \( \hat{I}_{(j+1)}^{(i)} \), \( i = 1, \ldots, i_{\text{max}} \), using equations (17) and (18). This loop was executed until it reached \( i = i_{\text{max}} \), giving \( \hat{C}_{(j+1)}^{(i_{\text{max}})} \) and \( \hat{I}_{(j+1)}^{(i_{\text{max}})} \). The change in \( \hat{C}_{(j+1)}^{(i)} \) can be observed in Figure 4. After reaching \( i_{\text{max}} \), the \( \hat{\theta}_{(j+1)}^{(\text{max})} \) values were passed to the LDPC for further decoding.

### 3.2 Initial Receiver Iteration

The receiver’s knowledge of the pilot-symbol information was exploited to obtain the initial estimates of the channel parameters. For convenience, \( n_p \) pilot symbols were assumed to
be 1’s. Because the transmitted bits were known to be 1, this gives the value of \( s_{\beta} \) to be zero.

Hence, substituting this into equations (25) or (27) gives the \( \hat{u}_{\beta} \) value to be 1, and the initial channel coefficient estimate can be written from equation (17) as

\[
\hat{C}_{(\lambda_{0})}^{(\lambda_{0})} = \frac{1}{n_{\beta}} \sum_{k=1}^{n_{\beta}} y_{k}^{\dagger} p_{k} \hat{g}
\]  

(29)

Similarly, from equation (18),

\[
\hat{i}_{\lambda_{0}}^{(\lambda_{0})} = \frac{1}{n_{\beta}} \sum_{k=1}^{n_{\beta}} |y_{k}|^{2} + |\hat{C}_{(\lambda_{0})}^{(\lambda_{0})}|^{2} - 2 \sum_{k=1}^{n_{\beta}} \text{Re}\left( y_{k}^{\dagger} p_{k} \hat{C}_{(\lambda_{0})}^{(\lambda_{0})} \right)
\]  

(30)

where

\[
I_{\lambda_{0}} = \begin{cases} 
N_{0} & \text{if User (B) is OFF} \\
N_{0} + E_{s} \cdot T_{s} / T_{c} & \text{if User (B) is ON} 
\end{cases}
\]  

(31)

**Channel State Information for LDPC Decoder**

After coherent demodulation and dispreading was done, the estimated fading coefficient’s phase was compensated, producing the complex envelop of the received signal as

\[
r_{\beta} = |C_{\beta}| u_{\beta} + n_{\beta}
\]  

(32)

where \( E\left[|n_{\beta}|^{2}\right] = I_{0} \)

The BPSK input to the LDPC decoder was

\[
\Lambda_{\beta} = \ln \left[ \frac{p\left(r_{\beta}|u_{\beta} = "1" \text{ bit}\right)}{p\left(r_{\beta}|u_{\beta} = "0" \text{ bit}\right)} \right] = \frac{4\text{Re}\left(r_{\beta}\right)}{I_{0,\lambda_{0}}^{(\lambda_{0})}} \hat{C}_{\beta,\lambda_{0}}^{(\lambda_{0})} \]  

(33)

Equation (33) is equivalent to equation (1-137) of from Torrieri’s work [5].

The QPSK constellation, after rotation by 45°, revealed that the odd code bits were determined by the real symbol point and similar principle applies for the even code bits.
Therefore, the input to the LDPC decoder was the following log-likelihood ratios (LLR) for the odd and even code bits $b_l$ carried by the $k^{th}$ symbol, $x_k$ respectively,

$$\Lambda(b_{l,i}) = \log \frac{\sum_{j=1}^{z-1} \exp \left\{ -\frac{1}{2\sigma_x^2} \left( y_{il} - a_{i,j} \right)^2 \right\}}{\sum_{j=1}^{z-1} \exp \left\{ -\frac{1}{2\sigma_x^2} \left( y_{il} - a_{i,j} \right)^2 \right\}}$$

$$\Lambda(b_{e,i}) = \log \frac{\sum_{j=1}^{z-1} \exp \left\{ -\frac{1}{2\sigma_y^2} \left( y_{il} - d_{i,j} \right)^2 \right\}}{\sum_{j=1}^{z-1} \exp \left\{ -\frac{1}{2\sigma_y^2} \left( y_{il} - d_{i,j} \right)^2 \right\}}$$

(34)

where $a_{i,j}$ and $d_{i,j}$ represent constellation points along the x-axis and y-axis, respectively.

When perfect channel state information was available, $\hat{C}_{\beta,ij}$ and $\hat{I}_{\nu,ij}$ were replaced with $C_\beta$ and $I_\theta$. 

17
This chapter discusses the performance of the proposed EM algorithm over both the single-user constant PSD model and the multiuser time-varying interference plus noise PSD model. In the first half, i.e., Figures 4, 5, 6, 7, 8, and 9, the single-user channel environment was considered, and in the second half, i.e., in Figures 10 and 11, a more realistic multiuser environment was tested. Single user channel environment was tested for both BPSK modulation with correlated Jakes and uncorrelated Rayleigh fading models, as shown in Figures 6 and 7, and QPSK modulation with correlated Jakes and uncorrelated Rayleigh fading models, as shown in Figures 8 and 9. The multiuser environment was tested under BPSK modulation with the uncorrelated Rayleigh fading model, as shown in Figures 10 and 11.

As discussed in the iterative receiver structure section, three iterations were considered throughout the simulation. Figure 4 shows the transient behavior of estimated fading values over $i = 1, \ldots, 10$ EM iterations. The initial receiver iteration, i.e., $j = 0$, was done with the aid of pilots, therefore, channel estimates $\hat{\theta}_{(j=0)} = \left(\hat{C}_{(j=0)}, \hat{N}_{0(j=0)}\right)$ were not updated using EM iterations but were obtained by simple averages in equations (29) and (30). This estimate was quite distant from the true fading value. Next, for the first receiver iteration, this value and the decoder feedback were used to generate new channel estimates. These new channel estimates were used to calculate improved values of $\bar{u}_{\beta, \lambda(j+1)}^{(i)}$ and $R_{\beta, \lambda(j+1)}^{(i)}$, which were again used to calculate refined channel estimates using equations (17) and (18). This loop was executed until it reached to $i = i_{\text{max}}$. The EM iterations during the $j = 1$ receiver iteration brought the channel estimate close to the actual fading value, thus improving the accuracy of the decoder input. Also, from Figure 4, it
is clear that it reached saturation point around 7 to 8 iterations, which indicates that more than 10 EM iterations were not useful.

Figure 4. Change in estimated fade values versus actual fading value for the $p^{th}$ subframe, where $p = 48$ at $E_b/N_0$ of zero dB (see the appendix for computer program number 1).

Figure 5 shows BER versus $E_b/N_0$ at receiver iterations $j = 9$ with the number of pilot symbols $n_p$ and the number of code symbols $n_{SF}$ per subframe as parameters. Block-fading coefficients were generated by a Jakes fading model with $f_D T_s$ equal to 0.005, where $f_D$ and $T_s$ are the Doppler frequency shift and the code-symbol duration, respectively. It was observed that the case of $(n_{SF} = 40, n_p = 4)$ whose pilot overhead is 9.1 percent, shows the best BER among the combinations considered at $E_b/N_0 = 6$ dB. This was due to enough $n_p$ giving a good initial channel estimate and the short size of the subframe providing LDPC gain. Even though the pilot overhead was 9.1 percent, because $n_p$ is too small, such as the $(n_{SF} = 20, n_p = 2)$ case, the quality of the initial channel estimate was bad and not helpful for the subsequent iterations. For similar reasons, using the pilot overhead of 4.8 percent resulted in less pilots per subframe such as the
The accuracy of channel parameter estimates is proportional to the ratio of the number of pilot symbols over the total number of bits in a codeword, but a higher overhead consumes bandwidth. Consequently, a proper pilot-to-data symbol ratio (say 9 percent for $n = 2,000$ in this report) was chosen to generate sufficiently good channel estimates in the initial and subsequent receiver iterations. The typical overhead ratio in a conventional system is about 20 percent, e.g., 16 percent in the 3GPP standard.

Figure 6 shows BER versus $E_b/N_0$ for codeword size $n = 2,000$ symbols and $j = 0, 1, 3,$ and 9 receiver iterations with 9 percent pilot symbol overhead. As discussed in Figure 5, among every 40 code symbols, four pilot symbols were multiplexed, and the fading block length was 44
code symbols. The performance improvement between receiver iteration 0 and 1 was about 0.5 dB at $10^{-3}$ BER. This gain was due to the use of four pilot bits per subframe of size 40 for the initial channel estimation, whereas the subsequent receiver iterations generated channel estimates derived from all 44 bits within each subframe leading to a more accurate estimate. The gain between first and ninth receiver iterations was 0.2 dB. The difference between perfect channel coefficients and the ninth receiver estimate was about 0.9 dB.

Figure 7 shows BER versus $E_b/N_0$ for $n = 2,000$ symbols and $j = 0, 1, 3$ and 9 receiver iterations with 9 percent pilot symbol overhead where block fading coefficients were generated assuming uncorrelated Rayleigh fading. The performance improvement between receiver iteration 0 and 1 was about 0.7 dB at $10^{-3}$ BER. The difference between perfect channel coefficients and the ninth receiver estimate was about 1.1 dB.

![Figure 6. BER versus $E_b/N_0$ for BPSK, for block length $n = 2,000$ bits and receiver iterations $j = 0, 1, 3$ and 9 with 9.1 percent pilot symbol overhead for correlated Jakes fading with normalized fade rate of 0.005 (see the appendix for computer program number 1).](image-url)
Figure 7. BER versus $E_b/N_0$ for BPSK, for block length $n = 2,000$ bits and receiver iterations $j = 0, 1, 3, \text{ and } 9$ with 9.1 percent pilot symbol overhead for uncorrelated Rayleigh fading (see the appendix for computer program number 2).

Figure 8 shows BER versus $E_b/N_0$ for codeword size $n = 2,000$ symbols and $j = 0, 1, 3, \text{ and } 9$ receiver iterations with 9 percent pilot symbol overhead. The performance improvement between receiver iteration 0 and 1 was about 0.5 dB at $10^{-3}$ BER. The gain between the first and ninth receiver iteration was 0.2 dB. The difference between perfect channel coefficients and the ninth receiver estimate was about 0.7 dB.

Figure 9 shows BER versus $E_b/N_0$ for $n = 2,000$ symbols and $j = 0, 1, \text{ and } 9$ receiver iterations with 9 percent pilot symbol overhead where block fading coefficients were generated assuming uncorrelated Rayleigh fading. The performance improvement between receiver iteration 0 and 1 was about 0.7 dB at $10^{-3}$ BER. The difference between perfect channel coefficients and the ninth receiver estimate was about 0.8 dB.
Figure 8. BER versus $E_b/N_0$ for QPSK, for block length $n = 2,000$ bits and receiver iterations $j = 0, 1, 3$ and $9$ with $9.1$ percent pilot symbol overhead for correlated Jakes fading with normalized fade rate of $0.005$ (see the appendix for computer program number 3).

Figure 9. BER versus $E_b/N_0$ for QPSK, for block length $n = 2,000$ bits and receiver iterations $j = 0, 1, 3$, and $9$ with $9.1$ percent pilot symbol overhead for uncorrelated Rayleigh fading (see the appendix for computer program number 3).
As seen from the plots, the uncorrelated fading channel for both BPSK and QPSK modulations, as shown in Figures 7 and 9, shows better performance than the correlated fading in Figures 6 and 8, because there was more channel diversity when the fading was uncorrelated.

Figure 10 shows BER versus $E_b/I_0$ at receiver iterations $j = 0, 1, 3, 5, 7, \text{ and } 9$ with adaptive estimation of fading coefficient and the interference plus noise PSD level. Block fading coefficients for User (A) and User (B) were generated by an independent Rayleigh block fading model. User (A) was always “ON”, whereas User (B) was turned “ON” and “OFF” periodically. The duty cycle of User (B) was 10 percent, i.e., in every ten frame, the nine frames were of User (A) affected by just noise, whereas in the tenth frame, User (B) data causes a interference with the noise. The performance improvement between receiver iteration 0 and 1 is about 1.3 dB at $10^{-3}$ BER. The gain between first and ninth receiver iteration was 1 dB. The difference between perfect channel coefficients and the ninth receiver estimate was about 1.3 dB.

Figure 10. BER versus $E_b/I_0$ for BPSK, for block length $n = 2,000$ code bits and receiver iterations $j = 0, 1, 3, 5, 7, \text{ and } 9$ with adaptive $I_0$ and fading coefficient estimation for independent Rayleigh block fading (see the appendix for computer program number 4).
Figure 11 shows BER versus $E_b/I_0$ under the same environment as shown in Figure 9 for $j = 0, 1, 3, 5, 7,$ and $9$ receiver iterations with only estimating fading coefficients, keeping interference plus noise PSD $I_0$ constant, i.e., non-adaptive. The performance improvement between non-adaptive $I_0$ PSD scheme and adaptive $I_0$ estimation scheme for $j = 9$ was significant such as $6.5$ dB at $10^{-3}$ BER. The difference between perfect channel coefficients and the ninth receiver estimate was about $7.3$ dB.

![Figure 11. BER versus $E_b/I_0$ for BPSK, for block length $n = 2,000$ code bits and receiver iterations $j = 0, 1, 3, 5, 7,$ and $9$ with non-adaptive $I_0$ and fading coefficient estimation for independent Rayleigh block fading (see the appendix for computer program number 5).](image)

Obviously, performance of the adaptive scheme, shown in Figure 10, was better than that of the non-adaptive scheme, shown Figure 11. This is because when User (B) was “ON,” it causes interference to User (A) data, and the User (A) receiver can estimate this $I_0$ level change correctly. In the initial receiver iterations, both the channel fading coefficients and interference...
PSD coefficients were estimated. This aids the decoder to decode more efficiently at the initial receiver iteration, thus giving a lower BER at the initial iteration, compared to conventional schemes where interference PSD is assumed to be constant over all time intervals, irrespective of change of interference PSD. Thus, at the initial iteration, there was significant gain of about 7dB in the decoder output. On further receiver iterations additional gain was achieved.
CHAPTER 5
CONCLUSIONS

This paper presented a refined iterative receiver with EM channel estimation and LDPC decoding. Conventional values of pilot-to-data symbol ratios are about 20 percent to have reasonably acceptable performance under a Rayleigh fading environment, compared with the perfect channel knowledge case. In contrast to that, the proposed receiver provided reasonably accurate performance with only a 9 percent ratio, thus greatly increasing the spectral efficiency. Also, the use of LDPC codes instead of turbo codes decreased the latency and need of large memory space, because LDPC codes have internal interleaving effects and do not require any external channel interleaving and deinterleaving.

The iterative receiver was also tested under the time-varying interference-plus-noise PSD level $I_0$, assuming two active users, where the desired user was always “ON” and the other user was “ON” and “OFF” periodically with a fixed duty cycle. The proposed adaptive PSD estimation scheme with fading coefficient estimation showed a significant gain over the non-adaptive scheme of constant PSD equal to $N_0$, which typically has been assumed in conventional receivers. Therefore, since the proposed receiver showed that it can adapt to a time-varying number of active users it means better performance than the conventional schemes.
REFERENCES
LIST OF REFERENCES


APPENDIX
COMPUTER PROGRAMS USED FOR SIMULATION

PROGRAM 1. EM ALGORITHM FOR BPSK MODULATION AND JAKES FADING

clc;
clear all;
frame_index=1000;
rows=1000;
cols=2000;
q=2;
H1=ldpc_generate(rows,cols,3,2,123);  

[H,G] = ldpc_h2g(H1,q);
G_Row = size(G,1); % K : size of information bits
G_Col = size(G,2);
ii=0;
err=[];
NumPilot=100; %%%%%%%%%Total number of pilot bits
max_iter=10; %%%%%%%%%%maximum receiver iterations

for EsN0dB =-3:2:25 %dB
    ii=ii+1;
    Es=1;
esn0dB(ii)=EsN0dB;
    EsN0(ii) = 10^(EsN0dB/10); % EbNo
    R=G_Row/G_Col;  
    Eb=Es/R;  
    Ecod=R*Eb;
    M=2;  
    E_sf=Es/sqrt(SF);
    N0(ii)=Es/EsN0(ii); % SF*1/Rate *10^(-EbN0dB/10)
    noise_sigma(ii) = sqrt( N0(ii)/(2));
    EbNo(ii)=Eb/N0(ii);
    ebn0dB(ii)=10*log10(EbNo(ii));
end

Rb=100e3;  
Tb=1/Rb;
error1=zeros(1,length(esn0dB));
error2=zeros(max_iter,length(esn0dB));
N0_table(1,:)=N0;

fb_num=1;%%counter for fade block number
for kk = 1:length(esn0dB)
    %%%Transmitter Starts Here
    %%%
    for ff=1:1:frame_index
        x=floor(rand(1,G_Row)*2);%%generates stream of message bits containing zeros and ones
        mG = mod(x*G,2);%%encoding the message bits in code symbols
        slots=20;%%number of divisions of each frame
        L=G_Col/slots;%%length of each slot

        division of frame into slots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        for i=1:1:slots
            xldpc(i,:)=mG((i-1)*L+1:i*L);
        end

        adding pilots at midamble position in each slot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        pilots=NumPilot/slots;
        for num=1:1:slots
            for i=1:1:(L/2+pilots);
                if i<=L/2
                    mG_plt(num,i)=xldpc(num,i);
                else
                    mG_plt(num,i)=1;
                end
            end
            for i=(L/2+1):1:L
                mG_plt(num,i+pilots)=xldpc(num,i);
            end
        end

        concatenating the slots in one row %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        mG_pilot=[];
        for i=1:1:slots
            MG_PLT=mG_plt(i,:);
            mG_pilot=[mG_pilot MG_PLT];
        end

        Tx_data=2*mG_pilot-1;
        codes=hadamard(SF);
        Sp_Code=codes(1,:);
        %%% Spreading Sequence Generation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:length(Tx_data)
    Spread.Tx_data(:,i) = Tx_data(i)*Sp_Code;
end

%$$$$$$$$$$$$$$ Making Spreaded data a column vector with spreaded data of 1st row of
Tx_data first and followed by others
Spread.Tx_data_col=reshape(Spread.Tx_data,[],1)/sqrt(SF);

%$$$$$$$$$$$$$$ JAKES FADING starts here $$$$$$$$$$$$$$$$$$$%
v=0.3*1000/3600;%velocity of vehicle
vc = 3e8;% velocity of light [m/sec]
fc=2e9;%carrier frequency
fd=v*fc/vc;% maximum doppler shift frequency
N_fb=(L+pilots);% number of bits in one fading block=one code symbol size

%%%%%%%%%%%%%%%%%%%%% Jakes Model Parameter
%%%%%%%%%%%%%%%%%%%%%%%
No = 8;% number of oscillators
N  = 4*No + 2;
alpha = pi/4;
% Other Parameters
k = linspace(1, No, No);
betan = pi*k/No;
wn=2*pi*fd*cos(2*pi*k/N);
tmpc=0;
tmps=0;
for ii=1:((G_Col+NumPilot)/N_fb)
    tmpc=0;
    tmps=0;
    t=fb_num*Tb;%%% synchronizing fading block time with Tb time
    for n=1:1:No
        c = cos(betan(n))*cos(wn(n)*t);
        s = sin(betan(n))*cos(wn(n)*t);
        tmpc=tmpc+c;
        tmps=tmps+s;
    end
    xc=(2*tmpc+(sqrt(2)*cos(alpha)*cos(2*pi*fd*t)))/sqrt(2*No);%%%sqrt(No) is to
    % normalize
    xs=(2*tmps+(sqrt(2)*sin(alpha)*cos(2*pi*fd*t)))/sqrt(2*(No+1));%%%% and 2 as we
    % have complex fading ie, two components sin and cos.
    fade(ii) = xc + sqrt(-1)*xs;
    fb_num=fb_num+N_fb;
end
fade_table(1,:)=fade;
chip_no=1;
for ii=1:1:((G_Col+NumPilot)/N_fb)
    for cc=1:1:(SF*N_fb)
        faded_Tx_data(chip_no)=Spread_Tx_data_col(chip_no)*fade(ii);
        chip_no=chip_no+1;
    end
end
noise=(sqrt(N0(kk)/2))*((randn(1,length(Spread_Tx_data_col)))+sqrt(-1)*(randn(1,length(Spread_Tx_data_col))));
Y_send=faded_Tx_data+noise;

% Transmier Ends Here and Receiver Starts Here

y_received=reshape(Y_send,[],1); % making column vector of received message
for i=1:1:slots
    Y_rec(i,:)=y_received((i-1)*(L+pilots)*SF+1:i*(L+pilots)*SF);
end

for i=1:1:(G_Col+NumPilot)
    y_dspr(i)=Sp_Code*y_received((i-1)*SF+1:i*SF)/sqrt(SF);
end
for i=1:1:slots
    y_Dspread(i,:)=y_dspr((i-1)*(L+pilots)+1:i*(L+pilots));
end
for j=1:1:slots
   sp=1;
   for i=1:1:L*SF/2
      y_info_spread(j,sp)=Y_rec(j,i);
      sp=sp+1;
   end
   for i=(L*SF/2+pilots*SF+1):1:length(Y_rec(1,:))
      y_info_spread(j,sp)=Y_rec(j,i);
      sp=sp+1;
   end
end
for i=1:1:slots
   for j=1:1:length(y_info_spread(1,:))
      y_info_spr(j,i)=y_info_spread(i,j);
   end
end
for row=1:1:slots
   col=1;
   for i=L*SF/2+1:1:(L*SF/2+pilots*SF)
      sprd_plt(row,col)=Y_rec(row,i);
      col=col+1;
   end
end
for i=1:1:slots
   for j=1:1:length(sprd_plt(1,:))
      spread_plt(j,i)=sprd_plt(i,j);
   end
end
spread_Pilot=[];
for i=1:1:slots
   SP_PLT=sprd_plt(i,:);
   spread_Pilot=[spread_Pilot SP_PLT];
end
for j=1:1:slots
   extracting despreaded message bits
end
d_sp=1;
for i=1:1:L/2
    y_info(j,d_sp)=y_Dspread(j,i);
    d_sp=d_sp+1;
end
for i=(L/2+pilots+1):1:length(y_Dspread(1,:))
    y_info(j,d_sp)=y_Dspread(j,i);
    d_sp=d_sp+1;
end
end

for row=1:1:slots
    col=1;
    for i=L/2+1:1:(L/2+pilots)
        Dspread_pilot(row,col)=y_Dspread(row,i);
        col=col+1;
    end
end

for i=1:1:slots
    C_hat1(i)=sum(Dspread_pilot(i,:))/pilots;
end
fade_table(2,:)=C_hat1;
for j=1:1:slots
    for i=1:1:pilots
        N_data(j,i)=norm(spread_plt(((i-1)*SF+1:i*SF),j))^2+(norm(C_hat1(j)*ones(SF,1))^2)/SF-sum(2*real((conj(spread_plt(((i-1)*SF+1:i*SF),j))).*(C_hat1(j)*ones(SF,1)).*Sp_Code'))/sqrt(SF);
    end
    N0_HAT(j)=sum(N_data(j,:))/(SF*pilots);
end
N0_hat1=sum(N0_HAT)/slots;
noise_sigma_hat1=sqrt(N0_hat1/2);
for i=1:1:slots
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat1(i)))*abs(C_hat1(i)));
end
Yss_Chat1=[ ];
Yss_Chat1=[Yss_Chat1 YSS_CHAT(i,:)];
end
f1=1./(1+exp(-4*Yss_Chat1/(noise_sigma_hat1^2)));
% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat1,f0,f1,H);
S_Beta=Q0;
Success=success;

x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2));
%G=[A|I] so end bits are mssg bits
'COLUMN Vector'
x_hat=x_hat';%making Column vector a ROW vector
delx=x-x_hat;
err=length(find(delx));
err1(:,ff)=err;
error1(kk)=error1(kk)+err;

for i=1:1:slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else
            s_beta(num,i)=0;
        end
    end
    for i=(L/2+1):1:L
        s_beta(num,i+pilots)=S_beta(num,i);
    end
end
for i=1:1:slots
    C_hat(i,1)=C_hat1(1,i);
end
N0_hat=N0_hat1;
itr=1;

C_hat1_R_itr1_row=C_hat1;
N0_hat1_itr1=N0_hat1;
while ((success == 0) & (itr < (max_iter))},
for rec_itr=1:1:10 %%% EM iteration loop
for j=1:1:slots
    ldpc_num=1; %%% index of bit given by LDPC
    for i=1:1:(L+pilots)%%% calculating u_beta_bar for all sent codesymbols
        R=exp(-1/N0_hat1_itr1*4*(real(conj(Y_rec(j,((i-1)*SF+1:i*SF)))*Sp_Code*C_hat1_R_itr1_row(j)/sqrt(SF))));%%% calculating R_beta
        if R==inf
            R=100000000;
        end
        u_beta_bar(j,i)=(1-s_beta(j,ldpc_num)-s_beta(j,ldpc_num)*R)/(1-s_beta(j,ldpc_num)+s_beta(j,ldpc_num)*R); %%% u_beta_bar
        if isnan(u_beta_bar(j,i))
            u_beta_bar(j,i) = -1;
        end
        ldpc_num=ldpc_num+1;
    end
    for j=1:1:slots %%% extracting u_beta values for just message bits
        u_b=1;
        for i=1:1:L/2
            U_Beta_Bar(j,u_b)=u_beta_bar(j,i);
            u_b=u_b+1;
        end
        for i=(L/2+pilots+1):1:length(u_beta_bar(1,:))
            U_Beta_Bar(j,u_b)=u_beta_bar(j,i);
            u_b=u_b+1;
        end
    end
end

for i=1:1:slots %%% calculating one fading value per subframe using all the bits in the subframe
    C_hat1_R_itr1_col(i,rec_itr)=sum(y_Dspread(i,:).*u_beta_bar(i,:))/(length(y_Dspread(i,:)));
end
for j=1:1:slots %%% calculating noise PSD for each subframe using all the bits in the subframe
    for i=1:1:length(y_info(1,:))
        N_data_itr(j,i)=norm(y_info_spr(((i-1)*SF+1:i*SF),j))^2+(norm(C_hat1_R_itr1_col(j,rec_itr)*ones(SF,1))^2)/SF-
    end
sum(2*real((conj(y_info_spr(((i-1)*SF+1:i*SF),j).)*C_hat1_R_itr1_col(j,rec_itr)*ones(SF,1)).*Sp_Code'.*(U_Beta_Bar(j,i)*ones(SF,1))))/sqrt(SF);
end
N0_HAT_itr(j)=sum(N_data_itr(j,:))/(SF*length(y_info(1,:)));
end
N0_hat1_R_itr1=sum(N0_HAT_itr)/slots;%%%%calculating noise PSD
for i=1:1:length(C_hat1_R_itr1_col(:,1))
    C_hat1_R_itr1_row(1,i)=C_hat1_R_itr1_col(i,rec_itr);
end
end
N0_hat=N0_hat1_R_itr1;
C_hat=C_hat1_R_itr1_col(:,rec_itr);
fade_table((itr+2),:)=C_hat;
N0_table((itr+2),:)=N0_hat;
noise_sigma_hat=sqrt(N0_hat/2);
for i=1:1:slots
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat(i)))*abs(C_hat(i)));
end
Yss_Chat=[];
for i=1:1:slots
    Yss_Chat=[Yss_Chat YSS_CHAT(i,:)];
end
f1=1./(1+exp(-2*Yss_Chat/noise_sigma_hat^2));% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat,f0,f1,H);
SUCCESS(itr)=success;
S_Beta=Q0;  % uB=-1=sB
x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2));  %G=[A|I] so end bits are mssg bits
x_hat=x_hat';
delx = x-x_hat;
err = length(find(delx));
error2(itr,kk)=error2(itr,kk)+err;

for i=1:1:slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else

for i=1:1:slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else

39
s_beta(num,i)=0;
end
end
for i=(L/2+1):1:L
    s_beta(num,i+pilots)=s_beta(num,i);
end
end
itr = itr+1;
end
end

Total_bits=frame_index*(G_Row);
j_0 = error1/Total_bits
j_1 = error2(1,:)/Total_bits
j_3 = error2(3,:)/Total_bits
j_5 = error2(5,:)/Total_bits
j_7 = error2(7,:)/Total_bits
j_9 = error2(9,:)/Total_bits
ebn0dB
figure(6)
semilogy(ebn0dB,j_0,'bo-',ebn0dB,j_1,'gx-',ebn0dB,j_3,'mx-',ebn0dB,j_5,'rx-',ebn0dB,j_7,'bx-',ebn0dB,j_9,'ro-');
grid on;
legend('j_0','j_1','j_3','j_5','j_7','j_9');
xlabel('Eb/No,dB');
ylabel('BER');
title('EM with C hat & No updated N_f = 1000, B.S = [1000,2000], N_p = 200');
PROGRAM 2. EM ALGORITHM FOR BPSK MODULATION AND RAYLEIGH FADING

clc;
clear all;
frame_index=1000;
rows=1000;
cols=2000;
q=2;
H1=ldpc_generate(rows,cols,3,2,123); % Generate sparse parity check matrix H1
[H,G] = ldpc_h2g(H1,q);
G_Row = size(G,1); % K : size of information bits
G_Col = size(G,2);
ii=0;
err=[];
NumPilot=100;
max_iter=10;

%%%%% Eb/N0 loop %%%%%
for EsN0dB =-3:2:25 %dB
    ii=ii+1;
    Es=1;
esn0dB(ii)=EsN0dB;
    EsN0(ii) = 10^(EsN0dB/10); % EbNo
    R=G_Row/G_Col; % rate of encoder
    Eb=Es/R; % data bit energy
    Ecod=R*Eb;
    M=2; for BPSK modulation
    Es=log2(M)*Ecod; % General relationship between Eb and Es
    SF=8; Spreading Factor
    E_sf=Es/sqrt(SF);
    N0(ii)=Es/EsN0(ii); % SF*1/Rate *10^(-EbN0dB/10)
    noise_sigma(ii) = sqrt( N0(ii)/(2));
    EbNo(ii)=Eb/N0(ii);
enb0dB(ii)=10*log10(EbNo(ii));
end

Rb=100e3; 10Kbits/sec
Tb=1/Rb;
error1=zeros(1,length(esn0dB));
error2=zeros(max_iter,length(esn0dB));
N0_table(1,:)=N0;
fb_num=1; % counter for fade block number
for kk = 1:length(esn0dB)
    % Transmitter Starts Here
    %
    for ff=1:1:frame_index
        x=floor(rand(1,G_Row)*2); % generates stream of message bits containing zeros and ones
        mG = mod(x*G,2); % encoding the message bits in code symbols
        slots=20; % number of divisions of each frame
        L=G_Col/slots; % length of each slot
        % division of frame into slots
        for i=1:1:slots
            xldpc(i,:)=mG((i-1)*L+1:i*L);
        end
        % adding pilots at midamble position in each slot
        pilots=NumPilot/slots;
        for num=1:1:slots
            for i=1:1:(L/2+pilots);
                if i<=L/2
                    mG_plt(num,i)=xldpc(num,i);
                else
                    mG_plt(num,i)=1;
                end
            end
            for i=(L/2+1):1:L
                mG_plt(num,i+pilots)=xldpc(num,i);
            end
        end
        % concatenating the slots in one row
        mG_pilot=[];
        for i=1:1:slots
            MG_PLT=mG_plt(i,:);
            mG_pilot=[mG_pilot MG_PLT];
        end
    end
    Tx_data=2*mG_pilot-1;
    codes=hadamard(SF);
    Sp_Code=codes(1,:); % Spreading Sequence Generation
end
Spread_Tx_data(:,i) = Tx_data(i)*Sp_Code;

%$$$$$$$$$$$$$$ Making Spreaded data a column vector with spreaded data of 1st row of 
Tx_data first and followed by others 
Spread_Tx_data_col=reshape(Spread_Tx_data,[],1)/sqrt(SF);

%$$$$$$$$$$$$$$ Rayleigh Fading starts here $$$$$$$$$$$$$$$$%

N_fb=(L+pilotes);% number of bits in one fading block=one code symbol size 

for ii=1:1:((G_Col+NumPilot)/N_fb) 
    fade(ii) = (sqrt(1/2))*((randn(1,1))+sqrt(-1)*(randn(1,1)));
end

fade_table(1,:)=fade;
chip_no=1;
for ii=1:1:((G_Col+NumPilot)/N_fb) 
    for cc=1:1:(SF*N_fb) 
        faded_Tx_data(chip_no)=Spread_Tx_data_col(chip_no)*fade(ii);
        chip_no=chip_no+1;
    end
end

noise=(sqrt(N0(kk)/2))*((randn(1,length(Spread_Tx_data_col)))+ sqrt(-
1)*(randn(1,length(Spread_Tx_data_col))));

Y_send=faded_Tx_data+noise;

%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%%%%%%%%
%%%%%%%%%%%%%%%%%%

%%%%%  Transmitter Ends Here and Receiver Starts Here

%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
%\%
y_receieved=reshape(Y_send,[],1); % making column vector of received message

for i=1:1:slots % dividing received spreaded data into slots
    Y_rec(i,:)=y_receieved((i-1)*(L+pilots)*SF+1:i*(L+pilots)*SF);
end

for i=1:1:(G_Col+NumPilot) % for despreading the complete received data
    y_dspr(i)=Sp_Code*y_receieved((i-1)*SF+1:i*SF)/sqrt(SF);
end

for i=1:1:slots % dividing complete despreaded data in slots
    y_Dspread(i,:)=y_dspr((i-1)*(L+pilots)+1:i*(L+pilots));
end

for j=1:1:slots % extracting only spreaded message bits
    sp=1;
    for i=1:1:L*SF/2
        y_info_spread(j,sp)=Y_rec(j,i);
        sp=sp+1;
    end
    for i=(L*SF/2+pilots*SF+1):1:length(Y_rec(1,:))
        y_info_spread(j,sp)=Y_rec(j,i);
        sp=sp+1;
    end
end

for i=1:1:slots % making rows columns and columns rows
    for j=1:1:length(y_info_spread(1,:))
        y_info_spr(j,i)=y_info_spread(i,j);
    end
end

for row=1:1:slots % extracting spreaded pilots
    col=1;
    for i=L*SF/2+1:1:(L*SF/2+pilots*SF)
        sprd_plt(row,col)=Y_rec(row,i);
end
end
for i=1:1:slots
    for j=1:1:length(sprd_plt(1,:))
        spread_plt(j,i)=sprd_plt(i,j);
    end
end
spread_Pilot=[];
for i=1:1:slots
    SP_PLT=sprd_plt(i,:);
    spread_Pilot=[spread_Pilot SP_PLT];
end

for j=1:1:slots
    d_sp=1;
    for i=1:1:L/2
        y_info(j,d_sp)=y_Dspread(j,i);
        d_sp=d_sp+1;
    end
    for i=(L/2+pilots+1):1:length(y_Dspread(1,:))
        y_info(j,d_sp)=y_Dspread(j,i);
        d_sp=d_sp+1;
    end
end

for row=1:1:slots
    col=1;
    for i=L/2+1:1:(L/2+pilots)
        Dspread_pilot(row,col)=y_Dspread(row,i);
        col=col+1;
    end
end

for i=1:1:slots
    C_hat1(i)=sum(Dspread_pilot(i,:))/pilots;
end
fade_table(2,:)=C_hat1;
for j=1:1:slots
    for i=1:1:pilots
N_data(j,i)=norm(spread_plt(((i-1)*SF+1:i*SF),j))^2+(norm(C_hat1(j)*ones(SF,1))^2)/SF-sum(2*real((conj(spread_plt(((i-1)*SF+1:i*SF),j))).*(C_hat1(j)*ones(SF,1)).*Sp_Code'))/sqrt(SF);

end
N0_HAT(j)=sum(N_data(j,:))/(SF*pilots);
end
N0_hat1=sum(N0_HAT)/slots;%%calculating initial value of noise PSD using pilots
N0_table(2,:)=N0_hat1;
noise_sigma_hat1=sqrt(N0_hat1/2);

for i=1:1:slots%%calculating values to pass to LDPC encoder
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat1(i)))*abs(C_hat1(i)));
end
Yss_Chat1=[];%% concatenating these slots values
for i=1:1:slots
    Yss_Chat1=[Yss_Chat1 YSS_CHAT(i,:)];
end
f1=1./(1+exp(-4*Yss_Chat1/(noise_sigma_hat1^2)));% likelihoods prob of sending '1'

end
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat1,f0,f1,H);
S_Beta=Q0;
Success=success;

x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2));   %G=[A|I] so end bits are mssg bits
"COLUMN Vector
x_hat=x_hat';%making Column vector a ROW vector
delx=x-x_hat;
err=length(find(delx));
err1(:,ff)=err;
error1(kk)=error1(kk)+err;

for i=1:1:slots%% dividing S_beta in slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots%% adding pilots s_beta at midamble position
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else
            s_beta(num,i)=0;
        end
    end
end
for i=(L/2+1):1:L
    s_beta(num,i+pilots)=S_beta(num,i);
end
end

for i=1:1:slots
    C_hat(i,1)=C_hat1(1,i);
end
N0_hat=N0_hat1;
itr=1;

C_hat1_R_itr1_row=C_hat1;
N0_hat1_itr1=N0_hat1;

while ((success == 0) & (itr < (max_iter)),
    for rec_itr=1:1:10 %%%%%EM iteration loop
        for j=1:1:slots
            ldpc_num=1;%%%%%index of bit given by LDPC
            for i=1:1:(L+pilots)%%%%%%%%%calculating u_beta_bar for all sent
codesymbols
                R=exp(-1/N0_hat1_itr1*4*(real(conj(Y_rec(j,(i-1)*SF+1:i*SF))*Sp_Code'*C_hat1_R_itr1_row(j)/sqrt(SF))));%%%%%calculating R_beta
                if R==inf
                    R=100000000;
                end
                u_beta_bar(j,i)=(1-s_beta(j,ldpc_num)-s_beta(j,ldpc_num)*R)/(1-s_beta(j,ldpc_num)+s_beta(j,ldpc_num)*R);%%%%%u_beta_bar
                if isnan(u_beta_bar(j,i))
                    u_beta_bar(j,i) = -1;
                end
            ldpc_num=ldpc_num+1;
        end
        for j=1:1:slots%%%%%%%extracting u_beta values for just message bits
            u_b=1;
            for i=1:1:L/2
                U_Beta_Bar(j,u_b)=u_beta_bar(j,i);
                u_b=u_b+1;
            end
        end
    end
end
for i=(L/2+pilots+1):1:length(u_beta_bar(1,:))
    U_Beta_Bar(j,u_b)=u_beta_bar(j,i);
    u_b=u_b+1;
end
end

for i=1:1:slots calculating one fading value per subframe using all the bits in the subframe
C_hat1_R_itr1_col(i,rec_itr)=sum(y_Dspread(i,:).*u_beta_bar(i,:))/(length(y_Dspread(1,:)));  
end
for j=1:1:slots calculating noise PSD for each subframe using all the bits in the subframe
for i=1:1:length(y_info(1,:))
    N_data_itr(j,i)=norm(y_info_spr(((i-1)*SF+1:i*SF),j))^2+(norm(C_hat1_R_itr1_col(j,rec_itr)*ones(SF,1))^2)/SF-
    sum(2*real((conj(y_info_spr(((i-1)*SF+1:i*SF),j))).*(C_hat1_R_itr1_col(j,rec_itr)*ones(SF,1)).*Sp_Code'.*(U_Beta_Bar(j,i)*ones(SF,1))))/sqrt(SF);
end
N0_HAT_itr(j)=sum(N_data_itr(j,:))/(SF*length(y_info(1,:)));  
end
N0_hat1_R_itr1=sum(N0_HAT_itr)/slots; calculating noise PSD
for i=1:1:length(C_hat1_R_itr1_col(:,1))
    C_hat1_R_itr1_row(1,i)=C_hat1_R_itr1_col(i,rec_itr);
end
N0_hat=N0_hat1_R_itr1;
C_hat=C_hat1_R_itr1_col(:,rec_itr);
fade_table((itr+2),:)=C_hat;
N0_table((itr+2),:)=N0_hat;
noise sigma_hat=sqrt(N0_hat/2);  
for i=1:1:slots
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat(i)))*abs(C_hat(i)));  
end
Yss_Chat=[];
for i=1:1:slots
    Yss_Chat=[Yss_Chat YSS_CHAT(i,:)];
end
f1=1./(1+exp(-2*Yss_Chat/(noise sigma_hat^2)));% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat,f0,f1,H);
SUCCESS(itr)=success;
S_Beta=Q0; % uB=-1=sB
\( x_\text{hat} = z_\text{hat}(\text{size}(G,2)+1-\text{size}(G,1):\text{size}(G,2)); \quad \%G=[A\mid I] \) so end bits are mssg bits
\( x_\text{hat}=x_\text{hat}'; \)
\( \text{delx} = x-x_\text{hat}; \)
\( \text{err} = \text{length}(`\text{find}(`\text{delx}))`; \)
\( \text{errr(itr,ff)}=\text{err}; \)
\( \text{error2(itr,kk)}=\text{error2(itr,kk)}+\text{err}; \)

\[
\begin{align*}
\text{for } i=1:1:\text{slots} \\
\quad S_\text{beta}(i,:) &= S_\text{Beta}((i-1)*L+1:i*L); \\
\text{end} \\
\text{for num=1:1:\text{slots}} \\
\quad \text{for } i=1:1:(L/2+pilots); \\
\quad \quad \quad \text{if } i<=L/2 \\
\quad \quad \quad \quad S_\text{beta}(\text{num},i) &= S_\text{beta}(\text{num},i); \\
\quad \quad \quad \text{else} \\
\quad \quad \quad \quad S_\text{beta}(\text{num},i) &= 0; \\
\quad \quad \text{end} \\
\quad \text{end} \\
\quad \text{for } i=(L/2+1):1:L \\
\quad \quad S_\text{beta}(\text{num},i+pilots) &= S_\text{beta}(\text{num},i); \\
\quad \text{end} \\
\quad \text{end} \\
\quad \text{itr} = \text{itr}+1; \\
\text{end} \\
\text{end} \\
\text{end}
\end{align*}
\]

\( \text{Total_bits} = \text{frame_index}*(\text{G_Row}); \)
\( j_0 = \text{error1/Total_bits} \)
\( j_1 = \text{error2(1,:)/Total_bits} \)
\( j_3 = \text{error2(3,:)/Total_bits} \)
\( j_5 = \text{error2(5,:)/Total_bits} \)
\( j_7 = \text{error2(7,:)/Total_bits} \)
\( j_9 = \text{error2(9,:)/Total_bits} \)
\( \text{ebn0dB} \)
\( \text{figure}(6) \)
\( \text{semilogy}(\text{ebn0dB},j_0,'bo-',\text{ebn0dB},j_1,'gx-',\text{ebn0dB},j_3,'mx-',\text{ebn0dB},j_5,'rx-',\text{ebn0dB},j_7,'bx-',\text{ebn0dB},j_9,'ro-'); \)
\( \text{grid on}; \)
\( \text{legend}('j_0','j_1','j_3','j_5','j_7','j_9'); \)
\( \text{xlabel}('\text{Eb/No,dB}'); \)
\( \text{ylabel}('\text{BER}'); \)
\( \text{title}('\text{EM with C hat & No updated N_f = 1000, B.S = [1000,2000], N_p = 200}') ;\)
PROGRAM 3. EM ALGORITHM FOR QPSK MODULATION

clc;
clear all;
frame_index=1000;
rows=1000;
cols=2000;
q=2;

H1=ldpc_generate(rows,cols,3,2,123); % Generate sparse parity check matrix H1
[H,G] = ldpc_h2g(H1,q);
G_Row = size(G,1);                   % K : size of information bits
G_Col = size(G,2);

for EsN0dB = -3:2:25
    ii=ii+1;
    Es=1;                       % Energy of transmitted symbol
    esn0dB(ii)=EsN0dB;
    EsN0(ii) = 10^(EsN0dB/10);
    N0(ii)=Es/EsN0(ii);          % SF*1/Rate * 10^(-EbN0dB/10)
    noise_sigma(ii) = sqrt( N0(ii)/(2));
end

error1 = zeros(1,length(esn0dB));

for kk = 1:length(esn0dB)
    % Transmitter Starts Here
    for ff=1:frame_index
        msg = randint(1,G_Row); % generates random stream of message bits containing zeros and ones
        ldpc = mod(msg*G,2);   % encoding the message bits to LDPC code bits
        for tt=0:length(msg)-1
            pair = [ldpc(2*tt+1) ldpc(2*tt+2)];
            if (isequal(pair,[0 0])) == 1) sym(tt+1) = sqrt(-1); end
            if (isequal(pair,[0 1]) == 1) sym(tt+1) = 1; end
            if (isequal(pair,[1 0]) == 1) sym(tt+1) = -1; end
            if (isequal(pair,[1 1]) == 1) sym(tt+1) = -sqrt(-1); end   % constellation mapping
        end
    end

end
modp =4; %Pilots as +1 symbol
sym_p = [sym modp];
noise = (sqrt(N0(kk)/2))*((randn(1,length(sym_p)))+ sqrt(-1)*(randn(1,length(sym_p))));
fade = sqrt(0.5)*randn;
z = fade*sym_p + noise; %Fading plus noise

%%% %Received Signal%%% %
for m = 1:length(sym)
z_msg(m) = z(m);
end
pilot = z(length(sym_p));
C_hatp = abs(pilot); %From pilots
abC_hatp = abs(C_hatp);
xk = (real(z_msg*exp(-sqrt(-1)*pi/4)))/abC_hatp; %determines odd code bits
yk = (imag(z_msg*exp(-sqrt(-1)*pi/4)))/abC_hatp; %determines even code bits

Np1 = abs(pilot)^2 + abC_hatp^2 - sum(2*real((conj(pilot)*C_hatp)));
N0_hatp = sum(Np1)/1; %From pilots

%Log-likelihood ratios
fxp1=1./(1+exp(4*xk*(1/sqrt(2))/(N0_hatp/2)));
fxp0=1-fxp1;
fyp1=1./(1+exp(4*yk*(1/sqrt(2))/(N0_hatp/2)));
fyp0=1-fyp1;

%%% Concatenation Part
mm=1;
for yy = 1:length(xk)
yss_chat1(mm) = xk(yy);
yss_chat1(mm+1) = yk(yy);
f1(mm)=fxp1(yy);
f1(mm+1) = fyp1(yy);
mm=mm+2;
end
f0 = 1-f1;

yss_chat = real(yss_chat1*exp(-sqrt(-1)*angle(C_hatp)*abs(C_hatp)));

[z_hat1, success1, k1, Q0, Q1] = ldpc_decode(yss_chat,f0,f1,H); %s_beta = Q0
x_hatp = z_hat1(size(G,2)+1-size(G,1):size(G,2)); %G=[A|I] so end bits are mssg bits
'COLUMN Vector
x_hatp=x_hatp'; %making Column vector a ROW vector
delx=msg-x_hatp;
errp(ff)=length(find(delx));
mm = 1;
for vv = 1:2:length(Q0)
  p1(mm) = Q0(vv);
  p2(mm) = Q0(vv + 1);
  mm = mm + 1;
end

s1 = p1.*(1-p2); % Qpsk: +1 ie. "01"
  s2 = p1.*p2;   % Qpsk: +j ie. "00"
  s3 = (1-p1).*p2; % Qpsk: -1 ie. "10"
  s4 = (1-p1).*(1-p2); % Qpsk: -j ie. "11"

R11 = exp((2*C_hatp/N0(kk))*xk);
R21 = exp((2*C_hatp/N0(kk))*yk);
R31 = exp((-2*C_hatp/N0(kk))*xk);
R41 = exp((-2*C_hatp/N0(kk))*yk);

xk_bar = (s1.*R11 + sqrt(-1)*s2.*R21 - s3.*R31 - sqrt(-1)*s4.*R41)./(s1.*R11 + s2.*R21
+ s3.*R31 + s4.*R41);

C_hat1 = (1/rows)*sum(real(conj(z_msg).*xk_bar));
N0_hat1 = (1/rows)*sum(abs(z_msg-C_hat1*xk_bar).^2);

xk1 = (real(z_msg*exp(-sqrt(-1)*pi/4))/C_hat1; % determines odd code bits
  yk1 = (imag(z_msg*exp(-sqrt(-1)*pi/4))/C_hat1; % determines even code bits

fx1 = 1./(1+exp(4*xk1*(1/sqrt(2))/(N0_hat1/2)));
fx0 = 1-fx1;
fy1 = 1./(1+exp(4*yk1*(1/sqrt(2))/(N0_hat1/2)));
fy0 = 1-fy1;

mm = 1;
for yy = 1:length(xk)
  yss_chat(mm) = xk1(yy);
  yss_chat(mm+1) = yk1(yy);
  f1(mm) = fx1(yy);
  f1(mm+1) = fy1(yy);
  mm = mm + 2;
end
f0 = 1-f1;

yss_chat = real(yss_chat1*exp(-sqrt(-1)*angle(C_hat1)*abs(C_hat1))); 
[z_hat1, success1, k1, Q0, Q1] = ldpc_decode(yss_chat,f0,f1,H); % s_beta = Q0
\[ x\_\text{hat} = z\_\text{hat}(\text{size}(G,2)+1-\text{size}(G,1):\text{size}(G,2)); \quad \%G=[A|I] \text{ so end bits are msg bits} \]

"COLUMN Vector"
\[ x\_\text{hat} = x\_\text{hat}'; \quad \%\text{making Column vector a ROW vector} \]

\[ \text{delx=msg-x\_hat;} \]
\[ \text{err1(ff)=length(find(delx));} \quad \%\text{Error count} \]

\[ \text{end} \quad \%\text{Monte Carlo Loop} \]
\[ \text{errorp\_snr(kk) = mean(errp)/rows;} \]
\[ \text{error1\_snr(kk) = mean(err1)/rows;} \]
\[ \text{end} \quad \%\text{SNR Loop} \]
PROGRAM 4. ADAPTIVE EM ALGORITHM FOR BPSK MODULATION, JAKES FADING AND MULTIUSER ENVIRONMENT

clc;
clear all;
frame_index=1000;
rows=1000;
cols=2000;
q=2;
H1=ldpc_generate(rows,cols,3,2,123); %% Generate sparse parity check matrix H1
[H,G] = ldpc_h2g(H1,q);
G_Row = size(G,1); % K : size of information bits
G_Col = size(G,2);
ii=0;
err=[];
NumPilot=200; %%%%%%%%%Total number of pilot bits
max_iter=10; %%% maximum receiver iterations

%%% Eb/N0 loop %%%
for EsN0dB = -3:2:25 %dB
    ii=ii+1;
    Es=1;
esn0dB(ii)=EsN0dB;
    EsN0(ii) = 10^(EsN0dB/10); % EbN0
    R=G_Row/G_Col;% rate of encoder
    Eb=Es/R; %data bit energy
    Ecod=R*Eb;
    M=2; %for BPSK modulation
    Es=log2(M)*Ecod; %General relationship between Eb and Es
    SF=8; %Spreading Factor
    E_sf=Es/sqrt(SF);
    N0(ii)=Es/EsN0(ii); % SF*1/Rate *10^-EbN0dB/10
    noise_sigma(ii) = sqrt( N0(ii)/(2));
    EbNo(ii)=Eb/N0(ii);
ebn0dB(ii)=10*log10(EbNo(ii));
end

Rb=100e3;%10Kbits/sec
Tb=1/Rb;
error1=zeros(1,length(esn0dB));
error2=zeros(max_iter,length(esn0dB));

N0_table(1,:)=N0;
fb_num=1;%counter for fade block number
for kk = 1:length(esn0dB)
  % Transmitter Starts Here
  intcount=1;
  for ff=1:frame_index
    x=floor(rand(1,G_Row)*2);%generates stream of message bits containing zeros and ones
    mG = mod(x*G,2);%encoding the message bits in code symbols
    slots=50;%number of divisions of each frame
    L=G_Col/slots;%length of each slot

    % division of frame into slots
    for i=1:slots
      xldpc(i,:)=mG((i-1)*L+1:i*L);
    end

    % adding pilots at midamble position in each slot
    pilots=NumPilot/slots;
    for num=1:slots
      for i=1:(L/2+pilots);
        if i<=L/2
          mG_plt(num,i)=xldpc(num,i);
        else
          mG_plt(num,i)=1;
        end
      end
      for i=(L/2+1):1:L
        mG_plt(num,i+pilots)=xldpc(num,i);
      end
    end

    % concatenating the slots in one row
    mG_pilot=[];
    for i=1:slots
      MG_PLT=mG_plt(i,:);
      mG_pilot=[mG_pilot MG_PLT];
    end

    Tx_data=2*mG_pilot-1;
    Tx_data2 = randsrc(1,length(Tx_data));
codes=hadamard(SF);
Sp_Code=codes(3,:);
Sp_Code2=randsrc(1,SF);

%$$$$$$$$$$$$ Spreading Sequence Generation $$$$$$$$$$$$$$$%
for i=1:length(Tx_data)
    Spread_Tx_data(:,i) = Tx_data(i)*Sp_Code;
    Spread_Tx_data2(:,i) = Tx_data2(i)*Sp_Code2;
end

%$$$$$$$$$$$$ Making Spreaded data a column vector with spreaded data of 1st row of
Tx_data first and followed by others
Spread_Tx_data_col=reshape(Spread_Tx_data,[],1)/sqrt(SF);
Spread_Tx_data_col2=reshape(Spread_Tx_data2,[],1)/sqrt(SF);

%$$$$$$$$$$$$ JAKES FADING starts here $$$$$$$$$$$$$$$$$$%
v=300*1000/3600;%velocity of vehicle
v2=100*1000/3600;%velocity of vehicle 2
vc = 3e8;% velocity of light [m/sec]
fC=2e9;%carrier frequency
fd=v*fc/vc;% maximum doppler shift frequency
fd2=v2*fc/vc;% maximum doppler shift frequency of vehicle 2
N_fb=(L+pilots);% number of bits in one fading block=one code symbol size

%%%%%%%%%%%%%%%%%%%%% Jakes Model Parameter
%%%%%%%%%%%%%%%%%%%%%%%
No = 8;% number of oscillators
N  = 4*No + 2;
alpha = pi/4;
% Other Parameters
k = linspace(1, No, No);
betan = pi*k/No;
wn=2*pi*fd*cos(2*pi*k/N);
wn2=2*pi*fd2*cos(2*pi*k/N); %%%%%   ADDITION    %%%%%%%%
tmpc=0;
tmph=0;
tmpc2=0;
tmph2=0;
for ii=1:1:((G_Col+NumPilot)/N_fb)
    tmpc=0;
tmph=0;
tmpc2=0;
tmph2=0;
tmpls2=0;
t=fb_num*Tb;%% synchronizing fading block time with Tb time
for n=1:1:No
    c = cos(betan(n))*cos(wn(n)*t);
    s = sin(betan(n))*cos(wn(n)*t);
    tmpc=tmpc+c;
    tmps=tmps+s;
    c2 = cos(betan(n))*cos(wn2(n)*t);
    s2 = sin(betan(n))*cos(wn2(n)*t);
    tmpc2=tmpc2+c2;
    tmps2=tmps2+s2;
end
xc=(2*tmpc+(sqrt(2)*cos(alpha)*cos(2*pi*fd*t)))/sqrt(2*No);%%sqrt(No) is to
normalize
xs=(2*tmps+(sqrt(2)*sin(alpha)*cos(2*pi*fd*t)))/sqrt(2*(No+1));%%and 2 as we
have complex fading i.e., two components sin and cos.
fade(ii) = xc + sqrt(-1)*xs;
xc2=(2*tmpc2+(sqrt(2)*cos(alpha)*cos(2*pi*fd2*t)))/sqrt(2*No);%%sqrt(No) is
to normalize
xs2=(2*tmps2+(sqrt(2)*sin(alpha)*cos(2*pi*fd2*t)))/sqrt(2*(No+1));%%and 2 as we
have complex fading i.e., two components sin and cos.
    fade2(ii) = xc2 + sqrt(-1)*xs2;
    fb_num=fb_num+N_fb;
end
fade_table(1,:)=fade;
fade_table2(1,:)=fade2;
chip_no=1;
for ii=1:1:((G_Col+NumPilot)/N_fb)
    for cc=1:1:(SF*N_fb)
        faded_Tx_data(chip_no)=Spread_Tx_data_col(chip_no)*fade(ii);
        faded_Tx_data2(chip_no)=Spread_Tx_data_col2(chip_no)*fade2(ii);
        chip_no=chip_no+1;
    end
end

noise=(sqrt(N0(kk)/2))*((randn(1,length(Spread_Tx_data_col)))+ sqrt(-1)*randn(1,length(Spread_Tx_data_col))));
abc=rem(intcount,10);
if (abc == 0)
    Y_send=faded_Tx_data+faded_Tx_data2+noise;%% Y=Hx+n row vector 1 x
else
    Y_send=faded_Tx_data + noise;
end
intcount=intcount+1;

y_recieved=reshape(Y_send,[],1);%%%%%making column vector of recieved message
for i=1:1:slots%%%%%dividing recieved SPREADED data into slots
    Y_rec(i,:)=y_recieved((i-1)*(L+pilots)*SF+1:i*(L+pilots)*SF);
end

for i=1:1:(G_Col+NumPilot)%%%%%for Despreading the complete recived data
    y_dspr(i)=Sp_Code*y_recieved((i-1)*SF+1:i*SF)/sqrt(SF);
end
for i=1:1:slots%%%%%dividing complete despreaded data in slots
    y_Dspread(i,:)=y_dspr((i-1)*(L+pilots)+1:i*(L+pilots));
end

for j=1:1:slots%%%%%extracting ony SPREADED message bits
    sp=1;
    for i=1:1:L*SF/2
        y_info_spread(j,sp)=Y_rec(j,i);
        sp=sp+1;
    end
    for i=(L*SF/2+pilots*SF+1):1:length(Y_rec(1,:))
        y_info_spread(j,sp)=Y_rec(j,i);
        sp=sp+1;
    end
end
end
for i=1:1:slots
    for j=1:1:length(y_info_spread(1,:))
        y_info_spread(j,i)=y_info_spread(i,j);
    end
end

for row=1:1:slots
    col=1;
    for i=L*SF/2+1:1:(L*SF/2+pilots*SF)
        sprd_plt(row,col)=Y_rec(row,i);
        col=col+1;
    end
end
spread_Pilot=[]; concatinating the SPREADED pilots together
for i=1:1:slots
    SP_PLT=sprd_plt(i,:);
    spread_Pilot=[spread_Pilot SP_PLT];
end

for j=1:1:slots
    d_sp=1;
    for i=1:1:L/2
        y_info(j,d_sp)=y_Dspread(j,i);
        d_sp=d_sp+1;
    end
    for i=(L/2+pilots+1):1:length(y_Dspread(1,:))
        y_info(j,d_sp)=y_Dspread(j,i);
        d_sp=d_sp+1;
    end
end

for row=1:1:slots
    extracting despreaded pilot bits
end
col=1;
for i=L/2+1:1:(L/2+pilots)
    Dspread_pilot(row,col)=y_Dspread(row,i);
    col=col+1;
end
end

for i=1:1:slots%calculating initial value of fading using pilots
    C_hat1(i)=sum(Dspread_pilot(i,:))/pilots;
end
fade_table(2,:)=C_hat1;
for j=1:1:slots
    for i=1:1:pilots
        N_data(j,i)=norm(spread_plt(((i-1)*SF+1:i*SF),j))^2+(norm(C_hat1(j)*ones(SF,1))^2)/SF-sum(2*real((conj(spread_plt(((i-1)*SF+1:i*SF),j))).*(C_hat1(j)*ones(SF,1)).*Sp_Code'))/sqrt(SF);
    end
    N0_HAT(j)=sum(N_data(j,:))/(SF*pilots);
end
N0_hat1=sum(N0_HAT)/slots;%calculating initial value of noise PSD using pilots
N0_table(2,:)=N0_hat1;
noise_sigma_hat1=sqrt(N0_hat1/2);

for i=1:1:slots%calculating values to pass to LDPC encoder
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat1(i)))*abs(C_hat1(i)));
end
Yss_Chat1=[];%concatinating these slots values
for i=1:1:slots
    Yss_Chat1=[Yss_Chat1 YSS_CHAT(i,:)];
end
f1=1./(1+exp(-4*Yss_Chat1/(noise_sigma_hat1^2)));% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat1,f0,f1,H);
S_Beta=Q0;
Success=success;

x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2)); %G=[A|I] so end bits are mssg bits
"COLUMN Vector
x_hat=x_hat';%making Column vector a ROW vector
delx=x-x_hat;
err=length(find(delx));
error1(:,ff)=err;
error1(kk)=error1(kk)+err;
for i=1:1:slots % dividing S_beta in slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end

for num=1:1:slots % adding pilots s_beta at midamble position
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else
            s_beta(num,i)=0;
        end
    end
    for i=(L/2+1):1:L
        s_beta(num,i+pilots)=S_beta(num,i);
    end
end

for i=1:1:slots
    C_hat(i,1)=C_hat1(1,i);
end

N0_hat=N0_hat1;
itr=1;

while ((success == 0) & (itr < (max_iter))),
    for rec_itr=1:1:6 % EM iteration loop
        for j=1:1:slots
            ldpc_num=1; % index of bit given by LDPC
            for i=1:1:(L+pilots)
                % calculating u_beta_bar for all sent codesymbols
                R=exp(-1/N0_hat1_itr1_row*4*(real(conj(Y_rec(j,((i-1)*SF+1:i)*SF))*Sp_Code'*C_hat1_R_itr1_row(j)/sqrt(SF))));
            end
            if R==inf
                R=100000000;
            end
        end
    end
\[
\begin{align*}
    u_{\beta_{\text{bar}}}(j,i) &= (1 - s_{\beta_{\text{j}}}(j,ldpc_{\text{num}}) - s_{\beta_{\text{j}}}(j,ldpc_{\text{num}})R) / (1 - s_{\beta_{\text{j}}}(j,ldpc_{\text{num}}) + s_{\beta_{\text{j}}}(j,ldpc_{\text{num}})R); \\
    &\text{if isnan}(u_{\beta_{\text{bar}}}(j,i)) \\
    &\quad u_{\beta_{\text{bar}}}(j,i) = -1; \\
    ldpc_{\text{num}} &= ldpc_{\text{num}} + 1;
\end{align*}
\]

for \( j = 1:1:\text{slots} \)
it\% extracting \( u_{\beta} \) values for just message bits

\[
\begin{align*}
    u_{\beta} &= 1; \\
    &\text{for } i = 1:1:L/2 \\
    &\quad U_{\text{Beta}_{\text{Bar}}}(j,u_{\beta}) = u_{\beta_{\text{bar}}}(j,i); \\
    &\quad u_{\beta} = u_{\beta} + 1; \\
    &\text{end} \\
    &\text{for } i = (L/2 + \text{pilots} + 1):1:\text{length}(u_{\beta_{\text{bar}}}(1,:)) \\
    &\quad U_{\text{Beta}_{\text{Bar}}}(j,u_{\beta}) = u_{\beta_{\text{bar}}}(j,i); \\
    &\quad u_{\beta} = u_{\beta} + 1; \\
    &\text{end}
\end{align*}
\]

for \( i = 1:1:\text{slots} \)
\% calculating one fading value per subframe using all the bits in
\% the subframe

\[
\begin{align*}
    C_{\text{hat1}_R}_{\text{itr1}_col}(i,\text{rec_itr}) &= \text{sum}(y_{\text{Dspread}}(i,:).*u_{\beta_{\text{bar}}}(i,:))/(\text{length}(y_{\text{Dspread}}(1,:))); \\
    &\text{end} \\
    &\text{for } j = 1:1:\text{slots} \% calculating noise PSD for each subframe using all the bits in
    &\% the subframe \\
    &\text{for } i = 1:1:\text{length}(y_{\text{info}}(1,:)) \\
    &\quad N_{\text{data}_{\text{itr}}}(j,i) = \text{norm}(y_{\text{info}_{\text{spr}}}(((i-1)*\text{SF} + 1 : i*\text{SF}),j))^2 + \text{norm}(C_{\text{hat1}_R}_{\text{itr1}_col}(j,\text{rec_itr})\times \text{ones}(\text{SF},1))^2 / \text{SF} - \text{sum}(2 \times \text{real}((\text{conj}(y_{\text{info}_{\text{spr}}}(((i-1)*\text{SF} + 1 : i*\text{SF}),j)) \times (C_{\text{hat1}_R}_{\text{itr1}_col}(j,\text{rec_itr})\times \text{ones}(\text{SF},1)) \times \text{Sp_{Code}}\times (U_{\text{Beta}_{\text{Bar}}}(j,i)\times \text{ones}(\text{SF},1)))) / \sqrt{\text{SF}}; \\
    &\quad N0_{\text{HAT}_{\text{itr}}}(j) = \text{sum}(N_{\text{data}_{\text{itr}}}(j,:))/(\text{SF} \times \text{length}(y_{\text{info}}(1,:))); \\
    &\quad N0_{\text{hat1}_R}_{\text{itr1}} = \text{sum}(N0_{\text{HAT}_{\text{itr}}})/\text{slots}; \% calculating noise PSD \\
    &\text{for } i = 1:1:\text{length}(C_{\text{hat1}_R}_{\text{itr1}_col(:,1)}) \\
    &\quad C_{\text{hat1}_R}_{\text{itr1}_row}(1,i) = C_{\text{hat1}_R}_{\text{itr1}_col}(i,\text{rec_itr}); \\
    &\text{end} \\
    N0_{\text{hat}} = N0_{\text{hat1}_R}_{\text{itr1}}; \\
    C_{\text{hat}} = C_{\text{hat1}_R}_{\text{itr1}_col(:,\text{rec_itr})};
\end{align*}
\]
fade_table((itr+2),:)=C_hat;
N0_table((itr+2),:)=N0_hat;
noise_sigma_hat=sqrt(N0_hat/2);
for i=1:1:slots
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat(i)))*abs(C_hat(i)));
end
Yss_Chat=[];
for i=1:1:slots
    Yss_Chat=[Yss_Chat YSS_CHAT(i,:)];
end
f1=1./(1+exp(-2*Yss_Chat/(noise_sigma_hat^2)));% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat,f0,f1,H);
SUCCESS(itr)=success;
S_Beta=Q0; % uB=-1=sB
x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2)); %G=[A|I] so end bits are msg bits
x_hat=x_hat';
delx = x-x_hat;
err = length(find(delx));
errr(itr,ff)=err;
error2(itr,kk)=error2(itr,kk)+err;

for i=1:1:slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else
            s_beta(num,i)=0;
        end
    end
    for i=(L/2+1):1:L
        s_beta(num,i+pilots)=S_beta(num,i);
    end
end
itr = itr+1;
end
Total_bits=frame_index*(G_Row);
j_0 = error1/Total_bits
j_1 = error2(1,:)/Total_bits
j_3 = error2(3,:)/Total_bits
j_5 = error2(5,:)/Total_bits
j_7 = error2(7,:)/Total_bits
j_9 = error2(9,:)/Total_bits

ebn0dB
figure(5)
semilogy(ebn0dB,j_0,'bo-',ebn0dB,j_1,'gx-',ebn0dB,j_3,'mx-',ebn0dB,j_5,'rx-',ebn0dB,j_7,'bx-'
',ebn0dB,j_9,'ro-');
grid on;
legend('j_0','j_1','j_3','j_5','j_7','j_9');
xlabel('Eb/No,dB');
ylabel('BER');
title('EM with C hat & Io updated N_f = 1000, B.S = [1000,2000], N_p = 200');
PROGRAM 5. NON-ADAPTIVE ALGORITHM FOR BPSK MODULATION, JAKES FADING AND MULTIUSER ENVIRONMENT

clc;
clear all;
frame_index=700;
rows=1000;
cols=2000;
q=2;
H1=ldpc_generate(rows,cols,3,2,123); %%% Generate sparse parity check matrix H1
[H,G] = ldpc_h2g(H1,q);
G_Row = size(G,1); % K : size of information bits
G_Col = size(G,2);
ii=0;
err=[];
NumPilot=200; %%%%%%%%%%%%Total number of pilot bits
max_iter=10; %%%%%%%% maximum reciever iterations

%%%%% Eb/N0 loop %%%%%%%%
for EsN0dB =-3:2:25 %dB
    ii=ii+1;
    Es=1;
esn0dB(ii)=EsN0dB;
    EsN0(ii) = 10^(EsN0dB/10); % EbNo
    R=G_Row/G_Col; % rate of encoder
    Eb=Es/R; % data bit energy
    Ecod=R*Eb;
    M=2; % for BPSK modulation
    Es=log2(M)*Ecod; % General relationship between Eb and Es
    SF=8; % Spreading Factor
    E_sf=Es/sqrt(SF);
    N0(ii)=Es/EsN0(ii); % SF*1/Rate *10^(-EbN0dB/10)
    noise_sigma(ii) = sqrt( N0(ii)/(2));
    EbNo(ii)=Eb/N0(ii);
    ebn0dB(ii)=10*log10(EbNo(ii));
end

Rb=100e3; % 10Kbits/sec
Tb=1/Rb;
error1=zeros(1,length(esn0dB));
error2=zeros(max_iter,length(esn0dB));

N0_table(1,:)=N0;
fb_num=1; % counter for fade block number
for kk = 1:length(esn0dB)
    TransmitStartsHere
    intcount=10;
    for ff=1:frame_index
        x=floor(rand(1,G_Row)*2); % generates stream of message bits containing zeros and ones
        mG = mod(x*G,2); % encoding the message bits in code symbols
        slots=50; % number of divisions of each frame
        L=G_Col/slots; % length of each slot
        % division of frame into slots
        for i=1:slots
            xldpc(i,:)=mG((i-1)*L+1:i*L);
        end
        % adding pilots at midamble position in each slot
        pilots=NumPilot/slots;
        for num=1:1:slots
            for i=1:(L/2+pilots);
                if i<=L/2
                    mG_plt(num,i)=xldpc(num,i);
                else
                    mG_plt(num,i)=1;
                end
            end
        end
        for i=(L/2+1):1:L
            mG_plt(num,i+pilots)=xldpc(num,i);
        end
    end
    % concatenating the slots in one row
    mG_pilot=[];
    for i=1:slots
        MG_PLT=mG_plt(i,:);
        mG_pilot=[mG_pilot MG_PLT];
    end
    Tx_data=2*mG_pilot-1;
    Tx_data2 = randsrc(1,length(Tx_data));

% division of frame into slots
for i=1:slots
    xldpc(i,:)=mG((i-1)*L+1:i*L);
end

% adding pilots at midamble position in each slot
pilots=NumPilot/slots;
for num=1:1:slots
    for i=1:(L/2+pilots);
        if i<=L/2
            mG_plt(num,i)=xldpc(num,i);
        else
            mG_plt(num,i)=1;
        end
    end
end
for i=(L/2+1):1:L
    mG_plt(num,i+pilots)=xldpc(num,i);
end
% concatenating the slots in one row
mG_pilot=[];
for i=1:slots
    MG_PLT=mG_plt(i,:);
    mG_pilot=[mG_pilot MG_PLT];
end
Tx_data=2*mG_pilot-1;
Tx_data2 = randsrc(1,length(Tx_data));
codes=hadamard(SF);
Sp_Code=codes(3,:);
Sp_Code2=randsrc(1, SF);

%$$$$$$$$$$$$$$$$$ Spreading Sequence Generation $$$$$$$$$$$$$$$$$%
for i=1:1:length(Tx_data)
    Spread_Tx_data(:,i) = Tx_data(i)*Sp_Code;
    Spread_Tx_data2(:,i) = Tx_data2(i)*Sp_Code2;
end

%$$$$$$$$$$$$$$$ Making Spreaded data a column vector with spreaded data of 1st row of
Tx_data first and followed by others
Spread_Tx_data_col=reshape(Spread_Tx_data,[],1)/sqrt(SF);
Spread_Tx_data_col2=reshape(Spread_Tx_data2,[],1)/sqrt(SF);

%$$$$$$$$$$$$$$$$$ JAKES FADING starts here $$$$$$$$$$$$$$$$$$
% velocity of vehicle
v=300*1000/3600;
v2=100*1000/3600;
vc = 3e8;% velocity of light [m/sec]
fc=2e9;%carrier frequency
fd=v*fc/vc;% maximum doppler shift frequency
fd2=v2*fc/vc;% maximum doppler shift frequency of vehicle 2
N_fb=(L+pilots);% number of bits in one fading block=one code symbol size

%%%%%%%%%%%%%%%%%%%%% Jakes Model Parameter
%%%%%%%%%%%%%%%%%%%%%%%
No = 8;% number of oscillators
N  = 4*No + 2;
alpha = pi/4;
% Other Parameters
k = linspace(1, No, No);
betan = pi*k/No;
wn=2*pi*fd*cos(2*pi*k/N);
wn2=2*pi*fd2*cos(2*pi*k/N);
tmpc=0;
tmps=0;
tmpc2=0;
tmps2=0;
for ii=1:1:((G_Col+NumPilot)/N_fb)
    tmpc=0;
tmps=0;
tmpc2=0;
tmps2=0;
t=fb_num*Tb;%%% synchronizing fading block time with Tb time
for n=1:1:No
    c = cos(betan(n))*cos(wn(n)*t);
    s = sin(betan(n))*cos(wn(n)*t);
    tmpc=tmpc+c;
    tmps=tmps+s;

    c2 = cos(betan(n))*cos(wn2(n)*t);
    s2 = sin(betan(n))*cos(wn2(n)*t);
    tmpc2=tmpc2+c2;
    tmps2=tmps2+s2;
end
xc=(2*tmpc+(sqrt(2)*cos(alpha)*cos(2*pi*fd*t)))/sqrt(2*No);%%%%%%sqrt(No) is to normalize
xs=(2*tmps+(sqrt(2)*sin(alpha)*cos(2*pi*fd*t)))/sqrt(2*(No+1));%%%%and 2 as we have comples fading ie., two components sin and cos.
fade(ii) = xc + sqrt(-1)*xs;

xc2=(2*tmpc2+(sqrt(2)*cos(alpha)*cos(2*pi*fd2*t)))/sqrt(2*No);%%%%%%sqrt(No) is to normalize
xs2=(2*tmps2+(sqrt(2)*sin(alpha)*cos(2*pi*fd2*t)))/sqrt(2*(No+1));%%%%and 2 as we have comples fading ie., two components sin and cos.
fade2(ii) = xc2 + sqrt(-1)*xs2;
fb_num=fb_num+N_fb;
end
fade_table(1,:)=fade;
fade_table2(1,:)=fade2;
chip_no=1;
for ii=1:1:((G_Col+NumPilot)/N_fb)
    for cc=1:1:(SF*N_fb)
        faded_Tx_data(chip_no)=Spread_Tx_data_col(chip_no)*fade(ii);
        faded_Tx_data2(chip_no)=Spread_Tx_data_col2(chip_no)*fade2(ii);
        chip_no=chip_no+1;
    end
end
noise=(sqrt(N0(kk)/2))*((randn(1,length(Spread_Tx_data_col)))+ sqrt(-1)*(randn(1,length(Spread_Tx_data_col))));
abc=rem(intcount,10);
if (abc == 0)
    Y_send=faded_Tx_data+faded_Tx_data2+noise;%%% Y=Hx+n row vector 1 x G_Col*SF
else
    Y_send=faded_Tx_data + noise;
end
intcount=intcount+1;
y_received=reshape(Y_send,[],1); % making column vector of received message

for i=1:1:slots
    Y_rec(i,:)=y_received((i-1)*(L+pilots)*SF+1:i*(L+pilots)*SF);
end

for i=1:1:(G_Col+NumPilot)
    y_dspr(i)=Sp_Code*y_received((i-1)*SF+1:i*SF)/sqrt(SF);
end

for i=1:1:slots
    y_Dspread(i,:)=y_dspr((i-1)*(L+pilots)+1:i*(L+pilots));
end

for j=1:1:slots
    sp=1;
    for i=1:1:L*SF/2
        y_info_spread(j,sp)=Y_rec(j,i);
        sp=sp+1;
    end
    for i=(L*SF/2+pilots*SF+1):1:length(Y_rec(1,:))
        y_info_spread(j,sp)=Y_rec(j,i);
        sp=sp+1;
    end
end
for i=1:1:slots
for j=1:1:length(y_info_spread(1,:))
    y_info_spr(j,i)=y_info_spread(i,j);
end
end

for row=1:1:slots
    col=1;
    for i=L*SF/2+1:1:(L*SF/2+pilots*SF)
        sprd_plt(row,col)=Y_rec(row,i);
        col=col+1;
    end
end

spread_Pilot=[];
for i=1:1:slots
    SP_PLT=sprd_plt(i,:);
    spread_Pilot=[spread_Pilot SP_PLT];
end

for j=1:1:slots
    d_sp=1;
    for i=1:1:L/2
        y_info(j,d_sp)=y_Dspread(j,i);
        d_sp=d_sp+1;
    end
    for i=(L/2+pilots+1):1:length(y_Dspread(1,:))
        y_info(j,d_sp)=y_Dspread(j,i);
        d_sp=d_sp+1;
    end
end

for row=1:1:slots
    col=1;
    for i=L/2+1:1:(L/2+pilots)
Dspread_pilot(row,col)=y_Dspread(row,i);
col=col+1;
end
end

for i=1:1:slots%%calculating initial value of fading using pilots
C_hat1(i)=sum(Dspread_pilot(i,:))/pilots;
end
fade_table(2,:)=C_hat1;

for i=1:1:slots%%calculating values to pass to LDPC encoder
YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat1(i)))*abs(C_hat1(i)));
end
Yss_Chat1=[];%%concatinating these slots values
for i=1:1:slots
    Yss_Chat1=[Yss_Chat1 YSS_CHAT(i,:)];
end
f1=1./(1+exp(-4*Yss_Chat1/(noise_sigma^2)));% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat1,f0,f1,H);
S_Beta=Q0;
Success=success;

x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2)); %G=[A|I] so end bits are mssg bits
"COLUMN Vector
x_hat=x_hat';%making Column vector a ROW vector
delx=x-x_hat;
err=length(find(delx));
error1(:,ff)=err;
error1(kk)=error1(kk)+err;

for i=1:1:slots%%dividing S_beta in slots
    S_beta(i,:)=S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots%%adding pilots s_beta at midamble position
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i)=S_beta(num,i);
        else
            s_beta(num,i)=0;
        end
    end
for i=(L/2+1):1:L
    s_beta(num,i+pilots)=S_beta(num,i);
end
end

for i=1:1:slots
    C_hat(i,1)=C_hat1(1,i);
end

% N0_hat=N0_hat1;
N0_hat=N0;
itr=1;

C_hat1_R_itr1_row=C_hat1;
N0_hat1_itr1=N0;

while ((success == 0) & (itr < (max_iter))),
    for rec_itr=1:1:6 %%%%%EM iteration loop
        for j=1:1:slots
            ldpc_num=1;%%%%%index of bit given by LDPC
            for i=1:1:(L+pilots)%%%%%%%%%calculating u_beta_bar for all sent
codesymbols
                R=exp(-1/N0_hat1_itr1*4*(real(conj(Y_rec(j,((i-1)*SF+1:i*SF))*Sp_Code*C_hat1_R_itr1_row(j)/sqrt(SF)))));%%%%%calculating R_beta
                if R==inf
                    R=100000000;
                end
                u_beta_bar(j,i)=(1-s_beta(j,ldpc_num)-s_beta(j,ldpc_num)*R)/(1-s_beta(j,ldpc_num)+s_beta(j,ldpc_num)*R);
            end
        end
    end
    C_hat1_R_itr1_row=C_hat1;
    N0_hat1_itr1=N0;
end

for j=1:1:slots%%extracting u_beta values for just message bits
    u_b=1;
    for i=1:1:L/2
        u_b=1;
        for i=1:1:L/2
U_Beta_Bar(j,u_b)=u_beta_bar(j,i);
u_b=u_b+1;
end
for i=(L/2+pilots+1):1:length(u_beta_bar(1,:))
    U_Beta_Bar(j,u_b)=u_beta_bar(j,i);
u_b=u_b+1;
end
end

for i=1:1:slots%% calculating one fading value per subframe using all the bits in
the subframe

C_hat1_R_itr1_col(i,rec_itr)=sum(y_Dspread(i,:).*u_beta_bar(i,:))/(length(y_Dspread(1,:)));
end
for i=1:1:length(C_hat1_R_itr1_col(:,1))
    C_hat1_R_itr1_row(1,i)=C_hat1_R_itr1_col(i,rec_itr);
end
N0_hat1_R_itr1=N0;
N0_hat=N0_hat1_R_itr1;
C_hat=C_hat1_R_itr1_col(:,rec_itr);
fade_table((itr+2),:)=C_hat;
N0_table((itr+2),:)=N0_hat;
oise_sigma_hat=sqrt(N0_hat/2);
for i=1:1:slots
    YSS_CHAT(i,:)=real(y_info(i,:)*exp(-sqrt(-1)*angle(C_hat(i)))*abs(C_hat(i)));
end
Yss_Chat=[];
for i=1:1:slots
    Yss_Chat=[Yss_Chat YSS_CHAT(i,:)];
end
f1=1./((1+exp(-2*Yss_Chat/(noise_sigma^2))));% likelihoods prob of sending '1'
f0=1-f1;
[z_hat, success, k, Q0, Q1] = avi_ldpc_decode(Yss_Chat,f0,f1,H);
SUCCESS(itr)=success;
S_Beta=Q0;
x_hat = z_hat(size(G,2)+1-size(G,1):size(G,2)); %G=[A|I] so end bits are mssg bits
x_hat=x_hat';
delx = x-x_hat;
err = length(find(delx));
errr(itr,ff)=err;
error2(itr,kk)=error2(itr,kk)+err;
for i=1:1:slots
    S_beta(i,:) = S_Beta((i-1)*L+1:i*L);
end
for num=1:1:slots
    for i=1:1:(L/2+pilots);
        if i<=L/2
            s_beta(num,i) = S_beta(num,i);
        else
            s_beta(num,i) = 0;
        end
    end
    for i=(L/2+1):1:L
        s_beta(num,i+pilots) = S_beta(num,i);
    end
    itr = itr+1;
end
end
end

Total_bits = frame_index*(G_Row);
j_0 = error1/Total_bits
j_1 = error2(1,:)/Total_bits
j_3 = error2(3,:)/Total_bits
j_5 = error2(5,:)/Total_bits
j_7 = error2(7,:)/Total_bits
j_9 = error2(9,:)/Total_bits
ebn0dB
figure(15)
semilogy(ebn0dB,j_0,'bo-',ebn0dB,j_1,'gx-',ebn0dB,j_3,'mx-',ebn0dB,j_5,'rx-',ebn0dB,j_7,'bx-',ebn0dB,j_9,'ro-');
grid on;
legend('j_0','j_1','j_3','j_5','j_7','j_9');
xlabel('Eb/No,dB');
ylabel('BER');
title('EM with C hat & Io N_f = 1000, B.S = [1000,2000], N_p = 200');