2012

Adaptive Channel Estimation for Turbo Decoding

YU QING GUO
University of Windsor

Adaptive Channel Estimation for Turbo Decoding

by

Yuqing Guo

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2012

Adaptive Channel Estimation for Turbo Decoding

by

Yuqing Guo

APPROVED BY:

_____

Dr. Luis Rueda
School of Computer Science

_____

Dr. Huapeng Wu
Department of Electrical and Computer Engineering

_____

Dr. Behnam Shahrrava, Advisor
Department of Electrical and Computer Engineering

_____

Dr. Chunhong Chen, Chair of Defense
Department of Electrical and Computer Engineering

April 30, 2012

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

A new adaptive filter is proposed for the turbo decoding on Rayleigh fading channels with noisy channel estimates. The turbo decoder that is used over Rayleigh fading channels is exactly the same as the one used on Additive White Gaussian Noise (AWGN) channel. The turbo decoder works very well on AWGN channel [1]-[2], but not as well on Rayleigh fading channels at that time. In [5], the author assumes there already exists a fading channel estimator with some estimation errors and develops a new channel reliability factor and new decision variables for turbo decoding on Rayleigh fading channels. Hence, Frenger, the author of [5] improved the performance of turbo decoding over Rayleigh fading channels. Since then, most research has focused on the channel estimation to reduce the error variances of estimating. However, the extrinsic information generated from the turbo decoder has some priority information about the transmitted data bits, which can help us better understand the channel characters. In this thesis, by using the soft extrinsic information after each iteration of decoding, we re-estimate the channel and the minimum mean square error (m.m.s.e.) and further update the channel reliability factor and decision variables at each iteration. Simulations show that signal to noise (SNR) gain is improved by up to about 1dB at bit error probability of 3.5×10^{-4}.

ACKNOWLEDGEMENTS

I would like to thank all of those who made this research possible: especially my advisor, Dr. Behnam Shahrrava, for his kindness guidance and advice at every step of this research. I would also like to thank the committee members, Dr. Luis Rueda and Dr. Huapeng Wu, for their comments and encouragement. I would also like to thank my wife Ying and my son Zhihao for their understanding and support of my study.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Review of the literature

Turbo codes, introduced in [1], have been proven to perform remarkably well on additive white Gaussian noise (AWGN) channels [1], [2]. The performance of the turbo codes on Rayleigh fading channels has also been studied since then [3] - [6]. In [6], the author, Frenger, gave out the exact decoding metric for binary phase-shift keying (BPSK) signalling on Rayleigh fading channels by assuming that there is a channel estimator prior to the turbo decoder to provide us with an unbiased channel estimate with a certain error variance. The conventional turbo decoding metric on AWGN channels needs the estimation of signal to noise ratio (SNR) [7]. The exact turbo decoding metric over Rayleigh fading channels needs both SNR and the channel fading factors [3], [8]. However, the channel parameters are assumed to be known by Frenger in [6]. Since then, many researches have focused on the estimation of channel parameters and the degradation caused by errors in these parameters, while there is not much research directly working on the results of Frenger in [5] and [6]. This could be seen from the number of citations in *IEEE*: [5] is only cited twice [9], while [6] is cited thirteen times so far [10], [12].

The effect of SNR mismatch on the performance of the turbo decoding has been studied in several works. Some research has been proposed for integrating the estimation process into the turbo decoder over fading channels [9], [11]. In [9], a modified version of Wiener filtering with initial pilot symbols is proposed, and the bit error rate (BER) performance has been improved by 0.5dB at BER of 10^{-3}, comparing to the Wiener

filtering algorithm with initial pilot symbols. In [10], the exact turbo decoding metric is simplified. The BER performance is between that of the conventional decoding metric and the exact decoding metric, but is very close to the BER performance of the exact decoding metric. In [11], an in-service estimation of the channel reliability factor is proposed, which uses the statistical computations of the block observations to get BER performance similar to the exact decoding metric in [6]. In [13], they do not use the fixed iterations with the turbo decoder, while they do use adaptive iterations for speeding up the decoding process by aiming at a fixed BER. Once the aimed BER, say10^{-4}, is reached, no further iterations for the turbo decoder are needed. All these estimation schemes can be seen as pilot symbol aided modulation (PSAM) or as blind channel estimation methods. Most of these estimation methods ignore the feedback from the turbo decoder. However, the extrinsic information generated during turbo decoding process has some priori information about the transmitted data bits, which can help us refine the channel fading factors.

In [12], a novel idea has been proposed for integrating the extrinsic information from the turbo decoder to re-estimate the fading channel. However, a mistake is made during the mathematical derivation approach. There is no relation between the re-estimate of the fading channel and the extrinsic information as expected. So an incorrect method is used to make such a connection, which is to approximate the new channel estimate and its error variance by taking their expected value on coded input data bits .There is no mathematical reason to support this kind of approximation.

1.2 A new adaptive algorithm for turbo decoding

In this thesis, based on the results in [6] and [12], we propose a new adaptive channel estimation algorithm for turbo decoding on Rayleigh fading channels. The mistake in [12] is corrected. However, the experiment does not go positively as expected after the correction. The results of the experiment show that the extrinsic information generated during the decoding process is not totally reliable. The extrinsic information of some bits is helpful to the channel re-estimation, while the others are not. Future researchers should pay attention to this point, avoiding unnecessary repeated experiments. The adaptive decoding metric proposed by this thesis has successfully overcome this problem by utilizing an effective stop-and-go strategy at the implementation stage as a selecting criterion. In addition to that, the steep-decent algorithm of Newton's method is used to co-operate with the iterative nature of the turbo decoder. The varying step size is also adopted to achieve faster convergence.

The observations received by the turbo decoder have two parts: the systematic portion and parity portion. The proposed adaptive filter takes only the systematic observations and the soft extrinsic information, which is the feedback from the turbo decoder, as its inputs. This is one of the unique choices of this thesis. Some research takes the hard decision as the input of the adaptive channel filter for only the amplitude estimation [14], while some takes only the soft information as the input of the adaptive filter for SNR estimation [13]. None of them split up the observations into two parts. The conventional decoding algorithm that is used for AWGN channels is unchanged in this thesis. However, the exact decoding metric that is derived by Frenger in [6] is updated iteratively during the turbo decoding process.

The proposed adaptive filter works better when the block size of the information gets smaller or the estimation errors of the channel estimator in [6] get bigger. The gain of using the proposed adaptive filter is about 1dB at the bit error probability of 3.5×10^{-4} with some settings. This gain is obtained with minimally increased complexity.

1.3 Organization of the thesis

The organization of this thesis is as follows: In Chapter 2, the basic elements of a digital system and the channel models are introduced. In Chapter 3, the turbo encoder and turbo decoding algorithm are reviewed. In Chapter 4, we propose an adaptive filter that uses the soft information to update the exact turbo decoding metric iteratively over Rayleigh fading channels. In Chapter 5, simulation results are presented. Finally, conclusions and future research directions are given in Chapter 6. The whole Matlab scripts of the proposed adaptive filter and the turbo encoder and decoder are presented at the end as an Appendix.

CHAPTER II

CHANNEL MODELS

To design a channel estimator and analyze the performance of turbo decoding algorithms, we need to understand the channels that the transmitted data experiences. The concept of the basic digital communication systems and two channel models are needed to discuss our contributions

2.1 Basic elements of digital communication systems

The demand for efficient and reliable digital communication systems has rapidly increased in recent years. It is necessary to minimize bit error probability at the receiver end for higher quality communication. A block diagram of a digital communication system is shown in Figure 1 [15].

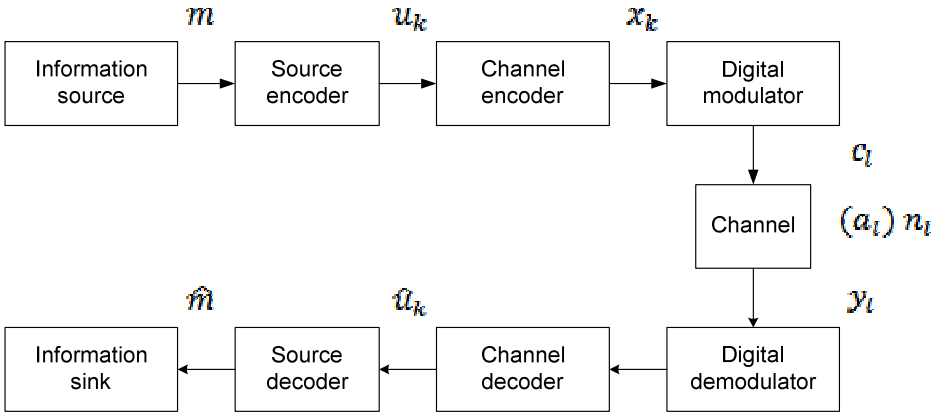

Figure 1 Block diagram of a digital communication system

The information source usually contains redundancy. The source encoder removes the redundancy of the information to achieve efficiency. The source encoder changes source information to information sequences. Then the channel encoder adds redundancy to the information sequences in a controlled way to increase communication reliability. Then the digital modulator transforms coded bits into a continuous time waveform, which

is suitable for a physical channel. The transmitted bits will be distorted randomly both in amplitude and phase due to many factors, such as reflection, refraction, multipath…

At the receiver end, the digital demodulator produces an estimation of the transmitted data. The channel decoder uses the redundancy and knowledge of the channel code to detect and correct errors. Finally the source decoder reconstructs the original information by using knowledge of the source encoding method.

In this thesis, the main concern is channel decoding for binary phase shift keying (BPSK) signalling over the Rayleigh fading channel.

2.2 Channel models

For better understanding of decoding strategies over the Rayleigh fading channel, we first need to introduce the additive white Gaussian noise (AWGN) channel model, and then the fading channel model.

2.2.1 AWGN channel

The AWGN channel model, together with BPSK modulator, is shown in Figure 2. Where $x_k \in (0,1)$ are coded data bits. The coded data (systematic bits and parity bits) are inputs to a BPSK modulator, which generates the transmitted channel symbols $c_l \in \left(-\sqrt{E_s}, \sqrt{E_s}\right)$. In an AWGN channel, Gaussian distributed random noise, n_l, with zero mean is added to the transmitted symbols. The variance of n_l is:

$$E[n_l] = \sigma_n^2 = \frac{N_0}{2} \tag{2.1}$$

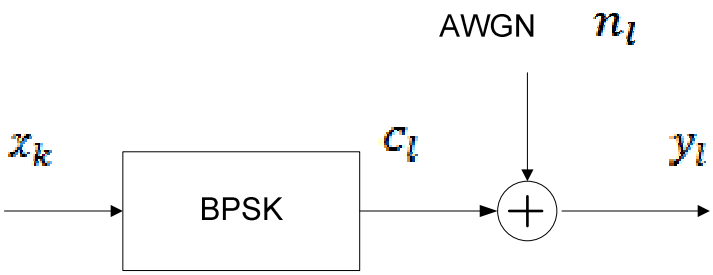

Figure 2 AWGN channel model

The signal to noise ratio (SNR) is:

$$\frac{E_b}{N_0} = \frac{E_s}{r \times 2\sigma_n^2} \tag{2.2}$$

where E_b is the energy per information bit, E_s is the energy per actual transmitted symbol, r is the coding rate, and we have the relationship between the energies and the code rate,

$$\frac{E_s}{E_b} = r \tag{2.3}$$

At the receiver end, we have,

$$y_l = c_l + n_l \tag{2.4}$$

2.2.2 Rayleigh fading channel

The Rayleigh fading channel is a statistic model mostly used by wireless system. The Rayleigh fading channel with independent additive white Gaussian noise and a BPSK modulator is shown in Figure 3. Each of the channel symbols, c_l, is transmitted on such model. At the receiver end, we have [16],

$$y_l = a_l c_l + n_l \tag{2.5}$$

where the noise n_l and the channel coefficient a_l are complex valued, Gaussian distributed random variables with zero mean that are independent of each other.

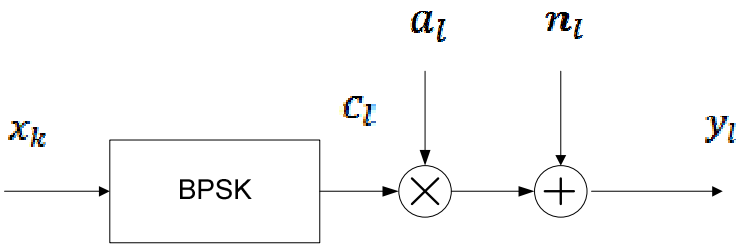

Figure 3 Fading channel model

The variances of a_l and n_l are

$$\begin{aligned} E[|a_l|^2] &= 2\sigma_a^2, \\ E[|n_l|^2] &= 2\sigma_n^2 \end{aligned} \tag{2.6}$$

At the receiver we need to know both the amplitude and phase distortion. Such analysis is more complex than the analysis of the AWGN channel model. We can express the complex valued channel coefficient a_l as follows,

$$a_l = a_{lr} + ja_{li} = r_l e^{j\theta_l} \tag{2.7}$$

The amplitude and phase probability density function (pdf) of the channel coefficient a_l is [17]:

$$\begin{aligned} f(r_l) &= \frac{r_l}{\sigma_a^2} e^{-\frac{r_l^2}{2\sigma_a^2}} \\ f(\theta_l) &= \frac{1}{2\pi} \quad [0, 2\pi] \end{aligned} \tag{2.8}$$

where the amplitude is Rayleigh distributed and the phase is uniform distributed.

CHAPTER III

TURBO CODES AND DECODING

Turbo codes with maximum a posteriori (MAP) algorithm have been proven to perform extraordinary well on AWGN channels [1], [2]. Turbo decoding on Rayleigh fading channels has also been studied in [5], [6]. In this chapter, we first introduce the concept of the Turbo encoder, then briefly review turbo decoding over AWGN channels and Rayleigh fading channels separately.

3.1 Turbo encoder

Normally, a Turbo encoder [1] consists of two recursive systematic convolution (RSC) encoders in parallel, separated by a random interleaver (I). The information sequences are sent to the first encoder directly, while the second encoder receives the interleaved information sequences. For code rate $r = 1/3$, there is no puncturing, the code words are $(x_l^s, x_l^{1p}, x_l^{2p} \cdots)$. We could puncture the code words to achieve a higher code rate of ½. In this case, the output code words are $(x_l^s, x_l^{1p}, x_{l+1}^s, x_{l+1}^{2p} \cdots)$.

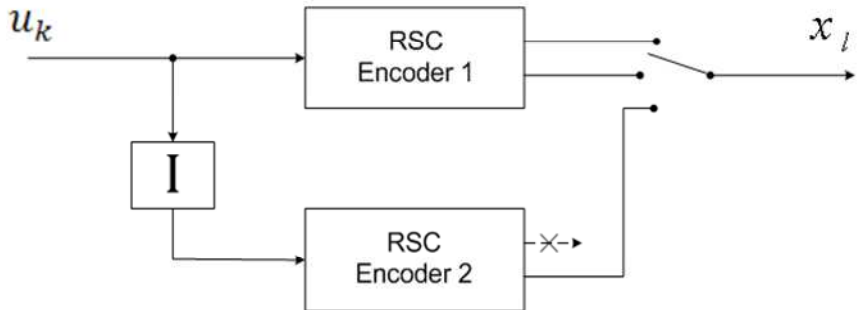

Figure 4 Turbo encoder

A typical RSC encoder is depicted in Figure 5, where the d_k is calculated as:

$$d_k = u_k + \sum_{i=1}^{K-1} g_{1i} d_{k-i} \tag{3.1}$$

The corresponding code words are (x_k^s, x_k^p),

$$\begin{cases} x_k^s = u_k \\ x_k^p = d_k + \sum_{i=1}^{K-1} g_{2i} d_{k-i} \end{cases} \tag{3.2}$$

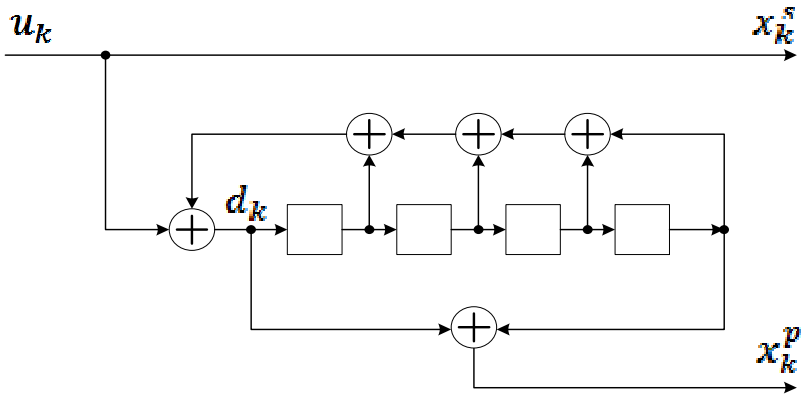

Figure 5 RSC encoder

where the feedback generator is $g_{1i} = (11111)$, and the forward generator is $g_{2i} = (10001)$. They correspond to octal notation $g_{1i} = 37$, and $g_{2i} = 21$.

3.2 Maximum *a posteriori* (MAP) algorithm over AWGN channel

MAP is the optimal symbol-by-symbol maximum *a posteriori* probability algorithm [18]. However, MAP is not practical for implementation, primarily because of the complexity associated with the representation of the probabilities. Log-MAP is a transform of MAP, and works in the logarithmic domain, which has equivalent performance and is more practical. We review the fundamentals of MAP/Log-MAP below, which are thoroughly discussed in [16] and [19].

For an information sequence of length *N*, we have $\vec{u} = (u_1, u_2, ..., u_N)$, where $u_i \in (0,1)$, and for the corresponding coded output sequence, we have $\vec{c} = (\vec{c}_1, \vec{c}_2, ..., \vec{c}_N)$, where the length of $\vec{c}_i$ is n for a code rate of $r = 1/n$. We denote the encoder state at time i is m_i. We know that the output and the current state of the convolutional code encoder depend on the previous state and input, so we have the functions:

$$\vec{c}_i = f_c(u_i, m_{i-1}) \tag{3.3}$$

$$m_i = f_s(u_i, m_{i-1}) \tag{3.4}$$

It is clear that any state pair (m_i, m_{i-1}) corresponds to either $u_i = 0$ or $u_i = 1$. Hence, we have two sets of state pairs S_0 and S_1, corresponding to $u_i = 0$ and $u_i = 1$. Based on observations at the receiver, $\vec{y} = (\vec{y}_1, \vec{y}_2, ..., \vec{y}_N)$, we can apply the MAP rule to find log-likelihood L values as:

$$\begin{aligned} L(u_i) &= \ln \frac{P(u_i = 1 \mid \vec{y})}{P(u_i = 0 \mid \vec{y})} = \ln \frac{P(u_i = 1, \vec{y})}{P(u_i = 0, \vec{y})} \\ &= \ln \frac{P(S_1, \vec{y})}{P(S_0, \vec{y})} = \ln \frac{\sum_{S_1} P(m_{i-1}, m_i, \vec{y})}{\sum_{S_0} P(m_{i-1}, m_i, \vec{y})} \end{aligned} \tag{3.5}$$

We define $\vec{y}_i^{(j)} = (\vec{y}_i ... \vec{y}_j)$, where $i \le j$. Then we can write

$$\vec{y} = (\vec{y}_1^{(i-1)}, \vec{y}_i, \vec{y}_{i+1}^{(N)}) \tag{3.6}$$

and we have

$$
\begin{aligned}
p(m_{i-1}, m_i, \vec{y}) &= p(m_{i-1}, m_i, \vec{y}_1^{(i-1)}, \vec{y}_i, \vec{y}_{i+1}^{(N)}) \\
&= p(m_{i-1}, m_i, \vec{y}_1^{(i-1)}, \vec{y}_i)\, p(\vec{y}_{i+1}^{(N)} \mid m_{i-1}, m_i, \vec{y}_1^{(i-1)}, \vec{y}_i) \\
&= p(m_{i-1}, \vec{y}_1^{(i-1)})\, p(m_i, \vec{y}_i \mid m_{i-1}, \vec{y}_1^{(i-1)})\, p(\vec{y}_{i+1}^{(N)} \mid m_{i-1}, m_i, \vec{y}_1^{(i-1)}, \vec{y}_i) \\
&= p(m_{i-1}, \vec{y}_1^{(i-1)})\, p(m_i, \vec{y}_i \mid m_{i-1})\, p(\vec{y}_{i+1}^{(N)} \mid m_i) \\
&= \alpha_{i-1}(m_{i-1})\gamma_i(m_{i-1}, m_i)\beta_i(m_i)
\end{aligned}
\tag{3.7}
$$

where the first three steps follow from the chain rule, the forth step follows from Markov properties [20], and the last step we define $\alpha_{i-1}(m_{i-1})$, $\beta_i(m_i)$, and $\gamma_i(m_{i-1}, m_i)$ as follows:

$$
\begin{aligned}
\alpha_{i-1}(m_{i-1}) &= p(m_{i-1}, \vec{y}_1^{(i-1)}) \\
\beta_i(m_i) &= p(\vec{y}_{i+1}^{(N)} \mid m_i) \\
\gamma_i(m_{i-1}, m_i) &= p(m_i, \vec{y}_i \mid m_{i-1})
\end{aligned}
\tag{3.8}
$$

Hence the log-likelihood, L, becomes:

$$
\begin{aligned}
L(u_i) &= \ln \frac{\sum_{(m_{i-1}, m_i) \in S_1} p(m_{i-1}, m_i, \vec{y})}{\sum_{(m_{i-1}, m_i) \in S_0} p(m_{i-1}, m_i, \vec{y})} \\
&= \ln \frac{\sum_{(m_{i-1}, m_i) \in S_1} \alpha_{i-1}(m_{i-1})\gamma_i(m_{i-1}, m_i)\beta_i(m_i)}{\sum_{(m_{i-1}, m_i) \in S_0} \alpha_{i-1}(m_{i-1})\gamma_i(m_{i-1}, m_i)\beta_i(m_i)}
\end{aligned}
\tag{3.9}
$$

We can compute $\alpha_i(m_i)$ forward recursively as following:

$$
\begin{aligned}
\alpha_i(m_i) &= p(m_i, \vec{y}_1^{(i)}) \\
&= \sum_{S_{m_{i-1}}} p(m_{i-1}, m_i, \vec{y}_1^{(i-1)}, \vec{y}_i) \\
&= \sum_{S_{m_{i-1}}} p(m_{i-1}, \vec{y}_1^{(i-1)})\, p(m_i, \vec{y}_i \mid m_{i-1}, \vec{y}_1^{(i-1)}) \\
&= \sum_{S_{m_{i-1}}} p(m_{i-1}, \vec{y}_1^{(i-1)})\, p(m_i, \vec{y}_i \mid m_{i-1}) \\
&= \sum_{S_{m_{i-1}}} \alpha_{i-1}(m_{i-1})\gamma_i(m_{i-1}, m_i)
\end{aligned}
\tag{3.10}
$$

assuming that all initial states have a value of zero, that is

$$\alpha_0(m_0) = \begin{cases} 1 & m_0 = 0 \\ 0 & m_0 \neq 0 \end{cases} \tag{3.11}$$

And we compute $\beta_{i-1}(m_{i-1})$ backward recursively as:

$$\begin{aligned} \beta_{i-1} &= p(\vec{y}_i^{(N)} \mid m_{i-1}) \\ &= \sum_{S_{m_i}} p(\vec{y}_i, \vec{y}_{i+1}^{N}, m_i \mid m_{i-1}) \\ &= \sum_{S_{m_i}} p(m_i, \vec{y}_i \mid m_{i-1}) p(\vec{y}_{i+1}^{(N)} \mid m_i, \vec{y}_i, m_{i-1}) \\ &= \sum_{S_{m_i}} p(m_i, \vec{y}_i \mid m_{i-1}) p(\vec{y}_{i+1}^{(N)} \mid m_i) \\ &= \sum_{S_{m_i}} \gamma_i(m_{i-1}, m_i) \beta_i(m_i) \end{aligned} \tag{3.12}$$

assuming that the trellis is terminated in the all-zero state. Hence,

$$\beta_N(m_N) = \begin{cases} 1 & m_N = 0 \\ 0 & m_N \neq 0 \end{cases} \tag{3.13}$$

We compute $\gamma_i(m_{i-1}, m_i)$ as follows:

$$\begin{aligned} \gamma_i(m_{i-1}, m_i) &= p(m_i, \vec{y}_i \mid m_{i-1}) \\ &= p(m_i \mid m_{i-1}) p(\vec{y}_i \mid m_i, m_{i-1}) \\ &= P(u_i) p(\vec{y}_i \mid u_i) \\ &= P(u_i) p(\vec{y}_i \mid \vec{c}_i) \end{aligned} \tag{3.14}$$

The expression clearly shows that $\gamma(m_{i-1}, m_i)$ depends on the prior probability of the information at time i, and the channel characteristics.

3.3 Turbo decoding over AWGN channel

For an AWGN channel, we have $y_l = c_l + n_l$. Let us consider the special case when code rate $r = 1/2$, and the systematic convolution code uses BPSK modulation. Under such a condition we have $\vec{y}_i = (y_i^s, y_i^p)$ and $\vec{c}_i = (c_i^s, c_i^p)$, where s and p represent systematic bit and parity bit, respectively. In order to calculate the log-likelihood L, we

need first to calculate the branch metric $\gamma_i(m_{i-1}, m_i)$ .According to formula (3.14), we further need to calculate the probability of $p(\vec{y}_i \mid \vec{c}_i)$.

The pdf of y given c could be calculated through its cumulative distribution function (CDF) as follows:

$$p(\vec{y}_i \mid \vec{c}_i) = \frac{\partial F_{y_l|c_l}(\vec{y}_i \mid \vec{c}_i)}{\partial \vec{y}_i} \tag{3.15}$$

while

$$\begin{aligned} F_{y_l|c_l}(\vec{y}_i \mid \vec{c}_i) &= P(y_l \le \vec{y}_i \mid c_l = \vec{c}_i) \\ &= P(c_l + n_l \le \vec{y}_i \mid c_l = \vec{c}_i) \\ &= P(n_l \le \vec{y}_i - c_l \mid c_l = \vec{c}_i) \\ &= P(n_l \le \vec{y}_i - \vec{c}_i) \\ &= \int_{-\infty}^{\vec{y}_i - \vec{c}_i} f_N(\alpha) d\alpha \end{aligned} \tag{3.16}$$

Therefore, we get:

$$\begin{aligned} p(\vec{y}_i \mid \vec{c}_i) &= \frac{\partial F_{y_l|c_l}(\vec{y}_i \mid \vec{c}_i)}{\partial \vec{y}_i} \\ &= \frac{\partial}{\partial \vec{y}_i} \int_{-\infty}^{\vec{y}_i - \vec{c}_i} f_N(\alpha) d\alpha \\ &= f_N(\alpha) \mid_{-\infty}^{\vec{y}_i - \vec{c}_i} = f_N(\vec{y}_i - \vec{c}_i) - f_N(-\infty) \\ &= f_N(\vec{y}_i - \vec{c}_i) - 0 \\ &= f_N(\vec{y}_i - \vec{c}_i) \end{aligned} \tag{3.17}$$

where $f_N(\alpha)$ is the pdf of the AWGN channel. So the branch metric is:

$$\begin{aligned} \gamma(m_i, m_{i-1}) &= P(u_i)\, p(\vec{y}_i \mid \vec{c}_i) \\ &= \frac{P(u_i)}{\pi N_0} \exp\left(-\frac{(y_i^s - c_i^s)^2 + (y_i^p - c_i^p)^2}{N_0} \right) \\ &= \frac{1}{\pi N_0} \exp\left(-\frac{(y_i^s)^2 + (y_i^p)^2 + (c_i^s)^2 + (c_i^p)^2}{N_0} \right) P(u_i) \exp\left(\frac{2 y_i^s c_i^s + 2 y_i^p c_i^p}{N_0} \right) \end{aligned} \tag{3.18}$$

Because of BPSK modulation, the term $\frac{1}{\pi N_0}\exp\left(-\frac{(y_i^s)^2+(y_i^p)^2+(c_i^s)^2+(c_i^p)^2}{N_0}\right)$

is independent of u_i, and it could be cancelled from the numerator and the denominator of the log-likelihood L values in the formula (3.9), as follows:

$$\begin{aligned}
L(u_i) &= \ln\frac{\sum_{(m_{i-1},m_i)\in S_1}\alpha_{i-1}(m_{i-1})P(u_i)\exp\left(\frac{2y_i^s c_i^s+2y_i^p c_i^p}{N_0}\right)\beta_i(m_i)}{\sum_{(m_{i-1},m_i)\in S_0}\alpha_{i-1}(m_{i-1})P(u_i)\exp\left(\frac{2y_i^s c_i^s+2y_i^p c_i^p}{N_0}\right)\beta_i(m_i)} \\
&= \frac{4\sqrt{E_c}}{N_0}y_i^s + \ln\frac{P(u_i=1)}{P(u_i=0)} + \ln\frac{\sum_{(m_{i-1},m_i)\in S_1}\alpha_{i-1}(m_{i-1})P(u_i)\exp\left(\frac{2y_i^p c_i^p}{N_0}\right)\beta_i(m_i)}{\sum_{(m_{i-1},m_i)\in S_0}\alpha_{i-1}(m_{i-1})P(u_i)\exp\left(\frac{2y_i^p c_i^p}{N_0}\right)\beta_i(m_i)} \qquad (3.19) \\
&= L_c y_i^s + L_a(u_i) + L_e(u_i)
\end{aligned}$$

where we define L_c as the channel reliability factor [7], $L_a(u_i)$ is a priori information, and $L_e(u_i)$ as the extrinsic information of the systematic bit y_i^s, which is dependent on the received parity bits.

$$\begin{aligned}
L_c &= \frac{4\sqrt{E_c}}{N_0} \\
L_a(u_i) &= \ln\frac{P(u_i=1)}{P(u_i=0)} \qquad (3.20) \\
L_e(u_i) &= \ln\frac{\sum_{(m_{i-1},m_i)\in S_1}\alpha_{i-1}(m_{i-1})P(u_i)\exp\left(\frac{2y_i^p c_i^p}{N_0}\right)\beta_i(m_i)}{\sum_{(m_{i-1},m_i)\in S_0}\alpha_{i-1}(m_{i-1})P(u_i)\exp\left(\frac{2y_i^p c_i^p}{N_0}\right)\beta_i(m_i)}
\end{aligned}$$

For turbo decoding, corresponding to the turbo encoder, we have two MAP decoders, DEC1 and DEC2, which iteratively exchange extrinsic information as a priori probability of each other. A de-multiplexer at beginning changes the received serial data

bits y_l into parallel data bits (y^s, y^{1p}, y^{2p}). Corresponding to the turbo encoder, the received systematic bits y^s and parity bits y^{1p} are sent to the first MAP decoder, which is depicted as DEC1 in Figure 6. The interleaved systematic data bits $y^s(i)$ and the received parity data bits y^{2p}, which are already interleaved at the turbo encoder, are sent to the second MAP decoder, which is DEC2 depicted in Figure 6.

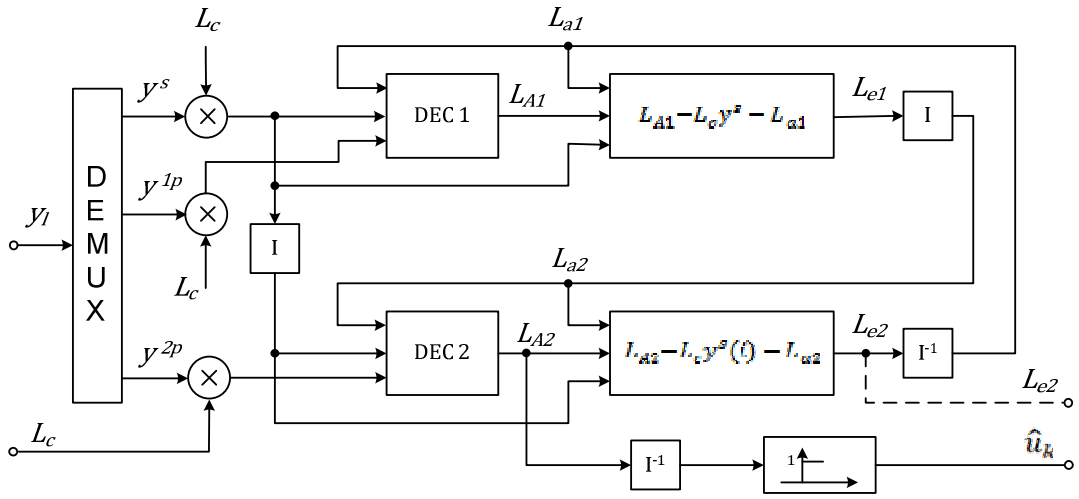

Figure 6 Turbo decoder

At the first iteration, we do not have the extrinsic information yet, assuming all bits are equiprobable, and so set a priori probability value L_{a1} to zero. Thus we get the first extrinsic information $L_{e1}^{(1)}$ from the first MAP decoder; the superscript represents the iteration number of the decoding process.

$$\begin{aligned} L_{e1}^{(1)} &= L_{A1}^{(1)} - L_c y^s - L_{a1}^{(1)} = L_{A1}^{(1)} - L_c y^s - 0 \\ &= L_{A1}^{(1)} - L_c y^s \end{aligned} \tag{3.21}$$

The first extrinsic information $L_{e1}^{(1)}$ after interleaved, becoming $L_{e1}^{(1)}(I)$, is sent to the second MAP decoder. The second MAP decoder will take the extrinsic information

from the first MAP decoder as its a priori probability L_{a2} of the transmitted data bits. In the second decoder, after decoding, we get a new extrinsic information $L_{e2}^{(1)}$:

$$\begin{aligned} L_{e2}^{(1)} &= L_{A2}^{(1)} - L_c y^s(I) - L_{a2}^{(1)} \\ &= L_{A2}^{(1)} - L_c y^s(I) - L_{e1}^{(1)}(I) \end{aligned} \tag{3.22}$$

The $L_{e2}^{(1)}$, after de-interleaved, becoming $L_{e2}^{(1)}(I^{-1})$, is fed back to the first MAP decoder as its a priori information of the next iteration.

$$\begin{aligned} L_{e1}^{(2)} &= L_{A1}^{(2)} - L_c y^s - L_{a1}^{(2)} \\ &= L_{A1}^{(2)} - L_c y^s - L_{e2}^{(1)}(I^{-1}) \end{aligned} \tag{3.23}$$

The general formula for the extrinsic information is as follows:

$$\begin{aligned} L_{e1}^{(i)} &= L_{A1}^{(i)} - L_c y^s - L_{a1}^{(i)} = L_{A1}^{(i)} - L_c y^s - L_{e2}^{(i-1)}(I^{-1}) \\ L_{e2}^{(i)} &= L_{A2}^{(i)} - L_c y^s(I) - L_{a2}^{(i)} = L_{A2}^{(i)} - L_c y^s(I) - L_{e1}^{(i)}(I) \end{aligned} \tag{3.24}$$

with the number of iterations $i \geq 1$, and $L_{a1}^{(1)} = L_{e2}^{(0)} = 0$. The capital letter I in brackets represents the interleaver and de-interleaver with a negative power of 1. The whole decoding process runs iteratively for the given times to improve the decoding performance.

The upper part and the lower part of the turbo decoder is identical, except that every piece of information that goes through the lower part must be interleaved and the output of the lower part must be de-interleaved before using.

At the end of the iterative decoding process, we can make a decision $\hat{u}_k$ by comparing $L_{A2}(u_k)$ to a threshold equal to zero,

$$\begin{aligned} \hat{u}_k &= 1 \quad if \quad L_{A2}(u_k) \geq 0, \\ \hat{u}_k &= 0 \quad if \quad L_{A2}(u_k) < 0, \end{aligned} \tag{3.25}$$

From Figure 6, we see that two pieces of information are needed by the turbo decoder, channel reliability factor L_c and observations y_l or decision variables.

3.4 Turbo decoding over the Rayleigh fading channel

The structure of the turbo decoder over the Rayleigh fading channel is identical to that over the AWGN channel. From the point of view of the turbo decoder, we still need two pieces of information, i.e., the channel reliability factor and decision variables. However, due to the different channel models, we need to modify those two pieces of information. In [6], the author assumes that there already exists a channel estimator, in this thesis we call it channel estimator (1). The channel estimator (1), h_l, is modeled as

$$h_l = a_l + m_l \tag{3.26}$$

where m_l is the estimate error of the Rayleigh fading channel a_l, which is complex valued and Gaussian distributed with

$$\begin{aligned} E[m_l] &= 0 \\ E[|m_l|^2] &= 2\sigma_m^2 \end{aligned} \tag{3.27}$$

The estimate error of the estimator (1), m_l, is independent of the channel a_l. Further we have:

$$\begin{aligned} E[h_l] &= E[a_l + m_l] = E[a_l] + E[m_l] = 0 \\ E[|h_l|^2] &= E[|a_l + m_l|^2] = E[|a_l|^2] + E[|m_l|^2] \end{aligned} \tag{3.28}$$

In [6], the author gives the new decision variables z_l based on the received bits y_l and the estimation h_l of the channel estimator (1):

$$z_l = y_l h_l^* \tag{3.29}$$

Further, the cross correlation coefficient of y_l and h_l is defined as follows:

$$\mu_l \triangleq \frac{E[y_l h_l^*]}{\sqrt{E[|y_l|^2]E[|h_l|^2]}} = \frac{\sigma_a^2 c_l}{\sqrt{(|c_l|^2\ \sigma_a^2+\sigma_n^2)(\sigma_a^2+\sigma_m^2)}} \tag{3.30}$$

The author of [6] also derived the probability density function of z_l conditioned on the transmitted code symbol c_l as follows:

$$\begin{aligned} p(z_l \mid c_l) &= \frac{1}{2\pi\sigma_h^2\sigma_y^2(1-|\mu|^2)} \exp\left[\frac{R[z_l\mu_l^*]}{\sigma_y\sigma_h(1-|\mu|^2)}\right] \\ &\times K_0\left(\frac{|z_l|}{\sigma_y\sigma_h(1-|\mu|^2)}\right) \end{aligned} \tag{3.31}$$

Where $K_o(x)$ is the zeroth order Hankel function of x, and $R(x)$ denotes the real component of x. The author of [6] then uses MAP algorithm as mentioned in section (3.2) to calculate the log likelihood ratio of *a posteriori* probabilities as follows:

$$\begin{aligned} \bar{L}(u_i) &= \ln\frac{P(u_i=1\mid z_l)}{P(u_i=0\mid z_l)} = \ln\frac{\sum_{(m_{i-1},m_i)\in S_1} p(m_{i-1},m_i,z_l)}{\sum_{(m_{i-1},m_i)\in S_0} p(m_{i-1},m_i,z_l)} \\ &= \ln\frac{\sum_{(m_{i-1},m_i)\in S_1} \bar{\alpha}_{i-1}(m_{i-1})\bar{\gamma}(m_{i-1},m_i)\bar{\beta}(m_i)}{\sum_{(m_{i-1},m_i)\in S_0} \bar{\alpha}_{i-1}(m_{i-1})\bar{\gamma}(m_{i-1},m_i)\bar{\beta}(m_i)} \end{aligned} \tag{3.32}$$

Similar to the formula (3.14), the author gets,

$$\bar{\gamma}_i(m_{i-1},m_i) = P(u_i)\,p(z_i\mid c_i) \tag{3.3}$$

Finally, the author of [6] gives out the new channel reliability factor $\bar{L}_c$ as follows:

$$\bar{L}_c = \frac{4\sqrt{E_c}}{N_0}\sigma_a^2\left[\sigma_m^2\left(\frac{2E_s}{N_0}\sigma_a^2+1\right)+\sigma_a^2\right]^{-1} \tag{3.34}$$

For a perfect channel estimator (1), $\sigma_m^2 = 0$, $\bar{L}_c = \frac{4\sqrt{E_c}}{N_0} = L_c$.

The BER performance may be improved by up to1 dB at a bit error probability of 10^{-3} by applying new decision variables and the new channel reliability factor.

CHAPTER IV

ADAPTIVE TURBO DECODER

The exact turbo decoding metric (3.29) over Rayleigh fading channels assumes there is an estimator (1) with an estimation error variance of σ_m^2. In [6], simulation results show that the smaller the error variance, the better the BER performance. Many works have been studied to reduce the error variance of the channel estimator (1). Most of them ignored the extrinsic information (L_{e2}) generated during the turbo decoding process.

In this Chapter, we first propose a new adaptive algorithm for turbo decoding, which uses the extrinsic information (L_{e2}) and the systematic observations (y^s) as its inputs. Then we review the basic theory of the estimation. The optimal solutions of minimum mean square error are modified to become more suitable to the iterative nature of turbo decoding by combining the steep-decent method. Further, the optimal step size of the steep-decent algorithm of the Newton's method is also adapted to the iterative nature of turbo decoding. At the implementation stage, the stop-and-go strategy makes the proposed adaptive filter more realistic. The boundary of estimation error variance of the estimator (2) is also discussed in detail.

4.1 Block diagram of the proposed adaptive filter

We follow the work of author [6]. There are two things that should be noticed. First, the channel estimator (1) is imperfect; secondly, the channel estimator (1) does not update iteratively as the Turbo decoder does. In other words, after getting the new channel reliability factor and new decision variables, we do not need channel estimator

(1) any more. However, the turbo decoder generates new information about the transmitted data bits after each iteration. The extrinsic information generated by the turbo decoder could help us better understand what we have received after each iteration. The proposed algorithm makes use of this kind of information to re-estimate the channel adaptively. In this thesis we will call it channel estimator (2), as depicted in Figure 7.

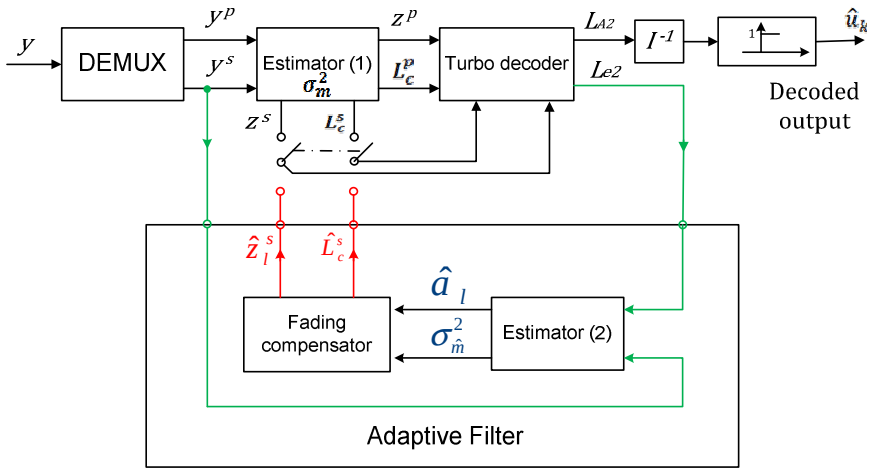

Figure 7 Block diagram of adaptive filter

Like in Figure 6, from the point view of the turbo decoder, we summarize two pieces of information, the decision variables (z or y) and the channel reliability factor (L_c), as inputs of the turbo decoder for all the three situations, which are the conventional turbo decoding metric, the exact decoding metric, and the proposed adaptive decoding metric. Further in Figure 7, we split up these two pieces of information into two sets of pairs. One set of pair is (z^p, L_c^p), which is related to the parity bits, the other is (z^s, L_c^s), which is related to the systematic bits. The superscripts (s, p) represent systematic and parity bits respectively. The proposed adaptive filter takes both the systematic observations (y^s) and the soft extrinsic information (L_{e2}), which is from the output of

the turbo decoder, as its inputs. This is one of the unique choices of this thesis. Some research takes the hard decision as its input of the adaptive channel filter [13], while some takes only the soft information as its input of the adaptive filter [14]. None of them split up the two pieces of information (z, L_c) into two sets of pairs. The basic motivation to make such kind of choice is that we do not want any delay or memory in the proposed algorithm. Any delay or register would increase the cost and complexity of the turbo decoder. Because the length of the extrinsic information is N , which is the same as the length of the systematic observations, we could simultaneously calculate the updated channel without any delay or register. We then re-estimate the channel, and finally update the channel reliability factor and decision variables after each iteration of the turbo decoding process. Because the extrinsic information generated by the turbo decoder is only related to the systematic data bits, we only update the channel reliability factors and the decision variables that are related to the systematic bits. Once we get the updated channel estimation and its variance, the fading compensator computes out the two updated pieces of information that the turbo decoder needed as depicted in Figure 6 and Figure 7, the updated channel reliability factor $\hat{L}_c^s$ and the updated decision variables $\hat{z}_l^s$.

4.2 Estimation theory review

According to the theory of estimation discussed in [21], the minimum mean square error estimator $\hat{a}$ of the unknown channel a , given observations y , is:

$$\hat{a} = w^o y \tag{4.1}$$

where w^o is any solution that satisfies the normal equation,

$$w^o = R_{ay} R_y^{-1} \tag{4.2}$$

while the covariance R_y and the cross-covariance R_{ay} are defined as follows:

$$\begin{aligned} R_y &= Eyy^* \\ R_{ay} &= Eay^* \end{aligned} \tag{4.3}$$

The solution w^o minimizes the cost function of the channel in the mean square error sense,

$$\min_{w^o} E(a - \hat{a})^2 \tag{4.4}$$

and the minimum mean square error (m.m.s.e.) is:

$$m.m.s.e. = R_a - R_{ay} R_y^{-1} R_{ya} \tag{4.5}$$

The optimal linear solution w^o is clearly not sensitive to the iterations of the turbo decoding process in general. In other words, no matter how many iterations we choose, the optimal minimum mean square error solution remains the same. This also means that the optimal linear solution w^o is optimal for the whole of the iterations, not for each of them. If we use the optimal linear solution w^o directly in the each iteration of the turbo decoding process, the turbo decoder must be disturbed at each of the iterations.

The any solution w^o could also be achieved by the steepest-decent algorithm of Newton's method [21] iteratively, as follows:

$$\begin{cases} w_i = w_{i-1} + \mu R_y^{-1} [R_{ya} - R_y w_{i-1}] \\ w_{-1} = any\ initial\ guess \\ \mu = step\ size \end{cases} \tag{4.6}$$

where i is the iterations of the Newton's method.

Because the steep-decent algorithm and the decoding process of the turbo decoder have such similar iterative characteristics, they could help each other during the decoding process.

4.3 The proposed estimator (2) of channel $\underline{a}$

First we calculate the covariance R_y and the cross-covariance R_{ay} by using the definition [20] as well as the channel model discussed in chapter 2,

$$\begin{aligned} R_y &= Eyy^* = E(ac+n)(ac+n)^* = 2\sigma_a^2 + 2\sigma_n^2 \\ R_{ay} &= Eay^* = Ea(ac+n)^* = 2\sigma_a^2 m_c \\ R_{ya} &= Eya^* = E(ac+n)a^* = 2\sigma_a^2 m_c \end{aligned} \tag{4.7}$$

Then we get the solution to the Newton's method

$$\begin{aligned} w_i &= w_{i-1} + \mu R_y^{-1}[R_{ya} - R_y w_{i-1}] \\ &= w_{i-1} + \frac{\mu[2\sigma_a^2 m_c - (2\sigma_a^2 + 2\sigma_n^2)w_{i-1}]}{2\sigma_a^2 + 2\sigma_n^2} \\ &= w_{i-1} + \frac{\mu[\sigma_a^2 m_c - (\sigma_a^2 + \sigma_n^2)w_{i-1}]}{\sigma_a^2 + \sigma_n^2} \end{aligned} \tag{4.8}$$

where the mean value of the coded bits (m_c) is related to the extrinsic information (L_{e2}). See formula (4.16) later.

The optimal step size μ^o is calculated as follows [21]:

$$\mu^o = \frac{2}{\lambda_{\max} + \lambda_{\min}} = \frac{1}{2(\sigma_a^2 + \sigma_n^2)} \tag{4.9}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ denote the maximum and minimum eigenvalues of the covariance R_y. Theoretically, the optimal step size is for the situation as the iteration $i \to \infty$. It is clear again that the optimal step size is not sensitive to the iterations of the turbo

decoding process as we see in the formula (4.9). Within the limited iterations of the decoding process, we naturally want to take the biggest step size first then gradually reduce the step size to reach the fastest convergence. In other words, we need relate the optimal step size to the iterations of the turbo decoding process in some specific way, as shown below.

We combine the solution to Newton's method and the normal equation, as well as based on the above considerations of the step size, the estimator (2) of the channel a is:

$$\begin{cases} \hat{a}_i^s = w_i y^s = \hat{a}_{i-1}^s + \dfrac{\mu[\sigma_a^2 m_c y^s - (\sigma_a^2 + \sigma_n^2)\hat{a}_{i-1}^s]}{\sigma_a^2 + \sigma_n^2} \\ \hat{a}_{-1} = h_l \\ \mu = \dfrac{k}{i \times (\sigma_a^2 + \sigma_n^2)} \end{cases} \tag{4.10}$$

In the above equation, we take the estimation of the estimator (1), both the systematic and parity bit part, as the initial value of the estimator (2), which is $\hat{a}_{-1} = h_l$ as shown in the formula. The size of the initial value $\hat{a}_{-1}$ is $n \times N$ with a code rate of $r = 1/n$, while the size of $\hat{a}_i^s$, m_c, w_i and y^s is N, which is the length of the information sequence. After the initialization, we only calculate and update the systematic part $\hat{a}_i^s$ as the superscript s indicated. And i is the iterations of the turbo decoder and/or Newton's method. Here, we combine them together and make no differentiation between them afterwards. The step size is reversely proportional to the iteration of the turbo decoder by practice and the above discussions.

Hence, the minimum mean square error (m.m.s.e.) of the estimator (2) and the variance of the estimation error is as following:

$$m.m.s.e. = R_a - R_{ay} R_y^{-1} R_{ya} = 2\sigma_a^2 - \frac{(2\sigma_a^2 m_c)^2}{2\sigma_a^2 + 2\sigma_n^2} \tag{4.11}$$

$$\sigma_{\hat{m}}^2 = \sigma_a^2 (1 - \frac{\sigma_a^2 m_c^2}{\sigma_a^2 + \sigma_n^2}) \tag{4.12}$$

Finally we get the updated channel reliability factor $\hat{L}_c^s$ and updated decision variables $\hat{z}_l^s$,

$$\begin{aligned} \hat{L}_c^s &= \frac{4\sqrt{E_s}}{N_0} \sigma_a^2 \left[\sigma_{\hat{m}}^2 (\frac{2E_s}{N_0} \sigma_a^2 + 1) + \sigma_a^2 \right]^{-1} \\ \hat{z}_l^s &= y_l^s \hat{a}_i^s \end{aligned} \tag{4.13}$$

where the size of $\hat{L}_c^s$ and $\hat{z}_l^s$ is also N, the length of the information sequence.

The proposed adaptive decoding metric (4.13) is the same as the one used in the exact decoding metric (3.29), except that the error variance $\sigma_{\hat{m}}^2$ is updated iteratively during the decoding process with the selecting criterion as follows.

$$\sigma_{\hat{m}}^2 < \sigma_m^2 \tag{4.14}$$

In the above equations we need to calculate the mean value of the coded bits. By definition [7], we have

$$\begin{aligned} L_{e2} = \ln \frac{P(u_k = 1)}{P(u_k = 0)} = \ln \frac{P(c = 1)}{P(c = -1)} &\triangleq \ln \frac{p}{1 - p} \\ p &= \frac{e^{L_{e2}}}{1 + e^{L_{e2}}} \\ f_c &= p\delta(c - 1) + (1 - p)\delta(c + 1) \end{aligned} \tag{4.15}$$

So the mean value of the coded bits is:

$$m_c = E[c] = \int_{-\infty}^{+\infty} c f_c(c) dc = 2p - 1 = \tanh(\frac{L_{e2}}{2}) \tag{4.16}$$

The mean value of the coded bits is only related to the systematic bits, so we did not put a superscript *s* around its right upper corner for simplicity.

One last thing we need to mention is that all the calculations are bit wised in the formulas. This also means each systematic bit has gone through the channel with different channel estimations. The square of m_c in the equation (4.12) is calculated by array power function with the Matlab, and the term $m_c y^s$ in the formula (4.10) is calculated by array multiplication with the Matlab, and so are the array operations in the other formulas. For simplicity, we did not put another notation around them to avoid notation confusion. But they should be clear by the context.

4.4 Implementation of the proposed adaptive filter

Considering the results of the calculation and some practical additions to the adaptive filter, we construct the adaptive filter as depicted in Figure 8. In the diagram below, the proposed adaptive filter has two inputs and two outputs.

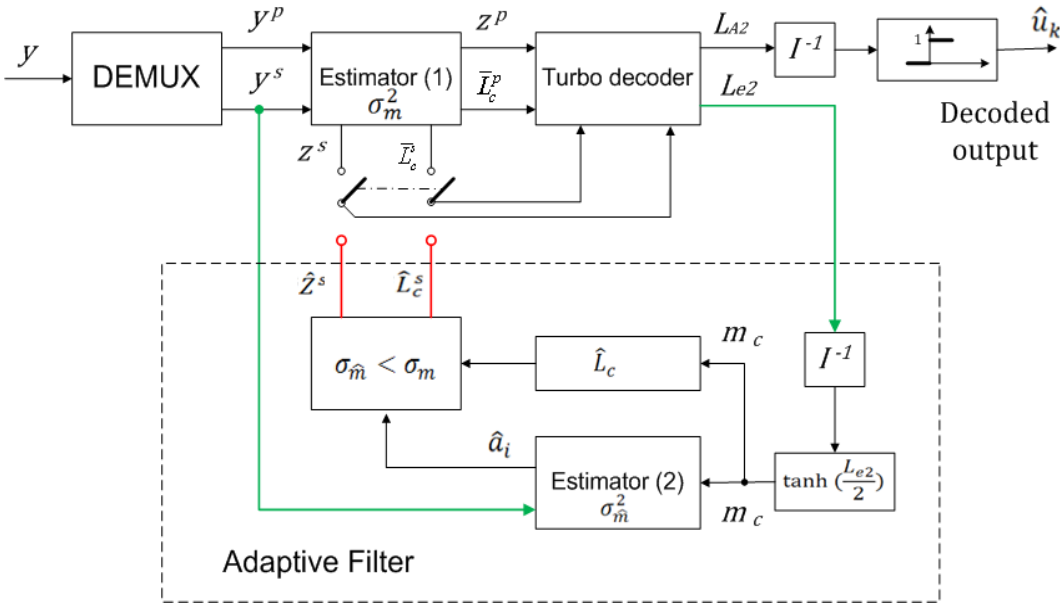

Figure 8 Implementation of the adaptive filter

The two inputs are the extrinsic information from the turbo decoder and the observations that are related to the systematic bits. The two outputs are the updated observations and updated new channel reliability factor.

At the first iteration, the received coded data bits go through the estimator (1). The estimator (1) produces two pieces of information, the new channel reliability factor $\bar{L}_c$ and the new decision variables z_l, that the turbo decoder needed, as depicted in Figure 6. We split these two pieces of information into a systematic part ($z^s, \bar{L}_c^s$) and a parity part ($z^p, \bar{L}_c^p$). Both systematic part and parity part are the inputs to the turbo decoder at the first iteration. After the first iteration, we get the extrinsic information from the turbo decoder, which could help us better understand what we have received about the transmitted data bits. In the meantime, we toggle the switch to the estimator (2). The extrinsic information from the turbo decoder is first de-interleaved, and then, by a simple function, we get the mean value of the systematic bits. Using the mean values we immediately get the error variances of the updated channel or estimator (2) through the formula (4.12). After the first iteration, we take the estimation values of the channel from the channel estimator (1) as the initial guess of the adaptive channels estimator (2). Then we get the updated channel reliability factor and decision variables by the formula (4.13). Through practice we compare the error variances of the estimator (2) to the error variances of estimator (1). We only update the information that has less error variances in estimator (2). If the error variances or standard derivations of estimator (2) are bigger than those of estimator (1), we skip further calculation for those bits. The comparison of the error variances of estimator (1) and estimator (2) provides the proposed adaptive filter with a stop-and-go character, which makes the adaptive filter more realistic.

The extrinsic information from the turbo decoder is only related to the systematic bits, so we only update the decision variables and the channel reliability factors ($\hat{z}^s, \hat{L}_c^s$) that are related to the systematic bits. While the parity part ($z^p, \overline{L}_c^p$) remains the same during the rest of the decoding iteration process.

4.5 The boundary of estimation error variance of the estimator (2)

The proposed adaptive filter has a selective criterion, as shown in Figure 8 and the formula (4.14). We rewrite the formula (4.12), (4.14) and (4.16) here for convenience.

$$\sigma_{\hat{m}}^2 = \sigma_a^2 (1 - \frac{\sigma_a^2 m_c^2}{\sigma_a^2 + \sigma_n^2}) \tag{4.12}$$

$$\sigma_{\hat{m}}^2 < \sigma_m^2 \tag{4.14}$$

$$m_c = E[c] = \int_{-\infty}^{+\infty} c f_c(c) dc = 2p - 1 = \tanh(\frac{L_{e2}}{2}) \tag{4.16}$$

The right side of the formula (4.14) is the estimation error variance of the estimator (1), while the left side is the estimation error variance of the estimator (2), which varies during the decoding process. We wish to get the smaller error variance of the adaptive filter. So, the minimum variance or the boundary of the adaptive filter happens when the mean value of the coded bits reach its maximum. From formula (4.16), we know that the maximum value of m_c^2 in the formula (4.12) is 1, so we get the boundary of estimation error variance of the estimator (2) as follows:

$$\sigma_a^2 (1 - \frac{\sigma_a^2}{\sigma_a^2 + \sigma_n^2}) < \sigma_{\hat{m}}^2 < \sigma_m^2 \tag{4.17}$$

which could be further simplified as:

$$\frac{\sigma_a^2\sigma_n^2}{\sigma_a^2+\sigma_n^2} < \sigma_{\hat{m}}^2 < \sigma_m^2 \tag{4.18}$$

After some calculations and from formula (2.1), we get,

$$\sigma_n^2 = \frac{N_0}{2} = \frac{\sigma_a^2 E_s}{r \times 10^{SNR(dB)/10}} \tag{4.19}$$

So, we relate the boundary to the signal-to-noise ratio as follows:

$$\frac{\sigma_a^2 E_s}{E_s + r \times 10^{SNR(dB)/10}} < \sigma_{\hat{m}}^2 < \sigma_m^2 \tag{4.20}$$

With the code rate of $r = 1/2$, and setting both σ_a^2 and E_s equal to 1, we get the boundary of estimation error variance of the estimator (2) for the special case,

$$\frac{2}{2+10^{SNR(dB)/10}} < \sigma_{\hat{m}}^2 < \sigma_m^2 \tag{4.21}$$

4.6 Decoding method comparison

In this chapter, we derived a new adaptive turbo decoding metric (4.13) for BPSK signaling on Rayleigh fading channels with the channel estimator (1) providing a certain error variance.

In some studies, the performance of turbo decoding on Rayleigh fading channels has also been studied [3], [4] and [22]. In [3], the amplitude and phase of the fading channels are assumed to be known, and then the Rayleigh fading channel can be modified as a special case of the AWGN channel conditioned on the known fading factors. In [4], the phase of the fading channels is assumed to be known and the amplitude is unknown, then the probability density function (pdf) of the received symbols is adopted approximately as Gaussian by averaging the fading process over all possible values. Thus,

the conventional decoding metric of AWGN may be used. In [22], the amplitude is assumed to be constant and the phase is unknown, the decision variables are also modified approximately as Gaussian and the conventional Turbo decoding metric is used again. However, in practical communication systems, the channel information is completely unknown at the receiver, and the fading channels must be estimated at the receiver. In [6], such an estimator is assumed to provide us with an unbiased channel estimate with a certain error variance, and the exact decoding metric on Rayleigh fading channels is derived. In [10] and [11], the exact turbo decoding metric derived in [6] is simplified with no performance degradation. All the above decoding methods for Rayleigh fading channels have no feedback from the turbo decoder, while the adaptive turbo decoding metric derived in this chapter takes the extrinsic information generated during the turbo decoding process as feedback from the turbo decoder.

CHAPTER V

SIMULATION RESULTS

5.1 General settings

In the simulation results, two generators of the constituent RSC encoder ($g1 = 37$ and $g2 = 21$, in octal notation) have been used in Figure 4 and Figure 5. The code rate is $r = 1/2$, and we set both σ_a^2 and E_s equal to 1. The channel estimator (1) in Figure 8 is simulated. That is, m_l in the formula (3.26) is generated randomly. The variance of m_l is set to $\sigma_m^2 = 0.4$ in Figure 9 and Figure 11, and the variance of m_l is set to $\sigma_m^2 = 0.4, 0.3,$ and 0.1 in Figure 10 respectively. The turbo decoder with 8 iterations is used in all situations. The block length of $N = 840$, 420, 210, and 100 are used in Figure 9 respectively, and the block length of $N = 100$ is used in Figure 10 and Figure 11.

5.2 BER performances with different settings

In Figure 9 and Figure 10, we present the simulation improvements when using the proposed adaptive filter (solid lines) against the results of Frenger's (dashed lines) in [6].

In Figure 9, we consider varying the block sizes of the information sequence. We can see that, as the block size of the information gets smaller, from $N = 840$ to $N = 100$, the performance of the turbo decoder degrades. The proposed adaptive filter does not improve the performance much when the information block size is $N = 840$ or greater than that. This could be explained due to the turbo decoder getting more information from the increased information size, which helps the decoding process. When the information block size is $N = 100$, the proposed adaptive filter could help the turbo

decoder to achieve better BER performance. Looking at the bit error rate of 3.5×10^{-4}, we see that the gain of using the proposed adaptive filter is about 1dB for the block length of $N=100$. The improvement of the turbo decoder with the proposed adaptive filter gets bigger when the information block size gets smaller.

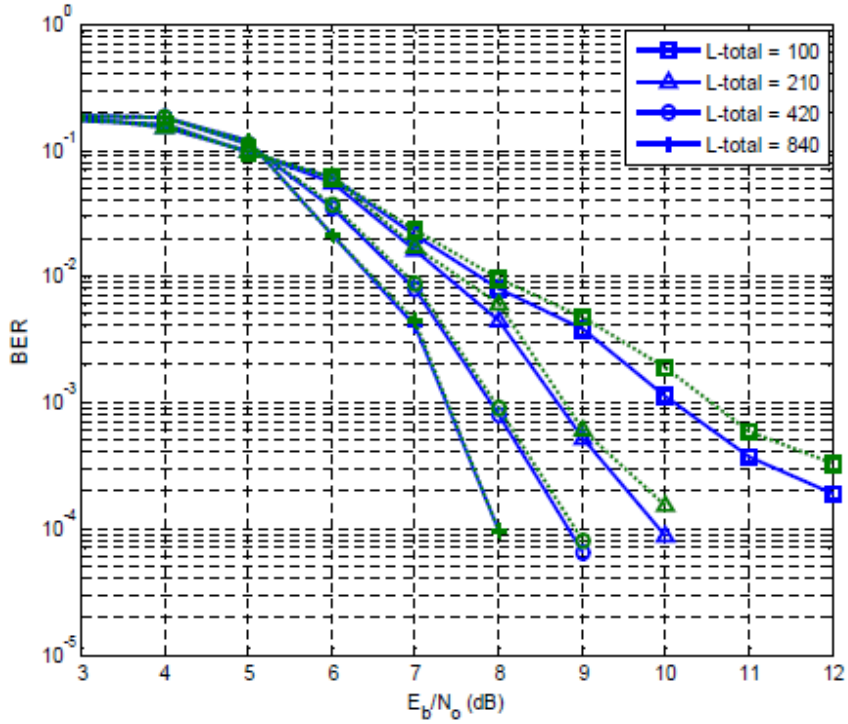

Figure 9 BER performance when using adaptive filter (solid) vs the results of Frenger's (dashed)

In Figure 10, we compare the simulation results of the proposed adaptive filter (solid lines) versus the results of Frenger's (dashed lines) in [6] with different error variances (σ_m^2) of the estimator (1) in Figure 8, while the information block size stays the same as $N=100$. When the error variance of the estimator (1) is $\sigma_m^2=0.1$ or less, we see that the proposed adaptive filter gets exactly the same curve with an SNR of less than 8dB. This is because we use the selection criterion as shown in Figure 8 and the formula (4.14), and there are no or few estimation errors from the adaptive filter that satisfies the

selection criterion. If the selection criterion is not satisfied, the proposed adaptive filter does not update the channel. This could be also explained as the estimator (1) in Figure 8 having already done a better estimation of the fading channels. When the error variance of the estimator (1) is $\sigma_m^2 = 0.4$, at the bit error rate of 3.5×10^{-4}, we see that the gain of using the proposed adaptive filter is about 1dB . We can see that as the error variance of the estimator (1) gets bigger, the improvement of the turbo decoder with the estimator (2) also gets bigger. This means when the channel estimator (1) gets worse, the proposed channels estimator (2) has more room to improve the BER performance.

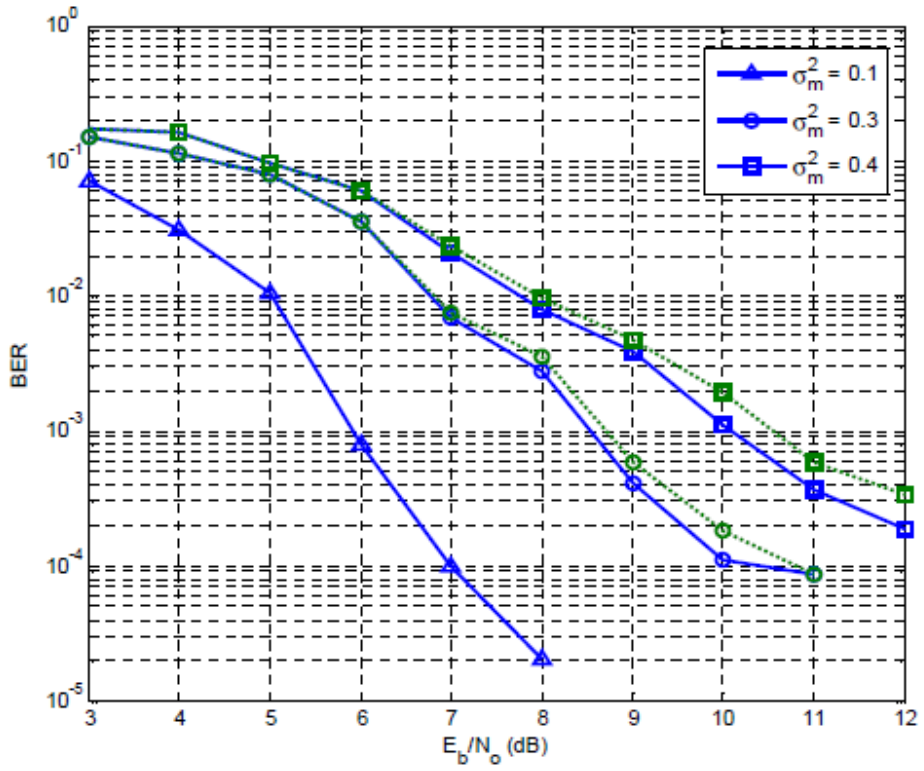

Figure 10 BER of the adaptive filter (solid) vs the results of Frenger's (dashed) with different σ_m^2

From both Figure 9 and Figure 10, we see that when either the block size of the information gets smaller or the estimation errors of the channel estimator (1) get bigger,

the proposed adaptive filter could help to improve the BER performance of the turbo decoder.

In comparison, we also give out the simulation results with the settings of $N = 100$, $\sigma_m^2 = 0.4$ and 8 iterations, but do not compare the error variance of the estimator (2) to those of the estimator (1). That is, there is no selecting criterion ($\sigma_{\hat{m}}^2 < \sigma_m^2$) for the adaptive filter in Figure 8. The adaptive filter does not provide better performance in this case.

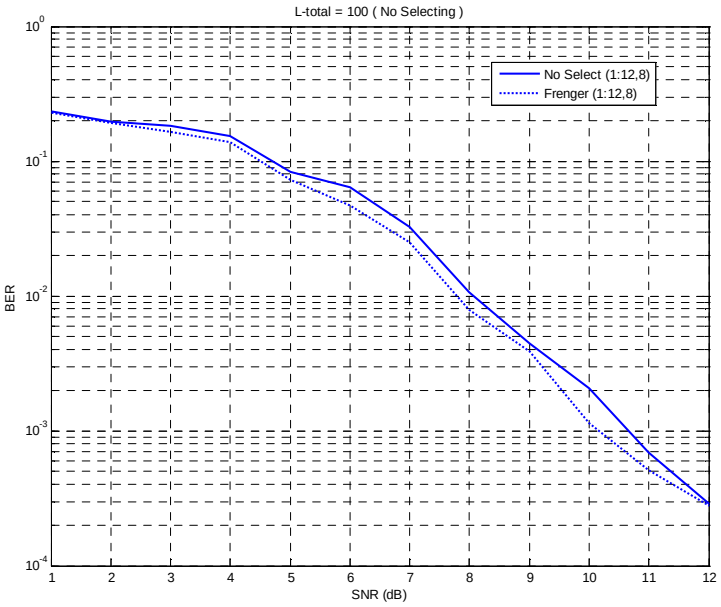

Figure 11 BER without selecting criteria

5.3 Step size and boundary

The step size of the steep-decent algorithm for the proposed adaptive filter, see formula (4.10), is depicted in Figure 12. Please note that the formula (4.10) follows the general convention of the steep-decent algorithm. The initial guess of the channel $\hat{a}_{-1}$ is actually the first iteration of the decoding process. So, the actual step size of μ in the

formula (4.10) begins to vary from iteration 2 of the decoding process. The adaptive channel estimator (2) in Figure 8 takes its biggest step at the iteration 2 of the decoding process to accelerate convergence, and then reduces the step size reversely to the iterations.

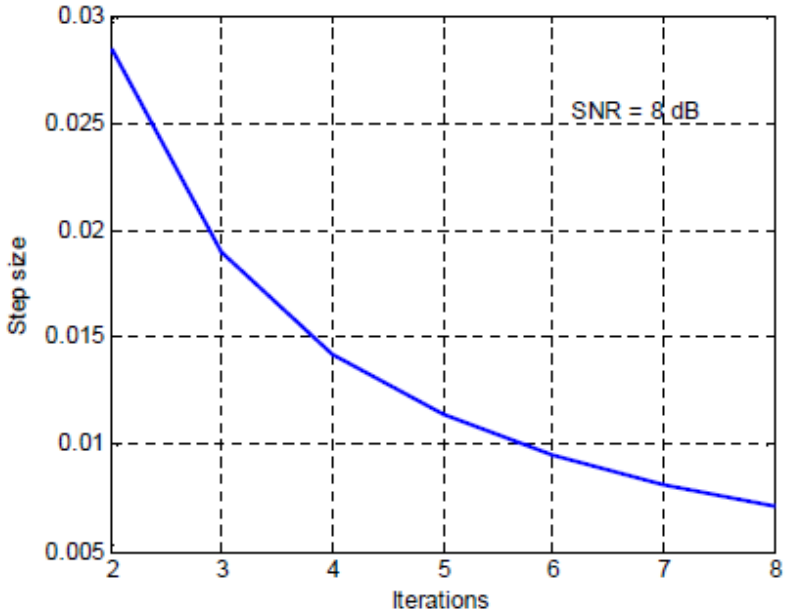

Figure 12 Step size versus iterations

In Figure 13, the boundary of estimation error variance of the estimator (2) for the special case is given according to the formula (4.21). That is, the code rate $r = 1/2$, and both σ_a^2 and E_s are set to 1. The arrow area is an example of the boundary with the estimation error variance $\sigma_m^2 = 0.4$ of the estimator (1). The arrow area shows that the adaptive filter starts to improve BER after SNR greater than 6dB when $\sigma_m^2 = 0.4$, the bigger SNR, the larger distance from $\sigma_m^2 = 0.4$ to the lower boundary. This means more ability to improve the BER performance. This could be verified by the BER performances with different settings in Figure 9. When σ_m^2 is 0.3, the adaptive filter starts

to improve the BER after SNR greater than 7dB, and when σ_m^2 is 0.1, the adaptive filter does not improve the BER before SNR greater than 13dB. These could also be verified by the BER performances with different settings in Figure 10.

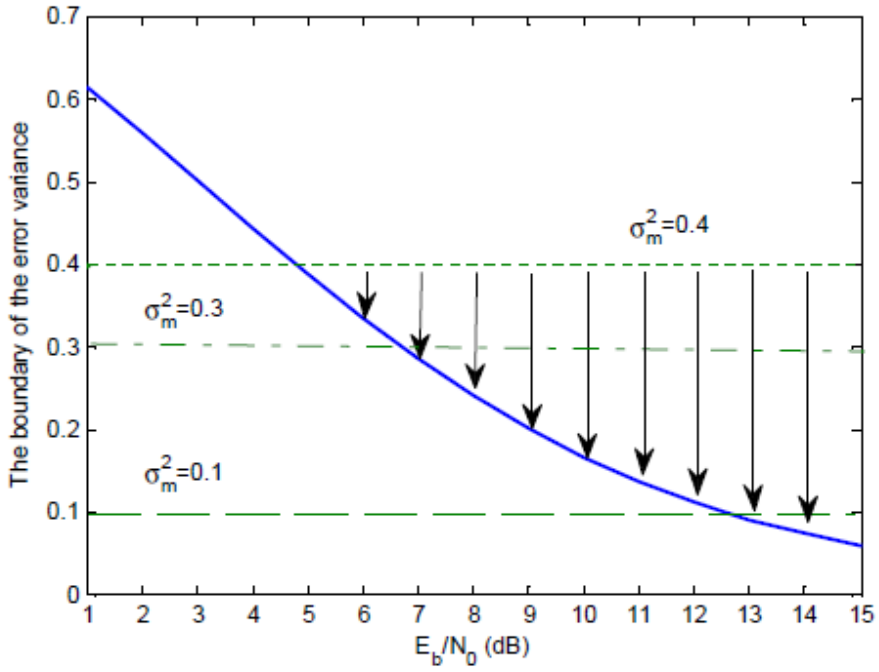

Figure 13 The boundary of estimation error variance of the estimator (2)

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

6.1 Summary of contributions

In this thesis, a number of contributions have been made in turbo decoding for BPSK signalling over Rayleigh fading channels with noise channel estimates.

First, a new adaptive channel filter with estimator (2) for Rayleigh fading channels is derived by assuming that the channel estimator (1) is available at first iteration of the decoding process. Channel estimator (1) is imperfect with some estimation errors. The proposed algorithm is based on the new turbo decoding metric which is derived by Frenger in [6]. However, the new decoding metric in [6] is fixed during the iterations of the turbo decoding process, see formula (3.29). The turbo decoder provides soft extrinsic information of the transmitted data bits which is used by the adaptive filter in this thesis to update the new decoding metric for the next iterations of decoding, see formula (4.13). The resulting iterations between the channel estimator (2) and the turbo decoder can improve the performance of both the channel estimator (2) and the turbo decoder by using the updated information. The proposed adaptive filter works better when the block size of the information gets smaller or the estimation errors of the channel estimator (1) get bigger. The gain of using the adaptive filter is about 1dB at the bit error probability of 3.5×10^{-4} with the information block length of $N=100$ and $\sigma_m^2=0.4$. This gain is obtained with minimally increased complexity.

The second highlight of this thesis is that we have proposed an effective stop-and-go strategy at the implementation stage of the adaptive filter. That is, we set the selecting criterion for the adaptive filter. If the estimation errors of the channel estimator (2) are

bigger than those of the channel estimator (1), the proposed algorithm stops updating the decoding metric. The proposed algorithm only continues when the selecting criterion is satisfied.

In the end, we argue that the steep-decent algorithm used in this thesis is suitable for the nature of the turbo decoder. The turbo decoder must go several iterations to achieve a better decoding result, while the steep-decent algorithm also takes several steps to get closer to the optimal point. They help each other during the decoding process although the steep-decent method is not an optimal method. Normally, an optimal method is achieved within one step comparing to the steep-decent method. If we use the normal optimal method at each iteration of the decoding process, the turbo decoder is disturbed by such a one step optimal method.

6.2 Recommendations for future studies

Because the proposed adaptive filter makes use of the extrinsic information from the turbo decoder and the extrinsic information produced by the turbo decoder currently is only related to the systematic bits, future research could develop a turbo decoder that could produce the extrinsic information that are related to both systematic and parity bits. Then, based on this research, it would be more interesting to develop an adaptive filter that uses the extrinsic information of both systematic bits and parity bits.

It is more important for the future research to develop higher-order modulation schemes based on the proposed algorithm, which is derived for the turbo decoding for the BPSK signalling over Rayleigh fading channels. The higher-order modulation schemes have much more spectrum efficiency in the modern wireless communication system. The

higher-order schemes could be, for example, quadrature phase shift keying (QPSK) or M-ary quadrature amplitude modulation (M-ary QAM).

Finally, the concept of the proposed adaptive turbo filter could be applied to adaptive channel equalizer by using the extrinsic information of the turbo decoder. It is important to develop an iterative turbo equalizer over the Rayleigh fading channels that have intersymbol interference.

APPENDICES

APPENDIX A

Matlab scripts of the turbo decoder with adaptive filter

1. Adaptive_Rayleigh_complex

```
% Turbo codes on Rayleigh fading channels using Log-MAP decoder
 % Copyright Oct. 2011 YuQing Guo
% Unversity of Windsor.  guo14@uwindsor.ca
% for academic use only
 % Rayleigh Fading Channels
% to modify Frenger's result

clear;
clc;

diary AdaptiveFilter_YQ.txt;

% Paul Frenger's paper
L_total = 100; % 420 is the parameter in Frenger's paper
g = [1 1 1 1 1; 1 0 0 0 1]; % Frenger. or g1=37, g2=21 in octal form
sigma_a = sqrt(1); % variance of fading coefficient @ Frenger
% alpha_factor = 0.5; % 0 0.1 0.5 1

[n,K] = size(g);
m = K - 1;
nstates = 2^m;
puncture = 0; %puncturing into rate 1/2; % exactly result of Frenger %puncture = 1; %no puncturing rate 1/3
rate = 1/(2+puncture);   % Code rate
niter = 8;% Number of iterations
Ferlim =[10];% Number of frame errors to count as a stop criterior
```

```
SNR = [11]; % Signal to noise ratio
k1=0.0725; % step size coefficient
Fetch_iter = 1;  % set up fetching iteration --yq
varian = 0.4;  % set up sigma_m^2  -- yq

Error = zeros(length(SNR), niter); % bit error
Error_hat = zeros(length(SNR), niter);
BER = zeros(length(SNR), niter); % bit error rate
BER_hat = zeros(length(SNR), niter); % bit error rate
ErrorFrame = zeros(length(SNR), niter); % frame error
FER = zeros(length(SNR), niter); % frame error rate
FrameNum = zeros(length(SNR), 1); % transmitted frame numbers for each SNR
mu = zeros(length(SNR), niter); % step size --yq
fprintf('\n\n-----------------------------------------------------\n');
fprintf(' Frame size = %6d\n',L_total);
fprintf(' code generator: \n');
for i = 1:n
   for j = 1:K
      fprintf( '%6d', g(i,j));
   end
   fprintf('\n');
end
if puncture==0
   fprintf(' Punctured, code rate = 1/2 \n');
else
   fprintf(' Unpunctured, code rate = 1/3 \n');
end
fprintf(' iteration number =  %6d\n', niter);
fprintf(' Eb / N0 (dB) = ');
for i = 1:length(SNR)
   fprintf('%10.2f',SNR(i));
```

```
end
fprintf('\n-------------------------------------------------\n\n');

fprintf('+ + + + Please be patient. Wait a while to get the result. + + + +\n');

for nEN = 1:length(SNR) % each SNR(dB)
    Eb_N0 = 10^(SNR(nEN)/10);      % convert Eb/N0 from unit db to normal numbers
    Es = 2*sigma_a*sigma_a;  % average power per symbol
    Eb = Es/rate;
    N0 = Eb/Eb_N0;
    sigma_n = sqrt(N0/2);   % standard deviation of AWGN noise Eb = Es
    sigma_m =sqrt(varian); %constant sigma_m^2  -- yq
    L_c_perfect = 4/N0; % the perfect value of channel reliability factor

    num = sigma_a^2;
    den = sigma_m.^2*(2*sigma_a^2/N0 + 1) + sigma_a^2;
    L_c = L_c_perfect*num./den; % Frenger's result % L_c = L_c_perfect; % conventional result
    a_a = 1+sigma_n^2/sigma_a^2;  % --- yq
    nframe = 1;
    Length=zeros(1,niter); % ---yq
    Lth=zeros(1,niter); % ---yq
    AverageIndex=zeros(1,niter); % ---yq

    while ErrorFrame(nEN, niter) < Ferlim(nEN)

        x = round(rand(1, L_total-m));    % info. bits
        [temp, alpha] = sort(rand(1,L_total));        % random interleaver mapping
        en_output = encoderm( x, g, alpha, puncture ) ; % encoder output (+1/-1)

        % Rayleigh Fading Channel (complex numbers)
        h = sigma_a*complex(randn(size(en_output)), randn(size(en_output)));
```

```
noise = sigma_n*complex(randn(size(en_output)), randn(size(en_output)));


r = h.*en_output + noise; % received signals
% channel estimates
   h_estimate = h + sigma_m.*complex(randn(size(h)), randn(size(h)));
   a_hat =  h_estimate;  % Adaptive start point ---yq
   sigma_m_hat(1:(puncture+2)*L_total) = sigma_m; % Adaptive start point ---yq
% decision variable after matached filter
z = r.*conj(h_estimate);
%z_real = real(z);
%yk = demultiplex(z_real,alpha,puncture); % demultiplex to get input for decoder 1 and 2
%rec_s = 0.5*L_c*yk;
z_real = L_c.*real(z);   % -------yq
yk = demultiplex(z_real,alpha,puncture); %
rec_s = 0.5*yk;          % -------yq
% Initialize extrinsic information
L_e(1:L_total) = zeros(1,L_total);
    index=[];  % --yq


for iter = 1:niter
   % Decoder one (turbo 1 from Frenger) -- yq
   % deinterleave the extrinsic information for first decoder   -yzh
   L_a(alpha) = L_e;  % a priori info.
   L_all = logmapo(rec_s(1,:), g, L_a, 1);  % complete info.
   L_e = L_all - 2*rec_s(1,1:2:2*L_total) - L_a;  % extrinsic info.


   % Decoder two
   L_a = L_e(alpha);  % a priori info.
   L_all = logmapo(rec_s(2,:), g, L_a, 2);  % complete info.
   L_e = L_all - 2*rec_s(2,1:2:2*L_total) - L_a;  % extrinsic info.
   x_hat(alpha) = (sign(L_all)+1)/2;  % Estimate the info. bits
```

```
 % caculate the BER at different SNR level %
Error(nEN,iter) = length(find(x_hat(1:L_total-m) ~= x)) + Error(nEN,iter);
BER(nEN,iter) = Error(nEN,iter)/(nframe*(L_total-m));

if length(find(x_hat(1:L_total-m) ~= x)) > 0 % this frame contains at least one bit error
    ErrorFrame(nEN,iter) = 1 + ErrorFrame(nEN,iter); % frame error
end
FER(nEN,iter) = ErrorFrame(nEN,iter)/nframe; % frame error rate
FrameNum(nEN) = nframe;  %?? from previous fellow student, not used --- yq

if iter ==  Fetch_iter  % BELOW -- yq
  L_e_hat = L_e;  % pick up extrinsic info at exact first iteration  ---yq
  mu(nEN,1) = k1;
end

if iter > Fetch_iter % refining channel from (Fetch_iter + 1)
    M_c(alpha) = tanh(L_e_hat/2); % soft info of codewords, mean value deinterleaved  ---yq
    mu(nEN,iter) = k1/(iter*(sigma_a^2+sigma_n^2));  %--- yq
    sigma=sigma_a*sqrt(1-M_c.^2/a_a);    % m.m.s.e.--------yq
      index = find ( sigma <sigma_m);   % find m.m.s.e. less than previous one ---yq
   if puncture > 0   % unpuntured  ---yq
      e_hat = (M_c(index).*r(3*index-2)/a_a - a_hat(3*index-2));  %  -- yq
      a_hat(3*index-2)= a_hat(3*index-2)+ mu(nEN,iter)*e_hat;   % -------yq
      sigma_m_hat(3*index-2)=sigma_a*sqrt(1-M_c(index).^2/a_a);   % -------yq
  else         % punctured
     e_hat = M_c(index).*r(2*index-1)/a_a - a_hat(2*index-1); % info bits error
     a_hat(2*index-1) = a_hat(2*index-1) + mu(nEN,iter)*e_hat;   % adaptive filter-------yq
     sigma_m_hat(2*index-1)=sigma_a*sqrt(1-M_c(index).^2/a_a);
  end
den_hat = sigma_m_hat.^2*(2*sigma_a^2/N0 + 1) + sigma_a^2;   % -------yq
L_c_hat = L_c_perfect*num./den_hat;   % -------yq
```

```
    z_hat = r.*conj(a_hat);  % -------yq
    z_real_hat =L_c_hat.*real(z_hat);       % -------yq
    yk_hat = demultiplex(z_real_hat,alpha,puncture); %    -------yq
    rec_s_hat = 0.5*yk_hat;                 % -------yq

   % Decoder one for refined channel (turbo 2 from YuQing for direct comparison)
    L_a_hat(alpha) = L_e_hat;  % a priori info.
    L_all_hat = logmapo(rec_s_hat(1,:), g, L_a_hat, 1);  % complete info.
    L_e_hat = L_all_hat - 2*rec_s_hat(1,1:2:2*L_total) - L_a_hat;  % extrinsic info.
   % Decoder two  for refined channel
    L_a_hat = L_e_hat(alpha);  % a priori info.
    L_all_hat = logmapo(rec_s_hat(2,:), g, L_a_hat, 2);  % complete info.
    L_e_hat = L_all_hat - 2*rec_s_hat(2,1:2:2*L_total) - L_a_hat;  % extrinsic info.

    x_hat_hat(alpha) = (sign(L_all_hat)+1)/2;
    % BER after refining channel for next iteration,  -- yq
    Error_hat(nEN,iter) = length(find(x_hat_hat(1:L_total-m) ~= x)) + Error_hat(nEN,iter);
    BER_hat(nEN,iter) = Error_hat(nEN,iter)/(nframe*(L_total-m));
   end % end refining
    Length(1,iter) = length(index);   % --- yq
    Lth(1,iter) = Length(1,iter)+Lth(1,iter);
    AverageIndex(1,iter) = Lth(1,iter)/nframe;    % ABOVE --- yq
end % iter

% display the results after each frame has been decoded
fprintf('***** SNR = %5.2f dB **********  Log-MAP    **********\n', SNR(nEN));
%fprintf('\n ******** Constant alpha_factor = %5.1f    *********\n', alpha_factor);
fprintf('\n ****  Constant Variance = %5.1f *****\n',varian);
fprintf('Info. size = %d, ', L_total);
fprintf('code  rate 1/%d, ', 2+puncture);
fprintf('  %d frame errors to stop the simulation \n', Ferlim(nEN));
```

```
    fprintf('%d frames transmitted, %d frames in error, ', nframe, ErrorFrame(nEN, niter));

    fprintf('\n CurrentindexLength (from iteration %d to iteration %d):\n',Fetch_iter+1,niter);
    for i=1:niter
        fprintf('%11.3d    ',Length(1,i));
    end

    fprintf('\n AverageIndexLength (from iteration %d to iteration %d):\n',Fetch_iter+1,niter);
    for i=1:niter
        fprintf('%11.1f    ', AverageIndex(1,i));
    end

    fprintf('\n K1 and Step Sizes (YuQing)  (from iteration %d to iteration %d):\n', Fetch_iter+1,niter);
    for i=1:niter
        fprintf('%11.8f    ', mu(nEN,i));
    end

    fprintf('\n Bit Error Rate (Frenger) (from iteration 1 to iteration %d):\n', niter);
    for i=1:niter
        fprintf('%8.4e    ', BER(nEN,i));
    end

    fprintf('\n Bit Error Rate (YuQing)  (from iteration %d to iteration %d):\n', Fetch_iter+1,niter);
    for i=1:niter
        fprintf('%8.4e    ', BER_hat(nEN,i));
    end

    fprintf('\n ***********************************************\n\n');

    nframe = nframe + 1;
end % while
```

```
  FrameNum(nEN) = nframe;

end %nEN

diary off
```

2. bin_state

```
function bin_state = bin_state( int_state, m )
% Copyright Matt C. Valenti
% MPRG lab, Virginia Tech
% for academic use only

% converts an vector of integer into a matrix; the i-th row is the binary form
% of m bits for the i-th integer

for j = 1:length( int_state )   % length(int_state)?=max_state?  --yzh
  for i = m:-1:1
     state(j,m-i+1) = fix( int_state(j)/ (2^(i-1)) );     % FIX(X) rounds the elements of X to the nearest integers towards zero.  --yzh
     int_state(j) = int_state(j) - state(j,m-i+1)*2^(i-1);    % remain of mod 2^(i-1), the leftmost bit is most significant  -yzh
  end
end

bin_state = state;
```

3. demultiplex

```
function subr = demultiplex(r, alpha, puncture);
% Copyright 1998, Yufei Wu
```

```
% MPRG lab, Virginia Tech.
% for academic use only

% At receiver end, serial to paralle demultiplex to get the code word of each
% encoder
% alpha: interleaver mapping
% puncture = 0: use puncturing to increase rate to 1/2;
% puncture = 1; unpunctured, rate 1/3;

% Frame size, which includes info. bits and tail bits
L_total = length(r)/(2+puncture);

% Extract the parity bits for both decoders
if puncture == 1        % unpunctured
  for i = 1:L_total
     x_sys(i) = r(3*(i-1)+1);
     for j = 1:2
        subr(j,2*i) = r(3*(i-1)+1+j); % 1/3 rate, one info.bit, two parity bits   -yzh
     end
  end
else           % punctured, 1/2 rate
  for i = 1:L_total
     x_sys(i) = r(2*(i-1)+1);
     for j = 1:2
        subr(j,2*i) = 0;
     end
     if rem(i,2)>0    % even position,one check bit from ENC1, one from ENC2 alternatively  --yzh
        subr(1,2*i) = r(2*i); % odd posisition is systematic bits,puntured parity bits are padded to zero   -yzh
     else
        subr(2,2*i) = r(2*i);
     end
```

```
  end
end

% Extract the systematic bits for both decoders
for j = 1:L_total
% For decoder one
  subr(1,2*(j-1)+1) = x_sys(j);    % odd positions is reserved for systematic bits  -yzh
% For decoder two: interleave the systematic bits
  subr(2,2*(j-1)+1) = x_sys(alpha(j)); % info.bits that are put into DEC2 are interleaved bits -yzh
end
```

4. encode_bit

```
function [output, state] = encode_bit(g, input, state)
% Copyright 1996 Matthew C. Valenti
% MPRG lab, Virginia Tech
% for academic use only

% This function takes as an input a single bit to be encoded,
% as well as the coeficients of the generator polynomials and
% the current state vector.
% It returns as output n encoded data bits, where 1/n is the
% code rate.

% the rate is 1/n
% k is the constraint length
% m is the amount of memory
[n,k] = size(g);
m = k-1;
 % determine the next output bit
```

```
for i=1:n
  output(i) = g(i,1)*input;    % the first bit a_k's contribution to output  --yzh
  for j = 2:k
    output(i) = xor(output(i),g(i,j)*state(j-1)); % a_(k-j)'s contribution to output  --yzh
    % why not use rem(g(i,j)*[input,state]'),j=1:k?  --yzh
  end
end

state = [input, state(1:m-1)];  % shift one bit  --yzh
```

5. encoderm

```
function en_output = encoderm( x, g, alpha, puncture )
% Copyright Nov. 1998 Yufei Wu
% MPRG lab, Virginia Tech.
% for academic use only

% uses interleaver map 'alpha'
% if puncture = 1, unpunctured, produces a rate 1/3 output of fixed length
% if puncture = 0, punctured, produces a rate 1/2 output
% multiplexer chooses odd check bits from RSC1
% and even check bits from RSC2

% determine the constraint length (K), memory (m)
% and number of information bits plus tail bits.

[n,K] = size(g);
m = K - 1;
L_info = length(x);
L_total = L_info + m;
% generate the codeword corresponding to the 1st RSC coder
```

```
% end = 1, perfectly terminated;
input = x;
output1 = rsc_encode(g,input,1);    % why 1? terminated?  --yzh

% make a matrix with first row corresponing to info sequence
% second row corresponsing to RSC #1's check bits.
% third row corresponsing to RSC #2's check bits.

y(1,:) = output1(1:2:2*L_total);    % y: unpuncture output of encoder; y(1,:) has m bits more than input bits   -yzh
y(2,:) = output1(2:2:2*L_total);

 % interleave input to second encoder
for i = 1:L_total
   input1(1,i) = y(1,alpha(i));     %alpha--index of interleaver,   --yzh
end
output2 = rsc_encode(g, input1(1,1:L_total), -1 );  %input has been interleaved. L_total bits already.(see y(1,:)) so
unterminated  --yzh
y(3,:) = output2(2:2:2*L_total);

% paralell to serial multiplex to get output vector
% puncture = 0: rate increase from 1/3 to 1/2;
% puncture = 1; unpunctured, rate = 1/3;

 if puncture > 0     % unpunctured
   for i = 1:L_total
      for j = 1:3
         en_output(1,3*(i-1)+j) = y(j,i); % put the 3 bits of the same colomn to a sequential outputs  -yzh
      end
   end
else           % punctured into rate 1/2
   for i=1:L_total
```

```
      en_output(1,n*(i-1)+1) = y(1,i);
      if rem(i,2)  % output check bits by turns    -yzh
     % odd check bits from RSC1
        en_output(1,n*i) = y(2,i);
      else
     % even check bits from RSC2
        en_output(1,n*i) = y(3,i);
      end
    end
end
 % antipodal modulation: +1/-1
en_output = 2 * en_output - ones(size(en_output));
```

6. int_state

```
function int_state = int_state( state )
% Copyright 1996 Matthew C. Valenti
% MPRG lab, Virginia Tech.
% for academic use only

% converts a row vector of m bits into a integer (base 10)
 [dummy, m] = size( state );

for i = 1:m
   vect(i) = 2^(m-i);
end

int_state = state*vect';
```

7. logmapo

```
function L_all = logmapo(rec_s,g,L_a,ind_dec)
```

```
% Copyright Nov 1998, Yufei Wu
% MPRG lab, Virginia Tech.
% for academic use only

% Log_MAP algorithm using straightforward method to compute branch metrics
% no approximation is used.
% Can be simplified to Max-Log-MAP by using approximation ln(e^x+e^y) = max(x,y).
% Input: rec_s: scaled received bits.
%              rec_s = 0.5 * L_c * yk = ( 2 * a * rate * Eb/N0 ) * yk
%       g: code generator for the component RSC code, in binary matrix form.
%       L_a: a priori info. for the current decoder,
%              scrambled version of extrinsic Inftyo. of the previous decoder.
%       ind_dec: index of decoder. Either 1 or 2.
%              Encoder 1 is assumed to be terminated, while encoder 2 is open.
%
% Output: L_all: log-likelihood ratio of the symbols. Complete information.

% Total number of bits: Inftyo. + tail
L_total = length(rec_s)/2;
[n,K] = size(g);
m = K - 1;
nstates = 2^m;          % number of states in the trellis

% Set up the trellis
[next_out, next_state, last_out, last_state] = trellis(g);

Infty = 1e10;

% Initialization of Alpha
Alpha(1,1) = 0;
Alpha(1,2:nstates) = -Infty*ones(1,nstates-1);  % first row of matrix Alpha
```

```
% Initialization of Beta
if ind_dec==1
  Beta(L_total,1) = 0;
  Beta(L_total,2:nstates) = -Infty*ones(1,nstates-1); % the last row of matrix Beta  --yzh
elseif ind_dec==2
  Beta(L_total,1:nstates) = zeros(1,nstates); % the last row of matrix Beta  --yzh
else
  fprintf('ind_dec is limited to 1 and 2!\n');
end
% what's the meaning of Alpha and Beta and gama?  --yzh
% Trace forward, compute Alpha
for k = 2:L_total+1
   for state2 = 1:nstates
     gamma = -Infty*ones(1,nstates);
     gamma(last_state(state2,1)) = (-rec_s(2*k-3)+rec_s(2*k-2)*last_out(state2,2))....
         -log(1+exp(L_a(k-1)));   % why is "-rec_s(2*k-3)?"  --yzh
     gamma(last_state(state2,2)) = (rec_s(2*k-3)+rec_s(2*k-2)*last_out(state2,4))....
         +L_a(k-1)-log(1+exp(L_a(k-1)));  % what's the meaning of "log(1+exp(L_a(k-1))"  --yzh

     if(sum(exp(gamma+Alpha(k-1,:)))<1e-300)
       Alpha(k,state2)=-Infty;
     else
       Alpha(k,state2) = log( sum( exp( gamma+Alpha(k-1,:) ) ) );
     end
   end
   tempmax(k) = max(Alpha(k,:));
   Alpha(k,:) = Alpha(k,:) - tempmax(k);
end

% Trace backward, compute Beta
for k = L_total-1:-1:1
```

```
  for state1 = 1:nstates
    gamma = -Infty*ones(1,nstates);
    gamma(next_state(state1,1)) = (-rec_s(2*k+1)+rec_s(2*k+2)*next_out(state1,2))....
        -log(1+exp(L_a(k+1)));
    gamma(next_state(state1,2)) = (rec_s(2*k+1)+rec_s(2*k+2)*next_out(state1,4))....
        +L_a(k+1)-log(1+exp(L_a(k+1)));
    if(sum(exp(gamma+Beta(k+1,:)))<1e-300)
      Beta(k,state1)=-Infty;
    else
      Beta(k,state1) = log(sum(exp(gamma+Beta(k+1,:))));
    end
  end
  Beta(k,:) = Beta(k,:) - tempmax(k+1);
end

% Compute the soft output, log-likelihood ratio of symbols in the frame
for k = 1:L_total
  for state2 = 1:nstates
    gamma0 = (-rec_s(2*k-1)+rec_s(2*k)*last_out(state2,2))....
        -log(1+exp(L_a(k)));
    gamma1 = (rec_s(2*k-1)+rec_s(2*k)*last_out(state2,4))...
        +L_a(k)-log(1+exp(L_a(k)));
    temp0(state2) = exp(gamma0 + Alpha(k,last_state(state2,1)) + Beta(k,state2));
    temp1(state2) = exp(gamma1 + Alpha(k,last_state(state2,2)) + Beta(k,state2));
  end
  L_all(k) = log(sum(temp1)) - log(sum(temp0));
end
```

8. rsc_encode

```
function y = rsc_encode(g, x, terminated)
```

```
% Copyright Nov. 1998 Yufei Wu
% MPRG lab, Virginia Tech.
% for academic use only

% encodes a block of data x (0/1)with a recursive systematic
% convolutional code with generator vectors in g, and
% returns the output in y (0/1).
% if terminated>0, the trellis is perfectly terminated
% if terminated<0, it is left unterminated;
% determine the constraint length (K), memory (m), and rate (1/n)
% and number of information bits.
[n,K] = size(g);
m = K - 1;
if terminated>0
  L_info = length(x);   % L_info: lenght of information sequence?  -yzh
  L_total = L_info + m; % L_total:m additional bits is used to terminate?  -yzh
else
  L_total = length(x);
  L_info = L_total - m; % see the sequence for untermated in function encoderm for reason. length of x is L_total  --yzh
end

 % initialize the state vector
state = zeros(1,m);

% generate the codeword
for i = 1:L_total
   if terminated<0 | (terminated>0 & i<=L_info)
      d_k = x(1,i);     % d_k: information sequence -yzh
   elseif terminated>0 & i>L_info
      % terminate the trellis
      d_k = rem( g(1,2:K)*state', 2 ); % g(1,2:K): why is g(1,2:K)? not other recursive polynomial?  -yzh
```

```
  end
  % a_k??feedback polynomial is g(1:)? --yzh
  % for terminated>0 & i>L_info, a_k will be zero(heihei)  -yzh
  % recursive encoding?right!    --yzh
  a_k = rem( g(1,:)*[d_k state]', 2 );     % a_k: the bit to be put into the register    -yzh
  [output_bits, state] = encode_bit(g, a_k, state);
  % since systematic, first output is input bit
  output_bits(1,1) = d_k;
  y(n*(i-1)+1:n*i) = output_bits;  % n output bits for 1 input bit(recursiv encoder)  --yzh
end
```

9. trellis

```
function [next_out, next_state, last_out, last_state] = trellis(g)
% copyright Nov. 1998 Yufei Wu
% MPRG lab, Virginia Tech
% for academic use only

% set up the trellis given code generator g
% g given in binary matrix form. e.g. g = [ 1 1 1; 1 0 1 ];
 % next_out(i,1:2): trellis next_out (systematic bit; parity bit) when input = 0, state = i; next_out(i,j) = -1 or 1
% next_out(i,3:4): trellis next_out  (systematic bit; parity bit) when input = 1, state = i;
% next_state(i,1): next state when input = 0, state = i; next_state(i,i) = 1,...2^m
% next_state(i,2): next state when input = 1, state = i;
% last_out(i,1:2): trellis last_out (systematic bit; parity bit) when input = 0, state = i; last_out(i,j) = -1 or 1
% last_out(i,3:4): trellis last_out  (systematic bit; parity bit) when input = 1, state = i;
% last_state(i,1): previous state that comes to state i when info. bit = 0;
% last_state(i,2): previous state that comes to state i when info. bit = 1;

[n,K] = size(g);
m = K - 1;
```

```
max_state = 2^m;

% set up next_out and next_state matrices for systematic code
for state=1:max_state
  state_vector = bin_state( state-1, m );  % matrix state_vector is of max_state rows and m columns  --yzh

  % when receive a 0
  d_k = 0;
  a_k = rem( g(1,:)*[0 state_vector]', 2 );
  [out_0, state_0] = encode_bit(g, a_k, state_vector);
  out_0(1) = 0;

  % when receive a 1
  d_k = 1;
  a_k = rem( g(1,:)*[1 state_vector]', 2 );
  [out_1, state_1] = encode_bit(g, a_k, state_vector);
  out_1(1) = 1;
  next_out(state,:) = 2*[out_0 out_1]-1;   % BPSK? Each row has two possible outputs(according to input 1 or 0)  --yzh
  next_state(state,:) = [(int_state(state_0)+1) (int_state(state_1)+1)];   % 2 next state for current state according to input
--yzh
end

% find out which two previous states can come to present state
last_state = zeros(max_state,2);
for bit=0:1
  for state=1:max_state
    last_state(next_state(state,bit+1), bit+1)=state; % row number is the next_state, column is the input bit  --yzh
    last_out(next_state(state, bit+1), bit*2+1:bit*2+2) ...   % row is the next_state value  --yzh
      = next_out(state, bit*2+1:bit*2+2);    % next_out is the output of current state with input 0 or 1  -yzh
  end
end
```

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," In *Proc.. IEEE International Conference on Communications (ICC'93),* Vol. 2, pp. 1064-1070, May 1993.

[2] S. Benedetto and G. Montorsi, "Unveling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409-428, Mar. 1996.

[3] S.A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, School of Elect. Eng., Faculty of Eng., Univ. South Australia, Feb. 1996.

[4] E. K. Hall and S. G. Wilson, "Design and analysis of turbo codes on Rayleigh fading channels," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 160-174, Feb. 1998.

[5] P. Frenger, "Turbo decoding on Rayleigh fading channels with noisy channel estimates," *Proc. IEEE VTC'99*, Huston, TX, pp. 884-888, May 1999.

[6] P. Frenger, "Turbo decoding for wireless systems with imperfect channel estimation," *IEEE Transactions on Communications*, Vol. 48, Issue 9, pp. 1437-1440, Sept. 2000

[7] B. Sklar, "A Prime on Turbo Code Concepts," *IEEE Communications Magazine*, Vol. 35, Issue 12, pp. 94-102, Dec. 1997.

[8] T. Summers and S. G. Wilson, "SNR mismatch and online estimation in turbo decoding," *IEEE Transactions on Communications*, vol. 46 no. 5, pp. 421-423. Apr. 1998.

[9] B. Mielczarek and A. Svensson, “Improved iterative channel estimation and turbo decoding over flat-fading channels,” In *Proc. VTC’02 Fall*, pp. 975-980, September 2002.

[10] K. Xu, J. Wang, and Z. Xu, “A simplified method for turbo decoding over Rayleigh fading channels,” S*ignal Processing (ICSP), 2010 IEEE 10th International Coference on*, pp. 1601-1603, Issue: 24-28, Oct. 2010.

[11] H. Shin, J. Lee, “Channel reliability estimation for turbo decoding in Rayleigh fading channels with imperfect channel estimates,” *IEEE Communications Letters*, Vol.6, Iss.11, pp.503, 2002

[12] Y. Zhou and B. Shahrrava, “Turbo decoding with integrated fading compensation”, *Radio and Wireless Symposium, 2008 IEEE*, On page(s): 535 -538, Volume: Issue: , 22-24 Jan. 2008

[13] R.Yao, Y. Wang, D. Wang, J. Xu, , "Adaptive Algorithm for Turbo Decoding Based on Logarithmic Cross Entropy," *Communications, 2005 Asia-Pacific Conference on* , vol., no., pp.232-236, 5-5 Oct. 2005

[14] C. Wang, J. Hsu, and T. Chang, , "Adaptive channel estimation using the GOBA algorithm for turbo codes in Rayleigh flat-fading channels," *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on* , vol.4, pp.45-48, 2000

[15] R. E. Ziemer and R. L. Peterson, *Introduction to Digital communication.* New York: Macmillan, Inc., 1992

[16] J. G. Proakis, *Digital Communications*, McGraw-Hill, 5rd edition, 2008.

[17] B. Black, P. Dipiazza, B. Ferguson, D. Voltmer, and F. Berry, *Introduction to Wireless Systems.* Prentice-Hall, May 2008.

[18] L. Bahl, J. Cocke, F, Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transaction on Information Theory*, Vol.20, Issue 2, pp. 284-287, Mar 1974

[19] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Eansactions on Communications*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.

[20] R. Yates and D. Goodman, *Probability and Stochastic Processes*, John Wiley & Sons, 2nd edition, 2005

[21] A. Sayed, *Adaptive Filters*, John Wiley & Sons, 2008

[22] E. K. Hall and S. G. Wilson, "Turbo codes for noncoherent channels," in *Proc. IEEE Communication Theory Mini-Conf.*, Phoenix, AZ, Nov. 3–8, 1997, pp. 66–70.

VITA AUCTORIS

Yuqing Guo was born in Nanjing, China, in 1960. He received his Bachelor of Engineering degree from the Radio Engineering Department of Southeast University in 1983. He taught at the Department of Information and Technology of Nanjing College for Population Program Management from 1999 to 2005. He is currently a candidate for the Master of Applied Science in Electrical and Computer Engineering at the University of Windsor.