An Application of Independent Component Analysis to DS-CDMA Detection

A Thesis Submitted
to the College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Electrical Engineering
University of Saskatchewan

by

Yue Fang

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, it is agreed that the Libraries of this University may make it freely available for inspection. Permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professors who supervised this thesis work or, in their absence, by the Head of the Department of Electrical Engineering or the Dean of the College of Graduate Studies and Research at the University of Saskatchewan. Any copying, publication, or use of this thesis, or parts thereof, for financial gain without the written permission of the author is strictly prohibited. Proper recognition shall be given to the author and to the University of Saskatchewan in any scholarly use which may be made of any material in this thesis.

Request for permission to copy or to make any other use of material in this thesis in whole or in part should be addressed to:

Head of the Department of Electrical Engineering
57 Campus Drive
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

ACKNOWLEDGMENTS

I would like to express my largest gratitude to my supervisor, Professor Kunio Takaya, for his continuous support and guidance throughout my thesis work.

My appreciation goes to Dr. Dodds and Dr. Ha. Nguyen for their time and suggestions rendered to this thesis. I wish to thank all of you not just for being on my committee but for all of your guidance and assistance.

A special thanks goes out to my wife for her endless encouragement.

ABSTRACT

This work presents the application of the theory and algorithms of Independent Component Analysis (ICA) to blind multiuser symbol estimation in downlink of Direct-Sequence Code Division Multiple Access (DS-CDMA) communication system. The main focus is on blind separation of convolved CDMA mixture and the improvement of the downlink symbol estimation. Term *blind* implies that the separation is performed based upon the observation only. Since the knowledge of system parameter is available only in the downlink environment, the blind multiuser detection algorithm is an attractive option in the downlink.

Firstly, the basic principles of ICA are introduced. The objective function and optimization algorithm of ICA are discussed. A typical ICA method, one of the benchmark methods for ICA, FastICA, is considered in details. Another typical ICA algorithm, InfoMAX, is introduced as well, followed by numerical experiment to evaluate two ICA algorithms

Secondly, FastICA is proposed for blind multiuser symbol estimation as the statistical independence condition of the source signals is always met. The system model of simulation in downlink of DS-CDMA system is discussed and then an ICA based DS-CDMA downlink detector has been implemented with MATLAB. A comparison between the conventional Single User Detection (SUD) receiver and ICA detector has been made and the simulation results are analyzed as well. The results show that ICA detector is capable of blindly solving multiuser symbol estimation problem in downlink of DS-CDMA system.

The convergence of ICA algorithm is, then, discussed to obtain more stable simulation results. A joint detector, which combines ICA and SUD and where ICA is considered as an additional element attached to SUD detector, has been implemented. It was demonstrated that the joint detector gives the lowest error probability compared to conventional SUD receiver and pure ICA

detector with training sequences.

Keywords: Direct-Sequence Code Division Multiple Access (DS-CDMA), FastICA, Independent Component Analysis (ICA), Principal Component Analysis (PCA)

LIST OF ABBREVIATIONS

3rd Generation (3G)

Additive White Gaussian Noise (AWGN)

Bit Error Rate (BER)

Blind Source Separation (BSS)

Direct-Sequence Code Division Multiple Access (DS-CDMA)

Direct Sequence Spread Spectrum (DSSS)

Double Sideband Suppressed Carrier (DSB-SC)

Frequency Division Multiple Access (FDMA)

Frequency-shift keying (FSK)

Independent Component (IC)

Independent Component Analysis (ICA)

Inter-Symbol Interference (ISI)

Multiple Access Interference (MAI)

Principal Component Analysis (PCA)

Probability Density Function (pdf)

Pseudo-Noise (PN) Code

Phase-Shift Keying (PSK)

Single User Detection (SUD)

Spread Spectrum (SS)

Time Division Multiple Access (TDMA)

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1 Motivation of Research

Code Division Multiple Access (CDMA) efficiently provides high quality voice services and high-speed packet data access and it has been selected as the promising solution to 3rd generation (3G) wireless communications. Direct-Sequence Code Division Multiple Access (DS-CDMA) is one of the required radio interface of these solutions.

In a DS-CDMA system, users share the same band of frequencies and the same time slots, but they are separated by unique spreading codes. The main sources of errors at the receiver are due to the Multiple Access Interference (MAI), the Inter-Symbol Interference (ISI), the asynchronism of users and the near-far problem. The last two problems only occur at the uplink (from the mobile stations to the base stations), because in the downlink (from the base stations to mobile stations) the information-bearing signals from the base stations are transmitted in a synchronous way and with the same power to all users.

The traditional way to estimate symbols, Single User Detection (SUD), considers interference as additional noise and thus ignore the structure of MAI. This leads to Matched Filtering. The optimum Multiuser Detection (MUD) strategy, maximum likelihood method, leads to a joint estimation of each user's symbols, and it thus increases computational complexity. Dr. S. Verdú took into account the inherent structure of interference in his classic work[16] and thus a significant computational gain was introduced. The maximum likelihood sequence detector consists of a banks of matched filters followed by a Viterbi-algorithm. Dr. Verdú's invention meant that the interference of multiple access and the near-far effect can be mitigated, although not by a conventional SUD receiver.

Although MUD receivers have been an attractive field of research, we will not use optimal MUD as a reference due to

- it is still quite computationally complex, since there is less processing power in the mobile station than in the base station.
- it is not applicable in downlink environments, since the codes of the interference users should be known and each mobile station knows only its own code while the codes of the others are unknown.

These features of downlink processing call for new, efficient and simple solutions. Independent Component Analysis (ICA), one of Blind Source Separation (BSS) techniques provides a promising new approach to the downlink signal processing of DS-CDMA systems using short spreading codes. In blind MUD, no knowledge of interference parameters are needed, but the structure of MAI is still exploited in the demodulation of a particular user.

To exploit the structure of MAI, the most intuitive way is to separate each user from the mixture of DS-CDMA signal, or to eliminate all the interference when demodulating a particular user's symbol in the presence of multiaccess interference. This is also the idea behind optimal MUD, i.e. the multiaccess interference is not considered as the additional noise but the info-bearing data stream. The only interference left is background noise.

ICA is a recently developed, useful extension of standard Principal Component Analysis (PCA). It has been proposed and implemented to take advantage of Blind Source Separation (BSS) algorithms. However, the performance and robustness of the algorithm are not satisfied.

1.2 Research Objectives

The objective of this research is to build an ICA-based blind multiuser detector for symbol estimation in downlink environment of DS-CDMA system. At first, an ICA-based build multiuser detector is considered. We then propose an ICA-SUD joint detector, in which ICA detector is considered as

an additional element attached to conventional SUD receiver. The results of numerical experiment are compared to conventional SUD receiver.

1.3 Outline of the Thesis

This thesis is organized into 6 chapters.

Chapter 1 provides background and motivation for the thesis. The objectives of this research are also stated.

Background information is provided in Chapters 2 and 3 to give the reader knowledge necessary to better understand and appreciate the ICA-based DS-CDMA downlink detector. Chapter 2 describes the characteristics of ICA Algorithm, which is used for this work. Two typical ICA methods, InforMAX and FastICA, have been implemented with MATLAB. The results of numerical experiment have been analyzed as well. It shows that FastICA is a better solution for ICA-based detector.

Chapter 3 provides an overview of DS-CDMA systems. The signal model of DS-CDMA is discussed as well.

Chapter 4 reviews the related literatures, in which the signal model of ICA-based multiuser detector is discussed. Also, the short spreading code, Gold code, is discussed.

Chapter 5 describes the detailed numerical experiment implementation of ICA-based blind multiuser detector in downlink of DS-CDMA system. At first, the simulation model and the Matlab implementations of ICA-based blind multiuser detector are discussed. Next, some factors are discussed to improve the convergence properties of ICA detector. At last, ICA-SUD joint detector is simulated.

Chapter 6 summarizes the research performed and the results. Future work which can be done to improve on aspects of this work is also discussed.

2. INDEPENDENT COMPONENT ANALYSIS (ICA)

2.1 Preliminaries

In the past two decades, a particular method for finding underlying factors or components from multivariate (multidimensional) statistical data, called Independent Component Analysis (ICA), has attracted a lot of interest both in statistical signal processing and neural network communities. Several good algorithms utilizing higher-order statistics of suitable nonlinearities either directly or indirectly are now available for solving the basic linear Blind Source Separation (BSS, also known as Blind Signal Separation) problem. Nonlinearities cause the statistical distribution of variables to be non-gaussian.

Imagine that four people are speaking simultaneously in a room and four microphones positioned at different locations give you four recorded time signals which are denoted by $x_1(t)$, $x_2(t)$, $x_3(t)$ and $x_4(t)$. Each of these recorded signals is a weighted sum of the source signals emitted by the four speakers, which are denoted by $s_1(t)$, $s_2(t)$, $s_3(t)$ and $s_4(t)$. We can express this system of linear equations as:

$$\begin{cases} x_1(t) &= a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t) + a_{14}s_4(t) \\ x_2(t) &= a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t) + a_{24}s_4(t) \\ x_3(t) &= a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t) + a_{34}s_4(t) \\ x_4(t) &= a_{41}s_1(t) + a_{42}s_2(t) + a_{43}s_3(t) + a_{44}s_4(t) \end{cases} \tag{2.1}$$

where a_{ij}, $i, j = \{1, 2, 3, 4\}$ are some parameters that depend on the distances from the microphones to the speakers. It would be interesting if we could now estimate the sources $s_1(t)$, $s_2(t)$, $s_3(t)$ and $s_4(t)$ using only the recordings $x_1(t)$, $x_2(t)$, $x_3(t)$ and $x_4(t)$. This problem is called *cocktail-party problem* because one tries to eavesdrop a particular person's talking from the noisy mixture

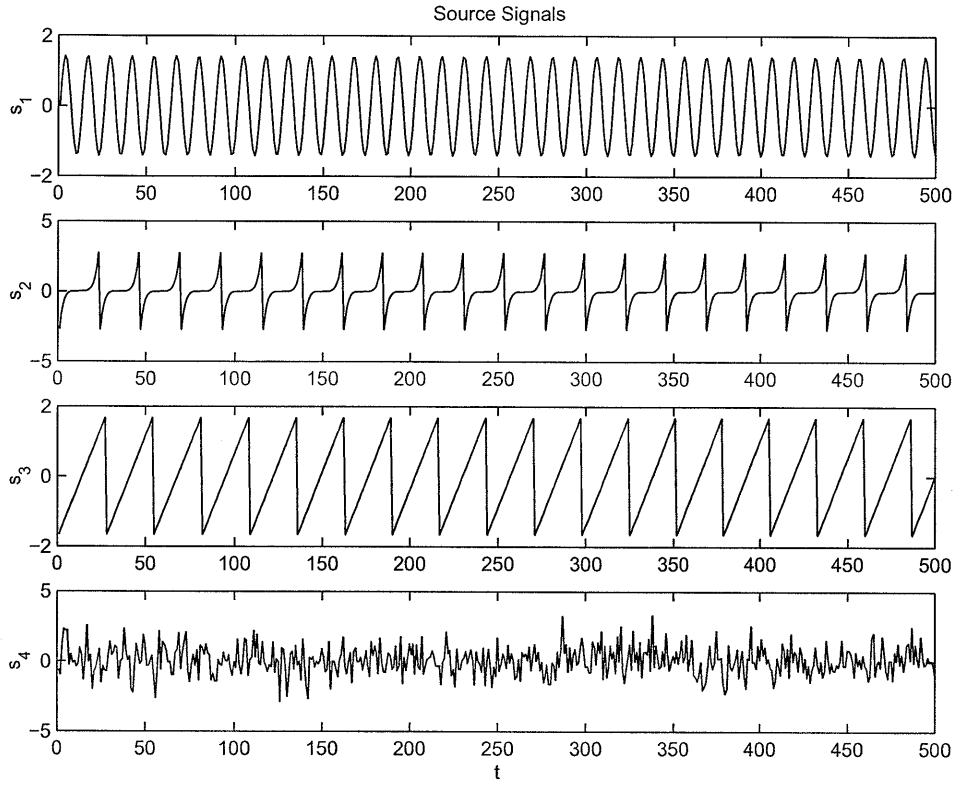

Figure 2.1 A set of four arbitrary source signals

of all conversations. As the name implies, ICA can be used to estimate the parameters $a_{ij},\ i,j = \{1,2,3,4\}$. This allows us to separate the source signals from the mixed recordings.

As an illustration, the four source waveforms are shown in Figure 2.1. Let us pretend for a moment that these waveforms represent sound signals from four speakers. Four recordings from four different positions could look something like the four mixtures shown in Figure 2.2. The problem is now to recover the signals shown in Figure 2.1 using only the data shown in Figure 2.2.

If the parameters $a_{ij},\ i,j = \{1,2,3,4\}$ are known, the source signals could have been easily extracted by linear algebraic methods. Unfortunately, since the parameters $a_{ij},\ i,j = \{1,2,3,4\}$ are unknown, the problem seems impossible. However, it turns out that the source signals can be separated from the mixtures with a realistic assumption that the four source signals $s_1(t)$, $s_2(t)$,

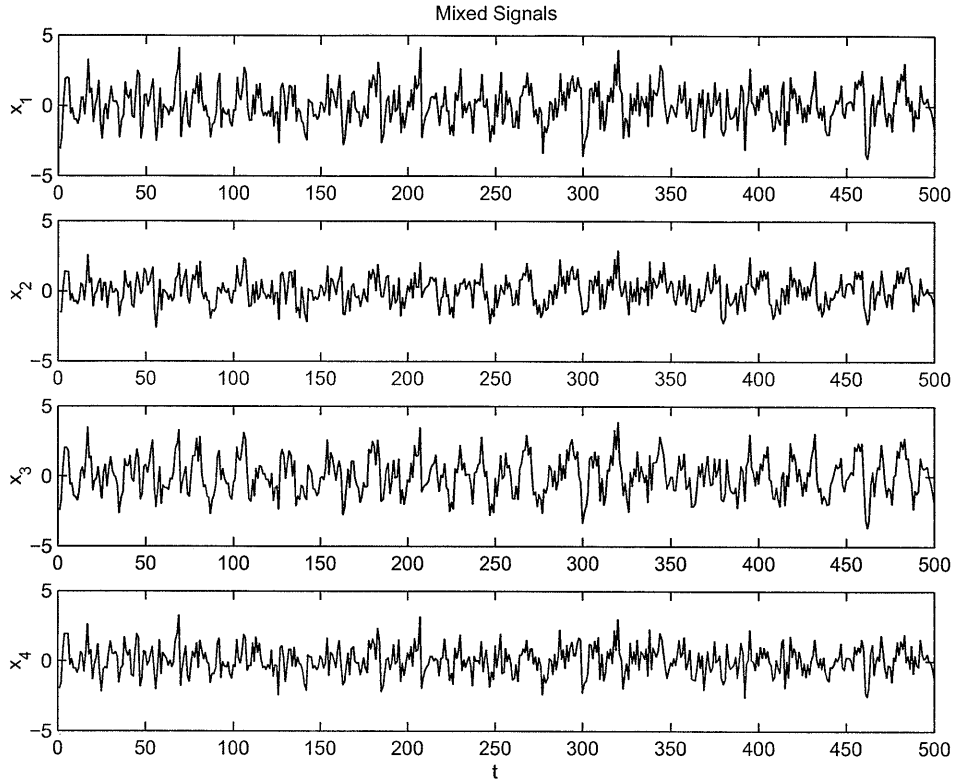

Figure 2.2 Four mixtures of the source signals in Figure 2.1

$s_3(t)$ and $s_4(t)$ are *statistically independent.*

Figure 2.3 gives an illustration of the four signals estimated by ICA method. It can be seen that the signals are very close to the original source signals. Some of the estimates are scaled by some positive or negative factor, but waveform were restored.

This *cocktail-party problem* can be extended to n sources and m mixtures, where $m \geq n$, and solved by ICA method as illustrated above. This system can be written in matrix form as:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{2.2}$$

the problem is to find out the estimation of source signals $\hat{\mathbf{s}} = \mathbf{W}\mathbf{x}$ with observed mixing signals $\mathbf{x}$ only. An illustration of ICA model is shown in Figure 2.4.

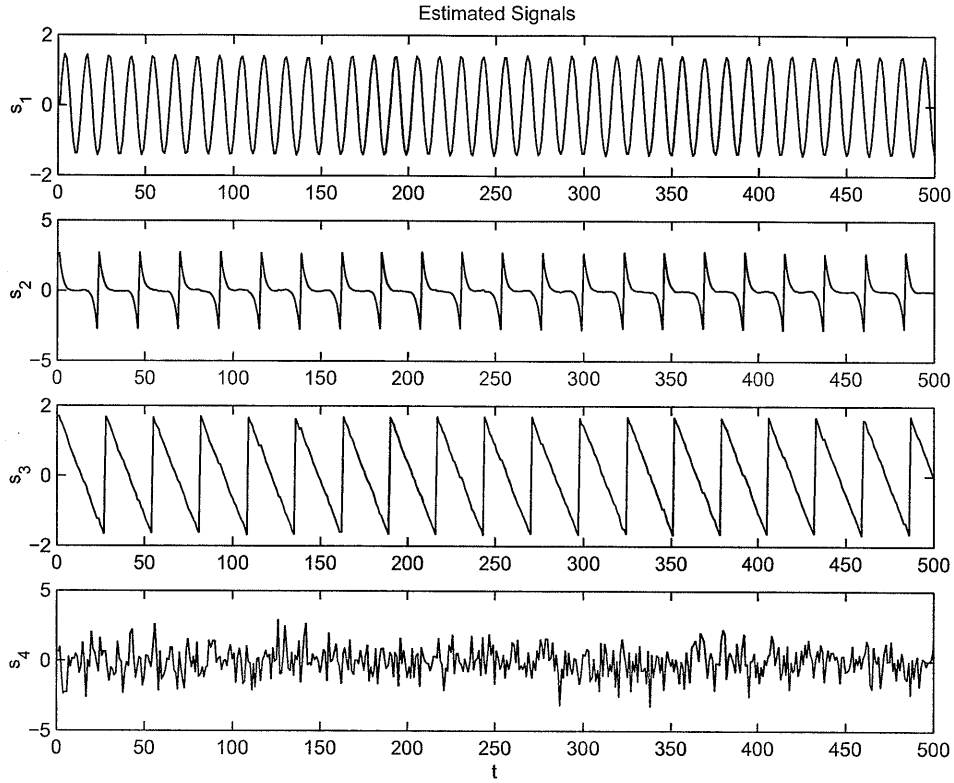

Figure 2.3 The ICA estimates of the original source signals

The ICA has many interesting applications that are similar to the *cocktail-party problem.* A brief literature review is given in Section 2.2, and a brief theoretical framework for ICA is given in next sections.

2.2 History of ICA

Source separation is an old problem and many algorithms exist depending on the nature of the mixed signals. The problem of Blind Source Separation (BSS) is usually difficult, without knowing the structure of the mixed signals.

The technique of ICA was introduced in the early 1980s by J. Hérault, C. Jutten and B. Ans [7] [8] [9]. At a meeting held in 1986 on Neural Networks for Computing, Jeanny Hérault and Christian Jutten contributed a research paper entitled "Space or time adaptive signal processing by neural network models" [11]. This paper announces the birth of ICA, although the name of

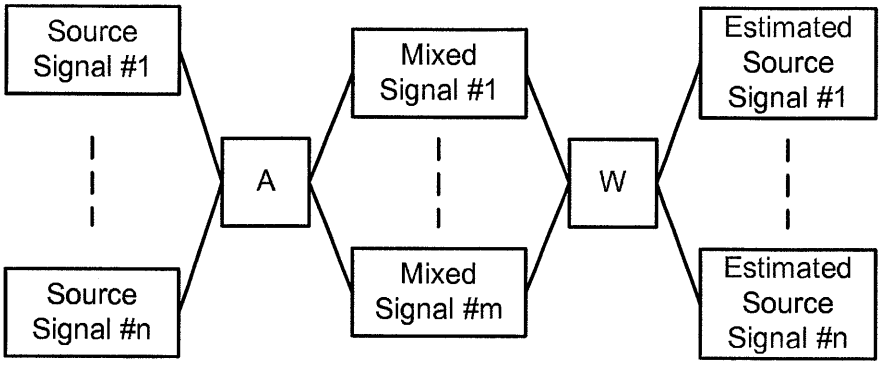

Figure 2.4 An illustration of ICA model

ICA was not mentioned.

The only assumption made by Hérault and Jutten was *statistical independence*, but additional constraints are needed on the probability distribution of the sources. Subsequent research has shown that the best performance was obtained by the Hérault-Jutten network when the source signals were sub-Gaussian, that is, for signals whose kurtosis[1] was less than that of a Gaussian distribution.

The general framework for ICA introduced by Hérault and Jutten is most clearly stated by P. Comon [14]. There are many other algorithms available in the literature. J. F. Cardoso [10] used algebraic methods, especially higher-order cumulant tensors, which eventually led to the JADE algorithm. ICA attained wider attention and growing interest after A. J. Bell and T. J. Sejnowski [5] published their approach based on the Information Maximization (InfoMAX) principle in 1995. This algorithm was further refined by S. Amari et al [15] using the natural gradient.

In 1997, A. Hyväinen and E. Oja [4] presented the fixed-point or FastICA algorithm, which has contributed to the application of ICA to large-scale problems due to its computational efficiency.

[1] In probability theory and statistics, kurtosis is a measure of the "peakedness" of the probability distribution of a real-valued random variable. The mathematical definition will be given in Section 2.4.

Since the mid-1990s, there has been a growing wave of papers, workshops, and special sessions devoted to ICA. The first international conference on ICA was held in Aussois, France, in January 1999, the second[2] followed in June 2000 in Helsinki, Finland, the third[3] in December 2001 in San Diego, California, USA, the fourth[4] in December 2002 at Whistler ski resort, Canada, the fifth[5] in September 2004 in Granada, Spain and the sixth[6] in March 2006 in Charleston, South Carolina, USA. Each gathered more than 100 researchers working on ICA and BSS, and contributed to the transformation of ICA to an established and mature field of research.

2.3 Principal Component vs. Independent Component

In this work, the problem of representing discrete-valued multidimensional variables is considered. Let $\mathbf{x}$ denote an observed m-dimensional random variable; the problem is then to find a matrix $\mathbf{W}$ so that the n-dimensional transform $\hat{\mathbf{s}} = (\hat{s_1}, \hat{s_2}, \ldots, \hat{s_n})^{\mathrm{T}}$ defined by

$$\hat{\mathbf{s}} = \mathbf{W}\mathbf{x} \tag{2.3}$$

can be estimated from the observed variable $\mathbf{x}$, where $\mathbf{x}$ is a linear mixed signal of source signal vector $\mathbf{s}$ shown as Equation 2.2 and $\mathbf{W}$ is an $n \times m$ matrix to be determined and in most cases, the linear transform of the observed variable $\mathbf{x}$ is considered.

The most popular methods for finding a linear transform as in Equation 2.3 are second-order methods, e.g. Principal Component Analysis, factor analysis and many more. Such methods use only the information contained in the covariance matrix of the data vector $\mathbf{x}$, since its distribution is completely determined by this second-order information, if the variable $\mathbf{x}$ has a normal, or Gaussian distribution.

[2]http://www.cis.hut.fi/ica2000/

[3]http://ica2001.ucsd.edu/

[4]http://www.kecl.ntt.co.jp/icl/signal/ica2003/

[5]http://ica2004.ugr.es/

[6]http://www.cnel.ufl.edu/ica2006/

Principal Component Analysis (PCA) is a common statistical method for analyzing variations within a set of data, and is known in several fields applying statistical techniques, e.g. image compression and pattern recognition. PCA is also known as the Karhunen-Loève transform (or KLT, named after Kari Karhunen and Michel Loève) or the Hotelling transform (in honor of Harold Hotelling) [2].

The aim of PCA is to find a set of n orthogonal vectors in data space that account for as much as possible the variance in the data. The data is projected from their original m-dimensional space onto the n-dimensional subspace spanned by the so called principal components[7]. In this way, the first principal component is taken to be along the direction with maximum variance. The second principal component is constrained to lie in the subspace perpendicular to the first.

In general, PCA is a decorrelation-based method that finds a linear transform $\mathbf{W}$ that satisfies Equation 2.3 so that the following three criteria are met [2]

1. The output vectors $\hat{\mathbf{s}}$ are uncorrelated.

2. The basis vectors of $\mathbf{W}$ are orthogonal to each other.

3. The eigenvalues of $\mathbf{W}$ are ordered according to eigenvalue.

Comparing to PCA, ICA provides a solution to the problem to represent multi-dimensional data by a set of their independent components. The principal components of multi-dimensional Gaussian data can be found by PCA as shown in Figure 2.5 (Left) for a 2-dimensional case. Figure 2.5 (Right) illustrates the case of non-Gaussian distribution which gives the same PCA components (p_1, p_2), but yields the basis vector (q_1, q_2) of data independence, provided that the statistical distributions of independent components are not of Gaussian.

[7] One of the basic goal in PCA is to reduce the dimension of the data, thus one usually chooses $n \ll m$.

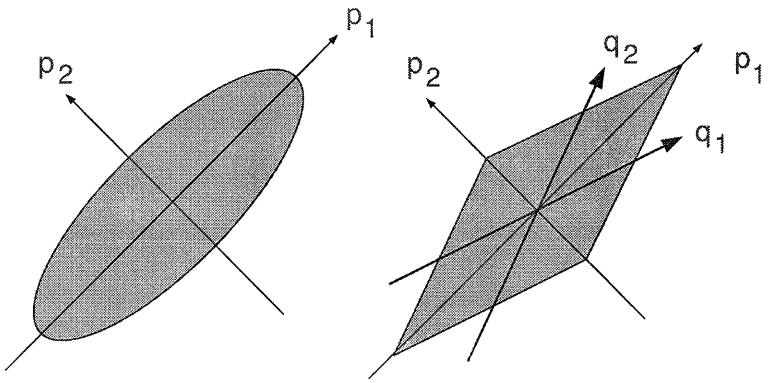

Figure 2.5 Two principal components (p_1, p_2) of two dimensional Gausian data by PCA (Left), and two independent components (q_1, q_2) of two dimensional non-Gaussian data by ICA superimposed by PCA vectors (p_1, p_2) (Right).

The reason for comparing PCA with ICA is their close relationship. Even though PCA works quite well in some cases, the situation is quite different when it comes to BSS problem. In fact, by using the well-known decorrelation methods, any linear mixture of the independent components can be transformed into uncorrelated components, in which case the mixing is orthogonal. Thus, the trick in ICA is to estimate the orthogonal transformation that is left after decorrelation. This is something that classic methods such as PCA can not estimate because they are based on essentially the same covariance information as decorrelation.

Typical algorithms for ICA use PCA or singular value decomposition as preprocessing steps in order to simplify and reduce the complexity of the actual iterative algorithm. Preprocessing ensures that all dimensions are treated equally a priori before the algorithm is run.

2.4 Independence and Nongaussianity

Consider two scalar-valued random variables s_1 and s_2, if information on the value of s_1 does not give any information on the value of s_2 and vice versa,

s_1 and s_2 are said to be *independent*.

In mathematical terms, independence is defined by probability density function (pdf). Random variables $s_1, s_2, \ldots, s_n$ are (mutually) *independent*, if the joint pdf can be factorized [6] as:

$$p(s_1, \ldots, s_n) = p_1(s_1)p_2(s_2)\ldots p_n(s_n) \tag{2.4}$$

where $p(s_1, \ldots, s_n)$ denotes the joint pdf of $s_1, s_2, \ldots, s_n$ and $p_i(s_i)$ denotes the marginal pdf of s_i. Note that independence must be distinguished from uncorrelation, which is defined by [6]

$$\mathbf{E}\Big\{(s_i - \mathbf{E}\{s_i\})(s_j - \mathbf{E}\{s_j\})\Big\} = 0, \quad \text{for} \quad i \neq j \tag{2.5}$$

or equivalently

$$\mathbf{E}\{s_i s_j\} - \mathbf{E}\{s_i\}\mathbf{E}\{s_j\} = 0, \quad \text{for} \quad i \neq j \tag{2.6}$$

where $\mathbf{E}\{\bullet\}$ denote expected value. Independence is, in general, a much stronger requirement than uncorrelation since independence implies uncorrelation, but uncorrelation does not imply independence. Indeed, if random variables $s_1, s_2, \ldots, s_n$ are said to be *independent*, it must satisfy [6]

$$\mathbf{E}\{g_1(s_i)g_2(s_j)\} - \mathbf{E}\{g_1(s_i)\}\mathbf{E}\{g_2(s_j)\} = 0, \quad \text{for} \quad i \neq j \tag{2.7}$$

for any nonlinear transformation functions $g_1(s_i)$ and $g_2(s_j)$ (in the sense that their covariance is zero). It is obvious that Equation 2.6 or Equation 2.5 is a special case where both $g_1(s_i)$ and $g_2(s_j)$ are linear only. Equation 2.7 can be proved as follows [6]

$$\begin{aligned}
\mathbf{E}\{g_1(s_i)g_2(s_j)\} &= \int\int g_1(s_i)g_2(s_j)p(s_i, s_j)\mathrm{d}s_1 s_2 \\
&= \int\int g_1(s_i)p_i(s_i)g_2(s_j)p_j(s_j)\mathrm{d}s_1 s_2 \\
&= \int g_1(s_i)p_i(s_i)\mathrm{d}s_1 \int g_2(s_j)p_j(s_j)\mathrm{d}s_2 \\
&= \mathbf{E}\{g_1(s_i)\}\mathbf{E}\{g_2(s_j)\}
\end{aligned} \tag{2.8}$$

Independence implies "nonlinear" uncorrelation, thus it gives a basic ICA estimation principle: By finding a matrix $\mathbf{W}$ for any $i \neq j$, the components

s_i and s_j are uncorrelated, as well as the transformed components $g_1(s_i)$ and $g_2(s_j)$ are uncorrelated. However, an important special case where independence and uncorrelation are equivalent, that is, $s_1, s_2, \ldots, s_n$ have Gaussian distribution. Hence, the fundamental restriction of ICA is that independent components must be nongaussian for ICA to be possible[8].

Another basic ICA estimation principle is thus given: By finding the local maxima of non-Gaussianity of a linear combination $x_i = \sum_i b_i s_i$ under the constraint that the variance of $\mathbf{x}$ is constant, each local maximum gives one independent component. This is because if $\mathbf{x}$ were a real mixture of two or more components s_i, it would be closer to a Gaussian distribution than its source components s_i, due to the central limit theorem.

Simply speaking, the key to estimate the ICA model is non-Gaussianity. Non-Gaussianity, motivated by the central limit theorem, is one method for measuring the independence of the components. The more details will be given in following sections.

2.5 Objective (Contrast) Functions for ICA

For assuring the identifiability of the ICA model, in this work, the following fundamental restrictions are imposed [2]

1. All the independent components s_i, with the possible exception of one component, must be non-Gaussian.

2. The number of observed linear mixtures m must be at least as large as the number of independent components n, i.e., $m \geq n$.

3. The mixing matrix $\mathbf{A}$ must be of full column rank so that $\hat{\mathbf{s}} = \mathbf{W}\mathbf{x}$ based on $\mathbf{x} = \mathbf{A}\mathbf{s}$).

Usually, it is also assumed that $\mathbf{x}$ and $\mathbf{s}$ are centered at zero, which is in practice no restriction, as this can always be accomplished by subtracting the

[8]Actually, if only one of the independent components is Gaussian, the ICA model can still be estimated

mean from the random vector $\mathbf{x}$.

From proceeding sections, it is known that the basic principle of ICA estimation is to find a set of estimated source signals $\hat{s}$ by maximizing non-Gaussianity. Thus the estimation of the data model of ICA is usually performed by formulating such an objective function and then minimizing or maximizing it. Often such a function is called an objective or contrast function (also the terms loss function or cost function are used). One might express this in the following 'equation':

$$\text{ICA methods} = \text{Objective function} + \text{Optimization algorithm} \tag{2.9}$$

Non-Gaussianity can be measured, for instance, by kurtosis or approximations of negentropy. Mutual information is another popular criteria for measuring statistical independence of signals.

Optimization algorithm is the subject of *optimization theory*. Most of multivariate function optimization are based on the gradients of the objective functions, which are considered in this work. Different optimization methods can be used to optimize a single objective function, and a single optimization method may be used to optimize different objective functions.

2.5.1 Mutual Information

One objective function for ICA estimation, inspired by information theory, is minimization of mutual information.

The most basic concept of information theory, entropy $H(y)$ (often called differential entropy), is defined for a continuous-valued random variable y, with pdf $p(y)$ as [5]:

$$H(y) = -\mathbf{E}\{\ln p(y)\} = -\int p(y) \ln p(y) \mathrm{d}y \tag{2.10}$$

where $\mathbf{E}\{\bullet\}$ denotes expected value. Mutual information $I(y, x)$ between the random variables y and x gives the relative entropy [5]:

$$I(y, x) = H(y) - H(y|x) \tag{2.11}$$

where y is considered as the output of a neural network processor and contains information about its input x. $H(y)$ is the entropy of the output, while $H(y|x)$ is whatever entropy the output has which didn't come from the input. Here the *gradient* of information theoretic quantities with respect to some parameter, say w, is considered. The above equation can be differentiated as follows, with respect to a parameter, w, involved in the mapping from x to y [5]:

$$\frac{\partial}{\partial w} I(y,x) = \frac{\partial}{\partial w} H(y) \tag{2.12}$$

Because $H(y|x)$ does not depend on w. No matter what the level of $H(y|x)$, maximization of the mutual information, $I(y,x)$, is equivalent to the maximization of the output entropy, $H(y)$. This principle leads to an ICA algorithm, InfoMAX, which will be discussed in following sections.

2.5.2 Kurtosis and Negentropy

Another classical measure of non-Gaussianity is kurtosis or the 4^{th}-order cumulant. The kurtosis of random variable y is classically defined by [2]

$$\mathbf{kurt}(y) = \mathbf{E}\{y^4\} - 3(\mathbf{E}\{y^2\})^2 = \mathbf{E}\{y^4\} - 3 \tag{2.13}$$

since y is assumed to have unit variance[9], i.e. $\mathbf{E}\{y^2\} = 1$. For a Gaussian y, the 4^{th} moment $\mathbf{E}\{y^4\}$ equals $3(\mathbf{E}\{y^2\})^2$. Thus kurtosis is zero for a gaussian random variable. For most non-Gaussian random variables, kurtosis is nonzero.

Kurtosis can be both positive or negative, corresponding to super-Gaussian and sub-Gaussian distribution respectively. Super-Gaussian random variables have typically a "spiky" pdf with heavy tails, e.g. Laplacian distribution, and sub-Gaussian random variables have a "flat" pdf, e.g. uniform distribution. Their pdfs are illustrated in Figure 2.6. Typically non-Gaussianity is measured by the absolute value of kurtosis.

Computationally, kurtosis can be simply estimated by using the 4^{th} moment of the sample data. Theoretically, kurtosis has the following linearity property:

[9]To simplify, y is assumed to have zero-mean and unit variance. Actually, one of the functions of preprocessing in ICA algorithms is to make this simplification possible.

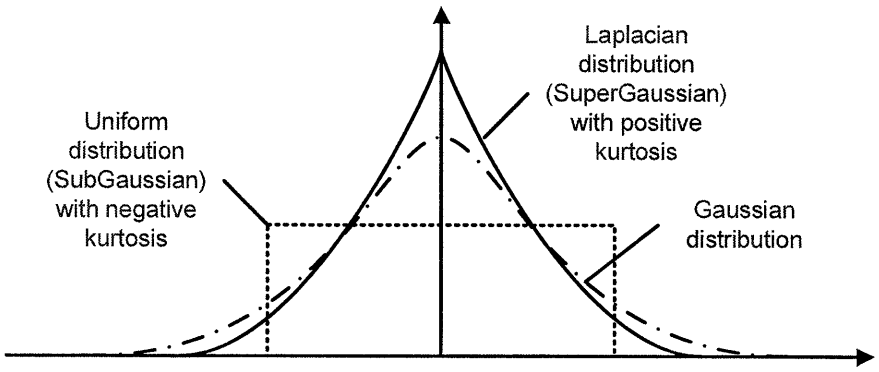

Figure 2.6 An illustration of density function with various kurtosis (modified from [2])

If x_1 and x_2 are two independent random variables, it holds [2]

$$\mathbf{kurt}(x_1 + x_2) = \mathbf{kurt}(x_1) + \mathbf{kurt}(x_2) \tag{2.14}$$

$$\mathbf{kurt}(\alpha x_1) = \alpha^4 \mathbf{kurt}(x_1) \tag{2.15}$$

where α is a scalar. However, kurtosis has also some drawbacks in practice, when its value has to be estimated from measured samples. The main problem is that kurtosis can be very sensitive to outliers. Its value may depend on only a few observations in the tails of the distribution, this means that kurtosis is not a robust measure of non-Gaussianity.

A basic result from information theory is that a variable with a Gaussian distribution has the largest entropy among all other random variables of equal variance. This means that differential entropy (calculated based on the definition of entropy given by Equation 2.10 on Page 14) could be used as a measure of non-Gaussianity. The measure should give the difference between the entropy of a Gaussian random variable y_{gauss} and a non-Gaussian random variable y. This measure is called *negentropy*, denoted by J, and is defined as

$$J(y) = H(y_{gauss}) - H(y) \tag{2.16}$$

under the requirement that y_{gauss} and y are of the same covariance matrix. The

problem in using *negentropy* is very difficult to calculate. Therefore, simpler approximations of *negentropy* are very useful.

The classical method of approximating negentropy is using higher-order cumulants, for example, as follows [4]:

$$J(y) \approx \frac{1}{12}\mathbf{E}\{y^3\}^2 + \frac{1}{48}\mathbf{kurt}(y)^2 \tag{2.17}$$

Unfortunately, the reservations made with respect to kurtosis are also valid here, since the cumulant-based approximations of negentropy are inaccurate, and in many cases too sensitive to outliers. New approximations of negenrtopy were therefore introduced. In the simplest case, new approximation is of the form [4]

$$J(y) \approx c[\mathbf{E}\{g(y)\} - \mathbf{E}\{g(v)\}]^2 \tag{2.18}$$

where g is practically any non-quadratic function, c is an irrelevant constant, and v is a Gaussian variable of zero mean and unit variance (i.e. standardized). For the practical choice of g, these approximations were shown to be more robust than the cumulant-based ones.

2.6 Algorithms for ICA

After choosing one of the objective (contrast) functions for ICA discussed in preceding section, one needs a practical method for its implementation. Usually, this means that one optimization method needs to be decided to optimize the objective function. In this section, the optimization problem will be discussed.

2.6.1 Preprocessing of the data

Typical algorithms for ICA use centering, whitening and dimensionality reduction as preprocessing steps in order to simplify and reduce the complexity of the problem for the actual iterative algorithm. The most basic and necessary preprocessing is to center the observed signal vector $\mathbf{x}$, i.e. subtract its mean vector $\mathrm{E}\{\mathbf{x}\}$ so as to make $\mathbf{x}$ a zero-mean variable. This implies that $\hat{\mathbf{s}}$ is zero-mean as well, as can be seen by taking expectations on both sides of

Equation 2.3.

Another useful preprocessing in ICA, often called whitening, is to transform the observed vector $\mathbf{x}$ linearly so that the components of a new vector $\tilde{\mathbf{x}}$ are uncorrelated and their variances equal unity, i.e. $\mathbf{E}\{\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\} = \mathbf{I}$.

The whitening transformation is always possible by performing

$$\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{x} \tag{2.19}$$

where $\mathbf{E}$ is the orthogonal matrix of eigenvectors of $\mathbf{E}\{\mathbf{x}\mathbf{x}^T\}$ and $\mathbf{D}$ is the diagonal matrix of its eigenvalues, $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$. Whitening and dimension reduction can be achieved with principal component analysis or singular value decomposition.

Next, two typical ICA algorithms, InfoMAX and FastICA will be introduced[10].

2.6.2 InfoMAX

Bell and Sejnowski [5] have shown that maximizing the joint entropy H(y) of the output of a neural network processor can approximately minimize the mutual information among the output components $y = g(u)$, where $g(u)$ is an invertible monotonic nonlinearity, e.g. $g(u) = \dfrac{1}{1+\mathrm{e}^{-u}}$ and $u = wx + w_0$.

When a single input x is passed through a transform function $g(x)$ to give an output variable y, both $I(y,x)$ and $H(y)$ are maximized when high density part of pdf of x is aligned with highly sloping parts of the function $g(x)$. When $g(x)$ is monotonically increasing or decreasing (i.e. has a unique inverse), the pdf of the output, $p_y(y)$, can be written as a function of the pdf of the input, $p_x(x)$ [6]

$$p_y(y) = \frac{p_x(x)}{|\partial y/\partial x|} \tag{2.20}$$

substituting Equation 2.20 into Equation 2.10 gives [5]

$$H(y) = E\left\{\ln\left|\frac{\partial y}{\partial x}\right|\right\} - \mathbf{E}\{\ln p_x(x)\} \tag{2.21}$$

[10]In this work, it is aimed for application of algorithms. The details of algorithms won't be given.

where the entropy is measured with natural logarithm. The second term on the right (the entropy of x) may be considered to be unaffected. Therefore in order to maximize the entropy of y by changing w, it only needs to maximize the first term , which is the average log of how the input affects the output. This can be done by considering the 'training set' of x's to approximate the density $p_x(x)$, and deriving an 'online', stochastic gradient ascent learning rule [5]

$$\Delta \mathbf{W} \propto \frac{\partial H}{\partial w} = \frac{\partial}{\partial w}\left(\ln\left|\frac{\partial y}{\partial x}\right|\right) = \left(\frac{\partial y}{\partial x}\right)^{-1} \frac{\partial}{\partial w}\left(\frac{\partial y}{\partial x}\right) \tag{2.22}$$

In the case of the logistic transfer function [5]

$$y = \frac{1}{1+\mathrm{e}^{-u}}, \; u = wx + w_0 \tag{2.23}$$

in which the input x is multiplied by a weight w and added to a bias-weight w_0, and then we have

$$\frac{\partial y}{\partial x} = wy(1-y) \tag{2.24}$$

$$\frac{\partial}{\partial w}\left(\frac{\partial y}{\partial x}\right) = y(1-y)\left[1 + wx(1-2y)\right] \tag{2.25}$$

then the learning rule for the logistic function for one input and one output is given [5]

$$\Delta w \propto \frac{1}{w} + x(1-2y) \tag{2.26}$$

as well as the rule for the bias weight

$$\Delta w_0 \propto 1 - 2y \tag{2.27}$$

for the input vector $\mathbf{x}$, a weight matrix $\mathbf{W}$, a bias vector $\mathbf{w_0}$ and a monotonically transformed output vector $\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{w_0})$, the resulting learning rules are familiar in form [5]

$$\Delta \mathbf{W} \propto \frac{1}{\mathbf{W}^T} + \mathbf{x}^T(1 - 2\mathbf{y})$$

$$\Delta \mathrm{w}_0 \propto 1 - 2\mathrm{y} \tag{2.28}$$

2.6.3 FastICA

A practical ICA algorithm, FastICA algorithm, is a fixed-point iteration scheme for finding a maximum of the non-Gaussianity or negentropy of $\mathbf{w}^T\mathbf{x}$ purposed by A. Hyvärinen and E. Oja [3] [4]. Its objective function is shown in the Equation 2.18 on Page 17. Since the objective function is the measure of non-Gaussian, many practical function could be used. To begin with, the Equation 2.18 on Page 17 could be modified as follow

$$J_G(y) = |\mathbf{E}\{G(y)\} - \mathbf{E}\{G(v)\}|^p \tag{2.29}$$

Note that the notation J_G should not be confused with the notation for negentropy, J, and the exponent $p = 1, 2$ typically.

Clearly, J_G can be considered a generalization of kurtosis. For $G(y) = y^4$, J_G becomes simply the modulus of kurtosis of y. Note that G must not be quadratic, because then J_G would be trivially zero for all distributions. Thus, it seems plausible that J_G could be a contrast function in the same way a kurtosis. The fact, for $p = 2$, J_G is coincides with the approximation of negentropy given in Equation 2.18 on Page 17.

In [6], the finite-sample statistical properties of the estimators based on optimizing such a general contrast function were analyzed. It was found that for a suitable choice of G, the statistical properties of the estimator (asymptotic variance and robustness) are considerably better than the properties cumulant-based estimators. The following choices of G were proposed [4]:

$$G_1(u) = \log \cosh a_1 u \tag{2.30}$$

$$G_2(u) = \exp\left(-\frac{a_2 u^2}{2}\right) \tag{2.31}$$

where $a_1, a_2 \geq 1$ are some suitable constants. Experimentally, it was found that especially the values $1 \leq a_1 \leq 2$, $a_2 = 1$ for the constants give good approximations. This approximations of negentropy gives a very good compromise between the properties of the two classical nongaussianity measures given by kurtosis and negentropy.

For one computational unit or neuron, the basic FastICA scheme is then as follows [4]

1. Choose an initial (e.g. random) weight vector $\mathbf{w}$
2. Let $\mathbf{w}^+ = \mathbf{E}\{\mathbf{x}g(\mathbf{w}^T\mathbf{x})\} - \mathbf{E}\{g'(\mathbf{w}^T\mathbf{x})\}\mathbf{w}$, where the nonquadratic function g is $g(u) = u^3$
3. Let $\mathbf{w} = \mathbf{w}^+/\|\mathbf{w}^+\|$
4. If not converged, go back to 2

Note that convergence means that the old and new values of $\mathbf{w}$ point in the same direction, i.e. their dot-product is (almost) equal to 1. The algorithm estimates just one of the independent components once. To estimate several independent components, the FastICA need to be run several times.[11]

2.7 Simulation of ICA

ICA can be applied to a variety of problems involving separation of mixed signals, multivariate analysis and almost every case where PCA is used. In this section, two ICA methods, InfoMAX discussed in Section 2.6.2 and FastICA discussed in Section 2.6.3 are simulated. Each algorithm was applied to two BSS applications. The first application is a *Cocktail Party Problem* which consists of 4 speakers and the same number of microphones. The second application is a demonstration of the example shown in Figure 2.1 on Page 5, and Figure 2.2 on Page 6.

2.7.1 InfoMAX

The principle of the algorithm is discussed in Section 2.6.2. The algorithm has been simulated with MATLAB, and its flowchart is shown in Figure 2.7.

At first, the *Cocktail Party Problem* which consists of 4 speakers is applied. The experiments presented here were obtained using speech segments recorded

[11] At `http://www.cis.hut.fi/projects/ica/fastica/`, the FastICA MATLAB package is available.

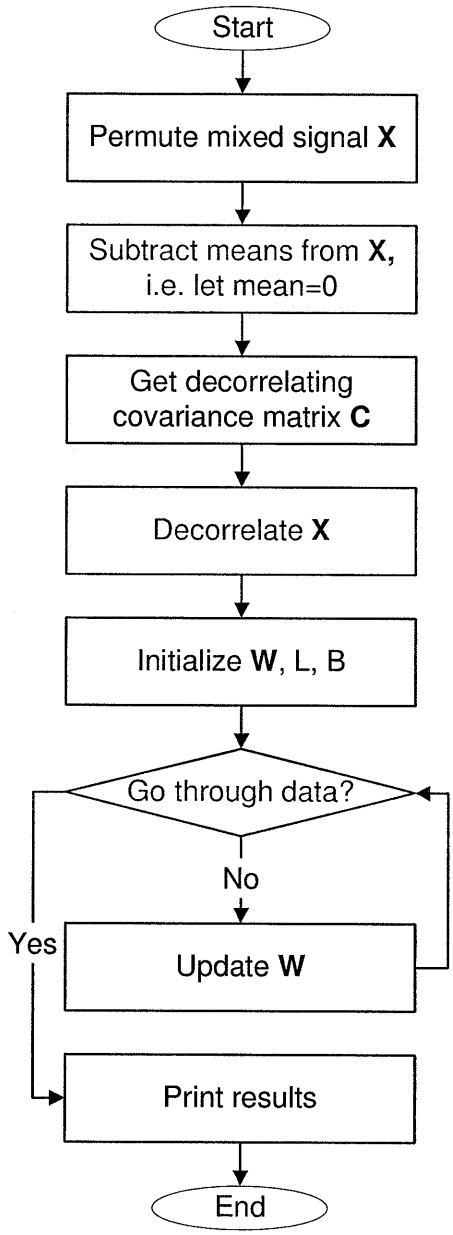

Figure 2.7 Flowchart of InfoMAX algorithm

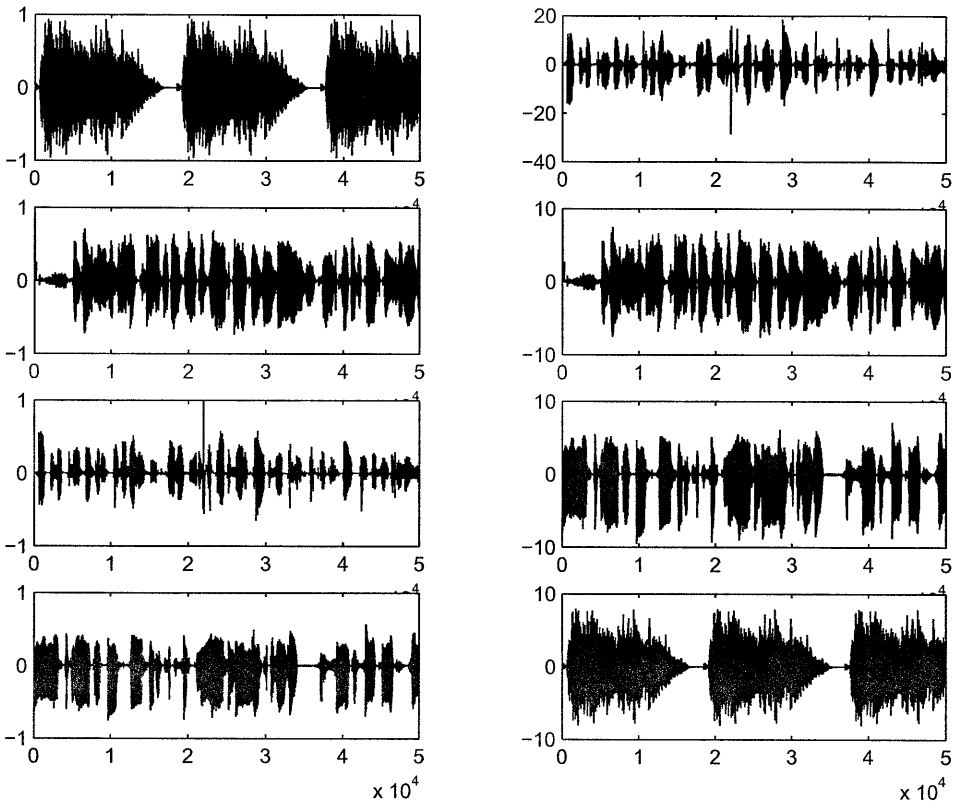

Figure 2.8 The comparison of source signals and estimates

from various speakers. All signals were sampled at 8 kHz with 50000 sample points. The method of training was stochastic gradient ascent, but the learning rule is multiplied by $\mathbf{W}^{\mathrm{T}} \times \mathbf{W}$, where $\mathbf{W}$ is unmixed matrix estimated by InfoMAX algorithm, as proposed by Amari, Cichocki and Yang [15]. This 'natural gradient' method speeds convergence and avoids the matrix inverse in the learning rule. Weights were usually adjusted based on the summed $\Delta\mathbf{W}$'s of small 'batches' of length $\mathbf{B}$, where $\mathbf{B}$=30 and various learning rates[12] were used (0.01 was typical). To ensure that the input ensemble was stationary in time, the time index of the signals was permuted.

An example is run with 4 speech sources. The mixtures, $\mathbf{x}$, is formed with a random mixture matrix $\mathbf{A}$. The unmixed solution was reached and is reflected

[12]The learning rate is defined as the proportionality constant in Equation 2.28 on Page 19

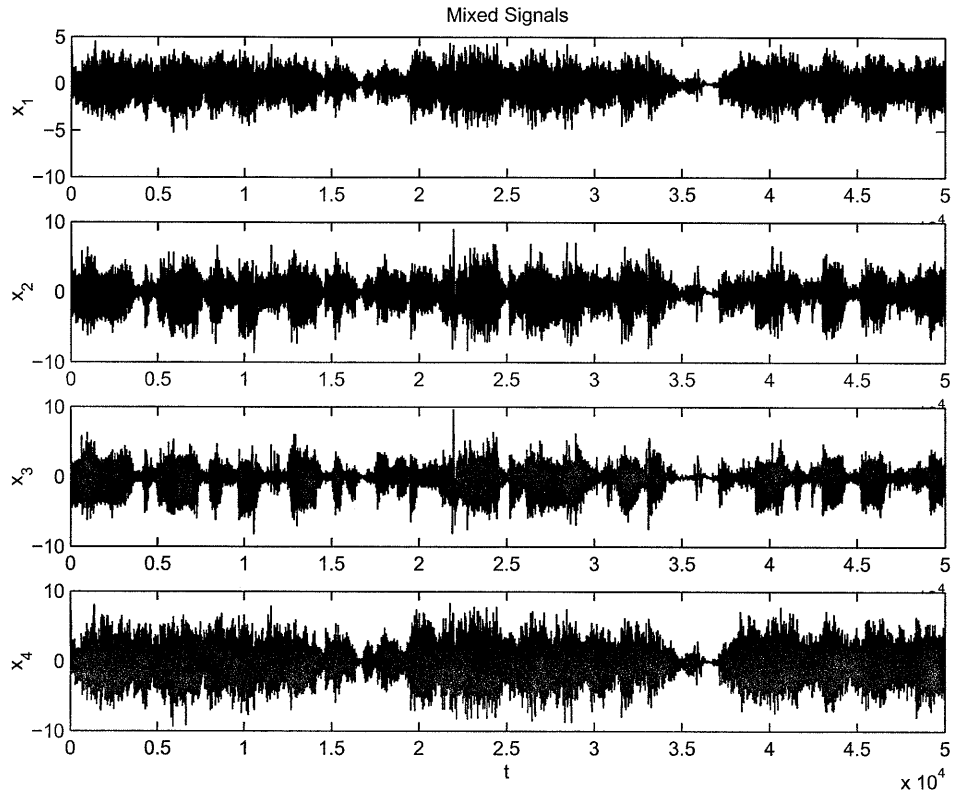

Figure 2.9 Four mixtures of the source signals in Figure 2.8

in the permutation structure of the matrix **WA**. As one typical result out of many trials, matrix **WA** is shown below

$$\mathbf{WA} = \begin{pmatrix} 0.0157 & 0.0809 & \boxed{\mathbf{-28.7406}} & -0.0966 \\ -0.0510 & \boxed{\mathbf{10.3998}} & 0.0523 & -0.1968 \\ -0.0149 & -0.0609 & -0.0647 & \boxed{\mathbf{12.5722}} \\ \boxed{\mathbf{8.4311}} & 0.0293 & -0.1328 & 0.0317 \end{pmatrix} \tag{2.32}$$

as can be seen, only one substantial entry (boldface and boxed) exists in each row and column. The interference was attenuated by between 34.5 and 65.3dB. The original source signals and their estimates are shown in Figure 2.8, and four observed mixtures are shown in Figure 2.9. Since both $\mathbf{s}$ and $\mathbf{A}$ are unknown in equation $\mathbf{x}=\mathbf{As}$, any scalar multiplier in one of the sources s_i could always be canceled by dividing the corresponding column a_i of $\mathbf{A}$ by the same scalar; Consequently, the variances (magnitudes or energies) of the

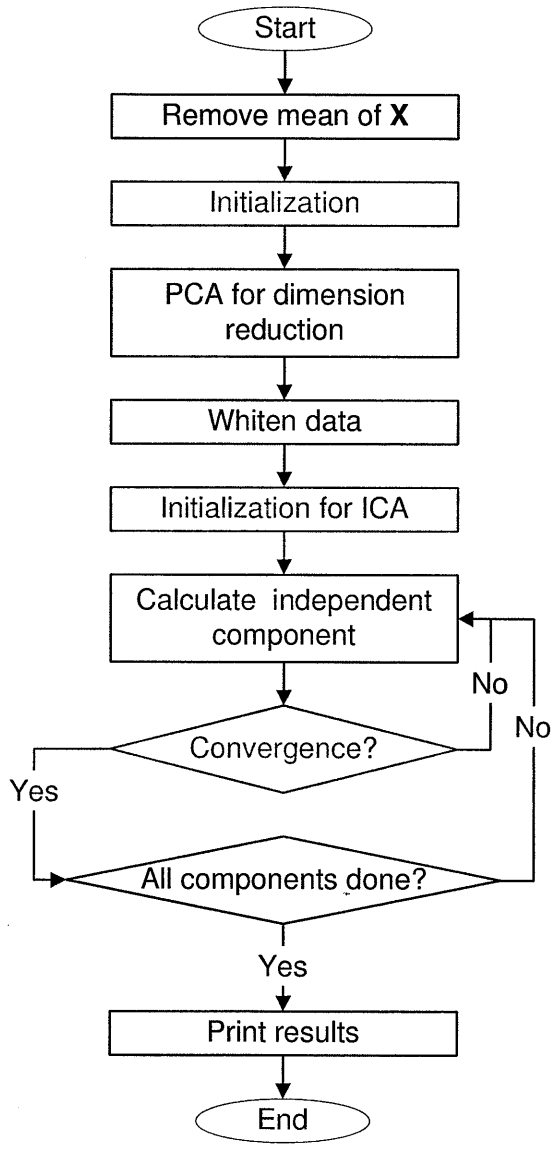

Figure 2.10 Flowchart of FastICA Algorithm

independent components cannot be determined. Note that the ambiguity of the sign is left too, when multiplying an independent component by -1 [2].

Secondly, the mixed sources, shown in Figure 2.2 on Page 6 are applied. Unfortunately, the unmixed solution cannot be reached, since the objective function of InfoMAX works for estimation of most super-Gaussian independent components; but for sub-Gaussian independent components, other objective functions must be used. Because two sub-Gaussian components existed in sample data, the attempt to unmix the independent components failed.

2.7.2 FastICA

Two BSS problems that are the same as those in preceding section, have been applied to FastICA algorithm as well. The MATLAB code is obtained from A. Hyvärinen's FastICA MATLAB package with default parameters, and its flowchart is shown in Figure 2.10.

At first, the speech signal test is applied. The mixtures, $\mathbf{x}$, is formed with a random mixture matrix $\mathbf{A}$. The unmixed solution can be reached. When the same mixture matrix, $\mathbf{A}$, as that in the trial of the Equation 2.32 on page 24, is applied, the corresponding matrix $\mathbf{WA}$ is shown below

$$\mathbf{WA} = \begin{pmatrix} \boxed{\mathbf{4.5418}} & 0.0356 & -0.0362 & 0.0324 \\ 0.0291 & 0.0836 & 0.0962 & \boxed{\mathbf{-6.2912}} \\ -0.0084 & 0.1576 & \boxed{\mathbf{-12.1829}} & -0.0848 \\ 0.0372 & \boxed{\mathbf{-5.5457}} & -0.3557 & -0.0648 \end{pmatrix} \tag{2.33}$$

where $\mathbf{W}$ is unmixed matrix. as can be seen, only one substantial entry (boldface and boxed) exists in each row and column. The interference was attenuated by between 23.9 and 63.2dB. Note that the order of the independent components must not be the same as the order in Equation 2.32 on page 24 because of the ambiguity of the ICA. The reason is that, in equation $\mathbf{x}=\mathbf{As}$, both $\mathbf{s}$ and $\mathbf{A}$ are unknown. The order of the terms can be freely changed, and any of the independent components can be the first one [2]. In the most ambitious attempt, nine sources were successfully separated.

The source example shown in Figure 2.1 on Page 5, and the mixed source shown in Figure 2.2 on Page 6 are applied, the unmixed solution shown in Figure 2.3 on Page 7 is reached. It proves that FastICA is more robust and has faster convergence than the InfoMAX method.

2.8 Summary of ICA Algorithm

PCA is a classical tool within multivariate analysis. However, PCA is not well suited to separate statistically independent signals that are mixed. This is because uncorrelated variables must not be independent.

Independent Component Analysis (ICA) is a general purpose statistical technique in which observed random data is separated in a way that makes the estimates maximally independent from each other under the assumption that not more than one of the sources has a Gaussian distribution.

ICA can be formulated with different algorithms, and its basic principle is mainly based upon nonlinear decorrelation by maximizing or minimizing negentropy or maximizing mutual information. In this section, no account has been given for cases where there is known noise in the inputs, so it impacts slightly the accuracy of the solution.

The InfoMAX algorithm presented in section 2.6.2 is limited. Firstly, since only single layer networks are used, the optimal mappings discovered are constrained to be linear, while some multi-layer system could be more powerful. Secondly, the diversified objective function can improve the performance in various problem. Despite these concerns, the information maximization approach serves as a guiding principle for further advances.

FastICA package is a good multi-purpose implementation of ICA. This algorithm is based on the fixed-point method and has the following advantages

- Fast convergence.
- Contrary to gradient-based algorithms, like InfoMAX, there is no learning rate or other adjustable parameters in the algorithm.

- Suitable to both super-Gaussian and sub-Gaussian components, as well as the mixtures of them.

In summary, the basic principles of ICA have been discussed. It has been demonstrated that ICA can solve the cocktail party problem and some other basic BSS problems. Two typical methods, InforMAX and FastICA, have been implemented.

In spite of the large amount of research conducted on this basic problem by several authors, this area of research is by no means exhausted. ICA has found many applications in diversified fields, like blind separation of electroencephalographic (EEG) and magnetoencephalographic (MEG), economic time series analysis, feature extraction of image and so forth. Recently, applications on telecommunications have also been published, where ICA is useful to separate the user's own signal from the interfering other users' signals in DS-CDMA. The use of ICA as a post-processing tool for conventional array receiver shows that is capable of mitigating the continuous-wave jamming in DS-CDMA system, especially with moderate/high signal to noise radio values [17]. It is an area of great potential. To this end, nonlinear transformation, higher number of estimated parameters, and signals separation in the presence of noise must be researched in further.

3. DIRECT SEQUENCE CODE DIVISION MULTIPLE ACCESS (DS-CDMA)

Design and development of future generation wireless systems is one of the most active and important areas of research and development in wireless communications. In November of 1999, the International Telecommunication Union (ITU) approved an industry standard for third-generation (3G) wireless networks. This standard, called International Mobile Telecommunications-2000 (IMT-2000), consists of 5 operating modes, including 3 based on CDMA technology. They are most commonly known as CDMA2000, WCDMA (UMTS) and TD-SCDMA. The standard defines three radio interfaces enabling interoperability: IMT-MC (Multi-Carrier), IMT-DS (Direct Spread) and IMT-TC (Time Code).

Code Division Multiple Access (CDMA) was originally developed in World War II for secure military communications. Today, CDMA offers significant advantages over other analog and digital cellular technologies. CDMA is a modulation and multiple-access scheme based on spread-spectrum communication. In this scheme, multiple users share the same frequency band simultaneously, by spreading the spectrum of their transmitted signals, so that each user's signal is pseudo-orthogonal to the signals of the other users.

3.1 Multiple Access

Multiple Access refers to the sharing of a communications resource, or in other word, the common transmission medium such as time, frequency among several users. The sharing of a communications resource is an essential question particularly in wireless communication systems since wireless is a broadcast medium. However, as the number of users in the system grows, the demand of using the common resources as efficiently as possible also arise.

Thus multiple access schemes are developed for this purpose. The most common multiple access schemes are [12]:

1. Frequency Division Multiple Access (FDMA): Each user is given a frequency slot in which one and only one user is allowed to operate. Interference of other users is thus easily prevented by assigning slots that do not overlap.

2. Time Division Multiple Access (TDMA): A simmilar idea is realized in the time domain, where each user is given a unique time period. One user can hence transmit and receive data only during its own time slot while others are silent.

3. Code Division Multiple Access (CDMA): As distinct from both FDMA and TDMA, each user occupies the same frequency band simultaneously. The users are now identified by so called codes, which are unique to each other. The signals of different users are first applied by its unique codes and then get mixed together before they are transmitted through a common medium. The unique codes, however, ensure that the signal of each user can be resolved at receiver.

In the simplest form, the code of CDMA is a sequence of ± 1's, also called chip sequence. This is so called Direct sequence spread spectrum, also known as Direct Sequence Code Division Multiple Access (DS-CDMA). DS-CDMA is one of three approaches to spread spectrum modulation for digital signal transmission [12]:

- Frequency Hopping (FH). The signal is rapidly switched between different frequencies within the hopping bandwidth pseudo-randomly, and the receiver knows before hand where to find the signal at any given time.

- Time Hopping (TH). The signal is transmitted in short bursts pseudo-randomly, and the receiver knows beforehand when to expect the burst.

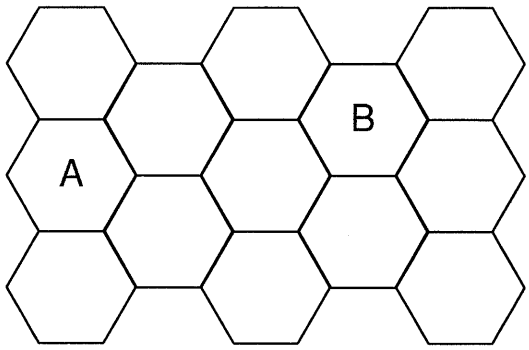

Figure 3.1 Cells in cellular communication system

- Direct Sequence (DS). The digital data is directly coded at a much higher bandwidth. The code is generated pseudo-randomly, the receiver knows how to generate the same code, and correlates the received signal with that code to extract the data.

Two types of digital modulation, Phase-shift keying (PSK) and Frequency-shift keying (FSK), are usually in conjunction with Spread Spectrum system. PSK is generally used with Direct Sequence Spread Spectrum (DSSS) system and FSK is commonly used with FH Spread Spectrum system.

Since available radio frequencies are limited resource, it should be used as efficiently as possible. This brings the question of frequency planning. In Figure 3.1, each cell (illstrated as a hexagon) corresponds to a geographical area served by a base station. In a randomly picked cell, A, certain predetermined frequencies are assigned. In all the adjacent cells of cell A, the frequencies assigned to them must not be same as the frequency slot of cell A.

Due to the availability of the frequencies, it would be desired to reuse the same frequencies in another existing cell, e.g. cell B. In FDMA or TDMA system, the reuse pattern must be carefully designed to prevent different cells interfering with each other. Or in other word, cells A and B must be far apart so that the signals from cell B are attenuated enough as they are received in cell A and vice versa. In CDMA system, no such design problems exists, since the same frequencies are used. This is the main advantage achieved by

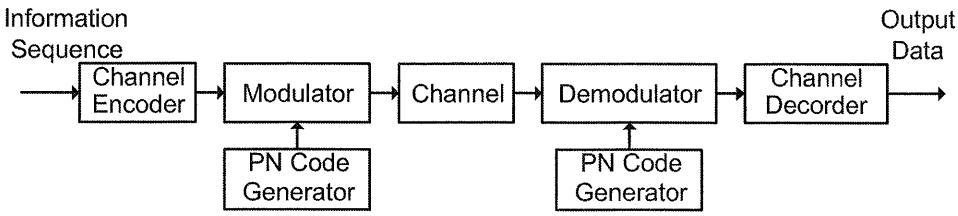

Figure 3.2 Model of Spread Spectrum Communications

CDMA, increasing the system capacity, .

3.2 Spread Spectrum

The basic elements of a spread spectrum digital communication system are illustrated in Figure 3.2. In general, Spread Spectrum communications is distinguished by three keys elements [12]:

1. The signal occupies a bandwidth much greater than that is necessary to send the information. This results in many benefits, such as immunity to interference and jamming and multi-user access.

2. The bandwidth is spread by means of a code which is independent of the data.

3. The receiver synchronizes to the code to recover the data. The use of independent codes and synchronous reception allows multiple users to access the same frequency band at the same time.

In order to protect the signal, the code used is pseudo-random. It appears random, but is actually deterministic, so that the receiver can reconstruct the code for synchronous detection. This pseudo-random code is also called Pseudo-Noise (PN) code. The code helps the signal resist narrow band jamming or interference and also enables the original data to be recovered if chip bits are damaged during transmission. Time synchronization of two identical pseudo-random sequence generators in Figure 3.2 is also required. A fixed PN bit pattern is designed and transmitted prior to the transmission of information so that the receiver will detect it with high probability in the presence of

interference.

In practice, since the codes may be as long as $2^{42} - 1$, it is difficult to eavesdrop. It is, however, possible. Hence, the spread spectrum methods does not offer security in a strict sense. Encryption is still necessary.

3.3 Signal Model of A DSSS System

In next two sections, the signal model of a DSSS system is briefly introduced and the transmission of a DSSS signal by means of binary PSK is considered [13].

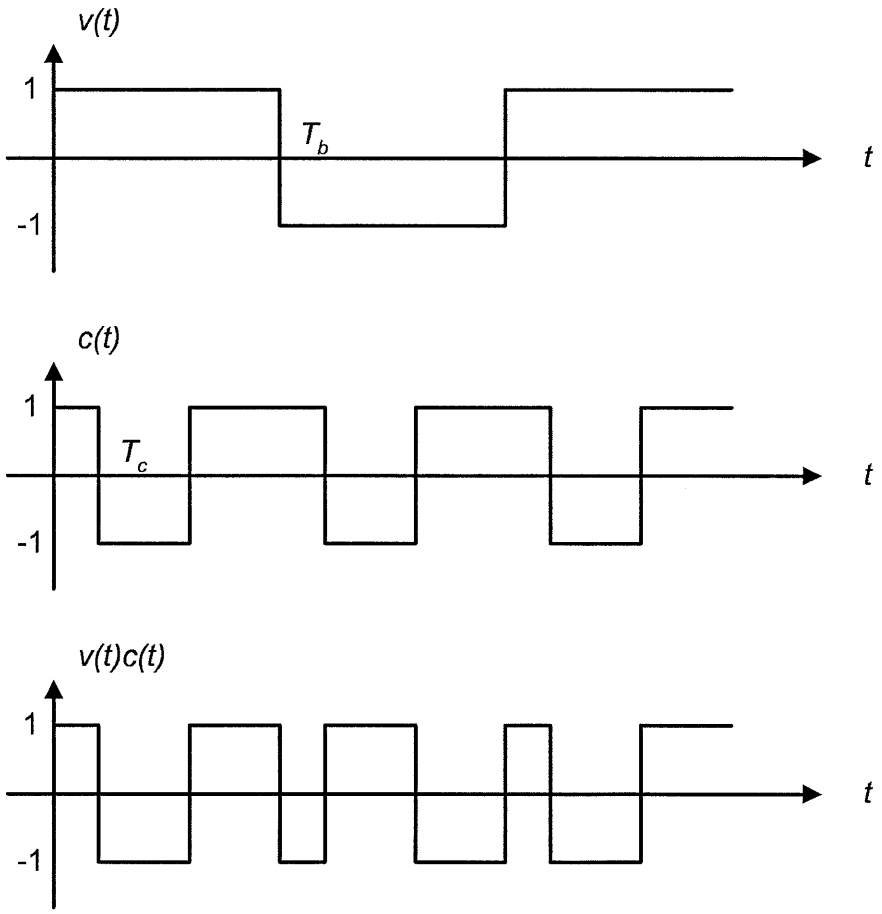

Figure 3.3 Generation of a DSSS signal

3.3.1 Signal Modulation

In Figure 3.3, the information-bearing baseband signal is denoted as $v(t)$ and can be expressed as

$$v(t) = \sum_{m=-\infty}^{\infty} b_m g(t - mT_b) \tag{3.1}$$

where

$\{b_m = \pm 1,\ -\infty < m < \infty\}$ is the information-bearing symbol stream

$g(t)$ is a rectangular pulse of duration T_b

This signal is then multiplied by the PN signal $c(t)$ generated from the PN sequence genertor

$$c(t) = \sum_{n=-\infty}^{\infty} c_n p(t - nT_c) \tag{3.2}$$

where

$\{c_n = \pm 1,\ -\infty < n < \infty\}$ is the binary PN sequence

$p(t)$ is a rectangular pulse of duration T_c.

The bandwidth of $c(t)$ is approximately $\frac{1}{T_c}$, and the bandwidth of $v(t)$ is approximately $\frac{1}{T_b}$, where $\frac{1}{T_c} \gg \frac{1}{T_b}$. This multiplication operation spreads the bandwidth of $v(t)$ to the much wider bandwidth of PN signal $c(t)$. In other words, it is equivalent to convolution operation in the frequency domain, i.e. the narrow spectrum signal, $V(f)$, is spread to a much wider spectrum $V(f) * C(f)$ by the PN signal $C(f)$, which are illustrated in Figure 3.4.

The product signal $v(t)c(t)$ is modulated by the carrier $A_c cos 2\pi f_c t$ and thus the Double Sideband Suppressed Carrier (DSB-SC) signal is

$$u(t) = A_c v(t) c(t) cos 2\pi f_c t \tag{3.3}$$

since $v(t)c(t) = \pm 1$ for any t, then $u(t)$ may also be expressed as

$$u(t) = A_c cos(2\pi f_c t + \theta(t)) \tag{3.4}$$

where $\theta(t) = 0$ when $v(t)c(t) = 1$ and $\theta(t) = \pi$ when $v(t)c(t) = -1$. Therefore, the $u(t)$ is a BPSK signal whose phase varies at the rate $\frac{1}{T_c}$.

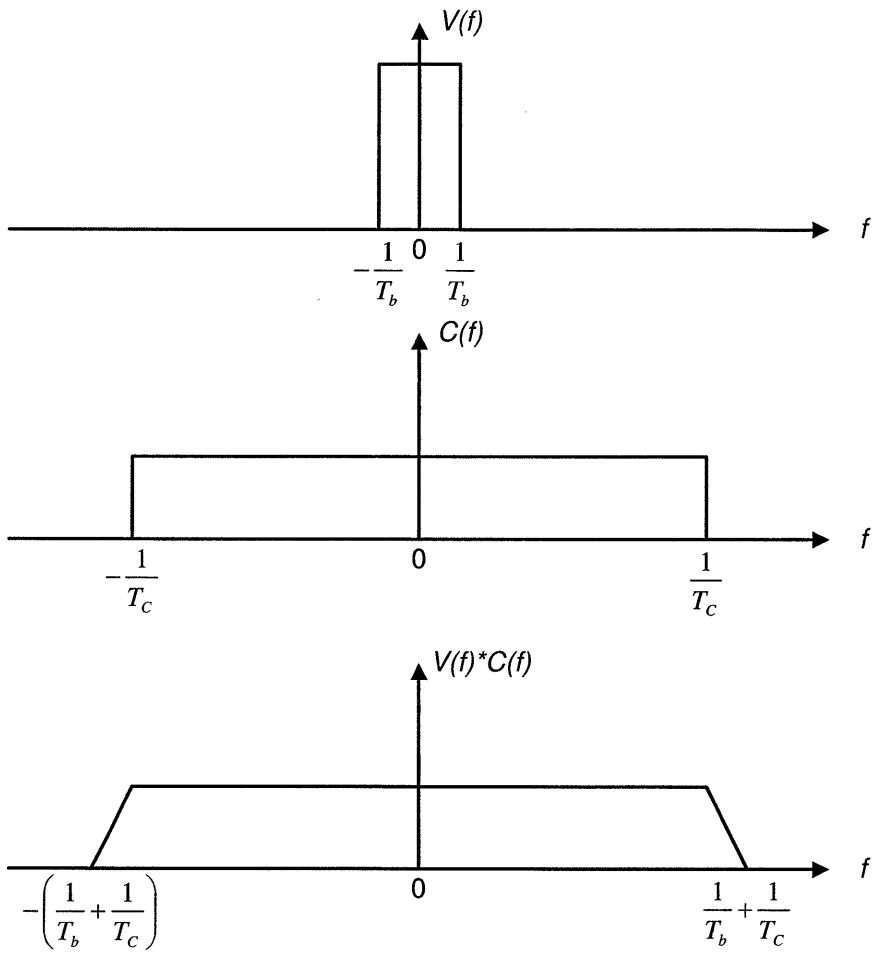

Figure 3.4 Convolution of spectra $V(f)$ and $C(f)$

3.3.2 Signal Demodulation

The signal demodulation is performed as illustrated in Figure 3.5. The received signal $r(t)$ is first multiplied by a synchronized local PN signal $c(t)$ at the receiver. This operation is called *spectrum despreading.* since the effect of multiplication by $c(t)$ at the receiver is to undo the spreading operation at the transmitter. Thus we have

$$r(t)c(t) = A_c v(t)c^2(t)cos2\pi f_c t \tag{3.5}$$

since $c^2(t) = 1$ for all t. The demodulator for the despread signal,

$$r(t)c(t) = A_c v(t)cos2\pi f_c t \tag{3.6}$$

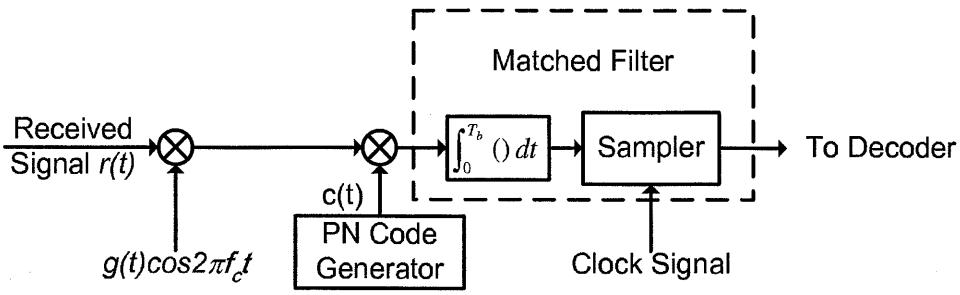

Figure 3.5 Demodulation of DS spread spectrum signal

is simply the matched filter.

3.4 Signal Model of DS-CDMA System

There are two main applications of DSSS system. First, the spread spectrum signal is transmitted at very low power so that a listener would encounter great difficulty in trying to detect the presence of the signal. We discussed the signal model of this system in last section. A second application is multiple-access radio communications, i.e. DS-CDMA system. As shown in Figure 3.3, the bit and chip durations are $T_b = 5$ and $T_c = 1$, respectively. The processing gain is thus determined by ratio $C = \frac{T_b}{T_c}$. For the orthogonal synchronous DS-CDMA systems, it gives a theoretical upper limit for the amount of users that can be supported.

In a DS-CDMA system, users share the same band of frequencies and the same time slots, but they are separated in code. The main sources of errors at the detector are due to the Multi-Access Interference (MAI). The conventional Inter-Symbol Interference (ISI) and channel noise are certainly the other sources of errors.

In Equation 3.1 and Equation 3.2, $c(t)$ is usually called Chip Sequence. In a practical DS-CDMA system, its length is not from $-\infty$ to ∞ and duration $T_c = \frac{T_b}{C}$.

We consider the signal after demodulation in this work. In Figure 3.5, let us denote m^{th} data symbol (information bit) by b_m, and the chip sequence with finite length C by $s(t)$. The number of bits in the observation interval is

denoted by M. Then we have

$$r(t) = \sum_{m=0}^{M} b_m s(t - mT_b) \tag{3.7}$$

where $r(t)$ denotes the transmitted CDMA baseband signal of one single user. The signal model of multi-user can be obtained and will be discussed in next chapter.

3.5 Probability of Error of DSSS System

In an Additive White Gaussian Noise (AWGN) downlink channel, let us consider a spreading signal with BPSK modulation. If we consider the signal after spreading as the transferred bit stream, the probability of error for this system with is identical to the probability of error for conventional BPSK, that is

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{3.8}$$

where E_b is power of signal and N_0 is power of noise.

On the basis of *Central Limit Theorem*, the MAI from other users in a DS-CDMA system can be approximately treated as the AWGN signal[1], when the number of user is big, e.g. 30. As mentioned above, the probability of error for the bit stream after spreading for one single user is therefore approximately equal to the probability of error for conventional BPSK, if we define the power of single user's signal as the power of signal, and the sum of the power of other users' signals and the power of AWGN as the power of noise, when we define Signal-to-Noise Ratio (SNR) in this calculation. The simulation is shown in following chapters.

[1]In practice, the PN code used in CDMA system usually are not orthogonal. Then MAI will include the effect of it

4. APPLICATION OF ICA TO DS-CDMA DETECTION

In last chapter, we briefly introduced the signal model of DS-CDMA system. The system model of ICA based multiuser detection in downlink environment of DS-CDMA is discussed in what follows.

4.1 Signal Model of DS-CDMA Downlink

In this section, we represent mathematically the DS-CDMA signal model, in a mobile receiver over an Additive White Gaussian Noise (AWGN) channel, the received signal (baseband) $r(t)$ can be easily described for K users and m symbols from Equation 3.7

$$r(t) = \sum_{k=1}^{K} \sum_{m=0}^{M} b_{km} s_k(t - mT_b) + n(t) \tag{4.1}$$

where $n(t)$ denotes additive noise. The signal model is not yet realistic, because it does not consider the effect of multipath propagation and fading. Our desired signal model including these factors therefore has the form

$$r(t) = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{m=0}^{M} a_{lm} b_{km} s_k(t - mT_b - d_l T_c) + n(t) \tag{4.2}$$

where

$l, k, m \Rightarrow$ path, user and symbol indices, respectively

$a_{lm} \Rightarrow$ path gain[1]

$b_{km} \Rightarrow$ symbol

$s_k(\cdot) \Rightarrow$ spreading code (chip sequence)

$C \Rightarrow$ number of chips per symbol

$t, T_b, T_c \Rightarrow$ time, symbol and chip duration, respectively

[1]In downlink model, path gain does not differ among the users since all the users' signal is sent together and the path gain a_{lm} and delay factor d_l only depend on the path.

$d_l \Rightarrow$ propagation delay factor, and $d_l \in \{0, \cdots, (C-1)/2\}$

$n(t) \Rightarrow$ Gaussian noise[2]

The signal is then sampled and chip-rate sampling is assumed. Here both code timing and channel estimation are often prerequisite tasks. The discrete data samples

$$r[n] = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{m=0}^{M} (a_{l,m-1} b_{k,m-1} \underline{\mathbf{s}}_{kl}[n] + a_{lm} b_{km} \overline{\mathbf{s}}_{kl}[n]) + n[n] \tag{4.3}$$

where $n[n]$ denotes the discrete noise and the "late" and "early" parts of the code vectors are

$$\begin{aligned} \underline{\mathbf{s}}_{kl}[n] &= [s_k(C - d_l + 1) \ \cdots \ s_k(C) \ \ 0 \ \cdots \ 0]^T \\ \overline{\mathbf{s}}_{kl}[n] &= [0 \ \cdots \ 0 \ \ s_k(1) \ \cdots \ s_k(C - d_l)]^T \end{aligned} \tag{4.4}$$

Having sampled, the samples are collected into $C \times M'$ matrix $\mathbf{R}$ from subsequently discrete data samples $r[n]$:

$$\mathbf{R} = \begin{pmatrix} r(mC) & r(mC+C) & \dots & r(m'C) \\ r(mC+1) & r(mC+C+1) & \dots & r(m'C+1) \\ \vdots & \vdots & \ddots & \vdots \\ r(mC+C-1) & r(mC+2C) & \dots & r(m'C+C-1) \end{pmatrix} \tag{4.5}$$

or in C-vectors $\mathbf{r}_m$ format

$$\mathbf{R} = [\mathbf{r}_m \ \mathbf{r}_{m+1} \ \dots \ \mathbf{r}_{m'}] \tag{4.6}$$

where $m' > m$, $M' = m' - m + 1$ and $m, m' \in \{0, \cdots, M\}$. The length M' is required by ICA algorithm and usually much greater than length of chips C. Although in practice, the length of chip codes C can be very long, M' still has to be greater than C. If the delay is shorter than a symbol duration, the length M' can be as shorter as requested by ICA algorithm. Then data vector $\mathbf{r}_m$ has the form

$$\begin{aligned} \mathbf{r}_m &= \sum_{k=1}^{K} \sum_{l=1}^{L} [a_{l,m-1} b_{k,m-1} \underline{\mathbf{s}}_{kl} + a_{lm} b_{km} \overline{\mathbf{s}}_{kl}] + \mathbf{n}_m \\ &= \sum_{k=1}^{K} \left[b_{k,m-1} \sum_{l=1}^{L} a_{l,m-1} \underline{\mathbf{s}}_{kl} + b_{km} \sum_{l=1}^{L} a_{lm} \overline{\mathbf{s}}_{kl} \right] + \mathbf{n}_m \end{aligned} \tag{4.7}$$

[2]The noise is the sum of all the background noise and will be considered as the last independent component.

where $\mathbf{n}_m$ denotes noise vector. The Equation 4.7 can be represented in more compact form

$$\mathbf{r}_m = \mathbf{G}\mathbf{b}_m + \mathbf{n}_m \tag{4.8}$$

where the $C \times 2KL$ matrix $\mathbf{G}$ contains all the $2KL$ "early" and "late" code vectors and fading terms

$$\mathbf{G} = [\underline{\mathbf{s}}_{11}, \overline{\mathbf{s}}_{11}, \cdots, \underline{\mathbf{s}}_{KL}, \overline{\mathbf{s}}_{KL}] \tag{4.9}$$

and the $2KL$ vector $\mathbf{b}_m$ contains the symbols

$$\mathbf{b}_m = [a_{1,m-1}b_{1,m-1}, a_{1m}b_{1m}, \cdots, a_{L,m-1}b_{K,m-1}, a_{Lm}b_{Km}] \tag{4.10}$$

The $C \times M'$ data matrix $\mathbf{R}$ then can be represented in pure matrix form

$$\mathbf{R} = \mathbf{G}\mathbf{B} + \mathbf{N} \tag{4.11}$$

where matrices $\mathbf{G}$, $\mathbf{B}$ and $\mathbf{N}$ denote the unknown mixed matrix, symbols and noise.

$$\begin{aligned} \mathbf{R} &= [\mathbf{r}_m, \cdots, \mathbf{r}_{m'}] \\ \mathbf{B} &= [\mathbf{b}_m, \cdots, \mathbf{b}_{m'}] \\ \mathbf{N} &= [\mathbf{n}_m, \cdots, \mathbf{n}_{m'}] \end{aligned} \tag{4.12}$$

Comparing with the model of linear ICA in Equation 2.2 on Page 6

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{4.13}$$

$\mathbf{B}$ is the source signal $\mathbf{s}$ need to be estimated, $\mathbf{R}$ is the observed mixed signal $\mathbf{x}$, and $\mathbf{G}$ is the unknown mixing matrix $\mathbf{A}$. The noise matrix $\mathbf{N}$ in Equation 4.11can be treated as an independent component to be added into $\mathbf{x}$.

4.2 Gold Code Generation

The generation of PN sequences for spread spectrum system simulation is necessary. By far the most widely known binary PN sequences are the maximum-length shift-register sequences, or m-sequences for short. Its periodic autocorrelation functions is

$$\phi(j) = \begin{cases} n & (j = 0) \\ -1 & (1 \leq j \leq n-1) \end{cases} \tag{4.14}$$

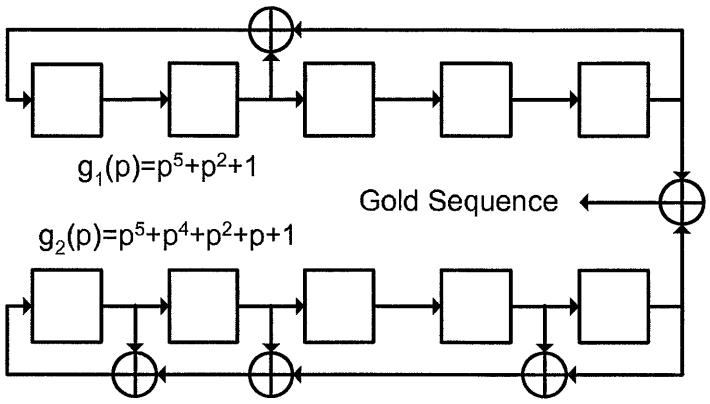

Figure 4.1 Generation of *Gold sequences* of length=31

and almost ideal, where n is the period.

In CDMA system, however, the cross-correlation properties of PN sequences are as important as the autocorrelation properties. The periodic cross-correlation function between any pairs of m-sequences with the same period can have relatively large peaks but still remains small. Thus PN sequences with better periodic cross-correlation properties is desirable in CDMA system.

The well-known Gold and Kasami sequences exhibit a three-valued periodic cross-correlation function with value $\{-1,\ -t(m),\ t(m)-2\}$, where

$$t(m) = \begin{cases} 2^{(m+1)/2}+1 & (\text{odd } m) \\ 2^{(m+2)/2}+1 & (\text{even } m) \end{cases} \tag{4.15}$$

and it has relatively lower peaks for its periodic cross-correlation function than m-sequence has.

From two m-sequences of length n with a periodic cross-correlation function that takes on the possible values $\{-1,\ -t(m),\ t(m)-2\}$, say $\mathbf{a}$ and $\mathbf{b}$, we construct a set of sequences of length n by taking the modulo-2 sum of $\mathbf{a}$ with the n cyclicly shifted versions of $\mathbf{b}$ or vice versa. Including the original sequences $\mathbf{a}$ and $\mathbf{b}$, the total $n+2$ sequences constructed in this manner are called *Gold sequences*.

An generation of *Gold sequences* of length 31 is shown in Figure 4.1 and the

0	0	0	0	0	0	1	1	1	1	0	1	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	1	1	1
1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	0
1	0	0	1	0	0	1	1	0	1	1	1	1	1	0	0	1	0	1	0	0	1	1	0	1	1	0	0	0	0	1
1	0	1	0	1	1	0	1	0	0	0	1	0	1	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	1
1	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0	0	1	0	0	1	0	0	0	1	1
1	1	0	1	1	0	1	1	0	0	1	0	1	0	1	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	0
0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	0	1	1	0	1	0	1	1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0	1	0	0	1	1	0	1	0	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0
0	1	1	0	1	0	1	0	1	1	0	1	1	0	0	0	1	0	0	0	1	1	0	1	0	1	1	1	0	0	1
0	1	0	1	1	1	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	1	0
0	0	1	1	0	1	1	1	0	1	0	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0
1	1	1	0	0	1	0	1	0	1	0	0	0	0	1	0	1	1	0	0	1	1	1	1	1	1	0	0	1	0	0
0	1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	1	0	1
0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	1	0	1	1	1	0
1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	1	0	0	0	1	0	1
1	1	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1	0	1	1
0	1	1	1	0	1	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	1	1	0	0	0	1	1	0	1	0	0	0	0
0	1	0	0	1	0	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0
0	0	0	1	1	1	0	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	0	0	0	1	1	1	1	0	0
1	0	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	0	1	1	0	0
1	1	1	0	1	1	1	1	1	0	1	0	1	1	1	0	1	0	1	1	0	0	1	1	0	0	0	1	1	0	1
0	1	0	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	0	1	0	0	1	0	1	0
1	1	0	0	1	1	1	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	1	0	1	0	1
1	0	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	1	0	0	0	0	1	1	1	1	1	1	0
1	1	0	0	0	1	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1

Figure 4.2 Example of 33 *Gold sequences* of length=31

codes generated by it plus two original m-sequences are shown in Figure 4.2. This sample short *Gold Sequences* will be selected as the PN sequences for numerical experiments illustrated in next chapter.

4.3 Application of ICA to DS-CDMA System

DS-CDMA provides several possibilities for applying ICA in a meaningful way such as a simplified complexity minimization approach for estimating fading channels [19], blind separation of convoluting mixtures using an extension of the natural gradient algorithm, and improvement of the performance of conventional DS-CDMA receivers. In this work, we focus on the last two applications.

In fact, the signal model in downlink environment of DS-CDMA system discussed in this chapter can be easily extended into the uplink environment for example in [21] and many more references on blind communications techniques

in recent conference proceeding and journals.

The FastICA algorithm has been applied to the CDMA signal separtion for the first time in [18], and later in [17], [20], and [21]. The basic feature of any ICA method is that it is able to estimate a set of independent components, but the order of estimated independent components is somewhat unpredictable. If the method is hierarchical, the sources tend to become separated in the order of decreasing non-Gaussianity, but this is not enough to identify each source. The utilization of the a prior information such as training sequences becomes thus important.

In next chapter, we focus a direct symbol estimation in downlink environment of DS-CDMA system by means of ICA method. The results of numerical experiment are compared with conventional match filter or Single User Detection (SUD).

5. SYSTEM SIMULATION OF ICA BASED DETECTOR

In this chapter, encouraged by the good results of numerical experiments for testing ICA algorithms in Chapter 2, a Monte-Carlo simulation is added to calculate the probability of error of SUD detector. The ICA detector and ICA-SUD detector are simulated by the Monte-Carlo simulation.

5.1 Simulation of Conventional DS-CDMA Detector

Having verified and compared two ICA algorithms in the previous chapters, a Monte-Carlo simulation of DSSS demodulation in the presence of AWGN has been implemented, and its simulation model is shown in Figure 5.1. Monte Carlo methods are a widely used class of computational algorithms for simulating the behavior of mathematical systems. They are distinguished from other simulation methods by being stochastic, that is nondeterministic in some manner - usually by using random numbers (or more often pseudo-random numbers) - as opposed to deterministic algorithms.

The short Gold Codes shown in Figure 4.2 are used in this numerical experiment. The parameters were as follows: The length of chip was $C = 31$; the number of symbols was $M = 10000$; the number of user was $K = 1$; Signal-to-Noise Ratio was varied from -20dB to 10dB. The simulation results are shown in Figure 5.2 and Table 5.1.

It is obvious that, as shown in Section 3.5, the probability of error for a DSSS signal before despreading with BPSK modulation is identical to the probability of error for conventional BPSK modulation in the presence of AWGN. And the probability of symbol error is much lower after despreading, comparing with Bit Error Rate(BER) of undespreading signal.

In this simulation, the processing gain of DSSS system is the chip length

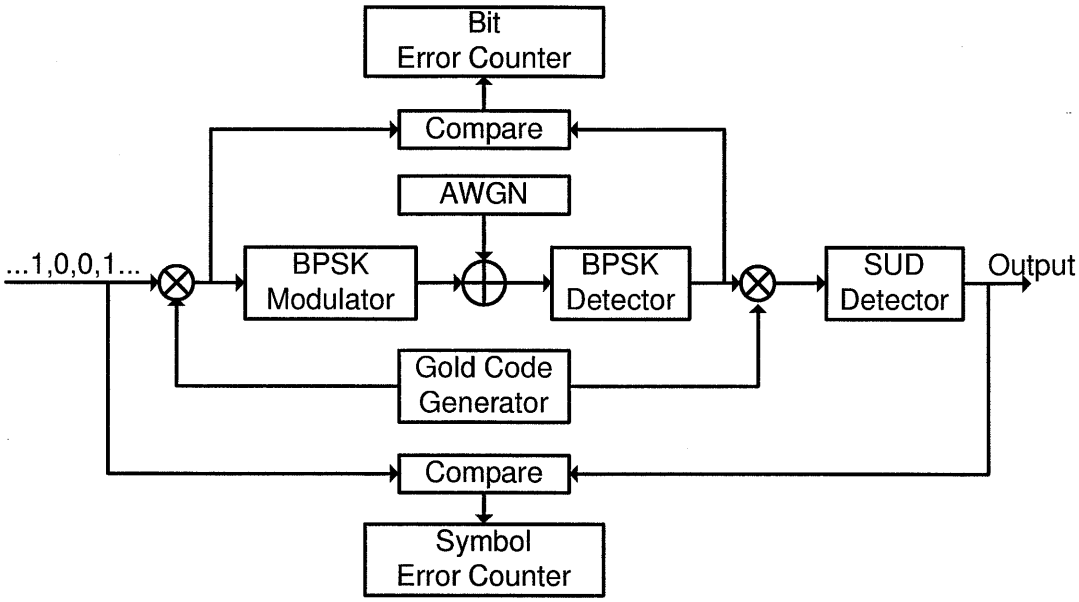

Figure 5.1 Model of DSSS for Monte Carlo Simulation

Table 5.1 Results of Conventional Detector with Monte Carlo Simulation

SNR(dB)	BER_{cal}	BER_{exp}	SNR(dB)	SER_{exp}
-5	0.21323	0.21469	- 20	0.2213
-4	0.18611	0.18744	- 19	0.1964
-3	0.15837	0.15926	- 18	0.1661
-2	0.13064	0.13138	- 17	0.1346
-1	0.10376	0.10437	- 16	0.1047
0	0.07865	0.07890	- 15	0.0790
1	0.05628	0.05635	- 14	0.0567
2	0.03751	0.03763	- 13	0.0367
3	0.02288	0.02265	- 12	0.0236
4	0.01250	0.01223	- 11	0.0139
5	0.00595	0.00595	- 10	0.0075

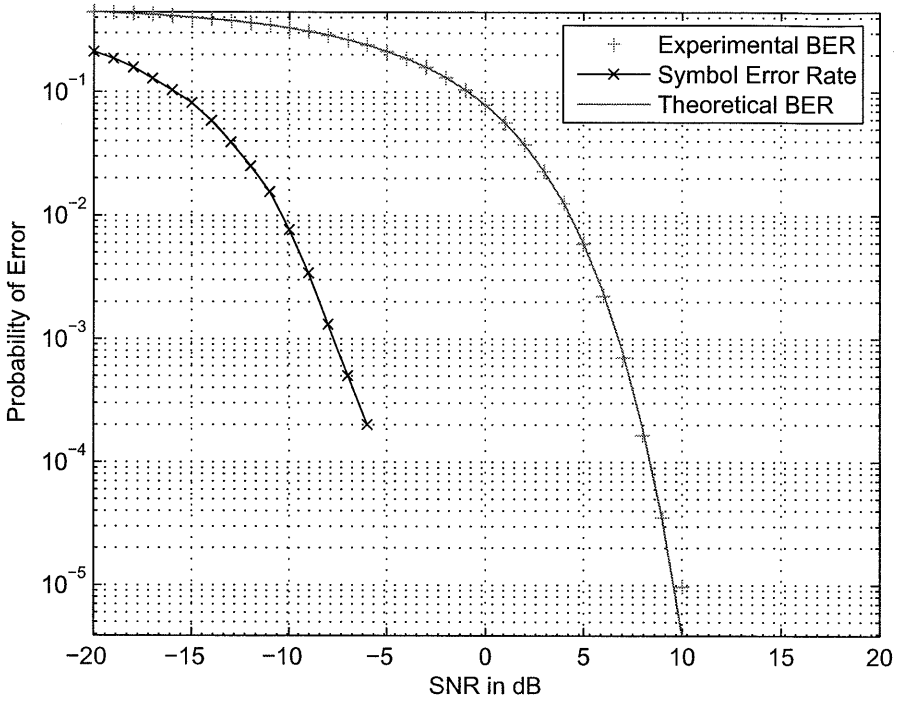

Figure 5.2 Results of DSSS performance with Monte Carlo Simulation

$C = 31$ or $10 \log C \approx 14.9136$dB. In Table 5.1, Symbol Error Rate (SER) has approximately equivalent probability of error than BER of signal before de-spreading when its SNR is about 15dB low. More precisely, when SER=0.2213 at SNR=-20dB, the corresponding $\text{BER}_{\text{cal}} = 0.22129$ at SNR=-5.305dB. The processing gain is -5.305-(-20)=14.695dB, which is approximately equal to $10 \log C \approx 14.9136$dB.

With this numerical experiment, the validation of system model of SUD is accomplished so that the SUD can be compared to ICA based detector.

5.2 Simulation of ICA-Based Detector

In this section, the ICA-Based DS-CDMA downlink detector (ICA detector) has been simulated. At first, a basic ICA detector is implemented. Next, the vector size required by ICA algorithm is studied. Some other factors such

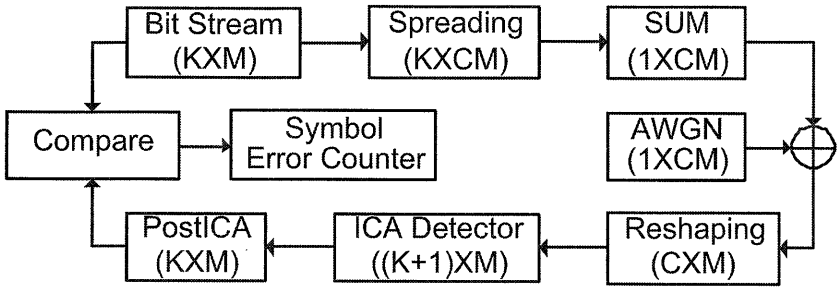

Figure 5.3 Model of ICA Detector with Monte Carlo Simulation

as the initial value of iteration, criteria of convergence are considered as well. Finally, the ICA-SUD joint detector is simulated and it gives lower probability of error than ICA detector or conventional SUD receiver.

5.2.1 The First ICA-Based Detector

The system model of DS-CDMA (see Equation 4.3 on Page 39) and the system model of ICA detector (see Equation 4.11 on Page 40) are discussed in Section 4.1. They are reproduced below

$$r[n] = \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{m=0}^{M} (a_{l,m-1} b_{k,m-1} \underline{\mathbf{s}}_{kl}[n] + a_{lm} b_{km} \overline{\mathbf{s}}_{kl}[n]) + n[n] \tag{5.1}$$

$$\mathbf{R} = \mathbf{GB} + \mathbf{N} \tag{5.2}$$

where **R**,**N** is $C \times M'$ matrix, **G** is $C \times 2KL$ matrix and **B** is $2KL \times M'$ matrix.

In this work, the code timing and channel estimation are assumed to be the prerequisite tasks. In other words, the received signal for ICA detector is assumed the sampled and synchronized data. Thus in Equation 5.1, the number of path l is down to 1; the propagation delay d_l is equal to 0; as well as the "late" and "early" parts, $\underline{\mathbf{s}}_{kl}$ and $\overline{\mathbf{s}}_{kl}$ in Equation 4.4 on Page 39 are merged to

$$\mathbf{s}_{kl} = [s_k(1) \ \cdots \ s_k(C)]^T \tag{5.3}$$

In view of ICA algorithm, the AWGN can be treated as one of the independent

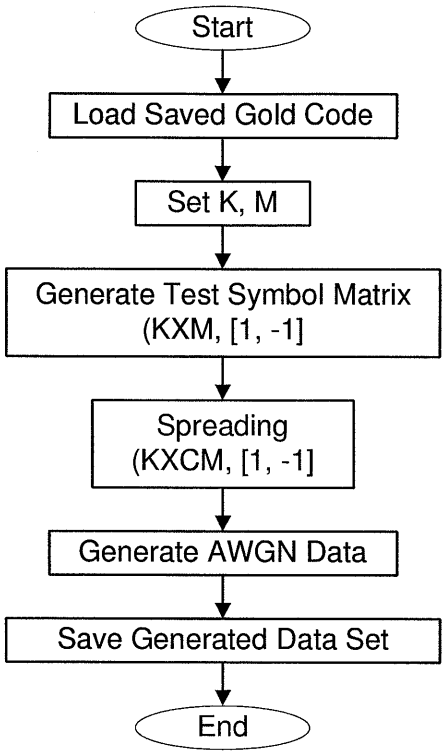

Figure 5.4 Flowchart of Simulation Data Set Generation with Monte Carlo Simulation

components. Then the Equation 5.2 is changed to

$$\mathbf{R} = \mathbf{GB} \tag{5.4}$$

where the dimension of matrices R, B and G are $C \times M'$, $C \times K$ and $K \times M'$, respectively.

The simulated DS-CDMA downlink data in the presence of AWGN is used to test the algorithm. The short Gold codes shown in Figure 4.1 on Page 41 are used. Thus the maximum number of users is $K = 30$ as the 31^{st} "user" is AWGN. The number of symbol is 1000. The process of data set generation is shown in Figure 5.4.

Matlab simulation is implemented for verification of the validity of the system model. Parameters were as follows: Number of symbol was $M = 1000$; Number of users was $K = 30$; Number of paths was $L = 1$; Signal-to-Noise

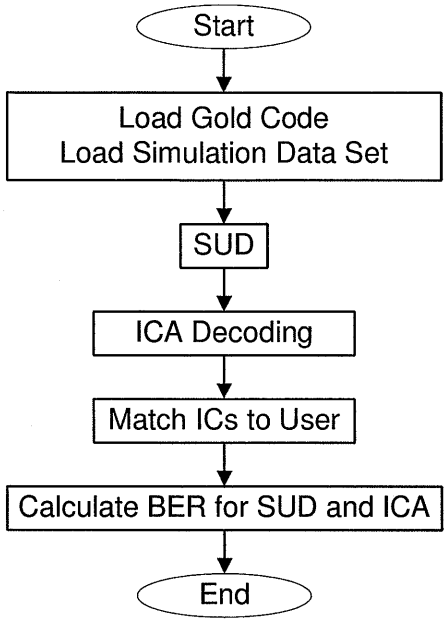

Figure 5.5 Flowchart of ICA Detector with Monte Carlo Simulation

Ratio (SNR) was varied with respect to the individual user from -10dB to 0dB. All the signals for each of users are sent in same power. The AWGN is treated as an IC. The simulation model is shown in Figure 5.3 ; the flowchart of simulation is shown in Figure 5.5and two of the typical simulation results are shown in Figure 5.6 and Figure 5.7.

We can easily observe that the ICA detector functionally works. It can semi-blindly estimate DS-CDMA downlink signal when the decoding is blind but it requires some user information such as the user's assigned chip code or training sequences to match the decoded bit stream to user who should receive it.

However, in this simulation, more than 10% of simulation trials can't converge. In other words, the ICA decoder is not working at all in these trials since no IC can be found during the iteration. In Figure 5.6, we also notice that, at $SNR = -8dB$ and $-4dB$, the symbol error rate of the ICA decoder are unreasonably high because some ICs can not be found, as well as in Figure

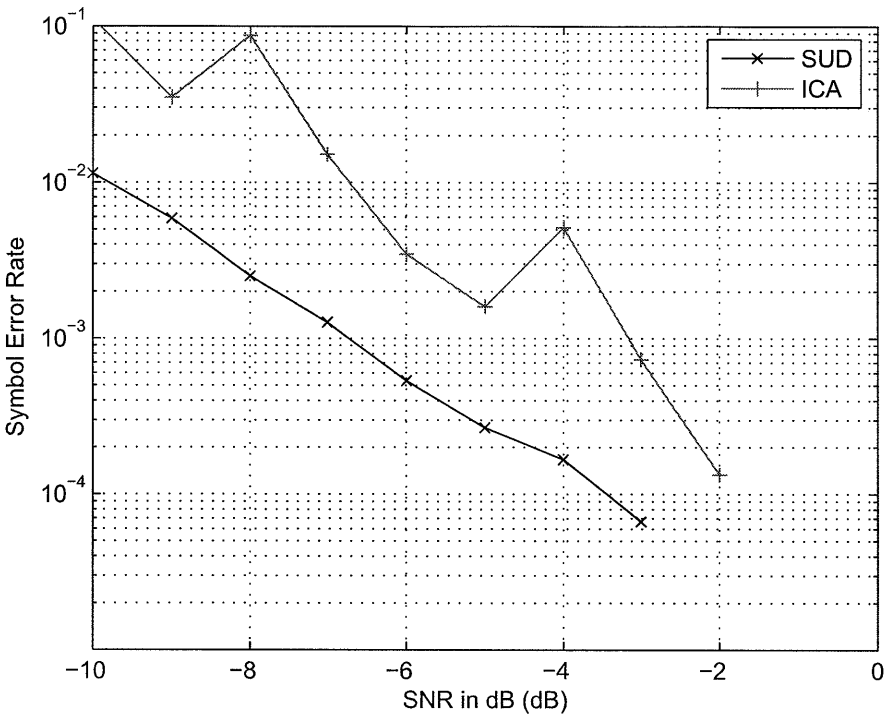

Figure 5.6 Results of ICA Detector with Monte Carlo Simulation

5.7 at $-5dB$. Therefore, the basic ICA detector need to be improved, though it functionally works in most cases.

5.2.2 Some Factors in ICA Algorithm

In the last section, the ICA detector has been implemented. However the consistency of the simulation results is not satisfied. In this section, some factors such as number of symbol M (also known as the vector size required by ICA algorithm) and the initial value of iteration in ICA algorithm are discussed.

Number of Symbol

The number of symbol involved in a single trial of ICA detection is usually $M = 1000$. It is used in many literatures [17] [18] [20] and the simulation in the last section. We test the ICA detector with $M = 500, 1000, 2000, 5000$ *and* 10000

Table 5.2 Results of ICA Detector with Monte Carlo Simulation, $M = 1000$, 2000, 5000 and 10000

SNR	$M = 1000$			$M = 2000$			$M = 5000$			$M = 10000$		
(dB)	#1	#2	#3	#1	#2	#3	#1	#2	#3	#1	#2	#3
-10	0.1779	0.1107	0.0654	0.0293	0.0392	0.0327	0.0282	0.0292	0.0282	0.0265	0.0262	0.0265
-9	0.0234	0.0349	0.0237	0.0167	0.0174	0.0174	0.0153	0.0153	0.0141	0.0135	0.0136	0.0134
-8	0.0113	0.0871	0.0113	0.0085	0.0090	0.0080	0.0064	0.0072	0.0068	0.0059	0.0060	0.0059
-7	0.0572	0.0151	0.0051	0.0038	0.0032	0.0037	0.0027	0.0028	0.0028	0.0021	0.0022	0.0024
-6	0.0023	0.0035	0.0019	0.0014	0.0013	0.0015	0.0006	0.0008	0.0009	0.0007	0.0006	0.0007
-5	0.0009	0.0016	0.0079	0.0004	0.0004	0.0003	0.0002	0.0002	0.0003	0.0001	0.0001	0.0002
-4	0.0006	0.0051	0.0004	0.0001	0.0001	0.0000	0.0001	0.0001	0.0001	0.0000	0.0000	0.0001
-3	0.0001	0.0007	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-2	0.0001	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 5.3 Results of ICA Detector with Monte Carlo Simulation with Zero, Random and Gold Code Initial Value

SNR	Zero			Random			Gold		
(dB)	#1	#2	#3	#1	#2	#3	#1	#2	#3
-10	0.0269	0.0264	0.0268	0.0267	0.0261	0.0267	0.0262	0.0265	0.0265
-9	0.0133	0.0132	0.0137	0.0137	0.0135	0.0134	0.0136	0.0134	0.0135
-8	0.0061	0.0064	0.0064	0.0059	0.0061	0.0062	0.0060	0.0059	0.0059
-7	0.0021	0.0023	0.0024	0.0023	0.0023	0.0023	0.0022	0.0024	0.0021
-6	0.0006	0.0007	0.0007	0.0007	0.0007	0.0006	0.0006	0.0007	0.0007
-5	0.0002	0.0002	0.0001	0.0001	0.0002	0.0001	0.0001	0.0002	0.0001
-4	0.0000	0.0001	0.0000	0.0000	0.0001	0.0000	0.0000	0.0001	0.0000
-3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
-1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

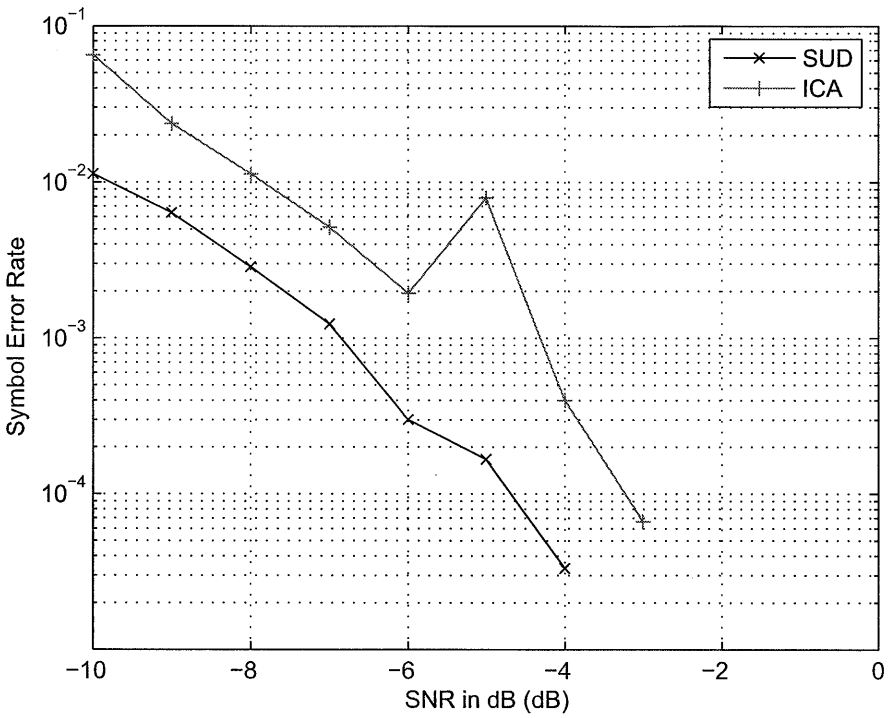

Figure 5.7 Results of ICA Detector with Monte Carlo Simulation

and typical simulation results from a trial (randomly picked) are shown in Figure 5.8 and 3 randomly picked simulation results are shown in Table 5.2 as well.

At first, the ICA detector can't find out any ICs at all as $M = 500$ because the algorithm can't converge and its result is not shown. Next, the computing load is heavier as M increases. At last, 100 simulations are performed for each M. In Figure 5.8 we can observe that the consistency of the simulation results is getting better and the symbol error rate is getting lower as M increases. Especially in Table 5.2, we notice that the consistency level of result is acceptable as $M = 10000$. Therefore, $M = 10000$ will be used for all the next simulations. Also, we find that ICA detector can even provide lower symbol error rate than conventional SUD detector as $SNR > -5dB$ and $M = 5000, 10000$.

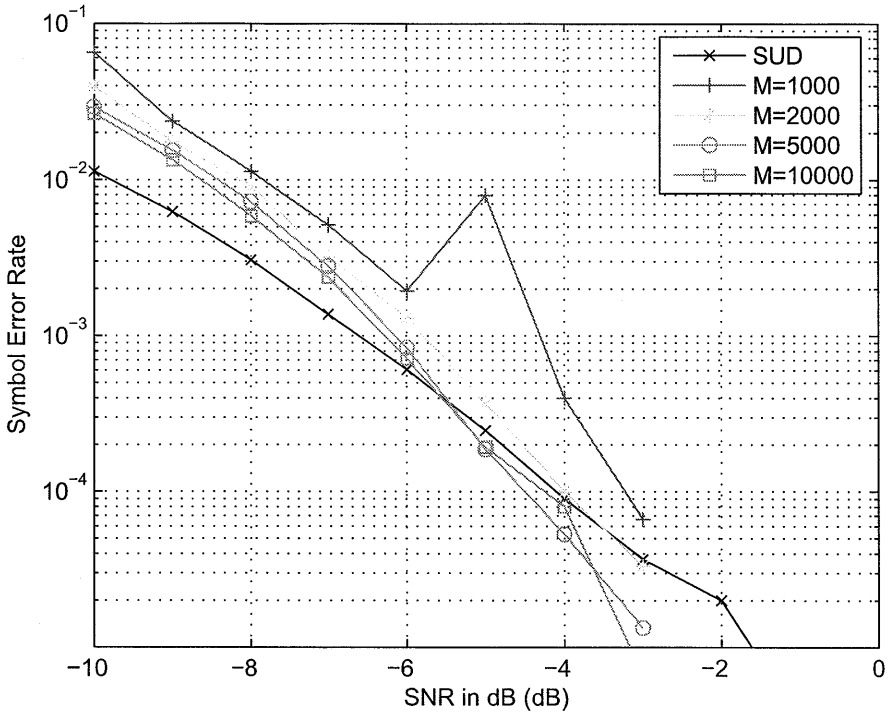

Figure 5.8 Results of ICA Detector with Monte Carlo Simulation, $M = 1000$, 2000, 5000 and 10000

Initial Value

Initial value is another important factor that affects the convergence of iteration algorithm. Table 5.3 shows the simulation results of ICA detector when the initial value is set to all-zero, random and Gold code.

From the simulation results, no significant difference among the initial value settings is observed. However, we can notice that the number of iteration is a little less when Gold code is used.

At last, we also attempt to change the criteria of convergence ϵ, but no difference is observed.

5.3 Simulation of ICA-SUD Detector

In the preceding sections, ICA detector has been implemented and discussed. The blind multi-user detection can be performed by ICA based detec-

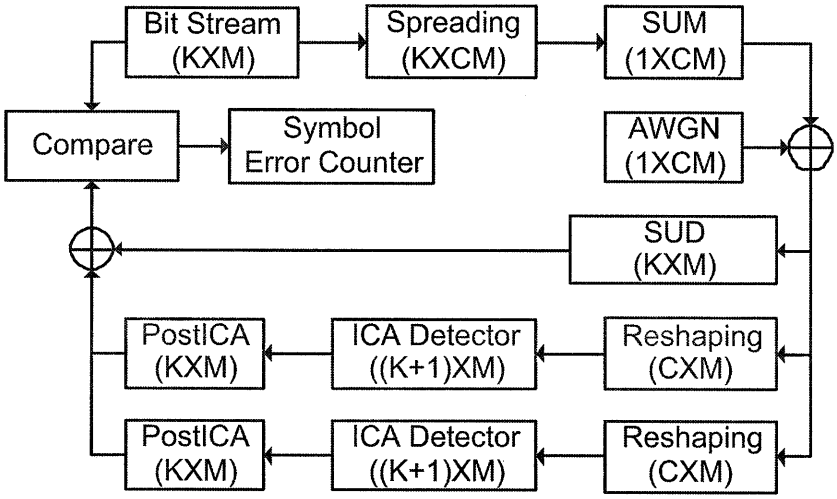

Figure 5.9 Model of ICA-SUD Detector for Monte Carlo Simulation

Table 5.4 Results of ICA-SUD Detector with Monte Carlo Simulation

SNR (dB)	ICA	SUD	ICA-SUD
-10	0.027047	0.011077	0.010930
-9	0.014180	0.005890	0.005213
-8	0.006273	0.002773	0.002227
-7	0.002207	0.001140	0.000753
-6	0.000727	0.000473	0.000203
-5	0.000187	0.000183	0.000070
-4	0.000027	0.000073	0.000010
-3	0.000000	0.000030	0.000000
-2	0.000000	0.000007	0.000000
-1	0.000000	0.000003	0.000000
0	0.000000	0.000000	0.000000

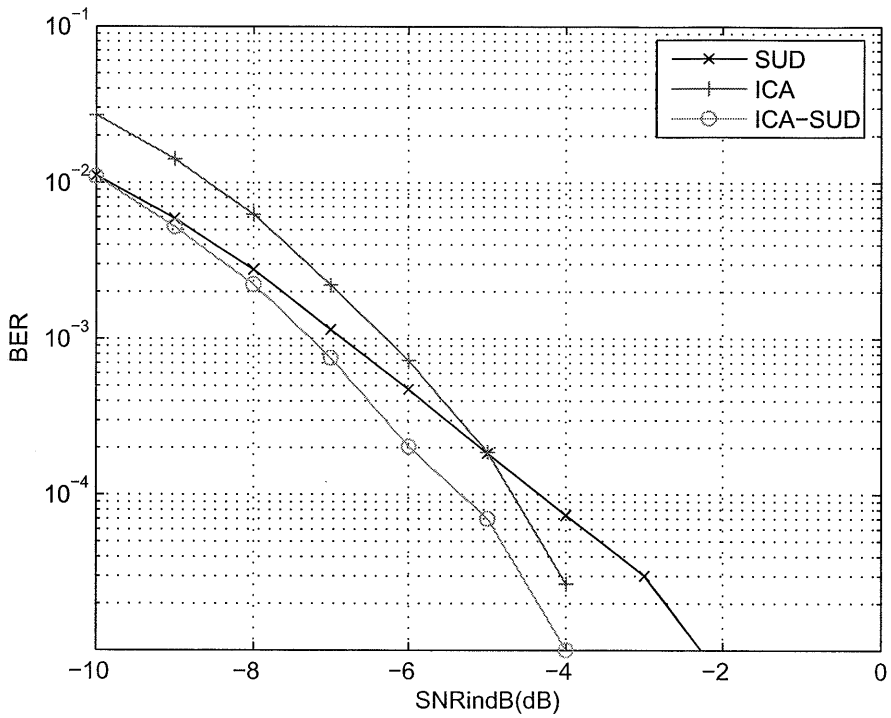

Figure 5.10 Result of ICA-SUD Detector for Monte Carlo Simulation

tor. The term *blind* implies that the detection can be performed by received signal only. Although the parameter of system is not necessary, use of some pilot or training sequence is desirable to identify the signal for desired user.

5.3.1 Ambiguities of ICA Detector

Independent Components (ICs) can be normally found by ICA detector, However, the ICA detector sometimes encounters the following ambiguities or indeterminacies.

Variances (energies) of the ICs can't be determined

As a consequence, we have to fix this magnitudes of the ICs. This is not a problem in our simulation. Since all of transmitted data stream is binomial distribution random variables with equal possibilities, the most natural way to do so is to assign $+1$ to all the positive magnitude and -1 to all the negative

magnitude. Note that this still leaves the ambiguity of the sign. Therefore, the pre-requisite info such as training sequences is required to identify the sign of the decoded data stream.

Order of ICs can't be determined

Because of two indeterminacies above, ICA detector can't be used independently as a DS-CDMA detector. It requires training sequence to help identify the owners of the decoded data stream. Therefore, the ICA detection may perform better as an additional part of conventional detector.

5.3.2 ICA-SUD Detector

The ICA-SUD detector is implemented. The simulation model is shown in Figure 5.9 and one randomly picked results of numerical experiment are shown in Figure 5.10 on and Table 5.4. Parameters were as follows: Number of symbol was $M = 10000$; Number of users was $K = 30$; Number of paths was $L = 1$; Signal-to-Noise Ratio (SNR) was varied with respect to the individual user from -10dB to 0dB. All the signals for each of users are sent in same power.

In the illustrative experiment, all the algorithm worked quite well. Two independent ICA detectors are used as the post-processing of SUD detector. In Figure 5.10 and Table 5.4 show the performance of these methods as the average bit-error-rate (BER) corresponding to all users. It is obvious that ICA-SUD gives the lowest BER.

6. CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

CDMA techniques are currently studied extensively in telecommunications, because they will be used in a form or another in future high performance mobile communications systems. A specific feature of ICA, semi-blind nature of source separation, provides useful means for CDMA demodulation. Where the receiver has more or less prior information on the communication system, typically at least the spreading code of the desired user is known, the ICA can be released from the user identification and concentrate on decoding. This prior information should be combined in a suitable way with blind ICA techniques for achieving optimal results. Another important design feature is that practical algorithms should not be computationally too demanding, making it possible to realize them in real time, though it still requires more effort in future work.

First of all, we began with the basic concepts of ICA. FastICA [4] was taken into deeper consideration. FastICA is a good multi-purpose implementation of ICA. This algorithm is based on the fixed-point method, so it has some advantages such as fast convergence, suitable to both super-Gaussian and sub-Gaussian components.

Next, basic concepts of multiple access, DSSS and DS-CDMA were discussed. The DS-CDMA signal model was introduced, as well as its discrete-time vector representation, followed by system model of ICA based DS-CDMA downlink detector.

Finally, ICA DS-CDMA downlink detector has been simulated and it demonstrated that ICA detector can solve the symbol estimation problem with no

spreading code required, though the spreading code should be utilized to identify each user. Thus an ICA-SUD detector has been proposed and its symbol error rate is lower than the conventional SUD detector concluded from numerical experiments. Even if the powers of the signals are the same, additional multiple access interference can be mitigated by ICA, thus improving the performance of SUD.

When the number of symbol involved into a single simulation run is getting bigger, e.g. from 1000 to 10000, we can observe the number of iteration is significantly reduced, e.g. form 20 down to 5 at average level. The bigger the number, the lower the symbol error rate. Of course, the more computation power is required.

6.2 Recommendations

Comparing to the conventional RAKE detector and subspace MMSE detector, SUD detector is simpler method. It will be considered to take ICA as an additional element of RAKE and MMSE detector in the presence of multi-path fading channel in future.

References

[1] A. Hyvärinen, "One-unit contrast functions for independent component analysis: A statistical analysis," in *Neural Networks for Signal Processing VII (Proc. IEEE Workshop on Neural Networks for Signal Processing)*, pp. 388-397, amelia Island, Flodia, USA, 1997.

[2] A. Hyvärinen, "Survey on independent component analysis," [Online], Available: `http://www.cis.hut.fi/ãapo/`

[3] A. Hyvärinen and Erkki Oja, "Independent Component Analysis: A Tutorial," [Online], Available: `http://www.cis.hut.fi/ãapo/`

[4] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation,* 9(7):1483-1492, 1997.

[5] A. J. Bell and T. J. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural Computation*, vol. 7, pp. 1129-1159, 1995.

[6] Athanasios Papoulis, *Probability, Random Variables and Stochastic Processes*, 3rd ed, McGraw-Hill, 1991.

[7] C. Jutten and J. Hérault, "Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture", *Signal Processing,* vol. 24, pp. 1-10, 1991.

[8] C. Jutten and J. Hérault, "Blind separation of sources, part II: Problems statement", *Signal Processing,* vol. 24, pp. 11-20, 1991.

[9] C. Jutten and J. Hérault, "Blind separation of sources, part III: Stability analysis", *Signal Processing,* vol. 24, pp. 21-29, 1991.

[10] J. F. Cardoso, "Source separation using higher order moments," in *Proc. ICASSP'89*, pp. 2109-2112, 1989.

[11] J. Hérault and C. Jutten, "Space or time adaptive signal processing by neural network models", in *Neural networks for computing: AIP conference proceedings 151*, J.S. Denker, Ed. New York: American Institute for Physics, 1986, pp. 206-211.

[12] J. G. Proakis, *Digital Communications*, 4th ed. McGraw-Hill, 2001.

[13] J. G. Proakis, Masoud Salehi, Gerhard Bauch, *Contemporary Communication Systems Using MATLAB And Simulink*, 2nd ed. THOMSON-Brooks/Cole, 2004.

[14] P. Comon, "Independent component analysis – a new concept?" *Signal Processing, Elsvier,* vol. 36, pp. 287-314, Apr. 1994. Special issue on Higher-Order Statistics.

[15] S. Amari, A. Cichocki, and H. H. Yang, "A new learning algorithm for blind source separation," in *Advances in Neural Information Processing 8*, Cambridge, MA:MIT Press, 1996, pp. 757-763.

[16] S. Verdú, *Multiuser Detection*, Cambridge University Press, 1998.

[17] T. Ristaniemi and R. Wu, "Mitigation of continuous-wave jamming in DS-CDMA systems using source separation techniques," *Proc. Third International Workshop on Independent component Analysis and Signal Separation,* (ICA2001), San Diego, CA, USA, December 2001, pp. 55-58.

[18] J. Joutsensalo and T. Ristaniemi, "Learning Algorithms for Blind Multiuser Detection in CDMA Downlink," *Proc. PIMRC'98,* Boston, USA, September 1998, pp. 1040-1044.

[19] R. Cristescu, T. Ristaniemi, J. Joutsensalo, and J. Karhunen, "Delay Estimation in CDMA Communications Using A FastICA Algorithm," *Proc. the International Workshop on Independent Component Analysis and Blind Signal Separation,* (ICA 2000), Helsinki, Finland, June 2000, pp. 105-110.

[20] T. Ristaniemi and J. Joutsensalo, "On the performance of Blind Source Separation in CDMA Downlink," *Proc. International Wrokshop on Independent Component Analysis and Signal Separation*, (ICA'99), Aussois, France, January 1999, pp.437-442.

[21] T. Ristaniemi and R Wu, "On the Performance of Multisensor Reception in CDMA by Fast Independent Component Analysis," *The 53th IEEE Vehicular Technology Conference*, Rhodes, Greece, May 6-9, 2001, vol. 3, pp. 1829-1833.

APPENDIX MATLAB SOURCE CODES

File Descriptions

```
#### Signal generation ####
  gold5gen.m  - Generates Gold code
  siggen1.m   - Generates mixed signals I for ICA simulation
                (Speeches)
  siggen2.m   - Generates mixed signals II for ICA simulation
  siggen3.m   - Generates mixed signal sets for CDMA simulation

#### InfoMAX simulation files ####
  infomax.m   - Main program
  sep.m       - The code for one learning pass thru the data
  wchange.m   - Tracks size and direction of weight changes

#### FastICA simulation files ####
  fastica.m   - Main function program
  fpica.m     - Main algorithm for calculating ICA
  remmean.m   - Function for removing mean
  pcamat.m    - Calculates the PCA for data
  whitenv.m   - Function for whitening data

#### CDMA simulation files ####
  SUDICA.m    - Main program
                SUD, ICA, SUD-ICA Detector
  yfastica.m - Modified fastica main program
                with Gold Code initial value
```

gold5gen.m

```
% Gold code generator
%  PN5_25: 5-bits MLSR sequence (2 5) with initial value [1 0 0 0 0]
%  PN5_1245: 5-bits MLSR sequence (1 2 4 5) with initial value [1 0 0 0 0]
%  gold: generated Gold code, total 31 chips.

clear all;clc;
PN5_25  =[1 -1 -1 -1 -1 1 -1 1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 1 1 -1 1 -1 -1];
PN5_1245=[1 -1 -1 -1 -1 1 1 -1 1 -1 1 -1 -1 1 -1 -1 -1 1 -1 1 1 1 1 1 -1 1 1 -1 -1 1 1];
gold=zeros(31,31);
for i=1:31
    gold(i,:)=PN5_1245.*PN5_25.*(-1);
    PN5_25=[PN5_25(2:31) PN5_25(1)];
end

save gold5.mat gold;
```

siggen1.m

```
% Generate X contains n mixtures of n source speeches
```

```
% n - # of sources, n=2,...,9
% A - mixing matrix
% X - mixed signal vector

% Read .wav files
[s1 fs]=wavread('source1');[s2 fs]=wavread('source2');
[s3 fs]=wavread('source3');[s4 fs]=wavread('source4');
[s5 fs]=wavread('source5');[s6 fs]=wavread('source6');
[s7 fs]=wavread('source7');[s8 fs]=wavread('source8');
[s9 fs]=wavread('source9');
S=[s1.';s2.';s3.';s4.';s5.';s6.';s7.';s8.';s9.'];
[n p]=size(S);              % p=50000, n=4, for example
A=rand(n); %a=[1 2;1 1];    % mixing matrix
X=A*S;                      % mix input signals
```

siggen2.m

```
N=500; v=[0:N-1];S=[];
S(1,:)=sin(v/2);                          % sinusoid
S(2,:)=((rem(v,23)-11)/9).^5;             % funny curve
S(3,:)=((rem(v,27)-13)/9);                % saw-tooth
S(4,:)=randn(1,N);                        % Gaussian signal
for t=1:4, S(t,:)=S(t,:)/std(S(t,:)); end
A=rand(size(S,1));[n p]=size(S);X=A*S;  % mix signals
```

siggen3.m

```
% Mixed Signal generator - Downlink signal generator
%  K: # of User, must <=30 (the 31th is AWGN noise)
%  SNR - Signal (power of 1 user) to AWGN Noise Ratio
%  M: # of symbols, M=10000
%  S: (K,310000) Transmitting signal
%  R: (1,310000) Received signal with
%  mixedsignal: (31,10000) decomposition from R
clear all; clc; load gold5.mat; K=30; M=10000;

% Bitstream: Source bit stream for K users & M symbols per user
Bitstream=randsrc(K,M,[1,-1]);

for i=1:K
    tmp=repmat(Bitstream(i,:),31,1);
    for j=1:M
        tmp(:,j)=tmp(:,j).*gold(i,:)';
    end
    S(i,:)=reshape(tmp,1,31*M);
end

AWGN=randn(1,31*M);

save mixedsig.mat Bitstream S AWGN K M;
```

infomax.m

```
% This file reads the mixing signals from vector X,
% and return the unmixed signal vector Shat.
% W*Wz - unmixed matrix

% Whiten data
X0=X;C=cov(X');permute=randperm(p);X=X(:,permute);
X=X-mean(X').'*ones(1,p);     % remove mean => mean=0;
Wz=2*inv(sqrtm(C));           % get decorrelating matrix
X=Wz*X;             % decorrelate mixes => cov(X.')=4*eye(n)

W=eye(n);           % init. unmixing matrix, or w=rand(M,N);
```

```
m=size(W,2);        % m=n  usually
sweep=0; Wold=W; olddelta=ones(1,n*n);Id=eye(m);

% should converge on 1 pass for 2->2 net but annealing
% will improve soln even more and so on
L=0.01; B=30; sep; L=0.001; B=30; sep
Shat=W*Wz*X0;         % make unmixed sources
```

sep.m

```
% SEP goes once through the mixed signals, X in batch blocks of size B,
% adjusting weights, W, at the end of each block.

sweep=sweep+1; t=1;noblocks=fix(p/B);BI=B*Id;
for t=t:B:t-1+noblocks*B,
  u=W*X(:,t:t+B-1);
  W=W+L*(BI+(1-2*(1./(1+exp(-u))))*u')*W;
end;
[change,olddelta,angle]=wchange(Wold,W,olddelta);Wold=W;
fprintf('****sweep=%d, change=%.4f angle=%.1f deg., [n%d,m%d,p%d,B%d,L%.5f] \n',...
   sweep,change,180*angle/pi,n,m,p,B,L);
W*Wz*A     %should be a permutation matrix for artif. mixed data.
```

wchange.m

```
function [change,delta,angle]=wchange(Wold,W,olddelta)
  [m,n]=size(W); delta=reshape(Wold-W,1,m*n);
  change=delta*delta';
  angle=acos((delta*olddelta')/sqrt((delta*delta')*(olddelta*olddelta')));
```

fastica.m

```
function [Out1, Out2, Out3] = fastica(mixedsig, varargin)
% [icasig, A, W] = FASTICA (mixedsig);
%   icasig   - estimated independent components
%   A        - estimated mixing matrix
%   W        - estimated unmixing matrix
%   mixedsig - mixed signal

% Remove the mean and check the data
[mixedsig, mixedmean] = remmean(mixedsig);

%[# of rows, # of columns]
[Dim, NumOfSampl] = size(mixedsig);

% Default values for 'pcamat' parameters
firstEig          = 1;
lastEig           = Dim;
interactivePCA    = 'off';
% Default values for 'fpica' parameters
approach          = 'defl';
numOfIC           = Dim;
g                 = 'pow3';
finetune          = 'off';
a1                = 1;
a2                = 1;
myy               = 1;
stabilization     = 'off';
epsilon           = 0.0001;
maxNumIterations  = 1000;
maxFinetune       = 5;
initState         = 'rand';
guess             = 0;
sampleSize        = 1;
displayMode       = 'signals';
displayInterval   = 1;
```

```
% Parameters for fastICA - i.e. this file
only = 3; userNumOfIC = 0;

% print information about data
fprintf('Number of signals: %d\n', Dim);
fprintf('Number of samples: %d\n', NumOfSampl);

% Check if the data has been entered the wrong way,
% but warn only... it may be on purpose
if Dim > NumOfSampl
   fprintf('Warning: ');
   fprintf('The signal matrix may be oriented in the wrong way.\n');
   fprintf('In that case transpose the matrix.\n\n');
end

% Calculating PCA
[E,D]=pcamat(mixedsig,firstEig,lastEig,interactivePCA);

% Whitening the data
[whitesig,whiteningMatrix,dewhiteningMatrix]=whitenv(mixedsig,E,D);

% Calculating the ICA
% Check some parameters
% The dimension of the data may have been reduced during PCA calculations.
% The original dimension is calculated from the data by default, and the
% number of IC is by default set to equal that dimension.

Dim = size(whitesig, 1);
% The number of IC's must be less or equal to the dimension of data
if numOfIC > Dim
   numOfIC = Dim;
   % Show warning only if verbose = 'on' and user suplied a value for 'numOfIC'
   fprintf('Warning: estimating only %d independent components\n', numOfIC);
   fprintf('(Can''t estimate more independent components than dimension of data)\n');
end

% Calculate the ICA with fixed point algorithm.
[A, W] = fpica (whitesig,  whiteningMatrix, dewhiteningMatrix, ...
     approach, numOfIC, g, finetune, a1, a2, myy, stabilization, ...
     epsilon, maxNumIterations, maxFinetune, initState, guess, ...
     sampleSize, displayMode, displayInterval);

% Check for valid return
if ~isempty(W)
  % Add the mean back in.
  fprintf('Adding the mean back to the data.\n');
  icasig = W * mixedsig + (W * mixedmean) * ones(1, NumOfSampl);
  fprintf('Note that the plots don''t have the mean added.\n');
else, icasig = []; end
Out1 = icasig;Out2 = A;Out3 = W;
```

fpica.m

```
function [A, W] = fpica(X, whiteningMatrix, dewhiteningMatrix, approach, ...
    numOfIC, g, finetune, a1, a2, myy, stabilization, ...
    epsilon, maxNumIterations, maxFinetune, initState, ...
    guess, sampleSize, displayMode, displayInterval);
%
% Perform independent component analysis using Hyvarinen's fixed point
% algorithm. Outputs an estimate of the mixing matrix A and its inverse W.
%
% whitesig                              :the whitened data as row vectors
% whiteningMatrix                       :can be obtained with function whitenv
% dewhiteningMatrix                     :can be obtained with function whitenv
% approach      [ 'symm' | 'defl' ]     :the approach used (deflation or symmetric)
% numOfIC       [ 0 - Dim of whitesig ] :number of independent components estimated
% g             [ 'pow3' | 'tanh' |     :the nonlinearity used
```

```
%                'gaus' | 'skew' ]
% finetune      [same as g + 'off']     :the nonlinearity used in finetuning.
% a1                                    :parameter for tuning 'tanh'
% a2                                    :parameter for tuning 'gaus'
% mu                                    :step size in stabilized algorithm
% stabilization [ 'on' | 'off' ]        :if mu < 1 then automatically on
% epsilon                               :stopping criterion
% maxNumIterations                      :maximum number of iterations
% maxFinetune                           :maximum number of iteretions for finetuning
% initState     [ 'rand' | 'guess' ]    :initial guess or random initial state. See below
% guess                                 :initial guess for A. Ignored if initState = 'rand'
% sampleSize    [ 0 - 1 ]               :percentage of the samples used in one iteration
% displayMode   [ 'signals' | 'basis' | :plot running estimate
%                 'filters' | 'off' ]
% displayInterval                       :number of iterations we take between plots

if nargin < 3, error('Not enough arguments!'); end
[vectorSize, numSamples] = size(X);

% Checking the data
if ~isreal(X), error('Input has an imaginary part.'); end

% Checking the value for approach
switch lower(approach)
case 'symm', approachMode = 1;
case 'defl', approachMode = 2;
otherwise
    error(sprintf('Illegal value [ %s ] for parameter: ''approach''\n', approach));
end
fprintf('Used approach [ %s ].\n', approach);

% Checking the value for numOfIC
if vectorSize < numOfIC, error('Must have numOfIC <= Dimension!');end

% Checking the sampleSize
if sampleSize > 1
    sampleSize = 1;fprintf('Warning: Setting ''sampleSize'' to 1.\n');
elseif sampleSize < 1
    if (sampleSize * numSamples) < 1000
        sampleSize = min(1000/numSamples, 1);
        fprintf('Warning: Setting ''sampleSize'' to %0.3f (%d samples).\n', ...
            sampleSize, floor(sampleSize * numSamples));
    end
end
if  (sampleSize < 1)
    fprintf('Using about %0.0f%% of the samples in random order in every step.\n',sampleSize*100);
end

% Checking the value for nonlinearity.
switch lower(g)
case 'pow3', gOrig = 10;
case 'tanh', gOrig = 20;
case {'gaus', 'gauss'}, gOrig = 30;
case 'skew', gOrig = 40;
otherwise
    error(sprintf('Illegal value [ %s ] for parameter: ''g''\n', g));
end
if sampleSize ~= 1, gOrig = gOrig + 2;end
if myy ~= 1, gOrig = gOrig + 1;end
fprintf('Used nonlinearity [ %s ].\n', g);

finetuningEnabled = 1;
switch lower(finetune)
case 'pow3', gFine = 10 + 1;
case 'tanh', gFine = 20 + 1;
case {'gaus', 'gauss'}, gFine = 30 + 1;
case 'skew', gFine = 40 + 1;
case 'off',
```

```
    if myy ~= 1, gFine = gOrig;
    else, gFine = gOrig + 1;end
    finetuningEnabled = 0;
otherwise
    error(sprintf('Illegal value [ %s ] for parameter: ''finetune''\n', ...
        finetune));
end

if finetuningEnabled
    fprintf('Finetuning enabled (nonlinearity: [ %s ]).\n', finetune);
end

switch lower(stabilization)
case 'on', stabilizationEnabled = 1;
case 'off'
    if myy ~= 1
        stabilizationEnabled = 1;
    else
        stabilizationEnabled = 0;
    end
otherwise
    error(sprintf('Illegal value [ %s ] for parameter: ''stabilization''\n', stabilization));
end

if stabilizationEnabled, fprintf('Using stabilized algorithm.\n');end

% Some other parameters
myyOrig = myy;
% When we start fine-tuning we'll set myy = myyK * myy
myyK = 0.01;
% How many times do we try for convergence until we give up.
failureLimit = 5;

usedNlinearity = gOrig;stroke = 0;notFine = 1;long = 0;

% Checking the value for initial state.
switch lower(initState)
case 'rand', initialStateMode = 0;
case 'guess',
    if size(guess,1) ~= size(whiteningMatrix,2)
        initialStateMode = 0;
        fprintf('Warning: size of initial guess is incorrect. Using random initial guess.\n');
    else
        initialStateMode = 1;
        if size(guess,2) < numOfIC
            fprintf('Warning: initial guess only for first ');
            fprintf('%d components. Using random initial guess for others.\n',size(guess,2));
            guess(:, size(guess, 2) + 1:numOfIC) = rand(vectorSize,numOfIC-size(guess,2))-.5;
        elseif size(guess,2)>numOfIC
            guess=guess(:,1:numOfIC);
            fprintf('Warning: Initial guess too large. The excess column are dropped.\n');
        end
        fprintf('Using initial guess.\n');
    end
otherwise
    error(sprintf('Illegal value [ %s ] for parameter: ''initState''\n', initState));
end

% Checking the value for display mode.
switch lower(displayMode)
case {'off', 'none'}, usedDisplay = 0;
case {'on', 'signals'}
    usedDisplay = 1;
    if (numSamples > 10000), fprintf('Warning: Data vectors are very long. Plotting may take long time.\n');end
    if (numOfIC > 25), fprintf('Warning: There are too many signals to plot. Plot may not look good.\n');end
case 'basis'
    usedDisplay = 2;
    if (numOfIC > 25),fprintf('Warning: There are too many signals to plot. Plot may not look good.\n');end
```

```
case 'filters'
    usedDisplay = 3;
    if (vectorSize > 25),fprintf('Warning: There are too many signals to plot. Plot may not look good.\n');end
otherwise
    error(sprintf('Illegal value [ %s ] for parameter: ''displayMode''\n', displayMode));
end

% The displayInterval can't be less than 1...
if displayInterval < 1, displayInterval = 1;end

fprintf('Starting ICA calculation...\n');
% SYMMETRIC APPROACH
if approachMode == 1,
    % set some parameters more...
    usedNlinearity = gOrig;stroke = 0;notFine = 1;long = 0;

    A = zeros(vectorSize, numOfIC);  % Dewhitened basis vectors.
    if initialStateMode == 0
        % Take random orthonormal initial vectors.
        B = orth(rand(vectorSize, numOfIC) - .5);
    elseif initialStateMode == 1
        % Use the given initial vector as the initial state
        B = whiteningMatrix * guess;
    end

    BOld = zeros(size(B));BOld2 = zeros(size(B));
    % This is the actual fixed-point iteration loop.
    for round = 1:maxNumIterations + 1,
        if round == maxNumIterations + 1,
            fprintf('No convergence after %d steps\n', maxNumIterations);
            fprintf('Note that the plots are probably wrong.\n');
            A=[]; W=[];return;
        end

        % Symmetric orthogonalization.
        B = B * real(inv(B' * B)^(1/2));

        % Test for termination condition. Note that we consider opposite
        % directions here as well.
        minAbsCos = min(abs(diag(B' * BOld)));minAbsCos2 = min(abs(diag(B' * BOld2)));

        if (1 - minAbsCos < epsilon)
            if finetuningEnabled & notFine
                fprintf('Initial convergence, fine-tuning: \n');
                notFine = 0; usedNlinearity = gFine; myy = myyK * myyOrig;
                BOld = zeros(size(B)); BOld2 = zeros(size(B));
            else
                fprintf('Convergence after %d steps\n', round);

                % Calculate the de-whitened vectors.
                A = dewhiteningMatrix * B; break;
            end
        elseif stabilizationEnabled
            if (~stroke) & (1 - minAbsCos2 < epsilon)
                fprintf('Stroke!\n');
                stroke = myy; myy = .5*myy;
                if mod(usedNlinearity,2) == 0, usedNlinearity = usedNlinearity + 1;end
            elseif stroke
                myy = stroke; stroke = 0;
                if (myy == 1) & (mod(usedNlinearity,2) ~= 0)
                    usedNlinearity = usedNlinearity - 1;
                end
            elseif (~long) & (round>maxNumIterations/2)
                fprintf('Taking long (reducing step size)\n');
                long = 1; myy = .5*myy;
                if mod(usedNlinearity,2) == 0
                    usedNlinearity = usedNlinearity + 1;
                end
```

```
    end
end

BOld2 = BOld;BOld = B;

% Show the progress...
if round == 1, fprintf('Step no. %d\n', round);
else, fprintf('Step no. %d, change in value of estimate: %.3f \n', round, 1 - minAbsCos);end

% Also plot the current state...
switch usedDisplay
case 1
    if rem(round, displayInterval) == 0,
        % There was and may still be other displaymodes 1D signals
        icaplot('dispsig',(X'*B)');
    end
case 2
    if rem(round, displayInterval) == 0,
        % ... and now there are :-)
        % 1D basis
        A = dewhiteningMatrix * B;icaplot('dispsig',A');
    end
case 3
    if rem(round, displayInterval) == 0,
        % ... and now there are :-)
        % 1D filters
        W = B' * whiteningMatrix;icaplot('dispsig',W);
    end
otherwise
end
drawnow;
switch usedNlinearity
case 10 % pow3
    B = (X * (( X' * B) .^ 3)) / numSamples - 3 * B;
case 11
    % optimoitu - epsilonin kokoisia eroja
    % tm on optimoitu koodi, katso vanha koodi esim.
    % aikaisemmista versioista kuten 2.0 beta3
    Y = X' * B; Gpow3 = Y .^ 3; Beta = sum(Y .* Gpow3);
    D = diag(1 ./ (Beta - 3 * numSamples));
    B = B + myy * B * (Y' * Gpow3 - diag(Beta)) * D;
case 12
    Xsub=X(:, getSamples(numSamples, sampleSize));
    B = (Xsub * (( Xsub' * B) .^ 3)) / size(Xsub,2) - 3 * B;
case 13
    % Optimoitu
    Ysub=X(:, getSamples(numSamples, sampleSize))' * B;
    Gpow3 = Ysub .^ 3; Beta = sum(Ysub .* Gpow3);
    D = diag(1 ./ (Beta - 3 * size(Ysub', 2)));
    B = B + myy * B * (Ysub' * Gpow3 - diag(Beta)) * D;
case 20 % tanh
    hypTan = tanh(a1 * X' * B);
    B = X * hypTan / numSamples - ...
        ones(size(B,1),1) * sum(1 - hypTan .^ 2) .* B / numSamples * a1;
case 21
    % optimoitu - epsilonin kokoisia
    Y = X' * B; hypTan = tanh(a1 * Y); Beta = sum(Y .* hypTan);
    D = diag(1 ./ (Beta - a1 * sum(1 - hypTan .^ 2)));
    B = B + myy * B * (Y' * hypTan - diag(Beta)) * D;
case 22
    Xsub=X(:, getSamples(numSamples, sampleSize));
    hypTan = tanh(a1 * Xsub' * B);
    B = Xsub * hypTan / size(Xsub, 2) - ...
        ones(size(B,1),1) * sum(1 - hypTan .^ 2) .* B / size(Xsub, 2) * a1;
case 23
    % Optimoitu
    Y = X(:, getSamples(numSamples, sampleSize))' * B;
    hypTan = tanh(a1 * Y); Beta = sum(Y .* hypTan);
```

```
        D = diag(1 ./ (Beta - a1 * sum(1 - hypTan .^ 2)));
        B = B + myy * B * (Y' * hypTan - diag(Beta)) * D;
        % gauss
    case 30
        U = X' * B; Usquared=U .^ 2; ex = exp(-a2 * Usquared / 2);
        gauss =  U .* ex; dGauss = (1 - a2 * Usquared) .*ex;
        B = X * gauss / numSamples - ...
            ones(size(B,1),1) * sum(dGauss)...
            .* B / numSamples ;
    case 31
        % optimoitu
        Y = X' * B; ex = exp(-a2 * (Y .^ 2) / 2); gauss = Y .* ex;
        Beta = sum(Y .* gauss);
        D = diag(1 ./ (Beta - sum((1 - a2 * (Y .^ 2)) .* ex)));
        B = B + myy * B * (Y' * gauss - diag(Beta)) * D;
    case 32
        Xsub=X(:, getSamples(numSamples, sampleSize));
        U = Xsub' * B; Usquared=U .^ 2;
        ex = exp(-a2 * Usquared / 2); gauss =  U .* ex;
        dGauss = (1 - a2 * Usquared) .*ex;
        B = Xsub * gauss / size(Xsub,2) - ...
            ones(size(B,1),1) * sum(dGauss)...
            .* B / size(Xsub,2) ;
    case 33
        % Optimoitu
        Y = X(:, getSamples(numSamples, sampleSize))' * B;
        ex = exp(-a2 * (Y .^ 2) / 2); gauss = Y .* ex;
        Beta = sum(Y .* gauss);
        D = diag(1 ./ (Beta - sum((1 - a2 * (Y .^ 2)) .* ex)));
        B = B + myy * B * (Y' * gauss - diag(Beta)) * D;
        % skew
    case 40
        B = (X * ((X' * B) .^ 2)) / numSamples;
    case 41
        % Optimoitu
        Y = X' * B; Gskew = Y .^ 2; Beta = sum(Y .* Gskew);
        D = diag(1 ./ (Beta));
        B = B + myy * B * (Y' * Gskew - diag(Beta)) * D;
    case 42
        Xsub=X(:, getSamples(numSamples, sampleSize));
        B = (Xsub * ((Xsub' * B) .^ 2)) / size(Xsub,2);
    case 43
        % Uusi optimoitu
        Y = X(:, getSamples(numSamples, sampleSize))' * B;
        Gskew = Y .^ 2; Beta = sum(Y .* Gskew);
        D = diag(1 ./ (Beta));
        B = B + myy * B * (Y' * Gskew - diag(Beta)) * D;
    otherwise
        error('Code for desired nonlinearity not found!');
    end
end

% Calculate ICA filters.
W = B' * whiteningMatrix;

% Also plot the last one...
switch usedDisplay
case 1
    % There was and may still be other displaymodes...
    % 1D signals
    icaplot('dispsig',(X'*B)');drawnow;
case 2
    % ... and now there are :-)
    % 1D basis
    icaplot('dispsig',A');drawnow;
case 3
    % ... and now there are :-)
    % 1D filters
```

```
        icaplot('dispsig',W);drawnow;
    otherwise
    end
end

% DEFLATION APPROACH
if approachMode == 2
    B = zeros(vectorSize);
    % The search for a basis vector is repeated numOfIC times.
    round = 1; numFailures = 0;
    while round <= numOfIC,
        myy = myyOrig; usedNlinearity = gOrig; stroke = 0; notFine = 1; long = 0; endFinetuning = 0;

        % Show the progress...
        fprintf('IC %d ', round);

        % Take a random initial vector of lenght 1 and orthogonalize it
        % with respect to the other vectors.
        if initialStateMode == 0
            w = rand(vectorSize, 1) - .5;
        elseif initialStateMode == 1
            w=whiteningMatrix*guess(:,round);
        end
        w = w - B * B' * w; w = w / norm(w);

        wOld = zeros(size(w));wOld2 = zeros(size(w));

        % This is the actual fixed-point iteration loop.
        %    for i = 1 : maxNumIterations + 1
        i = 1; gabba = 1;
        while i <= maxNumIterations + gabba
            drawnow;
            % Project the vector into the space orthogonal to the space
            % spanned by the earlier found basis vectors. Note that we can do
            % the projection with matrix B, since the zero entries do not
            % contribute to the projection.
            w = w - B * B' * w; w = w / norm(w);

            if notFine
                if i == maxNumIterations + 1
                    fprintf('\nComponent number %d did not converge in %d iterations.\n',...
                          round, maxNumIterations);
                    round = round - 1;
                    numFailures = numFailures + 1;
                    if numFailures > failureLimit
                        fprintf('Too many failures to converge (%d). Giving up.\n', numFailures);
                        if round == 0,  A=[]; W=[]; end
                        return;
                    end
                    % numFailures > failurelimit
                    break;
                end
                % i == maxNumIterations + 1
            else
                % if notFine
                if i >= endFinetuning
                    wOld = w; % So the algorithm will stop on the next test...
                end
            end
            % if notFine

            % Show the progress...
            fprintf('.');

            % Test for termination condition. Note that the algorithm has
            % converged if the direction of w and wOld is the same, this
            % is why we test the two cases.
            if norm(w - wOld) < epsilon | norm(w + wOld) < epsilon
```

```
    if finetuningEnabled & notFine
        fprintf('Initial convergence, fine-tuning: ');
        notFine = 0; gabba = maxFinetune; wOld = zeros(size(w));
        wOld2 = zeros(size(w)); usedNlinearity = gFine;
        myy = myyK * myyOrig; endFinetuning = maxFinetune + i;
    else
        numFailures = 0;
        % Save the vector
        B(:, round) = w;
        % Calculate the de-whitened vector.
        A(:,round) = dewhiteningMatrix * w;
        % Calculate ICA filter.
        W(round,:) = w' * whiteningMatrix;
        % Show the progress...
        fprintf('computed ( %d steps ) \n', i);
        % Also plot the current state...
        switch usedDisplay
        case 1
            if rem(round, displayInterval) == 0,
                % There was and may still be other displaymodes...
                % 1D signals
                temp = X'*B; icaplot('dispsig',temp(:,1:numOfIC)'); drawnow;
            end
        case 2
            if rem(round, displayInterval) == 0,
                % ... and now there are :-)
                % 1D basis
                icaplot('dispsig',A');drawnow;
            end
        case 3
            if rem(round, displayInterval) == 0,
                % ... and now there are :-)
                % 1D filters
                icaplot('dispsig',W); drawnow;
            end
        end
        % switch usedDisplay
        break; % IC ready - next...
    end
    %if finetuningEnabled & notFine
elseif stabilizationEnabled
    if (~stroke) & (norm(w - wOld2) < epsilon | norm(w + wOld2) < ...
            epsilon)
        stroke = myy; fprintf('Stroke!'); myy = .5*myy;
        if mod(usedNlinearity,2) == 0
            usedNlinearity = usedNlinearity + 1;
        end
    elseif stroke
        myy = stroke; stroke = 0;
        if (myy == 1) & (mod(usedNlinearity,2) ~= 0)
            usedNlinearity = usedNlinearity - 1;
        end
    elseif (notFine) & (~long) & (i > maxNumIterations / 2)
        fprintf('Taking long (reducing step size) ');
        long = 1; myy = .5*myy;
        if mod(usedNlinearity,2) == 0
            usedNlinearity = usedNlinearity + 1;
        end
    end
end
wOld2 = wOld; wOld = w;

switch usedNlinearity
    % pow3
case 10
    w = (X * ((X' * w) .^ 3)) / numSamples - 3 * w;
case 11
    EXGpow3 = (X * ((X' * w) .^ 3)) / numSamples;
```

```
    Beta = w' * EXGpow3;
    w = w - myy * (EXGpow3 - Beta * w) / (3 - Beta);
case 12
    Xsub=X(:,getSamples(numSamples, sampleSize));
    w = (Xsub * ((Xsub' * w) .^ 3)) / size(Xsub, 2) - 3 * w;
case 13
    Xsub=X(:,getSamples(numSamples, sampleSize));
    EXGpow3 = (Xsub * ((Xsub' * w) .^ 3)) / size(Xsub, 2);
    Beta = w' * EXGpow3;
    w = w - myy * (EXGpow3 - Beta * w) / (3 - Beta);
    % tanh
case 20
    hypTan = tanh(a1 * X' * w);
    w = (X * hypTan - a1 * sum(1 - hypTan .^ 2)' * w) / numSamples;
case 21
    hypTan = tanh(a1 * X' * w);
    Beta = w' * X * hypTan;
    w = w - myy * ((X * hypTan - Beta * w) / ...
        (a1 * sum((1-hypTan .^2)') - Beta));
case 22
    Xsub=X(:,getSamples(numSamples, sampleSize));
    hypTan = tanh(a1 * Xsub' * w);
    w = (Xsub * hypTan - a1 * sum(1 - hypTan .^ 2)' * w) / size(Xsub, 2);
case 23
    Xsub=X(:,getSamples(numSamples, sampleSize));
    hypTan = tanh(a1 * Xsub' * w);
    Beta = w' * Xsub * hypTan;
    w = w - myy * ((Xsub * hypTan - Beta * w) / ...
        (a1 * sum((1-hypTan .^2)') - Beta));
    % gauss
case 30
    % This has been split for performance reasons.
    u = X' * w; u2=u.^2; ex=exp(-a2 * u2/2);
    gauss =  u.*ex;
    dGauss = (1 - a2 * u2) .*ex;
    w = (X * gauss - sum(dGauss)' * w) / numSamples;
case 31
    u = X' * w; u2=u.^2; ex=exp(-a2 * u2/2);
    gauss =  u.*ex; dGauss = (1 - a2 * u2) .*ex;
    Beta = w' * X * gauss;
    w = w - myy * ((X * gauss - Beta * w) / ...
        (sum(dGauss)' - Beta));
case 32
    Xsub=X(:,getSamples(numSamples, sampleSize));
    u = Xsub' * w; u2=u.^2; ex=exp(-a2 * u2/2);
    gauss =  u.*ex; dGauss = (1 - a2 * u2) .*ex;
    w = (Xsub * gauss - sum(dGauss)' * w) / size(Xsub, 2);
case 33
    Xsub=X(:,getSamples(numSamples, sampleSize));
    u = Xsub' * w; u2=u.^2;
    ex=exp(-a2 * u2/2); gauss =  u.*ex;
    dGauss = (1 - a2 * u2) .*ex;
    Beta = w' * Xsub * gauss;
    w = w - myy * ((Xsub * gauss - Beta * w) / ...
        (sum(dGauss)' - Beta));
    % skew
case 40
    w = (X * ((X' * w) .^ 2)) / numSamples;
case 41
    EXGskew = (X * ((X' * w) .^ 2)) / numSamples;
    Beta = w' * EXGskew;
    w = w - myy * (EXGskew - Beta*w)/(-Beta);
case 42
    Xsub=X(:,getSamples(numSamples, sampleSize));
    w = (Xub * ((Xub' * w) .^ 2)) / size(Xsub, 2);
case 43
    Xsub=X(:,getSamples(numSamples, sampleSize));
    EXGskew = (Xsub * ((Xsub' * w) .^ 2)) / size(Xsub, 2);
```

```
                Beta = w' * EXGskew;
                w = w - myy * (EXGskew - Beta*w)/(-Beta);
            otherwise
                error('Code for desired nonlinearity not found!');
            end

            % Normalize the new w.
            w = w / norm(w);i = i + 1;
        end
        round = round + 1;
    end
    fprintf('Done.\n');

    % Also plot the ones that may not have been plotted.
    if (usedDisplay > 0) & (rem(round-1, displayInterval) ~= 0)
        switch usedDisplay
        case 1
            % There was and may still be other displaymodes...
            % 1D signals
            temp = X'*B;icaplot('dispsig',temp(:,1:numOfIC)');drawnow;
        case 2
            % ... and now there are :-)
            % 1D basis
            icaplot('dispsig',A');drawnow;
        case 3
            % ... and now there are :-)
            % 1D filters
            icaplot('dispsig',W); drawnow;
        otherwise
        end
    end
end

% In the end let's check the data for some security
if ~isreal(A)
    fprintf('Warning: removing the imaginary part from the result.\n');
    A = real(A);  W = real(W);
end

% Subfunction
% Calculates tanh simplier and faster than Matlab tanh.
function y=tanh(x)
y = 1 - 2 ./ (exp(2 * x) + 1);

%%%%%%%%%%1%%%%%%%%%%2%%%%%%%%%%3%%%%%%%%%%4%%%%%%%%%%5%%%%%%%%%%6
function Samples = getSamples(max, percentage)
Samples = find(rand(1, max) < percentage);
```

remmean.m

```
function [newVectors, meanValue] = remmean(vectors);
%REMMEAN - remove the mean from vectors
newVectors = zeros (size (vectors));
meanValue = mean (vectors')';
newVectors = vectors-meanValue*ones(1,size(vectors,2));
```

pcamat.m

```
%%%%%%%%%%1%%%%%%%%%%2%%%%%%%%%%3%%%%%%%%%%4%%%%%%%%%%5%%%%%%%%%%6
function [E,D]=pcamat(vectors,firstEig,lastEig,s_interactive);
%
% [E, D] = pcamat(vectors, firstEig, lastEig, ...
%                 interactive, verbose);
%
% Calculates the PCA matrices for given data (row) vectors. Returns
% the eigenvector (E) and diagonal eigenvalue (D) matrices containing the
% selected subspaces. Dimensionality reduction is controlled with
```

```
% the parameters 'firstEig' and 'lastEig' - but it can also be done
% interactively by setting parameter 'interactive' to 'on' or 'gui'.
%
% ARGUMENTS
%
% vectors       Data in row vectors.
% firstEig      Index of the largest eigenvalue to keep.
%               Default is 1.
% lastEig       Index of the smallest eigenvalue to keep.
%               Default is equal to dimension of vectors.
% interactive   Specify eigenvalues to keep interactively. Note that if
%               you set 'interactive' to 'on' or 'gui' then the values
%               for 'firstEig' and 'lastEig' will be ignored, but they
%               still have to be entered. If the value is 'gui' then the
%               same graphical user interface as in FASTICAG will be
%               used. Default is 'off'.
% verbose       Default is 'on'.
%
%
% EXAMPLE
%       [E, D] = pcamat(vectors);
%
% Note
%       The eigenvalues and eigenvectors returned by PCAMAT are not sorted.
%
% This function is needed by FASTICA and FASTICAG

% For historical reasons this version does not sort the eigenvalues or
% the eigen vectors in any ways. Therefore neither does the FASTICA or
% FASTICAG. Generally it seams that the components returned from
% whitening is almost in reversed order. (That means, they usually are,
% but sometime they are not - depends on the EIG-command of matlab.)

% 16.6.2000
% Hugo Gvert

% Check the optional parameters;

switch lower(s_interactive)
 case 'on', b_interactive = 1;
 case 'off', b_interactive = 0;
 case 'gui', b_interactive = 2;
 otherwise
  error(sprintf('Illegal value [ %s ] for parameter: ''interactive''\n', ...
        s_interactive));
end

oldDimension = size (vectors, 1);
if ~(b_interactive)
  if lastEig < 1 | lastEig > oldDimension
    error(sprintf('Illegal value [ %d ] for parameter: ''lastEig''\n', lastEig));
  end
  if firstEig < 1 | firstEig > lastEig
    error(sprintf('Illegal value [ %d ] for parameter: ''firstEig''\n', firstEig));
  end
end

% Calculate PCA
% Calculate the covariance matrix.
fprintf ('Calculating covariance...\n');covarianceMatrix = cov(vectors', 1);
maxLastEig = rank(covarianceMatrix, 1e-9);
% Calculate the eigenvalues and eigenvectors of covariance matrix.
[E, D] = eig(covarianceMatrix);
% Sort the eigenvalues - decending.
eigenvalues = flipud(sort(diag(D)));
% Interactive part - command-line
if b_interactive == 1
  % Show the eigenvalues to the user
```

```
  hndl_win=figure; bar(eigenvalues); title('Eigenvalues');

  % ask the range from the user...
  % ... and keep on asking until the range is valid :-)
  areValuesOK=0;
  while areValuesOK == 0
    firstEig = input('The index of the largest eigenvalue to keep? (1) ');
    lastEig = input(['The index of the smallest eigenvalue to keep? (' ...
                     int2str(oldDimension) ') ']);
    % Check the new values...
    % if they are empty then use default values
    if isempty(firstEig), firstEig = 1;end
    if isempty(lastEig), lastEig = oldDimension;end
    % Check that the entered values are within the range
    areValuesOK = 1;
    if lastEig < 1 | lastEig > oldDimension
      fprintf('Illegal number for the last eigenvalue.\n');
      areValuesOK = 0;
    end
    if firstEig < 1 | firstEig > lastEig
      fprintf('Illegal number for the first eigenvalue.\n');
      areValuesOK = 0;
    end
  end
  % close the window
  close(hndl_win);
end

% See if the user has reduced the dimension enough
if lastEig > maxLastEig
  lastEig = maxLastEig;
  fprintf('Dimension reduced to %d due to the singularity of covariance matrix\n',...
           lastEig-firstEig+1);
else
  % Reduce the dimensionality of the problem.
  if oldDimension == (lastEig - firstEig + 1)
     fprintf ('Dimension not reduced.\n');
  else
     fprintf ('Reducing dimension...\n');
  end
end

% Drop the smaller eigenvalues
if lastEig < oldDimension
  lowerLimitValue = (eigenvalues(lastEig) + eigenvalues(lastEig + 1)) / 2;
else
  lowerLimitValue = eigenvalues(oldDimension) - 1;
end
lowerColumns = diag(D) > lowerLimitValue;

% Drop the larger eigenvalues
if firstEig > 1
  higherLimitValue = (eigenvalues(firstEig - 1) + eigenvalues(firstEig)) / 2;
else
  higherLimitValue = eigenvalues(1) + 1;
end
higherColumns = diag(D) < higherLimitValue;

% Combine the results from above
selectedColumns = lowerColumns & higherColumns;

% print some info for the user
fprintf ('Selected [ %d ] dimensions.\n', sum (selectedColumns));
if sum (selectedColumns) ~= (lastEig - firstEig + 1),
  error ('Selected a wrong number of dimensions.');
end

fprintf ('Smallest remaining (non-zero) eigenvalue [ %g ]\n', eigenvalues(lastEig));
```

```
fprintf ('Largest remaining (non-zero) eigenvalue [ %g ]\n', eigenvalues(firstEig));
fprintf ('Sum of removed eigenvalues [ %g ]\n', sum(diag(D) .* (~selectedColumns)));

% Select the colums which correspond to the desired range
% of eigenvalues.
E = selcol(E, selectedColumns);
D = selcol(selcol(D, selectedColumns)', selectedColumns);

% Some more information
sumAll=sum(eigenvalues);
sumUsed=sum(diag(D));
retained = (sumUsed / sumAll) * 100;
fprintf('[ %g ] %% of (non-zero) eigenvalues retained.\n', retained);

function newMatrix = selcol(oldMatrix, maskVector);
% newMatrix = selcol(oldMatrix, maskVector);
%
% Selects the columns of the matrix that marked by one in the given vector.
% The maskVector is a column vector.

if size(maskVector, 1) ~= size(oldMatrix, 2),
  error ('The mask vector and matrix are of uncompatible size.');
end
numTaken = 0;

for i = 1 : size (maskVector, 1),
  if maskVector(i, 1) == 1,
    takingMask(1, numTaken + 1) = i; numTaken = numTaken + 1;
  end
end
newMatrix = oldMatrix(:, takingMask);
```

whitenv.m

```
function [newVectors,whiteningMatrix,dewhiteningMatrix]=whitenv(vectors, E, D);
%WHITENV - Whitenv vectors.
% Whitens the data (row vectors) and reduces dimension. Returns
% the whitened vectors, whitening and dewhitening matrices.
% vectors       Data in row vectors.
% E             Eigenvector matrix from function 'pcamat'
% D             Diagonal eigenvalue matrix from function 'pcamat'

% Calculate the whitening and dewhitening matrices
%  (these handle dimensionality simultaneously).
whiteningMatrix = inv (sqrt (D)) * E';
dewhiteningMatrix = E * sqrt (D);

% Project to the eigenvectors of the covariance matrix.
% Whiten the samples and reduce dimension simultaneously.
fprintf ('Whitening...\n');
newVectors =  whiteningMatrix * vectors;

% Just some security...
if ~isreal(newVectors)
  error ('Whitened vectors have imaginary values.');
end
fprintf('Check: covariance differs from identity by [ %g ].\n', ...
    max(max(abs(cov(newVectors',1)-eye(size(newVectors,1))))));
```

SUDICA.m

```
% Comparison: SUD vs ICA vs ICA-SUD over AWGN downlink
%  User number: K=30? The 31th is the Gussian Noise
%  SNR=0dB to 30dB?

%  # of symbols M=10000
clear all; clc; clf;
```

```
load gold5.mat; load mixedsig.mat;
low_SNRindB=-10; high_SNRindB=0;step=1;
SNRindB=low_SNRindB:step:high_SNRindB;
for ii=1:length(SNRindB)
    SNR=10^((SNRindB(ii))/10);
    R=sum(S)+AWGN/sqrt(2*SNR);

    clear tmp; clear BER; clear temp;
    R_ICA=reshape(R,K+1,M);
    [Bhat_ICA1 G1 G_inv1]=y1fastica(R_ICA);
    [Bhat_ICA2 G1 G_inv1]=y2fastica(R_ICA);
    [Bhat_ICA3 G1 G_inv1]=y1fastica(R_ICA);

    % BER_ICA cal
    Bhat_ICA=sign(Bhat_ICA1);
    for i=1:K
        for j=1:K+1
            % max or min value will be expected
            tmp(j)=sum(Bitstream(i,:).*Bhat_ICA(j,:));
        end
        BER(i)=1-max(abs(tmp))/M;
    end
    BER_mean_ICA(ii)=mean(BER);

    clear tmp;clear BER;
    for i=1:K
        tmp=reshape(R,31,size(R,2)/31);
        for j=1:M
            tmp(:,j)=tmp(:,j).*gold(i,:)';
        end
        Bhat_SUD(i,:)=sign(sum(tmp)); % SUD decision

        % BER_SUD cal
        BER(i)=sum((Bhat_SUD(i,:).*Bitstream(i,:)-1)/(-2))/M;

        % BER_SUDICA cal - compare with ICA1
        for j=1:K
            temp(j,:)=Bhat_SUD(i,:)+sign(Bhat_ICA1(j,:));
        end
        tmp=sum(abs(temp')); clear temp;
        if (min(tmp)>(2*M-max(tmp)))
            [temp(1) temp(2)]=max(tmp);
            Bhat_SUDICA(i,:)=Bhat_SUD(i,:)+sign(Bhat_ICA1(temp(2),:));
        else
            [temp(1) temp(2)]=min(tmp);
            Bhat_SUDICA(i,:)=Bhat_SUD(i,:)-sign(Bhat_ICA1(temp(2),:));
        end
        clear temp; clear tmp;

        % BER_SUDICA cal - compare with ICA2
        for j=1:K
            temp(j,:)=Bhat_SUD(i,:)+sign(Bhat_ICA2(j,:));
        end
        tmp=sum(abs(temp'));
        clear temp;
        if (min(tmp)>(2*M-max(tmp)))
            [temp(1) temp(2)]=max(tmp);
            Bhat_SUDICA(i,:)=Bhat_SUDICA(i,:)+sign(Bhat_ICA2(temp(2),:));
        else
            [temp(1) temp(2)]=min(tmp);
            Bhat_SUDICA(i,:)=Bhat_SUDICA(i,:)-sign(Bhat_ICA2(temp(2),:));
        end
        clear temp; clear tmp;

        Bhat_SUDICA(i,:)=sign(Bhat_SUDICA(i,:));
        BER_SUDICA(i)=sum((Bhat_SUDICA(i,:).*Bitstream(i,:)-1)/(-2))/M;
    end
    BER_mean_SUD(ii)=mean(BER);
```

```
    BER_mean_SUDICA(ii)=mean(BER_SUDICA);
end
clear tmp; clear BER; clear temp; clear BER_SUDICA;

save result11.mat BER_mean_SUD BER_mean_ICA BER_mean_SUDICA low_SNRindB high_SNRindB M;
```

yfastica.m

```
function [Out1, Out2, Out3] = yfastica(mixedsig)
%[mixedsig, mixedmean] = remmean(mixedsig);
[Dim, NumOfSampl] = size(mixedsig);

% Calculating PCA
% We need the results of PCA for whitening, but if we don't
% need to do whitening... then we dont need PCA...
covarianceMatrix = cov(mixedsig', 1); % Calculate the covariance matrix.
% Calculate the eigenvalues and eigenvectors of covariance matrix.
[E, D] = eig(covarianceMatrix);

% Calculate the whitening and dewhitening matrices
% (these handle dimensionality simultaneously).
whiteningMatrix = inv (sqrt (D)) * E';
dewhiteningMatrix = E * sqrt (D);
% Project to the eigenvectors of the covariance matrix.
% Whiten the samples and reduce dimension simultaneously.
whitesig =  whiteningMatrix * mixedsig;

% Calculate the ICA with fixed point algorithm.
[A, W] = yfpica (whitesig,  whiteningMatrix, dewhiteningMatrix);

Out1 = W * mixedsig; Out2 = A; Out3 = W;

function [A, W] = yfpica(X, whiteningMatrix, dewhiteningMatrix);
failureLimit = 5; maxNumIterations = 1000;
finetuningEnabled =0; notFine =1;

% epsilon = 0.0001;
 epsilon = 0.0001;

[vectorSize, numSamples] = size(X); numOfIC = vectorSize;

B = zeros(vectorSize); %579 DEFLATION APPROACH
% The search for a basis vector is repeated numOfIC times.
round = 1; numFailures = 0;
while round <= numOfIC, %586
    endFinetuning = 0;

    fprintf('IC %d ', round);    % Show the progress...

    % Take a random initial vector of lenght 1 and orthogonalize it
    % with respect to the other vectors.

    % random real initial value
    % w = rand(vectorSize, 1) - .5; %601
    % all-zero initial value
    % w = zeros(vectorSize, 1) -.5;
    % random binary initial value
    % load w.mat; w = w1; % w = randsrc(vectorSize, 1, [1,-1]);
    load gold5.mat; w=(gold(round,:))';
    % w=PN5_1245
    % w=[1 -1 -1 -1 -1 1 1 -1 1 -1 1 -1 -1 1 -1 -1 -1 1 -1 1 1 1 1 1 1 -1 1 1 -1 -1 1 1]';

    w = w - B * B' * w; w = w / norm(w);
    wOld = zeros(size(w)); wOld2 = zeros(size(w));

    % This is the actual fixed-point iteration loop.
    %    for i = 1 : maxNumIterations + 1
```

```
    i = 1; gabba = 1;
    while i <= maxNumIterations + gabba %615

        % Project the vector into the space orthogonal to the space
        % spanned by the earlier found basis vectors. Note that we can do
        % the projection with matrix B, since the zero entries do not
        % contribute to the projection.
        w = w - B * B' * w; %628
        w = w / norm(w);

        if notFine
            if i == maxNumIterations + 1
                fprintf('\nIC %d did not converge in %d iterations.\n', round, maxNumIterations);
                round = round - 1;  numFailures = numFailures + 1;
                if numFailures > failureLimit
                    fprintf('Too many failures to converge (%d).\n', numFailures);
                    if round == 0
                        A=[]; W=[];
                    end
                    return;
                end
                round = round + 1;
                break;
            end
        else
            if i >= endFinetuning
                wOld = w; % So the algorithm will stop on the next test...
            end
        end

        fprintf('.'); %662 Show the progress...

        % Test for termination condition. Note that the algorithm has
        % converged if the direction of w and wOld is the same, this
        % is why we test the two cases.
        if norm(w - wOld) < epsilon | norm(w + wOld) < epsilon %668
        % if abs(sum(sign(X'*w))-sum(sign(X'*wOld)))<10
            numFailures = 0; %681
            B(:, round) = w; % Save the vector
            A(:, round) = dewhiteningMatrix * w; % Calculate the de-whitened vector
            W(round,:) = w' * whiteningMatrix; % Calculate ICA filter

            fprintf('computed ( %d steps ) \n', i); % Show the progress...
            % temp = X'*B; icaplot('dispsig',temp(:,1:numOfIC)'); drawnow;

            break; % 721 IC ready - next...
        end
        wOld2 = wOld; wOld = w; %749

        w = (X * ((X' * w) .^ 3)) / numSamples - 3 * w; %755

        w = w / norm(w); i = i + 1; %844 Normalize the new w.
    end
    round = round + 1; %847
end
fprintf('Done.\n');

% In the end let's check the data for some security
if ~isreal(A)
    if b_verbose, fprintf('Warning: removing the imaginary part from the result.\n'); end
    A = real(A); W = real(W);
end
```