

# Iterative Blind Separation of Gaussian Data of Unknown Order

by

Amy Mueller

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

© Amy Mueller, MMIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and  
distribute publicly paper and electronic copies of this thesis document  
in whole or in part.

Author

Department of Electrical Engineering and Computer Science

May 9, 2003


Certified by . . . . .

  
David H. Staelin

Professor of Electrical Engineering

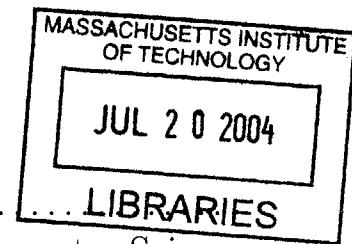
Thesis Supervisor

Accepted by . . . . .

  
Arthur C. Smith

Chairman, Department Committee on Graduate Students

This work was supported by the Department of the Air Force under  
contract F19628-00-C-0002.



**BARKER**



# Iterative Blind Separation of Gaussian Data of Unknown Order

by

Amy Mueller

Submitted to the Department of Electrical Engineering and Computer Science  
on May 9, 2003, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

A method for blind separation of noisy jointly Gaussian multivariate signals  $\mathbf{X}$  is presented, where  $\mathbf{X} = \mathbf{A}\mathbf{P} + \mathbf{N}$ ,  $\mathbf{X}$  is an observed set of vectors,  $\mathbf{A}$  is the mixing matrix,  $\mathbf{P}$  is the unknown signal matrix, and  $\mathbf{N}$  is white noise. The objective is to estimate all matrices on the right-hand side when even their dimensions (the system order) are unknown. The algorithms developed are extensions of the Iterative Order and Noise (ION) estimation algorithm [10]. Improvements made within the iterative structure of ION to better estimate the order and noise yield ION'. The addition of a second-order blind identification algorithm (SOBI, [4]) subsequently yields ONA, which fully characterizes a data set by estimating the (O)rder, (N)oise, and mixing matrix ( $\mathbf{A}$ ). Metrics are developed to evaluate the performance of these algorithms, and their applicability is discussed. Optimum algorithm constants for ION' and ONA are derived, and their range of applicability is outlined.

The algorithms are evaluated through application to three types of data: (1) simulated Gaussian data which spans the problem space, (2) a set of non-Gaussian factory data with 577 variables, and (3) a hyperspectral image with 224 channels. The ONA algorithm is extended to 2D (spatial) hyperspectral problems by exploiting spatial rather than time correlation. ONA produces a full characterization of the data with high signal-to-noise ratios for most unknown parameters in the Gaussian case, though the jointly Gaussian  $\mathbf{P}$  is shown to be most difficult to retrieve. In all three cases, ONA reduces the noise in the data, identifies small sets of highly correlated variables, and unmixes latent signals. The spatial ONA identifies surface features in the hyperspectral image and retrieves sources significantly more independent than those retrieved by PCA. Further exploration of the applicability of these algorithms to other types of data and further algorithmic improvement is recommended.

Thesis Supervisor: David H. Staelin  
Title: Professor of Electrical Engineering





## Acknowledgments

During my last month at MIT, I was asked to stage manage a play written by fellow student Max Goldman. In it, one of the characters reflects on a mundane–yet profound–interaction she has with a complete stranger:

“I was experiencing the power of my unconscious mind, the power that one person... can have on another person without either of them realizing it. That exchange is an essential feature of humanity.”

For everybody who has affected me, changed me, helped me grow as a person and make my way to where I am today, with or without my consciously realizing it, I am grateful.



# Contents

<b>1</b>	<b>Introduction to the Problem</b>	<b>17</b>
1.1	Introduction . . . . .	17
1.2	Previous work done with BSS . . . . .	17
1.3	Motivation for ION' and ONA . . . . .	18
<b>2</b>	<b>Improvements to the ION Algorithm: ION' and ONA</b>	<b>21</b>
2.1	Introduction . . . . .	21
2.2	Problem Definition . . . . .	22
2.3	Improved Iterative-Order-Noise (ION') Algorithm . . . . .	23
2.3.1	Algorithm Overview . . . . .	23
2.3.2	Improved Order Estimation . . . . .	24
2.3.3	EM Algorithm . . . . .	25
2.3.4	Iteration Determination . . . . .	26
2.4	Order-Noise-A (ONA) Algorithm . . . . .	28
2.4.1	Algorithm Overview . . . . .	28
2.4.2	Iteration Determination . . . . .	30
2.4.3	Addition of SOBI . . . . .	30
<b>3</b>	<b>Performance Evaluation: Metrics</b>	<b>35</b>
3.1	Introduction . . . . .	35
3.2	Signal-to-Noise Ratio (SNR) . . . . .	35
3.3	Order Estimation Metrics . . . . .	36
3.4	Concentration: Variable Grouping . . . . .	37

3.5	Evaluation of Estimated Source Quality . . . . .	38
3.5.1	Focus . . . . .	40
3.5.2	Visual Metric . . . . .	43
<b>4</b>	<b>Algorithm Optimization</b>	<b>47</b>
4.1	Introduction . . . . .	47
4.2	Order Estimation Parameters . . . . .	47
4.3	Trace of $\mathbf{G}$ . . . . .	48
4.4	Iteration Selection Parameters . . . . .	48
<b>5</b>	<b>Algorithm Evaluation: Exploring the Problem Space</b>	<b>51</b>
5.1	Introduction . . . . .	51
5.2	Degrees of Freedom in the Problem Space . . . . .	52
5.2.1	Number of Variables ( $n$ ) . . . . .	52
5.2.2	Fraction of Latent Sources ( $\frac{k}{n}$ ) . . . . .	52
5.2.3	Number of Samples ( $m$ ) . . . . .	52
5.2.4	Matrix Signal-to-Noise Ratio ( $SNR_{\mathbf{X}}$ ) . . . . .	53
5.2.5	Distribution of Singular Values of the Mixing Matrix ( $\Delta\lambda_{\mathbf{A}}$ ) . . . . .	53
5.3	Method for Exploring the Problem Space: Simulated Gaussian Tests . . . . .	54
5.3.1	Nominal Gaussian Test . . . . .	54
5.3.2	Varied Gaussian Tests . . . . .	56
5.3.3	Varied Gaussian Tests with less Noise . . . . .	58
<b>6</b>	<b>Example Data Set: Factory Data</b>	<b>61</b>
6.1	Introduction to Non-Gaussian Data . . . . .	61
6.2	Noise Reduction . . . . .	62
6.3	Variable Grouping . . . . .	64
6.4	Source Separation . . . . .	65
<b>7</b>	<b>Extension to 2D Spaces</b>	<b>73</b>
7.1	Applications of Extended Algorithm . . . . .	73
7.2	Use of Spatial Correlation in SOBI . . . . .	74

7.2.1	Nearest Neighbors . . . . .	74
7.2.2	Creation of Spatial Covariance Matrices . . . . .	76
7.2.3	Dealing with Edge Effects: Mirroring . . . . .	77
7.3	Memory Management . . . . .	78
<b>8</b>	<b>Evaluation of 2D ONA: Hyperspectral Data</b>	<b>81</b>
8.1	Hyperspectral Data Overview . . . . .	81
8.2	Evaluated Image . . . . .	82
8.3	Noise Reduction . . . . .	83
8.4	Variable Grouping . . . . .	87
8.5	Signal Separation . . . . .	87
<b>9</b>	<b>Conclusions and Suggested Further Work</b>	<b>93</b>
9.1	Conclusions . . . . .	93
9.2	Future Work . . . . .	94
9.2.1	Improvements to the Iteration Determination Algorithm . . . . .	94
9.2.2	Hyperspectral Applications . . . . .	94
9.2.3	Application to Varied Data . . . . .	95
9.2.4	Improvements and Extensions to the Algorithm . . . . .	95
<b>A</b>	<b>Matlab Code</b>	<b>97</b>
A.1	Useful Scripts . . . . .	97
A.2	ION/ION'/ONA Shared Scripts . . . . .	98
A.2.1	Unaltered ION Scripts . . . . .	98
A.2.2	Updated ION Scripts . . . . .	101
A.2.3	ION'/ONA Scripts . . . . .	104
A.3	ION' . . . . .	111
A.4	ONA . . . . .	114
A.4.1	SOBI . . . . .	119
A.4.2	Spatial SOBI helper function . . . . .	121
A.4.3	Joint Diagonalization . . . . .	122



# List of Figures

2-1	Scree plots of data before (PCA) and after (ION and ONA) noise-normalization . . . . .	25
2-2	ONA Algorithm Block Diagram . . . . .	29
2-3	SOBI Block Diagram . . . . .	33
3-1	Magnitude-ordered Normalized Count and Range Concentration measures for Factory Data . . . . .	39
3-2	Normalized Count and Range Concentration measures for Factory Data, ordered by eigenvalue magnitude . . . . .	39
3-3	Range of possible focus values for (top) Uniform, (upper-middle) Gaussian, (lower-middle) mixed, and (bottom) non-Gaussian PDFs. . . . .	41
3-4	PDF plots to show focus of original 59 <sup>th</sup> factory data PC and same PC with outliers removed. . . . .	44
3-5	Visual Metric Plots of factory data sources retrieved by PCA, ION, and ONA. . . . .	45
6-1	Scree plots of noise-normalized factory data . . . . .	63
6-2	Noise Variance per Variable for Factory Data . . . . .	63
6-3	Factory eigenvectors for PCA (top), ONA with time-scrambled data (middle), ONA with time-ordered data (bottom) . . . . .	64
6-4	First 10 Eigenvectors of Sequenced Factory Data determined by ONA . . . . .	66
6-5	Eigenvectors 10, 20, ... , 100 of Sequenced Factory Data determined by ONA . . . . .	67
6-6	Time sequences of groups of correlated $\mathbf{Z}_i$ for ONA . . . . .	68

6-7	Scatter plots of consecutive retrieved factory signals $\mathbf{P}_i(t)$ and $\mathbf{P}_{i+1}(t)$ for PCA (top), ION (middle-top), ONA for time-scrambled data (middle-bottom) and ONA for time-ordered data (bottom). . . . .	70
6-8	Scatter plots of consecutive retrieved factory signals $\mathbf{P}_i(t)$ and $\mathbf{P}_{i+1}(t)$ for PCA (top), ION (middle-top), ONA for time-scrambled data (middle-bottom) and ONA for time-ordered data (bottom). . . . .	71
6-9	Time plots of retrieved factory signals $\mathbf{P}_i(t)$ for PCA (top), ONA for time-scrambled data (middle) and ONA for time-ordered data (bottom). . . . .	72
7-1	First-Ring Nearest Neighbors for one pixel . . . . .	75
7-2	First- and Second-Ring Nearest Neighbors for one Pixel . . . . .	76
7-3	Mirroring technique used to pad an image for construction of covariance matrices . . . . .	78
8-1	Example Hyperspectral Data Set from AVIRIS, Complements of JPL [2] . . . . .	82
8-2	False Color Image of the Moffett Field Data Set from AVIRIS, Complements of JPL [2] . . . . .	83
8-3	Scree plots of noise-normalized Moffett Field image data . . . . .	84
8-4	Estimated Noise Variances for 224 Spectral Channels for Moffett Field sub-image . . . . .	85
8-5	Comparison of Original (top-left), noise-filtered (top-right), and corresponding Independent Source (bottom) for Moffett Field sub-image minimum noise channel (frequency channel 153) . . . . .	86
8-6	Two examples of noise reduction in mid-range noise levels: Original and Noise-Reduced Images for Channel 109 (top) and Channel 159 (bottom) . . . . .	88
8-7	Two examples of noise reduction in highest noise levels: Original and Noise-Reduced Images for Channel 2 (top) and Channel 3 (bottom) . . . . .	89
8-8	First 10 Eigenvectors of AVIRIS Moffett Field sub-image determined by ONA . . . . .	90



8-9 Comparison of first three principal components produced using PCA  
and three Independent Sources ( $\mathbf{P}_i$ ) retrieved by ONA for Moffett Field  
sub-image . . . . . 92



# List of Tables

2.1	The ION' algorithm . . . . .	23
2.2	The Expectation-Maximization Algorithm . . . . .	26
2.3	The ONA algorithm . . . . .	28
2.4	The SOBI Algorithm . . . . .	31
4.1	Legend of scoring for Table 4.2 . . . . .	49
4.2	Performance of varied $(c, d)$ pairs at parameter retrieval . . . . .	49
4.3	Performance of second set of varied $(c, d)$ pairs at parameter retrieval . . . . .	50
5.1	Nominal Test Parameters . . . . .	55
5.2	SNR (dB) performance of ONA, ION, and SOBI on nominal Gaussian test data (see Table 5.1) . . . . .	56
5.3	SNR (dB) performance of ONA on varied Gaussian test data . . . . .	57
5.4	Parameters defining the nominal less noisy BSS experiment . . . . .	58
5.5	SNR (dB) performance of ONA on less noisy Gaussian test data . . . . .	59



# Chapter 1

## Introduction to the Problem

### 1.1 Introduction

Blind separation of signals (BSS) is a diverse family of problems generally expressible in the form:

$$\mathbf{x} = \mathbf{A}\mathbf{p} + \mathbf{G}^{1/2}\mathbf{w} \quad (1.1)$$

where  $\mathbf{x}$  is one of a given set of  $m$  observed vectors of length  $n$ ,  $\mathbf{A}$  is the mixing matrix,  $\mathbf{p}$  is a signal vector of length  $k$ , and  $\mathbf{G}^{1/2}\mathbf{w}$  is a noise vector.<sup>1</sup> Each element of the vectors  $\mathbf{p}$  and  $\mathbf{w}$  has unit variance, thus reducing ambiguities in the separation. The most general problem statement seeks estimates of  $\mathbf{A}$ ,  $k$ ,  $\mathbf{p}$ ,  $\mathbf{G}$ , and  $\mathbf{w}$  based exclusively on  $m$  observations of  $\mathbf{x}$ . Usually  $\mathbf{w}$  is Gaussian white noise and  $k \leq n$ . Despite the simplicity of Equation 1.1, the array of problems it represents is extensive and evolving.

### 1.2 Previous work done with BSS

Most methods for BSS utilize the non-Gaussian nature of the signal set  $\mathbf{p}$  to facilitate signal separation, which is increasingly necessary as the signal order  $k$  approaches  $n$ . Methods utilizing cumulant matching and contrast functions are described in [6].

---

<sup>1</sup>In this thesis, I denote matrices by boldface capital letters, vectors by boldface lower-case, and scalars by lower-case.

Independent Component Analysis [ICA] also relies on the non-Gaussian nature of the mixed signals [7]. Other methods utilize maximum likelihood methods together with prior knowledge of the non-Gaussian nature of the signal set [8], [5]. Information maximization techniques have also been successfully used but require prior knowledge of the nature of the signal set [3].

The papers most relevant to the present Gaussian-signal case include [1], [13], [14], [4], [11], and [10]. Factor analysis [1] aims to unmix Gaussian sources but assumes that the order  $k$  is known in order to calculate prior probabilities for the unobserved signals. Bayesian BSS [13] assumes that  $k$  is unknown although its probability distribution is known. The second-order joint diagonalization algorithm introduced in [4] can be applied to both Gaussian and non-Gaussian sources, provided the sources have different spectral content. This Second-Order Blind Identification algorithm (SOBI), is incorporated in the algorithm introduced in this paper, designated the Order-Noise-A (ONA) estimation algorithm, and is discussed further then. Even more central to ONA is the Iterated Order-Noise (ION) estimation algorithm, which alternately estimates the order  $k$  of the system, for example by using a scree plot ([11] and Section 2.3.2), and then the noise covariance matrix  $\mathbf{G}$ , for example by using the EM algorithm [11], [10]. The EM algorithmic step in ION also estimates the signal matrix  $\mathbf{A}\mathbf{p}$  and the noise  $\mathbf{G}\mathbf{w}$  of Equation 1.1.

### 1.3 Motivation for ION' and ONA

Both algorithms developed in this thesis, ION' and ONA, are based on the ION algorithm developed in [10]. The ION algorithm was developed to address the BSS problem where the underlying order of the system is unknown. In this case, it is difficult to accurately estimate the order  $k$  without prior knowledge of the noise covariance matrix  $\mathbf{G}$  or to estimate  $\mathbf{G}$  without knowledge of  $k$ . Clearly, an iterative algorithm can take advantage of consecutive improved estimates of both  $k$  and  $\mathbf{G}$  to further improve the estimates of both parameters. The scree plot method of estimating the order  $k$  was chosen because of its ability to locate the noise plateau,

although other order-estimation algorithms could be used instead. Finally, the EM algorithm is the most successful known separation technique for Gaussian signals, so it was used to estimate  $\mathbf{G}$  based on the previous estimate of  $k$ . Again, an alternate separation technique could perform this function, but the goal of the ION algorithm was to combine the most reliable algorithms for order and noise estimation, and therefore the EM algorithm was chosen.

The ONA algorithm specifically addresses the case where both  $\mathbf{p}$  and  $\mathbf{w}$  are Gaussian and have covariance matrices equal to the identity matrix. Although it was designed for the case where the time sequence of the  $\mathbf{x}$  vectors is irrelevant, ONA will take advantage of any information latent in the sequencing of the  $\mathbf{x}$  vectors. It is assumed without loss of generality that the noise covariance matrix  $\mathbf{G}$  is diagonal. In the case where the noise is correlated across variables, ONA will detect this correlation and will consider it signal, thereby increasing the estimated order of  $\mathbf{p}$ . The end user must then determine which elements of the retrieved  $\mathbf{p}$  correspond to true signals and which correspond to correlated noise.

These algorithms are unique because they address one of the most difficult cases for blind separation: Gaussian data of unknown order. It is significant that they also perform well on non-Gaussian or mixed data despite this initial assumption. This makes ION, ION', and ONA flexible and applicable to a diverse range of BSS problems.





# Chapter 2

## Improvements to the ION

### Algorithm: ION' and ONA

#### 2.1 Introduction

ION was designed primarily to estimate the system order,  $k$ , and the noise covariance matrix  $\mathbf{G}$  in order to perform noise normalization. As a consequence of estimating the noise  $\mathbf{N}$ , ION also estimates the signal as  $\mathbf{Z} = \mathbf{X} - \mathbf{N}$ . The success of ION in estimating these parameters is striking, and more information on the subject can be found in [11] and [10].

Little analysis was done on the convergence properties of ION, however. Also, ION does not fully characterize the parameters of the most general problem statement given in Equation 1.1. (ION estimates  $\mathbf{Z}$  but does not estimate  $\mathbf{A}$  or  $\mathbf{P}$ .) Improvements were made to the ION algorithm to address these two issues.

ION' is an extension of ION that better initializes various variables at each iteration of the algorithm and that incorporates better criteria for terminating that iteration. These changes improve the estimates of  $\mathbf{Z}$ ,  $\mathbf{G}$ , and  $k$  relative to the original ION algorithm.

ONA further extends ION' to unmix the noise-free data matrix  $\mathbf{Z}$ . At the end of the ION' process, ONA uses SOBI to separate the estimated  $\mathbf{Z}$  into estimates of the mixing matrix  $\mathbf{A}$  and the source matrix  $\mathbf{P}$ . Thus ONA improves upon ION and ION'

by fully estimating all of the parameters described in Equation 2.1.

The details of ION' and ONA were determined after a large series of experiments on simulated Gaussian data. For example, it was found that SOBI was best used but once to conclude ONA rather than at each iteration. Also, it was found that successive iterations of ION' occasionally oscillate between a few estimated values for  $k$ , leading to the scree-like method for determining which iteration produced the best results. Finally, it was found that the initialization of parameters in the EM Algorithm significantly affected the convergence time of the EM algorithm, and thus of ION' or ONA, so the updated parameters were initialized to provide convergence in fewer iterations. In general, setting the number of EM iterations ( $j$ ) and the number of ION'/ONA iterations ( $i$ ) as  $j = 10$  and  $i \leq 5$  is sufficient. The corresponding changes and additions to the algorithm are described below.

## 2.2 Problem Definition

The general problem definition in Equation 2.1 is an extension of Equation 1.1 that represents the full set of  $m$  observed  $\mathbf{x}$  vectors by  $\mathbf{X}$ , which is a matrix of  $n$  row vectors, each of which represents one observed variable:

$$\mathbf{X} = \mathbf{A}\mathbf{P} + \mathbf{G}^{1/2}\mathbf{W} = \mathbf{Z} + \mathbf{N} \quad (2.1)$$

The  $k$ -by- $m$  matrix  $\mathbf{P}$  represents the unknown unmixed signal of interest, and  $\mathbf{Z}$  and  $\mathbf{N}$  represent the total signal and noise, respectively, where we define:

$$\mathbf{Z} = \mathbf{A}\mathbf{P} \quad (2.2)$$

$$\mathbf{N} = \mathbf{G}^{1/2}\mathbf{W} \quad (2.3)$$

The problem is to estimate with minimum mean square error the unknowns  $k$ ,  $\mathbf{A}$ ,  $\mathbf{P}$ ,  $\mathbf{G}$ , and  $\mathbf{W}$  given only a single matrix  $\mathbf{X}$ . Because the optimization is non-linear, these separate estimates for any matrix  $\mathbf{X}$  may not simultaneously

Table 2.1: The ION' algorithm

Step	
1.	Optionally normalize rows of $\mathbf{X}$ to zero mean and unit variance. Initialize $\mathbf{G}_0 = \mathbf{I}$ . Set maximum iterations $i_{max}$ (typically $i_{max} = 5$ is sufficient)
2.	Noise-normalize $\mathbf{X}$ : $\mathbf{X}_n = \mathbf{G}_{i-1}^{-1/2} \mathbf{X}$
3.	Estimate signal order $k_i$ using a scree plot of $\mathbf{X}_n$ and SVD
4.	Estimate $\mathbf{G}_i$ and $\mathbf{Z}_i$ using the EM algorithm and $k_i$
5.	Check for iteration termination conditions; if none, increment the index $i$ and return to Step 2.
6.	Determine which of the $i$ iterations produced the best results for $\mathbf{G}$ , $\mathbf{Z}$ , and $k$ .

satisfy (2.1) exactly.

## 2.3 Improved Iterative-Order-Noise (ION')

### Algorithm

#### 2.3.1 Algorithm Overview

The ION' algorithm is defined in Table 2.1 in a form that returns estimates of  $k$ ,  $\mathbf{Z}$ , and  $\mathbf{G}$  for a given  $\mathbf{X}$ . It, like ION (See [11] or [10]), iterates two principal steps: 1) order estimation (of  $k$ ), and then 2) estimation of  $\mathbf{G}$  and  $\mathbf{Z}$  using the EM algorithm. These two steps are elaborated below along with details of the improvements made in each. Also unlike ION, which simply returned the estimates produced by the last iteration of the loop, ION' compares the results of each iteration to determine which produced the best results.

Step 1 of ION' initializes  $\mathbf{G}$  to the identity matrix  $\mathbf{I}$  and optionally normalizes the rows of  $\mathbf{X}$  to zero mean and unit variance for each of the  $n$  variables. The normalization improves the results of the algorithm in cases where the noise variances vary widely over the variables, e.g. when the measurements were taken in different units, and ION' subsequently improves this normalization through iteration. Step 2 is the algorithm re-entry point and normalizes each row of  $\mathbf{X}$  so that its estimated

additive noise has unit variance based on the most recent estimate of  $\mathbf{G}$ . Steps 3 through 6 describe the major modules of ION and are discussed in more detail below.

### 2.3.2 Improved Order Estimation

Step 3 utilizes a scree plot to estimate the signal order  $k$ , where such plots present the logarithm of the magnitude-ordered eigenvalues of the correlation matrix for  $\mathbf{X}_n$  as a function of eigenvector number. Figure 2-1 illustrates typical scree plots, where the plateau to the right of the break generally represents noise, and the true order of the underlying process is 24. The estimated order is the number of eigenvalues that lie above the extrapolated plateau. Multivariate data with normalized additive noise yields eigenvalues representing that noise plus signal energy. Therefore, for noise-normalized signals, the eigenvalues corresponding to pure noise have approximately equal amplitudes and form a plateau of length  $n - k$ . When the data set ( $m$ ) is limited, these pure-noise eigenvalues differ slightly and form a slope in the resulting scree plot.<sup>1</sup> Round-off errors and statistics can cause the smallest eigenvalues to drop below this slope.

Occasionally this method alone significantly over- or under-estimates  $k$ , and therefore ION' simultaneously estimates  $k$  using Singular Value Decomposition (SVD). The SVD estimate  $k'$  is simply the number of eigenvalues that individually exceed 0.001 percent of the total variance of  $\mathbf{X}$ . For the range of parameters discussed later, the algorithm is not significantly sensitive to the percent cutoff used, although the estimate produced with 0.001 percent tends to approximate both the true order and the scree estimate. When consecutive estimates of  $k$  from the scree plot and from SVD have oppositely signed slopes, the SVD estimate of  $k$  is used. Except in special cases, e.g., when the scree plot is atypical or the problem parameters become very large, the SVD value is usually not required.

---

<sup>1</sup>The plateau and  $k'$  is typically determined by performing a linear regression on the 40<sup>th</sup> to 60<sup>th</sup> percentile of the ordered eigenvalues, although which percentiles are optimally included may change for data sets with extremely large or small order  $k$ . See 4.2 and [11] for more discussion.

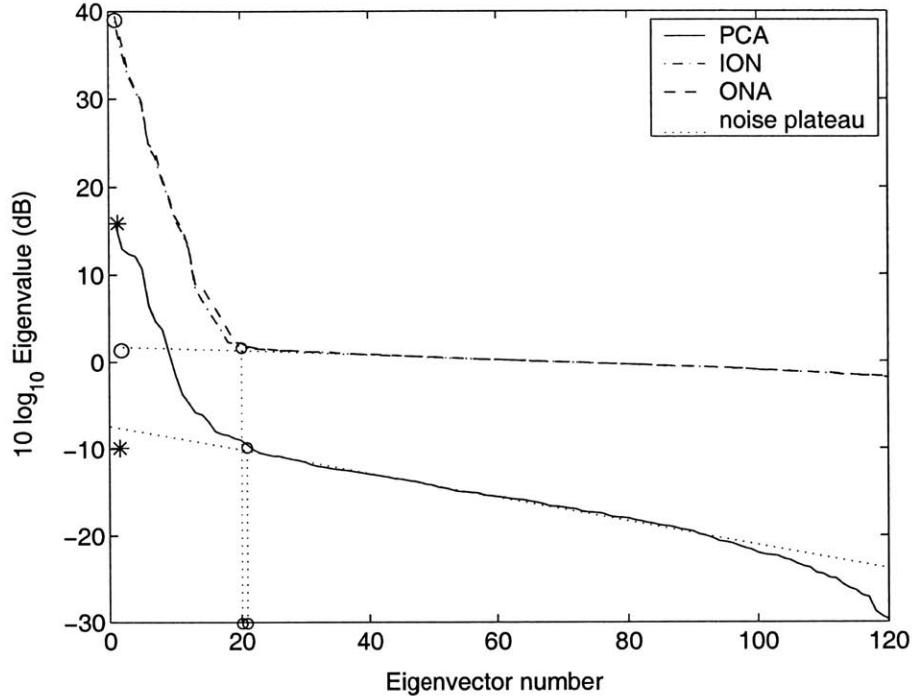


Figure 2-1: Scree plots of data before (PCA) and after (ION and ONA) noise-normalization

### 2.3.3 EM Algorithm

Step 4 of ION', as presented in Table 2.1, is the Estimation-Maximization (EM) algorithm, which is described in more detail in Table 2.2 and in [10]. The initialization step employed here is the only departure from the algorithm describe in [10]; there the initial  $\mathbf{A}'$  was the identity matrix augmented by  $n - k$  rows of zeros, or it was an  $n$ -by- $k$  matrix of unit-variance Gaussian random numbers. Here, for all iterations but the first, the initial  $\mathbf{A}'$  for each iteration is based on the latest estimate of  $\mathbf{A}$ .

The structure of the EM Algorithm is such that highly correlated columns of  $\mathbf{A}'$  often correspond to highly correlated rows of  $\mathbf{P}'$ . These highly correlated rows of  $\mathbf{P}'$  presumably correspond to a single true  $\mathbf{P}_i$  because the rows of  $\mathbf{P}$  are generally independent. In order to uncover more distinct rows of  $\mathbf{P}$ , only those  $d$  columns of  $\mathbf{A}'_{i-1}$  for which  $\sigma_{qq} > \sigma_{qr}$  for all  $r \neq q$  are therefore used to initialize  $\mathbf{A}_i$ , where  $\sigma_{qr}$  denotes the covariance of the  $q^{th}$  and  $r^{th}$  columns of  $\mathbf{A}'_{i-1}$ :

Table 2.2: The Expectation-Maximization Algorithm

Step	
1.	<p>Initialization:</p> <p>Iteration index <math>j = 1</math>; number of iterations <math>j_{max} = 10</math></p> <p><math>\mathbf{G}_1 = 0.5\mathbf{I}</math>, <math>\mathbf{I}</math> of order <math>n</math></p> <p><math>\mathbf{A}^s</math> = columns of <math>\mathbf{A}</math> for which <math>\sigma_q &gt; \sigma_r</math> for all <math>r \neq q, 1 \leq (r, q) \leq k'</math>,</p> <p><math>d</math> = number of columns in <math>\mathbf{A}^s</math></p> <p>If <math>\mathbf{A}_{i-1}</math> exists, <math>\mathbf{A}_{i,j=1} = \{ \mathbf{A}^s \mid \text{unit-variance } n \times (k_i - d) \text{ array of g.r.n.}^2 \}</math></p> <p>Otherwise, <math>\mathbf{A}_{i,j=1} = \{ \text{unit-variance } n \times k_i \text{ array of g.r.n.} \}</math></p>
2.	<p>Expectation Step:</p> <p><math>\mathbf{W}_j = \mathbf{A}_j^T \mathbf{G}_j^{-1} \mathbf{A}_j + \mathbf{I}</math></p> <p><math>\mathbf{C}_j = E[\mathbf{P}   \mathbf{X}, \mathbf{A}_j, \mathbf{G}_j] = \mathbf{X}^T \mathbf{G}_j^{-1} \mathbf{A}_j \mathbf{W}_j^{-1}</math></p> <p><math>\mathbf{D}_j = E[\mathbf{P}^T \mathbf{P}   \mathbf{X}, \mathbf{A}_j, \mathbf{G}_j] = m \mathbf{W}_j^{-1} + \mathbf{W}_j^{-1} \mathbf{A}_j^T \mathbf{G}_j^{-1} \mathbf{X} \mathbf{X}^T \mathbf{G}_j^{-1} \mathbf{A}_j \mathbf{W}_j^{-1}</math></p>
3.	<p>Maximization Step:</p> <p><math>\mathbf{A}_{j+1} = \mathbf{X} \mathbf{C}_j \mathbf{D}_j^{-1}</math></p> <p><math>G_{qq,j+1} = (\mathbf{Y}_q^T \mathbf{Y}_q - \mathbf{A}_{q,j+1} \mathbf{C}_j^T \mathbf{Y}_q) / m</math>;</p> <p>where <math>\mathbf{Y} = \mathbf{X}^T</math>; <math>q = 1, \dots, n</math> indexes the column vectors of <math>\mathbf{Y}</math> and <math>\mathbf{A}</math>;</p> <p><math>G_{qq}</math> references the diagonal elements of <math>\mathbf{G}</math>,</p>
4.	If $j < j_{max}$ , increment $j$ and go to Step 2 for further iteration.
5.	After last iteration, output $\mathbf{G}_{i+1}$ and $\mathbf{Z}_{i+1} = [\mathbf{X}^T \mathbf{G}_{j+1}^{-1} \mathbf{A}_{j+1} \mathbf{W}_{j+1}^{-1}] \mathbf{A}_{j+1}^T$

$$\sigma_{qr} = E[(A_q - E[A_q])(A_r - E[A_r])] \quad (2.4)$$

The remaining columns of the initialization matrix (up to the  $k^{th}$ ) are filled with random numbers (as in the ION algorithm). Alternatively, if the latest estimate of  $k$  has decreased (so that the number of columns of  $\mathbf{A}'_{i-1}$  is larger than the number desired), those columns with the smallest values  $\sigma_{qq}$  are dropped.

### 2.3.4 Iteration Determination

It is important to note the the ION' algorithm is one of two major components of the ONA algorithm, so improvements made to ION' also affect the performance of ONA. The algorithm developed to determine the best iteration of the ION' algorithm described here can be optimized differently depending upon whether it is being used alone or as a component of ONA. The general algorithm outline is detailed here, and optimizations specific to the ONA algorithm are discussed in Section 2.4.2.

The specific algorithm used in Step 6 of Table 2.1 to reject outliers and identify the best iteration  $i$  is similar to that used to determine  $k'$  from the scree plot but is based on the successive estimates of  $\text{Tr}\{\mathbf{G}\}$  and the corresponding estimated values of  $k$ . The set of  $\text{Tr}\{\mathbf{G}'\}$  is ordered by decreasing magnitude and plotted; the corresponding  $k$  values are re-ordered using the same transformation as on the set of  $\text{Tr}\{\mathbf{G}'\}$ . Note that this means the  $k$  values are not necessarily magnitude ordered. The steep slope on the left side of the plot typically indicates outlier iterations while the far right of the plateau typically indicates iterations that are "too good to be true," i.e. although lower values of  $\mathbf{G}$  imply that more signal has been retrieved from the noise, this is not always true. That is, those iterations for which the metric  $\text{Tr}\{\mathbf{G}'\}$  is near optimal have a statistical spread, and those values near the median (and therefore the knee) are less likely to have resulted in falsely optimistic estimates of  $\mathbf{G}$ .

Manual inspection of consecutive iterations of ION' on many different data sets led to the following heuristics for determining which iteration should be retained. Iterations are considered outliers if they have  $\text{Tr}\{\mathbf{G}_i'\} > (1 + c)\text{Tr}\{\mathbf{G}_{i+1}'\}$  (where  $i$  is the index into the ordered  $\text{Tr}\{\mathbf{G}'\}$  vector), or if they have  $k'_i > (1 + d)k'_{i+1}$  where  $c = 0.02$  and  $d = 0.15$ . (See Chapter 4 for the details of how this was determined.) Those which are "too good to be true" correspond to iterations for which  $\text{Tr}\{\mathbf{G}'\}$  decreases while  $k$  increases. An algorithm like that used to estimate  $k$  was used on the set of iterations remaining after the elimination of outliers above to determine which iteration is nearest the junction of the plateau and the steep slope on the left side; note that this algorithm produces acceptable results even when only a few iterations remain. This final valid iteration yields the best estimates of  $k$ ,  $\mathbf{G}$ , and  $\mathbf{Z}$  (and consequently of  $\mathbf{N}$  and  $\mathbf{W}$ ), except in the case when outlying iterations "accidentally" produce good results, i.e. when the algorithm converges to an answer inferior to an outlying iteration, which rarely occurs.

Stability of the performance of the algorithm with respect to the parameters  $c$  and  $d$  was tested for the same simulated data sets as described in Table 5.3. Performance is stable for variations of  $(c, d)$  by as much as a factor of two increase or decrease, though the results are more sensitive to changes in  $c$  than in  $d$ . It was found that

Table 2.3: The ONA algorithm

Step	
1.	Optionally normalize rows of $\mathbf{X}$ to zero mean and unit variance. Initialize $\mathbf{G}_0 = \mathbf{I}$ . Set maximum iterations $i_{max}$ (typically $i_{max} = 5$ is sufficient)
2.	Noise-normalize $\mathbf{X}$ : $\mathbf{X}_n = \mathbf{G}_{i-1}^{-1/2} \mathbf{X}$
3.	Estimate signal order $k_i$ using a scree plot of $\mathbf{X}_n$ and SVD
4.	Estimate $\mathbf{G}_i$ and $\mathbf{Z}_i$ using the EM algorithm and $k_i$
5.	Check for iteration termination conditions; if none, increment the index $i$ and return to Step 2.
6.	Determine which of the $i$ iterations produced the best results for $\mathbf{G}$ , $\mathbf{Z}$ , and $k$ .
7.	Using this estimate of $\mathbf{Z}$ , estimate $\mathbf{A}$ and $\mathbf{P}$ using the SOBI algorithm. Normalize $\mathbf{A}$ and $\mathbf{P}$ to guarantee that $\mathbf{P}$ has unit-variance rows.

the quality of the parameter estimates of the set  $\mathbf{G}$ ,  $\mathbf{Z}$ ,  $\mathbf{N}$  are often interrelated; for any choice of  $c$  and  $d$ , the three estimates are usually all good or all poor. The choice of  $c$  and  $d$  used here produce results which are near-optimal for all parameters, but changing the values used for  $c$  and  $d$  can change the precise optimality criterion of the algorithm. (This is discussed in more detail below where the use of ION' as a part of ONA is presented in Section 2.4.2.) Overall, the parameters for which the algorithmic estimation is most sensitive to changes in  $(c, d)$  are  $\mathbf{G}$  and the order  $k$ .

## 2.4 Order-Noise-A (ONA) Algorithm

### 2.4.1 Algorithm Overview

A block diagram of the ONA algorithm is shown in Figure 2-2, and the algorithm is defined in Table 2.3. ONA is identical to ION' (defined in Table 2.1) except for the addition of SOBI at Step 7. Like ION', ONA iterates two principal steps: 1) order estimation of  $k$  using a scree plot and SVD, and then 2) estimation of  $\mathbf{G}$  and  $\mathbf{Z}$  using the EM algorithm. After the completion of the ION' algorithm, however, ONA passes the estimate of  $\mathbf{Z}$  through the SOBI algorithm to estimate  $\mathbf{A}$  and  $\mathbf{P}$ .



X, optionally normalize rows to zero mean and unit variance

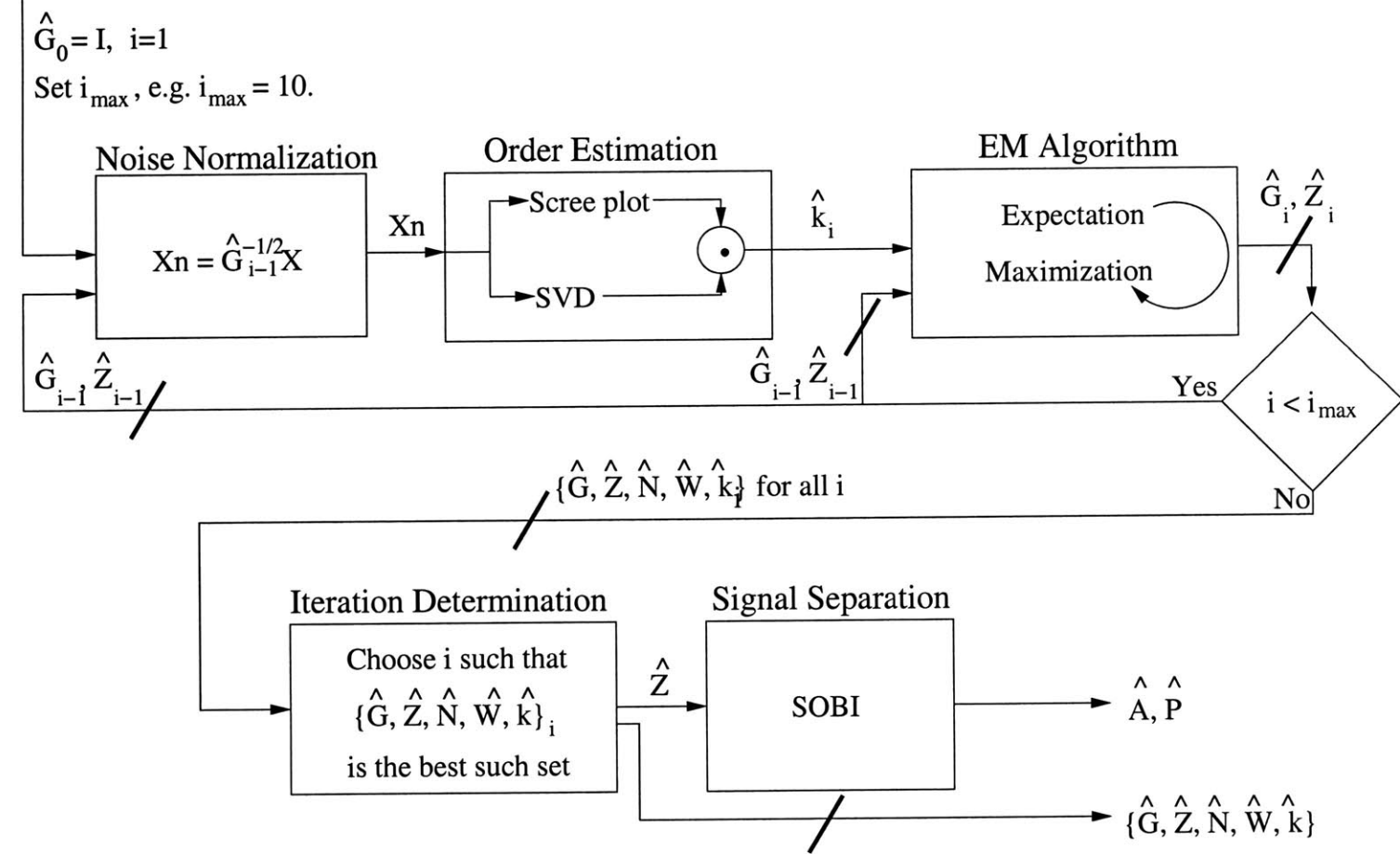


Figure 2-2: ONA Algorithm Block Diagram

## 2.4.2 Iteration Determination

As noted in Section 2.3.4, the specific optimization criterion of the algorithm which determines the best iteration can be changed by using different values for  $c$  and  $d$ . In order to use ION' as a component of ONA, additional work was done to determine the best such values of  $c$  and  $d$  with respect to the additional estimation of  $\mathbf{A}$  and  $\mathbf{P}$  done by ONA. Although  $\mathbf{A}$  and  $\mathbf{P}$  are not estimated in each iteration of ION', and thus they are not directly determined by choosing one iteration over another,  $\mathbf{A}$  and  $\mathbf{P}$  are entirely determined by which estimate of  $\mathbf{Z}$  is chosen. This estimate of  $\mathbf{Z}$  is passed to SOBI which then separates it into  $\mathbf{A}$  and  $\mathbf{P}$ , and the type of residual noise remaining in the estimate of  $\mathbf{Z}$  can affect the quality of this separation.

The optimal values of  $(c, d)$  determined for ION' as used in ONA are  $c = 0.02$  and  $d = 0.075$ . (See Section 4.4 for the details of how this was determined.) Using these parameter values, the final valid iteration yields the best estimates of  $k$ ,  $\mathbf{G}$ , and  $\mathbf{Z}$  (and consequently of  $\mathbf{N}$  and  $\mathbf{W}$ ) prior to use of SOBI in Step 7 of ONA to produce the final estimates of  $\mathbf{A}$  and  $\mathbf{P}$  from  $k'$  and  $\mathbf{Z}$ .

It was similarly found that the quality of the parameter estimates of  $\mathbf{A}$  and  $\mathbf{P}$  are often interrelated; for any choice of  $(c, d)$ , the coupled estimates are usually both good or both poor. The choice of  $(c, d)$  used here produces results which are near-optimal for all parameters, especially the order estimate and the estimates of  $\mathbf{A}$  and  $\mathbf{P}$ . Overall, the parameters for which the algorithmic estimates are most sensitive to changes in  $(c, d)$  are  $\mathbf{G}$ ,  $k$ , and  $\mathbf{A}$ .

## 2.4.3 Addition of SOBI

The most important improvement of the ONA algorithm over ION' is the addition of SOBI in Step 7 of Table 2.3 to estimate the mixing matrix  $\mathbf{A}$  and the source signals  $\mathbf{P}$ . The SOBI algorithm, as described in [4] and summarized in Table 2.4 for real signals characterized by (2.1), is a second-order blind source separation technique that estimates  $\mathbf{A}$  and  $\mathbf{P}$  by jointly diagonalizing a set of covariance matrices of the whitened signal matrix. Typically, the SOBI Algorithm assumes that the system

Table 2.4: The SOBI Algorithm

Step	
1.	<p><math>\mathbf{R}_{Z'}(0)</math> is the <math>n \times n</math> sample covariance of <math>\mathbf{Z}'</math> where we assume <math>\mathbf{Z}' = \mathbf{A}\mathbf{P} + \mathbf{N}_0</math> and <math>\mathbf{N}_0</math> is residual noise.</p> <p>Let <math>\mathbf{R}_{AP}(0)</math> be the <math>n \times n</math> sample covariance of the product <math>\mathbf{A}\mathbf{P}</math>.</p> <p>Find a whitening matrix <math>\mathbf{W}</math> such that <math>\mathbf{W}\mathbf{R}_{AP}(0)\mathbf{W}^T = \mathbf{W}\mathbf{A}\mathbf{A}^T\mathbf{W}^T = \mathbf{I}</math>.</p> <p>Because <math>(\mathbf{W}\mathbf{A})(\mathbf{W}\mathbf{A})^T = \mathbf{I}</math>, let the unitary matrix <math>\mathbf{U} = \mathbf{W}\mathbf{A}</math>.</p> <p>(<math>\mathbf{A}</math> is yet unknown.)</p> <p>(Note that finding <math>\mathbf{W}</math> requires that the covariance matrix of the noise <math>\mathbf{N}_0</math> be known or that it can be estimated. See [4])</p>
2.	<p>Calculate <math>\mathbf{\Psi} = \mathbf{W}\mathbf{Z}' = \mathbf{W}\mathbf{A}\mathbf{P} + \mathbf{W}\mathbf{N}_0 = \mathbf{U}\mathbf{P} + \mathbf{W}\mathbf{N}_0</math></p>
3.	<p>Calculate the set <math>\mathbf{R}_{\Psi}(\tau_j) \mid j = 1, 2, \dots, l</math> where <math>\mathbf{R}_{\Psi}(\tau_j)</math> are the estimated sample covariances of <math>\mathbf{\Psi}</math> for <math>l</math> fixed time lags <math>\tau_j</math>. (<math>l</math> is a variable parameter; here <math>l = 10</math>.)</p>
4.	<p>Find <math>\mathbf{U}</math> as the joint diagonalizer of the set <math>\mathbf{R}_{\Psi}(\tau_j) \mid j = 1, 2, \dots, l</math> where <math>\mathbf{R}_{\Psi}(\tau_j) = \mathbf{W}\mathbf{R}_{Z'}(\tau_j)\mathbf{W}^T = \mathbf{U}\mathbf{R}_P(\tau_j)\mathbf{U}^T</math>.</p>
5.	<p>Estimate <math>\mathbf{A}</math> and <math>\mathbf{P}</math>:</p> <p><math>\mathbf{A}' = \mathbf{W}\#\mathbf{U}</math> (where <math>\#</math> indicates the Moore-Penrose pseudo-inverse)</p> <p><math>\mathbf{P}' = (\mathbf{U}^T\mathbf{W})\mathbf{Z}'</math></p>

order  $k$  is known, so it could not be used directly on the matrix  $\mathbf{X}$ . ONA thus takes advantage of SOBI by giving it the order estimate  $k$  returned by ION' and then using SOBI to separate the noise-reduced estimate of  $\mathbf{Z}$ .

Step 1 of Table 2.4 describes how to calculate the whitening matrix which effectively noise-normalizes the data, producing the noise-normalized data in Step 2. A set of time-delayed covariance matrices of the whitened data is produced in Step 3; use of several covariance matrices instead of one allows additional information about the independent signals  $\mathbf{P}$  to be found as the independent noise on different variables is averaged out through the joint diagonalization of Step 4. Finally, the whitening and unitary matrices found in Steps 1-4 are used in Step 5 to estimate the mixing matrix  $\mathbf{A}$  and the source matrix  $\mathbf{P}$ . The block diagram of SOBI shown in Figure 2-3 also details this process.

It is important to note that, in Step 5, SOBI produces the best estimate of  $\mathbf{P}$  given  $\mathbf{A}$  and  $\mathbf{Z}$ , which is  $\mathbf{P}_{be} = \mathbf{A}\#\mathbf{Z}$  (where  $\#$  indicates the Moore-Penrose pseudo-inverse). It is the case that  $\mathbf{Z} = \mathbf{A}\mathbf{P}_{be}$  only when  $\mathbf{Z}$  and  $\mathbf{A}$  have an identical column space,

however, so any residual noise in the  $\mathbf{Z}$  matrix will produce a suboptimal estimate of  $\mathbf{P}$ . This fact makes the combined use of ION' and SOBI in ONA more effective in estimating  $\mathbf{P}$  because noise is reduced in both stages of the process.

Overall, the accuracy of  $\mathbf{P}'$  produced by using only SOBI to process  $\mathbf{X}$  is more sensitive to noise in the data and errors in the system order SOBI is given than is the accuracy of  $\mathbf{A}'$ , while the results are fairly insensitive to small changes in  $L$ , the number of covariance matrices which are jointly diagonalized (see Figure 2-3).

When SOBI is used as a component of ONA, the estimates it returns must be normalized to conform with the assumptions of Equation 2.1. That is, the magnitudes of the columns of  $\mathbf{A}$  and the rows of  $\mathbf{P}$  determined by SOBI are scaled to produce unit variance rows of  $\mathbf{P}$ , and these normalized matrices are used as the best estimates of  $\mathbf{A}$  and  $\mathbf{P}$ .

$\hat{k}$ , the estimated order of the system  
 $\hat{Z} = AP + N_o$ , where  $N_o$  is residual noise

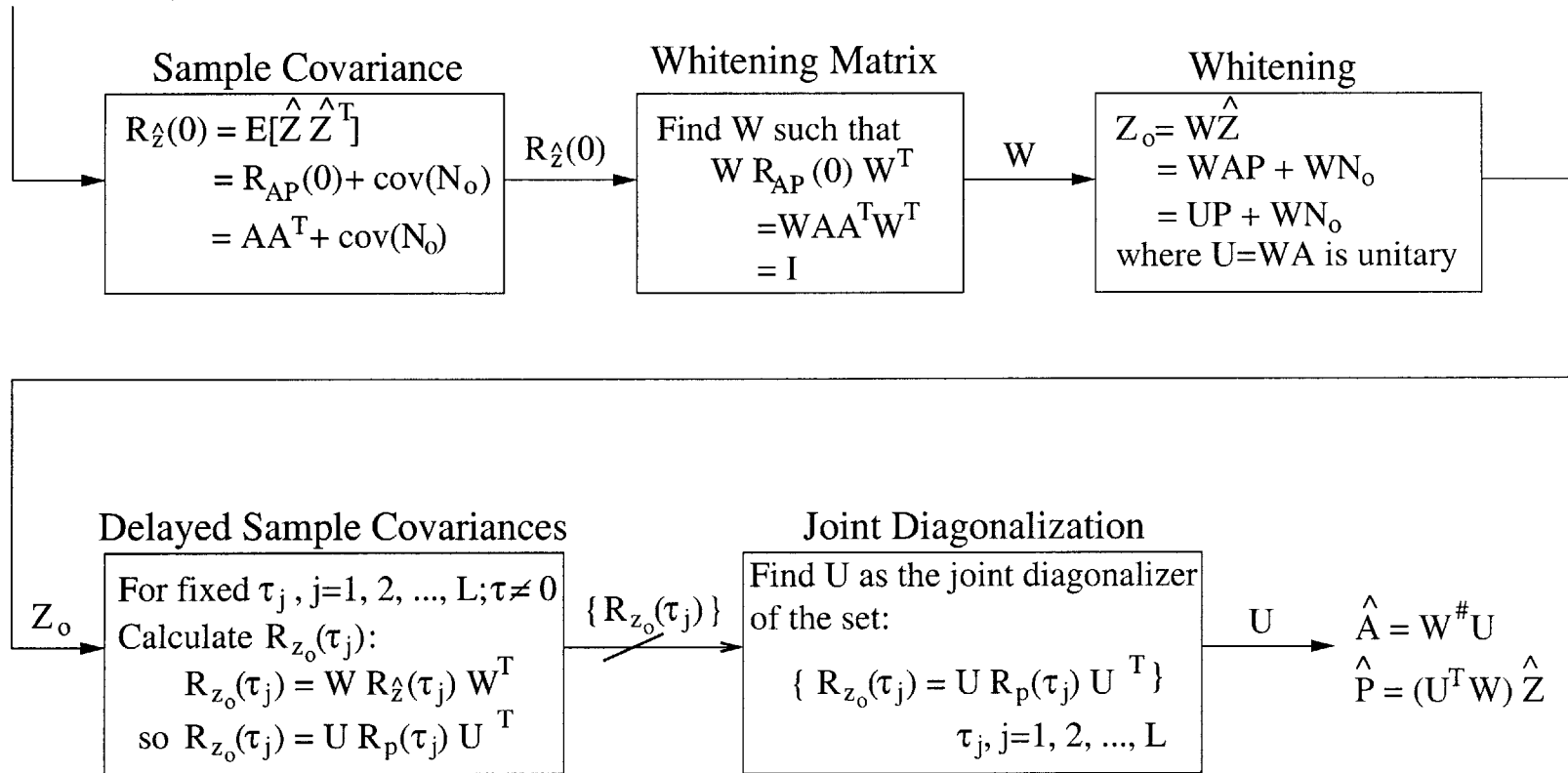


Figure 2-3: SOBI Block Diagram



# Chapter 3

## Performance Evaluation: Metrics

### 3.1 Introduction

Several of the parameters of Equation 1.1 are estimated by the ION' algorithm. The ONA algorithm is even more extensive, estimating all of the relevant matrix and scalar parameters, i.e.  $\mathbf{Z}$ ,  $\mathbf{A}$ ,  $\mathbf{P}$ ,  $\mathbf{N}$ ,  $\mathbf{G}$ ,  $\mathbf{W}$ , and  $k$ . Because the algorithms are simultaneously estimating such a diverse set of parameters, the performance can be quantified in many ways. Although not all of the metrics detailed in this chapter were ultimately used to evaluate the algorithms, their successes and failures lend insight into the nature of the problem being addressed.

### 3.2 Signal-to-Noise Ratio (SNR)

The most useful metric for evaluating the performance of the algorithm on simulated data is the signal-to-noise ratio for each of the estimated parameters. This measure gives some sense of the ability of the algorithm to detect signal buried in noise and to separate the two. Here we extend the definition of signal-to-reconstruction-noise ratio (SNR) for a matrix estimate  $\mathbf{Y}'$  of  $\mathbf{Y}$  as:

$$SNR(dB) = 10 \log_{10} \left( \frac{Tr\{S_S\}}{Tr\{S_N\}} \right) \quad 3.1$$

where

$$S_S = E[\mathbf{Y}\mathbf{Y}^T] \quad 3.2$$

$$S_N = E[(\mathbf{Y} - \mathbf{Y}')(\mathbf{Y} - \mathbf{Y}')^T] \quad 3.3$$

Also, we define the SNR of  $\mathbf{X}$  ( $SNR_X$ ) to be the ratio of the signal energy to the noise energy:

$$SNR_X = \frac{\sum_{i=1}^n \sum_{j=1}^m \mathbf{Z}_{i,j}^2}{\sum_{i=1}^n \sum_{j=1}^m \mathbf{N}_{i,j}^2} \quad 3.4$$

Because the system is noisy, it is not expected that  $SNR_{X'}$  will meet or exceed  $SNR_X$ . It is clear, however, that the algorithm is performing well when  $SNR_{X'}$  approaches  $SNR_X$ . Also, for this definition of  $SNR_{X'}$ , it is possible to calculate SNR for the ION' or ONA estimates of non-simulated data for which nothing is known a priori about the  $SNR_X$ .

### 3.3 Order Estimation Metrics

Unfortunately, it is impractical to evaluate the estimate of  $k$  using the SNR. Because  $k$  is often estimated precisely, the noise variance can be zero, making the SNR infinite. An alternate set of stable metrics is used instead.

For a single experiment, the quality of estimation of the system order is described by the percent error of the estimate and the root-mean-square error of the estimate. For a set of identical simulated test cases, the overall performance of the estimate of  $k$  is described as follows:

$$m_k \triangleq E\left[\frac{k - k'}{k}\right] \quad 3.5$$

$$\sigma_k \triangleq E[(k - k')^2]^{\frac{1}{2}} \quad 3.6$$

Together these two metrics describe both the absolute error in the estimate of  $k$  and the amount by which this estimate varies. Both metrics are stable and are used to quantify the performance of ION' and ONA on all simulated data sets.



### 3.4 Concentration: Variable Grouping

Concentration is a measure of how many variables are grouped together by a single eigenvector of the covariance matrix of the data, where fewer variables per group is usually considered to be better. If we let  $\mathbf{V}$  be the matrix whose columns are the eigenvectors of  $cov(\mathbf{G}^{-1/2}\mathbf{X})$ , then in general, the strength of the contribution of the  $i^{th}$  variable of a system to the  $j^{th}$  eigenvector is proportional to  $|\mathbf{V}_j(i) - mean(\mathbf{V}_j)|$ . (Note that a system of  $N$  variables will typically have  $N$  such eigenvectors of dimension  $N$ .) If  $\mathbf{V}_j$  has a small number of elements with magnitude significantly differing from the mean value, these elements likely distinguish a set of related variables. Concentration is developed as a measure of how well this grouping of variables into sets is accomplished. The concentration measure of all  $N$  eigenvectors can then be sequentially plotted to determine the number of significant variable groups found in the data. The concentration of an eigenvector can be determined with either of two related measures.

First, concentration can be related to a count of the number of elements in a single eigenvector whose magnitude significantly differs from the mean, although this method requires an arbitrary threshold for 'significance.' One simple way is to determine the number of elements which differ from the mean by more than *sigmacount* standard deviations (where *sigmacount* is arbitrarily chosen, e.g. *sigmacount* = 4). Given an eigenvector  $\mathbf{v}$  of order  $N$  with mean  $\mu$  and standard deviation  $\sigma$ , we define  $C_{count}$  as:

$$C_{count} = count(|\mathbf{v} - \mu| > (sigmacount)\sigma) \quad (3.7)$$

and it is the case that  $0 \leq C_{count} \leq N$ . This arbitrary choice of *sigmacount* here is not ideal, however; an alternate definition is proposed which yields comparable results without the need for an arbitrary threshold.

The second definition of concentration is a normalized measure of the range of values taken on by the elements of any particular eigenvector  $\mathbf{v}$ . If the Range of a vector is defined as:

$$R(\mathbf{v}) = max(\mathbf{v}) - min(\mathbf{v}) \quad (3.8)$$

then the concentration of the same vector can be defined:

$$C_{range} = \frac{R^2(\mathbf{v})}{\sigma^2} \quad (3.9)$$

where  $C_{range}$  can take on any non-negative real number. Although not directly related to  $C_{count}$ ,  $C_{range}$  reveals much of the same information without the use of an ad hoc threshold.

An example of sequentially plotted Concentration measures is shown in Figure 3-1 for the 577-variable factory data set discussed in Chapter 6. Both measures of Concentration are plotted with the curves normalized so that their maximums are unity. The shapes of the two curves are similar, showing that just much of the same information can be extracted from  $C_{range}$  as from  $C_{count}$ ; indeed, the discretized-curve of  $C_{count}$  obscures information about the variance of the magnitude of the elements exceeding the threshold. In both cases, it is clear that a fraction of the eigenvectors do much better at grouping a small set of variables; these are apparent in the steep slope at the left, especially for  $C_{range}$ .

Unfortunately, the magnitude of  $C(\mathbf{V}_i)$  is not correlated with  $|\lambda_i|$ . As can be seen in Figure 3-2, some eigenvectors corresponding to the smallest eigenvalues have a very large concentration measure. While these eigenvectors almost surely pick out small sets of correlated variables, it is unlikely that they are important groups because there is so little power in the corresponding eigenvalue. When they are significant groups, it is most often the case that these variable groups were also grouped by an earlier eigenvector, making the grouping redundant.

### 3.5 Evaluation of Estimated Source Quality

The ONA algorithm employs SOBI to separate the mixed source matrix  $\mathbf{Z}$  into the mixing matrix  $\mathbf{A}$  and the unmixed source matrix  $\mathbf{P}$ . Some metric is necessary, however, to evaluate the quality of these unmixed sources, both to determine the quality of the unmixing as compared to other algorithms and also to allow the end user to

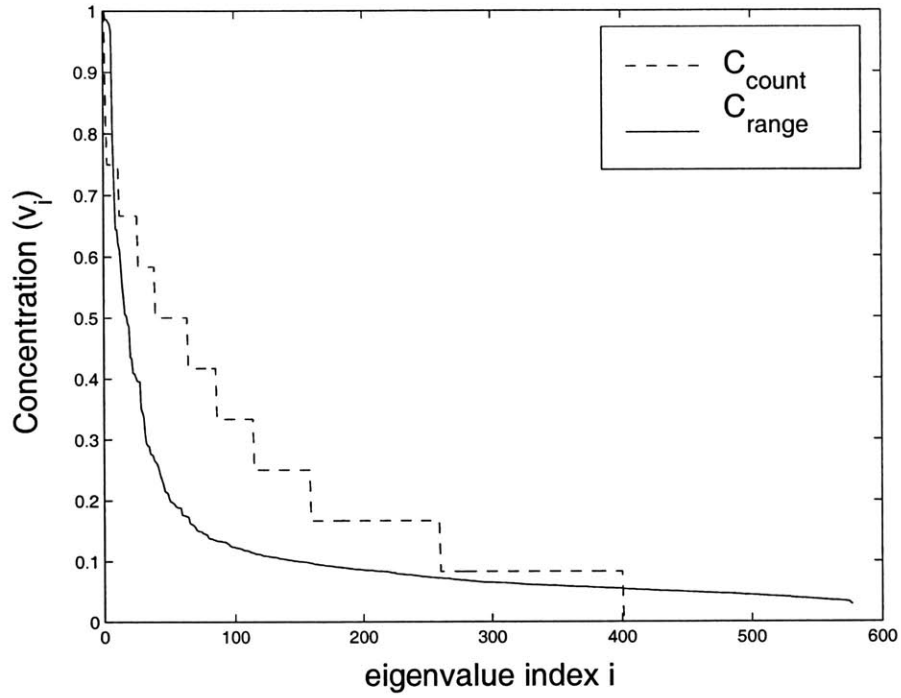


Figure 3-1: Magnitude-ordered Normalized Count and Range Concentration measures for Factory Data

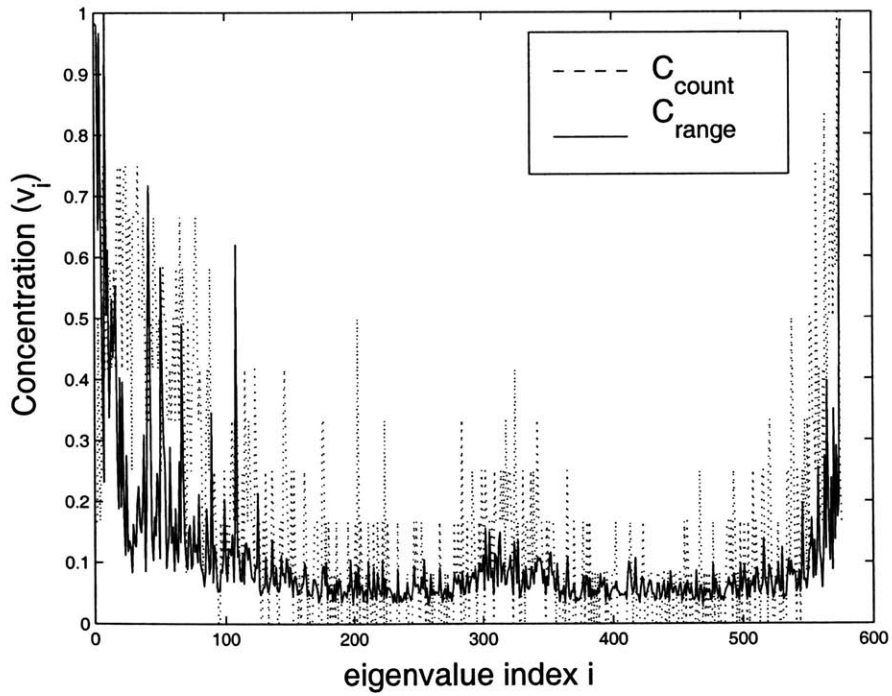


Figure 3-2: Normalized Count and Range Concentration measures for Factory Data, ordered by eigenvalue magnitude

separate significant source vectors from those primarily composed of correlated noise.<sup>1</sup>

The metrics discussed below quantify in some manner the quality of a source signal  $\mathbf{P}_i$  or of the set of source signals  $\mathbf{P}$ . None are used in the review of the algorithm in this thesis, but their strengths and weaknesses are detailed here for the edification of the reader.

### 3.5.1 Focus

The focus of a source signal  $\mathbf{P}_i$  is a general measure of the “sharpness” of the signal, i.e. the extent to which the probability distribution function (pdf) of  $\mathbf{P}_i$ , call it  $p_i(x)$ , is clustered around a few values. For the definition of focus provided below, a source with uniform pdf has a focus value of zero, a source with Gaussian probability has a very low focus value ( $\sim 0.1 - 0.3$ ), and a source with a pdf which consists only of a few delta functions will have a very high focus value ( $\sim 0.8 - 1.0$ ). Figure 3-3 illustrates the notional relation of focus to probability distribution. Here all calculations have been done with  $K = m$  for sources of dimension  $m=1001$ . The uniform distribution has range  $[-2500, 2500]$  and variance  $2 * 10^6$ . The Gaussian distribution has an approximate range of  $[-2800, 2600]$  and variance  $0.9 * 10^6$ . The mixed Gaussian has peaks with variance 100 centered at -2400, 0, and 2400. The range is  $[-2700, 2700]$  with overall variance of  $2.89 * 10^6$ . Finally, the non-Gaussian signal has range  $[-2400, 2400]$  with variance  $2.88 * 10^6$ . These signals all have similar ranges and variances, but the focus values produced differ drastically. While the variance measure describes the spread of the distribution, the focus measure is intended to capture information about the nature of spread of the distribution.

Specifically, focus is defined here as a measure of the entropy of  $p_i(x)$  relative to that of a uniform distribution over the range  $(\min(\mathbf{P}_i), \max(\mathbf{P}_i))$ . We define:

$$F(\mathbf{P}_i) = \frac{H_{unif} - H_{source}}{H_{unif}} \quad 3.10$$

---

<sup>1</sup>It is significant to note that if the noise is correlated with the data, ONA will consider it 'signal', and the estimated order  $k$  will increase. See Section 1.3 for more discussion.

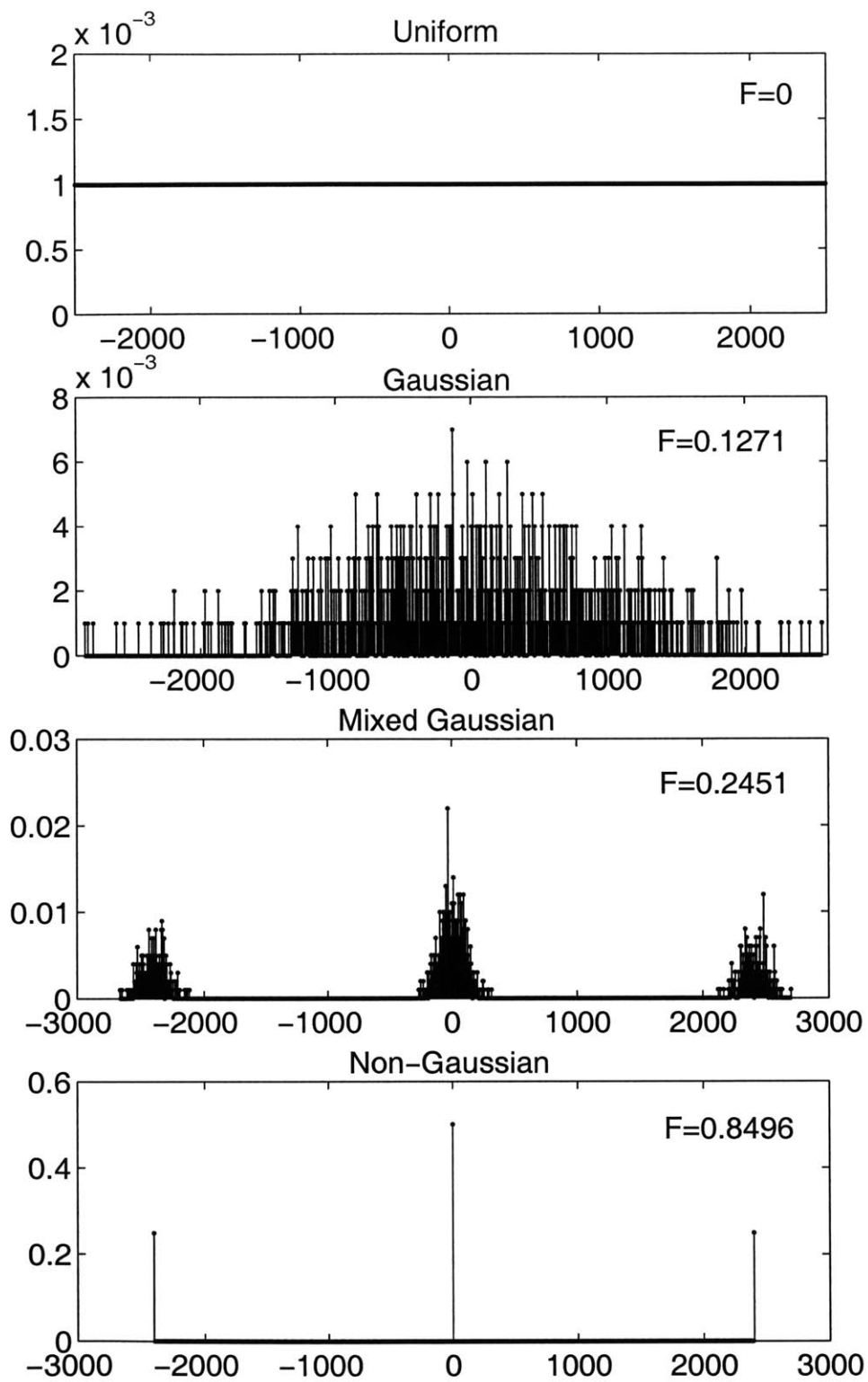


Figure 3-3: Range of possible focus values for (top) Uniform, (upper-middle) Gaussian, (lower-middle) mixed, and (bottom) non-Gaussian PDFs.

where the entropy of the source pdf is

$$H_{source} = \sum_{k=min(\mathbf{P}_i)}^{max(\mathbf{P}_i)} -p_i(k)\log(p_i(k)) \quad 3.11$$

and the entropy of the uniform pdf over the same range is

$$H_{unif} = \sum_{k=min(\mathbf{P}_i)}^{max(\mathbf{P}_i)} \frac{\log(max(\mathbf{P}_i) - min(\mathbf{P}_i))}{(max(\mathbf{P}_i) - min(\mathbf{P}_i))} \quad 3.12$$

where the summations are over  $K$  intervals within the given ranges. For  $K$  large enough, all interesting features of the pdf should be fully resolved, making  $F(\mathbf{P}_i)$  nearly independent of  $K$ . As with the concentration metric, the set  $\{F(\mathbf{P}_i)\}$  can be plotted in magnitude order to inspect the spread of well-focused to unfocused sources. Such plots can also be compared for competing algorithms to distinguish which produces more sharply-focused sources.

Unfortunately, extreme cases behave unexpectedly with this metric. Principal Components Analysis (PCA) has a tendency to align nearly Gaussian sources such that each also has a few extreme outliers. When this occurs and the outlier is far from the Gaussian center, the focus value for the source is greatly increased. This phenomenon is demonstrated in Figure 3-4. The pdf of the 59<sup>th</sup> principal component of the non-Gaussian factory data analyzed in Chapter 6 is shown on the top, calculated with interval sizes of 0.0998 ( $K = m/10$  where  $m$  is the number of samples in the 59<sup>th</sup> PC), with corresponding focus value of 0.4666. While a focus value near 0.5 does not indicate a source with great focus, it typically indicates at least that the source is non-Gaussian. Inspection of the pdf shows, however, that this source is essentially Gaussian except for the few extreme outliers at 15.98 and 68.48. When the two outliers are removed, and the pdf for this new (Gaussian) source is plotted below, calculated with interval sizes of 0.0932 ( $K = m/40$ ), the corresponding focus value decreases to 0.3146, which is near the high end of focus values indicating a Gaussian source.

In this example, the interval size was held nearly constant to preserve the shape of

the density function. If the number of intervals is instead held constant at  $K = m/10$ , the focus value for the outlier-removed source drops further to 0.2519 which is well within the Gaussian range. In the extreme, when  $K = m$ , the focus values become  $F = 0.3528$  for the unaltered PC and  $F = 0.2056$  for the outlier-removed source. Even here, the original focus value was above the Gaussian range though visual inspection of the pdf indicates that the source is nearly Gaussian. While outlier-removal reveals the Gaussian nature of the signal, it is not clear that this process should be automated; outlier detection and removal could quickly become ad hoc and may adversely affect the focus values obtained on non-Gaussian sources. It is also important to note that, in this case, the value chosen for  $K$  does affect the focus values, and this should be kept in mind when comparing the results of different algorithms.

Because the focus value of these undesirable retrieved signals can actually be increased above the typical Gaussian range of 0.1-0.3, it is difficult to use source focus values to identify Gaussian sources with outliers from well-focused non-Gaussian sources. For this reason, the focus metric is not used in the evaluation of ONA.

### 3.5.2 Visual Metric

A visual metric was also developed which helps to uncover patterns in the recovered signals. Essentially the signals are transformed into a new set of signals  $\mathbf{Q}$  by:

$$\mathbf{Q}_j(i) = \begin{cases} 1 & \text{if } |\mathbf{P}_j(i) - \mu_j| > mult * \sigma_j \\ 0 & \text{otherwise} \end{cases}$$

where *mult* is a constant chosen by the user.  $\mathbf{Q}$  is then plotted on a two-dimensional plot where black represents a 1 and white represents a 0. Clearly non-Gaussian  $\mathbf{P}_j$  will create 'streaks' of black in  $\mathbf{Q}_j$  as  $\mathbf{Q}_j(i) = 1$  for several consecutive  $i$ . Purely Gaussian signals have fewer extreme values overall, reducing the number of dots, and thus streaks, in the plot of  $\mathbf{Q}$ .

An example of this type of visual plot is shown in Figure 3-5 where a subset of the sources retrieved by PCA, ION, and ONA have been clipped at  $3\sigma_j$ . The

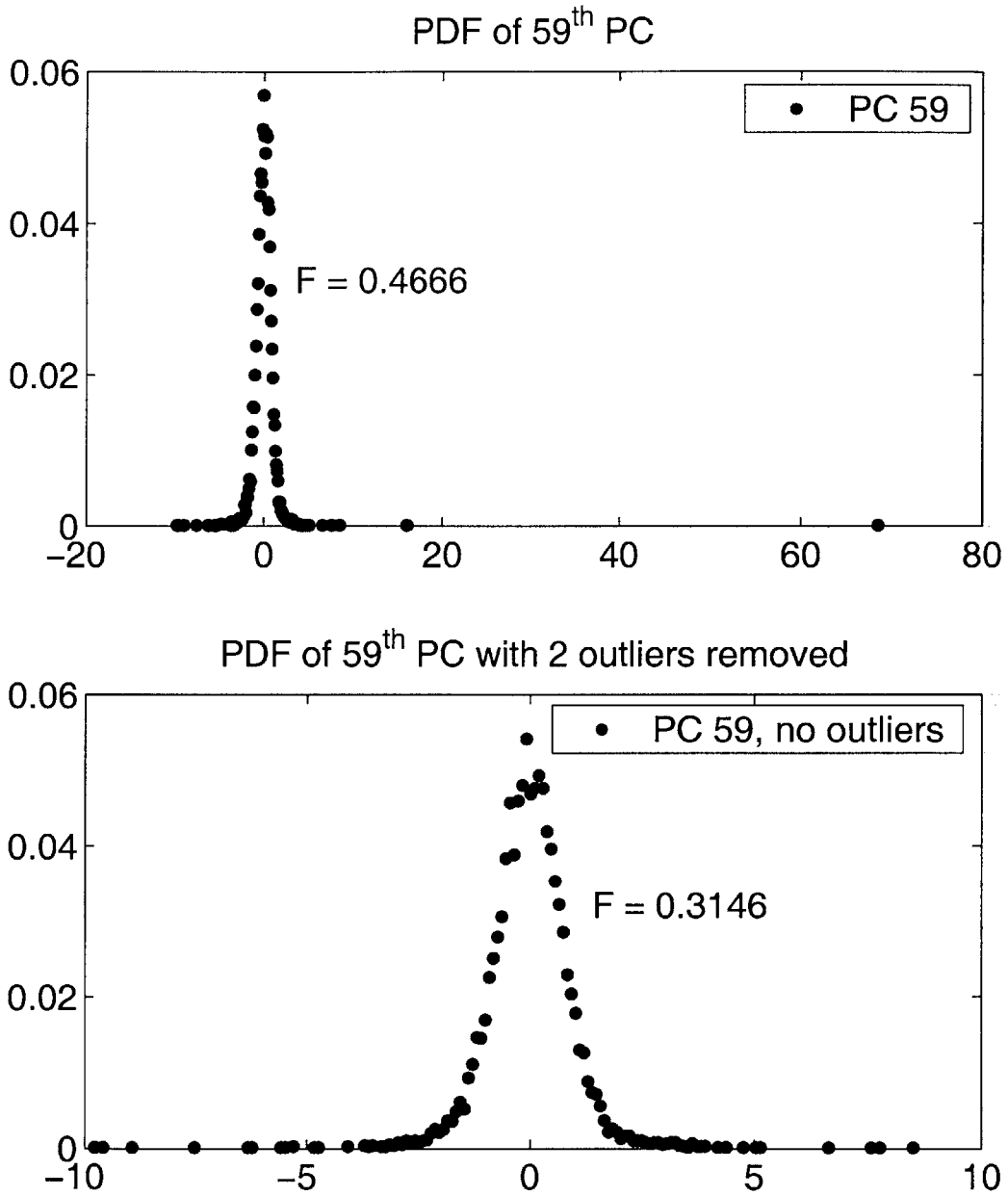


Figure 3-4: PDF plots to show focus of original 59<sup>th</sup> factory data PC and same PC with outliers removed.



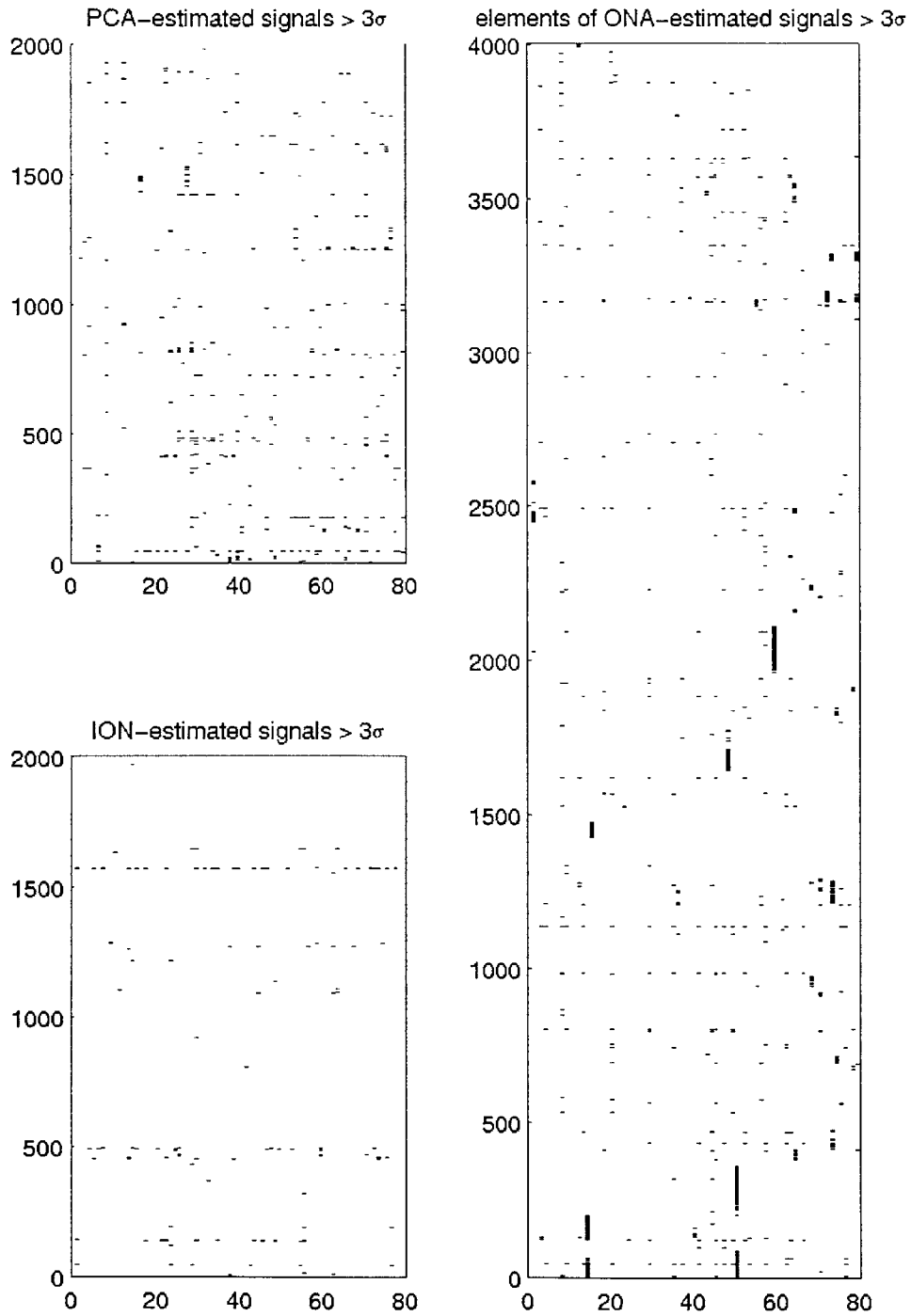


Figure 3-5: Visual Metric Plots of factory data sources retrieved by PCA, ION, and ONA.

corresponding set  $\mathbf{Q}$  is displayed as a set of column vectors. On the left, 2000 samples of the transformed PCA and ION sources  $\mathbf{Q}$  are shown, while on the right, 4000 samples of the transformed ONA sources are shown. It is immediately recognizable the the ONA plot shows significantly more vertical 'streaks' than do the other plots, demonstrating the power of the correlation assumptions made in the SOBI algorithm to unmix correlated sources. Although this type of plot gives a good indication of the type of sources retrieved, quantitative analysis of the plot quickly becomes more ad hoc than is desired in a comparative metric; the visual metric is thus used only as a qualitative measure of algorithm success.

# Chapter 4

## Algorithm Optimization

### 4.1 Introduction

The ION' and ONA algorithms have two sub-functions which rely upon empirically-set parameters. A discussion of the criterion for optimality and the approach used to determine optimal values for these parameters is included in this chapter. Section 4.2 discusses the parameters used for order estimation from a scree plot. Section 4.3 explores the use of the trace of  $\mathbf{G}$  as a metric for iteration selection. Section 4.4 explores the use of the trace of  $\mathbf{G}$  as well as determining the other parameters of the iteration selection algorithm. These parameters are optimized for the data used to evaluate the algorithm in Section 5.3.2, specifically those data sets reviewed in Table 5.3. The limitations of this training set are discussed further below.

### 4.2 Order Estimation Parameters

As documented in [11] and [10], the scree algorithm estimates the order of a data set by estimating the number of eigenvalues that rise above the noise plateau of the data. This requires estimation of the level of the noise plateau, done by the 'screeorder function' (see A.2) authored by Junehee Lee [10]. The plot analysis is controlled by the parameters  $(a, b)$  such that regression is performed on the  $100 * a^{th}$  to  $100 * b^{th}$  percentile of the eigenvalues to determine the estimated noise plateau. Typically,

$(a, b) = (0.4, 0.6)$ , the values used for ONA, but the optimal values can vary for different types of data. For further discussion, see [10].

### 4.3 Trace of $\mathbf{G}$

Recall that the noise on each channel is assumed independent, so the noise covariance matrix  $\mathbf{G}$  is considered diagonal and real. Because all of the noise variances lie on the diagonal, the trace of  $\mathbf{G}$ , denoted by  $\text{Tr}\{\mathbf{G}\}$ , is a rough measure of the total noise in the system. This means that the traces of  $\mathbf{G}'$  from consecutive iterations can be compared to determine whether the algorithm is increasing or decreasing its estimate of noise in the system. While this fact is straight-forward, how to use this information is not necessarily so. In general, the estimates of noise in the system are expected to increase for at least the first few iterations of ONA as the algorithm detects noise which is present. If, however, the noise estimates continue to increase drastically, it may be a sign that the algorithm has incorrectly classified a large portion of signal as noise. Careful interpretation is thus required to use this metric as an indicator of the quality of the results for a given iteration. The following section details how  $\text{Tr}\{\mathbf{G}\}$  is used in conjunction with other indicators to aid iteration selection.

### 4.4 Iteration Selection Parameters

The algorithm used to select the best iteration produced by ION' was briefly introduced in Sections 2.3.4 and 2.4.2. The Matlab code for this function can be found in Appendix A.4 and can be altered and optimized for different values of  $(c, d)$ . These two constants determine what relative changes in the order estimation and the noise estimation between two iterations are considered indicators that the iteration results are unreliable. As was discussed earlier, the choice of  $(c, d)$  can be used to optimize this selection process for a specific parameter, e.g.  $\mathbf{G}$ ; however it was desired that a set  $(c, d)$  be chosen which would optimize the results over the entire set of parameters. It is important to realize that this includes  $\mathbf{G}$ ,  $\mathbf{W}$ ,  $\mathbf{N}$ ,  $\mathbf{Z}$ , and  $k$ . It also includes the

Table 4.1: Legend of scoring for Table 4.2

(++)	good retrieval with high consistency
(+)	good retrieval with acceptable consistency
(/)	neutral or inconsistent retrieval
(-)	poor retrieval with some consistency
(- -)	poor retrieval with high consistency

Table 4.2: Performance of varied  $(c, d)$  pairs at parameter retrieval

$(c, d)$	<b>Z</b>	<b>N</b>	<b>G</b>	<b>A</b>	<b>P</b>	<b>k</b>	<b>W</b>
(0.04, 0.20)	+	+	+	/	/	--	++
(0.08, 0.25)	/	/	/	-	-	--	+
(0.02, 0.25)	+	+	+	-	-	--	++
(0.08, 0.15)	/	/	/	/	/	--	+
(0.02, 0.15)	++	++	++	/	/	++	++
(0.08, 0.10)	-	-	-	-	-	--	++
(0.02, 0.10)	+	+	+	/	/	++	++

**A** and **P** which will be unmixed from **Z**; for two **Z** with the same levels of noise, the specific contrasting noise structures may alter the ability of SOBI to unmix the data.

Many tests were run with a range of values for  $(c, d)$  to determine those values which identify the iteration that produced the best *set* of data. Table 4.2 shows generally how each pair  $(c, d)$  performed for each of the parameters estimated, with the best results highlighted. The scale used is presented in Table 4.1 as follows:

The labels were found by running the algorithm on the simulated tests of Table 5.3 where a pair  $(c, d)$  receives a (+) for performing within 10% of the best results as compared to the other  $(c, d)$  pairs tested for at least three of the six experiments and a (++) for doing so for more than four. A (-) indicates that the results using the pair  $(c, d)$  often varied more than 20% from the best value while a (- -) indicates that this performance is consistent over most of the experiments.

Recalling that ION' does not predict **A** or **P**, the contents of this table clearly indicated that the best parameters to be used in ION' were  $(c, d) = (0.02, 0.15)$ . None of the pairs tested showed promising results for **A** or **P**. A second set of tests was run with promising pairs from Table 4.2 and with some different pairs  $(c, d)$ ; the results

Table 4.3: Performance of second set of varied  $(c, d)$  pairs at parameter retrieval

$(c, d)$	<b>Z</b>	<b>N</b>	<b>G</b>	<b>A</b>	<b>P</b>	<b>k</b>	<b>W</b>
(0.04, 0.20)	++	++	++	-	-	--	++
(0.02, 0.25)	++	++	++	-	-	--	++
(0.02, 0.15)	++	++	++	/	/	++	++
(0.02, 0.125)	++	++	++	+	+	--	++
(0.02, 0.10)	++	++	++	/	/	++	++
(0.02, 0.075)	++	++	++	++	++	++	++

are presented in Table 4.3 with the best results again highlighted. Note that pairs which appear in both tables may have different scores since the scoring is relative to the other pairs  $(c, d)$  being tested.

It is clear from these results that the parameters best suited to estimation of all parameters well is the pair  $(c, d) = (0.02, 0.075)$ . These are the values selected for use in the ONA algorithm; while the value for  $c$  is the same as that chosen for ION', the value of  $d$  chosen here produces better results for **A** and **P** and thus gives a better complete set of parameter estimations for the context of ONA.

It is interesting to note that almost all choices of  $(c, d)$  do an equally good job of predicting **W**. Also, the predictions for the set  $\{\mathbf{G}, \mathbf{Z}, \mathbf{N}\}$  generally vary together, as do those for the set  $\{\mathbf{A}, \mathbf{P}\}$ . Finally, it is clear that altering these parameters can change the optimality criterion used by ONA in the iteration determination algorithm. This can be useful for applications in which only one or a few of the parameters are important. Finally, these adjustable parameters leave room for specialization of the algorithm to different types of data sets. Though the values chosen here worked well for the data sets analyzed in this thesis, it is probable that the pair  $(c, d)$  will have to be tweaked for data sets which differ significantly from those presented in the following chapters.

# Chapter 5

## Algorithm Evaluation: Exploring the Problem Space

### 5.1 Introduction

To compare ONA to PCA, SOBI, and ION, their performances on a suite of simulated test matrices  $\mathbf{X}$  were evaluated for various distributions of singular values for  $\mathbf{A}$  and  $\mathbf{G}$ , and for various values of  $n$ ,  $m$ , and  $k$ . Because ION and ONA were developed specifically to work on Gaussian data, the simulated data sets are Gaussian in nature and span the space of such problems. The tests evaluate the limits of the algorithms within the problem space, and the suite of tests described in this chapter demonstrate the algorithm strengths and limits.

It is important to note that, because the mixing matrix  $\mathbf{A}$  can only be determined to within some permutation of its columns, the column sequence of  $\mathbf{A}'$  must be unscrambled to maximize the correlation between  $\mathbf{A}$  and  $\mathbf{A}'$  prior to computing the error metric. This same unscrambling transformation is used to unscramble the row sequence of  $\mathbf{P}'$ .

## 5.2 Degrees of Freedom in the Problem Space

Recall the problem definition of Equation (2.1):

$$\mathbf{X} = \mathbf{A}\mathbf{P} + \mathbf{G}^{1/2}\mathbf{W} \quad (5.1)$$

The problem space of data sets of this form is diverse and extensive. It can be roughly characterized by five degrees of freedom.

### 5.2.1 Number of Variables ( $n$ )

The number of variables,  $n$ , is defined as the number of rows of  $\mathbf{X}$ , as written above. While it is usually the case that data sets with more variables are thought to be more difficult to unmix, the ratio of latent sources to the number of variables,  $\frac{k}{n}$ , is generally expected to have a greater impact on the difficulty of the problem, upheld both by discussion in source separation literature and results obtained here. For a given  $\frac{k}{n}$ , increasing  $n$  will increase the difficulty of the problem, but this is due more to the increase in  $k$  than the increase in  $n$ .

### 5.2.2 Fraction of Latent Sources ( $\frac{k}{n}$ )

The number of latent sources,  $k$ , is also referred to as the order of the system and appears in Equation 5.1 as the hidden dimension of  $\mathbf{A}$  and  $\mathbf{P}$ . In general, increasing the system order for a given  $n$  creates a more difficult problem; effectively, this increases the ratio  $\frac{k}{n}$ , showing again that part of the difficulty of a problem lies in the ratio of hidden sources to observed variables. It is also generally assumed here that  $\frac{k}{n} \leq \sim 0.3$ ; for Gaussian data, it is impractical to attempt to unmix data sets with a ratio higher than 0.3 due to the unstructured nature of the data.

### 5.2.3 Number of Samples ( $m$ )

The number of samples,  $m$ , is the number of columns of  $\mathbf{X}$  and of  $\mathbf{P}$ . As was true for the number of variables, it is often more useful to consider the ratio of the number



of samples to the number of latent variables, i.e.  $\frac{m}{k}$ , as this roughly indicates the proportional amount of redundancy data available per hidden source. Although data tends to be easier to unmix for larger  $m$ , the decrease in difficulty is more correlated with increased  $\frac{m}{k}$ .

#### 5.2.4 Matrix Signal-to-Noise Ratio ( $SNR_{\mathbf{X}}$ )

The matrix signal-to-noise ratio was defined in Equation 3.4 in Section 3.2 to be the ratio of the energy of the signal matrix  $\mathbf{Z}$  to the energy of the noise matrix  $\mathbf{N}$ . This parameter captures the extent to which the data is noisy. As such, a high  $SNR_{\mathbf{X}}$ , indicating little noise is present, corresponds to data which is more easily separated than data with a low  $SNR_{\mathbf{X}}$ .

#### 5.2.5 Distribution of Singular Values of the Mixing Matrix ( $\Delta\lambda_{\mathbf{A}}$ )

Finally, the structure of the mixing matrix  $\mathbf{A}$  affects the nature of the data set. This structure is the most difficult of any degree of freedom in the problem to quantify, so assumptions must be made in order to determine how to create adequate test data. First, it is assumed that  $\mathbf{A}$  has full column rank, although it is not assumed that the columns of  $\mathbf{A}$  are orthogonal. The non-orthogonality assumption describes a more general set of problems for which Principal Components Analysis (PCA) is no longer sufficient, allowing ION' and ONA to successfully unmix data sets which PCA cannot. That  $\mathbf{A}$  is full column rank guarantees that the unmixing process will not have singularity problems and allows for a unique mixing matrix to be found. The distribution of these  $k$  singular values could be anything that leaves  $\mathbf{A}$  with full column rank; this is a huge problem space. In order to simplify the exploration of the problem space defined by Equation 2.1 and reiterated in Equation 5.1, the singular values are assumed to be roughly evenly spaced with some  $\Delta\lambda_{\mathbf{A}}$ . The spacing of the singular values will not be precisely even, due to the random element present in construction of the mixing matrix, but the average spacing over many tests approximates the

chosen parameter. Although the entire range of possible  $\Delta\lambda_{\mathbf{A}}$  is not represented in the test suites, it is varied over a few values to explore the behavior of the algorithm for different structures of the mixing matrix.

## 5.3 Method for Exploring the Problem Space: Simulated Gaussian Tests

Preliminary tests were performed to determine acceptable nominal values for each of the five degrees of freedom described above. The algorithm performed adequately (but not necessarily optimally) for these nominal values. The problem space was then mapped by varying one parameter at a time; each parameter variation was chosen to create a data set which was more challenging than nominal. This was done for each of the five parameters characterizing the nominal Gaussian test data.

It was verified through these tests that the latent Gaussian sources,  $\mathbf{P}$ , are the most challenging parameters to estimate. The second set of tests were designed to explore the circumstances under which  $\mathbf{P}$  can be determined successfully. A second nominal test was defined for these less noisy trials. The parameters describing the degrees of freedom were again varied around nominal, but here an effort was made to map out the space where the algorithm is successful at unmixing  $\mathbf{P}$ . One parameter was altered to make the problem more challenging, and a second parameter was then varied to bring the problem back within the workable space. This is described in more detail in Section 5.3.3.

### 5.3.1 Nominal Gaussian Test

Table 5.1 summarizes the nominal parameters used for the algorithm comparison tests using simulated jointly Gaussian data. These nominal parameters explore a moderately noisy case ( $SNR_X = 16$  dB) for a system  $\mathbf{X}$  of 120 variables, each with 2400 test samples, where  $k = 24$ . The influence of each of these parametric choices is explored further in the subsequent section, with the results in Table 5.3. A series

Table 5.1: Nominal Test Parameters

Parameter	Value
n	120
k	0.2n ( $k = 24$ )
m	100k ( $m = 2400$ )
$SNR_X$ (SNR of $\mathbf{Z}$ to $\mathbf{N}$ )	16 dB
$\Delta\lambda_A$ (Singular values of $\mathbf{A}$ )	$\sim 2$ dB

of 50 test matrices  $\mathbf{X}$  was created using the set of parameters detailed in Table 5.1 with independent zero-mean unit-variance Gaussian random values chosen for the entries of  $\mathbf{P}$  and  $\mathbf{W}$ . Each test matrix  $\mathbf{X}$  had independent random values for  $\mathbf{A}$ ,  $\mathbf{P}$ ,  $\mathbf{G}$ , and  $\mathbf{W}$  while all have a system order of 24.  $\mathbf{A}$  was created by multiplying a matrix of Gaussian random numbers by a set of  $k$  orthonormal vectors with singular values separated by 2 dB, producing a matrix  $\mathbf{A}$  with singular values separated by  $\sim 2$  dB. The diagonal entries of the noise covariance matrix  $\mathbf{G}$  were chosen from an exponential distribution and scaled to produce the desired signal-to-noise ratio  $SNR_X$  for the test. It is important to note that it is actually this scaling which determined the precise dB spread of the entries of  $\mathbf{G}$ . The SNR performances of ONA, ION, and SOBI were then evaluated for these 50 tests, where SOBI was given the correct order  $k$ ; the SNR for each estimate (of  $\mathbf{G}$ ,  $\mathbf{Z}$ ,  $\mathbf{N}$ ,  $\mathbf{W}$ ,  $k$ ,  $\mathbf{A}$ , and  $\mathbf{P}$ ) appears in Table 5.2 ordered by the success of ONA. Note that the SNR for  $\mathbf{A}$  was calculated on the  $\min(k, k')$  most correlated columns of  $\mathbf{A}$  and  $\mathbf{A}'$ , and the SNR for  $\mathbf{P}$  was calculated on the corresponding rows of  $\mathbf{P}$  and  $\mathbf{P}'$ .

Table 5.2 demonstrates that for this test data, by adding SOBI and different initializations, ONA significantly improves the estimates of  $\mathbf{A}$  and  $\mathbf{G}$  in comparison to ION and improves the estimate of  $\mathbf{A}$  when compared to SOBI alone. Corresponding results for PCA are not included as they are unreliable when  $k > 1$  and in the case where the columns of  $\mathbf{A}$  are not assumed to be orthogonal. The initialization used by ONA accounts for its improvement in all parameters but  $\mathbf{A}$  and  $\mathbf{P}$ ; all other entries of the ONA column of Table 5.2 apply also to ION'. Only the estimates of  $\mathbf{A}$  and  $\mathbf{P}$  are improved by the addition of SOBI. The improvement relative to SOBI is due to

Table 5.2: SNR (dB) performance of ONA, ION, and SOBI on nominal Gaussian test data (see Table 5.1)

	ONA	ION	SOBI
<b>SNR of <math>\mathbf{G}</math></b>	31.8	31.5	0.296
<b>SNR of <math>\mathbf{Z}</math></b>	28.3	28.4	19.7
$m_k$	0.000	0.204	N/A
$\sigma_k$	0.775	5.03	N/A
<b>SNR of <math>\mathbf{N}</math></b>	12.3	12.4	3.73
<b>SNR of <math>\mathbf{W}</math></b>	8.25	8.23	3.65
<b>SNR of <math>\mathbf{A}</math></b>	2.56	-0.71	1.92
<b>SNR of <math>\mathbf{P}</math></b>	-2.06	-1.72	-2.76

the noise reduction by ONA before  $\mathbf{Z}$  is separated into  $\mathbf{A}$  and  $\mathbf{P}$ . The residual noise is responsible for the limited ability of all three algorithms to recover the signal matrix  $\mathbf{P}$ .

### 5.3.2 Varied Gaussian Tests

To further characterize the performance of these algorithms, a test suite was developed that explores the five axes of the problem space described in Section 5.2 by generally increasing the challenge. The test suite is summarized in Table 5.3, where the baseline experiment is that of Table 5.1. The second experiment increased  $n$  from 120 to 400 while keeping the ratio  $\frac{k}{n}$  constant at 0.2, thus increasing  $k$  to 80. The third experiment increased the ratio  $\frac{k}{n}$  from 0.2 to 0.3 so  $k = 36$  for  $n = 120$ . The fourth experiment decreased the ratio  $\frac{m}{k}$  from 100 to 20, thus reducing the redundancy in the data. The fifth experiment decreased  $SNR_X$  from 16 to 10 dB, and the sixth experiment assumed the singular values of the mixing matrix  $\mathbf{A}$  were each separated by an average of only 0.75 dB instead of 2 dB. The results suggest that increasing  $n$  and  $k$  significantly improves the SNR of estimates for  $\mathbf{Z}'$ ,  $\mathbf{N}'$ , and  $\mathbf{W}'$ , while noticeably degrading the SNR's for  $\mathbf{P}'$ ,  $\mathbf{G}'$ , and  $k'$ . The most significant effect of reducing the number  $m$  of test samples by a factor of five was a drop of 8 dB in the SNR for  $\mathbf{G}'$ . Also, the metrics improve for  $k'$  and  $\mathbf{P}'$  when the singular values of  $\mathbf{A}$  are separated by an average of only 0.75 dB.

Table 5.3: SNR (dB) performance of ONA on varied Gaussian test data

Experiment	Nominal	<b>n = 400</b>	<b>k = 0.3n</b>	<b>m = 20k</b>	<b>SNR<sub>X</sub>= 10 dB</b>	<b>Δλ<sub>A</sub>= 0.75 dB</b>
<b>SNR of G</b>	31.8	31.3	35.2	25.5	32.8	28.9
<b>SNR of Z</b>	28.3	32.4	28.2	26.7	23.0	26.7
$m_k$	0.000	-0.0025	0.0028	0.0083	-0.0167	-0.0021
$\sigma_k$	0.775	0.894	0.316	0.548	1.52	0.387
<b>SNR of N</b>	12.3	16.4	12.2	10.7	13.0	10.7
<b>SNR of W</b>	8.25	11.9	8.12	7.53	8.87	7.04
<b>SNR of A</b>	2.56	1.63	1.64	2.01	2.37	1.20
<b>SNR of P</b>	-2.06	-2.59	-2.10	-1.96	-1.97	-1.70

As the dimensions of the problem increase and as  $SNR_X$  decreases, the performance of ONA improves significantly relative to that of ION. The same experiments of Table 5.3 were run on ION and the results compared. For all six experiments, ONA increased the SNR of **A** by  $\sim 2$ -3 dB over ION. ONA also increased the SNR of **G** by  $\sim 3$ -5 dB for the experiments increasing  $n$  to 400, increasing  $k$  to  $0.3n$ , and decreasing  $SNR_X$ . In no case was ONA inferior to ION.

The performance of ONA and ION versus PCA can also be evaluated using the scree plot of Figure 2-1 discussed in Section 2.3.2, which presents the eigenvalues of the covariance matrix for the noise-normalized **X** determined for one of the nominal test matrices. For ION and ONA, **X** was normalized by **G'**, setting the noise plateau at 0 dB. For PCA, each row of **X** was simply normalized to unit variance, i.e. the energy in each variable was normalized, setting the mean eigenvalue to 0 dB. The figure suggests comparable performance for ONA and ION, both of which are substantially superior to PCA because the entries in **G** differ so greatly (they are exponentially distributed as described in Section 5.3.1) that noise normalization is critical. The best linear fits to the plateaus are shown, as are the intersections between the screens and the estimated plateaus, which suggest the value of  $k$ . All these methods appear to recover  $\sim 20$  significant eigenvalues. The gap between the largest eigenvalue and the largest noise is shown for each algorithm; the stars (\*) delineate the difference for PCA while the open circles (o) do so for ION and ONA. This eigenvalue difference for PCA is  $\sim 27$  dB and those for ONA and ION are  $\sim 38$  dB, i.e. about 11 dB greater.

Table 5.4: Parameters defining the nominal less noisy BSS experiment

Parameter	Value
n	200
k	4
m	2000
$SNR_X$ (SNR of $\mathbf{Z}$ to $\mathbf{N}$ )	60 dB
$\Delta\lambda_A$ (Singular values of $\mathbf{A}$ )	$\sim 2$ dB

The noise superimposed on the  $k$  signals is presumably typical of the plateau rather than the worst case, however.

### 5.3.3 Varied Gaussian Tests with less Noise

The parameters in the examples of Section 5.3.2 have not permitted  $\mathbf{P}$  to be recovered with SNR above 0 dB. Table 5.5 explores the problem space where successful recovery of  $\mathbf{P}$  is possible. The parameters of the nominal experiment are shown in Table 5.4, and the deviations from nominal are labeled in the heading of Table 5.5. For each successive experiment, one parameter was altered from the nominal value to make the data more difficult to separate and a second parameter was altered to bring the data set back to a working regime. In the second column, the order  $k$  is increased to 5, and the number of samples  $m$  is correspondingly increased to 3000. In the third column, the number of samples  $m$  is decreased to 1800 while the SNR is raised by 10 dB to 70 dB. In the fourth column,  $SNR_X$  is dropped to 55 dB, and the order is also dropped to 3. In the final column, the separation between successive singular values of  $\mathbf{A}$  is decreased to 0.75 dB, and the order is decreased to 2 to compensate. Each result is the average (dB) of ten simulated tests.

Table 5.5: SNR (dB) performance of ONA on less noisy Gaussian test data

<b>Experiment</b>	<b>Nominal</b> (Table 5.4)	<b>k = 5</b> <b>m = 3000</b>	<b>m = 1800</b> <b>SNR<sub>X</sub> =</b> <b>70 dB</b>	<b>k = 3</b> <b>SNR<sub>X</sub> =</b> <b>55 dB</b>	<b>k = 2</b> <b>Δλ<sub>A</sub> =</b> <b>~0.75 dB</b>
<b>SNR of G</b>	40.4	42.7	44.6	46.8	34.6
<b>SNR of Z</b>	79.9	91.5	77.5	84.5	79.2
<i>m<sub>k</sub></i>	-1.93	-0.280	-1.48	-0.233	-4.60
<i>σ<sub>k</sub></i>	9.89	1.67	9.21	1.05	10.8
<b>SNR of N</b>	19.9	21.5	22.5	24.5	19.2
<b>SNR of W</b>	14.4	16.3	16.8	19.3	14.4
<b>SNR of A</b>	4.00	5.12	6.37	10.6	4.43
<b>SNR of P</b>	-0.189	-0.513	2.45	6.07	-0.200





# Chapter 6

## Example Data Set: Factory Data

### 6.1 Introduction to Non-Gaussian Data

Since the EM algorithm embedded in ION (and thus ION' and ONA) was derived for Gaussian data, it is interesting to see how well these algorithms perform with real non-Gaussian data. For this purpose a data set characterizing a large single-product factory was chosen. The data comprise 7847 sequential test vectors of 577 variables, each of which was separately normalized to zero mean and unit variance. These variables include temperatures, pressures, power levels, product quality measures, switch positions, and many others, all recorded over many weeks.

This data was processed using Principal Components (PCA), the original ION algorithm of [10], and the final ONA algorithm.<sup>1</sup> The algorithms are compared for three distinct tasks: the ability of the algorithm to separate  $\mathbf{X}$  into  $\mathbf{Z}$  and  $\mathbf{N}$ , i.e. noise reduction of the data; the extent to which the algorithm groups the variables into small, correlated sets; and the quality of the sources, i.e.  $\mathbf{P}$ , retrieved from the data. Also, the ONA algorithm is used to process factory data which has been time-scrambled, that is the sequence of 7847 vectors is scrambled before processing and the results unscrambled accordingly. Testing ONA on scrambled data documents the ability of the ION portion of ONA to produce good results even on data with

---

<sup>1</sup>All of the results produced in this chapter for the ONA algorithm are identical to those produced with ION' except the separation of the sources  $\mathbf{P}$ . That is, all results for ONA in Sections 6.2 and 6.3 apply identically to ION'.

no correlated sequence. It also demonstrates that the SOBI portion of ONA takes advantage of such correlation in the vector sequence.

## 6.2 Noise Reduction

The log-scrree plots in Figure 6-1 show the results of applying PCA, ION, and ONA to the factory data. Note that the ONA curve is identical for both time-scrambled and time-sequenced data as the sequence of the data is not used to predict  $G$ . Although converting the x-axis to the logarithm of the eigenvalue index in Figure 6-1 better reveals the SNR behavior of these algorithms for small indices, it also obscures the scree-plot breakpoint that normally reveals the signal order  $k$ , so the final order estimations are indicated by circles on the plot. The figure shows that even the first iterations of ION and ONA exceed the performance of PCA, and that after three iterations the first few eigenvalues for ONA exceed those of ION by as much as two orders of magnitude. These results also suggest the benefits of iteration for both ION and ONA, and the ability of both algorithms to raise the eigenvalues above the 0 dB noise baseline where  $\lambda_i \cong 1$ .

The scree plot of Figure 6-1 is shown for the data noise-normalized by the estimated noise variances. In order to understand how significantly the noise normalization altered the original data, one can examine the spread of the estimated noise variances. The final noise variances for the 577 channels, i.e.  $\mathbf{G}'$ , estimated by ONA are shown in Figure 6-2. The values here are all less than unity because the data was normalized to unit variance before being processed by ONA. (This was important because the variance of the variables ranged from  $3.9514 * 10^{-16}$  to  $4.6309 * 10^8$ .) The variability of the estimated noise variances is extensive and indicates that noise levels for different variables also vary greatly. In such cases, even a technique as simple as noise-normalization of the data can significantly improve further data analysis.

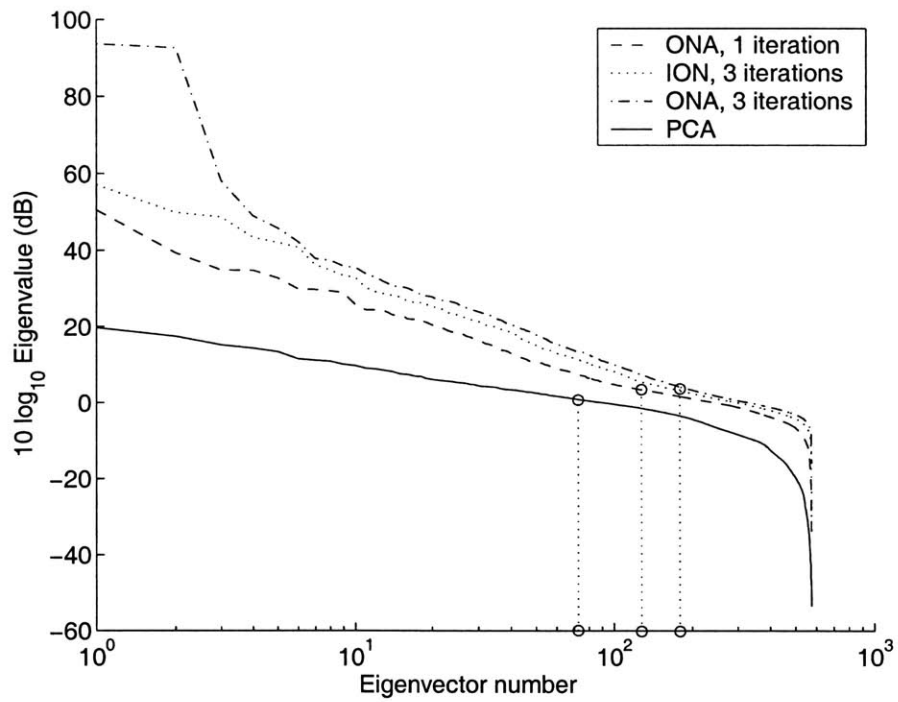


Figure 6-1: Scree plots of noise-normalized factory data

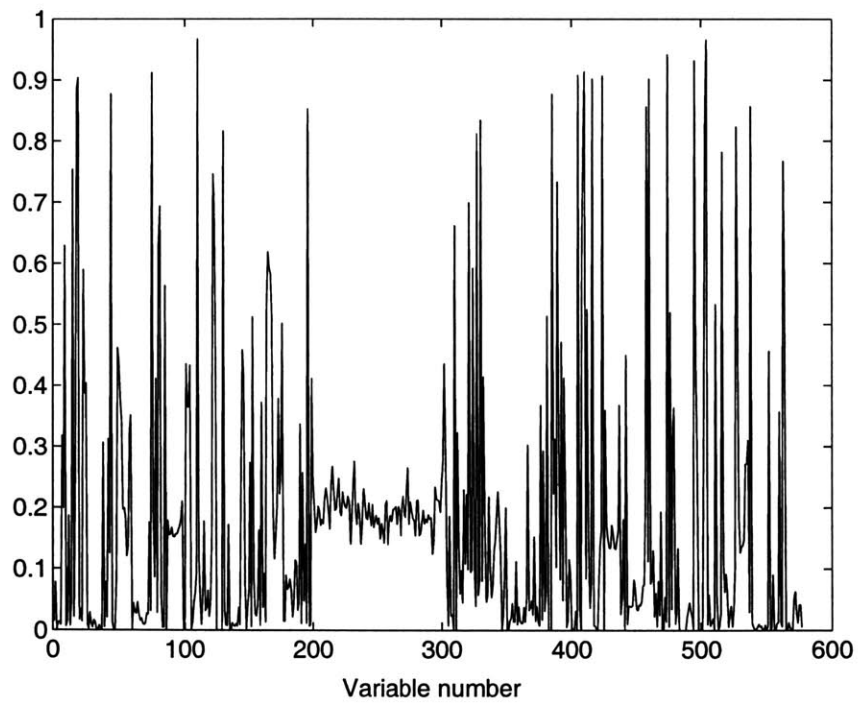


Figure 6-2: Noise Variance per Variable for Factory Data

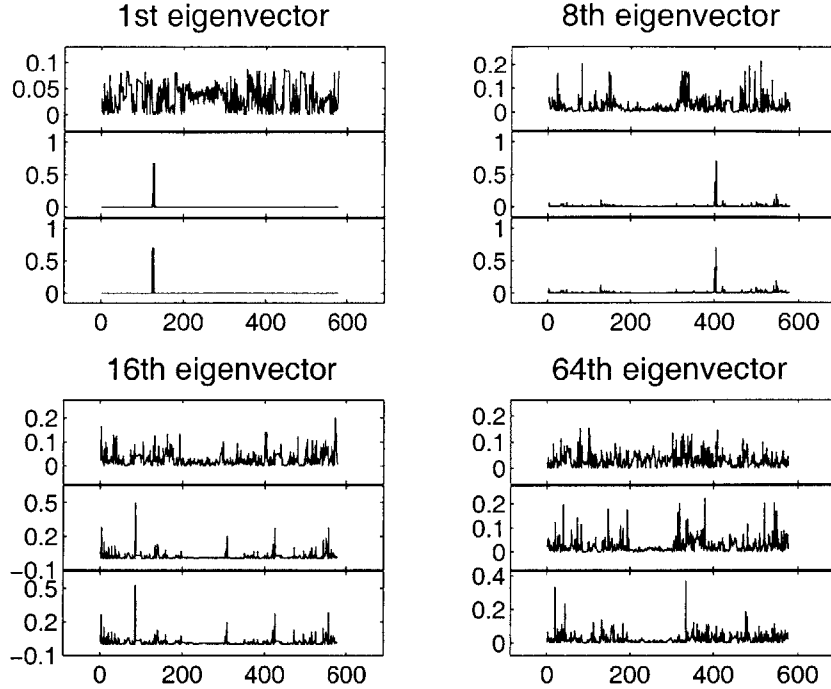


Figure 6-3: Factory eigenvectors for PCA (top), ONA with time-scrambled data (middle), ONA with time-ordered data (bottom)

### 6.3 Variable Grouping

Figure 6-3 characterizes the ability of these algorithms to identify groups of parameters in  $\mathbf{X}$  that are correlated. The figure presents the  $n$ -element eigenvectors derived for the factory data, i.e.  $\mathbf{X}'_n = (\mathbf{G}')^{-1}\mathbf{X}$ , for 1) PCA, 2) ONA for time-scrambled factory data, and 3) ONA for time-ordered factory data. (The data is normalized to unit-variance for PCA while it is normalized by the predicted noise covariance matrix for ONA.) Since even the low-order eigenvectors deduced using PCA involve contributions from nearly all 577 variables, it is fair to assume that these eigenvectors may not be very meaningful physically. In contrast, both ONA experiments single out a few dominant variables that are strongly correlated, and the time-sequential ONA retains some of this ability even at the 64th eigenvector.

In order to fully quantify the success of ONA, the extent to which all of the eigenvectors perform this grouping is measured. The first ten eigenvectors determined by ONA for time-sequenced factory data are plotted in Figure 6-4, and all ten clearly

identify small groups of correlated variables. This is expected for low-order eigenvectors, however. To determine the extent to which higher-order eigenvectors also perform this grouping, every tenth eigenvector is plotted in Figure 6-5 up to the 100<sup>th</sup> for ONA. While the highest-ordered eigenvectors have contributions from most of the variables, ONA retains some ability to categorize the variables even at the 50<sup>th</sup> eigenvector. The ability of the algorithm to produce so many physically significant variable groups indicates accurate estimation of the noise variances used for noise-normalization.

The first, second, fourth, and tenth factory eigenvectors were chosen and inspected to find such correlated groups of variables; these eigenvectors and the time sequences of the corresponding  $\mathbf{Z}_i$  are shown in Figure 6-6 where the peak excursions have been normalized to unity. Here the contribution of a variable to an eigenvector was considered significant if it was more than  $6\sigma$  from the mean. This criterion identifies variables 125 and 127 in the first eigenvector, 126 and 129 in the second eigenvector, 33 and 34 in the fourth eigenvector, and 128, 499, 500, and 501 in the tenth eigenvector. ONA has clearly grouped variables that exhibit correlated time histories.

## 6.4 Source Separation

Neither PCA nor ION (nor ION') takes advantage of possible time correlations in the data whereas SOBI and therefore ONA do. Recall from Section 2.4.3 that SOBI estimates the data correlation matrix by multiplying data vectors not only by themselves but also by adjacent correlated samples, thus reducing noise, particularly for those eigenfunctions that are most correlated in time. As a result, ONA is more successful than ION or PCA in identifying time-correlated variables that cluster well. Also, ONA used on time-ordered data does better than ONA on time-scrambled data due to the lack of correlation structure in the latter data. Figures 6-7 and 6-8 suggest the relative degrees to which the time behavior of the retrieved signals is organized for PCA, ION and ONA for time-scrambled and time-ordered data, respectively. These figures are scatter plots of the time histories of consecutive rows of  $\mathbf{P}$ , each row having

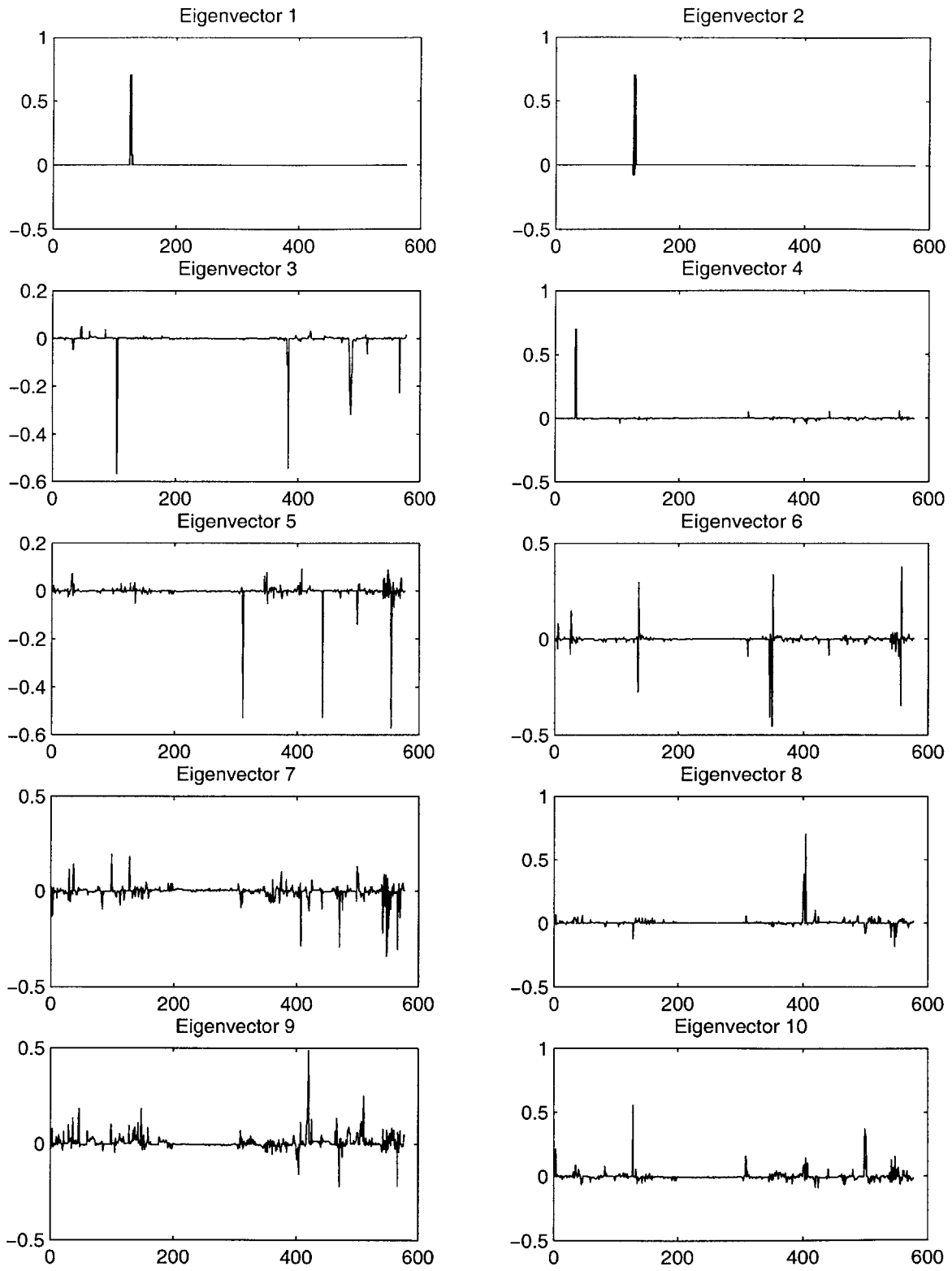


Figure 6-4: First 10 Eigenvectors of Sequenced Factory Data determined by ONA

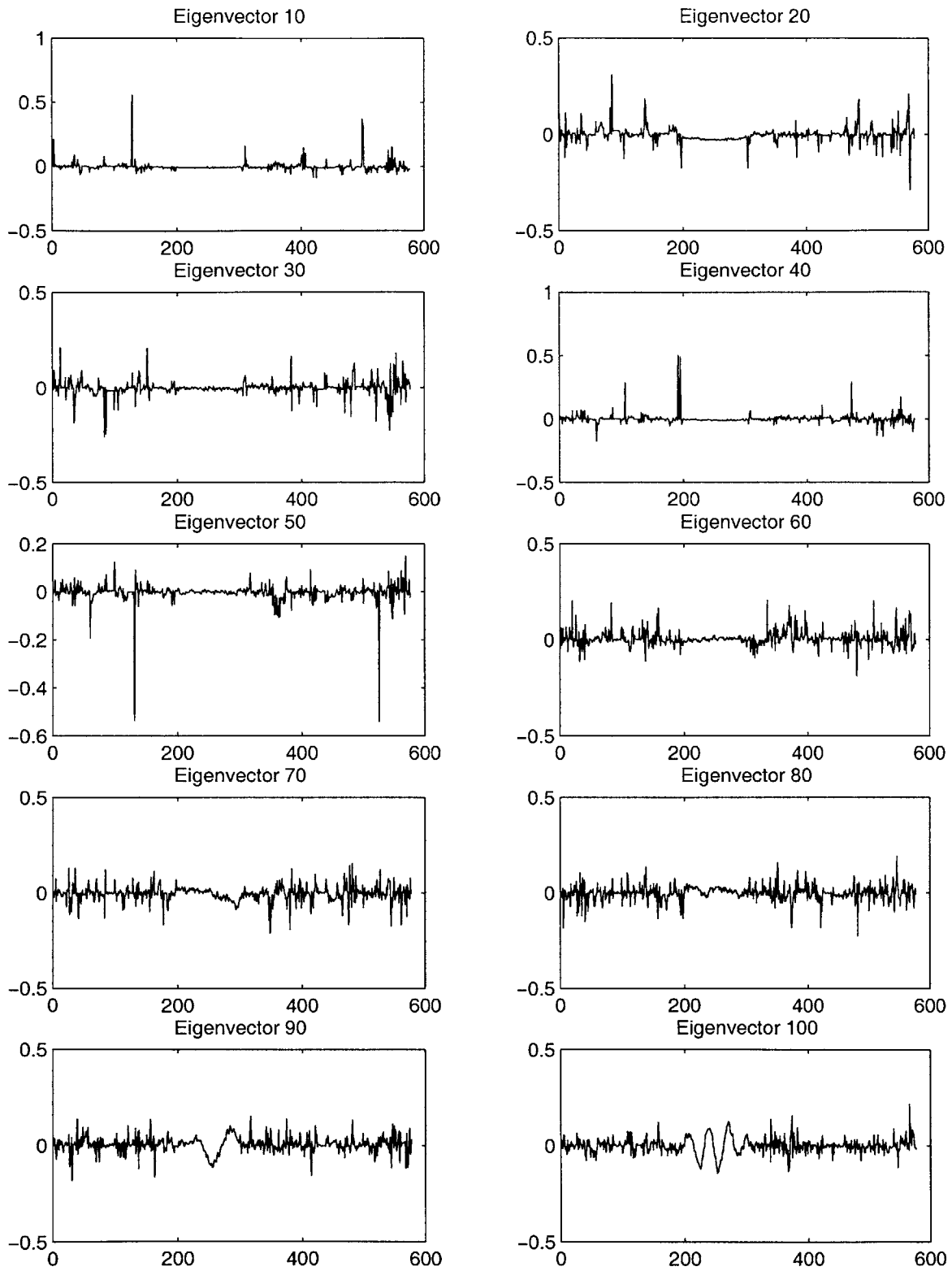


Figure 6-5: Eigenvectors 10, 20, ... , 100 of Sequenced Factory Data determined by ONA

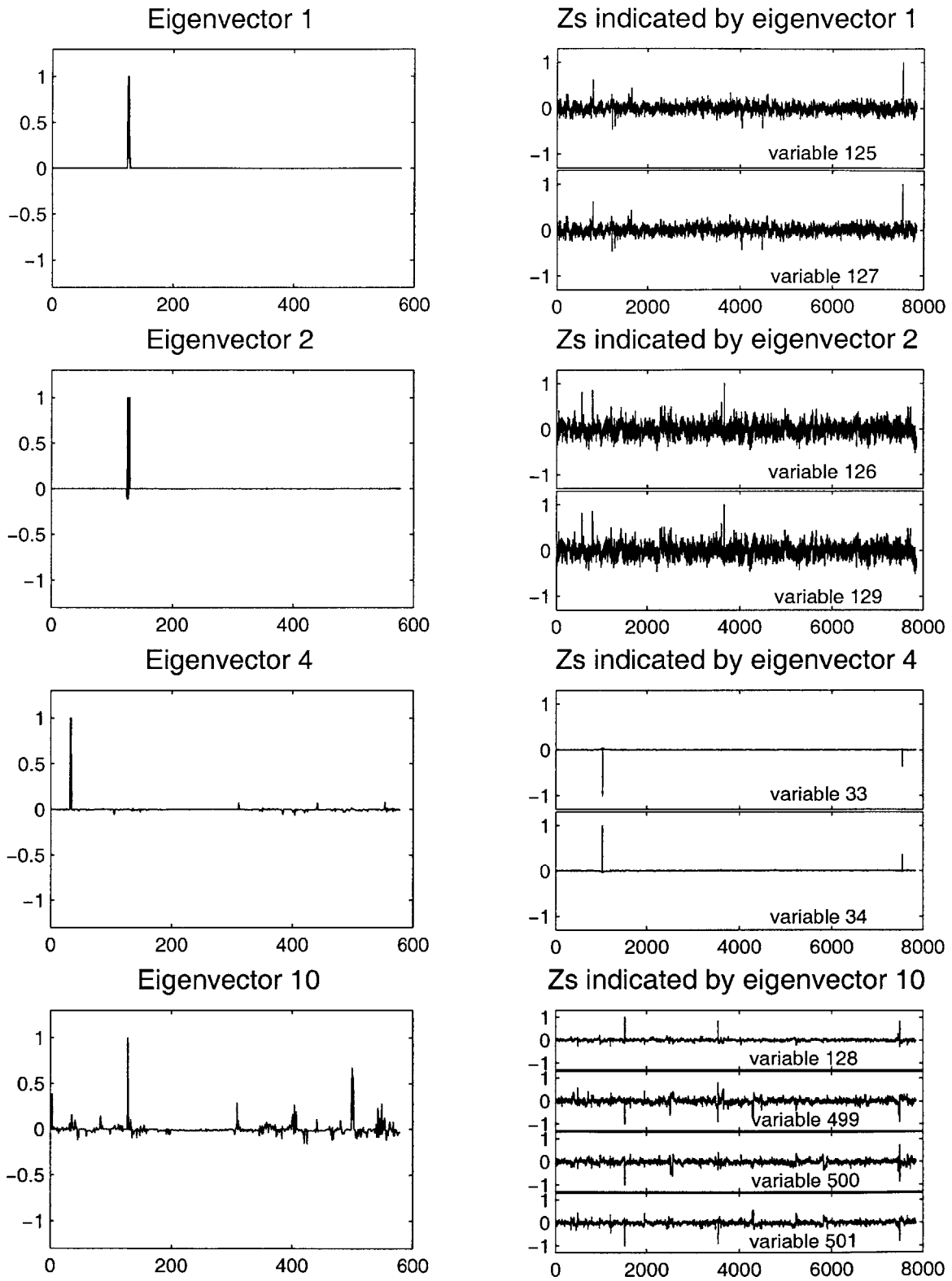


Figure 6-6: Time sequences of groups of correlated  $Z_i$  for ONA



unit variance.  $\mathbf{P}$  for PCA is obtained by normalizing to unit variance the eigenvectors produced by PCA of  $\mathbf{X}'$ . The scatter plots for PCA rapidly resemble Gaussian ellipses surrounded by outliers that contributed to the signal separation whereas the plots for ONA characteristically exhibit fewer outliers and crisp clustering within distinct time intervals.

Finally, inspection of the retrieved sources plotted as a function of time can indicate whether the sources are Gaussian or non-Gaussian and whether they are time correlated or not time correlated. The time sequences of several retrieved sources are plotted in Figure 6-9 for PCA, ONA for time-scrambled data, and ONA for time-ordered data. It is clear that the signals retrieved for ONA are in both cases more structured than those retrieved by PCA.

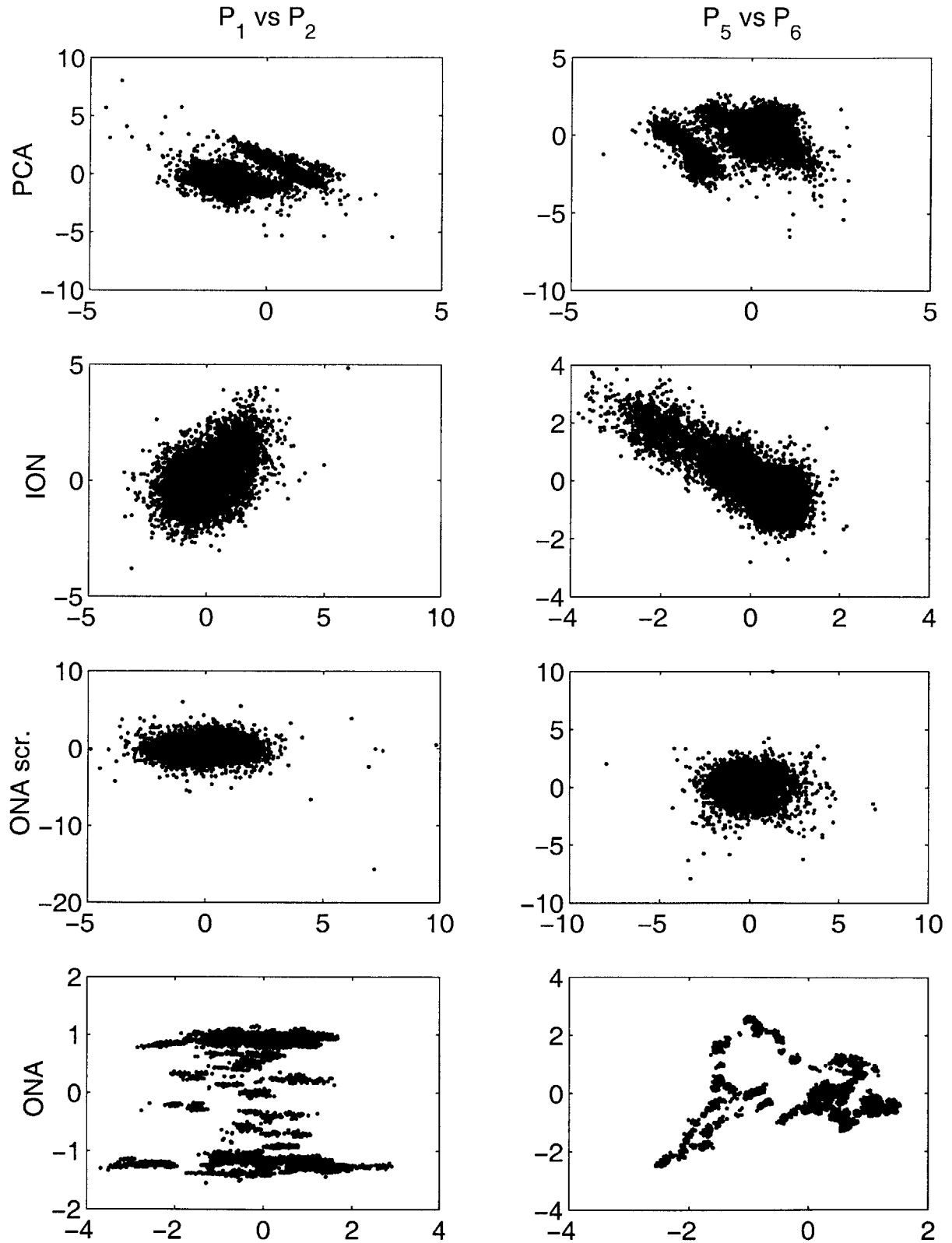


Figure 6-7: Scatter plots of consecutive retrieved factory signals  $\mathbf{P}_i(t)$  and  $\mathbf{P}_{i+1}(t)$  for PCA (top), ION (middle-top), ONA for time-scrambled data (middle-bottom) and ONA for time-ordered data (bottom).

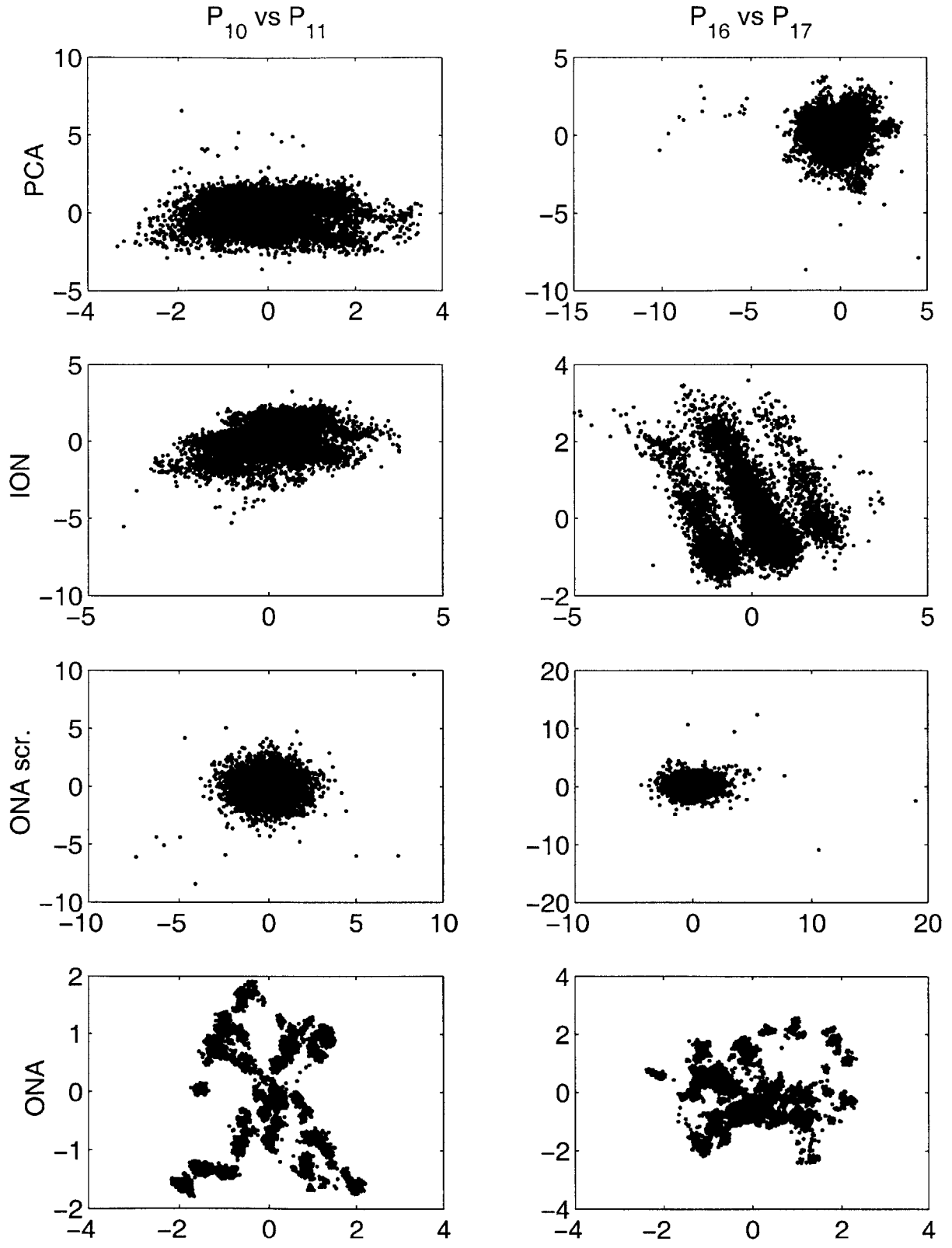


Figure 6-8: Scatter plots of consecutive retrieved factory signals  $P_i(t)$  and  $P_{i+1}(t)$  for PCA (top), ION (middle-top), ONA for time-scrambled data (middle-bottom) and ONA for time-ordered data (bottom).

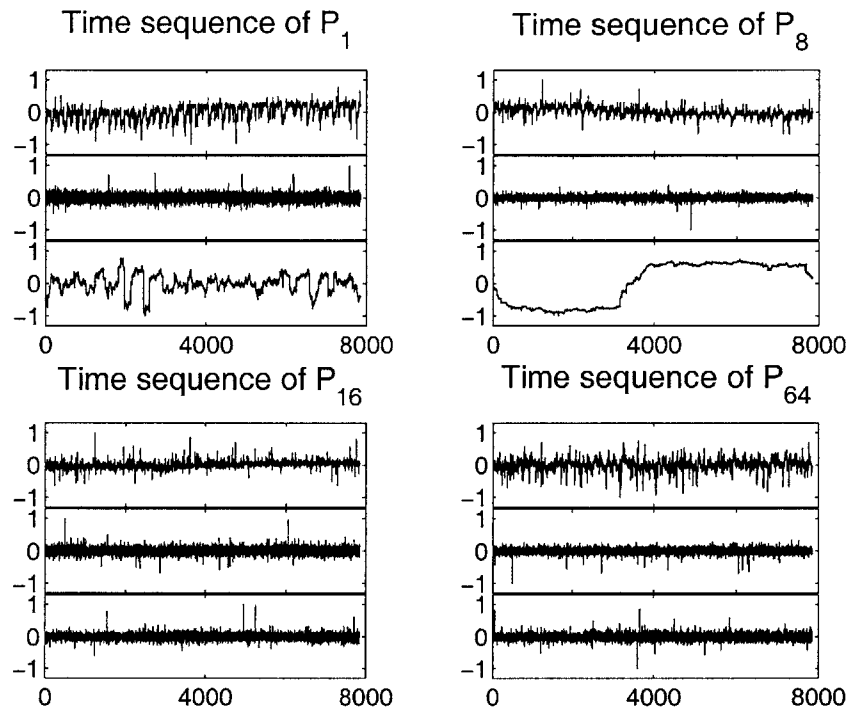


Figure 6-9: Time plots of retrieved factory signals  $P_i(t)$  for PCA (top), ONA for time-scrambled data (middle) and ONA for time-ordered data (bottom).

# Chapter 7

## Extension to 2D Spaces

### 7.1 Applications of Extended Algorithm

The ONA algorithm was developed for application to sets of one-dimensional data, e.g. 500 variable vectors, each with 8000 samples. In this case, the SOBI algorithm, which was developed for data which has one-dimensional correlation, detects and takes advantage of any correlation existing in the sample sequence. For many vector sets, the sample sequence corresponds to consecutive time intervals, and in these cases, SOBI effectively uses any sequential time correlation in the data to improve the estimates of  $\mathbf{A}$  and  $\mathbf{P}$ .

There are many data sets for which the inherent correlation structure is not sequential, however. Any type of spatial or image data, e.g. hyperspectral data, fits this description. If read line by line, the correlation structure of the image may well be sequential, but this method clearly fails to capture any correlation between lines. Also, even if these lines are concatenated to form a one-dimensional vector to take advantage of the per-line correlation, the areas of the vector where two lines are abutted clearly have no necessary local correlation. Here it is the case that the set of covariance matrices are computed created by SOBI fails to capture the true correlation in the data. The concept of jointly diagonalizing a set of covariance matrices is still valid, but the method by which these covariance matrices needs to be revised to support the two-dimensional spatial correlation inherent in the data. This extension

to the SOBI algorithm is developed in Section 7.2.

Because imaging and hyperspectral data sets are usually quite large, practical memory-saving alterations were made in the algorithm implementation. These are documented in Section 7.3 and in Appendix A.

## 7.2 Use of Spatial Correlation in SOBI

In a two-dimensional image, the most likely correlation structure is one where the strength of the correlation between two pixels decreases monotonically with the spatial distance between them. It is possible that no correlation may exist or that the signal may include cyclic patterns so the correlation is more complex; in a blind environment, however, no knowledge is given of these special cases, so the most likely case is assumed.

The power of SOBI comes through jointly diagonalizing a set of covariance matrices; the challenge for spatial data is to construct covariance matrices for 'spatially-shifted' data which correspond to the covariance matrices for time-shifted data constructed in the one-dimensional case. The concept of 'spatially-shifted' can be explored through pixel nearest neighbors.

### 7.2.1 Nearest Neighbors

In general, each pixel has 8 first-ring and 16 second-ring nearest neighbors. For the moment we ignore edge effects which are dealt with in Section 7.2.3. When an image is considered as a set of pixels, however, processing all 24 neighbors is redundant. Only half of the neighbors of each pixel need to be processed because the pixel itself will be considered a corresponding neighbor of the other half. For example, say that the neighbors immediately around a pixel  $P$  are labeled  $N_1$  through  $N_8$  as shown in Figure 7-1. If we choose to examine the correlation between a pixel and the pixel immediately to its left, we will first examine the pair  $\{P, N_8\}$ . We will also eventually examine the pair  $\{P, N_4\}$  because  $P$  is the left-neighbor of  $N_4$ . If the goal is simply to list all pairs of first-ring correlated pixels, it is clearly redundant to list both the

1	2	3
8		4
7	6	5

Figure 7-1: First-Ring Nearest Neighbors for one pixel

left-neighbors and the right-neighbors for all pixels. Here, the pair  $\{P, N_4\}$  would immediately be listed twice, as would all pairs of pixels by extension.

A diagram of the 24 nearest neighbors of one pixel is shown in Figure 7-2 where the correlation need only be measured between the black pixel and the set of white pixels because of the inherent redundancy explained above. Remaining are 4 first-ring neighbors and 8 second-ring neighbors. In general, it will be the case that a pixel will be most correlated with its first-ring neighbors and less correlated with those in the second ring, so the spatial correlation function is first done only for first-ring neighbors. Extension of the algorithm to include second-ring neighbors (and in fact any number of rings of neighbors) is straight-forward and thus will not be discussed further here.

In order for the joint diagonalization technique to work, a set of covariance matrices need to be constructed. The set proposed by the above discussion is a set of five covariance matrices as follows:

1. the self-covariance of a pixel
2. the covariance between a pixel and its immediate right neighbor
3. the covariance between a pixel and its immediate top-right neighbor
4. the covariance between a pixel and its immediate top neighbor
5. the covariance between a pixel and its immediate top-left neighbor

corresponding to the white first-ring neighbors in Figure 7-2.

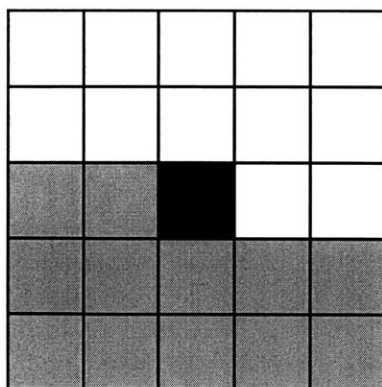


Figure 7-2: First- and Second-Ring Nearest Neighbors for one Pixel

### 7.2.2 Creation of Spatial Covariance Matrices

In order to more clearly discuss the construction of spatial covariance matrices, an example application is necessary. The application most appropriate, which is discussed further in Chapter 8, is data retrieved through hyperspectral imaging. By nature, this data has one spectral dimension and two spatial dimensions; the pixels in the two spatial dimensions are analogous to the time samples in one dimension in problems discussed earlier. Thus it is valid to create covariance matrices which capture the correlation structure between pixels in order to unmix the data in the spectral dimension.

Each spatial covariance matrix is created by transforming a pair of three-dimensional (one spectral, two spatial) images into a pair of two-dimensional (one spectral, one spatial) matrices, using the same transform for each. The covariance matrix is then computed in the normal manner for a pair of two-dimensional matrices (see Section 2.4.3 for more details). The information latent in the spatial correlation of the image is captured by altering the two images used to compute the covariance matrix. In all cases, the first image is identical to the original image while the second image corresponds to the original image spatially shifted in some direction.

As discussed previously, five such covariance matrices can be constructed for the first-ring nearest neighbors. The self-covariance matrix of the image is created by



performing the above calculations for two identical (un-shifted) copies of the original image. For the remaining four covariance matrices, the second image corresponds to a shifting of the original image by one unit in some direction (as listed in Section 7.2.1). While edge effects are caused by these shifts, they are minimized by a mirroring technique detailed in the following section.

### 7.2.3 Dealing with Edge Effects: Mirroring

Clearly not all pixels in a rectangular image actually have all eight first-ring nearest neighbors. At the extreme, pixels in a corner have only 3, while pixels along the edges typically have only 5. In order to develop a generalizable function that can spatially shift an image to create the desired shifted covariance matrices, mirroring is employed. This concept simply means that the original image is padded with reflections of itself so that all pixels have eight nearest neighbors, as shown in Figure 7-3. Processing is done only on the original image; this padding exists simply for ease of construction of the second images used above to create the corresponding covariance matrices. Visually, this means that the original image and the shifted image now overlap for all of the original pixels of the image, so the misalignment of these two images is no longer a problem.

The padding of Figure 7-3 is created simply by reflecting the entire image about each of its edges. The black and white image in the center is the original image, and the gray-scaled images are the padding. For example, the padding to the right is created by reflecting the image about the right edge, etc. This simple technique guarantees that all pixels have valid nearest neighbors by assuming that the pixels near the edge closely approximate those that would exist outside of the image borders. While this does in fact introduce inaccuracies in the spatial shift, they are much less than those that would exist if other padding techniques, such as simply re-copying the unreflected image about each edge, were used. It is also important to realize that the percentage of pixels for which the padded neighbors are used in the covariance matrices is small compared to the absolute size of the image, so any inaccuracies in the approximation have little impact on the final results.

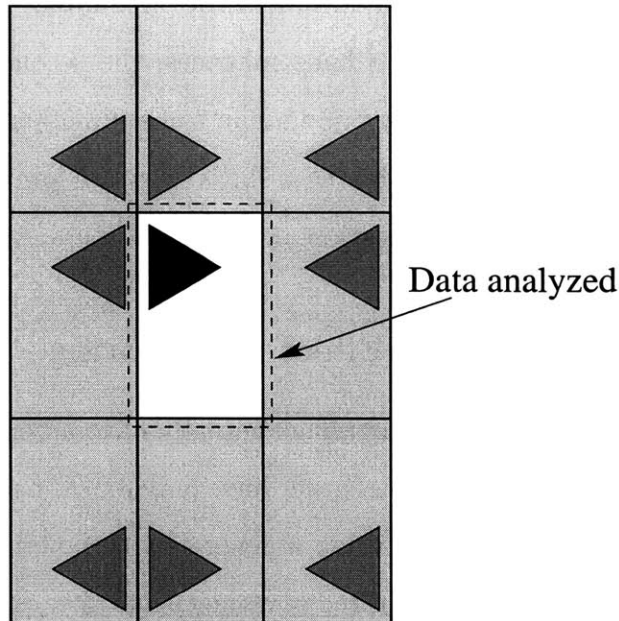


Figure 7-3: Mirroring technique used to pad an image for construction of covariance matrices

### 7.3 Memory Management

The addition of Step 6 of Table 2.1 and Table 2.3 requires that the results from each iteration be compared to determine which produced the best results. This means that  $\mathbf{G}_i'$ ,  $\mathbf{Z}_i'$ , and  $k_i'$  must be stored for all  $i < i_{max}$ . Although it is rarely necessary to set  $i_{max} > 10$ , as the size of the data sets grows it still becomes unmanageable to store this set of matrices in the Matlab workspace. Instead, at the end of each iteration, the new  $\mathbf{G}_i'$ ,  $\mathbf{Z}_i'$ , and  $k_i'$  are appended to a temporary file where the iterative results are stored. It is possible to move this data to disk because the iterative algorithm uses only the results of the previous iteration, not those of all previous iterations. In order to determine the best iteration, the contents of this file are simply read and the set  $\{\mathbf{G}_i', \mathbf{Z}_i, k_i\}$  is passed to the iteration determination algorithm.

Additionally, many of the matrix operations performed in the iterations, especially within the Expectation-Maximization Algorithm, are very memory-intensive. Care was taken to perform these operations sequentially, i.e. complex Matlab statements

were broken into several simple statements in order to minimize the memory demands of the program. This allowed for data sets of much greater size to be processed using ONA.

The most expensive matrix operation involved in these calculations is the matrix left-divide. In Matlab,  $\mathbf{A} \setminus \mathbf{B}$  is essentially the same as  $\mathbf{A}^{-1}\mathbf{B}$ , but it is calculated differently. In the specialized case where  $\mathbf{A}$  is diagonal, however, this calculation can be made more efficient by performing a row-wise division of  $\mathbf{B}$  by the inverted diagonal elements of  $\mathbf{A}$ . A script was created to do precisely this, further reducing the memory-intensive demands of the algorithm and increasing the size of the data sets that can be processed. The code which performs this special matrix left division is included in Appendix A.1.



# Chapter 8

## Evaluation of 2D ONA: Hyperspectral Data

### 8.1 Hyperspectral Data Overview

Hyperspectral visible and infrared data is used to sound both the surface of the earth and the earth's atmosphere. Hyperspectral refers to the simultaneous mapping of the surface by a large number of frequencies. By nature, this data has three dimensions, two spatial and one spectral; an example of this type of data is shown in Figure 8-1.

The data used to evaluate ONA in this chapter comes from AVIRIS, the Airborne Visible InfraRed Imaging Spectrometer. This instrument was developed at the Jet Propulsion Laboratory (JPL) at the California Institute of Technology with sponsorship by NASA. AVIRIS sounds in 224 contiguous spectral channels; the wavelengths used range from 370 to 2500 nanometers (nm). The image used here is of Moffett Field (taken in 1997), and it has resolution of approximately 17 meters [2].

The motivation behind applying ONA to hyperspectral data is the contrast between this and the previous data set. While the factory data is clearly non-Gaussian, hyperspectral data can be modeled fairly reliably as Gaussian. Also, the previous data had time correlations that could be exploited to improve separation. Hyperspectral data instead has inherent spatial correlation among pixels, and it was desirable to modify ONA to exploit this type of correlation as well. While many algorithms

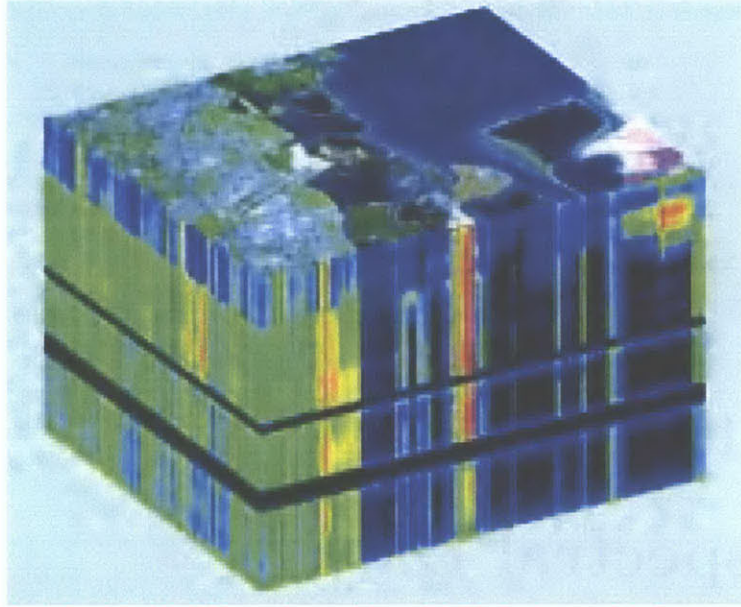


Figure 8-1: Example Hyperspectral Data Set from AVIRIS, Complements of JPL [2]

designed to process hyperspectral image data exist, most use clustering or filtering techniques or require that the noise variances be known, e.g. [12], [15], [9]. In this initial evaluation, ONA is not specifically intended to compete with these algorithms, although further work may reveal that its noise estimations may be used in conjunction with NAPC or other similar algorithms.

## 8.2 Evaluated Image

The data set chosen for analysis is of the Moffett Field site in California. This data set is appropriate for evaluation of the algorithm because it combines natural surface features (forest, mountain, lakes) with man-made surface features (roads, houses, etc.). While it is important to test the ability of the algorithm to deal with the existence of surface features, the most challenging aspect of the data is the extreme variation in surface types. A false color image of the entire Moffett Field data set is shown in Figure 8-2, where the data is one of several free data sets available on the JPL AVIRIS website [2]. This image has been processed by hand to accentuate ground features and clearly shows the variation between rural and urban areas. Because this

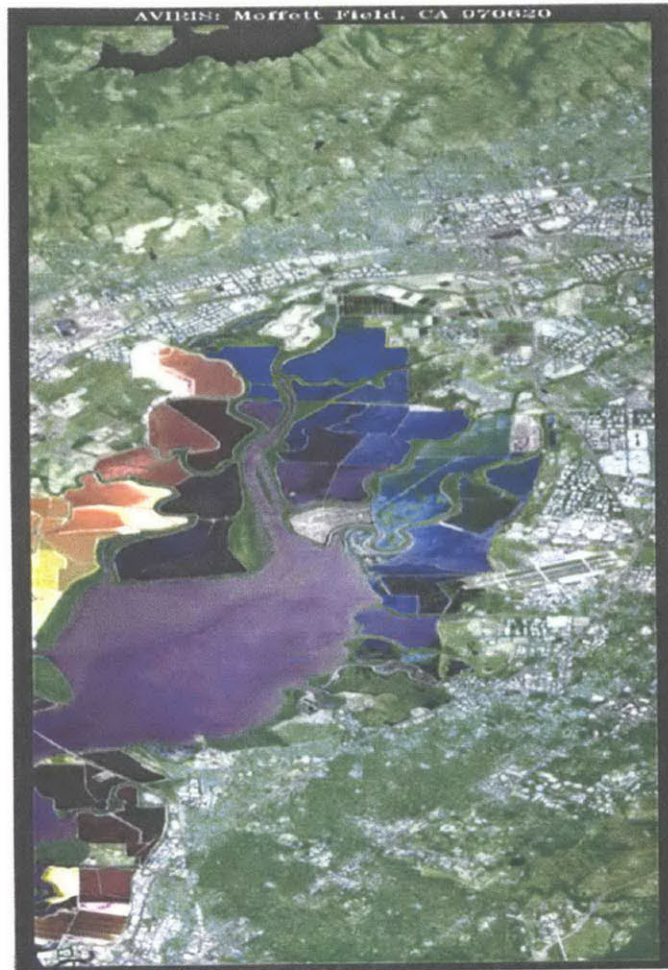


Figure 8-2: False Color Image of the Moffett Field Data Set from AVIRIS, Complements of JPL [2]

image actually spans many kilometers from top to bottom, only a small sub-image is used to evaluate ONA. This sub-image, which is comprised of approximately one-ninth the pixels of the whole image, is taken from the upper-right hand corner and is representative of the entire image in that it contains a corresponding ratio of rural and urban areas.

### 8.3 Noise Reduction

As discussed in 2.3.2, one good way to evaluate how well the noise has been removed from the data is through a scree plot. Such a plot is shown for the Moffett Field

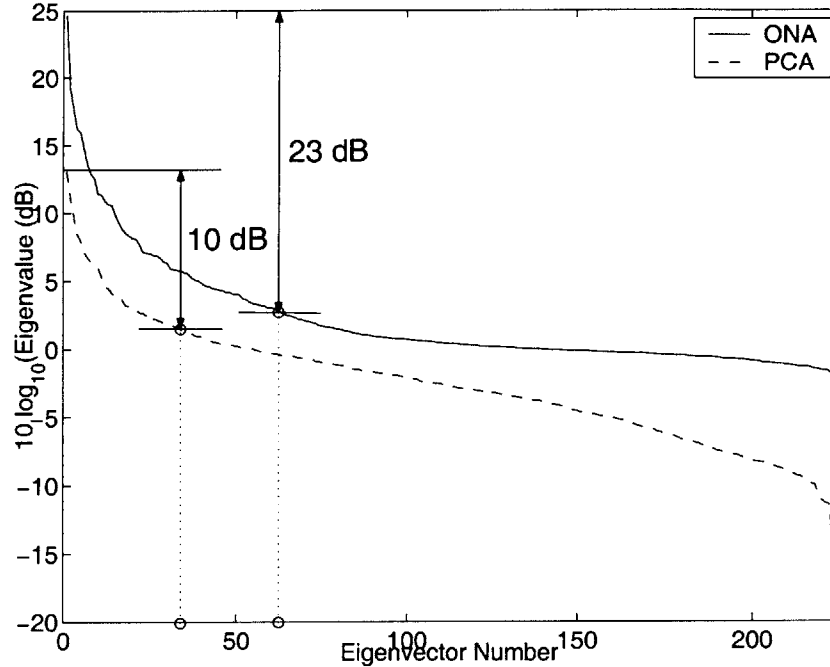


Figure 8-3: Scree plots of noise-normalized Moffett Field image data

sub-image data in Figure 8-3 where the ONA and PCA curves are compared. In this case, ONA estimates the order of the system at 60 while PCA estimates only 33. Also, the largest ONA eigenvalue is nearly 25 dB above the corresponding noise plateau, while this difference is only about 10 dB for PCA, where the noise plateau is less well defined.

The results of processing the Moffett Field sub-image produced estimates of the noise variances as shown in Figure 8-4, where it is again important to recall that the estimated noise variances are all less than or equal to one because the data was normalized to unit variance on each channel before processing. Note that some channels have very low noise levels; the information contained in these channels will be most heavily weighted in the retrieved signal  $\mathbf{Z}$  and in the retrieved sources  $\mathbf{P}$ . One example of this is detailed in Figure 8-5. Channel 153 was chosen because it corresponds to the minimum estimated noise variance in Figure 8-4. Figure 8-5 shows the original image of Channel 153 ( $\mathbf{X}_{153}$ ) in the top-left and the image for the same channel after the noise has been removed by ONA ( $\mathbf{Z}_{153}$ ) in the top-right.



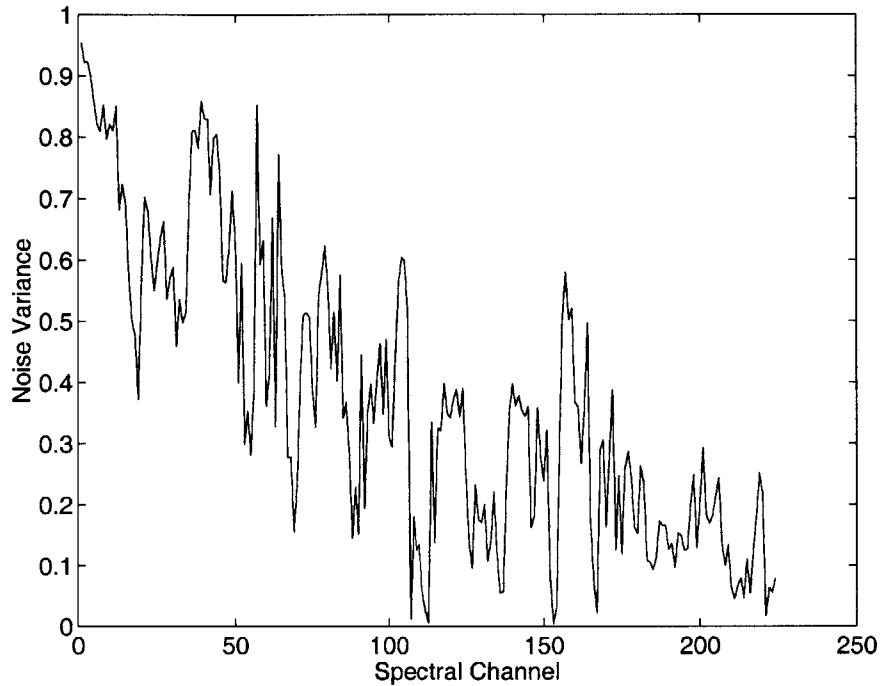


Figure 8-4: Estimated Noise Variances for 224 Spectral Channels for Moffett Field sub-image

Finally, the lower image of Figure 8-5 shows one of the independent sources unmixed by ONA ( $\mathbf{P}_1$ ). It is clear that the information content in these three images is virtually identical; ONA preserves the low-noise levels of Channel 153 (and thus its high information) through both the noise-filtering and the source unmixing steps, as would be expected.

Finally, Figures 8-6 and 8-7 show the ability of ONA to reduce the noise in the images for two channels with moderate noise levels and for two channels with very high noise levels. In Figure 8-6, the top shows the original ( $\mathbf{X}$ ) and noise-reduced ( $\mathbf{Z}$ ) images for channel 109 while the bottom shows the corresponding images for channel 159. The noise variances predicted on these two channels are respectively 0.3120 and 0.6269, and ONA has successfully improved the signal-to-noise ratio on each. Two examples of channels in the most challenging regime (noise variance  $> 0.9$ ) are shown in Figure 8-7. On the top are the original and noise-reduced images for channel 2 (noise variance predicted to be 0.9319), and below are the corresponding

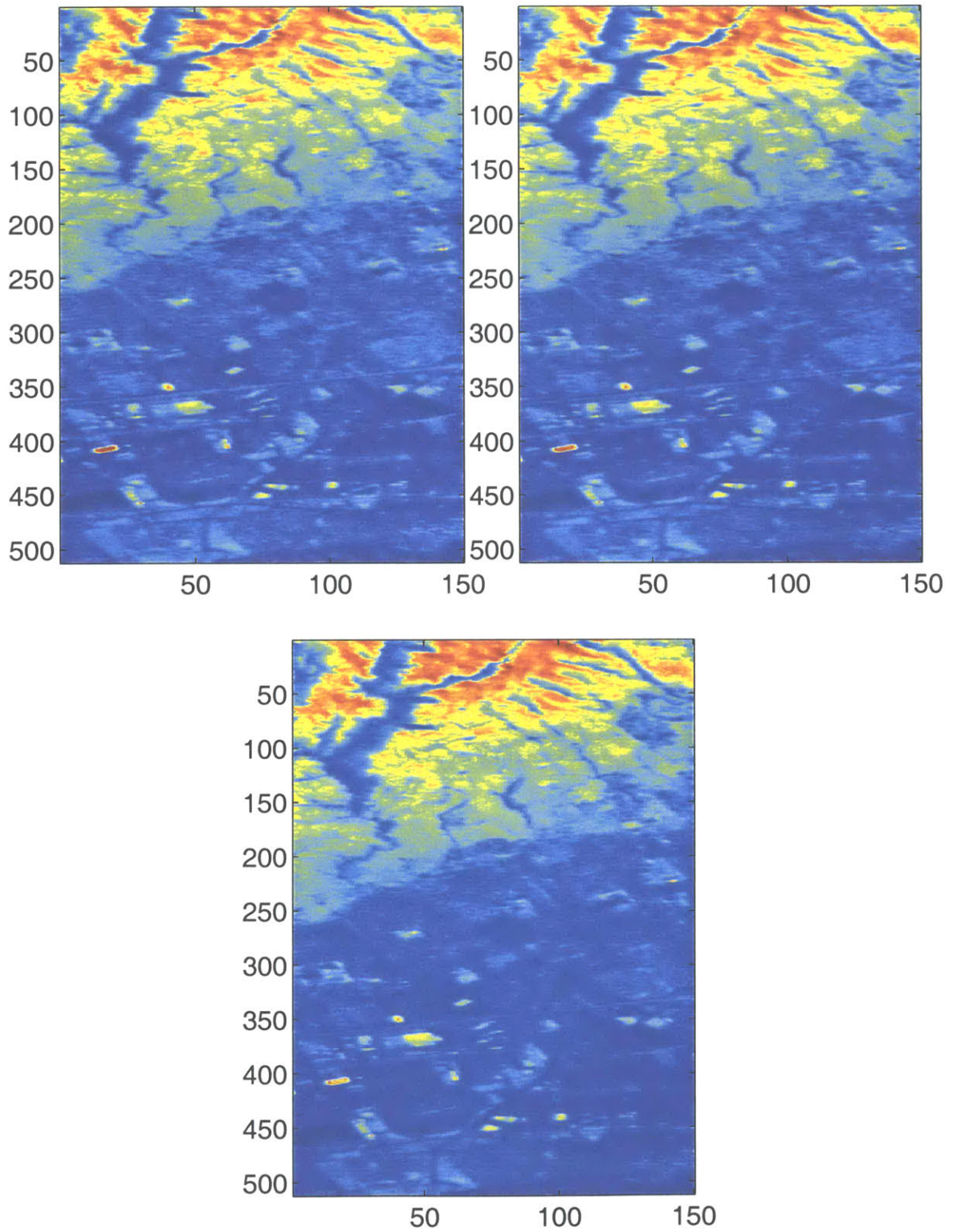


Figure 8-5: Comparison of Original (top-left), noise-filtered (top-right), and corresponding Independent Source (bottom) for Moffett Field sub-image minimum noise channel (frequency channel 153)

images for channel 3 (noise variance predicted as 0.9130). Even in this high-noise regime, ONA has successfully identified the signal portion of the data and removed the noise to produce a better quality image. Some of this apparent noise reduction may have been obtained, however, by overlaying a different signal with higher SNR. More study of this issue seems desired.

## 8.4 Variable Grouping

While previously the eigenvectors were inspected to determine if the algorithm had detected small groups of correlated data, in hyperspectral data the spectral channels will by definition have overlapping information content. This means the eigenvectors will in general have contributions from many channels, indicating the extent to which the channels are correlated. This is the case, as shown in Figure 8-8, for the first ten eigenvectors of the final noise-normalized data. It is significant, however, that few of the first ten eigenvectors contain contributions from channels in the lowest part of the spectrum, i.e. those which were determined to be most noisy as shown in Figure 8-4. This is as would be expected; since the information content, or possibly the novel information content, is higher in the less noisy channels, the correlation among these channels will also be higher.

## 8.5 Signal Separation

For hyperspectral data, it is actually more desirable for individual features of the earth's surface to be grouped into small highly correlated groups than that the frequency channels be grouped in such a manner. Figure 8-9 shows a comparison of the first three principal components retrieved using PCA and three independent sources ( $\mathbf{P}_i$ ) retrieved by ONA. Note that the first principal component detects features in the image, but the noise level increases drastically in the second and third PCs. In contrast, the sources retrieved by ONA, which are by definition unordered, are less noisy and group the image features into smaller independent clusters. For instance,



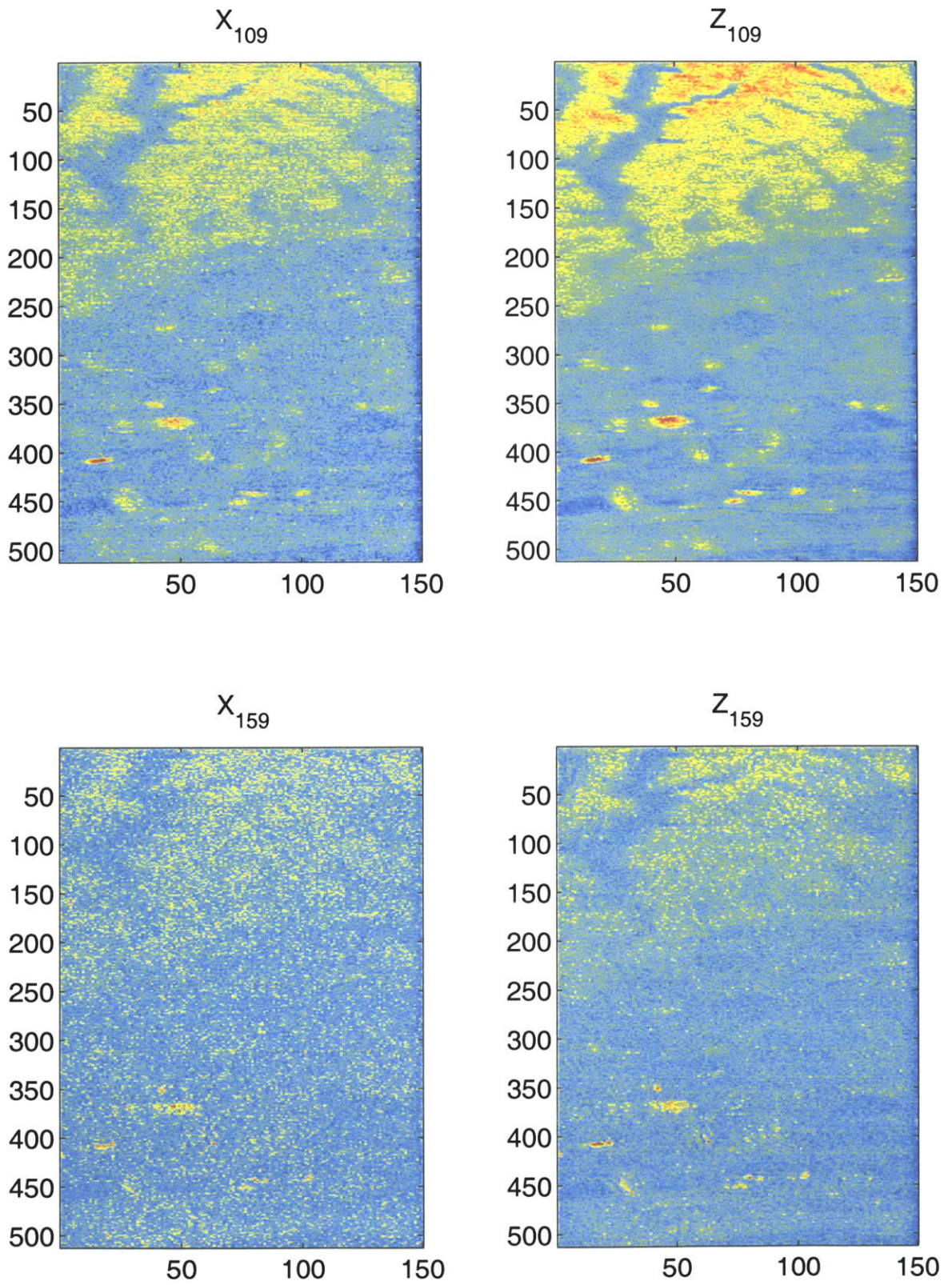


Figure 8-6: Two examples of noise reduction in mid-range noise levels: Original and Noise-Reduced Images for Channel 109 (top) and Channel 159 (bottom)



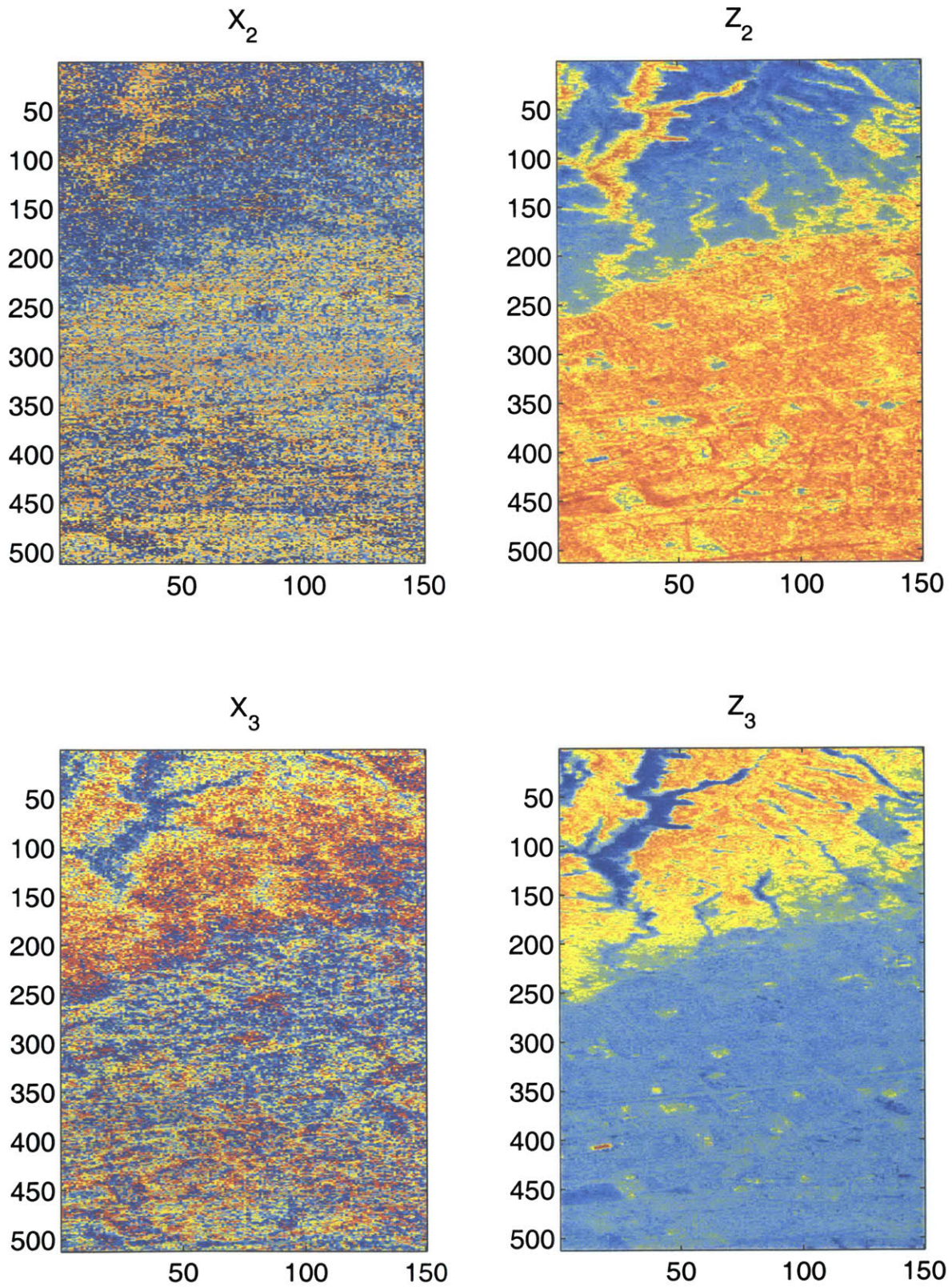


Figure 8-7: Two examples of noise reduction in highest noise levels: Original and Noise-Reduced Images for Channel 2 (top) and Channel 3 (bottom)

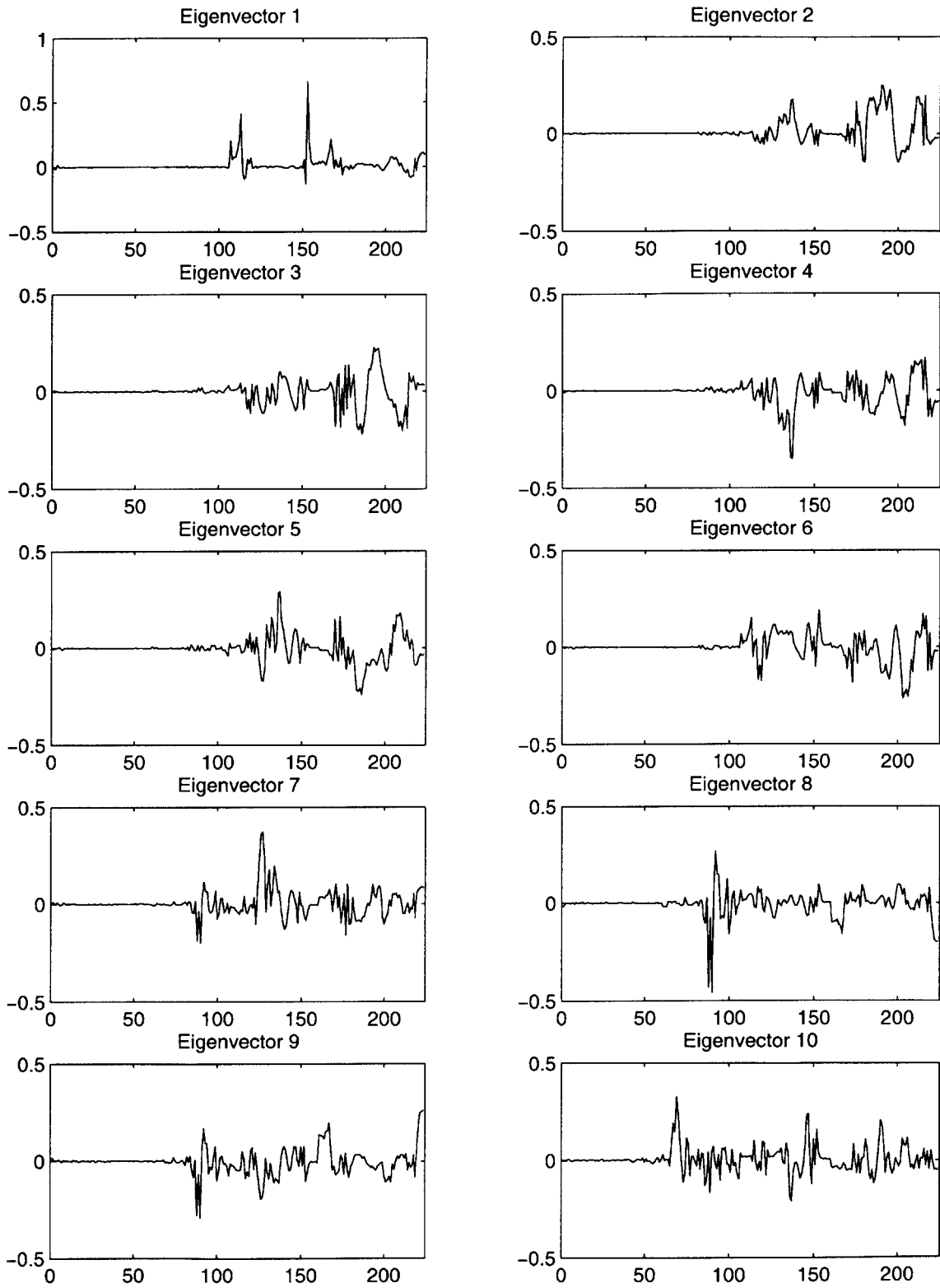


Figure 8-8: First 10 Eigenvectors of AVIRIS Moffett Field sub-image determined by ONA

the few bright features in the lower left corner of  $\mathbf{P}_1$  appear only slightly in  $\mathbf{P}_3$  and  $\mathbf{P}_{11}$ . Also, the road which runs up and to the right from pixel 350 on the left is clearly visible in  $\mathbf{P}_3$  but is not very visible in  $\mathbf{P}_1$  or  $\mathbf{P}_{11}$ . Finally, the features which cover larger areas in the lower left portion of the image in  $\mathbf{P}_{11}$  are virtually undetectable in the other two ONA sources. This ability of ONA both to identify important features in the data and also to separate it into sources which group fewer features together is an advantage over PCA and an asset to further analysis of the retrieved sources.



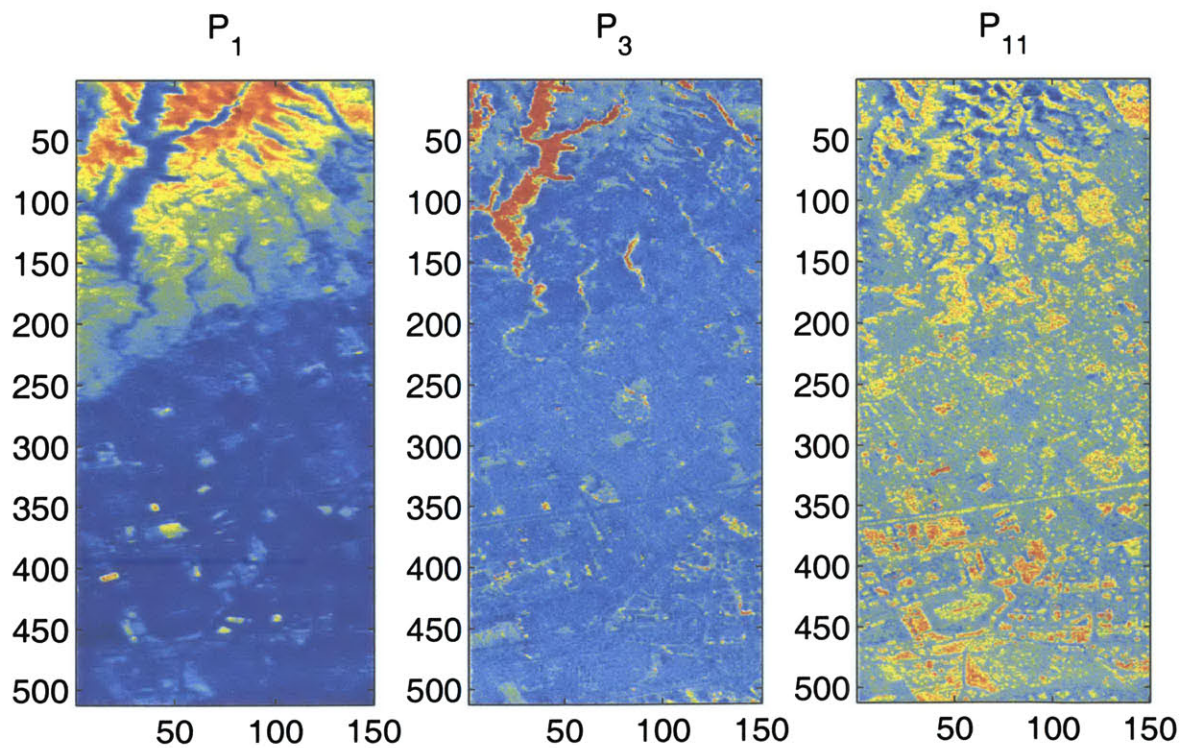
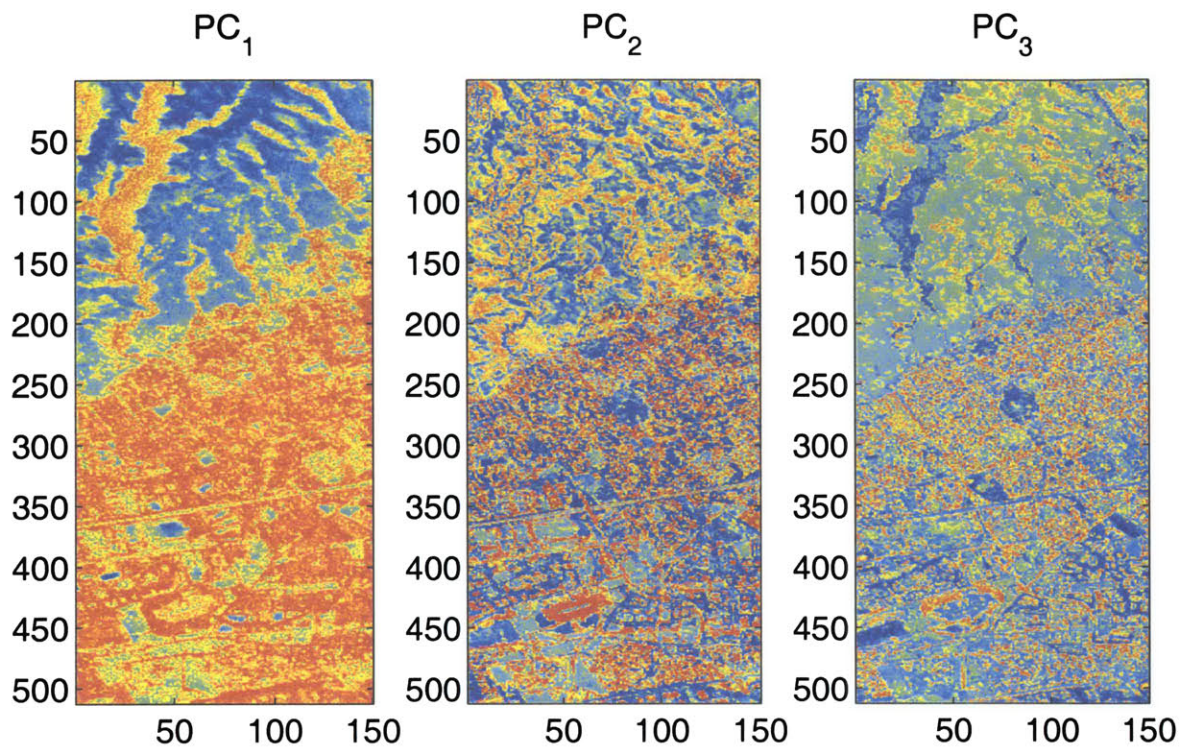


Figure 8-9: Comparison of first three principal components produced using PCA and three Independent Sources ( $P_i$ ) retrieved by ONA for Moffett Field sub-image



# Chapter 9

## Conclusions and Suggested Further Work

### 9.1 Conclusions

Both the ION' and ONA algorithm successfully blindly separate the signal from the noise given a single data matrix. Also, the addition of SOBI in ONA produces a final algorithm which can additionally unmix the hidden sources  $\mathbf{P}$  and the mixing matrix  $\mathbf{A}$  from the noise-filtered signal  $\mathbf{Z}$ . Good results are obtained for both Gaussian and non-Gaussian hidden sources  $\mathbf{P}$ ; also, it was shown that these sources need not have a specific time structure for successful separation to take place.

The problem space over which ION' and ONA produce reasonable results was mapped for purely Gaussian sources. It was demonstrated that there exists a regime where ONA can unmix even a set of purely Gaussian signals. The remaining area of the high-dimensionality problem space was mapped along each of the axes to indicate the areas where data is well-suited to ONA.

ONA was shown to be successful at reducing the noise, retrieving small sets of correlated variables, and retrieving non-Gaussian sources for the example set of factory data. When run on the same data which was time-scrambled to remove any time correlation, ONA and ION' produce results in line with those produced on the sequenced data. This indicates that the ION' portion of the algorithm does not assume

or exploit time correlation and that it is only recognized by the SOBI algorithm.

The joint diagonalization technique of SOBI was extended to take advantage of spatial correlation; this was proved successful by application of this 2D version of ONA to a sample hyperspectral data set. Here, the noise levels in the system were significantly reduced, and the retrieved sources were demonstrated to group smaller sets of features than did the sources produced by PCA. The results presented in Chapter 8 confirm the applicability of ONA to hyperspectral data in general, and the adaptability of the algorithm to new types of data.

## 9.2 Future Work

### 9.2.1 Improvements to the Iteration Determination

#### Algorithm

The iteration determination algorithm discussed in Section 4.4 can clearly be optimized differently depending upon the results desired. It was indicated that the choice of  $(c, d)$  determines those output parameters for which the algorithm is optimized. Further investigation into the optimal choice of  $(c, d)$  for each of  $\mathbf{Z}$ ,  $\mathbf{G}$ , etc. may enable even better results from ONA. It is possible that this algorithm could be run several times, once for each parameter to be estimated, producing fully optimal solutions for each. The results produced in this case will not be fully consistent with Equation 2.1, but for some applications, this may not be a problem.

### 9.2.2 Hyperspectral Applications

The array of problems associated with hyperspectral data is large, and ONA may be used to address some of them. The most obvious application would be to use ONA to identify surface types in hyperspectral images. Because ONA retrieves  $\mathbf{P}_i$ 's which are assumed independent, it may be possible to rotate these sources to a new basis in which each corresponds to a different surface type. If this is possible, it would eliminate the need for the time- and computation-intensive clustering algorithms which

perform this task currently.

It would also be revealing to compare the noise estimates of ONA to a priori noise levels for a given instrument. The amount by which the estimates and measurements differ will indicate the extent to which ONA has identified noise in the data (feature noise) in addition to noise in the measurements (instrument noise), allowing a better evaluation of the algorithm performance.

### 9.2.3 Application to Varied Data

The ability of ONA to identify and remove noise in the blind data setting makes it an obvious choice for application to data sets for which little knowledge exists which characterizes the latent sources  $\mathbf{P}$ . Data recorded from a suite of medical instruments during tests or during surgery could be analyzed. Data collected from a large number of patients for tests of new treatments or drugs could be filtered. The sets of variables identified by ION' and ONA may lend insight into the correlated functions of the human body in any of these cases.

### 9.2.4 Improvements and Extensions to the Algorithm

Because the algorithm was developed generically to be applicable to all blind data, much improvement can be gained by tailoring it for work on specific data types. The extension of the joint diagonalization algorithm to take advantage of spatial correlation to improve separation is one example of this. Allowing the user to specify any constraints which exist on the set of sources  $\mathbf{P}$  to exploit this extra knowledge if it exists would further improve results. This could involve the development or addition of other separation algorithms in place of SOBI; ONA would then determine at the outset which separation algorithm would be most appropriate for the data, given the a priori knowledge available. A similar multi-option setup could be developed for the order estimation portion of the algorithm. Should prior knowledge exist about the value of or statistics of the system order, use of this may also significantly improve the results attainable.



# Appendix A

## Matlab Code

### A.1 Useful Scripts

#### my\_leftdivide.m

```
function result = my_leftdivide(A,B)

% does
%   result = A \ B;
% more quickly and efficiently for the case where

%  $A \setminus B = INV(A)*B$ 

% A is a diagonal matrix, INV(A) is diagonal with elements which are
% the inverse of the elements in A                                     10

[ar ac] = size(A);
[br bc] = size(B);

if ((ar ~= ac) | (ac ~= br))
    warning('matrix dimensions do not match');
    return;
end;

result = zeros(br,bc); % same size as B
for l=1:br,
    result(l,:) = B(l,:) ./ A(l,l);
end;

return;                                                                20
```

#### my\_normalizer2.m

```
function [x] = my_normalizer2(x)

% memory-efficient normalization

% normalizes a matrix of column vectors:
```

```

%  $x_n = (x - \text{mean}_x) / \text{std}_x$ 
% where  $\text{mean}_x$ ,  $\text{std}_x$  are found for each column
%
%  $x$  :  $M \times N$ 
%  $\text{mean}_x$  :  $1 \times N$ 
%  $\text{std}_x$  :  $1 \times N$ 
%  $x_n$  :  $M \times N$ , normalized  $x$ 
10

if (size(x,1) == 1), % std dev will be zero
    x = NaN;
    warning('X cannot be normalized, unit dimension');
    return;
end;

[xr xc] = size(x);
for i=1:xc,
    v = x(:,i);
    m = mean(v);
    s = std(v);
    if (s~=0),
        v = v - mean(v);
        v = v/s;
    end;
    x(:,i) = v;
end;
20
30
30

```

## A.2 ION/ION'/ONA Shared Scripts

### A.2.1 Unaltered ION Scripts

The following functions are used in ION' and ONA but are unaltered from the original ION code written by Junehee Lee in [10].

#### lr1.m

```

function [a,b,R_SQUARED]=lr1(x,Y)
%
% LR1    [a,b,R_squared]=lr1(x,Y)
%        Multi-dimensional linear regression
%        x is a m by 1 vector
%        Y is a m by n matrix
%        a and b is ther coefficient which fits the data points on
%
%         $x = a + Y*b$ 
%
%        If Y and x is not normalized, this command will NOT normalize
%        them before the regression (as opposed to LR).
%
%        R_squared is the variability of x explained by Y
%        (in terms of percentage)
%
%
%        (c) Copyright 2000 M.I.T.
%
%        Permission is hereby granted, without written agreement or
%        royalty fee, for Hewlett Packard Corporation (HP) to use, copy,
10
20

```

```

%      modify, and distribute within HP this software and its
%      documentation for any purpose, provided that the above copyright
%      notice and the following three paragraphs appear in all copies of
%      this software.
%
%      In no event shall M.I.T. be liable to any party for direct,
%      indirect, special, incidental, or consequential damages arising
%      out of the use of this software and its documentation, even if
%      M.I.T. has been advised of the possibility of such damage.
%
%      M.I.T. specifically disclaims any warranties including, but not
%      limited to, the implied warranties of merchantability, fitness
%      for a particular purpose, and non-infringement.
%
%      The software is provided on an "as is" basis and M.I.T. has no
%      obligation to provide maintenance, support, updates, enhancements,
%      or modifications.

```

```

mY=mean(Y);
mx=mean(x);
new_Y=Y-ones(size(Y,1),1)*mY;
new_x=x-mx;
b=regress1(new_x,new_Y);
a=ones(size(x,1),1)*mx-mY*b;

Sxx=(x-mean(x))'*(x-mean(x));
Rss=(x-a-Y*b)'*(x-a-Y*b);
R_SQUARED=(Sxx-Rss)/Sxx*100;

```

## regress1.m

```

function b= regress1(y,X,alpha)
% REGRESS1 Performs multiple linear regression using least squares.
% b = REGRESS1(y,X) returns the vector of regression coefficients, B.
% Given the linear model: y = Xb,
% (X is an n x p matrix, y is the n x 1 vector of observations.)
%
% References:
% [1] Samprit Chatterjee and Ali S. Hadi, "Influential Observations,
% High Leverage Points, and Outliers in Linear Regression",
% Statistical Science 1986 Vol. 1 No. 3 pp. 379-416.
% [2] N. Draper and H. Smith, "Applied Regression Analysis, Second
% Edition", Wiley, 1981.
%
% B.A. Jones 3-04-93
% Copyright (c) 1993 by The MathWorks, Inc.
% $Revision: 1.4 $ $Date: 1993/10/04 12:26:29 $

if nargin < 2,
    error('REGRESS requires at least two input arguments.');
```

```

end

% Check that matrix (X) and left hand side (y) have compatible dimensions
[n,p] = size(X);
[n1,collhs] = size(y);
if n~=n1,
    error('The number of rows in Y must equal the number of rows in X.');
```

```

end

if collhs ~= 1,
    error('Y must be a vector, not a matrix');
```





## soebs.m

```
function [slope, intersect] = soebs(eigenvalues, a, b)
%
%   SOEBS [slope, intersect] = soebs(eigenvalues, a, b)
%   Signal Order Estimation by Scree plot
%   EIGENVALUES are the eigenvalues of the
%   covariance matrix of data in descending order.
%   SLOPE and INTERSECT are the slope and intesect of
%   the linear line which fits best the noise eigenvalues
%   when the scree plot is in logarithmic y-axis.
%   a, b control screeplot analysis.
%
%
%   (c) Copyright 2000 M.I.T.
%
%   Permission is hereby granted, without written agreement or
%   royalty fee, for Hewlett Packard Corporation (HP) to use, copy,
%   modify, and distribute within HP this software and its
%   documentation for any purpose, provided that the above copyright
%   notice and the following three paragraphs appear in all copies of
%   this software.
%
%   In no event shall M.I.T. be liable to any party for direct,
%   indirect, special, incidental, or consequential damages arising
%   out of the use of this software and its documentation, even if
%   M.I.T. has been advised of the possibility of such damage.
%
%   M.I.T. specifically disclaims any warranties including, but not
%   limited to, the implied warranties of merchantability, fitness
%   for a particular purpose, and non-infringement.
%
%   The software is provided on an "as is" basis and M.I.T. has no
%   obligation to provide maintenance, support, updates, enhancements,
%   or modifications.
%
if ( nargin==1),
    a = .4;
    b = .6;
end

n = length(eigenvalues);
logeig = log10(eigenvalues);

% ALSO SEE screeorder.m FOR DETAILS OF PARAMETERS a,b.
begins = floor(n * a);
ends = ceil(n * b);

[intersect, slope, foo] = lr1(logeig(begins:ends),[begins:ends]');
intersect = intersect(1);
```

### A.2.2 Updated ION Scripts

The following scripts were originally written by Junehee Lee ([10]) and revised by A. Mueller.

## fixeig.m

```
function [evect,evals] = fixeig(evect,evals)

%     FIXEIG [NewEvect, NewEvals] = fixeig(EVECT, EVALS)
%
%     EVALS is a vector of eigenvalues.
%     EVECT is a matrix whose column is the eigenvectors.
%     The first column of EVECT is the eigenvector corresponding
%     to the first element of EVALS.
%
%     NewEvals is the eigenvalue in descending order.
%     NewEvect is the re-ordered eigenvector matrix.
%
%
%     (c) Copyright 2000 M.I.T.
%
%     Permission is hereby granted, without written agreement or
%     royalty fee, for Hewlett Packard Corporation (HP) to use, copy,
%     modify, and distribute within HP this software and its
%     documentation for any purpose, provided that the above copyright
%     notice and the following three paragraphs appear in all copies of
%     this software.
%
%     In no event shall M.I.T. be liable to any party for direct,
%     indirect, special, incidental, or consequential damages arising
%     out of the use of this software and its documentation, even if
%     M.I.T. has been advised of the possibility of such damage.
%
%     M.I.T. specifically disclaims any warranties including, but not
%     limited to, the implied warranties of merchantability, fitness
%     for a particular purpose, and non-infringement.
%
%     The software is provided on an "as is" basis and M.I.T. has no
%     obligation to provide maintenance, support, updates, enhancements,
%     or modifications.

evals = diag(evals);
evals_temp = evals;

dim = size(evals);
for i = 1:dim(1)
    mx = max(evals);
    for j = 1:dim(1)
        if mx == evals(j,1)
            loc = j;
        end
    end
    evals_temp(i) = mx;
    evect_temp(:,i) = evect(:,loc);
    evals(loc,1) = (-1) * (abs(evals(loc,1)));
end
evals = evals_temp;
% if evals are below acceptable machine precision threshold
broken = find(abs(evals)<(10^-13));
minvalue = evals(min(broken)-1);
evals(broken) = minvalue;
evect = evect_temp;
```

## nebema.m

```
function [S, estimated_signal, A_est, p_est] = ...
    nebema(x, p, it, prevA)

% NEBEMA Noise Estimation through EM algorithm
%
%      x is an m-by-n data matrix. m represents the number of
%      observations, and n represents the number of variables.
%
%      p is the number of latent variables.                                     10
%
%      'it' represents how many iterations will be performed for EM.
%
%      S is the estimated noise variances for each variables.
%
%      prevA is used to initialize the estimate of A

[m, n] = size(x);
thresh = 1*10^(-6);
if (nargin==3),                                     20
    % randomly initialize A
    A_est = randn(n,p);
elseif(nargin==4),
    A_est = prevA;
end;
% randomly initialize G
G_est = diag(0.5*ones(n,1));

brokeninit = it+1;
S = [ ];                                          30
for repeat = 1 : it
    % first check that G is non-singular.
    % if it is not, fix that so that we don't break EM trying to do
    % pinv()
    if (rcond(G_est) < thresh),
        warning(strcat('singular G',num2str(rcond(G_est))));
        G_est = makeNonSingular(G_est,thresh);
    end;

    % E - STEP                                          40

    %AldG = (G_est \ A_est);
    % made more memory (RAM) efficient
    AldG = my_leftdivide(G_est,A_est);

    term1 = x * AldG;
    term2 = (A_est' * AldG + eye(p,p));
    if (rcond(term2) < 10^-10),
        brokeninit = repeat;
        warning(strcat('singular term2: ',num2str(rcond(term2))));
        break;                                          50
    end;

    Exp_lv = term1 / term2;
    clear term1;

    t1 = m * eye(p,p);
    t2 = (A_est' * pinv(G_est) * A_est + eye(p,p));
    t3 = (term2 \ A_est');
    %t4 = (G_est \ x');
    % made more memory (RAM) efficient
    t4 = my_leftdivide(G_est,x');
    t5 = AldG / term2;
    clear AldG term2;                                          60
end;
```

```

t55 = t3 * t4;
clear t3 t4;
t56 = x * t5;
clear t5;

t6 = t55 * t56;
clear t55 t56;
Exp_lv2 = t1/t2 + t6;
clear t6 t1 t2;

% M - STEP

A_est = (x'* Exp_lv) / Exp_lv2; %***
G_est = [ ];

for index_j = 1:n
    t1 = x(:,index_j)' * x(:,index_j);
    t2 = A_est(index_j,:) * Exp_lv' * x(:,index_j);
    G_est_j = (t1 - t2) / m;
    G_est = [G_est; G_est_j];
end

G_est = diag(G_est);
end

if (rcond(G_est) < thresh),
    warning(strcat('singular G',num2str(rcond(G_est))));
    G_est = makeNonSingular(G_est,thresh);
end;
S = diag(G_est);
if (brokeninit==1),
    A_est = [ ];
    estimated_signal = [ ];
    p_est = [ ];
else
    estimated_signal = Exp_lv * A_est' ;
    p_est = Exp_lv';
end;

```

## A.2.3 ION'/ONA Scripts

### checkevals.m

```

function checkevals(evals, it)

    whichbum=fnd(evals<0);
    if (prod(size(whichbum))>0)
        warning(strcat('negative eigenvalues in iter ',num2str(it)));
        %evals(whichbum)
    end;

return;

```

### essential\_equality.m

```

function [newA,newP] = essential_equality(realA,estA,estP);

% realA is of size n x k.
% estA is of size n x k'.

```

```

% estP MUST BE of size k' x m, i.e. it must share a dimension with
% estA.

% Given the truth (realA) and an estimate of the truth, determine
% the transformation matrix which maximizes the covariance of
% corresponding columns of realA and estA (i.e. maximizes
% trace(cov(realA,estA)) where the covariance is taken over the
% columns, not the rows). In order to be consistent, estP must
% also be transformed so that the product (estA*estP) does not
% change.
%
% Simply post-multiply estA by the transformation matrix and
% pre-multiply estP by the inverse of the transformation matrix.
%
% If only estA, realA are passed in, do not output a P.
%
% make sure that estA, realA are the same size
k = size(realA,2); % true number of latent variables

[ar ac] = size(estA);
if (ac < k),
    estA = [estA zeros(ar,k-ac)];
elseif (ac > k),
    estA = estA(:,1:k);
end; % end if

if (nargin==3),
    if (ac < k),
        estP = [estP; zeros(k-ac,size(estP,2))];
    elseif (ac > k),
        estP = estP(1:k,:);
    end; % end if;
end; % end if;

% determine the covariance between the columns of the two A
% matrices.
crosscov = zeros(k,k);
for i=1:k,
    for j=1:k,
        t = cov(estA(:,i),realA(:,j));
        crosscov(i,j) = t(1,2); % could be t(2,1)-symmetric matrix
    end;
end;

% transformation matrix
trans = zeros(k,k);

% loop variables
temp = abs(crosscov);
iterations = 0;

% exit the loop when all columns of estA have been optimally
% rearranged. (all elements of temp will be zeroed.)
% [sum(sum(temp))=0]
% However, don't want to exit prematurely. Every row & column of
% the transformation matrix must have a non-zero element.
while (iterations<k),
    m = max(max(temp)); % find the columns with the most in common
    w=find(temp==m); % locate this element in the covariance matrix

    % multiple elements may have the same value
    if (sum(sum(temp)) > 0),
        % just use the first element
        e = mod(w(1),k); % row index of the element
        if (e==0),
            e = k; % because matrix indices start at 1
        end;
        d = ceil(w(1)/k); % column index of the element
    end;
end;

```

```

else
    % only have zeros left (the matrix passed in was most likely
    % the incorrect size).
    % Must find some element for which the corresponding row and
    % column of the transformation matrix are all zero.
    notfound = 1;
    e = mod(w,k);
    d = ceil(w/k); % column index of the element
    es = 1; % loop variable
    while (notfound),
        if (e(es)==0), e(es) = k; end;
        if ((sum(trans(e(es),:)) ~= 0) | (sum(trans(:,d(es))) ~= 0)),
            es = es+1;
        else
            notfound=0;
        end;
    end;
    e = e(es);
    d = d(es);
end;

if (temp(e,d) ~= crosscov(e,d)),
    % the signs are wrong
    trans(e,d) = -1;
else
    trans(e,d) = 1;
end; % end if
% zero the row and column just marked in the transformation
% matrix to make sure they are only included once.
temp(e,:) = zeros(1,k);
temp(:,d) = zeros(k,1);
iterations = iterations+1;
end; % end while

% set the outputs
newA = estA*trans;
if (nargout==1), return, end;
if (nargin==3)
    newP = (trans')*estP;
else
    newP = [];
end;

```

## goodIter.m

```

function [status, newG] = goodIter(Git, thresh);

% Make sure that IONPRIME/ONA doesn't produce singular results.
% Detects if G will produce singular/imaginary results.

% THRESH should be changed depending upon the level of protection
% wanted against possibly singular results. It is possible that
% matrices with rcond<thresh may not be singular if the threshold is
% set too high (and also that matrices with rcond>thresh may be
% singular if the threshold is set too low). Keep in mind that this
% matrix needs to be stable enough to be inverted, and it will also be
% multiplied by another matrix, the result of which should still be
% non-singular.

if (nargin == 1),
    thresh = 1*10^-10;
end;
machprecision = (10^-13); % agrees with FIXEIG

% G is assumed REAL - fails if has any imaginary elements

```

```

imags = prod(size(find(imag(Git)~=0)));
if (imags>0),
    status = 0; % fails
else, % all real elements
    % if G has negative elements which are larger than machine
    % precision off from zero, throw G away. Else take the
    % absolute value of G to get rid of any bogus elements.
    negs = max(abs(Git(find(Git<=0)))); % largest magnitude of a
    % negative element
    if (size(negs,1)~=0), % Git has negative elements
        if (negs>machprecision),
            status = 0; % fails
        else
            % take absolute value to get rid of negative elements
            status = 1; % passes so far
            Git = abs(Git);
        end;
    else
        status = 1; % passes so far
    end;
    % final thing to check: singular cases
    sing = rcond(diag(Git));
    % if rcond is too low, make sure that iteration will not be chosen.
    if (sing < thresh),
        status = 0; % fails
        %else, it passes
    end;
end;

newG = Git;
return;

```

## initializer.m

```

function [newA] = initializer(est_A, est_order);

% constructs an initialization matrix NEWA from ESTA, the prior
% estimate of A, to be passed into the EM algorithm. If ESTA is
% of size n x k, NEWA should be of size n x est_order.

ca = cov(est_A);
nca = abs(ca)./(ones(size(ca))*diag(max(abs(ca))));
% keep a column of A if its self-covariance is larger than its
% covariance with other columns of A
[arows, acols] = size(est_A);
if (acols <= est_order),
    for vr=1:acols,
        if (nca(vr,vr)==1.0),
            newA(:,vr) = est_A(:,vr);
        else
            newA(:,vr) = randn(arows,1);
        end;
    end;
    for vr=(acols+1):est_order,
        newA(:,vr) = randn(arows,1);
    end;
else
    next=1;
    for vr=1:acols,
        if (nca(vr,vr)==1.0),
            newA(:,next) = est_A(:,vr);
            next = next+1;
        end;
    end;
    [nar, nac] = size(newA);

```

```

    if (nac < est_order)
        newA = [newA randn(arows,est_order-nac)];
    else
        newA = newA(:,1:est_order);
    end;
end;
% normalize the power in each column...should all be about the same
if (rcond(newA'*newA)<10^-10),
    variances = var(newA);
    for q=1:est_order,
        newA(:,q) = (1/variances(q))*newA(:,q);
    end;
end;

```

## makeNonSingular.m

```

function [betterX] = makeNonSingular(X,cutoff)

% given a diagonal matrix X, if the matrix is near singularity
% (according to cutoff), the function will repair this to the point
% where rcond(betterX) > cutoff.

% This version does not use scree plots at all. It 'raises the
% water level' of the screeplot by determining how many eigenvalues
% need to be altered in order for rcond(X)>cutoff. After
% alterations, the last several eigenvalues will have identical
% values.

% setup
if (nargin==1)
    cutoff = 1*10^(-6); % arbitrary cutoff
end;
[xr xc] = size(X);
if (xr==1 | xc==1),
    % X is actually a vector. assume it's the diagonal
    X = diag(X);
end;

tempX = diag(X);
vals = -1*sort(-1*diag(X)); % sort decreasing
upperbound = max(find(vals > cutoff*max(tempX)))+1;

% want to bring everything to the right of upperbound up to some
% minimum acceptable level
betterX = diag(tempX);
i=0;
while (rcond(betterX) < cutoff),
    i=i+1
    valueatbound = vals(upperbound-(i-1));
    needtofix = find(tempX <= valueatbound);
    newvalue = vals(upperbound-i);
    tempX(needtofix) = newvalue;
    betterX = diag(tempX);
end;

```

## pc\_ord2.m

```

function ord_est = pc_ord2(Z,tonormalize)
% Input: Z = {(A*p)^i}
% Estimate the hidden size of A,p by analyzing the number of
% significant principal components.

```



```

if (nargin==1)
    tonormalize=0;
end;

% Find the principal components of the data.
if (tonormalize==1),
    [zn] = my_normalizer2(Z');
    %-----%
    % code modified from princomp.m %
    %-----%
    [m,n] = size(zn);
    r = min(m-1,n); % max possible rank of x
    s = svd(zn./sqrt(m-1),0);
    latent = s.^2;
    if (r<n)
        zvar = [latent(1:r); zeros(n-r,1)];
    end
    clear zn;
else
    [zpc, zjunk, zvar] = princomp(Z');
    clear zpc zjunk;
end;
percExpl = 100*zvar/sum(zvar); % percent of var explained by pc's

% find # of principal components explaining any variance
for i=1:size(percExpl)
    if (percExpl(i) < .001)
        noSigZPC = i-1;
        break; % exit for loop
    end; % end if
end; % end for

ord_est = noSigZPC;

```

## traceG3.m

```

function bestIter = traceG3(orders, Gs, ordparam, tracegparam);

% Given the data of maxiter iterations of IONPRIM/ONA, determines
% which iteration provided the best prediction of the data.

% ORDERS is a matrix of size 1 x MAXITER (a parameter passed to
% ONA). It contains the estimated order at each iteration.
%
% GS is a matrix of size NVAR x MAXITER, where NVAR=size(x,2)=n.
% It contains the diagonal of the G matrix at each iteration.
% Recall that G is the noise variance matrix, so its elements will
% always be greater than zero (and the G matrix is always diagonal).

if (nargin<2),
    warning('not enough inputs')
    return;
elseif (nargin==2),
    % defaults
    ordparam = 0.075;
    tracegparam = .02;
    combinedparam = 0.15;
elseif (nargin==3),
    tracegparam = .02;
    combinedparam = ordparam - tracegparam;
elseif (nargin==4),
    combinedparam = ordparam - tracegparam;
end;

nvar=size(Gs,1);

```

```

iterations=size(Gs,2);
for it=1:iterations,
    traces(it) = sum(Gs(:,it));
end; % end for

alldata = [ traces' orders' [1:iterations]' ];
% sort them by traces; make sure to keep track of the original
% order in order to return this information.
sorteddata = -1*sortrows(-1*alldata); % sort descending
sorttraces = sorteddata(:,1);
sortorders = sorteddata(:,2);
sortiters = sorteddata(:,3);

% ----- %
% check for BOGUS iterations %
% ----- %
% check orders and trace(G)'s.
% find drastic changes in the highest iterations
maxTraceG = sorttraces(1);
% first check for next-largest of the same order
ofthatorder = sorttraces(find(sortorders==sortorders(1)));
if (size(ofthatorder,1) > 1);
    % first ofthatorder is always maxTraceG
    % if it has more than one element, take the second.
    nextTraceG = ofthatorder(2);
else
    % no other iterations with same order estimation; compare to the
    % next estimation
    nextTraceG = sorttraces(2);
end;

percentIncrease = (maxTraceG - nextTraceG) / nextTraceG;
maxTraceGorder = sortorders(1);
nextOrder = sortorders(2);
% if order is increasing, this will be positive
percOrderIncrease = (nextOrder-maxTraceGorder) / nextOrder;
% if order increases by a lot and traceG decreases by a lot, that
% iteration is likely bogus.
while ((percentIncrease > tracegparam) | ...
        (percOrderIncrease > ordparam) | ...
        (percOrderIncrease+percentIncrease > combinedparam)),
    % the iteration at the top of sorteddata is BAD
    % throw away the top iteration
    sorteddata = sorteddata(2:end,:);
    sorttraces = sorteddata(:,1);
    sortorders = sorteddata(:,2);
    sortiters = sorteddata(:,3);

    % do another check
    maxTraceG = sorttraces(1);
    ofthatorder = sorttraces(find(sortorders==sortorders(1)));
    if (size(ofthatorder,1) > 1),
        nextTraceG = ofthatorder(2);
    else
        if (size(sorteddata,1) > 1),
            nextTraceG = sorttraces(2);
        else
            nextTraceG = sorttraces(1);
        end;
    end;

percentIncrease = (maxTraceG - nextTraceG) / nextTraceG;
maxTraceGorder = sortorders(1);
if (size(sortorders,1) > 1),
    nextOrder = sortorders(2);
else
    nextOrder = sortorders(1);
end;
percOrderIncrease = (nextOrder-maxTraceGorder) / nextOrder;

```

```
end;
```

```
% now take the first row of sorteddata as best  
bestIter = sortiters(1);
```

100

## A.3 ION'

The ION' code was revised by A. Mueller from the original ION code authored by Junehee Lee [10].

### ionprime.m

```
function [S, noise, order] = ionprime(x, maxiter, a, b, p1, p2)  
  
% This version of ION, called ION', has improvements made in order  
% estimation, in initialization of the EM algorithm, in iteration  
% choosing. It does NOT estimate A or P. It does NOT use the SOBI  
% algorithm in any way.  
  
% Original code by Junehee Lee, updated by A. Mueller  
  
% IONPRIME Iterative Order, Noise estimation algorithm 10  
%  
% X is an m-by-n data matrix. m represents the number of  
% observations, and n represents the number of variables.  
%  $X = (A*P)' + (G^{-.5} * W)'$   
%  
% MAXITER is the desired max number of iterations of the ION  
% algorithm. We have found that MAXITER = 10 is generally more  
% than sufficient.  
%  
% S (n-by-1 vector) is the noise variance vector. 20  
% NOISE (m-by-n matrix) is the retrieved noise sequence(s).  
% ORDER is the estimated number of independent signals.  
%  
  
if (nargin==0), return;  
elseif (nargin==1), maxiter=10;  
end;  
if (nargin<4),  
    a = .4;  
    b = .6; 30  
end;  
if (nargin<6),  
    p1 = 0.15;  
    p2 = 0.02;  
end;  
thresh = 1*10-(10);  
EMiters = 10;% number of iterations of EM algorithm we want.  
randfileID = strcat('/usr/dicke2/ionprimetemp',...  
    num2str(round(rand(1,1)*1000))); 40  
  
% initialize variable used in the loop  
%-----  
[Nobs, Nvar] = size(x);  
G = ones(Nvar,1); S = ones(Nvar,1);  
  
allEstSignal = []; allS = zeros(Nvar,maxiter);  
save(randfileID,'allEstSignal','allS');
```

```

est_order = 0;
PCtypicallyToobig = 0;
PCtypicallyToosmall = 0;
ignorePC = 0;

nebemstatus = 1;
%-----

for it = 1 : maxiter
    xn = x*diag(S,~-5);
    Sx = cov(xn);
    [evec, evals] = eig(Sx);
    [evec, evals] = fixeig(evec, evals);
    checkevals(evals, it);

    screeorder = screeorder(evals,a,b);

    % Use the estimated signal and PCA to improve the signal order estimate.
    if (~ignorePC)
        if exist('est_signal'),
            pcordest = pc_ord2(est_signal* diag(sqrt(S_old)),1);
        else,
            pcordest = NaN;
        end; % end if
        % should work for any data sets, whether ION
        % overestimates or underestimates.
        if (it>1) % first iterations are always poor
            if ((pcordest < screeorder) & ~PCtypicallyToobig)
                PCtypicallyToosmall = 1;
            elseif ((pcordest > screeorder) & ~PCtypicallyToosmall)
                PCtypicallyToobig = 1;
            end; % end if
        end; % end if
        est_order_old = est_order;

        if ((it==5) & ~PCtypicallyToobig & ~PCtypicallyToosmall)
            % PC method generally agrees with ION, don't use it
            ignorePC = 1;
        end;
        if (~ignorePC),
            if (PCtypicallyToobig & (screeorder > pcordest)) | ...
                (PCtypicallyToosmall & (screeorder < pcordest)),
                warning('using pc estimate');
                est_order = pcordest;
            else
                est_order = screeorder;
            end; % end if
        else
            est_order = screeorder;
        end; % end if

        % Now that we have chosen an order, we need to construct
        % a prior A to pass in to the EM algorithm.
        if (it==1), % don't have prior estimates
            [G,est_signal,est_A]=nebema(xn, est_order, EMiters);
        else
            newA = initializer(est_A, est_order);
            [G, est_signal, est_A] = nebema(xn, est_order, EMiters, newA);
            clear newA;
            if (isempty(est_signal))
                nebemstatus = 0;
            end;
        end;

    S_old = S;

```

```

S = S.* G;

[status, newS] = goodIter(S,thresh);
S = newS;
clear newS;
if (status == 0 | nebemstatus==0),
    warning(strcat('status = 0 in iteration ',num2str(it)));
    maxiter = it-1; % how many iters there were
    % quit looping, LEAVE 'for' loop.
    break;
end;

if (rcond(diag(S))<thresh)
    warning(cat(2,'singular noise matrix in iterations: ', ...
        num2str(rcond(diag(S))))))
end;

% make sure that bogus runs don't get chosen:
if (isempty(est_signal)) allOrder(it) = Nvar;
else
    allOrder(it) = est_order;
end;

% Save all of the information from this iteration
load(randfileID,'allS');
allS(:,it) = S;
save(randfileID,'allS','-append');
clear allS;

load(randfileID,'allEstSignal');
allEstSignal(:,it) = est_signal* diag(sqrt(S_old));
save(randfileID,'allEstSignal','-append');
clear allEstSignal;
end; % end for

% get the best iteration's predictions
if (max(size(allOrder))>1),
    load(randfileID,'allS');
    % takes care of cases where the loop is exited early due to
    % status=0 and allS, allEstSignal are not saved for that iter.
    bestiter = traceG3(allOrder(1:maxiter), allS(:,1:maxiter), p1, p2)
else,
    bestiter=1;
end;

est_order = allOrder(bestiter);
load(randfileID,'allEstSignal');
est_signal = allEstSignal(:,bestiter);
clear allEstSignal;
S = allS(:,bestiter);
clear allS;

if (rcond(diag(S)) < thresh),
    % This should NEVER happen because all singular S matrices should
    % have been caught by GOODITER.m in the loop above.
    warning(cat(2,'singular noise matrix at end: ', ...
        num2str(rcond(diag(S))))))
end;

% other outputs %
noise = x - est_signal;
order = est_order;
% uncomment this line if you want to forcefully assume unit variance w
%S = (std(noise)).^2; % or S = var(noise);

% GET RID OF THE TEMP FILE SO AS TO FREE UP HARDDRIVE SPACE
unix(['rm ' randfileID '.mat']);

return;

```

## A.4 ONA

The ONA code was revised by A. Mueller from the original ION code authored by Junehee Lee [10].

### ona.m

```
function [S, noise, order, est_P, est_A] = ...
    ona(x, maxiter, a, b, p1, p2)

% IONPRIME with improvements made to estimate A (mixing matrix)
% and P (latent signals) with SOBI.

if (nargin==0), return;
elseif (nargin==1), maxiter=10;
end;
if (nargin<4),
    a = .4;
    b = .6;
end;
if (nargin<6),
    p1 = 0.075;
    p2 = 0.02;
end;
thresh = 1*10^(-10);
EMiters = 10;% number of iterations of EM algorithm we want.
randfileID = strcat('/usr/dicke2/onatemp', ...
    num2str(round(rand(1,1)*1000)));
randfileID1 = strcat(randfileID,'1');
randfileID2 = strcat(randfileID,'2');

% initialize variable used in the loop
%-----
[Nobs, Nvar] = size(x);
G = ones(Nvar,1); S = ones(Nvar,1);

allEstSignal = []; allS = zeros(Nvar,maxiter);
save(randfileID1,'allEstSignal');
save(randfileID2,'allS');

est_order = 0;
PCtypicallyToobig = 0;
PCtypicallyToosmall = 0;
ignorePC = 0;

nebemstatus = 1;
%-----

for it = 1 : maxiter
    xn = x*diag(S.^-5);
    Sx = cov(xn);
    [evect, evals] = eig(Sx);
    [evect, evals] = fixeig(evect, evals);
    checkevals(evals, it);

    screeorderdest = screeorder(evals,a,b);

% Use the estimated signal and PCA to improve the signal order estimate.
if (~ignorePC)
    if exist('est_signal'),
        pcordest = pc_ord2(est_signal* diag(sqrt(S_old)),1);
    else,
```

```

    pcordest = NaN;
end; % end if
% should work for any data sets, whether ION
% overestimates or underestimates.
if (it>1) % first iterations are always pretty poor
    if ((pcordest < screeordest) & ~PCtypicallyToobig)
        PCtypicallyToosmall = 1;
    elseif ((pcordest > screeordest) & ~PCtypicallyToosmall)
        PCtypicallyToobig = 1;
    end; % end if
end; % end if
end; % end if
est_order_old = est_order;

if ((it==5) & ~PCtypicallyToobig & ~PCtypicallyToosmall)
    % PC method generally agrees with ION, don't use it
    ignorePC = 1;
end;
if (~ignorePC),
    if (PCtypicallyToobig & (screeordest > pcordest)) | ...
        (PCtypicallyToosmall & (screeordest < pcordest)),
        warning('using pc estimate');
        est_order = pcordest;
    else
        est_order = screeordest;
    end; % end if
else
    est_order = screeordest;
end; % end if

% Now that we have chosen an order, we need to construct
% a prior A to pass in to the EM algorithm.
if (it==1), % don't have prior estimates
    [G,est_signal,est_A]=nebema(xn, est_order, EMiters);
else
    newA = initializer(est_A, est_order);
    [G, est_signal, est_A] = nebema(xn, est_order, EMiters, newA);
    clear newA;
    if (isempty(est_signal))
        nebemstatus = 0
    end;
end;

S_old = S;
S = S.* G;

[status, newS] = goodIter(S,thresh);
S = newS;
clear newS;
if (status == 0 | nebemstatus==0),
    warning(strcat('status = 0 in iteration ',num2str(it)));
    maxiter = it-1; % how many iters there were
    % quit looping over ONA, LEAVE 'for' loop.
    break;
end;

if (rcond(diag(S))<thresh)
    warning(cat(2,'singular noise matrix in iterations: ',...
        num2str(rcond(diag(S))))))
end;

% make sure that bogus runs don't get chosen:
if (isempty(est_signal))
    allOrder(it) = Nvar;
else
    allOrder(it) = est_order;
end;

```

```

    % Save all of the information from this iteration
    load(randfileID2,'allS');
    allS(:,it) = S;
    save(randfileID2,'allS');
    clear allS;

    load(randfileID1,'allEstSignal');
    allEstSignal(:,it) = est_signal* diag(sqrt(S_old));
    save(randfileID1,'allEstSignal');
    clear allEstSignal;
end; % end for

% get the best iteration's predictions
if (max(size(allOrder))>1),
    load(randfileID2,'allS');
    % takes care of cases where the loop is exited early due to
    % status=0 and allS, allEstSignal are not saved for that iter.
    bestiter = traceG3(allOrder(1:maxiter), allS(:,1:maxiter), p1, p2)
else,
    bestiter=1;
end;

est_order = allOrder(bestiter);
load(randfileID1,'allEstSignal');
est_signal = allEstSignal(:,bestiter);
clear allEstSignal;
S = allS(:,bestiter);
clear allS;

if (rcond(diag(S)) < thresh),
    % This should NEVER happen because all singular S matrices should
    % have been caught by GOODITER.m in the loop above.
    warning(cat(2,'singular noise matrix at end: ', ...
        num2str(rcond(diag(S))))))
end;

[est_A1] = sobi(est_signal',est_order);
est_P = pinv(est_A1)*(est_signal');

% normalize the results so that P has unit variance.
std_p = std(est_P');
est_P = diag(std_p.^(-1))*(est_P); % normalize P
est_A = est_A1 * diag(std_p); % move power to A

% other outputs %
noise = x - est_signal;
order = est_order;
%S = var(noise);

% GET RID OF THE TEMP FILES SO AS TO FREE UP HARDDRIVE SPACE
unix(['rm ' randfileID1 '.mat']);
unix(['rm ' randfileID2 '.mat']);

```

## ona\_spatial.m

```

function [S, noise, order, est_P, est_A] = ...
    ona_spatial(x, maxiter, a, b, p1, p2,q,r,s)

% ONA algorithm altered to work with 2D spatial data

if (nargin==0),    return;
elseif (nargin==1), maxiter=10;
end;
if (nargin<4),

```



```

a = .4;
b = .6;
end;
if (nargin<6),
    p1 = 0.075;
    p2 = 0.02;
end;
thresh = 1*10(-10);
EMiters = 10;% number of iterations of EM algorithm we want.
randfileID = strcat('/usr/dicke2/onatemp',...
                    num2str(round(rand(1,1)*1000)));
randfileID1 = strcat(randfileID,'1');
randfileID2 = strcat(randfileID,'2');

% initialize variable used in the loop
%-----
[Nobs, Nvar] = size(x);
G = ones(Nvar,1); S = ones(Nvar,1);

allEstSignal = [];%zeros(Nobs,Nvar,maxiter);
save(randfileID1,'allEstSignal');
allS = zeros(Nvar,maxiter);
save(randfileID2,'allS');

est_order = 0;
PCtypicallyToobig = 0;
PCtypicallyToosmall = 0;
ignorePC = 0;

nebemstatus = 1;
%-----

for it = 1 : maxiter
    xn = x*diag(S.^-5);
    Sx = cov(xn);
    [evect, evals] = eig(Sx);
    [evect, evals] = fixeig(evect, evals);
    checkevals(evals, it);

    screeorderdest = screeorder(evals,a,b);

    % Use the estimated signal and PCA to improve the signal order estimate.
    if (~ignorePC)
        if exist('est_signal'),
            pcordest = pc_ord2(est_signal* diag(sqrt(S_old)),1);
        else,
            pcordest = NaN;
        end; % end if
        % should work for any data sets, whether ION
        % overestimates or underestimates.
        if (it>1) % first iterations are always pretty poor
            if ((pcordest < screeorderdest) & ~PCtypicallyToobig)
                PCtypicallyToosmall = 1;
            elseif ((pcordest > screeorderdest) & ~PCtypicallyToosmall)
                PCtypicallyToobig = 1;
            end; % end if
        end; % end if
    end; % end if
    est_order_old = est_order;

    if ((it==5) & ~PCtypicallyToobig & ~PCtypicallyToosmall)
        % PC method generally agrees with ION, don't use it
        ignorePC = 1;
    end;
    if (~ignorePC),
        if (PCtypicallyToobig & (screeorderdest > pcordest)) | ...
            (PCtypicallyToosmall & (screeorderdest < pcordest)),
            warning('using pc estimate');
        end;
    end;
end;

```

```

    est_order = pcordest;
    else
    est_order = screeordest;
    end; % end if
else
    est_order = screeordest;
end; % end if

% Now that we have chosen an order, we need to construct
% a prior A to pass in to the EM algorithm.
if (it==1), % don't have prior estimates
    [G,est_signal,est_A]=nebema(xn, est_order, EMiters);
else
    newA = initializer(est_A, est_order);
    [G, est_signal, est_A] = nebema(xn, est_order, EMiters, newA);
    clear newA;
    if (isempty(est_signal))
        nebemstatus = 0;
    end;
end;

S_old = S;
S = S.* G;

[status, newS] = goodIter(S,thresh);
S = newS;
clear newS;
if (status == 0 | nebemstatus==0),
    warning(strcat('status = 0 in iteration ',num2str(it)));
    maxiter = it-1; % how many iters there were
    % quit looping over ONA, LEAVE 'for' loop.
    break;
end;

if (rcond(diag(S))<thresh)
    warning(cat(2,'singular noise matrix in iterations: ', ...
        num2str(rcond(diag(S))))))
end;

% make sure that bogus runs don't get chosen:
if (isempty(est_signal))
    allOrder(it) = Nvar;
else
    allOrder(it) = est_order;
end;

% Save all of the information from this iteration
load(randfileID2,'allS');
allS(:,it) = S;
save(randfileID2,'allS');
clear allS;

load(randfileID1,'allEstSignal');
allEstSignal(:,it) = est_signal* diag(sqrt(S_old));
save(randfileID1,'allEstSignal');
clear allEstSignal;
end; % end for

% get the best iteration's predictions
if (max(size(allOrder))>1),
    load(randfileID2,'allS');
    bestiter = traceG3(allOrder(1:maxiter), allS(:,1:maxiter), p1, p2)
else,
    bestiter=1;
end;

est_order = allOrder(bestiter);
load(randfileID1,'allEstSignal');

```

```

est_signal = allEstSignal(:,bestiter);
clear allEstSignal;
S = allS(:,bestiter);
clear allS;

if (rcond(diag(S)) < thresh),
    % This should NEVER happen because all singular S matrices should
    % have been caught by GOODITER.m in the loop above.
    warning(cat(2,'singular noise matrix at end: ', ...
        num2str(rcond(diag(S)))));
end;

es = reshape(reshape(est_signal',1,q*r*s),q,r,s);
es = shiftdim(es,1); % so it is spatial x spatial x spectral
est_A1 = sobi_spatial(es,est_order);
est_P = pinv(est_A1)*(est_signal');

% normalize the results so that P has unit variance.
std_p = std(est_P');
est_P = diag(std_p.^(-1))*(est_P); % normalize P
est_A = est_A1 * diag(std_p); % move power to A

% other outputs %
noise = x - est_signal;
order = est_order;
% uncomment this line if you want to forcefully assume unit variance w
%S = var(noise);

% GET RID OF THE TEMP FILE SO AS TO FREE UP HARDDRIVE SPACE
unix(['rm ' randfileID1 '.mat']);
unix(['rm ' randfileID2 '.mat']);

```

## A.4.1 SOBI

The following code was authored by William J. Blackwell and revised by A. Mueller to implement the SOBI algorithm described in [4].

### sobi.m

```

function [A_hat, P_hat] = sobi(x,k);

% Original code by William J. Blackwell, revised by A. Mueller

% x is the matrix we wish to separate into A,p
% assumingn x = A*p+noise
% if x is of size nxm (where each row is a variable, each
% column is a time sample), then A is of size n by k

T = length(x);

% calculate 2nd-order stats for a range of time lags
x_ = x - (mean(x')' * ones(1,T));
R_0 = x_ * x_' / (T - 1);
R_1 = x_(:,1:(end-1)) * x_(:,2:(end))' / (T - 2);
R_2 = x_(:,1:(end-2)) * x_(:,3:(end))' / (T - 3);
R_3 = x_(:,1:(end-3)) * x_(:,4:(end))' / (T - 4);
R_4 = x_(:,1:(end-4)) * x_(:,5:(end))' / (T - 5);
R_5 = x_(:,1:(end-5)) * x_(:,6:(end))' / (T - 6);
R_6 = x_(:,1:(end-6)) * x_(:,7:(end))' / (T - 7);
R_7 = x_(:,1:(end-7)) * x_(:,8:(end))' / (T - 8);

```

```

R_8 = x_(:,1:(end-8)) * x_(:,9:(end))' / (T - 9);
R_9 = x_(:,1:(end-9)) * x_(:,10:(end))' / (T - 10);

% calculate eigenv's
[evects, evals] = eig(R_0);
evals = diag(evals);
[junk, ind] = sort(evals);
evals = evals(ind(end:-1:1));
evects = evects(:, ind(end:-1:1));
30

% estimate residual noise
var_est = mean(evals(k+1:end));
% calculate whitening matrix W using k evecs...
for temp=1:k,
    col = 1/sqrt(evals(temp) - var_est) * evects(:,temp);
    W(:,temp) = col(:);
end;
W = W';
40

% Compute sample covariance matrices for x(t)-noise
wR_0 = W * R_0 * W';
wR_1 = W * R_1 * W';
wR_2 = W * R_2 * W';
wR_3 = W * R_3 * W';
wR_4 = W * R_4 * W';
wR_5 = W * R_5 * W';
wR_6 = W * R_6 * W';
wR_7 = W * R_7 * W';
wR_8 = W * R_8 * W';
wR_9 = W * R_9 * W';
50

[V, D] = joint_diag_r([wR_0 wR_1 wR_2 wR_3 wR_4 wR_5 wR_6 wR_7], 1e-6);
A_hat = pinv(W) * V;
P_hat = V' * W * x;

```

## sobi\_spatial.m

```

function [A_hat, P_hat] = sobi_spatial(x,k);

% Takes in a 3-d matrix of hyperspectral data of size (a,b,c):
% x is the matrix we wish to separate into A,p.
% The size of A is c by k.
% The size of P is k by a*b.

% (a,b) are the spatial dimensions of the data, 'c' is the spectral
% dimension of the data.
10

% NOTE: This function will normalize x before computing the
% spatially-shifted covariance matrices if the second parameter
% passed in is set to 1.
[M, M1, M2, M3, M4] = threeD_cov(x,1);
% Returns M, size (c, a*b) which is the data in x but made 2-D so
% that the first dimension is spectral, the second is spatial.
% (The corresponding mapping to the 2-D case is where each row is a
% variable and each column is a time sample, except now each column
% is a spectral distribution sample of one pixel.)
20

T = length(M);

% calculate 2nd-order stats for a range of spatial lags
% [Note that normalization should be done in threeD_cov.]

R_0 = M * M' / (T - 1);
R_1 = M * M1' / (T - 1);
R_2 = M * M2' / (T - 1);

```

```

R_3 = M * M3' / (T - 1);
R_4 = M * M4' / (T - 1);
% calculate eigenv's
[evects, evals] = eig(R_0);
evals = diag(evals);
[junk, ind] = sort(evals);
evals = evals(ind(end:-1:1));
evects = evects(:, ind(end:-1:1));

% estimate residual noise
var_est = mean(evals(k+1:end));
% calculate whitening matrix W using k evectors...
for temp=1:k,
    col = 1/sqrt(evals(temp) - var_est) * evects(:,temp);
    W(:,temp) = col(:);
end;
W = W';

% Compute sample covariance matrices for z(t)
wR_0 = W * R_0 * W';
wR_1 = W * R_1 * W';
wR_2 = W * R_2 * W';
wR_3 = W * R_3 * W';
wR_4 = W * R_4 * W';

[V, D] = joint_diag_r([wR_0 wR_1 wR_2 wR_3 wR_4], 1e-6);
A_hat = pinv(W) * V;
% note that V is unitary and real, so inv(V) = transpose(V)
P_hat = V' * W * M;

```

## A.4.2 Spatial SOBI helper function

### threeD\_cov.m

```

function [Mlong, M1long, M2long, M3long, M4long] = ...
    threeD_cov(M, normalize)

% Given the matrix M, return the first four spacial covariance
% matrices corresponding to the nearest pixels. All matrices
% returned are of appropriate size to be passed directly to the
% SOBI algorithm.

% given a matrix X of size x-by-y-by-v:

[a b c]=size(M);

if (nargin==1),
    normalize=1;
end;
if (normalize==1),
    tempM = reshape(shiftdim(M,-1), a*b,c)';
    T = length(tempM);
    tempM = tempM - (mean(tempM'))' * ones(1,T));
    M = reshape(shiftdim(tempM,1),a,b,c);
end;

% create the shifted matrices used to compute shift covariances

% CASE 1:
% |-|-|
% |-#|*| where # is the home entry, * is the entry to which

```

```

% | - | - | it is being compared in the covariance calculation
M1 = circshift(M,[0,-1]);
% correct the mistakes on the sides:
for k=1:a, % one mistake per row
    M1(k,b,:) = M(k,b-1,:);
end;

% CASE 2:
% | - * | - |
% | - # | - |
% | - | - |
M2 = circshift(M,1);
for k=1:b, % one mistake per column
    M2(1,k,:) = M(2,k,:);
end;

% CASE 3:
% | - | - * |
% | - | # | - |
% | - | - | - |
M3 = circshift(M,[1,-1]);
for k=2:a, % one mistake per row
    M3(k,b,:) = M(k-1,b-1,:);
end;
for k=1:b-1, % one mistake per column
    M3(1,k,:) = M(2,k+1,:);
end;
M3(1,b,:) = M(2,b-1,:);

% CASE 4:
% | * | - | - |
% | - | # | - |
% | - | - | - |
M4 = circshift(M,[1,1]);
for k=2:a,
    M4(k,1,:) = M(k-1,2,:);
end;
for k=2:b, % one mistake per column
    M4(1,k,:) = M(2,k-1,:);
end;
M4(1,1,:) = M(2,2,:);

% These are all of the first ring matrices necessary to calculate
% spatially-shifted covariance matrices.

Mlong = reshape(shiftdim(M,-1), a*b,c)'; % transpose for SOBI

% note that the columns of Mlong correspond to the spectral
% dimension of M. The columns are ordered from top-left pixel to
% bottom-right pixel BY COLUMN, then by row. i.e. the spectral
% distribution of pixel (x,y) of M corresponds to
% Mlong( (a-1)*y+x, :).
M1long = reshape(shiftdim(M1,-1), a*b,c)'; % transpose for SOBI
M2long = reshape(shiftdim(M2,-1), a*b,c)'; % transpose for SOBI
M3long = reshape(shiftdim(M3,-1), a*b,c)'; % transpose for SOBI
M4long = reshape(shiftdim(M4,-1), a*b,c)'; % transpose for SOBI

```

### A.4.3 Joint Diagonalization

The following script was obtained from the website of Jean-Francois Cardoso. cardoso@sig.enst.fr and used as a part of the ONA functions.

## joint\_diag\_r.m

```

function [ V , qDs ]= rjd(A,threshold)
%*****
% joint diagonalization (possibly
% approximate) of REAL matrices.
%*****
% This function minimizes a joint diagonality criterion
% through n matrices of size m by m.
%
% Input :
% * the n by nm matrix A is the concatenation of m matrices
% with size n by n. We denote  $A = [ A_1 A_2 \dots A_n ]$ 
% * threshold is an optional small number (typically = 1.0e-8 see below).
%
% Output :
% * V is a n by n orthogonal matrix.
% * qDs is the concatenation of (quasi)diagonal n by n matrices:
% qDs = [ D1 D2 ... Dn ] where  $A_1 = V*D_1*V'$  ,...,  $A_n = V*D_n*V'$ .
%
% The algorithm finds an orthogonal matrix V
% such that the matrices D1,...,Dn are as diagonal as possible,
% providing a kind of 'average eigen-structure' shared
% by the matrices A1 ,..., An.
% If the matrices A1,...,An do have an exact common eigen-structure
% ie a common orthonormal set eigenvectors, then the algorithm finds it.
% The eigenvectors THEN are the column vectors of V
% and D1, ...,Dn are diagonal matrices.
%
% The algorithm implements a properly extended Jacobi algorithm.
% The algorithm stops when all the Givens rotations in a sweep
% have sines smaller than 'threshold'.
%
% In many applications, the notion of approximate joint diagonalization
% is ad hoc and very small values of threshold do not make sense
% because the diagonality criterion itself is ad hoc.
% Hence, it is often not necessary to push the accuracy of
% the rotation matrix V to the machine precision.
% It is defaulted here to the square root of the machine precision.
%
%
% Author : Jean-Francois Cardoso. cardoso
% This software is for non commercial use only.
% It is freeware but not in the public domain.
% A version for the complex case is available
% upon request at cardoso
% -----
% Two References:
%
% The algorithm is explained in:
%
%
% HTML = "ftp://sig.enst.fr/pub/jfc/Papers/siam_note.ps.gz",
% author = "Jean-Fran\c{c}ois Cardoso and Antoine Souloumiac",
% journal = "{SIAM} J. Mat. Anal. Appl.",
% title = "Jacobi angles for simultaneous diagonalization",
% pages = "161-164",
% volume = "17",
% number = "1",
% month = jan,
% year = {1995}
%
% The perturbation analysis is described in
%
%
% author = "{J.F. Cardoso}",
% HTML = "ftp://sig.enst.fr/pub/jfc/Papers/joint_diag_pert_an.ps",
% institution = "T\{e}\{e}com {P}aris",

```

```

% title = "Perturbation of joint diagonalizers. Ref\# 94D027",
% year = "1994" }
%
%
%
%
[m,nm] = size(A);
V=eye(m);

if nargin==1, threshold=sqrt(eps); end;

encore=1;
while encore, encore=0;
for p=1:m-1,
for q=p+1:m,
%computation of Givens rotations
g=[ A(p,p:m:nm)-A(q,q:m:nm) ; A(p,q:m:nm)+A(q,p:m:nm) ];
g=g*g';
ton =g(1,1)-g(2,2); toff=g(1,2)+g(2,1);
theta=0.5*atan2( toff , ton+sqrt(ton*ton+toff*toff) );
c=cos(theta);s=sin(theta);
encore=encore | (abs(s)>threshold);
%update of the A and V matrices
if (abs(s)>threshold) ,
Mp=A(:,p:m:nm);Mq=A(:,q:m:nm);
A(:,p:m:nm)=c*Mp+s*Mq;A(:,q:m:nm)=c*Mq-s*Mp;
rowp=A(p,:);rowq=A(q,:);
A(p,:)=c*rowp+s*rowq;A(q,:)=c*rowq-s*rowp;
temp=V(:,p);V(:,p)=c*V(:,p)+s*V(:,q); V(:,q)=c*V(:,q)-s*temp;
end; %of the if
end; %of the loop on q
end; %of the loop on p
end; %of the while loop
qDs = A ;

```



# Bibliography

- [1] Attias, H. *Independent factor analysis*. Neural Computation, vol. 11: pp. 803-851, (1998).
- [2] AVIRIS. Jet Propulsion Laboratory, California Institute of Technology. 2000-2001 <<http://aviris.jpl.nasa.gov/>>. Accessed Feb.-May, 2003.
- [3] Bell, A.J. and T.J. Sejnowski. *An information-maximization approach to blind separation and blind deconvolution*. Neural Computation, Vol. 7: pp. 1129-1159, (1995).
- [4] Belouchrani, A., K. Abed-Meraim, J.-F. Cardoso, and E. Moulines. *A blind source separation technique using second-order statistics*. IEEE Transactions on Signal Processing, 45(2): 434 - 444, (1997).
- [5] Belouchrani, A. and J.-F. Cardoso. *Maximum likelihood source separation by the expectation-maximization technique: Deterministic and stochastic implementation*. In Proceedings of 1995 International Symposium on Non-Linear Theory and Applications. Las Vegas, NV: pp. 49-53, (1995).
- [6] Cardoso, J.-F., S. Bose, and B. Friedlander. *On optimal source separation based on second and fourth order cumulants*. In Proc. IEEE Workshop on SSAP, Corfou, Greece (1996).
- [7] Comon, P. *Independent Component Analysis - A New Concept?* Signal Processing, 36: pp. 287-314, (1994).
- [8] Dempster, A., N. Laird, and D. Rubin. *Maximum likelihood estimation from incomplete data*. Journal of the Royal Statistical Society (B): 39(1), (1977).
- [9] Green, Andrew, Mark Berman, Paul Switzer, and Maurice D. Craig. *A Transformation for Ordering Multispectral Data in Terms of Image Quality with Implications for Noise Removal*. IEEE Transactions on Geoscience and Remote Sensing, 26(1): 65 - 74, (1988).
- [10] Lee, J. *Blind noise estimation and compensation for improved characterization of multivariate processes*. PhD thesis, Department of Electrical Engineering

and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA. (March 2000).

- [11] Lee, J. and D.H. Staelin. *Iterative signal-order and noise estimation for multivariate data*. Electronics Letters, 37(2): pp. 134-135, (2001).
- [12] Richards, J.A. *Remote Sensing Digital Image Analysis, An Introduction*. Second Edition, Springer-Verlag, New York, NY, (1993).
- [13] Rowe, Daniel B. *A Bayesian approach to blind source separation*. Journal of Interdisciplinary Mathematics, 5(1): 49-76, (2002).
- [14] Rowe, Daniel B. *Multivariate Bayesian statistics: models for source separation and signal unmixing*. Chapman & Hall/CRC, Boca Raton, FL, (2003).
- [15] Schowengerdt, R.A. *Remote Sensing, Models and Methods for Image Processing*. Second Edition, Academic Press, San Diego, CA, (1997).
- [16] Vane, G. *1987: Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)*. JPL Publication 87-36: p.97, (1987).