Eigenvalue Pairing for Direction Finding with Vector Sensor Arrays

Kyle T. Martin
University of Rhode Island, kyletae@gmail.com

EIGENVALUE PAIRING FOR DIRECTION FINDING WITH VECTOR SENSOR ARRAYS

BY

KYLE T. MARTIN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2013

MASTER OF SCIENCE THESIS

OF

KYLE T. MARTIN

APPROVED:

Thesis Committee:

Major Professor	Richard J. Vaccaro
	Orlando Merino
	Ashwin Sarma
	Nasser H. Zawia
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2013

ABSTRACT

This thesis introduces a novel sorting pairing method for the azimuth-elevation estimates obtained from the Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) based closed-form source localization algorithm. The ESPRIT algorithm estimates Direction of Arrival (DOA) angles from a given source using arbitrarily spaced three-dimensional arrays of vector sensors, whose locations need not be known. In estimating the DOAs there can be miss-pairings of DOA azimuth and elevation direction cosines in certain cases. In such cases, the incident angles calculated can be permuted which produces miss matching of angles or misses. The sorting pairing method presented exploits the order of eigenvalues and thereby the DOAs estimated from the ESPRIT algorithm. The sorting pairing method is compared to two other pairing methods: the traditional pairing method, a method that is simplistic in nature when dealing with misses; and the exhaustive pairing method, a method that exhaustively attempts to find the eigenvalue pairing that yields the least amount of error. Simulation results provides strong evidence that the sorting pairing method gives good results over a much larger range of signal-to-noise ratios than the traditional method, and has an accuracy that is nearly co-linear with the exhaustive method.

ACKNOWLEDGMENTS

It is with great pleasure to thank the many people who have helped me along the way.

This thesis would not have been possible without the guidance, support, and understanding of my advisor, Dr. Richard J. Vaccaro. Without his enthusiasm, inspiration, and gentle encouragement none of this would have been possible. Throughout the process he has provided excellent instruction, great suggestions, and assisted me to get to the ’eureka’ moments. Without him I would have surely been lost at sea.

I would like to thank the many people who have taught me the principles of sound reasoning and engineering: my high school teachers Misters T.J. Baker, Doug Burbank and Robert Moore; the undergraduate faculty at Boston University, Profs. Michael Ruane and Joshua Semeter; my former advisor at Woods Hole Oceanographic Institute Prof. Kenneth G. Foote; and my thesis committee at the University of Rhode Island, Profs. Musa Jouaneh, Orlando Merino, and Ashwin Sarma.

I am indebted to many of my colleagues and mentors who chose to support me by writing letters, and giving wise advice, I wish to thank in particular Drs. Joseph DiBiase, Michael Garr, Mary Johnson, and Tod Luginbuhl.

I owe my deepest gratitude to my friends, particularly Dr. Thomas D. Kenny and Mr. Andrew M. Mara who have helped me get through the tedious process of research and writing.

To my parents, Richard and Debra Martin, and my grandmother Dorothy McVeigh to whom I owe the most thanks for having endured missed time together. Without their encouragement and understanding it would have been impossible to complete my work.

DEDICATION

To Marie Sabra Martin and John Joseph McVeigh; you are both missed.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Outline

This thesis is organized in a fashion that describes Direction of Arrival (DOA) estimation for vector sensors and classical and improved eigenvalue pairing methods. The thesis is structurally organized in 4 chapters:

1. Chapter 1 is an overview of vector sensors and their current applications. It includes a description of the topic and the relevance of the research within.

2. Chapter 2 presents relevant background and assumptions, basic system equations; the derivations of the noise-free and noisy data models for a field of vector sensors; the estimation of signal parameters via rotational invariant techniques (ESPRIT) algorithm and the eigenvalue pairing problem for noise-free and noisy data models.

3. Chapter 3 gives a detailed description of a novel eigenvalue pairing method and performs the detailed analysis of a vector-sensor array (VSAs) with and without noise for both classical and improved pairing methods for VSAs that have two or more sources.

4. Chapter 4 gives examples of simulation results for scenarios involving two, three, and four sources impinging upon a 13-element sensor array.

5. Chapter 5 concludes the thesis, and evaluates the research and its contribution. This chapter also discusses implications of the results and additional areas for potential research.

1.2 Hydrophones

Most detection systems in air or free space use electromagnetic waves, but these generally absorb quickly in salt water; however, sound waves, unlike electromagnetic waves, can travel great distances, sometimes thousands of kilometers, before being dissipated underwater. These underwater sound waves are reflected by objects and are produced by machinery, making sound useful for active and passive detection. Some undersea applications reject active sonar for two reasons:

- Active sonar pulses travel far beyond the maximum detection range; thus, targets can intercept these pulses at great distance and avoid detection.
- Active sonar is not covert, and sources actively transmitting energy in the water may be located and classified.

In order to achieve sensing underwater sound waves, one would require a hydrophone, a common sensor employed for sonar, which is at its barest an underwater microphone. Like an in air microphone, a hydrophone measures pressure only and sound waves passing over a hydrophone introduce changes that are measured and used for detection. Hydrophones that have no directionality are called omni-directional hydrophones and they are quite common due to the fact that they are easy to build, maintain, and analyze. Omnidirectional hydrophones measure the acoustic pressure as a scalar quantity.

1.3 Vector Sensors

Performance of a configuration of sensors can be improved upon by increasing the amount of information that is measured by each sensor. For acoustic measurements, the particle velocity of the medium in which the sound is traveling through can provide additional information about the direction of sound arrival. This type of sensor that measures the particle velocity is known as a vector sensor.

1.3.1 Description of Vector Sensors

Acoustic vector sensors measure the pressure as well as non-scalar, vector, components of an acoustic field such as a particle velocity. The non-scalar components of an acoustic field cannot be obtained using a single acoustic pressure sensor which has no directionality, an omni-directional sensor, but can be by a single acoustic vector sensor which contains one omni-directional hydrophone measuring pressure and two or three orthogonal velocity hydrophones measuring the components of particle velocity [1]. In the past few decades extensive research has been conducted on the theory and design of vector sensors [2][3], and vector sensors have been primarily used for underwater target localizations, acoustic communications, and **SO**und **N**avigation **A**nd **R**anging **SONAR** applications.

The concept of vector sensors has been within the acoustic community since the 1930s with the seminal paper by H.F. Olson [4]. In this work Olson provides the set up and derivation of how a vector sensor would work. Olson describes a velocity sensor structure that is a tube that contains a magnetic mass suspended by springs where any vibration along the axis of the tube will cause the mass to move; thus, inducing a current in a wire coil. This induced current yields a measurement of velocity along the velocity sensor axis. Olson proved that the response of a high quality microphone with uniform sensitivity over a wide frequency range set up in such a system can measure the velocity component of a sound wave.

Vector sensors can be categorized into two general categories: inertial and gradient [5]. Inertial vector sensors measure the acceleration or velocity by responding to particle motion, while gradient sensors use a finite-difference approximation to estimate the gradients and thereby the direction of the acoustic field. There are obvious benefits and drawbacks to each category of sensor, but inertial sensors offer a broad dynamic range. However, proper supporting and packaging of the sensor without affecting its response to motion is an issue. This is due to the fact that inertial sensors do not distinguish between acoustic and non-acoustic sources such as package vibrations; therefore, they must be properly shielded from such disturbances. This shielding, and supporting and packaging issue makes it difficult to manufacture accurate and small inertial sensors for higher frequencies. Gradient sensors, unlike inertial sensors can be manufactured at significantly smaller sizes as they do not suffer the same drawbacks as their counterpart, inertial sensors. Gradient sensors smaller size makes them more suitable for higher frequencies; however, the drawback to gradient sensors is that their method of determining the 'vector' of an acoustic source, finite-difference approximation, limits their operational use, e.g. their dynamic range.

The proposed use of vector sensors takes advantage of vector sensor components at the receiver and are not limited to a particular sensor type, inertial or gradient.

1.3.2 Current Applications of Vector Sensors

Vector sensors, despite their history dating back to 1931 [4], took 38 years to have interest from the United States Navy where they have been used in acoustic sensor systems such as the **D**irectional **F**requency **A**nalysis and **R**ecording (**DIFAR**), the passive **V**ertical **L**ine **A**rray **D**IFAR (**VLAD**), and the **DI**rectional **C**ommand **A**ctivated **S**onobuoy **S**ystem (**DICASS**) [6][7]. There are also other uses for vector sensors such as a receiver for underwater acoustic communication [8].

These systems that use vector sensors can do a wide variety of tasks such as **D**irection-**O**f-**A**rrival (**DOA**) estimation or source localization. DOA estimation is when the bearings of a number of far-field acoustic sources are determined by utilizing DOA recovery algorithms. Instead of using traditional sensor array fields, spatially distributed passive or active omni-directional scalar sensors, one can use vector sensors that have the benefits of higher directionality and decreased size when compared to omni-directional sensors. The directional information that can be acquired from a vector sensor can greatly improve the number of sources that can be detected as well as the localization accuracy without increasing the array aperture [9].

The vector sensor for the purpose of this thesis is assumed to be consisting of three sensors: two identical co-located but orthogonally oriented velocity hydrophone plus another pressure hydrophone. This is covered in greater detail in Chapter 2.

1.4 ESPRIT Algorithm overview

1.4.1 ESPRIT Algorithm and ESPRIT Algorithm for Vector Sensors

DOA estimation or source localization is important for a wide variety of reasons and there are two subspace methods that are typically used, **MU**ltiple **SI**gnal **C**lassification (**MUSIC**) [10] that was popularized by Schmidt [11], and the **E**stimation of **S**ignal **P**arameters via **R**otational **I**nvariance **T**echniques (**ESPRIT**) Algorithm. The ESPRIT method, first described in [12] and later described in more detail in [13, 14], can estimate source DOAs of narrowband sources that are highly correlated. The ESPRIT algorithm, an eigenstructure source localization algorithm, has been expanded over the years and adapted for a wide variety of different systems and purposes. One of the more recent purposes for vector sensors from Wong et. al. [15, 1] uses the ESPRIT algorithm to estimate DOA's for vector sensors.

The ESPRIT algorithm is an eigenstructure (subspace) DOA estimation algorithm that is similar to MUSIC as it exploits the underlying data model, and generally ESPRIT estimates are unbiased and computationally efficient to obtain.

ESPRIT and other eigenstructure source localization algorithms decompose the column space of the data correlation matrix into a signal subspace and a noise subspace. ESPRIT is favored over other DOA estimation methods due to the following features:

- ESPRIT requires less processing when compared to MUSIC, where the computational load for ESPRIT grows linearly, as opposed to MUSIC, which grows exponentially when the algorithms are extended to multidimensional systems.

- ESPRIT does not require a priori knowledge of the array geometry and element characteristics.

ESPRIT can be extended to vector sensors and DOA performance can be improved upon as shown by [1]. The improvements made by Wong et. al. is the advancement of the non-spatial realization of the ESPRIT algorithm and the extension of ESPRIT to the vector sensor. The improvement allows the ESPRIT algorithm to depend on only the impinging signals' direction-cosines and not on any array parameters as long as all sensors are oriented in the same direction; however, even if they are not oriented in the same direction Wong et. al. describes a method to align data received from non-aligned sensors as if it came from sensors in the same orientation. This extension of the ESPRIT algorithm removes previous constraints on a priori knowledge of the sensor array by allowing a DOA estimate when sensors are arbitrarily located in a field of sensors in an arbitrarily spaced 3-Dimensional region with sensors at unknown locations.

It is important to note that without vector sensors, the above claims and improvements would not be able to be achieved; that is to say, that the ESPRIT improvements are vector sensor dependent. In the vector sensor that was previous described, the co-located vector hydrophones in the vector sensors measures two of the three (or all three if the vector sensors has three velocity hydrophones) Cartesian components of the velocity field plus the over all pressure from the omni-directional sensor. Normalization of the velocity-field components would give three Cartesian direction-cosines of the acoustic signal.

Despite the ESPRIT algorithm's advantages, ESPRIT does have limitations and these limitations, as well as more information on ESPRIT, are covered in Chapter 2, Review of Existing ESPRIT Algorithm Development.

1.4.2 Eigenvalue Pairing

Using the ESPRIT algorithm, the DOA estimation of the azimuth and elevation angles are calculated and recovered in two separate eigenvalue decompositions (EVDs). The numerical methods used to calculate the eigenvalue decomposition do not give the eigenvalues in any particular order. Thus, additional calculations are needed to properly pair an elevation angle obtained from one EVD with the corresponding azimuth angle obtained from the other EVD. This issue of eigenvalue permutation and the pairing of permuted eigenvalue pairs is covered in Chapters 2 and 3.

1.4.3 Results and Limitations

Results that compare and contrast three different pairing methods are explored in Chapter 3: classical, sorting, and exhaustive. The classical pairing method is the pairing method that is described in [1] and the sorting method is a new method presented in this thesis. Finally the exhaustive method runs through every possible permutation and selects the permutation matrix that presents the least amount of error.

1.5 Contributions of thesis

The contribution of this thesis is a new pairing method that relies upon sorting. This method performs significantly better than the traditional pairing method. Simulation results in Chapter 4 show that the sorting pairing method performs better than the traditional method and that the sorting pairing method performs almost as well as the exhaustive pairing method.

List of References

[1] K. Wong and M. Zoltowski, "Closed-form underwater acoustic direction finding with arbitrarily spaced vector hydrophones at unknown locations," *IEEE Journal Of Ocean Engineering*, vol. 22, pp. 566–575, July 1997.

[2] *Proceedings Workshop Directional Acoustic Sensors (CD-ROM)*, Newport, RI, 2001.

[3] A. Nehorai and E. Paldi, "Acoustic vector-sensor array processing," *IEEE Transactions on Signal Processing*, vol. 42, pp. 2481–2491, 1994.

[4] H. F. Olson, "Mass controlled electrodynamic microphones: The ribbon microphone," *Journal Acoustical Society America*, vol. 3, pp. 56–58, 1931.

[5] T. B. Gabrielson, "Design problems and limitations in vector sensors," in *Proceedings Workshop Directional Acoustic Sensors (CD-ROM)*, Newport, RI, 2001.

[6] M. Higgins, "DIFAR system overview," in *Proceedings Workshop Directional Acoustic Sensors (CD-ROM)*, Newport, RI, 2001.

[7] M. T. Silvia and R. Richards, "A theoretical and experimental investigation of low-frequency acoustic vector sensors," in *Proceedings Oceans '02*, Beloxi, MS, 2002.

[8] A. Abdi, H. Guo, and P. Suthiwan, "A new vector sensor receiver for underwater acoustic communication."

[9] M. Hawkes and A. Nehorai, "Acoustic vector-sensor beamforming and capon direction estimation," *IEEE Transactions on Signal Processing*, vol. 46, pp. 2291–2304, Sept. 1998.

[10] K. Wong and M. Zoltowski, "Root-MUSIC-based azimuth-elevation angle-of-arrival estimation with uniformly spaced but arbitrarily oriented velocity hydrophones," *IEEE Transactions on Signal Processing*, vol. 47, pp. 3250–3260, December 1999.

[11] R. O. Schmidt, “Multiple emitter location and signal parameter estimation,” in *Proceedings RADC Spectrum Estimation Workshop*, Griffiths AFB, N.Y., 1979.

[12] A. Paulraj, R. Roy, and T. Kailath, “Estimation of signal parameters via rotational invariance techniques - ESPRIT,” in *Proc. Nineteenth Asilomar Conference on Circuits, Systems and Computers*, Aslomar, CA, November 1985.

[13] A. Paulraj, R. Roys, and T. Kailath, “Estimation of Signal Parameters Via Rotational Invariance Techniques ESPRIT,” in *Proc. Nineteenth Asilomar Conference on Circuits, Systems and Computers*, Asilomar, CA, November 1985.

[14] R. Roy, A. Paulraj, and T. Kailath, “Direction-of-arrival estimation by subspace rotation methods - ESPRIT,” *IEEE ICASSP*, pp. 2495–2498, 1986.

[15] K. Wong and M. Zoltowski, “Orthogonal-velocity-hydrophone esprit for sonar source localization,” *MTS/IEEE Oceans 96 Conference*, vol. 3, pp. 1307–1312.

CHAPTER 2

Review of Existing ESPRIT Algorithm Developments and Eigenvalue Pairing for VSAs

2.1 Assumptions for the ESPRIT Algorithm

To simplify the derivation and the entire document, the following assumptions are made throughout the entire thesis.

2.1.1 Coordinate Systems

For this thesis, the 3-Dimensional Cartesian coordinate system consisting of x, y, and z axes will be used. This system uses θ to refer to the angle from the vertical z-axis, elevation angle, and ϕ to refer to the angle from the x-axis in the xy-plane, azimuth angle.

2.1.2 Sensor and environmental Model

- The entire signal model is in a free-space environment where sound waves travel in a homogeneous, isotropic fluid wholespace which implies direct-path propagation only. Constructive and deconstructive interference or multi-path propagation is not assumed. It is also assumed that the speed of the sound wave is uniform across all sensors.

- The impinging signal is a narrow-band acoustic source that operates on one frequency f where the wavelength $\lambda = 1/f$. In practice this means that the signal is sufficiently band-limited to allow narrowband processing in the frequency domain. Mathematically speaking this would mean that the bandwidth β is significantly less than the carrier frequency f_c. Such a band-limited signal may be obtained by pre-filtering or computing the Discrete Fourier Transform of the time series measurements.

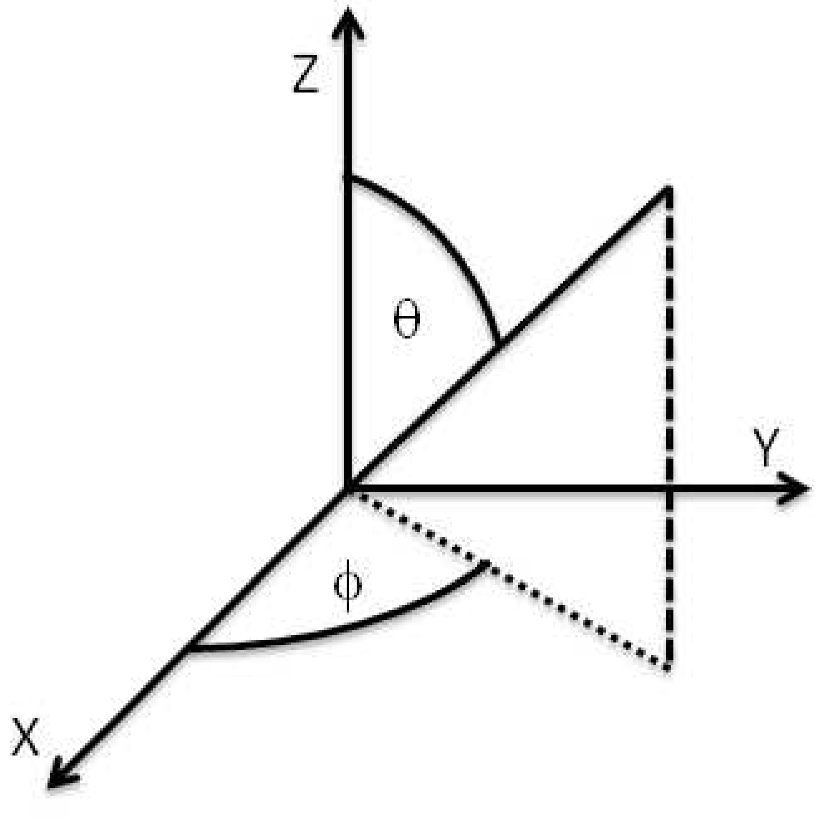

Figure 1. The 3-Dimensional Cartesian Coordinate System with illustrations to show X,Y, and Z axis; θ the elevation angle which is the angle from the z axis; and ϕ the azimuth angle which is the angle from the x-axis in the xy-plan

- The signal model assumes that the sound waves are planar across the array. This implies that the unit-vector from each sensor to a source or K sources is the same, regardless of the sensor location. In practice this would require sources to be in the far-field whose distances are much greater than the length of the array.

- The K source signals impinging upon the array of L vector sensors has the stipulation that $L > K$ at possibly arbitrary and possibly unknown locations in a 3-D region with arbitrary geometry.

2.1.3 Vector Sensor Array

- In this thesis it is assumed that each vector sensor consists of three components: one pressure hydrophone and two velocity sensors. In each vector sensor, all three components are located at the same point; in other words all three sensors are co-located. In practice this requires that the component spacing is small compared to the minimum wavelength, which is set by the highest operating frequency.

- Each vector-sensor is modeled as a single point. In practice this requires the sensor dimensions to be small when compared with the minimum wavelength.

- The signal response of each velocity hydrophone is proportional to the cosine of the angle between the velocity axis and the source. Cosine velocity response results from measuring the velocity, and is performed only along one axis.

- The axes of the two velocity hydrophones are orthogonal. In practice this is true where each vector-sensor is a static unit.

- The vector sensor array consists of L vector sensors and while all locations of the vector sensors need not be known, all the velocity hydrophones, as previously stated, need to be identically oriented, i.e. the array of L vector sensors are located at arbitrary and possibly unknown locations in a 3-Dimensional region with completely arbitrary geometry, and all vector hydrophones are oriented in the same direction.

2.2 Derivation of noise-free VSA data

2.2.1 Signal Measurement Model

Each vector sensor's (possibly unknown) location can be defined as

$$p_l = \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} \quad \text{where } l = 1, \ldots, L \tag{1}$$

where p_l denote the position of the l^{th} vector sensor in 3-Dimensional space, and each position is defined for the purpose of this paper in units of wavelengths.

The three-component vector hydrophone would produce the following 3×1 manifold with regard to the k^{th} source impinging from (θ_k, ϕ_k):

$$\mathbf{a}^{(3)}(\theta_k, \phi_k) \equiv \begin{bmatrix} u\,(\theta_k, \phi_k) \\ v\,(\theta_k, \phi_k) \\ 1 \end{bmatrix} = \begin{bmatrix} sin\,\theta_k \;\; cos\,\phi_k \\ sin\,\theta_k \;\; sin\;\phi_k \\ 1 \end{bmatrix} \tag{2}$$

where θ denotes the kth source's elevation angle measured from the vertical z axis parametrized by $\theta_k \in (0, 2\pi]$ and ϕ denotes the kth source's azimuth angle parametrized by $\phi_k \in [-\pi/2, \pi/2]$. Note that $u(\theta_k, \phi_k)$ and $v\,(\theta_k, \phi_k)$ or u and v respectively can be referred to as direction cosines. This nomenclature will be used later in the chapter and in Chapter 3.

2.2.2 Intervector hydrophone spatial phase factor

Each vector sensor is located at a particular 3-Dimensional (possibly unknown) location p_l (1). Given the different sensor locations and the nature of the signal, each sensor will have a different phase. In order to simulate this in the signal model, an intervector hydrophone spatial phase factor is needed. The phase response of the l^{th} vector sensor from a plane wave impinging from angle θ and ϕ is given by

$$\begin{aligned} q_l(\theta_k, \phi_k) &\equiv e^{j2\pi \frac{x_l\,u_k + y_l\,v_k + z_l}{\lambda}} \\ &= \underbrace{e^{j2\pi \frac{x_l\,u_k}{\lambda}}}_{\equiv\, q_l^x(u_k)} \quad \underbrace{e^{j2\pi \frac{y_l\,v_k}{\lambda}}}_{\equiv\, q_l^y(v_k)} \quad \underbrace{e^{j2\pi \frac{z_l\,w_k}{\lambda}}}_{\equiv\, q_l^z} \end{aligned} \tag{3}$$

The phase responds for all L sensors may be combined into a vector

$$\begin{bmatrix} q_1(\theta_k, \phi_k) \\ \vdots \\ q_L(\theta_k, \phi_k) \end{bmatrix} \equiv \boldsymbol{q}(\theta_k, \phi_k) = \boldsymbol{q}(u(\theta_k, \phi_k), v(\theta_k, \phi_k)) \tag{4}$$

This intervector hydrophone spatial phase factor as described in (3) can now be employed on a $3L \times 1$ array for the entire L-element vector hydrophone array in order to spatially relate the received signal on each individual sensor.

$$\boldsymbol{a}(\theta_k, \phi_k) \equiv \boldsymbol{a}^{(3)}(\theta_k, \phi_k) \otimes q(\theta_k, \phi_k) \tag{5}$$

where $\otimes$ symbolizes the Kronecker-product, or tensor product, that is defined as:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11} \begin{bmatrix} b_{11} \cdots b_{1m} \\ \vdots \ddots \vdots \\ b_{k1} \cdots b_{km} \end{bmatrix} & \cdots & a_{1j} \begin{bmatrix} b_{11} \cdots b_{1m} \\ \vdots \ddots \vdots \\ b_{k1} \cdots b_{km} \end{bmatrix} \\ \vdots & \ddots & \vdots \\ a_{i1} \begin{bmatrix} b_{11} \cdots b_{1m} \\ \vdots \ddots \vdots \\ b_{k1} \cdots b_{km} \end{bmatrix} & \cdots & a_{ij} \begin{bmatrix} b_{11} \cdots b_{1m} \\ \vdots \ddots \vdots \\ b_{k1} \cdots b_{km} \end{bmatrix} \end{bmatrix} \tag{6}$$

where $\boldsymbol{A}$ is an $i \times j$ matrix and $\boldsymbol{B}$ is a $k \times m$ matrix. This operation is possible because the phase vector $\mathbf{a}$ is simply the measurement vector for the corresponding pressure-sensor array. With the total of $K \leq L$ co-channel signals, the entire array would yield a $3L \times 1$ vector measurement $\mathbf{z}(t)$ at time t:

$$\mathbf{z}(t) = \sum_{k=1}^{K} \mathbf{a}(\theta_k, \phi_k) s_k(t) + \mathbf{n}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t) \tag{7}$$

$$s_k \equiv \sqrt{P_k} \sigma_k(t) e^{j(2\pi \frac{c}{\lambda} t + \psi_k)} \tag{8}$$

$$\mathbf{A} \equiv [\mathbf{a}(\theta_1, \phi_1), \ldots, \mathbf{a}(\theta_K, \phi_K)] \tag{9}$$

$$\mathbf{s}(t) \equiv \begin{bmatrix} s_1(t) \\ \vdots \\ s_K(t) \end{bmatrix}, \qquad \mathbf{n}(t) \equiv \begin{bmatrix} n_1(t) \\ \vdots \\ n_K(t) \end{bmatrix} \tag{10}$$

where $\mathbf{n}(t) = 0$, P_k denotes the kth signal's power, $\sigma_k(t)$ represents a zero-mean unit-variance complex random process, λ refers to the signals' wavelength, c represents the propagation speed, and ψ_k denotes the kth signal's uniformly distributed random carrier phase. The total of N snapshots (where $N > K$) taken at the distinct instants $t_n, n = 1, \cdots, N$, yields a $3L\ X\ N$ data matrix

$$\boldsymbol{Z} \equiv [\mathbf{z}(t_1)\ \cdots\ \mathbf{z}(t_N)]. \tag{11}$$

This matrix is a representation of the received acoustic energy from the vector-hydrophone sensor field. In it contains information that can be used to determine and estimate the DOA's through direction angles (θ_k, ϕ_k) where $k = 1, \cdots, K$.

2.3 ESPRIT Algorithm

This section will estimate the DOA's given the measurements from $\boldsymbol{Z}$ through the ESPRIT algorithm and makes no assumptions about array geometry, element characteristics, DOA's, or signal correlations.

2.3.1 Overview of ESPRIT Algorithm

DOA estimation is important for a wide variety of reasons and there are two methods that are typically used, the single subspace method, **MU**ltiple **SI**gnal **C**lassification (**MUSIC**) [1] that was popularized by Schmidt [2], and the **E**stimation of **S**ignal **P**arameters via **R**otational **I**nvariance **T**echniques (**ESPRIT**) Algorithm [3]. The MUSIC algorithm yields asymptotically unbiased and efficient estimates. The MUSIC algorithm estimates the signal subspace from the array measurements and then estimates the parameters of interest from the intersections between the array manifold and the estimated signal subspace; however, in order to use the MUSIC algorithm, a priori knowledge of the array geometry and element characteristics is required [4]. Therefore it is impossible to do DOA estimation using MUSIC when the array geometry and element characteristics are unknown.

The ESPRIT approach first proposed in [3] is similar to MUSIC as it exploits the underlying data model and produces generally unbiased estimates. [3, 4] in computationally efficient manner. The advantage of using ESPRIT over MUSIC is that

- ESPRIT does not require a priori knowledge of the array geometry and element characteristics.
- It is computationally less complex since it does not need the search procedure that MUSIC uses.

These benefits explain why ESPRIT has had a significant amount of research over the past two and a half decades. The ESPRIT algorithm is not perfect as it assumes that the signal source is in the far field; in other words, the ESPRIT algorithm can only operate with plane waves.

Despite the above limitations, there are evident reasons why the ESPRIT algorithm would be favored in a wide variety of cases over other algorithms. These same limitations give rise to the special conditions under which the ESPRIT algorithm can offer a unique solution to the DOA estimation problem.

2.3.2 ESPRIT Algorithm Derivation

It is important to realize that the goal of the ESPRIT algorithm is to effectively remove the effects of $\mathbf{q}$, shown in (3), the intervector phase factor between the sensors; thus, recovering the direction-cosines. By recovering the direction-cosines, the estimated DOA of the source can then be calculated.

2.3.3 Overview and Intuitive Analysis of the ESPRIT algorithm to estimate DOA's

Recall that each column the data matrix $\boldsymbol{Z}$ is a $3L \times 1$ vector. Each vector can be divided up into three $L \times 1$ sub-vectors. These $L \times 1$ sub-vectors are related to one another by invariant factors $\mathbf{q}$ that are dependent only on the positional locations p_l of the vector hydrophones and not the direction cosines of the sources. The $3L \times 1$ array manifold described as $\mathbf{a}$ from equation (5) can be rewritten as:

$$\mathbf{a}(\theta_k, \phi_k) = \mathbf{a}^{(3)}(\theta_k, \phi_k) \otimes \mathbf{q} = \begin{bmatrix} u\,(\theta_k, \phi_k)\mathbf{q}(\theta_k, \phi_k) \\ v\,(\theta_k, \phi_k)\mathbf{q}(\theta_k, \phi_k) \\ \mathbf{q}(\theta_k, \phi_k) \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1\,(\theta_k, \phi_k) \\ \mathbf{a}_2\,(\theta_k, \phi_k) \\ \mathbf{a}_3\,(\theta_k, \phi_k) \end{bmatrix}. \qquad (12)$$

Now that $\mathbf{a}$ is formed in a more explicit way one can see that the three sub-vectors are each related by the same $\boldsymbol{q}(\theta_k, \phi_k)$ factor. Furthermore, one can form three $L \times K$ data blocks out of the matrix $\mathbf{A}$, shown in (9), as follows

$$\mathbf{A}_j \equiv \mathbf{S}_j\mathbf{A}, \qquad j = 1, \ldots, J \qquad (13)$$

where $\mathbf{S}_j$ is an $L \times JL$ sub-array selection matrix

$$\mathbf{S}_j \equiv [\mathbf{O}_{L,L\times(j-1)} \vdots \mathbf{I}_L \vdots \mathbf{O}_{L,L\times(J-j)}] \qquad j = 1, \ldots, J \qquad (14)$$

where $\mathbf{O}_{m,n}$ denotes an $m \times n$ zero matrix, and $\mathbf{I}_m$ denotes an $m \times m$ matrix identity matrix. This matrix multiplication divides up $\mathbf{A}$ into 3 sub-blocks, which can be shown using (12) to be interrelated and described as:

$$\mathbf{A}_1 = \mathbf{A}_3 \underbrace{\begin{bmatrix} u(\theta_1, \phi_1) & & \\ & \ddots & \\ & & u(\theta_K, \phi_K) \end{bmatrix}}_{\equiv \boldsymbol{\Phi}^u} \qquad (15)$$

$$\mathbf{A}_2 = \mathbf{A}_3 \underbrace{\begin{bmatrix} v(\theta_1, \phi_1) & & \\ & \ddots & \\ & & v(\theta_K, \phi_K) \end{bmatrix}}_{\equiv \boldsymbol{\Phi}^v}. \qquad (16)$$

2.3.4 ESPRIT derivation for Vector Sensors

An alternative derivation to the literature derivation is as follows. Recall that $\mathbf{A}_1 = \mathbf{A}_3\mathbf{\Phi}^u$ from (15) where $\mathbf{\Phi}^u$ is a diagonal matrix containing the u direction cosines. Also, $\boldsymbol{A}_2 = \boldsymbol{A}_3\mathbf{\Phi}^v$.

In the absence of noise, equation (7) shows that $\boldsymbol{Z} = \boldsymbol{AS}$, and the column-space of $\boldsymbol{Z}$ is the same as the column space of $\boldsymbol{A}$ or

$$col(\mathbf{Z}) = col(\mathbf{A}) \tag{17}$$

where $col(\mathbf{Z})$ denotes column-space for $\mathbf{Z}$. If the singular value decomposition (SVD) is taken of $\mathbf{Z}$, it can be written as follows

$$\begin{aligned} \mathbf{Z} &= \underbrace{\mathbf{U}}_{3L \times 3L} \mathbf{\Sigma}\mathbf{V}^H \\ \mathbf{Z} &= \underbrace{\mathbf{U}_1}_{3L \times K} \mathbf{\Sigma}_1\mathbf{V}_1^H \end{aligned} \tag{18}$$

where $\mathbf{U}_1$ contains the left singular vectors of $\mathbf{Z}$ corresponding to the non-zero singular values. Recall that a property of the SVD is that $col(\mathbf{Z}) = col(\mathbf{U}_1)$ and using (17) this gives

$$col(\mathbf{A}) = col(\mathbf{U}_1). \tag{19}$$

Note that the relationship between the eigenvalue and singular value decomposition is that the left singular vectors of $\mathbf{Z}$ are the eigenvectors of $\mathbf{Z}\boldsymbol{Z}^H$. Thus $[\mathbf{U}, \mathbf{E}] = \text{eig}(\mathbf{Z}\boldsymbol{Z}^H)$ could be used in place of the SVD.

From (19) the following is true

$$\boldsymbol{A} = \boldsymbol{U}_1\boldsymbol{T} \tag{20}$$

where T is a non-singular change-of-basis matrix. Recall that

$$\boldsymbol{A} = \begin{array}{r} L\{ \\ L\{ \\ L\{ \end{array} \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \boldsymbol{A}_3 \end{bmatrix} \tag{21}$$

given this fact, $\boldsymbol{U}_1$ can be partitioned such as the following

$$\boldsymbol{U}_1 = \begin{array}{r} L\{ \\ L\{ \\ L\{ \end{array} \begin{bmatrix} \boldsymbol{U}_{11} \\ \boldsymbol{U}_{21} \\ \boldsymbol{U}_{31} \end{bmatrix} \quad . \tag{22}$$

From (20),

$$\begin{aligned} \boldsymbol{A}_1 &= \boldsymbol{U}_{11}\boldsymbol{T} \\ \boldsymbol{A}_2 &= \boldsymbol{U}_{21}\boldsymbol{T} \\ \boldsymbol{A}_3 &= \boldsymbol{U}_{31}\boldsymbol{T} \end{aligned} \tag{23}$$

and through manipulation and substitution of the first and third equations into $\boldsymbol{A}_1 = \boldsymbol{A}_3\boldsymbol{\Phi}^u$ (15) one can get the resultant equation:

$$\begin{aligned} \boldsymbol{U}_{11}\boldsymbol{T} &= \boldsymbol{U}_{31}\boldsymbol{T}\boldsymbol{\Phi}^u \\ \boldsymbol{U}_{11} &= \boldsymbol{U}_{31}\boldsymbol{T}\boldsymbol{\Phi}^u\boldsymbol{T}^{-1} \end{aligned} \tag{24}$$

where the expression $\boldsymbol{T}\boldsymbol{\Phi}^u\boldsymbol{T}^{-1}$ is given the name $\boldsymbol{F}_u$

$$\boldsymbol{F}_u = \boldsymbol{T}\boldsymbol{\Phi}^u\boldsymbol{T}^{-1}. \tag{25}$$

Combining (24) and (25) yields $\boldsymbol{U}_{11} = \boldsymbol{U}_{31}\boldsymbol{F}_u$, which may be solved for $\boldsymbol{F}_u$ to obtain

$$\boldsymbol{F}_u = (\boldsymbol{U}_{31}^H\boldsymbol{U}_31)^{-1}\boldsymbol{U}_{11}. \tag{26}$$

Using (25) and the fact that $\boldsymbol{\Phi}^u$ is diagonal, the eigenvalues of $\boldsymbol{F}_u$ are the diagonal elements of $\boldsymbol{\Phi}^u$.

Similarly through substitution of the second and third equations from (23) into $\boldsymbol{A}_2 = \boldsymbol{A}_3\boldsymbol{\Phi}^v$ (16) then one can get the resultant equation:

$$\begin{aligned} \boldsymbol{U}_{21}\boldsymbol{T} &= \boldsymbol{U}_{31}\boldsymbol{T}\boldsymbol{\Phi}^U \\ \boldsymbol{U}_{21} &= \boldsymbol{U}_{31}\boldsymbol{T}\boldsymbol{\Phi}^U\boldsymbol{T}^{-1} \end{aligned} \tag{27}$$

where the expression $\boldsymbol{T}\boldsymbol{\Phi}^v\boldsymbol{T}^{-1}$ is given the name $\boldsymbol{F}_v$,

$$\boldsymbol{F}_v = \boldsymbol{T}\boldsymbol{\Phi}^v\boldsymbol{T}^{-1}. \tag{28}$$

Using a derivation similar to that for $\boldsymbol{F}_u$, it can be shown that

$$\boldsymbol{F}_v = (\boldsymbol{U}_{31}^H \boldsymbol{U}_{31})^{-1} \boldsymbol{U}_{21} \tag{29}$$

Therefore, the eigenvalues of $\boldsymbol{F}_v$ are the diagonal elements of $\boldsymbol{\Phi}^u$.

From inspection (25) and (28) show that $\boldsymbol{F}_u$ and $\boldsymbol{F}_v$ have the same eigenvector matrix $\boldsymbol{T}$. This is true when the u direction cosines and v direction cosines are listed in corresponding order. If the eigenvectors of $\boldsymbol{F}_u$ and $\boldsymbol{F}_v$ are computed separately, it is unlikely that they will be in corresponding order. In general, the eigenvectors and eigenvalues of $\boldsymbol{F}_v$ will have to be permuted to get into corresponding order with the eigenvectors and eigenvalues of $\boldsymbol{F}_u$. Put differently, $[\boldsymbol{T}_1, \boldsymbol{\Lambda}_1] = eig(\boldsymbol{F}_u)$ and $[\boldsymbol{T}_2, \boldsymbol{\Lambda}_1] = eig(\boldsymbol{F}_v)$, where $\boldsymbol{T}_2 = \boldsymbol{T}_1 \boldsymbol{P}$ and $\boldsymbol{P}$ is a permutation matrix, and the diagonal elements of $\boldsymbol{P}\boldsymbol{\Lambda}_1\boldsymbol{P}^T$ are in corresponding order with the diagonal elements of $\boldsymbol{\Lambda}_2$. Note that this will be covered more in the next section, and in much greater detail in Chapter 3.

2.4 Pairing Problem

First recall known facts about permutations. Let π represent the permutation function that takes integers $i = 1, \ldots, N$ into some permuted order defined by $\pi(i)$. Let $\boldsymbol{e}_i$ be the ith standard basis vector in $\Re^n$. The permutation matrix corresponding to the permutation function π is

$$\mathbf{P} = [\boldsymbol{e}_{\pi(i)} \ \cdots \ \boldsymbol{e}_{\pi(i)}]. \tag{30}$$

Also recall some facts regarding permutations matrices

- Permutation matrices are orthogonal such that

$$\mathbf{P}^{-1} = \mathbf{P}^T \qquad \mathbf{P}\mathbf{P}^T = \mathbf{I} \qquad \text{where } \mathbf{I} \text{ is the identity matrix} \tag{31}$$

- Given a matrix $\mathbf{A}$=[$\boldsymbol{a}_1 \ \cdots \ \boldsymbol{a}_n$], multiplication on the right by $\mathbf{P}$ permutes the columns of $\mathbf{A}$ as follows:

$$\boldsymbol{AP} = [\mathbf{a}_{\pi(1)} \ \cdots \ \mathbf{a}_{\pi(n)}] \tag{32}$$

- A permutation matrix and its inverse may be used to permute the diagonal elements of a diagonal matrix.

$$\boldsymbol{D} = diag(d_1, \ldots, d_n) \quad \text{and} \quad \overline{\boldsymbol{D}} = diag(\overline{d}_1, \ldots, d_n), \quad \text{where } \overline{d}_i = d_{\pi(i)}. \tag{33}$$

  If $\mathbf{P}$ is defined by (30) then

$$\overline{\boldsymbol{D}} = \boldsymbol{P}^T \boldsymbol{DP} = \boldsymbol{P}^{-1} \boldsymbol{DP} \tag{34}$$

2.4.1 Eigenvalue Pairing

To develop the concept of eigenvalue pairing consider two different matrices, $\mathbf{F}$ and $\mathbf{G}$, that have the same eigenvectors but different eigenvalues.

$$\begin{aligned} \boldsymbol{F} &= \boldsymbol{T}_1 \boldsymbol{\Lambda} \boldsymbol{T}_1^{-1}, \quad \boldsymbol{\Lambda} = diag(\lambda_1, \ldots, \lambda_n) \\ \boldsymbol{G} &= \boldsymbol{T}_1 \boldsymbol{D} \boldsymbol{T}_1^{-1}, \quad \boldsymbol{D} = diag(d_1, \ldots, d_n) \end{aligned} \tag{35}$$

where the columns of $\boldsymbol{T}_1$ are eigenvectors of the matrices $\mathbf{F}$ and $\mathbf{G}$, and the corresponding eigenvalues are the diagonal elements of $\boldsymbol{\Lambda}$ and $\boldsymbol{D}$. Assume that the eigenvalues of $\mathbf{F}$ and $\mathbf{G}$ are real numbers and correspond to each other in the following order of pairs

$$(\lambda_1, d_1), (\lambda_2, d_2), \ldots, (\lambda_n, d_n) \tag{36}$$

Suppose that the columns of $\boldsymbol{T}_1$ are permuted into a new matrix $\boldsymbol{T}_2$ and that the diagonal elements of $\boldsymbol{D}$ are permuted by a permutation matrix $\boldsymbol{P}$ into a new matrix $\overline{\boldsymbol{D}}$ where

$$\boldsymbol{T}_2 = \boldsymbol{T}_1 \boldsymbol{P}, \quad \overline{\boldsymbol{D}} = \boldsymbol{P}^T \boldsymbol{DP}. \tag{37}$$

Note that the diagonal elements of $\overline{\boldsymbol{D}}$ are now a permutation of the diagonal elements of $\boldsymbol{D}$. That is to say that $\overline{d} = d_{\pi(i)}$ where π is the permutation function and defined in (33); thus, the columns of $\boldsymbol{T}_2$ are eigenvectors of $\boldsymbol{G}$ and the corresponding eigenvalues are the diagonal elements of $\overline{\boldsymbol{D}}$. As a result of the preceding equations

$$\boldsymbol{G} = \boldsymbol{T}_2\overline{\boldsymbol{D}}\boldsymbol{T}_2^{-1}. \tag{38}$$

In the eigenvalue pairing problem the computed eigendecomposition of $\boldsymbol{G}$ is (38) and not (35). Therefore given the eigendecomposition $\boldsymbol{G}$ in (38) and the eigendecomposition of $\boldsymbol{F}$ in (35) the eigenvalue pairing problem is to find the permutation matrix $\boldsymbol{P}$ or its associated permutation function π.

From equation (37) the permutation matrix $\boldsymbol{P}$ and, therefore, the permutation function π can be obtained from the eigenvector matrices $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$

$$\boldsymbol{P} = \boldsymbol{T}_1^{-1}\boldsymbol{T}_2. \tag{39}$$

The classical pairing method algorithm can be described by the following: given the matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ with identical eigenvectors, suppose that the eigendecomposition of each matrix can be defined as

$$\boldsymbol{F} = \boldsymbol{T}_1\boldsymbol{\Lambda}\boldsymbol{T}_1^{-1} \tag{40}$$

$$\boldsymbol{G} = \boldsymbol{T}_2\overline{\boldsymbol{D}}\boldsymbol{T}_2^{-1} \tag{41}$$

where the permutation matrix that puts the columns of $\boldsymbol{T}_2$ into the same order as the columns of $\boldsymbol{T}_1$ is

$$\boldsymbol{P} = \boldsymbol{T}_1^{-1}\boldsymbol{T}_2 \tag{42}$$

where the elements of $\boldsymbol{P}$ will be 0 or ±1, and where a -1 in the ijth element of P implies that the ith column of $\boldsymbol{T}_2$ is the negative of the jth column of $\boldsymbol{T}_1$. Note that the negative signs cancel out in the computation of the diagonal matrix

$$\boldsymbol{D} = \boldsymbol{P}\overline{\boldsymbol{D}}\boldsymbol{P}^T \tag{43}$$

Now that $\boldsymbol{D}$ has been permuted by $\boldsymbol{P}$, the diagonal elements of D correspond to the diagonal elements of $\boldsymbol{\Lambda}$.

The implementation in MATLAB M-code for calculating equation (42), $\boldsymbol{P}$, is as follows:

```
function [eig_F1,eig_F2]=...
pairing_methods(z,eig_F1,eig_F2,F1,F2,P_extensive,flag_type)
   % Find the Eigenvalues and Eigenvectors of the
   %  matrix from the SVD formulation for both matrices
   [T1,Db_1]=eig(F1);
   [T2,Db_2]=eig(F2);
   % Take the real part of the eigenvalues for both sets
   % of eigenvalues
   Db_1=real(Db_1);
   Db_2=real(Db_2);
   % Calculate the permutation matrix
   P=(T1\T2);
end
```

2.5 Consider Noisy Data

Recall from equations (8) and (10) that the additive noise, $\mathbf{n}(t)$, to the data snapshot, $z(t)$, was previously noiseless, $\mathbf{n}(t) = 0$. Now suppose that $\boldsymbol{n}(t) \neq 0$ and data matrix, which contains additive noise, is $\boldsymbol{Z}_{noise}$. When the DOAs are estimated using the ESPRIT algorithm from the steps described in (25) or (28) and the information provided about the eigenvalue permutation in section 2.4 are applied to $\boldsymbol{Z}_{noise}$ then the estimated noisy DOAs are the eigenvalues of the following matrices:

$$\begin{aligned} \tilde{\boldsymbol{F}}_u &= \boldsymbol{F}_u + \boldsymbol{\Delta}_1 \\ \tilde{\boldsymbol{F}}_v &= \boldsymbol{F}_v + \boldsymbol{\Delta}_2 \end{aligned} \tag{44}$$

where $\boldsymbol{\Delta}_1$ and $\boldsymbol{\Delta}_2$ represent perturbations due to noise. Note that one of the eigenvalue sets is ordered; the other has to be put into corresponding order. Let $\boldsymbol{F} = \boldsymbol{F}_u$ and $\boldsymbol{G} = \boldsymbol{F}_v$. In this case the eigendecomposition will yield

$$\begin{aligned} \mathbf{F} &= \boldsymbol{T}_{1_{noise}} \boldsymbol{\Lambda} \boldsymbol{T}_{1_{noise}}^{-1}, \quad & \boldsymbol{\Lambda} &= diag(\lambda_1, \ldots, \lambda_n) \\ \mathbf{G} &= \boldsymbol{T}_{2_{noise}} \boldsymbol{D} \boldsymbol{T}_{2_{noise}}^{-1}, \quad & \boldsymbol{D} &= diag(d_1, \ldots, d_n) \end{aligned} \tag{45}$$

In this particular case when the matrix $\boldsymbol{P}$ is calculated

$$\boldsymbol{P} = \boldsymbol{T}_{1_{noise}}^{-1}\boldsymbol{T}_{2_{noise}} \tag{46}$$

it is no longer a permutation matrix. The matrix $\boldsymbol{P}$ will instead take on values other than 0 and 1, which are the only element values of a permutation matrix.

In order to form a permutation matrix from this $\boldsymbol{P}$ that has values other than 0 and 1 one has to modify the classical noise-free algorithm to handle noisy matrices. The simplest approach, which is used by Wong et al. [5], is to compute the matrix $\boldsymbol{P}$ and then in each row put a 1 in place of the element with the largest magnitude and replace all other elements with zeros. In other words let (j_i) denote the row index of the matrix element with the largest absolute value in the ith column of $\boldsymbol{P}$ such that

$$(j_i) = \ \arg(max(|\boldsymbol{P}(j,i)|)) \text{ where } i = [1,\ldots,K] \tag{47}$$

The following MATLAB code can be added to the previous function pairing methods

```
% Create the permutation matrix such that it consists of all
% zeros and only 1's at the maximum (j_k,k) locations
for k=1:length(T1)
    [m,ind]=max(abs(P(:,k)));
    P(:,k)=zeros(length(T1),1);
    P(ind,k)=1;
end
```

This resulting matrix will likely be a permutation matrix. The only way that this process may fail is when a permutation matrix has the maximum elements from two different columns occur in the same row. This phenomena occurs when there are larger perturbations, or low SNR, in the matrices $\boldsymbol{U}_1$ or $\boldsymbol{U}_2$. The terminology that will be used in this thesis to describe the above phenomena is a miss. The authors in [5] suggest that with a two source problem $K = 2$ a miss occurs with a near-zero probability if the computer represents numbers by more than a few bits and that if a miss should occur, however rare, that it shall be handled by performing the pairing arbitrarily, i.e. coin-toss.

For the case of noiseless or the case of Gaussian additive noise with an infinite number of snapshots the approximation of $\boldsymbol{P}$ becomes exact which is to say that the step where the maximum value in $\boldsymbol{P}$ is found is not needed.

List of References

[1] K. Wong and M. Zoltowski, “Root-MUSIC-based azimuth-elevation angle-of-arrival estimation with uniformly spaced but arbitrarily oriented velocity hydrophones,” *IEEE Transactions on Signal Processing*, vol. 47, pp. 3250–3260, December 1999.

[2] R. O. Schmidt, “Multiple emitter location and signal parameter estimation,” in *Proceedings RADC Spectrum Estimation Workshop*, Griffiths AFB, N.Y., 1979.

[3] A. Paulraj, R. Roy, and T. Kailath, “Estimation of signal parameters via rotational invariance techniques - ESPRIT,” in *Proc. Nineteenth Asilomar Conference on Circuits, Systems and Computers*, Aslomar, CA, November 1985.

[4] R. Roy, A. Paulraj, and T. Kailath, “Direction-of-arrival estimation by subspace rotation methods - ESPRIT,” *IEEE ICASSP*, pp. 2495–2498, 1986.

[5] K. Wong and M. Zoltowski, “Closed-form underwater acoustic direction finding with arbitrarily spaced vector hydrophones at unknown locations,” *IEEE Journal Of Ocean Engineering*, vol. 22, pp. 566–575, July 1997.

CHAPTER 3

Improved Eigenvalue Pairing Methods for DOA Estimation for VSAs

Recall from Chapter 2 the situation where we assume noisy data and form the resulting permutation matrices from the ESPRIT calculated results. Let $\boldsymbol{P}$ represent a 'pseudo-permutation' matrix where $\boldsymbol{P}$ is a matrix that has values other than 0 and 1, which are the only element values of a permutation matrix. In order to form a permutation matrix, $\boldsymbol{P}$ from the 'pseudo-permutation' matrix $\boldsymbol{P}$ has to be modified to take on values of 0 and 1. The easiest approach to replace elements within the matrix $\boldsymbol{P}$ to values of 0 or 1 is calculated by first computing the matrix $\boldsymbol{P}$ and then in each row put a 1 in place of the element with the largest magnitude and replace all other elements in that row with zeros. This resulting matrix will likely be a permutation matrix.

The only way that this process may fail is when a permutation matrix has the maximum elements from two different columns occur in the same row. This phenomenon, a miss, occurs when there are larger perturbations, low SNR in the data. This chapter will present the method used by Wong et al. [1] to randomly assign tied elements or what will be referred to as missed elements in this thesis, e.g., coin-toss, in the event of a miss. The following two sections present respectively a new eigenvalue pairing method based on sorting, and an exhaustive eigenvalue method, respectively.

3.1 Coin-toss Eigenvalue Pairing Method

A miss, or what is referred to as a miss in this work, according to the authors in [1] suggest that with a two source problem $K = 2$ a miss occurs with a near-zero probability if the computer represents numbers by more than a few bits and that if a miss should occur, however rare, that it shall be handled by performing the pairing arbitrarily, i.e. coin-toss. The methodology to implement a coin-toss can be best demonstrated with an example.

Let $\boldsymbol{P}$ be a matrix which is a $K \times K$ or 2×2 matrix

$$\boldsymbol{P} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \tag{48}$$

where there exists a miss in $\boldsymbol{P}$ in row 1 where there are two ones and in row 2 where there are two zeros. Once a row is identified to have a miss, an element is randomly selected with probability $1/T$ where T is the number of missed elements. After the element is selected, it is moved to an empty row, a row that has all 0s and no 1s, and that has a probability of $1/T$ as well. This is done iteratively until there exists no missed rows. In order to ensure that the permutation matrix is well formed one can multiply $\boldsymbol{P}$ by $\boldsymbol{P}^T$ which should by definition return the identity matrix $\boldsymbol{I}$.

By the nature of the problem the maximum number of elements that can be in a missed row is $T = K$; therefore, if one extends K to a much larger number there exists a condition in which T becomes much larger as well. If this occurs, then the probability of the randomly distributed coin-toss matrix $\boldsymbol{P}$ matching the real or actual permutation matrix $\boldsymbol{P}_{real}$ becomes smaller according to the following equation:

$$Pr = \frac{1}{T!} \tag{49}$$

where Pr is the probability of the matrix $\boldsymbol{P}$ matching $\boldsymbol{P}_{real}$. Thus, if K increases there is a higher probability for more mismatches.

3.2 New Approach to Eigenvalue Pairing

3.2.1 Eigenvalue Pairing for Noise-Free Real-Valued Vectors via Sorting

Given the above issues with the coin-toss method and the likelihood of a mismatch of DOAs with the coin-toss method an alternative method is proposed.

Consider two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\Re^N$ whose elements are identical up to a permutation;

$$y(i) = x(\pi(i)) \qquad i = 1, \ldots, n \tag{50}$$

where $\boldsymbol{P}$ is some permutation matrix with corresponding permutation function π. For example, consider

$$\boldsymbol{x} = \begin{bmatrix} .2 \\ .1 \\ .4 \\ .3 \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} .4 \\ .1 \\ .3 \\ .2 \end{bmatrix}. \tag{51}$$

By looking at the corresponding vectors (51) and the definition (50) the permutation function π and the permutation matrix $\boldsymbol{P}$ for these two vectors are defined as

$$\pi = \begin{bmatrix} 3 \\ 2 \\ 4 \\ 1 \end{bmatrix}, \ \boldsymbol{P} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{52}$$

The permutation function for any two such vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ may be calculated using their sorting index vectors. A sorting index vector $\boldsymbol{s}_x$ corresponding to the given vector $\boldsymbol{x}$ shows how to arrange the elements of $\boldsymbol{x}$ in ascending order:

$$\boldsymbol{x}(\boldsymbol{s}_x(i)) \leq \boldsymbol{x}(\boldsymbol{s}_x(j)) \ \text{for}\, i < j. \tag{53}$$

Likewise the sorting index vector for y is denoted by $\boldsymbol{s}_y$. The permutation that relates the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$,$\boldsymbol{y}(i) = \boldsymbol{x}(\pi(i))$, may be written in terms of the sorting index vectors $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$

$$\pi(\boldsymbol{s}_y(i)) = \boldsymbol{s}_x(i), \ i = 1, \ldots, n. \tag{54}$$

Now consider the vectors given in (51). Their sorting vectors would be

$$\boldsymbol{s}_x = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 3 \end{bmatrix}, \ \boldsymbol{s}_y = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 1 \end{bmatrix} \tag{55}$$

which, when $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$ are put with (54), yields

$$\pi(2) = 2,\ \pi(4) = 1,\ \pi(1) = 4,\ \pi(3) = 3, \tag{56}$$

which gives the correct result,

$$\pi = \begin{bmatrix} 3 \\ 2 \\ 4 \\ 1 \end{bmatrix}. \tag{57}$$

Equation (54) may be used to solve the eigenvalue pairing problem. Once again suppose that the matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ have the following eigendecompositions

$$\begin{aligned} \mathbf{F} =& \boldsymbol{T}_1 \boldsymbol{\Lambda} \boldsymbol{T}_1^{-1}, \quad & \boldsymbol{\Lambda} =& diag(\lambda_1, \ldots, \lambda_n) \\ \mathbf{G} =& \boldsymbol{T}_2 \bar{\boldsymbol{D}} \boldsymbol{T}_2^{-1}, \quad & \bar{\boldsymbol{D}} =& diag(\bar{d}_1, \ldots, \bar{d}_n) \end{aligned} \tag{58}$$

where $\boldsymbol{T}_2 = \boldsymbol{T}_1 \boldsymbol{P}$ for some permutation matrix $\boldsymbol{P}$ with corresponding permutation function π (see (30)). The diagonal elements of $\bar{\boldsymbol{D}}$ must be permuted in order to have the elements of $\bar{\boldsymbol{D}}$ correspond to the diagonal elements of $\boldsymbol{\Lambda}$. The permutation is as follows:

$$\bar{d}_i \leftrightarrow \lambda_{\pi(i)} \tag{59}$$

where the permutation function π corresponds to the permutation matrix $\boldsymbol{P}$. Consider the following matrix $\boldsymbol{M}_1$

$$\boldsymbol{M}_1 = \boldsymbol{T}_2^{-1} \boldsymbol{F} \boldsymbol{T}_2 = \boldsymbol{P}^T \boldsymbol{T}_1^{-1} \boldsymbol{F} \boldsymbol{T}_1 \boldsymbol{P} = \boldsymbol{P}^T \boldsymbol{\Lambda} \boldsymbol{P} \tag{60}$$

where $\boldsymbol{P}$ represents the permutation matrix, and $\boldsymbol{T}_2$ represents the associated eigenvectors from the eigendecomposition of the matrix $\boldsymbol{G}$. The equation shows that $\boldsymbol{M}_1$ is a diagonal matrix whose diagonal elements, m_i, are a permuted version of the diagonal elements of $\boldsymbol{\Lambda}$

$$m_i = \lambda_{\pi(i)}, \ \ i = 1, \ldots, n. \tag{61}$$

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the diagonal elements of $\boldsymbol{\Lambda}$ and $\boldsymbol{M}_1$, respectively, and let $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$ be the corresponding sorting index vectors. Once again (54) may be used to calculate the permutation function π that relates the eigenvalues of $\boldsymbol{F}$ and the eigenvalues of $\boldsymbol{M}_1$ from $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$.

Consider another matrix $\boldsymbol{M}_2$

$$\boldsymbol{M}_2 = \boldsymbol{T}_1^{-1}\boldsymbol{G}\boldsymbol{T}_1 = \boldsymbol{P}\boldsymbol{T}_2^{-1}\boldsymbol{G}\boldsymbol{T}_2\boldsymbol{P}^T = \boldsymbol{P}\bar{\boldsymbol{D}}\boldsymbol{P}^T \tag{62}$$

where $\boldsymbol{P}$ represents the permutation matrix, and $\boldsymbol{T}_1$ represents the associated eigenvectors from the eigendecomposition of the matrix $\boldsymbol{F}$. The equation shows that $\boldsymbol{M}_2$ is a diagonal matrix whose diagonal elements, $\hat{m}_i$, are a permuted version of the the diagonal elements of $\bar{\boldsymbol{D}}$. Similarly (62) can be written as

$$\bar{\boldsymbol{D}} = \boldsymbol{P}^T\boldsymbol{M}_2\boldsymbol{P} \tag{63}$$

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the diagonal elements of $\bar{\boldsymbol{D}}$ and $\boldsymbol{M}_2$, respectively, and let $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$ be the corresponding sorting vector, and then (54) may be used to find the permutation π that relates the eigenvalues of $\boldsymbol{M}_2$ and the eigenvalues of $\boldsymbol{G}$.

3.2.2 Eigenvalue Pairing for Noisy Real-Valued Vectors via Sorting

Once again consider two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ whose elements are identical up to a permutation function π that is to say

$$y(i) = \pi(x(i)) \tag{64}$$

Now suppose though that only the perturbed versions of these vectors are known,

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta x} \quad , \quad \tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{\delta y} \tag{65}$$

where $\boldsymbol{\delta x}$ and $\boldsymbol{\delta y}$ are perturbations of $\boldsymbol{x}$ and $\boldsymbol{y}$. That is to say that the two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are perturbed by some amount $\boldsymbol{\delta x}$ or $\boldsymbol{\delta y}$. Recall from (54) that the sorting vectors can be calculated from $\boldsymbol{x}$ and $\boldsymbol{y}$. Now assume that the same methodology from (53) to find the sorting vectors that are associated with $\boldsymbol{x}$ and $\boldsymbol{y}$ (54) is used to determine the sorting vectors for $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$. If the perturbations of the vectors $\boldsymbol{x}$

and $\boldsymbol{y}$, $\boldsymbol{\delta x}$ and $\boldsymbol{\delta y}$ are so small that the sorting vectors for $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ are identical for $\boldsymbol{x}$ and $\boldsymbol{y}$ then (54) yields the desired permutation function π for both sets of vectors.

Now suppose the values of a pair of elements in $\boldsymbol{x}$ or $\boldsymbol{y}$ are very close to each other such that a small perturbation $\boldsymbol{\delta x}$ or $\boldsymbol{\delta y}$ may interchange the entries in the sorting index vectors, creating a transposition in the permutation function calculated by (54). For example, take the vector $\boldsymbol{x}$ where

$$\boldsymbol{x} = \begin{bmatrix} .2 \\ .1 \\ .4 \\ .3 \end{bmatrix} , \tag{66}$$

with the associated sorting matrix and apply the perturbation δx to x such that $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta x}$

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} .15 \\ .18 \\ .41 \\ .29 \end{bmatrix} . \tag{67}$$

The sorting vector for $\tilde{\boldsymbol{x}}$ would be

$$\boldsymbol{s}_{\tilde{x}} = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 3 \end{bmatrix} \tag{68}$$

instead of the sorting vector for $\boldsymbol{x}$

$$\boldsymbol{s}_{x} = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 3 \end{bmatrix} . \tag{69}$$

This error or transposition in the sorting matrix occurs only when $\boldsymbol{\delta x}$ and $\boldsymbol{\delta y}$ are large enough to cause elements within the vector to swap places with one another. That is to say that large perturbations in $\boldsymbol{x}$ and $\boldsymbol{y}$ will cause changes in the sorting index vectors, which will cause unavoidable errors in the estimated permutation function; thus, it is proposed to use (54) even in the case of noisy vectors.

Using the aforementioned realization, a novel sorting pairing method is explored. Once again consider matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ whose eigenvectors are identical, and suppose that for both $\boldsymbol{F}$ and $\boldsymbol{G}$ only the perturbed matrices are known, $\tilde{\boldsymbol{F}} = \boldsymbol{F} + \Delta \boldsymbol{F}$ and $\tilde{\boldsymbol{G}} = \boldsymbol{G} + \Delta \boldsymbol{G}$. Let the eigenvalue decomposition of each perturbed matrix be given by

$$\begin{aligned} \tilde{\boldsymbol{F}} =& \tilde{\boldsymbol{T}}_1 \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{T}}_1^{-1} \\ \tilde{\boldsymbol{G}} =& \tilde{\boldsymbol{T}}_2 \tilde{\tilde{\boldsymbol{D}}} \tilde{\boldsymbol{T}}_2^{-1} \end{aligned} \tag{70}$$

where the diagonal elements of $\tilde{\boldsymbol{\Lambda}}$ are $\boldsymbol{x}_1$, and the diagonal elements of $\tilde{\boldsymbol{D}}$ are $\boldsymbol{x}_2$. Similarly to (53) the sorting index vectors, $\boldsymbol{s}_{x1}$ and $\boldsymbol{s}_{x2}$, are calculated from $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Recall that a sorting index vector shows how to arrange elements of a given vector, $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$, in ascending order. Also recall that the columns of $\tilde{\boldsymbol{T}}_1$ and $\tilde{\boldsymbol{T}}_2$ are the eigenvectors of $\boldsymbol{F}$ and $\boldsymbol{G}$, and that the corresponding eigenvalues are the diagonal elements of $\tilde{\boldsymbol{\Lambda}}$ and $\tilde{\tilde{\boldsymbol{D}}}$ respectively.

Calculate the matrices $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ that are formed by taking the eigenvectors $\tilde{\boldsymbol{T}}_1$ from matrix $\boldsymbol{F}$ and replacing them for the eigenvectors $\tilde{\boldsymbol{T}}_2$ found in the matrix $\boldsymbol{G}$ and the eigenvectors $\tilde{\boldsymbol{T}}_2$ from matrix $\boldsymbol{G}$ and replacing them for the eigenvectors $\tilde{\boldsymbol{T}}_2$ found in the matrix $\boldsymbol{F}$.

$$\begin{aligned} \boldsymbol{M_1} =& \tilde{\boldsymbol{T}}_2^{-1} \tilde{\boldsymbol{F}} \tilde{\boldsymbol{T}}_2 \\ \boldsymbol{M_2} =& \tilde{\boldsymbol{T}}_1^{-1} \tilde{\tilde{\boldsymbol{D}}} \tilde{\boldsymbol{T}}_1 \end{aligned} \tag{71}$$

Let $\boldsymbol{y}_1$ be the diagonal elements of $\boldsymbol{M}_1$ and $\boldsymbol{y}_2$ be the diagonal elements of $\boldsymbol{M}_2$, and let $\boldsymbol{s}_{y1}$ and $\boldsymbol{s}_{y2}$ be sorting index vectors for $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ respectively.

Once the sorting index vectors are calculated the permutation functions, π_1 and π_2 can be calculated as follows:

$$\begin{aligned} \pi_1(\boldsymbol{s}_{y1}(i)) =& \boldsymbol{s}_{x1}(i), \quad i = 1, \ldots, n \\ \pi_2(\boldsymbol{s}_{y2}(i)) =& \boldsymbol{s}_{y2}(i), \quad i = 1, \ldots, n \end{aligned} \tag{72}$$

Now that the permutation functions, π_1 and π_2, have been calculated, a test can be done to see which permutation function might be used. If the permutation functions are equal to one another, that is to say that

$$\pi_1 = \pi_2 \tag{73}$$

then the permutation function $\pi = \pi_1$. If π_1 is not equal to π_2 then one must calculate the error from each different permutation function in order to determine which permutation function to use,

$$\alpha = \sum_{i=1}^{n} \{[y_1(i) - x_1(\pi_1(i))]^2 + [y_2(i) - x_2(\pi_1(i))]^2\}^2 \tag{74}$$

$$\beta = \sum_{i=1}^{n} \{[y_1(i) - x_1(\pi_2(i))]^2 + [y_2(i) - x_2(\pi_2(i))]^2\}^2 \tag{75}$$

Once the error values α and β are calculated then the following permutation function assignment is made based on the following

$$\alpha \leq \beta \rightarrow \pi = \pi_1 \tag{76}$$

$$\alpha > \beta \rightarrow \pi = \pi_2 \tag{77}$$

The advantage of the proposed permutation pairing method via sorting is that, unlike the classical approach, the proposed eigenvalue pairing method via sorting always returns a valid permutation function.

3.3 Exhaustive Pairing Method

The most exhaustive method is to permute through all possible pairing combinations. By utilizing a multitude of options and comparing the calculated error to one another, the best permutation option can be chosen. Consider a repository matrix that contains the order of eigenvalues for n different numbers. Then the following equation summarizes the number of distinct ways in which k eigenvalues can be picked in a distinct manner:

$$_nP_k = \frac{n!}{(n-k)!}. \tag{78}$$

where the equation assumes distinct numbers. In the case of the paper, it is assumed that n is finite; therefore, k is also finite. The variations of permutation listing, i.e. (1 2); (2 1), can be defined as π_i where π is the exhaustive list of permutation matrices and i is the selection of a given permutation within the exhaustive permutation matrix.

The method described in this paper to determine the proper permutation matrix from an exhaustive matrix of permutation matrices can be found by calculating the error between one set of eigenvalues and another set of eigenvalues whose order is permuted through all possible combinations. That is to say

$$Error_i = \text{norm}((\boldsymbol{M}_1 - \boldsymbol{y}_{\pi_i})^2) + \text{norm}((\boldsymbol{M}_1 - \boldsymbol{x}_{\pi_i})^2) \quad \text{where } i = 1 \ldots K! \tag{79}$$

where K is the number of eigenvalues, $Error_i$ is the error, $\boldsymbol{F}_u$ and $\boldsymbol{F}_v$ are labeled as $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively, $\boldsymbol{M}_1$ refers to equation (60) and $\boldsymbol{M}_2$ refers to equation (62) and π_i is the unique permutation as described above.

List of References

[1] K. Wong and M. Zoltowski, "Closed-form underwater acoustic direction finding with arbitrarily spaced vector hydrophones at unknown locations," *IEEE Journal Of Ocean Engineering*, vol. 22, pp. 566–575, July 1997.

CHAPTER 4

Simulation Results

4.1 Sensor Locations

The simulation results presented in the figures below have the sensor setup that is described in [1], which consists of a 13-element irregularly spaced 3-D array of vector sensors. The array is a nine-element non-uniformly spaced cross-shaped array with elements at Cartesian coordinates: $\lambda/2 \times \{(0, 0, 0,), (\pm 1, 0, 0,), (\pm 2.7, 0, 0), (0, \pm 1, 0), (0, \pm 2.7, 0)\}$ plus a four-element square array with elements at the Cartesian coordinates $\lambda/2 \times \{(\pm 4, \pm 4, 1)\}$.

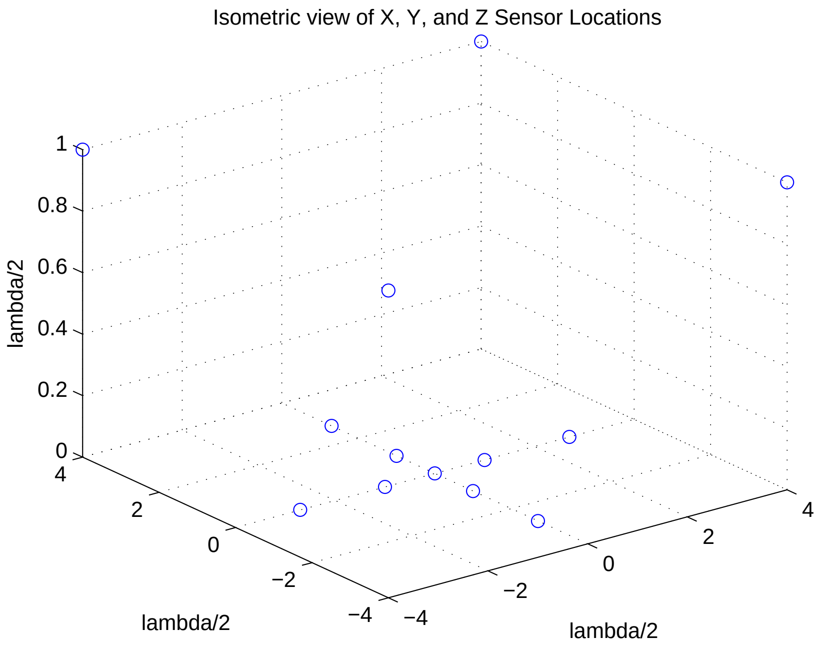

Figure 2. Isometric View of 13-Sensors in a non-uniformly spaced 3-D array at the Cartesian coordinates: $\lambda/2 \times \{(0, 0, 0,),(\pm 1, 0, 0,),(\pm 2.7, 0, 0), (0, \pm 1, 0), (0, \pm 2.7, 0)\}$ plus a four-element square array with elements at the Cartesian coordinates $\lambda/2 \times \{(\pm 4, \pm 4, 1)\}$.

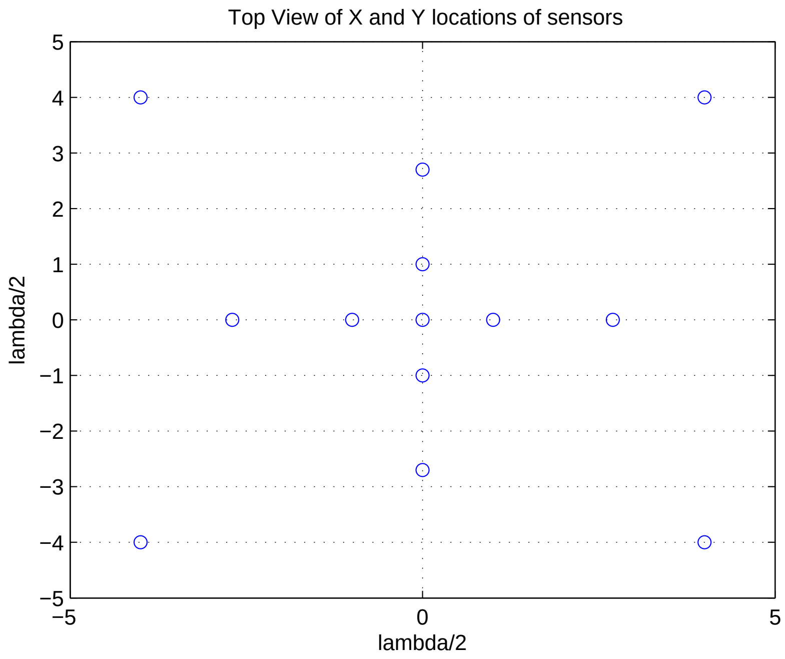

Figure 3. Top View of 13-Sensors in a non-uniformly spaced 3-D array at Cartesian coordinates $\lambda/2 \times \{(0,0,0,),(\pm1,0,0,),(\pm2.7,0,0),(0,\pm1,0),(0,\pm2.7,0)\}$ plus a four-element square array with elements at the Cartesian coordinates $\lambda/2 \times \{(\pm4,\pm4,1)\}$ This view is an X-Y Axis view with the Z-axis coming out of the page.

4.2 Number of Sources and Respective Parameters

In the results below there exist three separate scenarios consisting of 2, 3, or 4 sources, respectively, that impinge upon the sensor field. For each source there exists an angle pair, (θ,ϕ), and a direction cosine pair (u,v). Recall the relationship between the angle pair of direction cosine pair from Chapter 2:

$$\begin{bmatrix} u\,(\theta,\phi) \\ v\,(\theta,\phi) \end{bmatrix} = \begin{bmatrix} sin\,\theta \;\; cos\,\phi \\ sin\,\theta \;\; sin\;\phi \end{bmatrix}. \tag{80}$$

Note that the first two angle pairs and direction cosine pairs shown in Table 1 below are the same as those used in [1]. Following the structure laid out by Wong et al. the additional pairs are designated accordingly such that $u_2 - u_1 = v_1 - v_2 = 0.08$

or

$$u_k - u_{k-1} = v_{k-1} - v_k = 0.08 \qquad \text{where } k = 2, \ldots 4. \tag{81}$$

The following in Table 1 are the parameters used in the results below. For all cases

Table 1. Table of Parameter Values for Scenarios

Number of Sources	Angle Pairs (θ, ϕ)	Direction Cosines (u, v)
2	$(58.0700°, 44.0500°)$	$(0.61, 0.59)$
	$(59.1000°, 36.4700°)$	$(0.69, 0.51)$
3	$(58.0700°, 44.0500°)$	$(0.61, 0.59)$
	$(59.1000°, 36.4700°)$	$(0.69, 0.51)$
	$(61.8700°, 29.1800°)$	$(0.77, 0.43)$
4	$(58.0700°, 44.0500°)$	$(0.61, 0.59)$
	$(59.1000°, 36.4700°)$	$(0.69, 0.51)$
	$(61.8700°, 29.1800°)$	$(0.77, 0.43)$
	$(50.6866°, 26.8962°)$	$(0.53, 0.35)$

the signal power is set equal to unity, $P = 1$, and the sources impinge upon the previously outlined and described 13-element non-uniformly spaced 3-D array

4.3 Simulations

The simulation results in 5 through 12 illustrate the effectiveness of the presented eigenvalue pairing method (eigenvalue pairing via sorting). The figures are coupled such that all the figures for the two, three, and four source scenarios are presented together. Note that the miss rate is shown first. If the percentage of misses, previously described in Chapter 3, is too high, the classical sorting method is useless. Figures that compare the RMS standard deviation and RMS bias deviation compare three different pairing methods: 1) the traditional pairing method, the pairing method described in the [1]; 2) the new pairing method, the sorting pairing method described in this thesis; and 3) the exhaustive pairing method, a method that covers all possible permutations.

The simulations have SNR which is defined relative to each source and 100 snapshots are used in each of the 1000 independent Monte Carlo simulation experiments, i.e., 100 snapshots per experiment and 1000 independent experiments per data point.

The composite Root Mean Squared (RMS) standard deviation plotted is computed by taking the square root of the mean of the respective samples variances of $\hat{u}$ and $\hat{v}$. Likewise the composite bias is computed by taking the square root of the mean of the sample biases of $\hat{u}$ and $\hat{v}$. The classical pairing method misses are compiled together and computed to create a percent missed which is presented in its own figure %-missed. It is important to note that the data presented has 100 snapshots and 1000 independent Monte Carlo simulations. If the traditional pairing method gives away misses, its standard deviation and bias are not shown. The traditional method is considered useless in this case and the data is not presented.

4.3.1 Two Sources

The two source scenario is the same as the simulation presented in [1] as previously stated. The angle values can be found in Table 1 in the Two Source row. The results are as follows:

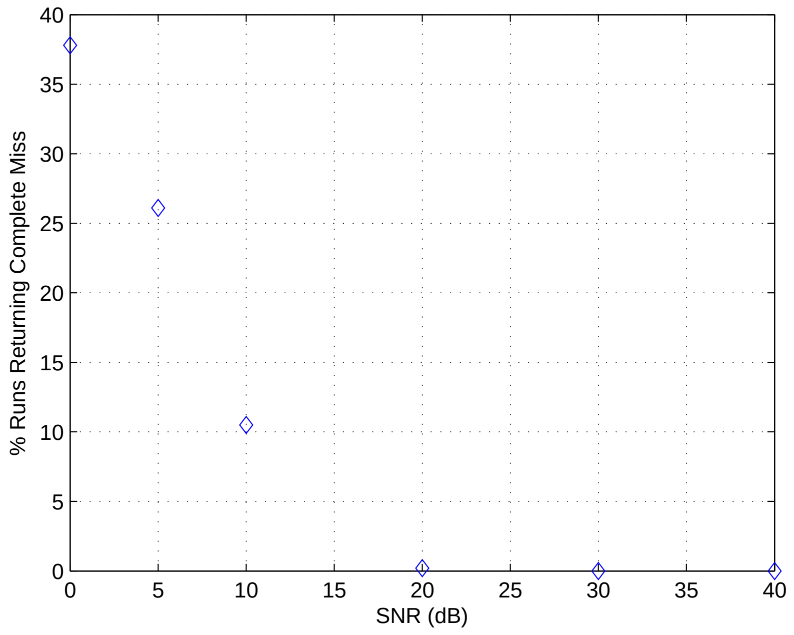

Figure 4. Miss Rate of $\{\hat{u}_1, \hat{u}_2, \hat{v}_1, \hat{v}_2\}$ for the Traditional Pairing Method (%) versus SNR (dB): two closely spaced equal-powered uncorrelated narrow-band sources with angular values found in Table 1. Note that the SNR must be greater than 20 dB for the traditional method to give valid results.

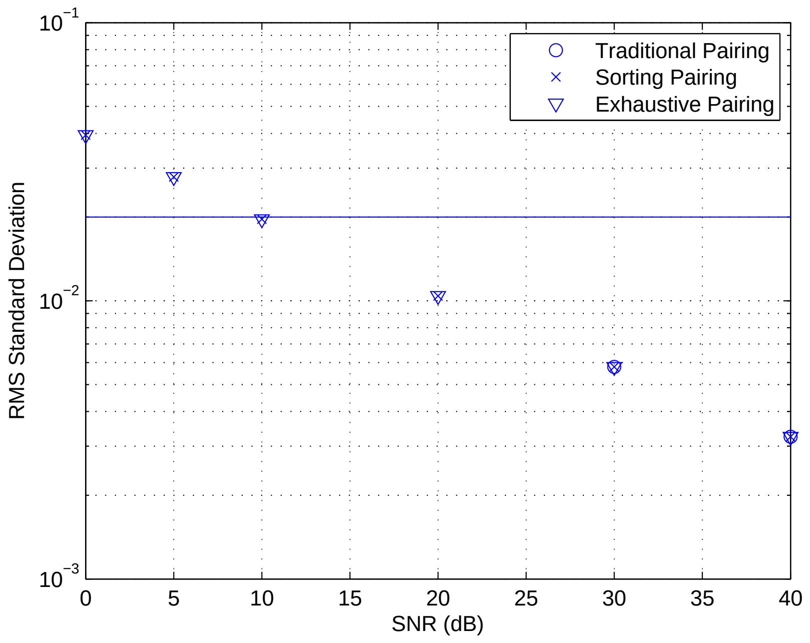

Figure 5. RMS standard deviation of $\{\hat{u}_1, \hat{u}_2, \hat{v}_1, \hat{v}_2\}$ versus SNR (dB): two closely spaced equal-powered uncorrelated narrow-band sources with angular values found in Table 1. There are three different pairing methods presented: classical, sorting, and exhaustive pairing method. The traditional method is not used below 30 dB (see Figure 4)

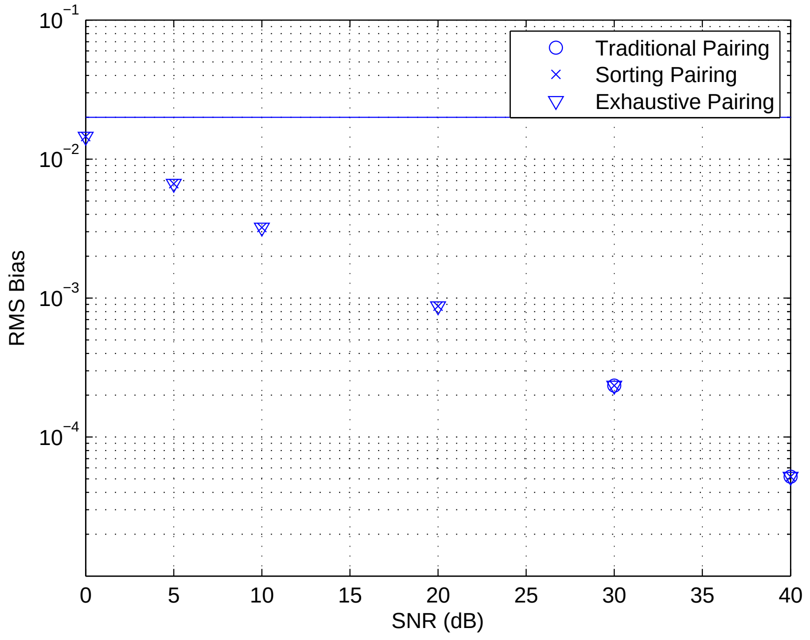

Figure 6. RMS Bias of $\{\hat{u}_1, \hat{u}_2, \hat{v}_1, \hat{v}_2\}$ versus SNR (dB): same settings as in Fig. 5

4.3.2 Three Source

The three source scenario includes the first two angle pair or direction cosine pair that the Wong et al. presents and an additional angle pair which follows the described methodology for choosing direction cosines that are separated by 0.08 by Wong et al. The angle values can be found in Table 1 in the Three Source row. The results are showing in Figures 7 - 9.

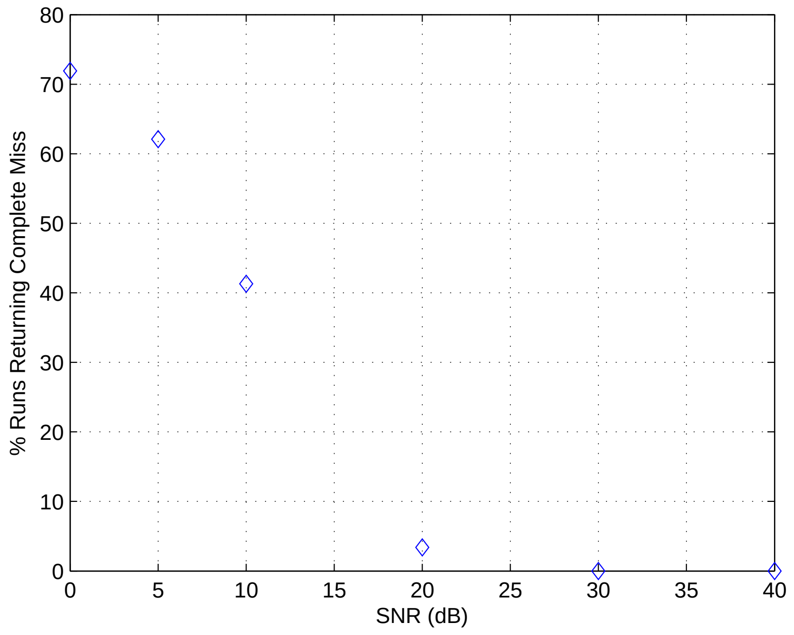

Figure 7. Miss Rate of $\{\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{v}_1, \hat{v}_2, \hat{v}_3\}$ for the Traditional Pairing Method (%) versus SNR (dB): three closely spaced equal-powered uncorrelated narrowband sources with angular values found in Table 1.

4.3.3 Four Source

The Four source scenario includes the first two angle pair or direction cosine pair that the Wong et al. presents and two additional angle pairs which follows the described methodology for choosing direction cosines that are separated by 0.08 by Wong et al. The angle values can be found in Table 1 in the Four Source row. The results are show in Figures 10 - 12.

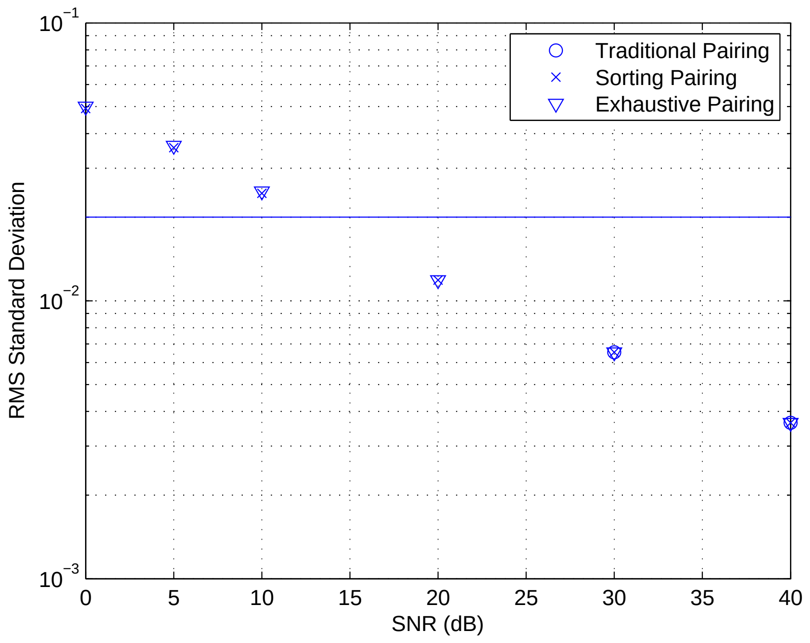

Figure 8. RMS standard deviation of $\{\hat{u}_1, \hat{u}_2, \hat{u}_3\hat{v}_1, \hat{v}_2\hat{v}_3\}$ versus SNR: three closely spaced equal-powered uncorrelated narrow-band sources with angular values found in Table 1. There are three different pairing methods presented: classical, sorting, and exhaustive pairing method. The traditional method is not used below 30 dB (see 7).

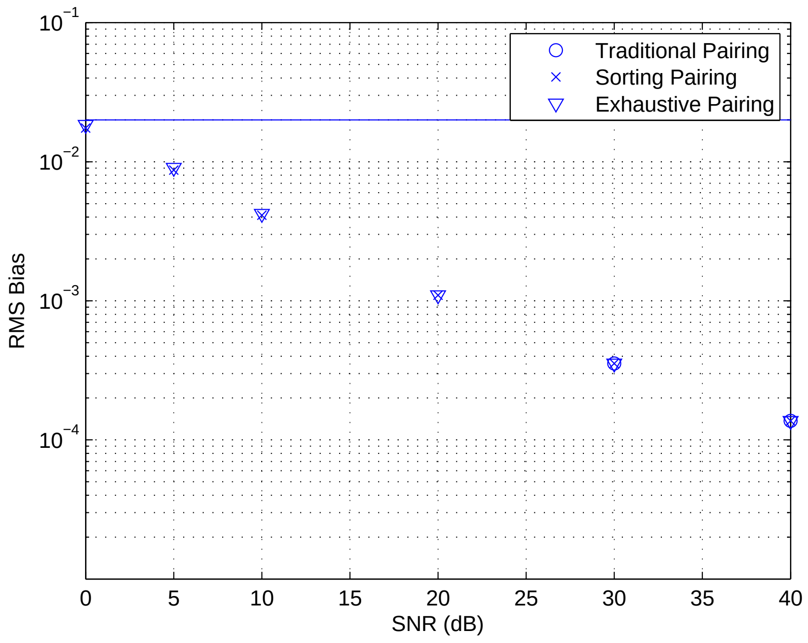

Figure 9. RMS Bias of $\{\hat{u}_1\hat{u}_2, \hat{u}_3, \hat{v}_1, \hat{v}_2, \hat{v}_3\}$ versus SNR (dB): same settings as in Fig. 8

4.4 Analysis

4.4.1 Two Source Analysis

Figures 5 and 6 plot the composite rms standard deviation and composite rms bias of two sources impinging upon a 3-Dimensional 13-element nonuniform array for the traditional, sorting, and exhaustive pairing methods. One can see that the methods of pairing, the exhaustive and the sorting pairing method are fairly co-linear which indicates that the method results from these two algorithms are similar. The traditional method has Miss % = 0 for only two SNR values as indicated by Figure 4. The number of misses made by the traditional pairing method is significantly high at lower SNRs 0 dB, 5 dB, and 10 dB at approximately 37%, 27%, and 12% respectively. The traditional, sorting and exhaustive pairing method all are co-linear at 30 and 40 dB.

Recall the relationship between $u_k - u_{k-1}$ and $v_{k-1} - v_k$ from Equation 81, where $u_k - u_{k-1} = v_{k-1} - v_k = .08$. Note that the two sources, due to this relation-

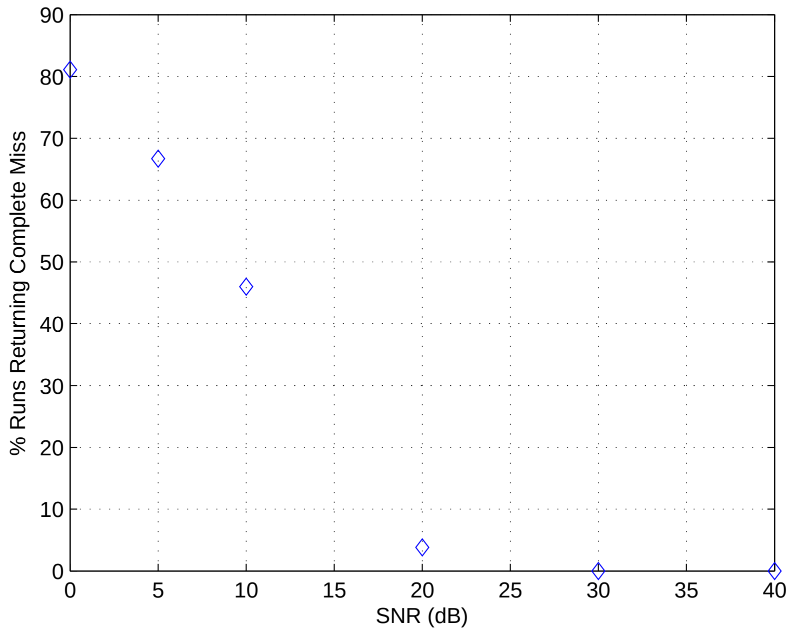

Figure 10. Miss Rate of $\{\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4\hat{v}_1, \hat{v}_2, \hat{v}_3.\hat{v}_4\}$ for the Traditional Pairing Method (%) versus SNR (dB): four closely spaced equal-powered uncorrelated narrow-band sources with angular values found in Table 1

ship, would be resolved and identified with high probability if both the estimation standard deviation and the bias are under approximately 0.02. The proposed sorting pairing method resolves these closely spaced sources for all SNR's at or above approximately 10 dB. Above the SNR resolution thresholds the estimation for both the standard deviation and bias both decrease for the both the exhaustive and the sorting pairing method fairly linearly with increasing SNR values. Since biases are typically one order of magnitude lower than the standard deviations the high biases are not the limiting factor for resolution of the estimated direction of arrival angles.

4.4.2 Three Source Analysis

Figures 8 and 9 plot the composite rms standard deviation and composite rms bias of three sources impinging upon a 3-Dimensional 13-element nonuniform array

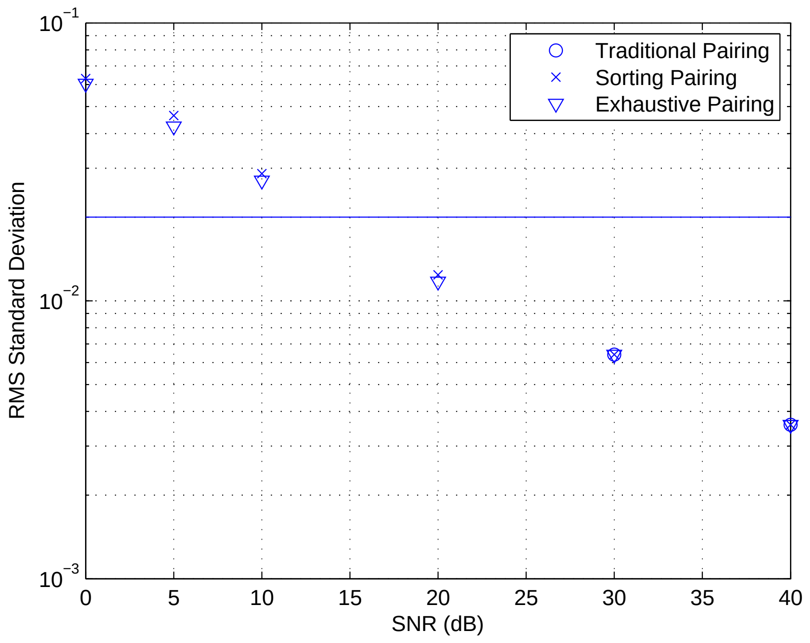

Figure 11. RMS standard deviation of $\{\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4\hat{v}_1, \hat{v}_2, \hat{v}_3.\hat{v}_4\}$ versus SNR: four closely spaced equal-powered uncorrelated narrow-band sources with angular values found in Table 1. There are three different pairing methods presented: classical, sorting, and exhaustive pairing method. The traditional method is not used below 30 dB (see Fig 10).

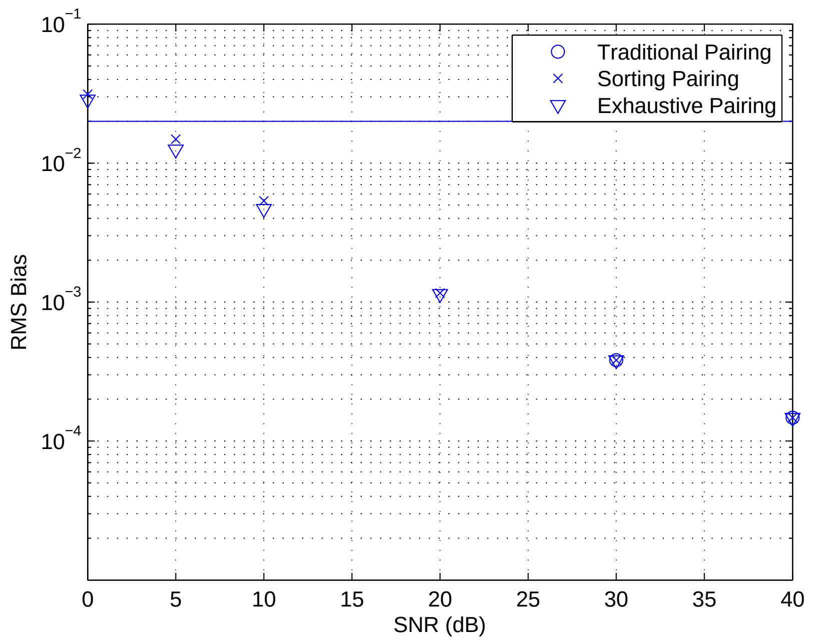

Figure 12. RMS Bias of $\{\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4\hat{v}_1, \hat{v}_2, \hat{v}_3.\hat{v}_4\}$ versus SNR (dB): same settings as in Figure 11

for the traditional, sorting, and exhaustive pairing methods. The composite rms standard deviation and composite rms bias for the sorting and exhaustive pairing method are approximately co-linear throughout all SNR, whereas the traditional method is co-linear for high SNR values for both the composite rms standard deviation and bias. The traditional method has Miss % = 0 for only two SNR values as indicated by Figure 7. The number of misses made by the traditional pairing method is significantly high at lower SNRs 0 dB, 5 dB, 10 dB, 20 dB at approximately 72%, 63%, 43%, and 3% respectively. The traditional, sorting and exhaustive pairing method all are co-linear at 30 and 40 dB.

Recall the relationship between $u_k - u_{k-1}$ and $v_{k-1} - v_k$ from Equation 81, where $u_k - u_{k-1} = v_{k-1} - v_k = .08$. Note that the three sources, due to this relationship, would be resolved and identified with high probability if both the estimation standard deviation and the bias are under approximately 0.02. The proposed sorting pairing method resolves these closely spaced sources for all SNR's at or above approximately 13 dB. Above the SNR resolution threshold the estimation for both the standard deviation and bias both decrease for both the exhaustive and the sorting pairing method fairly linearly with increasing SNR values. Since biases are typically one order of magnitude lower than the standard deviations, high biases do not matter as much as higher standard deviations.

4.4.3 Four Source Analysis

Figures 11 and 12 plot the composite rms standard deviation and composite rms bias of four sources impinging upon a 3-Dimensional 13-element nonuniform array for the traditional, sorting, and exhaustive pairing methods. The composite rms standard deviation and composite rms bias for the sorting and exhaustive pairing method are approximately co-linear with the sorting method having a higher composite RMS standard deviation and bias values than the exhaustive method. These two methods become more co-linear as SNR increases until 20 dB where the two results from the sorting and exhaustive pairing method resolve themselves to be co-linear. The traditional pairing method has Miss % = 0 for only two SNR values as indicated by Figure 10. The number of misses made by the traditional pairing method is significantly high at lower SNRs 0 dB, 5 dB, 10 dB, 20 dB at approximately 82%, 67%, 45%, and 3% respectively. All methods become co-linear as SNR increases particularly at 30 and 40 dB. Given this fact it is impressive that the difference between the traditional, and the sorting and exhaustive pairing method is not more than 0.01 at SNR values of 30 and 40 dB, and all SNR values for the sorting and exhaustive pairing method case.

Recall the relationship between $u_k - u_{k-1}$ and $v_{k-1} - v_k$ from Equation 81, where $u_k - u_{k-1} = v_{k-1} - v_k = .08$. Note that the four sources, due to this relationship, would be resolved and identified with high probability if both the estimation standard deviation and the bias are under approximately 0.02. The proposed sorting pairing method resolves these closely spaced sources for all SNRs at or above approximately 15 dB. Above the SNR resolution threshold the estimation for both the standard deviation and bias both decrease for both the exhaustive and the sorting pairing method fairly linearly with increasing SNR values. Since biases are typically one order of magnitude lower than the standard deviations, high biases do not matter as much as higher standard deviations.

4.4.4 Overall Analysis

Given the fact that the traditional method was only able to avoid producing misses at very high SNR, it is quite evident that the sorting and exhaustive method performs better than the traditional method. Due to the fact that the lower limit where the algorithm can successfully resolve closely spaced sources depends on the composite rms standard deviation value and the composite rms bias being under 0.02 it is quite evident from inspection that the sorting pairing method presented as well as the exhaustive method performed as well with SNR = 30 or 40 dB, and better in all other SNR values.

In all cases the sorting and exhaustive pairing method were relatively co-linear and would be able to resolve DOAs at approximately the same SNR. This is important to note as the exhaustive method is considered to be the best that can be achieved, which in turn means that in the presented scenarios, the sorting method would be the best pairing method out of the traditional and sorting pairing method.

List of References

[1] K. Wong and M. Zoltowski, “Closed-form underwater acoustic direction finding with arbitrarily spaced vector hydrophones at unknown locations,” *IEEE Journal Of Ocean Engineering*, vol. 22, pp. 566–575, July 1997.

CHAPTER 5

Conclusions and Recommendations for Future Work

5.1 Conclusions

Three different scenarios were presented as evidence for the case of the new eigenvalue pairing method via sorting. The results from said scenarios or case studies gives strong evidence that the sorting pairing method presents itself to be as effective as the exhaustive method for all SNR values, and as effective as the traditional pairing method for high SNR values.

Recall from Chapter 4 when there were four sources presented, that the sorting pairing method and the exhaustive pairing method achieved effectively the same composite rms standard deviation and bias values. This presents strong evidence towards saying that the sorting pairing method is nearly as effective as the exhaustive pairing method in the presented scenarios. Indeed, the traditional method failed at low SNR values as there was a high percentage of misses. This surely presents a strong case for the sorting method being more effective than the traditional method presented by Wong et al.

Finally, it is important to note that the exhaustive permutation method is computationally taxing as the number of possible permutations grows exponentially to the number of sources. Due to this fact, as the number of sources increase the number of computations must increase significantly more than the number of sources; thus, the sorting method, whose computational complexity increases approximately linearly to the number of sources, has a practical advantage.

5.2 Future Work

Future work could involve applying the same pairing method for the root-MUSIC algorithm for a similar case, but unlike the ESPRIT case where the vector sensor locations are unknown and the orientation is known, the root-MUSIC case has the sensor locations known and the orientation unknown. Despite the conclusion that the results would be likely similar in the root-MUSIC case as is found in the ESPRIT case, it would be of interest to research the future use of this sorting method in the root-MUSIC algorithm.

Another area that would be significantly interesting would be to formulate the CR-Bounds for this particular sensor setup. The lack of CR-bounds creates the ambiguity of how well the algorithm performed with the new pairing method. It would be of interest to see in future work how the results in this paper compare to the CR-bound for this field of sensors and different presented scenarios.

It is quite evident that the current algorithm has its limitations where the direction cosines can be resolved only to difference of 0.08. Therefore, it would be of great interest to research how high-resolution, a resolution smaller than 0.08 for close direction cosines, ESPRIT algorithm would perform with this new pairing method.

There are other eigenvalue pairing problems that can be found in other fields including but not limited to chemistry, physics, and engineering, it would be interesting to see whether or not any of these methods could benefit from the sorting pairing method and whether or not it would yield more accurate results.

Finally it would be interesting to retrieve a computational baseline of the ESPRIT algorithm in terms of the number of flops or pure computations that is required, particularly when it comes to the pairing schemes when dealing with eigenvalues. It would be interesting to see how the traditional, sorting, and exhaustive method would compare to one another. As it was mentioned before in Chapters 3 and 4, the computational complexity increases for the exhaustive method as the number of sources increase as well as the miss rate for the traditional pairing method. It would seem as if the sorting pairing method is somewhere in between, achieving similar composite rms standard deviations and composite rms bias as the exhaustive method but being significantly less computationally taxing. This would be of interest to scientists in the field of oceanography and signal processing where there can be a large number of sources in a large field of buoys, and when the battery life and/or processing power is limited by either the size or required longevity of the buoy.

APPENDIX

Appendix A - MATLAB Code

.1 script.m

```
% Kyle T. Martin
% 12/1/12
% script.m
%
clear all;
close all;
%
% SNR Value for the Added White Gaussian Noise in the Signal
%
SNR=[0 5 10 20 30 40];
axis_range=[min(SNR) max(SNR)];
%number of failures of traditional pairing at each SNR
miss=zeros(size(SNR));
rms_std=zeros(3,length(SNR));
rms_bias=zeros(3,length(SNR));
%
% N = Snapshots        Number_Trials= Number of Trials
N=100;                 Number_Trials=1000;
%
% theta = incident angle of arrival (elevation)
% phi = incident angle of arrival (azimuth)
% 4 angles
% comment if not needed or uncomment if needed
theta=[58.07 59.1 61.8760 39.4302];
phi=[44.05 36.47 29.1808 33.4399];
%
%
% 3 Angles
% comment if not needed or uncomment if needed
% theta=[58.07 59.1 61.8760];
% phi=[44.05 36.47 29.1808];
%
% 2 angles
% comment if not needed or uncomment if needed
% theta=[58.07 59.1];
% phi=[44.05 36.47];
%
ns=length(theta);
```

```
theta=theta*(pi/180);
phi=phi*(pi/180);
u_1=sin(theta).*cos(phi)
v_1=sin(theta).*sin(phi)
P_ext=combinator(length(u_1),length(u_1),'p')';
[u_1,ind]=sort(u_1);
v_1=v_1(ind);
%
%Locations of sensors
%
xLoc=[0 1 -1 2.7 -2.7  0   0    0     0  4   4  -4  -4];
yLoc=[0 0  0  0     0  1  -1  2.7  -2.7  4  -4   4  -4];
zLoc=[0 0  0  0     0  0   0    0     0  1   1   1   1];
%
% stores all "u" direction cosines
U=zeros(Number_Trials,ns);
% stores all "v" using traditional pairing
V1=zeros(Number_Trials,ns);
% stotes all "v" using new pairing
V2=zeros(Number_Trials,ns);
% stores all "v" using extensive pairing
V3=zeros(Number_Trials,ns);

for i = 1 : length(SNR)
  inderror=[];
  rand('seed',0);    randn('seed',0);
  for trials=1:Number_Trials;
     [z]=makeData(N,SNR(i),theta,phi,xLoc, yLoc, zLoc);
     [u1,v1,v2,v3,error]=...
 svd_results (z,length(xLoc),length(theta),theta,phi, P_ext);
     if error~=0
         miss(i)=miss(i)+1;
         inderror=[inderror;trials];
     end
     %if v2 ~= v3; keyboard;end
     U(trials,:)=u1;
     V1(trials,:)=v1;
     V2(trials,:)=v2;
     V3(trials,:)=v3;
  end
    U1=U;
    %remove all trials where traditional pairing failed
    U1(inderror,:)=[];
    %remove all trials where traditional pairing failed
    V1(inderror,:)=[];
    stdU=std(U);
    stdU1=std(U1);
    stdV1=std(V1);
    stdV2=std(V2);
    stdV3=std(V3);
    % traditional pairing
    rms_std(1,i)=sqrt(mean([stdU1.^2 stdV1.^2]));
    % new pairing
    rms_std(2,i)=sqrt(mean([stdU.^2 stdV2.^2]));
    % extensive pairing
    rms_std(3,i)=sqrt(mean([stdU.^2 stdV3.^2]));
    biasU=u_1-mean(U);
    biasU1=u_1-mean(U1);
    biasV1=v_1-mean(V1);
    biasV2=v_1-mean(V2);
    biasV3=v_1-mean(V3);
    % traditional pairing
    rms_bias(1,i)=sqrt(mean([biasU1.^2 biasV1.^2]));
    % new pairing
    rms_bias(2,i)=sqrt(mean([biasU.^2 biasV2.^2]));
    % extensive pairing
    rms_bias(3,i)=sqrt(mean([biasU.^2 biasV3.^2]));
end
miss
% figure 1
semilogy(SNR,rms_bias(1,:),'o');grid;
hold on
semilogy(SNR,rms_bias(2,:),'x');
semilogy(SNR,rms_bias(3,:),'v');
plot([axis_range], [.02 .02]);
axis([axis_range 1e-5 1e-1]);
a='Traditional Pairing'
b='Sorting Pairing'
c='Exhaustive Pairing'
legend(a,b,c);xlabel('SNR (dB)')
ylabel('RMS Bias')
% figure 2
figure
semilogy(SNR,rms_std(1,:),'o');grid;
hold on
semilogy(SNR,rms_std(2,:),'x');
semilogy(SNR,rms_std(3,:),'v');
plot([axis_range], [.02 .02]);
```

```
axis([axis_range 1e-3 1e-1]);
a='Traditional Pairing'
b='Sorting Pairing'
c='Exhaustive Pairing'
legend(a,b,c);
xlabel('SNR (dB)')
ylabel('RMS Standard Deviation')
% figure 3
figure
plot(SNR,N*miss/(Number_Trials),'d'); grid; hold on;
xlabel('SNR (dB)')
ylabel('% Runs Returning Complete Miss')
```

.2 makeData.m

```
%Kyle T. Martin
% July 20, 2010
% function to create data using makeKron.m
%and makeSignal.m functions
% N=number of snapshots
% Theta = values of theta
% Phi = value of phi
% x:the x off-set/location in amounts of lamda/2
% y:the y off-set/location in amounts of lamda/2
% z:the z off-set/location in amounts of lamda/2
%
%
function [Z]=makeData(N,SNR,theta,phi,xLoc,yLoc,zLoc)
  % Define Variable Inputs
  % Defines the Theta and Phi for X# of signal sources
  numberComponents=3; % per sensor .. i.e. 3x1 manifold
  A=makeKron(theta,phi,numberComponents, xLoc, yLoc,zLoc);
  m=size(A);
  [S,Noise] = makeSignal(SNR,N,length(theta),m(1));
  Z=A*S+Noise; % WITH NOISE
  %Z=A*S;   % NO NOISE
```

.3 makeKron

```
%Kyle T. Martin
%July 20, 2010
%function makeKron
% --Description
% This program creates the intervector hydrophone spatial
% phase factor in order to create simulated data for the
% ESPRIT Algorithm for vector sensors with arbitrarily
% spaced vector hydrophones at unknown locations
%
% The number of sound sources must be less than the number
% of elements or the co-channel signals.
%
% --Inputs
% theta: the elevation angle of the sources e.g. 45
% phi: the azimuth angle of the sources eg. 200
% numberComponents:
%the number of hydrophones in one location (3x1 or 4x1
% manifold)
% x:the x off-set/location in amounts of lamda/2
% y:the y off-set/location in amounts of lamda/2
% z:the z off-set/location in amounts of lamda/2
% lambda:the wavelength
% --Outputs
% A: matrix of intervector hydrophone spatial phase
% factor for each hydrophone based on a single source
%
function [A]=makeKron (theta,phi,numberSensors,x,y,z)
    numberSources=length(theta);
    clear j;
    if numberSensors==3
        u=sin(theta).*cos(phi);
        v=sin(theta).*sin(phi);
        A=[u;v;ones(1,numberSources)];
        for i = 1:1:length(x)
            %Q(i,:)= exp((j*2*pi*(x(i)*u+y(i)*v))/2);
            %Q(i,:)=exp(j*pi*(((x(i))*u)+((y(i))*v)));
            Q(i,:)=exp(j*pi*((x(i)*u)+(y(i)*v)));

        end
       % Q = exp(j*2*pi*(x.*u+y.*v)/2);
    elseif numberSensors==4
        u=sin(theta).*cos(phi);
        v=sin(theta).*sin(phi);
        w=cos(theta);
        A=[u;v;w;ones(1,numberSources)];
         for i = 1:1:length(x)
            Q(i,:)= exp(j*2*pi*(x(i)*u+y(i)*v+2*w)/2);
         end
      %  Q = exp(j*2*pi*(x.*u+y.*v+2.*w)/2);
    end
```

```
% Take the Kronecker product operator to determine
% the intervector hydrophone spatial phase factor
% Since amounts in Lamda/2 we can get rid of Lamda
% leaving just /2 in the exponent opposed to the above code.
A = mvkron(A,Q);
```

.4 makeSignal.m

```
function [s,n]=makeSignal(SNR,N,numberSources,m)
   %this function assumes all sources have the same SNR
    signalPower = sqrt(10^(SNR/20));
    phase=exp(i*2*pi*rand(numberSources,N));
    s=(randn(numberSources,N)+i*randn(numberSources,N))/sqrt(2);
    s=diag(signalPower)*s.*phase;
    n=(randn(m,N)+i*randn(m,N))/sqrt(2);
end
```

.5 combinator.m

```
function [A] = combinator(N,K,s1,s2)
%COMBINATOR  Perform basic permutation and combination
% samplings. COMBINATOR will return one of 4 different
% samplings on the set 1:N,  taken K at a time.
% These samplings are given as follows:
%
% PERMUTATIONS WITH REPETITION/REPLACEMENT
%   COMBINATOR(N,K,'p','r')  --  N >= 1, K >= 0
% PERMUTATIONS WITHOUT REPETITION/REPLACEMENT
%   COMBINATOR(N,K,'p')  --  N >= 1, N >= K >= 0
% COMBINATIONS WITH REPETITION/REPLACEMENT
%   COMBINATOR(N,K,'c','r')  --  N >= 1, K >= 0
% COMBINATIONS WITHOUT REPETITION/REPLACEMENT
%   COMBINATOR(N,K,'c')  --  N >= 1, N >= K >= 0
%
% Example:
%
% To see the subset relationships, do this:
% Permutations with repetition
%     combinator(4,2,'p','r')
% Permutations without repetition
%     combinator(4,2,'p')
% Combinations with repetition
%     combinator(4,2,'c','r')
 % Combinations without repetition
%     combinator(4,2,'c')
```

```
%
% If it is desired to use a set other than 1:N, simply use the
% output from COMBINATOR as an index into the set of interest.
% For example:
%
%    MySet = ['a' 'b' 'c' 'd'];
%    MySetperms = combinator(length(MySet),3,'p','r');
%  % Take 3 at a time.
%    MySetperms = MySet(MySetperms)
%
%
%    Class support for input N:
%       float: double, single
%       integers: int8,int16,int32
%
%
% Notes:
% All of these algorithms have the potential to create VERY
% large outputs.In each subfunction there is an anonymous function
% which can be used to calculate the number of row which will
% appear in the output.  If a rather large output is expected,
% consider using an integer class to conserve memory.  For example:
%
%          M = combinator(int8(30),3,'p','r');  % NOT uint8(30)
%
% will take up 1/8 the memory as passing the 30 as a double.
% See the note below on using the MEX-File.
%
% To make your own code easier to read, the fourth argument
% can be any string.  If the string begins with an 'r' (or 'R'),
% the function will be called with the replacement/repetition
% algorithm.  If not, the string will be ignored.  For instance,
% you could use:  'No replacement', or 'Repetition allowed' If
% only two inputs are used, the function will assume 'p','r'.
% The third argument must begin with either a 'p' or a 'c' but
% can be any string beyond that.
%
% The permutations with repetitions algorithm uses cumsum.
% So does the combinations without repetition algorithm for
% the special case of K=2. Unfortunately, MATLAB does not allow
% cumsum to work with integer classes. Thus a subfunction has been
% placed at the end for the case when these classes are passed.
% The subfunction will automatically pass the necessary matrix to
% the built-in cumsum when a single or double is used. When an
% integer class is used, the subfunction first looks to see if the
% accompanying MEX-File (cumsumall.cpp) has been compiled.  If not,
% then a MATLAB For loop is used to perform the cumsumming.  This is
% VERY slow!  Therefore it is recommended to compile the MEX-File
% when  using integer classes.
% The MEX-File was tested by the author using the
% Borland 5.5 C++ compiler.
%
% See also, perms, nchoosek, npermutek (on the FEX)
%
% Author:   Matt Fig
% Contact:  popkenai@yahoo.com
% Date:     5/30/2009
%
% Reference:  http://mathworld.wolfram.com/BallPicking.html

ng = nargin;

if ng == 2
    s1 = 'p';
    s2 = 'r';
elseif ng == 3
    s2 = 'n';
elseif ng ~= 4
    error('Only 2, 3 or 4 inputs are allowed.  See help.')
end

if isempty(N) || K == 0
   A = [];
   return
elseif numel(N)~=1 || N<=0 || ~isreal(N) || floor(N) ~= N
    error('N should be one real, positive integer. See help.')
elseif numel(K)~=1 || K<0 || ~isreal(K) || floor(K) ~= K
    error('K should be one real non-negative integer. See help.')
end

STR = lower(s1(1)); % We are only interested in the first letter.

if ~strcmpi(s2(1),'r')
    STR = [STR,'n'];
else
   STR = [STR,'r'];
end
```

```
try
    switch STR
        case 'pr'
            A = perms_rep(N,K);     % strings
        case 'pn'
            A = perms_no_rep(N,K);  % permutations
        case 'cr'
            A = combs_rep(N,K);     % multichoose
        case 'cn'
            A = combs_no_rep(N,K);  % choose
        otherwise
            error('Unknown option passed.  See help')
    end
catch
    rethrow(lasterror) % Throw error from here, not subfunction.
  %The only error thrown should be K>N for non-replacement calls.
end

function PR = perms_rep(N,K)
% This is (basically) the same as npermutek found on the FEX.
% It is the  fastest way to calculate these (in MATLAB) that
% I know.  pr = @(N,K) N^K;  Number of rows.
% A speed comparison could be made with COMBN.m, found on the
% FEX.  This is an excellent code which uses ndgrid.
% COMBN is written by Jos.
%
%  % All timings represent the best of 4 consecutive runs.
%  % All timings shown in subfunction notes used
%  % this configuration:
%  % 2007a 64-bit, Intel Xeon, win xp 64, 16 GB RAM
%  tic,Tc = combinator(single(9),7,'p','r');toc
%  %Elapsed time is 0.199397 seconds.  Allow Ctrl+T+C+R on block
%  tic,Tj = combn(single(1:9),7);toc
%  %Elapsed time is 0.934780 seconds.
%  isequal(Tc,Tj)  % Yes

if N==1
   PR = ones(1,K,class(N));
   return
elseif K==1
    PR = (1:N).';
    return
end
```

```
CN = class(N);
M = double(N);  % Single will give us trouble on indexing.
L = M^K;  % This is the number of rows the outputs will have.
PR = zeros(L,K,CN);  % Preallocation.
D = ones(1,N-1,CN);  % Use this for cumsumming later.
LD = M-1;  % See comment on N.
VL = [-(N-1) D].';  % These values will be put into PR.
% Now start building the matrix.
TMP = VL(:,ones(L/M,1,CN));  % Instead of repmatting.
PR(:,K) = TMP(:);  % We don't need to do two these in loop.
PR(1:M^(K-1):L,1) = VL;  % The first column is the simplest.
% Here we have to build the cols of PR the rest of the way.
for ii = K-1:-1:2
% Indices into the rows for this col.
    ROWS = 1:M^(ii-1):L;
% Match dimension.
    TMP = VL(:,ones(length(ROWS)/(LD+1),1,CN));
% Build it up, insert values.
    PR(ROWS,K-ii+1) = TMP(:);
end

PR(1,:) = 1;  % For proper cumsumming.
PR = cumsum2(PR);  % This is the time hog.
%
function PN = perms_no_rep(N,K)
% Subfunction: permutations without replacement.
% Uses the algorithm in combs_no_rep as a basis,
% then permutes each row.
% pn = @(N,K) prod(1:N)/(prod(1:(N-K)));
if N==K
    PN = perms_loop(N);  % Call helper function.
%     [id,id] = sort(PN(:,1));
%#ok  Not nec. uncomment for nice order.
%     PN = PN(id,:);  % Return values.
    return
elseif K==1
    PN = (1:N).';  % Easy case.
    return
end

if K>N
% Since there is no replacement, this cannot happen.
  error(['When no repetitions are allowed, '...
   'K must be less than or equal to N'])
```

```
end

M = double(N);  % Single will give us trouble on indexing.
WV = 1:K;  % Working vector.
lim = K;   % Sets the limit for working index.
inc = 1;   % Controls which element of WV
BC = prod(M-K+1:M);  % Pre-allocation of return arg.
BC1 = BC / ( prod(1:K)); % Number of comb blocks.
PN = zeros(round(BC),K,class(N));
L = prod(1:K) ;  % To get the size of the blocks.
cnt = 1+L;
P = perms_loop(K);  % Only need to use this once.
PN(1:(1+L-1),:) = WV(P);  % The first row.

for ii = 2:(BC1 - 1);
    if logical((inc+lim)-N)
    % The logical is nec. for class single
        stp = inc;  % This is where the for loop below stops.
        flg = 0;  % Used for resetting inc.
    else
        stp = 1;
        flg = 1;
    end
    for jj = 1:stp
        WV(K  + jj - inc) = lim + jj;
        % Faster than a vector assignment!
    end
    %
    PN(cnt:(cnt+L-1),:) = WV(P);  % Assign block.
    cnt = cnt + L;  % Increment base index.
    inc = inc*flg + 1;  % Increment the counter.
    lim = WV(K - inc + 1 );  % lim for next run.
end
V = (N-K+1):N;  % Final vector.
PN(cnt:(cnt+L-1),:) = V(P);  % Fill final block.
% The sorting below is NOT necessary.  If you prefer this nice
% order, the next two lines can be un-commented.
% [id,id] = sort(PN(:,1));
% PN = PN(id,:);  % Return values.

function P = perms_loop(N)
% Helper function to perms_no_rep.  This is
% basically the same as the  MATLAB function perms.
% It has been un-recursed for a runtime of around
% half the recursive version found in perms.m  For example:
%
%      tic,Tp = perms(1:9);toc
%      %Elapsed time is 0.222111 seconds.
%      tic,Tc = combinator(9,9,'p');toc
%      %Elapsed time is 0.143219 seconds.
%      isequal(Tc,Tp)  % Yes

M = double(N); % Single will give us trouble on indexing.
P = 1;  % Initializer.
G = cumprod(1:(M-1));  % Holds the sizes of P.
CN = class(N);

for n = 2:M
    q = P;
    m = G(n-1);
    P = zeros(n*m,n,CN);
    P(1:m, 1) = n;
    P(1:m, 2:n) = q;
    a = m + 1;

    for ii = n-1:-1:1,
        t = q;
        t(t == ii) = n;
        b = a + m - 1;
        P(a:b, 1) = ii;
        P(a:b, 2:n) = t;
        a = b + 1;
    end
end
function CR = combs_rep(N,K)
% Subfunction multichoose:  combinations with replacement.
% cr = @(N,K) prod((N):(N+K-1))/(prod(1:K)); Number of rows.

M = double(N);  % Single will give us trouble on indexing.
WV = ones(1,K,class(N));  % This is the working vector.
mch = prod((M:(M+K-1)) ./ (1:K));  % Pre-allocation.
CR = ones(round(mch),K,class(N));

for ii = 2:mch
    if WV(K) == N
        cnt = K-1;  % Work backwards in WV.
        while WV(cnt) == N
            cnt = cnt-1;  % Work backwards in WV.
        end
        WV(cnt:K) = WV(cnt) + 1;  % Fill forward.
    else
        WV(K) = WV(K)+1;   % Keep working in this group.
    end
    CR(ii,:) = WV;
end

function CN = combs_no_rep(N,K)
% Subfunction choose:  combinations w/o replacement.
% cn = @(N,K) prod(N-K+1:N)/(prod(1:K));  Number of rows.
% Same output as the MATLAB function nchoosek(1:N,K),
% but often faster for larger N.
% For example:
%
%       tic,Tn = nchoosek(1:17,8);toc
%       %Elapsed time is 0.430216 seconds.
%       tic,Tc = combinator(17,8,'c');toc
%       %Elapsed time is 0.024438 seconds.
%       isequal(Tc,Tn)  % Yes

if K>N
    error(['When no repetitions are allowed, '...
           'K must be less than or equal to N'])
end

M = double(N);  % Single will give us trouble on indexing.

if K == 1
   CN =(1:N).';  % These are simple cases.
   return
elseif K == N
    CN = (1:N);
    return
elseif K==2 && N>2 %This is an easy case to do quickly
    BC = (M-1)*M / 2;
    id1 = cumsum2((M-1):-1:2)+1;
    CN = zeros(BC,2,class(N));
    CN(:,2) = 1;
    CN(1,:) = [1 2];
    CN(id1,1) = 1;
    CN(id1,2) = -((N-3):-1:0);
    CN = cumsum2(CN);
    return
```

```
end

WV = 1:K; % Working vector.
lim = K;  % Sets the limit for working index.
inc = 1;  % Controls which element of WV is being worked on
BC = prod(M-K+1:M) / (prod(1:K));  % Pre-allocation.
CN = zeros(round(BC),K,class(N));
CN(1,:) = WV;  % The first row.

for ii = 2:(BC - 1);
    if logical((inc+lim)-N)
    % The logical is nec. for class single(?)
        stp = inc;  %This is where the for loop below stops
        flg = 0;  % Used for resetting inc.
    else
        stp = 1;
        flg = 1;
    end

    for jj = 1:stp
        WV(K  + jj - inc) = lim + jj;
        % Faster than a vector assignment.
    end

    CN(ii,:) = WV;  % Make assignment.
    inc = inc*flg + 1;  % Increment the counter.
    lim = WV(K - inc + 1 );  % lim for next run.
end

CN(ii+1,:) = (N-K+1):N;

function A = cumsum2(A)
%CUMSUM2, works with integer classes.
% Duplicates the action of cumsum, but for integer classes
% If Matlab ever allows cumsum to work for integer classes
% we can remove this.

if isfloat(A)
    A = cumsum(A);  % For single and double, use built-in
    return
else
 try
   A = cumsumall(A);  % User has the MEX-File ready?
 catch
```

```
  warning('Cumsum by loop MEX cumsumall.cpp for speed.')
  for ii = 2:size(A,1)
      A(ii,:) = A(ii,:) + A(ii-1,:);
  end
 end
end
```

.6 svdResults.m

```
% Kyle T. Martin
% svd_results.m
%
%
function [u1,v1,v2,v3,error]=...
svd_results (z,L,K,theta,phi,P_ext)
[u,s,v]=svd(z);% could use [u,e]=eig(z*z');
u_1=u(1:L,1:K);
u_2=u(L+1:2*L,1:K);
u_3=u(2*L+1:3*L,1:K);
%
F1=u_3\u_1;
F2=u_3\u_2;

%
%****
% PAIRING METHODS
%****
%
[u1,v1,v2,v3,error]=pairing_methods(F1,F2,P_ext);
%
```

.7 pairing_methods.m

```
%
% This code executes code to provide pairing methods
% and utilizes pre-made exhaustive permutation matrix
% 'P_ext'.  This code also executes the code
% to do sorting and traditional pairing methods
%
%
function [u1,v1,v2,v3,error]=pairing_methods(F1,F2,P_ext)
%v3=0;
error=0;
% Traditional pairing method
[T1,Db_1]=eig(F1);
```

```
[T2,Db_2]=eig(F2);
u1=diag(real(Db_1))';
v1=real(Db_2);
[u1,ind]=sort(u1);
T1=T1(:,ind);
%
P=T1\T2;
ns=length(T1);
for k=1:ns
    [m,ind]=max(abs(P(:,k)));
    P(:,k)=zeros(length(T1),1);
    P(ind,k)=1;
end
if rank(P)<ns
    error=1;
    v1=zeros(1,ns);
else
    v1=diag(P*v1*P')';
end
% Sorting pairing method
v2=diag(real(Db_2));
perm=pair_evals(F2,T2,v2,F1,T1,u1');
v2=v2(perm)';
m2=real(diag(T2\F1*T2));
m1=real(diag(T1\F2*T1));
y=u1';
%Exhaustive Pairing Method
v3=diag(real(Db_2));
for i = 1:length(P_ext(1,:))
   error_1(i)=...
(norm(m1-v3(P_ext(:,i)'))^2)+norm(y-m2(P_ext(:,i)))^2;
end
perm_1=P_ext(:,find(error_1==min(error_1)));
v3=v3(perm_1)';
```

.8 pair_evals.m

```
function perm=pair_evals(F,T1,x,G,T2,y)
% x is a vector of real-valued eigenvalues of
% F with corresponding e-vectors cols of T1
%
% y is a vector of real-valued eigenvalues of
% G with corresponding e-vectors cols of T2
% the result of the function is that y(i) is
% paired with x(perm(i))
m1=real(diag(T2\F*T2));
m2=real(diag(T1\G*T1));
[x1,indx]=sort(x);
[y1,indy]=sort(y);
[m11,indm1]=sort(m1);
[m12,indm2]=sort(m2);
p1=findperm(indm1,indx);
p2=findperm(indy,indm2);
% u=(x+m1)/2;
% v=(y+m2)/2;
perm=p1;
if norm(p1-p2)>0
    a=norm(m1-x(p1))^2+norm(y-m2(p1))^2;
    b=norm(m1-x(p2))^2+norm(y-m2(p2))^2;
    if a>b
        perm=p2;
    end
end
```

.9 findperm.m

```
function perm=findperm(x,y)
% x and y contain identical real-valued elements
% up to a permutation
%the result is that y(i) = x(perm(i))
n=length(y);
perm=zeros(n,1);
for k=1:n
    perm(k)=find(y(k)==x);
end
```

.10 recover.m

```
% Kyle T. Martin
% recover.m
%
% Recovers theta and phi from the direction cosine
% values using the following equations:
% a=sin(theta)cos(phi)
% b=sin(theta)sin(phi)
%
% b=sin(theta)sin(phi)   b   sin(phi)
% -------------------- = - = ------- = tan(phi)
% a=sin(theta)cos(phi)   a   cos(phi)
```

```
%
% inverseTangent(b/a) = phi
%
% a/cos(phi)=sin(theta)
% inverseSin(a/cos(phi))=theta
%
function [theta_recovered,phi_recovered]=...
         recover(a,b)
theta_recovered=asin(sqrt(a^2+b^2));
phi_recovered=atan(b/a);
```

.11 mvkron.m

```
% Executes kronecker product
function K=mvkron(A,B)
  [ma,na]=size(A);
  [mb,nb]=size(B);
  jab=1:na;
  t=0:(ma*mb-1);
  ia=fix(t/mb)+1;
  ib=rem(t,mb)+1;
  K=A(ia,jab).*B(ib,jab);
```

BIBLIOGRAPHY

Proceedings Workshop Directional Acoustic Sensors (CD-ROM), Newport, RI, 2001.

Abdi, A., Guo, H., and Suthiwan, P., "A new vector sensor receiver for underwater acoustic communication."

Gabrielson, T. B., "Design problems and limitations in vector sensors," in *Proceedings Workshop Directional Acoustic Sensors (CD-ROM)*, Newport, RI, 2001.

Hawkes, M. and Nehorai, A., "Acoustic vector-sensor beamforming and capon direction estimation," *IEEE Transactions on Signal Processing*, vol. 46, pp. 2291–2304, Sept. 1998.

Higgins, M., "DIFAR system overview," in *Proceedings Workshop Directional Acoustic Sensors (CD-ROM)*, Newport, RI, 2001.

Nehorai, A. and Paldi, E., "Acoustic vector-sensor array processing," *IEEE Transactions on Signal Processing*, vol. 42, pp. 2481–2491, 1994.

Olson, H. F., "Mass controlled electrodynamic microphones: The ribbon microphone," *Journal Acoustical Society America*, vol. 3, pp. 56–58, 1931.

Paulraj, A., Roy, R., and Kailath, T., "Estimation of signal parameters via rotational invariance techniques - ESPRIT," in *Proc. Nineteenth Asilomar Conference on Circuits, Systems and Computers*, Aslomar, CA, November 1985.

Paulraj, A., Roys, R., and Kailath, T., "Estimation of Signal Parameters Via Rotational Invariance Techniques ESPRIT," in *Proc. Nineteenth Asilomar Conference on Circuits, Systems and Computers*, Asilomar, CA, November 1985.

Roy, R., Paulraj, A., and Kailath, T., "Direction-of-arrival estimation by subspace rotation methods - ESPRIT," *IEEE ICASSP*, pp. 2495–2498, 1986.

Schmidt, R. O., "Multiple emitter location and signal parameter estimation," in *Proceedings RADC Spectrum Estimation Workshop*, Griffiths AFB, N.Y., 1979.

Silvia, M. T. and Richards, R., "A theoretical and experimental investigation of low-frequency acoustic vector sensors," in *Proceedings Oceans '02*, Beloxi, MS, 2002.

Wong, K. and Zoltowski, M., "Orthogonal-velocity-hydrophone esprit for sonar source localization," *MTS/IEEE Oceans 96 Conference*, vol. 3, pp. 1307–1312.

Wong, K. and Zoltowski, M., "Closed-form underwater acoustic direction finding with arbitrarily spaced vector hydrophones at unknown locations," *IEEE Journal Of Ocean Engineering*, vol. 22, pp. 566–575, July 1997.

Wong, K. and Zoltowski, M., "Root-MUSIC-based azimuth-elevation angle-of-arrival estimation with uniformly spaced but arbitrarily oriented velocity hydrophones," *IEEE Transactions on Signal Processing*, vol. 47, pp. 3250–3260, December 1999.