Technical University of Crete

Department of Electronic and Computer Engineering

Implementation in C of the algorithm that maximizes a quadratic form with a binary vector

Dissertation Thesis

Anastasia Barkalaki

November 29, 2009

Abstract

The maximization of a full rank quadratic form over the binary alphabet can be implemented in exponential complexity exhaustive search. It is proved that if the rank of the form is not a function of the problem size then it can be maximized in polynomial time. The binary vector that maximizes the quadratic form is testified to reside to the polynomial size set of candidate vectors. Therefore the candidate set is reduced from exponential to polynomial. We develop an algorithm, which is implemented in C that estimates the polynomial size candidate set in polynomial time and show that it is fully parallelizable and rank scalable. We implement the algorithm parallel by executing in a lot of processors simultaneously and as a result the execution time is improved according to the number of processors that are used. We compare this algorithm with sphere decoding and observe the distinctions. Finally we demonstrate the efficiency of the proposed algorithm in the context of multiple input multiple output signal detection by nesting the algorithm in an OSTBC system.

Contents

Chapter 1

Introduction

The maximization of a positive (semi) definite quadratic form that consists of a matrix parameter and a vector argument is common design problem in communications systems that appear at both the transmitter (signal design) and the receiver (signal processing) end. It must be illuminated that the complexity of such an optimization is determined by the characteristics of the matrix parameter (whose rank specifies the quadratic form) as well as the alphabet of the vector argument.

For example, if the alphabet of the vector argument is unconstrained, then the quadratic form is maximized by the maximum eigenvalue eigenvector of the matrix parameter. However, maximization of a full rank quadratic form over the binary alphabet is NP-hard in both a worst case and an average sense.

In digital communications, positive (semi) definite quadratic form maximization often involves a binary vector argument. Prime examples include maximum signal-to-noise ratio (SNR) spreading code design [5], [6], maximum likelihood signal detection in multiple input multiple output (MIMO) systems, and ML block no coherent detection of MPSK signals [9]. Furthermore, integer least - squares optimization [1], [10], is equivalent to positive (semi) definite quadratic form maximization, when the vector argument is binary.

Because of the exponential complexity of all the above problems in the general case, several reduced-complexity alternative approaches have appeared in the literature recently. Sphere decoding provides the exact ML solution at low computational cost for sufficiently high SNR and short vector argument. Finally a polynomial com-

plexity optimal method was developed for a special case of quadratics forms.

It has proved that the maximization of a quadratic form with a binary vector argument is no longer NP-hard on condition that the rank of the form is not a function of the problem size. Firstly it was developed an algorithm which computes with log-linear complexity the binary vector that maximizes a rank-2 quadratic form. The expansion was following by espousing the same idea and was concerned the maximization of a rank-3 quadratic form, constructing an algorithm that computes the optimal binary vector in log-quadratic time. It was implemented by utilizing auxiliary spherical coordinates and partitioning the three- dimensional space into a quadratic-size set of regions, where each region corresponds to a distinct binary vector. The optimal binary vector is testified to belong to the quadratic-size set of candidate vectors. Therefore the method reduces the size of the candidate vector set from exponential to quadratic. On the other hand, Computational Geometry adopts a different prespective about the maximization of any reduced-rank quadratic form over the 0/1 field and uses a variety of CG algorithms such as reverse search [3], [2]. This algorithm is optimal in terms of speed and memory efficient.

Now we are going to describe an analytic procedure to generalize the approach [4] and implement the algorithm in C in order to compute the binary vector that maximizes a reduced rank quadratic form. We introduce as many auxiliary spherical coordinates as the rank of the problem reduced by one and partition the multidimensional space into a polynomial-size set of regions. Each region is in-wrought with a distinct binary vector. The set of binary vectors we obtain has the same size as the set produced by the reverse search. However the set is constructed in a completely different manner in resulting time and memory issues. The algorithm that we are going to present is fully parallelizable and rank scalable.

Chapter 2

Problem Statement

We consider the quadratic form

$$\mathbf{x}^T\mathbf{A}\mathbf{x} \tag{2.1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a symmetric matrix and $\mathbf{x} \in \{\pm 1\}^N$ is a binary vector argument. Since $\mathbf{A}$ is symmetric, it can be decomposed as

$$\mathbf{A} = \sum_{n=1}^{N} \lambda_n \mathbf{q}_n \mathbf{q}_n^T, \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N, \quad \|\mathbf{q}_n\| = 1, \quad \mathbf{q}_n^T\mathbf{q}_k = 0, \quad n \neq k \tag{2.2}$$

$$n, k = 1, 2, \ldots, N,$$

where λ_n and $\mathbf{q}_n$ are its nth eigenvalue and eigenvector.

We are interested in computing the binary vector that maximizes the quadratic form

$$\mathbf{x}_{\text{opt}} \stackrel{\triangle}{=} \arg \max_{\mathbf{x}\in\{\pm 1\}^N} \mathbf{x}^T\mathbf{A}\mathbf{x}. \tag{2.3}$$

Without loss of generality we assume that $\lambda_N = 0$. Indeed, if $\lambda_N \neq 0$, then $\mathbf{A}$ can be substituted by $\mathbf{A} - \lambda_N\mathbf{I}$ so that the quadratic forms $\mathbf{x}^T(\mathbf{A} - \lambda_N\mathbf{I})\mathbf{x} = \mathbf{x}^T\mathbf{A}\mathbf{x} - N\lambda_N$ and $\mathbf{x}^T\mathbf{A}\mathbf{x}$ are maximized by the same binary vector and the minimum eigenvalue of $\mathbf{A} - \lambda_N\mathbf{I}$ equals zero. Therefore, in the following, $\mathbf{A}$ is assumed semidefinite positive with rank $D \leq N - 1$, i.e. $\mathbf{A} = \sum_{n=1}^{D} \lambda_n \mathbf{q}_n \mathbf{q}_n^T$, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D > 0$. we define the weighted principal component

$$\mathbf{v}_n \stackrel{\triangle}{=} \sqrt{\lambda_n}\mathbf{q}_n, \quad n = 1, 2, \ldots, D, \tag{2.4}$$

and the corresponding $N \times D$ matrix

$$\mathbf{V} \stackrel{\triangle}{=} [\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_D] \tag{2.5}$$

such that $\mathbf{A} = \mathbf{V}\mathbf{V}^T$ and

$$\mathbf{x}_{\text{opt}} = \arg \max_{\mathbf{x} \in \{\pm 1\}^N} \{\mathbf{x}^T \mathbf{V}\mathbf{V}^T \mathbf{x}\}. \tag{2.6}$$

Notice that $\mathbf{V}$ is full-rank and matrices $\mathbf{A}$ and $\mathbf{V}$ have the same rank $D \leq N-1$.

If $D = N-1$, then the computation is NP-hard and can be implemented by exhaustive search among all elements of $\{\pm 1\}^N$ with complexity $\mathcal{O}(2^N)$ since $|\{\pm 1\}^N| = 2^N$.However, if D is not a function of N, then we can achieve solutions with lower complexity.For example, if $D = 1$, then $\mathbf{x}_{\text{opt}} = \text{sgn}(q_1)$ where $\text{sgn}(\cdot)$ denotes the vector sign operation. If $D = 2$ (so, $\mathbf{V}$ has size $N \times 2$), then it has been shown that there exists a set $\mathcal{X}(\mathbf{V}_{N\times 2}) \subset \{\pm 1\}^N$ which has cardinality $|\mathcal{X}(\mathbf{V}_{N\times 2})| = N$ and is constructed with complexity $\mathcal{O}(N \log N)$ such that $\mathbf{x}_{\text{opt}}$ can be efficiently computed by numerical comparison of $\mathbf{x}^T\mathbf{A}\mathbf{x}$ among the elements of $\mathcal{X}(\mathbf{V}_{N\times 2})$. Therefore, maximization of a rank-2 quadratic form $\mathbf{x}^T\mathbf{A}\mathbf{x}$ with a binary argument $\mathbf{x}$ is efficiently achieved with log-linear complexity.

Special emphasis for the case $D = 3$ was given recently in [5] where an efficient algorithm for the computation of $\mathbf{x}_{\text{opt}}$ was developed. The algorithm utilizes auxiliary spherical coordinates and partitions the three-dimensional space into a quadratic-size set of regions. Each region corresponds to a distinct binary vector and the set $\mathcal{X}(\mathbf{V}_{N\times 3})$ that contains all binary vectors associated with regions has cardinality $|\mathcal{X}(\mathbf{V}_{N\times 3})| = \binom{N}{2} + 1 = \mathcal{O}(N^2)$, is constructed with complexity $\mathcal{O}(N^2 \log N)$, and contains the optimal vector $\mathbf{x}_{\text{opt}}$.

From a different perspective, several works in the area of computational geometry have treated the equivalent problem of maximization of a rank-D quadratic form $\mathbf{b}^T\mathbf{Q}\mathbf{b}$ over the 0/1 field, i.e. when $\mathbf{Q}$ is a matrix of rank D and $\mathbf{b} \in \{0,1\}^N$. They do so by identifying a subset of $\{0,1\}^N$ that contains $\sum_{d=0}^{D-1} \binom{N-1}{d}$ vectors among which one vector is the maximizer of $\mathbf{b}^T\mathbf{Q}\mathbf{b}$. The subset of interest is constructed by the incremental algorithm or the reverse search. The incremental algorithm

is theoretically faster but very complicated to implement due to its large memory requirement while the reverse search is simpler to implement and constructs the set of $\sum_{d=0}^{D-1} \binom{N-1}{d}$ candidate vectors with complexity $\mathcal{O}(\mathrm{LP}(N,D) \cdot N^D)$ where $\mathrm{LP}(N,D)$ denotes the time to solve a linear programming (LP) optimization problem with N inequalities and D variables .

In the next section, we generalize the approach of [5] to treat the problem of quadratic-form maximization for any $D \leq N-1$. Specifically, we introduce $D-1$ auxiliary spherical coordinates and show that there exists a set $\mathcal{X}(\mathbf{V}_{N\times D}) \subset \{\pm 1\}^N$ which has cardinality $|\mathcal{X}(\mathbf{V}_{N\times D})| = \sum_{d=0}^{D-1} \binom{N-1}{d}$, can be computed in polynomial time, and contains the optimal vector $\mathbf{x}_{\mathrm{opt}}$. We also develop an algorithm that constructs $\mathcal{X}(\mathbf{V}_{N\times D})$ with computational complexity $\mathcal{O}\left(N^D\right)$ and show that it is fully parallelizable and rank-scalable.

Chapter 3

Efficient maximization of a rank deficient quadratic form with a binary vector argument

3.1 Theoretic developments

Since $\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x} = \|\mathbf{V}^T\mathbf{x}\|^2$, we have to optimize the following form

$$\mathbf{x}_{\text{opt}} = \arg \max_{\mathbf{x}\in\{\pm 1\}^N} \|\mathbf{V}^T\mathbf{x}\|. \tag{3.1}$$

We recall that $\mathbf{V}$ is a full-rank $N \times D$ matrix, $D \leq N-1$. We make the assumption that each row of $\mathbf{V}$ has at least one nonzero element $\mathbf{V}_{n,1:D} \neq \mathbf{0}_{1\times D}$. If this condition doesnt exist such that $\mathbf{V}_{n,1:D} = \mathbf{0}_{1\times D}$ then neither $x_n = +1$ nor $x_n = -1$ have an effect on $\mathbf{V}^T\mathbf{x}$, implying that we can ignore the corresponding row of $\mathbf{V}$, assign an arbitrary value to $x_n = \pm 1$, and reduce the size of the original problem from N to $N-1$. In addition, we assume that $V_{n,1} \neq 0$, $n = 1, 2, \ldots, N$, because for any $\mathbf{V} \in \mathbb{R}^{N\times D}$ there exists an orthogonal matrix $\mathbf{U} \in \mathbb{R}^{D\times D}$ such that $\|\mathbf{V}^T\mathbf{x}\| = \|(\mathbf{V}\mathbf{U})^T\mathbf{x}\|$ and the $N \times D$ matrix $\mathbf{V}\mathbf{U}$ contains no zero in its first column, $[\mathbf{V}\mathbf{U}]_{n,1} \neq 0$, $n = 1, 2, \ldots, N$, as the following proposition states.

Proposition 1 *For any $N \times D$ matrix $\mathbf{V}$ there exists an orthogonal $D \times D$ matrix $\mathbf{U}$ such that the matrix $\mathbf{V}\mathbf{U}$ does not contain any zero in its first column.*

To develop an efficient method for the maximization in (3.1), we introduce the spherical coordinates $\phi_1 \in (-\pi, \pi]$, $\phi_2, \ldots, \phi_{D-1} \in (-\frac{\pi}{2}, \frac{\pi}{2}]$ and define the spherical coordinate vector

$$\boldsymbol{\phi}_{i:j} \stackrel{\triangle}{=} [\phi_i, \phi_{i+1}, \ldots, \phi_j]^T \tag{3.2}$$

and the hyperpolar vector

$$\mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \stackrel{\triangle}{=} \begin{bmatrix} \sin\phi_1 \\ \cos\phi_1 \sin\phi_2 \\ \cos\phi_1 \cos\phi_2 \sin\phi_3 \\ \vdots \\ \cos\phi_1 \cos\phi_2 \ldots \sin\phi_{D-1} \\ \cos\phi_1 \cos\phi_2 \ldots \cos\phi_{D-1} \end{bmatrix}. \tag{3.3}$$

A critical equality for our subsequent developments is

$$\max_{\mathbf{x}\in\{\pm 1\}^N} \left\|\mathbf{V}^T\mathbf{x}\right\| = \max_{\mathbf{x}\in\{\pm 1\}^N} \max_{\boldsymbol{\phi}_{1:D-1}\in(-\pi,\pi]\times(-\frac{\pi}{2},\frac{\pi}{2}]^{D-2}} \left\{\mathbf{x}^T\mathbf{V}\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\right\} \tag{3.4}$$

which results from Cauchy-Schwartz Inequality, since for any $\mathbf{a} \in \mathbb{R}^D$

$$\mathbf{a}^T\mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \leq \|\mathbf{a}\| \underbrace{\|\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\|}_{=1} \tag{3.5}$$

with equality if and only if $\phi_1, \ldots, \phi_{D-1}$ are the spherical coordinates of $\mathbf{a}$. We interchange the maximizations in (3.4) to obtain the equivalent problem

$$\max_{\boldsymbol{\phi}_{1:D-1}\in(-\pi,\pi]\times(-\frac{\pi}{2},\frac{\pi}{2}]^{D-2}} \sum_{n=1}^{N} \max_{x_n=\pm 1} \left\{x_n\mathbf{V}_{n,1:D}\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\right\}. \tag{3.6}$$

For a given point $\boldsymbol{\phi}_{1:D-1} \in (-\pi, \pi] \times \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]^{D-2}$, the maximizing argument of each term of the sum in (3.6) depends *only* on the corresponding row of $\mathbf{V}$ and is determined by

$$\mathbf{V}_{n,1:D}\,\mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \underset{x_n=-1}{\overset{x_n=+1}{\gtrless}} 0, \qquad n = 1, \ldots, N. \tag{3.7}$$

Motivated by the decision rule, for each $D \times 1$ vector $\mathbf{v}$ we define the *decision function* x that maps $\boldsymbol{\phi}_{1:D-1}$ to $+1$ or -1 according to

$$\mathbf{V}_{n,1:D}\, \mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \underset{x_n=-1}{\overset{x_n=+1}{\gtrless}} 0, \qquad n = 1, \ldots, N. \tag{3.8}$$

Motivated by the decision rule, for each $D \times 1$ vector $\mathbf{v}$ we define the *decision function* x that maps $\boldsymbol{\phi}_{1:D-1}$ to $+1$ or -1 according to

$$\mathbf{x}(\mathbf{V}_{N\times D}; \boldsymbol{\phi}_{1:D-1}) \triangleq \begin{bmatrix} x(\mathbf{V}_{1,1:D}; \mathbf{c}(\boldsymbol{\phi}_{1:D-1})) \\ x(\mathbf{V}_{2,1:D}; \mathbf{c}(\boldsymbol{\phi}_{1:D-1})) \\ \vdots \\ x(\mathbf{V}_{N,1:D}; \mathbf{c}(\boldsymbol{\phi}_{1:D-1})) \end{bmatrix} = \operatorname{sgn}(\mathbf{V}_{N\times D}\mathbf{c}(\boldsymbol{\phi}_{1:D-1})) \tag{3.9}$$

and the optimal vector $\mathbf{x}_{\text{opt}}$ belongs to $\bigcup_{\boldsymbol{\phi}_{1:D-1}\in(-\pi,\pi]\times(-\frac{\pi}{2},\frac{\pi}{2}]^{D-2}} \mathbf{x}(\mathbf{V}_{N\times D}; \boldsymbol{\phi}_{1:D-1})$.

We note that $x(\mathbf{v}^T; \phi_1 - \pi, \boldsymbol{\phi}_{2:D-1}) = -x(\mathbf{v}^T; \phi_1, \boldsymbol{\phi}_{2:D-1})$ for any $\mathbf{v} \in \mathbb{R}^D$ and $\boldsymbol{\phi}_{1:D-1} \in (-\pi, \pi]\times\left(-\frac{\pi}{2}, \frac{\pi}{2}\right]^{D-2}$, implying $\mathbf{x}(\mathbf{V}_{N\times D}; \phi_1-\pi, \boldsymbol{\phi}_{2:D-1}) = -\mathbf{x}(\mathbf{V}_{N\times D}; \phi_1, \boldsymbol{\phi}_{2:D-1})$ for any real matrix $\mathbf{V}_{N\times D}$ and $\boldsymbol{\phi}_{1:D-1} \in (-\pi, \pi]\times\left(-\frac{\pi}{2}, \frac{\pi}{2}\right]^{D-2}$. Since opposite binary vectors $\mathbf{x}$ and $-\mathbf{x}$ result in the same metric value in, we can ignore the values of ϕ_1 in $\left(-\pi, -\frac{\pi}{2}\right] \cup \left(\frac{\pi}{2}, \pi\right]$ and rewrite the optimization problem in (3.6) as

$$\max_{\boldsymbol{\phi}_{1:D-1}\in\Phi^{D-1}} \sum_{n=1}^{N} \max_{x_n=\pm 1} \left\{x_n \mathbf{V}_{n,1:D}\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\right\}, \quad \Phi \triangleq \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]. \tag{3.10}$$

Finally, we collect all candidate binary vectors into set

$$\begin{aligned} \mathcal{X}(\mathbf{V}_{N\times D}) &\triangleq \bigcup_{\boldsymbol{\phi}_{1:D-1}\in\Phi^{D-1}} \left\{\mathbf{x}(\mathbf{V}_{N\times D}; \boldsymbol{\phi}_{1:D-1})\right\} \\ &= \left\{\bar{\mathbf{x}} \in \{\pm 1\}^N : \exists\ \boldsymbol{\phi}_{1:D-1} \in \Phi^{D-1} \text{ such that } \mathbf{x}(\mathbf{V}_{N\times D}; \boldsymbol{\phi}_{1:D-1}) = \bar{\mathbf{x}}\right\} \\ &\subseteq \{\pm 1\}^N \end{aligned} \tag{3.11}$$

and observe that $\arg\max_{\mathbf{x}\in\{\pm 1\}^N} \left\{\mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}\right\} \in \mathcal{X}(\mathbf{V})$. In the following, we show that $|\mathcal{X}(\mathbf{V}_{N\times D})| = \sum_{d=0}^{D-1} \binom{N-1}{d}$ firstly and secondly develop an algorithm for the construction of $\mathcal{X}(\mathbf{V}_{N\times D})$ with complexity $\mathcal{O}(N^D)$.

We begin by observing that the decision function x determines a hypersurface that partitions the $(D-1)$-dimensional hypercube Φ^{D-1} into two regions; one corresponds to $x(\mathbf{v}^T;\boldsymbol{\phi}_{1:D-1})=+1$ and the other corresponds to $x(\mathbf{v}^T;\boldsymbol{\phi}_{1:D-1})=-1$. The following proposition presents the details of such a partition.

Proposition 2 *Let* $\mathbf{v}\in\mathbb{R}^D$, $v_1\neq 0$, *and* $\boldsymbol{\phi}_{1:D-1}\in\Phi^{D-1}$. *Then, the decision rule* $x(\mathbf{v}^T;\boldsymbol{\phi}_{1:D-1})=\operatorname{sgn}\left(\mathbf{v}^T\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\right)$ *is equivalent to*

$$x(\mathbf{v}^T;\boldsymbol{\phi}_{1:D-1})=\begin{cases}-\operatorname{sgn}(v_1), & \phi_1\in\left(-\frac{\pi}{2},\tan^{-1}\left(-\frac{\mathbf{v}_{2:D}^T\mathbf{c}(\boldsymbol{\phi}_{2:D-1})}{v_1}\right)\right],\\ \operatorname{sgn}(v_1), & \phi_1\in\left(\tan^{-1}\left(-\frac{\mathbf{v}_{2:D}^T\mathbf{c}(\boldsymbol{\phi}_{2:D-1})}{v_1}\right),\frac{\pi}{2}\right].\end{cases}\tag{3.12}$$

□

As seen in Proposition 2, for any $\mathbf{v}\in\mathbb{R}^D$ with $v_1\neq 0$ the function $\phi_1=\tan^{-1}\left(-\frac{\mathbf{v}_{2:D}^T\mathbf{c}(\boldsymbol{\phi}_{2:D-1})}{v_1}\right)$ is equivalent to $\mathbf{v}^T\mathbf{c}(\boldsymbol{\phi}_{1:D-1})=0$ and determines a hypersurface $S(\mathbf{v}^T)$ which partitions Φ^{D-1} into two regions that correspond to the two opposite values $x(\mathbf{v}^T;\boldsymbol{\phi}_{1:D-1})=\pm 1$. As a result, the $N\times D$ matrix $\mathbf{V}_{N\times D}$ is associated with N hypersurfaces $S(\mathbf{V}_{1,1:D}),S(\mathbf{V}_{2,1:D}),\ldots,S(\mathbf{V}_{N,1:D})$ that partition the hypercube Φ^{D-1} into K cells $C_1,C_2,\ldots,C_K$ such that $\bigcup_{k=1}^K C_k=\Phi^{D-1}$, $C_k\cap C_j\neq 0$ if $k\neq j$, and each cell C_k corresponds to a *distinct* $\mathbf{x}_k\in\{\pm 1\}^N$ in the sense that $\mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-1})=\mathbf{x}_k$ for any $\boldsymbol{\phi}_{1:D-1}\in C_k$ and $\mathbf{x}_k\neq\mathbf{x}_j$ if $k\neq j$, $k,j\in\{1,2,\ldots,K\}$.

Proposition 3 *Let* $\mathbf{V}_{N\times D}$ *be a rank-D matrix and* $V_{n,1}\neq 0$, $n=1,2,\ldots,N$. *The following hold true.*

(a) *Each subset of* $\{S(\mathbf{V}_{1,1:D}),S(\mathbf{V}_{2,1:D}),\ldots,S(\mathbf{V}_{N,1:D})\}$ *that consists of* $D-1$ *hypersurfaces has either a single intersection or uncountably many intersections in* Φ^{D-1}.

(b) *For any* $\phi_1,\phi_2,\ldots,\phi_{D-1}\in\Phi$,

(i) $\mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-2},\frac{\pi}{2})=\mathbf{x}(\mathbf{V}_{N\times(D-1)};\boldsymbol{\phi}_{1:D-2})$,

(ii) $\mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-2},-\frac{\pi}{2})=-\mathbf{x}(\mathbf{V}_{N\times D};-\boldsymbol{\phi}_{1:D-2},\frac{\pi}{2})$,

(iii) $\mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-3},\frac{\pi}{2},\phi_{D-1})=\mathbf{x}(\mathbf{V}_{N\times(D-2)};\boldsymbol{\phi}_{1:D-3})$,

(iv) $\mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-3},-\frac{\pi}{2},\phi_{D-1}) = -\mathbf{x}(\mathbf{V}_{N\times D};-\boldsymbol{\phi}_{1:D-3},\frac{\pi}{2},\phi'_{D-1}),\ \forall\phi'_{D-1}\in\Phi$,

and (v) $\mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-3},\pm\frac{\pi}{2},\phi_{D-1}) = \mathbf{x}(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-3},\pm\frac{\pi}{2},\phi'_{D-1}),\ \forall\phi'_{D-1}\in\Phi$. □

Let $\mathcal{I}_{D-1}\stackrel{\triangle}{=}\{i_1,i_2,\ldots,i_{D-1}\}\subset\{1,2,\ldots,N\}$ denote the subset of $D-1$ indices that correspond to hypersurfaces $S(\mathbf{V}_{i_1,1:D})$, $S(\mathbf{V}_{i_2,1:D})$, ..., $S(\mathbf{V}_{i_{D-1},1:D})$ and $\boldsymbol{\phi}(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})\in\Phi^{D-1}$ equal the vector of spherical coordinates of their intersection. If $\boldsymbol{\phi}(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$ is uniquely determined according to Proposition 3, Part (a), then it "leads" a cell, say $C(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$, associated with a distinct binary vector $\mathbf{x}(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$ in the sense that $\mathbf{x}\left(\mathbf{V}_{N\times D};\boldsymbol{\phi}_{1:D-1}\right)=\mathbf{x}(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$ for all $\boldsymbol{\phi}_{1:D-1}\in C(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$ and $\boldsymbol{\phi}(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$ is the single point of $C(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})$ for which ϕ_{D-1} is minimized.

We collect all such vectors into

$$J(\mathbf{V}_{N\times D})\stackrel{\triangle}{=}\bigcup_{\mathcal{I}_{D-1}\subset\{1,\ldots,N\}}\{\mathbf{x}(\mathbf{V}_{N\times D};\mathcal{I}_{D-1})\} \tag{3.13}$$

and observe that $J(\mathbf{V}_{N\times D})\subseteq\{\pm1\}^N$ and $|J(\mathbf{V}_{N\times D})|=\binom{N}{D-1}$.[1] In other words, $J(\mathbf{V}_{N\times D})$ contains $\binom{N}{D-1}$ binary vectors; each vector is associated with a cell in Φ^{D-1} that minimizes its ϕ_{D-1} component at a single point which constitutes the intersection of the corresponding $D-1$ hypersurfaces. We also note that there exist cells that are not associated with such a vertex and contain uncountably many points of the form $(\phi_1,\ldots,\phi_{D-2},-\frac{\pi}{2})$. However, according to Proposition 3, Part (b.*ii*), every such a cell can be ignored since there exists another cell that contains points of the form $(-\phi_1,\ldots,-\phi_{D-2},\frac{\pi}{2})$, is associated with the opposite vector, and is "led" by a vertex-intersection (thus, it belongs to $J(\mathbf{V}_{N\times D})$) unless the initial cell contains a point with $\phi_{D-2}=\pm\frac{\pi}{2}$, as Proposition 3, Part (b.*v*) mentions. For example, for $D=3$, the cells which are identified at the bottom of the plane, that is, for $\phi_2=-\frac{\pi}{2}$. We observe that the vectors that are associated with these cells are opposite to the vectors that are associated with the cells that are identified at the

[1] In general, $|J(\mathbf{V}_{N\times D})|\leq\binom{N}{D-1}$ with equality if and only if the $\binom{N}{D-1}$ intersections of hypersurfaces are distinct. In the sequel, we consider the most demanding case of distinct intersections.

top of the plane. Therefore, the former ones can be ignored. Similarly, for $D = 4$, the binary vectors that are determined for $\phi_3 = -\frac{\pi}{2}$ are opposite to the vectors determined for $\phi_3 = \frac{\pi}{2}$, hence the former ones can again be ignored. In addition, if $\phi_{D-2} = \pm\frac{\pi}{2}$ for a particular cell, then this cell "exists" for any $\phi_{D-1} \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$, implying that we can ignore ϕ_{D-1} (or, say, set it to an arbitrary value ϕ'_{D-1}), set ϕ_{D-2} to $\pm\frac{\pi}{2}$, and consider cells defined on $\Phi^{D-3} \times \left\{\pm\frac{\pi}{2}\right\} \times \left\{\phi'_{D-1}\right\}$.

Finally, due to Proposition 3, Part (b.*iv*), the cells that are defined when $\phi_{D-2} = -\frac{\pi}{2}$ are associated with vectors which are opposite to the vectors that are associated with cells defined when $\phi_{D-2} = \frac{\pi}{2}$. Therefore, we can ignore the case $\phi_{D-2} = -\frac{\pi}{2}$, set ϕ_{D-2} to $\frac{\pi}{2}$, ignore ϕ_{D-1}, and, according to Proposition 3, Part (b.*iii*), identify the cells that are determined by the *reduced-size* matrix $\mathbf{V}_{N\times(D-2)}$ over the hypercube Φ^{D-3}. For $D = 4$ we set $\phi_2 = \phi_3 = \frac{\pi}{2}$ and examine the cells that appear on the leftmost vertical edge of the cube. Hence, $\mathcal{X}(\mathbf{V}_{N\times D}) = J(\mathbf{V}_{N\times D}) \cup \mathcal{X}(\mathbf{V}_{N\times(D-2)})$ and, by induction,

$$\mathcal{X}(\mathbf{V}_{N\times d}) = J(\mathbf{V}_{N\times d}) \cup \mathcal{X}(\mathbf{V}_{N\times(d-2)}), \quad d = 3, 4, \ldots, D, \tag{3.14}$$

which implies that

$$\begin{aligned} \mathcal{X}(\mathbf{V}_{N\times D}) &= J(\mathbf{V}_{N\times D}) \cup J(\mathbf{V}_{N\times(D-2)}) \cup \ldots \cup J(\mathbf{V}_{N\times(D-2\left\lfloor\frac{D-1}{2}\right\rfloor)}) \\ &= \bigcup_{d=0}^{\left\lfloor\frac{D-1}{2}\right\rfloor} J(\mathbf{V}_{N\times(D-2d)}), \end{aligned} \tag{3.15}$$

since $\mathcal{X}(\mathbf{V}_{N\times 1}) = J(\mathbf{V}_{N\times 1})$ with $|\mathcal{X}(\mathbf{V}_{N\times 1})| = |J(\mathbf{V}_{N\times 1})| = 1$ and $\mathcal{X}(\mathbf{V}_{N\times 2}) = J(\mathbf{V}_{N\times 2})$ with $|\mathcal{X}(\mathbf{V}_{N\times 2})| = |J(\mathbf{V}_{N\times 2})| = N$. As a result, the cardinality of $\mathcal{X}(\mathbf{V}_{N\times D})$ is

$$\begin{aligned} |\mathcal{X}(\mathbf{V}_{N\times D})| &= |J(\mathbf{V}_{N\times D})| + |J(\mathbf{V}_{N\times(D-2)})| + \ldots + |J(\mathbf{V}_{N\times(D-2\left\lfloor\frac{D-1}{2}\right\rfloor)})| \\ &= \binom{N}{D-1} + \binom{N}{D-3} + \ldots + \binom{N}{D-1-2\left\lfloor\frac{D-1}{2}\right\rfloor} \\ &= \sum_{d=0}^{\left\lfloor\frac{D-1}{2}\right\rfloor} \binom{N}{D-1-2d} = \sum_{d=0}^{D-1} \binom{N-1}{d}. \end{aligned} \tag{3.16}$$

To summarize the developments in this subsection, we have utilized $D-1$ auxiliary spherical coordinates, partitioned the hypercube Φ^{D-1} into

$$\sum_{d=0}^{D-1} \binom{N-1}{d}$$

cells that are associated with distinct binary vectors which constitute

$$\mathcal{X}(\mathbf{V}_{N\times D}) \subseteq \{\pm 1\}^N,$$

and proved that $\mathbf{x}_{\text{opt}} \in \mathcal{X}(\mathbf{V}_{N\times D})$. Therefore, the initial problem in (3.1) has been converted into numerical maximization of $\|\mathbf{V}^T\mathbf{x}\|$ among all vectors $\mathbf{x} \in \mathcal{X}(\mathbf{V}_{N\times D})$. Such an optimization costs $\mathcal{O}\left(\sum_{d=0}^{D-1} \binom{N-1}{d}\right) = \mathcal{O}(N^{D-1})$ comparisons upon construction of $\mathcal{X}(\mathbf{V}_{N\times D})$. An efficient algorithm for the construction of $\mathcal{X}(\mathbf{V}_{N\times D})$ is developed in the next section.

3.2 Algorithmic developments

Let $\mathbf{V}_{N\times D}$ be a real matrix that satisfies the assumptions made in the beginning of Section III. According to (3.15), the construction of $\mathcal{X}(\mathbf{V}_{N\times D})$ reduces to the parallel construction of $J(\mathbf{V}_{N\times D}), J(\mathbf{V}_{N\times(D-2)}), \ldots, J(\mathbf{V}_{N\times 2})$ if D is even and $J(\mathbf{V}_{N\times D}), J(\mathbf{V}_{N\times(D-2)}), \ldots, J(\mathbf{V}_{N\times 1})$ if D is odd. We describe a way to construct $J(\mathbf{V}_{N\times d})$ for any d. Interestingly, from (3.13), we observe that the construction of $J(\mathbf{V}_{N\times d})$ can also be fully parallelized since the candidate vector $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ can be computed *independently* for each $\mathcal{I}_{d-1} \subset \{1, 2, \ldots, N\}$. As a result, we only need to present a method for the computation of $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)\ \forall\ \mathcal{I}_{d-1} \subset \{1, 2, \ldots, N\}$, $d \in \{3, 4, \ldots, D\}$.

We consider a certain value of $d \in \{3, 4, \ldots, D\}$ and a certain set of indices $\mathcal{I}_{d-1} \subset \{1, 2, \ldots, N\}$ such that the $d-1$ hypersurfaces

$$S(\mathbf{V}_{i_1,1:d}), S(\mathbf{V}_{i_2,1:d}), \ldots, S(\mathbf{V}_{i_{d-1},1:d})$$

intersect at a single point $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$. Cell $C\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ that is "led" by $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ is associated with the binary vector $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$.

To identify $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$, we consider its N elements separately and observe the following.

(i) For any index $n \in \{1, 2, \ldots, N\} - \mathcal{I}_{d-1}$, the corresponding element of $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ maintains its value at $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$, hence it is determined by

$$x_n\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right) = x\left(\mathbf{V}_{n,1:d}; \boldsymbol{\phi}(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1})\right). \tag{3.17}$$

(ii) For any index $n \in \mathcal{I}_{d-1}$, say $n = i_k$, the corresponding element of $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ cannot be determined at $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$. However, it maintains its value at the intersection of the remaining $d-2$ hypersurfaces $S(\mathbf{V}_{i_1,1:d-1})$, $S(\mathbf{V}_{i_2,1:d-1})$, $\ldots$, $S(\mathbf{V}_{i_{k-1},1:d-1})$, $S(\mathbf{V}_{i_{k+1},1:d-1})$, $S(\mathbf{V}_{i_{k+2},1:d-1})$, $\ldots$, $S(\mathbf{V}_{i_{d-1},1:d-1})$, hence it is determined by

$$x_n\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right) = x\left(\mathbf{V}_{n,1:d-1}; \boldsymbol{\phi}\left(\mathbf{V}_{N\times(d-1)}; \mathcal{I}_{d-1} - \{i_k\}\right)\right). \tag{3.18}$$

suggest the following construction of $\mathbf{x}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$. If the $\binom{N}{d-1}$ intersections of hypersurfaces are distinct, then only the $d-1$ hypersurfaces $S(\mathbf{V}_{i_1,1:d}), S(\mathbf{V}_{i_2,1:d}), \ldots, S(\mathbf{V}_{i_{d-1},1:d})$ pass through the "leading" vertex $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ of cell $C\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$. Therefore, if $n \in \{1, 2, \ldots, N\} - \mathcal{I}_{d-1}$, then the corresponding hypersurface $S(\mathbf{V}_{n,1:d})$ does not pass through $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$, implying that the polarity of $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ with respect to $S(\mathbf{V}_{n,1:d})$ is the same as the polarity of any point of the cell of interest $C\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ with respect to the same hypersurface. As a result, the sign of the corresponding binary element $x_n\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ is well-determined at the "leading" vertex, as (3.17) states. It remains to describe how the vector of spherical coordinates $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ is computed efficiently. Recall that $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ represents the intersection of $S(\mathbf{V}_{i_1,1:d}), S(\mathbf{V}_{i_2,1:d}), \ldots, S(\mathbf{V}_{i_{d-1},1:d})$, i.e. the solution of

$$\mathbf{V}_{\mathcal{I}_{d-1},1:d}\, \mathbf{c}(\boldsymbol{\phi}_{1:d-1}) = \mathbf{0}_{(d-1)\times 1}. \tag{3.19}$$

According to the proof of Proposition 3, Part (a), for a full-rank $(d-1)\times d$ real matrix eq. (3.19) has a unique solution $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right) \in \Phi^{d-1}$ which consists of the spherical coordinates of the zero right singular vector of $\mathbf{V}_{\mathcal{I}_{d-1},1:d}$. Therefore, to obtain $\boldsymbol{\phi}\left(\mathbf{V}_{N\times d}; \mathcal{I}_{d-1}\right)$ we just need to compute the zero right singular vector of $\mathbf{V}_{\mathcal{I}_{d-1},1:d}$ and calculate its spherical coordinates.

Chapter 4

Implementation of singular value decomposition

In this part, I describe the usage of singular value decomposition and the algorithm implementation that I have used in my program development.

4.1 Theoretic development

4.1.1 Introduction

In linear algebra, the **singular value decomposition (SVD)** is an important factorization of a rectangular real or complex matrix, with many applications in signal processing and statistics. Applications which employ the SVD include computing the pseudoinverse, least squares fitting of data, matrix approximation, and determining the rank, range and null space of a matrix.

Suppose $\mathbf{A}$ is an m-by-n matrix whose entries come from the field K, which is either the field of real numbers or the field of complex numbers. Then there exists a factorization of the form where $\mathbf{U}$ is an m-by-m unitary matrix over K, the matrix $\mathbf{S}$ is m-by-n diagonal matrix with nonnegative real numbers on the diagonal, and $\mathbf{V}^*$ denotes the conjugate transpose of $\mathbf{V}$, an n-by-n unitary matrix over K. Such a factorization is called a *singular-value decomposition* of $\boldsymbol{A}$.

A common convention is to order the diagonal entries $\mathbf{S}_{i,j}$ in non-increasing

fashion. In this case, the diagonal matrix **S** is uniquely determined by **A** (though the matrices $\boldsymbol{U}$ and **V** are not). The diagonal entries of **S** are known as the singular values of **A**.

4.1.2 Intuitive explanation

In

$$A = USV^* \tag{4.1}$$

- the columns of **V** form a set of orthonormal "input" or "analyzing" basis vector directions for **A**
- the columns of **U** form a set of orthonormal "output" basis vector directions for **A**
- The matrix **S** contains the singular values, which can be thought of as scalar "gain controls" by which each corresponding input is multiplied to give a corresponding output.

4.1.3 Problem Statement

In order to carry out computations with matrices, it is common to decompose them in some way to simplify and speed up the calculations. For a real m by n matrix **A**, the **QR** decomposition is particularly useful. This equates the matrix **A** with the product of an orthogonal matrix **Q** and a right- or upper-triangular matrix **R**, that is

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \tag{4.2}$$

Where **Q** is m by m and

$$\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{Q} = \boldsymbol{Q}\boldsymbol{Q}^{\boldsymbol{T}} = \mathbf{I} \tag{4.3}$$

and **R** is m by n with all elements

$$\boldsymbol{R_{i,j}} = \mathbf{0}, for\ i > j \tag{4.4}$$

The **QR** decomposition leads to the *singular-value decomposition* of the matrix **A** if the matrix **R** is identified with the product of a diagonal matrix **S** and an orthogonal matrix $\boldsymbol{V^T}$, that is

$$\boldsymbol{R} = \boldsymbol{S}\,\boldsymbol{V}^{\boldsymbol{T}} \tag{4.5}$$

Where the m by n matrix **S** is such that

$$S_{i,j} = 0, for\ i \neq j \tag{4.6}$$

And **V**, n by n, is such that

$$\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{V} = \boldsymbol{V}\,\boldsymbol{V}^{\boldsymbol{T}} = \mathbf{I} \tag{4.7}$$

Note that the zeros below the diagonal in both **R** and **S** imply that, apart from orthogonality conditions imposed by 4.3, the elements of columns $(n + 1)$, $(n + 2)$, . . . , m of **Q** are arbitrary. In fact, they are not needed in most calculations, so will be dropped, leaving the m by n matrix **U**, where

$$\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{U} = \mathbf{I_n} \tag{4.8}$$

Note that it is no longer possible to make any statement regarding **UUT**. Omitting rows $(n + 1)$ to m of both **R** and **S** allows the decompositions to be written as

$$\boldsymbol{A} = \boldsymbol{U}\,\boldsymbol{R} = \boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}^{\boldsymbol{T}} \tag{4.9}$$

Where **A** is m by n, **U** is m by n and, **R** is n by n and upper-triangular, **S** is n by n and diagonal, and **V** is n by n and orthogonal.

4.2 A SINGULAR-VALUE DECOMPOSITION ALGORITHM

While the svd is somewhat of a sledgehammer method for many nutshell problems, its versatility in finding the eigensolutions of a real symmetric matrix, in solving sets of simultaneous linear equations or in computing minimum-length solutions to least-squares problems makes it a valuable building block in programs used to tackle a variety of real problems. This versatility has been exploited in a single small program to carry out the above problems as well as to find inverses and generalised inverses of matrices and to solve nonlinear least-squares problems. However, for computational purposes, an alternative viewpoint is more useful. This considers the possibility of finding an orthogonal matrix $\mathbf{V}$, n by n, which transforms the real m by n matrix $\mathbf{A}$ into another real m by n $\mathbf{B}$ whose columns are orthogonal. The analysis is according to [7]. It is desired to find $\mathbf{V}$ such that

$$\mathbf{B} = \mathbf{AV} = \boldsymbol{(b_1, b_2, \ldots, b_n)} \tag{4.10}$$

Where

$$\boldsymbol{b_i^T b_j = S_i^2 \delta_{ij}} \tag{4.11}$$

And

$$\boldsymbol{V V^T = V^T V = \mathbf{I_n}} \tag{4.12}$$

The kronecker delta takes values

$$\delta_{i,j} = \begin{cases} 0, & for \quad i \neq j \\ 1, & for \quad i = j \end{cases} \tag{4.13}$$

The quantities Si may, as yet, be either positive or negative, since only their square is defined by equation 4.11. They will henceforth be taken arbitrarily as

positive and will be called *singular values* of the matrix **A**. The vectors

$$\boldsymbol{u_j} = \boldsymbol{b_j}/\boldsymbol{S_j} \tag{4.14}$$

which can be computed when none of the $\boldsymbol{Sj}$ is zero, are unit orthogonal vectors. Collecting these vectors into a real m by n matrix, and the singular values into a diagonal n by n matrix, it is possible to write

$$\mathbf{B} = \mathbf{US} \tag{4.15}$$

and

$$\boldsymbol{U^T U} = \mathbf{I_n} \tag{4.16}$$

Where **U** is a unit matrix of order n.

In the case that some of the Sj are zero, equations 4.10 and 4.11 are still valid, but the columns of **U** corresponding to zero singular values must now be constructed such that they are orthogonal to the columns of **U** computed via equation 4.14 and to each other. Thus equations 4.15 and 4.16 are also satisfied. An alternative approach is to set the columns of **U** corresponding to zero singular values to null vectors. By choosing the first k of the singular values to be the non-zero ones, which is always possible by simple permutations within the matrix **V**, this causes the matrix $\boldsymbol{U^T U}$ to be a unit matrix of order k augmented to order n with zeros. This will be written

$$\delta_{i,j} = \begin{pmatrix} 1_k, & \\ & 0_{n-k} \end{pmatrix} \tag{4.17}$$

While not part of the commonly used definition of the svd, it is useful to require the singular values to be sorted, so that

$$S_{11} > S_{22} > S_{33} > \cdots > S_{kk} > \cdots > S_{nn} \tag{4.18}$$

This allows (4.1.8) to be recast as a summation

$$\boldsymbol{A} = \sum_{j=1}^{n} \boldsymbol{u}_j S_{jj} \boldsymbol{v}_j^T \tag{4.19}$$

Partial sums of this series give a sequence of approximations $\widetilde{\mathbf{A}}\mathbf{1}, \widetilde{\mathbf{A}}\mathbf{2}, \ldots, \widetilde{\mathbf{A}}n$.Where, obviously, the last member of the sequence $\widetilde{\mathbf{A}}n = \mathbf{A}$. Since it corresponds to a complete reconstruction of the singular value decomposition. The rank-one matrices $\boldsymbol{u}_j S_{jj} \boldsymbol{v}_j^T$ can be referred to as singular planes, and the partial sums (in order of decreasing singular values) are partial svds (Nash and Shlien 1987).A combination of 4.10 and 4.15 gives

$$\boldsymbol{A}\,\boldsymbol{V} = \boldsymbol{U}\boldsymbol{S} \tag{4.20}$$

or, using 4.12, the orthogonality of $\mathbf{V}$, $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ 4.9 which expresses the svd of $\mathbf{A}$. The preceding discussion is conditional on the existence and computability of a suitable matrix $\mathbf{V}$. The next section shows how this task may be accomplished.

4.3 Orthogonalisation by plane rotations

The matrix $\mathbf{V}$ sought to accomplish the orthogonalisation 4.10 will be built up as a product of simpler matrices

$$\boldsymbol{V} = \prod_{k=1}^{z} \boldsymbol{V}^{(k)} \tag{4.21}$$

where z is some index not necessarily related to the dimensions m and n of $\mathbf{A}$, the matrix being decomposed. The matrices used in this product will be plane rotations. If $\boldsymbol{V}^{(k)}$ is a rotation of angle φ in the ij plane, then all elements of $\boldsymbol{V}^{(k)}$ will be the same as those in a unit matrix of order n except for

$$\begin{aligned} \mathbf{V}_{ii}^{(k)} &= \cos\varphi = \mathbf{V}_{jj}^{(k)} \\ \mathbf{V}_{ij}^{(k)} &= \sin\varphi = \mathbf{V}_{ji}^{(k)}. \end{aligned} \tag{4.22}$$

Thus $\boldsymbol{V}^{(k)}$ affects only two columns of any matrix it multiplies from the right. These columns will be labeled x and y. Consider the effect of a single rotation

involving these two columns

$$(\mathbf{x}, \mathbf{y}) \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} = (\mathbf{X}, \mathbf{Y}) \tag{4.23}$$

Thus we have

$$\begin{aligned} \mathbf{X} &= \mathbf{x}\cos\varphi + \mathbf{y}\sin\varphi \\ \mathbf{Y} &= -\mathbf{x}\sin\varphi + \mathbf{y}\cos\varphi \end{aligned} \tag{4.24}$$

If the resulting vectors X and Y are to be orthogonal, then

$$\mathbf{X}^T\mathbf{Y} = 0 = -(\mathbf{x}^T\mathbf{x} - \mathbf{y}^T\mathbf{y})\sin\varphi\cos\varphi + \mathbf{x}^T\mathbf{y}(\cos^2\varphi - \sin^2\varphi) \tag{4.25}$$

There is a variety of choices for the angle *f*, or more correctly for the sine and cosine of this angle, which satisfy 4.25. Some of these are mentioned by Hestenes (1958), Chartres (1962) and Nash (1975). However, it is convenient if the rotation can order the columns of the orthogonalised matrix **B** by length, so that the singular values are in decreasing order of size and those which are zero (or infinitesimal) are found in the lower right-hand corner of the matrix S as in equation 4.9. Therefore, a further condition on the rotation is that

$$\mathbf{X}^T\mathbf{X} - \mathbf{x}^T\mathbf{x} > 0 \tag{4.26}$$

For convenience, the columns of the product matrix

$$\boldsymbol{A}\prod_{k=1}^{z}\boldsymbol{V}^{(k)} \tag{4.27}$$

will be donated $a_i i = 1, 2, \ldots, n$. The progress of the orthogonalisation is then observable if a measure Z of the non-orthogonality is defined

$$\mathbf{Z} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(a_i^T a_j)^2 \tag{4.28}$$

Since two columns orthogonalised in one rotation may be made non- orthogonal in subsequent rotations, it is essential that this measure be reduced at each rotation.

Because only two columns are involved in the kth rotation, we have

$$\mathbf{Z}^{(k)} = \mathbf{Z}^{(k-1)} + (\mathbf{X}^T\mathbf{Y})^2 - (\mathbf{x}^T\mathbf{y})^2 \tag{4.29}$$

But condition 4.25 implies

$$\mathbf{Z}^(k) = \mathbf{Z}^(k-1) - (\mathbf{x}^T\mathbf{y})^2 \tag{4.30}$$

so that the non-orthogonality is reduced at each rotation.

The specific formulae for the sine and cosine of the angle of rotation are (see e.g. Nash 1975) given in terms of the quantities

$$p = \mathbf{x}^T\mathbf{y} \tag{4.31}$$

$$q = \mathbf{x}^T\mathbf{x} - \mathbf{y}^T\mathbf{y} \tag{4.32}$$

And

$$v = (4p^2 + q^2)^(1/2) \tag{4.33}$$

They are

$$\cos\phi = \left[\frac{(v+q)}{2v}\right]^{1/2}, for \quad q \geq 0 \tag{4.34}$$

$$\sin\phi = p/v\cos\phi, for \quad q \geq 0 \tag{4.35}$$

$$\sin\phi = sgn(p)[(v-q)/2v]^{1/2}, for \quad q < 0 \tag{4.36}$$

$$\cos\phi = p/v\sin\phi, for \quad q < 0 \tag{4.37}$$

$$sgn(p) = \begin{cases} 1, & p \geq 0 \\ -1, & p < 0 \end{cases} \tag{4.38}$$

Note that having two forms for the calculation of the functions of the angle of rotation permits the subtraction of nearly equal numbers to be avoided. As the matrix nears orthogonality p will become small, so that q and v are bound to have nearly equal magnitudes. It is choosen to be performed the computed rotation only when q r, and to use

$$\sin\phi = 1 \cos\phi = 0, when \qquad q < 0 \tag{4.39}$$

This affects an interchange of the columns of the current matrix **A**. However, it is more efficient to perform the rotations as defined in the code presented. The rotations 4.39 were used to force nearly null columns of the final working matrix to the right-hand side of the storage array. This will occur when the original matrix **A** suffers from linear dependencies between the columns (that is, is rank deficient). In such cases, the rightmost columns of the working matrix eventually reflect the lack of information in the data in directions corresponding to the null space of the matrix **A**. The current methods cannot do much about this lack of information, and it is not sensible to continue computations on these columns. In the current implementation of the method (Nash and Shlien 1987), we prefer to ignore columns at the right of the working matrix which become smaller than a specified tolerance. This has a side effect of speeding the calculations significantly when rank deficient matrices are encountered.

4.4 A fine point

Equations 4.30 and 4.26 cause the algorithm just described obviously to proceed *towards* both an orthogonalisation and an ordering of the columns of the resulting matrix $\boldsymbol{A}^{(z)}$. However the rotations must be arranged in some sequence to carry this task to completion. Furthermore, it remains to be shown that some sequences of rotations will not place the columns in disorder again. For suppose a_1 is orthogonal

to all other columns and larger than any of them individually. A sequential arrangement of the rotations to operate first on columns $(1,2)$, then $(1,3),(1,4)\ldots(1,n)$, followed by $(2,3)\ldots(2,n)(3,4)\ldots((n-1),n)$ will be called a cycle or sweep. Such a sweep applied to the matrix described can easily yield a new a_2 for which

$$\mathbf{a}_2^T\mathbf{a}_2 > \mathbf{a}_1^T\mathbf{a}_1 \tag{4.40}$$

If, for instance, the original matrix has $a_2 = a_3$ and the norm of these vectors is greater than $2^{-\frac{1}{2}}$ times the norm of a_1. Another sweep of rotations will put things right in this case by exchanging a_1 and a_2. However, once two columns have achieved a separation related in a certain way to the non-orthogonality measure 4.28, it can be shown that no subsequent rotation can exchange them. Suppose that the algorithm has proceeded so far that the non-orthogonality measure Z satisfies the inequality

$$Z < t^2 \tag{4.41}$$

where t is some positive tolerance. Then, for any subsequent rotation the parameter p, equation 4.32, must obey

$$p^2 < t^2 \tag{4.42}$$

Suppose that all adjacent columns are separated in size so that

$$\mathbf{a}_{k-1}^T\mathbf{a}_{k-1} - \mathbf{a}_k^T\mathbf{a}_k > t \tag{4.43}$$

Then a rotation which changes a_k (but not a_{k1}) cannot change the ordering of the two columns. If $x = a_{k-1}$, then straightforward use of equations 4.34 and 4.35 or 4.36 and 4.37 gives

$$\mathbf{X}^T\mathbf{X} - \mathbf{x}^T\mathbf{x} = (v-q)/2 \geq 0. \tag{4.44}$$

Using 4.42 and 4.33 in 4.44 gives

$$\mathbf{X}^T\mathbf{X} - \mathbf{x}^T\mathbf{x} \leq (v-q)/2 \geq [(4t^2+q^2)^(1/2) - q]/2 \leq [(2t+q)-q]/2 = t. \quad (4.45)$$

Thus, once columns become sufficiently separated by size and the onorthogonality sufficiently diminished, the column ordering is stable. When some columns are equal in norm but orthogonal, the above theorem can be applied to columns separated by size.

We present the algorithm that was implemented in C

procedure NashSVD(nRow, nCo1: integer; {size of problem} var W: wmatrix; {working matrix} var Z: rvector); {squares of singular values} {The algorithm forms a singular value decomposition of matrix A which is stored in the first nRow rows of working array W and the nCo1 columns of this array. The first nRow rows of W will become the product U * S of a conventional svd, where S is the diagonal matrix of singular values. The last nCo1 rows of W will be the matrix V of a conventional svd. On return, Z will contain the squares of the singular values. An extended form of this commentary can be displayed on the screen by removing the comment braces on the writeln statements below. }

```
Var i, j, k, EstColRank, RotCount, SweepCount,
slimit : integer; eps, e2, tol, vt, p, h2, x0, y0, q, r, c0, s0, c2, d1, d2 : real;
procedure rotate; (STEP 10 as a procedure)
(This rotation acts on both U and V, by storing V at the bottom of U)
begin (<< rotation )
for i :=1 to nRow+nCol do
begin
D1 := W[i,j]; D2:= W[i,k];
end; { rotation >>}
W[i,j] := D1*c0+D2*s0; W[i,k] := -D1*s0+D2*c0
end; { rotate }
begin { procedure SVD }
{STEP 0 Enter nRow, nCo1, the dimensions of the matrix to be decomposed.}
eps:= Calceps; {Set eps, the machine precision.}
slimit := nCo1 div 4; if slimit¡ then slimit := 6;
{Set slimit, a limit on the number of sweeps allowed. A suggested limit is
max([nCol/4], 6).}
SweepCount := 0; {to count the number of sweeps carried out}
e2 := 10.0*nRow*eps*eps;
tol:= eps*0.1;
{Set the tolerances used to decide if the algorithm has converged.
For further discussion of this, see the commentary under STEP 7.} EstColRank
:= nCo1; {current estimate of rank};
{Set V matrix to the unit matrix of order nCo1. V is stored in rows (nRow+1) to
(nRow+nCol) of array W.}
for i := 1 to nCo1 do
begin
for j := 1 to nCo1 do
W[nRow+i,j]:= 0.0; W[nRow+i,i]:= 1.0;
end; {loop on i, and initialization of V matrix}
{Main SVD calculations}
repeat {until convergence is achieved or too many sweeps are carried out}
RotCount := EstColRank*(EstColRank-1) div 2; {STEP 1 – rotation counter}
SweepCount := SweepCount+1
for j := 1 to EstColRank-1 do {STEP 2 – main cyclic Jacobi sweep}
```

```
begin {STEP 3}
for k := j+l to EstColRank do
begin {STEP 4}
p :=0.0; q:= 0.0; r := 0.0;
for i :=1 to nRow do {STEP 5}
begin
x0 := W[i,j]; y0 := W[i,k];
p := p+x0*y0; q:= q+x0*x0; r := r+y0*y0;
end;
Z[j] := q; Z[k]:=r;
{Now come important convergence test considerations. First we will decide if rotation will exchange order of columns.}
if q >= r then {STEP 6 – check if the columns are ordered.}
begin {STEP 7 Columns are ordered, so try convergence test.}
if (q<=e2*2[1]) or (abs(p)<= tol*q) then RotCount := RotCount-1
{There is no more work on this particular pair of columns in the current sweep. That is, we now go to STEP 11. The first condition checks for very small column norms in BOTH columns, for which no rotation makes sense. The second condition determines if the inner product is small with respect to the larger of the columns, which implies a very small rotation angle.}
else {columns are in order, but their inner product is not small}
begin {STEP 8}
p := p/q; r:= 1-r/q; vt:= sqrt(4*p*p + r*r);
c0 := sqrt(0.5*(1+r/vt)); s0 := p/(vt*c0);
rotate;
end end {columns in order with q>=r}
else {columns out of order – must rotate}
begin {STEP 9}
{note: r > q, and cannot be zero since both are sums of squares for
the svd. In the case of a real symmetric matrix, this assumption
must be questioned.}
p := p/r; q := q/r-1; vt :=sqrt(4*p*p + q*q);
s0 :=sqrt(0.5*(1-q/vt));
if p¡0 then s0 := -s0;
```

```
co :=p/(vt*s0);
rotate; {The rotation is STEP 10.}
end; {Both angle calculations have been set up so that large numbers do not occur in intermediate quantities. This is easy in the svd case, since quantities x2,y2 cannot be negative. An obvious scaling for the eigenvalue problem does not immediately suggest itself.} end; {loop on K – end-loop is STEP 11}
end; {loop on j – end-loop is STEP 12}
{STEP 13 – Set EstColRank to largest column index for which
Z[column index] > (Z[1]*tol + tol*tol)
Note how Pascal expresses this more precisely.}
while (EstColRank >=3) and (Z[EstColRank] <= Z[1]*tol + tol*tol)
do EstColRank:= EstColRank-1;
{STEP 14 – Goto STEP 1 to repeat sweep if rotations have been
performed and the sweep limit has not been reached.}
until (RotCount=0) or (SweepCount>slimit);
{STEP 15 – end SVD calculations}
if (SweepCount > slimit) then writeln(**** SWEEP LIMIT EXCEEDED);
if (SweepCount > slimit) then
{Note: the decomposition may still be useful, even if the sweep limit has been reached.}
end;
```

Chapter 5

Implementation in C programming language

5.1 Introduction

In this section, I am going to present the implementation of the algorithm in C programming language. At first place, I have to compute the combinations $\binom{N}{D-1}$, where N is the number of rows of the matrix and $D-1$ the cols of the matrix. As we have described above, I have to compute all possible combinations which correspond to $\binom{N}{D-1}$. To this way I have created a routine which takes as inputs the number of rows and the number of cols and gives us the file which contains all the possible combinations. The file is saved with spectacular onomatology which is donated by N and D. So I have computed all the combinations for a variety of N and D. After that, I load this file into my main program. The basic function that is used is **int **compute(double **V, int D, int N)** . As we observe, this function takes as inputs the matrix **V** of quadratic form, the dimensions, number of rows and columns of this matrix and returns the candidate set. We have to mention that we dont have fulfilled the processes in order to obtain all the candidate set, so according to the theory I reduce the dimensions by two scalars and call again the function compute until the$D$ parameter is equal to 1 or 2, which is translated that I take into consideration only the first one or two columns of matrix **V**correspondingly. I mention that each time that function compute is called I put in apposition all

vectors that are returned so we collect all vectors $J(\mathbf{V}_{N\times D})$, which constitutes the candidate set.

5.2 Functions that are used

In this section I am going to describe the components that were implemented and used.

- Function **combinations**

  void combinations(int n, int k, int *array, int finished)

  In first place, I have to compute all possible subsets. This function takes as inputs the number of rows and number of columns of matrix V of quadratic form and returns in an file all possible combinations recursively. The file has a specific onomatology, as we mentioned before , according the D and N. Based on the algorithm if the size of problem is D we have to estimate $\binom{N}{D-1}$ combinations.

- Function **find intersection**

  void find intersection(double *result, double **Ainitial, long rowsn, long cols, long D);

  Calculates the Cartesian coordinates of the intersection of the hyper faces that correspond to the rows of its input matrix within a sign ambiguity by the singular value decomposition, according to the proof of proposition 3, part(a). Then the conversion into spherical coordinates is only necessary to resolve the sign ambiguity and is performed by function **determine sign**.

  Actually this function estimates the singular value decomposition based on the algorithm development which was described on chapter 4. We introduce an auxiliary matrix A which must be pre-allocated with $2N$ rows and N columns. On calling the matrix to be decomposed is contained in the first N rows of **A**. On return the N first rows of **A** contain the product **US** and the lower$N$ rows contain **V**(not $\mathbf{V}'$). The $S2$ vector returns the square of the singular values. And the right singular vector is returned to the main program.

Note; We have to mention that the input matrix has $D-1$ rows and D columns, whose rows are determined by the file which contains all possible combinations $\binom{N}{D-1}$, as we described above.

- Function **determine sign**

  int determine sign(double *c, int cols)

  This function takes as inputs the right singular vector that is estimated in **find intersection** and the dimension D. In fact contains the decision rule that was described in theoretic development and returns the sign of this specific singular vector to the main program in order to be resolved the sign ambiguity.

- Function **total cols**

  int total cols(int cols, int rows)

  This function calculates the total columns of candidate set. As it has been shown in theoretic development the cardinality of candidate set is ;

$$\begin{aligned} |\mathcal{X}(\mathbf{V}_{N\times D})| &= |J(\mathbf{V}_{N\times D})| + |J(\mathbf{V}_{N\times (D-2)})| + \ldots + |J(\mathbf{V}_{N\times (D-2\lfloor \frac{D-1}{2} \rfloor)})| \\ &= \binom{N}{D-1} + \binom{N}{D-3} + \ldots + \binom{N}{D-1-2\lfloor \frac{D-1}{2} \rfloor} \\ &= \sum_{d=0}^{\lfloor \frac{D-1}{2} \rfloor} \binom{N}{D-1-2d} = \sum_{d=0}^{D-1} \binom{N-1}{d}. \end{aligned} \tag{5.1}$$

- Function **find subarrays**

  void find subarrays(double **Ainitial,double **V init sub, long cols, long n, long d ptr)

  This function estimates all possible subspaces as it has described in step two in algorithm development. It takes as input the initial matrix and returns the sub matrix except the row which concerns the variable **d ptr**.

- Function **compute**

  int **compute(double **V init,int cols, int rows)

This function takes as input the matrix $\mathbf{V}$ of quadratic form, the number of rows and columns and returns the candidate set which concern the specific dimension that is called each time. Firstly the function checks the dimension of the matrix (actually variable cols is checked). If the dimension is equal to one then the decision rule for the candidate vector takes into consideration only the first column of $\mathbf{V}$ matrix. If the dimension is equal to two then the decision rule for the candidate vector is based on the following relations;

$$\begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} \\ \vdots & \vdots \\ \mathbf{V}_{N1} & \mathbf{V}_{N2} \end{bmatrix} \tag{5.2}$$

$$\theta_i = -tan^{-1}\left(\frac{\mathbf{V}_{i2}}{\mathbf{V}_{i1}}\right) \tag{5.3}$$

$$c(\varphi) = \begin{bmatrix} cos\varphi \\ sin\varphi \end{bmatrix} \tag{5.4}$$

Where $\varphi_i = \frac{\theta_i + \theta_{i-1}}{2}$, $2 \leq i \leq N$ and $\varphi_1 = \frac{\frac{-\pi}{2} + \theta_1}{2}$

So the decision rule for the candidate vector $\mathbf{X} = sgn(\mathbf{V}\mathbf{c}(\boldsymbol{\phi}))$.

Otherwise, we call the function find intersection, and the input matrix is the first subset that is computed by the function combination. For example if the dimension is three $D = 3$ then the first input of function find intersection is the first line of the file that has been created by combination function which represents the first two rows of $\mathbf{V}$ matrix of quadratic form. After that this function returns the right singular vector which is placed as argument in determine sign function. After that, the output of determine sign function is multiplied with the output of function find intersection in order to solve the sign ambiguity and finally this result is multiplied with the initial matrix $\mathbf{V}$ of quadratic form. This quantity constitutes the criterion, which was formulated

in problem statement and was defined as decision function

$$x(\mathbf{v}^T; \boldsymbol{\phi}_{1:D-1}) \stackrel{\triangle}{=} \arg\max_{x=\pm 1} \left\{ x\mathbf{v}^T \mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \right\} = \mathrm{sgn}(\mathbf{v}^T \mathbf{c}(\boldsymbol{\phi}_{1:D-1})). \tag{5.5}$$

According to the instance above, we have to mention that, we cannot estimate the values of candidate vector in the first and second positions of candidate vector, which concerns the first two rows of matrix $\mathbf{V}$ of quadratic form. This happens because in these positions we cannot decide about $x(1,1)$ and $x(2,1)$ as the value of $\mathbf{x}$ is on curve that is created by $\mathbf{V}(1, 1:N)$ and $\mathbf{V}(2, 1:N)$ and produce the section $\boldsymbol{\phi}(\mathbf{V}_{N\times 3}; \{1,2\})$. In order to compute the values of $\mathbf{x}$ candidate vector in these positions, I pass the projection on axis label ϕ_2 and as a consequence $x(1,1)$ is computed. So I have to implement the proportional procedure to compute $x(2,1)$, that is I pass the projection on axis label ϕ_1. The programming development that was used to implement the procedure below was; firstly I call the function find subarrays, which returns a matrix with dimensions $D-2 \times D-1$. After that the routine find intersection was called with the first sub-array as input and then function determine sign was called and used as input the zero right singular vector from the previous function. Finally the multiplication between the fixed zero right singular vector, the sub-array and the $D-1$ rows of $\mathbf{V}$ matrix that were used on the first time that function find intersection was called produces the right sign for $\mathbf{x}$ candidate vector in these positions above. So we have fulfilled the process in order to compute the first vector $\mathbf{x}$. The function compute terminates when the same process repeated $\binom{N}{D-1}$ times with an exception that D is equal to 1 or 2 as it was mentioned before. So this function returns the candidate set for a specific dimension D.

Main procedure; I commit memory in order to save the candidate set. The number of columns of candidate set is computed by function total cols. The function computes takes as input the $\mathbf{V}$ matrix and the dimension D. Then the first candidate set which concerns dimension D is returned. Until now we have estimated the vectors which were extracted by cells that "lead" an

intersection. Cells that were created and surrounded by axis label in order to compute the intersection I reduce the dimensions by two scalars. Function compute is called again and the candidate set which is extracted concerns dimension $D-2$. I put in apposition the candidate matrix that is produced each time and reduce the dimensions by two scalars until the D parameter is equal to 1 or 2. So the algorithm is terminated and the whole candidate set is constructed with the cardinality that was proven above.

5.3 Comparisons C versus Matlab

It is mentioned the execution time of the algorithm implemented in C versus Matlab

- N=20 and a variety of D (D=3, 4, 5, 6, 7)

D=3		**D=4**		**D=5**		**D=6**		**D=7**	
C	**Matlab**	**C**	**Matlab**	**C**	**Matlab**	**C**	**Matlab**	**C**	**Matlab**
0.000	0.040	0.002	0.350	0.240	2.320	1.930	10.470	8.510	36.560
0.000	0.040	0.002	0.360	0.240	2.310	1.780	1.460	8.520	36.690
0.000	0.040	0.002	0.350	0.240	2.320	1.790	10.450	8.480	36.550
0.000	00240	0.002	0.360	0240	2.310	1.780	10.450	8.480	36.580
0.000	0.040	0.002	0.350	0.240	2.320	1.830	10.460	8.480	36.560
0.000	0.040	0.002	0.350	0.240	2.320	1.790	10.470	8.480	36.640
0.000	0.040	0.002	0.350	0.240	2.310	1.830	10.440	8.510	36.630
0.000	0.040	0.002	0.360	0.250	2.310	1.780	10.450	8.480	36.700
0.000	0.040	0.002	0.360	0.240	2.320	1.790	10.440	8.510	36.640
0.000	0.040	0.002	0.360	0.260	2.310	1.780	10.450	8.970	36.690
0.000	*0.040*	*0.020*	*0.355*	*0.243*	*2.315*	*1.808*	*10.454*	*8.563*	*36.624*

- N=40 and a variety of D (D=3, 4, 5, 6, 7)

D=3		D=4		D=5		D=6		D=7	
C	**Matlab**	**C**	**Matlab**	**C**	**Matlab**	**C**	**Matlab**	**C**	**Matlab**
0.000	0.200	0.200	3.090	4.740	43.130	72.670	426.400	800.560	out of memory
0.000	0.150	0.200	3.120	4.720	43.020	72.350	426.130	824.280	
0.000	0.160	0.200	3.07	4.720	43.060	72.250	426.140	810.160	
0.000	0.150	0.200	3.060	4.720	43.070	72.300	426.400	824.280	
0.000	0.150	0.200	3.050	4.700	43.070	72.460	426.300	812.600	
0.000	0.150	0.200	3.060	4.710	43.090	72.280	426.100	800.560	
0.000	0.150	0.200	3.060	4.700	43.070	72.490	426.300	824.280	
0.000	0.160	0.210	3.070	4.700	43.130	72.320	426.400	810.160	
0.000	0.150	0.210	3.060	4.710	43.070	72.320	426.500	800.560	
0.000	0.160	0.210	3.060	4.710	43.090	72.490	426.350	824.280	
0.000	*0.158*	*0.203*	*3.070*	*4.713*	*43.080*	*72.393*	*426.302*	*814.132*	

- N=60 and a variety of D (D=3, 4, 5, 6, 7)

D=3		D=4		D=5		D=6		D=7	
C	**Matlab**	**C**	**Matlab**	**C**	**Matlab**	**C**	**Matlab**	**C**	**Matlab**
0.010	0.360	0.740	10.660	25.770	230.080	612.250	out of memory	9811.450	out of memory
0.010	0.360	0.800	10.670	25.700	230.110	604.330		9813.350	
0.010	0.340	0.740	10.670	25.740	229.580	612.300		9813.450	
0.010	0.340	0.810	10.670	25.710	230.100	608.650		9813.350	
0.010	0.350	0.750	10.670	25.700	230.090	608.500		9813.350	
0.010	0.340	0.750	10.670	25.770	230.110	612.900		9811.450	
0.010	0.340	0.750	10.670	25.720	229.760	609.100		9812.500	
0.010	0.350	0.740	10.670	25.710	229.580	612.300		9820.100	
0.010	0.340	0.770	10.670	25.710	230.010	604.700		9821.200	
0.010	0.350	0.750	10.670	25.740	230.080	604.780		9821.100	
0.010	*0.347*	*0.760*	*10.669*	*25.727*	*229.950*	*608.981*		*9821.190*	

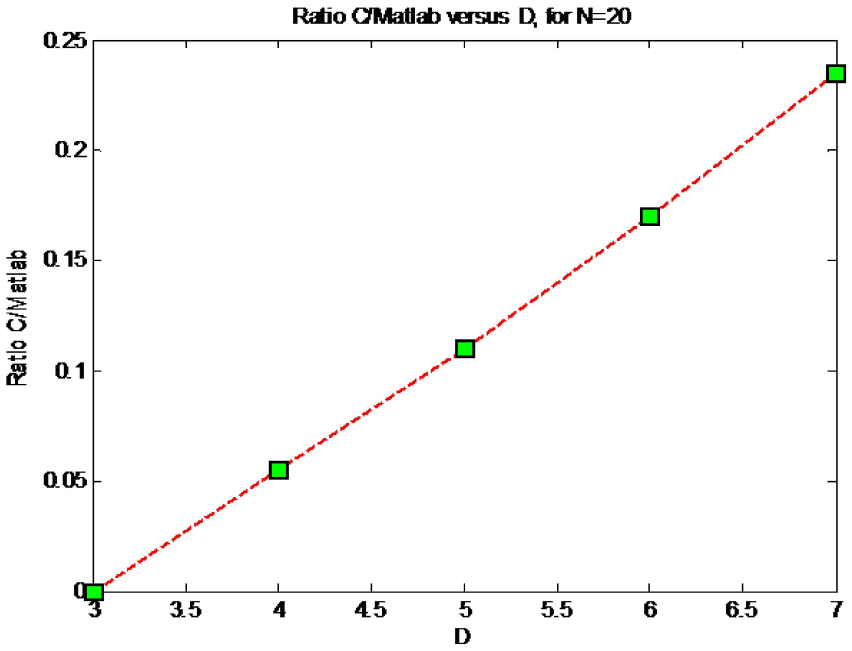

Figure 5.1: Ratio C/Matlab versus D, for N=20

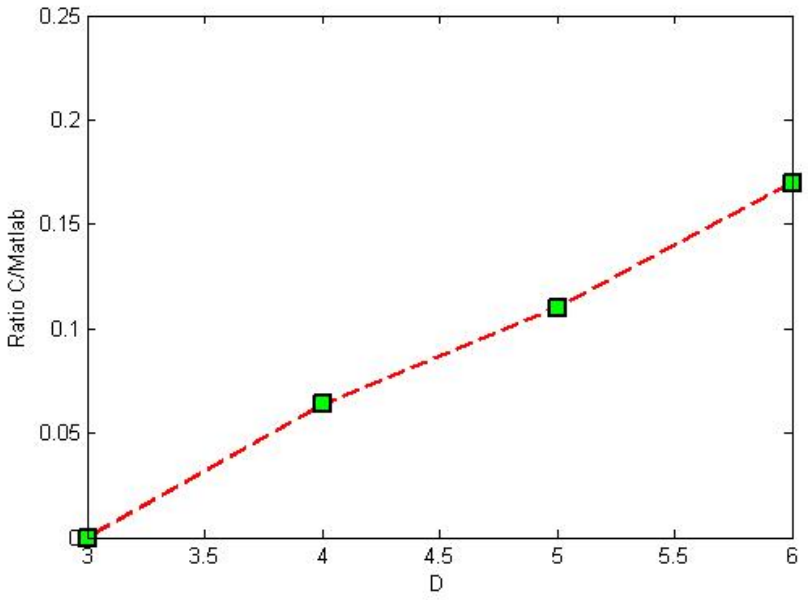

Figure 5.2: Ratio C/Matlab versus D, for N=40

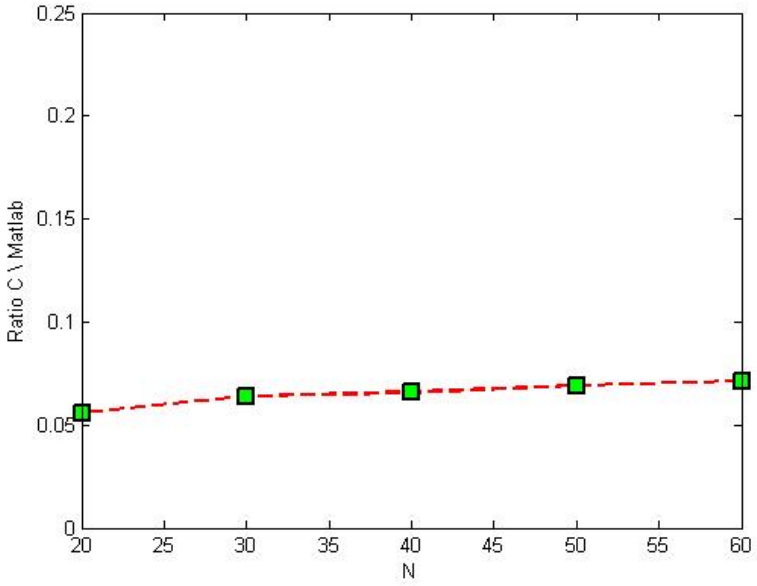

Figure 5.3: Ratio C/Matlab versus N, for D=4

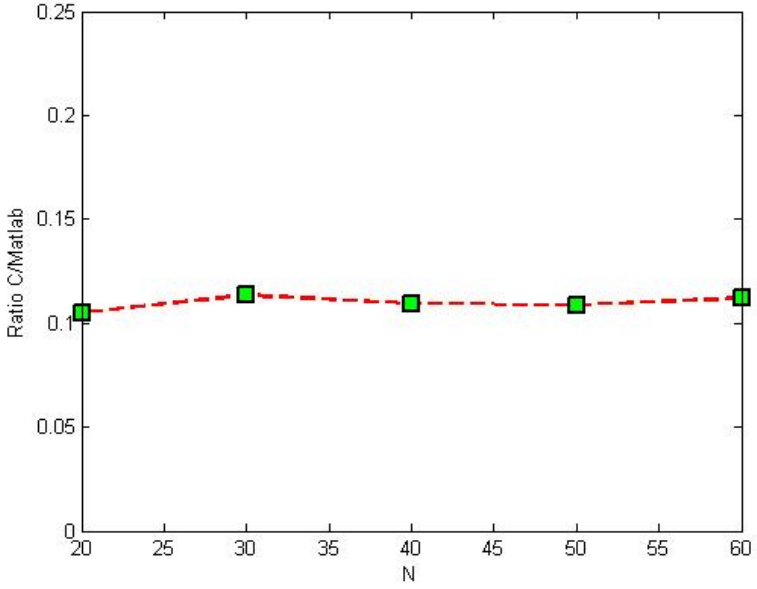

Figure 5.4: Ratio C/Matlab versus N, for D=5

It is noticable that for small values of N and D the ratio C towards Matlab is getting smaller distinguishablly. This is attributed to the interpenter that uses matlab each time which executes a program. Finally we observe, that the ratio grows up for bigger values of N and D but the ascendancy of C is discernible.

Chapter 6

Implementation in C programming language based on parallelizability

6.1 Implementation of parallizable C algorithm

A basic principle of the proposed algorithm concerns the parallelizability which means the $| \mathbf{X}(\mathbf{V}_{Nxd})| = \sum_{d=0}^{D-1} \binom{N-1}{d}$ cells are visited independently of each other so that the candidate vectors of $\mathbf{X}(\mathbf{V}_{NxD})$ are computed independently of each other. Hence, the algorithm is fully parallelizable. By taking into consideration this characteristic, our implementation in C was amended in order to execute our program simultaneously into a large number of processors.

For this purpose grid technology was used in order to implement and cover the needs of the algorithm. A definition for *Grid*; is an emerging infrastructure that provides seamless access to computing power and data storage capacity distributed over the globe.

Firstly it is noticeable that there were not any dependencies in our code so we don't have to use threads for this parallel implementation. The only control that it was used in order to testify that each processor was executing a single process was a bash script which denotes this test.

The modification, that was introduced, was included the following; firstly the number of inputs in main program except number of rows and columns were also included the number of processors(partsCount), which denote in how many parts the program is going to split and finally which part is being executed each time. In fact, the file that contains the combinations was split, and each time was being

executed the equivalent part according to the last argument. More specifically, the function compute takes three arguments in addition which are; firstly the variable startLines that denotes the number of lines that are being executed of the file that contains the combinations, secondly the variable parts that concerns which number of part will be run and finally the variable lines which concerns the number of the remaining lines of combination file of the last part that will be executed. For example if partsCount is k therefore

$$startLines = \left\lfloor \frac{\binom{N}{D-1}}{k} \right\rfloor$$

and

$$lines = \binom{N}{D-1} - startLines \times (k-1)$$

So each time we execute the nth part of the combination file until all parts will be executed. Each time an output file, that contains the nth part of candidate set of vectors, is produced and when all processes in all processors have been fulfilled, I put them in apposition in order to create the whole candidate set.

The only thing, which must be mentioned, is about the control of the processors in order to be ensured that each processor executes only one process each time. This can be achieved by creating a bash file that contains an if state. The conditions of this if statement include a variable named PBSARRAYID, which denotes the id of process that is implemented by a specific processor. So if PBSARRAYID is equal to one, it is ensured that the following statement will be executed by the first processor. We have to mention that the statement of this if declaration is the executable file of our program in which we assign the appropriate arguments. For example if number of processors is ten, I introduce an if condition which is parted by ten cases and the executable program in order to be fulfilled it must be run ten times separately. When all processes of both ten processors will end, the final candidate set will be constructed.

It must be noticed that I implement an additional source code, which links all output files into one which is the entire candidate set.

6.2 Parallel implementation versus single core implementation

In this part, ten processors are used in order to implement our program parallel. So the simulations concern with k=10 processors.

- D=3 and a variety of N (N=20, 30, 40, 50, 60) the required time in order to complete all processes both of ten processors is zero.

- D=5 and a variety of N (N=20, 30, 40, 50, 60)

N=20		**N=30**		**N=40**		**N=50**		**N=60**	
C parallel	**C**	**C parallel**	**C**	**C parallel**	**C**	**C parallel**	**C**	**C parallel**	**C**
0.025	0.240	0.160	1.470	0.540	4.740	1.410	11.750	3.130	25.770
0.025	0.240	0.160	1.460	0.540	4.720	1.400	11.740	3.110	25.700
0.025	0.240	0.160	1.460	0.540	4.720	1.400	11.760	3.130	25.740
0.025	0.240	0.160	1.460	0.540	4.720	1.400	11.750	3.130	25.710
0.025	0.240	0.160	1.460	0.540	4.700	1.400	11.740	3.130	25.700
0.025	0.240	0.160	1.460	0.540	4.710	1.400	11.740	3.110	25.770
0.025	0.240	0.160	1.470	0.540	4.700	1.410	11.760	3.130	25.720
0.025	0.250	0.160	1.470	0.540	4.700	1.410	11.760	3.110	25.710
0.025	0.240	0.160	1.470	0.540	4.710	1.410	11.750	3.110	25.710
0.025	0.260	0.160	1.470	0.540	4.710	1.410	11.750	3.110	25.740
0.025	*0.243*	*0.160*	*1.465*	*0.540*	*4.713*	*1.405*	*11.750*	*3.120*	*25.727*

- D=7 and a variety of N (N=20, 30, 40, 50, 60)

N=20		**N=30**		**N=40**		**N=50**		**N=60**	
C paral.	**C**	**C paral.**	**C**	**C paral.**	**C**	**C paral.**	**C**	**C paral.**	**C**
0.940	8.510	14.340	124.380	94.140	800.560	390.070	3267.300	1231.960	9811.450
0.940	8.520	14.330	124.310	93.630	824.280	390.520	3268.400	1233.450	9813.350
0.940	8.480	14.340	124.310	93.630	810.160	390.070	3269.500	1240.500	9813.450
0.940	8.480	14.330	124.380	94.140	824.280	390.520	3270.200	1230.450	9813.350
0.940	8.510	14.340	124.380	94.140	800.560	391.010	3267.400	1231.450	9813.350
0.940	8.510	14.330	124.380	93.630	824.280	390.520	3268.450	1231.960	9811.450
0.940	8.480	14.340	124.380	94.140	812.600	390.520	3267.340	1241.300	9812.500
0.940	8.660	14.330	124.310	93.700	800.560	390.450	3268.420	1231.960	9820.100
0.940	8.510	14.340	124.310	94.140	824.280	390.120	3269.500	1233.450	9821.200
0.940	8.970	14.330	124.310	94.140	810.160	391.800	3270.220	1230.070	9821.100
0.940	*8.563*	*14.335*	*124.310*	*93.943*	*814.132*	*390.560*	*3268.673*	*1233.655*	*9815.13*

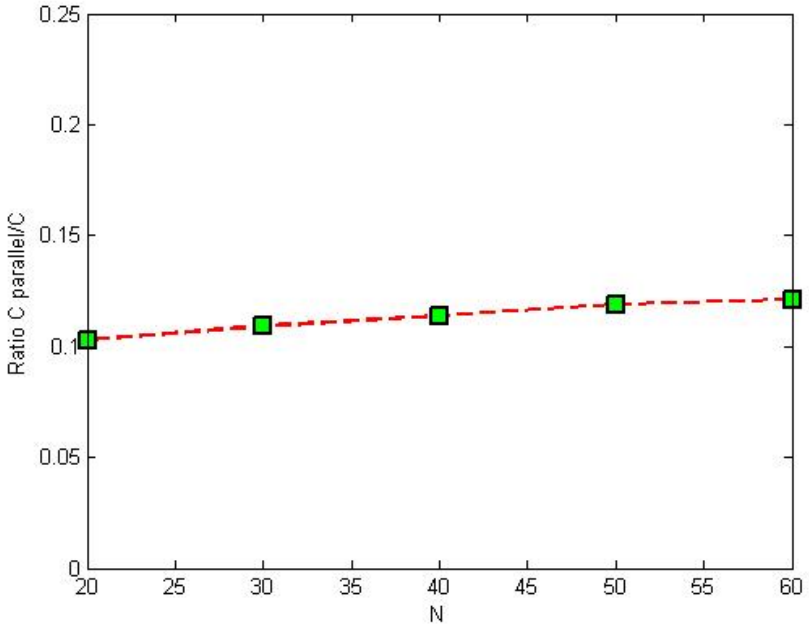

Figure 6.1: Ratio C parallel/C versus N, for D=5

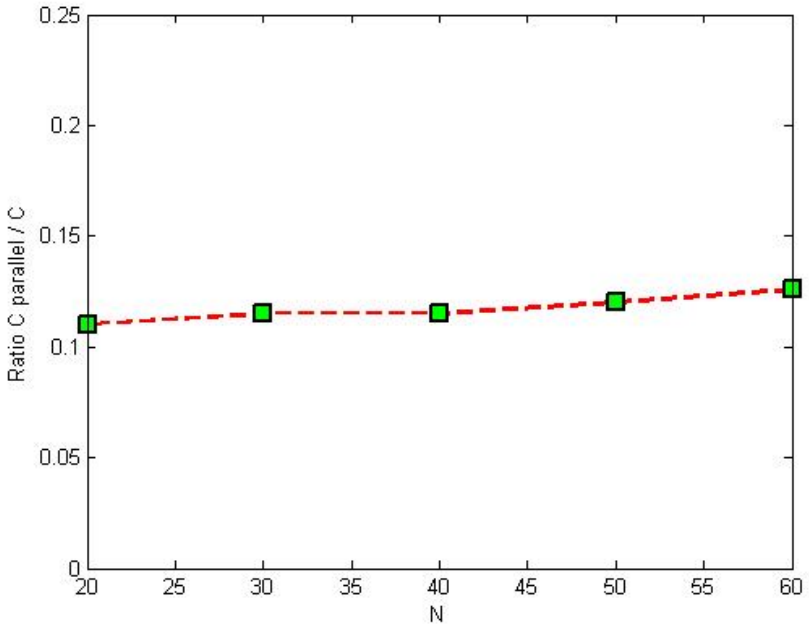

Figure 6.2: Ratio C parallel/C versus N, for D=7

In this part, twenty processors are used in order to implement our program parallel. So the simulations concern with k=20 processors.

- D=3 and a variety of N (N=20, 30, 40, 50, 60) the required time in order to complete all processes both of twenty processors is zero.

- D=5 and a variety of N (N=20, 30, 40, 50, 60)

N=20		**N=30**		**N=40**		**N=50**		**N=60**	
C parallel	**C**	**C parallel**	**C**	**C parallel**	**C**	**C parallel**	**C**	**C parallel**	**C**
0.013	0.240	0.080	1.470	0.270	4.740	0.700	11.750	1.550	25.770
0.013	0.240	0.080	1.460	0.270	4.720	0.710	11.740	1.560	25.700
0.013	0.240	0.080	1.460	0.280	4.720	0.700	11.760	1.550	25.740
0.013	0.240	0.080	1.460	0.280	4.720	0.700	11.750	1.560	25.710
0.013	0.240	0.080	1.460	0.280	4.700	0.700	11.740	1.560	25.700
0.013	0.240	0.080	1.460	0.280	4.710	0.700	11.740	1.560	25.770
0.013	0.240	0.080	1.470	0.270	4.700	0.710	11.760	1.560	25.720
0.013	0.250	0.080	1.470	0.280	4.700	0.710	11.760	1.550	25.710
0.013	0.240	0.080	1.470	0.270	4.710	0.710	11.750	1.550	25.710
0.013	0.260	0.080	1.470	0.270	4.710	0.710	11.750	1.550	25.740
0.013	*0.243*	*0.080*	*1.465*	*0.275*	*4.713*	*0.705*	*11.750*	*1.555*	*25.727*

- D=7 and a variety of N (N=20, 30, 40, 50, 60)

N=20		**N=30**		**N=40**		**N=50**		**N=60**	
C paral.	**C**	**C paral.**	**C**	**C paral.**	**C**	**C paral.**	**C**	**C paral.**	**C**
0.470	8.510	7.180	124.380	47.190	800.560	195.720	3267.300	619.920	9811.450
0.470	8.520	7.180	124.310	47.300	824.280	196.700	3268.400	620.220	9813.350
0.470	8.480	7.180	124.310	47.200	810.160	196.700	3269.500	618.120	9813.450
0.470	8.480	7.180	124.380	47.150	824.280	195.700	3270.200	620.300	9813.350
0.470	8.510	7.180	124.380	47.230	800.560	195.720	3267.400	621.500	9813.350
0.470	8.510	7.180	124.380	47.190	824.280	195.720	3268.450	615.150	9811.450
0.470	8.480	7.180	124.380	47.300	812.600	196.550	3267.340	616.300	9812.500
0.470	8.660	7.180	124.310	47.150	800.560	196.550	3268.420	619.200	9820.100
0.470	8.510	7.180	124.310	47.200	824.280	195.720	3269.500	619.100	9821.200
0.470	8.970	7.180	124.310	47.190	810.160	195.700	3270.220	618.120	9821.100
0.470	*8.563*	*7.180*	*124.345*	*47.210*	*814.132*	*196.078*	*3268.673*	*618.793*	*9815.13*

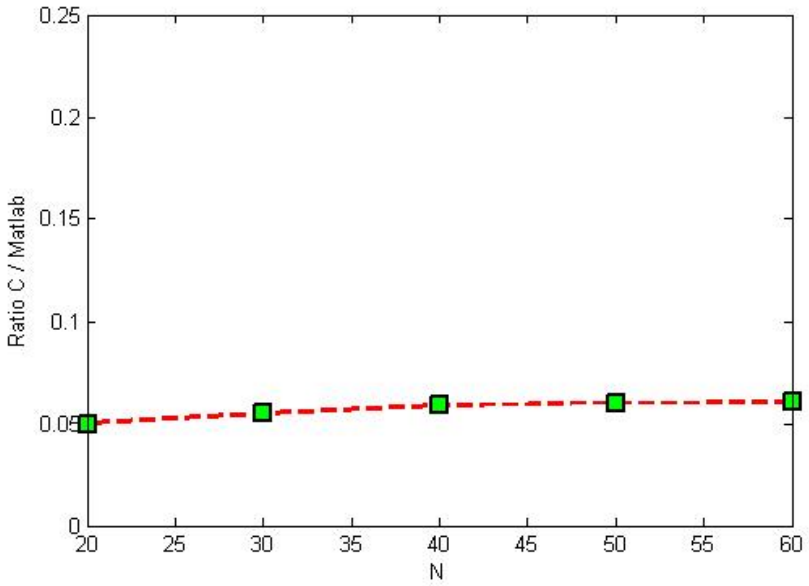

Figure 6.3: Ratio C parallel / C versus N, for D=5

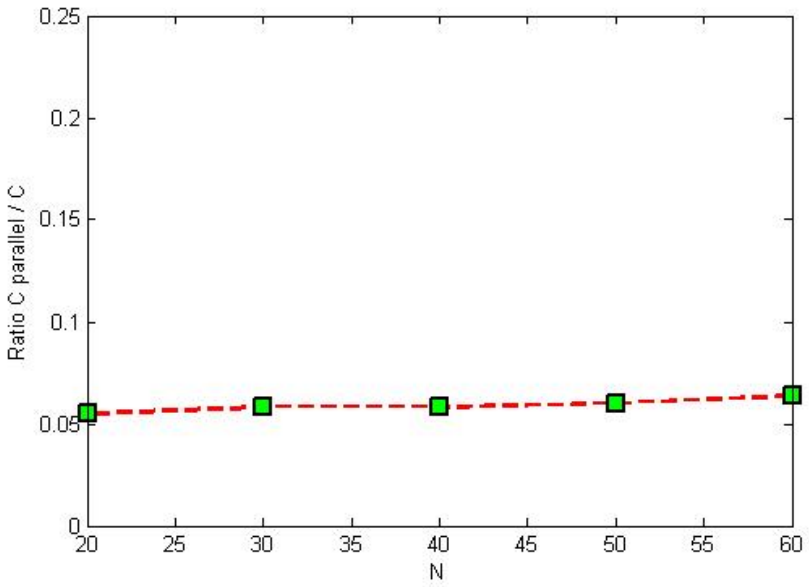

Figure 6.4: Ratio C parallel / C versus N, for D=7

Chapter 7

Sphere Decoding

7.1 Overview of Sphere Decoding and the Fixed SphereDecoder

This section provides a very brief overview of how the fixed sphere decoder differs from the standard sphere decoder

The maximum likelihood decoder for a MIMO receiver operates by comparing the received signal vector with all possible noiseless received signals corresponding to all possible transmitted signals. Under certain assumptions, this receiver achieves optimal performance in the sense of maximizing the probability of correct data detection. However, the complexity of this decoder increases exponentially with the number of transmit antennas, making it impossible to implement for large array sizes and high order digital modulation schemes.

The main idea of the Sphere Decoder is to reduce the computational complexity of the maximum likelihood detector by only searching over only the noiseless received signals that lie within a hypersphere of radius R around the received signal. Normally, this algorithm is implemented as a depth first tree search, where each level in the search represents one transmit antenna's signal. This is illustrated in Figure 2 below. If at a given level, a given branch exceeds the radius constraint, then that part of the tree can be removed from further consideration. Unfortunately, it is difficult to estimate how much of the tree needs to be searched in advance, since this depends on both the noise and the channel conditions. This means that the complexity of the sphere decoder is not fixed, but will typically vary with time.

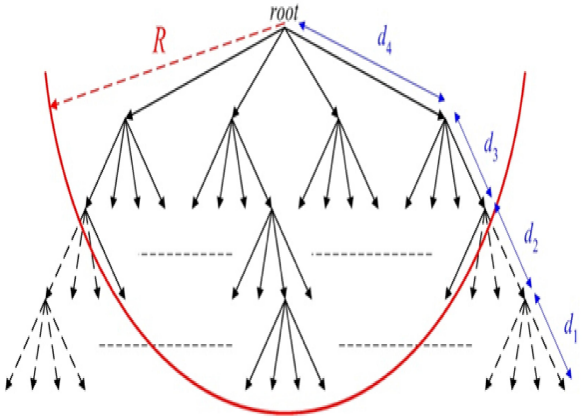

Figure 7.1: The Tree Structure and Sphere Constraint for the Sphere Decoder

The main idea behind the Fixed Sphere Decoder is to perform a search over only a fixed number of possible transmitted signals, generated by a small subset of all possible signals located around the received signal vector. This ensures that the detector complexity is fixed over time, a major advantage for hardward implementation. In order for such a search to operate efficiently, a key point is to order the antennas in such a way that most of the points considered relate to transmit antennas with the poorest signal-to-noise (SNR) conditions. Antennas with higher SNR conditions are much more likely to be detected correctly, based only on the received signal. Figure 3 below shows a hypothetical subset S in 4 transmit antenna, 4 receive antenna system with 4-QAM constellations used at each transmit antenna. The number of points considered per level (i.e. transmit antenna) is (n1, n2, n3, n4) = (1, 1, 2, 3). In each level, the closest points to the received signal are considered as components of the subset S. In this case, the Euclidean distance of only 6 transmitted vectors would be calculated, whereas the total number of possible vectors, 256, is much larger.

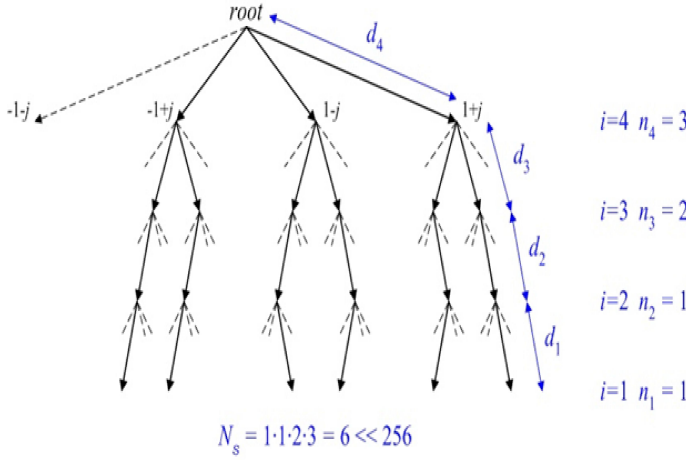

Figure 7.2: A Hypothetical Tree Structure for the Fixed Sphere Decoder

7.2 Reduction of algorithm that maximizes a quadratic form in sphere decoder

The basic principle of sphere decoder is related with the minimization of quantity

$$\min_{x} \|\mathbf{y} - \mathbf{Hx}\| \tag{7.1}$$

In order to use the tool that maximizes the quadratic form we follow the procedure below:

$$\begin{aligned}
&\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{Hx}\|^{\mathbf{2}} = \\
&\min_{\mathbf{x}} \| \mathbf{y} \|^{\mathbf{2}} + \mathbf{x^T H^T Hx} - \mathbf{y^T Hx} - \mathbf{x^T H^T y} = \\
&\min_{\widetilde{\mathbf{x}}=\left[\begin{smallmatrix}\mathbf{x}\\1\end{smallmatrix}\right]} \left\{ \widetilde{\mathbf{x}}^T \begin{bmatrix} \mathbf{H^T H} & \mathbf{H^T y} \\ -\mathbf{y^T H} & 0 \end{bmatrix} \widetilde{\mathbf{x}} \right\} = \\
&\max_{\widetilde{\mathbf{x}}=\left[\begin{smallmatrix}\mathbf{x}\\1\end{smallmatrix}\right]} \left\{ \widetilde{\mathbf{x}}^T (\mathbf{I} - \mathbf{B}) \widetilde{\mathbf{x}} \right\}
\end{aligned}$$

So relation 7.2 is equivalent with the asked type of quadratic form.

7.3 Reduction of sphere decoder in algorithm that maximizes a quadratic form

As we have mentioned above, we have to maximize the quantity

$$\max_{\mathbf{x}} \left\{ \mathbf{x^T A x} \right\} \tag{7.2}$$

In order to use the sphere decoder tool we follow the procedure below:

$$\max_{\mathbf{x}} \left\{ \mathbf{x^T A x} \right\} = min_{\mathbf{x}} \left\{ -\mathbf{x^T} \lambda \mathbf{x} + \lambda \mathbf{x^T I x} \right\} = \min_{\mathbf{x}} \left\{ \mathbf{x^T} \underbrace{(\lambda \mathbf{I} - \mathbf{A})}_{\mathbf{B}} \mathbf{x} \right\}$$

$$= \min_{\mathbf{x}} \begin{bmatrix} \widetilde{\mathbf{x}}^T & 1 \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{B}} & -\mathbf{b} \\ -\mathbf{b^T} & c \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{x}}^T \\ 1 \end{bmatrix} = \min_{\widetilde{\mathbf{x}}} \left\{ \widetilde{\mathbf{x}}^T \widetilde{\mathbf{B}} \widetilde{\mathbf{x}} - \mathbf{b^T} \widetilde{\mathbf{x}} - \widetilde{\mathbf{x}}^{\mathbf{T}} \mathbf{b} + \right\}$$

$$= \min_{\mathbf{x}} \left\{ \| \mathbf{a} \|^{\mathbf{2}} + \widetilde{\mathbf{x}}^{\mathbf{T}} \mathbf{L} \mathbf{L}^{\mathbf{T}} \widetilde{\mathbf{x}} - \widetilde{\mathbf{a}}^{\mathbf{T}} \mathbf{L}^{\mathbf{T}} \widetilde{\mathbf{x}} - \widetilde{\mathbf{x}}^{\mathbf{T}} \mathbf{L} \mathbf{a} \right\}$$

$$= \min_{\mathbf{x}} \| \mathbf{a} - \mathbf{L}^{\mathbf{T}} \widetilde{\mathbf{x}} \|^{\mathbf{2}}$$

And $\lambda - \lambda i \geq 0 \leftrightarrow \lambda \geq \lambda i \leftrightarrow \lambda = \max(\lambda i)$

So relation 7.3 is equivalent with the asked type of sphere decoder form.

7.4 Comparisons Single core implementation versus sphere decoding

- D=3 and a variety of N (N=30, 40, 50, 60)

N=30		**N=40**		**N=50**		**N=60**	
C	**Sphere**	**C**	**Sphere**	**C**	**Sphere**	**C**	**Sphere**
0.000	0.000	0.000	0.100	0.420	1.040	0.010	294.000
0.000	0.000	0.000	0.100	0.420	1.040	0.010	295.000
0.000	0.000	0.000	0.100	0.420	1.040	0.010	293.300
0.000	0.000	0.000	0.100	0.420	1.040	0.010	291.010
0.000	0.000	0.000	0.100	0.420	1.040	0.010	296.020
0.000	0.000	0.000	0.100	0.420	1.040	0.010	295.100
0.000	0.000	0.000	0.100	0.420	1.040	0.010	294.000
0.000	0.000	0.000	0.100	0.420	1.040	0.010	291.700
0.000	0.000	0.000	0.100	0.420	1.040	0.010	294.100
0.000	0.000	0.000	0.100	0.420	1.040	0.010	295.200
0.000	*0.000*	*0.000*	*0.100*	*0.420*	*1.040*	*0.010*	*293.943*

- D=4 and a variety of N (N= 40, 50, 60,70)

N=40		N=50		N=60		N=70	
C	**Sphere**	**C**	**Sphere**	**C**	**Sphere**	**C**	**Sphere**
0.200	0.400	0.420	1.220	0.740	64.930	1.230	1041.200
0.200	0.400	0.420	1.230	0.800	62.200	1.250	1041.200
0.200	0.400	0.420	1.220	0.740	62.270	1.240	1041.200
0.200	0.400	0.420	1.240	0.810	64.300	1.240	1060.400
0.200	0.400	0.420	1.220	0.750	62.380	1.240	1047.340
0.200	0.400	0.420	1.220	0.750	64.930	1.230	1041.200
0.200	0.400	0.420	1.220	0.750	62.100	1.250	1047.340
0.210	0.400	0.420	1.220	0.740	63.450	1.250	1047.340
0.210	0.400	0.420	1.220	0.770	62.600	1.230	1041.200
0.210	0.400	0.420	1.220	0.750	64.500	1.240	1060.400
0.203	*0.400*	*0.420*	*1.220*	*0.760*	*63.336*	*1.240*	*1046.882*

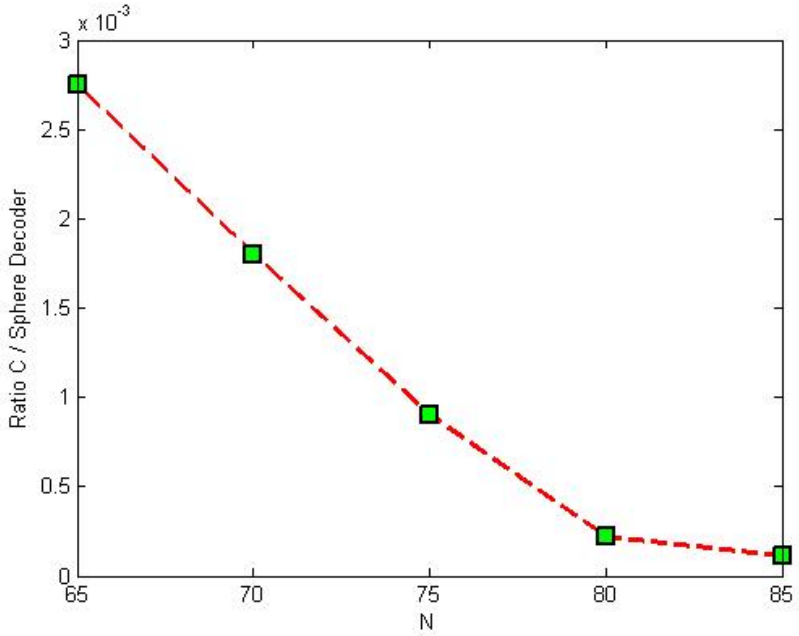

Figure 7.3: Ratio C / Sphere Decoder versus N, for D=3

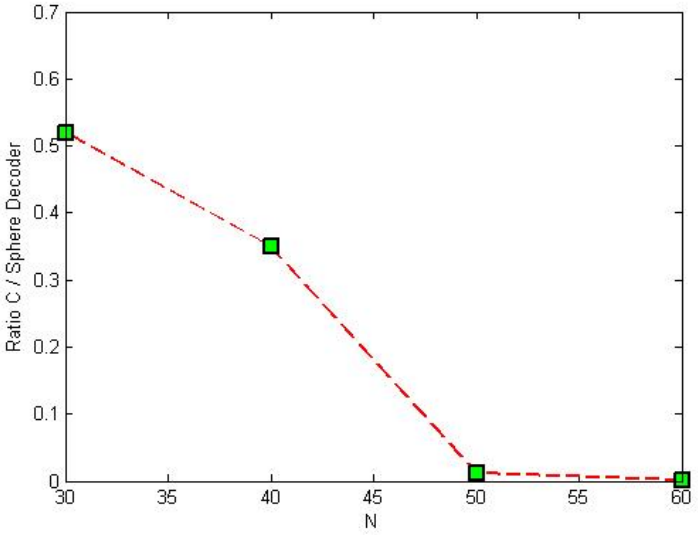

Figure 7.4: Ratio C / Sphere Decoder versus N, for D=4

Observe that the proposed algorithm for large N and small D is more efficient than sphere decoding. This is atributed to the implemantation of sphere decoder and it depends on the size of the tree. Furthermore for low SNR, the complexity of sphere decoder is affected. So complexity of sphere decoding algorithm depends on the size of the generated tree, which is equal to the sum of the number of lattice points in spheres of radius r and dimensions $d = 1, ..., D$. In our case, we choose an infinite radius and according to [1] the complexity of sphere decoder for large N and small D increases rapidly. The simulations that are gathered, concern $SNR = 7db$ and it is distinct that for these cases the proposed algorithm is the optimal solution.

Chapter 8

Applying the algorithm in a Space time block coding system

8.1 System model and problem statement

In this chapter, we are going to present the modeling of the system and define the problem. The main section is to proceed this model into the quadratic form in order to incorporate our algorithm.

We consider a multiple-input multiple-output (MIMO) system with M_t transmit and M_r receive antennas that employs orthogonal space-time coded transmission of size $M_t \times T$ and rate $R = \frac{N}{T}, N \leq T$. We assume transmission of binary data that are split into vectors of N bits. Each bit vector forms a corresponding space-time block (matrix) of size $M_t \times T$. The $M_t \times T$ space-time block $\mathbf{C}(\mathbf{s}) \in \mathbb{C}^{M_t \times T}$ that corresponds to the $N \times 1$ data vector $s \in \{pm1\}^N$ is given by

$$\mathbf{C}(\mathbf{s}) = \sum_{n=1}^{N} X_n s_n \tag{8.1}$$

where $s_n = \pm 1$ denotes the nth element of $s, n = 1, 2, \ldots, , , N$ and $X_n \in \mathbb{C}^{M_t \times T}, n = 1, 2, , , , , \ldots, N$, are orthogonal space-time codes that satisfy the equation

$$\mathbf{C}(\mathbf{s})\mathbf{C}^H(\mathbf{s}) = \| \ s \ \|^2 \ \mathbf{I}_{Mt} = T\mathbf{I}_{Mt} \tag{8.2}$$

For any $s_n = \pm 1$. Equation 8.2 denotes orthogonality and leads to maximum spatial diversity gain. Let $s^{(p)} = \left[s_1^{(p)} s_2^{(p)} \ldots s_N^{(p)} \right]$ denote the data vector contained in the pth transmitted code block $p = 1, 2, 3, \ldots$. The down converted and pulse-

matched equivalent pth received block of size $M_r \times T$ is

$$\mathbf{Y}^p = \mathbf{H}^p \mathbf{C}\left(\mathbf{s}^{(p)}\right) + \mathbf{V}^p \tag{8.3}$$

In 8.3, $\mathbf{H}^p \in \mathbb{C}^{M_r \times M_t}$ refers to the pth transmission and represents the channel matrix between the M_t transmit and M_r receive antennas. In general, $\mathbf{H}^p$ consists of correlated coefficients that are modeled as circular complex Gaussian random variables and account for flat fading. We assume that all collected energy is absorbed by the channel matrix $\mathbf{H}^p$. In addition, $\mathbf{V}^p \in \mathbb{C}^{M_r \times T}$ denotes zero-mean additive spatially and temporally white circular complex Gaussian noise with variance σ_ν^2. The channel and noise matrices $\mathbf{H}^p$ and $\mathbf{V}^p$ respectively,,,, $p = 1, 2, 3, \ldots,$ are independent of each other.

If the receiver has knowledge of the channel matrix, then coherent ML detection simplifies to one-shot block decisions according to

$$\widetilde{s}^{(p)} = \sinh n \left(\Re \left\{ tr \left\{ \mathbf{Y^{(p)}} \mathbf{X_n^H} \left(\mathbf{H^{(p)}} \right)^{\mathbf{H}} \right\} \right\} \right), n = 1, 2, \ldots, , , N, p = 1, 2, 3, \ldots \tag{8.4}$$

In this work, we assume that the channel matrices $\mathbf{H}^{(p)}, p = 1, 2, 3, \ldots$ are not available to the receiver. Hence, coherent detection in **??** cannot be utilized and the ML receiver takes the form of a sequence detector. We consider a sequence of P space-time blocks consecutively transmitted by the source and collected by the receiver, say $\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(p)}$ and form the $M_r \times T$ observation matrix

$$\mathbf{Y} \doteq \left[\mathbf{Y^{(1)}} \ldots \mathbf{Y^{(p)}} \right] = \left[\mathbf{H^{(1)}} \mathbf{C}\left(\mathbf{s^{(1)}}\right) \ldots \mathbf{H^{(p)}} \mathbf{C}\left(\mathbf{s^{(p)}}\right) \right] + \left[\mathbf{V^{(1)}} \ldots \mathbf{V^{(p)}} \right] \tag{8.5}$$

In the sequel, based on the observation of P blocks at the receiver we present ML noncoherent detection developments.

8.2 Maximum-Likelihood Noncoherent Detection and the special case of time-invariant Rayleigh fading

The ML maximizes the optimal decision according to the equation;

$\hat{\mathbf{s}}_{opt} = \arg\max f(\mathbf{Y}|\mathbf{s}) = \arg\max \mathbf{f}(\mathbf{vec}(\mathbf{Y})|\mathbf{s}) = \arg\max \mathbf{f}(\mathbf{y}|\mathbf{s})$. It has been proved that ML noncoherent OSTBC is attained with polynomial complexity if the mean channel vector belongs to the range of the channel covariance matrix whose rank is not a function of the sequence length. Since the time-invariant Rayleigh fading is a special case of channel model, this conclusion which concerns the complexity is immediately in effect for this case too.

The ML detector becomes [8] $\hat{\mathbf{s}}_{opt} = \arg\min_{\mathbf{s}^{(p)}\in\{\pm 1\}^N} \parallel \mathbf{V^T s} \parallel$ where $\mathbf{V} \doteq [\Re\{\mathbf{A}\}\, \Im\{\mathbf{A}\}]$ and $\mathbf{A} \doteq \mathbf{Z}\left(\mathbf{I_{Mt}} \otimes \mathbf{Y^H}\right) \underline{\mathbf{Q}} \left(\mathbf{I_D} + \frac{\mathbf{TP}}{\sigma_\nu^2}\underline{\mathbf{\Sigma}}\right)^{-\frac{\mathbf{1}}{\mathbf{2}}}$ Note that the latter is always feasible, since $\mu = 0$ belongs to the range of $\mathbf{C_h}$.

8.3 Integration of our algorithm in a 2 × 2 MIMO system

We consider a 2×2 MIMO system that employs Alamouti space-time coding (with rate $R = \frac{N}{T} = 1$, since $N = T = 2$ to transmit binary data in an unknown Rayleigh fading channel environment. Space-time ambiguity induced by the rotatability of the Alamouti code is resolved by employing differential space-time modulation due to which the pth transmitted space-time block is

$\mathbf{C}^{(p)} = \mathbf{C}^{(p-1)}\mathbf{X}^{(p)}$ where $\mathbf{X}^{(p)} = \begin{bmatrix} s_1^p & 0 \\ 0 & s_2^p \end{bmatrix}$ if $s_1^p s_2^p > 0, \mathbf{X}^{(p)} = \begin{bmatrix} 0 & s_2^p \\ -s_1^p & 0 \end{bmatrix}$ if $s_1^p s_2^p < 0$ and $\mathbf{C}^{(0)} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ so that $\mathbf{C}^{(p)}$ follows the Alamouti code structure, for any $p = 1, 2, \ldots$. According to the literacy, we adopt the model in which the covariance matrix is equal with $\mathbf{C}_{\underline{h}} = \begin{bmatrix} 1 & r & t & w_1 \\ r^* & 1 & w_2 & t \\ t^* & w_2^* & 1 & r \\ w_1^* & t^* & r^* & 1 \end{bmatrix}$. In our consideration we set $t = r = 0$ and $w_1 = w_2 = 1$, this setup provides us a higher ergodic capacity.

Observe that the rank of such a matrix is 2, therefore the overall complexity of the proposed ML receiver becomes $O\left(P^4\right)$.

We focus on Rayleigh fading channel and demonstrate the bit error rate of the non coherent receiver, by denoting a specific SNR(in our case is 8db), as a function of parameter N which concerns the size of the problem. Note; we have to mention that the size of the problem N and sequence lengths P are given by the relation $N = 2P$. For $D = 8$ and over 150 channel realizations, we present the fluctuation of BER versus N. We observe that as N increases the BER is reduced and for only small values of N BER is reduced too.

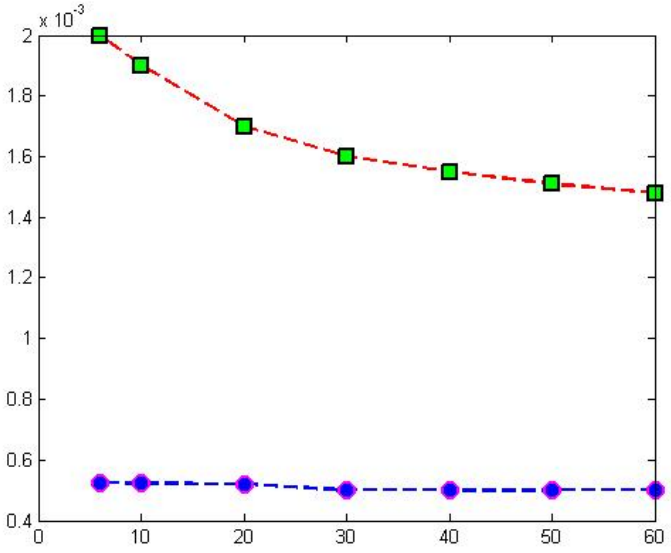

Figure 8.1: BER versus N, for SNR=8db Demonstrate Coherent Detection and Non Coherent Detection

Conclusions

We considered the problem of identyfing vector that maximizes a rank deficient quadratic form. I implemented this in C programming language. The algorithm was proved that is full parallizable. Taking the advantage of this property, I implement the algorithm parallel in many processors in order to increase the speed up of our program. Furthermore the proposed algorithm was compared with sphere decoder and observed that was more efficient on time issues for specific parameters. Finally, we demonstrate the efficiency of the algorithm by nesting in a OSTBC system and notice that the simulation time is really improved, when the proposed algorithm is implemented in C.

Bibliography

[1] B. Hassibi and H. Vikalo. "on the sphere - decoding algorithm i. expected complexity ". *IEEE Trans. Signal Proc, vol 53, pp. 2806-2818*, August 2005.

[2] K. Fukuda J. A. Ferrez and Th. M. Liebling. "solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm". *European Journal of operational research, vol. 166, pp 35-50*, 2005.

[3] T.M.Liebling K. Allemand, K. Fukuda and E. Steiner. "a polynomial case of unconstrained zero-one quadratic optimization". *Mathematical programming Series*, Oct 2001.

[4] G. N. Karystinos and A. P. Liavas. "efficient computation of the binary vector that maximizes a rank-deficient quadratic form". *IEEE Transactions on Information Theory*, 2006.

[5] G. N. Karystinos and D. A. Pados. "efficient computation of the binary vector that maximizes a rank-3 quadratic form". *Proc. 2006 Allerton Conference on Communication, Control, and Computing, Allerton House, Monticello*, 2006.

[6] G. N. Karystinos and D. A. Pados. "rank-2-optimal binary spreading codes". *Proc. 2006 Conf. on Inform. Sc. and Syst. (CISS 2006), Princeton University, Princeton, NJ*, 2006.

[7] J C Nash. *Compact Numerical Methods For Computers, Linear Algebra and Function Minimization, Second Edition.* 1990.

[8] D. S. Papailiopoulos and G. N. Karystinos. "maximum-likelihood noncoherent ostbc detection with polynomial complexity". *IEEE Transactions on Wireless Communications*, 2006.

[9] R. Schober V. Pauli, L. Lampe and K. Fukuda. "multiple-symbol differential detection based on computational geometry". *IEEE International conference on communications, Glasgow, Scotland*, June 2007.

[10] H. Vikalo and B. Hassibi. "on the sphere - decoding algorithm ii. generalizations, second-order statistics, and applications to communications ". *IEEE Trans. Signal Proc, vol 53, pp. 2819-2834*, August 2005.