1-1-2007

Adaptive vector greedy splitting algorithm

Evgeny Klavir
Ryerson University

"Adaptive Vector Greedy Splitting Algorithm"

by

Evgeny Klavir

A project
presented to Ryerson University
in partial fulfillment of the
requirement for the degree of
Master of Engineering
in the Program of
Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2007

Author's Declaration

I hereby declare that I am the sole author of this thesis.
I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.
Signature

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.
Signature

Instructions on Borrowers

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Title of Project: **"ADAPTIVE VECTOR GREEDY SPLITTING ALGORITHM"**

Evgeny Klavir, Master of Engineering, 2007

Department of Electrical and Computer Engineering,
School of Graduate Studies, Ryerson University

Project directed by: Dr. Sebastian Ferrando
Chair, Department of Mathematics

We introduce a new transform through a construction that we have called the Adaptive Vector Greedy Splitting algorithm. The main idea behind this algorithm is an optimization step based on the simple Bathtub Principle. We use the Vector Greedy Splitting algorithm to build orthonormal bases for a given vector of random variables (also called signals). A particular basis constructed in this way may be used for signal compression, audio pattern recognition and other applications of signal processing.
We compare performance of the Vector Greedy Splitting algorithm with the Haar wavelet transform applied to the same vector of input signals.
The implementation of the algorithms and statistics accumulation are made using the ANSI C computer language and Matlab. The work uses advanced methods of Computer Engineering and Digital Signal Processing.

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Sebastian Ferrando, who guided me along the way. Without his help and advice it would not have been possible to complete this project.
I would also like to thank my family for their constant support during my time at Ryerson University.

Contents

List of Figures

Chapter 1

Introduction

The general area related to the subject of this project is adaptive approximations. Some general references are given by: [9], [6], [14]. More especialized references are given by: [4], [5], [7], [8]. The thesis does not use any particular result from this area hence we will only describe, briefly, the main goals and ideas behind an adaptive approximation.

The idea of adaptive vector greedy splitting algorithm is described in [3] and the the project complements work [3] with:

- Formal proof of main theorems.
- Studying properties of vector greedy splitting algorithm.
- Efficient software implementation of the algorithm.
- Plotting results and comparison of vector greedy splitting algorithm vs. Haar wavelets.

Given a collection of analyzing signals, which we collect in a set called a dictionary $\mathcal{D}$, we try to obtain efficient representations of empirical signals. Namely, given a collection of data signals, which we will call $\mathcal{X}$, we look for an approximation to a given $X \in \mathcal{X}$ by using elements from $\mathcal{D}$. Some results from the theory indicate that having an efficient representation will provide optimal results in applications such as compression and denoising. Moreover, if the dictionary $\mathcal{D}$ is properly designed the approximations may also be interpreted in physical terms. Below we explain some of the ideas in mathematical notation, most of the mathematical symbols and notions will be introduced formally later in this document.

Consider a probability space $(\Omega, \mathcal{A}, P)$ and the associated Hilbert space $L^2(\Omega, \mathcal{A}, P)$. An orthonormal system of functions $\{u_k\}_{k\geq 0}$ defined on Ω is called an *H-system* if

and only if for any $X \in L^2(\Omega, \mathcal{A}, P)$

$$X_{\mathcal{A}_n} \equiv \mathbf{E}(X|u_0, u_1, \ldots, u_n) = \sum_{k=0}^{n} \langle X, u_k \rangle u_k, \quad \text{for all } n \geq 0. \tag{1.1}$$

This thesis studies some aspects of these type of orthonormal systems and their potential to perform lossy compression. From a theoretical point of view they are interesting as the sequence of approximations is a martingale. Moreover, for a given collection of random variables $\mathcal{X} = \{X_1, \ldots, X_d\}$, we can construct the system $\{u_k\}$ adaptively to obtain efficient approximations. Notice that the word *adapted* is used with two meanings. Firstly, the system $\{u_k\}$ will be constructed, in an optimal way, using the set $\mathcal{X}$ (and hence adapted to $\mathcal{X}$) and, secondly, the functions u_k are simple functions which are adapted (according to measure theory) to the sigma algebras $\sigma(u_0, \ldots, u_n)$ which form a natural filtration in the sense that they generate $\sigma(\mathcal{X})$.

The main contribution of the thesis is in the construction, via the greedy splitting algorithm, of adapted H-systems and in showing their usefulness in a concrete setting. The main step in our construction is explained next. Consider the dictionary of (generalized) Haar functions,

$$\mathcal{D} \equiv \{\psi = a\ \mathbf{1}_A + b\ \mathbf{1}_B, \quad a, b \in \mathbb{R}, \quad A \cap B = \emptyset, A, B \in \mathcal{A}\},$$

we also require that each $\psi \in \mathcal{D}$ satisfies

$$\int_\Omega \psi(\omega)\ dP(\omega) = 0, \qquad \int_\Omega \psi^2(\omega)\ dP(\omega) = 1.$$

Under appropriate conditions on a random variable X we show how to construct $\psi_0 = a\ \mathbf{1}_{A_{1,0}} + b\ \mathbf{1}_{A_{1,1}} \in \mathcal{D}$ such that

$$\langle X, \psi_0 \rangle = \sup_{\psi \in \mathcal{D}} \langle X, \psi \rangle. \tag{1.2}$$

The above result represents the main step to set up a greedy algorithm (see [19]), also called a pursuit algorithm (see [14]), on an overcomplete dictionary like $\mathcal{D}$. Equation (1.2) can be iterated by replacing X above by the residual $RX \equiv X - \langle X, \psi_0 \rangle \psi_0$. More generally, we set $R^{n+1}X \equiv R^n X - \langle R^n X, \psi_n \rangle \psi_n$ where ψ_n is the Haar function chosen at iteration n and, of course, $R^0 X \equiv X$.

If the resulting approximation is to be used to perform a (lossy) compression on the information contained in X, a main problem to address is the cost to encode the Haar functions ψ_k. The cost of this encoding can be greatly reduced by performing a restricted nonlinear approximation ([5], [4]) which in our case will imply to impose a tree structure on the supports of the Haar functions ψ_n. For example, in the first iteration, this restriction entails to apply (1.2) to $(\mathbf{1}_{A_{1,i}} RX)$, $i = 1, 2$, alternately.

Even when restricting the approximation to have a tree structure, the cost of storing a single H-system for each given input X is too high for a compression application. We approach this problem by extending the main step given by (1.2) to a vector setting, namely we will deal with vector valued Haar functions $\psi = a\ \mathbf{1}_A + b\ \mathbf{1}_B$ where now $a, b \in \mathbb{R}^d$ and $X = (X_1, \ldots, X_d) \in L^2(\Omega, \mathbb{R}^d)$. We will then show how to find ψ_0 so that

$$[X, \psi_0] \equiv \int_\Omega \langle X(w), \psi_0 \rangle \ dP(w) = \sup_{\psi \in \mathcal{D}} [X, \psi]. \tag{1.3}$$

In other words, we introduce a new transform for signal analysis. Our input is a collection of random variables, also called signals, which we collect in a single vector. In the special case of a single random variable (scalar signal case) we call our transform Greedy Splitting (GS) algorithm, while in the vector case, i.e. a given collection of random variables defined on the same probability space, the transform is called Vector Greedy Splitting (VGS) algorithm. We use the Vector Greedy Splitting algorithm to build orthonormal basis for a given input vector. This basis is used for signal approximation which is performed via an analysis step, then a compression step and finally a synthesis step. This basis is built iteratively and is constructed by splitting the domain of the signals and generating a sequence of partitions. To each new split we associate two vectors in our space and a corresponding basis function, called a (generalized) Haar function (or *generalized* Haar function). These Haar functions take only two non zero vector values. We start with our domain and split it into 2 parts, becoming sub-domains. This split and corresponding Haar function become the root of a tree and the sub-domains are leaves. In the next step we split one of the leaves depending which one carries more useful information about the given signals. Iteratively we continue this process of building a tree, in each iteration we split one of the sub-domains. The algorithm is called Adaptive Vector Greedy algorithm since we always choose best leaf to split and the choice depends on the vector of inputs. We call this tree *Partition Tree* and we present it in the Figure 1.1:

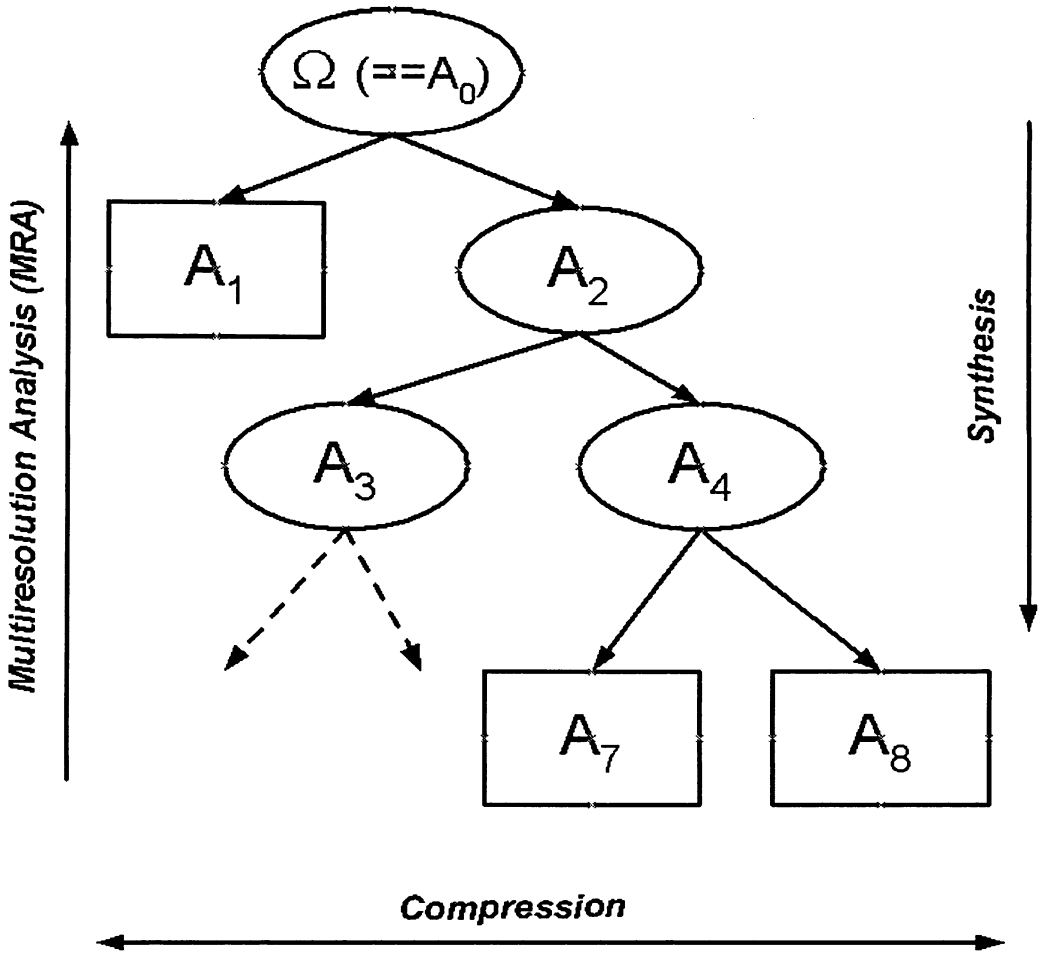

Figure 1.1: Partition tree and main steps of VGS algorithm.

We see at Figure 1.1 the partition tree. It is a binary tree that is not balanced and not complete since adaptive VGS algorithm at each iteration chooses the next leaf to split based on optimality criterion described in Chapter 2.2. It may happen that some leaves on the top of the tree will be never split. On the Figure 1.1 we observe that left child of the root, denoted as A_1 is not split.

In order to build tartition tree *adaptively* we define an inner product, and associated norm, in the vector space. The best Haar function and the best partition means that the associated Haar function carries the optimal amount of information about the input signals. The efficiency of the splitting algorithm has been achieved by using the Bathtub principle. After M iterations we obtain a tree with $M + 1$ leaves and M nodes and M Haar functions, corresponding to the nodes. The orthonormal set of basis functions so constructed is called an H-system. The next stage, multiresolution analysis, is the calculation of the inner products of the signals and all basis elements

(i.e. the Haar functions). The physical meaning of the inner product is the amount of signal energy represented by each Haar function. There are recursive formulae for the tree construction allowing fast calculation of these values, starting from the leaves and going up to the root, as shown in the Figure 1.1. The next optional stage is compression of the tree, by compression we mean setting to zero a certain percentage of the smallest inner products calculated in the analysis stage. The final stage is synthesis, or reconstruction of the signal. Also, there are recursive formulae starting from the root and going down to the leaves of the tree (see Figure 1.1), making this process fast and efficient. This concludes the VGS algorithm. The performance of the algorithm is compared to the classical Haar wavelet transform. The relative errors of both methods, applied to the same sets of input vector signals, have been compared and plotted. The implementation of the algorithms is done in ANSI C programming language and uses advanced methods of digital signal processing making the core algorithm easily portable to embedded systems. Matlab is used for statistical representation of the results.

1.1 Organization of the Document

The main notation used and mathematical formulae are presented in Section 1.2.

Chapter 2 explains in simple terms the procedure to construct adaptive sequence of partitions for the space (resulting in a binary tree), how this sequence defines an orthonormal basis and the special basis functions called Haar functions. Chapter 3 describes the major steps for an efficient implementation of Adaptive Vector Greedy Splitting Algorithm, in particular the recursive formulae needed to implement the Multiresolution Analysis Algorithm for a given vector of input signals, transform compression and synthesis. Chapter 4 presents testing results and comparison with the classical Haar wavelet transform. Chapter 5 summarizes the results of the thesis and indicates potential applications. The detailed proofs of all theoretical results are presented in Appendixes. Appendix A defines Haar functions and their properties. Appendix B describes how to build the VGS partition tree, provides proofs of formulae and presents a proof of the main theorem, that is, the bathtub principle. Appendix C describes how to cover multi-dimensional unit sphere with uniformly distributed points, that is an important step in the implementation of the Vector GS algorithm.

1.2 Notation

Consider a probability space $(\Omega, \mathcal{A}, P)$ and the associated Hilbert space $L^2(\Omega, \mathbb{R}^d)$. In the case when Ω has a finite number of elements, the collection $\mathcal{A}$ will then be the collection of all subsets of Ω. Elements from $L^2(\Omega, \mathbb{R}^d)$ are vector valued random variables $X : \Omega \to \mathbb{R}^d$, $X(w) = \{X_1(w), \ldots, X_d(w)\}$, the components X_i will be the given input signals. The inner product in $L^2(\Omega, \mathbb{R}^d)$, for two vector valued random variables X and Y, is given by

$$[X, Y] \equiv \int_\Omega \langle X(\omega), Y(\omega) \rangle \, dP(\omega), \tag{1.4}$$

where $\langle \ , \ \rangle$ is the Euclidean inner product in $\mathbb{R}^d$, namely,

$$\langle X(\omega), Y(\omega) \rangle = \sum_{i=1}^{d} X_i(\omega) \, Y_i(\omega). \tag{1.5}$$

For $d = 1$ we are getting scalar case and the inner product in $L^2(\Omega, \mathbb{R})$ is

$$[X, Y] = \int_\Omega X(\omega) \cdot Y(\omega) \, dP(\omega).$$

The reader should distinguish from the context whenever symbols have to be interpreted as being scalars or vectors, for example, 0 as belonging to $\mathbb{R}$ or to $\mathbb{R}^d$. Also, by abusing the notation a bit, we will use $\| \ \|^2$ for the square of the norm for the two different inner products, namely $\|X\|^2 = [X, X]$ and $\|a\|^2 = \langle a, a \rangle$.

Also we will use the following notation for the characteristic functions:

$$\mathbf{1}_A(\omega) = \begin{cases} 1 & if \ \omega \in A \\ 0 & otherwise. \end{cases}$$

This definition leads immediately to two main properties of characteristic functions:

$$\mathbf{1}_A \cdot \mathbf{1}_A = \mathbf{1}_A,$$

$$\mathbf{1}_A \cdot \mathbf{1}_B = 0, \quad \text{if} \quad A \cap B = \emptyset.$$

The d-dimensional unit sphere is denoted by S^d and defined by

$$S^d = \left\{ x = (x_1, \ldots, x_d) \in \mathbb{R}^d \ : \quad \|x\|^2 = \sum_{i=1}^{d} x_i^2 = 1 \right\}.$$

For $d = 2$ we obtain S^2, that is the unit circle on the plane of radius one with center at the origin.

Chapter 2

Haar systems and Vector Greedy Splitting Algorithm

This chapter describes how to build the adaptive tree of partitions, we define an associated H-system and how the orthonormal system of vector Haar functions is created. This process may be considered as the *analysis* of the input vector of signal.

2.1 H-Systems and Tree of Partitions

An H-system is an orthonormal basis (Haar-like) $\{\psi_k\}_{k\geq 0}$ constructed from a binary tree of partitions for our space Ω. Define $A_{0,0} = \Omega$, let the first partition be the trivial one $\mathcal{Q}_0 = \{A_{0,0}\}$, its physical meaning is that it provides the mean values, or DC levels of the signals. Now we split our space $\mathcal{Q}_0$ (or trivial partition $\mathcal{Q}_0$) into two disjoint sets $\mathcal{Q}_1 = \{A_{1,0}, A_{1,1}\}$ such that $A_{1,0} \bigcup A_{1,1} = A_{0,0}$. At this stage our tree consists of one node (root $A_{0,0}$) and its two children (leaves) $A_{1,0}$ and $A_{1,1}$. At the next step, we choose one of two atoms of $\mathcal{Q}_1$ to split. If we choose to split $A_{1,0}$, then it will become father (node) and its 2 children will be $A_{2,0}$ and $A_{2,1}$ and this partition will be $\mathcal{Q}_1 = \{A_{1,1}, A_{2,0}, A_{2,1}\}$, however, if we choose to split $A_{1,1}$, then it will become a node and its two leaves will be $A_{2,2}$ and $A_{2,3}$ and then the new partition will be $\mathcal{Q}_1 = \{A_{1,0}, A_{2,2}, A_{2,3}\}$. This last case is showed in the picture:

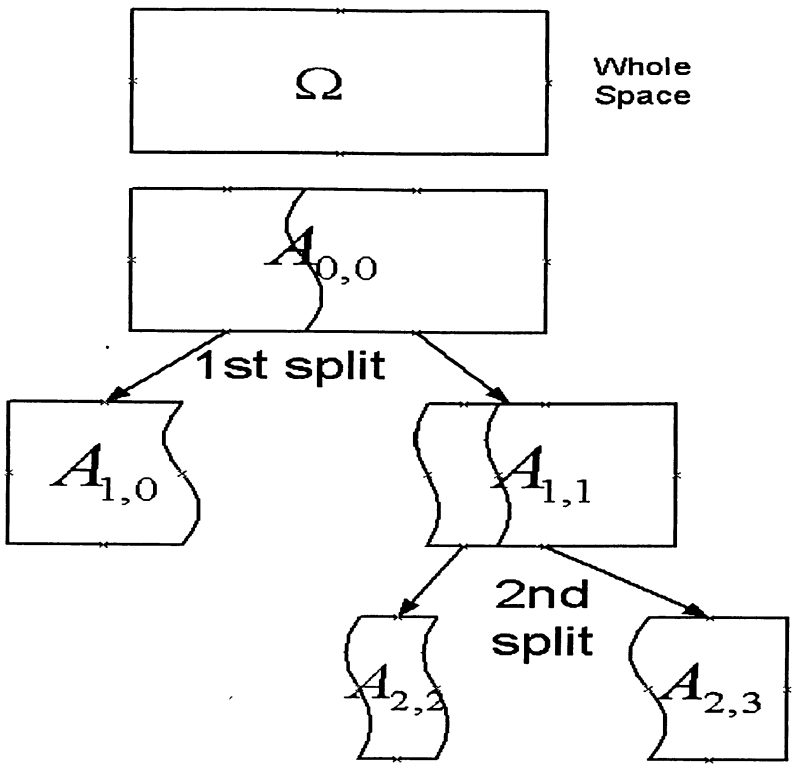

Figure 2.1: First two partitions

In general, the next partition $\mathcal{Q}_{n+1}$ is constructed from $\mathcal{Q}_n$ by splitting one of the atoms in $\mathcal{Q}_n$ in two pieces and leaving the other ones intact. This kind of sequence of partitions will be called *binary sequence of partitions.* According to our notation if an atom $A \in \mathcal{Q}_n$ then $A = A_{j,k}$, the parameter j indicates the depth or level in the tree, it is analogous to the *scale* parameter of the classical wavelets (in this case the scale is related to the size of the support of the wavelet). Assume that $A = A_{j,k} \in \mathcal{Q}_n$ is the atom chosen to be split to create $\mathcal{Q}_{n+1}$, we then have two new atoms $A_{j+1,2k}$ and $A_{j+1,2k+1}$ (of course $A_{j,k} = A_{j+1,2k} \cup A_{j+1,2k+1}$ and $\emptyset = A_{j+1,2k} \cap A_{j+1,2k+1}$). Obviously $\mathcal{Q}_{n+1} = \mathcal{Q}_n \backslash \{A_{j,k}\} \cup \{A_{j+1,2k}, A_{j+1,2k+1}\}$. Now the question is how do we choose the next leaf to split? The H-system of basis functions provides us with the arithmetical tool to make this decision. Every time we perform a split we construct the basis Haar function $\psi_{i,j}$ corresponding to the leaves of $A_{i,j}$. The inner product $\lambda_{i,j} = [X, \psi_{i,j}]$ gives us the information about how much of the signal energy is carried by $[X, \psi_{i,j}] \cdot \psi_{i,j}$. At every step we choose to split the leaf with highest value of inner product allowing for the most effective *analysis* of the input signal. This makes our algorithm *adaptive* so we call it the Adaptive VGS Algorithm. Chapter 3 will explain mathematical properties of the basis Haar functions. In the remaining of the present chapter we continue with the general description of the next stages of the algorithm, Multiresolution Analysis (MRA), compression and synthesis.

2.2 Multiresolution Analysis

Suppose we have built a tree of M atoms belonging to a *binary partition.* Therefore we have a set of M nodes with corresponding basis Haar functions ψ_i and inner products λ_i. In the next step we go through all $M+1$ leaves of our tree (atoms of $\mathcal{Q}_M$) and calculate mean values of each scalar component (i.e. of each input signal). Let's denote, for simplicity, the $M+1$ atoms of our dyadic partition as $\mathcal{Q}_M = \{A_0, A_1, \ldots, A_M\}$. In more explicit terms, the computations referred to above entail to compute the $d \cdot (M+1)$ quantities (d is the dimension of the input vector)

$$y_{A_i} = \frac{1}{P(A_i)} \int_{A_i} X_k(w) dP(w), \quad \text{where } i \in \{0, 1, \ldots, M\},\ k \in \{1, 2, \ldots, d\}. \tag{2.1}$$

Using these y_{A_i} values and probabilities $P(A_i)$ we have recursive formulae allowing us to perform a bottom up recursion (i.e. staring from the leaves) calculating scalar inner products (denoted d_{A_i}) for each input component signal at each node of the partition tree. This process completes MRA. The following Figure 2.2 illustrates the algorithm of recursive multiresolution analysis:

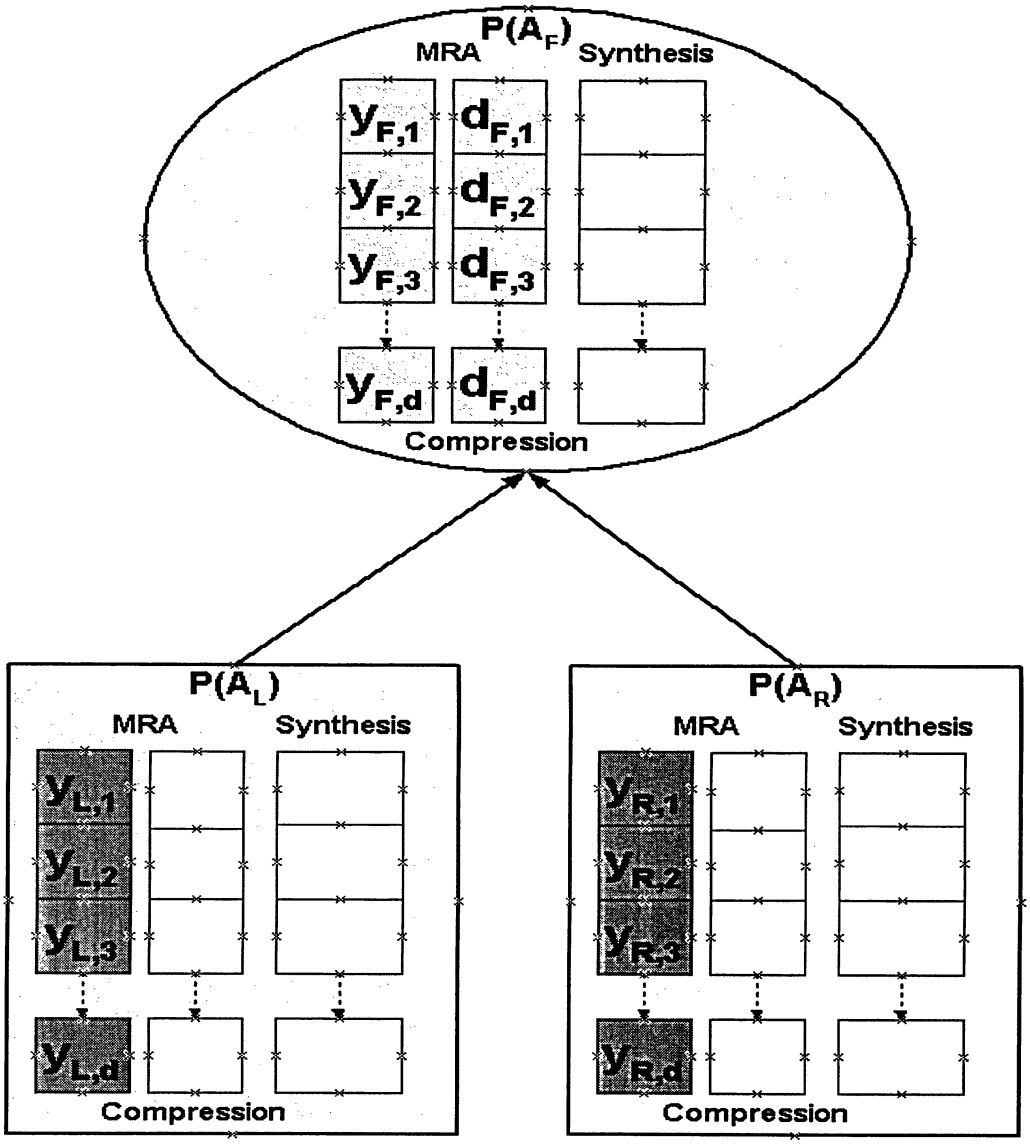

Figure 2.2: Recursive Multiresolution Analysis.

We see in the Figure 2.2 one node and its two children, the left one and the right one. We use the subscripts F, L and R for the parent (father) node, its left and right children correspondingly. The data structure for MRA consists of probability $P(A_i)$ and d-dimensional vector of y_{A_i} values of the node A_i. The recursion starts from leaves, and in our case the left child is assosiated with set A_L, its probability $P(A_L)$ and d-dimensional vector $\{y_{L,1}, \ldots, y_{L,d}\}$. On the same way the data structure for the right leaf contains probability $P(A_R)$ and d-dimensional vector $\{y_{R,1}, \ldots, y_{R,d}\}$.

The formula for the recursive MRA is

$$\begin{cases} p_F = p_L + p_R \\ y_F = \dfrac{1}{p_F}(p_L\, y_L + p_R\, y_R) \\ d_F = \sqrt{\dfrac{p_L\, p_R}{p_F}}\,(y_L - y_R) \end{cases}.$$

Using this formula we calculate vectors $\{y_{F,1}, \ldots, y_{F,d}\}$ and $\{d_{F,1}, \ldots, y_{F,d}\}$ for the parent node of our two children. The arrows on the lines connecting leaves and father node in the Figure 2.2 emphasize the fact that MRA recursion is implemented from the bottom to the top of the partition tree, starting from the leaves and ending in the root. The values $\{y_{A_i,1}, \ldots, y_{A_i,d}\}$ needed for MRA implementation, while the values $\{d_{A_i,1}, \ldots, d_{A_i,d}\}$ are necessary for the compression and the synthesis as described in the Chapters 2.3 and 2.4.

The mathematical justification for MRA are provided in Section 3.3 and the formal proof of recursive MRA formulae presented in Appendix B.3.

2.3 Compression

Once the multiresolution analysis algorithm terminates, we have available $d \cdot M$ scalar inner products one for for each node of the partition tree. The compression stage consists of setting to zero (in according to required compression ratio) the smallest scalar inner products. Suppose that compression ratio is 75%, in this case only a quarter of the inner products (corresponding to the largest values) will remain for each input signal, the smallest ones will be set to zero. These sets of inner products will, of course, be signal dependant. More specifically, for the same node A_i, there are d scalar inner products, and some of them may be set to zero and some not during compression. In practical terms, this means that for each input signal the remaining inner products carry most of the energy of the signal. We illustrate the compression on the following picture:

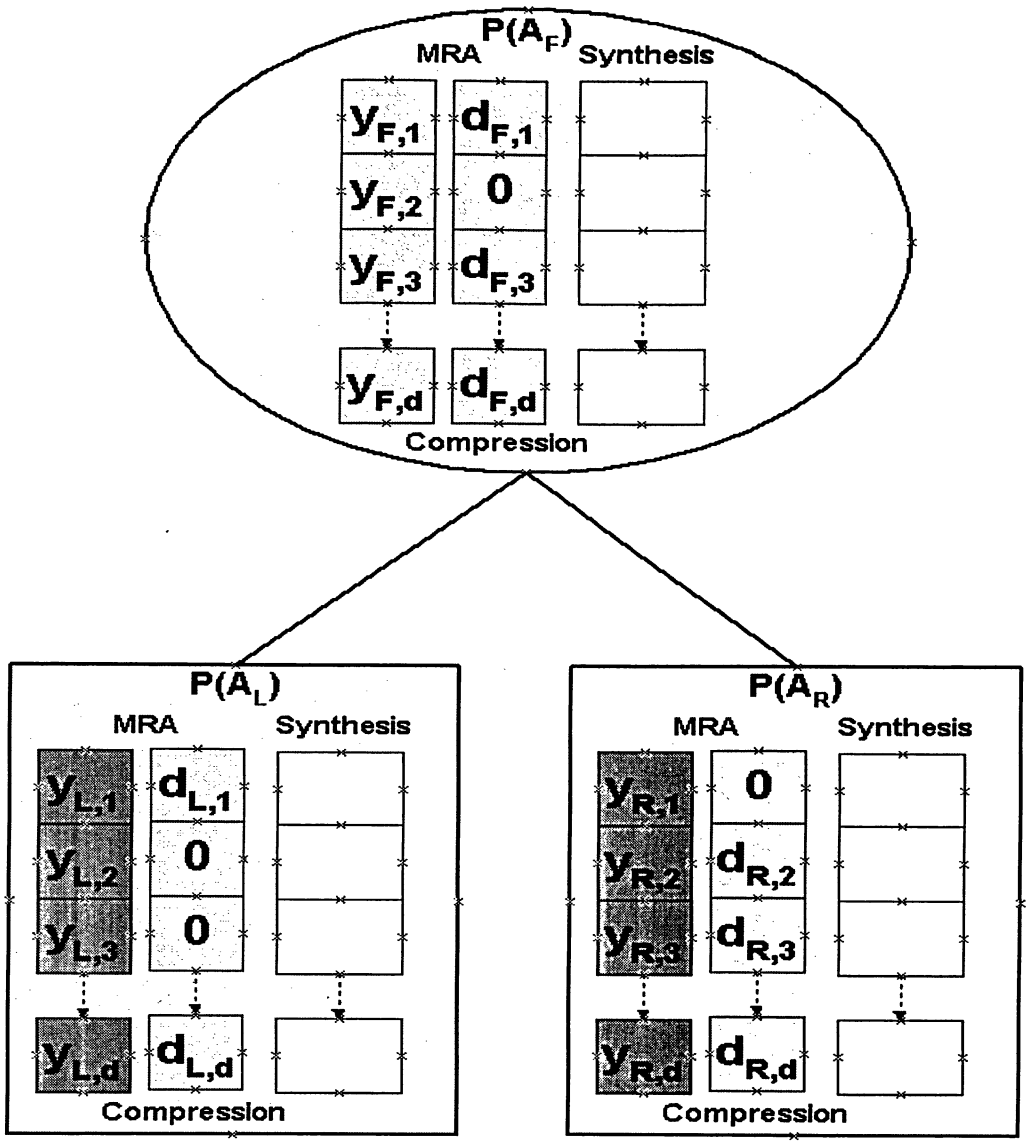

Figure 2.3: Compression stage of VGS algorithm.

We can see on the picture above father node and its two children, similarly to the Figure 2.2. However we observe that some of the inner products values $d_{A_i,j}$ are set to zero as a result of compression. For example, the values $d_{A_F,2}$ of the father node, $d_{A_L,2}$ and $d_{A_L,3}$ of the left child and also $d_{A_R,1}$ of the right child were all compressed.

It is interesting to note that the sum of squares of all inner products d_{k,A_i}, for the same node A_i, equals λ_i^2, which is calculated during construction of the partition tree. This result is one of the contributions of the thesis and it is proved in Appendix B.2.1.

2.4 Synthesis

The last step, after analysis and compression, is called synthesis and consists of computing the final approximation to the input vector of random variables. Similarly to the MRA there are recursive formulae, which provide the approximation to the input signals. This recursion starts at the root of the tree and goes top-down through all the leaves. This process is shown in Figure 2.4:

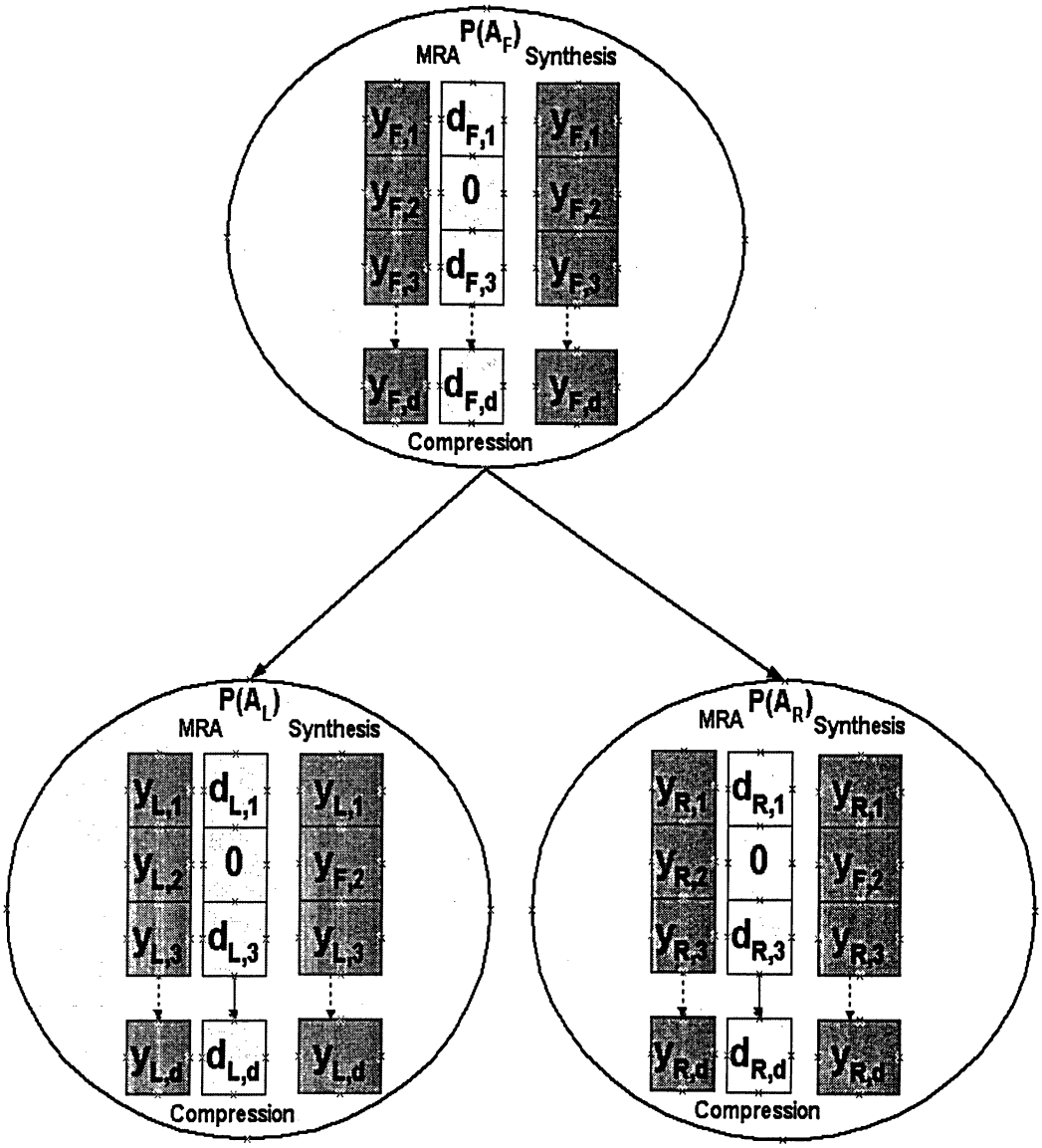

Figure 2.4: Recursive Multiresolution Synthesis.

We see in the Figure 2.4 parent node and its two children that are nodes as well. We use the same notation as in Figure 2.4 and its description in the Chapter 2.2. We use d-dimensional vector $\{y_{F,1}, \ldots, y_{F,d}\}$ and compressed vector of inner products $\{d_{F,1}, \ldots, d_{F,d}\}$ to implement recursive synthesis in according to the following

recursive formula:

$$\begin{cases} y_L = y_F + \sqrt{\dfrac{p_R}{p_F p_L}} d_F \,, \\ \\ y_R = y_L - \sqrt{\dfrac{p_F}{p_L p_R}} d_F \,. \end{cases}$$

It is important to note that the vector $\{y_{F,1}, \ldots, y_{F,d}\}$ used in the synthesis is not the same as the one received during MRA and shown in Figure 2.2. The actual values $y_{F,j}$ used in the formula above are received as an output of the recursive synthesis and they are stored in the synthesis part of the node data structure as has been illustrated in the Figure 2.4. The recursive synthesis calculates the vectors $\{y_{L,1}, \ldots, y_{L,d}\}$ and $\{y_{R,1}, \ldots, y_{R,d}\}$ for the left and right children of the father node. The initialization of the synthesis is actually copying of the vector $\{y_{Root,1}, \ldots, y_{Root,d}\}$ from the MRA data structure into synthesis data structure of the root of partition tree.

The section 3.3 contains additional description of the synthesis and formal proof of recursive formula is done in Appendix B.4.

Chapter 3

VGS implementation: Analysis, Compression and Synthesis

While the previous Chapter was an overview of all the steps performed by the Adaptive Vector Greedy Algorithm, this chapter contains formal definitions and mathematical formulae needed for the implementation of the VGS algorithm.

3.1 H-functions

Consider a probability space $(\Omega, \mathcal{A}, P)$ and the associated Hilbert space $L^2(\Omega, \mathbb{R}^d)$, in the case when Ω is finite, the collection $\mathcal{A}$ will then be the collection of all subsets of Ω.

Definition 1. *Given $A \in \mathcal{A}$, $P(A) > 0$, a function ψ_A is called a (vector valued) Haar function on A if there exist*
$A_i \in \mathcal{A}$, $i = 0, 1$, $A_i \subseteq A$, $A_0 \cap A_1 = \emptyset$, $A = A_0 \cup A_1$, and
$\psi_A = a\ \mathbf{1}_{A_0} + b\ \mathbf{1}_{A_1}$, $a, b \in \mathbb{R}^d$ satisfying the following two conditions:

$$\int_\Omega \psi_A(\omega)\ dP(\omega) = 0, \tag{3.1}$$

$$\|\psi_A\|^2 \equiv [\psi_A, \psi_A] \equiv \int_\Omega \|\psi_A(\omega)\|^2\ dP(\omega) = 1, \tag{3.2}$$

where ψ_A is norm in Hilbert $L^2(\Omega, \mathbb{R}^d)$ space and $\|\psi_A(\omega)\|$ is the Euclidean norm in $\mathbb{R}^d$.

The above equations can be used to relate a and b and also to set a constraint

on b (its norm is fixed), actually (3.1) gives

$$a = \frac{-\, b\, P(A_1)}{P(A_0)}. \tag{3.3}$$

Using this last equation in (3.2) gives

$$\|b\|^2 = \frac{P(A_0)}{P(A_1)\; P(A)}. \tag{3.4}$$

Also, if $b' \equiv b/\|b\|$ we get point b' on unit sphere S^d defined by direction of vector b. Finally, we can relate $b' \in S^d$ to b which is used to define the actual Haar function:

$$b = \pm\, b' \sqrt{\frac{P(A_0)}{P(A_1)\; P(A)}}. \tag{3.5}$$

The results presented above are equivalent to the following general form of vector valued Haar function:

$$\pm\, \psi_{A,b'} = \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\; \mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\; \mathbf{1}_{A_1} \right) b', \text{ where } b' \in S^d\,. \tag{3.6}$$

In particular, for $d = 1$ we obtain the one-dimensional case with $b' = \pm 1$ and, therefore, the scalar Haar function can be written as follows

$$\pm\, \psi_A = \sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\; \mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\; \mathbf{1}_{A_1}\,. \tag{3.7}$$

From equation (3.6) it is clear that for any disjoint sets A_i and A_j the corresponding Haar functions ψ_{A_i} and ψ_{A_j} are orthogonal, i.e. $[\psi_{A_i}, \psi_{A_j}] = 0$, moreover, they are orthonormal.

3.2 Construction of tree of partitions for VGS

Having as input the probability space $(\Omega, \mathcal{A}, P)$, vector random variable $X = \{X_1, \ldots, X_d\}$, belonging to the Hilbert space $L^2(\Omega, \mathbb{R}^d)$ and number of iterations M we approximate X using an adaptive set of functions consisting of the initial function ψ_0 and M basis vector Haar functions $\{\psi_0, \psi_1, \ldots, \psi_M\}$. Let us denote with $\hat{X}$ the resulting approximation to X. Therefore

$$\hat{X} = \sum_{k=0}^{M} \lambda_k \psi_k, \qquad \text{where } \lambda_k = [X, \psi_k]\,. \tag{3.8}$$

The goal is to build this basis of Haar functions adaptively, in such way, that the error norm $||X - \hat{X}||$ is minimal after M iterations. We approach this optimization via a greedy algorithm where we minimize this error norm *at each iteration.* Here, with the word *iteration*, we mean each splitting of an atom used to generate the tree of partitions.

This greedy optimization problem is equivalent to maximizing absolute value (modulus) of $[X, \psi]$. This follows from the fact that

$$\begin{aligned} \left|\left|X - \psi[X,\psi]\right|\right|^2 &= [X - \psi[X,\psi], X - \psi[X,\psi]] \\ &= ||X||^2 - [X, \psi[X,\psi]] - [\psi[X,\psi], X] + \left|\left|\psi[X,\psi]\right|\right|^2 \\ &= ||X||^2 - [X,\psi]^2 - [X,\psi] \cdot [\psi, X] + \left|\left|\psi[X,\psi]\right|\right|^2 \qquad (3.9) \\ &= ||X||^2 - [X,\psi]^2 - [X,\psi]^2 + ||\psi||^2 \cdot [X,\psi]^2 \\ &= ||X||^2 - [X,\psi]^2 \,. \end{aligned}$$

Here we used the fact that $||\psi||^2 = 1$.

The VGS algorithm constructs a binary tree by splitting subsets of Ω resulting on new partition after each of the M iterations. As a result of the algorithm we get a final partition $\mathcal{Q}_M$, a corresponding binary tree structure, an orthonormal basis $\{\psi_k\}_{k=0}^{k=M}$ and a collection of inner products $\{[X,\psi_k]\}_{k=0}^{k=M}$.

The first step of the VGS algorithm calculates ψ_0, the constant function over Ω. Each one of the next M steps will perform a split of one chosen leaf of our binary tree. The procedure that performs this split will be called `bestSplit`. After M iterations we get the final partition and tree.

The following are the steps performed by the VGS algorithm to build an adaptive tree of partitions:

- First, a simple step to construct the basis function ψ_0.
- Procedure to obtain the next partition (`bestSplit`).
- How to choose next leaf to split.

The partition tree is shown in the picture:

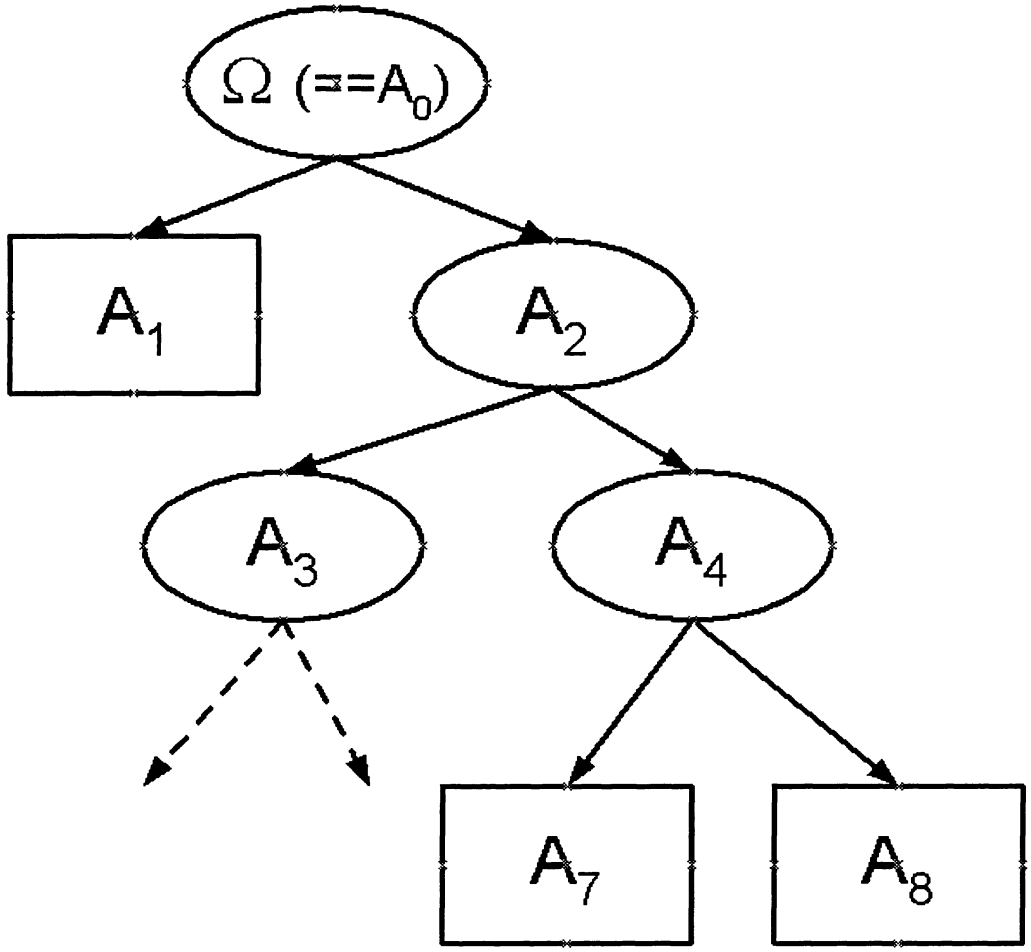

Figure 3.1: General layout of partition tree.

On the Figure 3.1 above we see partition tree that has three nodes (A_0, A_2 and A_4) and three leaves A_1, A_7 and A_8). The VGS alogorithm is performing split of the forth leaf (A_3), and in the end of its splitting it will become node and its two children will be leaves (A_9 and A_{10}).

3.2.1 First Basis Function ψ_0

In order to construct the first basis function ψ_0 we define the trivial partition $\mathcal{Q}_0$ as $\mathcal{Q}_0 = \{\Omega, \emptyset\}$. We look for a Haar function as in Definition 1. We are getting immediately that $\psi_0 = c\ \mathbf{1}_\Omega + d\ \mathbf{1}_\emptyset = c\ \mathbf{1}_\Omega$, i.e. ψ_0 is defined by one constant vector in $\mathbb{R}^d$ space. This basis function ψ_0 defines first approximation of X in the

form $X_{\mathcal{Q}_0} = [X, \psi_0]\psi_0$. The best approximation corresponds to the minimum of the remainder $||X - [X, \psi_0]\psi_0||$. But the problem of minimizing $||X - X_{\mathcal{Q}_0}||$ is equivalent to maximizing the inner product $\left|[X, \psi_0]\right|$. This is shown in formula (3.9). So we are interested in finding such ψ_0 that maximizes the inner product $\left|[X, \psi_0]\right|$.

We may use the method of Lagrange multipliers in order to solve this extremum problem. In Section B.1 of Appendix B there is a detailed description of the approach that results in the following equations for the first basis ψ_0, corresponding inner product $[X, \psi_0]$ and approximation $[X, \psi_0]\psi_0$. These results take the following form:

$$\begin{aligned}
&\psi_0(\omega) \equiv c = \{c_1, \ldots, c_d\} \quad \text{where} \quad c_i = \frac{\int_\Omega X_i(\omega) dP(\omega)}{\sqrt{\sum_{k=1}^{d} \left(\int_\Omega X_k(\omega) dP(\omega) \right)^2}}, \\
&\lambda_0 = [X, \psi_0] = \sqrt{\sum_{k=1}^{d} \left(\int_\Omega X_k(\omega) dP(\omega) \right)^2}, \\
&\{[X, \psi_0]\psi_0\}_i = \{X_{\mathcal{Q}_0}\}_i = \int_\Omega X_i(\omega) dP(\omega) .
\end{aligned} \tag{3.10}$$

For the scalar case ($d = 1$) the equations (3.10) have been simplified to

$$\begin{aligned}
&\psi_0(\omega) \equiv 1, \\
&\lambda_0 = [X, \psi_0] = \int_\Omega X(\omega) dP(\omega) , \\
&X_{\mathcal{Q}_0} = [X, \psi_0]\psi_0 = \int_\Omega X(\omega) dP(\omega) .
\end{aligned} \tag{3.11}$$

As we see from equation (3.11), inner product $[X, \psi_0]$ is the average, or DC level of the input scalar signal over the given domain.

3.2.2 Best Split Algorithm

Here we describe the procedure `bestSplit` that finds the best partition of a set A into two disjoint sets A_0 and A_1. Proofs of all formulae are presented in Appendix B.2.

The Best Split algorithm finds a vector Haar function $\psi_{A,b'}$ from the formula (3.6), that maximizes absolute value of the inner product

$$\lambda_A = [X, \psi_{b',A}] = \int_\Omega \langle\, X(\omega), \psi_{b',A}(\omega) \,\rangle\, dP(\omega)\,.$$

From the above formula and using expression (3.6), we obtain the following general form of the inner product:

$$\pm\,[X, \psi_{b',A}] = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}}\left(\frac{1}{P(A)}\int_A \langle X, b'\rangle\, dP - \frac{1}{P(A_0)}\int_{A_0}\langle X, b'\rangle dP\right). \quad (3.12)$$

There are two parameters in the equation above, partition $\{A_0, A_1\}$ and d-dimensional vector $b' \in S^d$. If we fix one parameter, either partition or unit vector, we may solve the optimization problem, namely the maximization of λ_A.

The first approach is to fix the partition $\{A_0, A_1\}$. In this case, as shown in Appendix B.2, the optimization problem may be solved using the method of Lagrange multipliers and the d-dimensional parameter is expressed by the following formula:

$$b'_i = \frac{\frac{1}{P(A)}\int_A X_i dP - \frac{1}{P(A_0)}\int_{A_0} X_i dP}{\sqrt{\sum_{k=1}^{d}\left(\frac{1}{P(A)}\int_A X_k dP - \frac{1}{P(A_0)}\int_{A_0} X_k dP\right)^2}}\,. \quad (3.13)$$

The second approach is to fix the unit vector b' on d-dimensional sphere S^d and to find best partition for parameter b'. It may be done easily through the use of the *Bathtub principle*, its discrete version is proved in Appendix section B.2.2. The Bathtub principle provides us with an effective tool to solve the optimization problem for finding the extremum of inner product for a scalar random variable. Suppose, we fix $b' \in S^d$, we then define the following scalar signal

$$X[b'](\omega) \equiv \langle X(\omega), b'\rangle\,.$$

In this case the formula (3.14) for inner product may be presented as:

$$\pm\,[X, \psi_{b',A}] = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}}\left(\frac{1}{P(A)}\int_A X[b']dP - \frac{1}{P(A_0)}\int_{A_0} X[b']dP\right). \quad (3.14)$$

Denote the set of range of values of $X[b'](\omega)$ (for $\omega \in A$) by $\mathcal{R}_A(X[b'])$ and sort them in increasing order, namely

$$\mathcal{R}_A(X[b']) = \{y_0, \ldots, y_K\} \;\text{ with }\; y_k < y_{k+1},\;\; k = 0, \ldots, K-1,$$

and for each y_k there exists $\omega \in A$ such that $y_k = \langle X(\omega), b' \rangle$. In short, $\mathcal{R}_A(X[b'])$ consists of the values $\langle X(\omega), b' \rangle$, $\omega \in A$, sorted in increasing order. Let's denote

$$R_k = \{y_0, \ldots, y_k\}, \quad k = 0, 1, \ldots, K-1 \, .$$

The bathtub principle is used to provide a solution for the extremum of the inner product $[X, \psi_{b',A}]$, providing a solution given by partition $\{A_0, A_1\}$, where $A_0 = X^{-1}[b'](R_k)$. Since, there are K sets R_k, we need K iterations to find best split for set A. This best split will be the one giving maximum absolute value of inner product $[X, \psi_{b',A}]$. After a best partition, corresponding to a given b', is found, we may calculate the inner product $\lambda_{A,b'}$. Iterating this step, we can choose another unit vector $b' \in S^d$ and repeat the procedure. The parameter b' corresponding to the largest value of $|\lambda_{A,b'}|$ will be the solution of the optimization problem. The found partition $\{A_0, A_1\}$ will be the best split for set A and will form two new leaves of the partition tree. We have chosen the second approach, namely, finding maximum for $[X, \psi_{b',A}]$ through sampling points in the unit sphere S^d and then using the bathtub principle to find the best split. In order to realize this approach for the VGS algorithm we have to be able to sample points, uniformly distributed, on the d-dimensional sphere. This algorithm is described thoroughly in Appendix C. Having obtained an algorithm for `bestSplit` step, the natural question is: what leaf should be chosen to split next?

3.2.3 Next Leaf to Split

Suppose that we have performed M `bestSplit` iterations on a given space Ω. The first iteration just splits the root of the tree Ω. For each one of the following $M-1$ steps we have to make a decision about which leaf of the tree to split next. Suppose that we have completed K `bestSplit` iterations. In particular, this means that there are K nodes and $K+1$ leaves of the tree, each node is defined by a subset $A_i \in \mathcal{A}$ of our input probability space $(\Omega, \mathcal{A}, P)$. Also, for each node A_i, we have a vector Haar function ψ_i and corresponding inner product $\lambda_i = [X, \psi_i]$. The criteria used, in order to select the next node to split, is to choose the node with largest absolute value of its inner product. From a physical point of view it means that this leaf carries most of the energy of the signal and it is optimal to choose it for the next iteration. Notice that the algorithm is greedy as it optimizes choices based only on information available at each stage. Moreover, the algorithm is adaptive as the optimality criteria used is dependent on the input vector (of signals).

3.3 Implementation of Multiresolution Analysis, Compression and Synthesis

After completing the construction of the tree of partitions, the next stage is to calculate the scalar inner products for each node of the tree. At each node, we have as many inner products as the number of input signals, namely d. As we see in formula (3.7), scalar inner products depend in partition only. There is an important relationship between the inner product λ_A, corresponding to the input vector, and the inner products λ_i corresponding to each component of the vector X:

$$\lambda^2 = \sum_{i=1}^{d} \lambda_i^2, \qquad \text{i.e.} \qquad \Big| [X, \psi_{b'}] \Big|^2 = \sum_{i=1}^{d} \Big| [X_i, \psi_A] \Big|^2 . \tag{3.15}$$

From formula (3.15), it follows that the decay of the sequence of values $\{|\lambda_1|, |\lambda_2|, \ldots, |\lambda_d|\}$, created by the VGS algorithm applied to the input vector $X = \{X_1, \ldots, X_d\}$, causes a corresponding decay of the sequence of values $\{|\lambda_{i,1}|, |\lambda_{i,2}|, \ldots, |\lambda_{i,d}|\}$, obtained by computing the inner products of each signal X_i. We emphasize that these last computations use the same partition tree. There are recursive formulae for the MRA, this algorithm starts from the leaves (down-top recursion). We perform the same task for each component signal X_i of the input vector $X = \{X_1, \ldots, X_d\}$. Suppose we are dealing with signal X_i and for each node A of the partition tree we use the following notation:

$$\begin{cases} p_{nodeA} = P(A) \\ y_{nodeA} = \frac{1}{P(A)} \int_A X_i(w) dP(w) \\ d_{nodeA} = \lambda_{i,nodeA} = \langle X_i, \psi_{i,nodeA} \rangle \end{cases}$$

For each leaf of the tree we know quantities p_{nodeA} and y_{nodeA}. The MRA performs the calculation of all inner products d_{nodeA} for all nodes of the tree per each signal component X_i. We will use subscripts F, L and R for father node and its left and right children. Suppose that we know p_L, p_R, y_L and y_R of children, then, the recursive formulae to calculate p_F, y_F and d_F are given by:

$$\begin{cases} p_F = p_L + p_R \\ y_F = \dfrac{1}{p_F}(p_L\, y_L + p_R\, y_R) \\ d_F = \sqrt{\dfrac{p_L\, p_R}{p_F}}\, (y_L - y_R) \end{cases} \tag{3.16}$$

We prove these formulae in APPENDIX B.3. The Figure 2.2 in Chapter 2.2 illustrate the algorithm of MRA.

The compression stage consists on setting to zero (in accordance to required compression ratio) the smallest scalar inner products d_{nodeA}. We perform compression independently for each component of vector input of signals. We shown the process in Figure 2.3 in Chapter 2.3.

The last step, after analysis and compression, is called synthesis and consists of computing the final approximation to the input vector of random variables. Similarly to the MRA there are recursive formulae, allowing for the approximation to the input signals. This recursion starts at the root of the tree and goes top-down through all the leaves. Here are recursive formulae for synthesis, using the same notation as in formula (3.16):

$$\begin{cases} y_L = y_F + \sqrt{\dfrac{p_R}{p_F\, p_L}}\, d_F \\ \\ y_R = y_L - \sqrt{\dfrac{p_F}{p_L\, p_R}}\, d_F \end{cases} . \tag{3.17}$$

The initial conditions for this recursion are y_F values for the root of the tree, corresponding to the space Ω, namely $y_{\Omega,i} = \int_\Omega X_i(w)\ d\ P(w)$, where X_i are components of the input vector of signals. We prove recursive formulae (3.17) in APPENDIX B.4 and the algorithm is illustrated in Figure 2.4 in Chapter 2.4.

Chapter 4

Performance Comparison of Vector Greedy Splitting Algorithm vs Haar Transform

We present some results and illustrations for the adaptive VGS algorithm. The implementation was done using software written in ANSI C programming language and MATLAB as a presentation tool (to gather the statistics and plotting). The software implements the following main tasks:

- Implementation of the adaptive VGS algorithm.
- Implementation of the classical Haar algorithm.
- Benchmark analysis and comparison of the VGS vs Haar wavelets.
- Accumulating of the statistics and plotting results.

During implementation of the VGS algorithm the emphasis was done on the efficiency of the calculations. For example, the 16-bit integers were used to store input samples and all corresponding buffers. The multiplication of the integers was done in fractional mode allowing storage of the result in 16-bit field as well. This technique is widely used in the commercial hi-tech DSP projects.

This chapter indicates how the VGS algorithm works for the scalar case (input vector is a single signal) and the vector case (up to 4 input signals). Also, we run the algorithm for five different input vectors, namely input vectors 1, 2, 3, 4 and 5. We compare adaptive VGS algorithm with classical Haar wavelet applied to the same input vectors. The following section presents these inputs.

4.1 Inputs for the VGS and Haar wavelet algorithm implementation

This section provides with illustration of the input vectors for the VGS implementation. The same inputs are used for both, adaptive VGS and Haar algorithms. We have built five different input vectors, enumerated from 1 to 5. Each of the input vectors may contain up to 4 signal components, namely signals 1 to 4. In the case when the dimension of the input vector is less than 4 it means that not all signals are used. For example if the dimension is 3, it means that first 3 signal components of the input vectors illustrated in the Figures 4.1 - 4.5 were input to the software. We present all input vectors in the following five figures, the first one illustrates input vector 1:

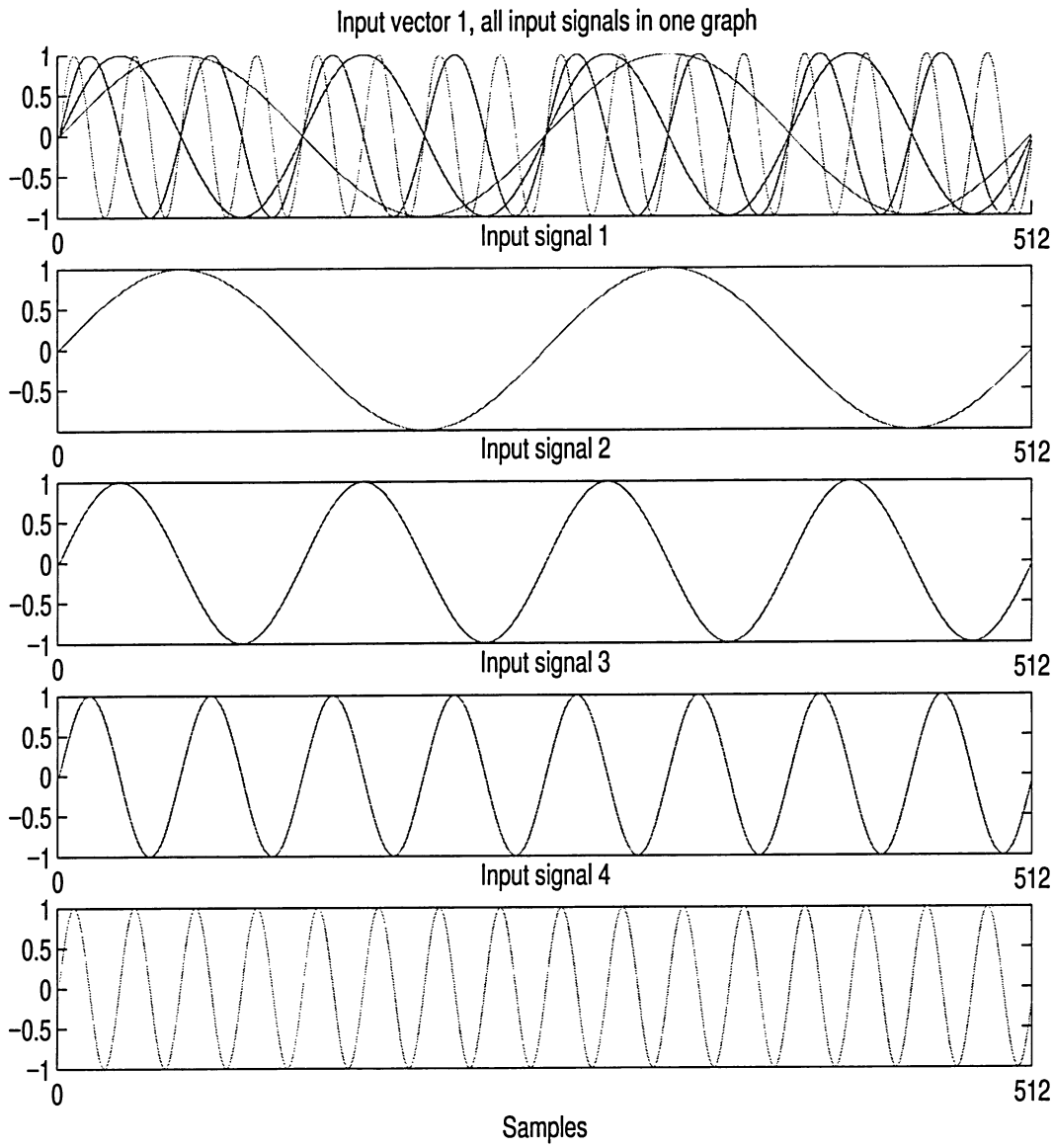

Figure 4.1: Input vector 1.

The picture above presents input vector 1. We see five subplots, the upper one presents all input signals of vector 1 in one graph, the 4 bottom subplots illustrate each of the input signals separately.

The components of the input vector 1 are signals described by the formula $X_i(t) = \sin(2\pi f_i t)$. In another words input vector 1 contains tones of the different frequencies with the same amplitude. All signals have 512 samples and their values are normalized to be in the segment $[-1, 1]$. The next figure shows input vector 2:

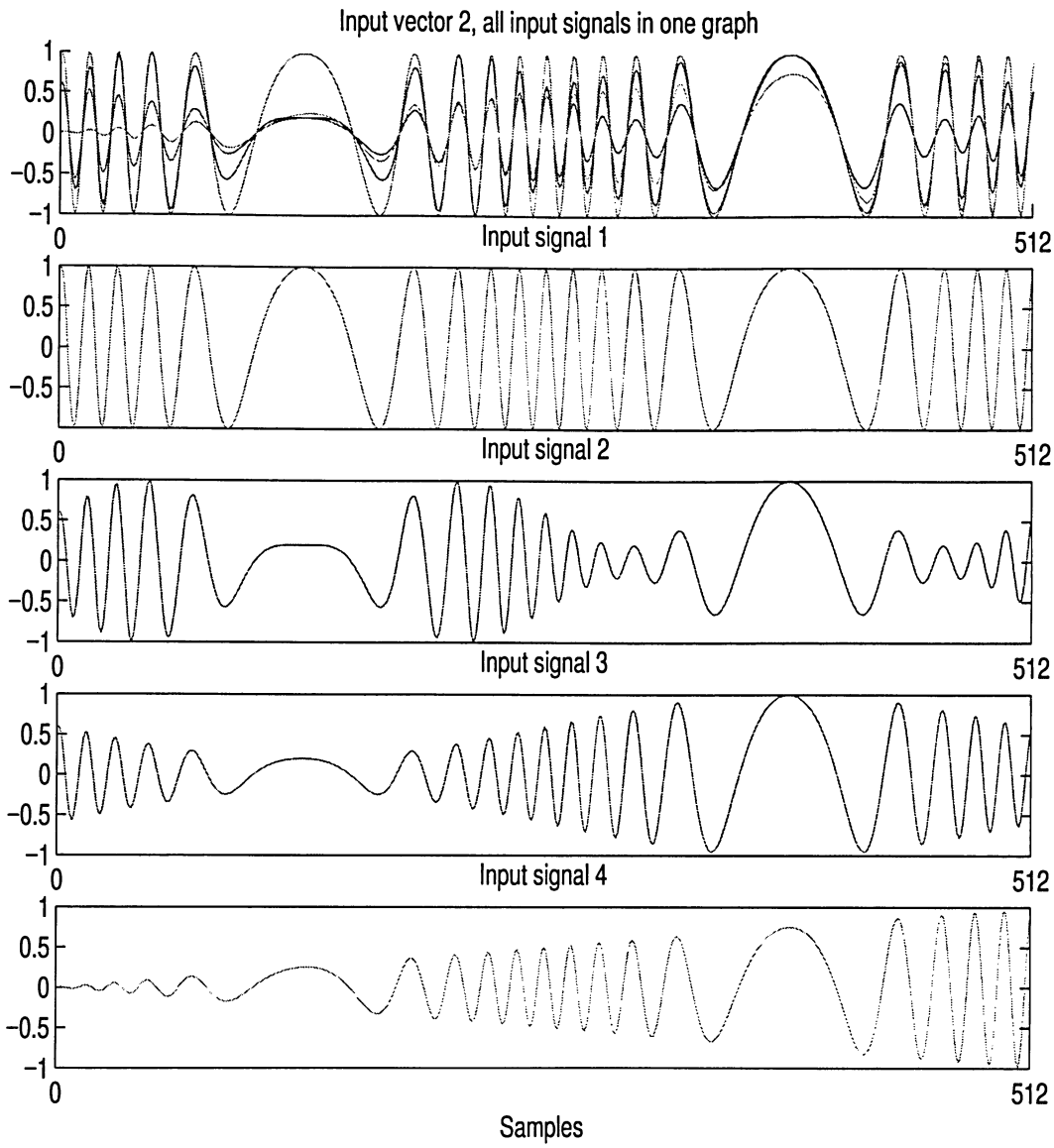

Figure 4.2: Input vector 2.

Input vector 2 is shown in the picture above. As in Figure 4.1 there are five subplots, the upper one presents all input signals in one graph, the 4 bottom subplots illustrate each of the input signals separately, all inputs are normalized and have 512 samples. The inputs are described by the formula $X_i(t) = A_i \sin(2\pi f_1 t) + B_i \cos(2\pi f_2 t)$. As we see all signals of the input vector 2 are in-phase, i.e. they oscillate with the same phase characteristic. The next figure shows input vector 3:

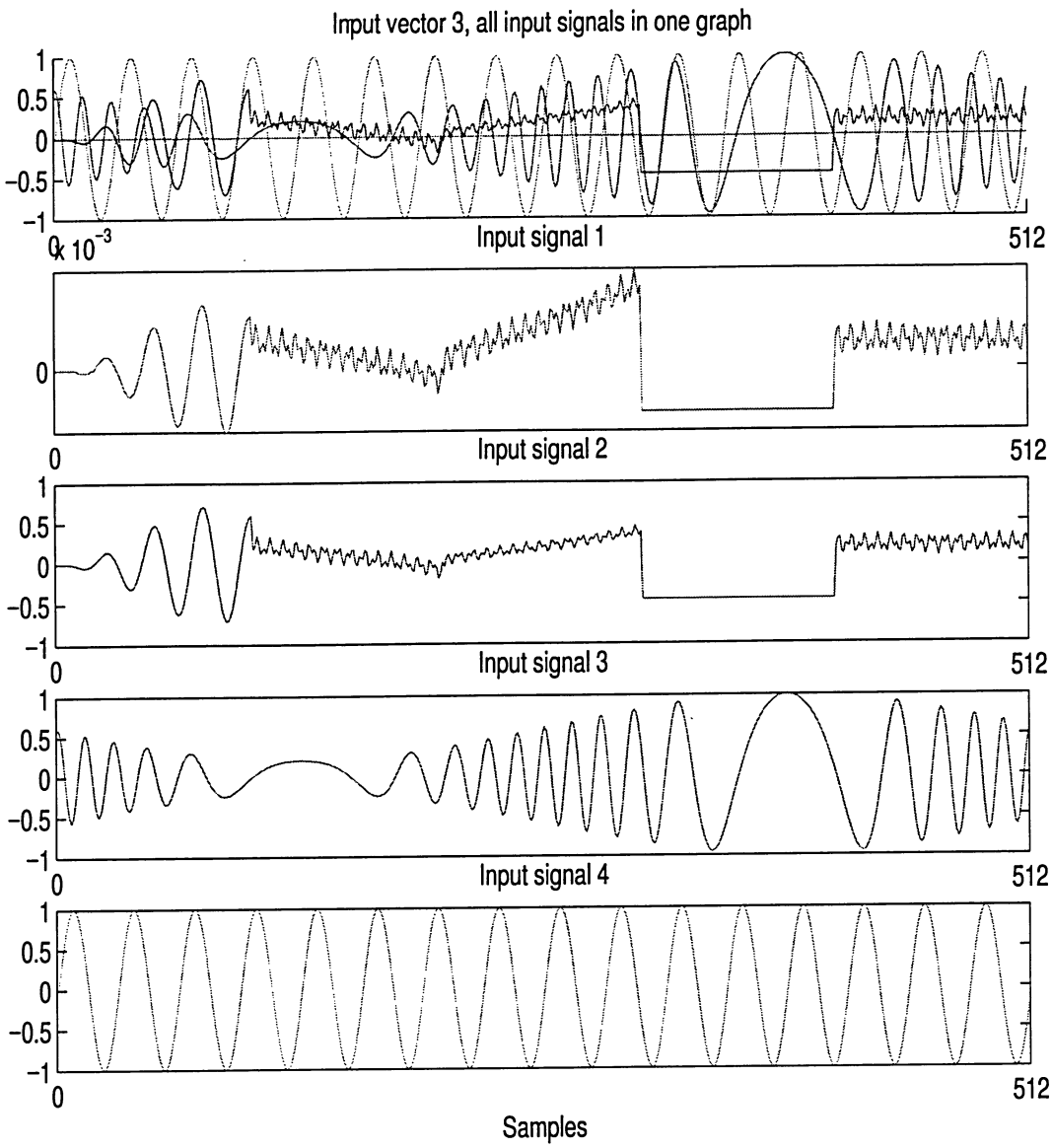

Figure 4.3: Input vector 3.

Input vector 3 has the most complicated form and contains signals with amplitude modulated tones, while the signal 4 is a simple tone. The same way as vectors 1 and 2, the input vector 3 has 512 samples. The next figure shows input vector 4:

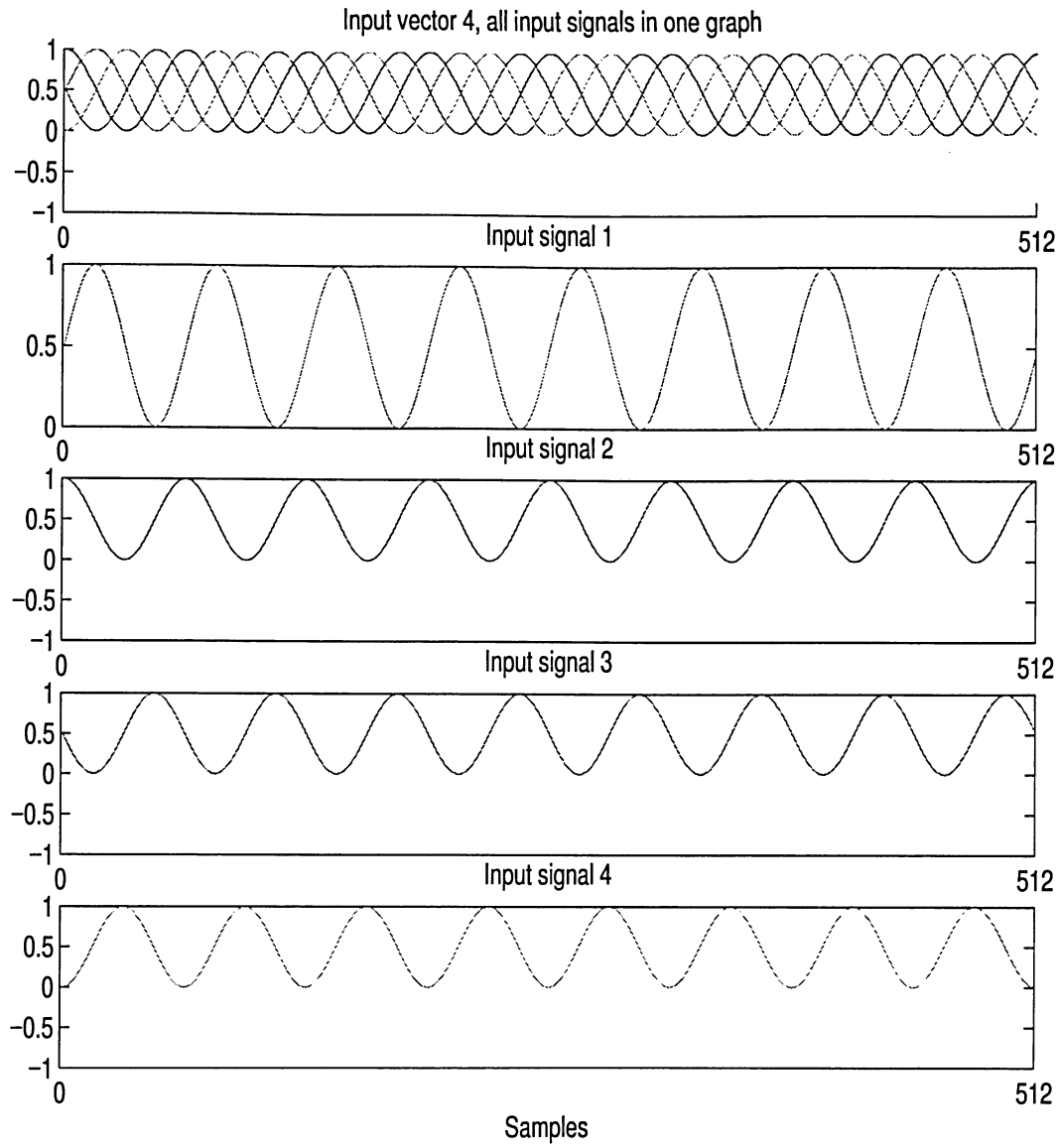

Figure 4.4: Input vector 4.

Presented above input vector 4 has 4 signals that are the same tone but with different initial phase, spaced by $\pi/2$. All these signals have 512 samples with values in segment $[0, 1]$. The next figure shows the last input vector 5:

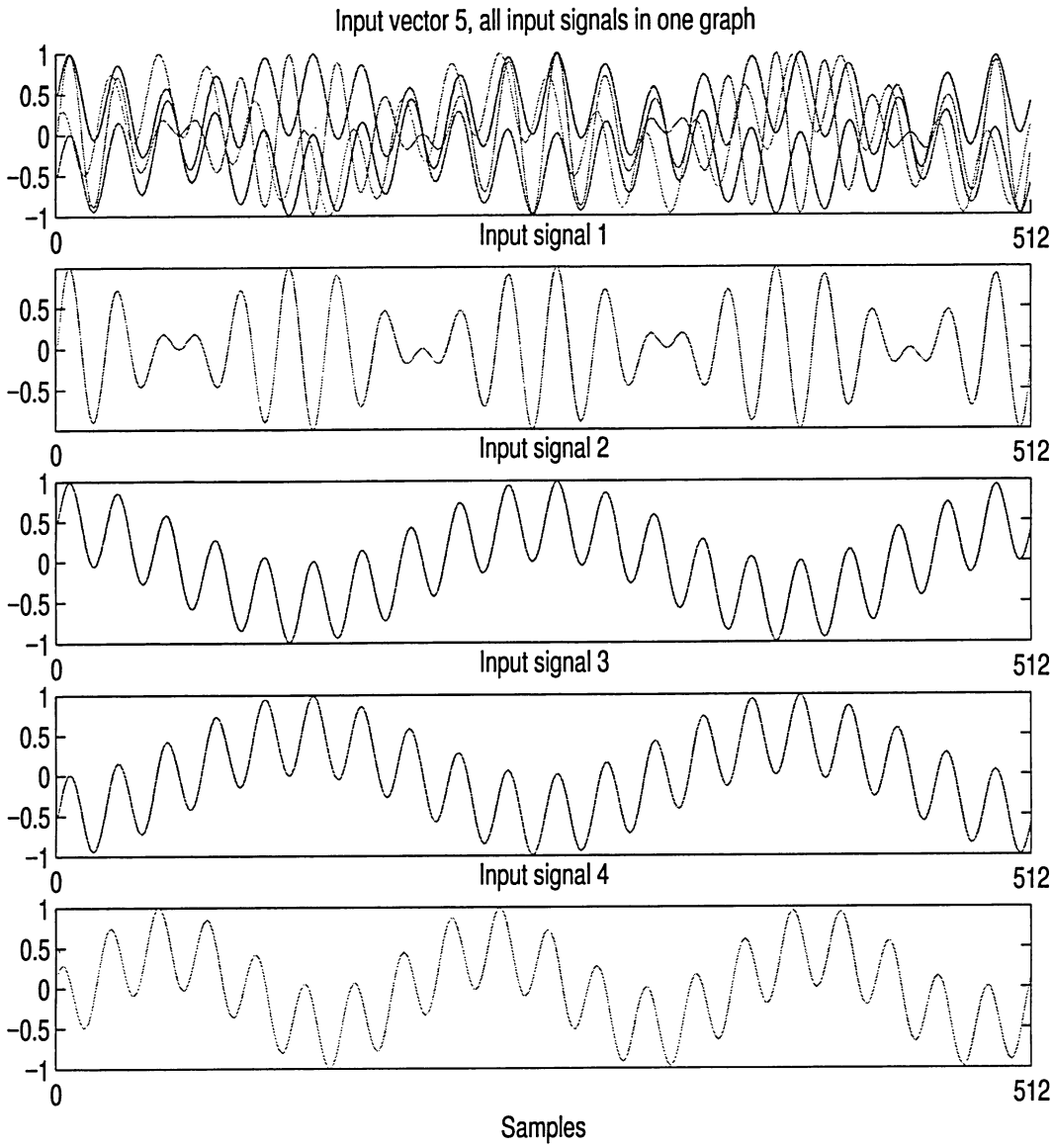

Figure 4.5: Input vector 5.

The input vector 5 has frequency-modulated signals, and each one has 512 samples, the true for all input vectors. All signals of input vector 5 are normalized so that their values lie in the segment $[-1, 1]$. This is the last input vector used in the VGS algorithm implementation.

4.2 Implementation of the Adaptive Vector Greedy Algorithm

We compare the adaptive VGS algorithm with the classical Haar Wavelet transform. We present plots for the relative errors generated by the signal approximations for both algorithms. In all figures the approximations to the inputs provided by the VGS algorithm are plotted in solid lines, while the Haar wavelet approximations are plotted in dotted lines. Also, the plotted VGS and Haar wavelets approximations of the signals can be easily distinguished given that they look like step functions and, therefore, they are easily differentiated from smooth original input signals.

The first picture presents approximation of the scalar input vector 2 by VGS algorithm after one split only:

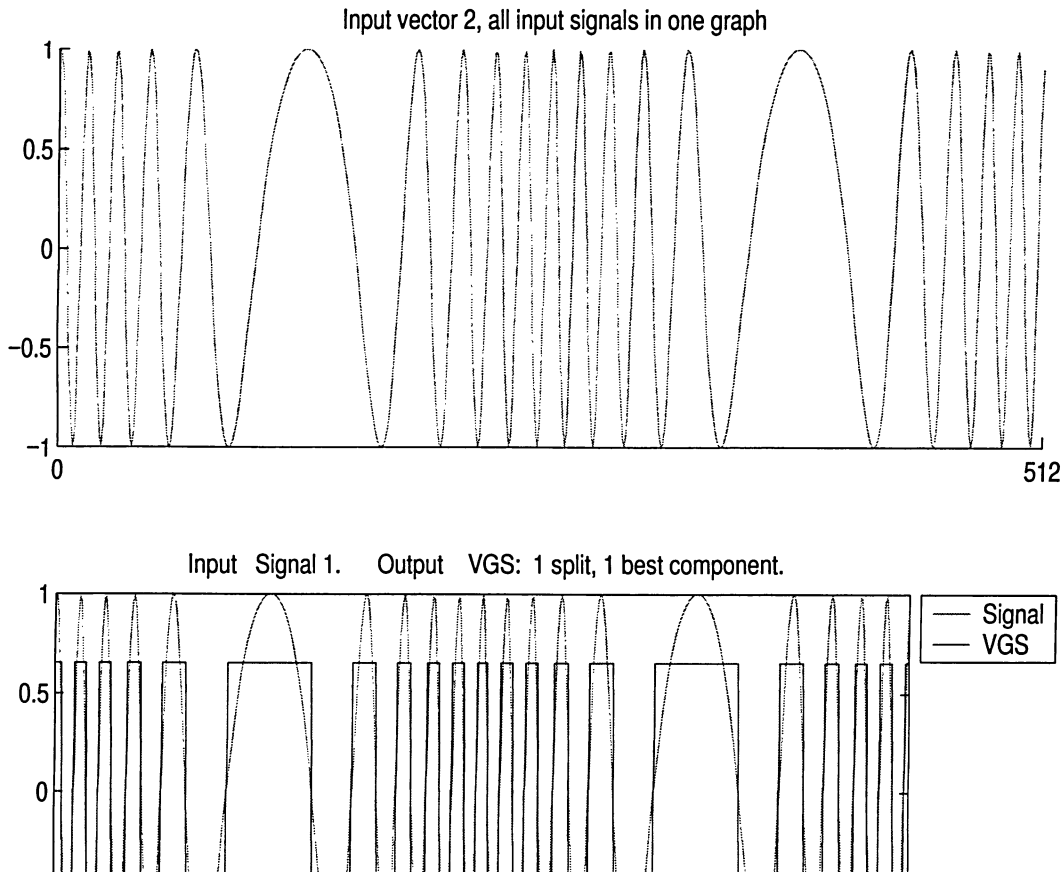

Figure 4.6: Scalar GS approximation of first signal of vector 2 using only one component and one split to build partition tree.

On the picture above and on the next two Figures 4.7 and 4.8 we illustrate VGS algorithm. On the Figure 4.6 we use input vector 2 with signal 1 only. We implemented only one iteration of VGS algorithm. It means that only root of the tree was split and the whole partition tree consists of the node, that is root and its two children that are leaves of the tree. In this case we have only one Haar function to approximate input signal. As we see from the graph, all oscillations of the input signals are reflected by VGS approximation.

Next we apply the algorithm to a 4-dimensional input vector 2 and perform 2 iterations of the adaptive VGS algorithm:

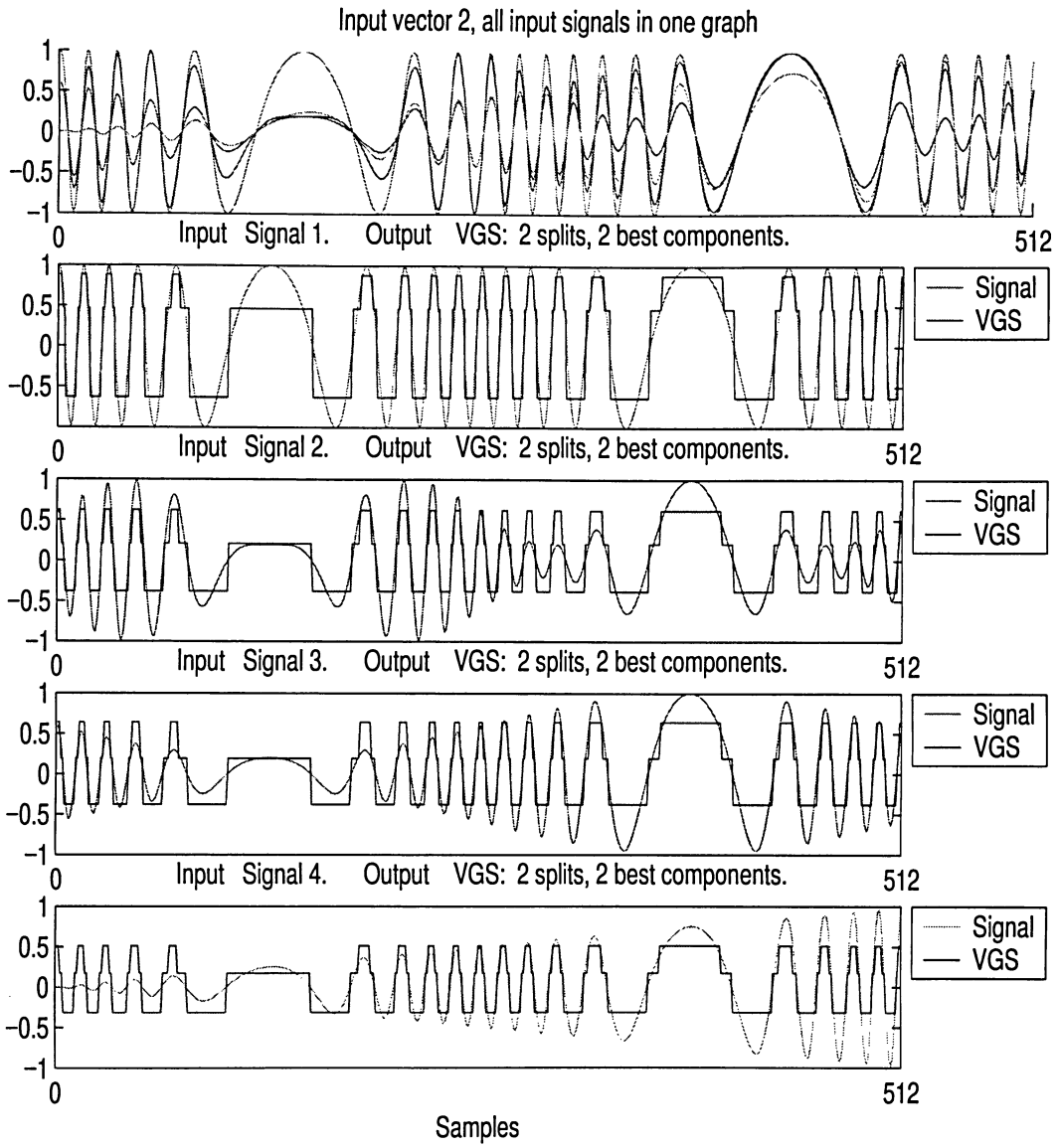

Figure 4.7: Adaptive VGS approximation of input vector 2 using two components after two splits.

As we see from the picture above the all oscillations of the input signals are reflected by the VGS algorithm and its approximation for each input component is presented by step-function of 3 values, corresponding to the fact that only two Haar functions were used. It is one step further than case presented in Figure 4.6, since now we have

partition tree of two nodes and 3 leaves. These two nodes are root of the tree and one of its children. We see the signals approximation consists of the 3 values reflecting the fact that partition tree has 3 leaves.

The next picture presents the approximation of a 3-dimensional input vector 2 by the VGS algorithm after 5 splits:

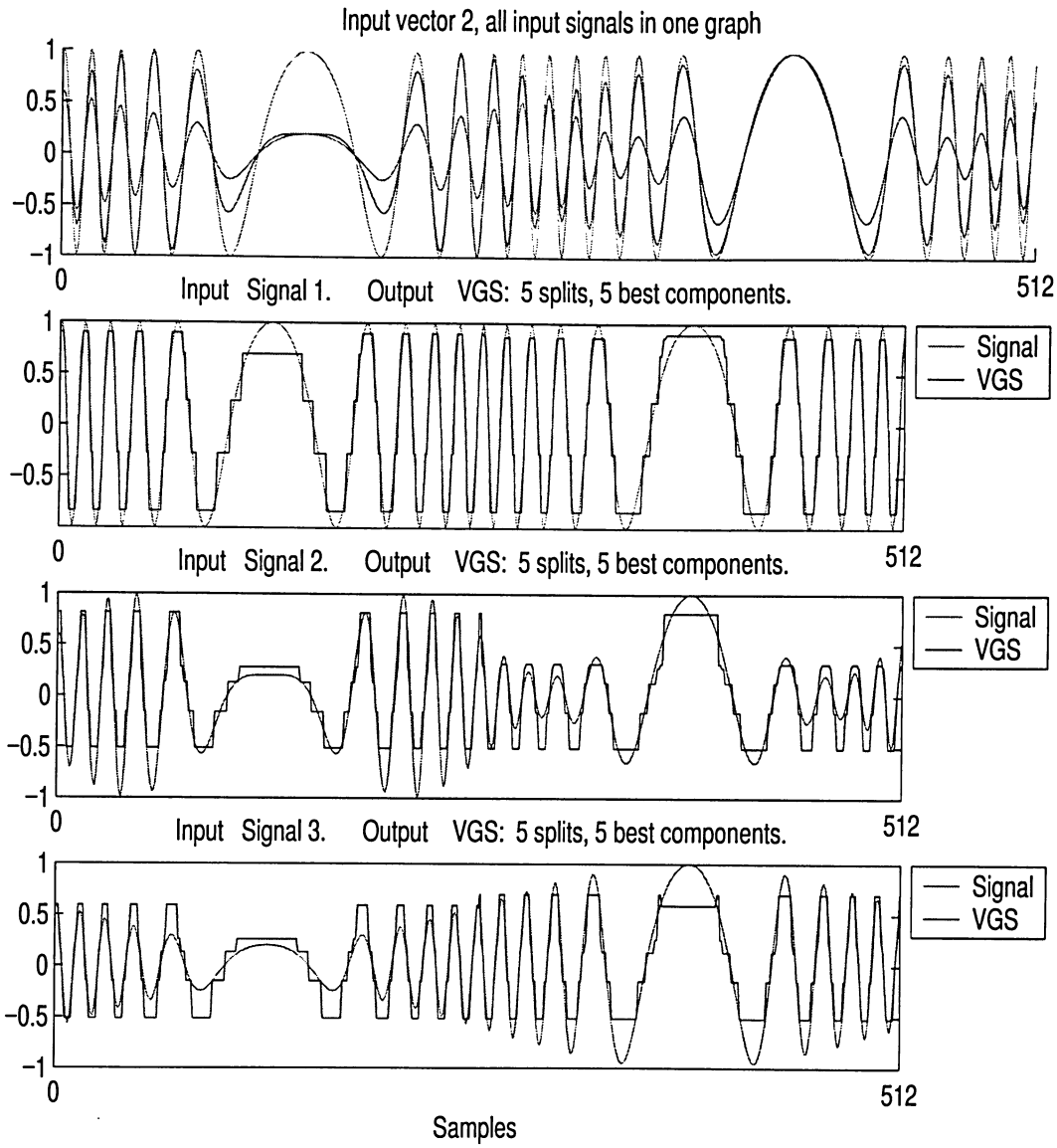

Figure 4.8: Adaptive VGS approximation of 3-dimensional input vector 2 using five components after five splits.

On the picture above we use all 5 Haar functions to represent input vector 2 after 5 iterations of VGS algorithm. The partition tree in this case has 5 nodes and 6 leaves. Correspondingly, each input signal is approximated through step-function taking 6 values. When we compare plots of this figure with previous Figure 4.7 we

note that the input signals are represented better as larger number of iterations and Haar functions is used.

In the next section we present VGS and Haar wavelet approximations for the same input vectors on the same plots.

4.3 Illustration of Adaptive VGS Algorithm vs Haar Transform

The next picture presents the approximation of a 3-dimensional input vector 2 by the VGS algorithm after 5 splits. We also show Haar Wavelets approximation using at least 5 best components.

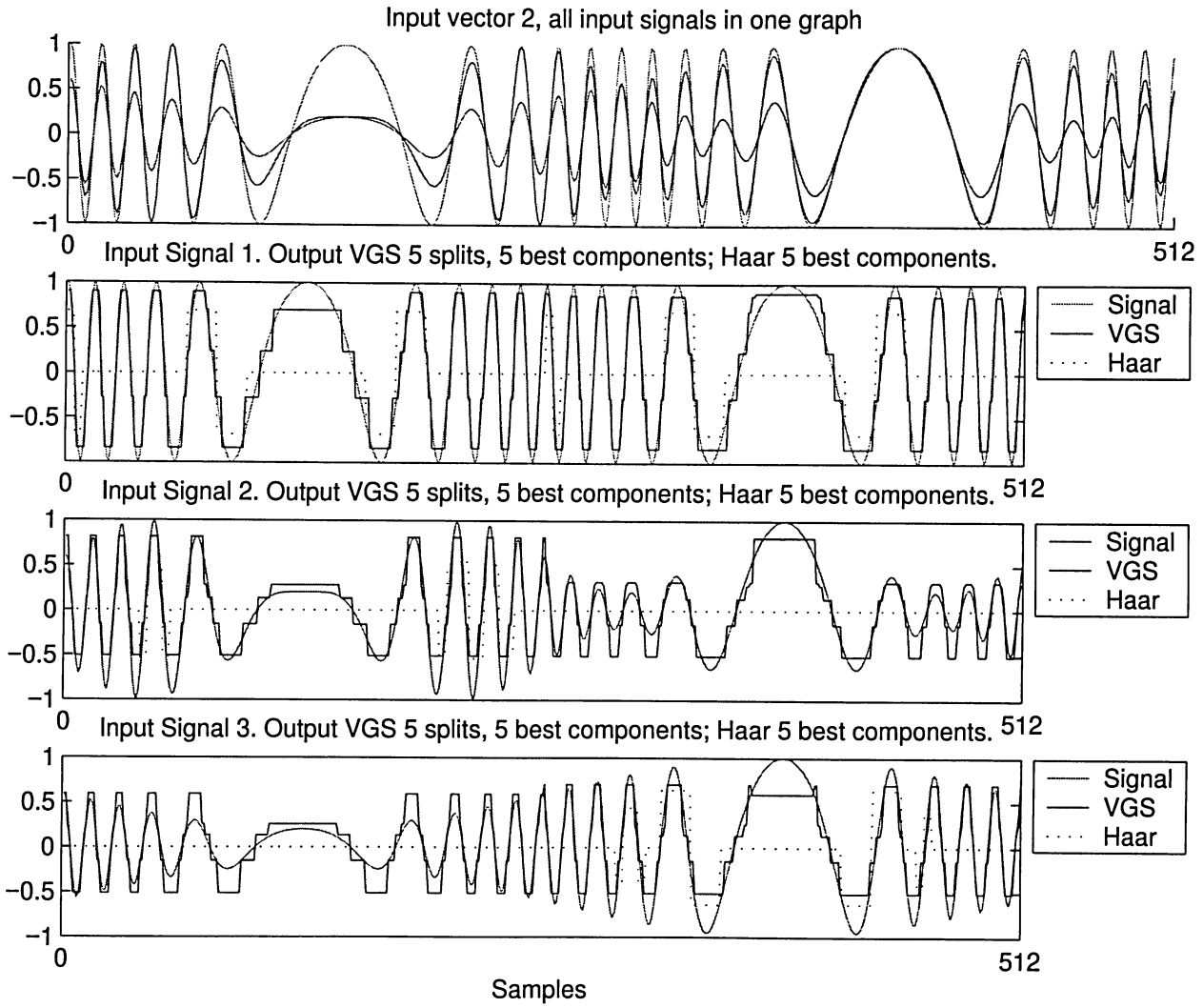

Figure 4.9: Approximation of input vector 2 using five components of Adaptive VGS after 5 splits and 5 best components of Haar Wavelets.

In the case presented above we used input vector 2 presented in the Figure 4.2. We implemented 5 iterations of the adaptive VGS algorithm. Also we implemented

classical Haar wavelet transform. All 5 components of VGS partition tree were used to represent input, in another words, compression was not involved. Also, at least 5 best components of the Haar wavelets were used to approximate the same input vector. Plots for the input signals and their approximations by VGS and Haar wavelets are plotted. We may see that VGS algorithm reflects all the oscilations of the signal and also, its approximation is closer to the input than in case of the Haar wavelet, that follows from the fact that each Haar wavelet has only 2 values.

The next picture presents the approximation of a 3-dimensional phase-shifted input vector 4 by the VGS algorithm after 5-splits. We also show the Haar transform approximation with at least 5 best components.

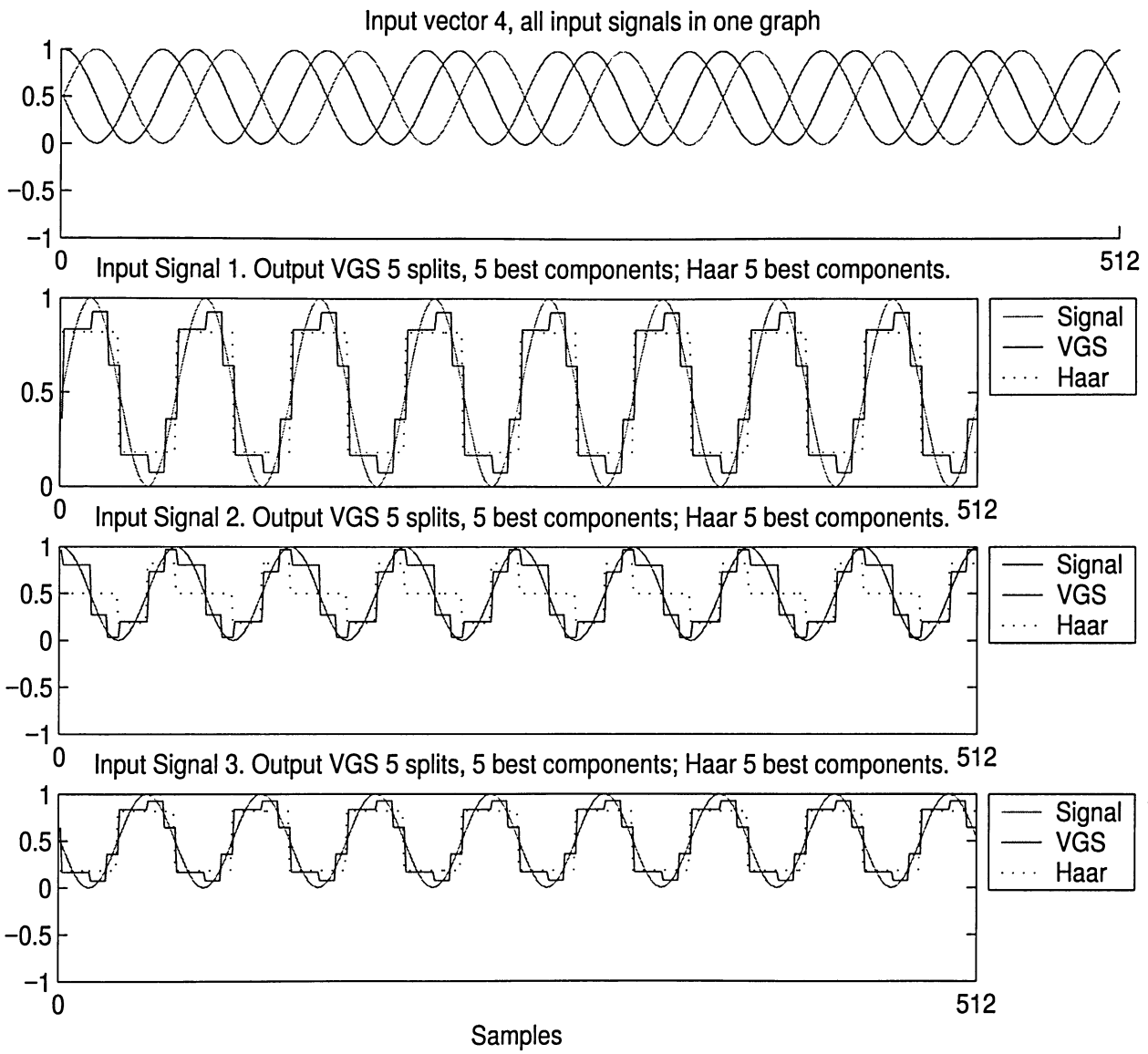

Figure 4.10: Approximation of input vector 4 using five components of Adaptive VGS after 5 splits and 5 best components of Haar Wavelets.

On the figure above we perform the same processing as in the Figure 4.9, but for the input vector 4. This input is presented in the Figure 4.4. Also, in this case we took more Haar wavelets. We required to use all Haar wavelets having 5 largest values.

For the input vector 4 it happen to be 8 Haar wavelets, all carying the same energy. As we see both algorithms, VGS and haar wavelets reflect all oscillations of the input harmonics.

The next picture presents the approximation of 3-dimensional input vector 3 by the VGS algorithm after 5-splits without compression. We also show 5 best components of the Haar wavelet transform.

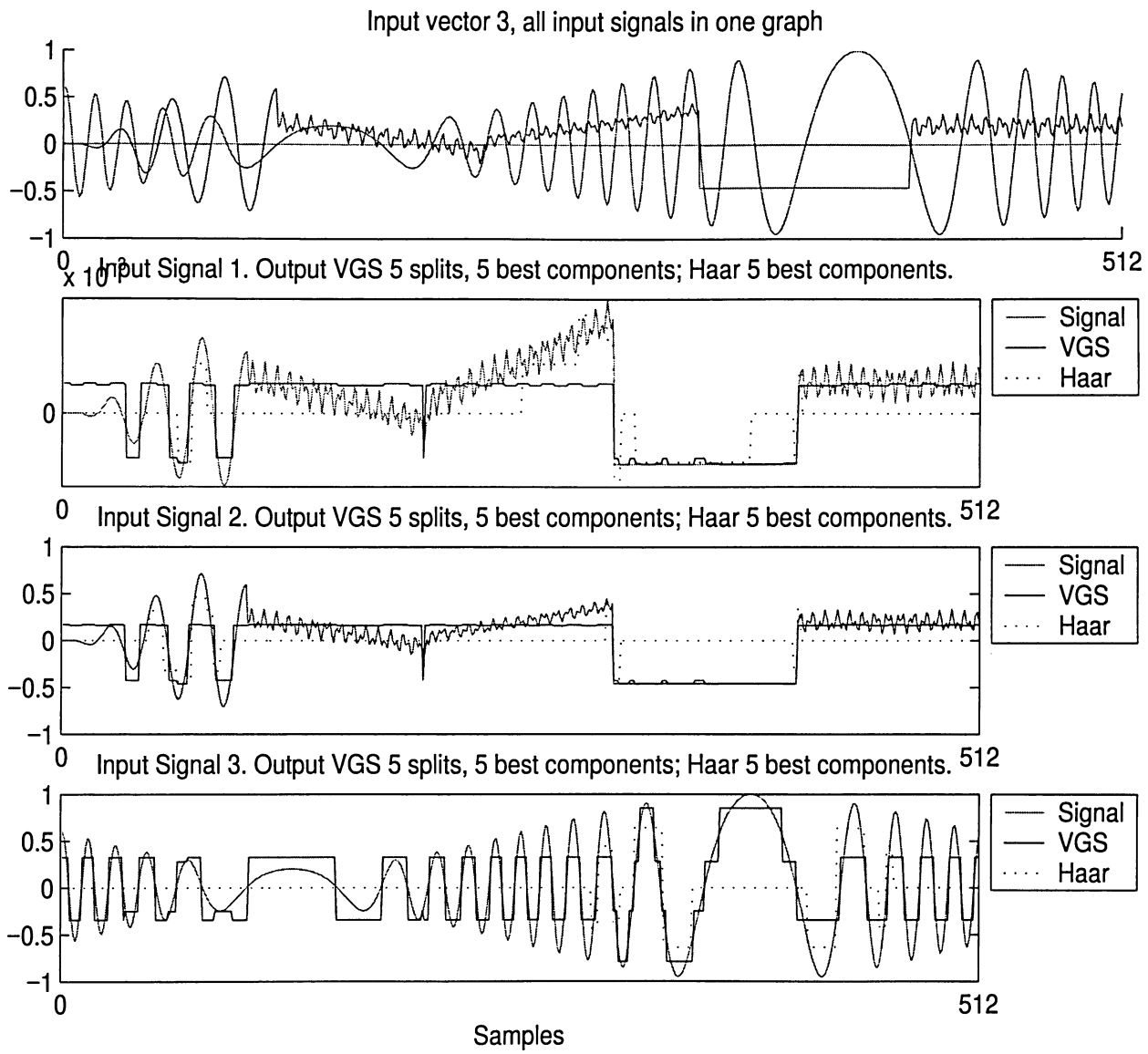

Figure 4.11: Approximation of input vector 3 using five components of Adaptive VGS after 5 splits and 5 best components of Haar Wavelets.

On he picture above we see that VGS algorithm does not reflects all high frequency oscillations. In order to provide better approximation the deeper partition tree must be used. However, VGS output approximates these high oscillations by averaging them,

while Haar wavelets reflects only 5 chosen harmonic periods. For such complicated input both algorithm require more components for proper approximation.

Next, we introduce a compression ratio of 50% for VGS algorithm.

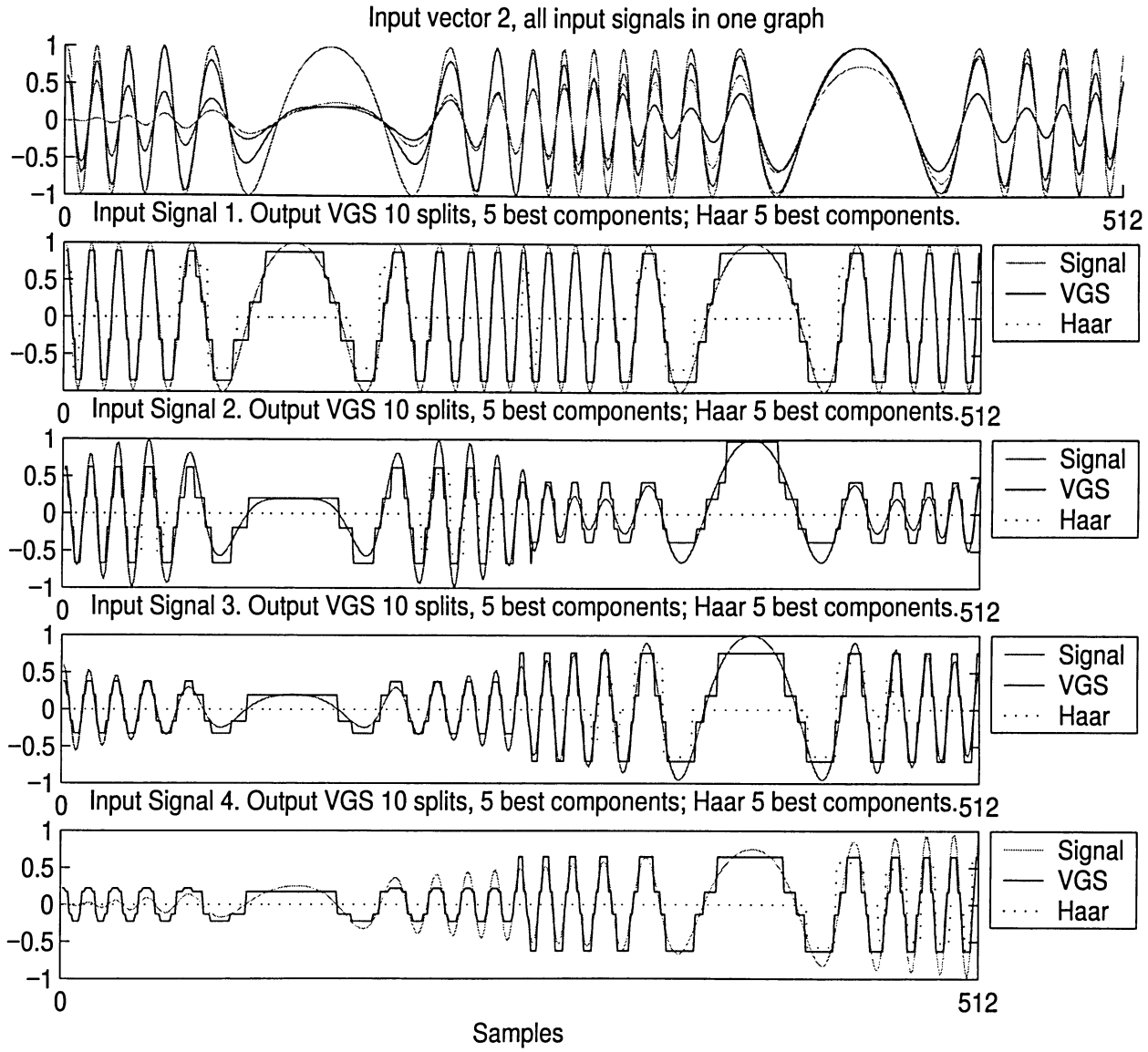

Figure 4.12: Approximation of input vector 2 using 5 best components of Adaptive VGS after 10 splits and 5 best components of Haar Wavelets.

In the figure above we present results of the case, where the input is 4-dimensional vector 2, approximated by 5 largest components after compressing half of the 10 components of the partition tree. We apply and compare results with 5 best splits of the Haar wavelet transform. As we see even after compression the VGS algorithm provides better representation of the input.

4.4 Statistical Comparison of VGS Algorithm vs Haar Transform

In this section we present statistical results for VGS vs Haar wavelets performance. We apply both algorithms to the all input vectors 1 to 5, each one having 4 signal components. The inputs are presented in Figures 4.1 - 4.5.

In order to provide statistics to compare the performance of our adaptive VGS Algorithm vs Haar Wavelets transform we define the error E for either of the two algorithms to be the difference between input vector of signals $X = \{X_1, \ldots, X_d\}$ and its approximation $\hat{X} = \{\hat{X}_1, \ldots, \hat{X}_d\}$. Using this notation we get error energy to be equal:

$$\begin{aligned} \|E\|^2 &= [E,E] = \int_\Omega \|E(w)\| dP(w) = \int_\Omega \Big(\sum_1^d E_i(w)^2\Big) dP(w) \\ &= \sum_1^d \|E_i(w)\|^2 \,. \end{aligned} \tag{4.1}$$

We then calculate the relative errors by calculating the ratio of the energy of the error and energy of the input signal, i.e.

$$RelativeError = \frac{\|E\|^2}{\|X\|^2} \quad \text{- for total vector of signals,}$$

$$RelativeError_i = \frac{\|E_i\|^2}{\|X_i\|^2} \quad \text{- for each component } i \in \{1, \ldots, d\} \,.$$

Here we show statistical comparison of performance for Adaptive VGS algorithm vs Haar Wavelets for all 5 different types of 4-dimensional inputs vectors. We plot the relative errors as a function of the compression rate (which goes in the x-axis). We have run 100 splits of the adaptive VGS algorithm and have performed 19 steps of compression, starting from 5% and going up to 95%. We plot the graphs for best, worst and average errors of VGS (solid lines) and Haar Wavelets (dashed lines) for

all compression ratios in according to the formulae:

$$min_i\left\{\frac{\|E_i\|^2}{\|X_i\|^2}\right\} \quad i = 1, \ldots, d \quad \text{- best error,}$$

$$max_i\left\{\frac{\|E_i\|^2}{\|X_i\|^2}\right\} \quad i = 1, \ldots, d \quad \text{- worst error,}$$

$$\frac{\sum_1^d \|E_i\|^2}{\sum_1^d \|X_i\|^2} \quad \text{- average error.}$$

For input vector 1 the results are:

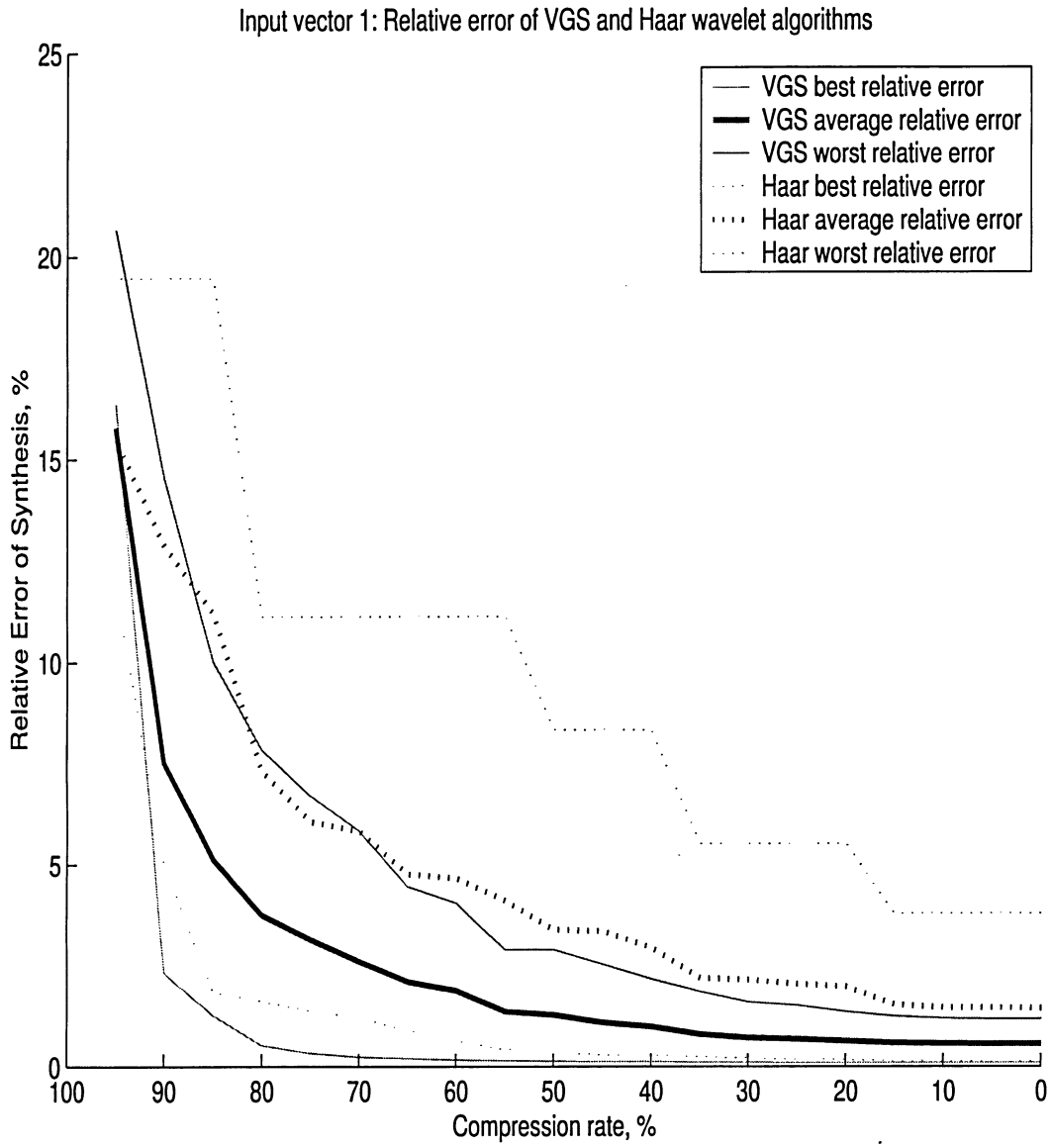

Figure 4.13: Relative error ratio of VGS vs Haar for input tones

On the graph above we see plots of best, average and worst relative error for VGS and Haar wavelets applied to the same input vector 1, as in Figure 4.1. As we know the input signals have 512 samples with possibility of taking up to 512 different values. We run 100 iterations for adaptive VS algorithm. That means that our partition tree has 100 nodes and 101 leaves. When we do not compress the tree, we use 101 values to represent input signals. These 101 values may represent without any error

only signal with maximum 101 values. However our signal takes up to 512 values, so even in the case of 0% compression, i.e. no compression at all, the error can not be eliminated completely. The same arguments are applied to the Haar wavelets, since we use maximum only 100 best of the Haar wavelets. When compression is applied we use only percentage of the best Haar wavelet components. For example, for compression rate 25% we use 75 best haar wavelets while compressing 25 worst ones. As we see from the plots in the Figure 4.13 the VGS algorithm gives lower relative errors when compression rate 90% and less. When compression rate 95%, i.e. only 5 best of the components are used the Haar wavelets give better results.

For input vector 2 the results are: On the graph above we see plots of best, average

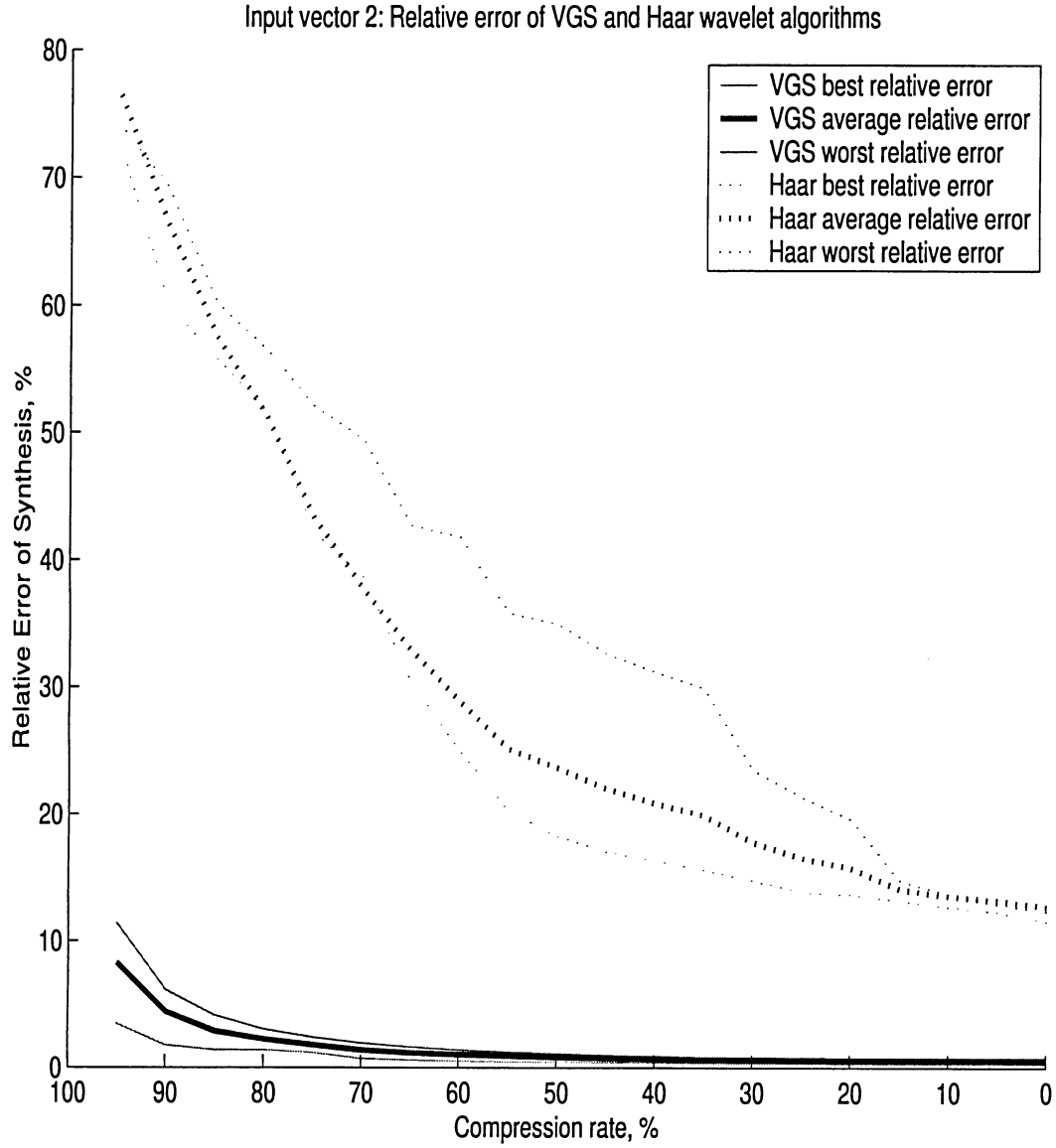

Figure 4.14: Relative error ratio of VGS vs Haar for input FM-modulated signals

and worst relative error for VGS and Haar wavelets applied to the same input vector 2, as in Figure 4.2. We can see, that for this input the VGS algorithm provides very fast decay of relative error. Also even the VGS approximation to the signal giving worst relative error is better than the best Haar wavelet approximation. With compressions approaching 0% the Haar wavelets relative error does not goes below 10%, while relative error of VGS algorithm is below 10% when compression is below 90%.

For input vector 3 the results are:

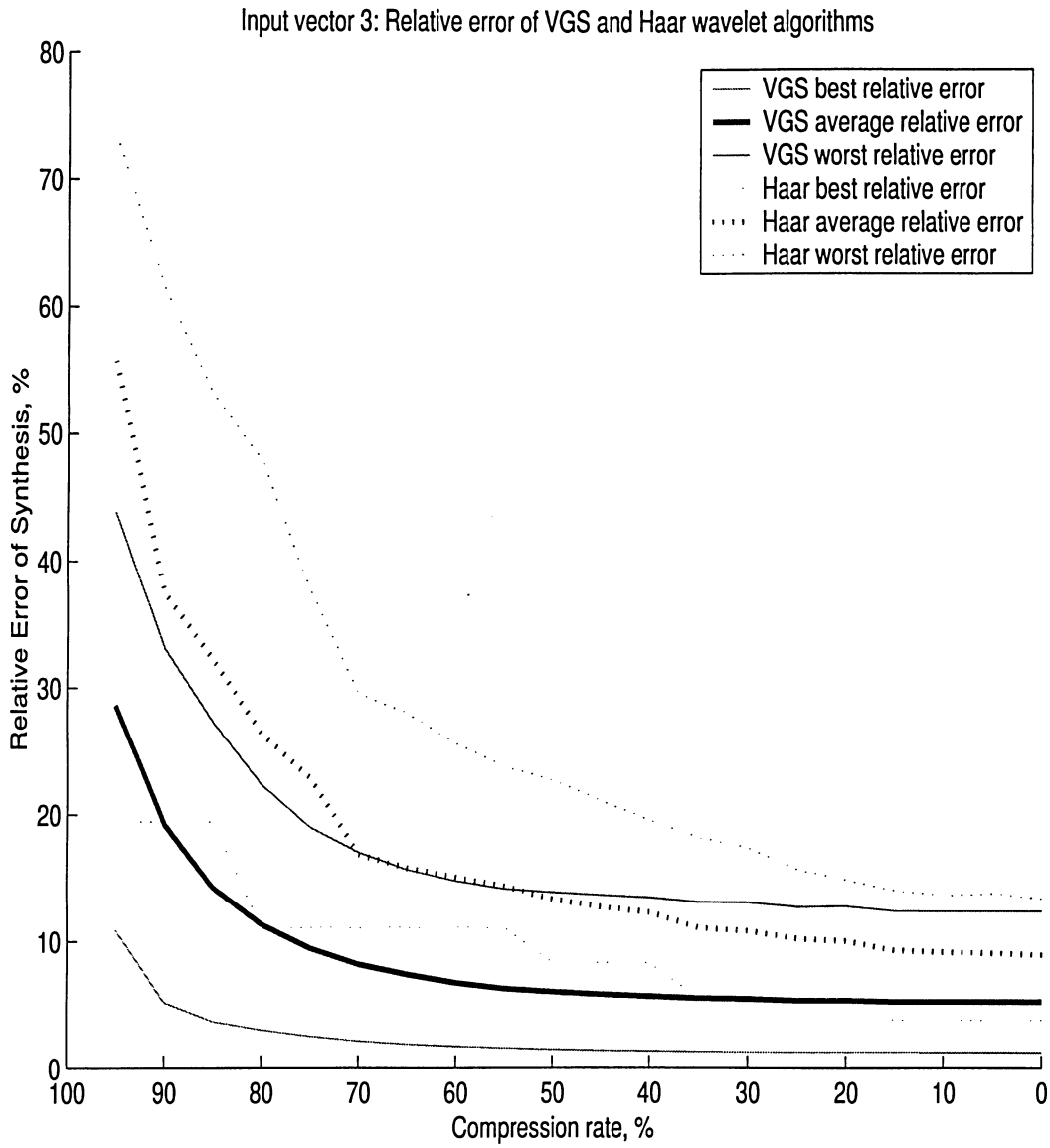

Figure 4.15: Relative error ratio of VGS vs Haar for arbitrary inputs

On the graph above we see plots of best, average and worst relative error for VGS and Haar wavelets applied to the same input vector 3, as in Figure 4.3. This is the most complicated input and the results show slow decay of the relative error for both, VGS and Haar wavelets. However relative error of best VGS approximation is lower than relative error of the best Haar wavelet, average relative error of VGS approximation is lower than average relative error of the Haar wavelet and relative error of the worst

VGS approximation is lower than relative error of the worst Haar wavelet.

For input vector 4 the results are:

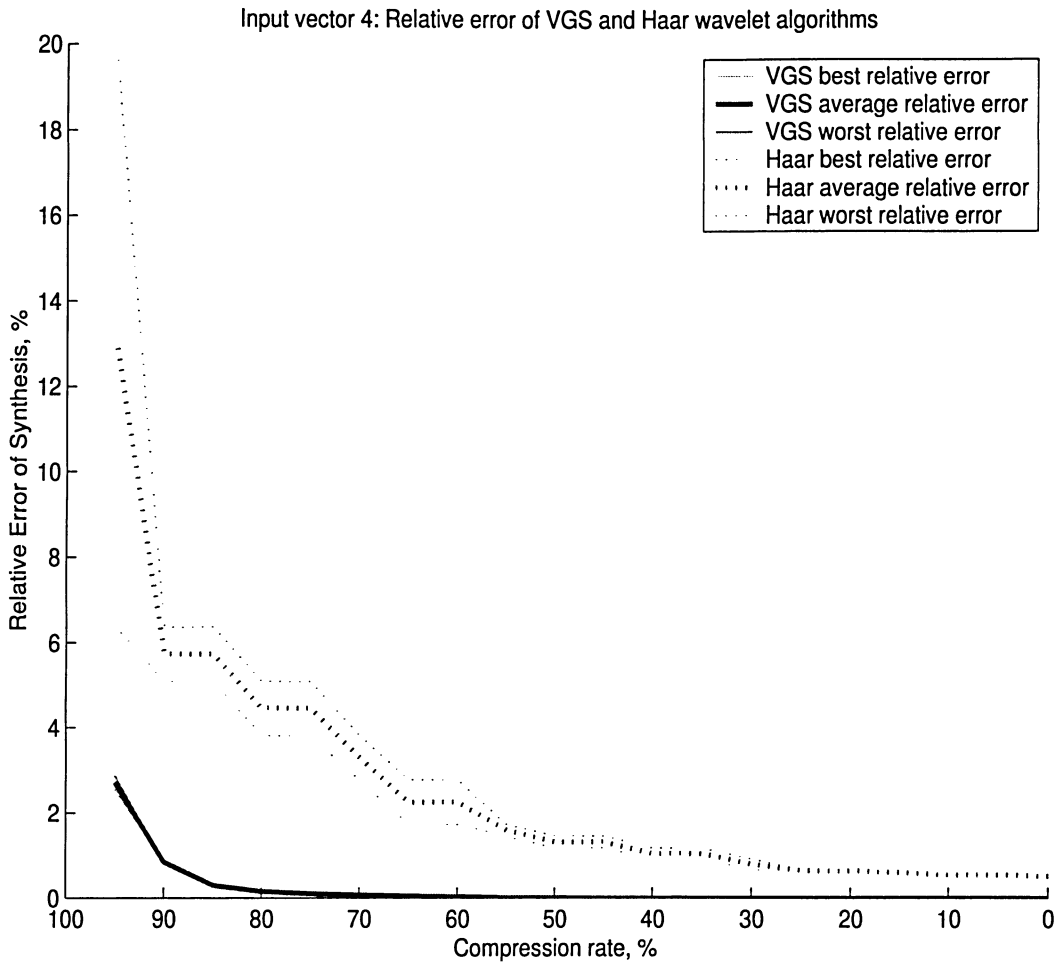

Figure 4.16: Relative error ratio of VGS vs Haar for input phase-shifted signals

On the graph above we see plots of best, average and worst relative error for VGS and Haar wavelets applied to the same input vector 3, as in Figure 4.4. In this case VGS algorithm is by far superior to the Haar wavelets. Also relative errors for the best and worst signal approximation almost coincide for the VGS algorithm. They decay very fast.

For input vector 5 the results are:

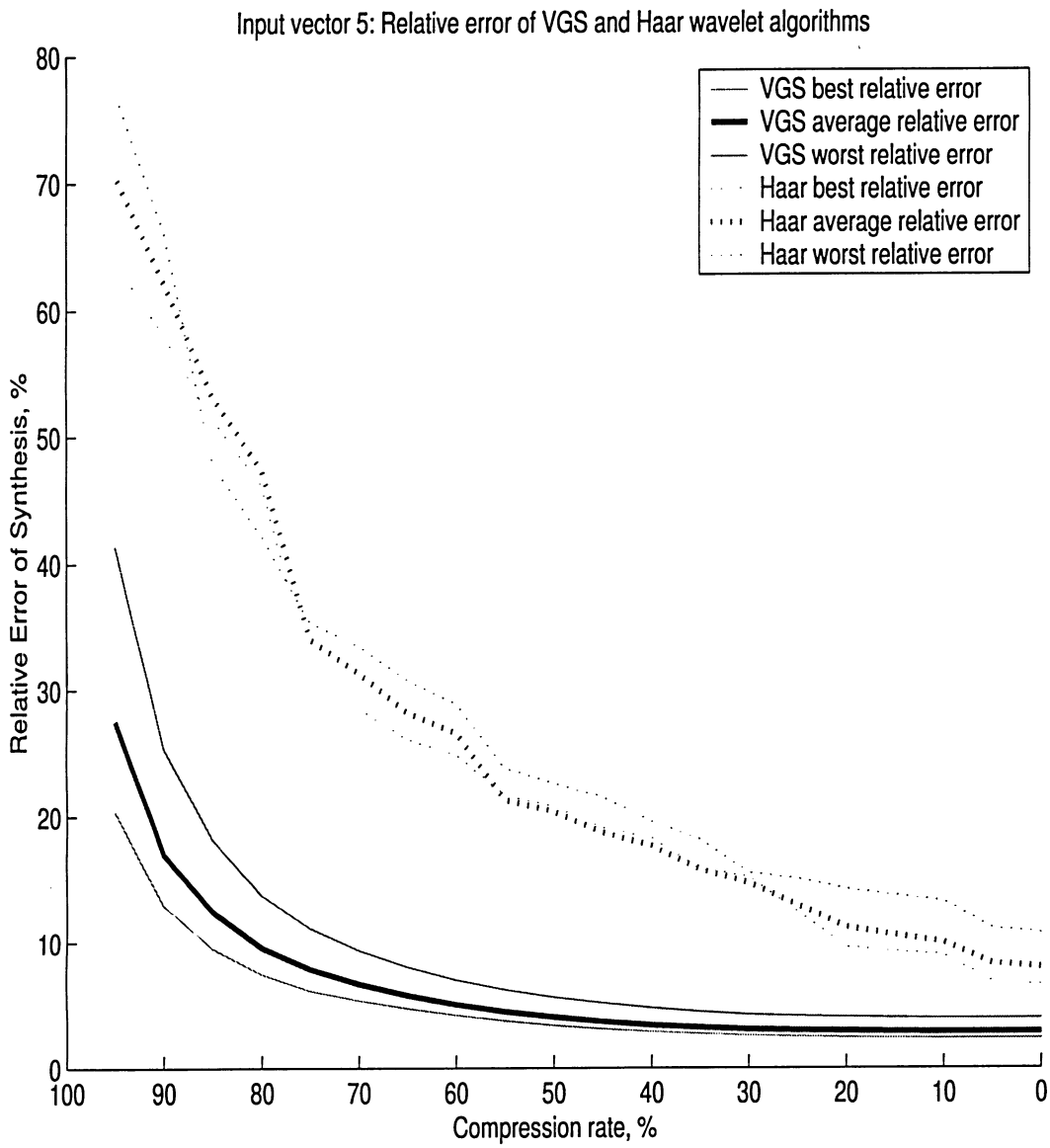

Figure 4.17: Relative error ratio of VGS vs Haar for input signals type 4

On the graph above we see plots of best, average and worst relative error for VGS and Haar wavelets applied to the same input vector 5, as in Figure 4.5. In this case VGS algorithm is also better than Haar wavelets. Even relative errors for the worst signal approximation by the VGS algorithm is lower than best result for the Haar wavelets.

Chapter 5

Conclusions

The Adaptive Vector Greedy Splitting Algorithm is a new tool available for signal analysis. It is applicable to different types of signals, especially multi-dimensional, like images. In the present work we have studied some of the basic aspects of the technique as well as performed a software implementation which gives an indication of the advantages offered by this new tool.

There are several interesting open questions related to the approach, for example: Is there a quantitative relationship between the input vector and the speed of convergence of the approximation? An answer to this question could indicate, a-priori, if a given collection can, or can not, be efficiently approximated by the technique.

Another question is whether we may estimate how large the dimension d of the input vector has to be so the cost (in a compression application) of storing the VGS basis is small relative to the total cost (in terms of bits)? Suppose, the length of each signal component for a d-dimensional input vector is N samples, and each sample is R bits. Let us say that we want to achieve compression rate k_t $(0 < k_t < 1)$, where t stands for a target compression rate. Using this notation we want to have the size of the output not larger than $N \cdot d \cdot R \cdot k_t$ bits. Suppose, we run M iterations of the adaptive VGS algorithm. We will denote our compression rate k_{VGS}, and we assume that we need c words (each one R bits) to contain data structure for each node of the partition tree. In this case $c \simeq 5$, since we need two words to contain pointers to the children, 2 words to be pointers to associated set A_i etc. So all together we need:

$$N \cdot R + (2M+1) \cdot c \cdot R \quad \text{bits to store} \quad 2M+1 \quad \text{nodes and leaves.}$$

In the formula above $N \cdot R$ is a size of the input probability space Ω, and $(2M+1)cR$ are pointers to chunks of partition subsets A_i. We only store non zero inner products d_i, for the whole partition tree this requires $(2M+1)dR \cdot 2K$ bits, where 2 means that we store only non-zero d_i and corresponding indexes i, i.e. 2 words per each non

zero component. All together we obtain the following equation:

$$NR + (2M+1)\cdot cR + (2M+1)\, d\, tR \cdot 2\, k_{VGS} \;<\; N\, d\, R\, k_t\,.$$

From here

$$d\big(N \cdot k_t - 2k_{VGS}(2M+1)\big) > N + (2M+1)\cdot c\,,$$

or, equivalently,

$$d > \frac{N + (2M+1)\cdot c}{N \cdot k_t - 2 \cdot k_{VGS} \cdot (2M+1)}\,.$$

If N is big, i.e. $M = o(N)$, and $c = o(N)$ we obtain a simpler expression for evaluating dimension d:

$$d > \frac{N}{N \cdot k_t - 4k_{VGS} \cdot M}\,.$$

If we simplify further by assuming $k_{VGS}M = o(N)$ we obtain the final simple formula

$$d > \frac{1}{k_t}\,. \tag{5.1}$$

As we see from the formula above for long signals (i.e. large N) there is a relationship between dimension of the signal and target compression rate. We may approach formula (5.1) from both sides. When dimension is fixed we have limit for the compression rate, while for the fixed compression rate we have requirement for the minimum dimension of the input vector.

There is a variety of potential applications for this method. The most straightforward application is signal compression. Another application is audio pattern recognition, which may be used, for example, in biometrics. In this case, we record a number of times the same short phrase (password) recorded by the same person. After applying some Digital Signal Processing techniques (activity detection, Automatic Gain Control), we may perform adaptive VGS for this set of input signals. We then build an H-system and associated tree. Next time we receive a new sample of the spoken phrase we may analyze it to see if it was spelled by the same person. This can be achieved by a comparison of the relative errors.

Appendix A

Vector H-Systems and their properties

A.1 Properties of H-functions

We will prove that formula (3.6), presenting the general form of the vector Haar function, follows from Definition 1.

Lemma 1. *Suppose, set $A \subseteq \Omega$ and may be presented as union of two disjoint subsets A_0 and A_1, s.t. $P(A_0) \neq 0$ and $P(A_1) \neq 0$. Then any point in the unit sphere $b' \in S^d$ defines Vector Haar function $\psi_{A,b'}$ of the following form:*

$$\pm\psi_{A,b'} = \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\, \mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\, \mathbf{1}_{A_1} \right) b' \,. \tag{A.1}$$

Here $b' \in S^d$ and $A = A_0 \cup A_1, A_0 \cap A_1 = \emptyset$.

Proof. We show that $\psi_{A,b'}$ meets the requirements of Definition 1, i.e.

$$\int_\Omega \psi_{A,b'}(\omega)\, dP(\omega) = 0, \quad \text{and} \quad \|\psi_{A,b'}\|^2 = \int_\Omega \|\psi_{A,b'}(\omega)\|^2\, dP(\omega) = 1 \,.$$

For the first integral we obtain

$$
\begin{aligned}
\pm \int_\Omega \psi_{A,b'} dP &= \int_\Omega b' \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\, \mathbf{1}_{A_0}(\omega) - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\, \mathbf{1}_{A_1}(\omega) \right) dP(\omega) \\
&= b' \left(\sqrt{\tfrac{P(A_1)}{P(A)P(A_0)}} \int_\Omega \mathbf{1}_{A_0} dP - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \int_\Omega \mathbf{1}_{A_1} dP \right) \\
&= b' \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}} P(A_0) - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} P(A_1) \right) \\
&= b' \left(\sqrt{\frac{P(A_1)P(A_0)}{P(A)}} - \sqrt{\frac{P(A_0)P(A_1)}{P(A)}} \right) = 0
\end{aligned}
$$

We use the following trivial identities:

$$
\mathbf{1}_{A_{1,0}} \cdot \mathbf{1}_{A_{1,1}} = 0, \qquad \left(\mathbf{1}_{A_{1,1}}\right)^2 = \mathbf{1}_{A_{1,1}}, \qquad \left(\mathbf{1}_{A_{1,0}}\right)^2 = \mathbf{1}_{A_{1,0}}, \qquad \|b'\|^2 = 1 \,.
$$

Then for the second condition we obtain:

$$
\begin{aligned}
\|\psi_{A,b'}\|^2 &= \int_\Omega \|b'\|^2 \left\| \sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\, \mathbf{1}_{A_0}(\omega) - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\, \mathbf{1}_{A_1}(\omega) \right\|^2 dP(\omega) \\
&= \|b'\|^2 \cdot \int_\Omega \left(\frac{P(A_1)}{P(A)P(A_0)}\, \mathbf{1}_{A_0}(\omega) + \frac{P(A_0)}{P(A)P(A_1)}\, \mathbf{1}_{A_1}(\omega) \right) dP(\omega) \\
&= \frac{P(A_1)}{P(A)P(A_0)} \int_\Omega \mathbf{1}_{A_0}(\omega) dP(\omega) + \frac{P(A_0)}{P(A)P(A_1)} \int_\Omega \mathbf{1}_{A_1}(\omega) dP(\omega) \\
&= \frac{P(A_1)}{P(A)P(A_0)} \int_{A_0} dP(\omega) + \frac{P(A_0)}{P(A)P(A_1)} \int_{A_1} dP(\omega) \\
&= \frac{P(A_1)}{P(A)P(A_0)} \cdot P(A_0) + \frac{P(A_0)}{P(A)P(A_1)} \cdot P(A_1) \\
&= \frac{P(A_1)}{P(A)} + \frac{P(A_0)}{P(A)} = \frac{P(A_1) + P(A_0)}{P(A)} = \frac{P(A)}{P(A)} = 1 \,.
\end{aligned}
$$

□

From equation (A.1) follows that for the scalar case, ($d = 1 \Rightarrow b' = 1$), the partition

$\{A_0, A_1\}$ of set A defines a scalar Haar function

$$\pm\psi_A = \sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\,\mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\,\mathbf{1}_{A_1}\,. \tag{A.2}$$

Appendix B

Vector Greedy Splitting Algorithm justification

B.1 Basis Function ψ_0 for Trivial Partition $\mathcal{Q}_0$

This section uses the method of Lagrange multipliers to obtain the expression for the first best basis function ψ_0 and also to evaluate the approximation $X_{\mathcal{Q}_0}$ associated to partition $\mathcal{Q}_0$.
Set $\mathcal{Q}_0$ to be the trivial partition $\mathcal{Q}_0 = \{\Omega, \emptyset\}$ it will be useful to set $A_{0,0} \equiv \Omega$. We need to introduce the vector valued constant function to be used for the first term in our approximations, so let's set

$$\psi_0(\omega) = c\ \mathbf{1}_{\mathbf{\Omega}}(\omega) \qquad \text{where } c \in S^d \text{ and } \mathbf{1}_{\mathbf{\Omega}}(\omega) = 1 \text{ for all } \omega \in \Omega.$$

As shown in formula (3.9) in order to construct the best basis function ψ_0 we are interested in finding such $c = \{c_1, \ldots, c_d\}$ that maximizes the inner product $\left|\lambda_0\right| = \left|[X, \psi_0]\right|$. Then:

$$
\begin{aligned}
[X, \psi_0] &= \int_\Omega \langle X(\omega), \psi_0(\omega) \rangle \, dP(\omega) \\
&= \int_\Omega \langle X(\omega), c \, \mathbf{1}_\Omega(\omega) \rangle \, dP(\omega) \\
&= \int_\Omega \sum_{k=1}^{d} (c_k X_k(\omega)) dP(\omega) \\
&= \sum_{k=1}^{d} \left(c_k \cdot \int_\Omega X_k(\omega) dP(\omega) \right) .
\end{aligned}
$$

Since we are building an orthonormal basis we require that $||\psi_0||^2 = 1$, therefore:

$$
\begin{aligned}
1 = ||\psi_0||^2 = [\psi_0, \psi_0] &= \int_\Omega \langle \psi_0(\omega), \psi_0(\omega) \rangle \, dP(\omega) \\
&= \int_\Omega \langle c \, \mathbf{1}_\Omega(\omega), c \mathbf{1}_\Omega(\omega) \rangle dP(\omega) \\
&= \int_\Omega \Big(\sum_{k=1}^{d} c_k^2 \Big) dP(\omega) \\
&= \sum_{k=1}^{d} \Big(c_k^2 \cdot \int_\Omega dP(\omega) \Big) \\
&= \sum_{k=1}^{d} c_k^2 \, .
\end{aligned}
$$

Now we use the method of Lagrange multipliers to find maximum of a scalar function of d variables

$f(c_1, \ldots, c_d) = \sum_{k=1}^{d} \Big(c_k \cdot \int_\Omega X_k(\omega) dP(\omega) \Big)$, subject to the constraint

$g(c_1, \ldots, c_d) = \sum_{k=1}^{d} c_k^2 = 1.$

The Lagrange Method consists of solving d equations of partial derivatives in the form $\nabla f(c_1, \ldots, c_d) = \mu \cdot \nabla g(c_1, \ldots, c_d)$, in other words $\dfrac{\partial f}{\partial c_i} = \mu \dfrac{\partial g}{\partial c_i}$, where the parameter μ is called the Lagrange multiplier. So for the $i-th$ coordinate we obtain the partial

derivative

$$\frac{\partial f}{\partial c_i} = \int_\Omega X_i(\omega) dP(\omega) = \mu \frac{\partial g}{\partial c_i} = 2\mu c_i \ .$$

From here we see that $c_i = \dfrac{\int_\Omega X_i(\omega) dP(\omega)}{2\mu}$. Now we use our constraint to derive μ:

$$\sum_{k=1}^{d} c_k^2 = 1 = \frac{\displaystyle\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}{(2\mu)^2} \ ,$$

i.e. $2\mu = \pm \sqrt{\displaystyle\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}$. Using the equation for 2μ we obtain for c_i:

$$c_i = \pm \frac{\displaystyle\int_\Omega X_i(\omega) dP(\omega)}{\sqrt{\displaystyle\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}} \ .$$

From here our inner product and single term approximation are given by

$$\begin{aligned}
[X, \psi_0] &= [X, c] = \int_\Omega \big\langle c, X(\omega) \big\rangle dP(\omega) \\
&= \int_\Omega \Big(\sum_{k=1}^{d} c_k X_k(\omega) \Big) dP(\omega) \\
&= \sum_{k=1}^{d} \Big(c_k \int_\Omega X_k(\omega) dP(\omega) \Big) \\
&= \pm \frac{\displaystyle\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}{\sqrt{\displaystyle\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}} \\
&= \pm \sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2} \ .
\end{aligned}$$

The first approximation of input vector X

$$X_{\mathcal{Q}_0}(\omega) = [X, \psi_0] \cdot \psi_0 = [X, c] \cdot c = \pm\, c \cdot \sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2} \,,$$

and its $i-th$ component:

$$\begin{aligned} \{X_{\mathcal{Q}_0}(\omega)\}_i &= \{[X, \psi_0] \cdot \psi_0\}_i = \pm c_i \sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2} \\ &= \frac{\int_\Omega X_i(\omega) dP(\omega)}{\sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}} \cdot \sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2} \\ &= \int_\Omega X_i(\omega) dP(\omega) \,. \end{aligned}$$

Summarizing the results we got so far, the first basis ψ_0 function and the corresponding inner product $[X, \psi_0]$ and the approximation $[X, \psi_0]\psi_0$ have the following form:

$$\begin{aligned} &\psi_0(\omega) \equiv c = \{c_1, \ldots, c_d\}, \quad \text{where} \quad c_i = \pm \frac{\int_\Omega X_i(\omega) dP(\omega)}{\sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2}} \,, \\ &\lambda_0 = [X, \psi_0] = \pm \sqrt{\sum_{k=1}^{d} \Big(\int_\Omega X_k(\omega) dP(\omega) \Big)^2} \,, \\ &\{[X, \psi_0]\psi_0\}_i = \{X_{\mathcal{Q}_0}\}_i = \int_\Omega X_i(\omega) dP(\omega) \,. \end{aligned} \tag{B.1}$$

Remark 1. *For the scalar case ($d = 1$) the formula (B.1) simplifies to:*

$$\begin{aligned} &\psi_0(\omega) \equiv \pm 1 \,, \\ &\lambda_0 = [X, \psi_0] = \pm \int_\Omega X(\omega) dP(\omega) \,, \\ &X_{\mathcal{Q}_0} = [X, \psi_0]\psi_0 = \int_\Omega X(\omega) dP(\omega) \,. \end{aligned} \tag{B.2}$$

B.2 Best Split and its Properties

We prove here the main results for the best split step. These results are used in the `bestSplit` procedure of the VGS algorithm described in Section 3.2.2.

Suppose $A \subseteq \Omega$ may be presented as the union of two disjoint subsets A_0 and A_1 with positive probabilities, as defined by Lemma 1. Using formula (A.1) we are looking at the inner product $[X, \psi_{b',A}]$:

$$\begin{aligned} \pm [X, \psi_{b',A}] &= \int_\Omega \langle\, X(\omega), \psi_{b',A}(\omega) \,\rangle\, dP(\omega) \\ &= \int_\Omega \left\langle X,\ b' \left(\sqrt{\tfrac{P(A_1)}{P(A_0)P(A)}} \mathbf{1}_{A_0} - \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \mathbf{1}_{A_1} \right) \right\rangle dP \\ &= \int_\Omega \left(\sqrt{\tfrac{P(A_1)}{P(A_0)P(A)}} \mathbf{1}_{A_0} - \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \mathbf{1}_{A_1} \right) \langle\, X,\ b' \rangle\, dP \\ &= \sqrt{\tfrac{P(A_1)}{P(A_0)P(A)}} \int_\Omega \mathbf{1}_{A_0} \langle X,\ b' \rangle dP - \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \int_\Omega \mathbf{1}_{A_1} \langle X,\ b' \rangle\, dP \\ &= \sqrt{\tfrac{P(A_1)}{P(A_0)P(A)}} \int_{A_0} \langle X,\ b' \rangle dP - \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \int_{A_1} \langle X,\ b' \rangle\, dP \\ &= \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \left(\tfrac{P(A_1)}{P(A_0)} \int_{A_0} \langle X,\ b' \rangle\, dP - \int_{A_1} \langle X,\ b' \rangle dP \right) \\ &= \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \left(\left(1 + \tfrac{P(A_1)}{P(A_0)} \right) \int_{A_0} \langle X,\ b' \rangle\, dP - \int_A \langle X,\ b' \rangle dP \right) \\ &= \sqrt{\tfrac{P(A_0)}{P(A_1)P(A)}} \left(\tfrac{P(A)}{P(A_0)} \int_{A_0} \langle X,\ b' \rangle\, dP - \tfrac{P(A)}{P(A)} \int_A \langle X,\ b' \rangle dP \right) . \end{aligned}$$

Therefore, we have obtained the following formula:

$$\pm [X, \psi_{b',A}] = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \left(\frac{1}{P(A)} \int_A \langle X, b' \rangle \, dP - \frac{1}{P(A_0)} \int_{A_0} \langle X, b' \rangle dP \right) . \quad \text{(B.3)}$$

We will show how to solve the maximization problem for $\left| [X, \psi_{b',A}] \right|$.

B.2.1 Best Split for fixed partition

Suppose $A \subseteq \Omega$ may be presented as union of two disjoint subsets A_0 and A_1 with positive probabilities, as defined by Lemma 1. We will show how to apply the technique of Lagrange multipliers to solve the given optimization problem.

Lemma 2. *Suppose we are given set A and its partition as defined in Lemma 1. Then for any given fixed partition the solution to finding the extremum $\sup_{b' \in S^d} \left| [X, \psi_{A,b'}] \right|$ is realized by the parameter $\pm\, b' = (b_1, \ldots, b_d) \in S^d$ whose $i-th$ coordinate is*

$$b'_i = \frac{\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP}{\sqrt{\sum_{k=1}^{d} \left(\frac{1}{P(A)} \int_A X_k dP - \frac{1}{P(A_0)} \int_{A_0} X_k dP \right)^2}} . \quad \text{(B.4)}$$

Proof. Let's rewrite the last equation by exploring inner products through summations of vector components:

$$\begin{aligned} \pm [X, \psi_{b'}] &= \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \left(\frac{1}{P(A)} \int_A \sum_{i=1}^{d} (X_i b'_i) dP - \frac{1}{P(A_0)} \int_{A_0} \sum_{i=1}^{d} (X_i b'_i) dP \right) \\ &= \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sum_{i=1}^{d} \left[b'_i \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) \right] . \end{aligned}$$

We are interested in finding such $\psi_{A,b'}$ that maximizing absolute value of the inner product $[X, \psi_{A,b'}]$. For the given partition $A = A_0 \cup A_1$ the function $\psi_{A,b'}$ depends on the vector b' only. We will use the method of Lagrange multipliers, like we did for the trivial partition and first basis function ψ_0. So we want to maximize the function $f(b')$ defined by

$$f(b'_1, b'_2, .., b'_d) = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sum_{i=1}^{d} \left[b'_i \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) \right] ,$$

where the restraint function $g(b')$ is

$$g(b'_1, b'_2, .., b'_d) = \sum_{i=1}^{d} (b'_i)^2 = 1 \, .$$

Since $\frac{\partial f}{\partial b'_i} = \mu \cdot \frac{\partial g}{\partial b'_i}$ we get:

$$\frac{\partial f}{\partial b'_i} = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) = \mu \cdot \frac{\partial g}{\partial b'_i} = 2 \, \mu \, b'_i \, .$$

Therefore we may express b'_i as

$$b'_i = \frac{1}{2\mu} \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) \, . \tag{B.5}$$

From our constraint and the last formula we may extract an equation for $(2\mu)^2$:

$$1 = \sum_{i=1}^{d} (b'_i)^2 = \frac{1}{(2\mu)^2} \frac{P(A)P(A_0)}{P(A_1)} \sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2 \, .$$

From here we are getting 2μ:

$$2\mu = \pm \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \cdot \int_{A_0} X_i dP \right)^2} \, . \tag{B.6}$$

Substitution of the expression for 2μ from (B.6) into (B.5) gives us that maximizing of $\left| [X, \psi_{b'}] \right|$ is achieved with the parameter $b' \in S^d$ defined by:

$$\pm b' = (b_1, \ldots, b_d), \quad \text{where} \quad b'_i = \frac{\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP}{\sqrt{\sum_{k=1}^{d} \left(\frac{1}{P(A)} \int_A X_k dP - \frac{1}{P(A_0)} \int_{A_0} X_k dP \right)^2}} \, .$$

□

From these results we may express inner product:

$$\pm [X, \psi_{A,b'}] = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sum_{i=1}^{d} \left[b'_i \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) \right]$$

$$= \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sum_{i=1}^{d} \frac{\left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2}{\sqrt{\sum_{k=1}^{d} \left(\frac{1}{P(A)} \int_A X_k dP - \frac{1}{P(A_0)} \int_{A_0} X_k dP \right)^2}}$$

$$= \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2}.$$

It is clear that the problem of finding the maximum for $\left| [X, \psi_{A,b'}] \right|$ is equivalent to the problem of maximization of $([X, \psi_{A,b'}])^2$:

$$([X, \psi_{A,b'}])^2 = \frac{P(A)P(A_0)}{P(A_1)} \sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2. \qquad \text{(B.7)}$$

Using the derived formulae for the vector Haar function $\psi_{A,b'}$ and the corresponding inner product $[X, \psi_{b'}]$ we obtain the approximation of X over set A:

$$[X, \psi_{A,b'}] \psi_{A,b'} =$$

$$\sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2} *$$

$$* \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}} \mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \mathbf{1}_{A_1} \right) b'.$$

And the same formula after simplification

$$[X, \psi_{A,b'}] \psi_{A,b'} = \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2} \cdot \left(\mathbf{1}_{A_0} - \frac{P(A_0)}{P(A_1)} \mathbf{1}_{A_1} \right) b'. \qquad \text{(B.8)}$$

From formula (B.8) the $i-th$ component of the vector $[X, \psi_{A,b'}]\,\psi_{A,b'}$ is

$$([X, \psi_{A,b'}]\,\psi_{A,b'})_i =$$

$$= \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2} \cdot \left(\mathbf{1}_{A_0} - \frac{P(A_0)}{P(A_1)}\, \mathbf{1}_{A_1} \right) b'_i =$$

$$= \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2} \cdot \left(\mathbf{1}_{A_0} - \frac{P(A_0)}{P(A_1)}\, \mathbf{1}_{A_1} \right) *$$

$$* \frac{\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP}{\sqrt{\sum_{k=1}^{d} \left(\frac{1}{P(A)} \int_A X_k dP - \frac{1}{P(A_0)} \int_{A_0} X_k dP \right)^2}} \cdot$$

Or

$$\begin{aligned}
([X, \psi_{A,b'}]\,\psi_{A,b'})_i &= \left(\mathbf{1}_{A_0} - \frac{P(A_0)}{P(A_1)}\, \mathbf{1}_{A_1} \right) \cdot \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) \\
&= \left(\mathbf{1}_{A_0} - \frac{P(A_0)}{P(A_1)}\, (\mathbf{1}_A - \mathbf{1}_{A_0}) \right) \cdot \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) \\
&= \left(\frac{P(A)}{P(A_1)} \mathbf{1}_{A_0} - \frac{P(A_0)}{P(A_1)}\, \mathbf{1}_A \right) \cdot \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right) .
\end{aligned}$$

Summarizing the results, we rewrite the formula for the best Haar function $\psi_{A,b'}$ and

related inner product $[X, \psi_{A,b'}]$ when the partition $\{A_0, A_1\}$ of $A = A_0 \cup A_1$ is given:

$$\pm\psi_{A,b'} = \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\, \mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\, \mathbf{1}_{A_1} \right) b', \text{ where}$$

$$b' = (b_1, \ldots, b_d), \text{ s.t. } b'_i = \frac{\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP}{\sqrt{\sum_{k=1}^{d} \left(\frac{1}{P(A)} \int_A X_k dP - \frac{1}{P(A_0)} \int_{A_0} X_k dP \right)^2}}, \tag{B.9}$$

$$\pm [X, \psi_{b'}] = \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \sqrt{\sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2}.$$

In particular, the last formula has some interest for the scalar case. In order to show it let us denote $\lambda = [X, \psi_{b'}]$. then $\lambda^2 = \left| [X, \psi_{b'}] \right|^2$. From our last formula immediate value for λ^2 is:

$$\lambda^2 = [X, \psi_{b'}]^2 = \frac{P(A)P(A_0)}{P(A_1)} \sum_{i=1}^{d} \left(\frac{1}{P(A)} \int_A X_i dP - \frac{1}{P(A_0)} \int_{A_0} X_i dP \right)^2. \tag{B.10}$$

Now suppose that we are building scalar Haar functions for components X_i of vector random variable $X = (X_1, \ldots, X_d)$ for the same partition $\{A_0, A_1\}$ of $A = A_0 \cup A_1$. As we had shown in formula (A.2) the scalar Haar function is defined by partition

only and we get for the inner product:

$$
\begin{aligned}
\pm[X_i, \psi_A] &= \int_\Omega X_i(\omega) \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}}\, \mathbf{1}_{A_0} - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}}\, \mathbf{1}_{A_1} \right) dP(\omega) \\
&= \sqrt{\frac{P(A_1)}{P(A)P(A_0)}} \int_{A_0} X_i(\omega) - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \int_{A_1} X_i(\omega) dP(\omega) \\
&= \left(\sqrt{\frac{P(A_1)}{P(A)P(A_0)}} + \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \right) \int_{A_0} X_i(\omega) dP(\omega) \\
&\quad - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \int_A X_i(\omega) dP(\omega) \\
&= \left(\frac{P(A_1)}{\sqrt{P(A)P(A_0)P(A_1)}} + \frac{P(A_0)}{\sqrt{P(A)P(A_1)P(A_0)}} \right) \int_{A_0} X_i(\omega) dP(\omega) \\
&\quad - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \int_A X_i(\omega) dP(\omega) \\
&= \sqrt{\frac{P(A)}{P(A_0)P(A_1)}} \int_{A_0} X_i(\omega) dP(\omega) - \sqrt{\frac{P(A_0)}{P(A)P(A_1)}} \int_A X_i(\omega) dP(\omega) \\
&= \sqrt{\frac{P(A)P(A_0)}{P(A_1)}} \left(\frac{1}{P(A_0)} \int_{A_0} X_i(\omega) dP(\omega) - \frac{1}{P(A)} \int_A X_i(\omega) dP(\omega) \right) .
\end{aligned}
\tag{B.11}
$$

From here we obtain equation for λ_i^2, the square of inner product for each scalar component X_i:

$$
\lambda_i^2 = [X_i, \psi_A]^2 = \frac{P(A)P(A_0)}{P(A_1)} \left(\frac{1}{P(A_0)} \int_{A_0} X_i dP - \frac{1}{P(A)} \int_A X_i dP \right)^2 . \tag{B.12}
$$

From formulae (B.10) and (B.12) follows the relation between them:

$$
\lambda^2 = \sum_{i=1}^{d} \lambda_i^2, \qquad \text{i.e.} \qquad \Big| [X, \psi_{b'}] \Big|^2 = \sum_{i=1}^{d} \Big| [X_i, \psi_A] \Big|^2 . \tag{B.13}
$$

Corollary 1. *From formula (B.13) it follows that the decay of the sequence of values* $\{|\lambda_1|, |\lambda_2|, \ldots, |\lambda_d|\}$, *created by the VGS algorithm applied to the of vector random*

variable $X = \{X_1, \ldots, X_d\}$, *causes a corresponding decay of the sequence of values* $\{|\lambda_{i,1}|, |\lambda_{i,2}|, \ldots, |\lambda_{i,d}|\}$, *obtained by computing the inner products for each input signal* X_i *using the same partition tree.*

B.2.2 Best Split for fixed b' - Bathtub theorem

The bathtub theorem provides us with an effective mechanism for the fast software implementation of the Greedy Splitting algorithm. We present the bathtub theorem in a simplified (discrete) form corresponding to the practical usage in our case. The problem is to find solution for optimization problem

$$I = \mathrm{opt}_{\varphi \in \mathcal{C}} \langle X, \varphi \rangle, \quad \text{where } \mathcal{C} = \{\varphi : 0 \le \varphi \le 1,\ E(\varphi) = u\}\ ,$$

where the expectation is defined by $E(\varphi) = \int_\Omega \varphi(\omega) dP(\omega)$.

We start with a simple lemma.

Lemma 3. *Suppose,* ψ *is a function, satisfying* $E(\psi) = 0$, $E(\psi^2) = 1$.
Then there exist two functions φ_1 *and* φ_2, *such that*
$\psi = \varphi_1 - \varphi_2$, *and* $\varphi_1 \ge 0$, $\varphi_2 \ge 0$, *and expectations*
$E(\varphi_1) = E(\varphi_2) = u$, *while* $E(\varphi_1 \cdot \varphi_2) = 0$.

Proof. Let's present our space Ω as union of two disjoint sets:

$$\begin{aligned} \Omega &= A_+ \cup A_-, \quad \text{where} \\ A_+ &= \{\omega \in \Omega : \psi(\omega) \ge 0\}, \\ A_- &= \{\omega \in \Omega : \psi(\omega) < 0\}. \end{aligned}$$

Let's define functions φ_1 and φ_2 as

$$\varphi_1(\omega) = \begin{cases} \psi(\omega), & \omega \in A_+ \\ 0, & \omega \in A_- \end{cases},$$

$$\varphi_2(\omega) = \begin{cases} 0, & \omega \in A_+ \\ -\psi(\omega), & \omega \in A_- \end{cases}.$$

From the definitions above immediately follows that

$$\varphi_1 \ge 0, \qquad \varphi_2 \ge 0, \qquad \psi = \varphi_1 - \varphi_2 \quad .$$

And for the expectations we get:

$$\begin{aligned} 0 = E(\psi) &= \int_\Omega \psi dP = \int_{A_+} \psi dP + \int_{A_-} \psi dP \\ &= \int_{A_+} \varphi_1 dP - \int_{A_-} \varphi_2 dP = \int_\Omega \varphi_1 dP - \int_\Omega \varphi_2 dP \\ &= E(\varphi_1) - E(\varphi_2)\ . \end{aligned}$$

Therefore, $E(\varphi_1) = E(\varphi_2) = u$. And for the multiplication $\varphi_1\varphi_2$:

$$E(\varphi_1\varphi_2) = \int_\Omega \varphi_1\varphi_2 dP = \int_{A_+} \varphi_1\varphi_2 dP + \int_{A_-} \varphi_1\varphi_2 dP = 0 \,.$$

And this concludes the proof. □

From Lemma 3 follows

Corollary 2. *If* $E(\varphi_1^2) = v_1$ *then* $E(\varphi_2^2) = 1 - v_1$.

Proof.

$$\begin{aligned} 1 = E(\psi^2) &= E\big((\varphi_1 - \varphi_2)\big) \\ &= \textstyle\int_{A_+} \varphi_1 dP - \int_{A_-} \varphi_2 dP = \int_\Omega \varphi_1 dP - \int_\Omega \varphi_2 dP \\ &= E(\varphi_1) - E(\varphi_2) \,. \end{aligned}$$

□

Using the previous lemma we find now expression for each φ_i using the following theorem, presenting discrete case of the Bathtub principle.

Theorem 1. *Suppose, we have discrete space* $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$, *where* $P(\omega_i) > 0$. *Then the solution to optimization problem* $opt_{\varphi \in C}\langle X, \varphi\rangle$, $C = \{\varphi : 0 \leq \varphi \leq 1,\ E(\varphi) = u$ *is done by function* φ, *taking all its values, but one at most, from the set* $\{0, 1\}$.

Proof. We use the following notation:

$$\begin{aligned} \varphi(\omega_i) &= x_i \,, \\ X(\omega_i) &= b_i \,, \\ P(\omega_i) &= a_i \,. \end{aligned}$$

Using these equations we obtain the inner product and mean value:

$$\begin{aligned} f = \langle X, \varphi \rangle &= \int_\Omega X(\omega)\varphi(\omega)dP(\omega) \\ &= \sum_{i=1}^{n} X(\omega_i)\varphi(\omega_i)P(\omega_i) \\ &= \sum_{i=1}^{n} a_i b_i x_i \ . \end{aligned}$$

$$\begin{aligned} E(\varphi) = u &= \int_\Omega \varphi(\omega)dP(\omega) \\ &= \sum_{i=1}^{n} \varphi(\omega_i)P(\omega_i) \\ &= \sum_{i=1}^{n} a_i x_i \ . \end{aligned}$$

So we are looking for $optf$ under constraints $g = \sum_i^n a_i x_i - u$, $0 \leq x_i \leq 1$. Assume, without loss of generality, that $b_1 \leq b_2 \leq \ldots \leq b_n$. We'll use method of Lagrange multipliers, defining Lagrangian as

$$L(x) = f(x) - \lambda g(x) = \sum_{i=1}^{n} a_i b_i x_i - \lambda \sum_{i=1}^{n} a_i x_i - \lambda u \ .$$

Using the fact that function $h(x) = \cos^2(t)$ is $1:1$ on segment $[0,1] \subseteq [0, \pi/2]$, we may change variables for our Lagrangian by $x = \cos^2(y)$. Then our Lagrangian becomes

$$L(y) = \sum_{i=1}^{n} a_i b_i y_i^2 - \lambda \sum_{i=1}^{n} a_i y_i^2 - \lambda u \ .$$

In according to the method of Lagrangian multipliers we require $\bigtriangledown L(y) = 0$, or, expressed in partial derivatives:

$$\begin{aligned} 0 = \frac{\partial L}{\partial y_i} = \frac{\partial f}{\partial y_i} - \lambda \frac{\partial f}{\partial y_i} &= -2a_i b_i \cos y_i \sin y_i + 2\lambda a_i \sin y_i \cos y_i \\ &= -2a_i \sin y_i \cos y_i \cdot (b_i - \lambda), \quad \forall\, i = 1, \ldots, n \ . \end{aligned}$$

Since $a_i = P(\omega_i) > 0$, we must satisfy $\lambda = b_i$ or $\sin y_i = 0$ or $\cos y_i = 0$. The last two conditions on our domain are equivalent to $\sin^2 y_i = 0$ and $\cos^2 y_i = 0$, i.e. $x_i = 0$ and $1 - x_i = 0$. Therefore:

$$x_i = 0 \quad \text{or} \quad x_i = 1 \quad \text{or} \quad \lambda = b_i \; . \tag{B.14}$$

From another side

$$u = \sum_i^n a_i x_i \leq \sum_i^n a_i \; . \tag{B.15}$$

Since a_i are positive and $\sum_i^n a_i \geq u$ follows that exists $k \in \{1, 2, \ldots, n-1\}$, such that

$$\sum_i^k a_i \leq u \quad \text{and} \quad \sum_1^{k+1} a_i \geq u \; .$$

Let's define solution to our optimization problem as

$$\begin{cases} \lambda = b_{k+1} \\ x_1 = x_2 = \ldots = x_k = 1 \\ x_{k+1} = \dfrac{u - \sum_1^k a_i}{a_{k+1}} \\ x_{k+2} = x_{k+3} = \ldots = x_n = 0 \end{cases}$$

In order to show that all constraints are satisfied we have to check that $0 \leq x_{k+1} \leq 1$. Since $a_{k+1} > 0$ and $u - \sum_1^k a_i \geq 0$ follows that $x_{k+1} \geq 0$. Also $\sum_1^{k+1} a_i \geq u$, hence

$$x_{k+1} = \frac{u - \sum_1^k a_i}{a_{k+1}} = \frac{u + a_{k+1} - \sum_1^{k+1} a_i}{a_{k+1}} = 1 - \frac{\sum_1^{k+1} a_i - u}{a_{k+1}} \leq 1 \; .$$

Therefore all constraints and met and we have completed the proof.

□

B.2.3 Software implementation of bestSplit iteration

We present here software implementation of bestSplit iteration written in ANSI C computer language. BestSplit implementation uses Bathtub principle:

```
//*************************************************************************
//***
//*** BESTSPLIT   O F   L I N E A R   G R E E D Y   A L G O R I T H M
//***
//*** Description:
//***    This function performs split of domain A in 2 domains A0 and A1 in
//***     according to the linear splitting algorithm.
//***
//*** Input:1) pBS_tree_node - pointer to the node of BS_tree;
//***       2) sp    - pointer to the buffer of P(A0)
//***       3) sh    - pointer to the buffer of integrals S_A0(X(W)*dP(w))
//***       4) NF    - # of samples in input signals
//***
//*** Output: return values are written in corresponding struct of BS_tree[]
//***
//*************************************************************************
void best_split(BS_NODE *ppBS_tree_node,  unsigned __int32 *sp,
                unsigned __int32 *sh, unsigned int NF)
{
    unsigned __int32 PA;
    unsigned __int32 IA;
    unsigned __int32 PA0;
    unsigned __int32 IA0;
    double           TempLambda;
    unsigned int     k;

    ppBS_tree_node->kmax= ppBS_tree_node->v0;
    ppBS_tree_node->Lambda_pow2 = 0;
    if(ppBS_tree_node->v0 == ppBS_tree_node->v1) {
        ppBS_tree_node->kmax= ppBS_tree_node->v0;
        return;
    }
    else {
        PA=sp[ppBS_tree_node->v1 +1] - sp[ppBS_tree_node->v0];//PA=NF*P(A)
        IA=sh[ppBS_tree_node->v1 +1] - sh[ppBS_tree_node->v0];//IA=NF*S_A
                                                              //(X(w)*dP(w))
        for( k = ppBS_tree_node->v0; k <= ppBS_tree_node->v1; k++) {
```

```
            PA0 = sp[k +1] - sp[ppBS_tree_node->v0]; //PA0=NF*P(A0)
            IA0 = sh[k +1] - sh[ppBS_tree_node->v0]; //IA0=NF*S_A0
                                                     // (X(w)*dP(w))
            if( (PA0 != 0) && (PA != 0) && (PA0 != PA) ) {
//-------------------------------------------------------------------
//--            __________  /      _                    _       \
//--           / P(A)P(A0)  |  1   |              1     |         |
//--lambda=\  /  --------- *| ---  |X(w)dP(w) - ----   |X(w)dP(w)|==[X,Psi]
//--        \/   P(A)-P(A0) | P(A)_|            P(A0)_|         |
//--                        \    A                  A0        /
//-------------------------------------------------------------------
//-------------------------------------------------------------------
//--                                  /     _                 _       \ 2
//--       2           1             |     |                 |         |
//-- lambda = -------------------- *|P(A0)|X(w)dP(w) - P(A)|X(w)dP(w)|
//--          P(A)P(A0)(P(A)-P(A0))  |    _|               _|         |
//--                                  \   A               A0        /
//-------------------------------------------------------------------
   TempLambda=(float)PA*PA0/((PA-PA0)*NF)*(IA/PA-IA0/PA0)*(IA/PA-IA0/PA0);
        }
        else {
            TempLambda = 0;
        }
        if(TempLambda > ppBS_tree_node->Lambda_pow2) {
         ppBS_tree_node->Lambda_pow2 = TempLambda;
         ppBS_tree_node->kmax = k; //next partition [v0,kmax] [kmax+1,v1]
        }
      }         // end of for()  - bathtub theorem loop
    }
}
//***   end of best_split()
//*********************************************************************
```

B.3 Recursive Multiresolution Analysis

We prove here formulae for the recursive Multiresolution Analysis (MRA) of the partition tree starting from the leaves (down-top recursion). We perform the same task for each component signal X_i of vector input $X = \{X_1, \ldots, X_d\}$. Suppose we

are dealing with signal X_i and we denote for each node A of the partition tree:

$$\begin{cases} p_{nodeA} = P(A) \ , \\ y_{nodeA} = \frac{1}{P(A)} \int_A X_i(w) dP(w) \ , \\ d_{nodeA} = \langle X_i, \psi_{i,nodeA} \rangle . \end{cases}$$

For each leaf of the tree we know quantities p_{nodeA} and y_{nodeA}. The MRA comprises calculation of all inner products d_{nodeA} for all the nodes of the tree per each signal component $X - i$. We'll use subscripts F, L and R for father node and its left and right children. Suppose that we know p_L, p_R, y_L and y_R of children. Here are recursive formulae to calculate p_F, y_F and d_F:

$$\begin{cases} p_F = p_L + p_R \ , \\ y_F = \dfrac{1}{p_F}(p_L y_L + p_R y_R) \ , \\ d_F = \sqrt{\dfrac{p_L \ p_R}{p_F}} \ (y_L - y_R) \ . \end{cases} \tag{B.16}$$

We prove these formulae.

Since $A_F = A_L \bigcup A_R$, and $A_L \bigcap A_R = \emptyset$, hence $p_F = p_L + p_R$. For y_F:

$$\begin{aligned} p_L y_L + p_R y_R &= p_L \frac{1}{p_L} \int_{A_L} X_i(w) dP(w) + p_R \frac{1}{p_R} \int_{A_R} X_i(w) dP(w) \\ &= \int_{A_L} X_i(w) dP(w) + \int_{A_R} X_i(w) dP(w) = \int_{A_L \bigcup A_R} X_i(w) dP(w) \\ &= \int_{A_F} X_i(w) dP(w) = y_F \cdot p_F \ . \end{aligned}$$

And, finally, for inner product:

$$
\begin{aligned}
d_F = \langle X_i, \psi_{i,A_F} \rangle &= \int_\Omega X_i(w)\psi_{i,A_F} dP(w) \\
&= \int_\Omega X_i(w)\Big(\sqrt{\frac{p_R}{p_L p_F}}\mathbf{1}_{A_L} - \sqrt{\frac{p_L}{p_R p_F}}\mathbf{1}_{A_R}\Big)(w) dP(w) \\
&= \int_{A_L} X_i(w)\sqrt{\frac{p_R}{p_L p_F}} dP(w) - \int_{A_R} X_i(w)\sqrt{\frac{p_L}{p_R p_F}} dP(w) \\
&= \sqrt{\frac{p_R}{p_L p_F}} \cdot p_L\Big(\frac{1}{p_L}\int_{A_L} X_i(w) dP(w)\Big) - \sqrt{\frac{p_L}{p_R p_F}} \cdot p_R\Big(\frac{1}{p_R}\int_{A_R} X_i(w) dP(w)\Big) \\
&= \sqrt{\frac{p_L p_R}{p_F}} y_L - \sqrt{\frac{p_L p_R}{p_F}} y_R = \sqrt{\frac{p_L p_R}{p_F}}(y_L - y_R) \ .
\end{aligned}
$$

And this concludes proof of MRA recursive formulae.

B.4 Recursive Multiresolution Synthesis

We prove here formulae for the recursive synthesis of the partition tree starting from the root (top-down recursion). We perform the same task for each component signal X_i of vector input $X = \{X_1, \ldots, X_d\}$. Suppose we are dealing with signal X_i and we use the same notation as in previous section. The recursive synthesis reconstructs approximation of the signal components by calculating values y_{nodeA} for each leaf of the partition tree. Here are recursive formulae to calculate y_L and y_R for left and right children A_L and A_R of node A_F:

$$
\begin{cases}
y_L = y_F + \sqrt{\dfrac{p_R}{p_F p_L}} d_F \ , \\[2ex]
y_R = y_L - \sqrt{\dfrac{p_F}{p_L p_R}} d_F \ .
\end{cases}
$$

We prove these formulae using formulae (B.16) for recursive MRA. So for the left child we obtain:

$$\begin{aligned} y_F + \sqrt{\frac{p_R}{p_F p_L}} d_F &= \frac{1}{p_F}(p_L y_L + p_R y_R) + \sqrt{\frac{p_R}{p_F p_L}} \cdot \sqrt{\frac{p_L\, p_R}{p_F}}\,(y_L - y_R) \\ &= \frac{1}{p_F}\big(p_L y_L + p_R y_R + p_R y_L - p_R y_R\big) \\ &= \frac{1}{p_F} \cdot y_L\big(p_L + p_R\big) = y_L\,. \end{aligned}$$

And for the right child we obtain:

$$\begin{aligned} y_L - \sqrt{\frac{p_F}{p_L p_R}} d_F &= y_L - \sqrt{\frac{p_F}{p_L p_R}} \sqrt{\frac{p_L\, p_R}{p_F}}\,(y_L - y_R) \\ &= y_R\,. \end{aligned}$$

Appendix C

Uniform distribution on multi-dimensional sphere

The key parameter for the Vector Greedy Splitting Algorithm is the point in unit d-dimensional sphere $b' \in S^d$. Suppose, that in order to build basis for VGS we have to cover unit sphere with N uniformly distributed points. This task is trivial for $d = 2$ only. For $d = 2$ we get unit circle in R^2, that may be easy covered using polar coordinates. In this case we have set of uniformly distributed M points of the form $(x_k, y_k) = (\cos(2\pi k/M), \sin(2\pi k/M))$. For other dimensions the only solution is to find way to distribute uniformly points using probability methods. There are two most efficient and popular methods: Box-Muller transformation (see [2]) and Marsaglia's ziggurat algorithm ([15]). Both methods transform uniform distribution into normal deviate. We use the polar form of the Box-Muller transformation. The whole algorithm consists of the following steps: generation of the two independent random numbers, transformation of these two random numbers into two uniform distributions on $(0, 1)$, polar form of Box-Muller transformation for Gaussian generator, and, finally, projection of vector in R^d Euclidian space into unit sphere S^d. Here is the description of the method.

C.1 Pseudorandom Generator

The first step is the generation of the random numbers. We use 32 bits pseudo random generator. The generator builds sequence $x_0, x_1, \ldots, x_n$ using iterative algorithm:

$$x_n = a \cdot x_{n-1} \, (mod \ m) \tag{C.1}$$

The main parameters of this equation are multiplier a and modulus m. Modulus m must be prime number to avoid generation of zero. Modern computers use 32 bits to

represent integer values. So the value $m = 2^{32} - 1$ is a natural choice for the modulus value. The multiplier a must satisfy $a < m$ and its choice affects the performance of the random number generator as shown in [17].

Once the modulus m ($m = 2^{32} - 1$) and multiplier a ($a < m$) are chosen the Schrage's factorization algorithm is needed in order to avoid 32 bits signed arithmetics computer overflow. Schrage's factorization algorithm is widely used but its proof is not trivial and textbook do not provide it. So we proved the Schrage's algorithm in the following lemma:

Lemma 4. *Suppose we may present natural number m as:*

$$m = aq + r, \quad i.e. \quad q = [m/a], \quad r = m \pmod{a} . \tag{C.2}$$

If $r < q$, the following equation holds for any $0 \leq x < m$:

$$ax \pmod{m} = \begin{cases} a(x \bmod q) - r[x/q] & \text{if it is} \geq 0, \\ a(x \bmod q) - r[x/q] + m & \text{otherwise.} \end{cases} \tag{C.3}$$

Proof. Let's denote $s = [x/q]$ and $t = x \bmod q$, i.e.

$$x = sq + t, \text{ where } 0 \leq t < q .$$

From here:

$$0 \leq at < aq = m - r < m, \text{ i.e. } at < m .$$

From inequities $x < m$ and $t < m$ follows

$$0 \leq rs < qs = x - t < m, \text{ i.e. } rs < m .$$

So $0 < at < m$ and $0 < rs < m$, therefore

$$-m < at - rs < m .$$

From another side

$$at - rs = a(x - sq) - s(m - aq) = ax - ms .$$

And we obtain

$$ax \bmod m = (at - rs) \bmod m$$

From here and the fact that $-m < at - rs < m$ follows that

$$ax \pmod{m} = \begin{cases} at - rs & \text{if it is} \geq 0, \\ at - rs & \text{otherwise.} \end{cases}$$

□

Using Schrage's factorization we avoid a problem of 32 bits arithmetics overflow and the iterative formula for a pseudorandom generator will look as:

$$x_{n+1} = \begin{cases} a(x_n \ mod \ q) - r[x_n/q] & \text{if it is} \geq 0, \\ a(x_n \ mod \ q) - r[x_n/q] \ + \ m & \text{otherwise,} \end{cases} \tag{C.4}$$

where the initial seed $0 < x_0 < m$, and the parameters m and a are modulus and multiplier of the pseudorandom generator correspondingly, and Schrage's factorization parameters r and q are defined as $r = m \ (mod \ a)$ and $q = [m/a]$.

C.2 Gaussian Distribution by Box-Muller Algorithm

In order to build a normal deviate we use Box-Muller algorithm from 1958 (see [2]):

Lemma 5. *Suppose we have two independent uniform distributions x_1 and x_2 on (0,1). Then the following transformation defines two Gaussian deviates y_1 and y_2:*

$$\begin{aligned} y_1 &= \sqrt{-2 \ln x_1} \cos 2\pi x_2 \ , \\ y_2 &= \sqrt{-2 \ln x_1} \sin 2\pi x_2 \ . \end{aligned} \tag{C.5}$$

Proof. From formula (C.5) we get:

$$\begin{aligned} {y_1}^2 + {y_2}^2 &= -2 \ln x_1 \ , \\ y_2/y_1 &= \tan 2\pi x_2 \ . \end{aligned}$$

And from here the inverse functions are:

$$\begin{aligned} x_1 &= e^{-\frac{{y_1}^2 + {y_2}^2}{2}} \ , \\ x_2 &= \tfrac{1}{2\pi} \cdot \arctan \tfrac{y_2}{y_1} \ . \end{aligned}$$

Changing variables for the integral of the probability density function we get the following formula relating joint pdfs $f_{X_1,X_2}(x_1, x_2)$ and $f_{Y_1,Y_2}(y_1, y_2)$:

$$f_{X_1,X_2}(x_1, x_2) = f_{Y_1,Y_2}(y_1, y_2) \cdot \left| \frac{\partial(y_1, y_2)}{\partial(x_1, x_2)} \right| \ .$$

From independency of the uniform distributions x_1 and x_2 we get for a joint pdf:

$$f_{X_1,X_2}(x_1,x_2) = f_{X_1}(x_1) \cdot f_{X_1}(x_1) = 1 \cdot 1 = 1 .$$

We obtain for the absolute value of the Jacobian:

$$\begin{aligned}
\left|\frac{\partial(y_1,y_2)}{\partial(x_1,x_2)}\right| &= \begin{vmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{vmatrix} \\
&= \begin{vmatrix} \frac{1}{2\sqrt{-2\ln x_1}}\frac{-2}{x_1}\cos 2\pi x_2 & -2\pi\sqrt{-2\ln x_1}\sin 2\pi x_2 \\ \frac{1}{2\sqrt{-2\ln x_1}}\frac{-2}{x_1}\sin 2\pi x_2 & 2\pi\sqrt{-2\ln x_1}\cos 2\pi x_2 \end{vmatrix} \\
&= \left|-\frac{2\pi}{x_1}\cos^2(2\pi x_2) - \frac{2\pi}{x_1}\sin^2(2\pi x_2)\right| \\
&= \frac{2\pi}{x_1} .
\end{aligned}$$

And from here

$$\begin{aligned}
f_{Y_1,Y_2}(y_1,y_2) &= \frac{f_{X_1,X_2}(x_1,x_2)}{\left|\frac{\partial(y_1,y_2)}{\partial(x_1,x_2)}\right|} \\
&= \frac{x_1}{2\pi} \\
&= \frac{1}{2\pi}\cdot e^{-\frac{{y_1}^2+{y_2}^2}{2}} \\
&= \frac{1}{\sqrt{2\pi}}e^{-\frac{{y_1}^2}{2}} \cdot \frac{1}{\sqrt{2\pi}}e^{-\frac{{y_2}^2}{2}} .
\end{aligned}$$

Therefore, random variables y_1 and y_2 are independent and their probability density

functions are

$$f_{Y_1}(y_1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{{y_1}^2}{2}} ,$$

$$f_{Y_2}(y_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{{y_2}^2}{2}} .$$

Hence y_1 and y_2 are Gaussian random variables. □

C.3 Polar Coordinates for Box-Muller Method

Looking back to formula (C.5) we may use polar coordinates in order to avoid using of trigonometric functions in uniform to normal deviates Box-Muller transformation. Let's take point (v_1, v_2) inside unit circle on plane, i.e. ${v_1}^2 + {v_1}^2 \leq 1$. Let's define polar coordinates by:

$$\begin{aligned} x_1 &= {v_1}^2 + {v_2}^2 , \\ \tan(2\pi x_2) &= v_2/v_1 . \end{aligned} \tag{C.6}$$

From here

$$\begin{aligned} \sin(2\pi x_2) &= \frac{v_2}{\sqrt{v_1^2 + v_2^2}} , \\ \cos(2\pi x_2) &= \frac{v_1}{\sqrt{v_1^2 + v_2^2}} . \end{aligned} \tag{C.7}$$

Defined this way the polar coordinates x_1 and x_2 are uniform random variables satisfying requirements for Box-Muller method of generating normal deviates. Using these polar coordinates we'll get free of trigonometric functions expressions for normal random variables y_1 and y_2:

$$\begin{aligned} y_1 &= \sqrt{-2\ln(v_1^2 + v_2^2)} \frac{v_1}{\sqrt{v_1^2 + v_2^2}} = v_1 \cdot \sqrt{\frac{-2\ln(v_1^2 + v_2^2)}{v_1^2 + v_2^2}} , \\ y_2 &= \sqrt{-2\ln(v_1^2 + v_2^2)} \frac{v_2}{\sqrt{v_1^2 + v_2^2}} = v_2 \cdot \sqrt{\frac{-2\ln(v_1^2 + v_2^2)}{v_1^2 + v_2^2}} , \end{aligned} \tag{C.8}$$

C.4 Sphere Uniform Distribution Algorithm

Summarizing the results above we present fast and efficient algorithm for the uniform distribution of points on unit sphere S^d.

1. Initial seeds $s_0^{(0)}, \ldots, s_0^{(2d-1)}$ for pseudorandom generator with arbitrary values, s.t.
$$0 < s_0^{(i)} < m, \quad \text{and} \quad s_0^{(i)} \neq s_0^{(j)} ,$$
where $m = 2^{31} - 1$ is modulus of algorithm. Also set to zeros flags $f_0, \ldots, f_{d-1}$. Set $i = 0$ and run steps (2) to (7) M times where M is a required number of vectors distributed on the sphere.

2. If flag $f_i == 1$ go to the step (6), otherwise go to the step (3).

3. Generate two next random values $s_{n+1}^{(2i)}$ and $s_{n+1}^{(2i+1)}$ using formula (C.4), i.e.:
$$s_{n+1} = \begin{cases} a(s_n \ mod \ q) - r[s_n/q] & \text{if it is} \geq 0, \\ a(s_n \ mod \ q) - r[s_n/q] \ + \ m & \text{otherwise.} \end{cases}$$
Save received seeds values for the next iteration and go to the step(4).

4. Transform received random values from the step (3) into uniform random variables v_1 and v_2 from (-1, 1) using the following formula
$$v_i = (s_i/m - 0.5) \cdot 2 .$$
If $v_1^2 + v_2^2 \leq 1$ goto the step (5), otherwise reject samples and return to the step (3).

5. Build two independent Gaussian random samples y_1 and y_2:
$$y_1 = v_1 \cdot \sqrt{\frac{-2\ln(v_1^2 + v_2^2)}{v_1^2 + v_2^2}} ,$$
$$y_2 = v_2 \cdot \sqrt{\frac{-2\ln(v_1^2 + v_2^2)}{v_1^2 + v_2^2}} .$$
Go to the step (6).

6. If the flag $f_i == 1$ set $b_i = y_2$, reset flag $f_i = 0$ and go to the step (7).
   If the flag $f_i == 0$ set $b_i = y_1$, save y_2 set flag $f_i = 0$ and go to the step (7).

7. Increment index i.
   If $i == d$ calculate the random vector $b' \in S^d$ by normalization:

$$b' = \left(b'_0, b'_1, \ldots, b'_{d-1}\right), \quad \text{where } b'_k = \frac{b_k}{\sqrt{b_0^2 + \ldots + b_{d-1}^2}}$$

   If $i < d$ reset i by setting it to zero. Save received vector b' and goto step (2).

This algorithm seems bulky but its implementation in ANSI C programming language requires only 30 lines of code.

Uniform distribution

```
//******************  CONSTANT PARAMETERS      **********************
#ifndef A_RANDOM
#define A_RANDOM    16807
#endif
#ifndef M_RANDOM
#define M_RANDOM    2147483647
#endif

#define     Q_RANDOM    (int)(M_RANDOM / A_RANDOM)
#define     R_RANDOM    (int)(M_RANDOM % A_RANDOM)

//****************      C O D E      ********************************

//*****************************************************************
//***
//***          R A N D O M    N U M B E R    G E N E R A T O R
//***
//*** Description:
//***       This function generates random number mod (2^31-1)
//***       using minimal standard generator
//***
//*****************************************************************
int random_32bit(int Seed_n)
{
    Seed_n = A_RANDOM*(Seed_n%Q_RANDOM)-R_RANDOM*(Seed_n/Q_RANDOM);
    if(Seed_n <= 0) {
        Seed_n += M_RANDOM;
    }
    return Seed_n;
}
//*** End of random_32bit ****************************************

//*****************************************************************
//***   G A U S S I A N     D I S T R I B U T I O N
//*****************************************************************
float gaussian_distribution(RANDOM_STRUCT* pRandomStruct)
{
```

```
    float   Value1;
    float   Value2;
    float   Value;
    int     Seed;

    if ((*pRandomStruct).Flag == ON) {
        (*pRandomStruct).Flag = OFF;
        return (*pRandomStruct).Gauss;
    }
    else {
        for(;;) {
            Seed = random_32bit( (*pRandomStruct).Seed1 );
            (*pRandomStruct).Seed1 = Seed;
            Value1 = ((float)Seed) / M_RANDOM;

            Seed = random_32bit( (*pRandomStruct).Seed2 );
            (*pRandomStruct).Seed2 = Seed;
            Value2 = ((float)Seed) / M_RANDOM;

            Value1 = 2*Value1 - 1;
            Value2 = 2*Value2 - 1;
            Value = Value1*Value1 + Value2*Value2;

            if( (Value >= (float)1.0) || (Value == 0) ) {
                continue;
            }
            else {
                Value = (float)sqrt((-2 * log(Value) )/ Value);
                (*pRandomStruct).Flag = ON;
                (*pRandomStruct).Gauss = Value1 * Value;
                return (Value2 * Value);
            }
        }
    }
}
//*** End of gaussian_distribution
//*****************************************************************
```

Here we present the plotting of he implementation of this algorithm for 2-dimensional case, when the unit circle is covered by 1024 points:

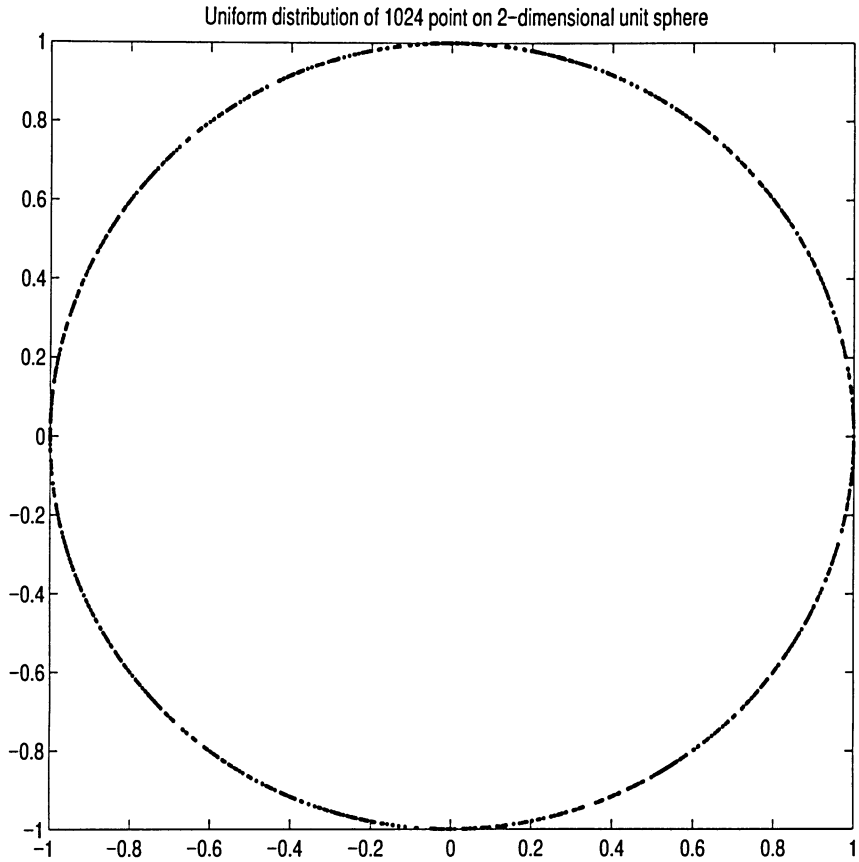

Figure C.1: Uniform distribution of 1024 points on unit circle

As we see from the figure above the algorithm performs well for 2-dimensional case and 1024 points covered the circle. However, for higher dimensions we need more points for good results. In the VGS algorithm implementation we build uniform distribution of the points on S^d only ones, and then use the same points through whole process of creating partition tree.

Appendix D

Abbreviations

ANSI	American National Standards Institute
DSP	Digital Signal Processing
GS	Greedy Splitting Algorithm
MRA	Multi-resolution Analysis
PDF	Probability Density Function
VGS	Vector Greedy Splitting Algorithm

Bibliography

[1] C. Blatter (1998): *Wavelets: a primer.* A K Peters, Ltd.

[2] Box, G.E.P, M.E. Muller (1958), A note on the generation of random normal deviates, *Annals Math. Stat*, V. 29, pp. 610-611.

[3] P.J. Catuogno, S.E. Ferrando, A.L. Gonzalez (2007), Adaptive Martingale Approximations, *in press.*

[4] A. Cohen, R.A. DeVore and R. Hochmuth (2000), Restricted nonlinear approximation, *Constructive Approx.* **16**, pp. 85-113.

[5] (2001), A. Cohen, W. Dahmen, I. Daubechies and R. DeVore. Tree approximation and optimal encoding. *Appl. Comput. Harmon. Anal.* 11(2), pp. 192-226.

[6] Ronald A. DeVore (Summer 1998), Nonlinear approximation. *Acta Numerica*, 1-99.

[7] Ronald A. DeVore, Guergana Petrova and Vladimir Temlyakov (May 2003), Best basis selection for approximation in L_p *Found. of Comp. Mathematics*, **3**, no. 2., pp. 161 - 185.

[8] D.L. Donoho (1993), Unconditional bases are optimal for data compression and for statistical estimation. *Appl. Comput. Harmonic Anal.*, **1**, no. 1., pp. 100-105.

[9] D.L. Donoho, M. Vetterli, R.A. DeVore and I Daubechies (1998), Data compression and harmonic analysis. *IEEE Trans. Inf. Theory*,**44** (6), pp. 2435-2476.

[10] G.P. Dwyer,Jr. and K.B. Williams (February 2003), Portable Random Number Generators. *Journal of Economic Dynamics and Control 27*, pp. 645-50.

[11] M. Girardi and W. Sweldens (1997), A new class of unbalanced Haar wavelets that form an unconditional basis basis for L^p on general measure spaces. *The Journal of Fourier Analysis and Applications*, **3**, 457-474.

[12] A. Jensen and A. la Cour-Harbo (2001): *Ripples in Mathematics. The Discrete Wavelet Transform.* Springer.

[13] E. Lieb and M. Loss (1997): *Analysis.* Graduate Studies in Mathematics 14 American Mathematical Society.

[14] S. Mallat and Z. Zhang (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions of Signal Processing*, Vol. **41**, 3397-3415.

[15] G. Marsaglia and W. W. Tsang (2000), The ziggurat method for generating random variables, *Journal of Statistical Software.*

[16] Moler, C.B. (2004), *Numerical Computing with MATLAB.* SIAM, .

[17] S.K. Park and K.W. Miller, " Random Number Generators: Good Ones Are Hard To Find", *Communications of the ACM*, (October 1988), pp. 1192-1201.

[18] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press; 2 edition (October 30, 1992).

[19] V.N. Temlyakov (1999), Greedy algorithms and m-term approximation with regard to redundant dictionaries. *J.Approx. Theory*, **98** pp. 117-145.