2012

Frames for Hilbert spaces and an application to signal processing

Kinney Thompson
Virginia Commonwealth University

Abstract

Frames for Hilbert spaces and an application to signal processing

Kinney Thompson

The goal of this paper will be to study how frame theory is applied within the field of signal processing. A frame is a redundant (i.e. not linearly independent) coordinate system for a vector space that satises a certain Parseval-type norm inequality. Frames provide a means for transmitting data and, when a certain about of loss is anticipated, their redundancy allows for better signal reconstruction. We will start with the basics of frame theory, give examples of frames and an application that illustrates how this redundancy can be exploited to achieve better signal reconstruction. We also include an introduction to the theory of frames in infinite dimensional Hilbert spaces as well as an interesting example.

Thesis Director: Dr. Kevin Beanland

Frames for Hilbert spaces and an application to signal processing

by

Kinney Thompson

Bachelor of Science
Virginia Polytechnic Institute and State University

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in the

Department of Mathematics

Virginia Commonwealth University

2012

Major Professor	Chair, Examining Committee
Committee Member	Committee Member
Committee Member	Dean of The Graduate School

Dedication

This manuscript is dedicated to my family and friends that have supported me as I took this journey with a special dedication to my friend David that left this world far too soon.

Acknowledgements

A special thanks to Dr. Kevin Beanland for all of his help and support with this work.

Abstract

The goal of this paper will be to study how frame theory is applied within the field of signal processing. A frame is a redundant (i.e. not linearly independent) coordinate system for a vector space that satises a certain Parseval-type norm inequality. Frames provide a means for transmitting data and, when a certain about of loss is anticipated, their redundancy allows for better signal reconstruction. We will start with the basics of frame theory, give examples of frames and an application that illustrates how this redundancy can be exploited to achieve better signal reconstruction. We also include an introduction to the theory of frames in infinite dimensional Hilbert spaces as well as an interesting example.

Contents

Chapter 1

Introduction

This paper consists of three chapters. In chapter 2 we lay the foundation for the theory of frames in finite dimensions. The tools necessary to construct a frame will be provided; starting from a complex scalar and all the way up to a finite dimensional frame and its associated operators. Necessary proofs and definitions will be provided along the way to allow for more clarity. Reference sources will be provided for any information outside the scope of this paper's discussion.

In chapter 3 of this paper we will introduce frame theory for an infinite dimensional Hilbert space. The main result in this section is the construction of an infinite dimensional frame such that no subset forms a Schauder basis for the space. This result is somewhat technical and it will broken into several pieces to clarify the exposition.

Following the previous two chapters one should have a good understanding of the basics of frame theory. In chapter 4 we discuss an application of frame theory to discrete data and the effects of quantization error compounded by the machine's word size limitations and provide conclusions reached. The frames used in the simulation are taken from chapter 2.

Chapter 2

Fundamentals of Frame Theory

Let us begin by defining what is our fundamental scalar element. Let

$$\mathbb{C} = \{x + iy : x, y \in \mathbb{R} \text{ and } i = \sqrt{-1}\}$$

Here $\mathbb{R}$ denotes the set of all real numbers. Let $|z| = \sqrt{x^2 + y^2}$. A complex number can also be expressed in polar coordinate form as $z = re^{i\theta}$ where $r \geq 0$ and θ is expressed in radians within the range $0 \leq \theta \leq 2\pi$. The relationship is given by Euler's identity

$$e^{i\theta} = \cos\theta + i\sin\theta.$$

Note that $|z| = |re^{i\theta}| = r$. $|e^{i\theta}| = 1$ for all values of θ. The complex conjugate for $z = x + iy$ is defined as $\overline{z} = x - iy$. Thus one can see that $|z|^2 = z\overline{z}$. Note that if we look at the polar coordinate form of complex numbers with $r = 1$ we can naturally define what are called the n^{th} roots of unity. These vectors have some nice properties that will be noticed later within the paper.

Definition 1. *Let n in $\mathbb{N}$, z in $\mathbb{C}$ is a n-th root of unity if $z^n = 1$. There are n such complex numbers.*

Remark: For each n in $\mathbb{N}$ the complex numbers

$$\{e^{i\frac{2\pi k}{n}} : k = 0, ..., n-1\}$$

are the n-th roots of unity.

For n in $\mathbb{N}$ let

$$\mathbb{C}^n = \{(a_i)_{i=1}^n : a_i \in \mathbb{C}\}.$$

DEFINITION 2. *Let n in $\mathbb{N}$ and define a function $\|\cdot\| : \mathbb{C}^n \to [0, \infty)$ by*

$$\|(a_i)_{i=1}^n\| = \sqrt{\sum_{i=1}^n |a_i|^2}.$$

Let $\mathcal{H}_n$ denote $\mathbb{C}$ equipped with this norm function. Let $x = (a_i)_{i=1}^n$ and $y = (b_i)_{i=1}^n$ in $\mathbb{C}^n$. Then $x + y = (a_i + b_i)_{i=1}^n$. This is to say, addition between vectors in this space is defined coordinate wise. Now for λ in $\mathbb{C}$, $\lambda x = \lambda(a_i)_{i=1}^n = (\lambda a_i)_{i=1}^n$. [**6**, Page 8]

Remark: For each n in $\mathbb{N}$ the function $\|\cdot\|$ satisfies the following

(1) $\|(a_i)_{i=1}^n\| = 0$ if and only if $a_i = 0$ for all $i = 1, ..., n$.

(2) $\|(a_i)_{i=1}^n + (b_i)_{i=1}^n\| \leq \|(a_i)_{i=1}^n\| + \|(b_i)_{i=1}^n\|$ which can also be written as

$$\sqrt{\sum_{i=1}^n |a_i + b_i|^2} \leq \sqrt{\sum_{i=1}^n |a_i|^2} + \sqrt{\sum_{i=1}^n |b_i|^2}.$$

(3) $\|\lambda(a_i)_{i=1}^n\| = |\lambda|\|(a_i)_{i=1}^n\|$.

Now we define the*inner product* function for $\mathcal{H}_n$. Define the following function $\langle\cdot,\cdot\rangle : \mathcal{H}_n \times \mathcal{H}_n \to \mathbb{C}$ by

$$\langle (a_i)_{i=1}^n, (b_i)_{i=1}^n \rangle = \sum_{i=1}^n a_i \overline{b_i}$$

Note that for x in $\mathcal{H}_n$

$$\|x\|^2 = \sum_{i=1}^n |a_i|^2 = \sum_{i=1}^n a_i \overline{a_i} = \langle x, x \rangle.$$

The following are all properties of the inner product, $\langle \cdot, \cdot \rangle$. Let x, y, z be elements in $\mathcal{H}_n$ and λ in $\mathbb{C}$.

(1) $\langle x, y \rangle = \overline{\langle y, x \rangle}$.

(2) $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$ and $\langle x, y + z \rangle = \langle x, y \rangle + \langle x, z \rangle$.

(3) $\langle x, \lambda y \rangle = \overline{\lambda} \langle x, y \rangle$ and $\langle \lambda x, y \rangle = \lambda \langle x, y \rangle$.

DEFINITION 3. *Let $A \subset \mathcal{H}_n$ then*

$$span(A) = \{\sum_{i=1}^{n} c_i x_i : n \in \mathbb{N}, c_i \in \mathbb{C}, x_i \in A \text{ for } i = 1, \dots n\}.$$

DEFINITION 4. *A sequence $\{x_k\}_{k=1}^n$ is a basis for $\mathbb{C}^n$ if*

(a) $\mathbb{C}^n = span\{x_k\}_{k=1}^n = \{\sum_{i=1}^n a_i x_i : a_i \in \mathbb{C}^n\}$

(b) $\{x_k\}_{k=1}^n$ is linearly independent. In other words, if $\sum_{k=1}^n a_k x_k = 0$ for some scalars $\{a_k\}_{k=1}^n$ then $a_k = 0$ for all $k = 1, ..., n$.

DEFINITION 5. *A basis $\{x_k\}_{k=1}^n$ is an orthonormal basis for $\mathbb{C}^n$ if $\{x_k\}_{k=1}^n$ is a basis, and*

$$\langle x_k, x_j \rangle = \delta_{k,j} = \begin{cases} 1 & k = j \\ 0 & k \neq j. \end{cases}$$

Define e_k as the vector with a 1 in the k-th position and 0 in all other places. This is the canonical orthonormal basis. For each $n \in \mathbb{N}$, $(e_k)_{k=1}^n$ is called the unit vector basis of $\mathbb{C}^n$ and will be used throughout this paper.

EXAMPLE 1. Another simple example for $\mathbb{C}^3$ is

$$e_1 = (2, 0, -1)$$

$$e_2 = (0, -1, 0)$$

$$e_3 = (2, 0, 4)$$

One can quickly see that these three vectors are orthonormal and therefore linearly independent.

Orthonormal bases have the following reconstruction property. Let x in $\mathcal{H}_n$. Then

$$x = \sum_{k=1}^{n} \langle x, x_k \rangle x_k.$$

In the case of the canonical orthonormal basis for $\mathcal{H}_n$ $x = (a_k)_{k=1}^n$, $\langle x, e_k \rangle = a_k$.

The next proposition is a basic fact from linear algebra. We omit the proof.

PROPOSITION 1. *Let n in $\mathbb{N}$ and M be a subspace of $\mathbb{C}^n$. Then there is a sequence $\{x_k\}_{k=1}^d$ in M that forms an orthonormal basis for M where $d \leq n$.*

Remark: Let $\{x_k\}_{k=1}^n$ be orthonormal basis, such that $\|x_k\| = 1$ for all $k = 1, \ldots, n$. Then for all x in $\mathcal{H}_n$

$$\sum_{k=1}^{n} |\langle x, x_n \rangle|^2 = \|x\|^2.$$

This is called Parseval's Identity.

PROOF. Let $\{x_k\}_{k=1}^n$ be an orthonormal basis of $\mathcal{H}_n$ and x in $\mathcal{H}_n$.

$$\begin{aligned}
\|x\|^2 &= |\langle x, x \rangle| \\
&= |\langle \sum_{i=1}^{n} \langle x, x_i \rangle x_i, \sum_{j=1}^{n} \langle x, x_j \rangle x_j \rangle| \\
&= \sum_{i=1}^{n} \langle x, x_j \rangle \langle x_i, \sum_{j=1}^{n} \langle x, x_j \rangle x_j \rangle \\
&= \sum_{i=1}^{n} \langle x, x_i \rangle \sum_{j=1}^{n} \overline{\langle x, x_j \rangle} \langle x_i, x_j \rangle \qquad (1) \\
&= \sum_{i=1}^{n} \langle x, x_i \rangle \overline{\langle x, x_i \rangle} \\
&= \sum_{i=1}^{n} |\langle x, x_i \rangle|^2
\end{aligned}$$

□

Now we introduce the definition of a frame. This is the focal point of study within this paper.

DEFINITION 6. [**3**, Definition 1.1.1] *Let $m \geq n$ in $\mathbb{N}$. A sequence $\{f_k\}_{k=1}^m$ in $\mathcal{H}_n$ is a frame for $\mathcal{H}_n$ if there exist constants $A, B > 0$ in $\mathbb{R}$ such that for all x in $\mathcal{H}_n$*

$$A\|x\|^2 \leq \sum_{k=1}^{m} |\langle x, f_k\rangle|^2 \leq B\|x\|^2$$

Remark: The sequence $\{f_k\}_{k=1}^m$ above *almost* satisfies Parseval's Identity. In fact if $A = B = 1$ then the frame is called a Parseval frame. If for all $k = 1, ..., m$, $\|f_k\| = 1$ we say the frame is normalized.

Note the following fundamental inequality. We omit the proof.

LEMMA 1. *(Cauchy - Schwartz Inequality) Let n in $\mathbb{N}$ and x, y in $\mathcal{H}_n$. Then*

$$|\langle x, y\rangle| \leq \|x\|\|y\|$$

Remark: For any collection of vectors $\{f_k\}_{k=1}^m$ in $\mathbb{C}^n$ taking $B = \sum_{k=1}^m \|f_k\|^2$ a apply the Cauchy-Schwartz inequality

$$\sum_{k=1}^{m} |\langle x, f_k\rangle|^2 \leq B\|x\|^2$$

for all x in $\mathcal{H}_n$. Therefore the an upper inequality always holds.

PROPOSITION 2. *If $\{f_k\}_{k=1}^m$ is in $\mathcal{H}_n$ and span$\{f_k\}_{k=1}^m = \mathcal{H}_n$ then $\{f_k\}_{k=1}^m$ is a frame for $\mathcal{H}_n$.*

PROOF. The upper inequality comes from the previous remark. Thus it suffices to prove the lower bound of the inequality. Let

$$S_{\mathcal{H}_n} = \{x \in \mathcal{H}_n : \|x\| = 1\},$$

$S_{\mathcal{H}_n}$ is a compact subset of $\mathcal{H}_n$. Define

$$\Phi : S_{\mathcal{H}_n} \to \mathbb{R} \text{ by } \Phi(x) = \sum_{k=1}^{m} |\langle x, f_k\rangle|^2.$$

Note that $\Phi(x) \neq 0$ for all x since $|\langle x, f_k \rangle| > 0$ for at least one k in $\{1, ..., m\}$. If this were not the case, x would be orthogonal to all of the members the collection $\{f_k\}_{k=1}^m$ which is impossible since $\text{span}\{f_k\} = \mathbb{C}^n$.

Since Φ is continuous and $S_{\mathcal{H}_n}$ is compact, there exists g in $S_{\mathcal{H}_n}$ such that $\Phi(g) = \sum_{k=1}^m |\langle g, f_k \rangle|^2$ is the minimum. It follows that for all x in $\mathcal{H}_n$,

$$\sum_{k=1}^m |\langle x, f_k \rangle|^2 = \|x\|^2 \sum_{k=1}^m |\langle \frac{x}{\|x\|}, f_k \rangle|^2 \geq \|x\|^2 \sum_{k=1}^m |\langle g, f_k \rangle|^2 = A\|x\|^2$$

□

The previous proposition tells us that in finite dimensions any spanning set is in fact a frame. From this fact it follows that if $\{f_k\}_{k=1}^m$ is a frame for $\mathcal{H}_n$ then for all x in $\mathcal{H}_n$ there exists coefficients $(c_k)_{k=1}^n$, not necessarily unique, such that

$$x = \sum_{k=1}^m c_k f_k.$$

Later on we will address the problem of finding coefficients that are minimal in a certain sense.

We will now switch gears a bit to introduce some important definitions from operator theory.

Let $m, n \in \mathbb{N}$. A map $T : \mathcal{H}_n \to \mathcal{H}_m$ is a linear operator if for all λ in $\mathbb{C}$ and x, y in $\mathcal{H}_n$ we have the following

$$T(\lambda x + y) = \lambda T x + T y.$$

Let $\mathcal{N}_T := \{x \in \mathcal{H}_n : Tx = 0\}$, the null-space of T. Note that T is one-to-one if and only if $\mathcal{N}_T = \{0\}$.

Let M be a subspace of $\mathcal{H}_n$. Define

$$M^\perp = \{y \in \mathcal{H}_n : \langle x, y \rangle = 0 \text{ for all } x \in M\}.$$

The space $M^\perp$ is called the annihilator of M and is a subspace of $\mathcal{H}_n$

PROPOSITION 3. *For all z in $\mathcal{H}_n$ there exists unique x in M and y in $M^\perp$ such that*

$$z = x + y \text{ and } \|z\| = \sqrt{\|x\|^2 + \|y\|^2}.$$

PROOF. Let $\{x_k\}_{k=1}^m$ be an orthonormal basis for M and define

$$P : \mathcal{H}_n \to M \text{ by } Px = \sum_{k=1}^{m} \langle x, x_k \rangle x_k.$$

It follows that for all x in M. $Px = x$. It is also easy to see that for all y in $\mathcal{H}_n$ we have $P(Py) = Py$, or, in other words, $P^2 = P$.

Also note that z in $M^\perp$ if and only if $Pz = 0$. Indeed, for z in $M^\perp$, $Pz = \sum_{k=1}^{d} \langle z, x_k \rangle x_k = 0$ since $\langle z, x_k \rangle = 0$ for all $k = 1, ..., m$. Suppose z in $\mathcal{H}_n$ and $Pz = 0$. Let x in M. Then

$$\begin{aligned} \langle z, x \rangle &= \langle z, \sum_{k=1}^{m} \langle x, x_k \rangle x_k \rangle = \sum_{k=1}^{m} \langle x, x_k \rangle, \langle z, x_k \rangle \\ &= \langle x, \sum_{k=1}^{m} \langle z, x_k \rangle x_k \rangle = \langle x, Pz \rangle = 0. \end{aligned} \tag{2}$$

For y in $\mathcal{H}_n$ let

$$y = Py + (y - Py).$$

By definition Py in M. Since $P(y - Py) = Py - P^2 y = 0$, $y - Py$ in $M^\perp$. This proves the first claim.

To prove the second part, let $\{x_k\}_{k=1}^n$ be an orthonormal basis of $\mathcal{H}_n$ such that $\{x_k\}_{k=1}^m$ is the orthonormal basis of M and $\{x_k\}_{k=m+1}^n$ is the orthonormal basis of $M^\perp$.

$$y = \sum_{k=1}^{n} \langle y, x_k \rangle x_k = \sum_{k=1}^{m} \langle y, x_k \rangle x_k + \sum_{k=m+1}^{n} \langle y, x_k \rangle x_k.$$

By Parseval's identity

$$\|y\|^2 = \sum_{k=1}^{n} |\langle y, x_k \rangle|^2 = \sum_{k=1}^{m} |\langle y, x_k \rangle|^2 + \sum_{k=m+1}^{n} |\langle y, x_k \rangle|^2 = \|Py\|^2 + \|y - Py\|^2$$

This proves the second part of the claim. □

We now introduce and study the adjoint of a linear operator. Let m, n in $\mathbb{N}$ and $T : \mathcal{H}_n \to \mathcal{H}_m$ be a linear operator. Define the operator $T^* : \mathcal{H}_m \to \mathcal{H}_n$ as follows

$$\langle T^* x, y\rangle := \langle x, Ty\rangle$$

DEFINITION 7. *An operator $T : \mathcal{H}_n \to \mathcal{H}_n$ is self-adjoint if for all x, y in $\mathcal{H}_n$*

$$\langle T^* x, y\rangle = \langle Tx, y\rangle.$$

In other words $T = T^*$.

Remark: Let m, n in $\mathbb{N}$ and $T : \mathcal{H}_n \to \mathcal{H}_m$. Then $T^{**} = T$.

PROOF. Let x, y in $\mathcal{H}$, then

$$\langle T^{**} x, y\rangle = \langle x, T^* y\rangle = \overline{\langle T^* y, x\rangle} = \overline{\langle y, Tx\rangle} = \langle Tx, y\rangle. \tag{3}$$

□

Remark: Let m, n in $\mathbb{N}$ and $T : \mathcal{H}_n \to \mathcal{H}_m$. Then $R = TT^*$ is self-adjoint.

PROOF. Let x, y in $\mathcal{H}_n$.

$$\langle R^* x, y\rangle = \langle x, TT^* y\rangle = \langle T^* x, T^* y\rangle = \langle T^{**} T^* x, y\rangle = \langle Rx, y\rangle. \tag{4}$$

□

Remark: If $R : \mathcal{H}_n \to \mathcal{H}_n$ is self-adjoint and invertible then R^{-1} is self-adjoint.

PROOF. Let x, y in $\mathcal{H}_n$ and $Rw = x, Rv = y$

$$\langle (R^{-1})^* x, y\rangle = \langle x, R^{-1}y\rangle = \langle Rw, v\rangle = \langle R^* w, v\rangle = \langle w, Rv\rangle = \langle R^{-1}x, y\rangle. \tag{5}$$

□

To each frame, $\{f_k\}_{k=1}^m$ of $\mathbb{C}^n$, there are *associated operators*. The first is called the *pre-frame* operator. $T : \mathbb{C}^m \to \mathbb{C}^n$ defined by

$$T(\sum_{i=1}^m a_i e_i) = \sum_{i=1}^m a_i f_i$$

where $(e_i)_{i=1}^m$ is the unit vector basis of $\mathbb{C}^m$. Note that T is well defined and onto but not one-to-one if $m > n$. In addition, the adjoint of T, which is called *analysis operator*, $T^* : \mathbb{C}^n \to \mathbb{C}^m$ satisfies

$$T^*(x) = \sum_{k=1}^m \langle x, f_k\rangle e_k.$$

Note that T^* is one-to-one but not onto. Observe that $TT^* : \mathbb{C}^n \to \mathbb{C}^n$. Note that

$$TT^* x = T(\sum_{k=1}^m \langle x, f_k\rangle e_k) = \sum_{k=1}^m \langle x, f_k\rangle f_k. \tag{6}$$

Let $S = TT^*$, then $Sx = \sum_{k=1}^m \langle x, f_k\rangle f_k$. S is called the *frame* operator. If $n = m$ and $\{f_k\}_{k=1}^m$ is an orthonormal basis then S is simply the identity operator. If a frame $\{f_k\}_{k=1}^m$ satisfies

$$\sum_{k=1}^m |\langle f_k, x\rangle|^2 = A\|x\|^2$$

for all x in $\mathcal{H}_n$ and some A in $\mathbb{R}$ then we say $\{f_k\}_{k=1}^m$ is a *tight* frame.

PROPOSITION 4. *If $\{f_k\}_{k=1}^m$ is a tight frame for $\mathcal{H}_n$ then $S = AI$.*

The above proposition follows directly from the definition. The proof can be found in [**3**, Proposition 1.1.4].

PROPOSITION 5. *If S is a frame operator then S is invertible.*

PROOF. Since S is a linear operator it is enough to prove that $Sx = 0$ implies $x = 0$. Assume that $Sx = 0$. Then by definition

$$0 = \langle Sx, x\rangle = \langle \sum_{k=1}^{m} \langle x, f_k\rangle f_k, x\rangle = \sum_{k=1}^{m} \langle x, f_k\rangle\langle f_k, x\rangle = \sum_{k=1}^{m} |\langle x, f_k\rangle|^2. \tag{7}$$

This implies that $\langle x, f_k\rangle = 0$ for all $k = 1, ..., m$, which means $x = 0$. Using the rank-plus-nullity theorem for linear maps S is surjective and thus invertible. □

PROPOSITION 6. *Let f in $\mathcal{H}_n$. If $f = \sum_{k=1}^{m} c_k f_k$ then*

$$\sum_{k=1}^{m} |c_k|^2 = \sum_{k=1}^{m} |\langle f, S^{-1} f_k\rangle|^2 + \sum_{k=1}^{m} |c_k - \langle f, S^{-1} f_k\rangle|^2$$

Moreover letting $c_k = \langle f, S^{-1} f_k\rangle$. We can see that for all f

$$\sum_{k=1}^{m} |\langle f, S^{-1} f_k\rangle|^2 = \inf\{\sum_{k=1}^{m} |c_k|^2 : f = \sum_{k=1}^{m} c_k f_k\}.$$

PROOF. Suppose $f = \sum_{k=1}^{m} c_k f_k$. First note that

$$f = SS^{-1} f = \sum_{k=1}^{m} \langle S^{-1} f, f_k\rangle f_k = \sum_{k=1}^{m} \langle f, S^{-1} f_k\rangle f_k$$

Since S is self-adjoint, S^{-1} is also self-adjoint so we have that $\langle S^{-1} f, f_k\rangle = \langle f, S^{-1} f_k\rangle$. In addition

$$T(\sum_{k=1}^{m} c_k e_k - \sum_{k=1}^{m} \langle f, S^{-1} f_k\rangle e_k) = 0$$

where $T : \mathbb{C}^m \to \mathbb{C}^m$ is the pre-frame operator. Therefore

$$\sum_{k=1}^{m} c_k e_k - \sum_{k=1}^{m} \langle f, S^{-1} f_k\rangle e_k \in \mathcal{N}_T.$$

Applying Proposition 3 for $M = \mathcal{N}^T$ we know that

$$\sum_{k=1}^{m} \langle f, S^{-1} f_k\rangle e_k \in \mathcal{N}_T^{\perp}$$

and so

$$\sum_{k=1}^{m} |c_k|^2 = \| \sum_{k=1}^{m} c_k e_k \|^2$$

$$= \| \sum_{k=1}^{m} \langle f, S^{-1} f_k \rangle e_k \|^2 + \| \sum_{k=1}^{m} (c_k - \langle f, S^{-1} f_k \rangle) e_k \|^2 \tag{8}$$

$$= \sum_{k=1}^{m} |\langle f, S^{-1} f_k \rangle|^2 + \sum_{k=1}^{m} |c_k - \langle f, S^{-1} f_k \rangle|^2.$$

The quantity on the right hand side is clearly minimized when $c_k = \langle f, S^{-1} f_k \rangle$.

□

Now we provide some simple examples of frames. Our first example is quite trivial.

EXAMPLE 2. Let $f_{2i+1} = 0$ and $f_{2i} = e_i$ for all i in $\mathbb{N}$. Then $\{f_k\}_{k=1}^{2n}$ is a Parseval frame for $\mathcal{H}_n$.

PROOF. Let n in $\mathbb{N}$ and f in $\mathcal{H}_n$. It is clear to see from the definition that

$$\sum_{k=1}^{2n} |\langle f, f_k \rangle|^2 = \sum_{k=1}^{n} |\langle f, e_k \rangle|^2 = \|f\|^2.$$

Thus $\{f_k\}_{k=1}^{2n}$ is a Parseval frame. □

EXAMPLE 3. Let n in $\mathbb{N}$. Define the sequence $\{f_k\}_{k=1}^{\frac{n(n+1)}{2}}$ as follows:

$$f_1 = e_1, f_2 = \frac{e_2}{\sqrt{2}}, f_3 = \frac{e_2}{\sqrt{2}}, f_4 = \frac{e_3}{\sqrt{3}}, f_5 = \frac{e_3}{\sqrt{3}}, f_6 = \frac{e_3}{\sqrt{3}}, ...$$

Note that for each e_k there are k elements in $\{f_k\}_{k=1}^{\frac{n(n+1)}{2}}$ defined by it, also note that for each e_i that $\|f_k\| = \frac{1}{\sqrt{i}}$ where $\frac{i(i+1)}{2} < k \leq \frac{(i+1)(i+2)}{2}$ where i is in $\mathbb{N}$. The sequence $\{f_k\}_{k=1}^{\frac{n(n+1)}{2}}$ is a Parseval frame for $\mathcal{H}_n$.

PROOF. Let n in $\mathbb{N}$ and f in $\mathcal{H}_n$. It is clear to see from the definition that

$$\sum_{k=1}^{\frac{n(n+1)}{2}} |\langle f, f_k \rangle|^2 = \sum_{k=1}^{n} |\langle f, e_k \rangle|^2 = \langle f, f \rangle = \|f\|^2$$

Thus $\{f_k\}_{k=1}^{\frac{n(n+1)}{2}}$ is a Parseval frame. □

The next frame will be used in the simulation software discussed later. This frame is called the *Discrete Fourier Transform* frame [**3**, Section 1.4].

First we define an orthonormal basis. Let n in $\mathbb{N}$ and define

$$x_k = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} e^{2\pi i \frac{(j-1)(k-1)}{n}} e_j$$

Note that for $k = 1, ..., n$ and $j = 1, ..., n$, $e^{2\pi i \frac{(j-1)(k-1)}{n}}$ is an n^{th} root of unity, i.e. $(e^{2\pi i \frac{(j-1)(k-1)}{n}})^n = 1$ and $|e^{2\pi i \frac{(j-1)(k-1)}{n}}| = 1$.

THEOREM 2. *$\{x_k\}_{k=1}^n$ is an orthonormal basis for $\mathbb{C}^n$.*

PROOF. First note that for $k = 1, ..., n$

$$\|x_k\| = \sqrt{\sum_{j=1}^{n} \left(\frac{|e^{2\pi i \frac{(j-1)(k-1)}{n}}|}{\sqrt{n}} \right)^2} = \sqrt{\frac{1}{n} \sum_{j=1}^{n} 1} = 1$$

Let $k \neq l$ with k, l in $\{1, ..., n\}$. A simple computation yields

$$\langle x_k, x_l \rangle = \frac{1}{n} \sum_{j=1}^{n} e^{2\pi i (j-1)(\frac{k-1}{n} + \frac{l-1}{n})} = \frac{1}{n} \sum_{j=1}^{n} (e^{2\pi i (\frac{k-l}{n})})^{(j-1)} \tag{9}$$

Using the identity $1 + ... + x^{n-1} = \dfrac{1 - x^n}{1 - x}$ we have

$$\frac{1}{n} \sum_{j=1}^{n} (e^{2\pi i (\frac{k-l}{n})})^{(j-1)} = \frac{1}{n} \left(\frac{1 - e^{2\pi i (k-l)}}{1 - e^{2\pi i (\frac{k-l}{n})}} \right) = 0.$$

□

Let m, n in $\mathbb{N}$ such that $m > n$. Define P as the projection, $P : \mathbb{C}^m \to \mathbb{C}^n$, such that for x in $\mathbb{C}^m$,

$$Px = \sum_{k=1}^{n} \langle x, e_k \rangle e_k$$

Note that the projection P is self-adjoint and that for all x in $\mathbb{C}^n$, $Px = x$. Now let $\{x_k\}_{k=1}^m$ be the Discrete Fourier Transform basis for $\mathbb{C}^m$. and define the sequence $\{x_k\}_{k=1}^m$ as follows

$$f_k = Px_k, \text{ for } k = 1, ..., m$$

PROPOSITION 7. *For $m > n$ with m, n in $\mathbb{N}$, $\{f_k\}_{k=1}^m$ is a Parseval frame for $\mathbb{C}^n$ and $\|f_k\| = \sqrt{\frac{m}{n}}$. for all $k = 1, ..., m$.*

PROOF. Let f in $\mathbb{C}^n$. Note that f also exists in $\mathbb{C}^m$ but with $f(j) = 0$ for $n < j \leq m$.

$$\|f\|^2 = \sum_{k=1}^{m} |\langle f, x_k\rangle|^2 = \sum_{k=1}^{m} |\langle Pf, x_k\rangle|^2 = \sum_{k=1}^{m} |\langle f, Px_k\rangle|^2 = \sum_{k=1}^{m} |\langle f, f_k\rangle|^2$$

Note that $\sum_{k=1}^{m} |\langle f, Px_k\rangle|^2 = \sum_{k=1}^{m} |\langle f, x_k\rangle|^2$ because f is in $\mathbb{C}^n$ and this $f(j) = 0$ for $n < j \leq m$. Also note that for all k

$$\|f_k\| = \|Px_k\| = \sqrt{\sum_{j=1}^{n} (\frac{|e^{2\pi i \frac{(j-1)(k-1)}{m}}|}{\sqrt{m}})^2} = \sqrt{\sum_{j=1}^{n} \frac{1}{m}} = \sqrt{\frac{n}{m}}$$

□

PROPOSITION 8. *Let $m > n$ and $\{f_k\}_{k=1}^m$ be the Discrete Fourier Frame for $\mathbb{C}^n$. Any subset of $\{f_k\}_{k=1}^m$ containing at least n elements will form a basis for $\mathbb{C}^n$.*

PROOF. Let $\{k_1, k_2, ..., k_n\}$ be and arbitrary subset of $\{1, 2, ..., m\}$. Write these vectors as rows in an $n \times n$ matrix.

$$\frac{1}{\sqrt{m}} \begin{pmatrix} 1 & e^{2\pi i \frac{k_1-1}{m}} & \cdot & \cdot & e^{2\pi i \frac{(k_1-1)(n-1)}{m}} \\ 1 & e^{2\pi i \frac{k_2-1}{m}} & \cdot & \cdot & e^{2\pi i \frac{(k_2-1)(n-1)}{m}} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & e^{2\pi i \frac{k_n-1}{m}} & \cdot & \cdot & e^{2\pi i \frac{(k_n-1)(n-1)}{m}} \end{pmatrix}$$

This matrix is a Vandermonde matrix and the determinant is given by

$$\frac{1}{m^{\frac{n}{2}}} \prod_{i,j=1, i\neq j}^{n} (e^{\frac{2\pi i}{m}(k_i-1)} - e^{\frac{2\pi i}{m}(k_j-1)}) \neq 0.$$

Thus $\{f_k\}_{k=1}^m$ is a basis for $\mathbb{C}^n$. □

Chapter 3

An Infinite Dimensional Example

In this chapter we introduce the concept of an infinite dimensional Hilbert space, denoted $\mathcal{H}$, and infinite dimensional frame for $\mathcal{H}$. It is easy to see that for each n in $\mathbb{N}$ if $\{f_k\}_{k=1}^m$, with $m > n$, is a frame for $\mathcal{H}_n$ then there exists a set $I \subseteq \{1, ..., m\}$ with $|I| = n$ such that $\{f_k\}_{k\in I}$ is a basis for $\mathcal{H}_n$. Indeed, this follows from the fact that any linearly independent set of n vectors form a basis for $\mathcal{H}_n$. In the case that $m > n$ and $\{f_k\}_{k=1}^m$ is the Discrete Fourier Transform frame the following, much stronger property is satisfied: for any $I \subseteq \{1, ..., m\}$ such that $|I| = n$, $\{f_k\}_{k\in I}$ is a basis for $\mathcal{H}_n$.

As we will show in this chapter, the infinite dimensional case is very different. This is due to the fact that the concept of basis is very different. In fact there are several notions in the literature that are infinite dimensional versions of bases (e.g. Hamel basis, Riesz basis, ω-linearly independent, biorthogonal systems, Schauder basis). We focus particularly on the notion of Schauder basis [**5**, Chapter 4] [**1**]. In particular we will give an example of a frame $\{f_k\}_{k=1}^\infty$ such that for any $I \subset \mathbb{N}$, $\{f_k\}_{k\in I}$ is not a Schauder basis for $\mathcal{H}$.

We begin by introducing the necessary definitions.

Definition 8. *Let*

$$\mathcal{H} = \{(a_i)_{n\in\mathbb{N}} : \sum_{n=1}^{\infty} |a_n|^2 < \infty \text{ and } a_i \in \mathbb{C} \text{ for } i \in \mathbb{N}\}$$

For $(a_i)_{i=1}^{\infty}, (b_i)_{i=1}^{\infty} \in \mathcal{H}$ *define*

$$\|(a_i)_{i=1}^{\infty}\| = \left(\sum_{n=1}^{\infty} |a_n|^2\right)^{1/2} \text{ and } \langle (a_i), (b_i)\rangle = \sum_{i=1}^{\infty} a_i \overline{b_i}.$$

The space $\mathcal{H}$ *with the above norm is and example of an infinite dimensional Hilbert space and is usually denoted* $\ell_2(\mathbb{N})$ *in the literature.*

We now will introduce the notion of Schauder basis in $\mathcal{H}_n$.

Definition 9. *Let* n *in* $\mathbb{N}$. *A sequence* $(x_k)_{k=1}^n$ *is called a Schauder basis if there exists* $C \geq 1$ *such that for all* $(c_i)_{i=1}^n \subseteq \mathbb{C}$ *and* $m < n$

$$\|\sum_{i=1}^{m} c_i x_i\| \leq C\|\sum_{i=1}^{n} c_i x_i\|.$$

The smallest such C *is called the basis constant of* $(x_i)_{i=1}^n$.

Remark: If $\{f_k\}_{k=1}^n$ is linearly-independent in $\mathcal{H}_n$ then $\{f_k\}_{k=1}^n$ is a Schauder basis for $\mathcal{H}_n$.

Proof. Let n in $\mathbb{N}$ and $\{f_k\}_{k=1}^n$ such that for x in $\mathcal{H}_n$

$$x = \sum_{k=1}^{n} \langle x, f_k\rangle f_k.$$

Let $m < n$. Then by the triangle inequality and Cauchy-Schwartz we have the following

$$\begin{aligned} \|\sum_{k=1}^{m} \langle x, f_k\rangle f_k\| &\leq \sum_{k=1}^{m} \|f_k\|^2 \|x\| \\ &\leq \sum_{k=1}^{m} \|f_k\|^2 \|x\| + \sum_{k=m+1}^{n} \|f_k\|^2 \|x\| \\ &= \|x\| \sum_{k=1}^{n} \|f_k\|^2. \end{aligned} \tag{10}$$

By letting $C = \sum_{k=1}^{n} \|f_k\|^2$ we have that $\{f_k\}_{k=1}^n$ is a Schauder basis. □

Remark: Suppose $\{x_k\}_{k=1}^n$ is linearly independent in $\mathcal{H}_n$ and $C \geq 1$. Then $\{x_k\}_{k=1}^n$ has Schauder basis constant greater than C if there exists $(c_k)_{k=1}^n \subset \mathbb{C}$ and there exists $m < n$ such that

$$\|\sum_{k=1}^{m} c_i x_k\| > C\|\sum_{k=1}^{n} c_k x_k\|.$$

We now give the definition of Schauder basis for an infinite dimensional Hilbert space $\mathcal{H}$. Note that a Schauder basis should not be confused with the notion of basis from linear algebra in which every vector in the space is a *finite* linear combination of vectors from the space. This type of basis is called a Hamel basis and is necessarily an uncountable collection of vectors [**6**] for an infinite dimensional space.

DEFINITION 10. *A sequence $\{x_k\}_{k=1}^\infty$ is a Schauder basis for $\mathcal{H}$ if $\overline{span\{x_k\}_{k=1}^\infty} = \mathcal{H}$ and there is a $C \geq 1$ such that for all $m < n$ scalars and $(a_i)_{i=1}^n$ then*

$$\|\sum_{i=1}^{m} a_i x_i\| \leq C\|\sum_{i=1}^{n} a_i x_i\|.$$

The smallest such C is the basis constant of $\{x_k\}_{k=1}^\infty$.

Remark: It is easy to see that if $\{x_k\}_{k=1}^\infty$ is a Schauder basis then every finite subset is linearly independent.

At first the above definition may seem a bit peculiar. Indeed this is not often given as the definition of Schauder basis, rather it is proved to be equivalent.

PROPOSITION 9. [**3**, pg. 47] *A sequence $\{x_k\}_{k=1}^\infty$ in $\mathcal{H}$ is a Schauder basis if and only if for every x in $\mathcal{H}$ there exist unique scalars $(c_i)_{i=1}^\infty$ such that*

$$\lim_{n\to\infty} \|\sum_{i=1}^{n} c_i x_i - x\| = 0$$

We write $x = \sum_{i=1}^\infty c_i x_i$, i.e. every x has a unique countably infinite representation.

Note: If $\{x_k\}_{k=1}^\infty$ is a Schauder basis for $\mathcal{H}$ and $\pi : \mathbb{N} \to \mathbb{N}$ is a permutation, then $\{x_{\pi(k)}\}_{k=1}^\infty$ need not be a Schauder basis for $\mathcal{H}$.

We are now introduce the notion of a Schauder frame for $\mathcal{H}$.

DEFINITION 11. *A sequence* $\{f_k\}_{k=1}^{\infty}$ *is a frame for* $\mathcal{H}$ *if there exist* $A, B > 0$ *such that for* x *in* $\mathcal{H}$

$$A\|x\|^2 \leq \sum_{k=1}^{\infty} |\langle f_k, x\rangle|^2 \leq B\|x\|^2.$$

As before if $A = B$ *then* $\{f_k\}_{k=1}^{\infty}$ *is a tight frame and if* $A = B = 1$ *then* $\{f_k\}_{k=1}^{\infty}$ *is a Parseval frame.*

THEOREM 3. [**3**, Theorem 5.5.1 page 102] *Let* $\{x_k\}_{k=1}^{\infty}$ *be a Schauder basis for* $\mathcal{H}$. *Then* $\{x_k\}_{k=1}^{\infty}$ *is a frame for* $\mathcal{H}$ *if there exists a* C *such that for all* $(c_i)_{i=1}^{\infty}$ *in* $\ell_2(\mathbb{N})$

$$\|\sum_{i=1}^{\infty} c_i x_i\| \leq C\sqrt{\sum_{i=1}^{\infty} |c_i|^2}.$$

We note that not every Schauder basis of $\mathcal{H}$ satisfies this inequality (see [**3**, Exercise 5.8 page 119]). However by Parseval's equality every orthonormal Schauder basis of $\mathcal{H}$ does have this property.

DEFINITION 12. *A sequence* $\{f_k\}_{k=1}^{\infty}$ *in* $\mathcal{H}$ *is seminormalized if*

$$0 < \inf_{k\in\mathbb{N}} \|f_k\| \leq \sup_{k\in\mathbb{N}} \|f_k\| < \infty.$$

We are now ready to state the main theorem of this chapter. This was proved in [**1**].

THEOREM 4. *There exists a seminormalized sequence* $\{f_k\}_{k=1}^{\infty}$ *in* $\mathcal{H}$ *such that*

(1) $\{f_k\}_{k=1}^{\infty}$ *is a tight frame for* $\mathcal{H}$.

(2) If $(n_i)_{i=1}^{\infty} \subseteq \mathbb{N}$, *not necessarily increasing, such that* $\overline{span\{f_{n_i}\}_{i=1}^{\infty}} = \mathcal{H}$ *then* $\{f_{n_i}\}_{i=1}^{\infty}$ *is not a Schauder basis for* $\mathcal{H}$.

This theorem illustrates the strong divergence between bases for finite and for infinite dimensional Hilbert spaces. We will now outline the proof of the theorem. First we find for each n in $\mathbb{N}$ a tight frame $\{f_k\}_{k=1}^{n+1}$ for $\mathcal{H}_n$ such that each spanning subset will have basis constant $\geq \frac{1}{4}\sqrt{n-2}$. Then $\mathcal{H}$ can be naturally identified with

the infinite direct sum of the spaces $(\mathcal{H}_i)_{i=1}^{\infty}$. Then we will consider a tight frame for $\mathcal{H}$ which is, roughly, given by $\{\{f_k\}_{k=1}^{n+1}\}_{n=1}^{\infty}$. Finally, we argue that any spanning subset must contain, for each n in $\mathbb{N}$, a finite subset which spans the corresponding $\mathcal{H}_n$. Since n is arbitrary it follows that $\{\{f_k\}_{k=1}^{n+1}\}_{n=1}^{\infty}$ cannot have a finite basis constant.

Our starting point is the following

LEMMA 5. *Let n in $\mathbb{N}$ and define*

$$
\begin{aligned}
f_k &= e_k - \frac{1}{n}\sum_{j=1}^{n} e_j, k = 1, ..., n \\
f_{n+1} &= \frac{1}{\sqrt{n}}\sum_{j=1}^{n} e_j.
\end{aligned}
\tag{11}
$$

Then:

(i) For all f in $\mathcal{H}_n$ we have that $\sum_{k=1}^{n+1} |\langle f_k, f\rangle|^2 = \|f\|^2$. i.e. $(f_k)_{k=1}^{n+1}$ is a Parseval frame.

(ii) Assume that $n > 2$, and let $\{f_i\}_{i\in I}$ be any subset of $\{f_i\}_{i=1}^{n+1}$ for which span$\{f_i\}_{i\in I} = \mathcal{H}_n$. Then there is a subset $G \subset I$ such that

$$\|\sum_{i\in G} f_i\| \geq \frac{1}{4}\sqrt{n-2}\|\sum_{i\in I} f_i\|.$$

(iii) For any subset $\{k_1, \ldots, k_n\}$ of $\{1, \ldots, n+1\}$ such that $\{f_{k_i}\}_{i=1}^{n}$ spans $\mathcal{H}_n$, the basis constant of $\{f_{k_i}\}_{i=1}^{n}$ is greater than or equal to $\frac{1}{4}\sqrt{n-2}$.

PROOF. First observe that *(iii)* follows directly from *(ii)* and the definition of basis constant.

We first show that $\{f_k\}_{k=1}^{n+1}$ is a Parseval tight frame for $\mathcal{H}_n$. Let $f = \sum_{j=1}^{n} a_j e_j \in \mathcal{H}_n$.

$$
\begin{aligned}
\sum_{k=1}^{n+1} |\langle f, f_k\rangle|^2 &= \sum_{k=1}^{n} \left[|a_k|^2 |1 - \frac{1}{n}|^2 + \sum_{i\neq k}^{n} \frac{|a_i^2|}{n^2} \right] \\
&= \sum_{k=1}^{n} [\frac{|a_k|^2}{n} + |a_k|^2 - \frac{2|a_k|^2}{n} + \frac{|a_k|^2}{n^2} + \sum_{i\neq k}^{n} \frac{|a_i|^2}{n^2}] \\
&= \sum_{k=1}^{n} (1 - \frac{1}{n} + \frac{1}{n^2})|a_k|^2 + \sum_{k=1}^{n} (\frac{1}{n} - \frac{1}{n^2})|a_k|^2 \\
&= \sum_{k=1}^{n} |a_k|^2 = \|f\|^2
\end{aligned} \tag{12}
$$

We now prove (ii). Note that

$$
\begin{aligned}
\sum_{k=1}^{n} f_k &= \sum_{k=1}^{n} [e_k - \frac{1}{n} \sum_{j=1}^{n} e_j] \\
&= \sum_{k=1}^{n} e_k - \sum_{k=1}^{n} \frac{1}{n} \sum_{j=1}^{n} e_j = 0
\end{aligned} \tag{13}
$$

Therefore $(f_k)_{k=1}^n$ are linearly independent and so any subset of the frame $\{f_k\}_{k=1}^{n+1}$ which spans $\mathcal{H}_n$ must contain f_{n+1} and at least $n-1$ of the terms $\{f_k\}_{k=1}^n$. We will make a few easy observations to be used later. Let $l \in \{1, \ldots, n\}$. Then

$$
\| \sum_{k\neq l}^{n} f_k \| = \sum_{k=1}^{n} f_k - f_l \| = \|f_l\| = \sqrt{\frac{n-1}{n}}.
$$

Also

$$
\langle f_{n+1}, f_l \rangle = \left(1 - \frac{1}{n}\right) \frac{1}{\sqrt{n}} - \sum_{k\neq l}^{n} \frac{1}{\sqrt{n} n} = 0.
$$

We isolate the following claim.

Claim 6. *Let $\Gamma \subset \{1, \ldots, n\}$ such that $|\Gamma| \leqslant n/2$. Then*

$$
\| \sum_{k\in\Gamma} f_k \| \geqslant \frac{|\Gamma|^{1/2}}{2}.
$$

Let Γ be as in the claim. We have

$$
\begin{aligned}
\|\sum_{k\in\Gamma} f_k\| &= \|\sum_{k\in\Gamma}(e_k - \frac{1}{n}\sum_{j=1}^{n} e_j)\| = \|\sum_{k\in\Gamma} e_k - \frac{1}{n}\sum_{j=1}^{n} e_j|\Gamma|\| \\
&= \|\sum_{k\in\Gamma} e_k - \frac{|\Gamma|}{n}\Big(\sum_{k\in\Gamma} e_k - \sum_{k\notin\Gamma}^{n} e_k\Big)\| \\
&= \|\sum_{k\in\Gamma}\Big(1-\frac{|\Gamma|}{n}\Big)e_k - \frac{|\Gamma|}{n}\sum_{k\notin\Gamma} e_k\| \\
&> \Big(\sum_{k\in\Gamma}\Big(1-\frac{|\Gamma|}{n}\Big)^2\Big)^{\frac{1}{2}} = |\Gamma|^{\frac{1}{2}}\Big(1-\frac{|\Gamma|}{n}\Big) \geq \frac{|\Gamma|^{\frac{1}{2}}}{2}
\end{aligned}
\tag{14}
$$

This proves the claim.

Suppose that $\Lambda \subset \{1,\dots,n+1\}$ is such that $\{f_i\}_{i\in\Lambda}$ is a linear independent spanning set. By our above observation $n+1 \in \Lambda$. We claim that $\|\sum_{k\in\Lambda} f_k\| \leq \sqrt{2}$. Find the unique $l \in \Lambda \setminus \{1,\dots,n\}$. Using the fact that $\langle f_{n+1}, f_l\rangle = 0$ and the above estimates we have

$$
\|\sum_{k\in\Lambda} f_k\| = \Big(\|f_{n+1}\|^2 + \|\sum_{k\in\Lambda\setminus\{1,\dots,n\}} f_k\|^2\Big)^{1/2} = (1+\frac{n-1}{n})^{1/2} \leq \sqrt{2}.
$$

Now consider a subset of G of Λ such that

$$
\frac{n}{2} - 1 \leq |G| \leq \frac{n}{2}.
$$

We claim that

$$
\|\sum_{k\in G} f_k\| \geq \frac{1}{4}\sqrt{n-2}\|\sum_{k\in\Lambda} f_k\|.
$$

This confirms item (ii). We have two cases.

Case 1 $f_{n+1} \notin \{f_k\}_{k\in G}$:

Using the claim for $G = \Gamma$ and the lower bound on $|G|$ we have

$$
\|\sum_{k\in G} f_k\| \geq \frac{1}{2}\sqrt{\frac{n}{2}-1}
$$

Combining this the with upper bound we have the desired inequality.

Case 2 $f_{n+1} \in \{f_k\}_{k\in\Lambda}$:

In this case let $\Gamma = G \cap \{1, \ldots, n\}$. Using the claim (for the set Γ) and the cardinality of G we have

$$\| \sum_{k \in \Gamma} f_k \| \geq \frac{|\Gamma|^{1/2}}{2} \geq \frac{1}{2}\sqrt{\frac{n}{2} - 2}.$$

Using this inequality and the fact that $\langle f_{n+1}, f_k \rangle = 0$ for all $k \in \{1, \ldots, n\}$, we have

$$\begin{aligned} \| \sum_{k \in G} f_k \| &= (\|f_{n+1}\|^2 + \| \sum_{k \in \Gamma} f_k \|^2)^{\frac{1}{2}} \\ &\geq (\| \frac{1}{\sqrt{n}} \sum_{j=1}^{n} e_j \|^2 + [\frac{1}{2}\sqrt{\frac{n}{2} - 2}]^2)^{\frac{1}{2}} \\ &= (1 + \frac{1}{4}(\frac{n}{2} - 2))^{1/2} \\ &> \frac{1}{2}\sqrt{\frac{n}{2} - 1}. \end{aligned} \tag{15}$$

Therefore, as before, we have the desired inequality. □

We are finally ready to prove Thereom 4 (2).

Proof. Define the desired frame as follows. Let $f_1^1 = e_1$, $f_1^2 = e_2$ and $f_2^2 = e_3$. For each $n \in \mathbb{N}$ with $n > 2$ let

$$\begin{aligned} f_k^n &= e_{\frac{(n-1)n}{2}+k} - \frac{1}{n} \sum_{j=1}^{n} e_{\frac{(n-1)n}{2}+j} \text{ for } 1 \leq k \leq n, \\ f_{n+1}^n &= \frac{1}{\sqrt{n}} \sum_{j=1}^{n} e_{\frac{(n-1)n}{2}+j}. \end{aligned} \tag{16}$$

We now show that $\{\{f_k^n\}_{k=1}^{n+1}\}_{n=3}^{\infty} \cup \{f_1^1, f_1^2, f_2^2\}$ is a Parseval frame for $\mathcal{H}$. Note that $\text{span}\{f_k^{n+1}\}$ can be identified with a 'copy' of $\mathcal{H}_n$ spanned by $\{e_k\}_{k=\frac{(n-1)n}{2}+1}^{\frac{n(n+1)}{2}}$. Let $\mathcal{E}_n$ denote this subspace of $\mathcal{H}$

By the previous proposition, for $n > 2$, $\{f_k^n\}_{k=1}^{n+1}$ is a Parseval frame for $\mathcal{E}_n$. Using this we can see that $\{\{f_k^n\}_{k=1}^{n+1}\}_{n=3}^{\infty} \cup \{f_1^1, f_1^2, f_2^2\}$ is a Parseval frame for $\mathcal{H}$. Let $g \in \mathcal{H}$. Let g_n denote the vector g restricted the coordinates in $\mathcal{E}_n$. Observe that

$$\sum_{n=1}^{3}\sum_{k=1}^{n}|\langle g,f_k^n\rangle|^2+\sum_{n=3}^{\infty}\sum_{k=1}^{n+1}|\langle g,f_k^n\rangle|^2=\sum_{n=1}^{3}\sum_{k=1}^{n}|\langle g_n,f_k^n\rangle|^2+\sum_{n=3}^{\infty}\sum_{k=1}^{n+1}|\langle g_n,f_k^n\rangle|^2$$
$$=\sum_{n=1}^{\infty}\|g_n\|_{\mathcal{H}_n}^2=\|g\|^2.$$

Therefore we have a Parseval frame. Also note that for $n>2$ and $1\leq k\leq n$,

$$\begin{aligned}\|f_k^n\|&=\|(1-\frac{1}{n})e_{\frac{(n-1)n}{2}+k}-\frac{1}{n}\sum_{j\neq k}e_{\frac{(n-1)n}{2}+j}\|\\&=\sqrt{(1-\frac{1}{n})^2+\frac{n-1}{n^2}}\\&\geq\frac{1}{4},\end{aligned}\tag{17}$$

and the other three frame vectors have norm one.

Now suppose that $\{h_k\}_{k=1}^{\infty}$ is a spanning subset. Let $C\geq 1$ and find $n\in\mathbb{N}$ such that $\frac{1}{4}\sqrt{n-2}\geq C$. Since $\{h_k\}_{k=1}^{\infty}$ spans $\mathcal{H}$, it must also span $\mathcal{E}_n$. Therefore there is an set I with cardinality at least n and not greater than $n+1$ such that $\{h_k\}_{k\in I}$ spans $\mathcal{E}_n$. Using Lemma 5 (ii) we know that there is a $G\subset I$ such that

$$\|\sum_{i\in G}h_i\|\geq\frac{1}{4}\sqrt{n-2}\|\sum_{i\in I}h_i\|.$$

By considering the coefficients that are one on the set I and zero off I we see that $\{h_k\}_{k=1}^{\infty}$ must have basis constant greater than $\frac{1}{4}\sqrt{n-2}$. This proves that $\{h_k\}_{k=1}^{\infty}$ is not a Schauder basis for $\mathcal{H}$. □

Chapter 4

Application of Frames To Discretized Data

The idea for performing a software simulation was originally inspired by the work in the paper titled "Quantized Frame Expansions with Erasures" [**4**], where the standard XOR is compared with a frame. The goals of the software component of this paper will be to gain a practical and working knowledge of the algorithmic application of frame theory within computer software. In this way I can directly apply the theory explored earlier and learn directly what are common implementation limitations and pitfalls. I decided to focus on the Discrete Fourier Transform frame as it is a Parseval frame which means it has a very direct implementation resulting from the fact that the frame operator is just the identity function. Since this is self-adjoint and has an inverse this allows for a straight forward calculation of the frame coefficients and for a relatively computationally cheap way to retrieve the original vector from those coefficients.

The first thing I realized when I approached the problem of implementing the Discrete Fourier Transform frame was that I'd need a fast and reliable framework capable of performing complex vector algebra. I choose to implement my own in C++. This provided me with the speed I needed to be able to work with large files quickly and reliably. Next I encountered the problem of accuracy within my floating point calculations. By moving up to 64-bit floating point components for my complex scalars I was able to work past most of this but even with such a high degree

of accuracy it was still necessary in the end to define an epsilon neighborhood around the discrete values of interest for which I would allow the computer to accurately round back to the appropriate integer after applying the frame operator.

With the frame simulation software completed I am now able to directly compare the data redundancy capabilities of the Discrete Fourier Frame with a traditional direct transmission where no other data redundancy or parity scheme is in use. The simulation software accepts a loss percentage as a parameter at run time and randomly will erase coefficients to simulate loss in transmission. To provide a visual comparison for this paper I found the following test image to work with.

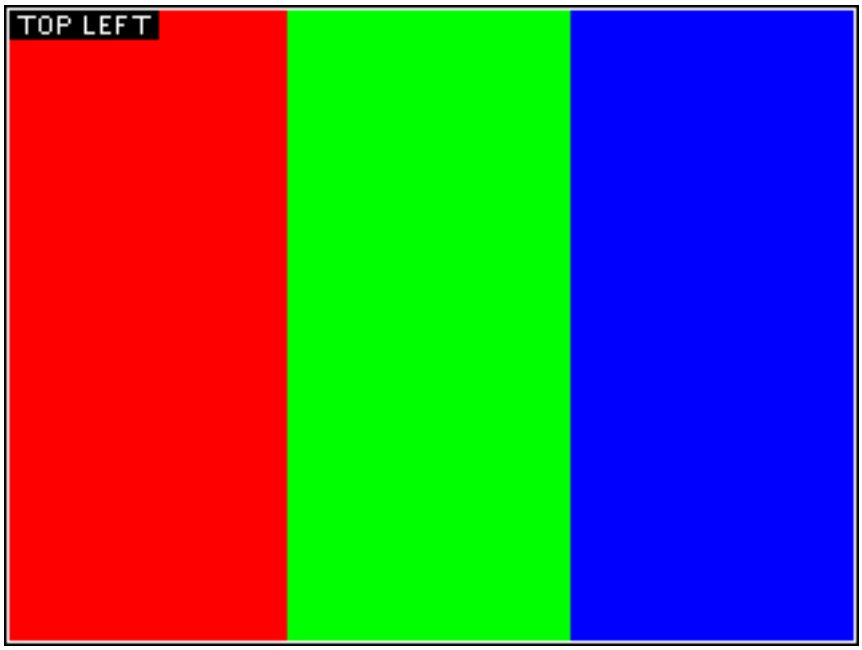

Figure 1 - Unaltered Test Image

Now let us introduce loss of 5% to this image in transmission and compare

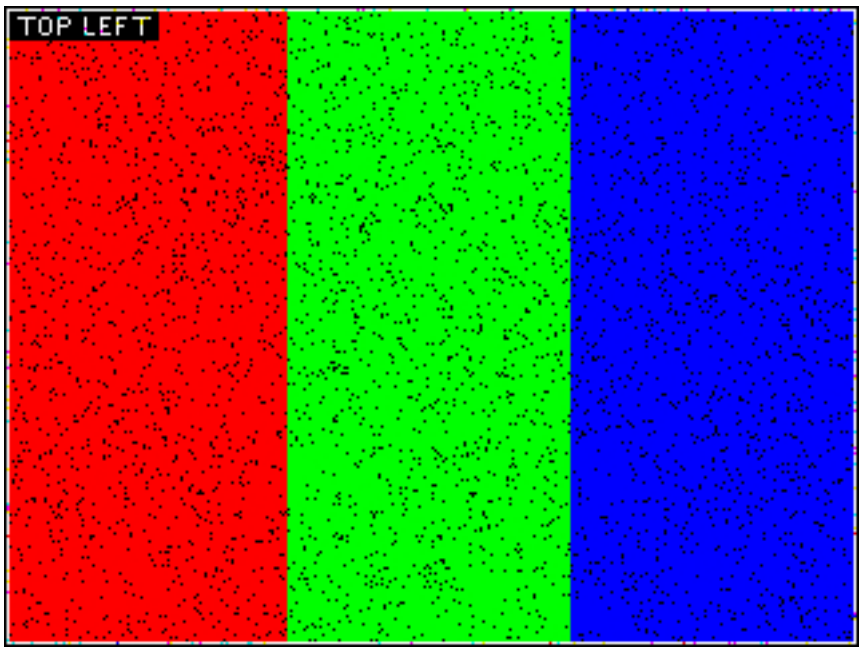

Figure 2 - Usual Orthonormal Basis with 10 dimensional chunks and 5% loss.

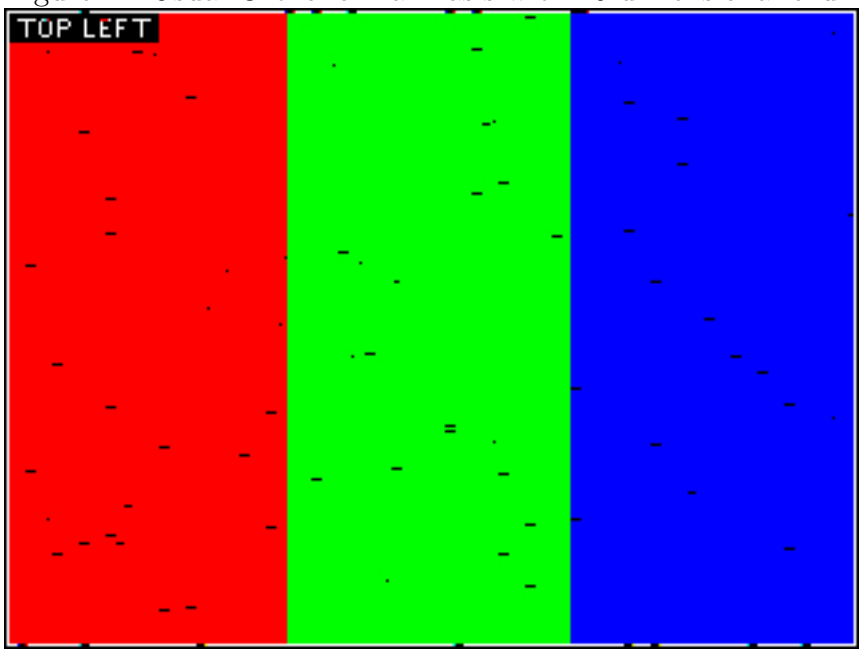

Figure 3 - Discrete Fourier Transfer 15 vector frame with 10 dimensional chunks and 5% loss.

Now let us introduce loss of 20% to this image in transmission and compare

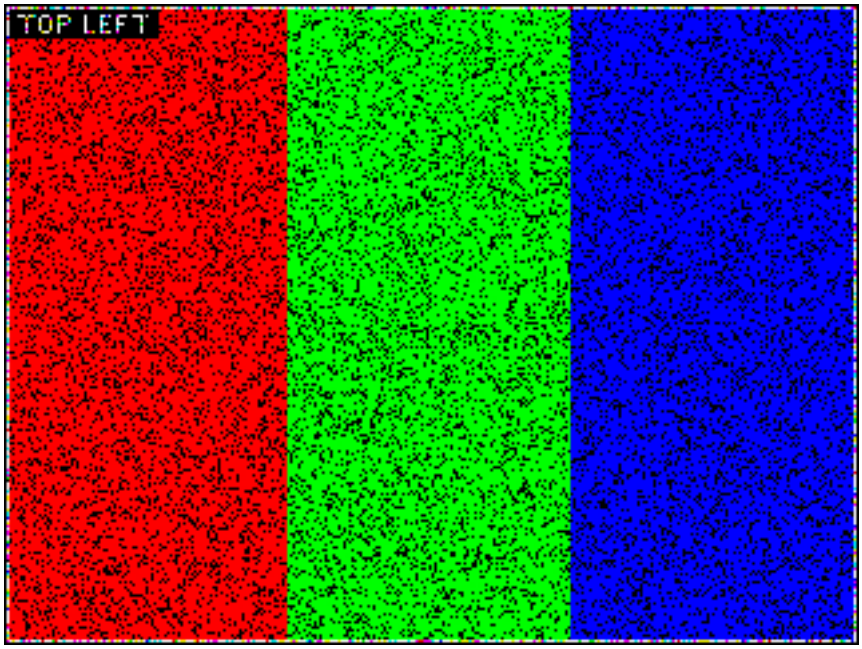

Figure 4 - Usual Orthonormal Basis with 10 dimensional chunks and 20% loss.

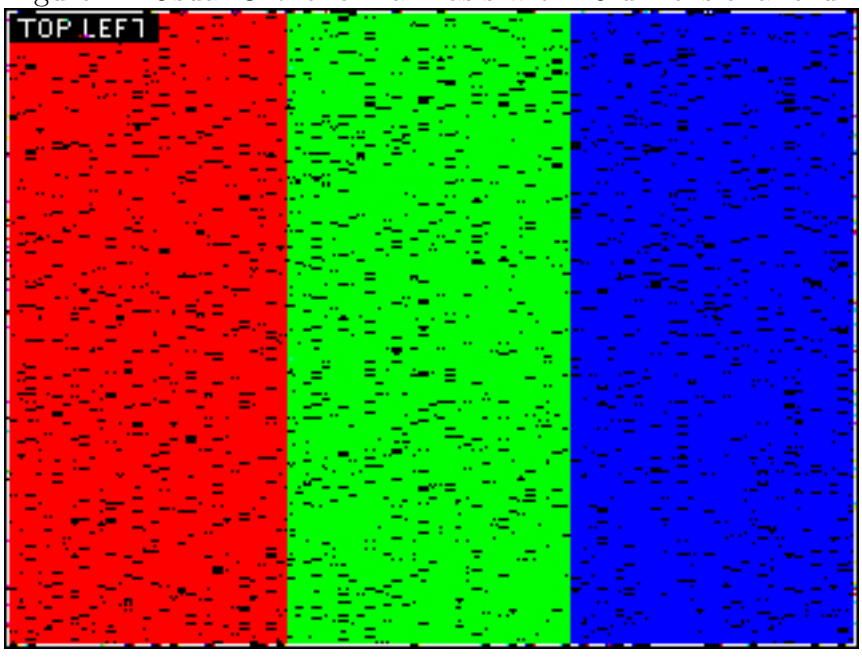

Figure 5 - Discrete Fourier Transfer 15 vector frame with 10 dimensional chunks and 20% loss.

Clearly from the test image results it can be seen there are great benefits to applying a frame and transmitting the coefficients, especially in an environment when you're expecting a level of data loss. This is exactly the case in most digital wireless transmission such as with cellular phones or radio modems. It should be noted though that even though the Discrete Fourier Transform frame drastically reduced the amount of noise introduced into our transmitted image by the random loss that there is still noise. This is a suitable scheme for reducing the apparent effects of loss at the receiving end when slight variance is acceptable in place of precision. This I found with my experiments as images, audio and video work well but text and discrete file headers suffer greatly for even the smallest amounts of disturbance.

It should also be noted that the computation for the frame coefficients for the Discrete Fourier Transform frame is fairly expensive and as they are complex values the transmission of them does require a fair amount of bandwidth overhead in addition to the CPU resources utilized on both the sender and receiver machines. I would not specifically recommend this frame for this purpose but would encourage the exploration of a more specialized tight frame. It should also be mentioned that there is potential for the application of frames in data encryption. For instance if one was to take the Discrete Fourier Frame as applied within the simulation software and populate the imaginary components of the data vectors with random data seeded by the system time. It would be possible to create an encryption algorithm that always generates slightly different output data but could always be decrypted to original data.

Bibliography

[1] Casazza, P.G. and Christensen, O. *Frames and Schauder bases* Approximation Theory: In Memory of A. K. Varna 133-139 (eds. Govil, N.K., Mohapatra, R. n., Nashed, Z., Sharma, A., Szabados, J.) Marcel Dekker, New York, 1998

[2] Peter G. Casazza and Janet C. Tremain *A Brief Introduction to Hilbert Space Frame Theory and its Applications* Frame Research Center Department of Mathematics University of Missouri, Columbia, MO

[3] Ole Christensen *An Introduction to Frames and Riesz Bases* Birkhäuser Boston, Basel, Berlin 2003

[4] Vivek K. Goyal and Jelena Kovačevič, *Quantized Frame Expansions with Erasures* Applied and Computational Harmonic Analysis 10, 203-233(2001)

[5] Robert E. Megginson, *An Introduction to Banach Space Theory.* Springer-Verlag New York, 1998

[6] Michael Reed, Barry Simon, *Functional Analysis.* Academic Press New York and London, 1972

Appendices

Chapter A

C++ Source Code

Listing A.1. Entry Point of Simulation software

```
#include <stdio.h>
#include <fstream>
#include <list>
#include <vector>
#include <time.h>

#include "DiscreteFourierTransformFrame.h"
#include "UsualOrthonormalBasis.h"
#include "Vector.h"

#define DFT_OUTPUT_FILE      "dft-n%d-k%d-loss%d-%s"
#define UOB_OUTPUT_FILE      "uob-n%d-k%d-loss%d-%s"

#define DFT_FRAME_TYPE 1
#define UOB_FRAME_TYPE 2

#define MACHINE_EPSILON .5

#define HEADER_SIZE 256
```

```
void Run_DFT_Simulation(unsigned nNumberOfFrames, unsigned
    nNumberOfDimensions, char* szInputFilename, unsigned nLossPercent
    = 0, unsigned nHeaderSize = HEADER_SIZE)
{
  std::ifstream inStream;
  std::vector<Vector*> lDFTCoefficients;
  Vector* pCoefficientChunk = NULL;
  Vector* pRawData = NULL;
  char  szOutputFilename[256];

    printf("Initializing Frames...\n");

  /* Initialize Frames */
  DFT_Frame oDFTFrame(nNumberOfDimensions, nNumberOfFrames);

  /* Read data and get coefficients */
  printf("Reading data from disk and calculating frame coefficients
      ...\n");

  oDFTFrame.Print();

  inStream.open(szInputFilename,std::ios::in | std::ios::binary);

  if(inStream.is_open())
  {
    printf("Successfully opened input data file...\n");
  }
  else
  {
    printf("Error opening input data file!\n");
```

```
  return;
}

printf("\n\n\n\n");

unsigned int nIterations = 0;

do
{
  pRawData = new Vector(nNumberOfDimensions);

  for (unsigned i = 1; i <= nNumberOfDimensions; i++)
  {
    char cData;

    inStream.read(&cData, sizeof(char));

    (*pRawData)[i] = ComplexScalar(cData);

    printf("%d", cData);

    if(inStream.eof()) break;
  }

  pCoefficientChunk = new Vector(nNumberOfFrames);

  oDFTFrame.GetFrameCoefficients(*pCoefficientChunk, *pRawData);

  /* Introduce artifical loss*/
  if( nIterations * nNumberOfDimensions > nHeaderSize )
```

```
  {
    for(unsigned i = 1; i <= nNumberOfFrames; i++)
    {
      if(((rand() % 100)+1) <= nLossPercent)
      {
        (*pCoefficientChunk)[i] = ComplexScalar(0,0);
      }
    }
  }
  else
  {
    ++nIterations;
  }


  lDFTCoefficients.push_back(pCoefficientChunk);


  delete pRawData;


  pRawData = NULL;
}
while (!inStream.eof());


inStream.close();


/* Application of Inverse */
std::vector<Vector*> lDFTInvCoefficients;


for(std::vector<Vector*>::iterator itCurrent = lDFTCoefficients.
   begin();
```

```
  itCurrent != lDFTCoefficients.end(); itCurrent++)
{
  Vector* pData = new Vector(nNumberOfDimensions);
  oDFTFrame.ApplyInverse(*pData, **itCurrent);
  lDFTInvCoefficients.push_back(pData);
}

/* Write new files to disk */
printf("\n\n\n\n");
printf("DFT\n");

std::ofstream outDFTStream;

sprintf(szOutputFilename, DFT_OUTPUT_FILE,nNumberOfDimensions,
    nNumberOfFrames, nLossPercent, szInputFilename);

outDFTStream.open(szOutputFilename,std::ios::out | std::ios::binary
    );

for(std::vector<Vector*>::iterator itCurrent = lDFTInvCoefficients.
    begin();
  itCurrent != lDFTInvCoefficients.end(); itCurrent++)
{
  for(unsigned i = 1; i <= nNumberOfDimensions; i++)
  {
    char  cTemp = 0;
    double dTemp = 0;

    dTemp = (**itCurrent)[i].GetRealComponent();
```

```
      if(dTemp > 0)
      {
        cTemp = dTemp+MACHINE_EPSILON;
      }
      else
      {
        cTemp = dTemp-MACHINE_EPSILON;
      }

      printf("%d", cTemp);

      outDFTStream.write(&cTemp,sizeof(char) );
    }
  }

  outDFTStream.close();
}



void Run_UOB_Simulation(unsigned nNumberOfFrames, unsigned
   nNumberOfDimensions, char* szInputFilename, unsigned nLossPercent
   = 0, unsigned nHeaderSize = HEADER_SIZE)
{
  std::ifstream inStream;
  std::vector<Vector*> lCoefficients;
  Vector* pCoefficientChunk = NULL;
  Vector* pRawData = NULL;
  char  szOutputFilename[256];
```

```
// insert code here...
  printf("Initializing Frames...\n");

/* Initialize Frames */
UOB_Frame oUOBFrame(nNumberOfDimensions, nNumberOfFrames);

/* Read data and get coefficients */
printf("Reading data from disk and calculating frame coefficients
    ...\n");

oUOBFrame.Print();

inStream.open(szInputFilename);

if(inStream.is_open())
{
  printf("Successfully opened input data file...\n");
}
else
{
  printf("Error opening input data file!\n");
  return;
}

printf("\n\n\n\n");

unsigned nIterations = 0;

do
{
```

```
pRawData = new Vector(nNumberOfDimensions);

for (unsigned i = 1; i <= nNumberOfDimensions; i++)
{
  char cData;

  inStream.get(cData);

  (*pRawData)[i] = ComplexScalar(cData);

  printf("%d", cData);

  if(inStream.eof()) break;
}

pCoefficientChunk = new Vector(nNumberOfFrames);

oUOBFrame.GetFrameCoefficients(*pCoefficientChunk, *pRawData);

/* Introduce artifical loss*/
if( nIterations * nNumberOfDimensions > nHeaderSize )
{
  for(unsigned i = 1; i <= nNumberOfFrames; i++)
  {
    if(((rand() % 100)+1) <= nLossPercent)
    {
      (*pCoefficientChunk)[i] = ComplexScalar(0,0);
    }
  }
}
```

```
  else
  {
    ++nIterations;
  }

  lCoefficients.push_back(pCoefficientChunk);

  delete pRawData;

  pRawData = NULL;
}
while (!inStream.eof());

inStream.close();

/* Application of Inverse */
std::vector<Vector*> lInvCoefficients;

for(std::vector<Vector*>::iterator itCurrent = lCoefficients.begin
    ();
  itCurrent != lCoefficients.end(); itCurrent++)
{
  Vector* pData = new Vector(nNumberOfDimensions);
  oUOBFrame.ApplyInverse(*pData, **itCurrent);
  lInvCoefficients.push_back(pData);
}

/* Write new files to disk */
printf("\n\n\n\n");
printf("UOB\n");
```

```
  std::ofstream outStream;

  sprintf(szOutputFilename, UOB_OUTPUT_FILE,nNumberOfDimensions,
     nNumberOfFrames, nLossPercent, szInputFilename);

  outStream.open(szOutputFilename);

  for(std::vector<Vector*>::iterator itCurrent = lInvCoefficients.
     begin();
    itCurrent != lInvCoefficients.end(); itCurrent++)
  {
    for(unsigned i = 1; i <= nNumberOfDimensions; i++)
    {
      char cTemp = 0;

      cTemp = (**itCurrent)[i].GetRealComponent();

      outStream.put(cTemp);

      printf("%d",cTemp);
    }
  }

  outStream.close();
}

int main (int argc, const char * argv[])
{
```

```
if(argc != 6)
{
  printf("Usuage: Frame_Simulator [# of frames] [# of dimensions] [
      Frame Type] [Loss Percent] [filename]\n\n");
  printf("Frame Types 1-DFT 2-UOB\n\n");
  return 0;
}

char  szInputFilename[256];

/* initialize random seed: */
srand ( time(NULL) );

sprintf(szInputFilename, "%s", argv[5]);

unsigned nNumberOfFrames = atoi(argv[1]);
unsigned nNumberOfDimensions = atoi(argv[2]);
unsigned nFrameType = atoi(argv[3]);
unsigned nLossPercent = atoi(argv[4]);

if(nFrameType == DFT_FRAME_TYPE)
{
  Run_DFT_Simulation(nNumberOfFrames, nNumberOfDimensions,
      szInputFilename, nLossPercent);
}
else if(nFrameType == UOB_FRAME_TYPE)
{
  Run_UOB_Simulation(nNumberOfFrames, nNumberOfDimensions,
      szInputFilename, nLossPercent);
}
```

```
    else
    {
      printf("Invalid Frame Type.\n");
    }


      return 0;
  }
```

Listing A.2. Complex Scalar Header

```
#pragma once

/*
 *  ComplexScalar.h
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include <stdio.h>

class UOB_FrameVector ;
class DFT_FrameVector ;

class ComplexScalar
{
public:
  ComplexScalar();
  ComplexScalar(const ComplexScalar & oComplexScalar);
  ComplexScalar(double fReal, double fImaginary = 0);
  ~ComplexScalar();
  double GetRealComponent();
  double GetImaginaryComponent();

  ComplexScalar GetComplexConjugate();
```

```
  ComplexScalar & operator=(const ComplexScalar &oRHS);
  ComplexScalar & operator+=(const ComplexScalar &oRHS);
  ComplexScalar & operator-=(const ComplexScalar &oRHS);
  ComplexScalar & operator*=(const ComplexScalar &oRHS);
  ComplexScalar & operator/=(const ComplexScalar &oRHS);

  const ComplexScalar operator+(const ComplexScalar & oRHS) const;
  const ComplexScalar operator-(const ComplexScalar & oRHS) const;
  const ComplexScalar operator*(const ComplexScalar & oRHS) const;
  const ComplexScalar operator/(const ComplexScalar & oRHS) const;

  bool operator==(const ComplexScalar & oRHS);
  bool operator!=(const ComplexScalar & oRHS);

  const UOB_FrameVector operator*(const UOB_FrameVector & oRHS) const
     ;
  const UOB_FrameVector operator/(const UOB_FrameVector & oRHS) const
     ;

  const DFT_FrameVector operator*(const DFT_FrameVector & oRHS) const
     ;
  const DFT_FrameVector operator/(const DFT_FrameVector & oRHS) const
     ;


  void Print();
private:
  double m_fReal;
  double m_fImaginary;
};
```

Listing A.3. Complex Scalar Source

```
/*
 *  ComplexScalar.cpp
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include <stdio.h>

#include "ComplexScalar.h"
#include "UsualOrthonormalBasis.h"
#include "DiscreteFourierTransformFrame.h"

ComplexScalar::ComplexScalar()
: m_fReal(0),
  m_fImaginary(0)
{
}

ComplexScalar::~ComplexScalar()
{
}

ComplexScalar::ComplexScalar(const ComplexScalar & oComplexScalar)
: m_fReal(0),
  m_fImaginary(0)
```

```
{
  m_fReal = oComplexScalar.m_fReal;
  m_fImaginary = oComplexScalar.m_fImaginary;
}

ComplexScalar::ComplexScalar(double fReal, double fImaginary)
: m_fReal(0),
  m_fImaginary(0)
{
  m_fReal = fReal;
  m_fImaginary = fImaginary;
}

double ComplexScalar::GetRealComponent()
{
  return m_fReal;
}

double ComplexScalar::GetImaginaryComponent()
{
  return m_fImaginary;
}

ComplexScalar & ComplexScalar::operator=(const ComplexScalar & oRHS)
{
  this->m_fReal = oRHS.m_fReal;
  this->m_fImaginary = oRHS.m_fImaginary;

  return *this;
}
```

```
ComplexScalar & ComplexScalar::operator+=(const ComplexScalar & oRHS)
{
  this->m_fReal += oRHS.m_fReal;
  this->m_fImaginary += oRHS.m_fImaginary;

  return *this;
}

ComplexScalar & ComplexScalar::operator-=(const ComplexScalar & oRHS)
{
  this->m_fReal -= oRHS.m_fReal;
  this->m_fImaginary -= oRHS.m_fImaginary;

  return *this;
}

ComplexScalar & ComplexScalar::operator*=(const ComplexScalar & oRHS)
{
  double fReal = (this->m_fReal * oRHS.m_fReal) - (this->m_fImaginary
       * oRHS.m_fImaginary);
  double fImaginary = (this->m_fReal * oRHS.m_fImaginary) + (this->
      m_fImaginary * oRHS.m_fReal);
  this->m_fReal = fReal;
  this->m_fImaginary = fImaginary;

  return *this;
}

ComplexScalar & ComplexScalar::operator/=(const ComplexScalar & oRHS)
```

```
{
  double fReal = ((this->m_fReal * oRHS.m_fReal) + (this->
     m_fImaginary * oRHS.m_fImaginary))/(oRHS.m_fReal * oRHS.m_fReal
     + oRHS.m_fImaginary * oRHS.m_fImaginary );
  double fImaginary = ((this->m_fImaginary * oRHS.m_fReal) - (oRHS.
     m_fImaginary * this->m_fReal) )/(oRHS.m_fReal * oRHS.m_fReal +
     oRHS.m_fImaginary * oRHS.m_fImaginary );
  this->m_fReal = fReal;
  this->m_fImaginary = fImaginary;

  return *this;
}

const ComplexScalar ComplexScalar::operator+(const ComplexScalar &
    oRHS) const
{
  return ComplexScalar(*this) += oRHS;
}

const ComplexScalar ComplexScalar::operator-(const ComplexScalar &
    oRHS) const
{
  return ComplexScalar(*this) -= oRHS;
}

const ComplexScalar ComplexScalar::operator*(const ComplexScalar &
    oRHS) const
{
  return ComplexScalar(*this) *= oRHS;
}
```

```
const ComplexScalar ComplexScalar::operator/(const ComplexScalar &
    oRHS) const
{
  return ComplexScalar(*this) /= oRHS;
}

bool ComplexScalar::operator==(const ComplexScalar & oRHS)
{
  if ((m_fReal == oRHS.m_fReal) && (m_fImaginary == oRHS.m_fImaginary
      )) {
    return true;
  }
  else
  {
    return false;
  }
}

bool ComplexScalar::operator!=(const ComplexScalar & oRHS)
{
  return !(*this == oRHS);
}

const UOB_FrameVector ComplexScalar::operator*(const UOB_FrameVector
    & oRHS) const
{
  return oRHS*(*this);
}
```

```
const UOB_FrameVector ComplexScalar::operator/(const UOB_FrameVector
    & oRHS) const
{
  return oRHS/(*this);
}

const DFT_FrameVector ComplexScalar::operator*(const DFT_FrameVector
    & oRHS) const
{
  return oRHS*(*this);
}

const DFT_FrameVector ComplexScalar::operator/(const DFT_FrameVector
    & oRHS) const
{
  return oRHS/(*this);
}


void ComplexScalar::Print()
{
  printf("%f i%f, ", m_fReal, m_fImaginary);
}

ComplexScalar ComplexScalar::GetComplexConjugate()
{
  return ComplexScalar(m_fReal, (-1*m_fImaginary));
}
```

Listing A.4. Vector Header

```
#pragma once

/*
 *  Vector.h
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include <stdio.h>

#include "ComplexScalar.h"

class DFT_FrameVector ;

class Vector
{
public:
  Vector();
  Vector(unsigned n);
  Vector(const Vector & oRHS);
  ~Vector();
  ComplexScalar & operator[](unsigned nIndex);
  Vector & operator=(const Vector & oRHS);
  Vector & operator+=(const Vector& oRHS);
  Vector & operator+=(const DFT_FrameVector& oRHS);
```

```
  Vector & operator-=(const Vector & oRHS);
  Vector & operator*=(const ComplexScalar & oRHS);
  Vector & operator/=(const ComplexScalar & oRHS);

  const Vector operator+(const Vector & oRHS) const;
  const Vector operator+(const DFT_FrameVector & oRHS) const;
  const Vector operator-(const Vector & oRHS) const;
  const Vector operator*(const ComplexScalar & oRHS) const;
  const Vector operator/(const ComplexScalar & oRHS) const;

  bool operator==(const Vector & oRHS);
  bool operator!=(const Vector & oRHS);

  void Print();
private:
  unsigned m_n;
  ComplexScalar* m_aScalarArray;
};
```

Listing A.5. Vector Source

```
/*
 *  Vector.cpp
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */
#include <assert.h>

#include "Vector.h"
#include "DiscreteFourierTransformFrame.h"

Vector::Vector()
: m_aScalarArray(NULL),
  m_n(0)
{
}

Vector::Vector(unsigned n)
: m_aScalarArray(NULL),
  m_n(0)
{
  m_n = n;

  m_aScalarArray = new ComplexScalar[n];
}
```

```
Vector::Vector(const Vector & oRHS)
: m_n(0),
m_aScalarArray(NULL)
{
  *this = oRHS;
}

Vector::~Vector()
{
  if(m_aScalarArray != NULL)
  {
    delete [] m_aScalarArray;
  }
}

ComplexScalar & Vector::operator[](unsigned nIndex)
{
  assert((nIndex > 0) && (nIndex <= m_n));
  return m_aScalarArray[nIndex-1];
}

Vector & Vector::operator=(const Vector & oRHS)
{
  m_n = oRHS.m_n;

  if(m_aScalarArray != NULL)
  {
    delete [] m_aScalarArray;
  }
```

```
  m_aScalarArray = new ComplexScalar[m_n];

  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = oRHS.m_aScalarArray[k];
  }

  return *this;
}

Vector & Vector::operator+=(const Vector & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] += oRHS.m_aScalarArray[k];
  }

  return *this;
}

Vector & Vector::operator+=(const DFT_FrameVector & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] += oRHS.m_aScalarArray[k];
  }

  return *this;
}
```

```
Vector & Vector::operator-=(const Vector & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] -= oRHS.m_aScalarArray[k];
  }

  return *this;
}

Vector & Vector::operator*=(const ComplexScalar & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = (m_aScalarArray[k] * oRHS);
  }

  return *this;
}

Vector & Vector::operator/=(const ComplexScalar & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = (m_aScalarArray[k] / oRHS);
  }

  return *this;
}
```

```
const Vector Vector::operator+(const Vector & oRHS) const
{
  return Vector(*this) += oRHS;
}

const Vector Vector::operator+(const DFT_FrameVector & oRHS) const
{
  return Vector(*this) += oRHS;
}

const Vector Vector::operator-(const Vector & oRHS) const
{
  return Vector(*this) -= oRHS;
}

const Vector Vector::operator*(const ComplexScalar & oRHS) const
{
  return Vector(*this) *= oRHS;
}

const Vector Vector::operator/(const ComplexScalar & oRHS) const
{
  return Vector(*this) /= oRHS;
}

bool Vector::operator==(const Vector & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
```

```
    if (m_aScalarArray[k] != oRHS.m_aScalarArray[k])
    {
      return false;
    }
  }

  return true;
}

bool Vector::operator!=(const Vector & oRHS)
{
  return !(*this == oRHS);
}

void Vector::Print()
{
  for(unsigned i = 0; i < m_n; i++)
  {
    m_aScalarArray[i].Print();
  }
}
```

Listing A.6. Discrete Fourier Transform Header

```
#pragma once

/*
 *  DiscreteFourierTransformFrame.h
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include <stdio.h>

#include "ComplexScalar.h"
#include "Vector.h"
#include "UsualOrthonormalBasis.h"

class DFT_FrameVector
{
public:
  DFT_FrameVector();
  DFT_FrameVector(const DFT_FrameVector & oRHS);
  DFT_FrameVector(unsigned n, unsigned m, unsigned k);
  ~DFT_FrameVector();
  ComplexScalar & operator[](unsigned nIndex);
  DFT_FrameVector & operator=(const DFT_FrameVector & oRHS);
  DFT_FrameVector & operator+=(const DFT_FrameVector& oRHS);
  DFT_FrameVector & operator-=(const DFT_FrameVector & oRHS);
```

```
  DFT_FrameVector & operator*=(const ComplexScalar & oRHS);
  DFT_FrameVector & operator/=(const ComplexScalar & oRHS);

  const DFT_FrameVector operator+(const DFT_FrameVector & oRHS) const
     ;
  const DFT_FrameVector operator-(const DFT_FrameVector & oRHS) const
     ;
  const DFT_FrameVector operator*(const ComplexScalar & oRHS) const;
  const DFT_FrameVector operator/(const ComplexScalar & oRHS) const;

  bool operator==(const DFT_FrameVector & oRHS);
  bool operator!=(const DFT_FrameVector & oRHS);

  void Print();
private:
  friend class Vector;

  unsigned m_n;
  unsigned m_m;
  unsigned m_k;

  ComplexScalar* m_aScalarArray;
};

class DFT_Frame
{
public:
  DFT_Frame();
  ~DFT_Frame();
```

```
  DFT_Frame(unsigned nDimensions /* n */, unsigned nFrameCount /* k
     */);
  unsigned GetFrameCount();
  void GetFrameCoefficients(Vector & oCoefficients,  Vector & oVector
     );
  void ApplyInverse(Vector & oCoefficents, Vector & oVector);
  DFT_FrameVector & operator[](unsigned nIndex);

  void Print();

private:
  DFT_FrameVector** m_aFrameArray;

  UOB_Frame*        m_poFrameInverse;

  unsigned m_nFrameCount;
  unsigned m_nDimensions;
};
```

Listing A.7. Discrete Fourier Transform Source

```
/*
 *  DiscreteFourierTransformFrame.cpp
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include <stdio.h>
#include <math.h>
#include <assert.h>

#include "DiscreteFourierTransformFrame.h"
#include "UsualOrthonormalBasis.h"

DFT_FrameVector::DFT_FrameVector()
: m_k(0),
  m_n(0),
  m_m(0),
  m_aScalarArray(NULL)
{
}

DFT_FrameVector::DFT_FrameVector(const DFT_FrameVector & oRHS)
: m_k(0),
  m_n(0),
  m_m(0),
```

```
  m_aScalarArray(NULL)
{
  *this = oRHS;
}

DFT_FrameVector::DFT_FrameVector(unsigned n, unsigned m, unsigned k)
: m_k(0),
  m_n(0),
  m_m(0),
  m_aScalarArray(NULL)
{
  m_n = n;
  m_k = k;
  m_m = m;

  m_aScalarArray = new ComplexScalar[n];

  /* Populate Frame Array */

  for(unsigned i=0; i < n; i++)
  {
    double fReal;
    double fImaginary;

    fReal = (cos(2 * M_PI * i * (m_k - 1)/m_m))/sqrt(m_m);
    fImaginary = (sin(2 * M_PI * i * (m_k - 1)/m_m))/sqrt(m_m);

    m_aScalarArray[i] = ComplexScalar(fReal,fImaginary);
  }
```

```
}

DFT_FrameVector::~DFT_FrameVector()
{
  if(m_aScalarArray != NULL)
  {
    delete [] m_aScalarArray;
  }
}

ComplexScalar & DFT_FrameVector::operator[](unsigned nIndex)
{
  assert((nIndex > 0) && (nIndex <= m_n));
  return m_aScalarArray[nIndex-1];
}

DFT_FrameVector & DFT_FrameVector::operator=(const DFT_FrameVector &
   oRHS)
{
  m_n = oRHS.m_n;
  m_k = oRHS.m_k;
  m_m = oRHS.m_m;

  if(m_aScalarArray != NULL)
  {
    delete [] m_aScalarArray;
  }

  m_aScalarArray = new ComplexScalar[m_n];
```

```
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = oRHS.m_aScalarArray[k];
  }

  return *this;
}

DFT_FrameVector & DFT_FrameVector::operator+=(const DFT_FrameVector &
    oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] += oRHS.m_aScalarArray[k];
  }

  return *this;
}

DFT_FrameVector & DFT_FrameVector::operator-=(const DFT_FrameVector &
    oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] -= oRHS.m_aScalarArray[k];
  }

  return *this;
}
```

```
DFT_FrameVector & DFT_FrameVector::operator*=(const ComplexScalar &
    oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = (m_aScalarArray[k] * oRHS);
  }

  return *this;
}

DFT_FrameVector & DFT_FrameVector::operator/=(const ComplexScalar &
    oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = (m_aScalarArray[k] / oRHS);
  }

  return *this;
}

const DFT_FrameVector DFT_FrameVector::operator+(const
    DFT_FrameVector & oRHS) const
{
  return DFT_FrameVector(*this) += oRHS;
}

const DFT_FrameVector DFT_FrameVector::operator-(const
    DFT_FrameVector & oRHS) const
```

```
{
  return DFT_FrameVector(*this) -= oRHS;
}

const DFT_FrameVector DFT_FrameVector::operator*(const ComplexScalar
   & oRHS) const
{
  return DFT_FrameVector(*this) *= oRHS;
}

const DFT_FrameVector DFT_FrameVector::operator/(const ComplexScalar
   & oRHS) const
{
  return DFT_FrameVector(*this) /= oRHS;
}

bool DFT_FrameVector::operator==(const DFT_FrameVector & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    if (m_aScalarArray[k] != oRHS.m_aScalarArray[k])
    {
      return false;
    }
  }

  return true;
}

bool DFT_FrameVector::operator!=(const DFT_FrameVector & oRHS)
```

```
{
  return !(*this == oRHS);
}

void DFT_FrameVector::Print()
{
  for(unsigned i = 0; i < m_n; i++)
  {
    m_aScalarArray[i].Print();
  }
}

DFT_Frame::DFT_Frame()
: m_aFrameArray(NULL),
  m_nFrameCount(0),
  m_nDimensions(0),
  m_poFrameInverse(NULL)
{
}

DFT_Frame::DFT_Frame(unsigned nDimensions /* n */, unsigned
   nFrameCount /* k */)
: m_aFrameArray(NULL),
  m_nFrameCount(0),
  m_nDimensions(0),
  m_poFrameInverse(NULL)
{
  m_nFrameCount = nFrameCount;
  m_nDimensions = nDimensions;
```

```
  m_aFrameArray = new DFT_FrameVector*[nFrameCount];

  /* Populate Frame Array */

  for(unsigned k = 0; k < nFrameCount ; k++)
  {
    m_aFrameArray[k] = new DFT_FrameVector(nDimensions, nFrameCount,
        k+1);
  }

}

DFT_Frame::~DFT_Frame()
{
  for(unsigned k = 0; k < m_nFrameCount ; k++)
  {
    delete m_aFrameArray[k];
  }

  delete [] m_aFrameArray;
}

unsigned DFT_Frame::GetFrameCount()
{
  return m_nFrameCount;
}

/*
 GetFrameCoefficenents takes a vector of size equal to the number of
    frames as the first argument and
```

```
 as the second arguement a vector of size equal to the number of
    dimensions for the frame vectors. The
 coefficients will be placed in the first vector.
 */

void DFT_Frame::GetFrameCoefficients(Vector & oCoefficients, Vector &
    oVector)
{
  ComplexScalar oCoefficient;

  for(unsigned k = 0; k < m_nFrameCount; k++)
  {
    oCoefficient = ComplexScalar(0,0);

    for(unsigned i = 1; i < m_nDimensions+1; i++)
    {
      ComplexScalar oFrameComponent;
      ComplexScalar oVectorComponent;
      ComplexScalar oProduct;

      oFrameComponent = (*m_aFrameArray[k])[i].GetComplexConjugate();
      oVectorComponent = oVector[i];

      oProduct = oFrameComponent * oVectorComponent;
      oCoefficient += oProduct;
    }

    oCoefficients[k+1] = oCoefficient;
  }
}
```

```
void DFT_Frame::ApplyInverse(Vector & oCoefficients, Vector & oVector
   )
{
  for(unsigned k = 1; k < m_nDimensions+1; k++)
  {
    oCoefficients[k] = ComplexScalar(0,0);
  }

  for(unsigned k = 0; k < m_nFrameCount; k++)
  {
    oCoefficients += (oVector[k+1] * (*m_aFrameArray[k]));
  }

}

DFT_FrameVector & DFT_Frame::operator[](unsigned nIndex)
{
  assert((nIndex >0) && (nIndex <= m_nFrameCount));
  return *(m_aFrameArray[nIndex-1]);
}

void DFT_Frame::Print()
{
  printf("Printing DFT Frame...\n");

  for(unsigned i = 0; i < m_nFrameCount; i++)
  {
    m_aFrameArray[i]->Print();
    printf("\n");
```

```
    }

    printf("\n");
  }
```

Listing A.8. Usual Orthonormal Basis Header

```
#pragma once
/*
 *  UsualOrthonormalBasis.h
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include <stdio.h>

#include "ComplexScalar.h"
#include "Vector.h"

class UOB_FrameVector
{
public:
  UOB_FrameVector();
  UOB_FrameVector(const UOB_FrameVector & oRHS);
  UOB_FrameVector(unsigned n, unsigned k);
  ~UOB_FrameVector();

  ComplexScalar & operator[](unsigned nIndex);
  UOB_FrameVector & operator=(const UOB_FrameVector & oRHS);
  UOB_FrameVector & operator+=(const UOB_FrameVector& oRHS);
  UOB_FrameVector & operator-=(const UOB_FrameVector & oRHS);
  UOB_FrameVector & operator*=(const ComplexScalar & oRHS);
```

```
  UOB_FrameVector & operator/=(const ComplexScalar & oRHS);

  const UOB_FrameVector operator+(const UOB_FrameVector & oRHS) const
     ;
  const UOB_FrameVector operator-(const UOB_FrameVector & oRHS) const
     ;
  const UOB_FrameVector operator*(const ComplexScalar & oRHS) const;
  const UOB_FrameVector operator/(const ComplexScalar & oRHS) const;

  bool operator==(const UOB_FrameVector & oRHS);
  bool operator!=(const UOB_FrameVector & oRHS);

  void Print();
private:
  friend class Vector;

  unsigned m_n;
  unsigned m_k;
  ComplexScalar* m_aScalarArray;
};

class UOB_Frame
{
public:
  UOB_Frame();
  ~UOB_Frame();
  UOB_Frame(unsigned nDimensions /* n */, unsigned nFrameCount /* k
     */);
  unsigned GetFrameCount();
```

```
  void GetFrameCoefficients(Vector & oCoefficients,  Vector & oVector
     );
  void GetFrameCoefficients_SRC(Vector & oCoefficients, Vector &
     oVector);
  void ApplyInverse(Vector & oCoefficients, Vector & oVector);
  UOB_FrameVector & operator[](unsigned nIndex);

  void Print();
private:
  UOB_FrameVector** m_aFrameArray;

  unsigned m_nFrameCount;
  unsigned m_nDimensions;
};
```

Listing A.9. Usual Orthonormal Basis Source

```
/*
 *  UsualOrthonormalBasis.cpp
 *  Frame_Simulator
 *
 *  Created by Kinney Thompson on 4/22/12.
 *  Copyright 2012. All rights reserved.
 *
 */

#include "UsualOrthonormalBasis.h"

#include <stdio.h>
#include <math.h>
#include <assert.h>



UOB_FrameVector::UOB_FrameVector()
: m_k(0),
m_n(0),
m_aScalarArray(NULL)
{
}

UOB_FrameVector::UOB_FrameVector(const UOB_FrameVector & oRHS)
: m_k(0),
  m_n(0),
  m_aScalarArray(NULL)
{
```

```
  *this = oRHS;
}

UOB_FrameVector::UOB_FrameVector(unsigned n, unsigned k)
: m_k(0),
m_n(0),
m_aScalarArray(NULL)
{
  m_n = n;
  m_k = k;

  m_aScalarArray = new ComplexScalar[n];

  for(unsigned i=0; i < n; i++)
  {
    double fReal;

    fReal = (i+1 == k);

    m_aScalarArray[i] = ComplexScalar(fReal,0);
  }
}

UOB_FrameVector::~UOB_FrameVector()
{
  if(m_aScalarArray != NULL)
  {
    delete [] m_aScalarArray;
  }
}
```

```
ComplexScalar & UOB_FrameVector::operator[](unsigned nIndex)
{
  assert((nIndex > 0) && (nIndex <= m_n));
  return m_aScalarArray[nIndex-1];
}

UOB_FrameVector & UOB_FrameVector::operator=(const UOB_FrameVector &
   oRHS)
{
  m_n = oRHS.m_n;
  m_k = oRHS.m_k;

  if(m_aScalarArray != NULL)
  {
    delete [] m_aScalarArray;
  }

  m_aScalarArray = new ComplexScalar[m_n];

  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] = oRHS.m_aScalarArray[k];
  }

  return *this;
}

UOB_FrameVector & UOB_FrameVector::operator+=(const UOB_FrameVector &
    oRHS)
```

```
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] += oRHS.m_aScalarArray[k];
  }

  return *this;
}

UOB_FrameVector & UOB_FrameVector::operator-=(const UOB_FrameVector &
    oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] -= oRHS.m_aScalarArray[k];
  }

  return *this;
}

UOB_FrameVector & UOB_FrameVector::operator*=(const ComplexScalar &
   oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] *= oRHS;
  }

  return *this;
}
```

```
UOB_FrameVector & UOB_FrameVector::operator/=(const ComplexScalar &
    oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    m_aScalarArray[k] /= oRHS;
  }

  return *this;
}

const UOB_FrameVector UOB_FrameVector::operator+(const
    UOB_FrameVector & oRHS) const
{
  return UOB_FrameVector(*this) += oRHS;
}

const UOB_FrameVector UOB_FrameVector::operator-(const
    UOB_FrameVector & oRHS) const
{
  return UOB_FrameVector(*this) -= oRHS;
}

const UOB_FrameVector UOB_FrameVector::operator*(const ComplexScalar
    & oRHS) const
{
  return UOB_FrameVector(*this) *= oRHS;
}
```

```
const UOB_FrameVector UOB_FrameVector::operator/(const ComplexScalar
    & oRHS) const
{
  return UOB_FrameVector(*this) /= oRHS;
}

bool UOB_FrameVector::operator==(const UOB_FrameVector & oRHS)
{
  for(unsigned k = 0; k < m_n ; k++)
  {
    if (m_aScalarArray[k] != oRHS.m_aScalarArray[k])
    {
      return false;
    }
  }

  return true;
}

bool UOB_FrameVector::operator!=(const UOB_FrameVector & oRHS)
{
  return !(*this == oRHS);
}

void UOB_FrameVector::Print()
{
  for(unsigned i = 0; i < m_n; i++)
  {
    m_aScalarArray[i].Print();
  }
```

```
}

UOB_Frame::UOB_Frame()
: m_aFrameArray(NULL),
m_nFrameCount(0),
m_nDimensions(0)
{
}

UOB_Frame::UOB_Frame(unsigned nDimensions /* n */, unsigned
    nFrameCount /* k */)
: m_aFrameArray(NULL),
m_nFrameCount(0),
m_nDimensions(0)
{
  m_nFrameCount = nFrameCount;
  m_nDimensions = nDimensions;

  m_aFrameArray = new UOB_FrameVector*[nFrameCount];

  for(unsigned k = 0; k < nFrameCount ; k++)
  {
    m_aFrameArray[k] = new UOB_FrameVector(nDimensions, k+1);
  }
}

UOB_Frame::~UOB_Frame()
{
  for(unsigned k = 0; k < m_nFrameCount ; k++)
  {
```

```
    delete m_aFrameArray[k];
  }

  delete [] m_aFrameArray;
}

unsigned UOB_Frame::GetFrameCount()
{
  return m_nFrameCount;
}

/*
 GetFrameCoefficenents takes a vector of size equal to the number of
    frames as the first argument and
 as the second arguement a vector of size equal to the number of
    dimensions for the frame vectors. The
 coefficients will be placed in the first vector.
 */

void UOB_Frame::GetFrameCoefficients(Vector & oCoefficients, Vector &
    oVector)
{
  for(unsigned k = 0; k < m_nFrameCount; k++)
  {
    ComplexScalar oCoefficient = ComplexScalar(0,0);

    for(unsigned i = 1; i < m_nDimensions+1; i++)
    {
      oCoefficient = oCoefficient + (*m_aFrameArray[k])[i] * oVector[
          i].GetComplexConjugate();
```

```
    }

    oCoefficients[k+1] = oCoefficient;
  }
}

void UOB_Frame::ApplyInverse(Vector & oCoefficients, Vector & oVector
   )
{
  GetFrameCoefficients(oCoefficients, oVector);
}

void UOB_Frame::GetFrameCoefficients_SRC(Vector & oCoefficients,
   Vector & oVector)
{
  for(unsigned k = 1; k < m_nDimensions+1; k++)
  {
    ComplexScalar oCoefficient = ComplexScalar(0,0);

    for(unsigned i = 1; i < m_nFrameCount+1; i++)
    {
      oCoefficient = oCoefficient + (*m_aFrameArray[i-1])[k] *
          oVector[i].GetComplexConjugate();
    }

    oCoefficients[k] = oCoefficient;
  }
}

UOB_FrameVector & UOB_Frame::operator[](unsigned nIndex)
```

```
{
  assert((nIndex >0) && (nIndex <= m_nFrameCount));
  return *(m_aFrameArray[nIndex-1]);
}

void UOB_Frame::Print()
{
  printf("Printing UOB...\n");

  for(unsigned i = 0; i < m_nFrameCount; i++)
  {
    m_aFrameArray[i]->Print();
    printf("\n");
  }

  printf("\n");
}
```