DISSERTATION

SPARSE MATRIX VARIETIES, DAUBECHIES SPACES, AND GOOD COMPRESSION REGIONS OF GRASSMANN MANIFOLDS

Submitted by

Brian Collery

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2024

Doctoral Committee:

Advisor: Chris Peterson
Co-Advisor: Clayton Shonkwiler

Renzo Cavalieri
Michael Kirby
Louis-Nöel Pouchet

ABSTRACT

SPARSE MATRIX VARIETIES, DAUBECHIES SPACES, AND GOOD COMPRESSION REGIONS OF GRASSMANN MANIFOLDS

The Grassmann manifold $\mathbf{Gr}(k, n)$ is a geometric object whose points parameterize k dimensional subspaces of $\mathbb{R}^n$. The flag manifold is a generalization in that its points parameterize flags of vector spaces in $\mathbb{R}^n$. This thesis concerns applications of the geometry of the Grassmann and flag manifolds, with an emphasis on image compression. As a motivating example, the discrete versions of Daubechies wavelets generate distinguished n-dimensional subspaces of $\mathbb{R}^{2n}$ that can be considered as distinguished points on $\mathbf{Gr}(n, 2n)$. We show that geodesic paths between "Daubechies points" parameterize families of "good" image compression matrices. Furthermore, we show that these paths lie on a distinguished Schubert cell in the Grassmannian. Inspired by the structure of Daubechies wavelets, we define and explore *sparse matrix varieties* as a generalization. Keeping in that theme, we are interested in understanding geometric considerations that constrain the "good" compression region of a Grassmann manifold.

ACKNOWLEDGEMENTS

Is aistear bliana é aon tráchtas, agus tá an iomarca múinteoirí ann chun aitheantas a thabhairt dóibh ar fad, ach táim buíoch as an teagasc agus an treoir a thug siad.

Eascraíonn an tráchtas seo agus an taighde atá ann ó na cruinnithe seachtainiúla a bhí agam le mo chomhairleoirí. Ba mhaith liom buíochas a ghabháil le Chris Peterson agus Clay Shonkwiler as a dtreoir agus iad ag iarraidh cineálacha cur chuige agus moltaí éagsúla a thriail maidir le bealaí torthúla le himscrúdú.

Táim buíoch de mo chlann—Mam, Daid, Aoife, Tommy, Eoghan, Lyra, agus Sadhbh—as a ngrá agus a dtacaíocht le linn tréimhsí deacra.

Ba mhaith liom buíochas a ghabháil leis na cairde atá déanta agam tríd an gclár seo, leithéidí Justin, Mats, Kelly, Amie, Seth, Anthony, Vlad, Jae, Michael, agus Colin. Is féidir le scoil grád a bheith deacair ach bhí sé taitneamhach le cairde.

A James Wilson, go raibh maith agat as an gcomhairle agus as na tuairimí malartacha, go háirithe ó thaobh ríomhaireachtúil de. Bhí siad seo thar a bheith úsáideach chun an tráchtas seo a dhéanamh.

A chara Pádraig, is mór agam an gáirí agus na moltaí chun rudaí a réiteach.

Do mo chairde ar ais sa bhaile: Fintan, Kieran, Lee, Mark, agus Eoin. Bhí sé iontach i gcónaí casadh leat go léir aon uair a d'éirigh mé abhaile, agus ba bhealach iontach iad glaonna Zoom chun an spiorad a choinneáil suas le linn Covid.

Fuair an obair seo tacaíocht i bpáirt le deontas ón National Science Foundation (DMS–2107700).

Any dissertation is a years-long journey, and there are too many teachers to acknowledge them all, but I am grateful for their instruction and guidance.

This dissertation and the research within it result from the weekly meetings I had with my advisors. I want to thank Chris Peterson and Clay Shonkwiler for their guidance in trying different approaches and recommendations for fruitful paths to investigate.

I am grateful to my family—Mom, Dad, Aoife, Tommy, Eoghan, Lyra, and Sadhbh—for their love and support during difficult times.

I want to thank the friends I have made through this program, such as Justin, Mats, Kelly, Amie, Seth, Anthony, Vlad, Jae, Michael, and Colin. Grad school can be difficult but it was enjoyable with friends.

To James Wilson, thank you for the advice and alternative perspectives, especially from a computational viewpoint. These were incredibly useful in making this dissertation.

Thank you to the people I collaborated with, Seth and Anthony, after Covid. You were extraordinarily gracious and helpful when restarting everything.

To my roommates, Cameron, Philip, and Emily, whom I have lived with and stayed in contact with over the years. Living with all of you was definitely a highlight of my time here.

To my friend Patrick, I appreciate the laughs and suggestions for sorting things out.

To my friends back home: Fintan, Kieran, Lee, Mark, and Eoin. It was great always to meeting you all whenever I got back home, and Zoom calls were a great way to keep the spirits up during Covid.

This work was partially supported by a grant from the National Science Foundation (DMS–2107700).

DEDICATION

I would like to dedicate to my family and all the friends I've made along the way.

TABLE OF CONTENTS

Chapter 1

Introduction

It has become apparent that geometric structure underlies many large data sets. This has led to the development of tools that leverage geometry to represent, compress, and extract information from such data sets. A broad area of data analysis and machine learning of particular relevance to this thesis concerns image processing, image compression, and image representation. We will exploit aspects of the geometric structure underlying collections of digital images to gain insights into good linear compressions for such data. A geometric tool that has surfaced in many applications for image compression and image processing is the Grassmannian [1]. The Grassmannian is a manifold whose points parameterize vector subspaces of a fixed dimension of a fixed ambient vector space. Hermann Grassmann described this geometric object in 1844, but its first applications by others were slow to appear as its importance was not well understood at the time. In recent years, people have built a collection of practical algorithms for processing data on Grassmannians. These include the ability to represent a cluster of Grassmann points with a representative "mean" or "median," and the ability to move along geodesic paths between points [2]. Basic linear algebraic constructions such as Gram-Schmidt orthogonalization [3], the singular value decomposition, and exponential and logarithmic maps for matrices allow for relatively efficient and practical computations on the Grassmannian and other related manifolds. Another collection of tools that have proven to be remarkably effective for data compression and data representation, particularly in the arena of image and signal compression, are wavelets. The Haar wavelet [4] is perhaps the most fundamental of this class, and its use stretches back over 100 years (though the term "wavelet" didn't appear until much later). In general, a wavelet can intuitively be thought of as a short-term wave or oscillation. Its mathematical structure has proven to be helpful for the purpose of extracting and compressing information within a signal. A broad family of wavelets that have proven to be particularly useful in applications are the Daubechies wavelets. The Daubechies wavelets grew out of the foundational work of Ingrid Daubechies, and their remarkable properties have

led to an explosion of applications. A highly recommended and influential source on this topic is the monograph *Ten lectures on wavelets* [5]. An interesting interaction happens when one hybridizes these two sets of tools (wavelets and Grassmannians) and applies them to problems in data representation, analysis, and compression. For instance, discrete versions of wavelets determine special points on a Grassmannian. In particular, the Daubechies wavelets $D_2, D_4, D_6, \ldots$ determine points that lie on a common distinguished subregion of the Grassmannian $\mathbf{Gr}(n, 2n)$. This subregion can be described concisely as the *Schubert Variety* of all n-dimensional subspaces in a $2n$-dimensional space containing the "*all ones*" vector. This kind of Schubert variety is known to be geodesically closed. Consequently, if two points lie on this Schubert variety, then a geodesic path passing through these two points also lies on the Schubert variety. In particular, the geodesic paths between Daubechies wavelet points on the Grassmann manifold stay within this Schubert variety. Computationally however, points along a geodesic path between a pair of Daubechies points appear to have an additional property in that all such points seem to correspond to subspaces that behave well with respect to compressing image data. We will show how to visualize paths on the Grassmann manifold through a corresponding projection action on an image. This allows one to gain insight into the region on a Grassmann manifold corresponding to subspaces that have good compression capability for image data.

A point on the Grassmann manifold $\mathbf{Gr}(k, n)$ corresponds to a k-dimensional subspace, V, of $\mathbb{R}^n$. If we have any $n \times k$ matrix A whose column space equals V, we can build a corresponding $n \times n$ projection matrix P_V that depends on V rather than the matrix A representing V. This matrix has the property that for any vector x in $\mathbb{R}^n$, the vector $P_V x$ will be the closest vector in V to the vector x. If we have two points on $\mathbf{Gr}(k, n)$ corresponding to two distinct k-dimensional subspaces, V and W of $\mathbb{R}^n$, then a geodesic path on $\mathbf{Gr}(k, n)$ between these two points corresponds to a "shortest" parameterized family of k-dimensional subspaces beginning at V and ending at W. This leads to a parameterized family of projection matrices that begin at P_V and end at P_W. Denote this family by $P_{V\to W}(t)$ with $P_{V\to W}(0) = P_V$ and $P_{V\to W}(1) = P_W$. We will be considering the effect of such families on a matrix M. In particular, we will consider the parameterized family

of matrices $PM(t)$ defined by $PM(t) = P_{V_1 \to W_1}(t) M P_{V_2 \to W_2}(t)$. When M corresponds to a black and white image, one can consider $PM(t)$ as a "movie". We will view this movie as a sequence of sampled images for various values of t. Keeping in mind that Daubechies wavelets determine points on a Grassmann manifold $\mathbf{Gr}(n, 2n)$, we will look at movies arising from a geodesic path between pairs of Daubechies points on $\mathbf{Gr}(n, 2n)$. A perhaps surprising feature is that such movies, built from a black-and-white image, have good features all along the path for values of t between 0 and 1. What if we continue on the path of the Grassmannian beyond values of t in the interval $[0, 1]$? Experimentally, I found that the points on the Grassmannian between two Daubechies wavelets lead to projection matrices that compress the image in a good way but that if you continue along the path beyond the interval $[0, 1]$, the projection matrices start to lose their good properties in that they no longer compress the image in a good way.

Extending these investigations, consider a "Daubechies triangle" created by taking three different Daubechies wavelet points and using them as the vertices of the triangle. Then, the edges of the triangle correspond to paths between the pairs, and paths between points on the edges correspond to points in the triangle's interior. If you explore within this region, will the projection matrices still retain the structure of the image? From the algorithms produced, the answer is yes. This leads to the broader question of whether there is some particular geometric region within the Grassmannian whose points include the Daubechies wavelet points and whose points all have good compression properties and have the additional property of being "convex" (geodesic paths between points in the region also lie in the region)? Among other results, in this thesis, we determine that Daubechies wavelet points live within a totally geodesic Schubert variety that we can describe explicitly and that the "convex hull" of the Daubechies points appear to lie in the geometric region we seek. Due to these results, there is the natural question of whether there could be additional geometric constraints that we can identify or additional Schubert varieties of interest that further identify the region of the Grassmannian corresponding to good projections.

Inspired by a specific sparsity feature associated with discrete Daubechies wavelets, we define a family of special subvarieties of $\mathbf{Gr}(n, 2n)$ that we refer to as sparse matrix varieties. These are

algebraic varieties that contain the Daubechies wavelets and whose points have some additional desirable properties partially mirroring the properties of discrete Daubechies wavelets. We develop a gradient descent algorithm that can be used to find points on the variety that minimize energy loss concerning projection perturbations of an image. We then generalize this family to sparse matrix varieties lying within $\mathbf{Gr}(n, kn)$ for $k = 3, 4, 5, \ldots$.

Chapter 2

Grassmann Manifolds

Grassmann manifolds (or Grassmannians) are a class of mathematical objects that have had a broad impact in both pure [6] and applied mathematics [7], [8]. Their use has been amplified due to various algorithms that have made practical computations on Grassmann manifolds feasible [9], [10]. We aim to extend old algorithms, develop new algorithms, and add new perspectives for these manifolds. Through these developments, we hope to extend their utility and applicability. One way to gain insight is to visualize paths on a Grassmann manifold through an associated action, of points along the path, on a digital image. By visualizing the effect of points on a Grassmannian, one gains intuition about the region of the Grassmannian that provides good image compression capability. This chapter will build up the fundamental tools that will serve as the building blocks for visualizing a Grassmann manifold path. We start with the definition of a Grassmann manifold and show that each point on the manifold can be associated with a unique projection matrix. Then, we explore how to construct geodesic paths on the Grassmannian to examine the effect of a parameterized family of projection matrices on an image. Next, we interpret a step in compressing a discretized signal, via a Daubechies wavelet, as a single point on a Grassmann manifold. Then, we explore paths along a Grassmannian that contain such a Daubechies wavelet point. This facilitates our understanding of the region of a Grassmannian we can characterize as having visually "good" projection properties with respect to image data.

2.1 Definition of a Grassmannian

To begin with, we will introduce some of the mathematical notation that will be used in the following chapters. In this thesis, we will focus on real vector spaces. Specifically, we will let $\mathbf{Gr}(k, n)$ denote the real Grassmannian whose points parameterize k dimensional real vector subspaces of $\mathbb{R}^n$. So, how do we represent a point on $\mathbf{Gr}(k, n)$? We will describe a k-dimensional vector space V through an orthonormal basis for the space, $v_1, \ldots, v_k$. We will refer to the vector

space V formed by these k vectors as a k-plane. Of course, many distinct orthonormal bases exist for the same space, and we must account for this non-unique representation. Throughout this thesis, we will use V to refer to either the vector space or the matrix, whose columns determine an orthonormal basis, representing V as its column space. Edelman et al. [11] introduced practical ways to use the matrix representation of V to describe geodesic paths on the Grassmannian. We will frequently use this description for our purposes.

We want to represent a point in $\mathbf{Gr}(k, n)$. As mentioned above, one way to do so is by starting with a set of k orthonormal vectors that span the associated k-dimensional space. On a computer, we can store this point as a matrix and carry out various operations on the matrix. The elements in the set of all $n \times k$ matrices with orthonormal columns correspond to points on an important geometric object known as a Stiefel manifold. More precisely, the Stiefel manifold, $\mathbf{St}(k, n)$, can be described as

$$\mathbf{St}(k, n) := \{A \in \mathbb{R}^{n \times k} \mid A^T A = I\}$$

where I denotes the $k \times k$ identity matrix. Another way of describing the Stiefel manifold is through the cosets of the group of $n \times n$ orthogonal matrices, $\mathbf{O}(n)$, modulo a subgroup isomorphic to $\mathbf{O}(n-k)$

$$\mathbf{St}(k, n) := \mathbf{O}(n)/\mathbf{O}(n-k).$$

The lower $\mathbf{O}(n-k)$ is realized as the set of block diagonal $n \times n$ matrices whose first block is a $k \times k$ identity matrix and whose second block is an $(n-k) \times (n-k)$ orthogonal matrix. This representation amounts to considering points on $\mathbf{St}(k, n)$ as the first k columns of an $n \times n$ orthogonal matrix. In other words, we identify two elements of $\mathbf{O}(n)$ if they have the same first k columns (i.e. we ignore the last $n-k$ columns).

There is a natural map from $\mathbf{St}(k, n)$ to $\mathbf{Gr}(k, n)$ by mapping an ordered orthonormal basis for a k-plane to the span of the ordered orthonormal basis. The fibers of this map are isomorphic to a

copy of $\mathbf{O}(k)$ that accounts for the collection of all orthonormal bases for the same k-plane. This leads us to describe $\mathbf{Gr}(k,n)$ as

$$\mathbf{Gr}(k,n) = \frac{\mathbf{O}(n)}{\mathbf{O}(k) \times \mathbf{O}(n-k)}$$

where the lower $\mathbf{O}(k) \times \mathbf{O}(n-k)$ is realized as the set of block diagonal $n \times n$ matrices whose first block is a $k \times k$ orthogonal matrix and whose second block is an $(n-k) \times (n-k)$ orthogonal matrix.

We can use the following lemma to help us understand a definition from Edelman et al [11].

Lemma 2.1. *Let $V \in \mathbb{R}^{n \times k}$ satisfy $V^T V = I$. Consider V as an element in $\mathbf{St}(k,n)$. If $M \in \mathbf{O}(k)$ then $VM \in \mathbf{St}(k,n)$.*

Proof. Note that the dimensions of V are $n \times k$, and the dimensions of M are $k \times k$, so the product VM has the correct dimensions to be an element of $\mathbf{St}(k,n)$. Next, we must check that $(VM)^T(VM) = I$.

$$\begin{aligned}(VM)^T(VM) &= M^T V^T V M \\ &= M^T I M \\ &= M^T M \\ &= I.\end{aligned}$$

□

This now lets us discuss a function with a *homogeneity* property described in Edelman et al.

Definition 2.1. *Let F be a function that takes $\mathbf{St}(k,n)$ as its set of inputs. We say that F is homogeneous if for $V \in \mathbf{St}(k,n)$ and $M \in \mathbf{O}(k)$, $F(VM) = F(V)$.*

In other words, the function is defined on orthogonal $n \times k$ matrices but is invariant to the right action of $k \times k$ orthogonal matrices. This allows us to consider a homogeneous function applied to points on a Stiefel manifold as a function applied to points on a Grassmann manifold.

2.2 Visualizing paths along a Grassmannian

2.2.1 Projection

This section will create tools to visualize paths along a Grassmannian on a computer. For any $V \in \mathbf{Gr}(k, n)$, we can store an orthonormal basis for V as a matrix that we can load into the computer. At this point, the stored matrix only corresponds to an element in $\mathbf{St}(k, n)$. To make this work, we will need a homogeneous function on $\mathbf{St}(k, n)$ that allows for visual interpretation. A natural candidate for this function is the map that takes the vector space V to its corresponding projection matrix P_V. We can then use this projection matrix to act on digital images represented as arrays. To ensure its viability, we must first check that it is a homogeneous function.

Lemma 2.2. *Let $V \in \mathbf{St}(k, n)$ and Proj:$\mathbf{St}(k, n) \to \mathbb{R}^{n \times n}$, where Proj(V) = VV^T. Proj is a homogeneous function.*

Proof. Let $M \in \mathbf{O}(k)$. Then

$$\begin{aligned} \textit{Proj}(VM) &= (VM)(VM)^T \\ &= VMM^TV^T \\ &= VIV^T \\ &= VV^T \\ &= \textit{Proj}(V). \end{aligned}$$

□

We aim to represent each point of a Grassmannian with an action, so if we have two points with the same action, they will be in the same equivalence class. The following Lemma proves this to be true.

Lemma 2.3. *Let $V, W \in \mathbf{St}(k,n)$. Then Proj(V) = Proj(W) iff $V^TW \in \mathbf{O}(k)$*

Proof.

$$\begin{aligned} &\text{If} && \textit{Proj}(V) = \textit{Proj}(W) \\ & && VV^T = WW^T \\ &\text{then} && W^TVV^TW = W^TWW^TW \\ &\text{so} && (V^TW)^T(V^TW) = I. \end{aligned}$$

Lemma 2.2 is the reverse direction. □

A meaningful interpretation of Lemma 2.3 is that there is a unique projection matrix for every point on a Grassmannian. Thus, each point on a Grassmann manifold $\mathbf{Gr}(k,n)$ is associated to a unique k-dimensional subspace V of $\mathbb{R}^n$ which in turn is associated with a unique projection matrix P_V [12]:

$$\mathbf{Gr}(k,n) = \{P \in \mathbb{R}^{n\times n} \mid P^2 = P, \quad \text{rank}(P) = k\}.$$

This interpretation will allow us to visualize the action, of each point along a geoedesic path, on an image. We now want to describe how to build geodesic paths on a Grassmannian, in its model as a quotient space built out of $\mathbf{O}(n)$, then use this to build paths on a Grassmannian in its model as a collection of rank k projection matrices.

We want to build paths on a Grassmannian by creating a path in $\mathbf{O}(n)$. The fact that $\mathbf{O}(n)$ is not a connected manifold slightly complicates our approach. With this in mind, we modify our definition of $\mathbf{Gr}(k,n)$ so that it is expressed in terms of a quotient space of the connected manifold

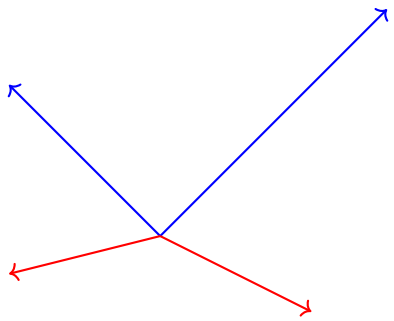
$\mathbf{SO}(n)$ [13] :

$$\mathbf{Gr}(k,n) \cong \frac{\mathbf{SO}(n)}{\mathbf{S}(\mathbf{O}(k) \times \mathbf{O}(n-k))}.$$

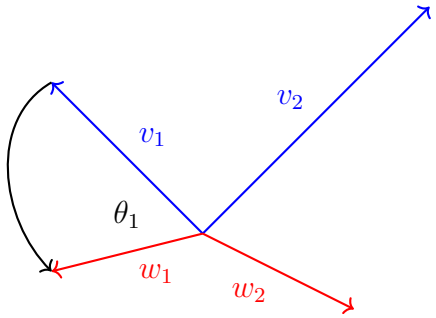
Note that in the above definition, $\mathbf{S}(\mathbf{O}(k) \times \mathbf{O}(n-k))$ denotes the set of block diagonal $n \times n$ matrices in $\mathbf{SO}(n)$ whose first block is a $k \times k$ orthogonal matrix and whose second block is an $(n-k) \times (n-k)$ orthogonal matrix. This representation of the Grassmannian has several advantages; for instance, as $\mathbf{SO}(n)$ is a matrix group and is connected, it has a closer connection to its Lie algebra. Furthermore, describing paths along a Grassmannian is already known. We can use principal angles between subspaces to create geodesic path within this space.

2.2.2 Principal Angles

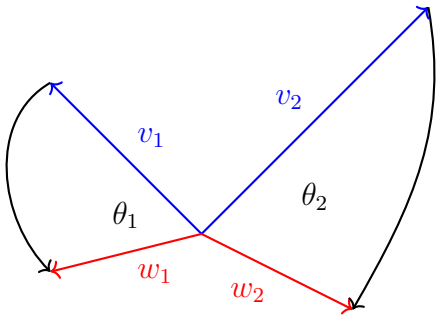
Principal angles between two subspaces characterize the smallest angles between vectors in the two subspaces. An important application of principal angles (and their associated principal vectors) is that they allow a description of the shortest paths between points on a Grassmannian. Each point on a Grassmannian corresponds to a vector space. The picture below represents two vector spaces through colored orthonormal basis vectors for each space. We begin with an example of two points on $\mathbf{Gr}(2,n)$ representing two distinct subspaces of $\mathbb{R}^n$. The blue vectors below represent the space V and the red vectors represent the space W.



We want to describe the shortest path between these two points. This effectively corresponds to finding an orthonormal basis for V that can be moved as little as possible to arrive at an orthonormal basis for W. To do this, we pick vectors v_1 in V and w_1 in W to achieve the smallest possible angle among all pairs of vectors, with one in V and one in W.



The vectors v_2 in V and w_2 in W are then chosen to minimize the angle between them, with v_2 constrained to be orthogonal to v_1 and w_2 constrained to being orthogonal to w_1. We then rotate v_1 towards w_1 while simultaneously rotating v_2 towards w_2.



We can parameterize this motion with a parameter t lying in the interval $[0, 1]$ such that when $t = 0$, the vectors are in their initial state and rotation proceeds at constant speed such that when $t = 1$, v_1 lands on w_1. At the same time, v_2 rotates towards w_2 such that at $t = 1$, v_2 lands on w_2. According to [14], Jordan in 1875 had already begun to progress on these ideas and there are many introductory articles on related topics in modern times [15].

Suppose we have two k-planes V and W in $\mathbb{R}^n$ given as the column spaces of orthogonal $n \times k$ matrices (denoted A and B). Thus, V and W are represented by elements A, B in the Stiefel manifold $\mathbf{St}(k, n)$. Recall that a homogeneous function F on $\mathbf{St}(k, n)$ is a function that satisfies, for any $M \in \mathbf{O}(n)$ and $A \in \mathbf{St}(k, n)$, the condition $F(AM) = F(A)$. We will now describe a way to characterize orthogonally invariant functions on pairs of k-planes in $\mathbf{R}^n$. Given two k-planes V and W in $\mathbb{R}^n$, we wish to describe an ordered list of k principal angles between V and W. We again let V and W be represented by a pair of orthogonal $n \times k$ matrices (denoted A and B). A function F on pairs of elements of $\mathbf{St}(k, n)$ is said to be orthogonally invariant on $\mathbf{St}(k, n)$ if

$F(A,B) = F(NA,NB)$ for any element $N \in \mathbf{O}(n)$ and F is said to be orthogonally invariant on $\mathbf{Gr}(k,n)$ if $F(A,B) = F(NAP,NAQ)$ for any $N \in \mathbf{O}(n)$ and any $P,Q \in \mathbf{O}(k)$. It is a fundamental theorem that any orthogonally invariant function on pairs of k-planes in $\mathbf{R}^n$ can be expressed as a function of the principal angles between A and B. With that in mind, we now describe principal angles between subspaces:

Definition 2.2. *Let A and B be $n \times k$ matrices whose respective columns, $a_1, a_2, \ldots a_k$ and $b_1, b_2, \ldots, b_k$, are orthonormal bases for k-dimensional subspaces V and W of $\mathbb{R}^n$. The first Principal angle, $\theta_1 = \theta_1(V,W)$, between V and W, is defined by*

$$\cos(\theta_1) = \max\{< v, w > \mid v \in V, w \in W, ||v|| = ||w|| = 1\}.$$

The first Principal vectors between V and W are vectors

$$x_1 \in V, y_1 \in W \text{ with } ||\mathrm{x_1}|| = ||\mathrm{y_1}|| = 1 \text{ and } < \mathrm{x_1}, \mathrm{y_1} >= \cos(\theta_1).$$

The i^{th} Principal angle is defined inductively as

$$\cos(\theta_i) = \max\{< v, w > \mid v \in V, w \in W, ||v|| = ||w|| = 1, < v, x_j >=< w, y_j >= 0 \text{ for } \mathrm{j} < \mathrm{i}\}.$$

The i^{th} Principal vectors between V and W are vectors

$$x_i \in V, y_i \in W \text{ with } ||\mathrm{x_i}|| = ||\mathrm{y_i}|| = 1, \ < \mathrm{x_i}, \mathrm{y_i} >= \cos(\theta_\mathrm{i}), < \mathrm{x_i}, \mathrm{x_j} >=< \mathrm{y_i}, \mathrm{y_j} >= 0 \text{ for } \mathrm{j} < \mathrm{i}.$$

Note that the first pair of principal vectors x_1, y_1, is a pair of vectors where x_1 is in the k-plane V and y_1 is in the k-plane W and they achieve the smallest possible angle between vectors in the two k-planes. We will discuss extensions of these ideas to flag manifolds in the conclusion.

Let X, Y denote the $n \times k$ matrices with columns $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$. There exist orthogonal matrices $U, V \in \mathbf{O}(k)$ such that:

$$X = \begin{bmatrix} | & | & \ldots & | \\ | & | & \ldots & | \\ \mathbf{x_1} & \mathbf{x_2} & \ldots & \mathbf{x_k} \\ | & | & \ldots & | \\ | & | & \ldots & | \end{bmatrix} = \begin{bmatrix} | & | & \ldots & | \\ | & | & \ldots & | \\ \mathbf{a_1} & \mathbf{a_2} & \ldots & \mathbf{a_k} \\ | & | & \ldots & | \\ | & | & \ldots & | \end{bmatrix} U = AU$$

$$Y = \begin{bmatrix} | & | & \ldots & | \\ | & | & \ldots & | \\ \mathbf{y_1} & \mathbf{y_2} & \ldots & \mathbf{y_k} \\ | & | & \ldots & | \\ | & | & \ldots & | \end{bmatrix} = \begin{bmatrix} | & | & \ldots & | \\ | & | & \ldots & | \\ \mathbf{b_1} & \mathbf{b_2} & \ldots & \mathbf{b_k} \\ | & | & \ldots & | \\ | & | & \ldots & | \end{bmatrix} V = BV.$$

Note that by construction, we have

$$U^T A^T B V = (AU)^T (BV) = X^T Y = \text{diag}(\cos(\theta_i)).$$

Therefore

$$A^T B = U \text{diag}(\cos(\theta_i)) V^T.$$

This expression shows that we can use the singular value decomposition to determine U and V. The computation of the singular value decomposition of a matrix is a standard linear algebra technique often used in data analysis.

2.2.3 Computing points along a path on a Grassmannian

Our goal is to create a path along a Grassmannian

$$\mathbf{Gr}(k,n) = \frac{\mathbf{SO}(n)}{\mathbf{S}(\mathbf{O}(n-k)\times\mathbf{O}(k))}.$$

For the algorithms we present, we will complete an orthogonal basis for a vector space V in $\mathbb{R}^n$ to an orthogonal basis for all of $\mathbb{R}^n$ in such a manner as to make an element of the special orthogonal group, $\mathbf{SO}(n)$. We will now focus on Grassmannians of the form $\mathbf{Gr}(n,2n)$. This means that V and its orthogonal complement will be of the same dimension n. For a space V, we denote its orthogonal complement as $V^\perp$. To get a representative of $\mathbf{SO}(2n)$ from our V, we concatenate orthonormal bases for V and $V^\perp$ to get an orthnormal basis for $\mathbf{R}^{2n}$, which we compile in a $2n\times 2n$ matrix $[V,V^\perp]$. If necessary, we can multiply the last column vector by -1 to ensure the matrix has a determinant equal to 1.

From the paper by Ji [16], if we have two points on our Grassmannian in the form $[V,V^\perp]$, $[W,W^\perp]$ where these are matrix representations made by principal vectors, $\{a_i\}$ and $\{b_i\}$, between the spaces V and W with complements $\{\tilde{a}_i\}$ and $\{\tilde{b}_i\}$ made by principal vectors between the spaces $V^\perp$ and $W^\perp$ then the product $[V,V^\perp]^T[W,W^\perp]$ looks as follows:

$$\begin{bmatrix} - & - & a_1 & - & - \\ - & - & a_2 & - & - \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ - & - & a_n & - & - \\ - & - & \tilde{a}_1 & - & - \\ - & - & \tilde{a}_2 & - & - \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ - & - & \tilde{a}_n & - & - \end{bmatrix} \begin{bmatrix} | & | & \dots & | & | & | & \dots & | \\ | & | & \dots & | & | & | & \dots & | \\ b_1 & b_2 & \dots & b_n & \tilde{b}_1 & \tilde{b}_2 & \dots & \tilde{b}_n \\ | & | & \dots & | & | & | & \dots & | \\ | & | & \dots & | & | & | & \dots & | \end{bmatrix}.$$

$$
\begin{array}{c}
 \\
a_1 \\
a_2 \\
\vdots \\
a_n \\
- \\
\tilde{a}_1 \\
\tilde{a}_2 \\
\vdots \\
\tilde{a}_n
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc}
b_1 & b_2 & \dots & b_n & | & \tilde{b}_1 & \tilde{b}_2 & \dots & \tilde{b}_n
\end{array} \\
\left[\begin{array}{cccccccccc}
\cos(\theta_1) & 0 & \dots & 0 & & \sin(\theta_1) & 0 & \dots & 0 \\
0 & \cos(\theta_2) & \dots & 0 & & 0 & \sin(\theta_2) & \dots & 0 \\
\vdots & & \ddots & & & & & \ddots & \\
0 & & & \cos(\theta_n) & & & & & \sin(\theta_n) \\
- & - & - & - & - & - & - & - & - \\
\sin(\theta_1) & 0 & \dots & 0 & & \cos(\theta_1) & 0 & \dots & 0 \\
0 & \sin(\theta_2) & \dots & 0 & & 0 & \cos(\theta_2) & \dots & 0 \\
\vdots & & \ddots & & & & & \ddots & \\
0 & & & \sin(\theta_n) & & 0 & & & \cos(\theta_n)
\end{array}\right]
\end{array}
$$

We will use this matrix to create geodesic paths between the two points on $\mathbf{Gr}(n, 2n)$. I let the basis vectors from the first matrix index the rows and the basis vectors of the second matrix index the columns of the matrix. Each entry in the matrix corresponds to an inner product between column vectors from each of the matrices. We will use the interpretation in Chapter 3 to create subvarieties of a Grassmannian.

From this we can parameterize a geodesic path from $[V, V^{\perp}]$ to $[W, W^{\perp}]$ as:

$$P(t) = [V, V^{\perp}] \begin{bmatrix} \cos(\theta_1 t) & 0 & \dots & 0 & \sin(\theta_1 t) & 0 & \dots & 0 \\ 0 & \cos(\theta_2 t) & \dots & 0 & 0 & \sin(\theta_2 t) & \dots & 0 \\ \vdots & & \ddots & & & & \ddots & \\ 0 & & & \cos(\theta_n t) & & & & \sin(\theta_n t) \\ - & - & - & - & - & - & - & - \\ \sin(\theta_1 t) & 0 & \dots & 0 & \cos(\theta_1 t) & 0 & \dots & 0 \\ 0 & \sin(\theta_2 t) & \dots & 0 & 0 & \cos(\theta_2 t) & \dots & 0 \\ \vdots & & \ddots & & & & \ddots & \\ 0 & & & \sin(\theta_n t) & 0 & & & \cos(\theta_n t) \end{bmatrix}.$$

Note that $P(0) = [V, V^{\perp}]$ and $P(1) = [V, V^{\perp}] \cdot [V, V^{\perp}]^T \cdot [W, W^{\perp}] = [W, W^{\perp}]$. Also note that, for each t, $P(t)$ is a product of two matrices in the special orthogonal group thus $P(t)$ lies in $\mathbf{SO}(2n)$ for each t. When using the complement space to move along the Grassmannian, we only consider the first half of the basis vectors to create the projection matrix. We do not wish to include the orthogonal complement space in our projection matrix. If we consider the basis representation at time t as $V(t)$, we can describe $P(t) = [V(t), V(t)^{\perp}]$ and to create our projection matrices we will only need $V(t)$.

2.2.4 Applying Projection matrices to images

Now that we have a method for creating a path on a Grassmannian, let's explore a way to visualize the path. We represent points on a Grassmannian via elements in the group $\mathbf{SO}(2n)$. Considering the induced action on matrices, we can take the matrix that represents an image as the object to be acted upon. As each point in the Grassmannian corresponds to a vector space, we can create a projection matrix for each point and observe its effect on an image, thus visualizing the path on the Grassmannian.

To demonstrate this approach, we apply it to images in the MNIST dataset [17]. This dataset was one of the early standards for machine learning. It consists of black-and-white images of handwritten digits. We will stick with MNIST images as our running example but it is elementary to extend the technique to larger images. As black and white images, each MNIST image is represented by a two dimensional array, i.e. a matrix. By applying a projection matrix and presenting the effect as a new image, we can see the action of each point on the Grassmannian.

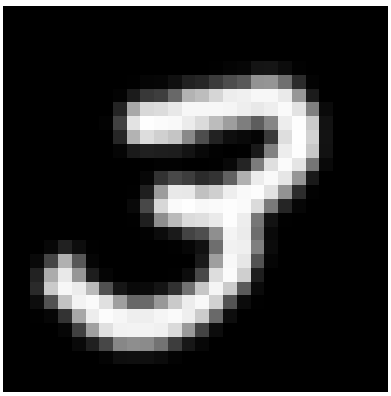


Figure 2.1: MNIST Image

The dimension of each MNIST image is 28 by 28. To examine this image properly, I used a Kronecker product to expand the image to highlight the changes that occur. The idea is to replace every pixel with an $m \times m$ block of pixels of the same color to make the image bigger. Many books in matrix analysis talk about this modification (e.g. see Horn and Johnson Chapter 4 [18]).

Now that we have the basics in place, we need to put everything together to create a method of visualizing the path. The strategy is to discretize the path between the two points in the Grassmannian. More precisely, we will discretize the path from 0 to 1 into segments and create a projection from an endpoint of each segment. If we then stack the images on top of one another, it will form a three dimensional array that we can visualize as a "film" showcasing how the action of the points, on the image, changes as we progress along the path.

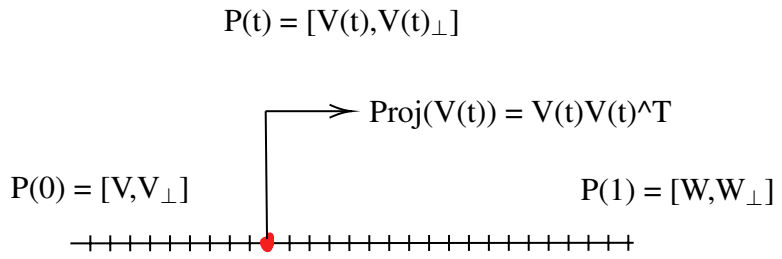


Figure 2.2: Discretize path between two points on a Grassmannian

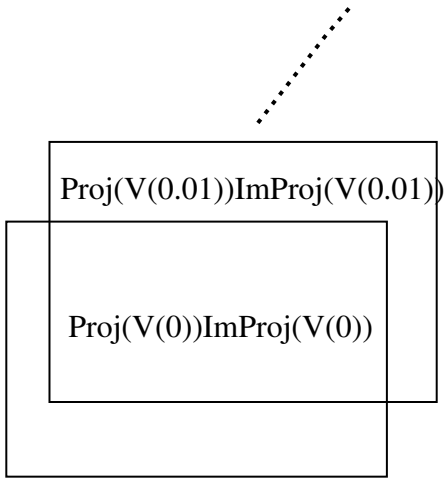


Figure 2.3: Creating a film of the action of the path

2.3 Method of choosing points on a Grassmannian

To classify a region of "good" compression within a Grassmannian, a starting point is to ask if there are any restrictions. Every point on a Grassmannian could, in theory, lead to a compression where the image is somewhat, visually retained. To test this hypothesis, I decided to see what would happen if we chose a point randomly and created an image from the projection matrices. The image is represented below.

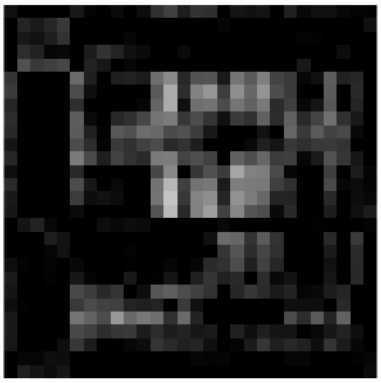
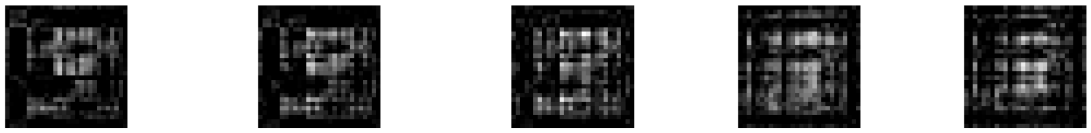


Figure 2.4: Using a random point to create a projection matrix

As you can see, the original image is severely altered. What if we follow along a geodesic path between two random points on the Grassmannian? Would we expect to run into a good projection space along the path? Experimentally, as we move on the Grassmannian from a random starting point to a random endpoint, the image's projection appears highly altered along the entire path. This is shown in figure 2.5.

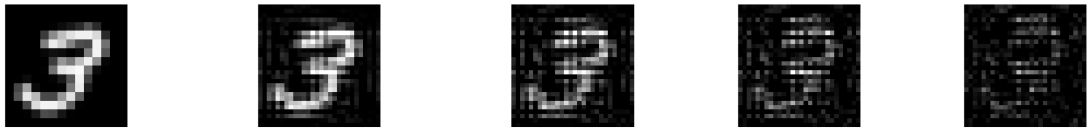


(a) Im(0) **(b)** Im(.25) **(c)** Im(0.5) **(d)** Im(0.75) **(e)** Im(1)

Figure 2.5: Moving between two random points

One of the motivating problems addressed in this thesis is to find regions of the Grassmannian that correspond to good projections of digital images (and correspondingly to good compression). Using two randomly chosen endpoints to create a path has led to an unsatisfactory compressed image at every point. To alter our strategy, we decided to change one of the endpoints to a point

known to compress images well. A commonly used, well-regarded method of compressing an image uses Daubechies wavelets [19]. As these are already accessible in Matlab, we pick our starting point to be a point built from a Daubechies wavelet and our endpoint to be some random point on the Grassmannian, as shown in figure 2.6.



(a) Im(0) **(b)** Im(0.25) **(c)** Im(0.5) **(d)** Im(0.75) **(e)** Im(1)

Figure 2.6: Moving from Daubechies 2 to a random point

This did not lead to a path where every point gave a recognizable three, but there is something to notice; close to my starting point, the point built from a Daubechies wavelet, the image stays recognizable, but further away, it becomes unrecognizable. To expand upon this observation, we built a point corresponding to the wavelet D_2 for the start of the path and a point corresponding to the Daubechies wavelet known as D_{10} to be the ending point. The projected images along this path are shown in figure 2.7.

(a) Im(0) **(b)** Im(0.25) **(c)** Im(0.5) **(d)** Im(0.75) **(e)** Im(1)

Figure 2.7: Moving from Daubechies 2 to Daubechies 10

Notably, not only do the endpoints of our path preserve the image, but the intermediate points also maintain their integrity. The only requirement for this path, apart from being the geodesic between two points on a Grassmannian, is that the start and end points are Daubechies wavelets.

This path has proven visually effective, but its success only continues for a while. To verify this, we followed the same route until $t = 6$, at which point the image's deterioration is evident:

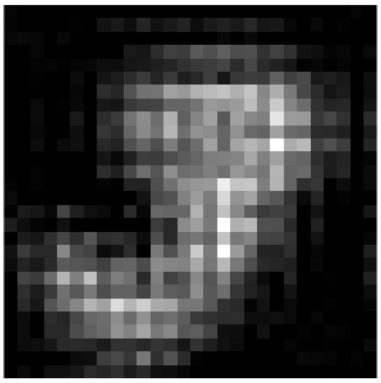


Figure 2.8: Im(6)

This strategy illustrates that the region of "good" compression on the Grassmannian may have an analog of a convexity property. To extend what has worked before, instead of using two Daubechies wavelets to create a path, we can use three to create a "triangular" region within our space. We make the vertices of our triangle points corresponding to different Daubechies wavelets and then we move within this triangle. To illustrate this property, I moved halfway between D_2 and D_{10} and then moved towards D_6.

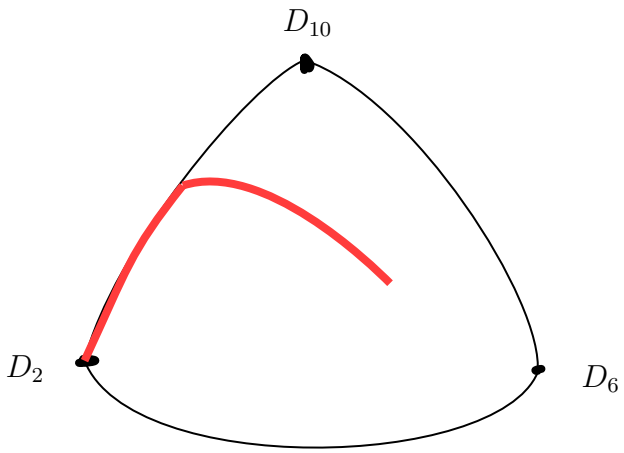


Figure 2.9: Path along triangle created from Daubechies wavelets

(a) Im(0) **(b)** Im(0.25) **(c)** Im(0.5) **(d)** Im(0.75) **(e)** Im(1)

Figure 2.10: Moving within the "Triangle"

This led us to conjecture that Daubechies wavelets have some shared property applicable for compressing images well and that this property is preserved along geodesic paths between the wavelets. We will now discuss how to construct Grassmann points corresponding to Daubechies wavelets in Matlab.

An essential part of this process was creating an element in $\mathbf{Gr}(n, 2n)$ from a Daubechies wavelet in Matlab. One restriction on Daubechies wavelets is that the ambient dimension to which it is applied must be even. Fortunately, Matlab has a command to make a length D_{2k} vector corresponding to the discrete Daubechies wavelet D_{2k}. To extend this to a vector, v, of length $2n$, we make the first $2k$ components of the vector the components from D_{2k} and the rest of the components of the vector zero. From v, we create n orthogonal vectors by shifting the components

of v down two places, then two more, etc. The command to shift components of a vector in Matlab is circshift. To create our point on the Grassmannian, we fill in a $2n$ by n matrix where each column is v with its components successively moved down two places as we move across. This process is given in Algorithm 1.

Algorithm 1 Making a subspace based off a Daubechies wavelet

Input: **k**$\in \mathbb{N}$,n $\in \mathbb{N}$
Output:Mat $\in \mathbb{R}^{2n\times n}$

```
function DAUBOMAT(k,n)
    d = dbaux(k)
    Mat = zeros(2*n,n)
    v = zeros(2*n,1)
    v(1:2*k) = d^T
    for i = 1:n do
        Mat(:,i) = circshift(v,2*i-2)
```

Visually, it would look as follows:

$$\begin{bmatrix} v_1 & 0 & 0 & \dots & v_3 \\ v_2 & 0 & 0 & \dots & v_4 \\ v_3 & v_1 & 0 & \dots & v_5 \\ v_4 & v_2 & 0 & \dots & v_6 \\ \vdots & \vdots & & & \vdots \\ v_{2k-1} & v_{2k-3} & v_{2k-5} & \dots & 0 \\ v_{2k} & v_{2k-2} & v_{2k-4} & \dots & 0 \\ 0 & v_{2k-1} & v_{2k-3} & \dots & 0 \\ \vdots & & & & \end{bmatrix}$$

To exemplify this process, I will build a couple of these matrices explicitly. For instance, if we look at the "wavelet" space $D_2 \in \mathbf{Gr}(4, 8)$, we would build the following matrix:

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

The following is an 8×4 matrix corresponding to a point, representing D_4, in $\mathbf{Gr}(4, 8)$:

$$\begin{bmatrix} \frac{1+\sqrt{3}}{4\sqrt{2}} & 0 & 0 & \frac{3-\sqrt{3}}{4\sqrt{2}} \\ \frac{3+\sqrt{3}}{4\sqrt{2}} & 0 & 0 & \frac{1-\sqrt{3}}{4\sqrt{2}} \\ \frac{3-\sqrt{3}}{4\sqrt{2}} & \frac{1+\sqrt{3}}{4\sqrt{2}} & 0 & 0 \\ \frac{1-\sqrt{3}}{4\sqrt{2}} & \frac{3+\sqrt{3}}{4\sqrt{2}} & 0 & 0 \\ 0 & \frac{3-\sqrt{3}}{4\sqrt{2}} & \frac{1+\sqrt{3}}{4\sqrt{2}} & 0 \\ 0 & \frac{1-\sqrt{3}}{4\sqrt{2}} & \frac{3+\sqrt{3}}{4\sqrt{2}} & 0 \\ 0 & 0 & \frac{3-\sqrt{3}}{4\sqrt{2}} & \frac{1+\sqrt{3}}{4\sqrt{2}} \\ 0 & 0 & \frac{1-\sqrt{3}}{4\sqrt{2}} & \frac{3+\sqrt{3}}{4\sqrt{2}} \end{bmatrix}.$$

In this thesis, we will refer to the space generated by the Daubechies wavelet rather than the wavelets themselves.

Definition 2.3. *Let $A \in \mathbf{Gr}(n, 2n)$. If a Daubechies wavelet generates A then we refer to A as a Daubechies space.*

In this chapter, we have established a method of traversing a Grassmannians and visualizing the action, of a point on the Grassmannian, on a digital image. We have an algorithm to make points on Grassmannians from Daubechies wavelets and can create “triangles” where each triangle vertex corresponds to a Daubechies wavelet. Remarkably, for each point within these “triangles”, the action of the point on the image retained much of the original image’s structure. This led to the conjecture that Daubechies wavelets have important properties within a Grassmannian that we could exploit. Chapter 3 is where we create special regions within a Grassmannian where all of the Daubechies wavelet spaces reside, and we show how to sample from this region. In Chapter 4, we examine the algebraic properties of Daubechies wavelets in the Stiefel manifold. This leads to the introduction of "sparse matrix varieties".

Chapter 3

The good compression region of the Grassmannian

My goal in this chapter is to describe a region of the Grassmannian that contains "good" projections. As each point on a Grassmannian corresponds to a vector space, one way to create a region in a Grassmannian is to describe features shared by the vector spaces in the region. One feature shared by multiple good projection points on Grassmannians is a tendency to contain or be close to containing a particular one-dimensional vector space. This feature can be captured by saying that the region lies near a specific Schubert variety. Schubert varieties in Grassmannians have played an important role in connecting Grassmannians with combinatorics, geometry, analysis, and many other disciplines. From this vantage point, it is not particularly surprising that they also connect to the good projection region of a Grassmannian. Papers such as Gillespie [20] and Coskun [21] are translating the tools used in more broad forms of Schubert varieties to the orthogonal form of Grassmannians utilized in this paper, but these are unnecessary for my purposes. In the following pages, I will use the notation $\mathbf{Sch}_W(k, n)$ for the particular type of Schubert variety we will be utilizing in this chapter. This chapter's structure describes Schubert varieties from a perspective that allows for the easy generation of random point samples. From here, we quickly examine the methodologies people use on images to determine whether the compression is adequate. We remove conditions on these techniques until later when we will have developed additional tools. At that point, we will better be able to create random projections that retain much of the "energy" in digital images by sampling from Schubert varieties defined using eigenvectors of special structured matrices.

3.1 Schubert Varieties

Definition 3.1. *Let W be a subspace of $\mathbb{R}^n$. We define the Schubert Variety $\mathbf{Sch}_W(k, n)$ by the formula:*

$$\mathbf{Sch}_W(k,n) := \{V \in \mathbf{Gr}(k,n) | W \subset V\}$$

This definition is helpful as we can use it to generate semi-randomized projection matrices, with the properties we want, by restricting to a Schubert variety with desired features. Let's look for properties from which to create Schubert varieties by looking at cases of suitable projection matrices associated to Daubechies wavelets.

3.1.1 A Schubert variety that contains Daubechies Spaces

A discrete Daubechies wavelet starts with a vector of length $2k$ satisfying a particular list of properties. For instance, one of the properties is that the even and odd coordinates have the same sum [22]. Using this vector of length $2k$, we can pad it with zeros to make a vector of length $2n$. If we cyclically permute this vector, shifting each coordinate by 2 with each cyclic shift, we get a collection of n vectors each of length $2n$. If we place these vectors as the columns of a matrix, we end up with a $2n \times n$ matrix of rank n. The column space of this matrix is referred to as a *Daubechies space*. As the sum of the even coordinates equals the sum of the odd coordinates in the discrete Daubechies vector, a simple observation is that the associated Daubechies space contains the all one's vector, which we will denote by $\mathbf{1}$. As each Daubechies space contains $\mathbf{1}$, each Daubechies space is contained in $\mathbf{Sch_1}(n, 2n)$. We can sample $\mathbf{Sch_1}(n, 2n)$ by picking points of the form $[\mathbf{1} \quad \mathbf{Gr}(n-1, 2n)^*]$, where $\mathbf{Gr}(n-1, 2n)^*$ means a collection of vectors, determining a point in $\mathbf{Gr}(n-1, 2n)$, whose span does not contain $\mathbf{1}$. We can then use Gram-Schmidt to create a matrix with the same column space but with orthonormal columns.

Algorithm 2 Creating an element of $\mathbf{Sch_1}(n, 2n)$

Input: n $\in \mathbb{N}$
Output:Mat $\in \mathbb{R}^{2n \times n}$

1: **function** SCHALLONES(n)
2: Mat = [ones(2n,1) nrmrnd(0,1,2n,2-1)]
3: Mat = GramSchmidt(Mat)

$\mathbf{Sch_1}(n, 2n)$ is a space from which we can draw samples that contains all Daubechies spaces. This space has a particular property, shared by all Schubert varieties of the form $\mathbf{Sch}_W(k, n)$, that we would like to now make explicit. More precisely, we can now show that geodesics on $\mathbf{Gr}(n, 2n)$, between points in $\mathbf{Sch_1}(n, 2n)$, stay in $\mathbf{Sch_1}(n, 2n)$ [23]. This can be restated as saying that $\mathbf{Sch_1}(n, 2n)$ is totally geodesic in $\mathbf{Gr}(n, 2n)$. The proof is informative and related to the strategies we used in Chapter 2. To show this, we first recall the following definition (which can be found on NLab) [24].

Definition 3.2. *A submanifold $S \subseteq M$ is totally geodesic if geodesics between points in S lie in S.*

Lemma 3.1. $\mathbf{Sch_1}(n, 2n)$ *is a totally geodesic submanifold of* $\mathbf{Gr}(n, 2n)$.

Proof. In section two, we refer back to the method of creating geodesics in the Grassmannian. We saw that if we take two points $V, W \in \mathbf{Sch_1}(n, 2n)$ then the geodesic between these points is created from the matrix:

$$
\begin{array}{c}
 \\
a_1 \\
a_2 \\
\vdots \\
a_n \\
- \\
\tilde{a}_1 \\
\tilde{a}_2 \\
\vdots \\
\tilde{a}_n
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc}
b_1 & b_2 & \dots & b_n & | & \tilde{b}_1 & \tilde{b}_2 & \dots & \tilde{b}_n
\end{array} \\
\left[\begin{array}{cccccccccc}
\cos(\theta_1) & 0 & \dots & 0 & & \sin(\theta_1) & 0 & \dots & 0 \\
0 & \cos(\theta_2) & \dots & 0 & & 0 & \sin(\theta_2) & \dots & 0 \\
\vdots & & \ddots & & & & & \ddots & \\
0 & & & \cos(\theta_n) & & & & & \sin(\theta_n) \\
- & - & - & - & - & - & - & - & - \\
\sin(\theta_1) & 0 & \dots & 0 & & \cos(\theta_1) & 0 & \dots & 0 \\
0 & \sin(\theta_2) & \dots & 0 & & 0 & \cos(\theta_2) & \dots & 0 \\
\vdots & & \ddots & & & & & \ddots & \\
0 & & & \sin(\theta_n) & & 0 & & & \cos(\theta_n)
\end{array}\right]
\end{array}
$$

Note that as both points are in $\mathbf{Sch_1}(k, n)$, we can assume that the first principle vector in both sets of principle vectors is the all ones vector and that $\theta_1 = 0$. Then, our geodesic along the Grassmannian will be

$$P(t) = [V(t), V(t)^{\perp}] \begin{bmatrix} 1 & 0 & \dots & 0 & & 0 & 0 & \dots & 0 \\ 0 & \cos(\theta_2 t) & \dots & 0 & & 0 & \sin(\theta_2 t) & \dots & 0 \\ \vdots & & \ddots & & & & & \ddots & \\ 0 & & & \cos(\theta_n t) & & & & & \sin(\theta_n) \\ - & - & - & - & - & - & - & - & - \\ 0 & 0 & \dots & 0 & & 1 & 0 & \dots & 0 \\ 0 & \sin(\theta_2 t) & \dots & 0 & & 0 & \cos(\theta_2 t) & \dots & 0 \\ \vdots & & \ddots & & & & & \ddots & \\ 0 & & & \sin(\theta_n t) & & 0 & & & \cos(\theta_n t) \end{bmatrix}$$

This path will preserve the first vector, our all ones vector, at every point. Therefore, every point on the path will contain the all ones vector. □

Note that a similar proof could be used to show that $\mathbf{Sch}_W(k, n)$ is a totally geodesic submanifold of $\mathbf{Gr}(k, n)$ for any k, n and any choice of W. We have now established that a Schubert variety of the form $\mathbf{Sch_1}(n, 2n)$ is totally geodesic and contains the Daubechies spaces. An interesting experiment is to create a projection matrix from a randomly sampled element in this Schubert variety and observe its effect on an image. The result when applied to an MNIST image of the digit "3" is shown in figure 3.1 below:

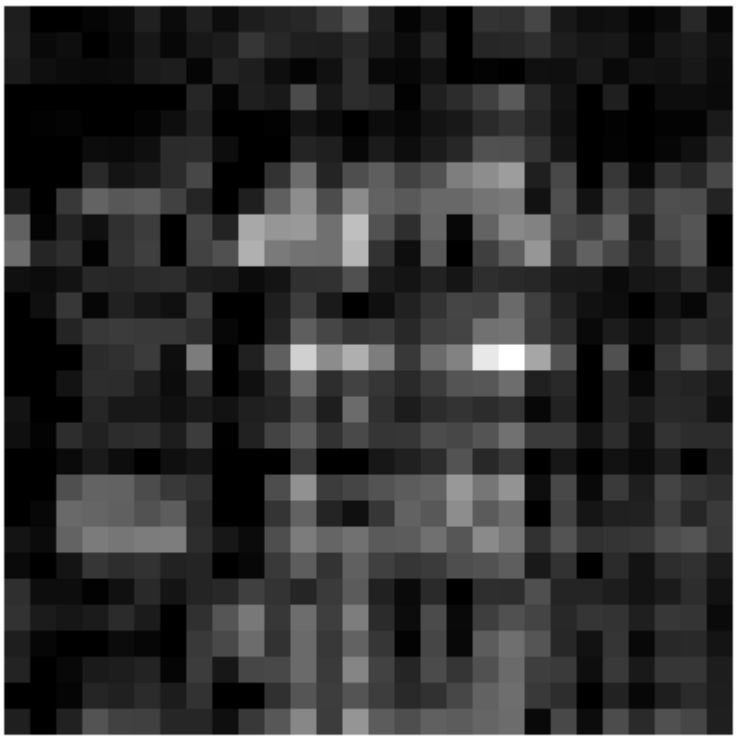


Figure 3.1: Projection to a random element in $\mathbf{Sch_1}(14, 28)$

Note that this is not an image that is legible as the digit "3". Sometimes, a random element in this Schubert variety leads to a somewhat recognizable projection, but typically, the result leads to an unrecognizable image. When we sampled points randomly, we typically did not get something useful. From this we concluded that containing the all ones vector in your subspace is helpful but not sufficient. We next examined further properties of wavelets to lead us to further constraints. To search for a Schubert variety containing “good” projection matrices, we will make our definition of “good” more rigorous. I employed a strategy to create matrix operations from functions related to denoising images. From there, I will create more constrained Schubert varieties from the eigenvalues of these matrices.

3.2 Total Difference Matrix

3.2.1 Introduction

To examine conditions that would lead to a good projection matrix, we will take our cue from a methodology already used in several fields of science. The sum of squares of differences of entries in a matrix is known as the total variance of the matrix. This measurement has been used in movie imaging [25], thermal imaging [26], image registration [27], and image regularization. The total variation of a matrix will depend quadratically on the entries of the matrix. As a related measurement, we will define the Total Difference Statistic of a matrix.

We start with a formula defined in terms of elementary matrices: $D_{ij} = E_{ii} - E_{ij}$. Then

Definition 3.3. *The Total Difference Statistic of a matrix M is defined as*

$$TD(M) = \sum_{i=1}^{n}\sum_{j=1}^{n}(M_{ii} - M_{ij}).$$

Lemma 3.2. *The total difference statistic of a matrix M can be represented through matrix operations as*

$$TD(M) = \textit{trace}\left(\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} MD_{ij}\right)$$

Proof.

$$\begin{aligned}\text{trace}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} MD_{ij}\right) &= \text{trace}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} MD_{ij}\right)\\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\text{trace}\,(MD_{ij})\\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}(M_{ii} - M_{ij})\end{aligned}$$

□

Now that we have this, what does this have to do with Schubert varieties? We will define Schubert varieties through constraints imposed by the eigenvectors of specific matrices. In particular, we will build Schubert varieties from eigenvectors of the Total Difference matrix, D:

$$D = \begin{bmatrix} n-1 & -1 & -1 & -1 & \dots & -1 & -1 \\ -1 & n-1 & -1 & -1 & \dots & -1 & -1 \\ -1 & -1 & n-1 & -1 & \dots & -1 & -1 \\ \vdots & \vdots & & & & & \\ -1 & -1 & -1 & -1 & \dots & -1 & n-1 \end{bmatrix}.$$

Lemma 3.3. *The all ones vector is an eigenvector of D with eigenvalue 0.*

Proof. To see this, we look at a row of this matrix applied to all ones vector.

$$\begin{bmatrix} n-1 & -1 & -1 & -1 & \dots & -1 & -1 \\ -1 & n-1 & -1 & -1 & \dots & -1 & -1 \\ -1 & -1 & n-1 & -1 & \dots & -1 & -1 \\ \vdots & \vdots & & & & & \\ -1 & -1 & -1 & -1 & \dots & -1 & n-1 \end{bmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

□

Lemma 3.4. *Every other eigenvector of D has eigenvalue n.*

Proof. To see this, if we produce $n-1$ linearly independent basis vectors with an eigenvalue of n, then the proof is done. We will pick a set of vectors v_i, $i \in \{2, \dots, n\}$ defined as follows.

$$v_i = e_1 - e_i$$

If we apply the first row of the matrix to v_i, we get $n - 1 - (-1) = n$. Similarly, if we apply the i^{th} row of the matrix to it, we get $-1 + (n-1)(-1) = -n$. Any other row is $-1 + 1 = 0$, so each v_i is an eigenvector with associated eigenvalue n, and they are linearly independent as each

v_i has a unique component e_i. Also, each v_i is perpendicular to the all ones vector. Together they form n linearly independent vectors in an n-dimensional space. □

If we want to minimize the total difference, we will look at the Schubert variety defined with respect to the all one's vector, but we have already examined the Schubert variety with this property, which was insufficient. Perhaps examining all the differences between the rows of a matrix is too much. If we want a reasonable projection, we want projections that, as best as possible, preserve image data from a visual perspective. Instead of a total difference matrix, a matrix that examines local differences may be preferable.

3.3 Local difference Matrix

3.3.1 Background

Definition 3.4. *In an n dimensional space, we will let the following be known as Fourier vectors.*

$$\overline{\sin}\left(\frac{2\pi\cdot k}{n}\right)=\begin{pmatrix}\sin\left(\frac{2\pi\cdot k\cdot 0}{n}\right)\\ \sin\left(\frac{2\pi\cdot k\cdot 1}{n}\right)\\ \vdots\\ \sin\left(\frac{2\pi\cdot k\cdot (n-1)}{n}\right)\end{pmatrix}$$

$$\overline{\cos}\left(\frac{2\pi\cdot k}{n}\right)=\begin{pmatrix}\cos\left(\frac{2\pi\cdot k\cdot 0}{n}\right)\\ \cos\left(\frac{2\pi\cdot k\cdot 1}{n}\right)\\ \vdots\\ \cos\left(\frac{2\pi\cdot k\cdot (n-1)}{n}\right)\end{pmatrix}.$$

The following two lemmas are used to establish formulas concerning eigenvalues and eigenvectors of local difference matrices.

Lemma 3.5.

$$2\sin(a) - (\sin(a+b) + \sin(a-b)) = 2\sin(a)(1-\cos(b)).$$

Proof. Two standard Trigonometric formulas are

$$\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b)$$
$$\sin(a-b) = \sin(a)\cos(b) - \cos(a)\sin(b).$$

From here, we have the following expression

$$\begin{aligned} 2\sin(a) - (\sin(a+b) + \sin(a-b)) &= 2\sin(a) - 2\sin(a)\cos(b) \\ &= 2\sin(a)(1-\cos(b)). \end{aligned}$$

□

Lemma 3.6.

$$2\cos(a) - (\cos(a+b) + \cos(a-b)) = 2\cos(a)(1-\cos(b)).$$

Proof. Two standard Trigonometric formulas are

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$
$$\cos(a-b) = \cos(a)\cos(b) - \sin(a)\sin(b).$$

Again, if we add these two terms together, the rightmost terms cancel each other, so

$$\begin{aligned} 2\cos(a) - (\cos(a+b) + \cos(a-b)) &= 2\cos(a) - 2\cos(a)\cos(b) \\ &= 2\cos(a)(1-\cos(b)). \end{aligned}$$

□

Definition 3.5. *The following is the $n \times n$ 1-Local Difference Matrix, $\Delta(1, n)$:*

$$\Delta(1,n) = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & \dots & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & \dots & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & & & & & & & & \\ 0 & 0 & & & & & & & & \\ \vdots & \vdots & & & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & \dots & 0 & -1 & 2k \end{bmatrix}.$$

Lemma 3.7. *Let n be an even integer greater than 2. Then, for each k with $0 \leq k \leq \frac{n}{2}$, $\overline{\sin}\left(\frac{2\pi \cdot k}{n}\right)$ is an eigenvector for the 1-local difference matrix with eigenvalue $2 - 2\cos\left(\frac{2\pi \cdot k}{n}\right)$.*

Proof. If we apply the 1-local difference matrix to $\overline{\sin}\left(\frac{2\pi \cdot k}{n}\right)$, on each row we essentially get the same equation

$$2\sin\left(\frac{2\pi \cdot k \cdot l}{n}\right) - \left(\sin\left(\frac{2\pi \cdot k \cdot l}{n} + \frac{2\pi \cdot k}{n}\right) + \sin\left(\frac{2\pi \cdot k \cdot l}{n} - \frac{2\pi \cdot k}{n}\right)\right)$$

where the value of l depends on what line you are on and $\sin\left(\frac{2\pi \cdot k \cdot l}{n}\right)$ is the value of that entry. By Lemma 3.5, we know that this equation is equal to

$$2\sin\left(\frac{2\pi \cdot k \cdot l}{n}\right)\left(1 - \cos\left(\frac{2\pi \cdot k}{n}\right)\right).$$

As each entry in the vector gets multiplied by the same value, it is an eigenvector, and its eigenvalue is $2\left(1 - \cos\left(\frac{2\pi \cdot k}{n}\right)\right)$.

□

Lemma 3.8. *Let n be an even integer greater than 2. Then, for each k with $0 \leq k \leq \frac{n}{2}$, $\overline{\cos}\left(\frac{2\pi \cdot k}{n}\right)$ is an eigenvector for the 1-local difference matrix with eigenvalue $2 - 2\cos\left(\frac{2\pi \cdot k}{n}\right)$.*

Proof. If we apply the 1-local difference matrix to $\overline{\cos}(\frac{2\pi \cdot k}{n})$, each line becomes

$$2\cos\left(\frac{2\pi \cdot k \cdot l}{n}\right) - \left(\cos\left(\frac{2\pi \cdot k \cdot l}{n} + \frac{2\pi \cdot k}{n}\right) + \cos\left(\frac{2\pi \cdot k \cdot l}{n} - \frac{2\pi \cdot k}{n}\right)\right).$$

By Lemma 3.6, we know that this equation is equal to

$$2\cos\left(\frac{2\pi \cdot k \cdot l}{n}\right)\left(1 - \cos\left(\frac{2\pi \cdot k}{n}\right)\right).$$

For the same reason as before, this means we have an eigenvector with eigenvalue $2\left(1 - \cos\left(\frac{2\pi \cdot k}{n}\right)\right)$. □

Lemma 3.9. *The 1-local difference matrix is Hermitian positive semidefinite.*

Proof. The 1-local difference matrix is real and symmetric, so it is Hermitian. Also, as all the eigenvalues of the matrix are greater than or equal to zero, it is a positive semidefinite matrix. □

Lemma 3.10. *The projection matrix is Hermitian positive semidefinite.*

Proof. The projection matrix is real and symmetric, so it is Hermitian. Also, as all the eigenvalues of the matrix are greater than or equal to zero, it is a positive semidefinite matrix. □

3.3.2 Sampling from Schubert varieties defined in terms of eigenvectors

We need a couple of theorems to glue everything together. The following is a theorem by von Neumann [28].

Theorem 1. *Let A, B be Hermitian positive semi-definite $n \times n$ matrices with eigenvalues $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ sorted in monotone increasing order. We have the following inequality:*

$$\sum_{i=1}^{n} a_i b_{n-i+1} \leq Tr(AB).$$

Theorem 2. *Let M be the 1-local difference matrix. Let $\lambda_1, \lambda_2, \ldots$ be the eigenvalues of M written in monotone increasing order. Let P be any projection matrix with rank k. Then*

$$\min_P trace(PM) = \sum_{i=1}^{k} \lambda_i.$$

Proof. By finishing the two lemmas and labeling them at the end of the last section, we know that our projection matrices and the 1-local difference matrix are Hermitian positive semidefinite. Therefore, we can use Theorem 1 to find a lower bound.

$$Tr(PM) = Tr(MP) \geq \sum_{i=1}^{n} \lambda_i p_{n-i+1}$$

but the eigenvalues of an $n \times n$ projection matrix of rank k will consist of k ones and $n-k$ zeroes. Therefore

$$\begin{aligned} Tr(MP) &\geq \sum_{i=1}^{n} \lambda_i p_{n-i+1} \\ &\geq \sum_{i=1}^{k} \lambda_i. \end{aligned}$$

We know the eigenvectors of the 1-local difference matrix and we can create a projection matrix, Q, using the eigenvectors $v_1, \ldots, v_k$ associated to the k smallest eigenvalues $\lambda_1, \ldots, \lambda_k$ of M. The matrix QM can be seen to have $\lambda_1, \ldots, \lambda_k$ as eigenvalues with the same eigenvectors $v_1, \ldots, v_k$. We have

$$trace(QM) = \sum_{i=1}^{k} \lambda_i$$

which is the smallest possible trace as allowed by von Neumann's Theorem. □

3.4 k-Local Difference Matrix

Definition 3.6. *The k-local difference of a projection matrix M is defined as*

$$LV(M,k) = \sum_{i=1}^{n} \sum_{j=i-k, j\neq i}^{i+k} (x_{ii} - x_{ij})$$

Definition 3.7. *We can define the $n \times n$ k-local difference matrix as*

$$\Delta(k,n) = \sum_{i=1}^{n} \sum_{j=i-k, j\neq i}^{i+k} D_{ij} = \begin{bmatrix} 2k & -1 & -1 & -1 & \dots & 0 & \dots & -1 & -1 & -1 \\ -1 & 2k & -1 & -1 & \dots & 0 & \dots & 0 & -1 & -1 \\ -1 & -1 & 2k & -1 & \dots & 0 & \dots & 0 & 0 & -1 \\ \vdots & \vdots & & & & & & & & \\ 0 & 0 & & & & & & & & \\ \vdots & \vdots & & & & & & & & \\ -1 & -1 & -1 & 0 & \dots & 0 & \dots & -1 & -1 & 2k \end{bmatrix}.$$

Lemma 3.11. *Let n be an even integer greater than 2. Then, for k with $0 < k < \frac{n}{2}$, we have $\overline{\sin}\left(\frac{2\pi \cdot l}{n}\right)$ is an eigenvector for the k-local difference matrix with eigenvalue $2 - 2\cos\left(\frac{2\pi \cdot i \cdot l}{n}\right)$.*

Proof. If we apply the local difference matrix to $\overline{\sin}\left(\frac{2\pi \cdot k}{n}\right)$, on each row we essentially get the same equation

$$2\sin\left(\frac{2\pi \cdot k \cdot l}{n}\right) - \left(\sin\left(\frac{2\pi \cdot k \cdot l}{n} + \frac{2\pi \cdot k}{n}\right) + \sin\left(\frac{2\pi \cdot k \cdot l}{n} - \frac{2\pi \cdot k}{n}\right)\right)$$

where the value of l depends on what line you are on and $\sin(\frac{2\pi \cdot k \cdot l}{n})$ is the value of that entry. By Lemma 3.5, we know that this equation is equal to

$$2\sin\left(\frac{2\pi \cdot k \cdot l}{n}\right)\left(1 - \cos\left(\frac{2\pi \cdot k}{n}\right)\right).$$

As each entry in the vector gets multiplied by the same value, it is an eigenvector, and its eigenvalue is $2\left(1-\cos\left(\frac{2\pi\cdot k}{n}\right)\right)$.

□

Lemma 3.12. *Let n be an even integer greater than 2. Then, for k with $0 < k < \frac{n}{2}$, we have $\overline{\cos}(\frac{2\pi\cdot l}{n})$ is an eigenvector for the k-local difference matrix with eigenvalue $2-2\cos(\frac{2\pi\cdot l}{n})$.*

Proof. If we apply the local difference matrix to $\overline{\cos}\left(\frac{2\pi\cdot l}{n}\right)$, each line becomes

$$2\cos\left(\frac{2\pi\cdot k\cdot l}{n}\right)-\left(\cos\left(\frac{2\pi\cdot k\cdot l}{n}+\frac{2\pi\cdot k}{n}\right)+\cos\left(\frac{2\pi\cdot k\cdot l}{n}-\frac{2\pi\cdot k}{n}\right)\right).$$

By Lemma 3.6 we know that this equation is equal to

$$2\cos\left(\frac{2\pi\cdot k\cdot l}{n}\right)\left(1-\cos\left(\frac{2\pi\cdot k}{n}\right)\right).$$

As before, this means we have an eigenvector with eigenvalue $2\left(1-\cos\left(\frac{2\pi\cdot k}{n}\right)\right)$. □

This means that the eigenvectors that we use to define Schubert varieties, from the local difference matrix, are the same eigenvectors we use to form Schubert varieties associated to a k-local difference matrix. While convenient to know, this does not extend to later sections where we enforce a boundary to the local difference matrix.

3.5 Schubert varieties from k-local Difference Matrices.

We associated a distinguished Schubert variety to the total difference matrix, $\mathbf{Sch}_1(n,2n)$. It is natural to ask whether a useful Schubert variety can be constructed from eigenvectors of a k-local difference matrix. With total variation, the Schubert variety was created by picking the eigenvectors that minimized the total variation, so what if we apply the same strategy with k-local difference matrices?

Note that the submanifold we create will be a submanifold of $\mathbf{Sch}_1(n,2n)$ as the all ones vector is an eigenvector of the k-local difference matrices (with or without boundary) with eigenvalue 0.

Definition 3.8. *Let $\Delta(k,n)$ denote the k-local difference matrix of size $n \times n$. Let $v_1, \ldots, v_n$ be its eigenvectors arranged so that the corresponding eigenvalues go from smallest to largest. We define the subspace $\mathbf{Diff}(j,k,n)$ of $\mathbb{R}^n$ and the Schubert variety $\mathbf{Sch}(j,k|l,n)$ by*

$$\mathbf{Diff}(j,k,n) := span\{v_1, v_2, \ldots, v_j\}$$

$$\mathbf{Sch}(j,k|l,n) = \mathbf{Sch}_{\mathbf{Diff}(j,k,n)}(l,n).$$

We now examine the effect of projection matrices built from random samples from $\mathbf{Sch}(j,k|l,n)$.

3.6 Random projections from k-local difference Schubert Varieties

MNIST samples are 28×28 matrices corresponding to fairly low resolution black and white images of hand draw digits. We will illustrate the effect of random projections on such images by compressing to 14×14 black and white images. To do this, we will start with $l = 1, k = 14$, and $n = 28$ and let j vary. Then, we will pick random elements in $\mathbf{Sch}(j,1|14,28)$ with various values for j and observe the effect of the projection matrices we create from these random elements.

(a) j=2 **(b)** j=4 **(c)** j=6 **(d)** j=8 **(e)** j=10

Figure 3.2: Sampling with different values of j in $\mathbf{Sch}(j,1|14,28)$

These should be compared to the image we obtain if we pick a random point on the Grassmannian $\mathbf{Gr}(14,28)$ as in figure 3.3:

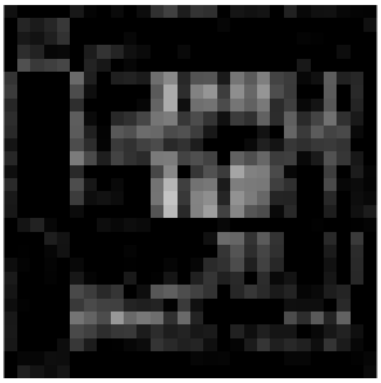


Figure 3.3: Using a random point on $\mathbf{Gr}(14, 28)$ to create a projection matrix

These random projections from the special Schubert variety $\mathbf{Sch}(j, 1|14, 28)$ are significantly better than a random projection (particularly for larger j). This leads to several questions. Are there related Schubert varieties that also have good properties? If we enforce a boundary in our local difference matrices instead of full circulant difference matrices, will that harm or improve our results?

3.7 Local Difference Matrices with boundary

Digital pictures have boundaries and we would like to take this into account by defining local distance matrices with boundaries, like the one below.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & & & & & \\ 0 & 0 & 0 & 0 & \dots & -1 & 1 \end{bmatrix}$$

Fortunately, we can use a lot of machinery we have already developed in earlier sections. To denote local difference with boundary, I use Δ_B.

Definition 3.9. *The* 1*-local difference with boundary of a matrix* M *is defined as*

$$TDB(M) = \sum_{i=1}^{n} \sum_{j=\max\{1,i-1\}, j\neq i}^{\min\{i+1,n\}} (x_{ii} - x_{ij}).$$

Definition 3.10. *We define the* 1*-local difference with boundary matrix as*

$$\Delta_B(1,n) = \sum_{i=1}^{n} \sum_{j=\max\{1,i-1\}, j\neq i}^{\min\{i+1,n\}} D_{ij}.$$

Lemma 3.13. *The all ones vector is an eigenvector of* $\Delta_B(1,n)$ *with eigenvalue 0.*

Proof. Each line of the matrix sums to 0, and if you apply a matrix to the all ones vector, each entry in the vector becomes the sum of the entries in the row of the matrix. □

Lemma 3.14. *Let* n *be an even integer greater than 2. Then, for* $1 \leq l \leq \frac{n}{2}$, $\overline{\cos}\left(\frac{2\pi \cdot l}{n}\right)$ *is not an eigenvector for the* 1*-local difference with boundary matrix,* $\Delta_B(1,n)$.

Proof. We can split the behavior of the matrix below into two parts.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & & & & & \\ 0 & 0 & 0 & 0 & \dots & -1 & 1 \end{bmatrix} \begin{pmatrix} \cos\left(\frac{2\pi \cdot l \cdot 0}{n}\right) \\ \cos\left(\frac{2\pi \cdot l \cdot 1}{n}\right) \\ \cos\left(\frac{2\pi \cdot l \cdot 2}{n}\right) \\ \vdots \\ \cos\left(\frac{2\pi \cdot l \cdot (n-1)}{n}\right) \end{pmatrix}$$

The second and the second-to-the-last rows of the matrix act on the vector differently from how the first and and the last rows act on the vector.

The entries in our first part are just repetitions of what we did in lemma 3.6 where it scales that component by $2\left(1-\cos\left(\frac{2\pi\cdot l}{n}\right)\right)$. For our second case, the action is different, so it will not scale that component the same way as before. Therefore, $\overline{\cos}\left(\frac{2\pi\cdot l}{n}\right)$ is not an eigenvector for $\Delta_B(1,n)$. □

Lemma 3.15. *Let n be an even integer greater than 2. Then, $\overline{\sin}\left(\frac{2\pi\cdot l}{n}\right), 1 \leq l \leq \frac{n}{2}$ is not an eigenvector for the local difference with boundary matrix.*

Proof. We can split the behavior of the matrix below into two parts.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & & & & & \\ 0 & 0 & 0 & 0 & \dots & -1 & 1 \end{bmatrix} \begin{pmatrix} \sin\left(\frac{2\pi\cdot l\cdot 0}{n}\right) \\ \sin\left(\frac{2\pi\cdot l\cdot 1}{n}\right) \\ \sin\left(\frac{2\pi\cdot l\cdot 2}{n}\right) \\ \vdots \\ \sin\left(\frac{2\pi\cdot l\cdot (n-1)}{n}\right) \end{pmatrix}$$

The second to second-last rows of the matrix act on the vector differently from how the first and last rows act on the vector.

The entries in our first part are just repetitions of what we did in lemma 3.5 where it scales that component by $2\left(1-\cos\left(\frac{2\pi\cdot l}{n}\right)\right)$. For our second case, the action is different, so it will not scale that component the same way as the other. Therefore, $\overline{\sin}\left(\frac{2\pi\cdot l}{n}\right)$ is not an eigenvector for $\Delta_B(1,n)$. □

This means we will obtain different Schubert varieties from the eigenvectors of local difference matrices with boundary than from the eigenvectors of local difference matrices without boundary.

However, a more precise description of eigenvectors of local difference matrices with boundary is challenging. To circumvent this challenge, we will rely on computer-generated eigenvectors to build the Schubert variety. Another problem found experimentally is the "localness" of the variance when we include boundary changes in the eigenvectors we find. This means we do not have the "uniqueness" of eigenvectors consisting of just the Fourier modes like we had when we excluded the boundary.

Definition 3.11. *Let $\Delta_B(k,n)$ denote the k-local difference matrix with boundary of size $n \times n$. Let $v_1, \ldots, v_n$ be its eigenvectors arranged so that the corresponding eigenvalues go from smallest to largest. We define the subspace $\mathbf{Diff}_B(j,k,n)$ of $\mathbb{R}^n$ and the Schubert cycle $\mathbf{Sch}_B(j,k|l,n)$ by*

$$\mathbf{Diff}_B(j,k,n) := span\{v_1, v_2, \ldots, v_j\}$$

$$\mathbf{Sch}_B(j,k|l,n) = \mathbf{Sch}_{\mathbf{Diff}_B(j,k,n)}(l,n).$$

(a) j=2 **(b)** j=4 **(c)** j=6 **(d)** j=8 **(e)** j=10

Figure 3.4: Sampling with different values of j in $\mathbf{Sch}_B(j,1|14,28)$

Chapter 4

Sparse Matrix Varieties

Wavelets have been used for several decades now in areas ranging from quantum chemistry [29] and atmospheric science [30] to signal processing, data compression, and image analysis. Daubechies spaces, the vectors spaces built from discrete Daubechies wavelets, correspond to particular points on Grassmannians. In general, Grassmannians are high-dimensional spaces; we would like to focus our attention on a much smaller region of the Grassmannian. In particular, we are interested in the region of the Grassmannian that yields suitable projection matrices for the purpose of compressing digital images. In Chapter 2, we found methods for creating geodesic paths in a Grassmannian. This involved building a matrix that incorporated both the space under consideration and its orthogonal complement. We saw in Chapter 3 that we could restrict to more fruitful regions of Grassmannians by requiring certain subspace containment conditions. Doing so allowed us to create "random" projections that retained more structure in projected/compressed digital images. This chapter focuses on how to move on a Grassmannian to an advantageous subregion (typically lying close to the region containing Daubechies spaces). We apply optimization algorithms using rotation matrices. This method has computational benefits and avoids the use of the orthogonal complement space.

Our goal is to describe, build, and sample *sparse matrix varieties*. To create these subvarieties, we look at constraints satisfied by discrete Daubechies wavelets and their associated Daubechies spaces. Using orthogonal matrices, we describe a geometric approach for moving towards spaces with better projection properties with respect to a digital image. An essential observation, allowing for optimizing over these varieties, is that orthogonal matrices preserve inner products.

4.1 Further structure within the Daubechies wavelets

4.1.1 A closer look at D_4

In the last chapter, we looked at different ways of discovering potentially suitable projection matrices. Furthering our goals in this direction, we will look more closely at the discrete Daubechies wavelets D_4 and D_6 to motivate the constructions of a family of distinguished subvarieties of a Grassmannian.

With D_4, you have four nonzero entries such that when you cyclically shift (by two) you obtain an orthogonal $2n \times n$ matrix. In Chapter 2, we had the algorithm, algorithm 1 on page 23, to create Daubechies spaces in $\mathbf{Gr}(n, 2n)$ from Daubechies wavelets. In doing so, we create a matrix, M, of the form:

$$M = \begin{bmatrix} v_1 & 0 & 0 & \dots & v_3 \\ v_2 & 0 & 0 & \dots & v_4 \\ v_3 & v_1 & 0 & \dots & 0 \\ v_4 & v_2 & 0 & \dots & 0 \\ \vdots & & & & \end{bmatrix}.$$

While the values of the v_i's are well known for D_4, we will treat them as unknowns and consider constraints such that cyclic shifts (by two) leads to a $2n \times n$ matrix M that satisfies $M^T M = I$. Note that this matrix is quite sparse (i.e. most of the matrix is filled with zeros). To be clear, we have the matrix expression $M^T M = I$:

$$\begin{bmatrix} v_1 & v_2 & v_3 & v_4 & 0 & \dots & 0 \\ 0 & 0 & v_1 & v_2 & v_3 & \dots & 0 \\ \vdots & & & & & & \end{bmatrix} \begin{bmatrix} v_1 & 0 & 0 & \dots & v_3 \\ v_2 & 0 & 0 & \dots & v_4 \\ v_3 & v_1 & 0 & \dots & 0 \\ v_4 & v_2 & 0 & \dots & 0 \\ \vdots & & & & \end{bmatrix} = I.$$

Notice that the circulant nature of the entries and the sparsity lead to only a small number of conditions forced upon the entries of M. In the four variable case, there are two constraints placed upon our v_i's:

$$v_1^2 + v_2^2 + v_3^2 + v_4^2 = 1$$
$$v_1 \cdot v_3 + v_2 \cdot v_4 = 0.$$

We can view these conditions in terms of dot products of length 2 vectors as follows:

$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} v_3 & v_4 \end{pmatrix} \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} = 1$$
$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} = 0.$$

A strategy used in Chapter 2 was to form a matrix where the columns were vectors with certain constraints. We have the same situation here. We now consider the variables in M arranged in a 2×2 matrix:

$$V = \begin{bmatrix} v_1 & v_3 \\ v_2 & v_4 \end{bmatrix}.$$

Applying an orthogonal matrix on the left corresponds to a row operation that changes these components while keeping the inner products constraints listed above. This changes the column space of the matrix we build which in turn changes the projection. We illustrate this with an example. Let $R \in \mathbf{O}(2)$ and consider the entries a_1, a_2, a_3, a_4 obtained after we multiply R and V:

$$\begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} = R \cdot \begin{bmatrix} v_1 & v_3 \\ v_2 & v_4 \end{bmatrix}.$$

Then our dot product equations are transformed as:

$$\begin{aligned}\begin{pmatrix} a_1 & a_2 \end{pmatrix}\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \begin{pmatrix} a_3 & a_4 \end{pmatrix}\begin{pmatrix} a_3 \\ a_4 \end{pmatrix} &= \begin{pmatrix} v_1 & v_2 \end{pmatrix} R^T R \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} v_3 & v_4 \end{pmatrix} R^T R \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} \\ &= \begin{pmatrix} v_1 & v_2 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} v_3 & v_4 \end{pmatrix}\begin{pmatrix} v_3 \\ v_4 \end{pmatrix} \\ &= 1\end{aligned}$$

and

$$\begin{aligned}\begin{pmatrix} a_1 & a_2 \end{pmatrix}\begin{pmatrix} a_3 \\ a_4 \end{pmatrix} &= \begin{pmatrix} v_1 & v_2 \end{pmatrix} R^T R \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} \\ &= \begin{pmatrix} v_1 & v_2 \end{pmatrix}\begin{pmatrix} v_3 \\ v_4 \end{pmatrix} \\ &= 0.\end{aligned}$$

We can thus create a new matrix with properties similar to the Daubechies spaces' matrix.

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & 0 & \dots & 0 \\ 0 & 0 & a_1 & a_2 & a_3 & \dots & 0 \\ \vdots & & & & & & \end{bmatrix}\begin{bmatrix} a_1 & 0 & 0 & \dots & 0 \\ a_2 & 0 & 0 & \dots & 0 \\ a_3 & a_1 & 0 & \dots & 0 \\ a_4 & a_2 & 0 & \dots & 0 \\ \vdots & & & & \end{bmatrix} = I$$

Earlier in this thesis, we mentioned in Lemma 2.2 that the map F, that sends an element M on a Stiefel manifold to the projection matrix MM^T, is homogeneous. In other words, right multiplying an orthogonal $n \times k$ matrix M by an element $R \in \mathbf{O}(k)$ produces a new matrix, $N = MR$ that

satisfies $NN^T = MM^T$. Altering the basis of vectors for the columns space of a matrix does not change the projection matrix. Will applying an orthogonal matrix to V lead to a matrix with a different column space? To examine this, consider the matrix operations that take us from our original column vectors to our new ones.

Let $R \in \mathbf{O}(2)$ and let's examine how we convert from one element in our space to another:

$$\begin{aligned} A = \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} &= R \cdot \begin{bmatrix} v_1 & v_3 \\ v_2 & v_4 \end{bmatrix} \\ &= \begin{bmatrix} R\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} & R\begin{pmatrix} v_3 \\ v_4 \end{pmatrix} \end{bmatrix} \end{aligned}$$

So now, if we form a matrix from the vector determined by the entries in A, with the same construction we had for D_4, we get:

$$\begin{bmatrix} R\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \dots & R\begin{pmatrix} v_3 \\ v_4 \end{pmatrix} \\ R\begin{pmatrix} v_3 \\ v_4 \end{pmatrix} & R\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \dots & \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & R\begin{pmatrix} v_3 \\ v_4 \end{pmatrix} & R\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} & \dots & \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{bmatrix}.$$

which we can factor as:

$$\begin{bmatrix} R & 0 & 0 & \dots & 0 \\ 0 & R & 0 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & R \end{bmatrix} \begin{bmatrix} v_1 & 0 & 0 & \dots & v_3 \\ v_2 & 0 & 0 & \dots & v_4 \\ v_3 & v_1 & 0 & \dots & 0 \\ v_4 & v_2 & 0 & \dots & 0 \\ \vdots & & & & \end{bmatrix}.$$

Then we note that the projection matrix, $ProjMat(A) = AA^T$, onto the column space of A can be written as:

$$\begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & R \end{bmatrix} \begin{bmatrix} v_1 & 0 & \dots & v_3 \\ v_2 & 0 & \dots & v_4 \\ v_3 & v_1 & \dots & 0 \\ v_4 & v_2 & \dots & 0 \\ \vdots & & & \end{bmatrix} \cdot \left(\begin{bmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & R \end{bmatrix} \begin{bmatrix} v_1 & 0 & \dots & v_3 \\ v_2 & 0 & \dots & v_4 \\ v_3 & v_1 & \dots & 0 \\ v_4 & v_2 & \dots & 0 \\ \vdots & & & \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} R & 0 & 0 & \dots & 0 \\ 0 & R & 0 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & R \end{bmatrix} ProjMat(V) \begin{bmatrix} R^T & 0 & 0 & \dots & 0 \\ 0 & R^T & 0 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & R^T \end{bmatrix}.$$

Here, the induced action of R is applied to either side of the projection matrix VV^T. The actions of the orthogonal matrix created from R does not cancel out, which is fundamental for the proof of Lemma 2.2.

One more thing to notice about this situation: we have four variables and two constraints, and thus, we have two degrees of freedom. So far, we have found a way to realize one degree of freedom via left multiplication with elements of $\mathbf{O}(2)$. Is there another degree of freedom hiding within these equations that we can identify? By reexamining our equations and rearranging the

entries of V, we create a new matrix to which we can multiply elements of $\mathbf{O}(2)$ while retaining our original constraints:

$$v_1^2 + v_2^2 + v_3^2 + v_4^2 = 1$$
$$v_1 \cdot v_3 + v_2 \cdot v_4 = 0.$$

To do so, we arrange the entries of V to create slightly different vectors with constraints:

$$\begin{pmatrix} v_1 & v_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_4 \end{pmatrix} + \begin{pmatrix} v_3 & v_2 \end{pmatrix} \begin{pmatrix} v_3 \\ v_2 \end{pmatrix} = 1$$

and

$$\begin{pmatrix} v_1 & v_4 \end{pmatrix} \begin{pmatrix} v_3 \\ v_2 \end{pmatrix} = 0.$$

From this permutation of the entries of V, we can create a new matrix

$$V' = \begin{bmatrix} v_1 & v_3 \\ v_4 & v_2 \end{bmatrix}.$$

If we apply an orthogonal matrix to V', we will keep the conditions on the original matrix V, but it will have a different effect after reversing our permutation. By rearranging the components of our vector, we are able to apply an orthogonal 2×2 matrix in two distinct ways. Elements of $\mathbf{O}(2)$ can be identified with the manifold consisting of two copies of a circle. We can see from above that there are two distinct actions of $\mathbf{O}(2)$ acting on our starting matrix which maintain the required conditions on the dot products. These two actions correspond to our two local degrees of freedom.

4.1.2 A closer look at D_6

Looking at the D_6 case, we once more make a matrix by cyclic permutation (shifting two steps) of a starting vector, but this time the starting vector has six nonzero entries:

$$\begin{bmatrix} v_1 & 0 & 0 & \dots & v_3 \\ v_2 & 0 & 0 & \dots & v_4 \\ v_3 & v_1 & 0 & \dots & v_5 \\ v_4 & v_2 & 0 & \dots & v_6 \\ v_5 & v_3 & v_1 & \dots & 0 \\ v_6 & v_4 & v_2 & \dots & 0 \\ 0 & v_5 & v_3 & \dots & 0 \\ \vdots & & & & \end{bmatrix}.$$

In the case of D_6, the product of this matrix times its transpose is again the identity matrix. We have the following matrix expression:

$$\begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & 0 & \dots & 0 \\ 0 & 0 & v_1 & v_2 & v_3 & v_4 & v_5 & \dots & 0 \\ \vdots & & & & & & & & \end{bmatrix} \begin{bmatrix} v_1 & 0 & 0 & \dots & v_3 \\ v_2 & 0 & 0 & \dots & v_4 \\ v_3 & v_1 & 0 & \dots & v_5 \\ v_4 & v_2 & 0 & \dots & v_6 \\ v_5 & v_3 & v_1 & \dots & 0 \\ v_6 & v_4 & v_2 & \dots & 0 \\ 0 & v_5 & v_3 & \dots & 0 \\ \vdots & & & & \end{bmatrix} = I.$$

From this, we can conclude there are three algebraic constraints instead of two constraints as in the case with four variables:

$$\begin{aligned} v_1^2 + v_2^2 + v_3^2 + v_4^2 + v_6^2 + v_5^2 &= 1 \\ v_1 \cdot v_5 + v_2 \cdot v_6 &= 0 \\ v_1 \cdot v_3 + v_2 \cdot v_4 + v_3 \cdot v_5 + v_4 \cdot v_6 &= 0 \end{aligned}$$

Similar to the earlier case with 4 variables, we can rearrange these into the following dot product expressions:

$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} + \begin{pmatrix} v_3 & v_4 \end{pmatrix} \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} + \begin{pmatrix} v_5 & v_6 \end{pmatrix} \begin{pmatrix} v_5 \\ v_6 \end{pmatrix} = 1$$

$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_5 \\ v_6 \end{pmatrix} = 0$$

$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} + \begin{pmatrix} v_3 & v_4 \end{pmatrix} \begin{pmatrix} v_5 \\ v_6 \end{pmatrix} = 0.$$

From here, we create a 2×3 matrix where the columns of the matrix are the vector building blocks:

$$V = \begin{bmatrix} v_1 & v_3 & v_5 \\ v_2 & v_4 & v_6 \end{bmatrix}.$$

We have now seen behavior for the case of 4 variables that is similar to behavior for the case of 6 variables. This behavior continues both with the Daubechies wavelets $D_8, D_{10}, \ldots$ and, analogously, with 8 variables, 10 variables, etc. These matrix varieties are sparse and contain Daubechies wavelets. Furthermore, we can move locally on this manifold using rotation matrices. We refer to elements in the family of varieties built in this manner as *Sparse Matrix Varieties*.

Definition 4.1. *Let $j \in \mathbb{N}$. Define the sparse matrix variety, SparseMat(2,j), as follows:*

$$\textit{SparseMat}(2,j) = \left\{ \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1j} \\ v_{21} & v_{22} & \dots & v_{2j} \end{bmatrix} \mid v_{ij} \in \mathbb{R} \text{ satisfy C1 and C2} \right\}.$$

The first condition, C1, is that for each $r \in \{1, \dots, j-1\}$

$$\sum_{i=1}^{r} \sum_{m=1}^{2} v_{im} v_{(n+1-i)m} = 0$$

The second condition, C2, is that

$$\sum_{i=1}^{j} \sum_{m=1}^{2} v_{im}^2 = 1.$$

We now return to our examination of the 6 variable case. As seen in the D_4 case, we can swap the end components of the vector and preserve the inner product formulas:

$$V' = \begin{bmatrix} v_1 & v_3 & v_5 \\ v_6 & v_4 & v_2 \end{bmatrix}.$$

We have six variables with three algebraic constraints. We expect that this will define a three dimensional variety. We have two local directions by applying rotation matrices when v_2 and v_6 are swapped and when they are not swapped. This indicates that we should look for one more local degree of freedom. To get this third direction, we need to be more flexible in rearranging before applying rotation matrices.

So far, we have been applying a permutation along the bottom row of the matrix. This strategy will no longer work since if we apply a permutation that moves v_4 then we can no longer guarantee that the dot product equations will still be satisfied. One possibility is to alter the basis before permuting elements while maintaining the dot products using the same permutation strategy.

To do this, we focus on a suitable set of basis vectors. Ideally, we should pick vectors built from the entries of V. Fortunately, the column vectors at the start and end of the element have

to be orthogonal. Using the normalized version of these vectors, we will change the elements as follows:

$$w_1 = \begin{bmatrix} v_1 \\ \\ v_2 \end{bmatrix} \quad w_2 = \begin{bmatrix} v_5 \\ \\ v_6 \end{bmatrix}.$$

Then, we normalize w_1 and w_2 by redefining them as:

$$w_1 := \frac{w_1}{|w_1|} \quad w_2 := \frac{w_2}{|w_2|}.$$

We use these two vectors to build the rows of an orthogonal matrix

$$W = \begin{bmatrix} w_1^T \\ \\ w_2^T \end{bmatrix}.$$

We get the result by applying our matrix W to the matrix V resulting in:

$$WV = \begin{bmatrix} a & b & 0 \\ \\ 0 & c & d \end{bmatrix}.$$

Note that the two zeroes in the matrix WV arise from the orthogonality of the first and third column of V. Now, we can again permute these entries. If we swap c and d, we will keep all our previous conditions intact, although we will have a new issue. How do we unpermute these in a way that is related to our original matrix if we start applying rotation matrices?

One strategy is to apply the permutation to our element so that we end up with a matrix of the form:

$$\begin{bmatrix} a & b & 0 \\ \\ c & d & 0 \end{bmatrix}.$$

We keep track of our operations so that after applying permutations, we can revert to our original basis. We will show an example of using permutations and rotations to alter our elements in SparseMat(2,3). Still, one question we should ask ourselves before continuing is how can we ensure that the action of the third direction is not simply a combination of the previous two actions? To do this, we examine the effect of each action on the magnitude of the columns of the element. We apply rotation matrices to the original matrix

$$V = \begin{bmatrix} v_1 & v_3 & v_5 \\ v_2 & v_4 & v_6 \end{bmatrix}.$$

We notice that this is simply rotating all the vectors, so the magnitude of the column vectors will remain unchanged. The magnitude of the first and last column vectors will change if we apply a rotation matrix to our second arrangement:

$$V' = \begin{bmatrix} v_1 & v_3 & v_5 \\ v_6 & v_4 & v_2 \end{bmatrix}.$$

Here, the components of the first and last columns interact. This changes the magnitudes of the vectors $\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ and $\begin{pmatrix} v_5 \\ v_6 \end{pmatrix}$, but it leaves the magnitude of the vector comprised of the second column alone as that vector is rotated again. Note that we can even make one of the vectors 0 here by looking at the operations. To see this, let's apply a rotation matrix to make $v_1 = 0$ with the assumption that $v_6 \neq 0$

$$\begin{aligned}\cos(\theta)v_1 + \sin(\theta)v_6 &= 0\\ \sin(\theta)v_6 &= -\cos(\theta)v_1\\ \tan(\theta) &= -\frac{v_1}{v_6}\\ \theta &= \arctan\left(-\frac{v_1}{v_6}\right).\end{aligned}$$

Note that if we apply $R(\arctan(-\frac{v_1}{v_6}))$ to the matrix we get the following form

$$\begin{bmatrix} 0 & v_3' & v_5' \\ v_6' & v_4' & v_2' \end{bmatrix}.$$

The conditions on our inner products are still the same, so:

$$\begin{aligned}\begin{pmatrix} 0 & v_6' \end{pmatrix}\begin{pmatrix} v_5' \\ v_2' \end{pmatrix} &= 0\\ v_6' \cdot v_2' &= 0.\end{aligned}$$

Next note that

$$\begin{aligned}\begin{pmatrix} 0 & v_6' \end{pmatrix}\begin{pmatrix} 0 \\ v_6' \end{pmatrix} &= \begin{pmatrix} v_1 & v_6 \end{pmatrix}\begin{pmatrix} v_1 \\ v_6 \end{pmatrix}\\ (v_6')^2 &= v_1^2 + v_6^2\\ v_6' &= \sqrt{v_1^2 + v_6^2}\end{aligned}$$

as $v_6' \neq 0$, $v_2' = 0$. Thus, we can make the magnitude of the vector $\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ zero by permuting the second row of our matrix and applying rotation matrices.

Lastly, from our third perspective, we can simultaneously alter the magnitude of all three columns. Consider the following matrix again:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \end{bmatrix}.$$

We know that a is the magnitude of the first column of our element in SparseMat(2,3), and d is the magnitude of the third column. Using the same strategy as we did when constructing the second matrix V', we can show the magnitudes of all three column vectors change simultaneously. We do this by determining the conditions it would take to make a equal to zero:

$$\begin{aligned} \cos(\theta)a + \sin(\theta)c &= 0 \\ \sin(\theta)c &= -\cos(\theta)a \\ \tan(\theta) &= -\frac{c}{a} \\ \theta &= \arctan\left(-\frac{c}{a}\right). \end{aligned}$$

Note that if we apply $R(\arctan(-\frac{c}{a}))$ to the matrix we get a matrix with the following form:

$$\begin{bmatrix} 0 & b' & 0 \\ c' & d' & 0 \end{bmatrix}.$$

Yet again dot products are preserved so:

$$\begin{pmatrix} 0 & c' \end{pmatrix} \begin{pmatrix} b' \\ d' \end{pmatrix} = 0$$

$$c' \cdot d' = 0.$$

As $c' \neq 0$ this means that $d' = 0$. This means that applying rotation matrices to this altered form of our element can reduce the magnitude of the first and third columns to zero. Furthermore, as we still have the constraint that the sum of the squares of each term sum to one, the magnitude of the second column has to increase to one, so this "direction" has a different effect on the elements of SparseMat(2,j) than the other two directions.

Algorithm 3 Convert an element of SparseMat(2,j) to a Column Matrix

Input: $M \in$ SparseMat(2,j), $n \in \mathbb{N}$
Output: ColMat $\in$ Mat(2n,n)

```
1: function ColMat(M,n)
2:     vec = zeros(2*n,1)
3:     j = ColSize(M)
4:     for i =1:j  do
5:         vec(2*(i-1)+1:2*i) = M(:,i)
6:     ColMat=zeros(2n,n)
7:     for i=1:n do
8:         ColMat(:,i) = circVec(vec,2*(i-1))
```

Algorithm 4 Convert an element of SparseMat(2,j) to a projection Matrix

Input: $M \in$ SparseMat(2,j), $n \in \mathbb{N}$
Output: Proj $\in$ Mat(2n,2n)

```
1: function CSTP(M,n)
2:     ColMatrix = ColMat(M,n)
3:     Proj = ColMatrix*ColMatrix^T
```

4.2 Energy Function

Now that we know how to move along a sparse matrix variety, how do we decide the direction to move for a given purpose? As an example, we will pick a function we would like to optimize on a sparse matrix variety. The Frobenius norm is a useful matrix statistic that has found applications in area such as image denoising [31]. The square of the Frobenius norm of a matrix is the sum of the squares of the entries in the matrix. Letting $\|\cdot\|_F$ denote the Frobenius norm, we have the following expression for a matrix $M \in \mathbb{R}^{n\times m}$:

$$\|M\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{m} M_{ij}^2 = \mathrm{Trace}(M^T M).$$

In the algorithms that follow, I will sometimes denote the code to get the square of the Frobenius norm of a matrix M as $fb(M)$. We will define an energy function in terms of the Frobenius norm.

In the case we have discussed so far, if we use special orthogonal matrices, these correspond to rotations which can be identified with points on a circle. On a circle, we can only move clockwise or counterclockwise. We will monitor whether our energy function increases or decreases when we rotate in a given direction. When we have found the correct direction in which to move, we will continue in this direction until it stops increasing the value of the energy function.

We now examine the effect of applying rotation matrices, on the energy function, as we move along a sparse matrix variety. Below is the algorithm that does this monitoring. To examine the behavior, we can input D_6 as an element of SparseMat(2,3) into this algorithm and investigate any behavior about how the Frobenius energy changes.

Algorithm 5 Rotate along the path

Input: Image, $M \in$SparseMat(2,j)

function ROTATE2π(Image,M)
    Energy = []
    for θ = 0:0.01:2π **do**
        tempM = Rotate(θ)$\cdot$ M
        tempProj = CSTP(tempM,n)
        tempImage = tempProj*Image
        TempEnergy = $\|$TempImage$\|_F$
        Energy.append(TempEnergy)
    Plot(θ,Energy)

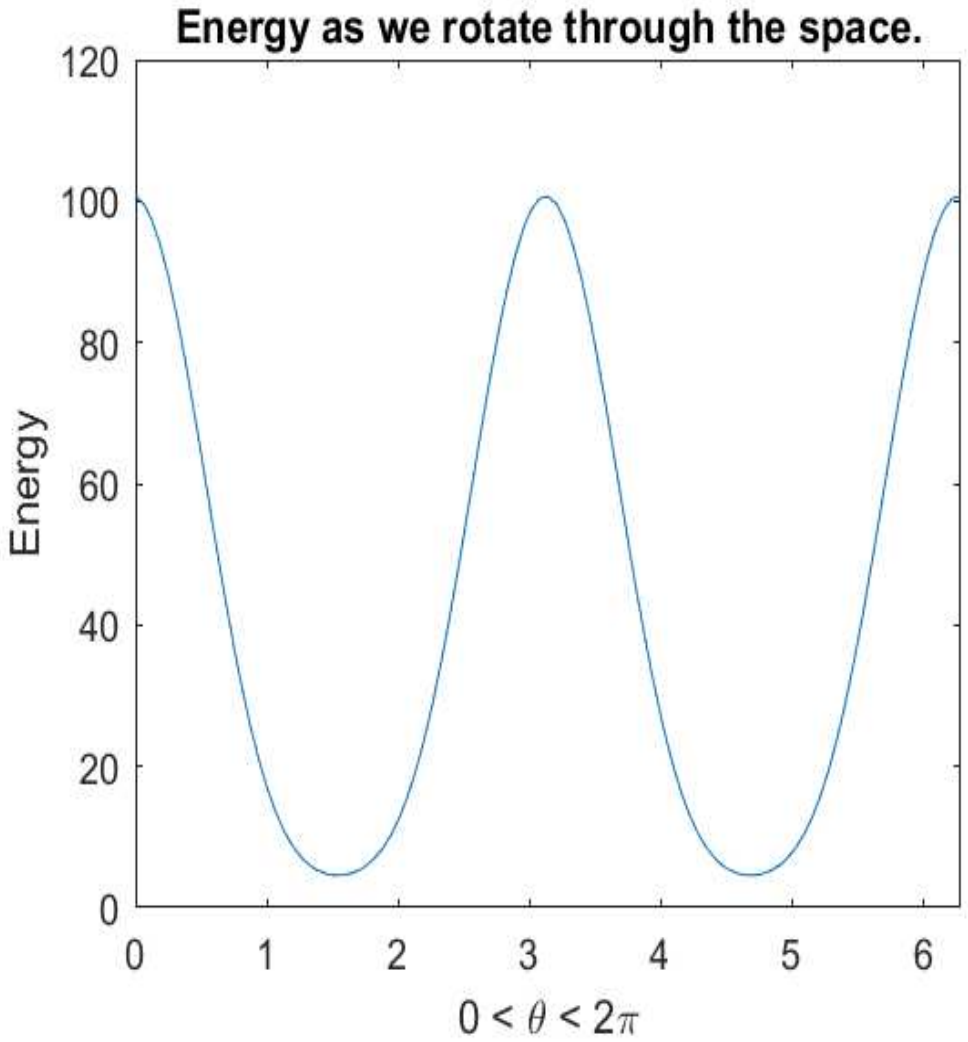


Figure 4.1: Energy versus Rotation with D_6 as a starting point

(a) $\theta = 0$ **(b)** $\theta = \frac{\pi}{2}$ **(c)** $\theta = \pi$ **(d)** $\theta = \frac{3\pi}{2}$ **(e)** $\theta = 2\pi$

Figure 4.2: The projection after different rotations are applied

From this, we notice that the behavior of the energy function resembles a cosine wave though the valleys are more expansive and the peaks are more narrow.

4.3 Generating starting points

The goal of this section is to introduce a method for generating random points on sparse matrix varieties. We can then use these start points in optimization algorithms for functions on sparse matrix varieties. We then show that the algorithms lead to points that do not correspond to a Daubechies space. We will illustrate the method in the setting of SparseMat(2,3), i.e. we will create matrices of the form

$$\begin{bmatrix} v_1 & v_3 & v_5 \\ v_2 & v_4 & v_6 \end{bmatrix}$$

that satisfy the three constraints on the entries. If we picked our six entries randomly, we will not land on the variety but we can perform a basic algorithm to obtain a point on the variety from this random starting point.

The main idea is to use Gram-Schmidt and normalization. To see this in action, let's examine one dot product:

$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_5 \\ v_6 \end{pmatrix}.$$

We would like this dot product to be zero. We begin by normalizing the first column:

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} := \frac{1}{\sqrt{v_1^2 + v_2^2}} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}.$$

Then, we use Gram Schmidt orthogonalization [32] to modify the last column of the matrix to be orthogonal to the new first column as follows:

$$\begin{pmatrix} v_5 \\ v_6 \end{pmatrix} := \begin{pmatrix} v_5 \\ v_6 \end{pmatrix} - \left(\begin{pmatrix} v_5 & v_6 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \right) \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}.$$

We now satisfy one of the constraints on the variables $v_!, \ldots, v_6$. Our next task is to satisfy the constraint:

$$\begin{pmatrix} v_1 & v_2 \end{pmatrix} \begin{pmatrix} v_3 \\ v_4 \end{pmatrix} + \begin{pmatrix} v_3 & v_4 \end{pmatrix} \begin{pmatrix} v_5 \\ v_6 \end{pmatrix} = 0.$$

We focus our attention on the middle column, the column vector comprised of v_3 and v_4. To do so, we view the above equation as

$$\begin{pmatrix} v_3 & v_4 \end{pmatrix} \begin{pmatrix} v_1 + v_5 \\ v_2 + v_6 \end{pmatrix} = 0.$$

From here, we modify v_3 and v_4 as above to obtain this orthogonality constraint then we normalize the vector.

It is important to note that this procedure is biased in that we will not produce sample points from SparseMat(2,3) in a uniform manner. When using Gram-Schmidt, the projection biases a vector's magnitude to be smaller and thus it will be less likely to be of the same magnitude as the other vectors in our final renormalization. For SparseMat(2,3), we have a method of choosing a point randomly, but it does not generalize.

For this section, I described a method to generate points in SparseMat(2,3) and the approach generalizes to other sparse matrix varieties. In Appendix A.3.3, I include code for producing

Algorithm 6 Generating an element in SparseMat(2,3)

Input: Nothing
Output: A $\in$ **SparseMat(2,3)**

function WAVEGEN23( )
    components = normrnd(0,1,2,3);
    components(:,1) = components(:,1) /sqrt(components(:,1)'*components(:,1));
    components(:,3) = components(:,3) - (components(:,3)'*components(:,1))*components(:,1)
    temp = components(:,1) +components(:,3);
    components(:,2) = components(:,2) - (1/(temp'*temp))*(temp'*components(:,2))*temp;
    length = $\sqrt{tr(\text{components} * \text{components}^T)}$
    components = $\frac{1}{length} \cdot$ components

sample points on SparseMat(2,4) and SparseMat(2,5) following the same strategies as seen in this section.

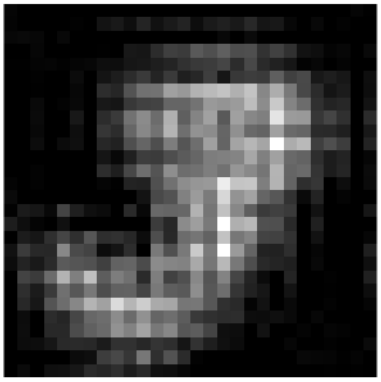


Figure 4.3: Using a generated point in SparseMat(2,3)

4.4 Optimization

Optimization functions on Grassmannians have been utilized in other contexts [33], [34]. Now that we have an energy function, we use methods to travel along our variety as seen in 4.1.2 on page 52. Rather than using derivatives to alter our elements, we will use permutations and rotations

to improve our points. Each rotation corresponds to one direction and in our algorithm, we cycle through the directions multiple times. To illustrate the algorithm, we will go over an example.

Let's look at the case where we have six nonzero entries. Note that we have three directions we can go in, as described before. The optimization problem is below:

$$\max \|\mathbf{CSTP}(A) \cdot Image \cdot \mathbf{CSTP}(A))\|_F$$

subject to

$$A \in \text{SparseMat}(2, j).$$

To examine the effect of the algorithm on an point in SparseMat(2,3), we pick a point $M \in$ SparseMat(2,3), investigate how energy the changes at the various steps, and show its effect on the image. Below is the M we generated to start the process:

$$M = \begin{bmatrix} 0.7019 & -0.3558 & -0.0244 \\ 0.7122 & 0.0319 & 0.0240 \end{bmatrix}.$$

Algorithm 7 Using Rotation Matrices to change the projection Matrix

Input: Image, $M \in$SparseMat(2,j),tolerance
Output: $M \in$SparseMat(2,j)

```
1:  function GRADROTATION(Image,M,tolerance)
2:      NewEnergy = 1
3:      energy = 0
4:      RotateMat = RotateMatrix(delta)
5:      ProjMat = ConverseSparseToProj(M,n)
6:      newEnergy = fb(ProjMat*Im*ProjMat)
7:      newEnergy = energy+.1
8:      while NewEnergy - energy > tolerance do
9:          energy = NewEnergy
10:         energy = ||ProjMat*Im*ProjMat||_F
11:         newEnergy = fb(tempProj*Im*tempProj)
12:         direction = sgn(tempEnergy - energy)
13:         for i = 1:j do
14:             delta = .01
15:             RotateMat = RotateMatrix(delta)
16:             tempM = RotateMat * permute(M,j)
17:             tempM = unpermute(tempM,j)
18:             ProjMat = ConverseSparseToProj(M,n)
19:             tempProj = ConverseSparseToProj(tempM,n)
20:             tempEnergy = ||tempProj*Im*tempProj||_F
21:             direction = sgn(tempEnergy-NewEnergy)
22:             delta = delta*direction
23:             if direction <0 then
24:                 RotateMat = RotateMatrix(delta)
25:                 tempM = RotateMat * permute(M,j)
26:                 tempM = unpermute(tempM,j)
27:                 tempProj = ConverseSparseToProj(tempM,n)
28:                 tempEnergy = ||tempProj*Im*tempProj||_F
29:             while tempEnergy - NewEnergy >tolerance do
30:                 M = tempM
31:                 newEnergy = tempEnergy
32:                 Rotate = Rotate + delta
33:                 RotateMat = RotateMatrix(Rotate)
34:                 tempM = RotateMat * permute(M,j)
35:                 tempM = Unpermute(tempM,j)
36:                 ProjMat = ConverseSparseToProj(M,n)
37:                 tempProj = ConverseSparseToProj(tempM,n)
38:                 tempEnergy = ||tempProj*Im*tempProj||_F
```

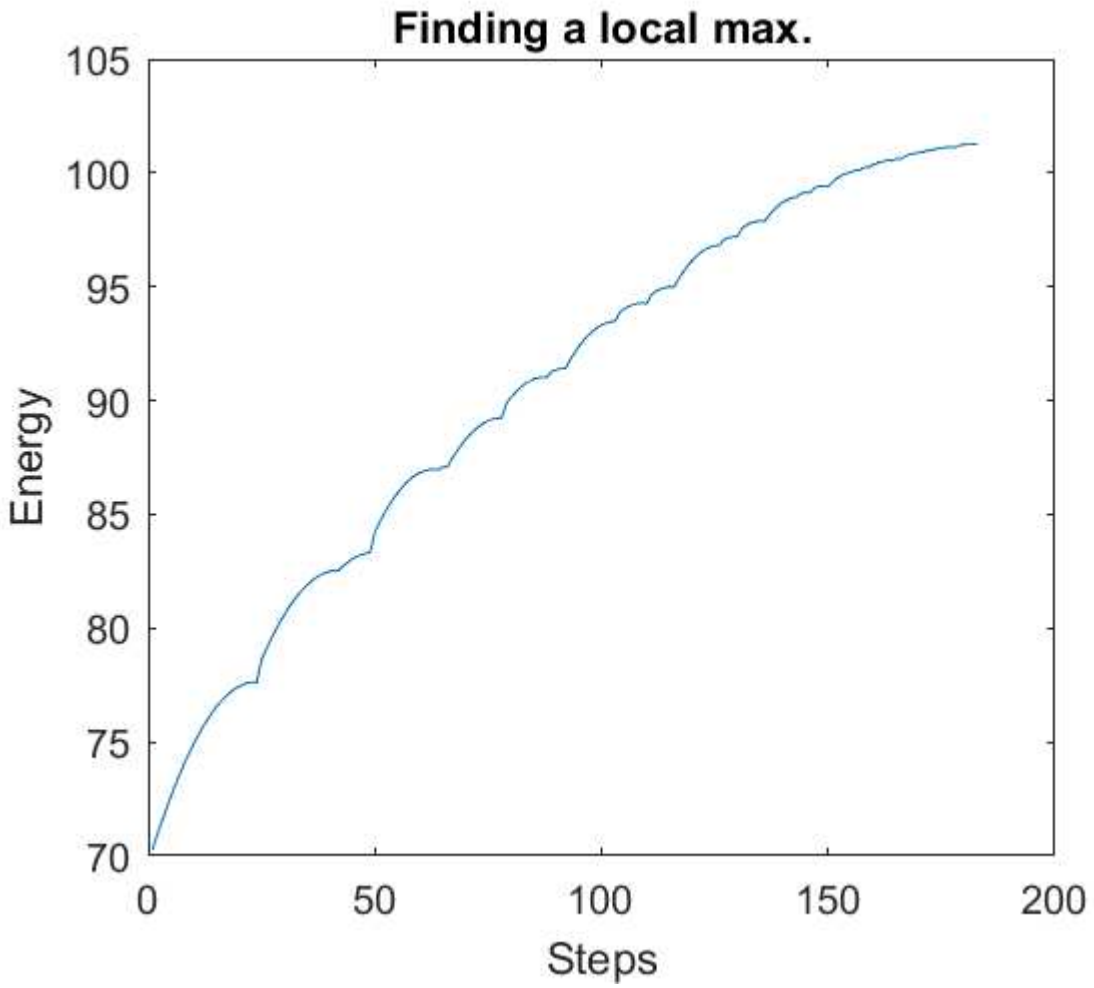
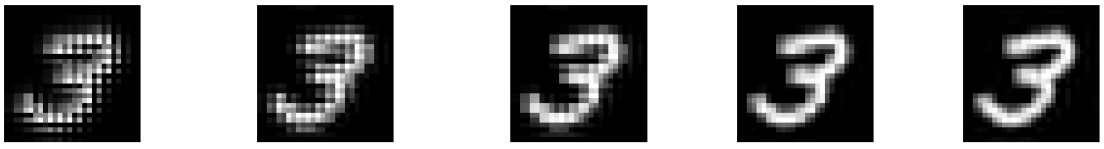


Figure 4.4: Finding a local maximum



(a) Starting point **(b)** 50 steps **(c)** 100 Steps **(d)** 150 Steps **(e)** 183 Steps

Figure 4.5: Moving along this variety using rotation matrices

From here, we notice the increase in energy as the image begins to improve. As we can fill in a SparseMat(2,3) generated point, it works on generated elements and not just Daubechies wavelets.

A feature of the energy graph is that we can see the specific points where the optimization algorithm changes directions as kinks in the plot of the energy function. The final element generated in SparseMat(2,3) in this case was:

$$\begin{bmatrix} 0.4838 & 0.3230 & -0.0909 \\ 0.7924 & -0.1497 & 0.0555 \end{bmatrix}.$$

This is interesting to compare to the point on SparseMat(2,3) that we obtain to the Daubechies point D_6:

$$\begin{bmatrix} 0.3327 & 0.4599 & -0.0854 \\ 0.8069 & -0.1350 & 0.0352 \end{bmatrix}.$$

It is possible that these correspond to the same element in the Grassmannian. To check, we build the matrices corresponding to each of the points on SparseMAt(2,3) and look at the principal angles between the column spaces of the matrices. In this case, the principal angles are plotted as follows:

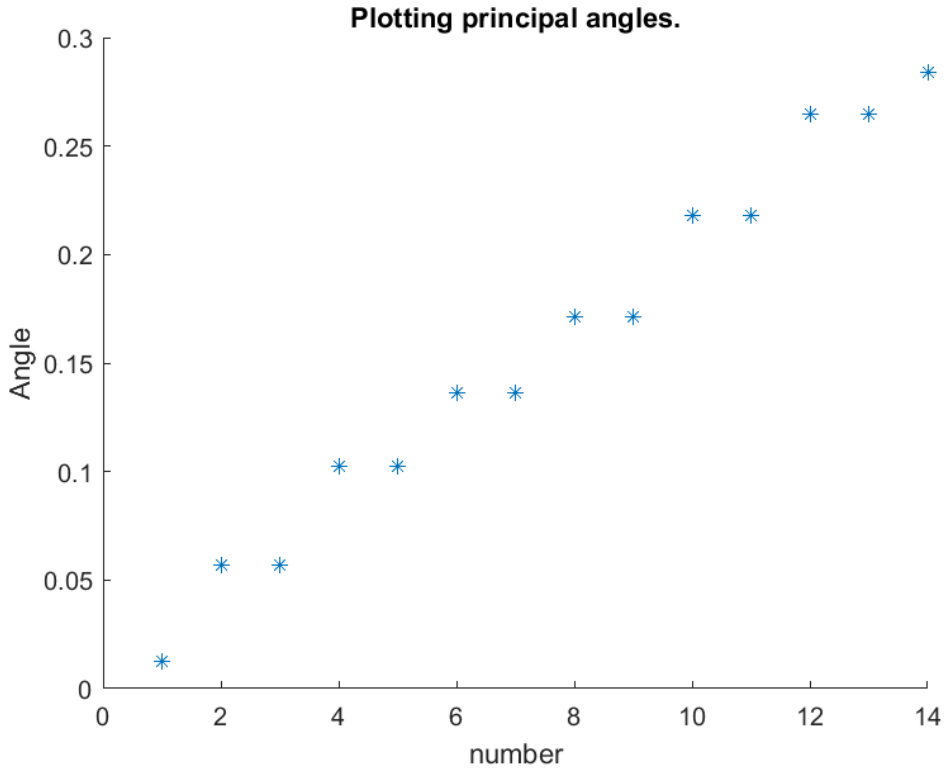


Figure 4.6: Caption

From the plot, we can tell that there are no non-zero vectors in the intersection of the two vector spaces. Note that, similar to what we saw in 3, there is a pairing of principal angles similar to the pairing of eigenvalues. Also, the principal angles created using the all-ones vector and the component found through the optimization algorithm form the same angle as the minimum angle between these two spaces. Another point to note is that when I start the algorithm with the Daubechies point D_6, it did not move.

4.5 Defining and extending this variety

The definition of sparse matrix varieties, and their associated algebraic constraints, is inspired by some of the properties of discrete Daubechies wavelets. From there, we saw many ways to modify points on the variety to new points on the variety. A method to extend this is to build matrices, satisfying similar orthogonality constraints from elements in an $l \times j$ matrix rather than from a $2 \times j$ matrix. Consider the entries in an element of SparseMat(2,j):

$$\begin{bmatrix} v_{11} & v_{12} & \dots & v_{1j} \\ v_{21} & v_{22} & \dots & v_{2j} \end{bmatrix}.$$

The algebraic constraints for points on this variety are as follows. For each $r \in \{1, \dots, n-1\}$

$$\sum_{i=1}^{r} \sum_{m=1}^{2} v_{im} v_{(n+1-r)m} = 0.$$

and

$$\sum_{i=1}^{l} \sum_{j=1}^{2} v_{ij}^2 = 1.$$

In practice, we only need to expand the restriction from 2 to any length l.

Definition 4.2. *Let $j, l \in \mathbb{N}$ where $j, l > 2$*

$$\textit{SparseMat}(l,j) = \left\{ \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1j} \\ v_{21} & v_{22} & \dots & v_{2j} \\ \vdots & \vdots & & \vdots \\ v_{l1} & v_{l2} & \dots & v_{lj} \end{bmatrix} \mid v_{im} \in \mathbb{R} \text{ satisfy C1 and C2} \right\}$$

The conditions for this variety could be described as follows. Condition C1 is that for each $r \in \{1, \dots, j-1\}$

$$\sum_{i=1}^{r} \sum_{m=1}^{l} v_{im} v_{(n+1-i)m} = 0$$

and condition C2 is that

$$\sum_{i=1}^{l} \sum_{m=1}^{l} v_{im}^2 = 1.$$

We used points on SparseMat$(2,j)$ to create points on $\mathbf{Gr}(2,2n)$. In other words, we have a map from SparseMat$(2,j)$ to $\mathbf{Gr}(n,2n)$. In a similar manner we have a map from SparseMat(l,j) to the Grassmannian $\mathbf{Gr}(n,ln)$.

Lemma 4.1. *Let* $l, j, n \in \mathbb{N}$ *such that* $n \geq j$*. Then SparseMat*(l,j) *maps into* $\mathbf{Gr}(n,ln)$.

Proof. Let's go through why we can do this mathematically. To create a function, we need to show that we can map any element in SparseMat(l,j) into $\mathbf{Gr}(n,ln)$. To do this, we will map SparseMat(l,j) into $\mathbf{St}(n,ln)$. To begin, we will associate an element of SparseMat(l,j) with a vector of length ln and then, through circulating the vector as we did in the SparseMat$(2,l)$ case, create a matrix with the property that the transpose of the matrix times the matrix is the identity. First, let's create a vector of length ln from an element of SparseMat(j,l). Our approach begins with "vectorization" [35]. Visually, we want to convert our matrix representation of an element in SparseMat(l,j) into a vector of length ln. For $A \in$ SparseMat(l,j), we are going to create vec(A), a vector of length ln as follows: The first l elements of vec(A) is the first column of A, the second l components of vec(A) is the second column of A,..., the j^{th} l components of vec(A) are the j^{th}

column of A. The rest of the entries are zero. Note for this process to be possible, we would need $ln \geq lj$, which would require $n \geq j$. As a function, we will call this SparseVec.

Visually, we can understand the function as below:

$$\text{SparseVec}\left(\begin{bmatrix} v_{11} & v_{12} & \dots & v_{1j} \\ v_{21} & v_{22} & \dots & v_{2j} \\ \vdots & \vdots & & \vdots \\ v_{l1} & v_{l2} & \dots & v_{lj} \end{bmatrix}, n\right) = \begin{pmatrix} v_{11} \\ v_{21} \\ \vdots \\ v_{l1} \\ v_{12} \\ v_{22} \\ \vdots \\ v_{l2} \\ \vdots \\ v_{lj} \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

From this we can create a shift vector Shift:$\mathbb{R}^{ln} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{ln}$ where $B =$shift(SparseVec(A, n),l, r) is defined as follows:

$$B_i = A_{i+r \cdot l}.$$

Then for $1 \leq r \leq j$ shift(vec(A),$l, r)^T \cdot$ vec(A)) is

$$\sum_{i=1}^{r} \sum_{m=1}^{2} v_{im} v_{(n+1-r)m} = 0.$$

If $r \geq j$, their product is zero because each multiplication will be the sum of the product of two numbers where at least one is zero. Lastly, note that vec($A)^T \cdot$ vec(A)= 1, as this is just the sum of

the squares of each component which is one by definition. Then we can create an element of $S \in \mathbf{St}(n, ln)$ where each column S_r is defined as follows

$$S_r = \text{shift}(\text{SparseVec}(A, n), l, r-1).$$

This is an orthogonal matrix, so it is an element of $\mathbf{St}(n, ln)$ and thus a representative of an element in $\mathbf{Gr}(n, ln)$. □

We extend algorithm 3 on page 59 to the more general form as follows:

Algorithm 8 Convert an element of SparseMat(l,j) to a Column Matrix

Input: $M \in$ SparseMat(l,j), $l \in \mathbb{N}$,$n \in \mathbb{N}$
Output: ColMat $\in$ Mat(2n,n)

```
1: function ColMat(M,n)
2:     l = RowSize(M)
3:     j = ColSize(M)
4:     vec = zeros(l*n,1)
5:     for i =1:j  do
6:         vec(l*(i-1)+1:l*i) = M(:,i)
7:     ColMat=zeros(ln,n)
8:     for i=1:n do
9:         ColMat(:,i) = circVec(vec,l*(i-1))
```

Chapter 5

Conclusion

5.1 Contributions

This dissertation focused on two main topics. The first topic is to develop tools to help visualize features of high dimensional Grassmannians so as to aid in building intuition. The second topic is to identify and exploit regions of a Grassmann manifold that contained subspaces with good projection properties with respect to digital images. Specifically, this thesis makes the following contributions:

- Described a method for visualizing paths in a Grassmannian using projection matrices applied to images.
- Described a family of Schubert varieties based on eigenvectors of matrices corresponding to local variations in an array and showed that these Schubert varieties contained subspaces with good projection properties.
- Defined sparse matrix varieties and Daubechies spaces. Developed algorithms for sampling and optimizing functions on this family of varieties.

5.2 Future Work

This dissertation presents several avenues for future research. A strategy that yielded results in this thesis was to create tools for visualizations of a process to aid intuition and formulate conjectures. Elements of SparseMat(2,j) can be represented by an ordered collection of j vectors of length two. This collection can be visualized as a collection of j labeled points in $\mathbb{R}^2$. While proceeding through the steps in optimizing a function over SparseMat(2,j), we follow a path on the variety. This will lead to a family of j labeled paths in $\mathbb{R}^2$. It would be interesting to better understand the features of these paths under different optimization conditions. We can extend this

approach to the sparse matrix family SparseMat(3,j) to produce a family of j labeled paths in $\mathbb{R}^3$ for each run of the optimization procedure. Further questions include the following. Within the large family SparseMat(l, j), can we find interesting wavelets for various purposes? Can we adapt the strategy to understand local degrees of freedom as developed for SparseMat(2,j) to the more general setting of SparseMat(l, j)? Is there a better formulation or additional visualization tools that can be developed in order to make progress towards these generalizations? One possibility is to consider stereographic projections with the vectors in SparseMat(3,j). This leads to the question of whether there are different types of visualization, for higher dimensional n spheres, that can be used in conjunction with the tools developed in this dissertation. In order to answer these questions, we expect the presence of multiple, potentially difficult, hurdles to overcome. However, we feel that the potential payoffs make this a compelling area for further research.

Throughout this thesis, we have used Grassmann manifolds as a starting point for our results. Recall that the Grassmann manifold $\mathbf{Gr}(k, n)$ parameterizes k-dimensional subspaces of $\mathbb{R}^n$, i.e. pairs $V \subset \mathbb{R}^n$ with $dim(V) = k$. A flag of vector spaces in $\mathbb{R}^n$ is a nested sequence of vector spaces $V_1 \subset V_2 \cdots \subset V_k \subset \mathbb{R}^n$. If we fix the dimensions of $V_1, \ldots, V_k$ to be $d_1, d_2, \ldots, d_k$ then the parameter space of all length k flags of vector spaces in $\mathbb{R}^n$ with the same dimensions is an example of a flag manifold. The signature of the flag manifold is the data $(d_1, d_2, \ldots, d_k, n)$. From a flag, i.e. a point on a flag manifold, we can create multiple kinds of projection matrices. Furthermore, strategies to make the objects in Chap 4 could be employed within Flag manifolds to see the effects in a different context.

Bibliography

[1] Hermann Grassmann. *Die lineale Ausdehnungslehre ein neuer Zweig der Mathematik: dargestellt und durch Anwendungen auf die übrigen Zweige der Mathematik, wie auch auf die Statik, Mechanik, die Lehre vom Magnetismus und die Krystallonomie erläutert*, volume 1. O. Wigand, 1844.

[2] Andrew V Knyazev and Merico E Argentati. Principal angles between subspaces in an A-based scalar product: algorithms and perturbation estimates. *SIAM Journal on Scientific Computing*, 23(6):2008–2040, 2002.

[3] Steven J Leon, Åke Björck, and Walter Gander. Gram-Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20(3):492–532, 2013.

[4] Alfred Haar. *Zur theorie der orthogonalen funktionensysteme*. Georg-August-Universitat, Gottingen., 1909.

[5] Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

[6] David Speyer and Bernd Sturmfels. The tropical Grassmannian. *Advances in Geometry*, 4(3):389–411, 2004.

[7] Pavan Turaga, Ashok Veeraraghavan, Anuj Srivastava, and Rama Chellappa. Statistical computations on Grassmann and Stiefel manifolds for image and video-based recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2273–2286, 2011.

[8] Soren Hauberg, Aasa Feragen, and Michael J Black. Grassmann averages for scalable robust PCA. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3810–3817, 2014.

[9] Kyle A Gallivan, Anuj Srivastava, Xiuwen Liu, and Paul Van Dooren. Efficient algorithms for inferences on Grassmann manifolds. In *IEEE Workshop on Statistical Signal Processing, 2003*, pages 315–318. IEEE, 2003.

[10] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. Riemannian geometry of Grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematica*, 80:199–220, 2004.

[11] Alan Edelman, Tomás A Arias, and Steven T Smith. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[12] Liviu I Nicolaescu. *Lectures on the Geometry of Manifolds*. World Scientific, 2020.

[13] Ke Ye, Ken Sze-Wai Wong, and Lek-Heng Lim. Optimization on flag manifolds. *Mathematical Programming*, pages 1–40, 2021.

[14] Christopher C Paige and Musheng Wei. History and generality of the CS decomposition. *Linear Algebra and its Applications*, 208:303–326, 1994.

[15] Clay Shonkwiler. Principal angles in terms of inner products. *Unpublished note, http://www. math. colostate. edu/˜ clayton/research/notes/PrincipalAngles. pdf*, 2009.

[16] Sun Ji-Guang. Perturbation of angles between linear subspaces. *Journal of Computational Mathematics*, pages 58–61, 1987.

[17] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[18] Roger A Horn and Charles R Johnson. *Topics in matrix analysis*. Cambridge university press, 1994.

[19] Ashish Khare and Uma Shanker Tiwary. Daubechies complex wavelet transform based technique for denoising of medical images. *International Journal of Image and Graphics*, 7(04):663–687, 2007.

[20] Maria Gillespie. Variations on a theme of Schubert calculus. *Recent Trends in Algebraic Combinatorics*, pages 115–158, 2019.

[21] Izzet Coskun. Rigidity of Schubert classes in orthogonal Grassmannians. *Israel Journal of Mathematics*, 200, 06 2014.

[22] Brani Vidakovic. Basics of wavelets. *https://www2.isye.gatech.edu/isyebayes/bank/handout20.pdf*, 2004.

[23] James Simons. Minimal varieties in Riemannian manifolds. *Annals of Mathematics*, pages 62–105, 1968.

[24] nLab authors. totally geodesic submanifold. https://ncatlab.org/nlab/show/totally+geodesic+submanifold, April 2024.

[25] Dayong Wang, Yu Sun, Ce Zhu, Weisheng Li, Frederic Dufaux, and Jiangtao Luo. Fast depth and mode decision in intra prediction for quality SHVC. *IEEE Transactions on Image Processing*, 29:6136–6150, 2020.

[26] Grzegorz Bieszczad, Tomasz Sosnowski, and Henryk Madura. Improved sum-of-squared-differences tracking algorithm for thermal vision systems. In *International Symposium on Photoelectronic Detection and Imaging 2011: Advances in Infrared Imaging and Applications*, volume 8193, pages 746–749. SPIE, 2011.

[27] Javanshir Khosravi, Mohammad Shams Esfand Abadi, and Reza Ebrahimpour. Image registration based on sum of square difference cost function. *Journal of Electrical and Computer Engineering Innovations (JECEI)*, 6(2):273–281, 2018.

[28] John Von Neumann. *Some matrix-inequalities and metrization of matric space*. Pergamon Press, 1962.

[29] Matthew Sundling, Nagamani Sukumar, Hongmei Zhang, Mark J Embrechts, and Curt M Breneman. Wavelets in chemistry and cheminformatics. *Reviews in Computational Chemistry*, 22:295–329, 2006.

[30] Margarete Oliveira Domingues, Odim Mendes Jr, and Aracy Mendes da Costa. On wavelet techniques in atmospheric sciences. *Advances in Space Research*, 35(5):831–842, 2005.

[31] Yiwen Shan, Dong Hu, Zhi Wang, and Tao Jia. Multi-channel nuclear norm minus Frobenius norm minimization for color image denoising. *Signal Processing*, 207:108959, 2023.

[32] Åke Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra and Its Applications*, 197:297–316, 1994.

[33] Dejiao Zhang and Laura Balzano. Global convergence of a Grassmannian gradient descent algorithm for subspace estimation. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1460–1468, Cadiz, Spain, 09–11 May 2016. PMLR.

[34] Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Global optimality of local search for low rank matrix recovery. *Advances in Neural Information Processing Systems*, 29, 2016.

[35] Phoebus J Dhrymes. Matrix vectorization. *Mathematics for Econometrics*, pages 117–145, 2000.

Appendix A

Code Used

This appendix contains user-friendly Matlab code designed to test the examples created in this thesis. To use the code, copy and paste it into a Matlab file and save it as the function's name. You can then start experimenting with the code yourself. I have also included a Test Function to test these functions, so at the end, you can quickly write a Test Function with the parameters you want to examine the effect of certain conditions. This straightforward process empowers you to explore and understand the code quickly.

A.1 Grassmannian Algorithms

```
function [A,G1Bar,G2Bar] = GrassShortPath(G1,G2,grassType)
    n = sum(grassType);
    G1Bar = Thicken(G1,grassType);
    G2Bar = Thicken(G2,grassType);
    G1Slice1 = G1Bar(:,1:grassType(1));
    G2Slice1 = G2Bar(:,1:grassType(1));
    [U,S,V] = svd(G1Slice1'*G2Slice1);
    G1Slice1 =G1Slice1*U;
    G2Slice1 =G2Slice1*V;
    G1Slice2 = G1Bar(:,grassType(1)+1:n);
    G2Slice2 = G2Bar(:,grassType(1)+1:n);
    [U,S,V] = svd(G1Slice2'*G2Slice2);
    G1Slice2 =G1Slice2*U;
    G2Slice2 =G2Slice2*V;
    G1Bar = [G1Slice1 G1Slice2];
    G2Bar = [G2Slice1 G2Slice2];
```

```
    A = logm(G1Bar'*G2Bar);
end

function [rowTangent] = Daubechies(imagePath,D1,D2,path,...
        resizeFactor )
    I= imread(imagePath);
    [row, column, depth] = size(I);
    rowMatD1 = makeD2k(D1,row/2);
    rowMatD2 = makeD2k(D2,row/2);
    columnMatD1 = makeD2k(D1,column/2);
    columnMatD2 = makeD2k(D2,column/2);

    [rowTangent, rowStart] = geodesicMat(rowMatD1,...
        rowMatD2,[row/2,row/2]);
    [columnTangent, columnStart] = geodesicMat(columnMatD1,...
        columnMatD2,[column/2,column/2]);
    frameMaker(rowStart,columnStart, rowTangent, columnTangent,...
        row,column,depth, imagePath, path,resizeFactor);
end

function [frameNum] = frameMaker(rowStartingPoint,...
    columnStartingPoint, rowTangentVector, columnTangentVector...
    ,row,column,depth, imagePath, path,resizeFactor,frameNum )
    % Need to use depth to make the file work for arrays.
    I= imread(imagePath);
    Im = im2double(I);
    checkStart = exist('frameNum','var');
    if (checkStart~= 1)
        frameNum = 1;
```

```
end
% tempImage is where we store the image pre scaling
tempImage = zeros(row,column,depth);
tempKron = zeros(row*resizeFactor,column*resizeFactor,depth);
for i = 1:length(path)
    rowMat              = rowStartingPoint*...
        expm(rowTangentVector*path(i));
    rowMat              = rowMat(:,(1:row/2));
    columnMat           = columnStartingPoint*...
        expm(columnTangentVector*path(i));
    columnMat           = columnMat(:,(1:column/2));
    rowProjection       = rowMat*rowMat';
    columnProjection    = columnMat*columnMat';
    % The depth part should be here so that
    % we can apply it to each layer of the picture.
    for j = 1:depth
        tempImage(:,:,j) = rowProjection*...
            Im(:,:,j)*columnProjection;
    end
    % This should rescale the picture.
    for j = 1:depth
        tempKron(:,:,j) = ...
            kron( tempImage(:,:,j),ones(resizeFactor));
    end
    frameMade = cat(depth,tempKron);
    frameName = makeFrameName(frameNum);
    saveas(imshow(frameMade),frameName);
```

```
        frameNum = frameNum +1;
    end
end
```

A.2 Schubert Variety Algorithms

A.2.1 Local Variance Matrix

```
function [locVar] = localVariance(k,n)
    if ( (2*k+1) >= n)
        locVar = eye(n) - (ones(n)/n);
    else
        tempVec = zeros(1,n);
        locVar = zeros(n);
        tempVec(1) = 1;
        for i =1:k
            tempVec(1+i) = 1;
            tempVec(n-i+1) = 1;
        end
        for i =1:n
            locVar(i,:) = circshift(tempVec,i-1);
        end
        locVar = locVar *(1/(2*k+1));
        locVar = eye(n) - locVar;
    end
end
```

A.2.2 Local Variance with Boundary

```
function [locVar] = localVarianceWithBoundary(k,n)
```

```
if ( (2*k+1) >= n)
    locVar = n*eye(n) - ones(n);
else
    locVar = zeros(n);
    % Creating the boundary cases.
    for i = 1:k
        locVar(i,i) =  k + i - 1;
        locVar(n-i+1,n-i+1) = k+i-1;
        for j = 1:i-1
            locVar(i,j) = -1;
            locVar(n-i+1,n-j+1)=-1;
        end
        for j = 1:k
            locVar(i,i+j) = -1;
            locVar(n-i+1,n-i+1-j)=-1;
        end
    end
    % Create the rest of the matrix.
    % Begin with the vector that we will create.
    tempVec = zeros(1,n);
    for i=1:k
        tempVec(i) = -1;
        tempVec(i+k+1) = -1;
    end
    tempVec(k+1) = 2*k;

    for i = k+1:n-k
```

```
            locVar(i,:) = circshift(tempVec,i-k-1);
        end
    end
end
```

A.3 Sparse Matrix Algorithms

A.3.1 Rotating D_6 by 2π

```
%% This script is to get a gradient condition going.
%  It will be split into two parts. Getting the direction
%  to go in and then the stop condition.

% Initial condition
k = 3;
d = dbaux(k,1);
n=28;

components = zeros(2,k);
for i=1:k
    components(:,i)= d(2*(i-1)+1:2*i);
end
components = ...
        (1/sqrt(trace(components'*components)))*components;
imagePath="MNISTExample.png";
I= imread(imagePath);
Im = im2double(I);
components;
thetaNum=1;
```

```
delta = .01;

% Here is where we figure out the direction we move in
circVec = zeros(n,1);
circVec(1:2*k) = components;
origCircMat = circulantMatrixGen(circVec,n);
origProjMat = origCircMat*origCircMat';
origProjection = origProjMat*Im*origProjMat;
origEnergy = trace(origProjection'*origProjection);
rotationMat = [cos(delta) -sin(delta); sin(delta) cos(delta)];
newDaub = rotationMat*components;
circVec = zeros(n,1);
circVec(1:2*k) = newDaub;
moveCircMat = circulantMatrixGen(circVec,n);
moveProjMat = moveCircMat*moveCircMat';
moveProjection = moveProjMat*Im*moveProjMat;
moveEnergy = trace(moveProjection'*moveProjection);
direction =sign(moveEnergy - origEnergy);
delta = delta*direction;
theta = delta;
path = 0:2*pi/100:2*pi;
for theta= 0:100
    rotationMat = [cos(theta*2*pi/100) -sin(theta*2*pi/100);...
        sin(theta*2*pi/100) cos(theta*2*pi/100)];
    newDaub = rotationMat*components;
    circVec = zeros(n,1);
    circVec(1:2*k) = newDaub;
```

```
    genMat = zeros(n,n/2);
    size(genMat)
    for i = 1:n/2
        genMat(:,i) = circshift(circVec,2*i-2);
    end
    ProjMat = genMat*genMat';
    tempImage = ProjMat*Im*ProjMat;
    points(thetaNum) = trace(tempImage'*tempImage);
    tempImage = kron(tempImage,ones(13));
    thetaName = makeThetaName(thetaNum);
    saveas(imshow(tempImage),thetaName);
    thetaNum = thetaNum +1;
end

plot(path, points)
title('Energy as we rotate through the space.')
ylabel('Energy')
xlabel('0 < \theta < 2\pi')
```

A.3.2 SparseMat(2,3) algorithm

```
%% This script is to get a gradiant condition going.
%% It will be split into two parts. Getting the direction
%% to go in and then the stop condition.

k = 2;
components = zeros(2,k);
```

```
d = OrthogonalCircVectorGen(2,k,n);
for i=1:k
    components(:,i)= d(2*(i-1)+1:2*i);
end
imagePath="MNISTExample.png";
I= imread(imagePath);
Im = im2double(I);
components;
thetaNum=1;
delta = .01;

% Here is where we figure out the direction we move in
circVec = zeros(n,1);
circVec(1:2*k) = components;
origCircMat = circulantMatrixGen(circVec,n);
origProjMat = origCircMat*origCircMat';
origProjection = origProjMat*Im*origProjMat;
origEnergy = trace(origProjection'*origProjection);
rotationMat = [cos(delta) -sin(delta); sin(delta) cos(delta)];
newDaub = rotationMat*components;
circVec = zeros(n,1);
circVec(1:2*k) = newDaub;
moveCircMat = circulantMatrixGen(circVec,n);
moveProjMat = moveCircMat*moveCircMat';
moveProjection = moveProjMat*Im*moveProjMat;

moveEnergy = trace(moveProjection'*moveProjection);
```

```
direction =sign(moveEnergy - origEnergy);

delta = delta*direction;

theta = delta;

moveEnergy = direction*moveEnergy;
direction = 1
while direction > 0
    origEnergy = moveEnergy;
    theta = delta + theta;
    rotationMat = [cos(theta) -sin(theta);...
        sin(theta) cos(theta)];
    newDaub = rotationMat*components;
    circVec = zeros(n,1);
    circVec(1:2*k) = newDaub;
    moveCircMat = circulantMatrixGen(circVec,n);
    moveProjMat = moveCircMat*moveCircMat';
    moveImage = moveProjMat*Im*moveProjMat;
    moveEnergy = trace(moveImage'*moveImage);
    moveImage = kron(moveImage,ones(13));
    direction = moveEnergy - origEnergy;
    thetaName = makeThetaName(thetaNum);
    saveas(imshow(moveImage),thetaName);
    thetaNum = thetaNum +1;
end
```

```
function[genMat] = circulantMatrixGen(circVec,n)
genMat = zeros(n,n/2);
for i = 1:n/2
    genMat(:,i) = circshift(circVec,2*i-2);
end

function[frameName]= makeThetaName(thetaNumber)
if( thetaNumber < 10 )
    frameName =strcat('theta00',int2str(thetaNumber),'.png');
elseif( thetaNumber < 100)
    frameName =strcat('theta0',int2str(thetaNumber),'.png');
else
    frameName =strcat('theta',int2str(thetaNumber),'.png');
end
```

A.3.3 Element in SparseMat Generator

```
%% Script to take a vector and make it piecewise orthogonal
%% along d parts to itself
% Strip the vector into d components of length k and make each
% part that is recieved through circulation orthogonal. Do so
% in a way that only changes one component at a time.

% Test case: My first case is going to try do this with either 2
% components or three. The hope is that by seeing how to do this
% for small scale the method of extending this will stretch.
function [components] = SparseMatGen(l,j)
```

```
components = normrnd(0,1,1,j);
components = (1/sqrt(trace(components*components')))*...
components;
if j == 2
    components(:,1) = components(:,1) /...
        sqrt(components(:,1)'*components(:,1) );
    components(:,2)
    components(:,2) = components(:,2) -...
        (components(:,2)'*components(:,1))*components(:,1);
end
if j ==3
    components(:,1) = components(:,1) /...
        sqrt(components(:,1)'*components(:,1) );
    components(:,3) = components(:,3) -...
        (components(:,3)'*components(:,1))*components(:,1);
    tempVec = components(:,1) +components(:,3);
    components(:,2) = components(:,2) -...
        (1/(tempVec'*tempVec))*...
        (tempVec'*components(:,2))*tempVec;
end
if j==4
    components(:,1) = components(:,1) /...
        sqrt(components(:,1)'*components(:,1) );
    components(:,4) = components(:,4) -...
        (components(:,4)'*components(:,1))*components(:,1);
    tempProd1 = components(:,3)'*components(:,1);
    tempProd2 = components(:,4)'*components(:,2);
```

```
        alpha = -tempProd1/tempProd2;
        components(:,2) = alpha*components(:,2);
        tempProd1 = components(:,2)'*components(:,1)...
            +components(:,3)'*components(:,2) + ...
            components(:,3)'*components(:,4);
        tempVec = zeros(2,1);
        tempVec_2 = components(:,4);
        tempVec(1) = -tempVec_2(2);
        tempVec(2) = tempVec_2(1);
        tempProd2 = tempVec'*(components(:,1)+components(:,3));
        alpha = -tempProd1/tempProd2;
        components(:,2) = components(:,2) + alpha*tempVec;
    end
    components =...
    (1/sqrt(trace(components*components')))*components;
end
```