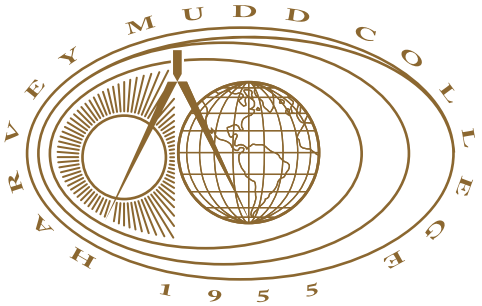
2003

Generalized Spectral Analysis for Large Sets of Approval Voting Data

David Uminsky
Harvey Mudd College



Generalized Spectral Analysis for Large Sets of Approval Voting Data

by

David Thomas Uminsky

Michael Orrison, Advisor

Advisor: _____

Second Reader: _____

(Francis Su)

May 2003

Department of Mathematics

HARVEY MUDD COLLEGE

Abstract

Generalized Spectral Analysis for Large Sets of Approval Voting Data

by David Thomas Uminsky

May 2003

Generalized Spectral analysis of approval voting data uses representation theory and the symmetry of the data to project the approval voting data into orthogonal and interpretable subspaces. Unfortunately, as the number of voters grows, the data space becomes prohibitively large to compute the decomposition of the data vector. To attack these large data sets we develop a method to partition the data set into equivalence classes, in order to drastically reduce the size of the space while retaining the necessary characteristics of the data set. We also make progress on the needed statistical tools to explain the results of the spectral analysis. The standard spectral analysis will be demonstrated, and our partitioning technique is applied to U.S. Senate roll call data.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Professor Henry Krieger for his patience and help in the statistical analysis. I would also like to thank Professor Brian Lawson for the discussions of this work and providing so much data in which to test these methods. I would like to thank Nate Eldredge for staying up late writing his own thesis and providing Latex help at 4:00 am. Lastly, I would like to thank Professor Mike Orrison for advising this work. His guidance has been invaluable and, above all else, he has taught me how to be a better mathematician.

Chapter 1

Introduction

1.1 The Problem

The objective of this paper is to present a new method of analyzing large sets of approval voting data. Approval voting is a familiar process to anyone who has observed the federal government during legislation. For example, members of the U.S. Senate vote "yea" or "nay" on any given bill on the floor. While the question of what passes or does not pass is usually decided by whether a majority is achieved or not, an interesting question is whether blocks or coalitions of voters form during the process. Being able to completely describe coalition formation can be helpful in understanding how legislators act in a governing body. For example, in [9] we are able to use our analysis of coalition formation to describe the voting behavior over the last fifty years of the U.S. Supreme Court.

The standard techniques for analyzing approval voting data, including the spatial model presented in Chapter 1, have difficulty describing coalition effects. In this paper we will describe the method of generalized spectral analysis, developed by Persi Diaconis as a way to analyze coalition behavior of our data. We will present a new partitioning technique as a tool to apply spectral analysis to large sets of data whose analysis would not be computationally feasible otherwise. In addition, we will present some statistical techniques to provide a way to formally report the results of the analysis. Finally, we will provide a program and manual in the appendix that fully implements these techniques using efficient methods

developed in [10], and [11].

1.2 Approval Voting

Approval voting or roll-call voting is a "yea" or "nay" voting system used in many academic and political institutions to decide on issues and policy. For example, the U.S. Senate, U.S. Supreme Court, and the HMC Math Department are all examples of institutions that use approval voting. Many questions arise from this data such as: How do the voters behave? Do they form coalitions? Can coalitions be determined from the data?

For the sake of mathematical analysis, roll-call data can be recorded into a large matrix with columns corresponding to the voting members and each row corresponding to a roll-call vote that was decided by the governing body. We encode each roll-call as follows: For each voter we record a 0 if the decision places the voter in the minority and 1 if the voter agrees with the majority.

Example 1.2.1 *A voting body of 4 members and 10 bills might have a data matrix like the following:*

$$D = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

For a data matrix like the one in example 1.2.1, each vote (i.e. row) can also be thought of as a vote on a subset of the voting body where the subset is the set of members who voted in the minority. Thus our data can now be thought of as a function f on the elements of the power set of the voting body where f simply tallies up how many times a given subset formed a minority in a given data set. It is clear from this definition that the domain of our function is actually the set of all subsets of our governing body of size $\lfloor n/2 \rfloor$ and smaller.

Example 1.2.2 *The data matrix from example 1.2.1 can be written as a tally function on the minority subsets. In this case our function looks like:*

$$f(\{1\}) = 1, \quad f(\{3\}) = 1, \quad f(\{4\}) = 3,$$
$$f(\{2,4\}) = 1, \quad f(\{1,4\}) = 1, \quad f(\{2,3\}) = 3,$$

and f is 0 on all other subsets.

It is in this context that we can now describe the space in which our data resides. We will continue this discussion in Chapters 2. First, a brief survey of the standard spatial model is given.

1.3 The Spatial Model

A very popular method of analyzing approval voting or roll-call data is the spatial model approach. This approach, first presented in [13] by Keith Poole and Howard Rosenthal, has become a popular way in which political scientists analyze this kind of data. The Poole-Rosenthal model assumes "probabilistic voting based on a spatial utility function" [13]. The parameters and the spatial coordinates of the utility functions are then estimated by the data. NOMINATE was written by Keith Poole and Howard Rosenthal to perform the estimation of these utility functions. The

FORTRAN source code of an updated version of NOMINATE, D-NOMINATE, can be found on the web at [12].

The results of such analysis on the U.S. Senate and Congress suggests that the data can be described in having a one-dimensional, " liberal/conservative" line. Results of applying the analysis to the U.S. senate places all the senators on a "liberal/conservative" line. Your position on this line dictates how you vote to a degree of accuracy.

There are a few limitations of the spatial model that leave room for more research. In general, the placing of voters on a line does describe the behavior of the voters but actually does not give precise coalition behavior. Being placed very close to another senator usually means the two senators have similar voting behavior. Unfortunately the low dimensional analysis leaves little room for explaining exactly which coalitions form and which do not. In addition, no precise gradiancy of coalition behavior can be derived.

Another limitation of this method is that certain bills are not useable in estimating the positions and must be removed from the original data set. It is precisely the bills that have too "small" a minority that are removed since they seem to negatively effect the estimates computed by NOMINATE. Finally, it has been pointed out by some that the standard errors on the estimates can sometimes be quite large when estimating the positions of the voters, suggesting that more study is needed.

To conclude, the NOMINATE program and the spatial model describes very well how individual voters behave, but the low dimensional analysis of the data leaves much to be desired in giving a complete description of coalition behavior. For this description, we turn to generalized spectral analysis.

Chapter 2

Generalized Spectral Analysis

Here we present the necessary definitions and theorems in representation theory to understand spectral analysis. The more experienced reader will feel comfortable skipping the first section of this chapter. In Section 2.1, we define a representation of a group. We also present some useful definitions and theorems in module theory. In Section 2.2, we draw an explicit connection between representation theory and approval voting data and explain how the resulting homogeneous space decomposes into irreducible, orthogonal and interpretable subspaces. In Section 2.3, we apply the theory described in Section 2.1 and Section 2.2 to a small example. In Section 2.4, we present bootstrapping as a statistical tool to analyze the results of spectral analysis. Lastly, in Section 2.5, we summarize our discussion of generalized spectral analysis and how it differs from the spatial model method mentioned in Chapter 1.

2.1 Representation of a Group, Modules

To understand generalized spectral analysis, we will need to build up some important theory from algebra. Two excellent books on the subject are [2] and [5].

Definition 2.1.1 *Let R be a ring. A left R-***module** *is a set M together with*

1. *a binary operation $+$ on M under which M is an abelian group, and*

2. *an action of R on M (that is, a map $R \times M \to M$) such that for all $r, s \in R$, $m, n \in M$,*

- $(r+s)m = rm + sm$,
- $(rs)m = r(sm)$, *and*
- $r(m+n) = rm + rn$.

  If the ring R has a multiplicative identity, 1, we impose the additional axiom:
- $1m = m, \forall m \in M$.

All our modules will be left modules, so we will drop the word left in our discussion of modules. Also note that modules over a field F are equivalent to vector spaces over F. It is in these algebraic structures where we will perform spectral analysis.

Definition 2.1.2 *Let R be a commutative ring with identity. Let G be any finite group with the group operation written as multiplication. Then the* **group ring**, *RG, is the set of all formal sums*

$$a_1g_1 + a_2g_2 + \cdots + a_ng_n \qquad\qquad a_i \in R, g_i \in G.$$

where addition is component wise and $(ag_i)(bg_j) = (ab)g_ig_j$.

Definition 2.1.3 *Let F be a field. A* **representation** *of a group G is an $FG-module$.*

It is these representations or modules that primarily concerns our study.

Definition 2.1.4 *A* **submodule** *N of a FG-module is a subspace of an FG-module which is invariant under the action of FG, i.e., $rn \in N$, for all $r \in FG$, $n \in N$.*

Definition 2.1.5 *A module M is called* **irreducible** *if its only submodules are 0 or M.*

Theorem 2.1.1 *Let F be a field with characteristic 0, then every FG-module is a direct sum of irreducible submodules.*

Theorem 2.1.1 implies that the study of any given FG-module may be reduced to studying its FG-stable irreducible decomposition. Applying generalized spectral analysis involves studying these decompositions.

2.2 Homogeneous Spaces, Interpretable Subspaces

In this section we connect the above representation theory to our problem of spectral analysis of approval voting data. We first define what a homogeneous space is and discuss why it is important.

Definition 2.2.1 *Let G be a finite group and X be a finite set. Define an equivalence on X by $x \sim y$ if for some $s \in G, sx = y$. The equivalence classes are called orbits. G operates transitively on X if there is only one orbit. A set with a group acting transitively on it is called a* **homogeneous space**.

This might be a good time to ask: How does this theory relate to the analysis of approval voting data? The answer is that we can view the approval voting data as residing in a $\mathbb{C}S_n$-module. To see this and how homogeneous spaces relate, we now construct our space.

As we demonstrated in the previous chapter, we can view approval voting data as a function, f, on a subset of the power set of the voting members. In fact, in the case where we only care about who is in the minority, the domain of f is all subcollections of voters up to size $\lfloor n/2 \rfloor$, where n is the total number of voters. To reiterate, if we label or voters 1 to n, then X is exactly the elements of the power set of X with up to $\lfloor n/2 \rfloor$ voters in the set.

Now let us consider the vector space of all functions, M, from X into $\mathbb{C}$. We index the basis elements of this vector space as follows: first we list the singleton coalitions in order $\{1\}$ to $\{n\}$, then followed by all doubleton coalitions in the order {1,2},{1,3},...,{1,n},{2,3},...,$\{n-1, n\}$ and similarly for all coalitions of up to size $\lfloor n/2 \rfloor$.

Example 2.2.1 *For an example of 4 voters on ten bills we recall the data matrix from Example 1.2.1. Then as shown above, our data function, f, can be written as :*

$$f(\{1\}) = 1, \quad f(\{3\}) = 1, \quad f(\{4\}) = 3,$$
$$f(\{2,4\}) = 1, \quad f(\{1,4\}) = 1, \quad f(\{2,3\}) = 3,$$

and f is 0 on all other subsets. We now rewrite the data in vector form with the ordered basis described above:

$$f = \begin{pmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 3 \\ 3 & 4 \\ 0 & 12 \\ 0 & 13 \\ 1 & 14 \\ 3 & 23 \\ 1 & 24 \\ 0 & 34 \end{pmatrix}.$$

The column of numbers on the right of the data vector represents the elements of the basis to which value corresponds.

We transform M into a $\mathbb{C}S_n$-module by defining the action of the group ring, $\mathbb{C}S_n$ on the basis elements of M in a natural way. That is, let x_i be a basis element of M. Then an element of the group ring acts on x_i as follows:

$$(a_1 g_1 + a_2 g_2 + \cdots + a_k g_k) x_i = a_1(g_1 x_i) + a_2(g_2 x_i) + \cdots + a_k(g_k x_i)$$

where $g_j x_i$ is just the action of the permutation on the subset x_i.

Example 2.2.2 *To see the action of the group ring on our data vectors, we consider the ring element $2(12) + (34)$ acting on the data vector f from Example 2.2.1.*

$$(2(12)+(34))f = 2(12)\begin{pmatrix}2\\0\\1\\0\\1\\0\\0\\0\\1\\0\end{pmatrix} +(34)\begin{pmatrix}2\\0\\1\\0\\1\\0\\0\\0\\1\\0\end{pmatrix}$$

$$= \begin{pmatrix}0\\4\\2\\0\\2\\0\\2\\0\\0\\0\end{pmatrix} + \begin{pmatrix}2\\0\\0\\1\\1\\0\\0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}2\\4\\2\\1\\3\\0\\2\\1\\0\\0\end{pmatrix}.$$

To reiterate, our "tally" data vectors sits inside M, which is a $\mathbb{C}S_n$-module. Drawing upon definition 2.2.1, we notice that the basis elements (of subsets) of M decompose into a collection of homogeneous spaces with respect to S_n. This decomposition is simply grouping up all the subsets of size k for $k \leq \lfloor n/2 \rfloor$. This is clear, since the action of permutation defined on a set does not change the size of the set. We now extend the definition of a homogeneous space to the function

space *on* the corresponding homogeneous space. Thus for a data vector with voters $\{1...n\}$, M decomposes as

$$M = M^{(n,0)} \oplus M^{(n-1,1)} \oplus \cdots \oplus M^{(n-k,k)},$$

where each $M^{(n-i,i)}$ is an orthogonal, homogeneous space.

For example, in the case of the U.S. Supreme Court we have $n = 9$ justices. In the cases where the Supreme Court unanimously approved, the corresponding one dimensional space is $M^{(9,0)}$. For the cases where a single justice dissented, the corresponding $\binom{9}{1}$ dimensional space is denoted $M^{(8,1)}$. In general, when there is k justices dissenting, the corresponding space is denoted $M^{(9-k,k)}$ and has dimension $\binom{9}{k}$.

There are natural statistics associated with data functions such as f. The first and most basic is the mean response of f, which is the average number of times any element of $M^{(n-k,k)}$. The first order summary counts the number of times an individual voted in the minority. The second order summary counts the number of times any given pair voted in the minority, and similarly for i^{th} order summaries. As a result of this counting method, each of these summaries contains a great deal of redundancy. To see the amount of redundancy consider the following example.

Example 2.2.3 *Suppose we have 7 voters, A, B, C, D, E, F and G and just a single roll-call is made. Suppose that the dissenting minority in this vote is* $\{A, B, C\}$. *Then the first order summaries are* $\{A\} = 1, \{B\} = 1, \{C\} = 1$ *and the rest of the individuals are 0. The second order summaries are* $\{A, B\} = 1, \{B, C\} = 1, \{A, C\} = 1$ *and the rest of the pairs are 0. The third order summaries are just* $\{A, B, C\} = 1$ *and the rest of the triples are 0. So as a result of just a single bill we get a great deal of over counting.* $f(\{A, B, C\}) = 1$ *contributes to seven different summaries even though it is only a single vote, hence the the data function* f *is used over and over again in these summaries. In general, the larger the subset in the minority the greater degree of over counting and redundant use of information.*

Removing this redundancy in the higher order effects is one of the amazing results of generalized spectral analysis. From Theorem 2.1.1, we know M decomposes into orthogonal, irreducible subspaces. The result (see [2]), is that, if $k \leq \lfloor n/2 \rfloor$ the space $M^{(n-k,k)}$ decomposes as follows:

$$M^{(n-k,k)} = M_0^{(n-k,k)} \oplus M_1^{(n-k,k)} \oplus \cdots \oplus M_k^{(n-k,k)} \tag{2.1}$$

where $M_i^{(n-k,k)}$ is an irreducible, orthogonal subspace corresponding to the *pure* "i^{th} order coalition effects." We state that they are pure in the sense that all redundant information has been removed. The i^{th} order effect space is exactly the i order summaries with the mean response, first order summaries,..., $(i-1)^{st}$ order summaries removed. The i^{th} order effect space measures of how much coalitions of size i are influential in the data after accounting for all coalitions small than i. Hence, M_1 describes the behavior of individual voters, M_2 describes the behavior of pairs of voters and so on. We can then apply "Mallow's Method" which is simply choosing some interpretable vectors that lie in these subspaces that correspond to specific coalitions and take their inner product with our data vector f to find exactly which coalitions have the greatest effect.

To restate, spectral analysis involves the study of projections of f onto its irreducible invariant subspaces of the corresponding $\mathbb{C}S_n$-module M. These projections tell us a story about which coalition effects explain the behavior of the voters. In the case of the U.S. Supreme Court, spectral analysis will consist of studying the following decomposition of M:

$$
\begin{aligned}
M^{(9,0)} &= M_0^{(9,0)} \\
M^{(8,1)} &= M_0^{(8,1)} \oplus M_1^{(8,1)} \\
M^{(7,2)} &= M_0^{(7,2)} \oplus M_1^{(7,2)} \oplus M_2^{(7,2)} \\
M^{(6,3)} &= M_0^{(6,3)} \oplus M_1^{(6,3)} \oplus M_2 \oplus M_3^{(6,3)} \\
M^{(5,4)} &= M_0^{(5,4)} \oplus M_1^{(5,4)} \oplus M_2^{(5,4)} \oplus M_3^{(5,4)} \oplus M_4^{(5,4)}
\end{aligned}
$$

and determining which coalitions have the greatest effect on the data. In general, we ignore the space corresponding to unanimous votes, in this case $M^{(9,0)}$. We do so since the analysis has trivial decomposition as shown above. The analysis of the Supreme Court is performed in great detail in [9].

2.3 An Example

In this section, we present an example that is also discussed in [9]. This analysis was done using the programs written in Appendix B. This example demonstrates the application of the theory stated above in analyzing approval voting data. Here one can view the data as the result of a committee of five professors (A, B, C, D, E) voting on the approval of a 150 different text books. We would like to apply spectral analysis to describe the behavior of the five professors.

The first step in our analysis of our data function f is to decompose f into homogeneous spaces, as noted in Section 2.2. Following the decomposition shown above, $f^{(4,1)}$ is the data corresponding to the decisions that had one person in the minority and $f^{(3,2)}$ is the decisions that had two people in the minority. Thus the data function f is comprised of two parts, $f^{(4,1)}$ and $f^{(3,2)}$. If the original data vector

f is:

$$f = \begin{pmatrix} 4 & A \\ 9 & B \\ 15 & C \\ 5 & D \\ 7 & E \\ 11 & AB \\ 10 & AC \\ 4 & AD \\ 7 & AE \\ 9 & BC \\ 7 & BD \\ 6 & BE \\ 24 & CD \\ 17 & CE \\ 15 & DE \end{pmatrix}$$

then f decomposes as follows:

$$f^{(4,1)} = \begin{pmatrix} 4 & A \\ 9 & B \\ 15 & C \\ 5 & D \\ 7 & E \end{pmatrix}$$

$$f^{(3,2)} = \begin{pmatrix} 11 & AB \\ 10 & AC \\ 4 & AD \\ 7 & AE \\ 9 & BC \\ 7 & BD \\ 6 & BE \\ 24 & CD \\ 17 & CE \\ 15 & DE \end{pmatrix}.$$

For example, there was 4 votes in which A was in the minority against the other committee members, and there were 11 votes in which A and B were in the minority against C, D and E.

As demonstrated in Section 2.2, we define M to be the vector space of functions defined on one and two person subsets of individuals. Then M decomposes as

$$M = M^{(4,1)} \oplus M^{(3,2)}$$

which further decomposes as

$$\begin{aligned} M^{(4,1)} &= M_0^{(4,1)} \oplus M_1^{(4,1)} \\ M^{(3,2)} &= M_0^{(3,2)} \oplus M_1^{(3,2)} \oplus M_2^{(3,2)}. \end{aligned}$$

For both $M^{(4,1)}$ and $M^{(3,2)}$, we again point out that M_0 contains the "mean response" which is simply the average number of times a subset would be in the minority for a given number of votes. In general, the mean response contains no important information in regards to voter behavior and is simply the average number of votes for that space. For $i \geq 1$, however, M_i contains the "coalition" or "ith order" effects of subsets of i individuals, as described in [2] and [14].

2.3.1 *Decomposition of One-person Minorities*

Here we will demonstrate the decomposition of $f^{(4,1)}$, which is done by projecting $f^{(4,1)}$ onto $M_0^{(4,1)}$ and $M_1^{(4,1)}$.

This gives the following decomposition: $f^{(4,1)} = f_0^{(4,1)} + f_1^{(4,1)}$ where

$$f^{(4,1)} = \begin{pmatrix} 4 \\ 9 \\ 15 \\ 5 \\ 7 \end{pmatrix} \quad f_0^{(4,1)} = \begin{pmatrix} 8 \\ 8 \\ 8 \\ 8 \\ 8 \end{pmatrix} \quad f_1^{(4,1)} = \begin{pmatrix} -4 \\ 1 \\ 7 \\ -3 \\ -1 \end{pmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix} .$$

The mean response function $f_0^{(4,1)}$ is a vector of 8's which is the average number of votes that occurred. The first order effect $f_1^{(4,1)}$ is just the amount which each individual differs from the mean. The interpretation of this decomposition is natural: the largest value is for professor C which tells us that C was in the minority the most (a rebel against the other committee members) where A had the smallest value which indicates professor A's desire to not vote alone in the voting process.

This is a typical interpretation of the decomposition of the data. In a split such as $(n-1, 1)$ a careful observer could see this behavior without the decomposition but, it becomes less obvious in higher coalition effects as demonstrated below.

2.3.2 *Decomposition of Two-person Minorities*

With two people in the minority there is now the mean response, the individual effects, and pair effects, thus the two-person minority data is decomposed into three parts. This is done by computing the projection of $f^{(3,2)}$ onto $M_0^{(3,2)}$, $M_1^{(3,2)}$, and $M_2^{(3,2)}$. This results in the decomposition

$$f^{(3,2)} = f_0^{(3,2)} + f_1^{(3,2)} + f_2^{(3,2)}$$

where

$$f^{(3,2)} = \begin{pmatrix} 11 \\ 10 \\ 4 \\ 7 \\ 9 \\ 7 \\ 6 \\ 24 \\ 17 \\ 15 \end{pmatrix} \quad f_0^{(3,2)} = \begin{pmatrix} 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \end{pmatrix} \quad f_1^{(3,2)} = \begin{pmatrix} -7.7 \\ 1.3 \\ -2.0 \\ -3.7 \\ 1.7 \\ -1.7 \\ -3.3 \\ 7.3 \\ 5.7 \\ 2.3 \end{pmatrix} \quad f_2^{(3,2)} = \begin{pmatrix} 7.7 \\ -2.3 \\ -5.0 \\ -0.3 \\ -3.7 \\ -2.3 \\ -1.7 \\ 5.7 \\ 0.3 \\ 1.7 \end{pmatrix} \quad \begin{matrix} AB \\ AC \\ AD \\ AE \\ BC \\ BD \\ BE \\ CD \\ CE \\ DE \end{matrix} .$$

The interpretation of these projections takes a little more work. Again the mean vector, $f^{(3,2)}$, is simply the number of votes (110) with two people in the minority divided by the $\binom{5}{2} = 10$ possible coalitions of size two. Therefore, because there are 110 votes with two people in the minority, the mean value for each pair is 11 (110 divided by 10). To interpret the pure first and second order effects, $f_1^{(3,2)}$ and $f_2^{(3,2)}$, we introduce Mallow's method.

2.3.3 Mallow's Method

The vector $f_1^{(3,2)}$ is the resulting pure first order effects but the resulting vector is not as easy to interpret as was $f_1^{(4,1)}$. We also point out that $f_1^{(3,2)}$ are pure effects in the sense that it is simply the first order summaries without the mean response. In order to get a measure of how influential each single professor is we apply Mallow's method as described in [2] and [14]. Mallow's method is simply computing

the inner products of naturally interpretable vectors with our projection, in this case, $f_1^{(3,2)}$. The idea behind Mallow's method is that our projection $f_1^{(3,2)}$ is just a vector lying in $M_1^{(3,2)}$, hence it has a length and a direction. We must find to which coalitions $f_1^{(3,2)}$ lies closest. To do this we create vectors, described below, for each coalition and take their inner product with $f_1^{(3,2)}$ where the value represents how close the data lies in each coalition's direction. The interpretation of the values is largely geometric where large positive values are interpreted as the data lying close in that particular coalition's direction, large negative values are interpreted as pointing in the opposite direction of that coalition.

These coalition vectors are easy to create. For example if we wanted to find Professor A's effect we would define a vector f_A which would have a 1 if it contains A and 0 if not. Below are the interpretable vectors for A, B and C:

$$f_A = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad f_B = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad f_C = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \begin{matrix} AB \\ AC \\ AD \\ AE \\ BC \\ BD \\ BE \\ CD \\ CE \\ DE \end{matrix} .$$

The inner product of $f_1^{(3,2)}$ with each f_Y describes how much $f_1^{(3,2)}$ "lies in the direction of" f_Y, for $Y \in \{A, B, C, D, E\}$. Table 2.1 shows the inner products of $f_1^{(3,2)}$ with each of the "naturally interpretable" functions.

Table 2.1: Pure first order effects in 3-2 splits using Mallow's method

	f_A	f_B	f_C	f_D	f_E
$f^{(3,2)}$	-12	-11	16	6	1

Table 2.1 completely describes the pure first order effect in the $M^{(3,2)}$ space. By inspection, we can see that the prominent direction in which our data resides is in the direction of professor C. It would also appear that the data lies in the opposite direction of professor A. This gives rise to the explanation that in the case where there is a 3-2 split, professor C is very much in the minority and professor A is not.

We can continue this analysis for the second order data but in this case it is clear how to interpret the results. Now our Mallow's vectors are just a 1 in the corresponding pair and 0 everywhere else, thus the actual vector $f_2^{(3,2)}$ suffices to explain the second order effects. This is true in general, i.e., the vector $f_k^{(n-k,k)}$ is already interpretable using Mallow's Method. We list the result in Table 2.2 for clarity.

Table 2.2: Pure second order effects in 3-2 splits

	$f_{A,B}$	$f_{A,C}$	$f_{A,D}$	$f_{A,E}$	$f_{B,C}$	$f_{B,D}$	$f_{B,E}$	$f_{C,D}$	$f_{C,E}$	$f_{D,E}$
$f^{(3,2)}$	7.7	-2.3	-5.0	-0.3	-3.7	-2.3	-1.7	5.7	0.3	1.7

Now it is clear that after removing the mean and the *pure* first order effects there is still a story to tell about the *pure* second order effects. It would appear the the pair $\{A, B\}$ is prone to working together and are often in the minority together where as the pair $\{A, D\}$ avoids being in the minority together.

2.4 The Statistics of Spectral Analysis

2.4.1 Bootstrapping

Statistical tools for spectral analysis have been developed in several papers, although there still appears to be open questions as to the best way of analyzing this data. A discussion of the Binomial and Multinomial methods can be found in [1]. Advanced techniques using Gröbner Basis is described in [4].

Given the number of bills and the number of voters, how do we know when the projection of $f_i^{(n-k,k)}$ is significant, i.e., how do we know this did not occur by chance? One common way of determining significance is to compute the squared norm, $||f_i^{(n-k,k)}||^2$ and divide through by the dimension of the subspace $M_i^{(n-k,k)}$. The idea would be that if this data was generated randomly that, in general, the amount of the vector that would lie in the subspace $M_i^{(n-k,k)}$ would be proportional to the dimension of that space. Thus deviations from this proportions might indicate structured or significant data. Diaconis [2], however, notes that

> It is customary in comparing sums of squares to divide by the dimension of the subspace. This makes sense if it is thought that the projection is reasonably spread out between a natural system of coordinate vectors so that the sum of squares is a measure of noise. If, as in the present example, the projections are likely to be quite structured, lying close to a few interpretable vectors, dividing by dimension is likely to be deceptive.

As our example above shows, structure is very likely to be found in human voting behavior and as such we will determine significance of the squared norms of our projections without dividing by their corresponding dimension. To perform this analysis we apply the computational method of bootstrapping.

The idea of bootstrapping is a relatively simple one. Given the number of bills and the number of voters, we can compute a distribution on the projections by sampling at random from all possible voting results on a bill. So for our example above we indicate to our bootstrapping program, found in Appendix B, that there are 5 voters, 40 bills split 4-1 and 110 bills split 3-2. Our approach will be to find which projections have a significant squared norm with respect to the distribution computed by the program. This will tell us which subspaces vary significantly from noise, or a distribution computed by randomly selecting data given the constraints above. Once inside these subspaces, we then can look at the actual coalition behavior and give a description of how the data behaves in the same manner that we did in the previous section using Mallow's method.

The table 2.3 below shows the resulting mean and standard deviations of the squared norms from 10000 samples with respect to the example in section 2.3.

Table 2.3: Significance via bootstrapping

		observed	bootstrapped		Significance
split	subspace	norm^2	mean	std. dev.	
4-1	M_1	76.0	32.2	22.3	projection not sign'f't
3-2	M_2	146.0	44.0	31.0	Significant
3-2	M_1	186.0	55.0	34.7	Significant

If an observed value is 3 standard deviations away from the mean, we will say that it is significant at the .05 level, meaning the observed value has exceeded the expected value by more than 3 standard deviations (above or below). This indicates that the value falls farther away from the mean than 95% of the possible projections. To see this, the histogram in Figure 2.1 is the distribution computed

from 10000 random samples of $||f_2^{(3,2)}||^2$.

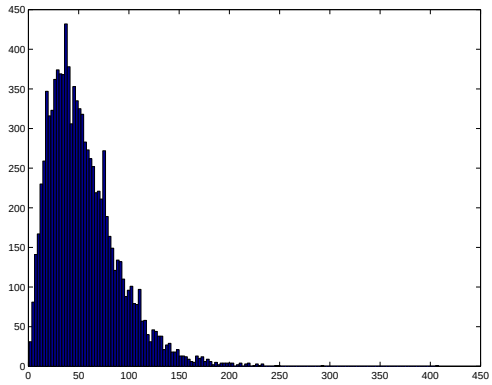


Figure 2.1: The distribution computed by bootstrapping technique on the squared norm of $f_2^{(3,2)}$

It should be pointed out here that these statistical techniques are rather coarse. In general, this method of bootstrapping will conclude that most projections are significant. The reason for this is that data with "mild" coalition behavior, which most data has, will seemingly always differ from random samples enough to pass this significance test. More refined techniques are discussed in [4] and more will be said in Section 4.

2.4.2 A Short Note on the Connection to ANOVA

The additive decomposition of generalized spectral analysis can be viewed as a generalization of the Analysis of Variance (ANOVA) as discussed in Chapter 8 of [2]. In two-way ANOVA with 1 repetition per cell, data is collected in an $I \times J$ matrix $X = \{x_{ij}\}$, where each row corresponds to a different level of one factor and each column is a different level of the second factor. Similar to spectral analysis, in order to analyze which factors are important, an additive decomposition of the data matrix X is computed where the "row effect", the "column effect" and the

grand mean are all separated. In this case, ANOVA is simply studying this additive decomposition of the data to determine the importance of both factors.

2.5 Discussion of the method

As can be seen from the example demonstrated above, the computation of spectral analysis is easy to interpret and finding large coalition effects, both positive and negative, is also easy. Thus, if we were handed some data vector f and were not provided any background on the data, we could compute this analysis and be able to tell stories about the data as we did with the five professors above. This ability makes spectral analysis a very powerful exploratory tool.

If we try to apply this to sets of data with a large number of voters we run into some difficulty. The amount of computational time grows quickly with the number of voters. For the Supreme Court our space is of size $\sim 2^9 = 252$ which is still reasonable for our computer to tackle, but for the U.S. senate with dimension of $\sim 2^{100}$ our methods fail. Thus, in order to apply spectral analysis to these larger data sets, we develop a partitioning technique that drastically reduces the size of our set while maintaining the characteristics of the data. This technique is the focus of the next chapter.

Chapter 3

Approach to Large Sets: the Partitioning Technique

3.1 Description of the Partitioning Technique

Suppose we were handed a "large" set of data, i.e., a data set with a lot of voters to analyze. We attempt to run the data set through our standard techniques but the computer grinds to a halt and error messages begin flashing. We are at first discouraged that our computer can no longer compute the spectral analysis on this data. We decide that if we cannot analyze the individual voter's coalition behavior, we might instead collect up the individuals into separate groups and now ask if we can understand the voting behavior between *groups*. The idea is to take the original n members and partition them into k groups where $k \ll n$. For example, in the case of the U.S. Senate, dividing groups up by party, gender, and region might be a useful way to study the Senate.

Now given these groups, we would like to apply spectral analysis to measure the coalition effects between the groups. To do this, we need to translate the original data collected from each bill into data in the smaller space where each group is considered a single voting member. We solve this problem by asking the following question: Given the result of a single bill that corresponds to a value of 0 for the subset of senators that were in the minority and a 1 otherwise, how can we distribute this value in the resulting partitioned space? In other words, how can we distribute this value as weights for the subsets in our new space?

We answer this question by asking another. For a given bill, we look at our first group, and ask: If I picked a member of this group at random, what is the

probability, p_1, that I choose someone who was in the majority? We distribute that weight to all subsets in our new space that places group 1 in the majority set. We then distribute the weight $1 - p_1$ to all subsets that place group 1 in the minority. We then look at the second group. We ask the same question and get a probability, p_2. We now distribute the weight p_2 to all subsets in our new space that places the second group in the majority and likewise for $1 - p_2$. Continue this for all k groups. The result is a collection of weights in front of each element of the power set of the new space.

Example 3.1.1 *Suppose we had 5 different groups voting on a single bill. Each group has probability p_i that a member of the group voted in the majority. The weight in front of the dissenting subset $\{1, 2, 4\}$ would then be $(1 - p_1)(1 - p_2)p_3(1 - p_4)p_5$.*

The weight in front of the dissenting subset that contains all k groups is $(1 - p_1)(1 - p_2) \dots (1 - p_k)$. The total weight in front of the dissenting subset that contains all k groups except group i and j has weight, $(1 - p_1)(1 - p_2) \dots (p_i)(1 - p_{i+1}) \dots (p_j)(1 - p_{j+1}) \dots (1 - p_k)$. We now state a simple proposition.

Proposition 3.1.1 *For a single bill, the sum over all possible weights is equal to 1.*

PROOF: Since each weight corresponds to the probability that a certain subset of senators are in the majority and the complement of that subset is in the minority, summing over all possible weights corresponds to summing over all the possible probabilities, hence this sum must equal 1.

■

Now we have a way of translating each bill to a bill in our partitioned space with equal weight. By iterating this process and tallying the weights, we now can transform a general data set into data in our partitioned space.

We can now translate each bill into weights of the power set of our n groups, but as you will recall, in spectral analysis we are only concerned with the elements

of the power set of up to size $\lfloor n/2 \rfloor$. Hence we have distribute weights in front of elements we currently do not analyze. The natural correction is to identify the elements of the power set that are larger than $\lfloor n/2 \rfloor$ with its complement and add the corresponding weight.

Example 3.1.2 *After computing the partitioning technique for three groups, A, B, and C, with corresponding probabilities,* $p_A = .8, p_B = .6$ *and* $p_C = .1$*, we obtain the following distribution of weights on the power set:*

$$f(\{\emptyset\}) = .048, \quad f(\{A\}) = .012, \quad f(\{B\}) = .032, \quad f(\{C\}) = .432,$$
$$f(\{A,B\}) = .008, \quad f(\{A,C\}) = .108, \quad f(\{B,C\}) = .288, \quad f(\{A,B,C\}) = .072.$$

We now identify the corresponding subsets to obtain:

$$f(\{\emptyset\}) = .12, \quad f(\{A\}) = .3, \quad f(\{B\}) = .14, \quad f(\{C\}) = .44 \quad .$$

Notice that the corresponding data vector $f^{(2,1)}$ *is:*

$$f^{(2,1)} = \begin{pmatrix} .3 \\ .14 \\ .44 \end{pmatrix} \quad \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

By proceeding in general with this identification, we obtain the data vector which is now ready for spectral decomposition.

3.2 Application of Partitioning: the U.S. Senate

How do we know this technique is useful or valid? What does it mean to be valid? Well if the technique is valid then it should not result in contradictory information, meaning, if an obvious and important coalition among groups is not detected then we should have cause to worry.

A test of whether a technique is "valid" is if, when applied, it gives "valid" results. Thus we apply this technique to data that has been well studied, which will make it easier to decide if our analysis is correct. The data comes from the first session of the year 1999 of the 106^{th} U.S. Senate. Analysis of the U.S. Senate data can be found in [13] using the spatial model approach.

As we noted above for $n = 100$ senators, our ambient space has dimension of order $\sim 2^{100}$! As it turns out this level of computation is fantastically difficult for computers. In fact, Matlab can run our programs up to about $n = 20$, but beyond that, Matlab fails.

Hence, the Senate is a good candidate for the partitioning method. The first question that arises is how do we partition the Senate? The obvious partition would be along some of the known characteristics of the Senate. For our case we choose three characteristics:

1. Party Affiliation: Republican or Democrat

2. Gender: Male or Female

3. Region: North or South.

This results into 2^3 distinct subsets of voters. For example: Southern, Republican, Females is one group and Northern, Republican, Males are another. This choice of partitioning was picked as it can serve as a means to compare to what is popularly known about the behavior of the senate, i.e., that party affiliation is the most important factor in deciding who you align yourself with, much more important than region and gender is. A failure to detect this using this partitioning technique would challenge its validity. What is also desired is the detection of other lesser known coalitions among these characteristics.

The raw data is shown graphically in Figure 3.2 where blue is a vote in the minority and white is a vote in the majority. There are eighty-one bills and ninety-

nine senators in this data set. (One senator abstained a great deal in this data and thus was left out. How to handle abstentions is briefly discussed in the chapter 4)

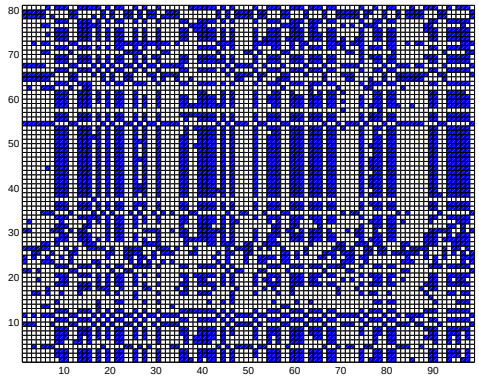


Figure 3.1: Raw Data "signal" for 106^{th} U.S. Senate

After applying our partitioning technique to the above data, we still have 81 bills listed but now we have 8 columns corresponding to the eight subsets as shown in Figure 3.2.

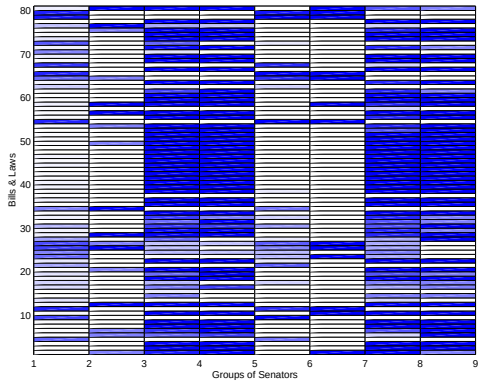


Figure 3.2: Results of apply the partitioning technique to the 106^{th} U.S. Senate Data

One important observation to make is that the values of the matrix no longer are one or zero but range between the values of zero and one corresponding to the

different shades of blue.

After rewriting the original data into its partitioned form, we calculate the projections of this data. To verify whether party is the most important factor of the three, we need to check the projections of our data into the $M^{(4,4)}$ subspace, where the votes split four-four. Continuing the notation established in Chapter 2, the irreducible subspace corresponding to the fourth order effects, $M_4^{(4,4)}$, contains all the important information about all pure fourth order coalitions. Below are the graphic projections of $M_4^{(4,4)}$ using Mallow's method so that each bar corresponds to a particular coalition of 4.

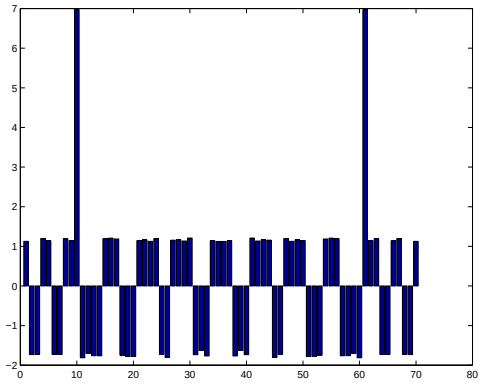


Figure 3.3: Plot of $M_4^{(4,4)}$ derived from data of the 106^{th} U.S. Senate

The two large "spikes" correspond exactly when the Democrats and the Republicans all vote together. The projections of these spikes dominate the other projections, dictating that data in this irreducible subspace has a great deal of the data pointing in the direction of the two party coalitions. This result is very encouraging and provides strong evidence to the validity of this new technique. In this data set we have been able to demonstrate that this reduction in dimension of the original data preserves the characteristic of party voting as the dominant coalition behavior. In the next section we discuss how one can go about describing the dynamics of the technique which will provide another verification of why we seem to get valid results.

3.3 Understanding the Technique

The result above is highly encouraging. Alas if we could make rigorous what it means to preserve the characteristics of the original data, we would have a fine result that would explain why the method above produced a good analysis of the data. Our first attempt at doing so is using a perturbation analysis on random sets of data to get an understanding of the dynamics of the partitioning technique.

For example, given a set of data that we partition and then compute the projections, it is our hope that if we perturb the original data a little, the resulting projections are stable. How do we go about "perturbing" our data?

The idea is relatively simple. Our method is a way of performing spectral analysis between groups of voters. If for all the groups, each voter in a given group votes exactly the same, then the partitioning effect is essentially comparing the uniform voting records of each subset against each other. The result of this complete agreement for each group results in a smaller data matrix, but of still ones and zeros (there are no fractional values since there is no disagreement).

Suppose we had 80 voters broken into 8 groups of 10 who participate in 100

roll-call votes. If every member in each group agreed completely on the bills, then the partitioning technique reduces the original 80×100 data matrix to an 8×100 matrix of ones and zeros, where each column is exactly the voting record of each group. Thus the spectral analysis of the 80 voters reduces to the spectral analysis of 8 voting records.

Given the spectral decomposition of the 8 voting records, what would happen if we perturbed the voting data of each group? For example, what would happen to the spectral analysis if each group now disagreed a little among each other? Can we discover the "behavior" of the spectral decomposition given the amount of disagreement among each group? We will use Hamming distance, denoted $H(x, y)$, as our metric in performing this analysis. Recall that the Hamming distance between two vectors is the number of coordinates in which the vectors differ. Below we extend this definition to a set of vectors.

Definition 3.3.1 *The* **diameter** *of a set of vectors X with respect to Hamming distance, denoted $D_H(X)$, is defined by*

$$D_H(X) = max\{H(x_i, x_j) : \quad x_i, x_j \in X\}.$$

For example, if a set of vectors X has $D_H(X) = 1$, then there are at most two different vectors in our set X. Moreover, the two different vectors differ by one coordinate.

With this definition we will now discuss what happens when our groups begin to disagree. The measure of how much they disagree among each other will be measured by the diameter, D_H, of the set of data vectors of each group. Again if the diameter of a given group is 1, then all the members of that group agreed on every bill but one, in which case the voters disagreed (not necessarily 50/50) on this bill. Thus there is only two distinct different vectors in this subset.

The question then arises: As the diameter of each group grows, can we predict

what the effect on the spectral analysis of the higher order effects might be? To answer this question we turn to numerical simulations.

We continue the example above with the 8 groups of 10 voters deciding on 100 roll-calls. One measure of understanding the dynamics is to try and measure the squared norms of the projections of our data into each irreducible subspace as the diameter changes.

To obtain these measurements we ran 1000 random simulations of the 80 voters in the case where each group had a diameter of $\{0, 1, 2, 5, 10, 15, 25, 50, 75\}$, and recorded the squared norm of the projections corresponding to every irreducible subspace. We then computed the mean for each squared norm of each subspace and plotted them with respect to diameter. We provide two sample plots below of the projection into $M_4^{(4,4)}$ and $M_1^{(1,1)}$.

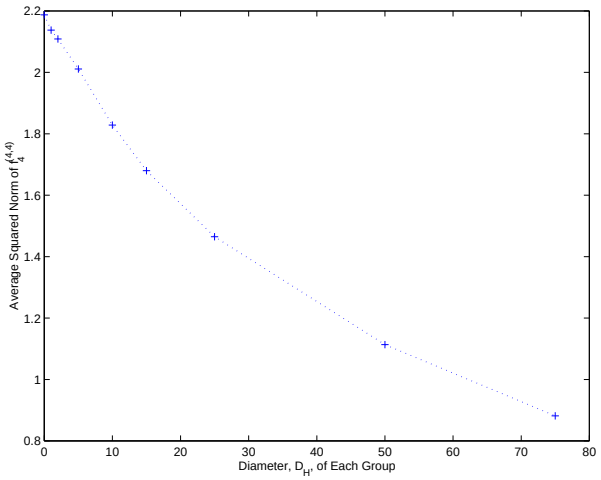


Figure 3.4: Plot of the average squared norm of $f_4^{(4,4)}$ with respect to the diameter

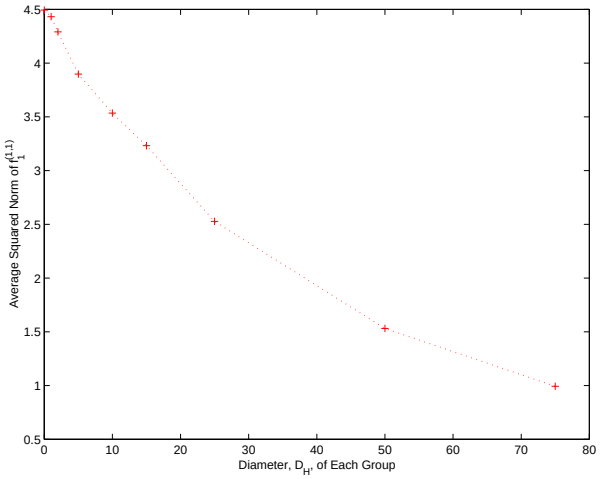


Figure 3.5: Plot of the average squared norm of $f_1^{(7,1)}$ with respect to the diameter

As we can see from Figures 3.4 and 3.5, the size of the projections drop of as the the groups disagree more and more. These graphs are representative of all the non-mean subspaces projections of the data.

What can we learn from this? It would appear that by choosing groups that agree a good deal, we can minimize this drop off in squared norms which correspond to a loss of detectable coalition behavior. The fact that along party lines there is such a high degree of agreement might explain why party was so easily detected in the U.S. Senate example.

The other interesting result is the behavior of the mean response as the diameter among each group grew as shown in Figure 3.6.

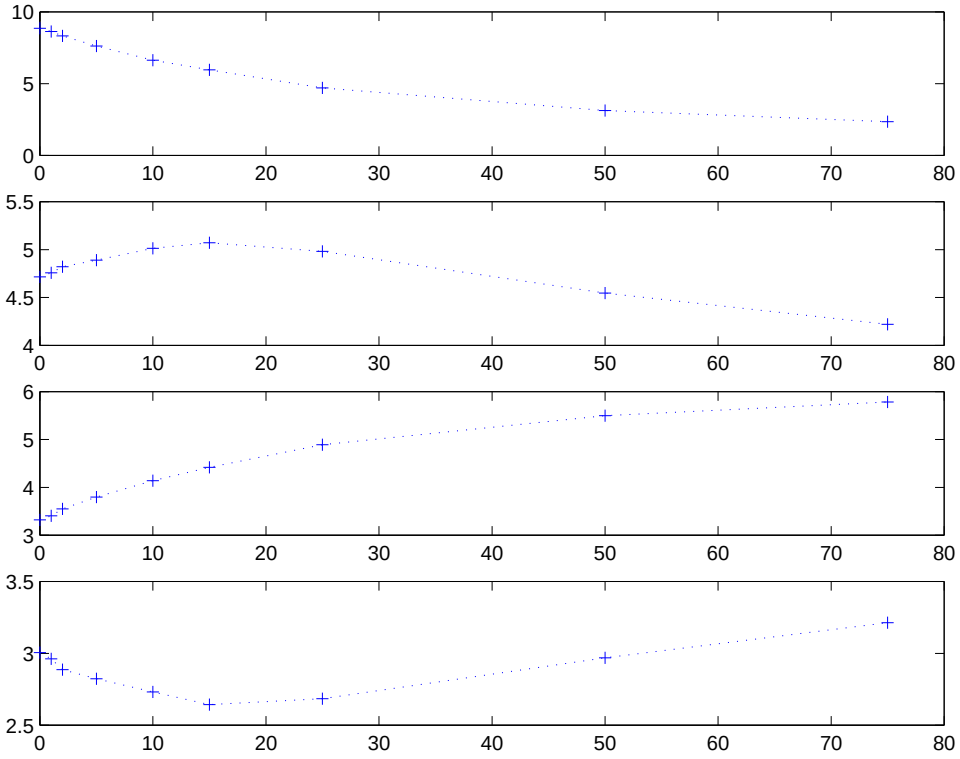


Figure 3.6: Plot of the average squared norm of $f_0^{(7,1)}$, $f_0^{(6,2)}$, $f_0^{(5,3)}$, and $f_0^{(4,4)}$ with respect to the diameter

The changing parabolic behavior from $M_0^{(7,1)}$ to $M_0^{(4,4)}$ is most intriguing. It would appear that the the data vector that resides in the $M^{(4,4)}$ space appears to collect in the mean response space as the the diameter grows. One explanation of this might be the fact that as the groups disagree more and more among each other no discernable coalitions *between* groups can be detected and thus all data residing in this space will be noise, or unrecognizable from the mean response.

These numerical results begin to give us an understanding of how the partitioning technique affects the data. The decaying squared norms in the non-mean response spaces suggest that a good choice of groups, i.e., members of a group agree with each other well, will produce good detectable results. The behavior of the squared norms of the mean response spaces are interesting and a closer look at the projection formulas might reveal how the partitioning technique is affecting these spaces.

3.4 A Short Comment on The Statistics

The statistical analysis of the partitioning method is just an extension of the statistics described in section 2.4. We introduce the actual partitioning technique as step in between generating random samples and computing their projections. After generating the appropriate random samples, we run these samples through the partitioning technique and compute the lower dimension projections.

While this seems like the natural extension of the bootstrapping method, the test becomes even more sensitive to any structure found in the data. As a result, all the projections found above are significant. One explanation for this is that the number of possible vectors to sample out of grows immensely with the size of the original space, hence the probability that a group of vectors are chosen that indicate any coalition behavior becomes quite small. A more refined technique is needed here and is discussed in the conclusion.

Chapter 4

Conclusion and Future Work

We have been able to fully apply the abstract theory of generalized spectral analysis to approval voting data. Given a set of data we can fully analyze the coalition behavior using our programs. If the set of data has many legislators, we have designed a new technique to reduce the dimension of the data space by analyzing the coalition behavior between subsets of legislators. The numerical results in Chapter 4 begin to reveal an understanding of the dynamics of this new technique. We have also designed a simple test for the significance of our spectral decomposition of the data. The majority of this work resulted in fast algorithms to implement the spectral analysis. Hence, the main result of this thesis is a rather complete package of programs.

As a result of this work, my co-author Brian Dawson, a political scientist at the University of Cincinnati, has been able to analyze the Supreme Court (the past 50 years) using generalized spectral analysis. These new methods have been presented at several conferences and appear to be generating interest.

There is a great deal of work that still needs to be done on this project. The first would be to continue to work on the statistics. One method of refining our significance technique would be to run this coarse test first and find the most significant projection. We would then hold this value fixed and sample again with the condition that this projection is preserved. The idea is to recover significant projections recursively. A technique like this is described in [4], but it would appear that there is yet to be a computationally efficient way to perform such analysis. Any refinement of our statistics would be useful, especially when testing data that has been

partitioned. Our statistical methods for this technique need to be improved.

Beyond statistics is the idea of generalizing these methods to non-binary data, such as partially ranked or ranked data. The representation theory has been developed to support such analysis but the implementation is more difficult. Progress in this area would make these techniques more versatile. One immediate result of such a generalization is the ability to incorporate abstentions in voting by placing an abstention "vote" between a majority and minority vote.

It should also be pointed out that although the discussion and implementation of these techniques are framed in the context of approval voting, the methods are just as valid in analyzing most longitudinal binary data. That is, binary data arising from random variables that is recorded in discrete time steps. Analyzing this data using spectral analysis is similar to a generalization of higher order correlation among the random variables. This encompasses a great deal of data taken by scientists and thus could be applied to their data. For example, after presenting a poster on this material at conference of mostly biologists, I was offered binary data to analyze that was coming out of medical trials to test the effect certain medication had on finger coordination.

Lastly, it would be nice to develop more of the theory on our partitioning technique. A great deal of data falls into the category of "large" and it would be nice to establish this technique as a useful and effective tool in measuring the coalition effects between subsets of legislators. The numerical results presented in Chapter 4 are just small steps in understanding the behavior of this technique in the context of spectral analysis.

Appendix A

Manual for the Program

In this appendix, I provide instruction on how to use the main components of the program. A copy of Matlab is necessary to run the programs found in Appendix B. These programs are multifaceted and have several options, so I will only discuss a subset of them.

Let us begin with handling the data from an outside file. Your data should be an $m \times n$ matrix where the columns correspond to the voters and the rows correspond to a particular roll call vote. Their should be no space in between the entries of the matrix and no lines skipped between rows. If your data is to be partitioned then the top row should be reserved to indicate the partition. To label your data into n groups, you need to place in the first row of a particular column a number, i, from 0 to $n-1$ that indicates membership into group i. The program will be able to read the partitions this way. The rest of the matrix should be nothing but 1's and 0's. In addition, 0 is the default indication of a minority vote and thus there should always at least as many 1's as 0's in each row.

Suppose that your data file's name is Foo.txt. To read in this file, type:

```
DATA = dlmread('Foo.txt', ' ');
```

Your data will now be the $m \times n$ matrix labeled DATA. To check this you can type:

```
DATA
```

in the command line and your matrix should appear. To prep your data for spectral analysis we need to rewrite this data into a collection of vectors corresponding to

$f^{(n-1,1)}$, $f^{(n-2,2)}$, ..., $f^{(n-k,k)}$. To do this we use the FinalDataRetrieval.m program. If your data needs to be partitioned, type:

```
[D] = FinalDataRetrieval(DATA, 1)
```

otherwise type:

```
[D] = FinalDataRetrieval(DATA)
```

This produces a $1 \times k$ cell, D, where the i^{th} cell entry is the data vector $f^{(n-i,i)}$. With the cell D, you are almost ready to compute the spectral analysis of your data. The projection method onto the irreducible subspaces is done very efficiently using an eigenspace approach. This eigenspace method was developed in [11] and is also discussed in [10]. To perform this efficient projection method we need to feed our program the adjacency matrices of certain Johnson graphs of appropriate size. To generate these matrices type:

```
[M] = GenerateJohnson(N);
```

where N is either the number of voters in your original space if no partitioning is needed or is the number of different groups in the partition.

This computes the necessary matrices for our eigenspace projection method, which are stored in the $1 \times k$ cell M. We will use M several times so be careful not to reassign M. From here you are ready to compute the spectral analysis of your data. Here you have a couple options. To produce the projections in the fashion seen in Chapter 2, you will use the FinalDecomposition.M program. To do so type:

```
[B,Q,q,R] = FinalDecomposition(D, M);
```

This command produces four different cells, B, Q, q, and R. Each cell will be $k \times (k+1)$. The i, j entry of B is the value of $||f_j^{(n-i,i)}||^2$ for all $i, j \leq k$ and $j \leq i$. The

$k+1$ column corresponds to the squared norm of the mean for each vector $f^{(n-i,i)}$. Thus, B is a lower diagonal matrix except for its last column and indicates all the important squared norm values.

The cell Q has the same indexing as B but in each cell entry you have the projection data after applying "Mallow's method" when appropriate. It is here that one can begin to inspect which coalitions appear to be important. You can compute graphics similar to Figure 3.3 by typing:

```
bar(Q{i,j})
```

The cell q has the same indexing as Q but each cell entry has the raw projection data without applying "Mallow's method." This cell is less helpful for discerning important coalitions but is useful to have nonetheless.

The last cell, R, is very similar to Q but with a modification. All the projections have been sorted from least to greatest and next to their value has been appended the explicit coalition vector. A 1 in the coalition vector means "not in the coalition" and a 2 means "in the coalition." It is here that one would look first to see which coalitions are important. This organization of the data makes it very easy to get an understanding of the important coalitions of the data.

After inspecting the cells you might want to compute a significance test on your data vector. To do this you again have a couple of options. If your data is not partitioned, then you will be using the FStatsNP.M program. Your command will have the form

```
[F, T] = FStatsNP(X,Y,M,V);
```

where X corresponds to the number of legislators (should be N). Y is the number of trials you would like to run. As a general rule of thumb, 1000 should be the minimum number of trials. M is the previously computed cell that contains all the adjacency matrices. V is a vector who's i^{th} entry indicate the number of bills

decided by a $[n-i, i]$ split vote. You can hand count this or you can run Vbuilder.M by typing:

```
[V] = Vbuilder(DATA);
```

After this you are ready to run FStatsNP. The screen will tick off which trial it is running and how long it takes so you will know about when the computation will be done. When finished, the variable F is a $k \times (k+1)$ cell that is indexed exactly the same as B. In each entry of F is two numbers, the first is the mean value of the squared norm of the particular projection and the second value is the standard deviation. T is also a $k \times (k+1)$ cell indexed in the same fashion but each cell contains all the squared norms from each trial. By typing:

```
hist(T{i,j},100);
```

you can produce the distribution graphics in the same fashion as graphic 2.1. It is using the values from T that let you now test your data for significance as described in section 2.4 and organize your results into tables such as Table 2.3.

If you do have partitioned data then you will use FinalStatsA.m by typing:

```
[F, T] = FinalStatsA(X,P,Y,M,V);
```

where all the variables are the same as above for FStatsNP.m except you need to include the actual partition vector P. To do this just use the first line of your original data by typing:

```
P =DATA(1,:);
```

The rest is exactly the same as above.

After this you are essentially done. All the projections are computed nicely and the statistics have been generated. The other version of the programs such as FinalDecompositionA.M do similar computations but normalize the projections, when it is desirable at times to reporting on scales between -1 and 1. Good luck!

Appendix B

Programs

Here we attach the source code from our Matlab programs. Some were written by Michael Orrison, some jointly written by both myself and Michael Orrison, and most were written by just me. The comments in the code should indicate the authors. I've included the main programs first which are then followed by many of the smaller supporting programs.

B.1 FinalDataRetrieval.m

```
function [D] = FinalDataRetrieval(Raw, sig)

%Exactly the same as DataREtrieval except data can be a matlab matrix.
%FinalDataRetrieval(Raw, sig)
%
%Raw should be off the form foo.txt.  This is how DataRetrieval works. If your
%incoming data file foo.txt is already partition or none is need than just type
%DataRetrieval('foo.txt') and you will recieve back a cell, D, of iwasaki data. If you
%need partitioning done then you will type DataRetrieval('foo.txt',1).  The 1 signals
%a need to partition. If partitioning is done then you will receive the combined
%iwasakied data. DataRetrieval is a %collection of old m files:
%iwasaki.m, readpart.m, partition.m, and ubersaki.m.
%Still uses choose.m, index_to_set.m and set_to_index.m
%
% Author: David Uminsky, Fall 2002

DATA = Raw;


if nargin < 2,

   D = sorting(DATA);
end



if nargin == 2,

[DD P] = readpart(DATA);

[DDD PP] = partition(DD,P);

[DD] = UltraSorting(DDD);
```

```
   D = DD;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
function D = sorting(DATA)

% sorting(DATA)
%
% takes takes in a matrix DATA whose rows are consist of
% seqences of 0's and 1's, where each row has no more 0's
% than 1's.  The idea is to view the rows of DATA as the
% partitions of a set, where 1 means "winner" and 0 means
% "loser".  The output is the tabloid version of the incoming
% data in DATA.  This can then be run through BUDR or EMMYR
% to complete the spectral analysis.  Note: we also assume
% that there is at least one loser in each partition.

V = size(DATA,1);                  % the number of votes
N = size(DATA,2);                  % the number of things to partition
K = floor(N/2);                    % the most number of "losers"
D = cell(1,K);                     % where we store the output

for k = 1:K,
  data = sparse(choose(k,N),1);         % what we'll eventually store
  for i = 1:V
    d = DATA(i,:);
    if (sum(d)==N-k)
      d = 2.-d;                         % to tabloid mode
      n = double(set_to_index(d));              % the tabloid's number
      data(n) = data(n)+1;              % creates the data vector
    end
  end
  D{k} = data;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [D, P] = readpart(kata)

%readpart will take in a vector of assignments to partition and return a cell of
%partitions need for the function partition. It will also trim off the index 1st row.

L = size(kata,2);
K = kata(1,:);
N = max(K);
P = cell(N+1,1);


for i = 0:N
   g =[];
   for j = 1:L
      if K(j) == i
         g= [g j];
      end
   end
   P{i+1,1} = g;
end
```

```
D = kata(2:size(kata,1),:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [D, PP] = partition(DATA, P)

%partion(D,P)
%
%partion should taken in a set of raw data and "collapse" the data
%by combining the "voters" into disjoint classes. For now let P be a k x 1 cell,
%where each cell is a vector who's entries are the members of ki's group.  Notice
%that |ki|= # of dudes in that group.

N = size(DATA,2); % number of voters
k = size(P,1); % how many different classes.
D = zeros(size(DATA,1),k);
PP = P';
for i=1:k
   Sum = zeros(size(DATA,1),1);
   u = size(P{i,1},2);
   for j=1:u
      Sum(:,1) = Sum(:,1) + DATA(:,P{i,1}(j));
   end
   D(:,i) = Sum/u;

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [D] = UltraSorting(DATA)

%Ultra big sorting. This combines the good and bad tabloids into one data vector.
%Big Sorting. It takes in a set of data of equivalence class votes so the
%values of DATA can range between 0-1. It will look at a row and distribute the
%weight for each tabloid and keep seperate the bad and good tabloids.

N = size(DATA,2);              % number of objects
M = size(DATA,1);                      % number of different situations
DD = zeros(1,2^N);
K = floor(N/2);                     % the most number of "losers"
FD = fliplr(DATA);  %to keep index correct on binary tree
GT = cell(1,K);                     % where we store the output for "good tabloids"
BT = cell(1,K);            %where we store the "bad tabloids"
D = cell(1,K);              %where we combine both data sets into one

for i=1:M
   L=[1- FD(i,1), FD(i,1)];

   for j=2:N;
      L1 = [(1-FD(i,j))*L, FD(i,j)*L]; %where entry is prob vote in majority (1)
      L= L1;
   end

   DD = DD + L;   %DD is a vector of sums of all weights for
end               %each tabloid shape good and bad

for k=1:K
   data = sparse(choose(k,N),1);       % what we'll eventually store
   bdata= sparse(choose(k,N),1);       %eventually store Bad Tabloids.
   for j=1:choose(k,N)                 %grab a tabloid shape in order and
      B = 0;                           %find its binary rep and that number
      bB = 0;                          %bB is for bad tabloids.
```

```
      q = 2.-index_to_set(j,[N-k,k]);   %gives us what B index from DD to put in
                bq= -1.+index_to_set(j,[N-k,k]); %that particular spot in data
        for uu=1:N
        B1 = 2^(N - uu)*q(uu);
        B = B1 + B;
        B2 = 2^(N - uu)*bq(uu);
        bB = B2 + bB;
      end
      data(j) = DD(B+1);           %-1 for shift over.
      bdata(j) = DD(bB+1);
  end
  GT{k} = data;
  BT{k} = bdata;
end

if floor(N/2) == (N/2)             %Take care of issues with even and odd numbers
   for a=1:(N/2 -1);               %If even, we don't want to add back last guy
      D{a} = GT{a}+BT{a};          %because its the same in bt as in gt
   end
D{N/2}  = GT{N/2};
else
   for a=1:floor(N/2);
      D{a} = GT{a}+BT{a};
   end
end


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function newset = index_to_set2(index, shapeVector)
% index_to_set2(number,shape)
% Maps to tabloid given integer index and a vector containing the shape of
% the tabloid.  This vector is defined in accordance with the following:
% 1 chosen from 4 chosen from 5 chosen from 9 yields rows of width
% 9-5, 5-4, 4-1, 1 respectively.  So the vector would be passed as
% [4 1 3 1]


index=index-1;                 %program originally written for indecies starting at 0.
                               %now they will start at 1
setSize = 0;
for counter=shapeVector
     setSize = setSize + counter;   %set setSize to the size of the array
end

newset = zeros(1, setSize);

nextSize=setSize;

for i = 1:1:length(shapeVector)
     internalSum = 0;                %the working sum just for this line
     selectedSize = shapeVector(i);  %these two vars are used for
     workingSize = nextSize;         %actual calculations

     nextSize = nextSize - shapeVector(i);    %saved for next iteration
     base = multiplier(shapeVector, i+1, nextSize);
     internalIndex = floor(index/base);        %to serve as this steps index
     index = index - (internalIndex * base);   %saved for next iteration

     for j = 1:1:setSize             %the workhorse loop.  loops through elements
                                     %and sets appropriate ones to current line
                                     %in the tabloid
```

```
          if newset(j) == 0                  %only look at unplaced elements
               if (selectedSize == 0)        %only look while we have space
                   workingSize = workingSize - 1;
               elseif (internalSum + choose(selectedSize, workingSize - 1) > internalIndex)
                   workingSize = workingSize - 1;
               else
               internalSum = internalSum + choose(selectedSize, workingSize - 1);
                   newset(j) = i;
                   workingSize = workingSize - 1;
                   selectedSize = selectedSize - 1;
               end
          end
     end
end


%-------------------------------------------------------

function answer = multiplier(countedSet, threshhold, size)
%multiplier.  Computes the base for the next level in the tabloid above
%the one passed to it as the threshhold.  Size is the number of elements
%on level of threshhold or below


answer = factorial(size);
for q = threshhold:1:length(countedSet)
     answer = answer / factorial(countedSet(q));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function index = set_to_index(setArray)
% set_to_index(tabloid)
% Maps from multiple-deep tabloid or set to an integer index of possible
% sets of that shape.

index = 1;

for p = 1:1:length(setArray)
    setArray(p) = setArray(p) + 1;
                  %this is unfortunately necessary since MATLAB won't index
end               %vector elements to 0.  0 will become 1 with this.

highestLevel = max(setArray);
setSize = length(setArray);

for i = 1:1:highestLevel
     countset(i) = count(setArray, i);
     countsetfact(i) = factorial(countset(i));
end

for levelCounter = 1:1:highestLevel-1
     workingSize = setSize;                %size still in use
     thisLevel =  countset(levelCounter);   %size selected in this level
     multiplier =
     factorial(workingSize - thisLevel)/factProduct(countset, levelCounter + 1, countsetfact);
     for k = setArray
         if k > levelCounter
              workingSize = workingSize - 1;
         elseif k == levelCounter
              index = index + (choose(thisLevel, workingSize - 1) * multiplier);
              workingSize = workingSize - 1;
              thisLevel = thisLevel - 1;
```

```
        end
    end
    setSize = setSize - countset(levelCounter);
end


function product = factProduct(countedSet, threshhold, countsetfact)
%factProduct
%multiplies the factorials of all elements in countedSet that are > threshhold

product = 1;
for q = threshhold:1:length(countedSet)
    product = product * countsetfact(q);     %factorial(countedSet(q));
end
```

B.2 FinalDecomposition.m

```
function [B,Q,q,R] = FinalDecomposition(D, JJ)

% FinalDecomposition(D)
%
% takes in data D = iwasaki(DATA) and returns the corresponding
% isotypic projections.  It then returns "stuff". In particular
% q is the raw projections, Q is the projections filtered using the
% up operator. R is the sorted values with the particular tabloid appended for each
% value.
%JJ are a cell of precomputed matrices for projrplus!!
%
% Author: David Uminsky, Spring 2003, modification of scdecompx.m

N = size(D{1},1);              % number of objects
K = size(D, 2);                % number of different situations
Q = cell(K,K+1);
R = cell(K,K+1);
q = cell(K,K+1);
B = cell(K,K+1);


%=================================================
if nargin < 2,
   for k = 1:K
  [L,P] = EMMYRk([N-k,k],D{k});  % our friend EMMYR, updated with k
  %---k,k case
q{k,k} = P(:,k+1);
Q{k,k} = P(:,k+1);
B{k,k} = L(1,F);

%---k,k+1 case
q{k,K+1} = P(:,1);
Q{k,K+1} = P(:,1);
B{k,K+1} = L(1,F);

%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
    U = up([N-k+l,k-l])*U;
```

```
  end
  q{k,k+1-j} = P(:,k+1+1-j) ;
  Q{k,k+1-j} = U*P(:,k+1+1-j)./factorial(j);  %NEED TO DIVIDE TO NORMALIZE
  B{k,k+1-j} = L(1,F);

end
end

else

   for k = 1:K
      [L,P] = PROJRplus(JJ{k},D{k});  % Projrplus takes in the johnson graphs of the
                                      % appropriate split and goes to work!


% ALL this find calling is to keep the projections index correctly.  We use the
% eigenvalues in L to do this. for the johnson graphs the formula for the jth
% projection in the kth order split is Ej = (k-j)(N-k-j)-j;

%---k,k case
[II,F] = find(L(2,:)>=-k-.00001 & L(2,:)< (-k+.00001));

if isempty(F) == 0

q{k,k} = P(:,F);
Q{k,k} = P(:,F);
B{k,k} = L(1,F);

else
q{k,k} = 0;
Q{k,k} = 0;
B{k,k} = 0;

end

%---k,k+1 case
[II,F] = find(L(2,:)>= (k*(N-k))-.00001 & L(2,:)< ((k*(N-k)) +.00001));

if isempty(F) == 0

q{k,K+1} = P(:,F);
Q{k,K+1} = P(:,F);
B{k,K+1} = L(1,F);

else
   q{k,K+1} = 0;
   Q{k,K+1} = 0;
   B{k,K+1} = 0;
end

%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
    U = up([N-k+l,k-l])*U;
 end

 H = ((k-(k+1-j))*(N-k-(k+1-j))-(k+1-j));

 [II,F] = find(L(2,:) >= (H-.00001)  & L(2,:) < (H +.00001));
 if isempty(F) == 0
```

```
 [LL,PP] = PROJRplus(JJ{k},U(1,:)');
 [III,FF] = find(LL(2,:) >= (H-.00001)  & LL(2,:) < (H +.00001));

 q{k,k+1-j} = P(:,F);  %This normalizes our projections
 Q{k,k+1-j} = U*P(:,F)./(factorial(j-1));  %NEED TO DIVIDE TO NORMALIZE
 B{k,k+1-j} = L(1,F);

else
 q{k,k+1-j} = 0;
 Q{k,k+1-j} = 0;
 B{k,k+1-j} = 0;
end


end
end
end


%=================================================

for i=1:K
  for j=1:i
    [Y,I] = sort(Q{i,j});
    sze = size(Y,1);
    T = zeros(sze,N);
    for l=1:sze
      T(l,:) = index_to_set(I(l),[N-j,j]);
    end
    R{i,j} = [Y,T];
  end
end


%=================================================
%=================================================
```

B.3 FinalDecompositionA.m

```
function [B,Q,q,R] = FinalDecompositionA(D, JJ)

% FinalDecompositionA(D)
%
% takes in data D = iwasaki(DATA) and returns the corresponding
% isotypic projections.  It then returns "stuff". In particular
% q is the raw projections, Q is the projections filtered using the
% up operator. R is the sorted values with the particular tabloid
%appended for each value.
%JJ are a cell of precomputed matrices for projrplus!!
%
% Author: David Uminsky, Spring 2003, modification of scdecompx.m

N = size(D{1},1);              % number of objects
```

```
K = size(D, 2);                % number of different situations
Q = cell(K,K+1);
R = cell(K,K+1);
q = cell(K,K+1);
B = cell(K,K+1);


%=================================================
if nargin < 2,
   for k = 1:K
  [L,P] = EMMYRk([N-k,k],D{k});  % our friend EMMYR, updated with k
  %---k,k case
q{k,k} = P(:,k+1);
Q{k,k} = P(:,k+1);
B{k,k} = L(1,k+1);

%---k,k+1 case
q{k,K+1} = P(:,1);
Q{k,K+1} = P(:,1);
B{k,K+1} = L(1,1);

%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
    U = up([N-k+l,k-l])*U;
  end
  q{k,k+1-j} = P(:,k+1+1-j) ;
  Q{k,k+1-j} = U*P(:,k+1+1-j)./factorial(j);  %NEED TO DIVIDE TO NORMALIZE
  B{k,k+1-j} = L(1,k+1+1-j);

end
end

else

   for k = 1:K
      [L,P] = PROJRplus(JJ{k},D{k});  % Projrplus takes in the johnson graphs of the
                                      % appropriate split and goes to work!
     if isempty(L) == 1;
     elseif isempty(L) == 0;
     XX = norm(L(1,:)')^2;

% ALL this find calling is to keep the projections index correctly.  We use the
% eigenvalues in L to do this. for the johnson graphs the formula for the jth
% projection in the kth order split is Ej = (k-j)(N-k-j)-j;

%---k,k case
[II,F] = find(L(2,:)>=-k-.00001 & L(2,:)< (-k+.00001));

if isempty(F) == 0

q{k,k} = P(:,F)./L(1,F);
Q{k,k} = P(:,F)./L(1,F);
B{k,k} = L(1,F)^2/XX;
else
q{k,k} = 0;
Q{k,k} = 0;
B{k,k} = 0;
end

%---k,k+1 case
[II,F] = find(L(2,:)>= (k*(N-k))-.00001 & L(2,:)< ((k*(N-k)) +.00001));
```

```
if isempty(F) == 0

   q{k,K+1} = P(:,F)./L(1,F);
   Q{k,K+1} = P(:,F)./L(1,F);
   B{k,K+1} = L(1,F)^2/XX;
else
   q{k,K+1} = 0;
   Q{k,K+1} = 0;
   B{k,K+1} = 0;
end

%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
    U = up([N-k+l,k-l])*U;
 end
 H = ((k-(k+1-j))*(N-k-(k+1-j))-(k+1-j));

 [II,F] = find(L(2,:) >= (H-.00001)  & L(2,:) < (H +.00001));
 if isempty(F) == 0

 [LL,PP] = PROJRplus(JJ{k},U(1,:)');
 [III,FF] = find(LL(2,:) >= (H-.00001)  & LL(2,:) < (H +.00001));


 q{k,k+1-j} = P(:,F)./L(1,F);  %This normalizes our projections
 Q{k,k+1-j} = U*P(:,F)./(factorial(j-1)*L(1,F)*LL(1,FF));  %NEED TO DIVIDE TO NORMALIZE
 B{k,k+1-j} = L(1,F)^2/XX;
else
 q{k,k+1-j} = 0;
 Q{k,k+1-j} = 0;
 B{k,k+1-j} = 0;
end
end
end
end
end

%=================================================

for i=1:K
  for j=1:i
    [Y,I] = sort(Q{i,j});
    sze = size(Y,1);
    T = zeros(sze,N);
    for l=1:sze
      T(l,:) = index_to_set(I(l),[N-j,j]);
    end
    R{i,j} = [Y,T];
  end
end


%=================================================
%=================================================
```

B.4 FinalStats.m

```
function [F, T] = FinalStats(senators,Partition,trials, JJ,V)

% Takes in the number of Bills, senators, the partion on the senate, the number
% of trial samples, the Johnson matrices (JJ), and a vector V that dictates how
% the data was distributed over the homogeneous spaces. Returns the result of
% sampling from uniformity, given V. F will contain entry values that correspond
% to the 90 percentile. T will actually be the concatenated trials.
%
% Author: David Uminsky, Spring 2003

M = sum(V);                          % the number of votes
%N = senators;              % the number of voters
%K = floor(N/2);                          % the most number of "losers"
w = max(Partition)+1;
z = floor(w/2);
T = cell(z,z+1);
F = cell(z,z+1);


for j=1:trials;
   j
   tic
   [B] = sample(senators,Partition,JJ, V);

  for i=1:z;
     for k=1:z+1;
        T{i,k} =[T{i,k}  B{i,k}];
     end
  end
  toc
end

for ii=1:z;
   for jj=1:z+1;
      if isempty(T{ii,jj}) == 0
         [A,S] = Ave90A(T{ii,jj},1);
         F{ii,jj} = [A,S];
      else
         F{ii,jj} = [];
      end
   end
end


function [B] = sample(senators,Partition,JJ, V)

% Sample grabs random data rows till its got the same number as Bills.
% sample also takes in a horizontal vector V which is a vector of length
% Floor(senators/2) whose entries are the number of bills that had that
% exact split.  THis lets us sample from the correct homogeneous spaces.
% To generate V, iwasaki the original data and sum each cell to get value.
% It then tacks on the horizontal partition vector and sends it
% to finalDataRetrieval.

DATA1= [];

                                  % the number of "votes" in uniformity Matrix
K = floor(senators/2);
C = senators;
```

```
% Using the so-called Fisher-Yates shuffle which can be found at
% http://theoryx5.uwinnipeg.ca/CPAN/perl/pod/perlfaq4/
% How_do_I_shuffle_an_array_randomly.html
%
% This method has been proven to produce uniformly random shuffles :)
% Thanks to Nate Eldredge for the help on finding this algorithm

                                    % the number of "votes" in uniformity Matrix


for i=1:K;
   if V(i) == 0;
   else
      for j=1:V(i);
         X = [zeros(1,i) ones(1,senators-i)];
         for ii=1:C;
            d = ceil((C+1-ii)*rand(1));
            a = X(1,d);
            X(1,d) = X(1, C+1-ii);
            X(1, C+1-ii) = a;
         end
         DATA1 = [DATA1 X'];
      end
   end
end

DATA1 = [Partition' DATA1]';

[DD] = FinalDataRetrieval(DATA1, 1);


[B] = StatDecomposition(DD, JJ);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [A,S] = Ave90A(Q,i);

% Ave90 will take in a vector Q and return the average A which we will
% assume to be zero and return N which is the value to beat, i.e. the value
% that is in the 90th percentile.

A = mean(Q(i,:));
%A = sum(Q(i,:))/size(Q,2);
%CC = [sort(abs(Q(i,:)))];
%N = CC(1,round(.90*size(Q(i,:),2)));
S = std(Q(i,:));
```

B.5 FinalStatsA.m

```
function [F, T] = FinalStatsA(senators,Partition,trials, JJ,V)

% Takes in the number of Bills, senators, the partion on the senate, the number
% of trial samples, the Johnson matrices (JJ), and a vector V that dictates how
% the data was distributed over the homogeneous spaces. Returns the result of
% sampling from uniformity, given V. F will contain entry values that correspond
% to the 90 percentile. T will actually be the concatenated trials.
%
% Author: David Uminsky, Spring 2003
```

```
M = sum(V);                          % the number of votes
%N = senators;              % the number of voters
%K = floor(N/2);                             % the most number of "losers"
w = max(Partition)+1;
z = floor(w/2);
T = cell(z,z+1);
F = cell(z,z+1);


for j=1:trials;
   j
   tic
   [B] = sample(senators,Partition,JJ, V);

  for i=1:z;
     for k=1:z+1;
        T{i,k} =[T{i,k}  B{i,k}];
     end
  end
  toc
end

for ii=1:z;
   for jj=1:z+1;
      if isempty(T{ii,jj}) == 0
         [A,S] = Ave90A(T{ii,jj},1);
         F{ii,jj} = [A,S];
      else
         F{ii,jj} = [];
      end
   end
end


function [B] = sample(senators,Partition,JJ, V)

% Sample grabs random data rows till its got the same number as Bills.
% sample also takes in a horizontal vector V which is a vector of length
% Floor(senators/2) whose entries are the number of bills that had that
% exact split.  THis lets us sample from the correct homogeneous spaces.
% To generate V, iwasaki the original data and sum each cell to get value.
% It then tacks on the horizontal partition vector and sends it
% to finalDataRetrieval.

DATA1= [];

                                     % the number of "votes" in uniformity Matrix
K = floor(senators/2);
C = senators;

% Using the so-called Fisher-Yates shuffle which can be found at
% http://theoryx5.uwinnipeg.ca/CPAN/perl/pod/perlfaq4/
% How_do_I_shuffle_an_array_randomly.html
%
% This method has been proven to produce uniformly random shuffles :)
% Thanks to Nate Eldredge for the help on finding this algorithm

                                     % the number of "votes" in uniformity Matrix


for i=1:K;
   if V(i) == 0;
   else
```

```
      for j=1:V(i);
         X = [zeros(1,i) ones(1,senators-i)];
         for ii=1:C;
            d = ceil((C+1-ii)*rand(1));
            a = X(1,d);
            X(1,d) = X(1, C+1-ii);
            X(1, C+1-ii) = a;
         end
         DATA1 = [DATA1 X'];
      end
   end
end

DATA1 = [Partition' DATA1]';

[DD] = FinalDataRetrieval(DATA1, 1);


[B] = StatDecompositionA(DD, JJ);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [A,S] = Ave90A(Q,i);

% Ave90 will take in a vector Q and return the average A which we will
% assume to be zero and return N which is the value to beat, i.e. the value
% that is in the 90th percentile.

A = mean(Q(i,:));
%A = sum(Q(i,:))/size(Q,2);
%CC = [sort(abs(Q(i,:)))];
%N = CC(1,round(.90*size(Q(i,:),2)));
S = std(Q(i,:));
```

B.6 FStatsNP.m

```
function [F, T] = FStatsNP(senators,trials, JJ,V)

% Takes in the number of Bills, senators, the partion on the senate, the number
% of trial samples, the Johnson matrices (JJ), and a vector V that dictates how
% the data was distributed over the homogeneous spaces. Returns the result of
% sampling from uniformity, given V. F will contain entry values that correspond
% to the 90 percentile. T will actually be the concatenated trials.
%
% Author: David Uminsky, Spring 2003

M = sum(V);;                % the number of votes
N = senators;               % the number of voters
K = floor(N/2);                     % the most number of "losers"
%w = max(Partition)+1;
z = floor(N/2);
T = cell(z,z+1);
F = cell(z,z+1);


for j=1:trials;
   j
   tic
  [B] = GsampleNP(senators,JJ, V);
```

```
  for i=1:z;
     for k=1:z+1;
        T{i,k} =[T{i,k}  B{i,k}];
     end
  end
  toc
end

for ii=1:z;
   for jj=1:z+1;
      if isempty(T{ii,jj}) == 0
         [A,S] = Ave90A(T{ii,jj},1);
         F{ii,jj} = [A,S];
      else
         F{ii,jj} = [];
      end
   end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [B] = GsampleNP(senators,JJ, V)

% Sample grabs random data rows till its got the same number as Bills.
% sample also takes in a horizontal vector V which is a vector of length
% Floor(senators/2) whose entries are the number of bills that had that
% exact split.  THis lets us sample from the correct homogeneous spaces.
% To generate V, iwasaki the original data and sum each cell to get value.
% It then tacks on the horizontal partition vector and sends it
% to finalDataRetrieval.

DATA1= [];

                                     % the number of "votes" in uniformity Matrix
K = floor(senators/2);
C = senators;

% Using the so-called Fisher-Yates shuffle which can be found at
% http://theoryx5.uwinnipeg.ca/CPAN/perl/pod/perlfaq4/
% How_do_I_shuffle_an_array_randomly.html
%
% This method has been proven to produce uniformly random shuffles :)
% Thanks to Nate Eldredge for the help on finding this algorithm

                                     % the number of "votes" in uniformity Matrix


for i=1:K;
   if V(i) == 0;
   else
      for j=1:V(i);
         X = [zeros(1,i) ones(1,senators-i)];
         for ii=1:C;
            d = ceil((C+1-ii)*rand(1));
            a = X(1,d);
            X(1,d) = X(1, C+1-ii);
            X(1, C+1-ii) = a;
         end
         DATA1 = [DATA1 X'];
      end
   end
end
```

```
%DATA1 = [Partition' DATA1]';

[DD] = FinalDataRetrieval(DATA1');


[B] = FDecompA(DD, JJ);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [B] = FDecompA(D, JJ)

% FinalDecomposition(D)
%
% takes in data D = iwasaki(DATA) and returns the corresponding
% isotypic projections.  It then returns "stuff". In particular
% q is the raw projections, Q is the projections filtered using the
% up operator. R is the sorted values with the particular tabloid appended for each
% value.
%JJ are a cell of precomputed matrices for projrplus!!
%
% Author: David Uminsky, Spring 2003, modification of scdecompx.m

N = size(D{1},1);              % number of objects
K = size(D, 2);                % number of different situations
Q = cell(K,K+1);
R = cell(K,K+1);
q = cell(K,K+1);
B = cell(K,K+1);


%=====================================================
if nargin < 2,
   for k = 1:K
  [L,P] = EMMYRk([N-k,k],D{k});  % our friend EMMYR, updated with k
  %---k,k case
%q{k,k} = P(:,k+1);
%Q{k,k} = P(:,k+1);
B{k,k} = L(1,k+1);

%---k,k+1 case
%q{k,K+1} = P(:,1);
%Q{k,K+1} = P(:,1);
B{k,K+1} = L(1,1);

%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
%    U = up([N-k+l,k-l])*U;
  end
 % q{k,k+1-j} = P(:,k+1+1-j) ;
  %Q{k,k+1-j} = U*P(:,k+1+1-j)./factorial(j);  %NEED TO DIVIDE TO NORMALIZE
  B{k,k+1-j} = L(1,k+1+1-j);

end
end

else

   for k = 1:K
      [L,P] = PROJRplus(JJ{k},D{k});  % Projrplus takes in the johnson graphs of the
                                      % appropriate split and goes to work!
```

```
     if isempty(L) == 1;
     elseif isempty(L) == 0;
     %XX = norm(L(1,:))^2;

% ALL this find calling is to keep the projections index correctly.  We use the
% eigenvalues in L to do this. for the johnson graphs the formula for the jth
% projection in the kth order split is Ej = (k-j)(N-k-j)-j;

%---k,k case
[II,F] = find(L(2,:)>=-k-.00001 & L(2,:)< (-k+.00001));
if isempty(F) == 0
   %q{k,k} = P(:,F)./L(1,F);
   %Q{k,k} = P(:,F)./L(1,F);
   B{k,k} = L(1,F)^2;
else
   B{k,k} = 0;

end

%---k,k+1 case
[II,F] = find(L(2,:)>= (k*(N-k))-.00001 & L(2,:)< ((k*(N-k)) +.00001));
if isempty(F) == 0
%q{k,K+1} = P(:,F)./L(1,F);
%Q{k,K+1} = P(:,F)./L(1,F);
B{k,K+1} = L(1,F)^2;
else
   B{k,K+1} = 0;
end
%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
    U = up([N-k+l,k-l])*U;
 end
 H = ((k-(k+1-j))*(N-k-(k+1-j))-(k+1-j));

 [II,F] = find(L(2,:) >= (H-.00001)  & L(2,:) < (H +.00001));
 if isempty(F) == 0
% q{k,k+1-j} = P(:,F)./L(1,F);  %This normalizes our projections
% Q{k,k+1-j} = U*P(:,F)./(factorial(j)*L(1,F));  %NEED TO DIVIDE TO NORMALIZE
B{k,k+1-j} = L(1,F)*L(1,F);
else
   B{k,k+1-j} = 0;
end
end
end
end
end

%=================================================
%=================================================


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [A,S] = Ave90A(Q,i);

% Ave90 will take in a vector Q and return the average A which we will
% assume to be zero and return N which is the value to beat, i.e. the value
% that is in the 90th percentile.

A = mean(Q(i,:));
%A = sum(Q(i,:))/size(Q,2);
```

```
%CC = [sort(abs(Q(i,:)))];
%N = CC(1,round(.90*size(Q(i,:),2)));
S = std(Q(i,:));
```

B.7 GenerateJohnson.m

```
function [M] = GenerateJohnson(N);

% Generate Johnson takes in a value N and computes the adjacency matrices for the
% (N choose k) johnson graphs k = 1...floor(N/2) and returns a cell of the matrices
% M.
%
% Author: David Uminsky, Spring 2003

K = floor(N/2);

M = cell(1, K);

for i=1:K;
   tic
   M{1,i} = Johnson([N-i,i]);
   toc
end


function [A] = Johnson(S);

% Johnson returns the adjacency matrix of the Johnson graph for the particular split
% [N,k]. Such as (7,1),(6,2)...
%
% Author: David Uminsky, January 2003

N = S(1)+S(2);  % the total number elements in the set

K= choose(S(2),N);  %size of matrix
A = sparse(K,K);

for i=1:K
   for j=1:K
      if sum(abs((2.-index_to_set(i,S))-(2.-index_to_set(j,S)))) == 2
         A(i,j) = 1;
      else
         A(i,j) = 0;
      end
   end
end
```

B.8 sample.m

```
function [Q,q,R] = sample(senators,Partition,JJ, V)

% Sample grabs random data rows till its got the same number as Bills.
% sample also takes in a horizontal vector V which is a vector of length
% Floor(senators/2) whose entries are the number of bills that had that
% exact split.  THis lets us sample from the correct homogeneous spaces.
```

```
% To generate V, iwasaki the original data and sum each cell to get value.
% It then tacks on the horizontal partition vector and sends it
% to finalDataRetrieval.
%
% Author: David Uminsky, Spring 2003

DATA1= [];

                                    % the number of "votes" in uniformity Matrix
K = floor(senators/2);

for i=1:K;
   if V(i) == 0;
   else
      for j=1:V(i);
         X = 2.-index_to_set(ceil(choose(i,senators)*rand(1)),[senators-i,i])';
         DATA1 = [DATA1 X];
      end
   end
end

DATA1 = [Partition' DATA1]';

[DD] = FinalDataRetrieval(DATA1, 1);


[Q,q,R] = FinalDecomposition(DD, JJ)
```

B.9 Up.m

```
function R = up(lam)

% R = up(lam)
%
% returns the Radon transform R: M^lam --> M^lam+ where lam is a
% partition of an integer n.
% Author: Michael Orrison


r = size(lam,2);                                % the number of rows of lam
s = sum(lam);                        % the number we're partitioning

new_lam  = lam;                                 % the new partition
new_lam(1) = lam(1)+1;
new_lam(r) = lam(r)- 1;

M = bloids(lam); m = size(M,1);         % where we're starting
N = bloids(new_lam); n = size(N,1);     % where we're going

R = sparse(n,m);                                % the Radon transform

for i = 1:m
    count = 0;
    j = 1;
    temp = zeros(1,s);
    while count<lam(r)
        if M(i,j)==r
            temp = M(i,:);
            temp(j) = 1;
```

```
            ind = set_to_index(temp);
            R(ind,i) = R(ind,i) + 1;
            count = count + 1;
        end
        j = j+1;
    end
end
```

B.10 down.m

```
function R = down(lam)

% R = down(lam)
%
% returns the Radon transform R: M^lam --> M^lam- where lam is a
% partition of an integer n.
% Author: Michael Orrison


r = size(lam,2);                                % the number of rows of lam
s = sum(lam);                        % the number we're partitioning

new_lam  = lam;                                 % the new partition
new_lam(1) = lam(1)-1;
new_lam(r) = lam(r)+1;

M = bloids(lam); m = size(M,1);         % where we're starting
N = bloids(new_lam); n = size(N,1);     % where we're going

R = sparse(n,m);                                % the Radon transform

for i = 1:m
    count = 0;
    j = 1;
    temp = zeros(1,s);
    while count<lam(1)
        if M(i,j)==1
            temp = M(i,:);
            temp(j) = r;
            ind = set_to_index(temp);
            R(ind,i) = R(ind,i) + 1;
            count = count + 1;
        end
        j = j+1;
    end
end
```

B.11 bloids.m

```
function M = bloids(mu)

%BLOIDS(mu) takes vector mu of positive integers and
%returns a matrix whose rows correspond to the
%tabloids of shape mu.
% Author: Michael Orrison


%tic
mu = sort(mu);               %the "right" order
sze = size(mu,2);            %the number of rows
summu = sum(mu);

num = factorial(summu);
for i = 1:size(mu,2)
 num = num/factorial(mu(i)); %the number of tabloids of shape mu
end
M = ones(num,sum(mu));       %the setup

if sze == 1                  %...then we're all set.

else
 count = 1;
 lam = mu;
 lam(:,sze) = [];            %remove the "top row"
 sumlam = sum(lam);          %the number of "boxes" left
 K = kset(summu,sumlam);     %find the k-sets for the lower rows
 J = 1 + bloids(lam);        %we use bloids here!
  for k = 1:size(K,1)        %for each k-set...
   for j = 1:size(J,1)        %...get new tabloid of shape lam...
    temp = ones(1,summu);
     for i = 1:sumlam
      temp(K(k,i)) = J(j,i);   %...create new tabloid of shape mu...
     end
    M(count,:) = temp;        %...then add it to the list.
    count = count + 1;
   end
  end
end
%toc
```

B.12 choose.m

```
function permutations = choose(X, Y)
% choose
% returns the number of non-ordered permutations of X items chosen from Y
% Author: Michael Orrison

if (X > Y)
     permutations=0;
elseif (X == 0 & Y == 0)
```

```
     permutations=0;
else
     permutations = factorial(Y) / (factorial(Y-X) * factorial(X));
end
```

B.13 count.m

```
function counter = count(set, number)
%count
%Takes a vector and a number, and returns the number of occurances of the
%number in that set.
% Author: Michael Orrison


counter = 0;

for i = set
    if (i == number)
        counter = counter + 1;
    end
end
```

B.14 Emmyrk.m

```
function [L,P] = EMMYRk(lambda,v,Rs)

% [L,P]=EMMYR(lambda,v)
%
% does what BUDR does but is a little more clever with the
% JM-elements and takes advantage of the branching rules
% associated to the representations of the symmetric group.
%
% [L,P]=EMMYR(lambda,v,Rs) does the above, but takes advantage
% of the precomputed JM elements in Rs.
% Author: Michael Orrison


%tic

if nargin < 3, Rs = JMs(bloids(lambda)); end

d = size(Rs,1);
n = size(Rs,2)/d + 1;
A = Rs(1:d,1:d);

%['Computing the projections of v onto the eigenspaces of R_2.']
[L,P] = PROJRplus(A,v);
[L,P] = gather(L,P);

for i = 3:n
  %['Computing the projections onto the eigenspaces of R_', num2str(i),'.']
  A = Rs(1:d,1+(i-2)*d:(i-1)*d);
  [L,P] = PROJRplus(A,P,L);
  [L,P] = gather(L,P);
end

% Ugly ordering procedure to make sure P lines up in order of: mean, 1st order, 2nd order...
```

```
[r c] = size(L);
U = [];
for i = 1:c
   for f = 1:c
      if L(r,f) == n - i, U(:,i) = P(:,f);
      end
   end
end
P = U;

%toc
```

B.15 gather.m

```
function [new_L,new_P] = gather(L,P)

% [new_L,new_P] = gather(L,P)
%
% ...uses the information in L (created using JODR
% or EMMYR) to rearrange and gather those projections in P
% that are in the same isotypic subspace under the restriction
% of the permutation representation in question to "the" subgroup
% S_m where m < n.
% Author: Michael Orrison

temp_P = P;
temp_L = L;
new_P  = [];
new_L  = [];
new_proj_lengths = [];

temp_L(1,:) = [];               %% Remove the lengths of the projections.
temp_L = round(temp_L);         %% Make sure we're dealing with integers.

if size(temp_L,1) > 1
   temp_L = sort(temp_L);       %% Sort the columns of L first.
end

check = 0;              %% So we know when we've finished comparing.
count = 1;              %% So we know where to start comparing.
coord = [];             %% So we can "gather" the information we need.
while check == 0
    coord = [];
    if count > size(temp_L,2)
       check = 1;
    elseif count == size(temp_L,2)
       new_proj_lengths = [new_proj_lengths norm(temp_P(:,count))];
       check = 1;
    else
       current_shape = temp_L(:,count);     %% The shape we're comparing.
       for i = count+1:size(temp_L,2)
           if current_shape == temp_L(:,i)  %% Find coordinates of those
              coord = [coord i];            %% projections with the same shape.
           end
       end
```

```
      A = temp_P(:,[count coord]);
      summed_proj = zeros(size(A,1),1);
      for i=1:size(A,2)
          summed_proj = summed_proj + A(:,i);
      end

      temp_P(:,count) = summed_proj;       %% Gather the info into the
      temp_L(:,count) = current_shape;     %% the right places.

      new_proj_lengths = [new_proj_lengths norm(summed_proj)];

      temp_P(:,coord) = [];                %% Get rid of the extra info
      temp_L(:,coord) = [];                %% that we have just gathered.
      count = count+1;
    end

end

new_L = [new_proj_lengths; temp_L];
new_P = temp_P;
```

B.16 iwasaki.m

```
function D = iwasaki(DATA)

% iwasaki(DATA)
%
% takes takes in a matrix DATA whose rows are consist of
% seqences of 0's and 1's, where each row has no more 0's
% than 1's.  The idea is to view the rows of DATA as the
% partitions of a set, where 1 means "winner" and 0 means
% "loser".  The output is the tabloid version of the incoming
% data in DATA.  This can then be run through BUDR or EMMYR
% to complete the spectral analysis.  Note: we also assume
% that there is at least one loser in each partition.
% Authors: Michael Orrison and David Uminsky, Fall 2002


V = size(DATA,1);                  % the number of votes
N = size(DATA,2);                  % the number of things to partition
K = floor(N/2);                    % the most number of "losers"
D = cell(1,K);                     % where we store the output

for k = 1:K,
  data = sparse(choose(k,N),1);         % what we'll eventually store
  for i = 1:V
    d = DATA(i,:);
    if (sum(d)==N-k)
      d = 2.-d;                         % to tabloid mode
      n = set_to_index(d);              % the tabloid's number
      data(n) = data(n)+1;              % creates the data vector
    end
  end
  D{k} = data;
end
```

B.17 kset.m

```
function M = kset(n,k)

%KSET(n,k) produces a matrix whose rows are the
%k-element subsets of an n-element set.
% Author: Michael Orrison

nchsk = nchoosek(n,k);

M = zeros(nchsk,k);

if (n < k | k <= 0)
 M = [];

elseif k==1
 M = [1:n]';

else
 J = kset(n-1,k-1);
 count = 1;
 for i = 1:size(J,1)
  for j = (J(i,k-1)+1):n
   M(count,:) = [J(i,:) j];
   count = count + 1;
  end
 end

end
```

B.18 JM.m

```
function R = JM(M,i)

%JM(M,i) returns the Jucys-Murphy element R_i associated with
%the space of tabloids given to us by the tabloid matrix M.
% Author: Michael Orrison

dim = size(M,1);     %the dimension of our tabloid space
R = sparse(dim,dim);
temp = zeros(1,size(M,2));
check = 0;

for k = 1:dim
  for j = 1:(i-1)
    if M(k,i)==M(k,j)
      R(k,k) = R(k,k) + 1;
    else
      temp = M(k,:);
      temp(j) = M(k,i);
      temp(i) = M(k,j);
%
%new
      temp_index = set_to_index(temp);
      R(temp_index,k) = R(temp_index,k) +1;
```

```
%old
%      check = 1;
%      count = 1;
%      while check==1
%        if temp==M(count,:)
%         R(count,k) = R(count,k) + 1;
%         check = 0;
%        else
%         count = count + 1;
%        end
%      end
    end
  end
end
```

B.19 JMs.m

```
function Rs = JMs(M)

% JMs(M)
%
% computes all of the Jucys-Murphy elements for the tabloid
% space given by M.  The result is the matrix [R_2 ... R_n].
% Author: Michael Orrison

n = size(M,2);
d = size(M,1);

Rs = sparse(d,(n-2)*d);
%tic
for i = 2:n
%  ['Computing R_',num2str(i),'.  Please be patient.']
  Rs(1:d,1+(i-2)*d:(i-1)*d) = JM(M,i);
end
%toc
```

B.20 lanr.m

```
function [Q,R] = lanr(A,f,epsilon)

% [Q,R] = lanr(A,f)
%
% uses the "Lanczos Iteration with re-orthogonaliztion" to compute the
% QR factorization of the matrix [f Af A^2f...] where A is a symmetric
% matrix.
%
% [Q,R] = lanr(A,f,espilon)
%
% also does the above but lets the user determine how small the residue
% vectors must be before terminating.  If lanr(A,f) is used, then epsilon
% is set to 10e-8.
% Author: Michael Orrison
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% tic

Q = f/norm(f);                      % our starting vector
R = sparse([]);                     % our tridiagonal matrix
a = []; b = []; ep = 0;             % vectors used in the iteration
check = 1; n = 1;                   % counters and checks
sze = size(A,2);                    % in case espsilon doesn't work
v = zeros(sze,1);                   % v is the main character

if nargin == 2                      % if we don't choose "small"
  small = 10e-8;
end

if nargin == 3                      % if we do choose "small"
  small = epsilon;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  v = A*Q(:,1);                     % this is our first pass to
  a(1) = Q(:,1)'*v;                 % see if we should continue and
  v = v - a(1)*Q(:,1);              % to work our way into the iteration
  R(1,1) = a(1);                    % portion of the algorithm
  b(1) = norm(v);
  if b(1) > small
   Q(:,2) = v/b(1);
   R(1,2) = b(1);
   R(2,1) = b(1);
   n = 2;
  else
   check = 0;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while check > 0                     % this is where the three-term
  v = A*Q(:,n) - b(n-1)*Q(:,n-1);   % recurrence appears
  a(n) = Q(:,n)'*v;
  v = v - a(n)*Q(:,n);
  R(n,n) = a(n);

  for j = 1:n                       % where we re-orthogonalize
   ep = Q(:,j)'*v;
   v = v - ep*Q(:,j);
  end
  b(n) = norm(v);
  if (b(n) > small & n < sze)
   Q(:,n+1) = v/b(n);
   R(n,n+1) = b(n);
   R(n+1,n) = b(n);
   n = n+1;
  else
   check = 0;
  end
end

% toc
```

B.21 Projrplus.m

```
function [L,P] = PROJRplus(A,X,Y)

% [L,P] = PROJRplus(A,X)
%
% uses the "Lanczos Iteration with re-orthogonalization" to compute
% the projections P of the column vectors of X onto the eigenspaces
% of the symmetric matrix A.  It also computes the lengths of each of
% these projections and returns a two-rowed matrix L with the length
% and the corresponding eigenvalue making up one column.
%
% [L,P] = PROJRplus(A,X,Y)
%
% also does the above, but keeps track of previous computations of
% eigenvalues (in Y) for iteration.
% Author: Michael Orrison


Q = []; R = []; P = []; L = []; U = []; D = [];
nrm = 0;

if nargin==2
  for i = 1:size(X,2)
    [Q,R] = lanr(A,X(:,i));       %the QR decomp of [x Ax...]
    R = full(R);  %Needed?        %to get the eig vals/vecs of R
    [U,D] = eig(R);               %the eig vals/vecs of R
    nrm = norm(X(:,i));           %to remind us of the size of x
    P = [P Q*U*diag(nrm*U(1,:))]; %the projections
    L = [L [nrm*abs(U(1,:)); ones(1,size(D,1))*D]]; %the lengths
  end

% Recall that the first row of U contains the lengths
% of the projections of x/||x|| onto the eigenvectors
% of R in the basis Q.  We also take advantage of the
% fact that the eigenvectors in U have length 1 to
% compute the lengths in L by using abs.

elseif nargin==3
  Y(1,:) = [];                    %forgets the old lengths
  for i = 1:size(X,2)
    [Q,R] = lanr(A,X(:,i));       %see above
    R = full(R);  %Needed?        %see above
    [U,D] = eig(R);               %see above
    nrm = norm(X(:,i));           %see above
    P = [P Q*U*diag(nrm*U(1,:))]; %see above
    L = [L [nrm*abs(U(1,:)); Y(:,i)*ones(1,size(D,1));
                               ones(1,size(D,1))*D   ]];  %see above
  end
end
```

B.22 readpart.m

```
function [D, P] = readpart(kata)

% readpart will take in a vector of assignments to partition and return a cell of
% partitions need for the function partition. It will also trim off the index 1st row.
% Author: David Uminsky, Fall 2002


L = size(kata,2);
K = kata(1,:);
N = max(K);
P = cell(N+1,1);


for i = 0:N
   g =[];
   for j = 1:L
      if K(j) == i
         g= [g j];
      end
   end
   P{i+1,1} = g;
end

D = kata(2:size(kata,1),:);
```

B.23 scdecompx.m

```
function [q,Q,R,B,G,E] = scdecompx(D)

% scdecomp(D)
%
% takes in data D = iwasaki(DATA) and returns the corresponding
% isotypic projections.  It then returns "stuff". In particular
% q is the raw projections, Q is the projections filtered using the
% up operator. R is the sorted values with the particular tabloid
% appended for each value. B is the sum of the squares divided by
% the dimension, G.
% Authors: Michael Orrison and David Uminsky, Fall 2002


N = size(D{1},1);              % number of objects
K = size(D, 2);                % number of different situations
Q = cell(K,K+1);
R = cell(K,K+1);
B = cell(K,K+1);
G = cell(K,K+1);
E = cell(K,K+1);
q = cell(K,K+1);


%=====================================================

for k = 1:K
  [L,P] = EMMYRk([N-k,k],D{k});  % our friend EMMYR, updated with k

%---k,k case
```

```
q{k,k} = P(:,k+1);
Q{k,k} = P(:,k+1);
B{k,k} = (norm(P(:,k+1)))^2/dim(N-k,k); % SS/dim -- k-th order effects
E{k,k} = dim(N-k,k)/choose(k,N);
G{k,k} = (norm(q{k,k})/norm(D{k}))^2;

%---k,k+1 case
q{k,K+1} = P(:,1);
Q{k,K+1} = P(:,1);
B{k,K+1} = (norm(P(:,1)))^2;            % SS/1 -- the mean response
E{k,K+1} = 1/choose(k,N);
G{k,K+1} = (norm(q{k,K+1})/norm(D{k}))^2;

%---k,j (j=2:k-1)
for j=2:k
  U = eye(choose(k,N));
  for l=0:j-2
    U = up([N-k+l,k-l])*U;
  end
  q{k,k+1-j} = P(:,k+1+1-j) ;
  Q{k,k+1-j} = U*P(:,k+1+1-j)./factorial(j);  %NEED TO DIVIDE TO NORMALIZE
  B{k,k+1-j} = (norm(P(:,k+1+1-j)))^2/dim(N-j,j); % SS/dim -- j-th order effects
  E{k,k+1-j} = dim(N-(j-1),j-1)/choose(k,N);
  G{k,k+1-j} = (norm(q{k,k+1-j})/norm(D{k}))^2;
end

for j=1:k-1
  E{k,j} = dim(N-j,j)/choose(k,N);
end

end

%=====================================================

for i=1:K
  for j=1:i
    [Y,I] = sort(Q{i,j});
    sze = size(Y,1);
    T = zeros(sze,N);
    for l=1:sze
      T(l,:) = index_to_set(I(l),[N-j,j]);
    end
    R{i,j} = [Y,T];
  end
end


%=====================================================
%=====================================================

function d = dim(a,b)

% d = dim(a,b)
%
% computes the dimension of simple module S^(a,b).

if b==0
  d = 1;
else
  d = choose(b,a+b);
  for i=0:(b-1)
     d = d - dim(a+b-i,i);
  end
```

```
end
```

B.24 ubersaki.m

```
function [GT, BT, DD] = ubersaki(DATA)

%Big iwasaki. It takes in a set of data of equivalence class votes so the
%values of DATA can range between 0-1. It will look at a row and distribute the
%weight for each tabloid and keep seperate the bad and good tabloids.
% Author: David Uminsky, Fall 2002

N = size(DATA,2)                % number of objects
M = size(DATA,1)                        % number of different situations
DD = zeros(1,2^N);
K = floor(N/2);                     % the most number of "losers"
FD = fliplr(DATA);  %to keep index correct on binary tree
GT = cell(1,K);                     % where we store the output for "good tabloids"
BT = cell(1,K); %where we store the "bad tabloids"

for i=1:M
   L=[1- FD(i,1), FD(i,1)];

   for j=2:N;
      L1 = [(1-FD(i,j))*L, FD(i,j)*L]; %where entry is prob vote yes (1)
      L= L1;
   end

   DD = DD + L;                         %DD is a vector of sums of all weights for
end                                     %each tabloid shape good and bad

for k=1:K
   data = sparse(choose(k,N),1);        % what we'll eventually store
   bdata= sparse(choose(k,N),1);        %eventually store Bad Tabloids.
   for j=1:choose(k,N)                  %grab a tabloid shape in order and
      B = 0;                            %find its binary rep and that number
      bB = 0;                           %bB is for bad tabloids.
      q = 2.-index_to_set(j,[N-k,k]);  %gives us what B index from DD to put in
                bq= -1.+index_to_set(j,[N-k,k]); %that particular spot in data
        for uu=1:N
        B1 = 2^(N - uu)*q(uu);
        B = B1 + B;
        B2 = 2^(N - uu)*bq(uu);
        bB = B2 + bB;
     end
     data(j) = DD(B+1);                 %-1 for shift over.
     bdata(j) = DD(bB+1);
  end
  GT{k} = data;
  BT{k} = bdata;
end
```

B.25 Vbuilder.m

```
function [V] = Vbuilder(DATA);

% This builds our V vector for FinalStats.m It takes in a matrix of
% zeros and ones and sorts them into how many bills had k zeros and
% so on. The senators are the columns and the rows are each bill.
% Author: David Uminsky, spring 2003

N = size(DATA,1); % number of bills
M = size(DATA,2); % number of senators;
K = floor(M/2);

V= zeros(1,K);

for i=1:N;
   a = sum(DATA(i,:));
   V(1,M-a) = V(1,M-a) +1;
end
```

Bibliography

[1] Laurel Beckett and Persi Diaconis. Spectral analysis for discrete longitudinal data. *Adv. Math.*, 103(1):107–128, 1994. This article provides the foundation for the statistics of spectral analysis.

[2] Persi Diaconis. *Group representations in probability and statistics*, volume 11 of *Institute of Mathematical Statistics Lecture Notes—Monograph Series*. Institute of Mathematical Statistics, Hayward, CA, 1988. This is an incredible book that lays down the theory for applying representation theory to analyzing data.

[3] Persi Diaconis. A generalization of spectral analysis with application to ranked data. *Ann. Statist.*, 17(3):949–979, 1989. This article carries out an example of the spectral analysis on both ranked and partially ranked data.

[4] Persi Diaconis and Bernd Sturmfels. Algebraic algorithms for sampling from conditional distributions. *Ann. Statist.*, 26(1):363–397, 1998. This article talks about applying Grobner basis to the question of statistcal analysis of data.

[5] David S. Dummit and Richard M. Foote. *Abstract Algebra*. Prentice Hall Inc., Englewood Cliffs, NJ, 1991. This was my algebra text book where I learned most of abstract algebra and is a good source for representation theory.

[6] E. J. Hannan. *Group representations and applied probability*. Methuen's Supplementary Review Series in Applied Probability, Vol. 3. Methuen & Co. Ltd., London, 1965. An old article that precedes Diaconis's book on the beginnings of generalized spectral analysis, It draws connections between spectral analysis and ANOVA.

[7] E. J. Hannan. Group representations and applied probability. *J. Appl. Probability*, 2:1–68, 1965. This is a technical paper that predates Persi Diaconis's work on group representations.

[8] Manabu Iwasaki. Spectral analysis of multivariate binary data. *J. Japan Statist. Soc.*, 22(1):45–65, 1992. Iwasaki carries out spectral analysis on a set of data using both the symetric group and $Z/2Z$.

[9] Brian Lawson, Michael Orrison, and David Uminsky. Noncommuntative harmonic analysis of voting in committees. *submitted*, pages 1–26, 2002. This is an article that we wrote from work done over the summer that focuses on the Supreme Court.

[10] David K. Maslen, Michael E. Orrison, and Daniel N. Rockmore. Computing isotypic projections witht the lanczos iteration. *submitted*, pages 1–21. This is an article that describes, in detail, the algorithms we use to compute the spectral analysis.

[11] Michael E. Orrison. *An eigenspace approach to decomposing representations of finite groups*. PhD thesis, Dartmouth College, 2001. Prof. Orrison's Thesis where he gives an excellent build up of the needed theory and explains the algorithms we are implementing to efficiently compute the projections.

[12] Keith T. Poole. `http://voteview.uh.edu/ideal_point_d_nominate.htm`. *D-NOMINATE*, 1986. Website that contains the Fortran code for D-NOMINATE.

[13] Keith T. Poole and Howard Rosenthal. A spatial model for legislative roll call analysis. *American Journal of Political Science*, 29(2):357–384, 1985. This

article is the seminal work done by Poole and Rosenthal on the spatial model approach.

[14] Daniel N. Rockmore. Some applications of generalized FFTs. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 329–369. Amer. Math. Soc., Providence, RI, 1997. This is a comprehensive discussion of spectral analaysis and generalized FFTs.

[15] Bruce E. Sagan. *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*. Number 203 in Graduate Texts in Mathematics. Springer-Verlag, second edition, 2001. Provides a very nice, in depth presentation of the irreducible reprsentations of the symmetric group.