ipe - r Package for Ideal Point Estimation

Michael Peress*

July 2021

*SUNY-Stony Brook, michael.peress@stonybrook.edu

1 Overview

ipe is an *r* package for ideal point estimation. The main algorithm optimizes a penalized maximum likelihood objective function using limited memory BFGS and a trust region line search. Standard errors are calculated using the asymptotic Hessian. The main algorithm works efficiently on dense and sparse data matrices and can be used to estimate models with a very large number of parameters. The package includes utilities for creating data matrices, estimating starting values, bridging together multiple chambers, and visualizing and interpreting results. The methods included in this package were primarily developed in Peress (forthcoming), though some methods were also developed in Peress and Spirling (2010), Battista, Peress and Richman (2013), Peress (2013), and Battista, Peress and Richman (Forthcoming).

1.1 Installation

ipe can be installed in *r* using the command:

```
source("https://sites.google.com/a/stonybrook.edu/mperess/install_ipe.r")
```

The package can the be loaded using the command:

```
library(ipe)
```

The most computationally intensive functions call *c++* code compiled into a dynamic-link library (DLL). Since DLLs are specific to the *Windows* operating system, these functions will not work directly in other operating systems. It is possible to run these functions using *Wine* on a different operating system, though this has only been tested on *Linux*.

1.2 Model and Notation

Assume that data takes the form of an $N \times T$ matrix y with elements y_{nt}, where n denotes individuals or voters and t denotes items or votes. Assume that $y_{nt} \in \{0, 1, 2\}$ where $y_{nt} = 2$ denotes a positive outcome or yea vote, $y_{nt} = 1$ denotes a negative outcome or nay vote, and $y_{nt} = 0$ denotes a missing outcome or vote. Each individual is characterized by $\alpha_n \in \mathbb{R}^D$ where $D \geq 1$ and each item is characterized by $a_t \in \mathbb{R}$ and $b_t \in \mathbb{R}^D$. Let $\delta_t = (a_t, b_t)$. Assume that conditional on observing an outcome (i.e. $y_{nt} \neq 0$), we observe $y_{nt} = 2$ with probability $\Phi(a_t + b_t'\alpha_n)$ and $y_{nt} = 1$ with probability $1 - \Phi(a_t + b_t'\alpha_n)$, where these probabilities are independent across n and t. To avoid confusion due to the similarity of the terms vote and voter, we will use the terms individual to refer to the n index and item to refer to the t index. Individuals parameters or ideal points refer to α_n and item parameters refer to (a_t, b_t) or δ_t.

2 Loading Data

2.1 Dense Data

ipe handles both dense and sparse data. Dense data is stored as a matrix. Each row of the matrix represents an individual and each column represents an item. The elements of the matrix are 2 (indicating a positive outcome), 1 (indicating a negative outcome), and 0 (indicating a missing outcome). The rows and columns of the matrix may be named, and if so, they should have unique names.

Example 1 (Roll Call Voting in the California House)**.**

This example loads data from the Representation in the America's Legislature Project (Wright, 2007). Since the data is stored in matrix form, it can be loaded directly as a matrix. The values representing a yea vote, nay vote, and missing vote can then be recoded to match the format used by *ipe*. The data matrix can be formed using the *r* code,

```
data1 <- read.delim(paste(dir1,"cah9900_rc.tab",sep=""))
Y <- as.matrix(data1[,6:2220]) # get the votes
Y <- Y + 1 # recode yea/nay
Y[is.na(Y)] <- 0 # recode NA
```

The rows and columns can be named as follows,

```
rownames(Y) <- paste(data1$first,data1$last,sep=" ")
colnames(Y) <- paste("CAH",1:ncol(Y),sep="")
```

Party can be saved for future use,

```
party <- data1$pid
party[party==0] <- "R"
party[party==1] <- "D"
party[party==2] <- "I"
```

2.2 Sparse Data

Sparse data is used to store ideal points in chambers where most of the votes are missing. Such a situation is typical when the "chamber" is the result of bridging together many chambers. Sparse data is stored as a list with named elements *N*, *T*, *S*, *Nids*, *Tids*, *r*, *c*, and *v*. The sparse matrix is stored in row-column-value format (Saad, 2003). *N* and *T* should be the number of rows and columns of the matrix. The vectors *r* and *c* refer to the rows and columns of the entries of the matrix, and are 0-indexed, meaning that 0 refers to the first row or column, 1 refers to the second row or column, etc. *v* stores the values of the matrix, which can be 2 (indicating a positive outcome), 1 (indicating a negative outcome), or 0 (indicating a missing outcome). While the software allows

v to include 0, these should be excluded for efficiency reasons. If one includes all the 0s from the matrix, there will be no computational advantage to the sparse algorithms. The vector r, c, and v should be all be of the same length, and this length should be supplied as S. *Nids* and *Tids* should be the (unique) row names and column names.

Occasionally, data that is dense (i.e. contains few missing votes) will be stored in a sparse format. In this case, the data can be read as sparse, and then converted to dense data.

Example 2 (Roll Call Voting in the 90th Senate)**.**

Though voting in a given session of the Senate is dense, *Voteview* stores the roll call data in a sparse format. The following code loads roll call data for the 90th Senate as a sparse matrix:

```
data1 <- read.csv("https://voteview.com/static/data/out/votes/S090_votes.
    csv")
Y <- list()
Y$Nids <- sort(unique(data1$icpsr)) # voter ids (icpsr code)
Y$Tids <- sort(unique(data1$rollnumber)) # vote ids
Y$r <- fmatch(data1$icpsr,Y$Nids) - 1 # 0-indexed
Y$c <- fmatch(data1$rollnumber,Y$Tids) - 1 # 0-indexed
Y$v <- data1$cast_code # votes
Y$v[Y$v!=1&Y$v!=6] <- 0 # missing votes (present, not voting, pair, etc.)
Y$v[Y$v==1] <- 2 # yeas
Y$v[Y$v==6] <- 1 # nays
Y$N <- length(Y$Nids) # number of voters
Y$T <- length(Y$Tids) # number of votes
Y$S <- length(Y$v) # number of non-missing entries in matrix
```

To check if the matrix is in a proper format, the function *ipe_check_mat* can be used:

```
ipe_check_mat(Y)
```

A return value of zero indicates that the matrix is in proper format. It is also possible to create a sparse matrix by first creating a dense matrix and using the function *ipe_dense_to_sparse*:

Example 1 (continued).

```
Ys <- ipe_dense_to_sparse(Y)
```

The function *ipe_sparse_to_dense* has similar functionality for converting in the reverse direction.

Example 2 (continued).

```
Yd <- ipe_sparse_to_dense(Y)
```

2.3 Chamber Data Structures

A chamber object is a list with the following elements: Y (a dense or sparse matrix of the form described in sections 2.1 and 2.2), *name* (the name of the chamber), and optionally, *inddat*, *itemdat*, *source*, and *member*. If provided, *inddat* should be a data frame where each column indicates an individual-specific variable (e.g. party) and where the number of rows is equal to N (the number of individuals). Similarly, *itemdat* should be a data frame where each column indicates an item-specific variable (e.g. the number of the bill the vote relates to) and where the number of rows is equal to T (the number of items). *source* and *member* are generated when a chamber is constructed by merging a collection of chambers, as described below.

Chambers can be loaded or saved using the functions *ipe_load_chamber* and *ipe_save_chamber*. The chamber object thus allows for a convenient way to save the response matrix along with individual-specific and item-specific information. *ipe_save_chamber* takes a chamber object and a file name, which should have the suffix "rds". *ipe_save_chamber* is a wrapper for the function *save.list* from the *rlist* package, which verifies the format of the data. The function *ipe_load_chamber* takes in a file name and returns a chamber object. This is also a wrapper for a function from the *rlist* package which verifies the format of the data.

Example 1 (continued).

The following example saves a chamber for later use:

```
chamber <- list()
chamber$Y <- Y
chamber$inddat <- as.data.frame(party)
ipe_save_chamber(chamber,paste(dir1,"cali1.rds",sep=""))
```

The following code loads the saved data:

```
chamber2 <- ipe_load_chamber(paste(dir1,"cali1.rds",sep=""))
```

The validity of a chamber object can be checked using the function *ipe_check_chamber*. The function takes in a chamber object and returns an integer. Table 1 details the meaning of the return values for *ipe_check_chamber*.

A collection of chambers is a list numbered 1, 2, ..., L, where L is the number of chambers and each entry in the list is a chamber object.

Example 3 (1st through 113th Senate)**.**

We can use the following code to get a list of chambers with votes, individual-level data, and names for the chambers:

```
chambers <- list()
for(cong in 1:113)
```

0	No Error
1	Y is missing (the entry Y should be present in the list)
2	Invalid entries in chamber (the list contains an entry other that Y, name, inddat, itemdat, member, and source)
3	Y is not a valid vote matrix (ipe_check_mat returned an error when applied to Y)
4	Number of rows in inddat is not equal to N
5	Number of columns in itemdat is not equal to T
6	Dense chambers should not have a source

Table 1: Return values for *ipe_check_chamber*.

```
{
        print(cong)
        flush.console()
        cong1 <- paste(rep("0",1*(cong < 10)),rep("0",1*(cong < 100)),cong
            ,sep="")
        data1 <- read.csv(paste("https://voteview.com/static/data/out/
            votes/S",cong1,"_votes.csv",sep=""))
        data1 <- data1[data1$cast_code==1|data1$cast_code==6,] # drop the
            missing votes
        data2 <- read.csv(paste("https://voteview.com/static/data/out/
            members/S",cong1,"_members.csv",sep=""))
        Y <- list()
        Y$Nids <- sort(unique(as.character(data1$icpsr))) # voter ids (
            icpsr code)
        Y$Tids <- paste(cong,sort(unique(data1$rollnumber)),sep="_") #
            vote ids
        Y$r <- fmatch(as.character(data1$icpsr),Y$Nids) - 1 # 0-indexed
        Y$c <- fmatch(paste(cong,data1$rollnumber,sep="_"),Y$Tids) - 1 #
            0-indexed
        Y$v <- data1$cast_code # votes
        Y$v[Y$v!=1&Y$v!=6] <- 0 # missing votes (present, not voting, pair
            , etc.)
        Y$v[Y$v==1] <- 2 # yeas
        Y$v[Y$v==6] <- 1 # nays
        Y$N <- length(Y$Nids) # number of voters
        Y$T <- length(Y$Tids) # number of votes
        Y$S <- length(Y$v) # number of non-missing entries in matrix
        idx <- fmatch(Y$Nids,as.character(data2$icpsr))
        inddat <- as.data.frame(cbind(data2$bioname[idx],data2$state_
            abbrev[idx],data2$party_code[idx]))
        colnames(inddat) <- c("name","state","party")
```

```
        inddat$party[is.na(inddat$party)] <- "NA"
        inddat$party[inddat$party==100] <- "D" # recode Democrats
        inddat$party[inddat$party==200] <- "R" # recode Republicans
        inddat$party[inddat$party==1] <- "Federalist"
        inddat$party[inddat$party==13] <- "Democratic-Republican"
        inddat$party[inddat$party==29] <- "Whig"
        inddat$party[inddat$party==555] <- "Jackson"
        chambers[[cong]] <- list(Y=Y,inddat=inddat,name=paste("S",cong,sep
            =""))
}
```

The code does not populate *itemdat* because there aren't interesting vote-specific variables to include. When constructing a list of chambers, it is essential to properly name the voters and the votes (stored in *Nids* and *Tids*, respectively). This example uses a Senator's ICPSR id as the unique identifier for voters. It is this identifier that will allow proper merging of the chambers into one large chamber. The example concatenates the chamber and the roll call number as the unique identifier for votes. This is because the roll call number is only unique within a given session of the Senate. Failing to include the Congress number in the identifier would lead to improper merging since merging would treat roll call 46 in the 1st session of the Senate as the same vote as roll call 46 in the 18th session of the Senate (for example), even though these votes are unrelated.

A collection of chambers can be saved or loaded using the commands *ipe_save_chambers* and *ipe_load_chambers*, which have similar functionality to *ipe_save_chamber* and *ipe_load_chamber* and are illustrated below:

Example 3 (continued).

The collection of chambers object can be saved as:

```
ipe_save_chambers(chambers,paste(dir1,"sen1_113.rds",sep=""))
```

and can be loaded as:

```
chambers2 <- ipe_load_chambers(paste(dir1,"sen1_113.rds",sep=""))
```

2.4 Ordered Data

In it possible to convert ordered responses to binary responses. Consider an ordered response of the following form,

$$x^*_{nt} = a_t + b'_t \alpha_n + \epsilon_{nt} \tag{1}$$

where we observe $x_{nt} = j$ if an only if $\tau_{tj} \leq x^*_n \leq \tau_{t,j+1}$ and where τ_{tj} are cutoffs for vote t, $j \in \{0, ..., J\}$, J is number of responses for vote j, $\tau_0 = -\infty$, and $\tau_J = \infty$. Based on this, we can derive,

$$\Pr(x_{nt} = j) = \Phi(\tau_{tj} - a_t - b_t'\alpha_n) \tag{2}$$

If we recode ordered responses based on whether the response is less than or equal to j for each $j \in \{1, ..., J-1\}$ and define the reduced form parameters $\tilde{a}_{tj} = \tau_{tj} - a_t$, then these recoded ordered responses fit into our binary outcome framework.[1] Data can be converted as such using the function *ipe_expand*. This functions takes in a (dense) matrix where missing values denote missing values and integers denote ordered responses, and produces a dense matrix of the format required by *ipe*. The columns of the inputed matrix must be named. Below, we illustrate one example:

Example 4 (2000 National Annenberg Election Study)**.**

The 2000 National Annenberg Election Study asked a number of policy questions which can be used to estimate the ideology of the mass public. Few of these are binary, but the vast majority are ordered. The following code loads the binary and ordered policy questions from the NAES and converts the ordered items to binary items.

```
data1 <- read_excel(paste(dir1,"npat_key.xlsx",sep="")) # item key
data2 <- read.delim(paste(dir1,"rcs.dat",sep="")) # the NAES
Y <- data2[,fmatch(data1$var,names(data2))]
Y[Y>4] <- NA # get rid of invalid responses
colnames(Y) <- data1$var # name the items
party <- data2$CV01
party[party==1] <- "R"
party[party==2] <- "D"
party[party!="D"&party!="R"] <- "I"
inddat <- data.frame(party=party,state=data2$CST) # individual data
itemdat <- data.frame(lab=data1$lab,type=data1$type) # item data
chamber <- ipe_ordered_to_binary(Y,itemdat=itemdat)
chamber$inddat <- inddat
chamber$Y <- ipe_dense_to_sparse(chamber$Y) # convert to sparse, for
   efficiency
```

Note that the *ipe_ordered_to_binary* function only works with dense data, but the dense data created by this function can then be converted to sparse data.

2.5 Bridging Chambers

Suppose that we have a number of chambers that are connected using a small number of bridge voters or bridge votes. For example, we may have 113 sessions of the U.S. Senate connected by

[1] Though the marginal probabilities are correct, the observations would no longer be independent, leading standard errors to potentially be too small if there are many ordered observations. This can be accounted for by clustering by ordered response, but this is not yet implemented in *ipe*.

Senators who served in multiple sessions. Such problems pose special challenges. Here, we demonstrate how to construct a large "chamber" by bridging together a number of smaller chambers, with the next subsection describing how to check if there are enough bridges between the smaller chambers and Section 3.2 demonstrating how to construct starting values for estimation problems involving bridging.

Chambers can be bridged together using the function *ipe_merge_chambers*, which takes in a collection of chambers object and returns a chamber object—a list with the components Y (the merged data matrix, in sparse form), *source* (which lists for each entry in Y, the chamber the entry came from), *inddat* (the merged individual level data), *itemdat* (the merged item data), and *member* (a membership matrix for individuals). Note that if an individual appears in two chambers, *ipe_merge_chambers* assumes that their individual level data does not vary by chamber. This is violated in *voteview*'s roll call data for party (considered below)—many legislators changed their party without the ICPSR code changing. This generally occurred when the individual's party changed due to a change in the party system. Users wanting to collect individual level data that changed across chambers will have to collect this data separately from the *ipe_merge_chambers* function.

Example 3 (continued).

The sessions of the U.S. Senate can be merged into a single chamber using as bridge individuals Senators who served in multiple sessions. The following code accomplishes this:

```
merge1 <- ipe_merge_chambers(chambers)
```

The function *ipe_split_chamber* takes a single chamber and splits it into a collection of chambers. By default, *ipe_split_chamber* will split by *source* variable, but this can be over-ridded by specifying a split variable. This is illustrated below:

```
chamber3 <- ipe_split_chamber(merge1,split=merge1$source)
```

In this example, *split* could have been omitted as *ipe_split_chamber* would have split the chambers using *source* by default. Note that *ipe_split_chamber* will fail on dense chambers. If it is desired to split a dense chamber, the data matrix can be converted from dense to sparse and the chamber can then be split.

2.6 Bridge Analysis

Before merging the chambers, it is important to check that enough bridges are present to obtain accurate results. Shor, McCarty and Berry (2008) and Battista, Peress and Richman (Forthcoming) have found in Monte Carlo experiments that good performance can be achieved with relatively few bridge individuals or bridge items—as few as 5 bridges may be enough, with 20 bridges being a very safe amount. The function *ipe_bridge_analysis* can help in checking if enough bridges are available.

This function takes in collection of chambers. An optional parameter is *minnum* (default is 20) which is the minimum number of bridges to require. The function first counts the raw number of bridges between any pair of chambers (summing bridge individuals and bridge items). Next, it applies that Ford-Fulkerson Algorithm (Ford and Fulkerson, 1956) to compute the maximum flow between any pair of chambers. To see why this is useful for, there are 0 senators who served in the 1st Congress as well as the 10th Congress. However, there are 21 who served in the 1st and the second, 22 who served in the 2nd and the 3rd, 23 who served in the 3rd and the 4th, etc. Connecting any pair of congresses between the 1st and the 10th, we never have fewer that 21 bridge individuals. The maximum flow algorithm is one way of accounting for the fact that one chamber may be bridged to another chamber indirectly.

The function *ipe_bridge_analysis* returns a list with elements *count*, *countind*, *countitem*, and *flow*. *flow* contains the flow between each pair of chambers. In the event that there is a flow less than *minnum* between between at least two chambers, the function will also return *clusters*. The connections between chambers with a flow of at least *minnum* generate an undirected graph which is partitioned into connected components (Skiena, 1997). If the graph is not connected, *clusters* will contain the connected components.

Example 3 (continued).

The following code analyzes the number of bridges in the U.S. Senate:

```
bridge1 <- ipe_bridge_analysis(chambers)
```

As the flow is at least 20 for each pair of chambers, *clusters* is not returned. *countind* and *countitem* contain the number of bridge individuals and bridge items between every pair of chambers. *count* will contain the total number of bridges (the sum of the number of bridge individuals and bridge items). Below are *countind*, *countitem*, and *flow* for the first 10 sessions of the Senate:

```
> bridge1$countind[1:10,1:10]
    S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
S1  NA NA NA NA NA NA NA NA NA  NA
S2  21 NA NA NA NA NA NA NA NA  NA
S3  12 21 NA NA NA NA NA NA NA  NA
S4   8 14 23 NA NA NA NA NA NA  NA
S5   5  6 13 31 NA NA NA NA NA  NA
S6   3  4  7 19 26 NA NA NA NA  NA
S7   1  3  5  9 12 22 NA NA NA  NA
S8   1  4  5  7  7 12 24 NA NA  NA
S9   0  1  2  3  4  4 12 27 NA  NA
S10  0  1  1  1  3  3  6 19 25  NA
> bridge1$countitem[1:10,1:10]
    S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
S1  NA NA NA NA NA NA NA NA NA  NA
```

```
S2    0 NA NA NA NA NA NA NA NA   NA
S3    0  0 NA NA NA NA NA NA NA   NA
S4    0  0  0 NA NA NA NA NA NA   NA
S5    0  0  0  0 NA NA NA NA NA   NA
S6    0  0  0  0  0 NA NA NA NA   NA
S7    0  0  0  0  0  0 NA NA NA   NA
S8    0  0  0  0  0  0  0 NA NA   NA
S9    0  0  0  0  0  0  0  0 NA   NA
S10   0  0  0  0  0  0  0  0  0   NA
> bridge1$flow[1:10,1:10]
     S1 S2 S3  S4  S5  S6  S7  S8  S9 S10
S1   NA 57 57  57  57  57  57  57  57  57
S2   57 NA 85  85  85  85  85  85  85  85
S3   57 85 NA  99  99  99  99  99  99  99
S4   57 85 99  NA 119 108 107 125 125 125
S5   57 85 99 119  NA 108 107 119 119 119
S6   57 85 99 108 108  NA 107 108 108 108
S7   57 85 99 107 107 107  NA 107 107 107
S8   57 85 99 125 119 108 107  NA 141 133
S9   57 85 99 125 119 108 107 141  NA 133
S10 57 85 99 125 119 108 107 133 133  NA
```

The function reports that there are no bridge items (as expected) and few bridge individuals between distant sessions of the Senate, but that results demonstrate that the bridging is not problematic as distant chambers are connected indirectly though chains of bridge individuals.

The example below is intended to demonstrate an instant where bridging is problematic.

Example 5 (Merging the National Political Awareness Test and Congressional Roll Call Voting)**.**

The National Political Awareness Test (NPAT) is a survey of Congressional candidates that was first fielded in 1992. Candidates for office are asked a series of policy questions, with the same questions asked of House and Senate candidates, and many identical questions asked in different years. In Example 3, different sessions of the Senate were bridged by assuming that Senators do not change their positions over time. If we wanted to develop a dynamic measure of the positions of Senators, this assumption could not be made and other bridges would have to be identified. Here, we consider using common questions asked in the NPAT over different years as bridges over time, and members of the House and Senate who responded to the NPAT as bridges between the roll call voting in a Session of the House and Senate and the NPAT. For purpose of illustration, we assume that there are sufficient bridges between the NPAT over time and investigate whether there are sufficient bridge individuals between each session of roll call voting and the NPAT.

The following code loads the NPAT/Roll Call data, uses *source* from the loaded chamber to split

up the data into separate chambers, and analyzes the bridges between these chambers (checking for pairs of chambers where the flow is less than 10).

```
chamber <- ipe_load_chamber(paste(dir1,"npat1.rds",sep=""))
chambers <- ipe_split_chamber(chamber)
bridge1 <- ipe_bridge_analysis(chambers,minnum=10)
```

The function identifies problematic pairs of chambers with the output,

```
[1] "cluster 1: NPAT/H102/H103/H104/H105/S105/H106/H107/H108/H109/H110/
   H111/S111/H112/S112/H113/S113/H114/S114/H115/S115"
[1] "cluster 2: S102"
[1] "cluster 3: S103"
[1] "cluster 4: S104"
[1] "cluster 5: S106"
[1] "cluster 6: S107"
[1] "cluster 7: S108"
[1] "cluster 8: S109"
[1] "cluster 9: S110"
```

indicating that the 102nd through 110th Senates are connected to the other chambers by too few bridge individuals. Analyzing *countind* identifies the source of the problem—in these Senates, there were too few Senators who responded to the NPAT (and due to the structure of the data, there are no bridge items and no indirect bridges, so *flow* is equal to *countind*).

To fix the problem, we can bridge the Senate indirectly to the NPAT using bridge items between concurrent sessions of the House and Senate. The following code loads similar data and checks it:

```
merge6 <- ipe_load_chamber("c:\\Users\\mike0\\Desktop\\npat2.rds")
chambers6 <- ipe_split_chamber2(merge6)
bridge6 <- ipe_bridge_analysis2(chambers6,minnum=10)
```

Examining the flow matrix, we find that it is always greater than 24, suggesting more confidence that we have a sufficient number of bridges.

3 Estimation

3.1 Estimation

Once the data matrix is formed, estimation can be performed in three steps—computing starting values, minimizing the objective function, and normalizing the estimates. Functions for each of these interact with ideal point objects. An ideal point object has the elements *Alpha* and *Delta*, representing the individual parameters and the item parameters, and optionally, *AlphaSe*, *AlphaVarDiag*, *DeltaSe*, *DeltaVarDiag*, *Fit*, *IndFit*, and *ItemFit*.

Example 1 (continued).

The following code illustrates the three steps in the estimation process.

```
ipe1s <- ipe_start(Y,1) # starting values
ipe1u <- ipe_pmle(Y,1,ipe=ipe1s) # main estimation
ipe1 <- ipe_normalize_by_group(ipe1u,party,list("D"=-1,"R"=1)) # normalize
    output
```

The first command, *ipe_start*, takes a chamber object and a number of dimensions (in this case set to 1) and returns an ideal point object—a list with elements *Alpha* and *Delta* containing the starting values for α and $\delta = (a, b)$, respectively. The command itself does the following: starting values for α are computed using method A described on page 7 of Peress (forthcoming). Starting values for δ are computed using vote-specific (penalized) probits, taking the starting values for α as given. The starting values for α and δ are then re-normalized based on the scale suggested by the prior as described on pages 8 and 9 of Peress (forthcoming). Optional parameters include *AlphaFac* and *DeltaFac*, which determine the penalty parameters λ_α and λ_δ. These may be provided as scalars (in which case it is assumed that all individuals and items have the same penalty term) or as vectors of length N and T respectively. Default values are 1 for both. As described in Peress (forthcoming), applying penalized maximum likelihood estimation is equivalent to maximizing a posterior distribution, so the penalty parameter can be interpreted as determining the dispersion of the prior distribution. *IndWeights* and *ItemWeights* allow for unequal weighting of individuals and items, with the default being equal weighting.[2]

The second command, *ipe_pmle*, takes a dense or sparse data matrix and a number of dimensions (again set to 1 in the example) and returns an ideal point object—a list with elements *Alpha* (individual parameter estimates), *Delta* (item parameter estimates), *AlphaSe* and *DeltaSe* (standard errors), *AlphaVarDiag* and *DeltaVarDaig* (diagonal blocks of the variance-covariance matrix), *Fit* (overall model fit statistics), *IndFit* (individual fit statistics), and *ItemFit* (item fit statistics). Optional parameters again include *AlphaFac*, *DeltaFac*, *IndWeights*, and *ItemWeights*, which have the same default values as with *ipe_start*. *MaxIter* is the maximum number of iterations (default is 2500). *TolD* is the tolerance required of the maximum derivative for convergence (default is 1.0e-6, and changing this is not recommended). *TolX* is the tolerance required of the maximum change in parameters (default is 1.0e-12, and changing this is not recommended). *SkipInference* will skip the computation of standard errors if set to 1 (default is 0). *LineSearchOpt* determines the type of line search used by the optimizer, with 1 being trust region and 2 being backtracking.[3] The default is trust region, which is generally faster, but can be less robust than backtracking. Starting values can be supplied in two ways. In the example, they are supplied to the parameter *ipe* as an ideal point object (a list with elements *Alpha* and *Delta*). Alternatively, if it is desired to supply starting values for only α or δ, the parameters *AlphaStart* or *DeltaStart* can be used, with each taking in a matrix of size equal to $N \times D$ or $T \times (D + 1)$, respectively.

[2] See Battista, Peress and Richman (Forthcoming), for example.
[3] See Dennis and Schnabel (1983) for a discussion of line search methods.

NumThreads (default value 1) determines the number of threads to use in the estimation. In principle, it should be possible to divide work efficiently across multiple processors or cores. In practice, the efficiency will depend on the cooling characteristics of the system. It is increasingly common for multiple core computers to throttle their CPU when multiple cores are running for extended periods of time. This is particularly likely to be the case for laptop computers. As such, it is possible that increasing the number of threads will slow down performance. Users should experiment with various choices of *NumThreads* rather than assuming that increasing the number of threads will increase performance.

The estimates for α and δ produced by *ipe_pmle* are on the (likely arbitrary) scale suggested by the penalty parameters. The third command, *ipe_normalize_by_group*, normalizes the ideal points by setting a number of group averages to pre-selected values. The number of groups normalized should be set equal to $D + 1$. The function takes an ideal point object (a list with elements *Alpha* and *Delta*), a group to normalize by, and a list with normalized values for $D + 1$ groups. If the ideal point object includes *AlphaSe*, *DeltaSe*, *AlphaVarDiag*, and *DeltaVarDiag*, these will be transformed accordingly as well. The output will be an ideal point object with the estimates, standard errors, and variance blocks appropriately transformed. Optional parameters include *stat*, which is "mean" by default, but can be changed to "median" if $D = 1$.[4] *num* (default=0) sets the minimum number of observations to use when calculating the group averages. For example, if *num* is set to 20, individuals with 19 or fewer items will not be used in computing the group means.

The list of individual parameters supplied to *ipe_normalize_by_group* should be such that the groups do not align along a lower dimensional subspace. Technically, this can be achieved by having the differences between the individual parameter vectors span $\mathbb{R}^D$. When $D = 1$, this amounts to requiring that the two groups have distinct individual parameters. When $D = 2$, this would require that the three groups not fall on a single line.

In computing the adjusted standard errors, *ipe_normalize_by_group* assumes that the groups are growing in size as the number of individuals increases. One can force *ipe_normalize_by_group* to normalize by $D + 1$ individuals instead of $D + 1$ groups in the following way:

Example 1 (continued).

```
ipe1a <- ipe_normalize_by_group(ipe1u,rownames(Y),list("Michael M. Honda"
   =-1,"Tom McClintock"=1))
```

This would retain the validity of the estimates for α and δ, but the standard errors would be incorrect. There are two reasons why *ipe* does not allow for correct standard errors when scales are normalized by small group means. First, uncertainty will generally be larger on these scales, so these types of normalizations should be avoided. Second, once such a transformation is made,

[4] The group medians cannot be used when $D > 1$ because dimension-by-dimension medians are not preserved by linear transformations.

the variance-covariance matrix would no longer be block-diagonal, so performing any subsequent inferences would require storing the entire variance-covariance matrix (which would be large even in smaller examples) or storing the history of such transformations.

In some cases, it may be difficult to identify $D+1$ groups to normalize. In these cases, the functions *ipe_normalize_0I*, *ipe_normalize_flip*, *ipe_normalize_permute*, and *ipe_normalize_varimax* may be used.

Example 4 (continued).

In normalizing the ideal points of the mass public in two dimensions, there are no obvious groups that can be used to define the second dimension. Here, we can instead normalize the ideal points to be mean zero, variance one, and uncorrelated across dimension:

```
ipe3s <- ipe_start(Y3,2) # starting values
ipe3u <- ipe_pmle(Y3,2,ipe=ipe3s) # main estimation
ipe3 <- ipe_normalize_0I(ipe3u)
```

After plotting the data, one may determine that reversing the order of one of the dimensions is appropriate:

```
ipe3 <- ipe_normalize_flip(ipe3,1)
```

Here, the 1 indicates that the first dimension will be flipped around zero. It may also be desired to swap the order of the dimensions:

```
ipe3 <- ipe_normalize_permute(ipe3,1,2)
```

In this case, the second and third parameters determine which dimensions are flipped, with this example flipping the first and second dimension. The three transformations are not sufficient to uniquely determine the scale of the ideal points. *ipe_normalize_0I* selects a transformation $C\alpha_n + d$ such that:

$$\tfrac{1}{N}\sum_{n=1}^{N}(C\alpha_n + d) = 0 \tag{3}$$

$$\tfrac{1}{N-1}\sum_{n=1}^{N}\left(C\alpha_n + d - \tfrac{1}{N}\sum_{m=1}^{N}(C\alpha_m + d)\right)\left(C\alpha_n + d - \tfrac{1}{N}\sum_{m=1}^{N}(C\alpha_m + d)\right)' = I \tag{4}$$

The solution to second equation indicates that,

$$CC' = \left[\tfrac{1}{N-1}\sum_{n=1}^{N}\left(\alpha_n - \tfrac{1}{N}\sum_{m=1}^{N}\alpha_m\right)\left(\alpha_n - \tfrac{1}{N}\sum_{m=1}^{N}\alpha_m\right)'\right]^{-1} \tag{5}$$

If $D > 1$, there are multiple solutions. *ipe_normalize_0I* selects the transformation C such that C is lower triangular. An additional transformation of the form $C\alpha_n + d$ with $d = 0$ and $CC' = I$ will

preserve the mean and variance of α_n. One such approach is the Varimax rotation (Kaiser, 1958) applied to $\frac{b_t}{a_t}$. To preserve $a_t + b_t'\alpha_n$, a transformation with $d = 0$ will transform $\frac{b_t}{a_t}$ as $C^{-1}\frac{b_t}{a_t}$ and leave a_t untransformed. Since $CC' = I$, we have $C^{-1} = C'$. In this context, the Varimax rotation is defined by,

$$C_{varimax} = \arg\max_{C} \sum_{d} \left[\frac{1}{T} \sum_{t} \left(\frac{[C'b_t]_d}{a_t} \right)^4 - \left(\frac{1}{T} \sum_{t} \left(\frac{[C'b_t]_d}{a_t} \right)^2 \right)^2 \right] \tag{6}$$

The Varimax rotation transforms $\frac{b_t}{a_t}$ such that the variance of the squares is minimized, averaging over dimensions. This rotation privileges item parameters that have large coefficients on a small number of items, along each dimension. In the Factor Analysis literature, this rotation is often argued to lead to more interpretable item parameters, corresponding to cutting lines that are parallel to one of the D dimensions. Even if this is not explicitly desired, applying the rotation after *ipe_normalize_0I* is useful for comparing two sets of estimated ideal points in multiple dimensions, as it ensures that the ideal points are on the same scale.

Example 4 (continued).

The Varimax rotation is implemented in the function *ipe_normalize_varimax*:

```
ipe3 <- ipe_normalize_varimax(ipe3)
```

Optional parameter *num* (default value 0) will ignore δ_t with fewer that *num* observations.

Due to the penalty terms, *ipe_pmle* will produces estimates for individuals or items, even if these estimates are based on little data (or in the extreme case, no data). In some cases, this is not problematic, as illustrated in the example below:

Example 1 (continued).

In the California House, the individual parameters are estimated based on at least 1133 observations:

```
> table(ipe1$IndFit$IndTot)
1133 1341 1502 1861 1904 1927 1931 1943 1949 1958 1970 1974 1975 1978 1990 1992
   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
1999 2003 2007 2037 2039 2041 2045 2046 2053 2058 2063 2067 2068 2072 2073 2074
   1    1    1    1    1    1    1    1    1    1    1    2    1    1    1    1
2075 2076 2078 2082 2083 2086 2087 2088 2089 2091 2093 2094 2096 2097 2099 2101
   1    1    1    1    1    1    3    2    1    2    1    1    1    1    1    1
2108 2114 2115 2118 2119 2122 2125 2127 2128 2132 2147 2151 2154 2155 2157 2168
   1    2    2    2    1    1    1    2    2    1    2    1    1    1    1    1
2170 2177 2181 2191
   2    1    1    1
```

Example 4 (continued).

In the NAES, some individual parameters are estimated based on few observations or no observations:

```
> table(ipe1$IndFit$IndTot)
   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
  47  951   17   39   83   58  269 1312   14   19   29   37   57   36   61  128
  16   17   18   19   20   21   22   23   24   25   26   27   28   29   30   31
  85  183  483  255  743 5431  955  892 1592 4450 3834  664 1322 3402 6029   15
  32   33   34   35   36   37   38   39   40   41   42   43   44   45   46   47
  18   11    6   13   16   20   28   31   27   31   41   43   46   59   62  100
  48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63
  87  107  125  181  209  270  480  411  706 2606  976 1042 1546 3786 3400  636
  64   65   66
 938 2316 4507
```

A minimum number of observations per estimate can be maintained in two ways. After estimation, the user can ignore all ideal points with fewer than 10 (for example) observations:

```
ipe1$Alpha[ipe1$IndFit$IndTot<10,] <- NA
```

An alternative approach is to drop observations before estimation:

```
chamber2 <- ipe_reduce(chamber,minind=10)
ipe2s <- ipe_start(chamber2$Y,2)
ipe2u <- ipe_pmle(chamber2$Y,2,ipe=ipe2s,LineSearchOpt=2)
ipe2 <- ipe_normalize_flip(ipe_normalize_varimax(ipe_normalize_0I(ipe2u,
   num=10)),c(1,1))
```

We can compare the two sets of results as follows:

```
> cor(ipe2$Alpha[fmatch(chamber$Y$Nids,chamber2$Y$Nids),1],ipe1$Alpha[,1],
   use="pairwise.complete.obs")
[1] 0.9997205
> cor(ipe2$Alpha[fmatch(chamber$Y$Nids,chamber2$Y$Nids),2],ipe1$Alpha[,2],
   use="pairwise.complete.obs")
[1] 0.9997303
```

As can be seen, both approaches produce nearly identical estimates for the ideal points based on at least 10 observations.

Above, the function *ipe_reduce* takes in a chamber and returns a reduced chamber. *minind* (default value 0) is the minimum number of observations for retaining an individual. *minitem* (default value 0) is the minimum number of observations for retaining an item. *minmarg* (default value 0) is the minimum margin for retaining an item. Note that it is possible for *ipe_reduce* to result in a chamber with individuals having fewer than *minind* observations or items with fewer than *minitem* observations if both are set to a value greater than 0. This could occur (for example) if some of the items that allowed individual to achieve *minind* observations were dropped due to the

item having fewer than *minitem* observations. This can be avoided by applying *ipe_reduce* multiple times.

3.2 Estimating Bridge Chambers

As described in Peress (forthcoming), *ipe_start* is likely to fail to produce good starting values when applied to very sparse chambers. The online appendix of Peress (forthcoming) describes an alternative approach for computing starting values for bridged chambers, which is implemented in the function *ipe_bridge_start*. The function takes in a collection of chambers, a number of dimensions to estimate, and optional parameters. *AlphaFac* and *DeltaFac* supply the penalty terms for penalized maximum likelihood estimation, and ensure that the starting values are on the scale suggested by the penalty terms. *MinNumAlpha* (default value 5) is the minimum number of bridge individuals. *MinNumDelta* (default value 5) is the minimum number of bridge items. In the first step, *ipe_bridge_start* bridges together clusters of chambers that are connected by at least *MinNumAlpha* bridge individuals. The connections between chambers generate an undirected graph which is partitioned into connected components. In the second step, these clusters are connected if there are at least *MinNumDelta* bridge items between the clusters. An undirected graph is again formed and if the graph is not connected, *ipe_bridge_start* will fail. If a connected graph was found in the first step, the second step is skipped. Note that it is possible for *ipe_bridge_start* to fail even when *ipe_bridge_analysis* finds that there are sufficient bridges—this can occur if a chamber is connected by a small number of bridges to many chambers. In this case, one can try lowering *MinNumAlpha* and *MinNumDelta*, but additional caution should be maintained as *ipe_bridge_start* attempts to find starting values by re-mapping chamber estimates of the individual parameters and the item parameters.

The optional parameters *AlgorithmAlpha* and *AlgorithmDelta* (default values 1) determine the algorithm that is used in each step.[5] If the algorithm for a step is set to 1, a penalized function is optimized. If the algorithm for a step is set to 2, a constrained function is optimized. If algorithm 1 is used, *Lambda* (default value 1) determines the penalty term. If algorithm 2 is used, the constraint is applied to the first chamber in the chambers object within the current cluster of chambers.

Algorithm 1 privileges not flipping the left-right orientation of chambers. Before applying algorithm 1, it is thus necessary to guess the correct orientations. Within each cluster in the first step, algorithm 1 builds a matrix where entry (n, m) is the correlation between the individual parameters estimated in chambers n and m. If there are no individuals bridging the chambers, or if the number of bridge individuals is less than optional parameter *MinNumCorr* (default value 10), the entry is set to 0. An eigenvalue decomposition is applied to the matrix and the sign of the individual parameter is flipped whenever the corresponding entry of the first eigenvector is negative. This procedure is designed to alter the orientation of the chamber-by-chamber ideal points such

[5] See the discussion on the bottom of page 2 of the online appendix of Peress (forthcoming).

that they are positively correlated across pairs of chambers. If $D > 1$, this same procedure is applied one dimension at a time.

Algorithm 2 privileges constraining chambers with a lower index. The parameter *order* (a permutation of $\{1, 2, ..., L\}$, where L is the number of chambers) re-orders the chambers, and thus allows for the constraint to be applied to an arbitrary chamber. Suppose algorithm 2 is used. In the first step, within each cluster of chambers, the lowest order chamber is constrained. In the event that the chambers are connected in the first step, all chambers are in the first cluster and the chamber with *order* equal to 1 is constrained. In the second step, as there is always one cluster, the first chamber in the order is constrained.

In choosing an algorithm to use, one should consider whether a single chamber is connected to many other chambers, or whether the chambers are connected in a long chain. For example, in Battista, Peress and Richman (2013), 99 state legislature are connected to each other through a common survey. All 100 chambers are connected using bridge individuals—individuals who responded to the survey and voted in one of the 99 legislative chambers. In this case, algorithm 2 is appropriate, though *order* should be set so that the normalization is applied to this common chamber. In the DW-Nominate Common Space, all chambers are connected using bridge individuals, but only through long chains. In this case, algorithm 1 is more appropriate. The example below is a simplified version of the DW-Nominate Common Space.

Example 3 (continued).

One could try estimating the ideal points without providing starting values as follows:

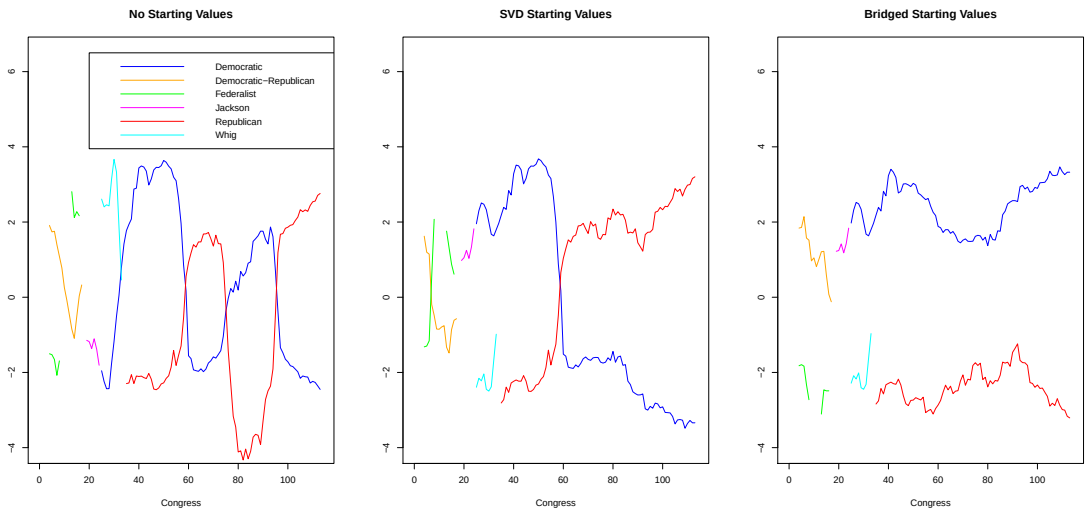
```
set.seed(1234) # set seed since results will be affected by random
   starting values
ipe1u <- ipe_pmle(merge1$Y,D=1)
ipe1 <- ipe_normalize_by_group(ipe1u,merge1$inddat$party,list(D=-1,R=1))
```

A plot of the party means over time is given in the first panel of Figure 3.2. The plot identifies implausible flips in the relative positions of the Democratic and Republican parties that are due to failing to find a global optimum of the objective function due to poor starting values—in this case, as no starting values were provided, random starting values were used. One can instead use *ipe_start* to compute starting values using the singular value decomposition:

```
ipe2s <- ipe_start(merge1$Y,D=1)
ipe2u <- ipe_pmle(merge1$Y,D=1,ipe=ipe2s)
```

These results are given in the second panel of Figure 3.2. Again, we find implausible flips in the relative positions of the Democratic and Republican parties. Appropriate starting values can be obtained using *ipe_bridge_start*:

```
ipe3s <- ipe_bridge_start(chambers,D=1)
ipe3u <- ipe_pmle(ipe3s$Y,D=1,ipe=ipe3s$ipe)
```



In the third panel of Figure 3.2, the relative positions of the parties appear correct (apart from the overall scale needing to be flipped). We can verify that these differences are due to better starting values by comparing the optimized values of the objective function:

```
> ipe1u$Fit$Objective
[1] 0.01300827
> ipe2u$Fit$Objective
[1] 0.01221757
> ipe3u$Fit$Objective
[1] 0.01204369
```

The starting values produced by *ipe_bridge_start* indeed lead to the smallest (and thus, best) value for the objective function.

Example 6 (State Legislatures Example)**.**

Gerald Wright's Representation in the America's Legislature Project (Wright, 2004) collected data from 99 state legislatures for the 1999-2000 time period. These chambers can be bridged using the National Political Awareness Test (NPAT), a survey filed out can candidates for office. State legislators who responded to the NPAT serve as bridge individuals between the NPAT and roll call voting in the state legislatures, with the state legislatures indirectly bridged to each other through the NPAT. This example differs from Example 3 in there are no long chains to worry about. It is also a more difficult example due to the fact that some chambers have very few bridges and a few chambers exhibit some strange voting behavior.

We first try to use algorithm 1. For some state legislatures, there are only 3 bridge individuals between the NPAT and the roll call data. Though this is on the very low end of what we would need for reliable bridging, setting *MinNumAlpha* to 3 forces *ipe_bridge_start* to provide starting values.

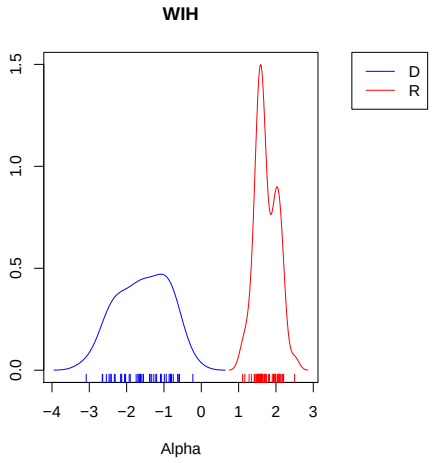


Figure 1: Density, Wisconsin House

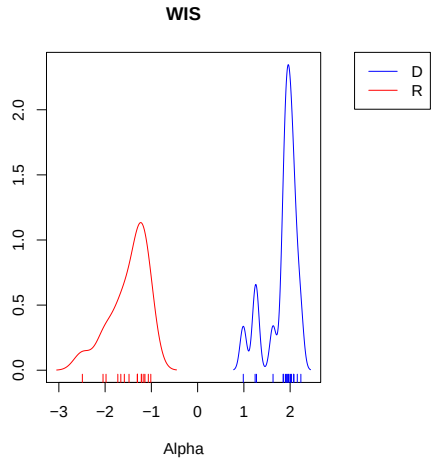


Figure 2: Density, Wisconsin Senate

```
ipe1s <- ipe_bridge_start(chambers,1,MinNumAlpha=3)
ipe1u <- ipe_pmle(ipe1s$Y,D=1,ipe=ipe1s$ipe)
ipe1 <- ipe_normalize_by_group(ipe1u,ipe1s$inddat$Party,list("D"=-1,"R"=1)
    )
```

We can demonstrate a likely problem with the following code:

```
ipe_density(ipe1,by=ipe1s$inddat$Party,main="WIH",subset=ipe1s$inddat$
    State=="WI"&ipe1s$inddat$Chamber=="H"&substr(ipe1s$inddat$Bridge,1,1)==
    "R",bys=c("D","R"),col=c('blue','red'),rug=T)
ipe_density(ipe1,by=ipe1s$inddat$Party,main="WIS",subset=ipe1s$inddat$
    State=="WI"&ipe1s$inddat$Chamber=="S"&substr(ipe1s$inddat$Bridge,1,1)==
    "R",bys=c("D","R"),col=c('blue','red'),rug=T)
```

The plots are given in Figure 3.2. The Democrat and Republican parties have opposite orientations in the Wisconsin House and the Wisconsin Senate. This is unlikely to be the case in reality, and likely occurred because of poor starting values. The following code instead uses algorithm 2, which may be more effective when there are no long chains:

```
ipe2s <- ipe_bridge_start(chambers,1,MinNumAlpha=3,AlgorithmAlpha=0,order=
    c(9,1:8,10:100))
ipe2u <- ipe_pmle(ipe2s$Y,D=1,ipe=ipe2s$ipe)
ipe2 <- ipe_normalize_by_group(ipe2u,ipe2s$inddat$Party,list("D"=-1,"R"=1)
    )
```

Setting *order* equal to *c(9,1:8,10:100)* ensures that the constraint is applied to the scale of NPAT ideal points, which is the obvious choice as it is the only chamber connected to every other chamber. Examining the objective function values, it appears that a better estimate was found:

```
> ipe1$Fit$Objective
[1] 0.003082549
> ipe2$Fit$Objective
[1] 0.003077109
```

A further check examines whether the left-right orientation of the Democratic and Republican parties is consistent across chambers:

```
ipe_means(ipe2,by=paste(ipe2s$inddat$State,ipe2s$inddat$Chamber),subset=
    ipe2s$inddat$Party=="R")-ipe_means(ipe2,by=paste(ipe2s$inddat$State,
    ipe2s$inddat$Chamber),subset=ipe2s$inddat$Party=="D")
```

Examining these, only the Rhode Island House has the wrong orientation.

Finding a few suspect chambers is common is harder problems such as this one. In these instances, one may want to perturb some of the estimates and re-optimize. This can be accomplished with the help of the function *ipe_init_delta*, which takes in a chamber and a matrix containing the current α. The function optimizes over δ given the current α. Optional parameters include *AlphaFac* (default value 1), *DeltaFac* (default value 1), *IndWeights* (default equal weighting), and *ItemWeights* (default equal weighting).

We can attempt to fix this problem with Rhode Island as follows:

```
ipe3s <- ipe2u
ipe3s$Alpha[ipe2s$inddat$Chamber=="H" & ipe2s$inddat$State =="RI"] <- -
    ipe3s$Alpha[ipe2s$inddat$Chamber=="H" & ipe2s$inddat$State =="RI"]
ipe3s$Delta <- ipe_init_delta_sparse(ipe2s$Y,ipe3s$Alpha)
ipe3u <- ipe_pmle(ipe2s$Y,D=1,ipe=ipe3s)
ipe3 <- ipe_normalize_by_group(ipe3u,ipe2s$inddat$Party,list("D"=-1,"R"=1)
    )
```

The code starts with the previous estimates, but adjusts the ideal points for the Rhode Island House by flipping the scale around 0. The estimates for δ are then re-optimized. Skipping this later step would leave the δ as the values that optimize the αs before the scale was flipped, which could prevent the optimizer for exploring this potentially better orientation. These new starting values lead to slightly improved estimates:

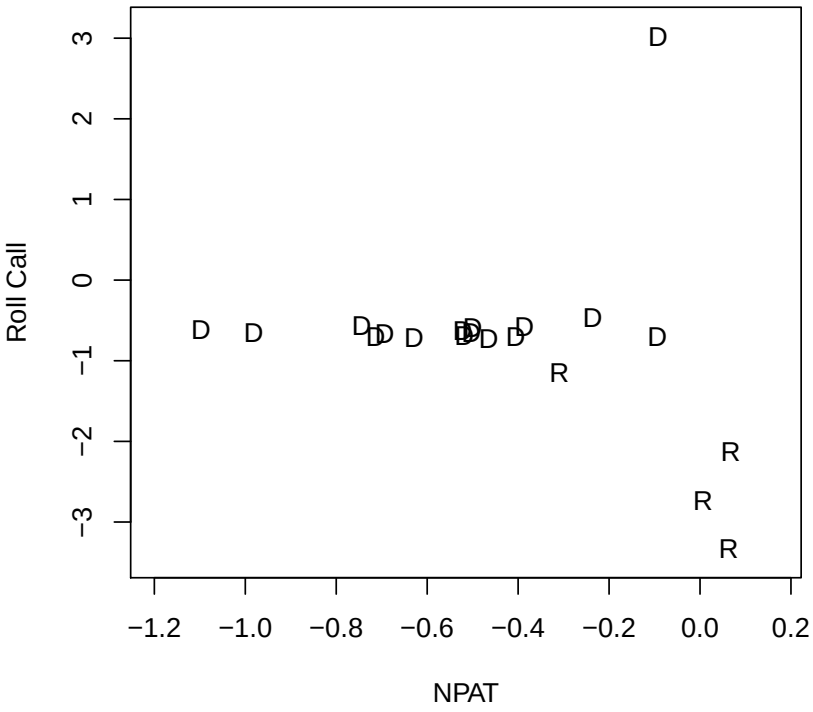
```
> ipe3$Fit$Objective
[1] 0.003076813
```

Further information can be obtained by setting *SaveAlphaArray* equal to true in *ipe_bridge_start*:

```
ipe14s <- ipe_bridge_start(chambers,1,MinNumAlpha=3,AlgorithmAlpha=0,order
    =c(9,1:8,10:100),SaveAlphaArray=T)
```

Here, we can directly examine the chamber-by-chamber estimates:

```
plot0(ipe4s$AlphaArray[[1]][,,1],ipe4s$AlphaArray[[1]][,,70],xlab="NPAT",
    ylab="Roll Call",labels=ipe2s$inddat$Party)
```


Roll Call
NPAT
3
2
1
0
−1
−2
−3
−1.2
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
D
R

The code above generates Figure 3.2. Among the bridge individuals, a large number of Democrats have similar roll call ideal points while having distinct NPAT ideal points. One Democrat that is on the extreme left in the roll call data has a more conservative NPAT ideal point than some Republicans. To examine whether the estimates converged properly, we can restart the estimates for the Illinois House, while leaving the other ideal points unchanged:

```
ipe5s <- ipe3u
ipe5s$Alpha[ipe2s$inddat$Chamber=="H" & ipe2s$inddat$State =="IL"] <- 0
ipe5s$Delta <- ipe_init_delta_sparse(ipe2s$Y,ipe5s$Alpha)
ipe5u <- ipe_pmle(ipe2s$Y,D=1,ipe=ipe5s)
ipe5 <- ipe_normalize_by_group(ipe5u,ipe2s$inddat$Party,list("D"=-1,"R"=1)
   )
```

We can first check whether the fit improved:

```
> ipe5$Fit$Objective
[1] 0.003071611
```

Indeed the fit improved.

An important point to consider when applying *ipe_bridge_start* relates to how the data is partitioned into chambers. In Example 3, the partitioning was the only obvious one. In Example 6, different NPAT surveys were fielded in different states, with many identical items which served as bridges. One could worry that treating the merged NPAT in a single chamber would lead poor starting values. We compare two approaches in the example below:

Example 6 (continued).

Basic starting values can be computed as:

```
npat <- chambers[[9]]
ipe6s <- ipe_start(npat$Y,1)
ipe6u <- ipe_pmle(npat$Y,1,ipe=ipe6s$ipe)
```

Bridged starting values can be computed as:

```
chambers2 <- ipe_split_chamber(npat,npat$inddat$State[npat$Y$r+1])
ipe7s <- ipe_bridge_start(chambers2,1)
ipe7u <- ipe_pmle(ipe7s$Y,1,ipe=ipe7s$ipe)
```

Comparing the objectives,

```
> ipe6u$Fit$Objective
[1] 0.3769918
> ipe7u$Fit$Objective
[1] 0.3769918
```

we find that the fit is identical. Comparing the estimates themselves also indicates that both methods of producing starting values appeared to be effective.

Example 6 demonstrates the need for caution with starting values. Using random starting values will cause problems in difficult examples, but all starting value algorithms remain ad-hoc to an extent. The starting value algorithms available in *ipe* are intended to handle many types of data, but the user should always consider the following checks on the estimates:

1. Examine patterns within each chamber with respect to individual level variable (e.g. party)
2. Examine the scatter plots of chamber-by-chamber estimates among pairs of chambers using bridge individuals
3. Apply greater scrutiny to connections that are held together by few bridged voters

Even easier cases (e.g. Example 3) may pose difficulties without good starting values. In more difficult cases, such as those with few bridge observations, or those where behavior appears to differ by chamber among the bridge individuals, even good starting values algorithms may fail to produce appropriate starting values for all chambers.

4 Visualization and Interpretation

4.1 Visualizing Ideal Points

The function *ipe_density* can be used to visualize the density of the data along one dimension. The function takes an ideal point object as well as optional parameters listed in Table 2.

Example 1 (continued).

For example, the following can be used:

```
ipe_density(ipe1,party)
```

Results are given in Figure 3. Here, the colors are chosen by default. To specify the colors, add the additional parameters *bys* and *cols*. For example,

```
ipe_density(ipe1,by=party,bys=c("D","R"),cols=c('blue','red'))
```

These results are given in Figure 4.

ipe_density can also be applied for dimension-by-dimension visualization when $D > 1$, provided that the parameter d is specified.

Example 2 (continued).

The density of ideal points for the two dimensions can be obtained as follows:

```
ipe_density(ipe1,by=party,bys=c("D","SD","R"),cols=c('blue','cyan','red'),
    d=1)
ipe_density(ipe1,by=party,bys=c("D","SD","R"),cols=c('blue','cyan','red'),
    d=2)
```

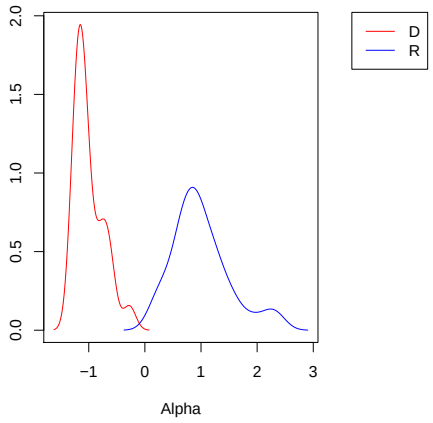


Figure 3: CA House Density, Default Colors

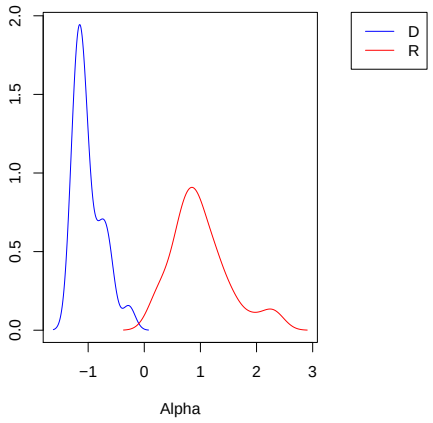


Figure 4: CA House Density, Custom Colors

These results are given in the Figures 5 and 6. Along the first dimension, non-Southern and Southern Democrats are on the left (with the Southern Democrats being somewhat more moderate) while the Republicans are on the right. The second dimension divides the non-Southern and Southern Democrats, with the Republican party being in the middle.

For two and higher-dimensional visualization of ideal points, the functions *ipe_scatter* and *ipe_contour* can also be used. Which of these is chosen depends on the number of individuals as scatter plots will be hard to interpret when there are a large number of data points, unless modified in some way. *ipe_scatter* and *ipe_contour* take in an ideal point object. *ipe_scatter* can be used to produce a scatter plot or a plot of group averages while *ipe_contour* produces contour plots. Optional parameters are given in Tables 3 and 4.

Example 2 (continued).

We can use *ipe_scatter* to plot the two dimensional ideal points of members of the 90th Senate:

```
ipe_scatter(ipe2,by=party)
```

with results given in Figure 7. Again, these results can be improved by selecting appropriate colors:

```
ipe_scatter(ipe2,by=party2,bys=c("D","SD","R"),cols=c('blue','cyan','red')
    )
```

with results given in Figure 8.

Example 4 (continued).

Considering the NAES data once again, we can attempt to report a scatter plot of ideal points by party (Figure 9):

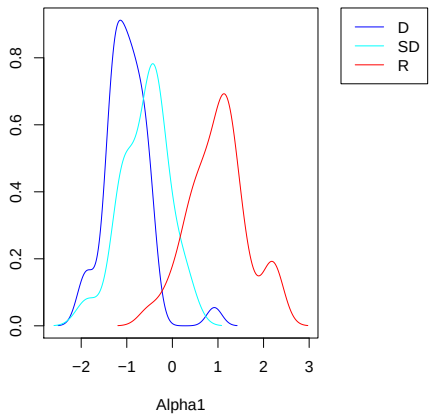


Figure 5: US Senate Density, First Dimension

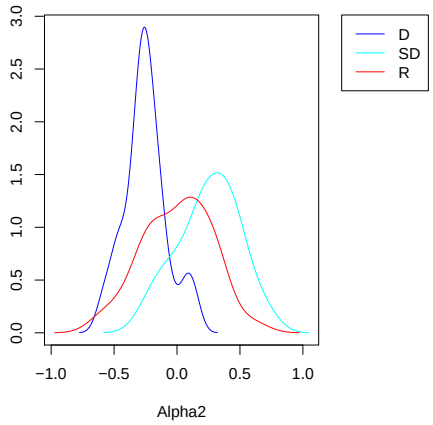


Figure 6: US Senate Density, Second Dimension

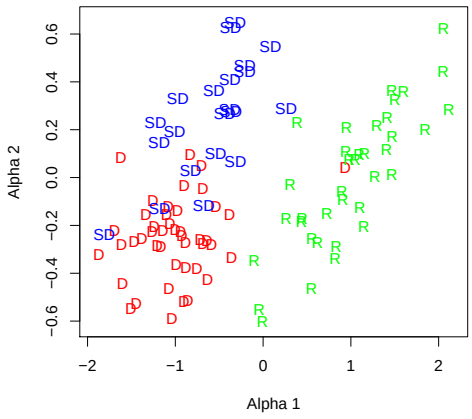


Figure 7: US Senate Scatter Plot, Default Colors

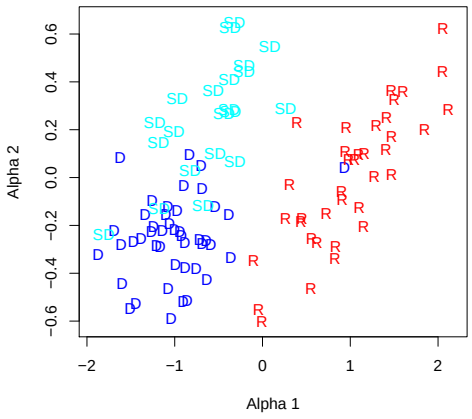


Figure 8: US Senate Scatter Plot, Custom Colors

```
ipe_scatter(ipe1,by=inddat$party,bys=c("D","I","R"),cols=c('blue','black',
    'red'))
```

The results are clearly not informative, as there are too many points being plotted. One alternative is to examine contours instead (Figure 10):

```
ipe_contour(ipe1,by=party,bys=c("D","I","R"),cols=c('blue','black','red'))
```

The results indicate that there is a lot of overlap between the parties in the ideology of the mass public. An alternative approach would be to report a scatter plot of the state means (Figure 11):

```
ipe_scatter(ipe1,by=inddat$state,stat="mean")
```

The figure produces different insights than the previous two figures, focusing on state differences rather than party differences, but is useful for beginning to understand the patterns. The second dimension appears to separate Southern states from non-Southern states.

Example 6 (continued).

Below, we estimate two dimensional ideal points using only the State Legislative NPATs:

```
ipe8s <- ipe_start(npat$Y,2)
ipe8u <- ipe_pmle(npat$Y,2,ipe=ipe8s)
ipe8 <- ipe_normalize_by_group(ipe8u,npat$inddat$Party,list("D"=c(-1,0),"R
    "=c(1,0),"L"=c(1,4)),num=20)
ipe8$Alpha[ipe8$IndFit$IndTot<20] <- NA
```

In normalizing the ideal points, we postulate that one dimension will divide the Democratic and Republican parties and one dimension will divide the Libertarian party from the major parties. The code below reports a scatter plot of the ideal points:

```
ipe_scatter(ipe8,by=npat$inddat$Party,bys=c("D","R","L"),cols=c('blue','
    red','yellow'))
```

Figure 12 shows a clear pattern, but is somewhat hard to decipher due to the large number of Democratic and Republican candidates. The code below only plots 10% of the Democratic and Republican candidates for clarity:

```
ipe_scatter(ipe8,by=npat$inddat$Party,bys=c("D","R","L"),cols=c('blue','
    red','yellow'),subsamp=c(.1,.1,1))
```

Alternatively, one can rely on a contour plot:

```
ipe_contours(ipe8,by=npat$inddat$Party,bys=c("D","R","L"),cols=c('blue','
    red','yellow'))
```

Both Figures 13 and 14 demonstrate a very sharp divide between the Libertarian candidates and the major party candidates along the second dimension, with the Libertarian candidates aligned with the Republican candidates along the first dimension.

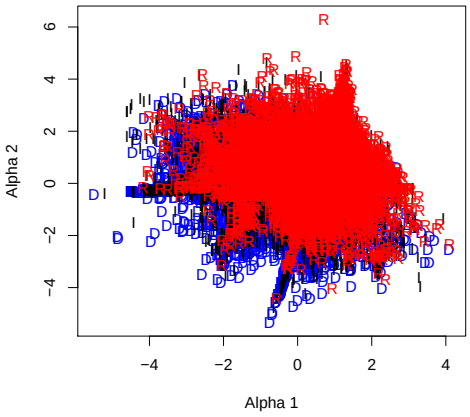


Figure 9: NAES Scatter Plot

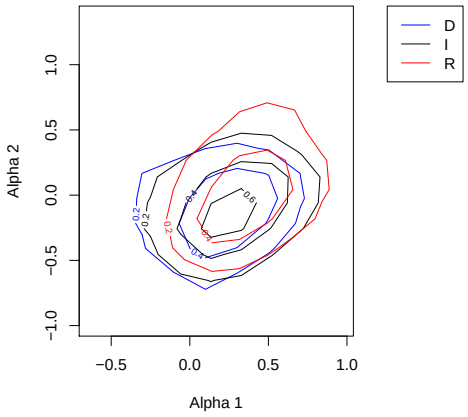


Figure 10: NAES Contours

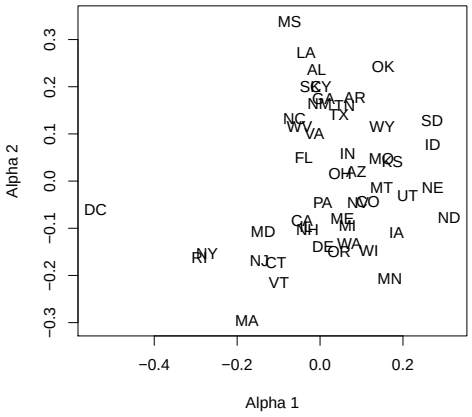


Figure 11: NAES State Means

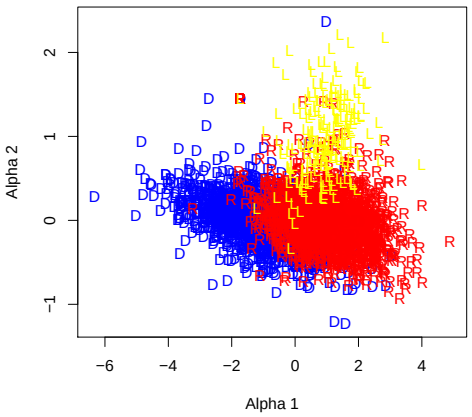


Figure 12: Scatter Plot, 2D State Legislative NPAT Ideal Points

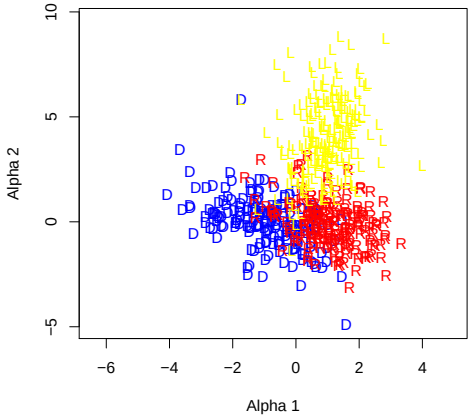


Figure 13: Scatter Plot, 2D State Legislative NPAT Ideal Points (10% sample used for Dem. and Rep. candidates)

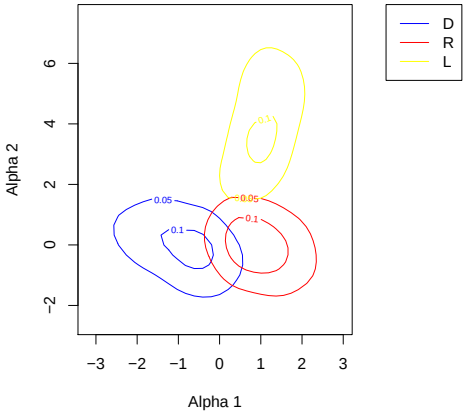


Figure 14: Contours, 2D state Legislative NPAT ideal points

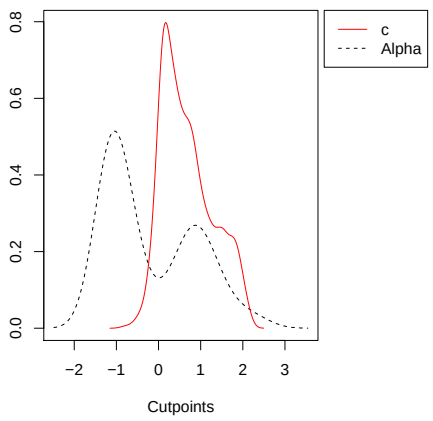


Figure 15: CA House Cutpoints Compared to Chamber Density

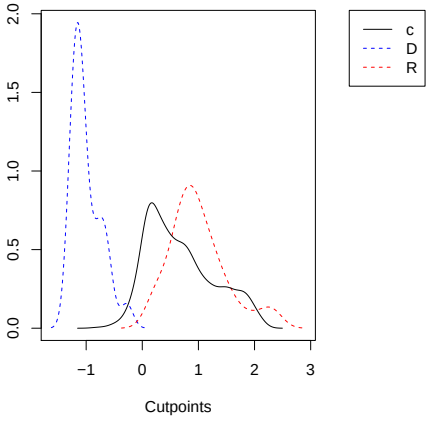


Figure 16: CA House Cutpoints Compared to Party Densities

4.2 Visualizing Cutting Lines

The function *ipe_cutlines* can plot cutpoints (when $D = 1$), cutting lines (when $D = 2$), and cutting line angles (when $D = 2$). The function takes in an ideal point object, as well as optional parameters. Optional parameters are listed in Table 5.

Example 1 (continued).

When $D = 1$, the following command can be used to plot the cutpoints (see Figure 15):

```
ipe_cutlines(ipe1)
```

A more informative version plots for reference the density of Democratic and Republican members of the California House (see Figure 16):

```
ipe_cutlines(ipe1,cols='black',byalpha=party,bysalpha=c("D","R"),colsalpha
    =c('blue','red'))
```

In the figure, the distribution of cutpoints indicates that the most common votes split the Democratic and Republican parties, but votes that split the Republican party are far more common than votes that split the Democratic party. Since the Democratic party held a majority of the seats, this finding is consistent with theories of majority party negative agenda setting power (Cox and McCubbins, 2005).

Example 2 (continued).

When $D = 2$, *ipe_cutlines* allows plotting the cutting lines and cutting line angles, along with the ideal points for reference. The command below plots the cutpoints (see Figure 17):

```
ipe_cutlines(ipe1)
```

Because there are large number of cutting lines, the plot is largely uninformative, though one can perhaps make out that there are more cutting lines that divide legislators along the second dimension. One option is to report a random subset of the cutting lines (Figure 18):

```
ipe_cutlines(ipe1,subsamp=.1,cols='black',byalpha=party2,bysalpha=c("D","
    SD","R"),colsalpha=c('blue','cyan','red'))
```

A random 10% of the cutting lines are reported. In addition, the ideal points are reported by party. These results are reported in Figure 18. A majority of votes in the 90th Senate appear to split the Southern and non-Southern Democrats, rather than splitting the two major parties. An alternative approach to obtaining more interpretable results is to report the distribution of cutting line angles (see Figure 19):

```
ipe_cutlines(ipe1,angle=T)
```

The cutting line angles range from -90 to 90, with 0 indicating a cutting line that divides legislators along the first dimension and -90 and 90 both indicating cutting lines that divide the legislators along the second dimension, with positive (negative) angles corresponding to positive (negative) slopes in Figure 18. The data here a circular in nature (Gill and Hangartner, 2010) and the presentation of the data reflect the fact that cutting line angles of -89 and 89 represent very similar cutting lines. The third panel of the figure again demonstrates the surprising pattern that votes that split the Southern and non-Southern Democrats where most common in the 90th Senate. Note that in comparing the cutting lines in Figure 18 with the cutting line angles in Figure 19, the visual angles of the former figure will depend on the x and y scale chosen for the figure.

Example 4 (continued).

In order to aid in the interpretation of the dimensions in the NAES data, we can code the items into categories. The items were coded as economic, social, or other. Economic items included preferences on taxation, social security spending, education spending, etc. Social items included preferences on abortion, the death penalty, effort to reduce discrimination, etc. Other issues included preferences related to campaign finance, the military, and the environment. The code below reports the results in two ways—using cutting lines and cutting line angles:

```
ipe_cutlines(ipe1,ref=F,by=chamber$itemdat$type)
ipe_cutlines(ipe1,ref=F,by=chamber$itemdat$type,angle=T,bw=20)
```

The cutting lines are reported in Figure 20. The economic cutting lines largely divide respondents along the first dimension. Though there is more variation among social and other cutting lines, there is some tendency for these cutting lines to divide people along the second dimension, with this begin more pronounced for other issues. The cutting line angles are reported in Figure 20. Here, we can again see that economic issues tend to divide respondents along the first dimension. The peak

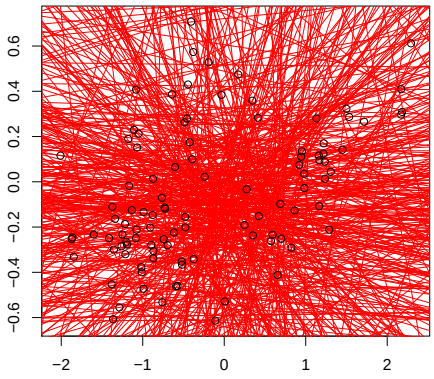


Figure 17: US Senate, All Cutting Lines

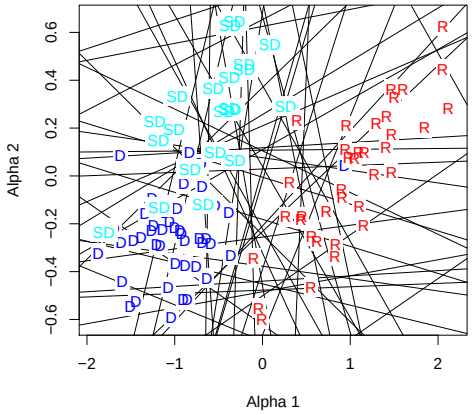


Figure 18: US Senate, Random Sample of Cutting Lines

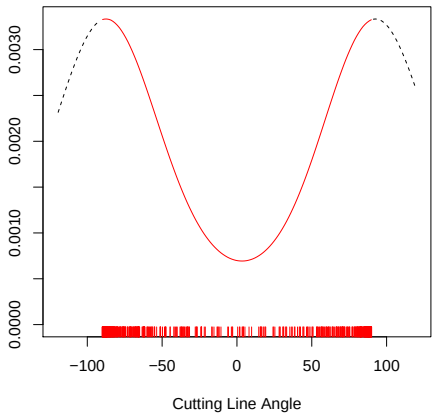


Figure 19: US Senate, Cutting Line Angles

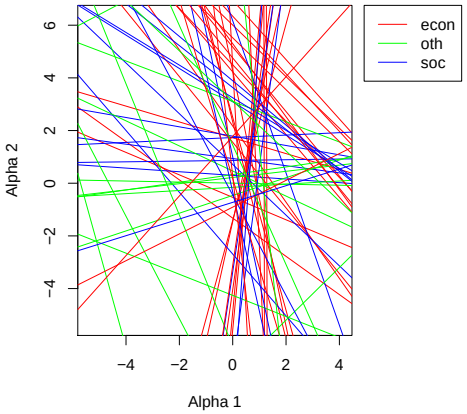


Figure 20: NAES, Cutting Lines

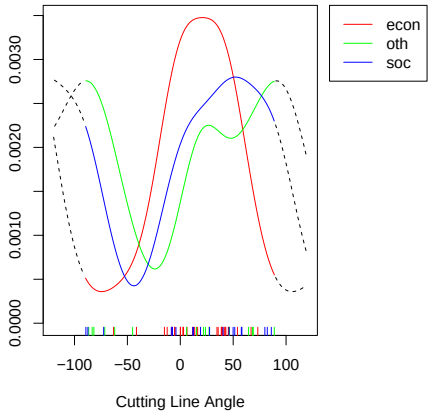


Figure 21: NAES, Cutting Line Angles

of the distribution for other cutting line angles is around 90 degrees suggesting that the average cutting line divides voters along the second dimension. For social issues, the distribution peaks around 65 degrees. Overall, these results could be summarized as suggesting that the first dimension is economic and the second dimension is social, with the caveat that the prototypical social issues are different than what we might expect—instead of abortion and efforts to reduce discrimination coinciding directly with the second dimension, we find that preferences over campaign finance and the military coincide more directly.

4.3 Summary Statistics

The functions *ipe_median*, *ipe_mean*, and *ipe_var* compute summary statistics for the ideal points. *ipe_median* computes dimension-by-dimension medians, *ipe_mean* computes dimension by dimension means, and *ipe_var* computes variance covariance matrices. Each function takes in an ideal point object and optional parameters. *by* specifies a grouping. *bys* (default is all values of *by* sorted) determines the groups to report results for. *subset* applies the functions to a subset of the data. *num* (default value 0) specifies the minimum number of items for reporting ideals points. For example, if *num* is set to 10, only ideal points computed with at least 10 items with be included in the means, medians, and variances.

4.4 Assessing Dimensionality

ipe provides a number of tool for selecting the number of dimensions to estimate. *ipe_pmle* produces various fit statistics including the percent correctly predicted and the geometric mean probability. The ideal point object contains three elements, *Fit*, *IndFit*, and *ItemFit*, which contain relevant

information. *Fit* is a data frame with elements listed in Table 6, *IndFit* is a data frame with elements listed in Table 7, and *ItemFit* is a data frame with elements listed in Table 8.

The function *ipe_dim_fit* aids in comparing the overall fit statistics for multiple dimensions at once. The function takes in a data matrix and a maximum number of dimensions. The optional parameter *out* (default value false) specifies whether to compute the fit statistics out-of sample. By default, *ipe_dim_fit* will use all observation in both estimation and measuring fit. If *out* is set to true, a hold-out sample is set aside, estimation in done in sample, and measuring fit is computed out-of-sample.[6] The optional parameter *frac* (default value 0.8) determine how much of the sample to use for estimation, with 0.8 indicating that 80% of the sample is use for estimation and 20% of the sample is the hold-out sample. Each observation (an individual's response on a particular item) has a *frac* percent chance of appearing in the training sample and a 1 - *frac* percent chance of appearing in the holdout sample.

Example 2 (continued).

The following command can help assess dimensionality of the 90th Senate in sample:

```
ipe_dim_fit(Y,10)
```

The results are given in Figure 22. Exactly how to use this information is left up to the user, but applying the elbow rule to the figure suggests a two-dimensional model is appropriate. The following code can help assess dimensionality of the 90th Senate out-of-sample:

```
ipe_dim_fit(Y,10,out=T)
```

The results are given in Figure 23. Applying the elbow rule to the out-of-sample fit similarly suggests a two dimensional model. The 6-dimensional model provides the best fit, though the improvement over the two-dimensional model is 2.0 percentage points and 0.9 percentage points relative to the 3-dimensional model.

Applying the elbow rule (to either in-sample or out-of-sample fit statistics) provides one approach for selecting the number of dimensions. Strictly choosing the best fit in the hold out sample provides a second approach. A third approach (illustrated in Sections 4.1 and 4.2) involves including dimensions that can be interpreted. For example, Examples 2 and 4 clearly point to at least two dimensions being present. In Example 6, the second dimension picked up differences between Libertarian and major party candidates. This was despite the fact that there were many fewer Libertarian candidates in the sample, suggesting that ability to improve fit among the average major party candidate was much smaller than the ability to improve fit among the average Libertarian candidate. Unless we are inherently interested in the Libertarian candidates, this suggest a strong case for a one-dimensional model for state legislative candidates.

[6] This particular approach to cross-validation is developed in Peress and Spirling (2010).

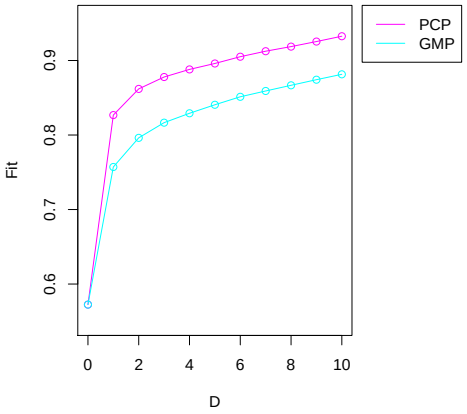


Figure 22: 90th Senate, Fit, In-Sample

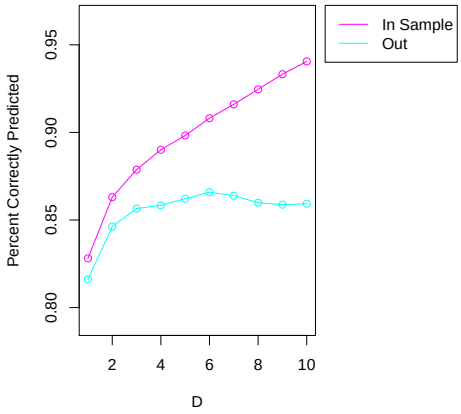


Figure 23: 90th Senate, Fit, Out-of-Sample

References

Battista, James, Michael Peress and Jesse Richman. 2013. "Common Space Ideal Points, Committee Assignments, and Financial Interests in the State Legislatures." *State Politics and Policy Quarterly* 13:70–87.

Battista, James, Michael Peress and Jesse Richman. Forthcoming. "Estimating the Locations of Voters, Legislators, Policy Outcomes, and Status Quos on a Common Scale." *PoliticalScienceResearchMethods* .

Cox, Gary W. and Matthew D. McCubbins. 2005. *Setting the Agenda: Responsible Party Government in the U.S. House*. Cambridge: Cambridge University Press.

Dennis, John E. and Robert B. Schnabel. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs: Prentice-Hall.

Ford, L. R. and D. R. Fulkerson. 1956. "Maximal Flow through a Network." *Canadian Journal of Mathematics* 8:399–404.

Gill, Jeff and Dominik Hangartner. 2010. "Circular Data in Political Science and How to Handle It." *Political Analysis* 18:316–336.

Kaiser, Henry F. 1958. "The Varimax Criterion for Analytic Rotation in Factor Analysis." *Psychometrika* 23:187–200.

Peress, Michael. 2013. "Estimating Proposal and Status Quo Locations using Voting and Cosponsorship Data." *Journal of Politics* 75:613–631.

Peress, Michael. forthcoming. "Large Scale Ideal Point Estimation." *Political Analysis* .

Peress, Michael and Arthur Spirling. 2010. "Scaling the Critics: Uncovering the Latent Dimensions of Movie Criticism." *Journal of the American Statistical Association* 105:71–83.

Saad, Yousef. 2003. *Iterative Methods for Sparse Linear Systems*. Society for Industrial Mathematics.

Shor, Boris, Nolan McCarty and Christopher Berry. 2008. "Methodological Issues in Bridging Ideal Points in Disparate Institutions in a Data Sparse Environment." Working Paper.

Skiena, Steven S. 1997. *The Algorithm Design Manual*. Springer.

Wright, Gerald. 2004. "Representation in America's Legislatures." Indiana University: National Science Foundation Grant.

Wright, Gerald. 2007. "Representation in the America's Legislature Project.". `https://doi.org/10.7910/DVN/LFULHR`.

A Optional Parameters

Variable	Default	Description
d	-	The dimension to compute the density along. When $D > 1$, d (an integer between 1 and D) must be supplied.
by		A grouping of the data, for example, party.
bys		A list of groups to report results for, with values of *by* not found in groups ignored. If not provided, groups will be a list of all unique values in *group* appearing in alphabetical order
labs		A labeling applied to the groups, which override the default of using the values from *by* as their own labels
cols		Line colors of density plots—a scalar if *by* is not provided, and a vector with the same length as *bys* otherwise
ltys		Line types of density plots—a scalar if *by* is not provided, and a vector with the same length as *bys* otherwise
num	0	Ideal point estimated from a sample of less than *num* items are dropped from the analysis
xlim		A two-element vector containing the x-axis limits of the plot
ylim		A two-element vector containing the y-axis limits of the plot
xlab		A label for the x-axis of the plot
bw		Bandwidth to use in Kernel density estimation (the same bandwidth is applied to all groups, except for groups of size less than 10, which are skipped over)
subset		A subset of the data to analyze
rug	true	Include a rug plot with the density
main		Label for the figure
inset	$(-.4, 0)$	Spacing for the legend

Table 2: Optional parameters for *ipe_density*

Variable	Default	Description
d1, *d2*		The dimensions to compute the scatter plot along. When $D > 2$, *d1* and *d2* (integers between 1 and D) must be supplied.
by		A grouping of the data, for example, party.
bys		A list of groups to report results for, with values of *by* not found in groups ignored. If not provided, groups will be a list of all unique values in *group* appearing in alphabetical order
xlim		A two-element vector containing the x-axis limits of the plot
ylim		A two-element vector containing the y-axis limits of the plot
cols		Line colors of density plots—a scalar if *by* is not provided, and a vector with the same length as *bys* otherwise
main		Label for the figure
subsamp	1	Determines whether to report only a sub-sample of points in the scatter plot. If scalar, must be a probability between 0 and 1. If a vector is supplied, different groups are reported with different probabilities.
stat	""	If set equal to "mean" or "median", *ipe_scatter* will report group means of medians, respectively.
xlab		A label for the x-axis of the plot
ylab		A label for the y-axis of the plot
subset		A subset of the data to analyze

Table 3: Optional parameter for *ipe_scatter*

Variable	Default	Description
d1, *d2*		The dimensions to compute the scatter plot along. When $D > 2$, *d1* and *d2* (integers between 1 and D) must be supplied.
by		A grouping of the data, for example, party.
bys		A list of groups to report results for, with values of *by* not found in groups ignored. If not provided, groups will be a list of all unique values in *group* appearing in alphabetical order
xlim		A two-element vector containing the x-axis limits of the plot
ylim		A two-element vector containing the y-axis limits of the plot
labs		A labeling applied to the groups, which override the default of using the values from *by* as their own labels
cols		Line colors of density plots—a scalar if *by* is not provided, and a vector with the same length as *bys* otherwise
ltys		Line types of density plots—a scalar if *by* is not provided, and a vector with the same length as *bys* otherwise
num	0	Ideal point estimated from a sample of less than *num* items are dropped from the analysis
bw		Bandwidth to use in Kernel density estimation (the same bandwidth is applied to all groups, except for groups of size less than 10, which are skipped over)
levels	3	Number of levels of contours to report
inset	$(-.4, 0)$	Spacing for the legend
subset		A subset of the data to analyze

Table 4: Optional parameter for *ipe_contour*

Variable	Default	Description
by		A grouping of the data, for example, party.
bys		A list of groups to report results for, with values of *by* not found in groups ignored. If not provided, *bys* will be a list of all unique values in *group* appearing in alphabetical order
labs		A labeling applied to the groups, which override the default of using the values from *by* as their own labels
cols		Line colors of density plots—a scalar if *group* is not provided, and a vector with the same length as *bys* otherwise
ltys		Line types of density plots—a scalar if *group* is not provided, and a vector with the same length as *bys* otherwise
num	0	Ideal point estimated from a sample of less than *num* items are dropped from the analysis
xlim		A two-element vector containing the x-axis limits of the plot
ylim		A two-element vector containing the y-axis limits of the plot
xlab		A label for the x-axis of the plot
ylab		A label for the y-axis of the plot
bw		Bandwidth to use in Kernel density estimation (the same bandwidth is applied to all groups, except for groups of size less than 10, which are skipped over)
droplow, drophigh		Drop cutlines outside the range.
ref	true	If TRUE, Include a scatter plot of the ideal points for comparison.
nby	10	Results are not reported for groups of size less that *nby*.
dither	0	Standard deviation of random normal noise to add to cutting lines and ideal points.
angle	false	If FALSE, producing cutting line plot, if TRUE, produce cutting angle density plot.
byalpha		Grouping variable for the ideal points, α.
bysalpha		Groups for the ideal points.
colsalpha		Colors for α groups.
ltysalpha		Line types for α groups.
subsamp	1	Include cutting lines with probably *subsamp*. For example, if *subsamp* = .1, only 10% of cutting lines are included. Has no effect if *angle* is set to TRUE.
inset	$(-.4, 0)$	Spacing for the legend

Table 5: Optional parameter for *ipe_cutlines*

Variable	Description
Objective	The minimized (normalized) penalized log-likelihood
BasePerCorr1	Percent correctly predicted when all individuals chose the modal response on each item
BasePerCorr2	Percent correctly predicted when all individuals vote like the modal individual
Tot	Total number of observations (i.e. the number of non-missing elements in Y
TotCorr	Total number of correct predictions
TotPerCorr	Percent correctly predicted
APRE1	Average Percent Reduction in Error, using *BasePerCorr1* as a baseline
APRE2	Average Percent Reduction in Error, using *BasePerCorr2* as a baseline
GeoMeanProb	Geometric Mean Probability

Table 6: Elements of *Fit*

Variable	Description
IndTot	Number of observations for each individual
IndCorr	Number of correct predictions for each individual
IndPerCorr	Percent correctly predicted for each individual
IndGeoMeanProb	Geometric Mean Probability for each individual

Table 7: Elements of *IndFit*

Variable	Description
ItemTot	Number of observations for each item
ItemCorr	Number of correct predictions for each item
ItemPerCorr	Percent correctly predicted for each item
ItemGeoMeanProb	Geometric Mean Probability for each item

Table 8: Elements of *ItemFit*