Reaper: More Simulations for the "Thresher" Paper

Kevin R. Coombes

29 November 2013

Contents

List of Figures

1 Executive Summary

1.1 Introduction

This report describes the (second) analysis of simulated data sets to test the behavior of our proposed methods for analyzing continuous pathway data.

1.1.1 Aims/Objectives

We want to see whether the methods can identify the correct number of protein clusters (which should be between 1 and 4 in our simulated datasets).

1.2 Methods

1.2.1 Description of the Data

In the previous report, we simulated and saved 2500 datasets with a few proteins (around 10–20) and many samples (median: 304, range: 126–506). Each dataset exhibits either one or two independent correlated signals. Each signal can be unsigned (all proteins are positively correlated, so a reasonable summary would be a simple average of all proteins) or signed (including both positively and negatively correlated proteins, so a reasonable summary requires looking at a difference between two group averages). Each dataset also contains two "noise" genes that are not correlated with any of the simulated signals.

1.2.2 Statistical Methods

We use the "Thresher" algorithm described in the previous report, with a cutoff $\Delta \leq 0.3$, to detect outliers or "noise" proteins. We use the Auer-Gervini approach to estimate the number K of signfiicant principal components. We fit a mixture of von Mises - Fisher distributions to cluster the protein directions (on a unit sphere) into $N = K, K+1, \ldots, 2K+1$ protein groups. To select the optimal number of protein groups, we compute the Bayesian Information Criterion (BIC) for each N; the best number corresponds to the minimum BIC.

1.3 Results

- The estimated number of principal components is (a) always correct if the true dimension equals 1 and (b) is correct 83% of the time when the true dimension equals 2 (**Section 4.1**).
- When clustering in the space of principal component loadings, the estimated number of protein groups is correct 68% of the time (**Section 4.2**). If you only consider situations where the PC dimenion was correctly estimated, then the number of protein groups is correct 73% or the time.
- When clustering in the complete protein-sample space, the estimated number of protein groups is correct 81% of the time (**Section 4.3**). If you only consider situations where the PC dimenion was correctly estimated, then the number of protein groups is correct 89% or the time.
- After removing outliers and estimating the number of protien groups, the plots give a clearer idea of the true underlying structure. (For loadings, compare **Figure 6** to **Figure 7**. For heatmaps, compare **Figure 3** to **Figure 8**. For samples in principal component space, compare **Figure 5** to **Figure 9**.)

1.4 Conclusions

The Thresher-Reaper methods provide effective tools for removing outliers and deteminein
g the correct number of protein groups in (simulated) data sets containing about 10–20 proteins.

2 Preliminaries / Methods

2.1 Library Packages

We start by loading all of the R library packages that we need for this analysis.

```
> library(Thresher)
> library(RColorBrewer)   # for sensible color schemes
```

2.2 The Data Sets

Next, we load the simulated datasets from the first report.

```
> load("savedSims.rda")
> class(savedSims)
```

```
[1] "list"
```

```
> length(savedSims)
```

```
[1] 2500
```

3 Three Examples

We run the following loop of code to create the five standard figures for several different sample datasets.

```
> if (!file.exists("SimFigs")) {
+   dir.create("SimFigs")
+   for (idx in 1:40) { # really, do not do 2500 of these ...
+     makeFigures(savedSims[[idx]], DIR="SimFigs")
+   }
+ }
```

We now plot a series of standard figures for a (related) trio of example datasets. **Figure 1** shows the (true) correlation matrices that were used to simulate the data. All three datasets contain two groups of proteins, and these are the same size in each dataset. In the dataset on top, all proteins within a group are positively correlated; in the datasets in the middle and on the bottom, some proteins are positively and some are negatively correlated.

Figure 3 presents clustered heatmaps of the three simulated datasets. In all cases, simulated proteins P21 and P22 are outliers or "noise" proteins (which are forced to cluster somewhere). For the samples, however, the main structure seems to be the "off" or "on" status of different sets of genes. Interestingly, the top two figures look similar *even though we know the underlying structure is different*. In the top dataset, all proteins within a "signal group" are positively correlated, so we expect the "blue"-or-"red" and "green" proteins to mark the two clusters of "signal" proteins. In the middle dataset, by contrast, we expect to have both postive and negative correlation. So, we expect to see four clusters of proteins (one for the positive and one for the negative of each of the two signal groups). While there are apparently four protein clusters, it is not clear how to get the correlation information from this plot. The bottom "mixed" dataset correctly shows three groups of proteins, with the "blue" and "red" being negatively correlated and the "green" being independent of the others.

Figure 2 shows the Auer-Gervini step function that was used to determine the number of significant principal components.

Figure 4 contains scree plots for the PCA on each of the three datasets. The overlaid blue curve shows the expected values from the broken stick model. As expected from how we know the data were simulated, we see two significant components in each dataset.

Figure 5 is a plot of the samples from the PCA on each dataset. Colors assigned to the samples in this plot are the same as the color bars in the heatmaps (**Figure 3**). How to interperet the structure from these plots is not clear.

Figure 6 illustrates the loadings of each protein on the first two principal ccomponents. This figure gives us the clearest picture of the known structure. The top plot (of the unsigned dataset) shows two independent groups of proteins; the middle plot, four groups; the bottom plot, three groups. In the middle and bottom plots, we see that a contrast between the "positive – blue" group and "negative – red" group represents one factor. The other factor comes from either the "green" group alone (bottom) or a contgrast between the "green" and "purple" groups (middle),

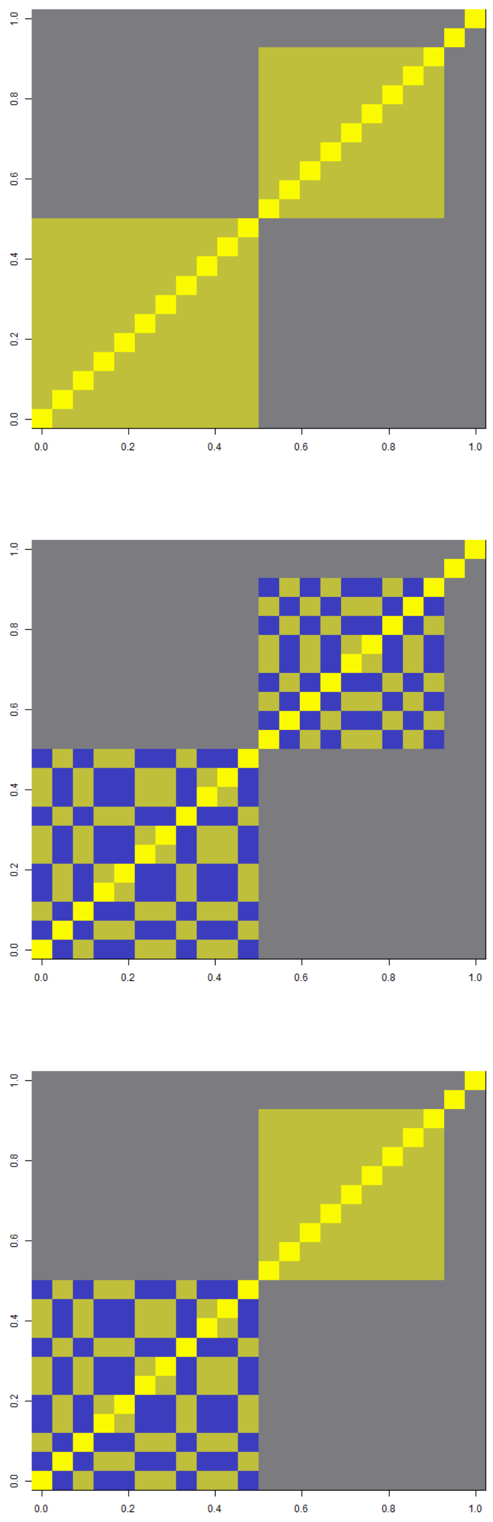


Figure 1: Covariance matrices for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

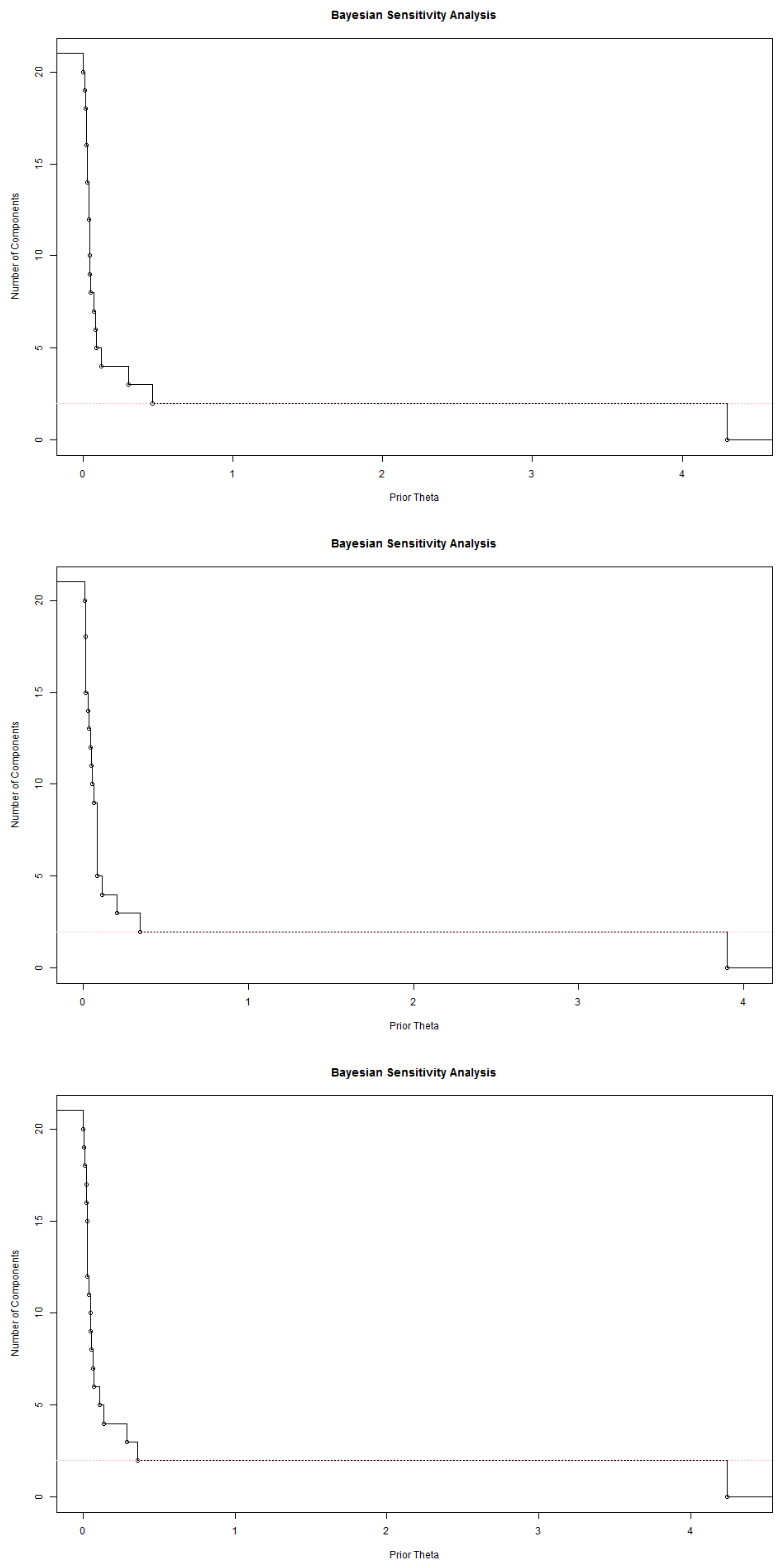


Figure 2: ; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

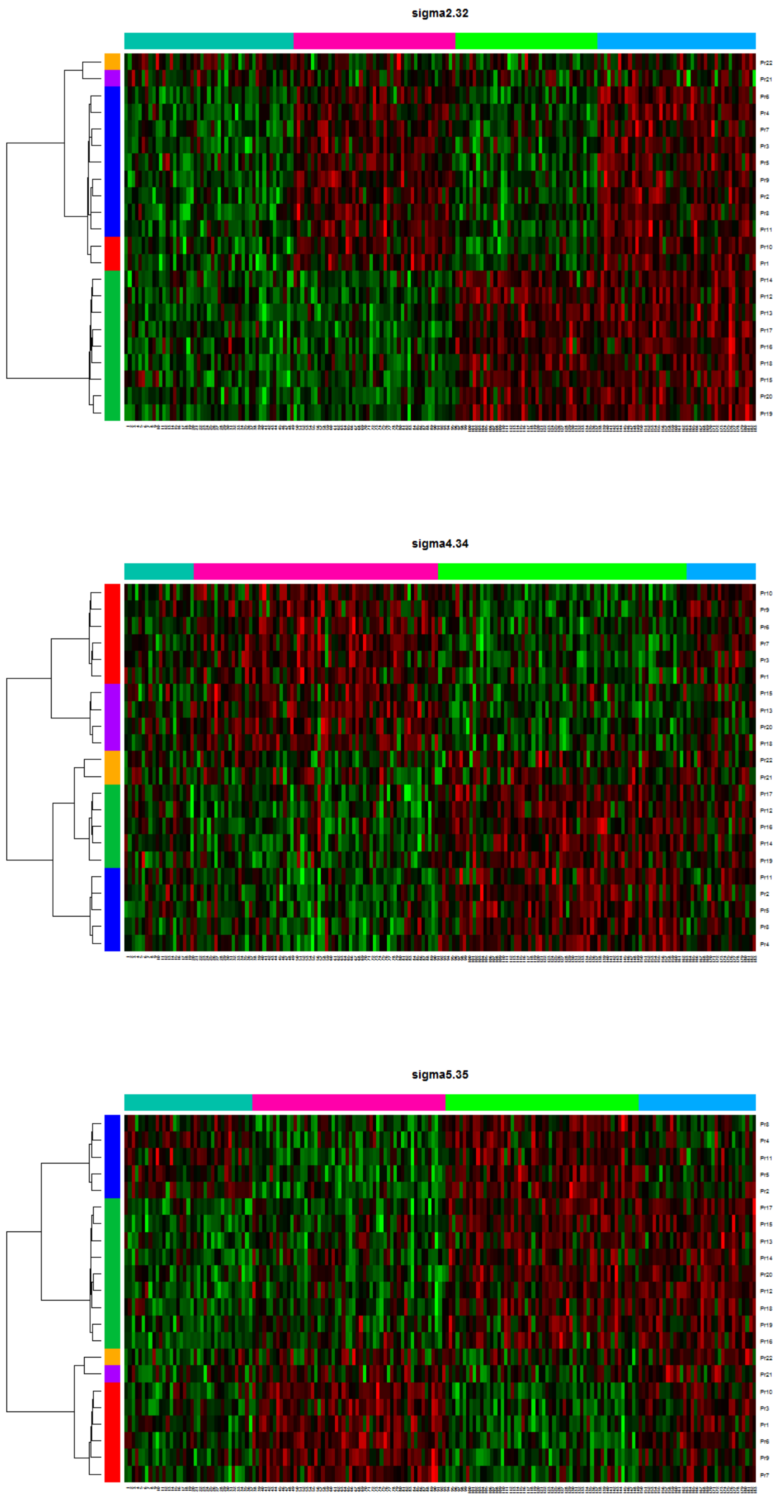


Figure 3: Clustered heatmaps for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

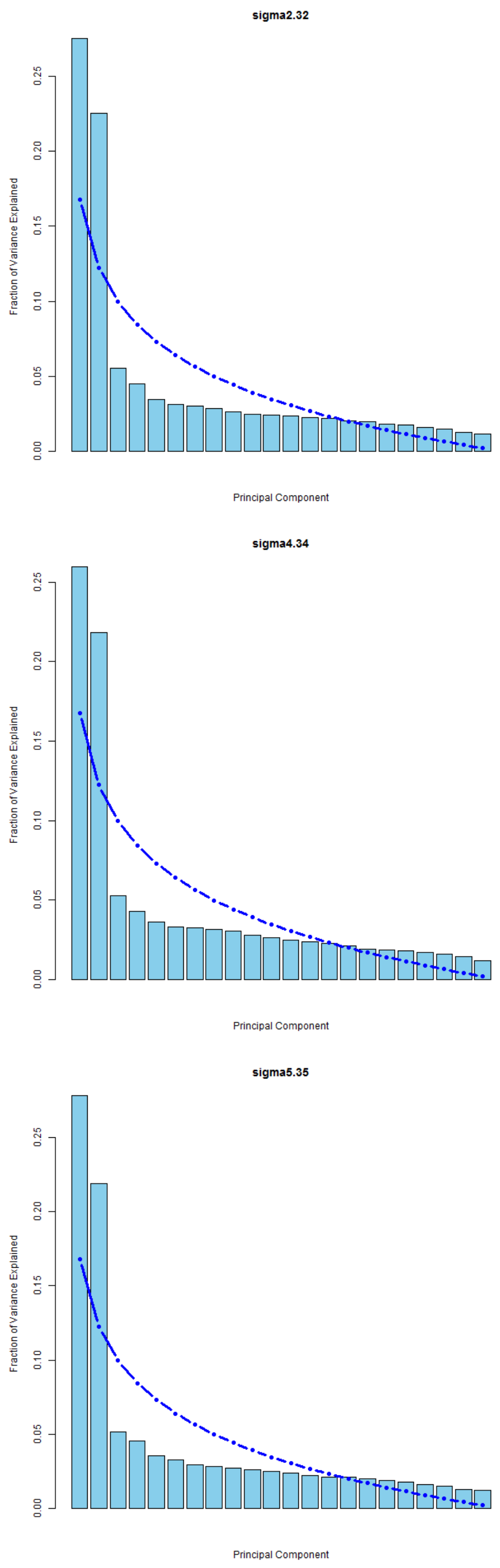
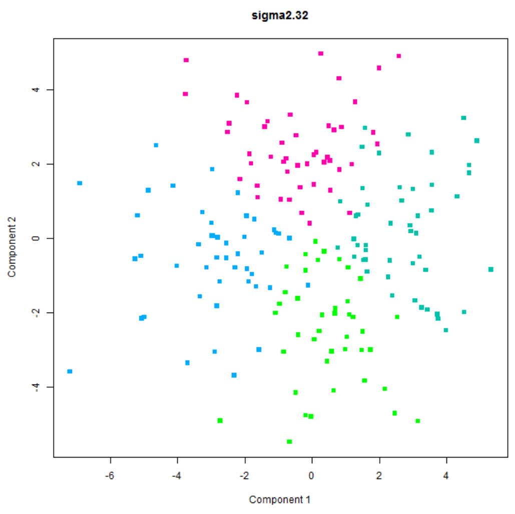
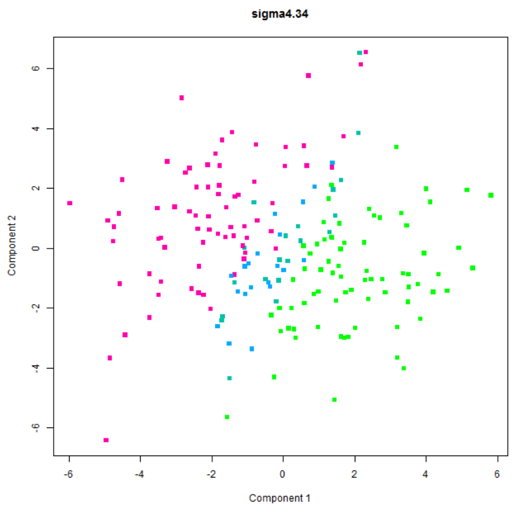


Figure 4: Screeplots for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.





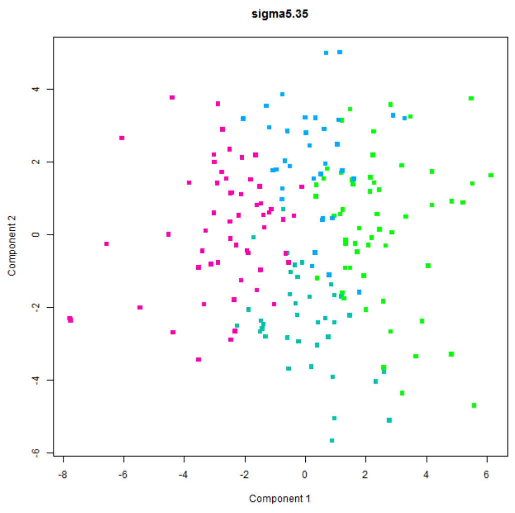


Figure 5: Principal components scatterplot for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

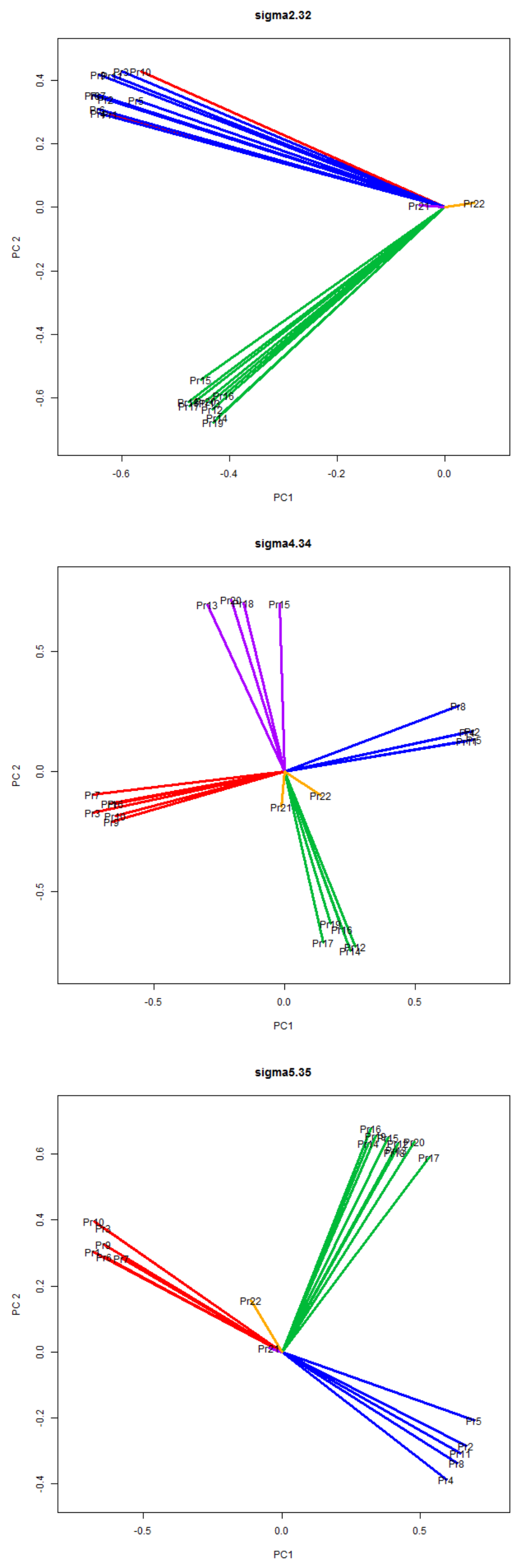


Figure 6: PCA loadings plots for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

4 Finding Protein Groups

We repeat the following block of code from the previous report in order to have a vector defining the true “type” of each simulated dataset.

```
> nSimSets <- length(savedSims)/5
> simpleType <- paste(rep(c("OneGroup", "TwoGroups"), times=2),
+                     rep(c("Unsigned", "Signed"), each=2), sep="")
> simpleType <- c(simpleType, "TwoGroupsMixed")
> rt <- rep(1:5, nSimSets)
> simType <- factor(simpleType[rt], levels=simpleType)
> typer <- rep(simType, each=2)
> evens <- sort(c(seq(2, 5*nSimSets, 5),
+                 seq(4, 5*nSimSets, 5),
+                 seq(5, 5*nSimSets, 5)))
> sim.type <- factor(simType[evens])
> rm(rt)
> summary(simType)

 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned
              500               500               500               500
   TwoGroupsMixed
              500

> summary(sim.type)

TwoGroupsUnsigned   TwoGroupsSigned    TwoGroupsMixed
              500               500               500
```

We first apply the `reaper` algorithm to the directions in PC space.

```
> f <- "vmfMixturesLoaded.rda"
> if(file.exists(f)) {
+   load(f)
+ } else {
+   set.seed(743634)
+   vmfMixturesLoaded <- lapply(savedSims, Reaper, useLoadings=TRUE,
+                               method="auer.gervini")
+   save(vmfMixturesLoaded, file=f)
+ }
> rm(f)
```

Next, we apply the algorithm in the full protein-sample space.

```
> f <- "vmfMixtures.rda"
> if(file.exists(f)) {
+   load(f)
+ } else {
+   set.seed(115234)
+   vmfMixtures <- lapply(savedSims, Reaper, useLoadings=FALSE,
+                         method="auer.gervini")
+   save(vmfMixtures, file=f)
+ }
> rm(f)
```

4.1 Number of Principal Components

Since both applications use the same code to determine the correct PC dimension, K, we want to see how this compares both to the value before removing outliers. and to the true value.

```
> pcDimension <- sapply(vmfMixtures, function(x) x@pcdim)
> pcDimension0 <- sapply(savedSims, function(x) x@pcdim)
> table(pcDimension, pcDimension0)

           pcDimension0
pcDimension    1    2    4
          1 1081    0    0
          2    0 1410    0
          4    0    0    9
```

Only five out of the 2500 simulated datasets have the estimated dimension changed when removing outliers. This finding is not terribly surprising, since we saw in the previous report that the main explanation of the failure to find the correct dimension was attributable to few signal proteins and small correlation, neither of which has anything to do with the outliers.

Here are the “true” dimensions, which we know because we have simulated these datasets.

```
> trueDim <- c(1, 2, 1, 2, 2)
> names(trueDim) <- simpleType
> trueDim

 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned 
                1                 2                 1                 2 
   TwoGroupsMixed 
                2 

> trueDimensions <- trueDim[as.character(simType)]
```

To be thorough, we check that, for all five times where removing the outliers caused a change, it actually changed to the correct answer.

```
> temp <- data.frame(pcDimension0, pcDimension, trueDimensions)
> temp[pcDimension> pcDimension0,]

[1] pcDimension0   pcDimension    trueDimensions
<0 rows> (or 0-length row.names)
```

We can also compare the estimated dimensions to the true dimensions as a function of the type of simulated dataset.

```
> table(simType, pcDimension)

                   pcDimension
simType               1   2   4
  OneGroupUnsigned  500   0   0
  TwoGroupsUnsigned  26 471   3
  OneGroupSigned    500   0   0
  TwoGroupsSigned    29 469   2
  TwoGroupsMixed     26 470   4
```

Here we compute the accuracy rate for each type.

```
> accuDim <- sapply(simpleType, function(s) {
+   results <- pcDimension[simType==s]
+   mean(results == trueDim[s])
+ })
> accuDim

 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned
            1.000             0.942             1.000             0.938
   TwoGroupsMixed
            0.940

> mean(accuDim)

[1] 0.964

> mean(ok <- trueDimensions==pcDimension)

[1] 0.964
```

So, the accuracy when the correct dimension is 1 is 100%, while the accuracy when the correct dimension is 2 is only 83%.

4.2 Number of Protein Groups: PC Loadings

Now we explore how often clustering the proteins (using a mixture of von Mises - Fisher distributions) in principal component space gets the correct number of protein groups.

```
> ngL <- sapply(vmfMixturesLoaded, function(x) x@nGroups)
> table(simType, ngL)
```

```
                   ngL
simType               1   2   3   4   5
  OneGroupUnsigned  381  58  61   0   0
  TwoGroupsUnsigned  16 354  52  29  48
  OneGroupSigned      0 394 106   0   0
  TwoGroupsSigned     0  32  90 347  31
  TwoGroupsMixed      5  59 342  52  42
```

Of course, we want to compare this formally to the true values, which are given by:

```
> trueGroups <- c(1, 2, 2, 4, 3)
> names(trueGroups) <- simpleType
> trueGroups
```

```
 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned 
                1                 2                 2                 4 
   TwoGroupsMixed 
                3 
```

We need to measure “accuracy” in two ways. First, we look at the complete method (which is likely to work poorly when it gets the PC dimension wrong).

```
> accurall <- sapply(simpleType, function(s) {
+   results <- ngL[simType==s]
+   sum(results == trueGroups[s], na.rm=TRUE)/length(results)
+ })
> accurall
```

```
 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned 
            0.762             0.708             0.788             0.694 
   TwoGroupsMixed 
            0.684 
```

```
> mean(accurall)
```

```
[1] 0.7272
```

So, the overall accuracy is only about 68%. But the accuracy is slightly higher when there is only one group, and declines when there are more true groups. Since getting the dimension wrong only happens when there are more groups, we can compute the "conditional accuracy", which meaures how often we get the number of groups right after knowing that we have gotten the PC dimension right.

```
> condaccu <- sapply(simpleType, function(s) {
+   results <- ngL[ok & simType==s]
+   sum(results == trueGroups[s], na.rm=TRUE)/length(results)
+ })
> condaccu
```

```
 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned 
        0.7620000         0.7367304         0.7880000         0.7377399 
   TwoGroupsMixed 
        0.7170213 
```

```
> mean(condaccu)
```

```
[1] 0.7482983
```

This is slightly better, but still only yields an accuracy of 73%. It is, however, consistent across the simulation types, suggesting that the clustering works equally well in all types provided the dimension is identified correctly.

4.3 Number of Protein Groups: Protein-Sample Space

The alternative method performs the clustering in the full protein-sample space, not just in the truncated principal component space. The overall perfomance clearly looks better:

```
> ng <- sapply(vmfMixtures, function(x) x@nGroups)
> table(simType, ng)
```

```
                   ng
simType               1   2   3   4   5
  OneGroupUnsigned  477  11  12   0   0
  TwoGroupsUnsigned  24 444  15  16   1
  OneGroupSigned      0 478  22   0   0
  TwoGroupsSigned     0  67  60 352  21
  TwoGroupsMixed      9  66 393  21  11
```

as does the (full) accuracy:

```
> accurall <- sapply(simpleType, function(s) {
+   results <- ng[simType==s]
+   sum(results == trueGroups[s], na.rm=TRUE)/length(results)
+ })
> accurall

 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned 
            0.954             0.888             0.956             0.704 
   TwoGroupsMixed 
            0.786 

> mean(accurall)

[1] 0.8576
```

Using this method, the overall accuracy is about 82%. Again, the accuracy is slightly higher when there is only one group, and declines when there are more groups.

```
> condaccu <- sapply(simpleType, function(s) {
+   results <- ng[ok & simType==s]
+   sum(results == trueGroups[s], na.rm=TRUE)/length(results)
+ })
> condaccu

 OneGroupUnsigned TwoGroupsUnsigned    OneGroupSigned   TwoGroupsSigned 
        0.9540000         0.9426752         0.9560000         0.7462687 
   TwoGroupsMixed 
        0.8255319 

> mean(condaccu)

[1] 0.8848951
```

This is slightly better, yielding a conditional accuracy of 89%, which is consistent across the simulation types.

5 Examples Revisited

We no return to our earlie set of examples, and generate new figures after removing outliers and estimating the "true" number of protein groups. We plot the loadings (**Figure 7**), the revised heatmap (**Figure 8**) and the samples in principal component space (**Figure 9**).

```
> for (idx in 31:35) {
+   makeFigures(vmfMixtures[[idx]], "SimFigs")
+ }
```

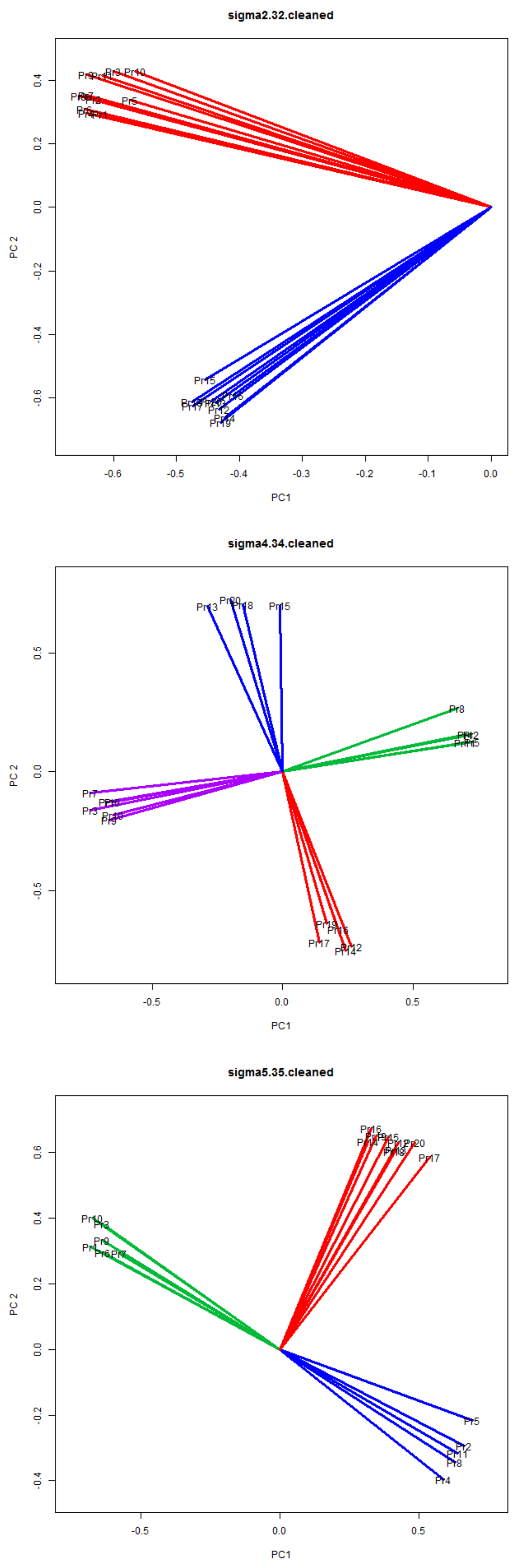


Figure 7: PCA loadings plots, after “reaping”, for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

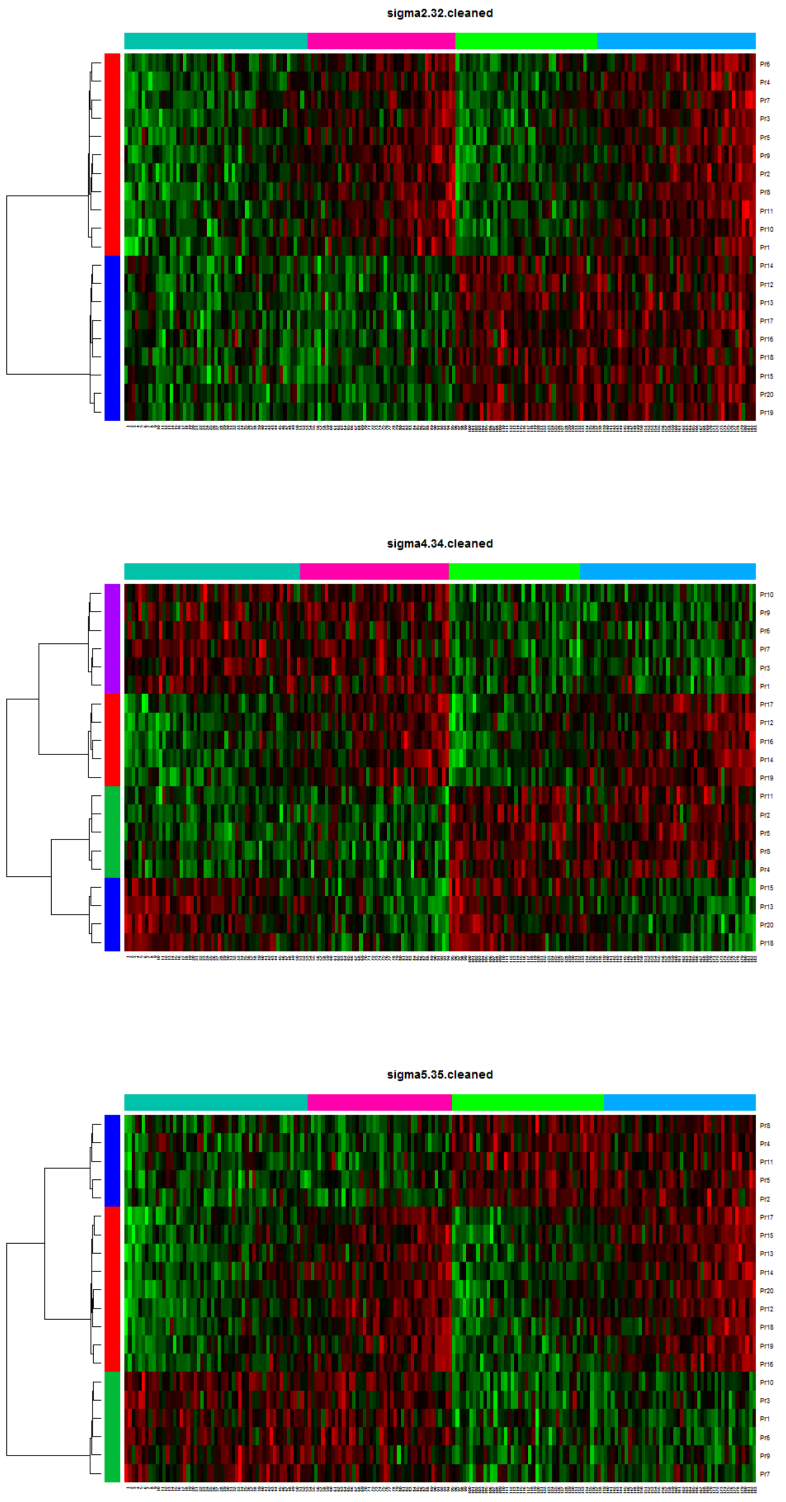


Figure 8: Heatmaps, after ”reaping”, for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

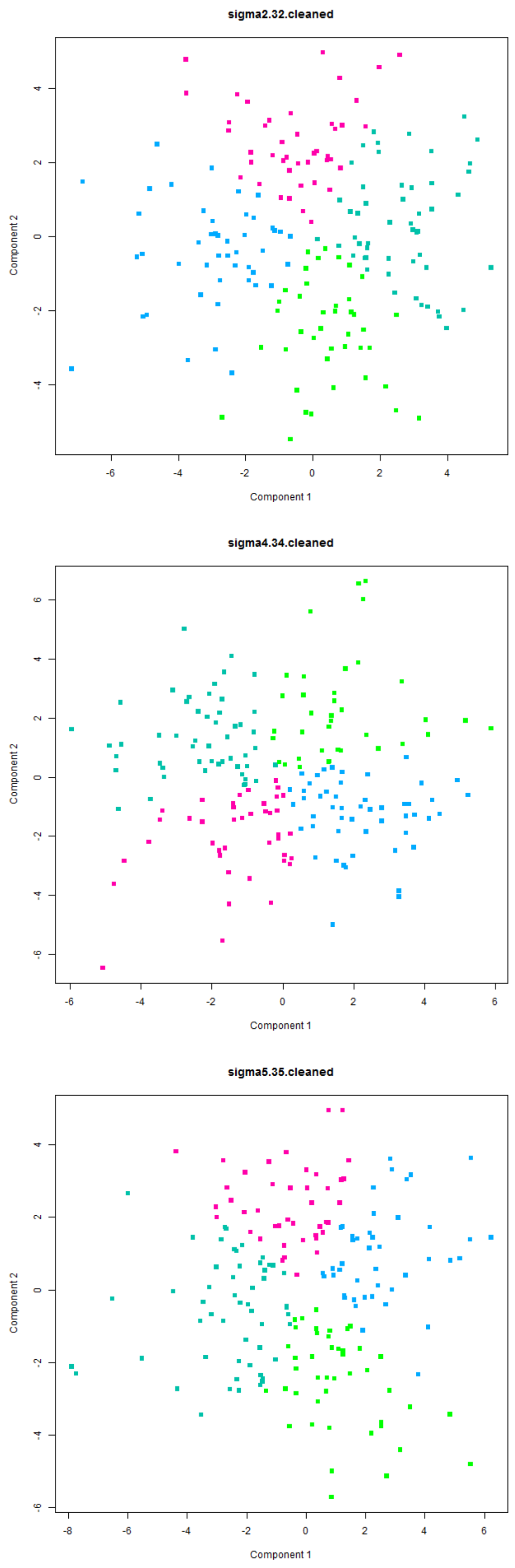


Figure 9: PCA scatter plots, afte ”reaping”, for example datasets with two groups of proteins; **(top)** unsigned, **(middle)** signed, and **(bottom)** mixed.

6 More Detailed Methods

Now we have the function that actually performs the main analysis.

```
> Reaper

function (thresher, useLoadings = FALSE, cutoff = 0.3, metric = NULL,
    verbose = TRUE, ...)
{
    if (verbose)
        cat(thresher@name, "\n", file = stderr())
    keep <- thresher@delta > cutoff
    m <- ifelse(is.null(metric), "pearson", metric)
    cleaned <- Thresher(thresher@data[, keep], paste(thresher@name,
        "cleaned", sep = "."), metric = m, ...)
    tab <- 0
    counter <- 0
    while (any(tab == 0) & counter < 5) {
        counter <- counter + 1
        if (verbose)
            print(counter)
        fits <- .fitModels(cleaned, useLoadings)
        if (length(fits) == 0)
            next
        bic <- sapply(fits, BIC)
        woo <- which(bic == min(bic))
        ng <- as.integer(sub("NC=", "", names(woo)))
        fit <- fits[[woo]]
        gassign <- factor(predict(fit), levels = 1:ng)
        tab <- table(gassign)
    }
    if (length(fits) == 0) {
        bic <- ng <- fit <- NA
        metric <- "no fit"
        sigset <- new("SignalSet")
    }
    else {
        if (is.null(metric)) {
            pp <- factor(paste("G", predict(fit), sep = ""))
            metric <- bestMetric(cleaned@data, pp)
            cleaned@gc <- hclust(distanceMatrix(cleaned@data,
                metric, p = 1), "ward")
        }
        if (any(tab == 0)) {
            sigset <- new("SignalSet")
        }
        else {
            sigset <- .findSignals(cleaned, fit, ng)
        }
    }
    new("Reaper", cleaned, useLoadings = useLoadings, keep = keep,
        nGroups = ng, fit = fit, allfits = fits, bic = bic, metric = metric,
        signalSet = sigset)
}
<environment: namespace:Thresher>
```

The algorithm used by the `reaper` function is as follows:

1. We start by using the `Thresher` method from the previous report. We use the cutoff determined there ($\Delta \leq 0.3$) to identify and remove outliers. (Recall that, based on these datasets, this cutoff should have a 2% false negative rate and a 0.1% false positive rate.)
2. Next, we apply the broken-stick model to the dataset with outliers removed to determine the correct number K of principal components to use to characterize the data.
3. As noted in the examples in **Section 3**, the number of protein groups should range between K and $2K$, depending on how many of the K signal protein groups include negative correlation. Because the estimation of K is likely to be slightly conservative, we actually allow the upper bound to go to $2K + 2$.
4. We now use just the directions/angles of the proteins (in the full space) of their loadings (in the K-dimensional principal component space). Since these directions are points on a (possibly high-dimensional) unit sphere, we model them as a mixture of von Mises - Fisher distributions. We use the Bayesian Information Criterion (BIC) to select the optimal number N of protein groups out of the range of candidates ($K \ldots 2K + 2$).

6.0.1 Plotting routines

We have a series of plotting routines for both "`Thresher`" and "`Reaper`" objects. This includes `makeFigures`, which is a wrapper that produces a complete set of five plots:

1. `image`, which only applies to simulated datasets, produces an image of the correlation matrix used in the simulations.
2. `screeplot` produces a "scree plot" of the amount of variance explained by each principal component (PC), with an overlay of the expected values from the broken stick model.
3. `plot` produces a plot of the loadings on each protein feature in PC space.

4. `scatter` produces a scatter plot of the samples in PC space.
5. `heat` produces a heatmap, in which the protein features are clustered but the samples are ordered to highlight the strongest signals in the data.

The ordering routine is critical, and this is different for "Thresher" objects than for "Reaper" objects. For "Threshers", samples are simply sorted by the sign of the first two principal components.

```
> getMethod("getSplit", "Thresher")

Method Definition:

function (object, ...)
{
    .local <- function (object)
    {
        colors <- rev(thresherPalette)[1:4]
        std <- scale(object@data)
        bb <- cutree(object@gc, k = 2)
        b1 <- apply(std[, bb == 1, drop = FALSE], 1, mean)
        b2 <- apply(std[, bb == 2, drop = FALSE], 1, mean)
        c1 <- 1 * (b1 > 0)
        c2 <- 1 * (b2 > 0)
        colset <- colors[1 + c1 + 2 * c2]
        fc <- factor(colset, levels = colors)
        op <- order(c1 + 2 * c2)
        list(fc = fc, op = op, colset = colset)
    }
    .local(object, ...)
}
<environment: namespace:Thresher>

Signatures:
        object
target  "Thresher"
defined "Thresher"
```

For "Reapers", the algorithm is more complex, since it uses the fact that protein features have already been clustered into groups (on a unit sphere in PC space, using mixtures of von Mises - Fisher distributions). We average the protein loadings for each group, and identify pairs that point in opposite directions and thus correspond to positively and negatively correlated members of the same PC.

```
> getMethod("getSplit", "Reaper")
```

```
Method Definition:

function (object, ...)
{
    .local <- function (object)
    {
        binSignal <- object@signalSet@binary
        contSignal <- object@signalSet@continuous
        nSig <- ncol(binSignal)
        weights <- matrix(2^(-1 + (1:nSig)), ncol = 1)
        sclass <- binSignal %*% weights
        op <- order(sclass, contSignal[, 1])
        colorscheme <- .makeColorScheme(2^nSig)
        colset <- colorscheme[1 + sclass]
        fc <- factor(colset, levels = colorscheme)
        list(fc = fc, op = op, colset = colset)
    }
    .local(object, ...)
}
<environment: namespace:Thresher>

Signatures:
        object
target  "Reaper"
defined "Reaper"
```

7 Appendix

This analysis was run in the following directory:

```
> getwd()
```

```
[1] "d:/Work/Reaper/Manuscript"
```

This analysis was run in the following software environment:

```
> sessionInfo()
```

```
R version 3.0.0 (2013-04-03)
Platform: x86_64-w64-mingw32/x64 (64-bit)

locale:
[1] LC_COLLATE=English_United States.1252  LC_CTYPE=English_United States.1252
```

```
[3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] RColorBrewer_1.0-5   Thresher_0.8.8       ClassDiscovery_3.0.0 oompaBase_3.0.1
 [5] mclust_4.0           cluster_1.14.4       ade4_1.5-2           movMF_0.1-2
 [9] colorspace_1.2-4     MASS_7.3-26

loaded via a namespace (and not attached):
[1] compiler_3.0.0 tools_3.0.0
```