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 (first) analysis of simulated data sets to test the behavior of our proposed methods for analyzing continuous pathway data.

1.1.1 Aims/Objectives

The primary objective is to create a large number of simulated datasets so we can see how the methods should work in an ideal theoretical setting. We also want to explore how well the outlier-detection method works to separate good "signal" proteins from bad "noise" proteins. Future analyses will look at other aspects of the method.

1.2 Methods

1.2.1 Description of the Data

We simulate 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 (i.e., 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

After standardizing (mean zero, standard deviation one) the measurements from each protein, we perform principal components analysis (PCA). We use both the Auer-Gervini [2008] Bayesian approach and the "broken stick" model to estimate the number K of significant components. For each protein, we then compute its distance Δ from the origin in the K-dimensional space of PCA loadings, where K is chosen using the Auer-Gervini approach. We use histograms and receiver operating characteristic (ROC) curves to explore how well Δ separates signal from noise.

1.3 Results

- The broken stick model appears to be somewhat conservative. It never finds more than the true number of components, but in 16% of the cases with two components, it only finds one of them.
- The Auer-Gervini approach is more accurate (94% compared to 84%) but less conservative (i..e, it finds extra components about 0.36% of the time) than the broken-stick model.
- Failure of the Auer-Gervini method to find the true number of components is associated with a small number of proteins (**Figure 3**) or a small number of samples (**Figure 1**), but does not depend on the correlation (**Figure 2**).
- The distance Δ from the origin is highly effective at separating good "signal" proteins from bad "noise" proteins (**Figure 4**, **Figure 5**).
- Using a cutoff $\Delta > 0.3$ is expected to eliminate 99.6% of noise proteins while retaining 99.5$ of signal proteins (**Section 3.5.1**).

1.4 Conclusions

Using the distance from the origin in a principal components loadings space with dimension determined by the Auer-Gervini Bayesian approach appears to be an effective way to separate signal from noise.

2 Preliminaries

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
```

We also define two default color sets.

```
> col6 <- c(brewer.pal(5, "Set1"), "#4ea3a3")
> col4 <- c("gray", "red", "blue", "purple")
```

2.2 Simulated Data Sets

With the packages loaded, we can start simulating datasets. We simulate five different kinds of datasets. The simulated datasets can have either one or two true underlying signals, and each signal can either be all postiively correlated or can include roughly half positive and half negative correlation. We use the following procedure:

1. We select an even number of features (proteins, antibodies, genes, etc.) between 10 and 20.
2. We split the set of features roughly in half so that we can later specify two groups.
3. We also split the features in half to allow for positive and negative correlation.
4. We randomly choose a correlation coefficient from a normal distribution with mean 0.5 and standard deviaiton 0.1.
5. We construct five different correlation/covariance matrices for the five kinds of simulated datasets.
6. We then select a number of samples for each dataset from a normal distribution with mean 300 and standard deviation 60.
7. Finally, we add two “noise” genes to each data set to represent uncorrelated outliers.

We repeat this procedure 500 times, which produces a total of 2500 simulated datasets.

```
> nSimSets <- 500
> f <- "savedSims.rda"
> if (file.exists(f)) {
+   load (f)
+ } else {
+   set.seed(159708)
+   savedSims <- list()
+   bsdim <- list()
+   counter <- 0
+   for (idx in 1:nSimSets) {
+     cat(idx, "\n", file=stderr())
+ # parameters
+     nProtein <- 2*sample(5:10,1)
+     splinter <- sample((nProtein/2) + (-3:3), 1)
+     positive <- sample(nProtein, nProtein/2)
+     negative <- (1:nProtein)[!((1:nProtein) %in% positive)]
+     posi <- positive[positive <= splinter]
+     nega <- negative[negative <= splinter]
+ # one group, unsigned
+     rho <- rnorm(1, 0.5, 0.1)
+     sigma1 <- matrix(rho, ncol=nProtein, nrow=nProtein)
+     diag(sigma1) <- 1
+ # two groups, unsigned
+     sigma2 <- sigma1
+     sigma2[(1+splinter):nProtein, 1:splinter] <- 0
+     sigma2[1:splinter, (1+splinter):nProtein] <- 0
+ # one group, signed
+     sigma3 <- sigma1
+     sigma3[positive, negative] <- -sigma3[positive, negative]
+     sigma3[negative, positive] <- -sigma3[negative, positive]
+ # two groups, signed
+     sigma4 <- sigma2
+     sigma4[positive, negative] <- -sigma4[positive, negative]
+     sigma4[negative, positive] <- -sigma4[negative, positive]
+ # two groups, mixed
+     sigma5 <- sigma2
+     sigma5[posi, nega] <- -sigma5[posi, nega]
+     sigma5[nega, posi] <- -sigma5[nega, posi]
+ # number of samples
+     nSample <- round(rnorm(1, 300, 60))
+     for (nm in paste("sigma", 1:5, sep='')) {
+ # add noise
+       ss <- matrix(0, nProtein+2, nProtein+2)
+       diag(ss) <- 1
+       ss[1:nProtein, 1:nProtein] <- get(nm)
+ # basic analysis
+       counter <- counter+1
+       value <- SimThresher(ss, nSample,
+                            paste(nm, counter, sep="."),
+                            method='auer.gervini')
+       savedSims[[counter]] <- value
+       bsdim[[counter]] <- bsDimension(value@spca)
+     }
+   }
+   save(savedSims, bsdim, file=f)
+ }
> rm(f)
```

3 Interpreting the Results

We need to extract various useful pieces of data from the simulation results. We start by defining the "type" of simulated dataset.

```
> 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
```

3.1 The broken stick model sometimes finds too few components

We are trying two different automated methods to estimate the number of "statistically significant" components in each principal component space: the Auer-Gervini Bayesian approach and the broken-stick model. Here we look at how well the estimates from the broken-stick model match the true number of components that we put into the simulation.

```
> bsdim <- unlist(bsdim)
> table(simType, bsdim)

                   bsdim
simType               0   1   2
  OneGroupUnsigned    1 499   0
  TwoGroupsUnsigned  22  62 416
  OneGroupSigned      1 499   0
  TwoGroupsSigned    21  65 414
  TwoGroupsMixed     24  64 412
```

We see that the estimate appears to be somewhat conservative. It never finds more than the true number of components, but in at least sixteen percent (80/500) of the cases with two true components, it only finds one of them.

3.2 The Auer-Gervini approach is more accurate but less conservative

Now we look at how well the Auer-Gervini estimates match the true number of components.

```
> estNComponents <- unlist(lapply(savedSims, function(v) v@pcdim))
> table(simType, estNComponents)
```

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

In rare cases (9/2500, about one-third of one percent), the Auer-Gervini method overestimates the number of components, possibly because of a weak signal in the presence of the two noise features. However, it only gets a wrong answer in the presence of two true components about 6 percent of the time (30/500).

The following table shows that the broken-stick model is almost always more conservative than the Auer-Gervini approach.

```
> table(bsdim, estNComponents)
```

```
     estNComponents
bsdim    1    2    4
    0    5   64    0
    1 1075  114    0
    2    1 1232    9
```

3.3 Understanding conservatism

There are several possible explanations for the failure of the Auer-Gervini method to to find the second true component.

1. It may be harder when the number of samples is small.
2. It may be harder when the true correlation is small.
3. It may be harder when the number of true signal proteins is small.

We can explore each of these potential explanation graphically, starting with the number of samples (**Figure 1**). This figure suggests that the number of samples may be ver weakly related to whether or not the algorithm detects the correct number of components.

```
> ns <- unlist(lapply(savedSims, function(val) val@nSample))
> summary(ns)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  126.0   263.8   304.0   303.3   343.2   506.0 
```

Next, we look at the true correlation coefficients from each simulation (**Figure 2**. This figure supports the idea that when the algorithm should find two components but only finds one, then the true correlation is likely to be irrelevant.

```
> correl <- unlist(lapply(savedSims, function(v) v@rho))
```

Finally, we look at the effect of the number of true signal proteins (**Figure 3**). This figure shows that the number of proteins is likely to be an important predictor of the failure to identify both components.

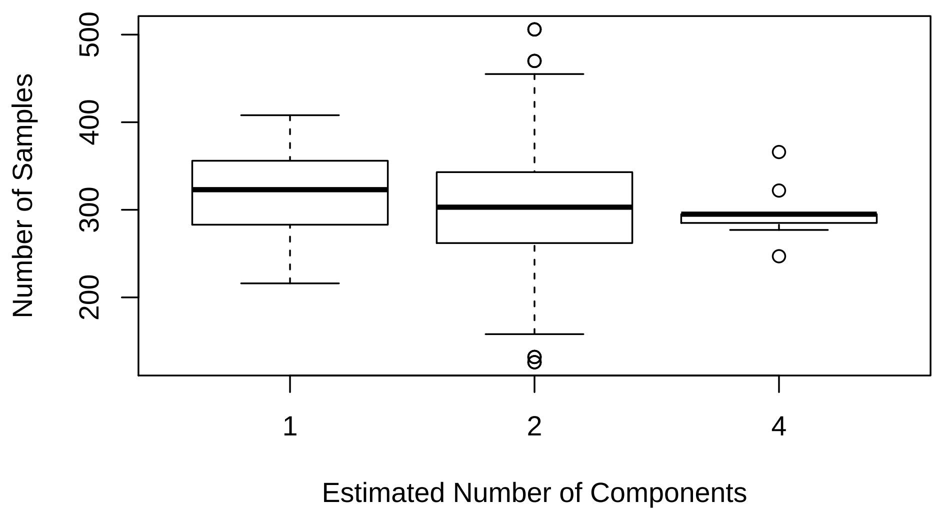
```
> np <- unlist(lapply(savedSims, function(val) ncol(val@data)))
> temp <- rep(1:5,  nSimSets)
> classy <- rep(temp, times=np)
```

We can use logistic regression to determine whether the number of proteins and the correlation give independent predictors of the inability to find both true compnents. First, we try an additive model:

```
> slayer <- data.frame(NC=1*(estNComponents > 1), ns, np, correl)[evens,]
> model <- glm(NC ~ ns + np + correl, data=slayer, family=binomial)
> anova(model, test="Chisq")
```

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: NC

Terms added sequentially (first to last)


       Df Deviance Resid. Df Resid. Dev Pr(>Chi)    
NULL                    1499     630.39             
ns      1    4.970      1498     625.42  0.02579 *  
np      1  250.234      1497     375.18  < 2e-16 ***
correl  1    0.411      1496     374.77  0.52162    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As we expected from the graphical views, the number of proteins is highly significant, the number of samples is significant, and the correlation coefficient is irrelevant. So, we can ignore the correlation in what folows.



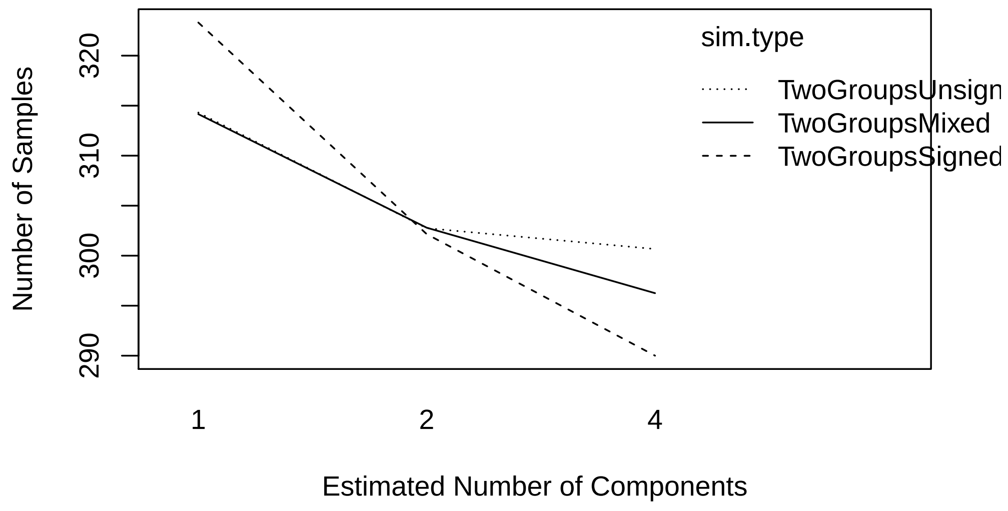


Figure 1: Number of components versus the number of samples; **(top)** box-and-whisker plot, **(bottom)** interaction plot.

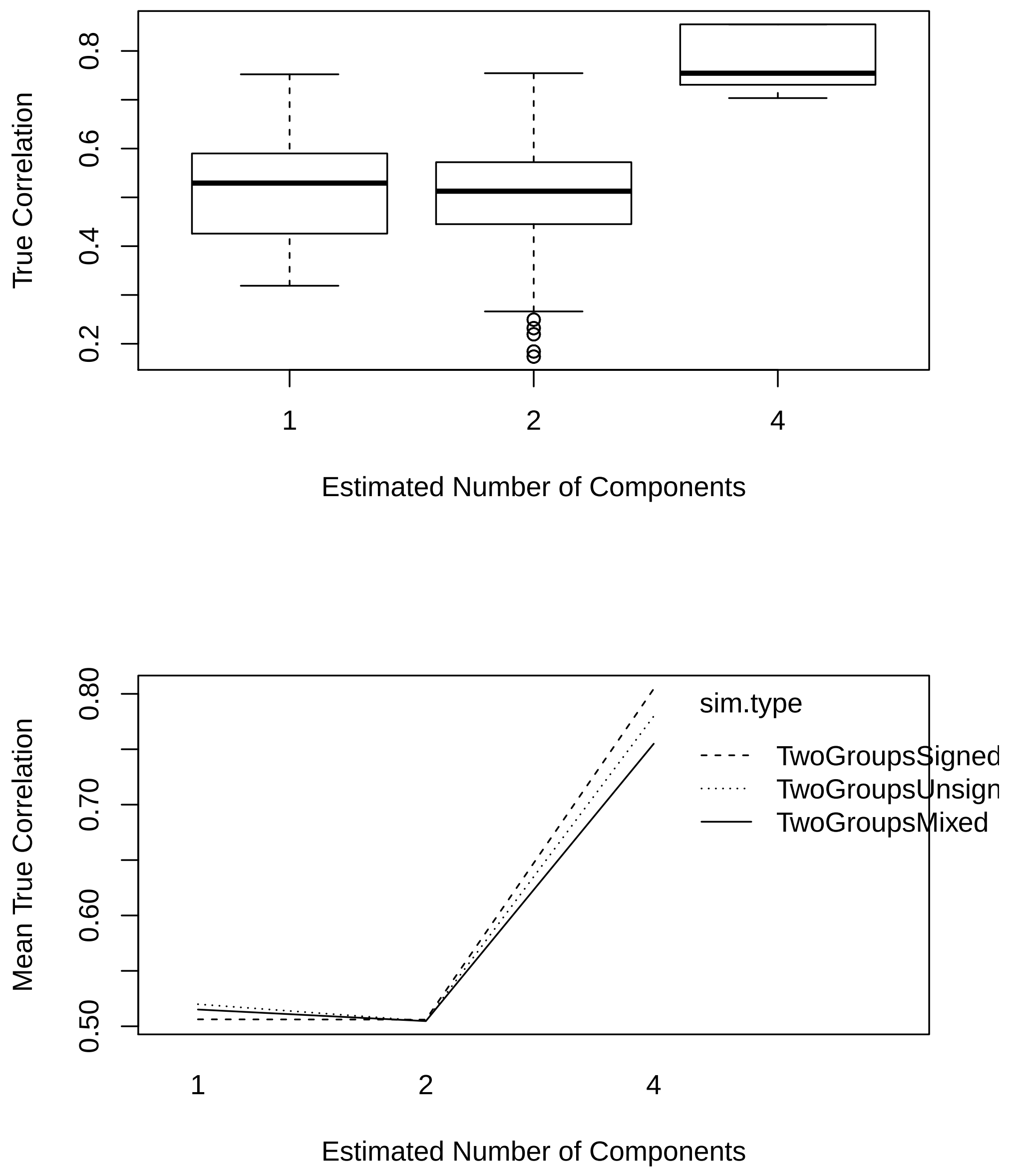
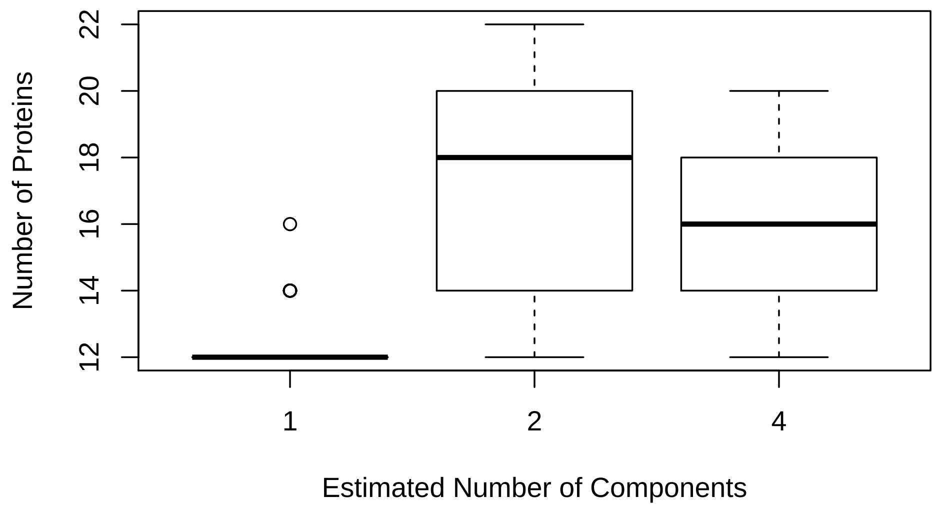


Figure 2: Number of components versus the true correlation between proteins; **(top)** box-and-whisker plot, **(bottom)** interaction plot.



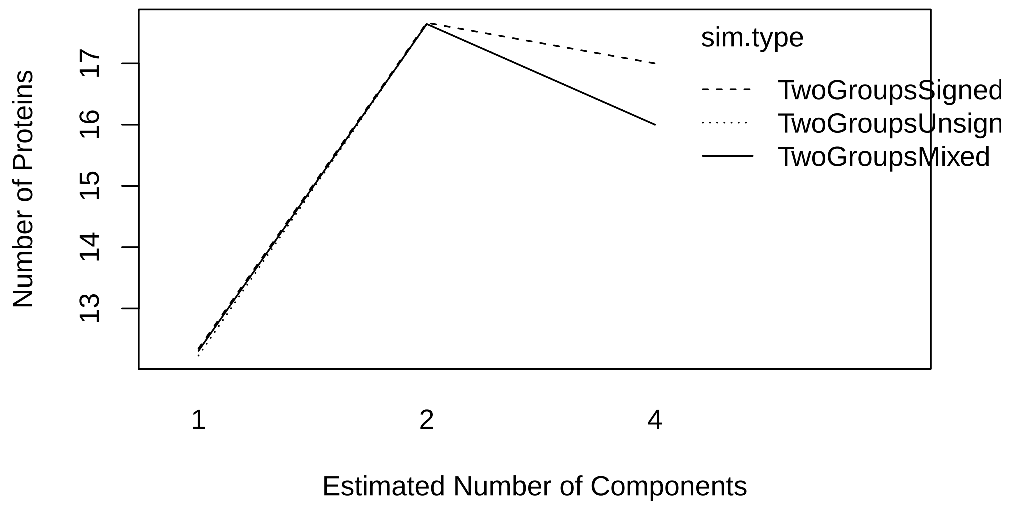


Figure 3: Number of components versus the number of true "signal" proteins; **(top)** box-and-whisker plot, **(bottom)** interaction plot.

```
> slayer <- data.frame(NC=1*(estNComponents > 1), ns, np, correl)[evens,]
> addmodel <- glm(NC ~ ns + np, data=slayer, family=binomial)
> anova(addmodel, test="Chisq")

Analysis of Deviance Table

Model: binomial, link: logit

Response: NC

Terms added sequentially (first to last)


     Df Deviance Resid. Df Resid. Dev Pr(>Chi)
NULL                  1499     630.39
ns    1     4.97      1498     625.42  0.02579 *
np    1   250.23      1497     375.18  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the aditive model that ignores correlation, both terms are statistically significant, and independent. We now check whether there is an interaction between the predictive factors.

```
> intmodel <- glm(NC ~ np*ns, data=slayer, family=binomial)
> anova(intmodel, test="Chisq")

Analysis of Deviance Table

Model: binomial, link: logit

Response: NC

Terms added sequentially (first to last)


      Df Deviance Resid. Df Resid. Dev Pr(>Chi)
NULL                   1499     630.39
np     1  252.302      1498     378.08  < 2e-16 ***
ns     1    2.902      1497     375.18  0.08846 .
np:ns  1    0.284      1496     374.90  0.59440
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The interaction term is not significant. So, we record the coefficients in the additive model:

```
> coef(addmodel)

  (Intercept)            ns            np
-11.607568740  -0.003564007   1.128698445
```

3.4 Separating the Wheat from the Chaff

In the code loop above where we simulated the datasets, we computed the distance Δ from the origin (in a Bayesian-defined K-dimensional space) for each protein in the dataset. These separate into "good" proteins (which are part of the signals we are trying to detect) and "bad" proteins (which are the noise we are trying to remove. The next block of code extracts the distances Δ for these two sets of proteins.

```
> goody <-   unlist(lapply(savedSims, function(val) {
+   np <- ncol(val@data)-2
+   val@delta[1:np]
+ }))

> baddy <-   unlist(lapply(savedSims, function(val) {
+   np <- ncol(val@data)-2
+   val@delta[np+(1:2)]
+ }))

> breakpoints <- seq(0, 1, length=201)
```

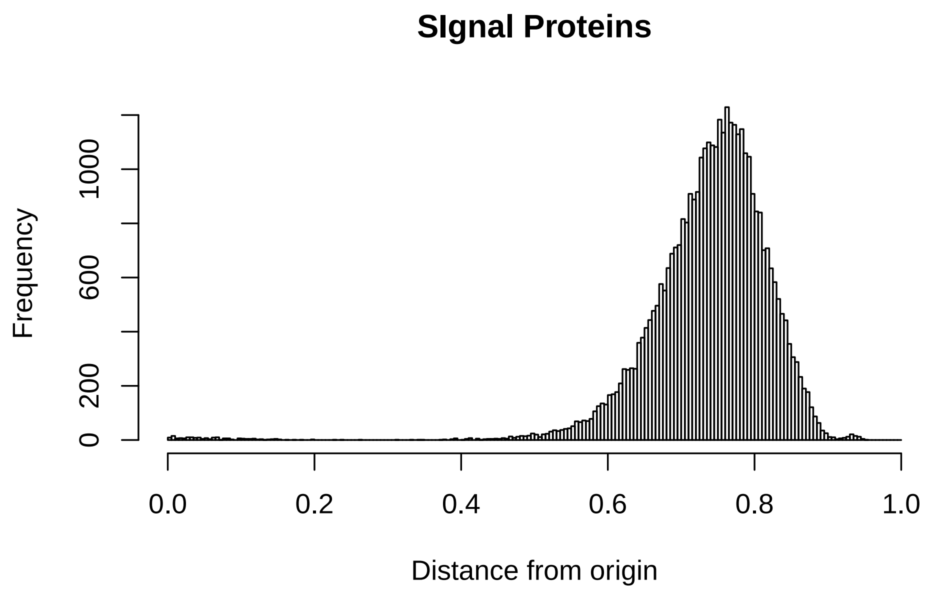
Plotting histograms of the distances Δ of the protein features from the origin suggests that this procedure does a pretty good job of separating the signal from the noise (**Figure 4**). We also note, however, the small "bump" of noise proteins with very large coefficients (near 1.0), suggesting that no amount of parameter-tuning or cutoff-choosing is going to exclude them.

We also prepared a plot of the receive operating characteristics (ROC) curve for using Δ to ditinguish signal from noise proteins (**Figure 5**). As suggested above, the separation given by the ROC curve is excellent.

3.5 Understanding Anti-conservatism

Here we start by looking at the "bad = noise" features that have a large distance from the origin.

```
> w <- unique(round(which(baddy > 0.5)/2))
> data.frame(EstNC=estNComponents, nSamples=ns, nProteins=np, Correl=correl)[w,]

     EstNC nSamples nProteins    Correl
212      4      295        18 0.8545704
214      4      295        18 0.8545704
215      4      295        18 0.8545704
1122     4      285        16 0.7544456
```



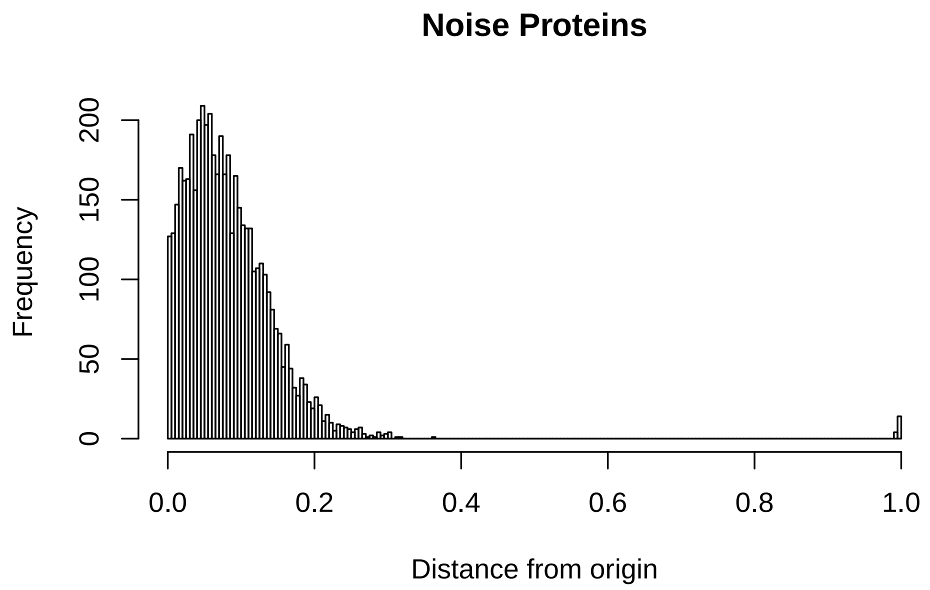


Figure 4: Histograms of the distance from the origin of the "true" signal proteins and the additional noise proteins.

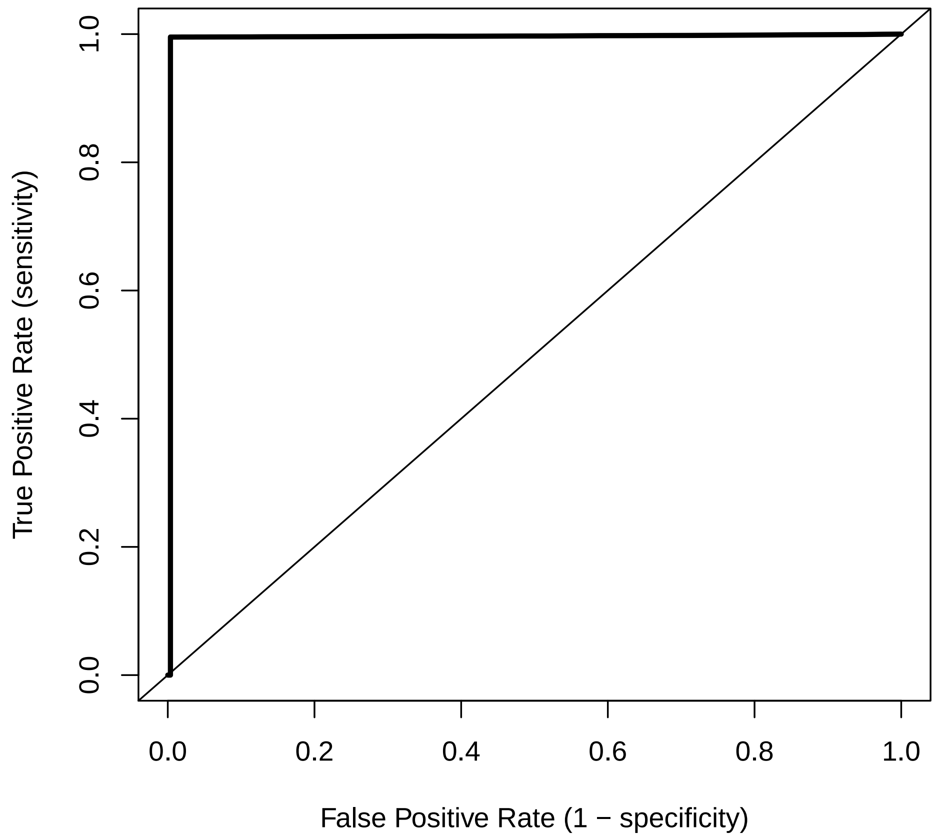


Figure 5: Receiver operating characteristic (ROC) curve, based on simulated datasets, for the proposed method of separating (true) signal from (false) noise among the protein features.

```
1124     4       285           16 0.7544456
1214     2       366           14 0.7034335
1215     4       366           14 0.7034335
1770     4       247           20 0.7307377
1892     4       322           14 0.7304254
2354     2       277           12 0.7311215
2355     4       277           12 0.7311215
```

We see that there appears to be nothing special about the number of samples or the number of proteins. However, this set includes all nine cases where the number of components is estimated to equal four (i.e., more than truth). More importantly, all of the correlation coefficients are very high (i.e., in the top 3% for this simulation):

```
> summary(correl)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.1737  0.4442  0.5138  0.5072  0.5727  0.8546 
```

```
> mean(correl > 0.7)
```

```
[1] 0.028
```

This finding suggest that Auer-Gervini is only likely to include the two noise features as independent significant components if the features in the true signals are very highly correlated.

3.5.1 Selecting a Cutoff

The next task is to try to find a reasonable place to draw a cutoff, based on Δ, between the good signal proteins and the bad noise proteins. Of course, this depends on whether we are more concerned about omitting true signals or about including noise. To begin, we compute the quantiles of Δ for the signal proteins.

```
> quantile(goody, seq(0.01, 0.05, by=0.01))
```

```
       1%        2%        3%        4%        5% 
0.5121072 0.5604375 0.5848545 0.5997724 0.6111055 
```

This computation suggests that a reasonable cutoff is certainly less than 0.6.

```
> testRange <- seq(0.2, 0.6, by=0.05)
```

The next function uses the cutoff on Δ to compute the true positive and false positive rates at the corresponding point on the ROC curve.

```
> cutter <- function(r) {
+   w <- 1+sum(breakpoints <= r)
+   c(R=r, FP=fp[w], TP=tp[w])
+ }
```

We apply this function at the points in the test range.

```
> cutterTable <- t(matrix(unlist(lapply(testRange, cutter)), nrow=3))
> colnames(cutterTable) <- c("Delta", "FP", "TP")
> cutterTable

      Delta     FP        TP
 [1,]  0.20 0.0300 0.9954664
 [2,]  0.25 0.0108 0.9954143
 [3,]  0.30 0.0042 0.9953882
 [4,]  0.35 0.0038 0.9952840
 [5,]  0.40 0.0036 0.9949453
 [6,]  0.45 0.0036 0.9939812
 [7,]  0.50 0.0036 0.9904898
 [8,]  0.55 0.0036 0.9819698
 [9,]  0.60 0.0036 0.9554455
```

Well, this helps narrow things down. We probably don't want to set a cutoff below $\Delta = 0.25$, since this starts admitting too many (i.e., at least 1%) noise proteins. And we probably don't want to go above $\Delta = 0.35$, since we keep losing signal proteins without corresponding gains in reducing the noise proteins.

```
> refine <- t(matrix(unlist(lapply(seq(0.25, 0.35, by=0.01), cutter)), nrow=3))
> colnames(refine) <- c("Delta", "FP", "TP")
> refine

      Delta     FP        TP
 [1,]  0.25 0.0108 0.9954143
 [2,]  0.26 0.0082 0.9953882
 [3,]  0.27 0.0074 0.9953882
 [4,]  0.28 0.0068 0.9953882
 [5,]  0.29 0.0056 0.9953882
 [6,]  0.30 0.0042 0.9953882
 [7,]  0.31 0.0040 0.9953622
 [8,]  0.32 0.0038 0.9953622
 [9,]  0.33 0.0038 0.9953361
[10,]  0.34 0.0038 0.9953361
[11,]  0.35 0.0038 0.9952840
```

It seems to me that a cutoff anywhere between 0.30 and 0.40 is reasonable, which gives a false negative rate of about 5 in 1000 and a false positive rate about 4 in 1000. Because the peformance is so consistent across this range, we select the smaller value (0.3) as our default cutoff, since this will eventually retain as many true positives as possible.

4 More Detailed Methods

In this appendix, we document the main functions in the Thresher package.

4.1 Thresher

The first major step in the analysis is to identify and remove outliers from the set of antibodies (features; predictors; or pathway members). Outliers are features that appear to be simply noise, without being correlated to anything else in the putative pathway. The function that carries out this procedure is called `Thresher`, by analogy with the well-known idiom about separating the wheat from the chaff.

```
> Thresher

function (data, nm = deparse(substitute(data)), FUZZ = 0.005,
    metric = "pearson", method = c("broken.stick", "auer.gervini"))
{
    std <- scale(data)
    spca <- SamplePCA(t(std))
    ag <- AuerGervini(spca)
    method <- match.arg(method)
    pcdim <- switch(method, broken.stick = bsDimension(spca),
        auer.gervini = agDimension(ag))
    deltaDim <- max(1, pcdim)
    lambda <- sqrt(spca@variances)
    loadings <- sweep(spca@components, 2, lambda, "*")
    delta <- sqrt(apply(loadings[, 1:deltaDim, drop = FALSE]^2,
        1, sum))
    gc <- hclust(distanceMatrix(std, metric), "ward")
    new("Thresher", name = nm, data = data, spca = spca, loadings = loadings,
        gc = gc, pcdim = pcdim, delta = delta, ag = ag)
}
<environment: namespace:Thresher>
```

The algorithm in `thresher` basically works as follows.

1. We start by standardizing the data from each antibody (so it has mean zero and standard deviation one).
2. We perform a PCA on the standardized pathway dataset.
3. We apply the Auer-Gervini approach to select the number K of statistically significant components. (See below.)

4. Next, for each antibody, we compute its (Euclidean) distance from the origin in the K-dimensional space of loadings; this is a reflection of the amount of "influence" that the antibody has in clustering the samples.

5. Antibodies whose influence is smaller than a specfied cutoff will be identified as outliers and removed from the dataset.

4.2 Auer-Gervini

The original method described in the Auer-Gervini 2008 paper is a graphical tool for selecting the number K of components. We have developed an additional rule that allows us to automate this procedure. We can illustrate how this works using one of our simulated data sets.

```
> s <- savedSims[[5]]
```

Auer and Gervini put an exponential prior distribution on the number of principal components. This prior distribution imposes the condition that the probability associated with a dimension K is a non-increasing function of K; in other words, more components are less likely than fewer components. The prior distribution has a tunable hyperparameter, θ, that governs how swiftly the prior probability decreases. When $\theta = 0$, every possible number of components is equally likely, and in the posterior distribtion, all components are declared significant. As θ increases, the number of components selected by the posterior distribution decreases in the form of a step function (**Figure 6**). (Interestingly, some values of K can never be chosen by the Bayesian procedure.) For large anough values $\theta > \theta_0$, the posterior distribution will always choose zero components.

Auer and Gervini suggest selecting the largest value of K for which the step-length is "significant", since this value of K would be chosen by the largest number of "reaonable" prior values of θ. One problem with their approach is that "significant" is still subjective. A second problem is that the step length corresponding to chossing $K = 0$ is actually infinite (since every value greater than θ_0 is on this step). Our innovations are as follows:

1. We compute an upper bound on "reaonable" values of θ, which corresponds to the value that assigns at most 99% of the prior probability to $K = 0$, and keeps at least 1% of the prior probability on $K \geq 1$.

2. We define "significant length" to be at least twice the mean length.

```
> Thresher:::estimateTop

function (object)
{
    max(-2 * log(0.01)/length(object@Lambda), 1.03 * object@changePoints)
}
<environment: namespace:Thresher>

> agDimension
```

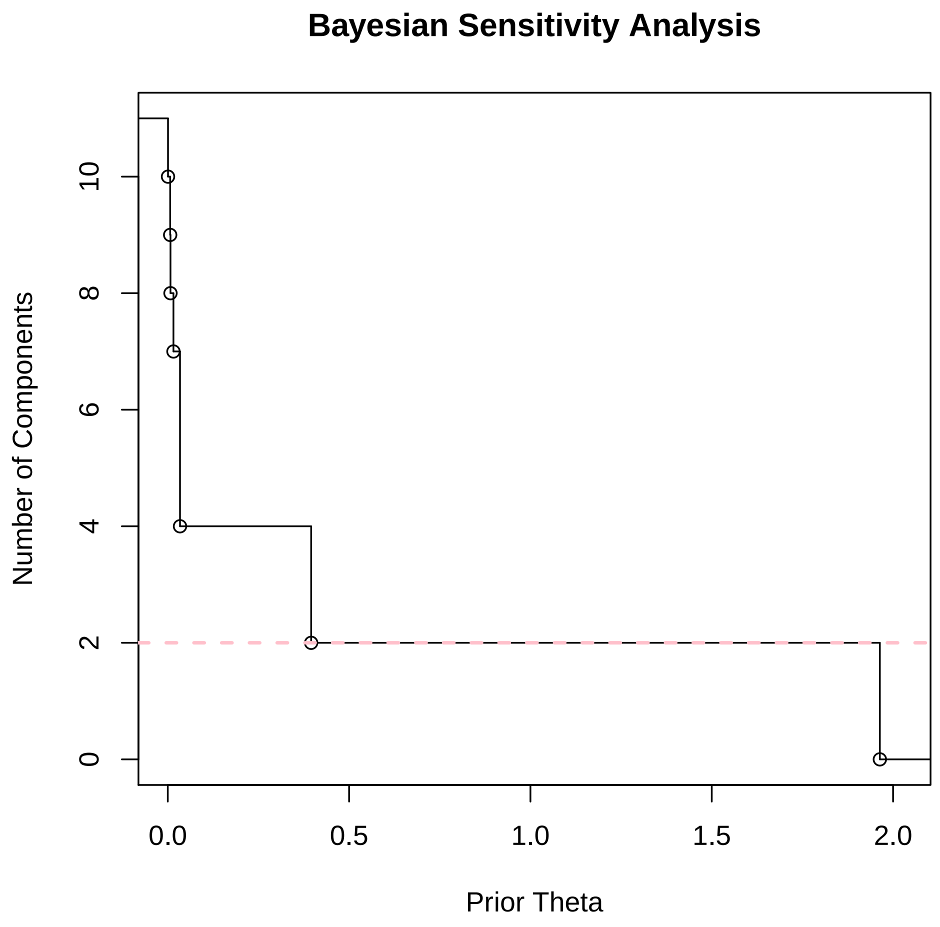


Figure 6: Auer-Gervini step function relating the prior hyperparameter θ to the maximum posterior estimate of the nunber K of significant principal components. In this example, our automated procedure selects $K = 2$.

```
function (object)
{
    stepLength <- diff(c(object@changePoints, estimateTop(object)))
    if (length(stepLength) > 3) {
        magic <- (stepLength > 2 * mean(stepLength))
    }
    else {
        magic <- (stepLength == max(stepLength))
    }
    object@dLevels[1 + which(magic)[1]]
}
<environment: namespace:Thresher>
```

5 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_0.9.0       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
```