AC-PCA: Principal Component Analysis Adjusting for Confounding Variation User's Guide

Zhixiang Lin, Hongyu Zhao & Wing Hung Wong

March 1, 2016

If you use AC-PCA in published research, please cite:
Z. Lin, C. Yang, Y. Zhu, J. C. Duchi, Y. Fu, Y. Wang, B. Jiang, M. Zamanighomi, X. Xu, M. Li, N. Sestan, H. Zhao, W. H. Wong:
AC-PCA adjusts for confounding variation in transcriptome data and recovers the anatomical structure of neocortex.
bioRxiv, `http://dx.doi.org/10.1101/040485`

Contents

1 Introduction

1.1 Scope

Dimension reduction methods are commonly applied to visualize datapoints in a lower dimensional space and identify dominant patterns in the data. Confounding variation, technically and biologically originated, may affect the performance of these methods, and hence the visualization and interpretation of the results. AC-PCA simultaneously performs dimension reduction and adjustment for confounding variation. The intuition of AC-PCA is that it captures the variations that are invariant to the confounding factors. One major application area for AC-PCA is the transcriptome data, and it can be implemented to other data types as well.

1.2 How to get help

This user's guide addresses many scenarios for confounding factors. Additional questions about AC-PCA can be sent to `linzx06@gmail.com`. Comments on possible improvements are very welcome.

2 Specific experimental designs and case studies

2.1 Input for AC-PCA

AC-PCA requires two inputs: the data matrix X and the confounder matrix Y. X is of dimension $n \times p$, where n is the number of samples and p is the number of variables (genes). Y is of dimension $n \times q$, where n is the number of samples and q is the number of confounding factors. Depending on the scientific question, confounding factors can be different for the same experiment. As an example, consider a transcriptome experiment where gene expression levels were measured in multiple tissues from multiple species. If one wants to capture the variation across tissues that is preserved among species, then the species labels are the confounding factors. On the other hand, if the variations across species but shared among tissues are desirable, the tissue labels are the confounding factors. Missing data is allowed in both X and Y. In the following sections, we provide implementation details on how to design Y for different types of confounders.

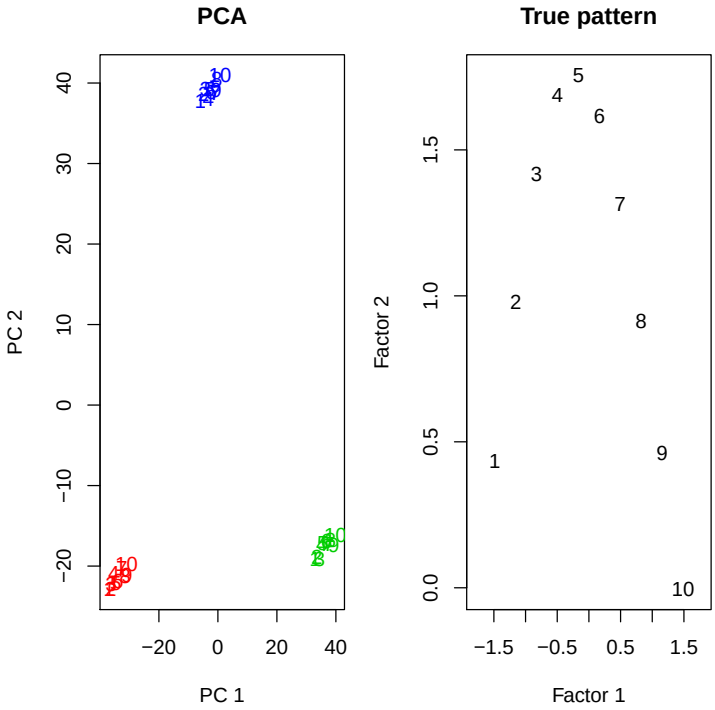
2.2 Categorical confounding factors

Categorical confounding factors are commonly observed in biological data, it can be technical (different experimental batches, etc.) and biological (donor labels, different races, species, etc.). To adjust for categorical confounding factors, the factor labels need to be converted to dummy variables.
Simulated example 1:

```
> #library(acpca)
> load("~/Dropbox/AIPCA/R_package/data/data_example1.rda")
> X <- data_example1$X ###the data matrix
> dim(X)

[1]  30 400

> Y <- data_example1$Y ###the confounder matrix
> Y

      [,1] [,2] [,3]
 [1,]    1    0    0
 [2,]    1    0    0
 [3,]    1    0    0
 [4,]    1    0    0
 [5,]    1    0    0
 [6,]    1    0    0
 [7,]    1    0    0
 [8,]    1    0    0
 [9,]    1    0    0
[10,]    1    0    0
[11,]    0    1    0
[12,]    0    1    0
[13,]    0    1    0
[14,]    0    1    0
[15,]    0    1    0
[16,]    0    1    0
[17,]    0    1    0
[18,]    0    1    0
[19,]    0    1    0
[20,]    0    1    0
[21,]    0    0    1
[22,]    0    0    1
[23,]    0    0    1
[24,]    0    0    1
[25,]    0    0    1
[26,]    0    0    1
[27,]    0    0    1
[28,]    0    0    1
[29,]    0    0    1
[30,]    0    0    1
```

In this simulated example, the confounder is a factor with three levels, corresponding to three groups. The confounder contributes globally to all genes. This corresponds to simulation setting 1 in the manuscript. Next, we perform regular PCA and compare the result with the true simulated pattern:

```
> pca <- prcomp(X, center=T) ###regular PCA
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(pca$x[,1], pca$x[,2], xlab="PC 1", ylab="PC 2", col="black", type="n", main="PCA")
> text(pca$x[,1], pca$x[,2], labels = data_example1$lab, col=data_example1$colors+1)
> plot(data_example1$true_pattern[1,], data_example1$true_pattern[2,],
+      xlab="Factor 1", ylab="Factor 2", col="black", type="n", main="True pattern",
+      xlim=c(min(data_example1$true_pattern[1,])-0.3,
+             max(data_example1$true_pattern[1,])+0.3) )
> text(data_example1$true_pattern[1,], data_example1$true_pattern[2,], labels = 1:10)
```

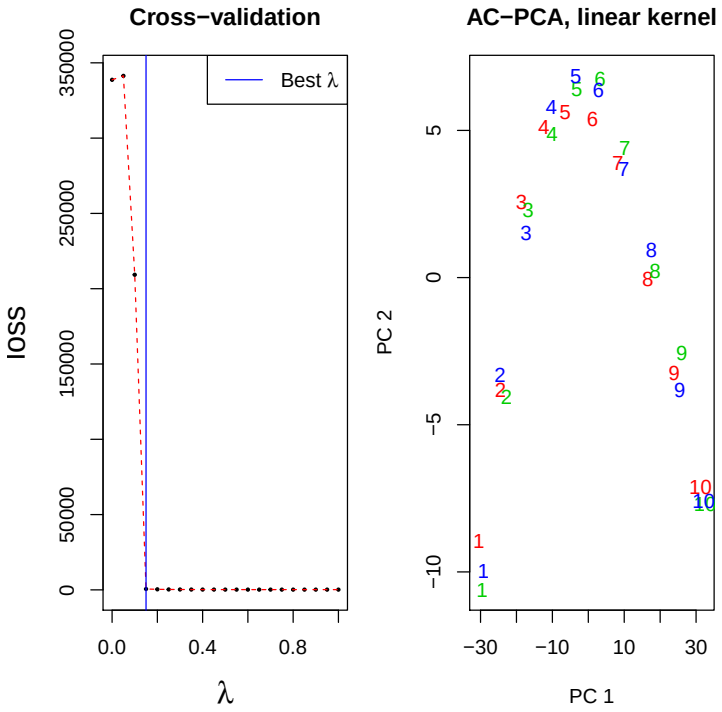


In the PCA plot, each color represents a group of samples with the same confounder label. The confounding variation dominates variation of the true pattern. Next we apply AC-PCA with linear kernel:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> ###perform cross-validation to select lambda, generates a plot of lambda vs. loss
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 1, 0.05), kernel="linear", nPC=2, plot=T) ##

[1] "Running fold 1"
[1] "Running fold 2"
[1] "Running fold 3"
```

```
[1] "Running fold 4"
[1] "Running fold 5"
```

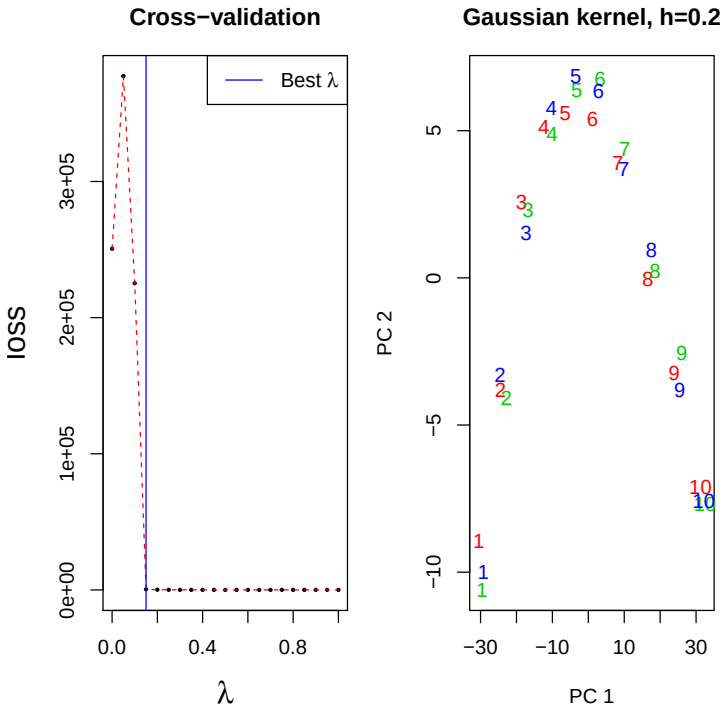
```
> ###run with the best lambda
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="linear", nPC=2)
> ###the signs of the PCs are meaningless
> plot(-result$Xv[,1], -result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="AC-PCA, linear kernel")
> text(-result$Xv[,1], -result$Xv[,2], labels = data_example1$lab,
+      col=data_example1$colors+1)
```



AC-PCA is able to recover the true latent structure. The signs of the PCs have been changed to match with the true pattern. We can also implement AC-PCA with Gaussian kernel (bandwidth $h = 0.2$):

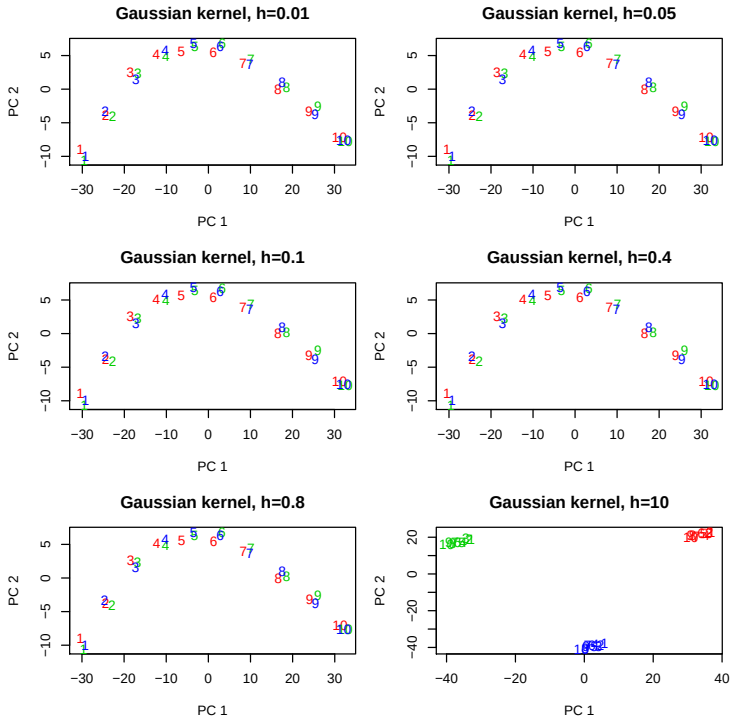
```
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 1, 0.05),
+                      kernel="gaussian", bandwidth=0.2, nPC=2, plot=T, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="gaussian",
+                 bandwidth=0.2, nPC=2)
> ###the signs of the PCs are meaningless
> plot(-result$Xv[,1], -result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="Gaussian kernel, h=0.2")
```

```
> text(-result$Xv[,1], -result$Xv[,2], labels = data_example1$lab,
+      col=data_example1$colors+1)
```



Gaussian kernel also works well. Note that the mean of $abs(Y)$: $mean(abs(Y)) = 0.33$. Next we try different bandwidths:

```
> par(mfrow=c(3,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> hs <- c(0.01, 0.05, 0.1, 0.4, 0.8, 10)
> for (h in hs){
+   result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 1, 0.05),
+                        kernel="gaussian", bandwidth=h, nPC=2, plot=F, quiet=T)
+ result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="gaussian",
+                  bandwidth=h, nPC=2)
+ plot(-result$Xv[,1], -result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main=paste("Gaussian kernel, h=", h, sep="") )
+ text(-result$Xv[,1], -result$Xv[,2], labels = data_example1$lab,
+      col=data_example1$colors+1)
+ }
```



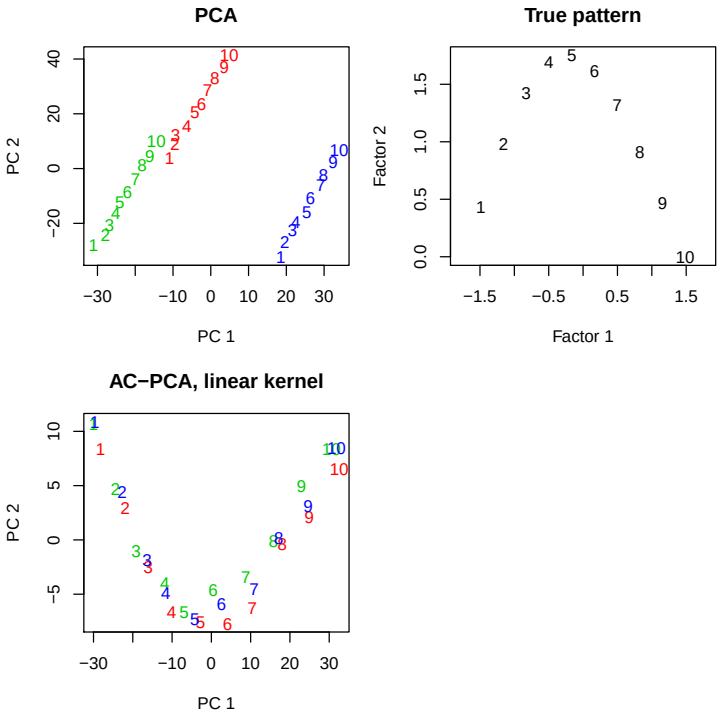
When bandwidth is too large, AC-PCA fails to adjust for the confounding variation. For detailed discussion on the selection of bandwidth selection, please refer to Chapter 3 in

Simulated example 2: in this simulated example, the confounder is a factor with three levels, corresponding to three groups. Instead of assuming that the confounder affects all genes (example 1), we assume that it affects only a subset of genes (half of the genes in this example).

We perform regular PCA and compare the result with the true simulated pattern:

```
> load("~/Dropbox/AIPCA/R_package/data/data_example2.rda")
> X <- data_example2$X ###the data matrix
> Y <- data_example2$Y ###the confounder matrix
> pca <- prcomp(X, center=T) ###regular PCA
> par(mfrow=c(2,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(pca$x[,1], pca$x[,2], xlab="PC 1", ylab="PC 2", col="black", type="n", main="PCA")
> text(pca$x[,1], pca$x[,2], labels = data_example2$lab, col=data_example2$colors+1)
> plot(data_example2$true_pattern[1,], data_example2$true_pattern[2,],
+      xlab="Factor 1", ylab="Factor 2", col="black", type="n", main="True pattern",
+      xlim=c(min(data_example2$true_pattern[1,])-0.3,
+             max(data_example2$true_pattern[1,])+0.3) )
> text(data_example2$true_pattern[1,], data_example2$true_pattern[2,], labels = 1:10)
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 1, 0.05), kernel="linear", nPC=2,
```

```
+                       plot=F, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="linear", nPC=2)
> plot(-result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="AC-PCA, linear kernel")
> text(-result$Xv[,1], result$Xv[,2], labels = data_example2$lab,
+      col=data_example2$colors+1)
```



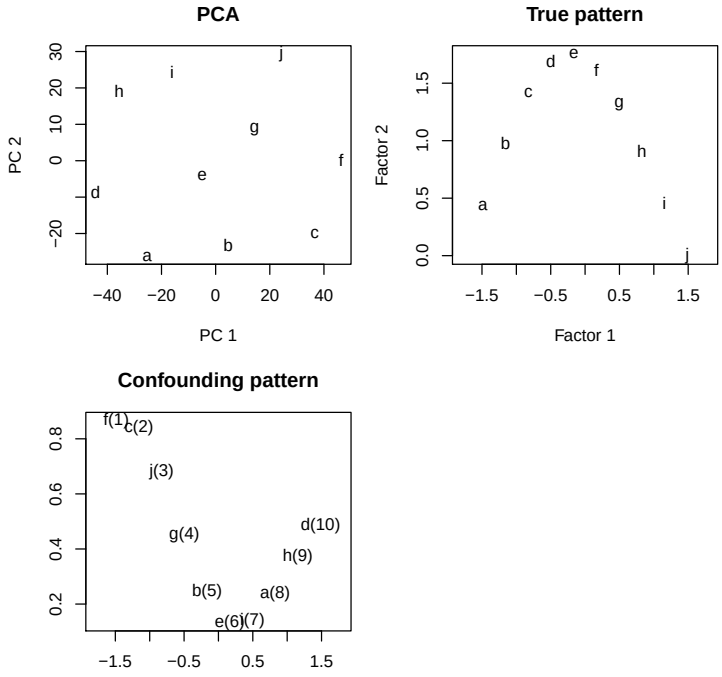
2.3 Continuous confounding factors

Continuous confounding factors may be present in biological data, for example, age.
Simulated example 3: the confounder is assumed to be continuous and it contributes a global trend to the gene expression levels.

We first perform regular PCA:

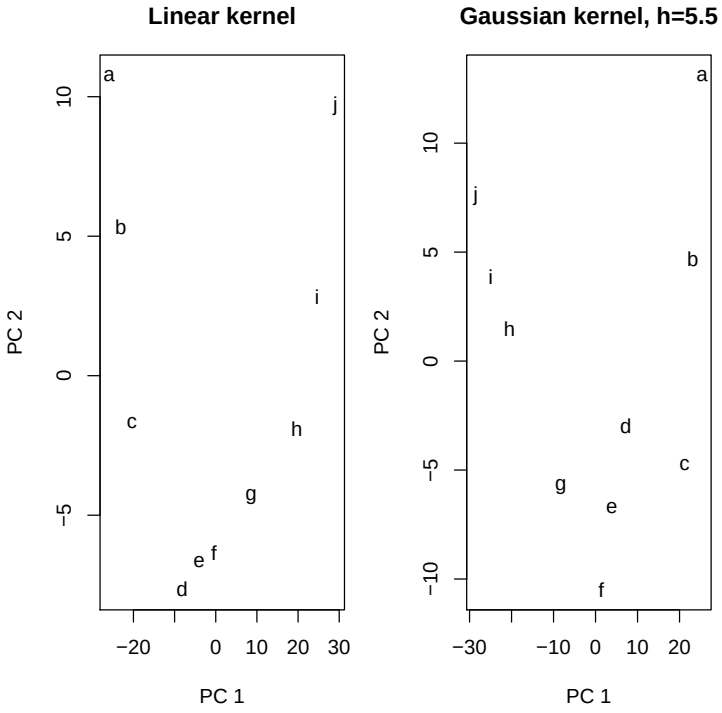
```
> load("~/Dropbox/AIPCA/R_package/data/data_example3.rda")
> X <- data_example3$X ###the data matrix
> Y <- data_example3$Y ###the confounder matrix
> Y

      [,1]
 [1,]    8
```

```
 [2,]    5
 [3,]    2
 [4,]   10
 [5,]    6
 [6,]    1
 [7,]    4
 [8,]    9
 [9,]    7
[10,]    3

> pca <- prcomp(X, center=T) ###regular PCA
> par(mfrow=c(2,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(pca$x[,1], pca$x[,2], xlab="PC 1", ylab="PC 2", col="black", type="n", main="PCA")
> text(pca$x[,1], pca$x[,2], labels = data_example3$lab)
> plot(data_example3$true_pattern[1,], data_example3$true_pattern[2,],
+      xlab="Factor 1", ylab="Factor 2", col="black", type="n", main="True pattern",
+      xlim=c(min(data_example3$true_pattern[1,])-0.3,
+             max(data_example3$true_pattern[1,])+0.3) )
> text(data_example3$true_pattern[1,], data_example3$true_pattern[2,],
+      labels = data_example3$lab)
> plot(data_example3$confound_pattern[1,], data_example3$confound_pattern[2,],
+      xlab="", ylab="", col="black", type="n", main="Confounding pattern",
+      xlim=c(min(data_example3$confound_pattern[1,])-0.3,
+             max(data_example3$confound_pattern[1,])+0.3) )
> text(data_example3$confound_pattern[1,], data_example3$confound_pattern[2,], labels =
+        paste(data_example3$lab, '(',data_example3$Y, ')', sep="") )
```



Next, we implement AC-PCA with linear and Gaussian kernels:

```
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 1, 0.05), kernel="linear", nPC=2,
+                      plot=F, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="linear", nPC=2)
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="Linear kernel")
> text(result$Xv[,1], result$Xv[,2], labels = data_example3$lab)
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 1, 0.05), kernel="gaussian",
+                      bandwidth=mean(abs(Y)), nPC=2, plot=F, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="gaussian",
+                 bandwidth=mean(abs(Y)), nPC=2)
> plot(result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main=paste("Gaussian kernel, h=", mean(abs(Y)), sep="") )
> text(result$Xv[,1], result$Xv[,2], labels = data_example3$lab)
```



The results of linear and Gaussian kernels are comparable.

2.4 Experiments with replicates/unobserved confounding factors

Sometimes the confounding factors are unknown/unobserved. Consider an experiment where the gene expression levels are measured under multiple biological conditions with several replicates. It may be desirable to capture the variation across biological conditions but shared among replicates.
Simulated example 4: there are 10 biological conditions, each with 3 replicates. The variation is shared among replicates for half of the genes and not shared for the other genes. Suppose we want to capture the variation shared among replicates, the confounder matrix can be chosen such that samples within the same biological condition are "pushed" together:

```
> load("~/Dropbox/AIPCA/R_package/data/data_example4.rda")
> X <- data_example4$X ###the data matrix
> dim(X)

[1]  30 400

> Y <- data_example4$Y ###the confounder matrix
> dim(Y)
```

```
[1] 30 30
```

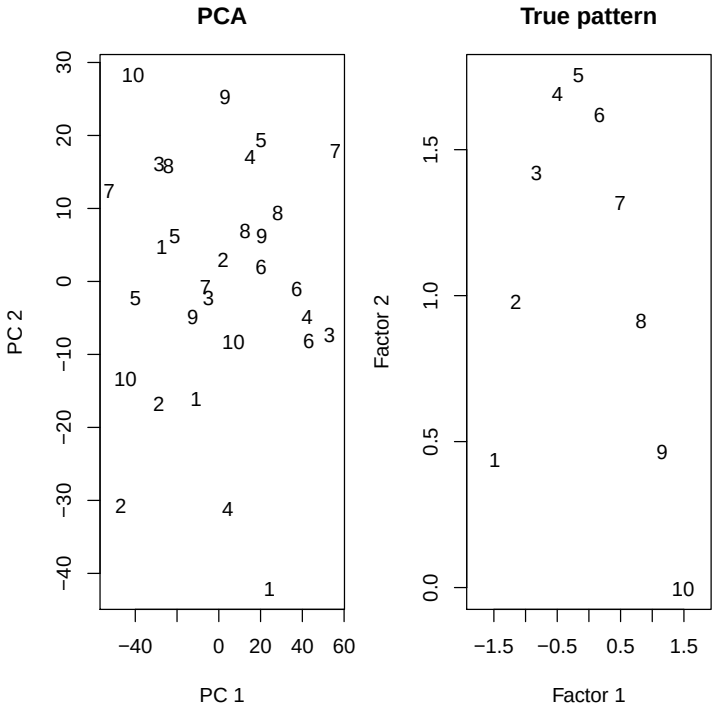
```
> Y[,1:2]
```

```
      [,1] [,2]
 [1,]    1    1
 [2,]   -1    0
 [3,]    0   -1
 [4,]    0    0
 [5,]    0    0
 [6,]    0    0
 [7,]    0    0
 [8,]    0    0
 [9,]    0    0
[10,]    0    0
[11,]    0    0
[12,]    0    0
[13,]    0    0
[14,]    0    0
[15,]    0    0
[16,]    0    0
[17,]    0    0
[18,]    0    0
[19,]    0    0
[20,]    0    0
[21,]    0    0
[22,]    0    0
[23,]    0    0
[24,]    0    0
[25,]    0    0
[26,]    0    0
[27,]    0    0
[28,]    0    0
[29,]    0    0
[30,]    0    0
```

In each column of Y, there is one 1 and one -1 corresponding to a pair of samples from the same biological condition. So the number of columns in Y is $10 \times 3 \times 2/2$. The implementation of AC-PCA with uneven number of replicates across biological condition is straightforward. We first implement PCA and compare it with the true pattern:

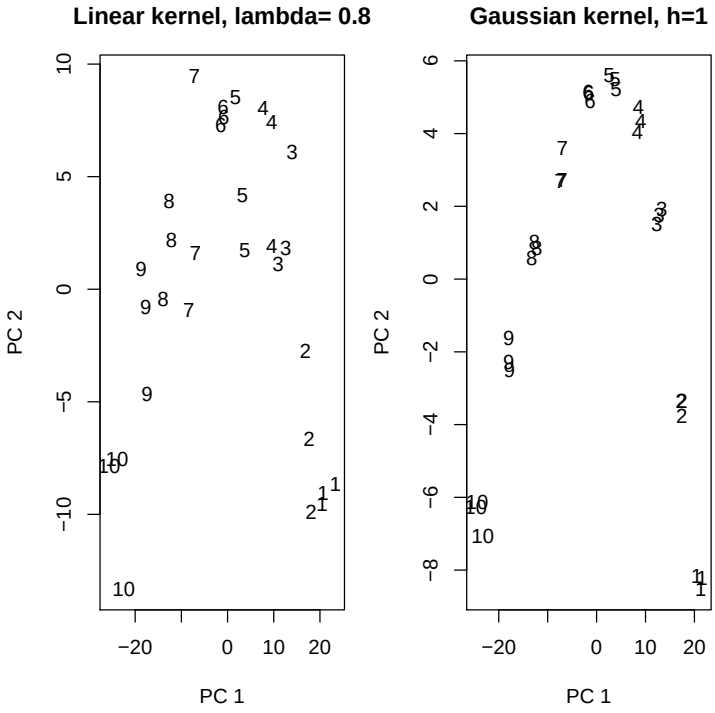
```
> pca <- prcomp(X, center=T) ###regular PCA
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(pca$x[,1], pca$x[,2], xlab="PC 1", ylab="PC 2", col="black", type="n", main="PCA")
> text(pca$x[,1], pca$x[,2], labels = data_example4$lab)
> plot(data_example4$true_pattern[1,], data_example4$true_pattern[2,],
```

```
+      xlab="Factor 1", ylab="Factor 2", col="black", type="n", main="True pattern",
+      xlim=c(min(data_example4$true_pattern[1,])-0.3,
+             max(data_example4$true_pattern[1,])+0.3) )
> text(data_example4$true_pattern[1,], data_example4$true_pattern[2,],
+      labels = 1:10)
```



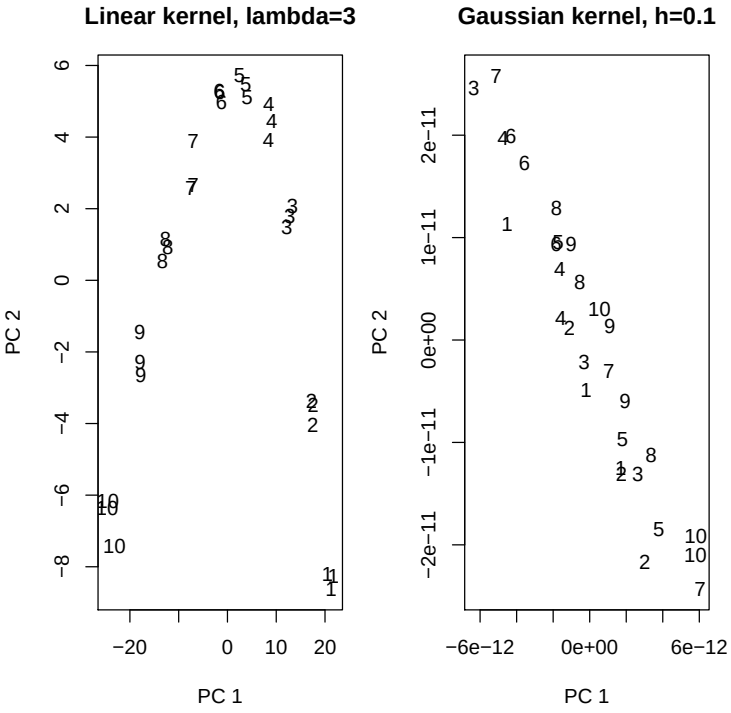
Next we implement AC-PCA with linear and Gaussian kernels:

```
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 2, 0.05), kernel="linear", nPC=2,
+                      plot=F, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="linear", nPC=2)
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main=paste("Linear kernel, lambda=", result_cv$best_lambda) )
> text(result$Xv[,1], result$Xv[,2], labels = data_example4$lab)
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 2, 0.05), kernel="gaussian",
+                      bandwidth=1, nPC=2, plot=F, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="gaussian",
+                 bandwidth=1, nPC=2)
> plot(result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main=paste("Gaussian kernel, h=", 1, sep="") )
> text(result$Xv[,1], result$Xv[,2], labels = data_example4$lab)
```



The linear kernel does not seem work as well as the Gaussian kernel. The bandwidth for the Gaussian kernel is chosen to be 1. Note that $mean(abs(Y)) = 0.07$. We implement linear kernel with larger λ and Gaussian kernel with smaller bandwidth:

```
> result <- acPCA(X, Y, lambda=3, kernel="linear", nPC=2)
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> plot(result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="Linear kernel, lambda=3")
> text(result$Xv[,1], result$Xv[,2], labels = data_example4$lab)
> result_cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 2, 0.05), kernel="gaussian",
+                      bandwidth=0.1, nPC=2, plot=F, quiet=T)
> result <- acPCA(X, Y, lambda=result_cv$best_lambda, kernel="gaussian",
+                 bandwidth=0.1, nPC=2)
> plot(result$Xv[,1], result$Xv[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main=paste("Gaussian kernel, h=", 0.1, sep="") )
> text(result$Xv[,1], result$Xv[,2], labels = data_example4$lab)
```

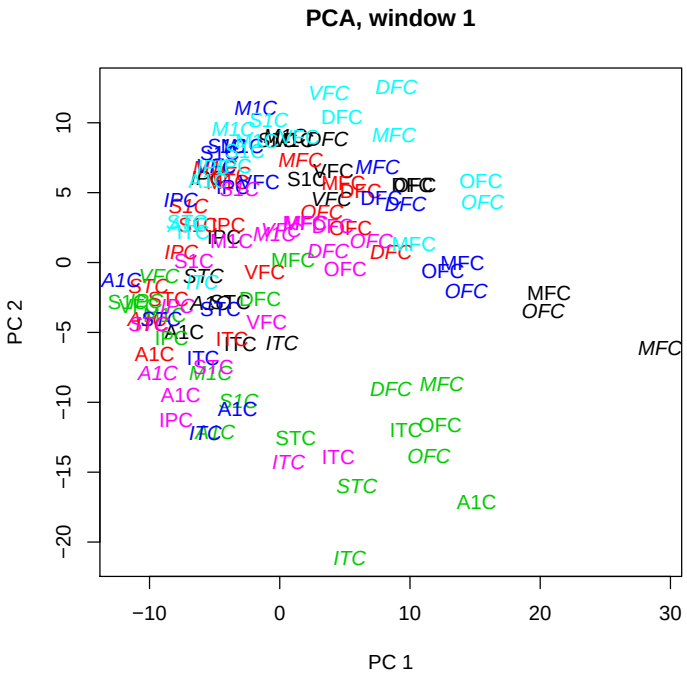


With larger λ than the optimal tuned value, the visualization result for linear kernel is better. Smaller bandwidth makes the visualization result for Gaussian kernel worse.

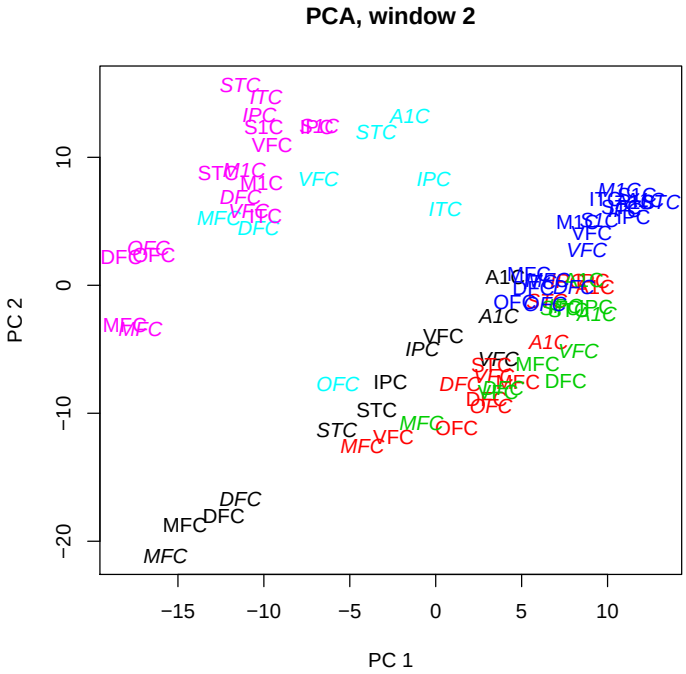
2.5 Application to the human brain exon array data

We analyze the human brain exon array data reported in (cite). We use a subset of $1,000$ genes for demonstration purpose. For time window 1 and 2, we first perform PCA to visualize the brain regions:

```
> load("~/Dropbox/AIPCA/R_package/data/data_w1NN.rda")
> load("~/Dropbox/AIPCA/R_package/data/data_w2NN.rda")
> X1 <- data_w1$X; X2 <- data_w2$X
> table(data_w1$hemispheres) ###1: left hemisphere, 3: right hemisphere

 1  3
59 60

> pca1 <- prcomp(X1, center=T) ###regular PCA
> pca2 <- prcomp(X2, center=T) ###regular PCA

> plot(pca1$x[,1], pca1$x[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="PCA, window 1")
> text(pca1$x[,1], pca1$x[,2], labels = data_w1$regions,
+      col=data_w1$donor_labs, font=data_w1$hemispheres)
```

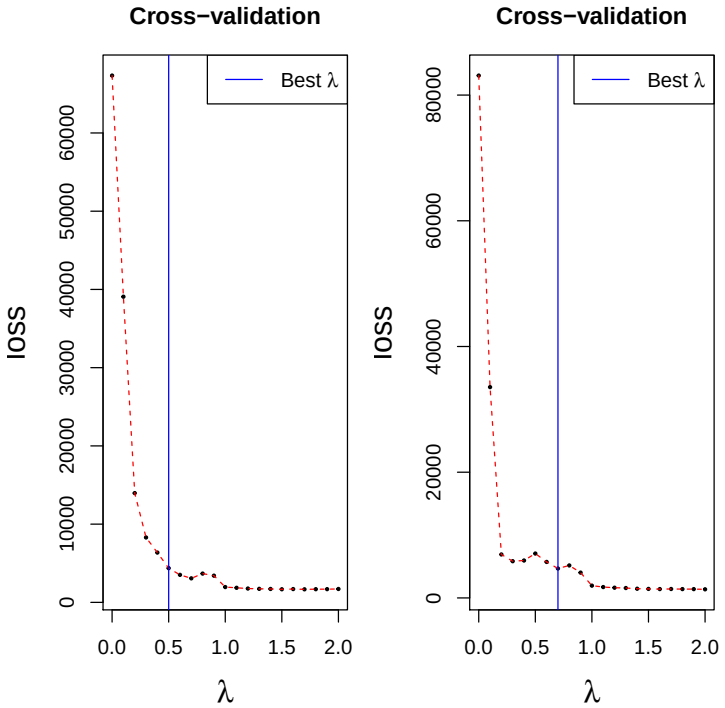



```
> plot(pca2$x[,1], pca2$x[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="PCA, window 2")
> text(pca2$x[,1], pca2$x[,2], labels = data_w2$regions,
+      col=data_w2$donor_labs, font=data_w2$hemispheres)
```



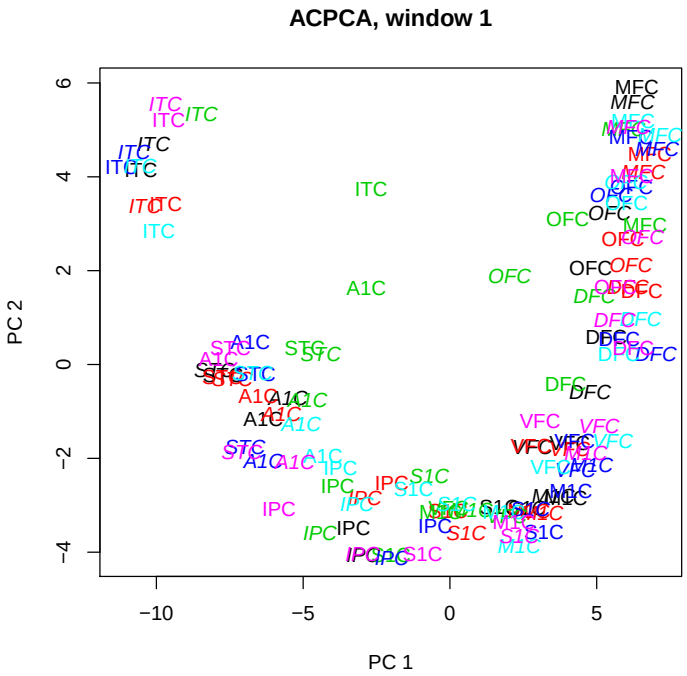
In window 1, there is no clear patterns across the regions. In window 2, samples from the same donor tend to form a cluster. To capture the interregional variation that is shared among donors, we implement AC-PCA with the confounder matrix designed similarly as that in the previous section. We first tune λ:

```
> Y1 <- data_w1$Y; Y2 <- data_w2$Y
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> result1cv <- acPCAcv(X=X1, Y=Y1, lambdas=seq(0, 2, 0.1),
+                      kernel="linear", nPC=2, plot=T, quiet=T)
> result2cv <- acPCAcv(X=X2, Y=Y2, lambdas=seq(0, 2, 0.1),
+                      kernel="linear", nPC=2, plot=T, quiet=T)
```

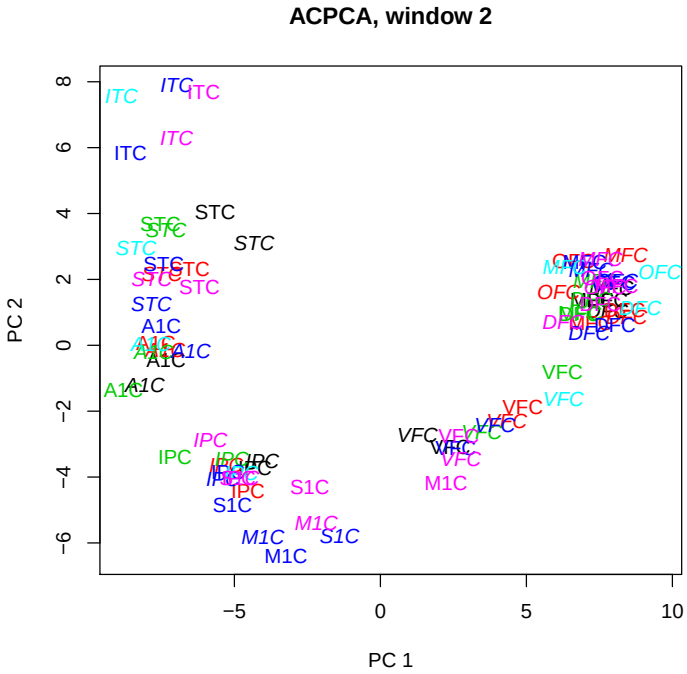


The best λs are used to implement AC-PCA:

```
> result1 <- acPCA(X=X1, Y=Y1, lambda=result1cv$best_lambda,
+                  kernel="linear", nPC=2)
> result2 <- acPCA(X=X2, Y=Y2, lambda=result2cv$best_lambda,
+                  kernel="linear", nPC=2)
> Xv1 <- result1$Xv
> Xv2 <- result2$Xv

> plot(Xv1[,1], Xv1[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="ACPCA, window 1")
> text(Xv1[,1], Xv1[,2], labels = data_w1$regions,
+      col=data_w1$donor_labs, font=data_w1$hemispheres)
```

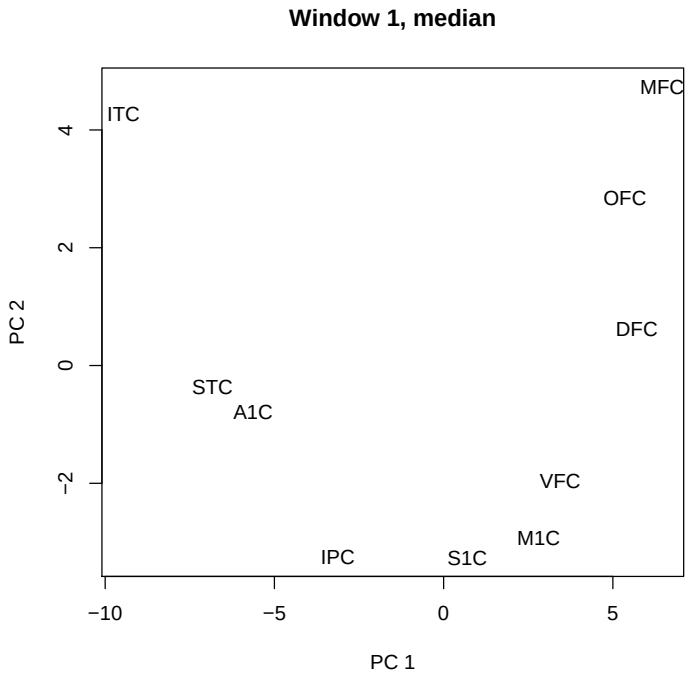



```
> plot(Xv2[,1], Xv2[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="ACPCA, window 2")
> text(Xv2[,1], Xv2[,2], labels = data_w2$regions,
+      col=data_w2$donor_labs, font=data_w2$hemispheres)
```



When we take median across donors, the pattern is easier to visualize:

```
> Xv1_med <- Xv2_med <- c()
> regs <- unique(data_w1$regions)
> for (reg in regs){
+   Xv1_med <- rbind(Xv1_med, apply(Xv1[which(data_w1$region==reg),], 2, mean) )
+   Xv2_med <- rbind(Xv2_med, apply(Xv2[which(data_w2$region==reg),], 2, mean) )
+ }

> plot(Xv1_med[,1], Xv1_med[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="Window 1, median")
> text(Xv1_med[,1], Xv1_med[,2], labels = regs)
```




```
> plot(Xv2_med[,1], Xv2_med[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="Window 2, median")
> text(Xv2_med[,1], Xv2_med[,2], labels = regs)
```

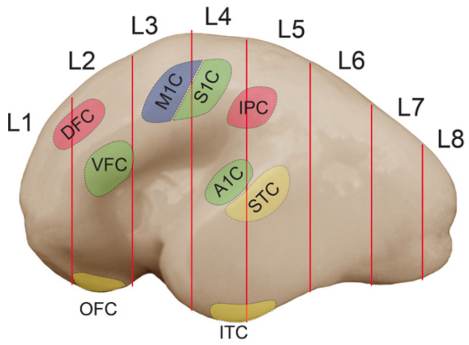
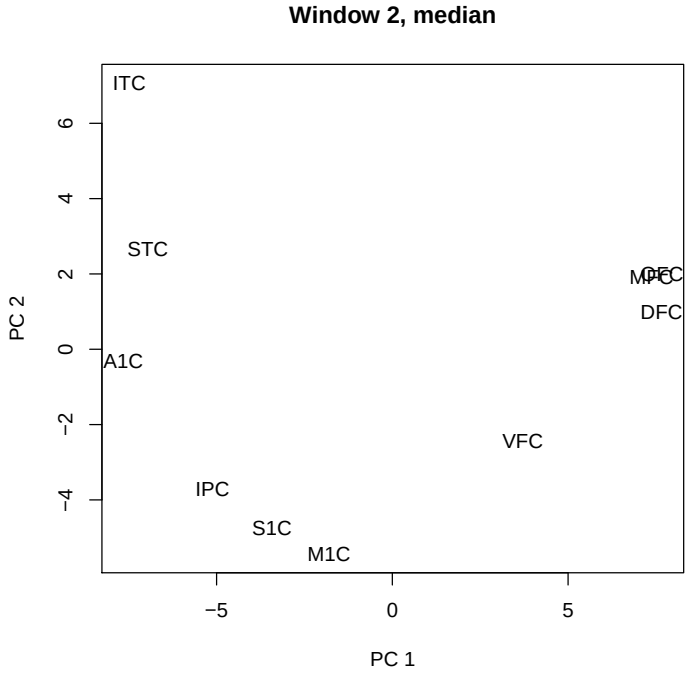


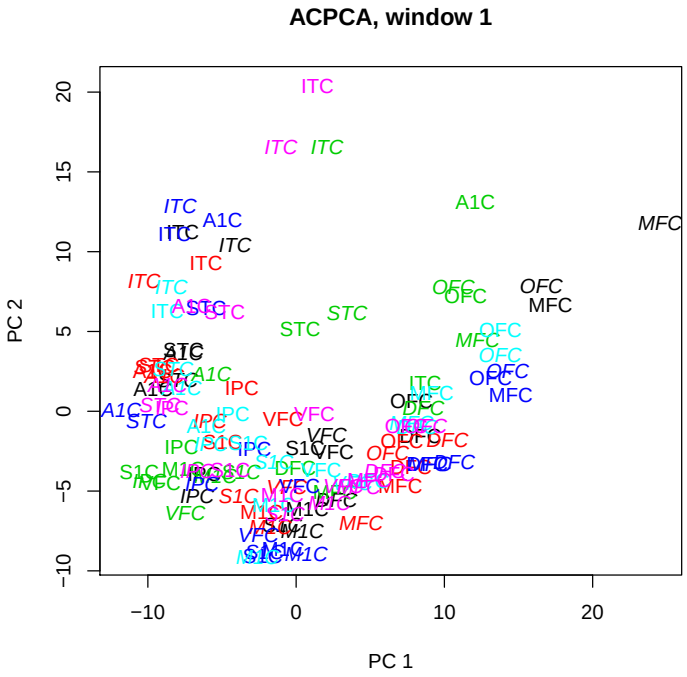
Figure 1: Representative fetal human brain, lateral surface of hemisphere cite



The pattern is in consistent with the anatomical structure of neocortex (Figure 1 cite):

An alternative way to adjust the confounding variation of donors is to set Y to be the factor levels of donors. However, the visualization result is not as good as the previous plot:

```
> Yid <- data_w1$Yid
> result1cv_2 <- acPCAcv(X=X1, Y=Yid, lambdas=seq(0, 2, 0.1),
+                        kernel="linear", nPC=2, plot=F, quiet=T)
> result1_2 <- acPCA(X=X1, Y=Yid, lambda=result1cv_2$best_lambda,
+                    kernel="linear", nPC=2)
> Xv1_2 <- result1_2$Xv
> plot(Xv1_2[,1], Xv1_2[,2], xlab="PC 1", ylab="PC 2",
+      col="black", type="n", main="ACPCA, window 1")
> text(Xv1_2[,1], Xv1_2[,2], labels = data_w1$regions,
+      col=data_w1$donor_labs, font=data_w1$hemispheres)
```



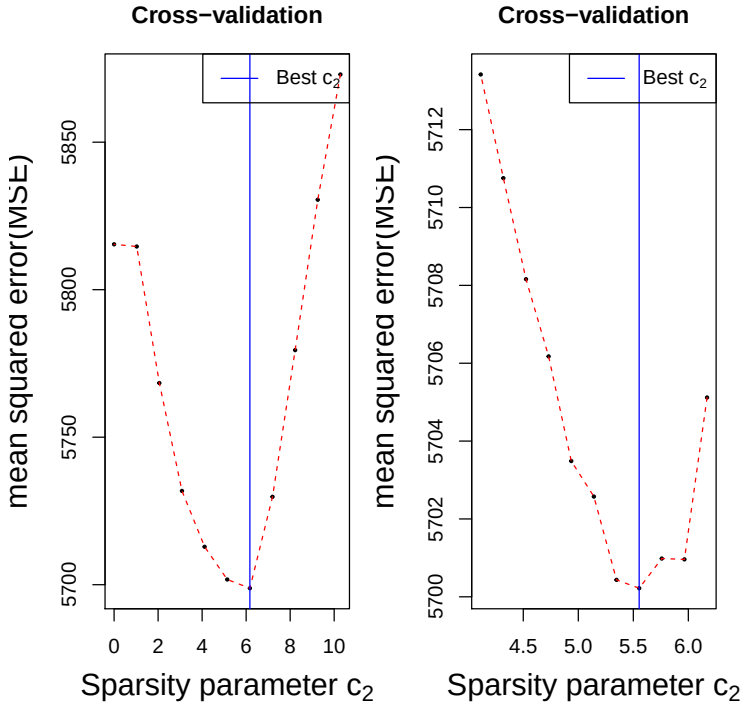
2.6 AC-PCA with sparsity

For AC-PCA with sparsity constrains, there are two tuning parameters, c_1 and c_2: c_1 controls the penalization of dependency between Xv and Y, and c_2 controls the sparsity of v. The parameters are tuned sequencially: c_1 is tuned without the sparsity constrain, and then c_2 is tuned with c_1 fixed.

Simulated example 5: the setting is similar to example 4, except that the true loadings are sparse.

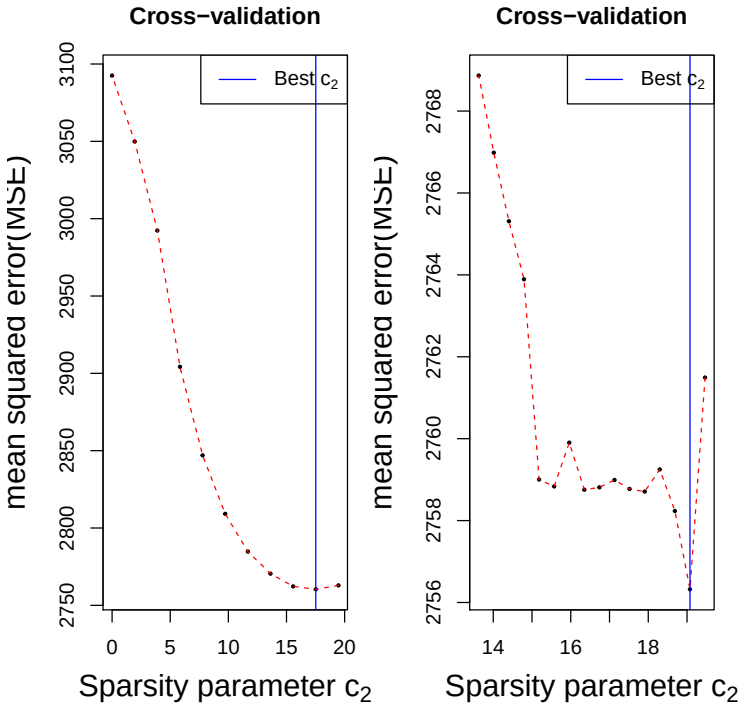
```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
```

```
> load("~/Dropbox/AIPCA/R_package/data/data_example5.rda")
> #data_w1 <- data_w4
> X <- data_example5$X ###the data matrix
> Y <- data_example5$Y
> result1cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 2, 0.05),
+                      kernel="linear", nPC=2, plot=F, quiet=T)
> result1 <- acPCA(X=X, Y=Y, lambda=result1cv$best_lambda,
+                  kernel="linear", nPC=2)
> v_ini <- as.matrix(result1$v[,1])
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> resultcv_spc1_coarse <- acSPCcvM( X=X, Y=Y, c2s=seq(1, 0, -0.1)*sum(abs(v_ini)),
+                                   v_ini=v_ini, kernel="linear", plot=T, quiet=T, fold=10)
> resultcv_spc1_fine <- acSPCcvM( X=X, Y=Y, c2s=seq(0.6, 0.4, -0.02)*sum(abs(v_ini)),
+                                 v_ini=v_ini, kernel="linear", plot=T, quiet=T, fold=10)
> result_spc1 <- acSPCM( X=X, Y=Y, c2=resultcv_spc1_fine$best_c2,
+                        v_ini=v_ini, kernel="linear")
> v1 <- result_spc1$v
> sum(v1!=0)

[1] 71

> sum(v1[301:400]!=0)

[1] 59
```



In simulated example 5, to speed-up the computational time, a coarse search for c_2 was performed first, followed by a finer grid. The 301 to 400 entries are non-zero in the true loading factor. Next we implement the procedure on the brain exon array data, time window 2:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_w2NN.rda")
> X <- data_w2$X ###the data matrix
> Y <- data_w2$Y
> result1cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 2, 0.05),
+                      kernel="linear", nPC=2, plot=F, quiet=T)
> result1 <- acPCA(X=X, Y=Y, lambda=result1cv$best_lambda,
+                  kernel="linear", nPC=2)
> v_ini <- as.matrix(result1$v[,1])
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> c2s <- seq(1, 0, -0.1)*sum(abs(v_ini))
> resultcv_spc1_coarse <- acSPCcvM( X=X, Y=Y, c2s=c2s, v_ini=v_ini,
+                                   kernel="linear", quiet=T, fold=10)
> c2s <- seq(1, 0.7, -0.02)*sum(abs(v_ini))
> resultcv_spc1_fine <- acSPCcvM( X=X, Y=Y, c2s=c2s, v_ini=v_ini,
+                                 kernel="linear", quiet=T, fold=10)
> result_spc1 <- acSPCM( X=X, Y=Y, c2=resultcv_spc1_fine$best_c2,
+                        v_ini=v_ini, kernel="linear")
> v1 <- result_spc1$v
> sum(v1!=0)

[1] 998
```

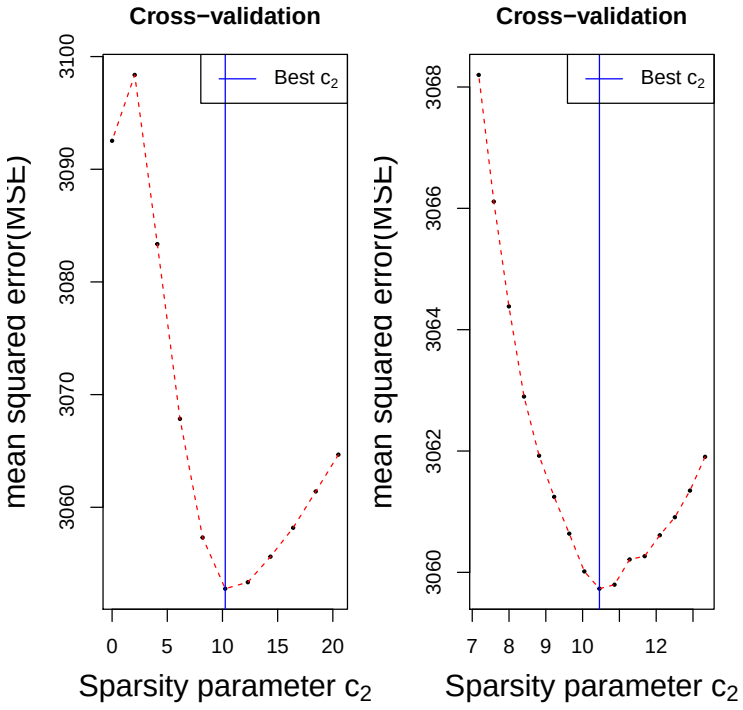


2.7 Multiple sparse principal components

Multiple sparse principal components can be obtained by substracting the first several principal components, and then implement the algorithm. We provide an example implementing the procedure on the brain exon array data, time window 2:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_w2NN.rda")
> X <- data_w2$X ###the data matrix
> Y <- data_w2$Y
> result1cv <- acPCAcv(X=X, Y=Y, lambdas=seq(0, 2, 0.05),
+                      kernel="linear", nPC=2, plot=F, quiet=T)
> result1 <- acPCA(X=X, Y=Y, lambda=result1cv$best_lambda,
+                  kernel="linear", nPC=2)
> v_substract <- as.matrix(result1$v[,1])
> v_ini <- as.matrix(result1$v[,2])
> par(mfrow=c(1,2), pin=c(2.5,2.5), mar=c(4.1, 3.9, 3.2, 1.1))
> c2s <- seq(1, 0, -0.1)*sum(abs(v_ini))
> resultcv_spc2_coarse <- acSPCcvM( X=X, Y=Y, c2s=c2s, v_ini=v_ini,
+                                   v_substract=v_substract, kernel="linear",
```

```
+                                    quiet=T, fold=10)
> ws <- seq(1, 0, -0.1)[which.min(resultcv_spc2_coarse$mse)]
> c2s <- seq(ws+0.15, ws-0.15, -0.02)*sum(abs(v_ini))
> resultcv_spc2_fine <- acSPCcvM( X=X, Y=Y, c2s=c2s, v_ini=v_ini,
+                                    v_substract=v_substract, kernel="linear",
+                                    quiet=T, fold=10)
> result_spc2 <- acSPCM( X=X, Y=Y, c2=resultcv_spc2_fine$best_c2, v_ini=v_ini,
+                        v_substract=v_substract, kernel="linear")
> v2 <- result_spc2$v
> sum(v2!=0)

[1] 249
```



In the example, the non-sparse first PC was substracted. We can also substract the sparse PC:

```
> v_substract <- v1
```

2.8 Implementation for RNA-Seq data

PCA generally works for "normal-like" data. For data that are far from "normal", the result may be driven by genes with large variation.

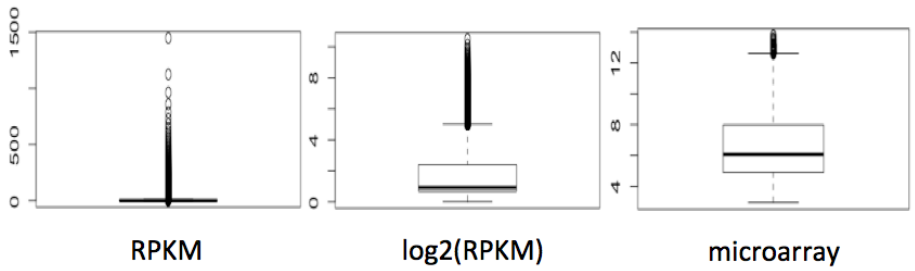


Figure 2: Boxplots comparing RNA-Seq and microarray, same biological sample

Figure 2 compares the distribution of gene expression levels between RNA-Seq and microarray experiments for the same biological sample. Log transformation of RPKM makes the data more "normal-like", but may not be sufficient: there are still many outliers. One way to get around this is to use rank:
(a) Rank among samples: for each gene, rank the expression levels among samples, then the expression levels will be 1,...,number of samples
(b) Rank among genes: in each sample, rank the gene expression levels among all genes, then the expression level will be 1,...,number of genes.
Quantile normalization to a normal distribution can be further implemented for the rank among genes.

Based on our experience, both approaches seem to work well for visualization purpose.

3 Theory behind AC-PCA

3.1 AC-PCA in a general form

Let X be the $N \times p$ data matrix and Y be the $N \times l$ matrix for l confounding factors. Denote y_i the ith row in Y. We propose the following objective function to adjust for confounding variation:

$$\begin{aligned} \underset{v \in \mathbb{R}^p}{\text{maximize}} \quad & v^T X^T X v - \lambda v^T X^T K X v \\ \text{subject to} \quad & ||v||_2^2 \leq 1, \end{aligned} \tag{1}$$

where K is the $N \times N$ kernel matrix, and $K_{ij} = k(y_i, y_j)$. It can be shown that $v^T X^T K X v$ is the same as the empirical Hilbert-Schmidt independence criterion for Xv and Y, when linear kernel is applied on Xv. In the objective function, we are penalizing the dependence between Xv (i.e. extracted feature) and the confounding factors.

3.2 AC-PCA with sparse loading

A sparse solution for v can be achieved by adding ℓ_1 constraint:

$$\underset{v\in\mathbb{R}^p}{\text{maximize}}\ v^T X^T X v \quad \text{subject to } v^T X^T K X v \le c_1,\ ||v||_1 \le c_2,\ ||v||_2^2 \le 1. \quad (2)$$

4 When to use AC-PCA?

The loss $l = v^T X^T K X v$ is a measure of dependency between Y and Xv. The greater the loss, Y and Xv tend to be more dependent. When the variation associated with Y is small, the loss tends to be small for $\lambda = 0$. Therefore a quick way to check is to generate random Ys and compare the loss:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_example1.rda")
> X <- data_example1$X ###the data matrix
> Y <- data_example1$Y ###the confounder matrix
> v <- prcomp(X, center=T)$rotation[,1]
> Xv <- X%*%as.matrix(v)
> K <- calkernel(Y, kernel="linear")
> loss <- crossprod(Xv, K%*%Xv) ###the loss for Y
> loss_random <- c() ###the loss for random Y
> for (run in 1:10){
+   Yrandom <- apply(Y, 2, sample) ###a random Y
+   Krandom <- calkernel(Yrandom, kernel="linear")
+   loss_random <- c(loss_random, crossprod(Xv, Krandom%*%Xv))
+ }
> loss/loss_random

 [1]  8.266089 69.381071 15.561694 39.842222  6.543025 42.007434 16.555028
 [8] 49.717780 13.092926 56.519476
```

The loss for using Y is much larger than using random Y. Here is the result for the brain exon array data using donor labels as Y:

Window 1:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_w1NN.rda")
> X <- data_w1$X ###the data matrix
> Y <- data_w1$Yid
> ###left and right hemispheres are treated as different individuals
> v <- prcomp(X, center=T)$rotation[,1]
> Xv <- X%*%as.matrix(v)
> K <- calkernel(Y, kernel="linear")
> loss <- crossprod(Xv, K%*%Xv) ###the loss for Y
```

```
> loss_random <- c() ###the loss for random Y
> for (run in 1:10){
+   Yrandom <- apply(Y, 2, sample) ###a random Y
+   Krandom <- calkernel(Yrandom, kernel="linear")
+   loss_random <- c(loss_random, crossprod(Xv, Krandom%*%Xv))
+ }
> loss/loss_random

 [1] 1.3069309 0.6020095 0.6073669 1.0415394 1.2274799 2.4073358 1.4212115
 [8] 1.5732521 2.7743790 1.2521630
```

Window 2:

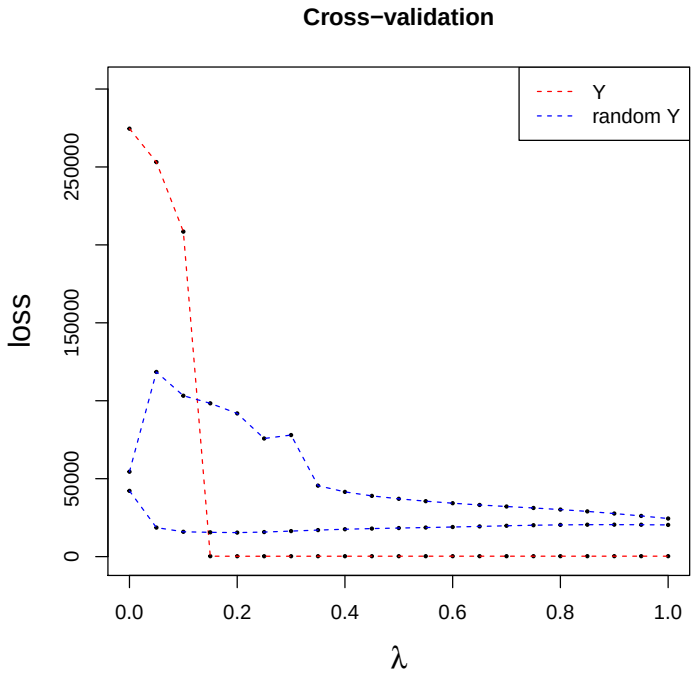
```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_w2NN.rda")
> X <- data_w2$X ###the data matrix
> Y <- data_w2$Yid ###the confounder matrix
> v <- prcomp(X, center=T)$rotation[,1]
> Xv <- X%*%as.matrix(v)
> K <- calkernel(Y, kernel="linear")
> loss <- crossprod(Xv, K%*%Xv) ###the loss for Y
> loss_random <- c() ###the loss for random Y
> for (run in 1:10){
+   Yrandom <- apply(Y, 2, sample) ###a random Y
+   Krandom <- calkernel(Yrandom, kernel="linear")
+   loss_random <- c(loss_random, crossprod(Xv, Krandom%*%Xv))
+ }
> loss/loss_random

 [1]  8.876579  7.248348 12.930141  5.855964 19.334887  5.030696  7.481942
 [8] 13.913875 11.911634  5.660333
```

The variation associated with donors is much stronger in window 2.

The λ vs. loss plot also provides evidence of confounding variation. When there is confounding variation associated with Y, the loss will drop sharply and then become stable when λ increases. When there is little confounding variation associated with Y, the λ vs. loss plot tend to be flat:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_example1.rda")
> X <- data_example1$X ###the data matrix
> Y <- data_example1$Y ###the confounder matrix
> Yrandom1 <- apply(Y, 2, sample)
> Yrandom2 <- apply(Y, 2, sample)
> lambdas <- seq(0, 1, 0.05)
```

```
> result_cv1 <- acPCAcv(X=X, Y=Y, lambdas=lambdas,
+                       kernel="linear", nPC=2, plot=F, quiet=T)
> result_cv2 <- acPCAcv(X=X, Y=Yrandom1, lambdas=lambdas,
+                       kernel="linear", nPC=2, plot=F, quiet=T)
> result_cv3 <- acPCAcv(X=X, Y=Yrandom2, lambdas=lambdas,
+                       kernel="linear", nPC=2, plot=F, quiet=T)
> loss1 <- result_cv1$loss; loss2 <- result_cv2$loss; loss3 <- result_cv3$loss
> plot(lambdas, loss1, ylab="loss", xlab=expression(lambda), pch=20, cex=0.5,
+      ylim=c(0, max(c(loss1,loss2))*1.1), main="Cross-validation", cex.lab=1.5)
> lines(lambdas, loss1, col="red", lty=2)
> points(lambdas, loss2, pch=20, cex=0.5)
> lines(lambdas, loss2, col="blue", lty=2)
> points(lambdas, loss3, pch=20, cex=0.5)
> lines(lambdas, loss3, col="blue", lty=2)
> legend( "topright", legend=c("Y", "random Y"), lty=2, col=c("red", "blue"))
```



However, special attention is required when Y is designed as in section unobserved confounders. The loss for Y can be similar or even smaller than random Y:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_example4.rda")
```

```
> X <- data_example4$X ###the data matrix
> Y <- data_example4$Y ###the confounder matrix
> v <- prcomp(X, center=T)$rotation[,1]
> Xv <- X%*%as.matrix(v)
> K <- calkernel(Y, kernel="linear")
> loss <- crossprod(Xv, K%*%Xv) ###the loss for Y
> loss_random <- c() ###the loss for random Y
> for (run in 1:10){
+   Yrandom <- apply(Y, 2, sample) ###a random Y
+   Krandom <- calkernel(Yrandom, kernel="linear")
+   loss_random <- c(loss_random, crossprod(Xv, Krandom%*%Xv))
+ }
> loss/loss_random

 [1] 0.9315521 1.0702277 1.0421758 1.2071408 0.8741620 0.9914818 1.1043110
 [8] 0.8164731 0.9305006 1.0893215
```

Here is the result for the brain exon array data. Window 1:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_w1NN.rda")
> X <- data_w1$X ###the data matrix
> Y <- data_w1$Y ###the confounder matrix
> v <- prcomp(X, center=T)$rotation[,1]
> Xv <- X%*%as.matrix(v)
> K <- calkernel(Y, kernel="linear")
> loss <- crossprod(Xv, K%*%Xv) ###the loss for Y
> loss_random <- c() ###the loss for random Y
> for (run in 1:10){
+   Yrandom <- apply(Y, 2, sample) ###a random Y
+   Krandom <- calkernel(Yrandom, kernel="linear")
+   loss_random <- c(loss_random, crossprod(Xv, Krandom%*%Xv))
+ }
> loss/loss_random

 [1] 0.4150267 0.4282357 0.4293525 0.4770684 0.4432589 0.4305454 0.3926965
 [8] 0.4338578 0.4203871 0.3895510
```

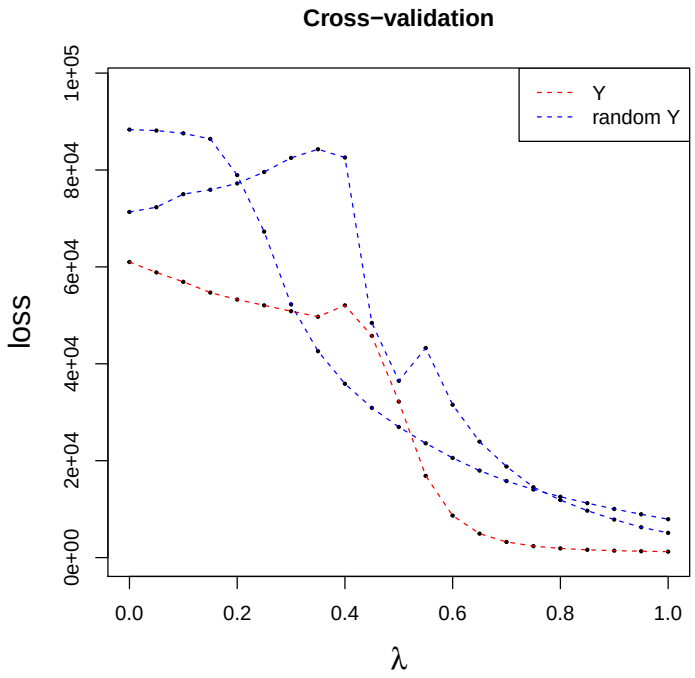
Window 2:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_w2NN.rda")
> X <- data_w2$X ###the data matrix
> Y <- data_w2$Y ###the confounder matrix
> v <- prcomp(X, center=T)$rotation[,1]
```

```
> Xv <- X%*%as.matrix(v)
> K <- calkernel(Y, kernel="linear")
> loss <- crossprod(Xv, K%*%Xv) ###the loss for Y
> loss_random <- c() ###the loss for random Y
> for (run in 1:10){
+   Yrandom <- apply(Y, 2, sample) ###a random Y
+   Krandom <- calkernel(Yrandom, kernel="linear")
+   loss_random <- c(loss_random, crossprod(Xv, Krandom%*%Xv))
+ }
> loss/loss_random

 [1] 0.9030121 0.8466617 1.0296871 1.0249619 0.8101540 0.8667743 0.8926253
 [8] 0.9561484 0.8629107 0.8600722
```

The trend for the λ vs. loss plot can also be similar:

```
> library(rARPACK)
> source("~/Dropbox/brain_gradient/code/function_acpca.R")
> load("~/Dropbox/AIPCA/R_package/data/data_example4.rda")
> X <- data_example4$X ###the data matrix
> Y <- data_example4$Y ###the confounder matrix
> Yrandom1 <- apply(Y, 2, sample)
> Yrandom2 <- apply(Y, 2, sample)
> lambdas <- seq(0, 1, 0.05)
> result_cv1 <- acPCAcv(X=X, Y=Y, lambdas=lambdas,
+                       kernel="linear", nPC=2, plot=F, quiet=T)
> result_cv2 <- acPCAcv(X=X, Y=Yrandom1, lambdas=lambdas,
+                       kernel="linear", nPC=2, plot=F, quiet=T)
> result_cv3 <- acPCAcv(X=X, Y=Yrandom2, lambdas=lambdas,
+                       kernel="linear", nPC=2, plot=F, quiet=T)
> loss1 <- result_cv1$loss; loss2 <- result_cv2$loss; loss3 <- result_cv3$loss
> plot(lambdas, loss1, ylab="loss", xlab=expression(lambda), pch=20, cex=0.5,
+      ylim=c(0, max(c(loss1,loss2))*1.1), main="Cross-validation", cex.lab=1.5)
> lines(lambdas, loss1, col="red", lty=2)
> points(lambdas, loss2, pch=20, cex=0.5)
> lines(lambdas, loss2, col="blue", lty=2)
> points(lambdas, loss3, pch=20, cex=0.5)
> lines(lambdas, loss3, col="blue", lty=2)
> legend( "topright", legend=c("Y", "random Y"), lty=2, col=c("red", "blue"))
```



Howver, under this setting, we are more explicit on the desired variation: the variation that is shared across replicates. Therefore the choice between AC-PCA and regular PCA is less of an issue.