A Mixture of Flexible Multivariate Distributions

In this document, I will compute the posterior distribution for β_j using the derivation in the document 'next Steps', where the prior on β_j is modeled as a mixture of multivariate normals.

```
tuto.dir=getwd()
##' @param b.gp.hat PxR matrix of standardized effect sizes across all `R' tissue types,
##' @param se.gp.hat RxR estimated covariance matrix of standardized standard errors, corresponding to
##' @param t.stat PxR matrix of t statistics for each gene-snp Pair across all R tissues
##' @param U.0kl RxR prior covariance matrix for posterior.covariance matrix K and weight l
##' @param pi LxK matrix of prior weights estimated from the EM algorithm which correspond to optimal w

b.gp.hat=na.omit(read.table("16008genesnppairs_43tissues_beta.hat.std.txt",header=F,skip=1)[,-c(1,2)])
se.gp.hat=na.omit(read.table("16008genesnppairs_43tissues_beta.hat.std.txt",header=F,skip=1)[,-c(1,2)])
t.stat=na.omit(read.table("16008genesnppairs_43tissues_t.stat.txt",header=F,skip=1)[,-c(1,2)])
L = #(number of grid weights to use)
R=ncol(b.gp.hat)#number of tissues
X.t=as.matrix(t.stat)
X.c=apply(X.t,2,function(x) x-mean(x)) ##Column centered matrix of t statistics
#colMeans(X.c)
```

Now, we need to load in the prior matrices which we will try to find the optimal combination. First we load in the 'K' RxR prior covariance matrices specifying the relative importance of a particular tissue direction in each component.

```
##' @param R tissues from which to estimate covariance matrices
##' @param X.c a matrix of column center tstatistics
##' @param K number of PCs to keep in approximation
##' @return return K component list of covariance matrices (SFA or SVD approximations)
get.prior.covar.U.0=function(R,X.c,P,omega){

  U.0.=list()
  U.0.[[1]]=omega*diag(1,R)# the first covariance matrix will be the 'sopped up' identity
  U.0.[[2]]=omega*(t(X.c)%*%X.c)/(nrow(X.c))
  svd.X=svd(X.t)##perform SVD on uncentered matrix
  v=svd.X$v;u=svd.X$u;d=svd.X$d

cov.pc=1/P*v[,1:P]%*%diag(d[1:P])%*%t(v[,1:P])##Use the rank P summary representation

  U.0.[[3]]=omega*cov.pc
return(U.0.)}

U.0=get.prior.covar.U.0(R,X.c,13,omega=0.2)
```

Now, for every prior covariance matrix U_k, we compute L 'stretches' specifying the width of this distribution.

```
##' @param function to return the K component list of prior covariance matrices
##' @param X.c a matrix of column center tstatistics
##' @param K number of PCs to keep in approximation
##' @return return L dimensional list of K dimensional lists where each L,K contains the Lth grid compo

omega.table=read.table("~/Dropbox/cyclingstatistician/beta_gp_continuous/omega2.txt")
```

```
get.prior.covar.Ukl=function(P,L,R){
  U.0kl=lapply(seq(1:L),function(l){ ##For each element of omega, computes the J covariance matrices
  omega=omega.table[l,]
get.prior.covar.U.0(R,X.c,P,omega)
})
return(U.0kl)}
```

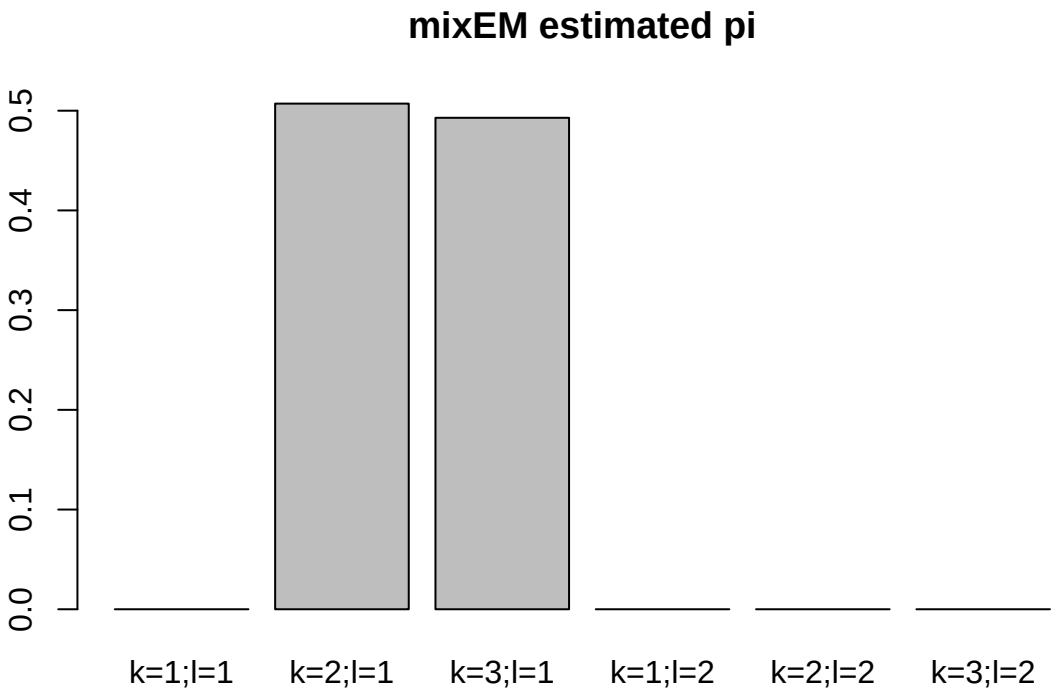
Now that we have the prior covariance matrices, we can maximize the likelihood $L(\pi; \hat{\beta}_j, se_j)$ by computing $Pr(\hat{\beta}_j | Componentk, l)$ for each gene componenet at each each gene SNP pair and then finding the optimal combination among all gene SNP pairs.

```
##' @return Compute a likelihood using the prior covariance matrix for each gene SNP pair in the rows ar
##' @param b.gp.hat and se.gp.hat matrix of MLES for all J gene snp pairs
##'
##' @param U.0kl L dimensional list of K dimensional list with prior covairnace matrix for each grid we
#install.packages("SQUAREM") note you'll need to have SQUAREM installed
library("SQUAREM")
K=3
L=2
R=43
U.0kl=get.prior.covar.Ukl(P=13,L,R)
#U.0kl[[L+1]]=diag(1,R) ##Add the identity matrix to the list

##J = Number of gene-snp pairs to consider
library("mvtnorm")
likelihood=function(b.gp.hat,se.gp.hat,J=nrow(b.gp.hat))
  {
  lik.i=list()
lapply(seq(1:J),function(j){
  b.mle=b.gp.hat[j,]
  V.gp.hat=diag(se.gp.hat[j,])^2

   lik=matrix(NA,nrow=K,ncol=L)
 for(l in 1:L){
for (k in 1:K){
  lik[k,l]=dmvnorm(x=b.mle, sigma=U.0kl[[l]][[k]] + V.gp.hat)
  }}
   lik.i[[j]] = as.vector(lik) ## concatenates column by column (e.g., [K=1,1],[K=2,1],[K=3,1],[2,1])


})}

##Likelihood of each component in cols, gene=snp pairs in rows##
global.lik=matrix(unlist(likelihood(b.gp.hat,se.gp.hat)),ncol=K*L,byrow=TRUE)
```

Now, to use the EM algorithm:

```
#install.packages("SQUAREM")
library("SQUAREM")
pis=mixEM(matrix_lik=global.lik,prior=rep(1,K*L))
names.vec=matrix(NA,nrow=K,ncol=L)
  for(l in 1:L){
```

```
    for(k in 1:K){
      names.vec[k,l]=paste0("k=",k,";l=",l)}}

write.table((cbind(as.vector(names.vec),pis$pihat)),quote=FALSE,file="piklhat.txt")
pi.hat=read.table(file="piklhat.txt")
barplot(pis$pihat,names=as.vector(names.vec),main="mixEM estimated pi")
```

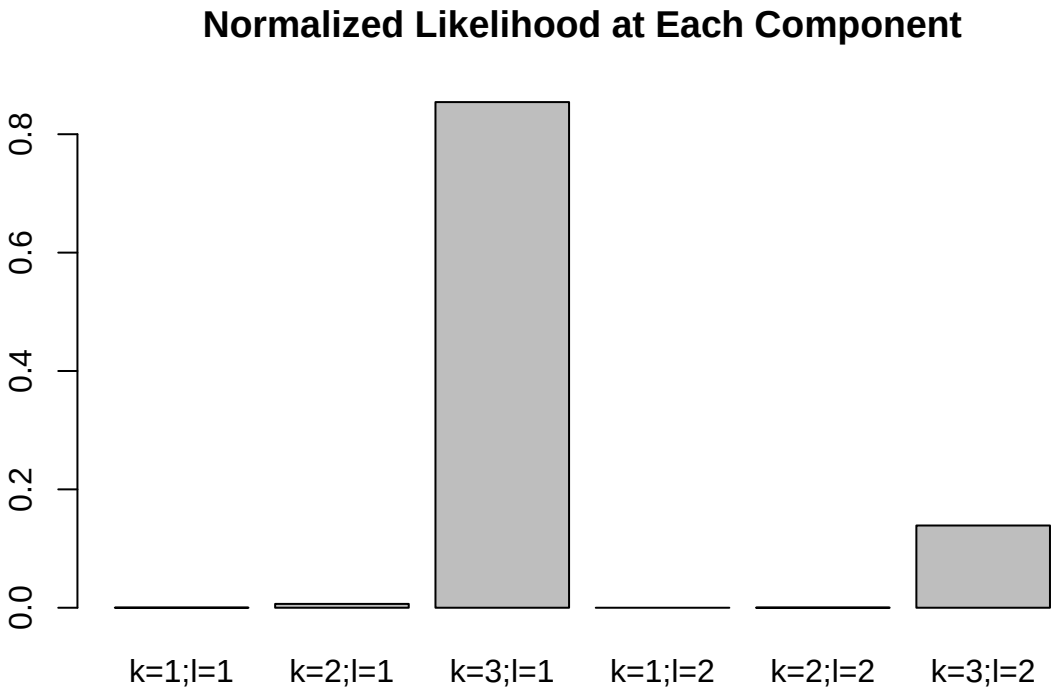


We can see that the majority of the weight is put on the covariance matrix of t statistics, $X_t'X$ and the estimated covariance matrix, $V_{t,1..P}\lambda V'_{t,1..P}$.

I compare with naiive iteration, which simply weights the relative likelihood across individuals.

```
##' @param global.lik Computes the likelihood matrices for each componenet for each of j gene snp pairs
##' @return global.sum Sums the likelihood for each component across j gene snp pairs
##' @return global.norm Sums the likelihood for all components across all j pairs


library("mvtnorm")

updated.weights=function(b.gp.hat,se.gp.hat){
  #global.lik=likelihood(b.gp.hat,se.gp.hat)
  global.sums=Reduce('+', likelihood(b.gp.hat,se.gp.hat)) ##This sums all matrices, so we get one KxL ma

global.norm=Reduce('+',lapply(global.lik,sum))
return(updated.weights=global.sums/global.norm)}

names.vec=matrix(NA,nrow=K,ncol=L)
  for(l in 1:L){
    for(k in 1:K){
      names.vec[k,l]=paste0("k=",k,";l=",l)}}
```

```
x=updated.weights(b.gp.hat,se.gp.hat)
barplot(as.vector(x),names=as.vector(names.vec),main="Normalized Likelihood at Each Component")
```




```
##Test with prior weights as 1/31
pi=rep(1/length(K*L),K*L)
```

Here, I've plotted the relative importance of each component after one iteration with a uniform prior weight on each π.

Recall:

$$L(\beta_j; k, l) = Pr(\hat{b}_j | k, l)$$
$$= Pr(\hat{b}_j; 0, \omega_l^2 U_k + \hat{V})$$

$$w_{jkl} = \frac{\pi_{k,l} L(\beta_j; k, l)}{\sum_{k,l} \pi_{k,l} L(\beta_j; k, l)}$$

We can then update our estimate of $\pi_{k,l}$

$$\pi_{kl}^{i+1} = \frac{\sum_j w_{jkl}}{\sum_{j,k,l} w_{jkl}}$$

Posterior computation

Now that we have the hierarchical weights $\pi_{k,l}$, we can compute the posterior distribution for each component.

For a given prior covariance matrix, compute posterior covariance and posterior mean. Here, let U.0k.l represent a specific matrix in U.0kl (.e.g, $U.0kl[[l]][[k]]$)

```
##' @param U.0k.l let U.0k.l represent a specific matrix in U.0kl (.e.g, U.0kl[[l]][[k]])
post.b.gpkl.cov <- function(V.gp.hat.inv, U.0k.l){
        U.gp1kl <- U.0k.l %*% solve(V.gp.hat.inv %*% U.0k.l + diag(nrow(U.0k.l)))
            return(U.gp1kl)
    }

post.b.gpkl.mean <- function(b.mle, V.gp.hat.inv, U.gp1kl){
        mu.gp1kl <- U.gp1kl %*% V.gp.hat.inv %*% b.mle
            return(mu.gp1kl)
    }
```

We also need to compute the "posterior weights" corresponding to each prior covariance matrix, which is simply the likelihood evaluated at that componenet times the prior weigth, $pi_{k,l}$ normalized by the marginal likelihood over all components.

$p(k=1,l=1|D) = \frac{p(D|k=1,l=1)*p(k=1,l=1)}{p(D)}$

```
##' @param pi.hat = matrix of prior weights
##' @return a vector of posterior weights
pis=pi.hat[,2]
post.weight.func=function(pis,U.0kl,V.gp.hat,b.mle){
post.weight.num = matrix(NA,nrow=K,ncol=L)
for(k in 1:K){
for(l in 1:L){
  wts=matrix(pis,nrow=3,ncol=2)
  pi=wts[k,l]
  post.weight.num[k,l]=pi*dmvnorm(x=b.mle, sigma=U.0kl[[l]][[k]] + V.gp.hat)}
}
post.weight=post.weight.num/sum(post.weight.num)
return(as.vector(post.weight))}
```

Now, for each gene-snp pair j and each prior covariance matrix U_{0kl} I will generate a 43 x 43 posterior covariance matrix and 43 x 1 vector of posterior means.

```
##' @param U.0kl = l dimensional list of k dimensional list of prior covariance matrices
##' @return all, a J dimensional list of k dimensional lists of L dimensional list of posterior mean an
##' @return post.weight.matrix a J x (K*L) matrix of posterior weights coresponding to p(J,L|D) for eac


all.covs=list()
all.means=list()
J=dim(b.gp.hat)[1]
#lapply(seq(1:J),function(j){
  for(j in 1:J){
    b.mle=as.vector(t(b.gp.hat[j,]))##turn i into a 43 x 1 vector
    V.gp.hat=diag(se.gp.hat[j,])^2
    all.covs[[j]]=list()
    all.means[[j]]=list()
    temp.mean=matrix(NA,nrow=)

            V.gp.hat.inv <- solve(V.gp.hat)
          for(k in 1:K){
            all.covs[[j]][[k]]=list()
```

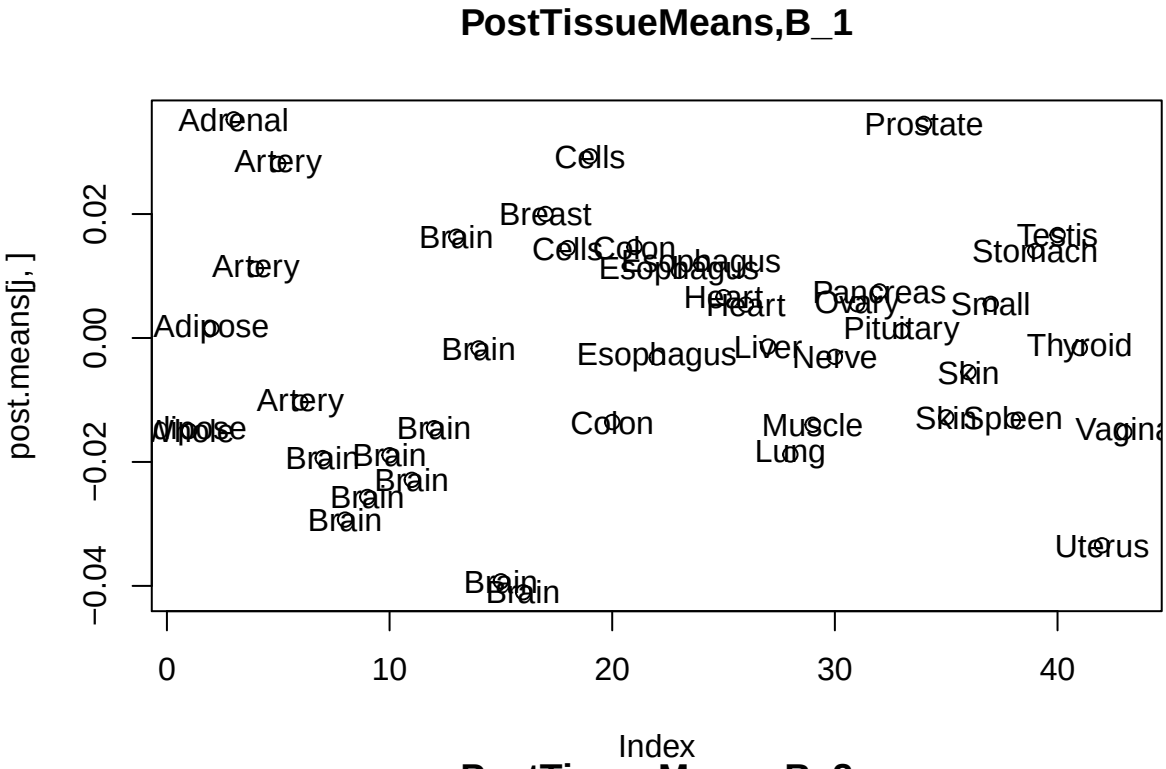
```
            all.means[[j]][[k]]=list()
            for (l in 1:L){
                  all.means[[j]][[k]][[l]]=list()
                    all.covs[[j]][[k]][[l]]=list()
                  U.gp1kl <- post.b.gpkl.cov(V.gp.hat.inv, U.0kl[[l]][[k]]) ## returns an R*R posterior
                  mu.gp1kl <- post.b.gpkl.mean(b.mle, V.gp.hat.inv, U.gp1kl) ##return a 1 * R vector of
                  all.means[[j]][[k]][[l]]=mu.gp1kl
                  all.covs[[j]][[k]][[l]]=U.gp1kl
                      }
          }}
```

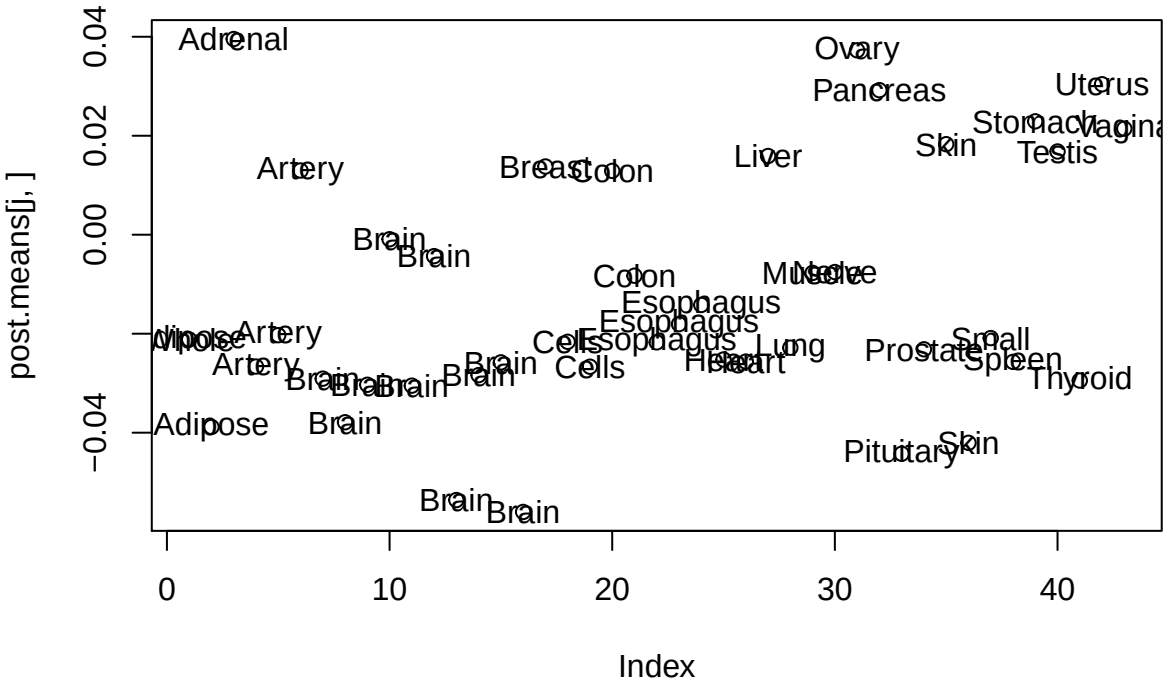
Now, for each of the J gene-snp pairs we generate a matrix of posterior weight matrix. Remember, this will not be tissue specific information, so we need to do it only once and can store in an Jx(K*L) matrix

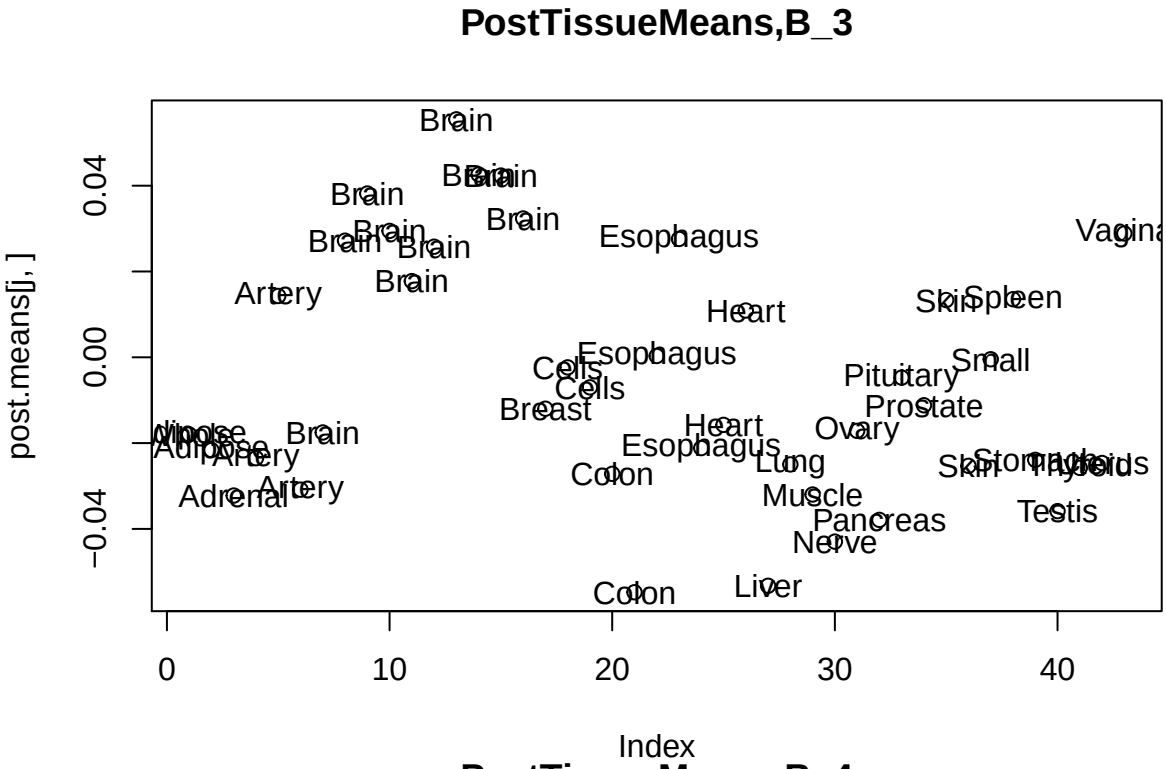
```
library("mvtnorm")
post.weight.matrix=matrix(NA,nrow=J,ncol=L*K)
J=dim(b.gp.hat)[1]


for(j in 1:J){
    b.mle=as.vector(t(b.gp.hat[j,]))##turn i into a 43 x 1 vector
    V.gp.hat=diag(se.gp.hat[j,])^2
    pis=pi.hat[,2]
    U.0kl=U.0kl
    post.weight.matrix[j,]=as.vector(post.weight.func(pis,U.0kl,V.gp.hat,b.mle))
}
```

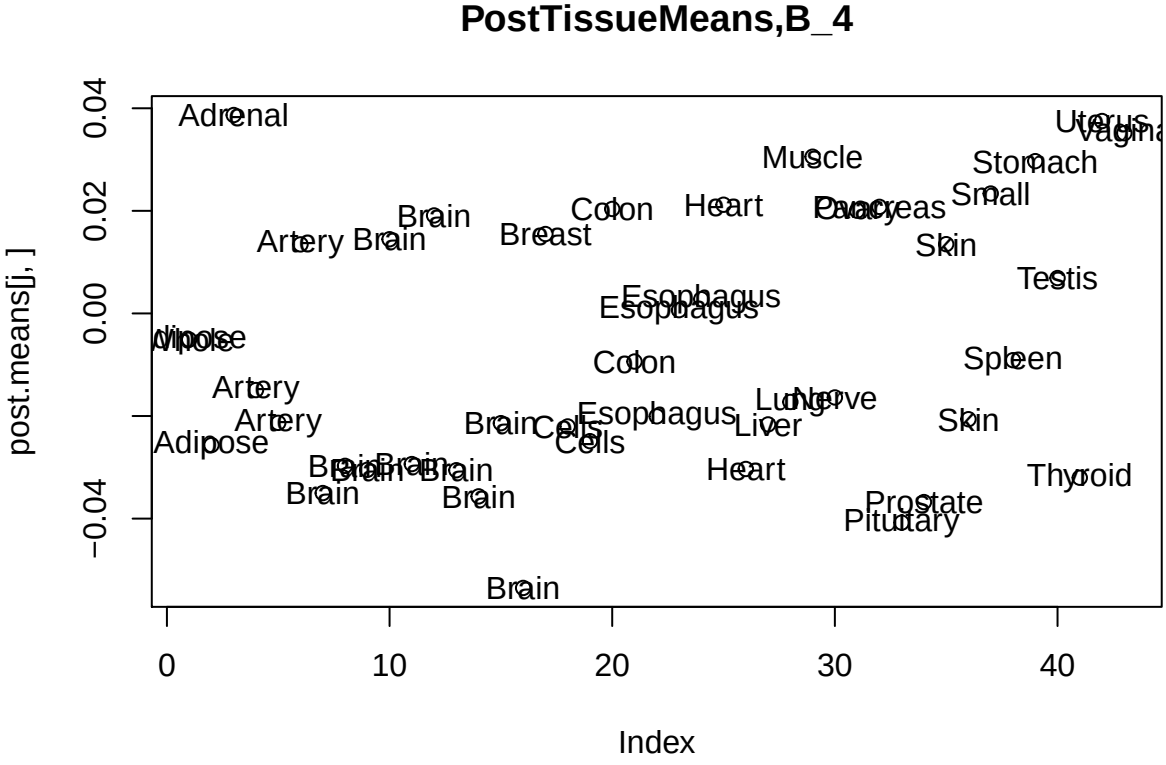
We can compute the overall posterior mean for each gene-SNP pair across tissues (i.e., the weighted R dimensional posterior mean vector of mu_j). Here, I do it for 10 gene SNP pairs.

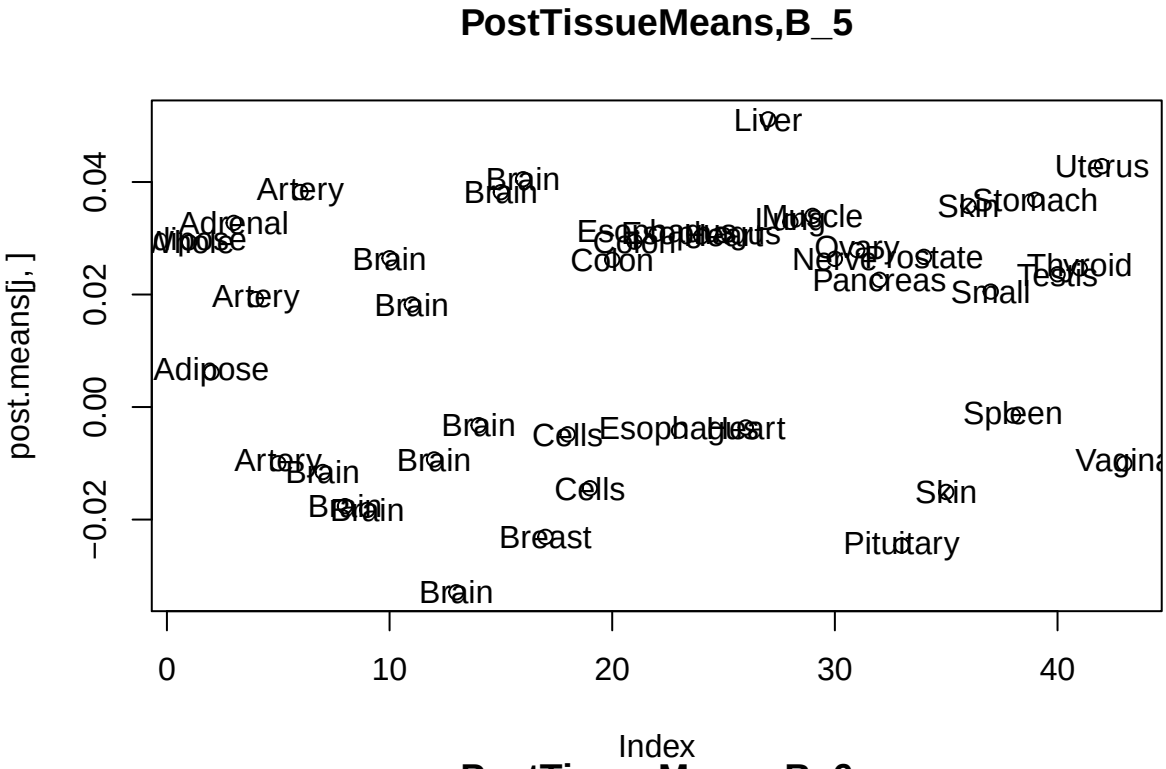
```
J=10
for(j in 1:J){
post.means=matrix(NA,nrow=J,ncol=R)
mean.list=all.means[[j]]
post.weights=post.weight.matrix[j,]
post.weight.mat=matrix(post.weights,nrow=3,ncol=2)
post.weight.mat[is.na(post.weight.mat)] <- 0

temp=NULL
for(k in 1:K){
for(l in 1:L){
temp=rbind(as.vector(post.weight.mat[k,l]*mean.list[[k]][[l]]),temp)##creates a 6x43 matrix of posterio
}
}
post.means[j,]=t(colSums(temp))

plot(post.means[j,],main=paste0("PostTissueMeans,B_",j))
text(post.means[j,],labels=names)
}
```

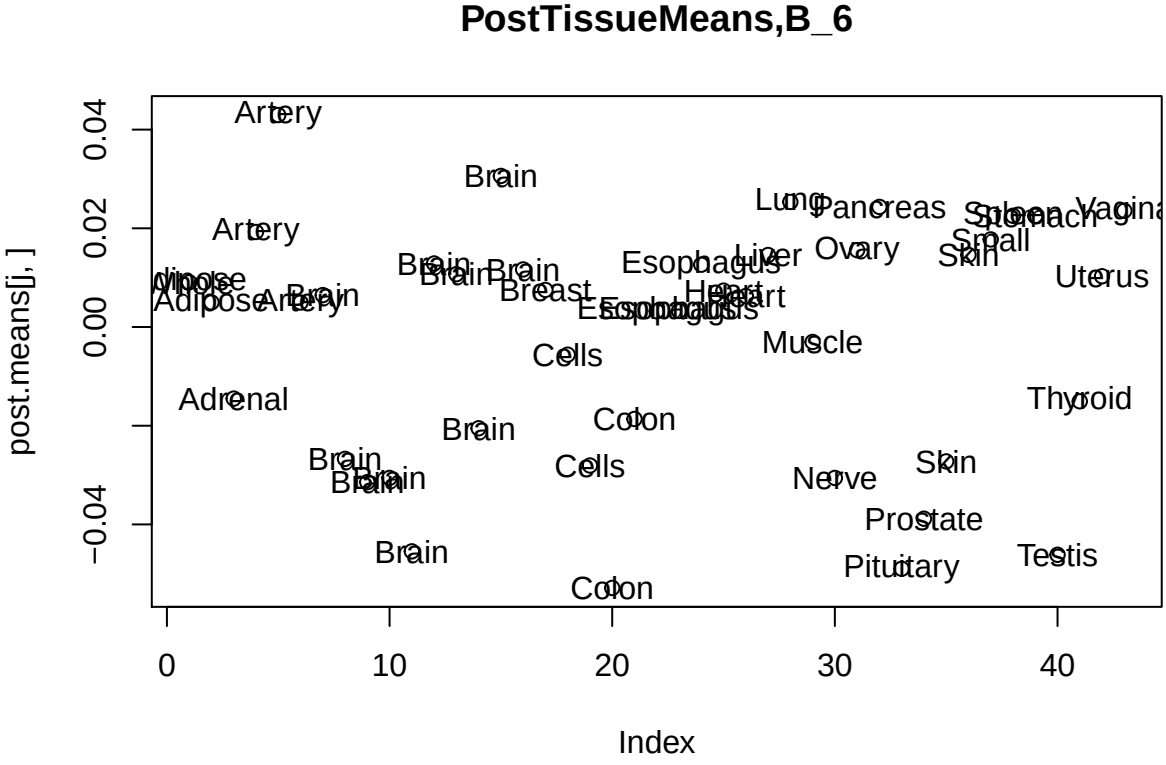

PostTissueMeans,B_1
post.means[j,]
0.02
0.00
−0.02
−0.04
0
10
20
30
40
Index

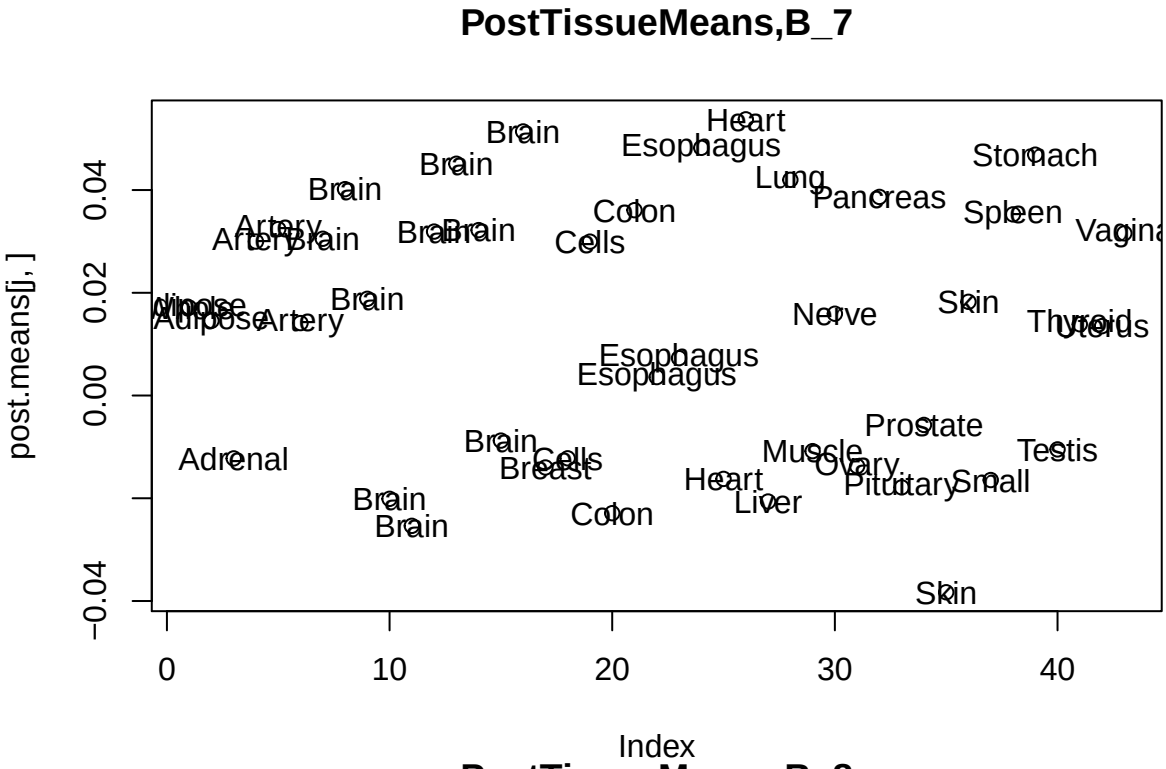

PostTissueMeans,B_2
post.means[j,]
0.04
0.02
0.00
−0.04
0
10
20
30
40
Index

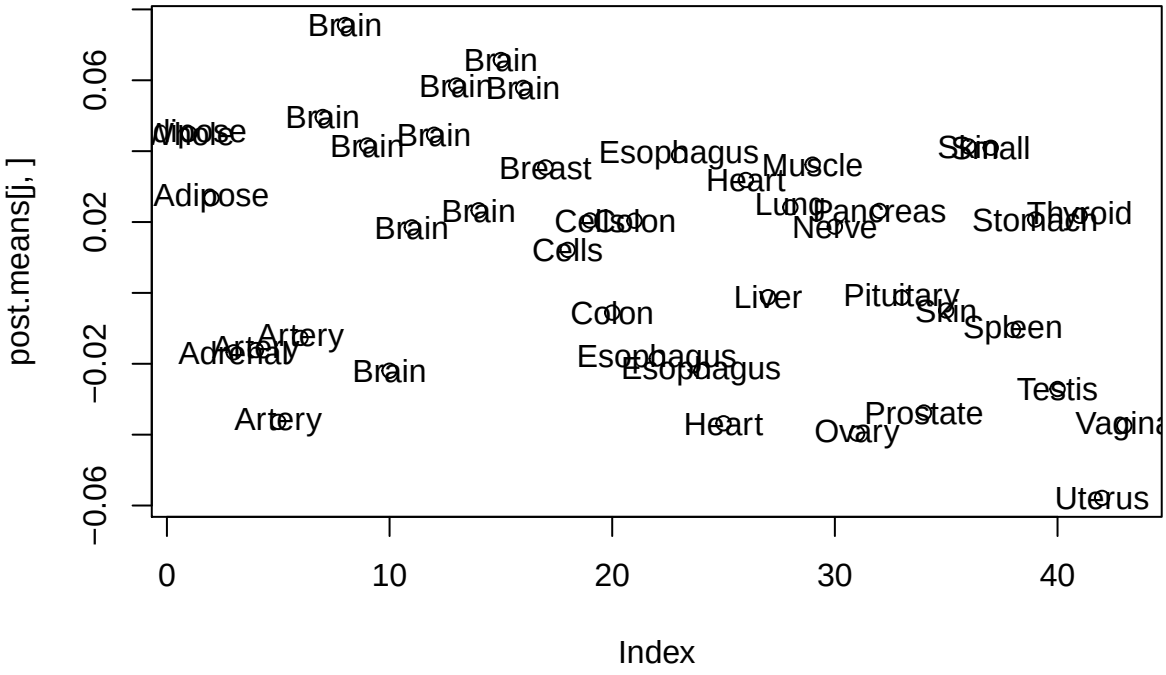

PostTissueMeans,B_3
post.means[j,]
0.04
0.00
−0.04
0
10
20
30
40
Index

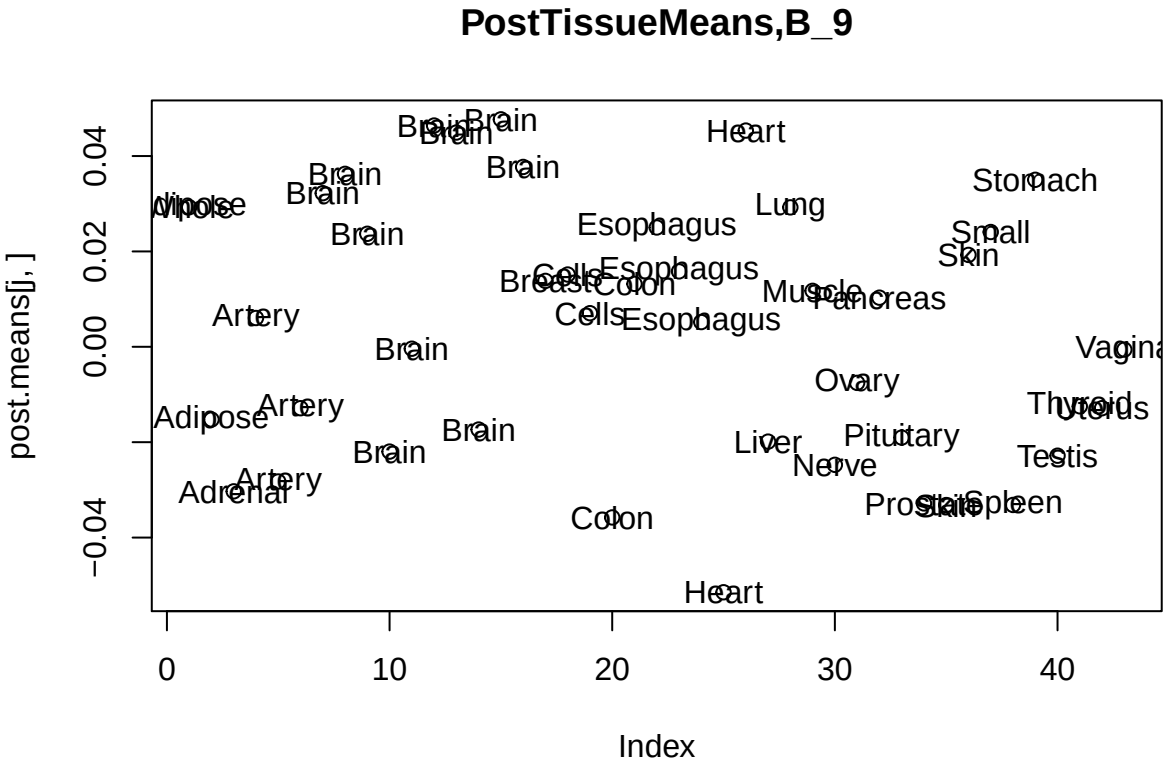

PostTissueMeans,B_4
post.means[j,]
0.04
0.02
0.00
−0.04
0
10
20
30
40
Index

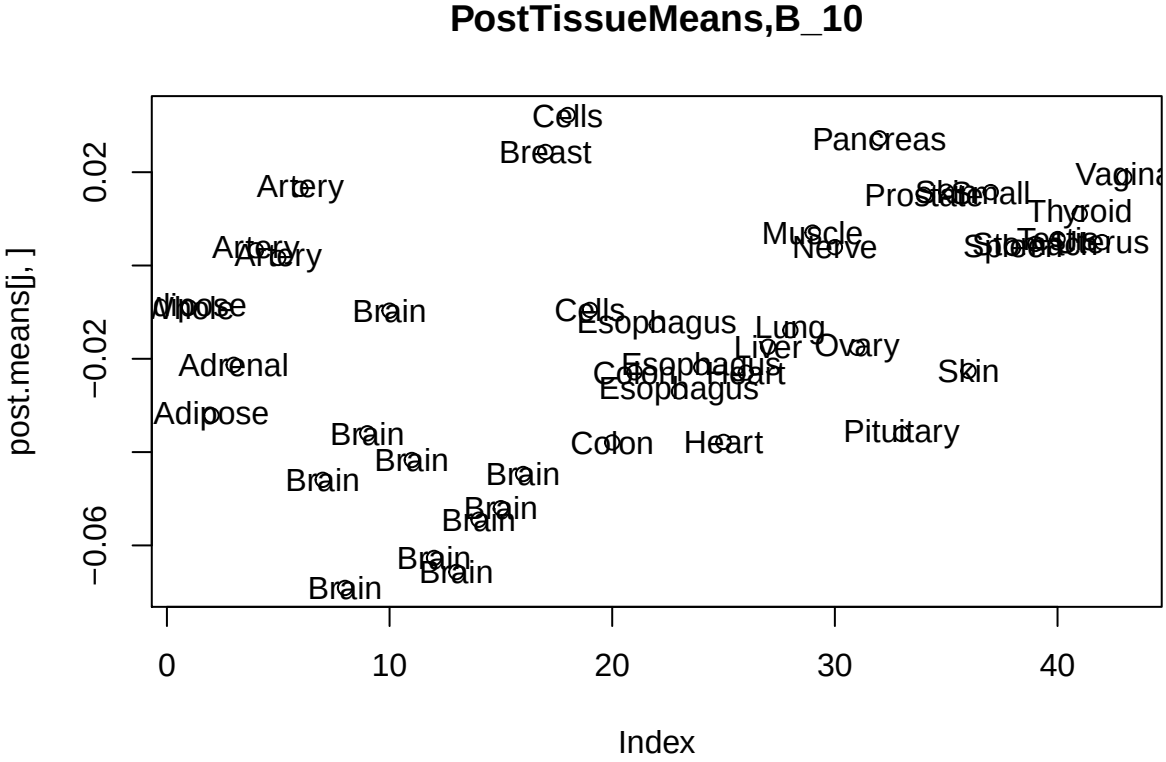

PostTissueMeans,B_5
post.means[j,]
0.04
0.02
0.00
−0.02
0
10
20
30
40
Index


PostTissueMeans,B_6
post.means[j,]
0.04
0.02
0.00
−0.04
0
10
20
30
40
Index


PostTissueMeans,B_7
post.means[j,]
Index
0.04
0.02
0.00
−0.04
0
10
20
30
40


PostTissueMeans,B_8
post.means[j,]
Index
0.06
0.02
−0.02
−0.06
0
10
20
30
40


PostTissueMeans,B_9
post.means[j,]
0.04
0.02
0.00
−0.04
0
10
20
30
40
Index


PostTissueMeans,B_10
post.means[j,]
0.02
−0.02
−0.06
0
10
20
30
40
Index