Exact Procedure: With Z stats

Sarah Urbut

July 6, 2015

First, I load the 16,069 x 44 matrix of maximum z statistics statistics into memory. I had previously computed the SFA decomposition on this matrix using 5 factors on the **PPS Cluster**. The steps that follow were run on the **Midway Cluster**.

```
z.stat <- read.table("maxz.txt")
lambda.mat=as.matrix(read.table("zsfa_lambda.out"))
factor.mat=as.matrix(read.table("zsfa_F.out"))
```

1 Deconvolution Step

I then perform deconvolution.em which initiatlizes the EM algorithm with the $X'X$, where X is the 16,069 x 44 matrix of z statistics from Matrix QTL. In brief, this is how the deconvolution.em algorithm works.

1. Produce a 2 element list of initialization parameters containing the initial covariance matrix (or matrices if you choose to fit more than one component) and a vector of their initial weights (again, here only 1: $\pi = 1$.)
2. Choose 'zstrong' as the top 10,000 gene-pairs and 'learn' the denoised covariance matrix from this choice.
3. Return a list with the denoised covariance matrix and corresponding mixture weights (here a vector of length 1 with $\pi = 1$)

```
max.step=deconvolution.em(t.stat = z.stat,factor.mat = factor.mat,lambda.mat = lambda.mat,K=1,P=2,per
```

2 Generation of List of Covariance Matirce

I then use this covariance matrix in place of our original choice of $X'X$ to create a KxL list of covariance matrices. The function **compute.hm.covmat** chooses a grid according to the range of effect sizes present in the initial 16,069 x 44 matrix of strong Z statistics. Here, I also used the Identity, rank 2 PC matrix, 5 factors, the rank 5 SFA approximation and the 44+1 eqtlbma.lite configurations. This is 54 matrices, and in this data set, **autoselect.mixgrid** chose a grid with 22 omegas for a total of 1188 covariance matrices.

```
covmat=compute.hm.covmat(z.stat,s.j,Q=5,lambda.mat,P=2,A="jul3",factor.mat,max.step=max.step)
```

covmat is thus a list of 1188 covariance matrices.

3 Mixture Weights

We now need to compute the mixture weights hierarchically. I use a randomly chosen set of 20000 gene snp pairs from the matrix QTL output to estimate these mixture proportions. This set does not contain the strongest gene-snp pairs, and thus will allow for substantial shrinkage, as a majority of these gene-snp pairs will have their likelihood maximized at low ω components.

```
compute.hm.train(train.b = train.z,se.train = train.s,covmat = covmat,A="jul3")
```

compute.hm.train produces an rds object with prior weights, a likelihood matrix, and a pdf of the barplot of these weights.

4 Posterior Quantities

Now that I have the estimated mixture weights stored in the vector **pis**. I proceed to the inference step, where I compute the posterior weights and corresponding posterior quantities across all original 16069 gene snp pairs. In brief, the posterior mean, post covariance matrix and tissue specific tail probabilities are computed across all K components for each gene snp pair, and then weighted according to the posterior weights. This is performed in the **weightedquants** step.

```
weightedquants=lapply(seq(1:nrow(z.stat)),function(j){total.quant.per.snp(j,covmat,b.gp.hat=z.stat,se
```

This will return a set of 6 files containing the JxR matrix of posterior means, marginal variances, tail probabilities, local false sign rates, and the JxK matrix of posterior weights. Checkpoint = FALSE means that the files will be created (rather than simply outputting an object array which contains the posterior quantities.

I have also simulated, for each gene snp pair, 100 draws from the multivariate normal distribution characterized by the posterior mean and covariance produced at a component chosen according to its posterior weight (responsibility). For each gene-snp pair, I count the number of simulations in which at least two signs differed.

```
sim.array.generation = function(j,b.j.hat,se.j.hat,covmat,pis,sim){
K=length(covmat)
b.mle=as.vector(t(b.j.hat[j,]))##turn i into a R x 1 vector
V.j.hat=diag(se.j.hat[j,]^2)
lik=lik.func(b.mle,V.j.hat,covmat)
post.weights=lik*pis/sum(lik*pis)
component=apply(rmultinom(100,1,prob = post.weights),2,function(x){which(x==1)})##choose a component
tinv=lapply(seq(1:sim),function(sim){k=component[sim];solve(covmat[[k]]+V.j.hat)})##generate a list c
b.j.=lapply(seq(1:sim),function(sim){ k=component[sim];post.b.jk.ed.mean(b.mle,tinv=tinv[[sim]],covma
B.j.=lapply(seq(1:sim),function(sim){ k=component[sim];post.b.jk.ed.cov(tinv=tinv[[sim]],covmat[[k]])
simulations=sapply(seq(1:sim),function(x){
  dat=mvrnorm(1,mu = b.j.[[x]],Sigma = B.j.[[x]])##for each simulation, generate a multivariate norma
   pos=sum(dat>0);neg=sum(dat<0);pos*neg!=0})##for each simulation, ask if they are all one direction

return(sum(simulations)/length(simulations)
)}
```