Manual for the Use of Matrix-Ash

Sarah Urbut

July 1, 2015

Contents

This document describes the proper usage and integrates the mathematical framework introduced by Stephens and Urbut *et al.*, 2015.

1 Posterior Effect Computation

By maximum likelihood in each tissue separately, we can easily obtain the estimates of the standardized genotype effect sizes, $\hat{\boldsymbol{b}}_j$, and their squared standard errors recorded on the diagonal of an $R \times R$ matrix noted $\hat{V}_j = \mathbb{V}(\hat{\boldsymbol{b}}_j)$. We assume that the matrix of standard errors of $\hat{\boldsymbol{b}}_j$, V_j as approximated by $\hat{V}_j$ is diagonal and that $\hat{V}_j$ is an accurate point estimate for the standard error and that these standard errors are independent between tissues.

If we now view $\hat{\boldsymbol{b}}_j$ and $\hat{V}_j$ as *observations* (i.e. known), we can write a new "likelihood" (using only the sufficient statistics) where we employ the use of bold typeset to indicate vector notation:

$$\hat{\boldsymbol{b}}_j | \boldsymbol{b}_j \sim \mathcal{N}_R(\boldsymbol{b}_j, \hat{V}_j) \tag{1}$$

For all j gene-snp pairs, $\boldsymbol{b}_j$ represents the unknown standardized effect of the the j_{th} gene-snp pair. We model the prior distribution from which $\boldsymbol{b}_j$ is drawn as a mixture of multivariate *Normals*.

$$\boldsymbol{b}_j|\boldsymbol{\pi}_{,0} \sim \sum_{k,l} \pi_{k,l}\, \mathcal{N}_R(\mathbf{0}, \omega_l U_k) \tag{2}$$

We allow $\pi_{k,l}$ to represent the (unknown) prior weight on prior covariance matrix U_k, which represents a direction k upon which the effects $\boldsymbol{b}_j$ in each tissue may lie, and 'stretch factor' $\omega_{1\cdots L}$. The novelty of our approach is in modeling $\boldsymbol{b}_j$ as a mixture of multivariate *Normals*, where each component of the mixture is defined by its data-sensitive estimate of the prior covariance matrix $U_{k,l}$. We allow the latent variable z_j to indicate which combination of covariance matrix and stretch factor we are considering, z_j can take on KxL values $z_j = [1,1]\cdots[k,l]$.

We know that for a single multivariate *Normal* the posterior on $\boldsymbol{b}_j|_0$ is simply:

$$\boldsymbol{b}_j|\hat{\boldsymbol{b}}_j \sim \mathcal{N}_R(\boldsymbol{\mu}_{j1}, U_{j1})$$

where:

- $\boldsymbol{\mu}_{j1} = U_{j1}(\hat{V}_j^{-1}\hat{\boldsymbol{b}}_j)$;
- $U_{j1} = (U_0^{-1} + \hat{V}_j^{-1})^{-1}$.

This leads us to a corresponding posterior that is also a mixture of multivariate *Normals* as this prior is conjugate to likelihood.

$$\begin{aligned} p(\boldsymbol{b}_j|\hat{\boldsymbol{b}}_j, \hat{V}_j, \hat{\boldsymbol{\pi}}) &= \sum_{k=1,l=1}^{K,L} p(\boldsymbol{b}_j|\hat{\boldsymbol{b}}_j, \hat{V}_j, k, l)p(z_j = k, l|\hat{\boldsymbol{b}}_j, \hat{V}_j, \hat{\boldsymbol{\pi}}), \\ &= \sum_{k=1,l=1}^{K,L} p(\boldsymbol{b}_j|\hat{\boldsymbol{b}}_j, \hat{V}_j, z_j = k, l)\tilde{\pi}_{k,l} \end{aligned} \tag{3}$$

Here, the posterior weight $\tilde{\pi}_{k,l}$ is simply

$$\tilde{\pi}_{k,l} = \frac{p(\hat{\boldsymbol{b}}_j|\hat{V}_j, z_j = k, l)\hat{\pi}_{kl}}{\sum_{k=1,l=1}^{K,L} p(\hat{\boldsymbol{b}}_j|\hat{V}_j, z_j = k, l)\hat{\pi}_{kl}} \tag{4}$$

Note also that $\hat{\pi}_{kl}$ represents the prior weights which are estimated hierarchically, using an EM algorithm which assumes that all $\boldsymbol{b}_j$ arise from a shared multivariate-*Normal* distribution. The proportional representation of each component of the prior on $\boldsymbol{b}_j$ is 'learned' from the data by maximizing the likelihood across all gene SNP pairs.

2 Intuition

So why does the likelihood increase at the 'right component'?

Consider the univariate single tissue case case case. Here, think about x as $\hat{b_1}$ as the Maximum Likelihood Estimate of the effect size in tissue 1, and σ as $U_{k[1,1]} + \hat{V}_{j[1,1]}$ as the marginal variance at a given component (integrating over the uncertainty in $\boldsymbol{b}$.

To compute the likelihood at each component:

$$f(x \mid \sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x)^2}{2\sigma^2}} \tag{5}$$

We can see that this will be largest when σ^2 approaches the MLE, which is simply $\hat{b_r}^2$. $\widehat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{1}(x)^2 = \frac{1}{1}\sum_{i=1}^{1} x^2$ This is the intuition behind the bayes factor being the largest at the 'true component'.

Furthermore, now we know that the posterior mean is $\boldsymbol{\mu}_1 = U_1(\hat{V}^{-1}\hat{\boldsymbol{b}})$ where $U_1 = (U_0^{-1} + \hat{V}^{-1})^{-1}$ and so quite obviously, the posterior mean for a particular tissue in the components with a large prior variance will also be large because they are 'very roughly' $\propto U_{0k}(\hat{V}^{-1}\hat{\boldsymbol{b}})$.

2.1 Intuition

Putting these things together, we see that a large likelihood in a particular component will mean that the majority of the posterior weight is at this component, and correspondingly, the majority of the posterior mean will be comprised of the posterior mean at this component which will specify a large effect at this component. If a SNP shows a relatively 'flat likelihood' at all components, than the posterior weights will appear similar to the prior weights, and correspondingly, the posterior mean will look a lot like the null case (since the prior weights are computed from 'mostly null data' and thus the prior weights will heavily weight the components with small posterior means (as determined by small prior variance in U_k).

3 Focus on a single gene-SNP pair

The critical innovation of our method is the use of U_k matrix which aim to capture the wide array of patterns of sharing in the data. Each U_k represent the covariance matrix of the underlying distribution from which each vector of standardized effect sizes is thought o have been drawn. To select these U_k, we want to choose a set that best represents the 'strongest' directions in the data set, and allow our model to 'learn' the relative proportions. In brief, we choose a set of covariance matrices that resembles the following:

4 Choice of Covariance Matrices

Choice of Covariance Matrices For a given ω_l, we specify 5 'types' of RxR prior covariance matrices $U_{k,l}$.

- $U_{k=1,l} = \omega_l\ \mathbf{I}_R$
- $U_{k=2,l} = \omega_l\ \frac{1}{J}\ X_t^t\ X_t$ The (naively) estimated tissue covariance matrix as estimated from the column-centered J $\times R$ matrix of t statistics,
- $U_{k=3,l} = \omega_l\ \frac{1}{J}\ \mathrm{V}_{1\ldots p}\ d^2_{1\ldots p}\ V^t_{1..p}$
- $U_{k=4:4+Q-1,l} = \frac{1}{J}[(\Lambda\mathbf{F})^t\Lambda\mathbf{F}]_q$
- $U_{k=4+Q,l} = \frac{1}{J}\ (\Lambda\mathbf{F})^t\Lambda\mathbf{F}$
- Λ represent the J x Q matrix of loadings
- F is then the KxR matrix of factors.

- Sparse Prior on Λ such that each SNP can be a member of a minimal number of factor classes

The function **compute.covmat** performs this using a JxR matrix of the 'strongest T statistics' across all gene-snp pairs. The intuition here is that we want to capture the strong patterns and allow our model to learn the relative proportions, so that when trained on a relatively 'null' training set, the 'shrinkage' will occur by putting emphasis on low while allowing the strongest effects to find their true match because the large likelihood will overwhelm the prior.

In order to run the model, one first executes

```
covmat=compute.covmat(b.j.hat,se.j.hat,Q=5,t.stat,lambda.mat,P=2,A="simulation",factor.mat=factors,bm
```

where b.j.hat and se.j.hat represent the JxR matrix of strong $\hat{\beta}$ and their standard errors, t.stat represents the MxR matrix of strong t statistics, and lambda, factor.mat, Q represent the matrices of loadings and factors and the corresponding number of single rank approximations to use. P represents the rank approximation using singular value decomposition of the PC matrix. If the bma option is true, then R singleton and 1 'full' (i.e., active with the same prior variance on the effect size in all tissues) is used.

```
omega=mult.tissue.grid(mult=sqrt(2),b.j.hat,se.j.hat)
```

We allow for the input of b.j.hat and se.j.hat because there may be situations in which the Exchangeable Effects (and not the Exchangeable T) shows a larger likelihood, and so we would want the scaling by ω to reflect this grid choice. The *covmat* will then be a list of length K, where K corresponds to the total number of components (grid weights times directions). In the event that you are interesting in only scaling these models to reflect the ET model, you can simply replace b.j.hat and se.j.hat with t.stat and v.j

5 Denoising Our Covariance Matrices

The observed data likelihood function can be optimized in several ways, one of which is to use a generic optimizer to increase the likelihood until it reaches a maximum. This approach is complicated by parameter constraints (e.g., the amplitudes π_k must all be nonnegative and add up to one, the variance matrices must be positive definite and symmetric) and multimodality of the likelihood surface. In what follows we will describe a different approach that is natural in this setting: An EM algorithm that iteratively and monotonically maximizes the likelihood, while naturally respecting the restrictions on the parameters. Because these t-statistics need to be 'denoised' (they represent the statistics of the 'observed effects' and not the real effects') we can use an EM algorithm to obtain a set of denoised full rank approximations from which to generate further covariance matrices.

This option is achieved with the use of **deconvolutionEM.R** which follows directly from Bovy *et al.*. Briefly, we fit a K=3 component mixture model to obtain the RxR 'true covariance matrix' that is initialized with the empirical covariance matrix $X'X$, the rank P SVD approximation and the rank Q SFA approximation. This returns a set of true covariance matrices in a $KxRxR$ arrays and an K vector of their corresponding mixture weights. We can then create a list of covariance matrices using these 3 full rank matrices as matrices K=2, K=3, and K=4+Q - each of these matrices will be scaled and the grid weighting procedure will then proceed as defined in the step above. For derivation, please see the document "EMUpdates" in the matrix$_a sh/DocumentsDirectory$.

Because this can take some time on a large data set, you might initially choose to denoise only the empirical covariance matrix, thus fitting a K=1 component mixture model initialized with the empirical covariance matric. The function **deconvolution.em.R** updates each of the prior covariance matrices and corresponding mixture weights by computing the posterior mean and covariance quantities for each

gene-pair at each of the K components. This is performed in the E step and contained in the function **emarray.generator.R**, generating a JxKxRxR array of posterior covariances and a JxKxR array of posterior means at each iteration.

```
for(j in 1:J){
    b.mle=as.vector(t(b.j.hat[j,]))##turn into a R x 1 vector
    V.j.hat=diag(se.j.hat[j,]^2)
    lik=lik.func.em(true.covs,b.mle,V.j.hat,K)

 tinv=lapply(seq(1:K),function(k){ ##generate a list of K inverted marginal covariance matrices
solve(true.covs[k,,]+V.j.hat)}

 ##create a K dimensional list of covariance matrices

B.j.=lapply(seq(1:K),function(k){
       post.b.jk.ed.cov(tinv=tinv,true.covs[k,,])
       })
post.covs[j,,,] <- tarray(array(unlist(B.j.), c( R, R,K))) ###store a K dimensional list of posterior

##compute a K dimensional list of posterior means for each J
   b.j.=(lapply(seq(1:K),function(k){
      tinv=solve(true.covs[k,,]+V.j.hat)##covariance matrix of the marginal distribution
      b=post.b.jk.ed.mean(b.mle,tinv=tinv,true.covs[k,,])##for each component, compute posterior mean
    }
    ))
post.means[j,,]=matrix(unlist(b.j.),ncol=R,byrow=TRUE) ##store as KxR matrix for each individual
#compute a k dimensional normalized likelihood for each individual
    q.mat[j,]=pi*lik/sum(pi*lik)
}
```

The M-step then proceeds by averaging across gene-snp pairs for each component, to return a list of K 'true covariance matrices' and K mixture weights, reflecting the relative proportion of each covariance matrix in the data set. However, with K =1, a good check on the method is that the π output at the end is in fact 1.

deconvolution.em(t.stat = t.stat,factor.mat = factor.mat,lambda.mat = lambda.mat,K = 1,P=2,permsnp

6 Training our Model

Now, we remove the 'strong data' and train our model on a relatively 'null' data set. This allows for shrinkage (see Stephens et al, *Ash*). Essentially while every gene-snp pair falls along a particular direction defined by the relationship in effect sizes between subgroups, some SNPs have much larger effects than others across subgroups. Because the majority of snp-gene associations in the genome are 'null' we expect this training set to reflect heavy loadings on low ω but reflect the patterns of the relative directions present in the data well. When we then compute the posterior on strong SNPa of interest, strong likelihoods will overwhelm low priors for large .

After defining our training data, we use the function **compute.hm.train(train.b = b.train,se.train = se.train,covma** to produce a matrix of likelihood based based on the set of covariance matrices contained in the K list

covmat.

```
J=nrow(train.b)
R=ncol(train.b)
lik.mat=t(sapply(seq(1:J),function(x){lik.func(b.mle,V.j.hat,covmat)}))
train=lik.mat
pis=mixEM(matrix_lik=train,prior=rep(1,ncol(train)))
barplot(t(as.matrix(pis$pihat)))
```

The matrix of likelihoods is then used to estimate the optimal combination of that maximize the probability of observing the data. This function thus returns a matrix of likelihoods, an .rds object of training weights and a corresponding barplot indicating the significance of each weight in the training data set. This is useful in visualizing the relative shrinkage of the method - a method which places large weight on low will shrink associations with relatively flat likelihoods.

7 Quantity Per SNP

After removing the training data set, we now choose a set of SNP upon which we want to make an inference.

If we want to compare between methods, we can use the **compute.lik.test** function on a matrix of test $\hat{\boldsymbol{\beta}}$ and their standard errors to compute the observed data likelihood using the hierarchical prior weights estimated from the training data. A method which captures the data well will not be overly biased to the choice of training data and thus maximize the probability of observing both the training (as reflected in the EM-chosen weights) and the test data set. If the weights are estimated on training data, they are then treated as fixed parameters. A method which chooses too many components will not fit the test data well (be overly biased, similar to nearest neighbor) and so this allows for easy comparison between methods.

In order to perform this step, simply separate training and testing data and load in the vector of computed on training data.

```
A="filename"
pis=readRDS(paste0("pis",A,".rds"))$pihat
compute.lik.test(b.test = b.test,J = 100,se.test = se.test,covmat,A="filename",pis)
```

This will output a matrix of likelihoods, a text file with the sum of the log-likelihoods of the test data set, and a matrix of posterior weights for the test data set (i.e., where **post.weight.func** computes the likelihood times the prior weight, normalized by the marginal likelihood across all components).

We also wish to produce a matrix of posterior quantities for all gene snp pairs in our test data set. Because we now have the hierarchical quantities (the vector of and the corresponding covariance matrices), we can compute an array of posterior quantities at each component for each gene snp pair and compute the corresponding posterior weights as the product of the prior weigh π_k at that component and the likelihood at that component. The function **total.quant.per.snp** thus neatly performs this.

For a given gene-snp pair j and the corresponding matrix of it's standard effect sizes and their standard errors, this function computes a list of KxR arrays containing the component specific posterior means, tissue-specific variances (obtained from the diagonal of the component specific posterior covariance matrix) and component specific marginal tail probabilities for each tissue. This occurs in the nested **post.array.per.snp** portion of the function.

The function then computes the posterior weights for that gene snp pair as the aforementioned produce of the prior weigh π_k at that component and the likelihood at that component. The total weighted quantity for each gene snp pair is simply a linear combination of each tissue specific quantity, where each is weighted by the posterior weight corresponding to that tissue specific quantity This weighting procedure is accomplished by each of the **total.X.per.snp** functions.

The function then returns 6 files, containing the R vector of posterior means (as weighted across all component), posterior upper and lower tail probabilities, the marginal tissue-specific weighted posterior variances and marginal tissue-specific local false sign rate. The function also returns the K vector of posterior weights for each gene snp pair. We can submit a job to loop through all J pairs under interrogation, and thus generate 5 JxR test files and 1 JxK posterior weights file.

```
weightedquants=lapply(seq(1:100),function(j){total.quant.per.snp(j,covmat,b.gp.hat=b.test,se.gp.hat =

  all.arrays=post.array.per.snp(j,covmat,b.gp.hat,se.gp.hat)
  post.means=all.arrays$post.means
  post.ups=all.arrays$post.ups
  post.downs=all.arrays$post.downs
  post.covs=all.arrays$post.covs
  post.nulls=all.arrays$post.nulls

  all.mus=total.mean.per.snp(post.weights,post.means)
  all.ups=total.up.per.snp(post.weights,post.ups)
  all.downs=total.down.per.snp(post.weights,post.downs )
  lfsr=t(lfsr.per.snp(all.ups,all.downs))
  all.covs.partone=total.covs.partone.persnp(post.means,post.covs,post.weights)
  marginal.var=all.covs.partone-all.mus^2
```

8 Putting it All together

First, we load the strong t statistics from which to derive the covariance matrices. Note that we remove the row-names for this portion and store the vector of names in 'genesnpnames'.

```
gene.snp.names=na.omit(read.table("~/Dropbox/AllGeneSNPStuff/max_tscore_eQTL.table.txt"))[,1]
t.stat=na.omit(read.table("~/Dropbox/AllGeneSNPStuff/max_tscore_eQTL.table.txt"))[,-c(1,2,47)]
v.j=matrix(rep(1,nrow(t.stat)*ncol(t.stat)),nrow=nrow(t.stat),ncol(t.stat))
lambda.mat=as.matrix(read.table("~/Dropbox/AllGeneSNPStuff//tri_gtex_allstrongt_lambda.out"))
factor.mat=as.matrix(read.table("~/Dropbox/AllGeneSNPStuff/tri_gtex_allstrongt_F.out"))
```

Now, we perform the matrix deconvolution in using the EM algorithm to estimate a 'denoised' empirical covariance matrix using the strongest 1000 gene snp pairs.

```
s=deconvolution.em(t.stat = t.stat,factor.mat = factor.mat,lambda.mat = lambda.mat,K = 1,P=2,permsnp
```

We now feed this object containing the 'denoised' empirical covariance matrix into our compute covariance matrices structure to produce the full set of covariance matrices.

```
covmat=compute.hm.covmat(t.stat,v.j,Q,lambda.mat,P,A,factor.mat,max.step=s)
```

We now proceed as before, in estimating the weights on a set of largely null training data, here I use the largely null 'first batch' of summary statistics and compute the hierarchical mixture weights on this data.

You can see that this function computes a matrix of likelihoods as described above, and then finds the optimal combination of weights to maximize this likelihood.

```
start="~/Dropbox/cyclingstatistician/beta_gp_continuous/matched/firstbatch"
t.stat=na.omit(read.table(paste0(start,"t.stat.txt"),header=F,skip=1)[,-c(1,2)])
v.j=matrix(rep(1,nrow(t.stat)*ncol(t.stat)),nrow=nrow(t.stat),ncol(t.stat))
b.train=t.stat[1:1000,]
se.train=v.j[1:1000,]
compute.hm.train(train.b = b.train,se.train = se.train,covmat = covmat,A="1000trained")
```

If we want to compare with previous approaches, we can compute the test-set likelihood. Critically, note that our test set does NOT contain our training set.

```
b.test=t.stat[1001:nrow(t.stat),]
se.test=v.j[1001:nrow(v.j),]
A="1000trained"
pis=readRDS(paste0("pis",A,".rds"))$pihat
compute.lik.test(b.test = b.test,J = nrow(b.test),se.test = se.test,covmat,A,pis)
```

We now simply compute the posterior quantities.

```
A="1000trained"
pis=readRDS(paste0("pis",A,".rds"))$pihat
rownames(b.test)=gene.snp.names
weightedquants=lapply(seq(1:100),function(j){total.quant.per.snp(j,covmat,b.gp.hat=b.test,se.gp.hat =
```

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.