Week 5 Drosophila Work

Eric Weine

3/30/2022

Subsetting Data by LD Blocks

This week, all data that I'll be working with will be subset by kilobase pairs. The code to do this is shown below, with the only addition from last week that I am now sampling from each LD block as opposed to merely creating them.

```
sites_df <- data.frame(stringr::str_split_fixed(summary_table$site, ":", 2))
colnames(sites_df) <- c("chromosome", "site_id")
sites_df <- sites_df %>%
  dplyr::mutate(site_id = as.numeric(site_id))

# split into blocks of a certain length
split_into_LD_blocks <- function(df, block_length) {

  block_range <- seq(from = min(df$site_id), to = max(df$site_id), by = block_length)
  df %>%
    dplyr::mutate(block_id = plyr::laply(site_id, function(x) sum(x > block_range)))

}

# group by chromosome and then split into blocks
sites_df <- sites_df %>%
  dplyr::group_by(chromosome) %>%
  dplyr::group_modify(~ split_into_LD_blocks(.x, 1e4))

# Now, want to sample one SNP from each group
sites_sample_df <- sites_df %>%
  dplyr::ungroup() %>%
  dplyr::sample_frac() %>% #randomly shuffle df
  dplyr::distinct(chromosome, block_id, .keep_all = TRUE) %>%
  dplyr::select(chromosome, site_id)

# Reconstruct site names
selected_sites <- purrr::pmap_chr(
  list(sites_sample_df$chromosome, sites_sample_df$site_id),
  function(x, y) glue::glue("{x}:{y}")
)

summary_table_samp <- summary_table %>%
  dplyr::filter(site %in% selected_sites) %>%
  dplyr::select(c(site, pval_CTRL, pval_HS, coef_CTRL, coef_HS))
```

```r
# replace 0 p-values with small numbers
summary_table_samp <- summary_table_samp %>%
  dplyr::mutate(
    pval_CTRL = pmax(.00000000001, pval_CTRL),
    pval_HS = pmax(.00000000001, pval_HS)
  )

# construct std error estimates from coefficients and p-values
summary_table_samp <- summary_table_samp %>%
  dplyr::mutate(
    std_error_ctrl = abs(coef_CTRL) / qnorm((2 - pval_CTRL) / 2),
    std_error_hs = abs(coef_HS) / qnorm((2 - pval_HS) / 2)
  )
```

In total, there are about 12K variants sampled in the reduced dataset.

Fitting MASH to subset dataset

Now, we fit MASH to the subset dataset, first without additional added penalty in the prior. The code is the same as last week, except that I realized that MASH automatically adds a null matrix for you, and so I have removed the "no_effect" matrix that I was feeding mash, which always had 0 posterior weight and was clearly redundant.

```r
reg_fx_mat <- t(matrix(
  data = c(summary_table_samp$coef_CTRL, summary_table_samp$coef_HS),
  nrow = 2,
  byrow = TRUE
))
colnames(reg_fx_mat) <- c("ctrl", "hs")

reg_se_mat <- t(matrix(
  data = c(summary_table_samp$std_error_ctrl, summary_table_samp$std_error_hs),
  nrow = 2,
  byrow = TRUE
))
colnames(reg_se_mat) <- c("ctrl", "hs")

mash_data <- mashr::mash_set_data(reg_fx_mat, reg_se_mat)

# Now, want to construct covariance matrices to feed into mash
cov_mat_list <- list()

cov_mat_list[['hs_spec']] <- matrix(
  data = c(0, 0, 0, 1), nrow = 2, byrow = TRUE, dimnames = list(
    rows = c("ctrl", "hs"), cols = c("ctrl", "hs")
  )
)

cov_mat_list[['ctrl_spec']] <- matrix(
  data = c(1, 0, 0, 0), nrow = 2, byrow = TRUE, dimnames = list(
    rows = c("ctrl", "hs"), cols = c("ctrl", "hs")
  )
)
```

```
desired_corrs <- seq(from = -1, to = 1, by = .25)
desired_amp <- c(3, 2, 1.5)

for(corr in desired_corrs) {

  cov_mat_list[[glue::glue('equal_corr_{corr}')]] <- make_amp_cov_mat(
    desired_corr = corr, amp = FALSE
  )

  for(cond in c("hs", "ctrl")) {

    for(amp in desired_amp) {

      cov_mat_list[[glue::glue('{cond}_amp_{amp}_corr_{corr}')]] <- make_amp_cov_mat(
        desired_corr = corr, amp_hs = (cond == "hs"), amp_coef = amp
      )

    }

  }

}

mash_out <- mashr::mash(
  data = mash_data,
  Ulist = cov_mat_list,
  algorithm.version = "Rcpp",
  outputlevel = 1
)

cov_mat_ests <- mashr::get_estimated_pi(mash_out)
```

The updated posterior estimates on the mixing parameters (above 1e-3) of the mash model are shown below:

```
##              null      equal_corr_1 hs_amp_1.5_corr_1
##        0.07150051        0.73275924        0.19531411
```

The estimate on the null covariance matrix has increased from about 1% to about 7%, so clearly subsetting the data is making some amount of a difference. However, the weight on the null matrix is still substantially lower than what we saw in GxSex.

Experimenting with null penalization in MASH

Another avenue of exploration in MASH is attempting to bias the fitting procedure to fit more null effects. The function `mash` has a `prior` parameter, that can be set to either `'nullbiased'` or `'uniform'` (the default). In every dataset I have examined, including simulated datasets used in the mash tutorial, I have only noticed negligible differences between the posterior weights on the mixture parameters when changing between these two settings. So, I wanted to better understand the "nullbiased" setting and see if it was possible to make the bias towards the null stronger than whatever was being implemented currently by MASH.

Based on the mash source code, it seems that mash calls the ash code in order to estimate the mixture components. This makes sense, because once the likelihoods are evaluated for the data, estimating the mixture components involves the same procedure for ash and mash. To encourage sparsity, ash uses a penalization term in the likelihood. This term is of the form

$$\prod_{k=0}^{K} \pi_k^{\lambda_k - 1},$$

where π_0 is the weight on the null matrix, $\pi_1, ...\pi_K$ is the weight on the K non-null mixture components, and λ_i is the penalty term on the i^{th} mixture component. In ash, λ_0 can be set via the `nullweight` parameter in the `ash` function (assuming one sets `prior = "nullbiased"`), and all other λ_i are set to 1. In `mash`, there is no `nullweight` parameter (as of the writing of this), and one simply has the ability to set `prior = "nullbiased"` which sets λ_0 to 10 (and keeps all other values λ_i at 1). When testing this feature, it seems that in mash, setting λ_0 to 10 is insufficient to induce sparsity in most cases. When going through the mash tutorial as well as testing that value on our data, the weight on π_0 essentially did not change.

I submitted a PR to add this parameter to mash, but in the meantime I was able to build mash locally and run it with a higher penalty on non-null matrices. I set λ_0 to 100, with the results shown below:

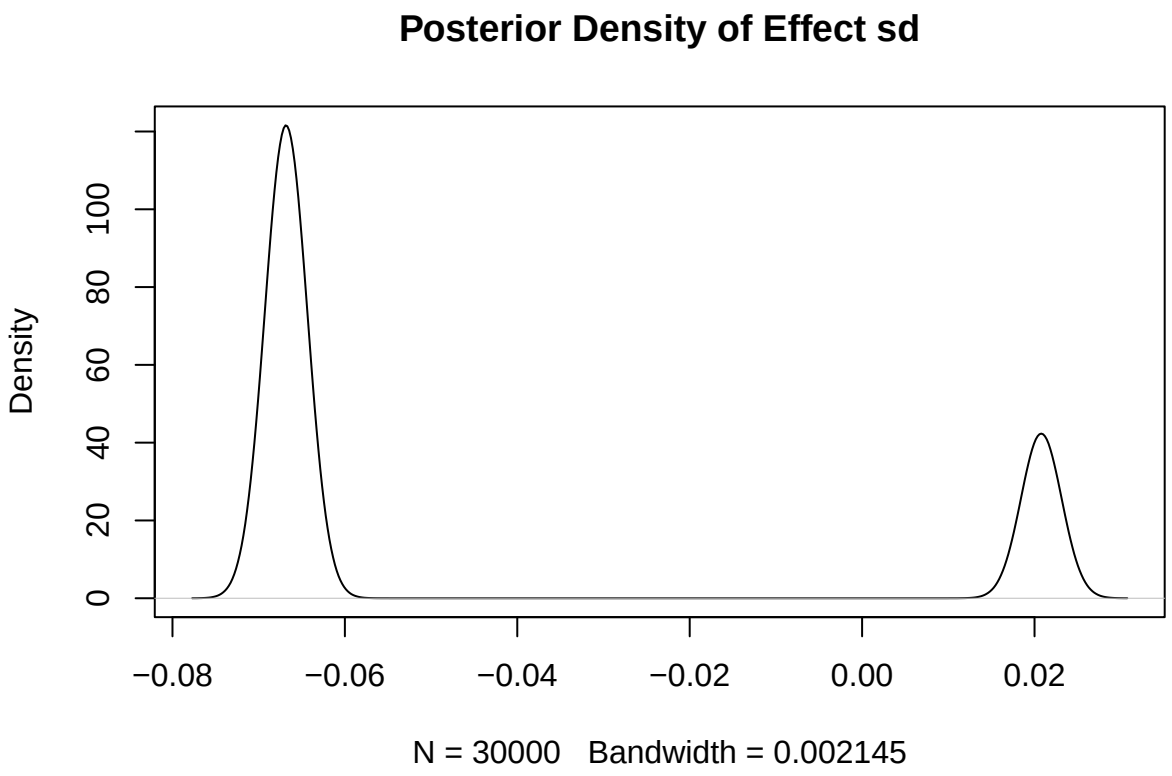
```
##              null      equal_corr_1 hs_amp_1.5_corr_1
##        0.22475462        0.69227736        0.08147361
```

Interestingly, it seems most of the weight that transferred to the null mixture component came from the amplification covariance matrix. This could be because many of the signals that were increasing the weight on the amplification matrix had very small signals for the high sugar group, and thus with a higher penalty were "classified" as null. In any event, the weight on the null matrix is still much lower than GxSex, even with a high penalty on non-null weights.

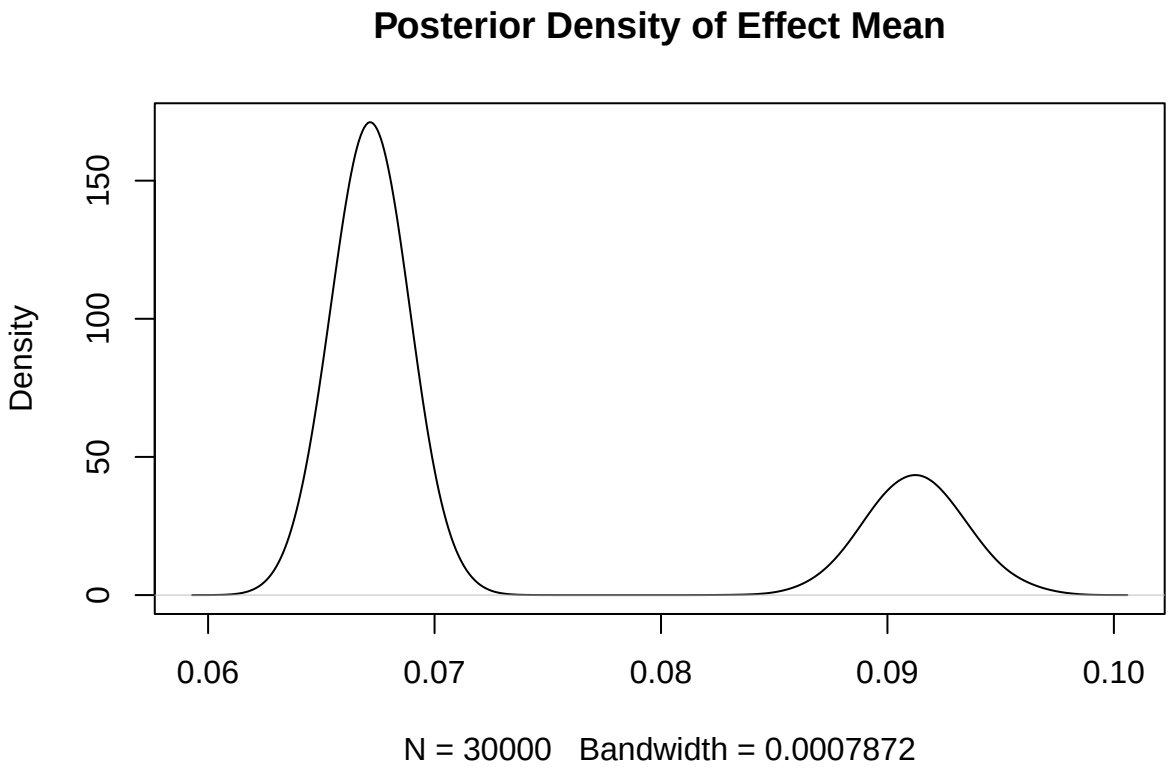
Troubleshooting STAN model

This week, I also wanted to continue to improve the stan model. Since the dataset is of size 12k, my local would run out of RAM when running the model, but I was able to spin up an AWS instance with much more RAM that could get the job done. The key insight this weak was abandoning the normality assumption for the distribution of effect sizes. Following Arbel's advice, I first attempted to fit two models. One where I set the mixture component π_0 to 0 (corresponding to amplification in every signal) and one where I set the mixture component π_0 to 1, corresponding to no amplification. I was still having some difficulty fitting these models, and I required quite a few samples until the chains mixed (on the order of 10K, which is always a bad sign). What I noticed was that the posterior distribution seemed to be bimodal in the mean and variance parameters, which was driving poor chain mixing. Below is the plot of the mean and variance parameters from setting π_0 to 1.

```
post_group_diff <- readr::read_rds("rds_data/stan_dros_post_group_diff.rds")
stan_ld_data <- readr::read_rds("rds_data/stan_ld_data.rds")

plot(density(post_group_diff$mu_theta), main = "Posterior Density of Effect sd")
```

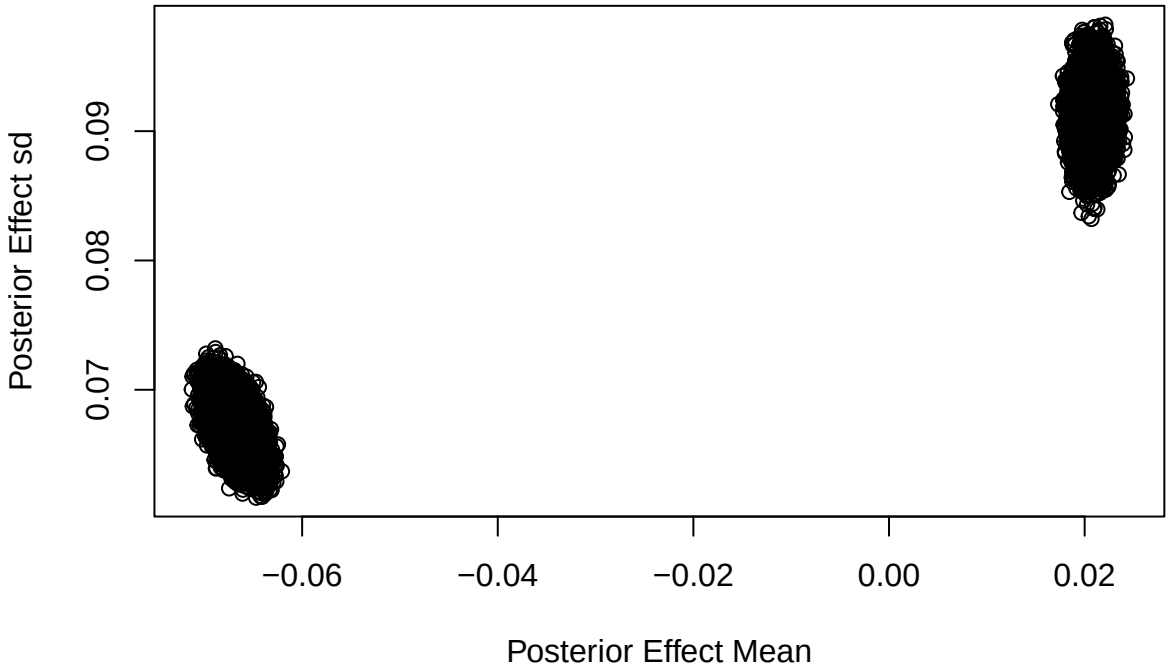



```
plot(density(post_group_diff$sigma_theta), main = "Posterior Density of Effect Mean")
```




```
plot(
  post_group_diff$mu_theta,
  post_group_diff$sigma_theta,
```

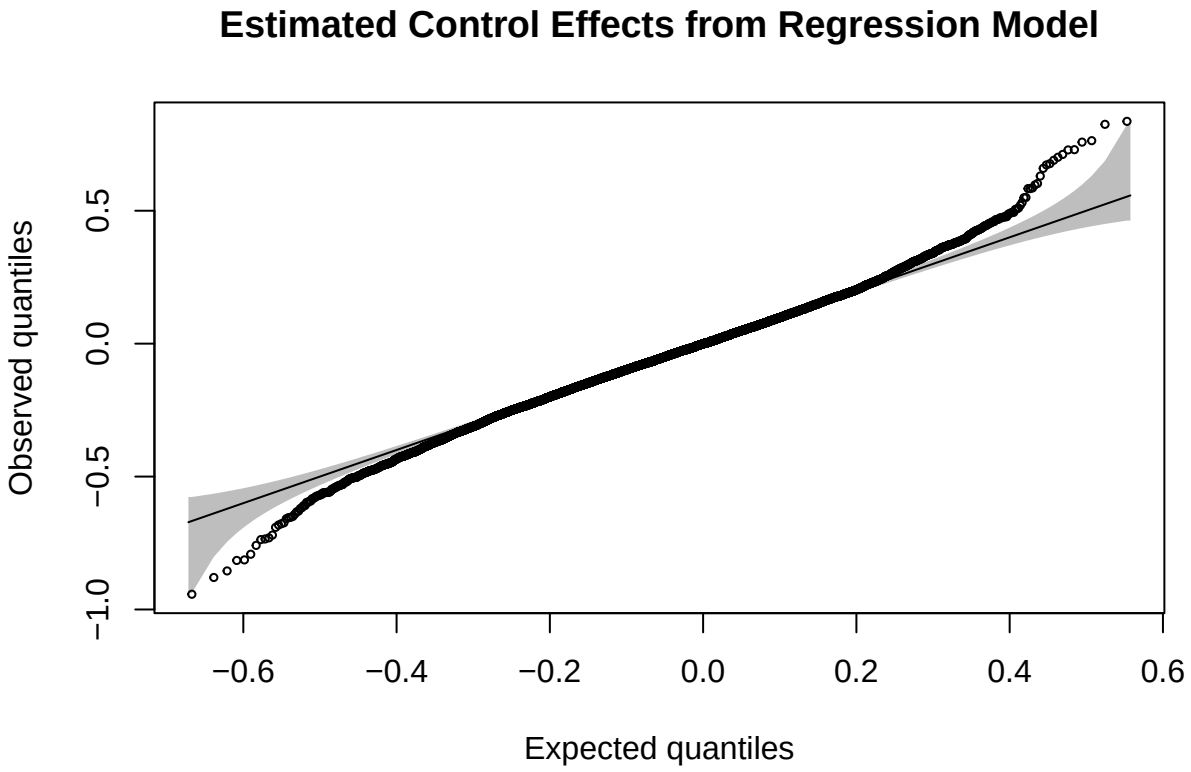
```
  xlab = "Posterior Effect Mean",
  ylab = "Posterior Effect sd"
  )
```



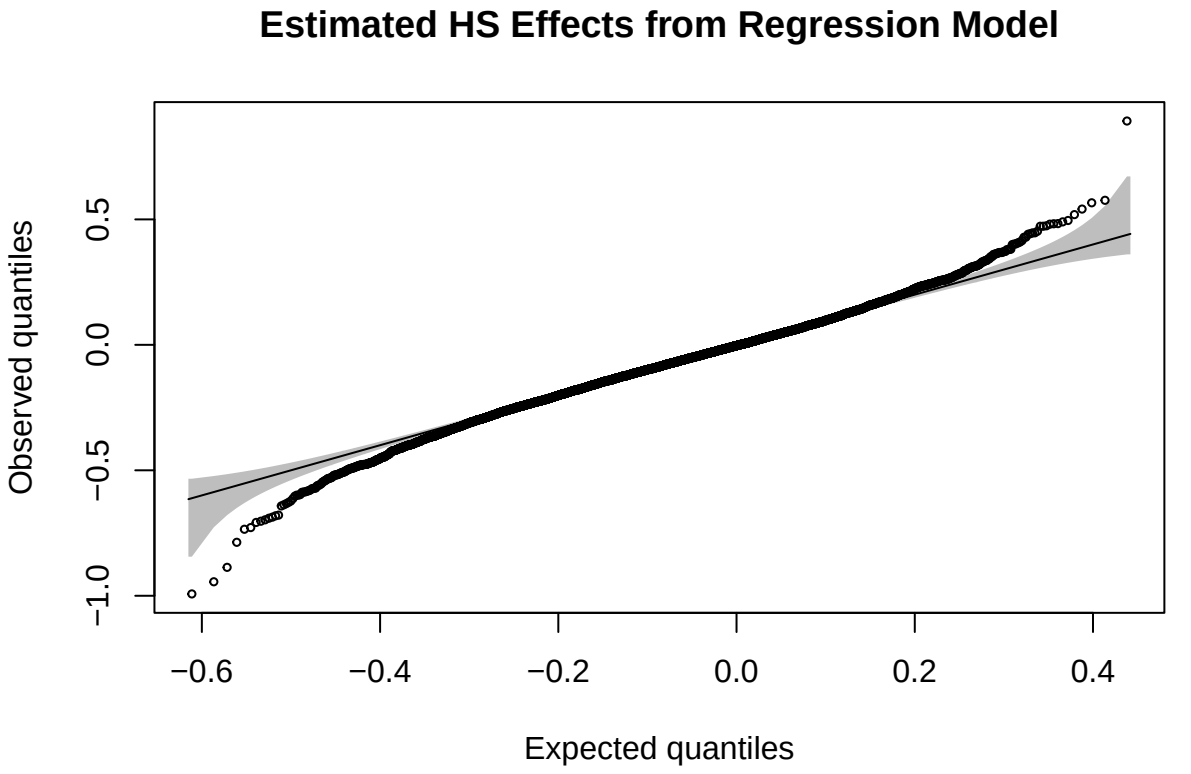
To me, this indicated that perhaps the variance of the data was too large for a normal distribution, as it seemed that some outlying observations were pulling the likelihood into two modes.

To confirm this, I plotted a qq-plot of the estimated beta-binomial regression effects for both groups against the normal distribution (with parameters estimated by the EXCELLENT package qqconf).

```
qqconf::qq_conf_plot(
  obs = stan_ld_data$c,
  points_params = list(cex = .5),
  main = "Estimated Control Effects from Regression Model"
)
```




```
qqconf::qq_conf_plot(
  obs = stan_ld_data$h,
  points_params = list(cex = .5),
  main = "Estimated HS Effects from Regression Model"
)
```



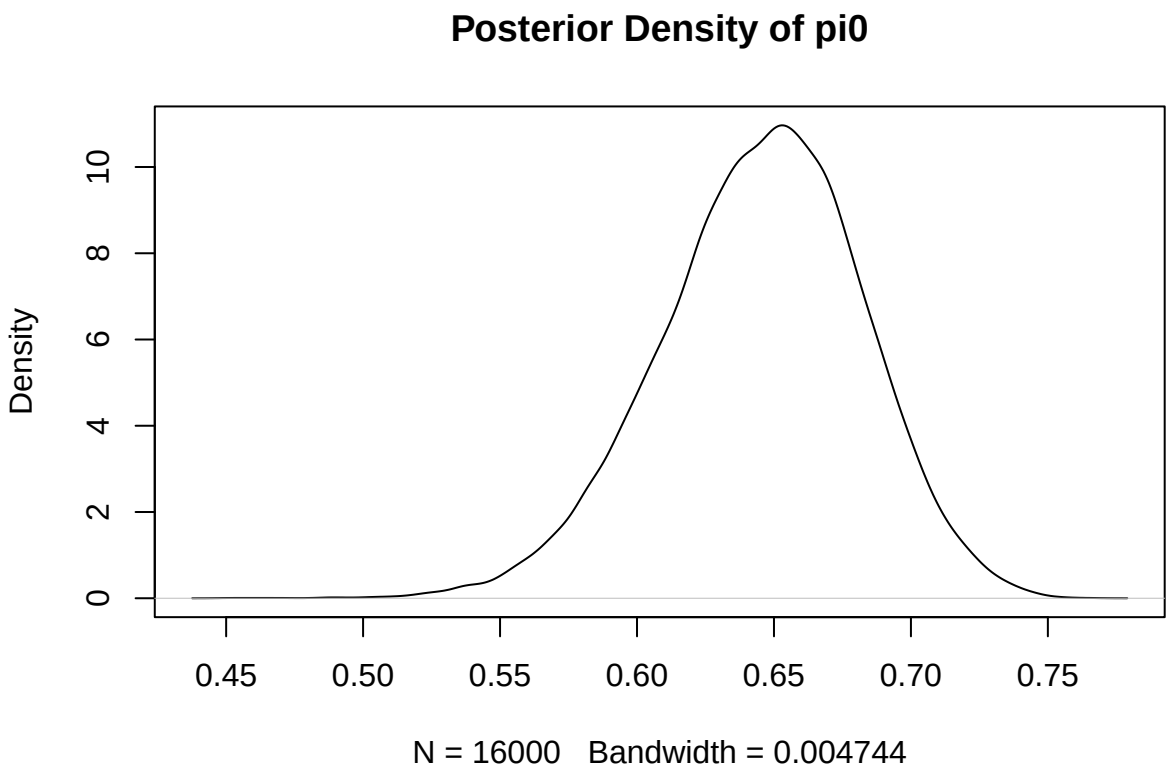
The deviation from non-normality is not extreme, but the tails do seem to be large enough such that it could make a difference. To remedy this, I decided to use a t-likelihood (with one degree of freedom). The mixture model with the t likelihood is shown below:

```
data {
 int < lower = 1 > N; // Sample size
 vector[N] h; // high sugar measured effects
 vector[N] c; // control measured effects
 vector<lower = 0>[N] s_h; // high sugar se
 vector<lower = 0>[N] s_c; // control se
}

parameters {

 real<lower = 0, upper = 1> pi_0; // Mixture model proportion
 real mu_theta; // mean of theta parameters
 real<lower = 0> sigma_theta; // sd of the theta parameters
 real alpha; // amplification coefficient

}

model {

 // uninformative prior on pi_0
 mu_theta ~ normal(0, sqrt(3));
 sigma_theta ~ normal(0, sqrt(3));
 alpha ~ normal(0, sqrt(3));

 c ~ normal(mu_theta, sqrt(square(s_c) + square(sigma_theta)));

 for(i in 1:N) {

   target += log_sum_exp(
   log(pi_0) +
    student_t_lpdf(h[i] | 1,
      mu_theta,
      sqrt(square(s_h[i]) + square(sigma_theta))),
   log(1 - pi_0) +
    student_t_lpdf(h[i] | 1,
      (1 + alpha) * mu_theta,
      sqrt(square(s_h[i]) + square(1 + alpha) * square(sigma_theta))));

 }

}
```

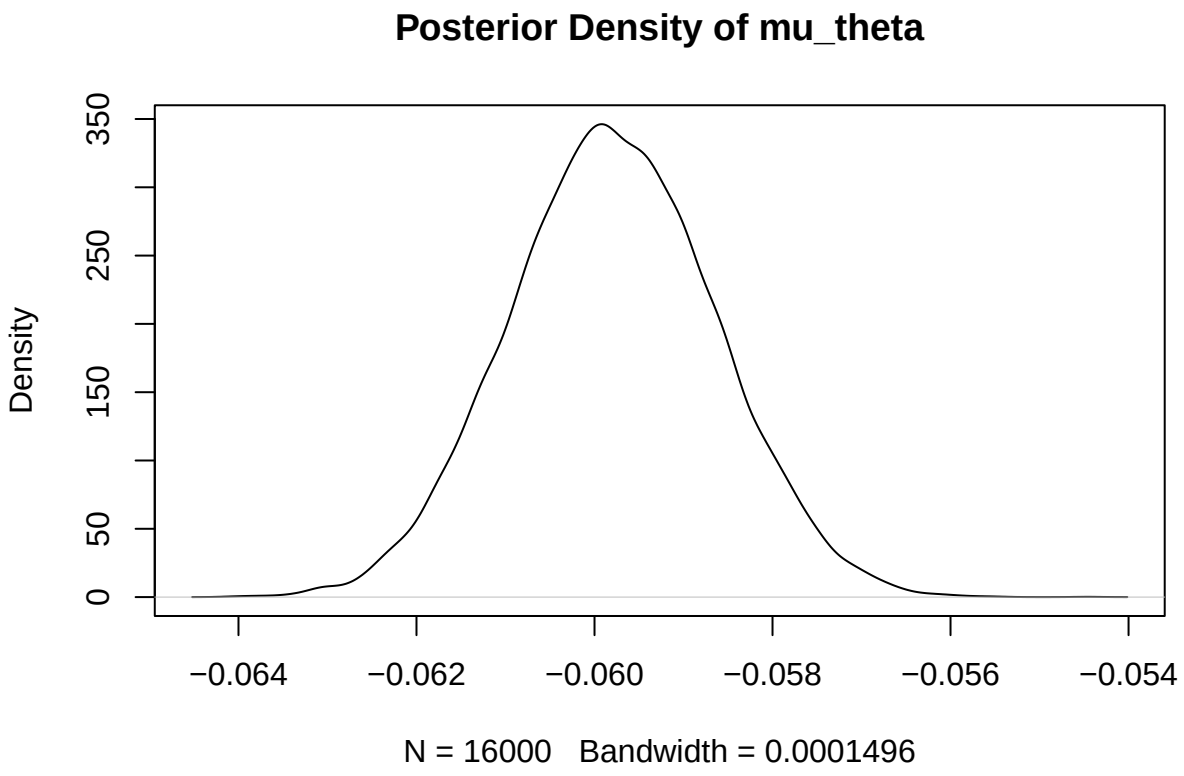
The inclusion of the t-likelihood seemed to make a very big difference, and convergence occurred quickly.

Below are the posterior densities of the parameters of interest.

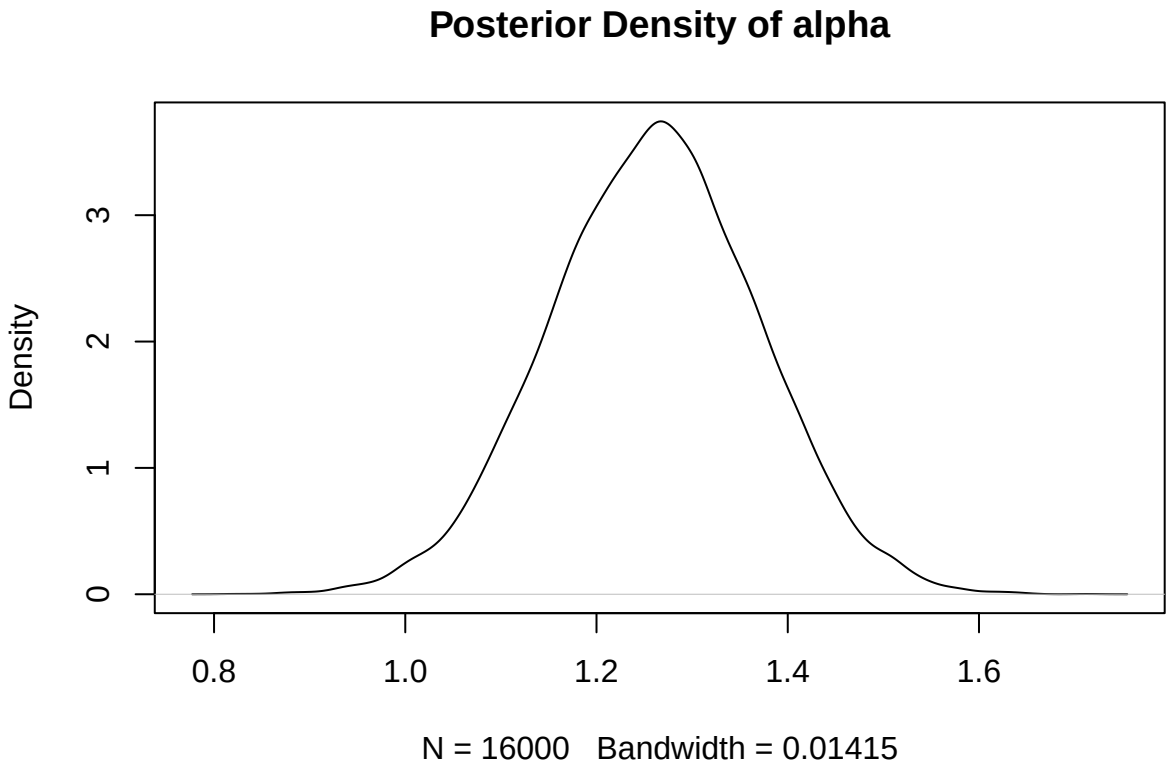
```
plot(density(amp_mixt_t_post$pi_0), main = "Posterior Density of pi0")
```




```
plot(density(amp_mixt_t_post$mu_theta), main = "Posterior Density of mu_theta")
```




```
plot(density(amp_mixt_t_post$alpha), main = "Posterior Density of alpha")
```



This model is much less flexible than the MASH model (most notably there's no null effect here). However, the two models seem to "agree" with each other, relatively speaking. The posterior mean of π_0 is about .65, which is pretty close to the estimates that we've gotten on matrix of equal effects with a correlation of 1. The posterior density of α is about 1.25 (which indicates that the effect of the control group is multiplied by 2.25 in the high sugar group). The MASH model seemed to suggest that the amplification was closer to 1.5, which seems more plausible based on the density plots of the regression coefficients I looked at in week 1.

Adding a null effect mixture component could be the next logical step in this model. However, before spending more money on AWS instances, I'd like to discuss where we want to go with this model so that I can reduce cost as much as possible.

MASH Bootstrap

I was mistaken about what I said last week. I believe that the current method is valid for creating bootstrap samples. I had misremembered the procedure for generating bootstrap samples, which I thought only generated sample sizes of some fraction of the whole dataset (e.g. $.66 \cdot n$), but in fact if the dataset is of size n then for each bootstrap sample you sample with replacement n datapoints. Obviously any datapoint has a correlation of 1 with itself, so sampling 1 SNP per block will have less than a correlation of 1 between samples, and thus is valid for this case. It is necessary that the blocks are sampled with replacement though, in order to follow the standard bootstrapping procedure.

Fitting MASH for GxExTrait

One other idea I was thinking about this week is extending MASH to the case in which the effect matrices estimated are 3D instead of 2D. In the GxSex project, we had SNP effects for a number of traits in each sex. In this case, the posterior effects were estimated completely separately. However, it seems possible ot imagine a matrix normal generalization of the mash procedure as follows.

Assume that we have n SNPs of interest, where we have estimated effect sizes and standard errors for c conditions and t traits. Let the true effect matrix for the i^{th} SNP be represented as the cxt matrix X_i. Then, we assume that

$$X_1, ..., Xn \sim MN_{cxt}(0, U, V)$$

where MN_{cxt} is the cxt matrix normal distribution and U is a cxc matrix representing the correlation between conditions and V is a txt matrix representing the correlation between traits. Now, to actually fit this model with MASH, we can take advantage of the fact that the matrix normal distribution reduces to the multivariate normal distribution in the following way. If

$$X \sim MN_{nxp}(M, U, V),$$

Then

$$vec(X) \sim N_{np}(vec(M), U \otimes V)$$

where the vec operator converts a matrix to a column vector by stacking the columns atop each other. Thus, to actually fit this model with mash, we would first define a set of potential covariance matrices for U and V

$$\begin{aligned} W &= \{U_1, ..., U_z\} \\ P &= \{V_1, ..., V_Q\} \\ B &= PxW \end{aligned}$$

then, we apply the kronecker product to each element of B, and we have our set of potential covariance matrices. Then, we simply convert our observations $\hat{X}_1, ..., \hat{X}_n$ to $vec(\hat{X}_1), ..., vec(\hat{X}_n)$ and proceed as usual.