Week 7 Work

Eric Weine

4/18/2022

Drosophila MASH

This week, I investigated the differences between fitting a mash model with a "tight" grid of possible amplifications (e.g. 1, 1.125, 1.25, ...) vs. a loose grid of amplifications (e.g. 1, 1.5, 2, 3). This question is very important from the perspective of interpreting the posterior weights on the mixture components, because I found that I get very different results depending on which model I use.

First, I constructed tight and loose grids of covariance matrices

```
#' Create 2x2 covariance matrices with optional levels of amplification
#'
#' The function assumes that we have two groups, with the standard deviation of
#' the effect size in one group being 1, and the standard deviation of the effect
#' size in the other group being \code{1 * amp_coef} if \code{amp} is set to
#' \code{TRUE}
#'
#' @param desired_corr Desired level of correlation. Must be in [-1, 1].
#' @param amp_coef Coefficient of amplification, as described above.
#' @param amp Boolean indicating if any amplificaiton should take place
#' @param amp_hs Boolean indicating if amplification should take place in the hs
#' group or in the c group. Only used if \code{amp} is set to \code{TRUE}.
#'
#' @return 2x2 covariance matrix
#' @export
#'
#' @examples
make_amp_cov_mat <- function(
  desired_corr, amp_coef = 1, amp = TRUE, amp_hs = TRUE
) {

  if (amp_hs && amp) {

    ctrl_sd <- 1
    hs_sd <- ctrl_sd * amp_coef

  } else if(!amp_hs && amp) {

    hs_sd <- 1
    ctrl_sd <- hs_sd * amp_coef

  } else {

    hs_sd <- 1
```

```
    ctrl_sd <- 1

  }

  # derive covariance from correlation and sds
  cov_hs_ctrl <- desired_corr * hs_sd * ctrl_sd

  cov_mat <- matrix(
    data = c(ctrl_sd ^ 2, cov_hs_ctrl, cov_hs_ctrl, hs_sd ^ 2),
    nrow = 2,
    byrow = TRUE,
    dimnames = list(
      rows = c("ctrl", "hs"), cols = c("ctrl", "hs")
    )
  )

  return(cov_mat)

}
```

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

cov_mat_list_loose[['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_loose[['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(1.5, 2, 3)

for(corr in desired_corrs) {

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

    }
```

```
  }

}
```

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

cov_mat_list_tight[['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_tight[['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(1.125, 1.25, 1.375, 1.5, 1.625, 1.75, 1.875, 2, 3)

for(corr in desired_corrs) {

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

    }

  }

}
```

Then, I sampled the dataset in the usual manner. I set aside a "test" dataset for later as well.

```
summary_table <- read.delim('data/SummaryTable_allsites_12Nov20.txt')

# replace 0 p-values with small numbers
summary_table <- summary_table %>%
  dplyr::select(c(site, pval_CTRL, pval_HS, coef_CTRL, coef_HS, sig_cat)) %>%
  dplyr::mutate(
    pval_CTRL = pmax(.00000000001, pval_CTRL),
    pval_HS = pmax(.00000000001, pval_HS)
  )

# construct std error estimates from coefficients and p-values
```

```r
summary_table <- summary_table %>%
  dplyr::mutate(
    std_error_ctrl = abs(coef_CTRL) / qnorm((2 - pval_CTRL) / 2),
    std_error_hs = abs(coef_HS) / qnorm((2 - pval_HS) / 2)
  )

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)

summary_table_signif <- summary_table %>%
  dplyr::filter(sig_cat != 'NS')

# generate a test df
sites_test_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_test <- purrr::pmap_chr(
```

```
  list(sites_test_df$chromosome, sites_test_df$site_id),
  function(x, y) glue::glue("{x}:{y}")
)

summary_table_test <- summary_table %>%
  dplyr::filter(site %in% selected_sites_test)

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

reg_fx_mat_test <- t(matrix(
  data = c(summary_table_test$coef_CTRL, summary_table_test$coef_HS),
  nrow = 2,
  byrow = TRUE
))
colnames(reg_fx_mat_test) <- c("ctrl", "hs")

reg_fx_mat_signif <- t(matrix(
  data = c(summary_table_signif$coef_CTRL, summary_table_signif$coef_HS),
  nrow = 2,
  byrow = TRUE
))
colnames(reg_fx_mat_signif) <- c("ctrl", "hs")

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

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

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

mash_samp_data <- mashr::mash_set_data(reg_fx_samp_mat, reg_se_samp_mat)
mash_test_data <- mashr::mash_set_data(reg_fx_mat_test, reg_se_mat_test)
mash_signif_data <- mashr::mash_set_data(reg_fx_mat_signif, reg_se_mat_signif)
```

Then, I fit the two models with the training dataset.

```
mash_loose_fit <- mashr::mash(
  data = mash_samp_data,
  Ulist = cov_mat_list_loose,
  outputlevel = 3
)

mash_loose_fit <- mashr::mash(
  data = mash_samp_data,
  Ulist = cov_mat_list_loose,
  outputlevel = 3
)
```

These two fits lead to quite different results. First, below are the posterior weights for the loose fit.

```
print(loose_fit$pi[loose_fit$pi > 1e-4])
```

```
##                 null      equal_corr_1.11 hs_amp_1.5_corr_1.11
##         0.0715005864         0.6013971894         0.1948948189
##      equal_corr_1.12         ctrl_spec.14  hs_amp_3_corr_-1.14
##         0.1313614707         0.0001529778         0.0002731683
## hs_amp_1.5_corr_1.15
##         0.0004197886
```

And for the tight fit:

```
print(tight_fit$pi[tight_fit$pi > 1e-4])
```

```
##                   null hs_amp_1.125_corr_1.11        equal_corr_1.12
##           0.0689234064           0.7809967021           0.1492406627
##           ctrl_spec.14    hs_amp_3_corr_-1.14   hs_amp_1.5_corr_1.15
##           0.0002085024           0.0002723314           0.0003583950
```

The tight fit classifies much more points into the amplification mixture component, but that mixture component has a much lower level of amplification.

Clearly, we need some way to be able to select between these two models. In Bayesian models there are typically two ways to do this. The first is to evaluate the log likelihood of the models on out of sample data. Here, we can use the test data we set aside before and evaluate the average log likelihood

```
print(mashr::mash_compute_loglik(
  g = tight_fit,
  data = mash_test_data
) / 12030)
```

```
## Warning: Please make sure the alpha in data is consistent with the `alpha` used to compute g.
```

```
## [1] 0.852874
```

```
print(mashr::mash_compute_loglik(
  g = loose_fit,
  data = mash_test_data
) / 12030)
```

```
## Warning: Please make sure the alpha in data is consistent with the `alpha` used to compute g.
```

```
## [1] 0.8522244
```

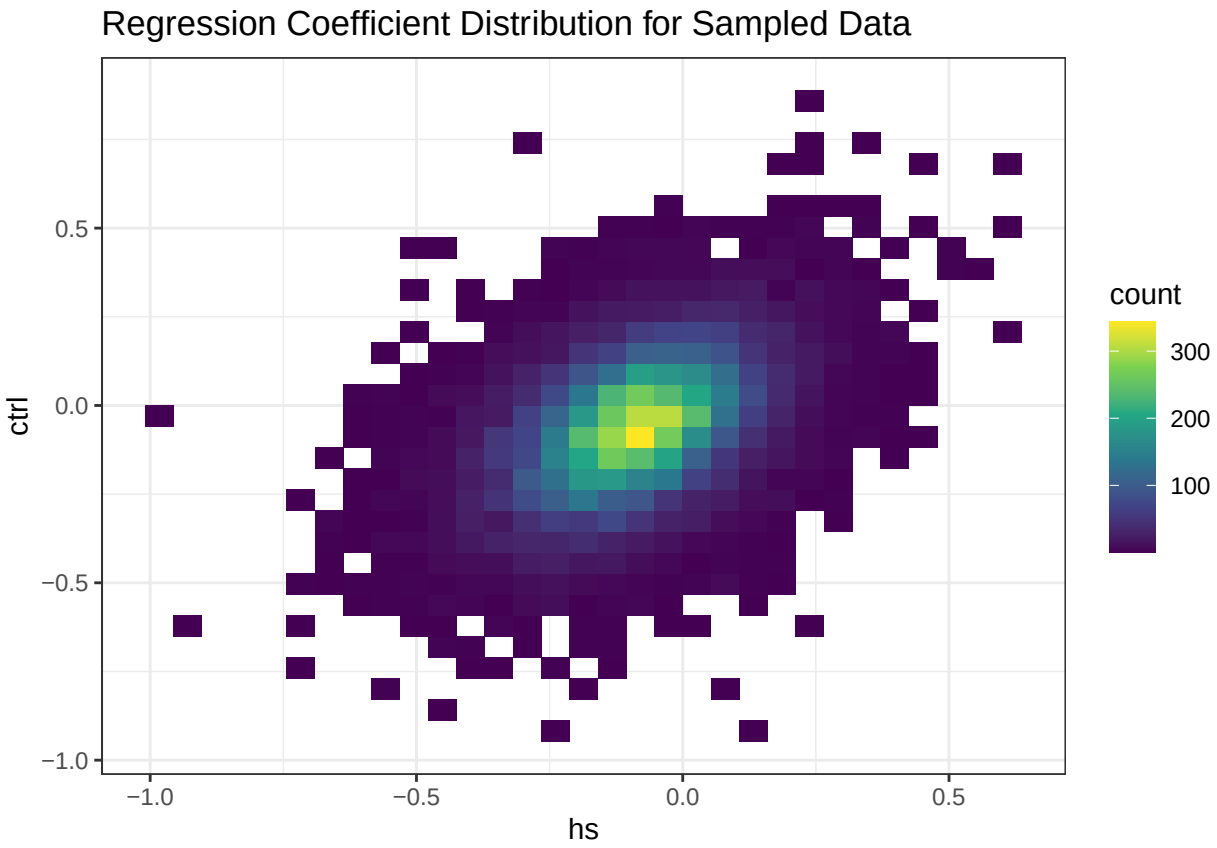
The average log-likelihoods are incredibly close, indicating that both models seem to provide essentially equivalent quality of fits to out of sample data.

Another potential avenue of model comparison is what are referred to as "posterior predictive checks." This involves simulating data from the posterior fitted model and comparing it to the actual data. If the simulated data looks substantially different from the fitted data, then one can conclude that the model is not a good fit for the data. We can do this with the mash models by simulating from the posterior of effects and then adding noise using the standard errors of the regression coefficients.

```
generate_posterior_predictive <- function(
  mash_samples,
  mash_se_mat,
  n_samples,
  n_conditions = 2
) {

  out_datasets <- list()
  cov_mat_list <- sample_cov_mats <- apply(
    mash_se_mat, MARGIN = 1, function(x) {diag(x ^ 2)}, simplify = FALSE
  )

  for (i in 1:n_samples) {

    sample_post_effect <- mash_samples[,,glue::glue("sample_{i}")]
    sample_noise <- plyr::laply(
      cov_mat_list,
      MASS::mvrnorm,
      mu = rep(0, n_conditions),
      n = 1
    )
    posterior_predictive_samp <- sample_post_effect + sample_noise
    out_datasets[[i]] <- data.frame(posterior_predictive_samp)

  }

  return(out_datasets)

}
```

The most obvious dimension on which to compare simulated datasets to the actual datasets is via the joint density of the control and high sugar effects. If we take our sampled data, we can see what this plot looks like.

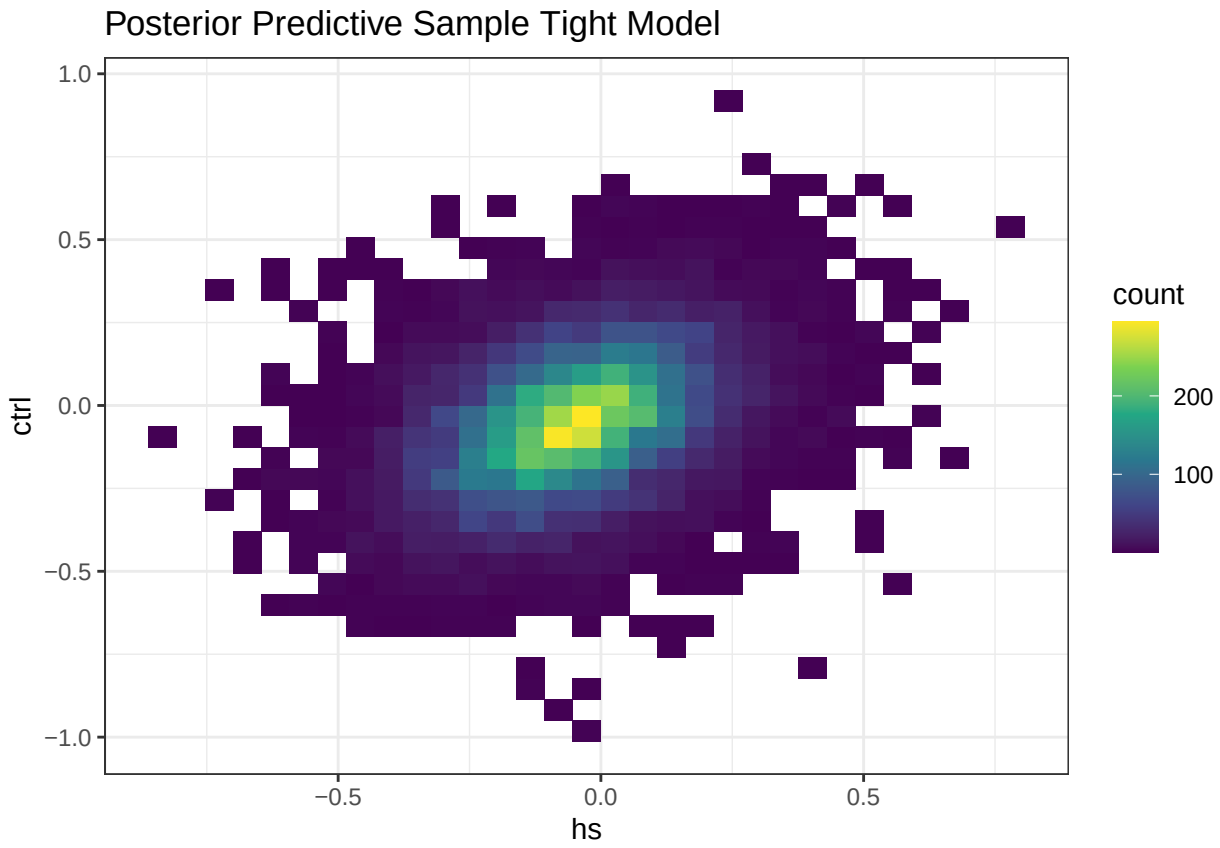
```
ggplot(data = data.frame(reg_fx_samp_mat), aes(x = hs, y = ctrl)) +
  geom_bin2d() +
  scale_fill_continuous(type = "viridis") +
  theme_bw() +
  ggtitle("Regression Coefficient Distribution for Sampled Data")
```



This plot looks like a multivariate normal distribution with a relatively strong correlation. If we take a sample from the posterior distribution of either model, it should look similar.

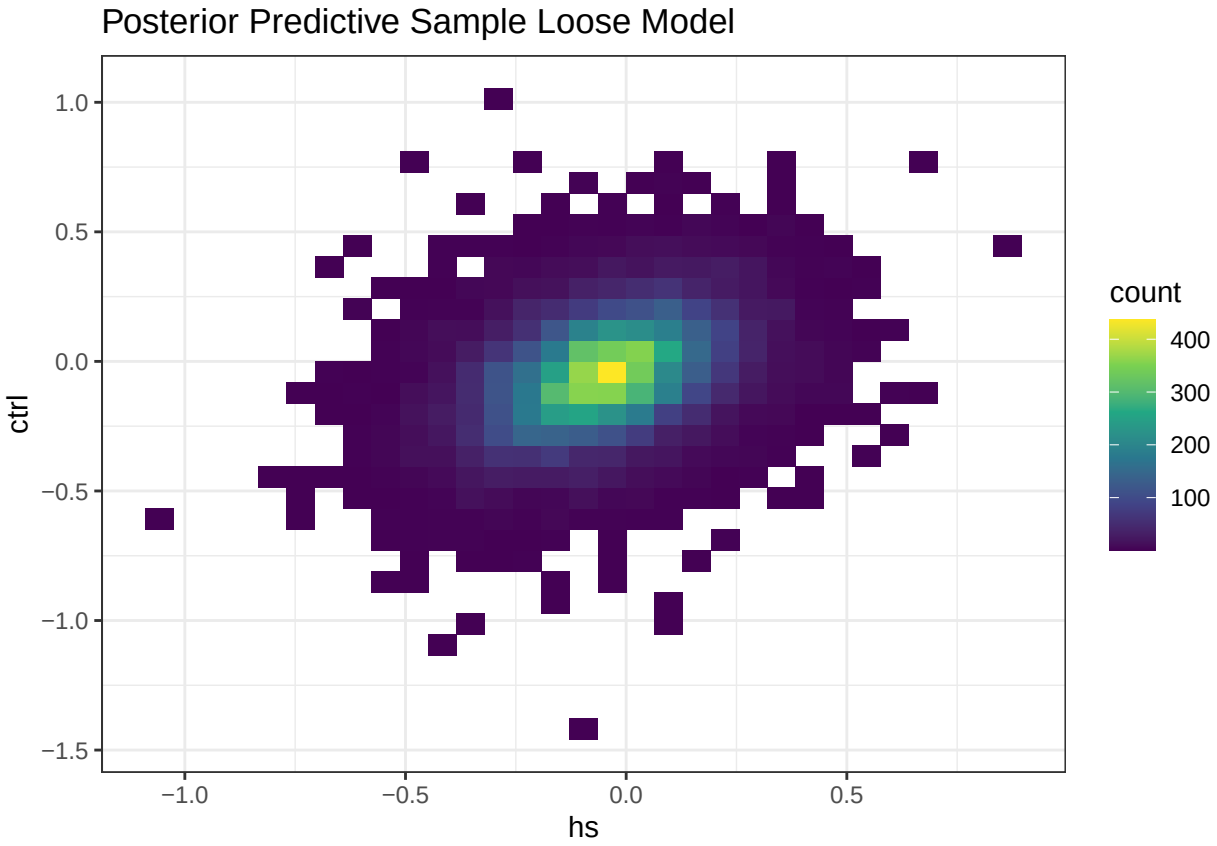
```
post_preds_tight <- generate_posterior_predictive(
  mash_samples = post_samp_tight,
  mash_se_mat = reg_se_samp_mat,
  n_samples = 5
)
```

```
ggplot(data = post_preds_tight[[3]], aes(x = hs, y = ctrl)) +
  geom_bin2d() +
  scale_fill_continuous(type = "viridis") +
  theme_bw() +
  ggtitle("Posterior Predictive Sample Tight Model")
```




```
post_preds_loose <- generate_posterior_predictive(
  mash_samples = post_samp_loose,
  mash_se_mat = reg_se_samp_mat,
  n_samples = 5
)
```

```
ggplot(data = post_preds_loose[[3]], aes(x = hs, y = ctrl)) +
  geom_bin2d() +
  scale_fill_continuous(type = "viridis") +
  theme_bw() +
  ggtitle("Posterior Predictive Sample Loose Model")
```



Visually, both models look relatively reasonable and quite similar. Comparing summary statistics gives additional information.

Mean and covariance of the sampled data:

```
cov(data.frame(reg_fx_samp_mat))
```

```
##            ctrl         hs
## ctrl 0.03028915 0.01124314
## hs   0.01124314 0.02204988
```

```
colMeans(data.frame(reg_fx_samp_mat))
```

```
##        ctrl          hs
## -0.06102910 -0.09081548
```

Mean and covariance of one posterior predictive sample for the loose fit:

```
cov(post_preds_loose[[3]])
```

```
##            ctrl         hs
## ctrl 0.03522145 0.01135230
## hs   0.01135230 0.02851517
```

```
colMeans(post_preds_loose[[3]])
```

```
##        ctrl          hs
## -0.04901843 -0.04931643
```

Mean and covariance of one posterior predictive sample for the tight fit:

```
cov(post_preds_tight[[3]])
```

```
##            ctrl         hs
## ctrl 0.03408092 0.01114231
## hs   0.01114231 0.02806400
```

```
colMeans(post_preds_tight[[3]])
```

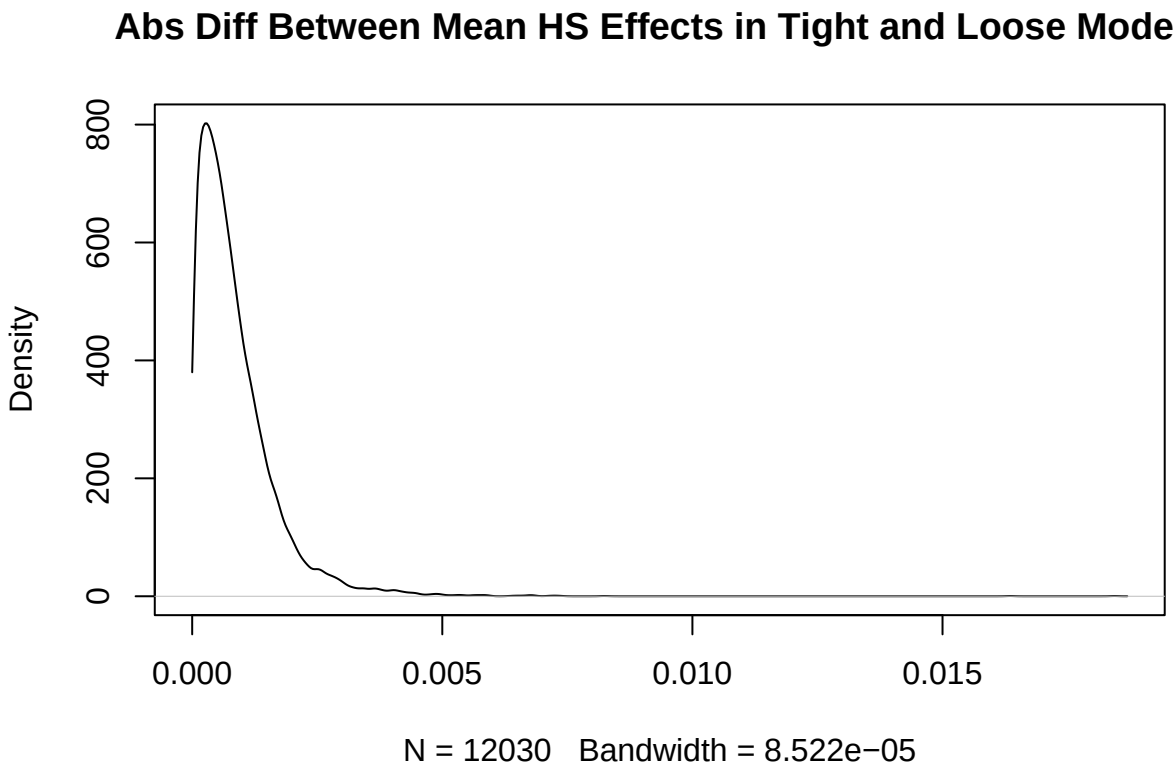
```
##        ctrl          hs
## -0.04849291 -0.05172822
```

Booth fitted models seem to underestimate the difference in means between the HS and CTRL effects. They also seem to overestimate the variance for the HS effect. With that said, between the fitted models there seems to be very little difference.

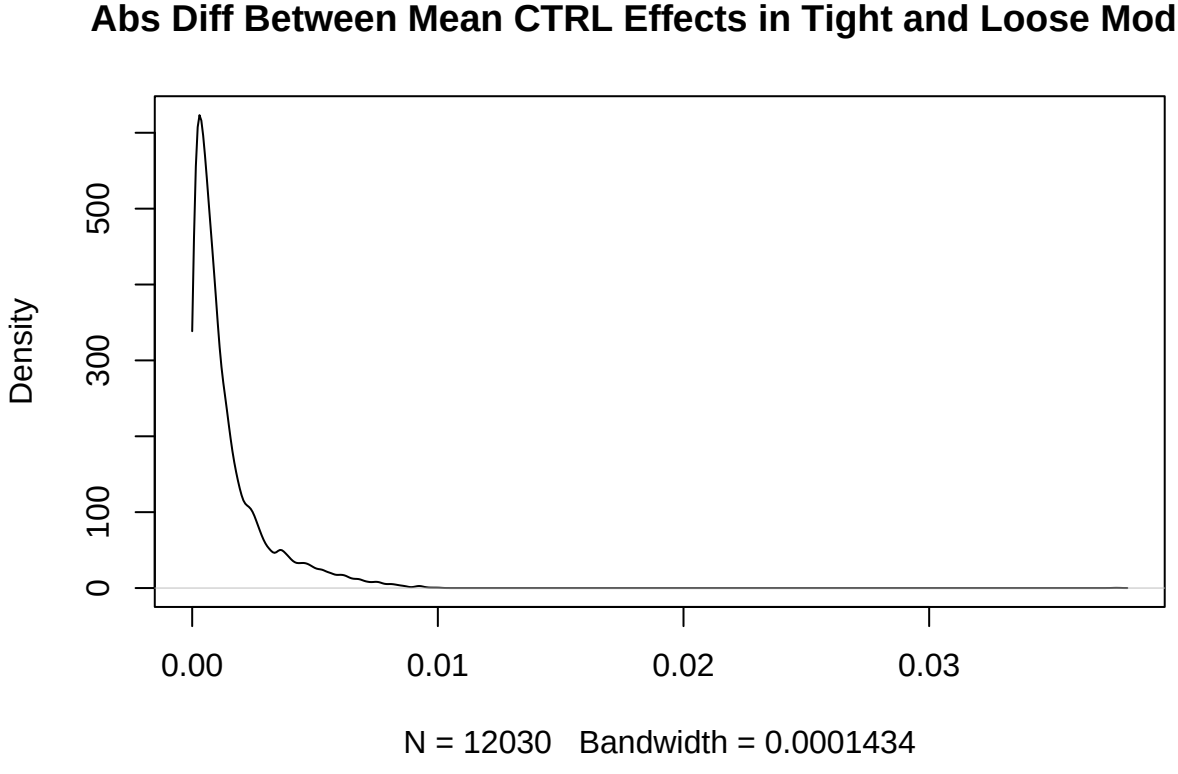
Moreover, when examining the estimated true effects on the training dataset between these two models, they look incredibly similar. Below are density plots that show the difference of the estimated effects between the two models. What we can see is that the different fitted priors have very little influence on the fitted effects.

```
summary_table_samp <- summary_table_samp %>%
  dplyr::mutate(
    post_mean_hs_loose = mash_loose_posterior$result$PosteriorMean[,'hs'],
    post_mean_ctrl_loose = mash_loose_posterior$result$PosteriorMean[,'ctrl'],
    post_mean_hs_tight = mash_tight_posterior$result$PosteriorMean[,'hs'],
    post_mean_ctrl_tight = mash_tight_posterior$result$PosteriorMean[,'ctrl'],
  )

summary_table_samp <- summary_table_samp %>%
  dplyr::mutate(
    hs_tight_loose_diff = abs(post_mean_hs_loose - post_mean_hs_tight),
    ctrl_tight_loose_diff = abs(post_mean_ctrl_loose - post_mean_ctrl_tight),
  )
```

```
plot(
  density(summary_table_samp$hs_tight_loose_diff, from = 0),
  main = "Abs Diff Between Mean HS Effects in Tight and Loose Models"
)
```




```
plot(
  density(summary_table_samp$ctrl_tight_loose_diff, from = 0),
   main = "Abs Diff Between Mean CTRL Effects in Tight and Loose Models"
  )
```



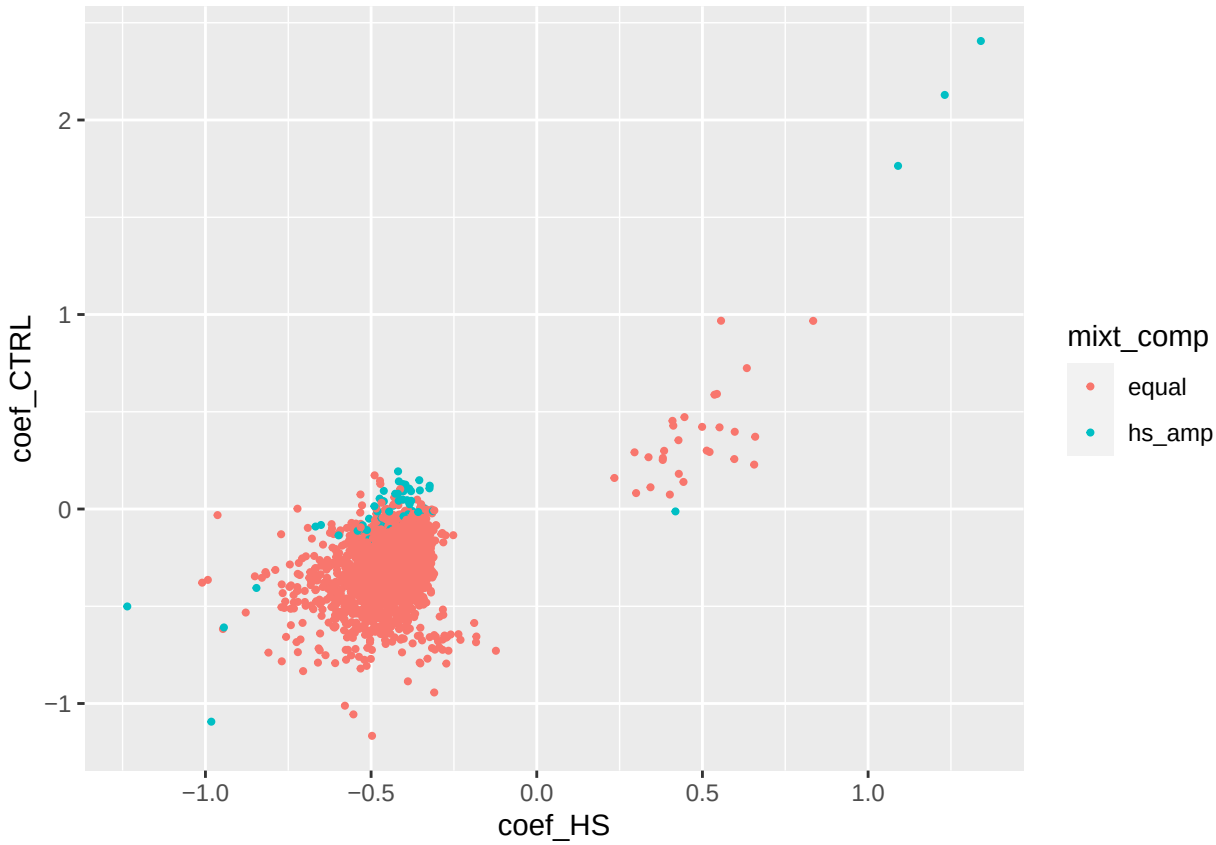
So, there seems to be no obvious "winner" when it comes to these two models. They both seem to fit the data reasonably well, and also they seem to have similar disadvantages. The difficulty in this conclusion is that the classification of points looks quite different between the two models.

Unfortunately, unless we sample down to the set of sites that Pallares et al. labelled significant, it is difficult to see what's going on. In the loose fitted model without sampling, almost all SNPs are labelled as equal. In the tight fitted model without sampling, almost all SNPs are labelled as 1.125 amplification. Part of this effect may be empirical Bayes coming back to bite us. Because we assume no uncertainty in the prior when calculating the posterior class membership probabilities, the prior seems to be overwhelming the likelihood. There are some interesting insights to glean from the plots of the significant SNPs in Pallares et al.

```
cov_mat_names_loose <- colnames(mash_loose_posterior$posterior_weights)
map_mash_loose <- cov_mat_names_loose[max.col(mash_loose_posterior$posterior_weights)]
map_mash_loose_df <- data.frame(
  site = summary_table_samp$site,
  cov_map = map_mash_loose
)
summary_table_samp_loose <- summary_table_samp %>%
  dplyr::inner_join(map_mash_loose_df, by=c("site"))
```

```
cov_mat_names_loose_sig <- colnames(mash_loose_posterior_signif$posterior_weights)
map_mash_loose_sig <- cov_mat_names_loose_sig[
  max.col(mash_loose_posterior_signif$posterior_weights)
]
map_mash_loose_sig_df <- data.frame(
  site = summary_table_signif$site,
  cov_map = map_mash_loose_sig
)
summary_table_sig_loose <- summary_table_signif %>%
  dplyr::inner_join(map_mash_loose_sig_df, by=c("site"))
```

```
summary_table_sig_loose_samp <- summary_table_sig_loose %>%
  dplyr::filter(
    cov_map %in% c(
      "equal_corr_1.11", "equal_corr_1.12", "hs_amp_1.5_corr_1.11", "hs_amp_1.5_corr_1.15"
      )
  ) %>%
  dplyr::mutate(
    mixt_comp = dplyr::case_when(
      cov_map %in% c("equal_corr_1.11", "equal_corr_1.12") ~ "equal",
      TRUE ~ "hs_amp"
    )
  )

ggplot(
  data = summary_table_sig_loose_samp,
  aes(x = coef_HS, y = coef_CTRL, color = mixt_comp)
) +
  geom_point(size = .75)
```

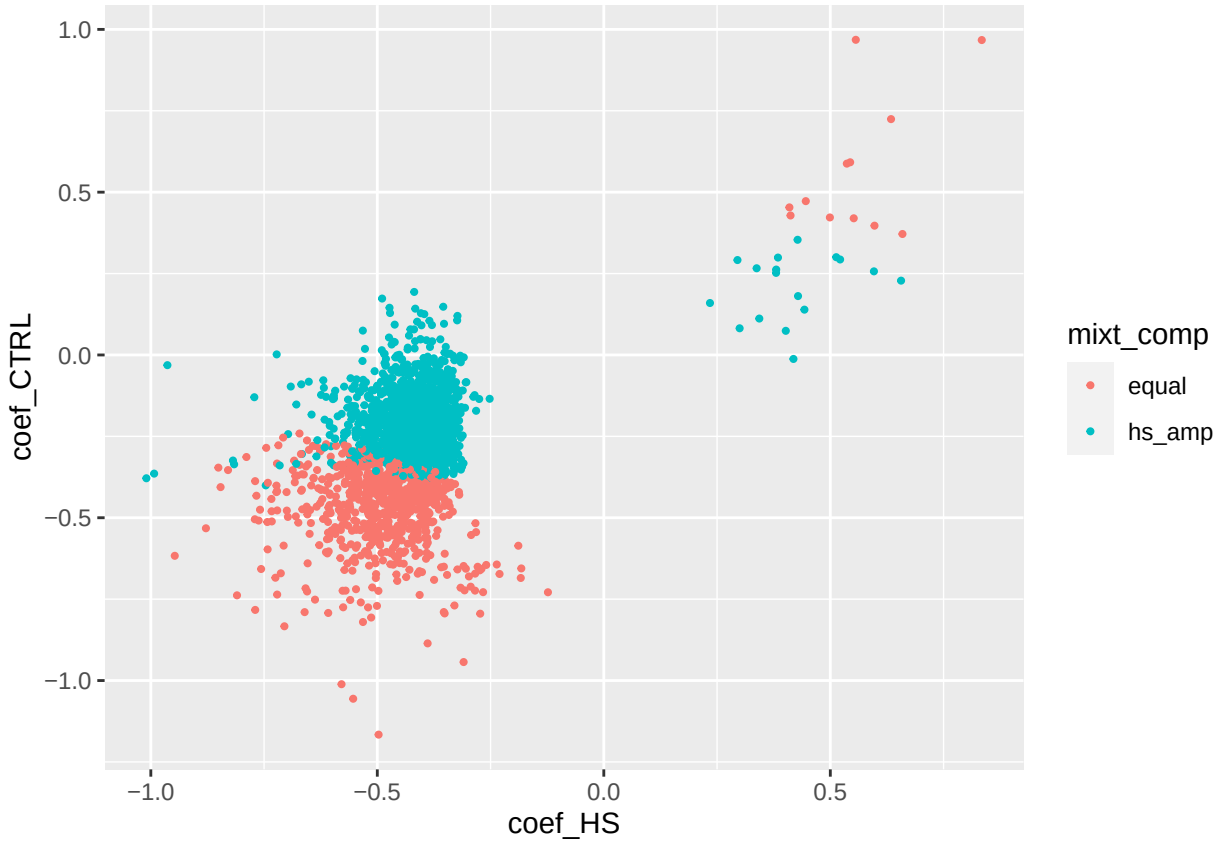



```
cov_mat_names_tight <- colnames(mash_tight_posterior$posterior_weights)
map_mash_tight <- cov_mat_names_tight[
  max.col(mash_tight_posterior$posterior_weights)
]
map_mash_tight_df <- data.frame(
  site = summary_table_samp$site,
  cov_map = map_mash_tight
)
summary_table_samp_tight <- summary_table_samp %>%
  dplyr::inner_join(map_mash_tight_df, by=c("site"))
```

```
cov_mat_names_tight_sig <- colnames(mash_tight_posterior_signif$posterior_weights)
map_mash_tight_sig <- cov_mat_names_tight_sig[
  max.col(mash_tight_posterior_signif$posterior_weights)
]
map_mash_tight_sig_df <- data.frame(
  site = summary_table_signif$site,
  cov_map = map_mash_tight_sig
)
summary_table_sig_tight <- summary_table_signif %>%
  dplyr::inner_join(map_mash_tight_sig_df, by=c("site"))
```

```
summary_table_sig_tight_samp <- summary_table_sig_tight %>%
  dplyr::filter(cov_map %in% c("equal_corr_1.12", "hs_amp_1.125_corr_1.11")) %>%
  dplyr::mutate(mixt_comp = dplyr::case_when(
    cov_map == "equal_corr_1.12" ~ "equal",
    TRUE ~ "hs_amp"
```

```
  ))

ggplot(
  data = summary_table_sig_tight_samp,
  aes(x = coef_HS, y = coef_CTRL, color = mixt_comp)
) +
  geom_point(size = .75)
```



We see that the main difference between the results of the two models is the point at which effects are most likely to come from HS amplification vs. equal effects.

It's also interesting to look at the tables which classify effects in the tight and loose models.

For the loose model:

```
table(summary_table_sig_loose_samp$sig_cat, summary_table_sig_loose_samp$mixt_comp)
```

```
##
##           equal hs_amp
##   CTRL        1      0
##   HS        641     59
##   shared   1538      7
```

For the tight model:

```
table(summary_table_sig_tight_samp$sig_cat, summary_table_sig_tight_samp$mixt_comp)
```

```
##
##           equal hs_amp
##   CTRL        1      0
```

```
##   HS         3    697
##   shared   782    757
```

It's very difficult to say which of the models is preferable. They clearly have very different properties from the perspective of clustering. I'm becoming skeptical of the ability of mash to effectively give accurate cluster labels to different mixture components under at least some set of conditions. It may be useful to do a sensitivity analysis simulation here. Another possibility is to attempt to use a Dirichlet mixture model that is intended for clustering.