Week 6 Work

Eric Weine

4/7/2022

Stan modelling

This week I wanted to add additional complexity to the stan model by adding a null component to the model. This allows us to more easily compare the results of the stan model with the results of mash. Now, there are three possible signals in the stan model.

(1) The true signal is null in both environments.

(2) The true signal is non-null but equal in both environments.

(3) The true signal is non-null in both environments and amplified in the high sugar environment.

The model code is shown below:

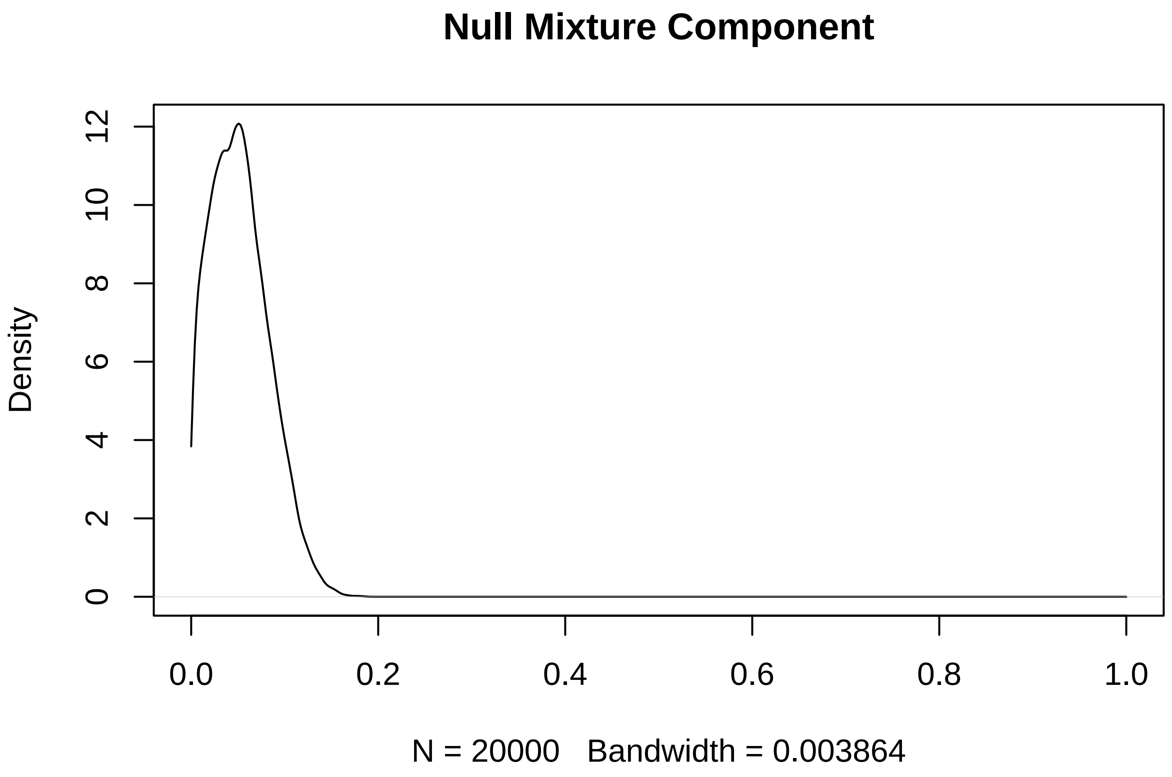
```
data {
 int < lower = 1 > N; // Sample size
 vector[N] h; // high sugar measured effects
 vector[N] c; // control measured effects
 vector<lower = 0>[N] s_h; // high sugar se
 vector<lower = 0>[N] s_c; // control se
}

parameters {

 simplex[3] pi; // Mixture model proportions
 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
 vector[3] hs_contributions;
 mu_theta ~ normal(0, sqrt(3));
 sigma_theta ~ normal(0, sqrt(3));
 alpha ~ normal(0, sqrt(4));

 for(i in 1:N) {

   target += log_sum_exp(
     log(pi[1]) +
     normal_lpdf(c[i] | 0, s_c[i]),
     log(1 - pi[1]) +
     normal_lpdf(c[i] | mu_theta, sqrt(square(s_c[i]) + square(sigma_theta)))
   );

   hs_contributions[1] = log(pi[1]) + normal_lpdf(h[i] | 0, s_h[i]);
   hs_contributions[2] = log(pi[2]) + normal_lpdf(h[i] |
    mu_theta, sqrt(square(s_h[i]) + square(sigma_theta))
  );
   hs_contributions[3] = log(pi[3]) +
     normal_lpdf(
       h[i] | (1 + alpha) * mu_theta,
       sqrt(square(s_h[i]) + square(1 + alpha) * square(sigma_theta)));

   target += log_sum_exp(hs_contributions);

 }

}
```

Essentially, the only difference between this model and previous models is that we draw the true effects for each SNP from a spike-slab prior, which includes the possibility of true effects of 0.

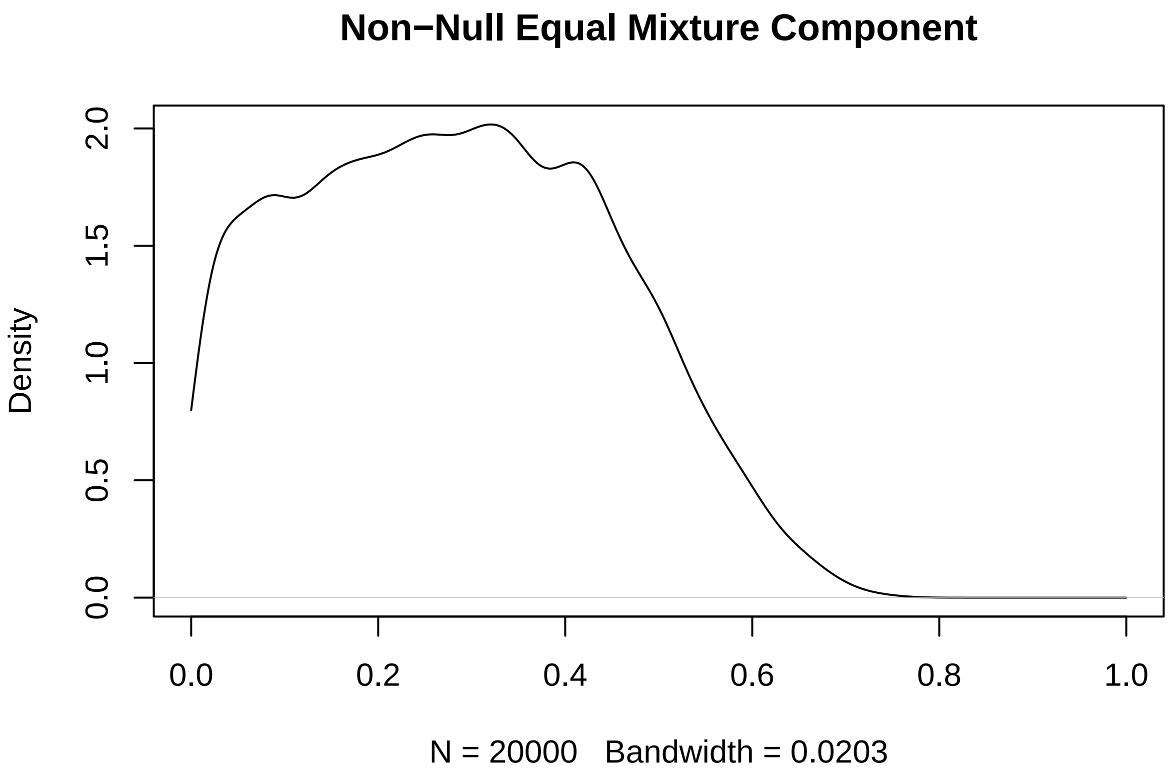
With a normal likelihood, this model seemed to have good results and converged quickly. Many of the posterior parameters had a high degree of uncertainty, but this may just be because of the complexity of the model.

First, we look at the posterior distributions of the mixture proportions.

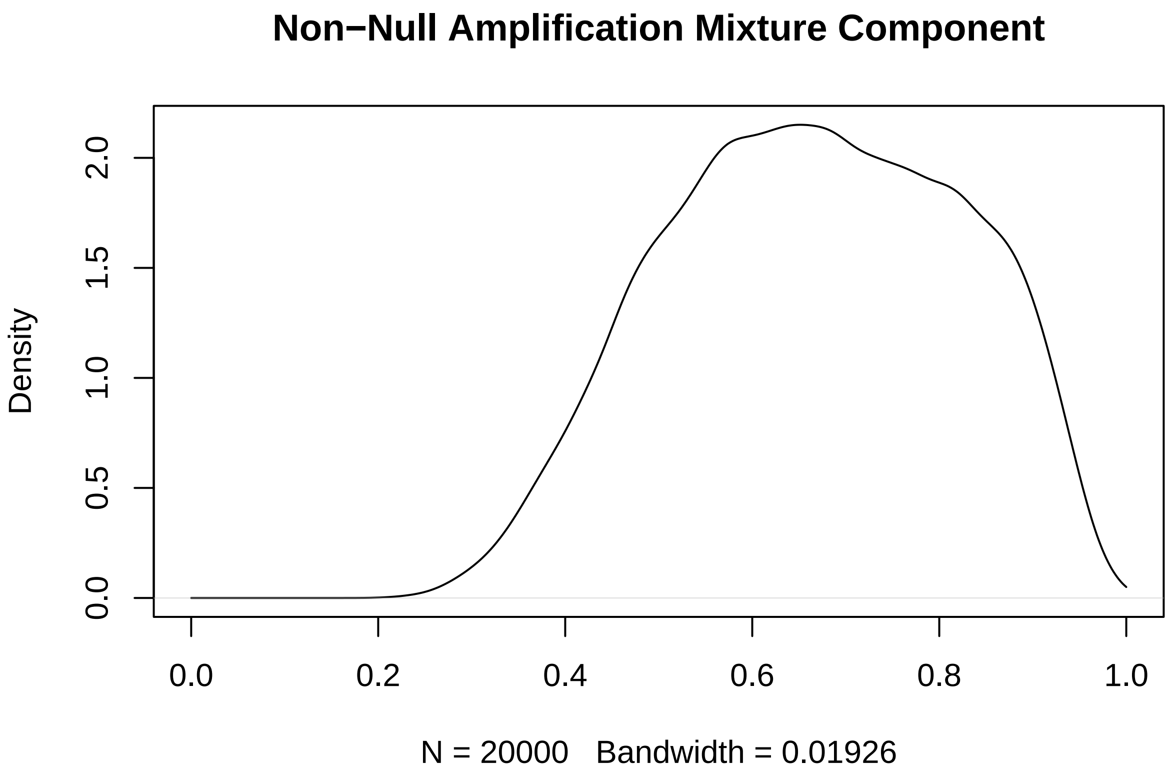
```
plot(density(post_dist$pi[,1], from = 0, to = 1), main = "Null Mixture Component")
```



```
plot(density(post_dist$pi[,2], from = 0, to = 1), main = "Non-Null Equal Mixture Component")
```



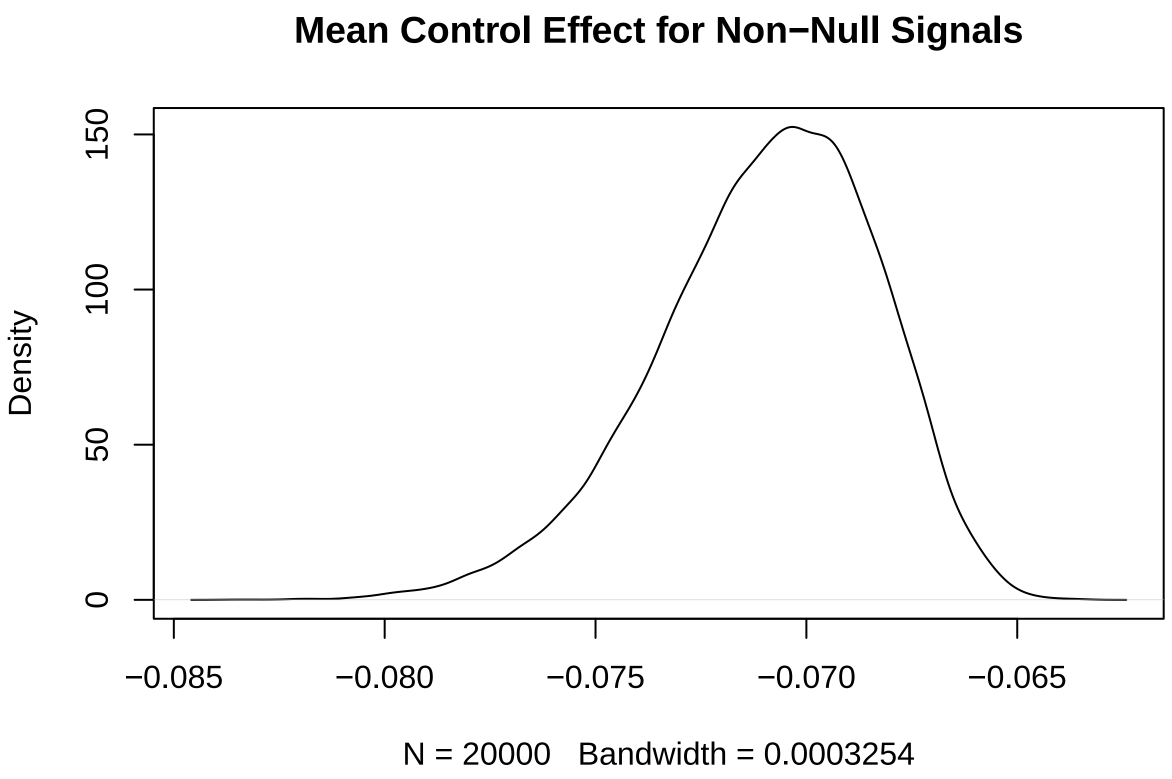
```
plot(density(post_dist$pi[,3], from = 0, to = 1), main = "Non-Null Amplification Mixture Component")
```



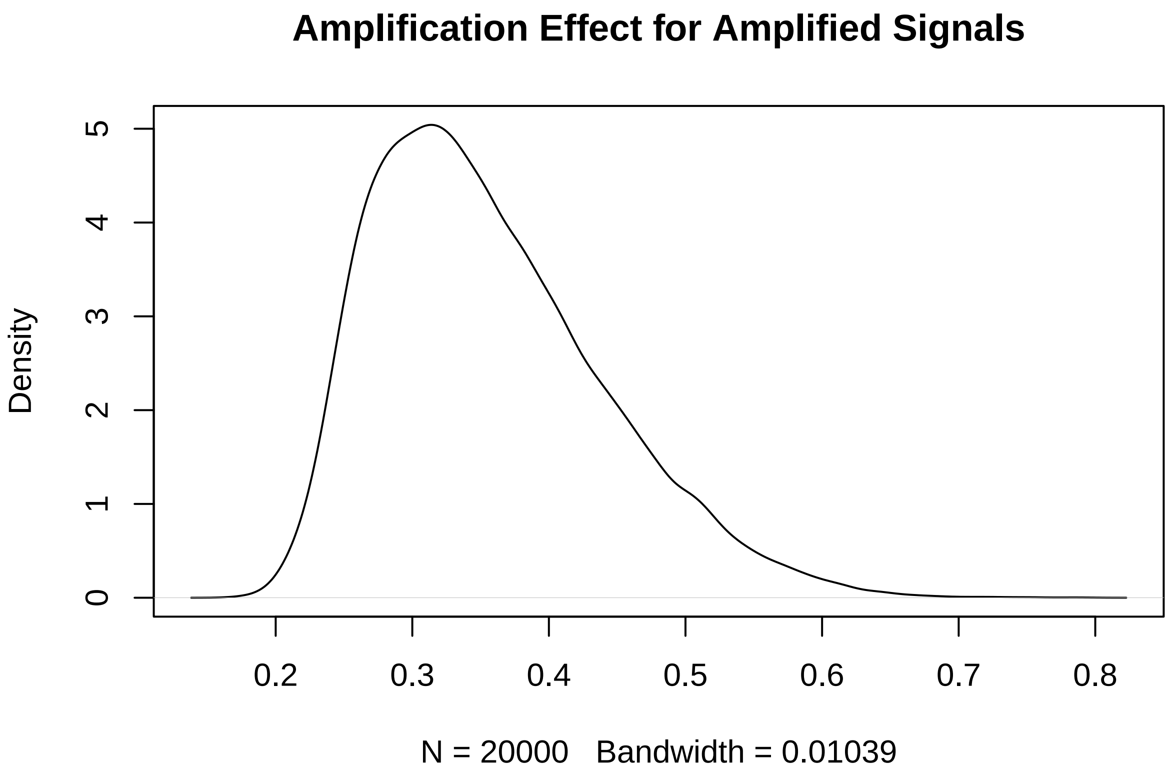
Overall, the mean posterior estimates for the parameters were about 5.2% for null, 28.4% for non-null and equal, and 66.5% for non-null and amplified.

It's also instructive to look at the mean and amplification parameters. Their posterior densities are shown below:

```r
plot(density(post_dist$mu_theta), main = "Mean Control Effect for Non-Null Signals")
```



```r
plot(density(post_dist$alpha), main = "Amplification Effect for Amplified Signals")
```



Thus, the model predicts that on average effects in the high sugar environment are 1.36X the effect in the control environment.

Finally, we can also classify signals based on the posterior probabilities of their membership to different mixture components. This is perhaps most interesting to look at for the set of signals that Pallares et al. classifies as "significant".

```r
summary_table_sig <- read.delim('data/SummaryTable_allsites_12Nov20.txt') %>%
  dplyr::filter(sig_cat != 'NS')

# replace 0 p-values with small numbers
summary_table_sig <- summary_table_sig %>%
  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_sig <- summary_table_sig %>%
  dplyr::mutate(
    std_error_ctrl = abs(coef_CTRL) / qnorm((2 - pval_CTRL) / 2),
    std_error_hs = abs(coef_HS) / qnorm((2 - pval_HS) / 2)
  )
```

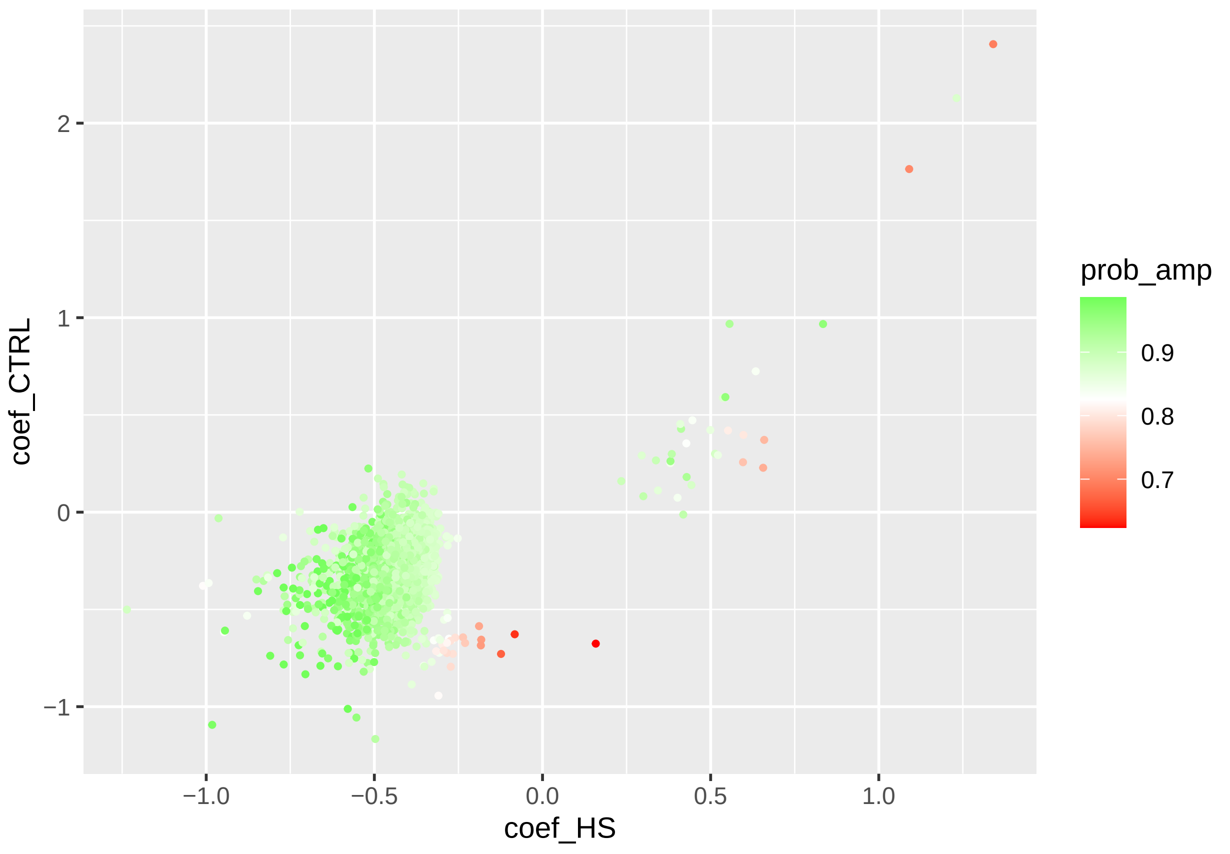
```r
obs <- summary_table_sig$coef_HS
sds <- summary_table_sig$std_error_hs
mu_theta <- mean(post_dist$mu_theta)
var_theta <- mean(post_dist$sigma_theta ^ 2)
alpha <- mean(post_dist$alpha)

n_obs <- length(obs)
```

```r
n_classes <- 3
class_lik_mat <- matrix(nrow = n_obs, ncol = n_classes)

for (i in 1:n_obs) {

  for (j in 1:n_classes) {

    if (j == 1) {

      lik <- dnorm(x = obs[i], mean = 0, sd = sds[i])

    } else if(j == 2) {

      lik <- dnorm(
        x = obs[i],
        mean = mu_theta,
        sd = sqrt(var_theta + sds[i] ^ 2)
      )

    } else {

      lik <- dnorm(
        x = obs[i],
        mean = mu_theta * (1 + alpha),
        sd = sqrt(((1 + alpha) ^ 2) * var_theta + sds[i] ^ 2)
      )

    }

      class_lik_mat[i, j] <- lik

  }

}

class_prior_mat <- matrix(
  nrow = n_obs,
  ncol = n_classes,
  data = rep(c(mean(post_dist$pi[,1]), mean(post_dist$pi[,2]), mean(post_dist$pi[,3])), n_obs),
  byrow = TRUE
)

class_joint_mat <- class_lik_mat * class_prior_mat
# normalize rows
class_prob_mat <- t(apply(class_joint_mat, 1, function(x) x/sum(x)))
```

The difficulty with this is that because of the prior probabilities, literally every signal in the set of significant signals is classified as most likely being from the amplification mixture component. This is likely because the normal distribution corresponding to the mixture component of non-amplification is quite similar to the distribution corresponding to the amplification component, and so the prior becomes more important than the likelihood.

Below is a scatterplot of the regression coefficients colored by the probability that the high-sugar point comes from the amplification mixture component

```r
summary_table_sig$prob_amp <- class_prob_mat[,3]
ggplot(data = summary_table_sig, aes(x = coef_HS, y = coef_CTRL, col = prob_amp)) +
  geom_point(size = .75) +
  scale_color_gradient2(low = "red", high = "green", midpoint = .825)
```



Overall, it seems that trying to use MAP or MLE estimates to classify the variants might run into some difficulties with the stan model. One other possibility is to classify the variants without applying the prior, though this seems statistically suspect.

MASH modelling

I also wanted to examine the posterior mixture component membership when using the mash model. However, I also changed the grid of covariance matrices for MASH, because the stan model indicated that the average amplification is only 1.35X, and the smallest amplification effect that previous iterations of the mash model allowed for was 1.5X.

In this iteration of the model, I created covariance matrices with the usual grid of correlations and with amplification effects in the range of 1.125, 1.25, 1.375, 1.5, 1.625, and 1.75.

```r
set.seed(2034)

#' 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}
```

```r
#'
#' @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)

}

# read in 50% of data and select relevant columns
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
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 <- summary_table %>%
  dplyr::filter(sig_cat != 'NS')

reg_fx_samp_mat <- t(matrix(
  data = c(summary_table_samp$coef_CTRL, summary_table_samp$coef_HS),
```

```r
  nrow = 2,
  byrow = TRUE
))
colnames(reg_fx_samp_mat) <- c("ctrl", "hs")

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

mash_samp_data <- mashr::mash_set_data(reg_fx_samp_mat, reg_se_samp_mat)
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(1.125, 1.25, 1.375, 1.5, 1.625, 1.75)

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

mash_full_fit <- mashr::mash(
  data = mash_data,
  g = ashr::get_fitted_g(mash_samp_fit),
  fixg = TRUE
)
```

The mixture components for the new model are shown below:

```
estimated_pi_tight_grid <- readr::read_rds(
  "rds_data/mash_estimated_pi_tight_grid.rds"
)
print(estimated_pi_tight_grid[estimated_pi_tight_grid >= 1e-3])
```

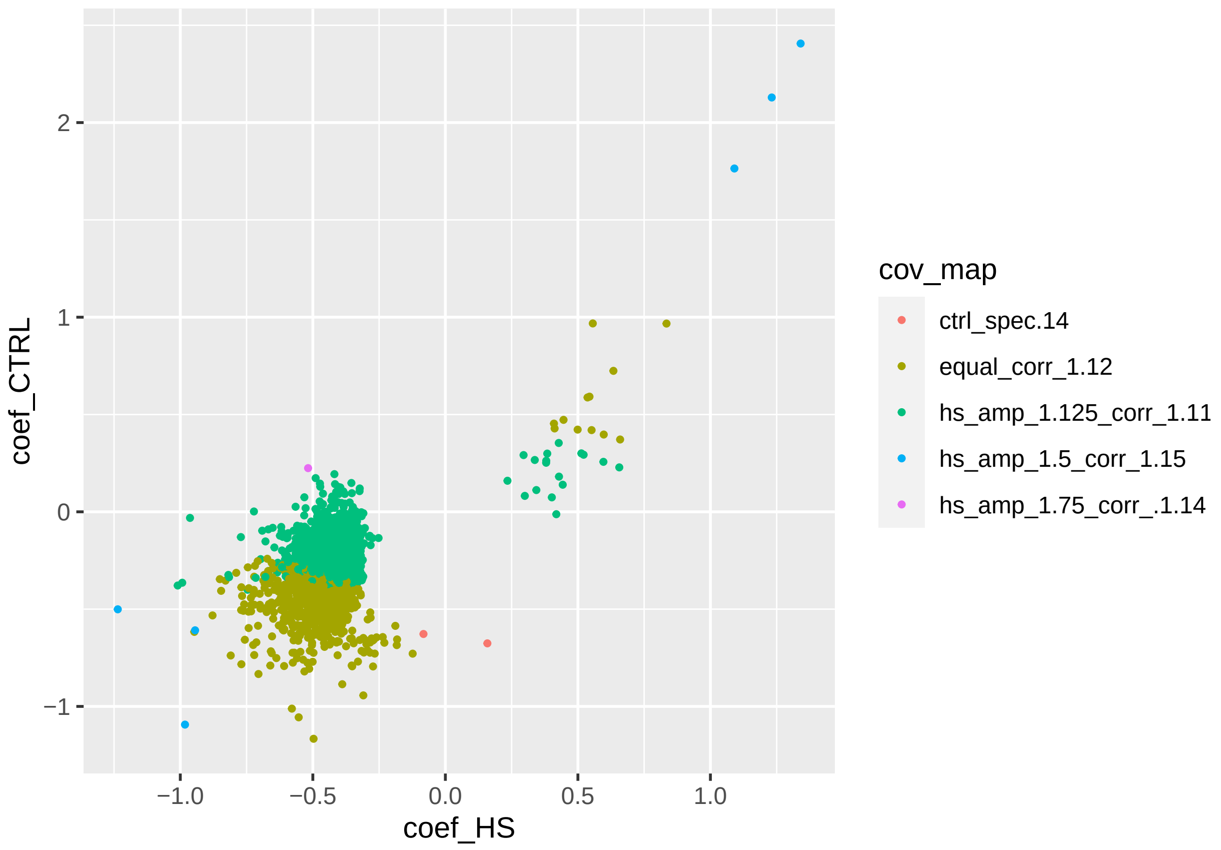
```
##                  null          equal_corr_1   hs_amp_1.125_corr_1
##           0.086182500           0.147569352           0.759064029
## ctrl_amp_1.125_corr_1   ctrl_amp_1.5_corr_1
##           0.003057941           0.003600747
```

With this new covariance grid, we see much more weight being put on HS signals with a very slight amount of amplification. This coheres more closely with the STAN model, though the level of amplification is slightly less.

```
post_preds_mash <- readr::read_rds(
 "~/Documents/academic/drosophila_longevity/drosophila-longevity/rds_data/mash_sig_posterior_weights_df
)
```

With this new model in hand, I wanted to look at the posterior predictions of class membership given by mash. Below is a scatterplot of the regression coefficients colored by their most likely mixture component membership.

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



The most interesting part of the plot is on the bottom left, where we can see that there are two clusters: one of equal effects and the other of amplified effects, with a relatively well defined dividing line between the clusters.

Testing Approach

A third approach to demonstrating different effect sizes between control and high sugar effects is to use a test to compare the coefficient in the high sugar group to the coefficient in the control group. While in some ways this is less appealing because it does not specify a generative model for GxE, it also requires fewer assumptions about the mathematical nature of potential amplifications.

Specifically, we could envision conducting the hypothesis test that the effect of a SNP in the high sugar group has a more significant negative effect on longevity than the effect of a SNP in the control group.

For each SNP, let $\hat{\beta}_{hs}$ and $\hat{\beta}_c$ be the estimated high sugar and control effects, respectively. Also, let $\hat{\sigma}_{hs}$ and $\hat{\sigma}_c$ be the standard error estimates for the regression estimates of the high sugar and control effects, respectively. Then, we can define a Z-score as:

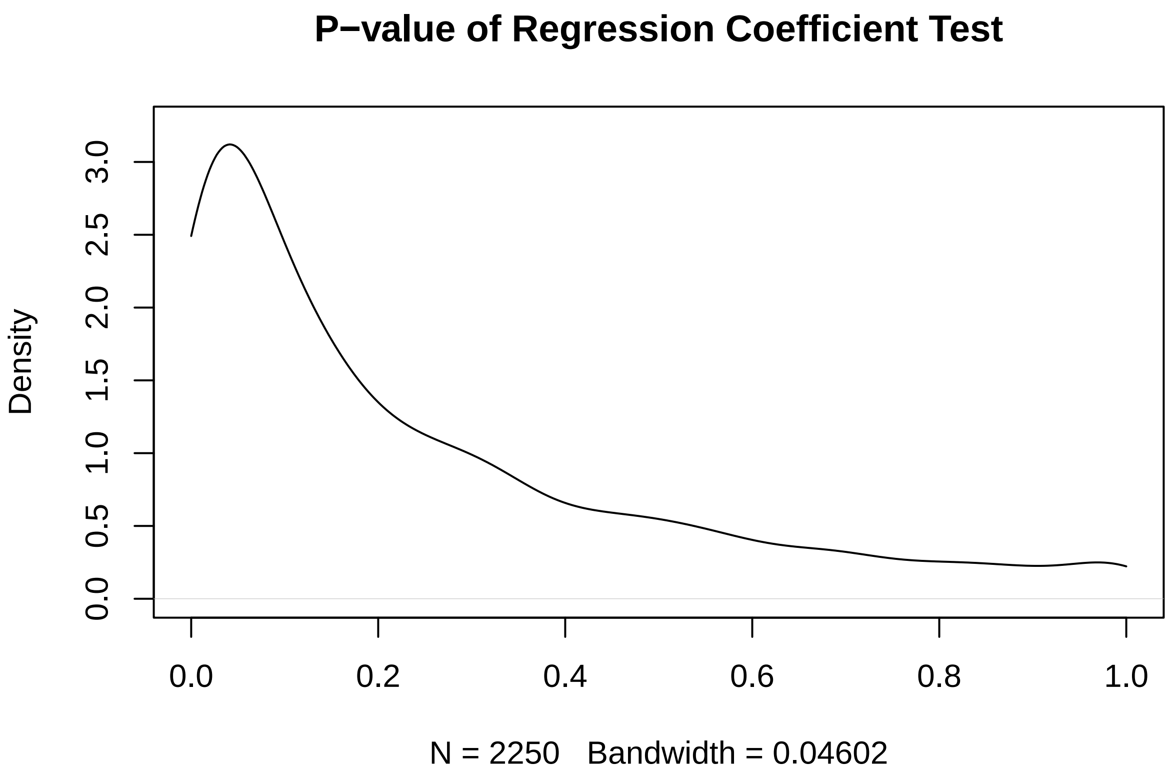
$$z = \frac{\hat{\beta}_{hs} - \hat{\beta}_c}{\sqrt{\hat{\sigma}_{hs}^2 + \hat{\sigma}_c^2}}$$

The density plot of p-values in shown below for this test:

```r
summary_table_sig <- summary_table_sig %>%
  dplyr::mutate(reg_z = (coef_HS - coef_CTRL) / sqrt(std_error_ctrl ^ 2 + std_error_hs ^ 2))
```

```
summary_table_sig <- summary_table_sig %>%
  dplyr::mutate(reg_p = pnorm(reg_z))
```

```
plot(density(summary_table_sig$reg_p, from = 0, to = 1), main = "P-value of Regression Coefficient Test
```



Clearly, this distribution deviates very strongly from uniformity, indicating there are likely many signals that are stronger in the case of the high sugar group than the control group

```
summary_table_sig <- summary_table_sig %>%
  dplyr::mutate(reg_fdr = qvalue::lfdr(reg_p))

num_shared_hs_amp <- summary_table_sig %>%
  dplyr::filter(sig_cat == 'shared' & reg_fdr <= .1) %>%
  nrow()
```

If we look at the signals that are described as "shared" by Pallares et al. but have a local FDR below .1, we identify 72 candidate SNPs for amplification.