Week 2 Drosophila Work

Overview

The purpose of my work this week is to build simulated datasets similar to those used by Pallares et al. The purpose of these simulated datasets is to be able to test the power and Type I error of both the model prosposed by Pallares et al. as well as the interaction model proposed by Rebecca / Arbel. Ideally, we aim to show that the interaction model is able to effectively identify types of genetic effects (e.g. amplification) that the origin model is unable to.

Dataset Simulation

As a reference for the dataset simulations, I used the simulation setup that Pallares et al. uses to compare the CMH and beta-binomial approaches for detecting genetic signal. I made some small modifications to this setup (e.g. allowing for different effects in the control and high sugar environments) that I felt would be useful in evaluating the interaction model. Specifically, we assume that n individuals are sampled throughout the experiment ($\frac{1}{2}$ at T_0, $\frac{1}{4}$ in C at T_N, $\frac{1}{4}$ in HS at T_N). For each individual, l loci are sampled. Let π_{ij} be the genotype of individual i at locus j (1 for homozygous alternative, 0.5 for heterozygous, 0 otherwise). Let y_{ij} be the number of alternative allele reads of individual i at locus j, and let r_j be the total number of allele reads at locus j (assume this is the same for all individuals). Finally, let $\bar{\pi}_{\cdot j}^{(c)}$ be the average alternative allele frequency for all individuals at locus j in condition c, where $c \in \{T_0, C, HS\}$.

Then, we use the following procedure to generate a simulated dataset:

1.

- Generate $\bar{\pi}_{\cdot 1}^{(T_0)}, ..., \bar{\pi}_{\cdot l}^{(T_0)} \sim \hat{F}_{maf}$ independently, where $\hat{F}_{maf}$ is the estimated distribution of minor allele frequencies from the data.
- Generate the change in minor allele frequencies for each site j and each condition c, $\Delta_j^{(c)}$ according to a mixture distribution. For more details on the possible parameters of this mixtures please see the code below. But, in summary, this mixture distribution can put probability on 5 different cases. (1) Both conditions null, (2) HS alt, C null, (3) C alt, HS null, (4) HS and C alt, same magnitude, (5) HS and C alt, HS effect is amplified. Once these change values are calculated, use them to calculate the minor allele frequencies for each condition at each site at T_N.

2. For each locus j, generate genotypes for each individual, $\pi_{1j}, ..., \pi_{1n}$ according to the corresponding minor allele frequency for each site and condition.
3. For each locus j, generate the total number of reads at that locus, r_j from a mixture of negative binomial distributions. This mixture is determined empirically by fitting a negative binomial to 5000 randomly selected alleles, and then putting a weight of $\frac{1}{5000}$ on each distribution. This is exactly how Pallares et al. conducts their power simulation.
4. Finally, generate the number of alternative reads at each locus for each individual as:

$$y_{ij} \sim Bin(r_j, \pi_{ij})$$

Once these steps are completed, we have a dataset on which to compute analysis. Below is the code to generate datasets of this form.

```r
#' Generate n genotypes with a minor allele frequency
#'
#' @param maf Minor allele frequency. Must be in [0, 1]
#' @param n Total number of genotypes to generate
#'
#' @return Vector of genotypes, where 0 is homozygous major allele,
#' .5 is heterozygous, and 1 is homozygous minor allele
#' @export
#'
#' @examples
generate_genotypes <- function(maf, n) {

  sample(
    x=c(0, 0.5, 1),
    size=n,
    prob=c((1 - maf) ^ 2, 2 * maf * (1 - maf), maf ^ 2),
    replace = TRUE
  )

}

#' Generate Allele Frequency Changes for Simulated Dataset
#'
#' The function assumes that the true allele frequency changes
#' for each site follows one of a few cases:
#' (1) The minor allele frequency change is 0 in both cases.
#' (2) The minor allele frequency changes by \code{cx} percent in the HS case only.
#' (3) The minor allele frequency changes by \code{cx} percent in the C case only.
#' (4) The minor allele frequency changes by \code{cx} percent in both cases.
#' (5) The minor allele frequency changes by \code{cx} percent in C
#' and \code{cx} * \code{hs_amp_coef} in HS.
#' The user is able to input the magnitude of changes as well as
#' the probability of landing in each case.
#'
#' @param sites Number of SNPs for which to generate data
#' @param prob_both_null Probability of being in case 1 in the description
#' @param prob_hs_only Probability of being in case 2 in the description
#' @param prob_c_only Probability of being in case 3 in the description
#' @param prob_same Probability of being in case 4 in the description
#' @param prob_amp Probability of being in case 5 in the description
#' @param cx Percentage change of minor allele frequency at time TN
#' @param hs_amp_coef Amplification constant, see description for more details
#'
#' @return Dataframe with minor allele frequency changes for each site in the HS and C cases.
#' @export
#'
#' @examples
generate_tn_allele_cxs <- function(
  sites,
  prob_both_null,
  prob_hs_only,
  prob_c_only,
  prob_same,
```

```
  prob_amp,
  cx,
  hs_amp_coef = 1.0
) {

  # Create matrices to select from so that only one call to sample is made
  # c(both_null, hs_only, c_only, same, amp)
  c_matrix_data <- c(rep(0, 2 * sites), rep(cx, 3 * sites))
  c_matrix <- matrix(data = c_matrix_data, ncol = sites, byrow = TRUE)
  hs_matrix_data <- c(
    rep(0, sites),
    rep(cx, sites),
    rep(0, sites),
    rep(cx, sites),
    rep(cx * hs_amp_coef, sites)
  )
  hs_matrix <- matrix(data = hs_matrix_data, ncol = sites, byrow = TRUE)

  # Select from 5 cases with specified probability
  case_gen <- sample(
    x = c(1:5),
    size = sites,
    replace = TRUE,
    prob = c(prob_both_null, prob_hs_only, prob_c_only, prob_same, prob_amp)
  )
  str_possibilities <- c("both_null", "hs_only", "c_only", "same", "hs_amp")
  str_selections <- str_possibilities[case_gen]

  # Find proper indices in matrix
  selected_indices <- c(0:(sites - 1)) * 5 + case_gen
  selected_c_cxs <- c_matrix[selected_indices]
  selected_hs_cxs <- hs_matrix[selected_indices]

  maf_cx_tn_df <- data.frame(c_cx = selected_c_cxs, hs_cx = selected_hs_cxs)
  return(list(cx_df = maf_cx_tn_df, true_signal = str_selections))

}

#' Generate Simulated Drosophila Dataset With Specified Signal Setup
#'
#' The function assumes that the true allele frequency changes
#' for each site follows one of a few cases:
#' (1) The minor allele frequency change is 0 in both cases.
#' (2) The minor allele frequency changes by \code{cx}
#' percent in the HS case only.
#' (3) The minor allele frequency changes by \code{cx}
#' percent in the C case only.
#' (4) The minor allele frequency changes by \code{cx} percent in both cases.
#' (5) The minor allele frequency changes by \code{cx}
#' percent in C and \code{cx} * \code{hs_amp_coef} in HS.
#' The user is able to input the magnitude of changes as well as the probability
#' of landing in each case.
#'
```

```r
#' @param n Total number of flies. 1/2 are at time 0,
#' 1/4 are high sugar at time N, 1/4 are control at time N
#' @param sites Total number of SNP sites for each fly
#' @param prob_both_null Probability of being in case 1 in the description
#' @param prob_hs_only Probability of being in case 2 in the description
#' @param prob_c_only Probability of being in case 3 in the description
#' @param prob_same Probability of being in case 4 in the description
#' @param prob_amp Probability of being in case 5 in the description
#' @param cx Percentage change of minor allele frequency at time TN
#' @param hs_amp_coef Amplification constant, see description for more details
#' @param nb_params_dist Vector of lists of negative binomial parameters to be
#' sampled from to determine reads for each SNP.
#' For each SNP, one set of these parameters are sampled, and then a negative
#' binomial distribution with these parameters
#' is used to generate \code{n} read counts for each SNP. Each list should
#' contain an element t corresponding to
#' the \code{n} argument in \code{rnbinom} and \code{p} corresponding to the
#' \code{prob} argument in \code{rnbinom}.
#'
#' @return A list with (1) a dataframe containing information from
#' simulations and (2) a dataframe containing
#' the generated signal classicification for each site
#' @export
#'
#' @examples
generate_simulated_dataset <- function(
  n,
  sites,
  maf_dist,
  prob_both_null,
  prob_hs_only,
  prob_c_only,
  prob_same,
  prob_amp,
  cx,
  nb_params_dist,
  hs_amp_coef = 1.0
) {

  # Divide fly counts into groups
  n_t0 <- floor(n / 2)
  n_hs_tn <- floor((n - n_t0) / 2)
  n_c_tn <- n_hs_tn

  # Sample parameters before loop for speed
  maf_t0_freqs <- sample(x = maf_dist, size = sites, replace = TRUE)
  freq_change_list <- generate_tn_allele_cxs(
    sites,
    prob_both_null,
    prob_hs_only,
    prob_c_only,
    prob_same,
    prob_amp,
```

```
  cx,
  hs_amp_coef
)
freq_change_df <- freq_change_list$cx_df
true_signal_df <- data.frame(
  site = c(1:sites), true_signal = freq_change_list$true_signal
)

maf_hs_tn_freqs <- maf_t0_freqs * (
  1 + freq_change_df$hs_cx
)
maf_c_tn_freqs <- maf_t0_freqs * (
  1 + freq_change_df$c_cx
)
nb_site_params <- sample(x = nb_params_dist, size = sites, replace = TRUE)

site_vec <- c()
total_reads_vec <- c()
alt_reads_vec <- c()
condition_vec <- c()
time_vec <- c()

for(i in c(1:sites)) {

  # Generate genotypes for HS and Control at T0 and TN
  site_t0_maf <- maf_t0_freqs[i]
  genotypes_t0 <- generate_genotypes(site_t0_maf, n_t0)

  site_tn_hs_maf <- maf_hs_tn_freqs[i]
  genotypes_hs_tn <- generate_genotypes(site_tn_hs_maf, n_hs_tn)

  site_tn_c_maf <- maf_c_tn_freqs[i]
  genotypes_c_tn <- generate_genotypes(site_tn_c_maf, n_c_tn)

  # Sample reads based on negative binomial distribution
  # Make sure there are never 0 reads, this should happen very rarely
  total_reads <- pmax(
    1,
    rnbinom(
      n = n, size = nb_site_params[i][[1]]$t, prob = nb_site_params[i][[1]]$p
    )
  )
  alt_reads <- rbinom(
    n = n,
    size = total_reads,
    prob = c(genotypes_t0, genotypes_hs_tn, genotypes_c_tn)
  )

  site_vec <- c(site_vec, rep(i, n))
  total_reads_vec <- c(total_reads_vec, total_reads)
  alt_reads_vec <- c(alt_reads_vec, alt_reads)
  condition_vec <- c(
    condition_vec, rep("both", n_t0), rep("HS", n_hs_tn), rep("C", n_c_tn)
  )
```

```
    time_vec <- c(time_vec, rep("T0", n_t0), rep("TN", n_hs_tn + n_c_tn))

  }

  sim_df <- data.frame(
    site = site_vec,
    total_reads = total_reads_vec,
    alt_reads = alt_reads_vec,
    condition = condition_vec,
    time = time_vec
  )

  return(list(sim_df = sim_df, true_signal_df = true_signal_df))

}
```

To ensure correctness of this simulation process, I built code to run some automated tests.

```
test_generate_simulated_dataset <- function() {

  sim_df <- generate_simulated_dataset(
    n = 20,
    sites = 10,
    maf_dist = c(.25, .4),
    prob_both_null = .5,
    prob_hs_only = 0.1,
    prob_c_only = 0.1,
    prob_same = 0.1,
    prob_amp = 0.2,
    cx = -0.15,
    hs_amp_coef = 1.5,
    nb_params_dist = list(list(t = 25, p = .4), list(t = 10, p = .3))
  )$sim_df

  # Test basic expected features of simulated dataset
  num_rows_t0 <- sim_df %>% filter(time == 'T0') %>% nrow()
  stopifnot(num_rows_t0 == 100)

  num_rows_tn <- sim_df %>% filter(time == 'TN') %>% nrow()
  stopifnot(num_rows_tn == 100)

  num_rows_both <- sim_df %>% filter(condition == 'both') %>% nrow()
  stopifnot(num_rows_both == 100)

  num_rows_c <- sim_df %>% filter(condition == 'C') %>% nrow()
  stopifnot(num_rows_c == 50)

  num_rows_hs <- sim_df %>% filter(condition == 'HS') %>% nrow()
  stopifnot(num_rows_hs == 50)

  sim_df2 <- generate_simulated_dataset(
    n = 500,
    sites = 250,
    maf_dist = c(.25, .4),
```

```
  prob_both_null = 1.0,
  prob_hs_only = 0.0,
  prob_c_only = 0.0,
  prob_same = 0.0,
  prob_amp = 0.0,
  cx = -0.15,
  nb_params_dist = list(list(t = 25, p = .4), list(t = 10, p = .3))
)$sim_df

# Make sure allele frequencies don't vary
# accross time and condition in the all null case
sim_df2 <- sim_df2 %>% mutate(allele_prop = alt_reads / total_reads)
sim_df2_summary <- sim_df2 %>%
  group_by(time, condition) %>%
  summarize(maf = mean(allele_prop))
max_diff <- max(sim_df2_summary$maf) - min(sim_df2_summary$maf)
stopifnot(max_diff < 0.01)

sim_df3 <- generate_simulated_dataset(
  n = 500,
  sites = 250,
  maf_dist = c(.25, .4),
  prob_both_null = 0.0,
  prob_hs_only = 0.5,
  prob_c_only = 0.5,
  prob_same = 0.0,
  prob_amp = 0.0,
  cx = -0.15,
  nb_params_dist = list(list(t = 25, p = .4), list(t = 10, p = .3))
)$sim_df

sim_df3 <- sim_df3 %>% mutate(allele_prop = alt_reads / total_reads)
sim_df3_summary <- sim_df3 %>%
  group_by(condition) %>%
  summarize(maf = mean(allele_prop)) %>%
  filter(condition != 'both')
maf_tn_diff <- max(sim_df3_summary$maf) - min(sim_df3_summary$maf)
stopifnot(maf_tn_diff < 0.01)

sim_df4 <- generate_simulated_dataset(
  n = 500,
  sites = 250,
  maf_dist = c(.25, .4),
  prob_both_null = 0.0,
  prob_hs_only = 0.0,
  prob_c_only = 0.0,
  prob_same = 1.0,
  prob_amp = 0.0,
  cx = -0.15,
  nb_params_dist = list(list(t = 25, p = .4), list(t = 10, p = .3))
)$sim_df

sim_df4 <- sim_df4 %>% mutate(allele_prop = alt_reads / total_reads)
```

```
  sim_df4_summary <- sim_df4 %>%
    group_by(time) %>%
    summarize(maf = mean(allele_prop))
  maf_t0 <- (sim_df4_summary %>% filter(time == 'T0'))$maf[1]
  maf_tn <- (sim_df4_summary %>% filter(time == 'TN'))$maf[1]
  maf_cx <- (maf_tn / maf_t0) - 1
  maf_diff_vs_expected <- maf_cx - .15
  stopifnot(maf_diff_vs_expected < 0.01)

  # Finally, I want to do a test of the amplification
  # Again, check for correct MAFs in this case
  sim_df5 <- generate_simulated_dataset(
    n = 2000,
    sites = 1000,
    maf_dist = c(.25, .4),
    prob_both_null = 0.5,
    prob_hs_only = 0.0,
    prob_c_only = 0.0,
    prob_same = 0.0,
    prob_amp = 0.5,
    hs_amp_coef = 2.0,
    cx = -0.1,
    nb_params_dist = list(list(t = 25, p = .4), list(t = 10, p = .3))
  )$sim_df
  sim_df5 <- sim_df5 %>% mutate(allele_prop = alt_reads / total_reads)
  sim_df5_summary <- sim_df5 %>%
    group_by(time, condition) %>%
    summarize(maf = mean(allele_prop))
  t0_maf <- (sim_df5_summary %>% filter(condition == "both"))$maf[1]
  stopifnot(abs(t0_maf - .325) < .01)
  tn_maf_c <- (sim_df5_summary %>% filter(condition == "C"))$maf[1]
  stopifnot(abs(tn_maf_c - .95 * .325) < .01)
  tn_maf_hs <- (sim_df5_summary %>% filter(condition == "HS"))$maf[1]
  stopifnot(abs(tn_maf_hs - .9 * .325) < .01)

}
```

I also built the code to run the model proposed by Pallares et al. on the simulated data.

```
#' Calculate results for simulated data based on the methods of Pallares et al.
#'
#' The function fits a beta-binomial regression
#' of allele frequency on a 3-level treatment factor(T0, C TN, HS TN).
#' Let the regression coefficient for HS TN be Bh and the regression
#'  coefficient for C TN be Bc. Let the p-value
#' for the HS TN regression coefficient be pBh and the q-value be qBh
#' (defined similarly for C TN). Then, for a pval thresh of .05 and a qval
#' thresh of .1, the function classifies each variant as either
#' (1) a shared signal (if one of pBh or pBc < .05 and
#' one of qBh or qBc < .1), (2) an HS only signal (if qBh < .1 and pBc >= .05),
#' (3) a c only signal (if qBc < .1 and pBch >= .05),
#' or (4) null (if none of the other conditions are met).
#'
#' @param sim_df Simulated dataframe in the formated produced
```

```r
#' by \code{generate_simulated_data}
#' @param n Total number of individuals per site
#' @param sites Total number of sites
#' @param pval_thresh pvalue threshold for individual regression coefficients
#' to declare significance in cases above.
#' @param qval_thresh qvalue threshold for individual regression coefficients
#' to declare significan in cases above.
#'
#' @return Dataframe with site numbers and signal classifications
#' @export
#'
#' @examples
calc_pallares_results <- function(
  sim_df, n, sites, pval_thresh = .05, qval_thresh = .1
) {

  site_vec <- c()
  test_result_vec <- c()

  # Make sure that the sim_df is sorted
  sim_df <- sim_df %>% arrange(site)
  cond_c_pval_vec <- numeric(sites)
  cond_hs_pval_vec <- numeric(sites)

  for(i in c(1:sites)) {

    # slice instead of filter for speed
    site_df <- sim_df %>% slice(c((1 + (i - 1) * n):(i * n)))
    bbn_model <- aod::betabin(
      cbind(alt_reads, total_reads - alt_reads) ~ condition, ~1, data=site_df
    )
    mod_sum <- aod::summary(bbn_model)
    cond_c_pval_vec[i] <- attributes(mod_sum)$Coef['conditionC', 4]
    cond_hs_pval_vec[i] <- attributes(mod_sum)$Coef['conditionHS', 4]

  }

  hs_pval_test <- cond_hs_pval_vec < pval_thresh
  c_pval_test <- cond_c_pval_vec < pval_thresh

  hs_qval_test <- qvalue::qvalue(
    p = cond_hs_pval_vec, fdr.level = qval_thresh
  )$significant
  c_qval_test <- qvalue::qvalue(
    p = cond_c_pval_vec, fdr.level = qval_thresh
  )$significant

  # shared signal if the p-value < .05 in one environment and < 10% FDR in the other
  shared_test <- (hs_pval_test & c_qval_test) | (c_pval_test & hs_qval_test)

  # hs only if the fdr test passes for hs but the pval test fails for c
  hs_only_test <- !c_pval_test & hs_qval_test
```

```
  # c only if the fdr test passes for c but the pval test fails for hs
  c_only_test <- !hs_pval_test & c_qval_test

  results_df <- data.frame(
    site = c(1:sites),
    shared_test = shared_test,
    hs_only_test = hs_only_test,
    c_only_test = c_only_test
  )

  results_df <- results_df %>%
    mutate(
      classification = case_when(
        shared_test ~ "shared",
        hs_only_test ~ "hs_only",
        c_only_test ~ "c_only",
        TRUE ~ "null"
      )
    )

  results_df <- results_df %>% select(c(site, classification))
  return(results_df)

}
```

Below is a simple example of how to use this code:

```
sim_df3 <- generate_simulated_dataset(
  n = 1000,
  sites = 50,
  maf_dist = c(.25, .4),
  prob_both_null = 0.5,
  prob_hs_only = 0.5,
  prob_c_only = 0.0,
  prob_same = 0.0,
  prob_amp = 0.0,
  cx = -0.25,
  nb_params_dist = list(list(t = 25, p = .4), list(t = 10, p = .3))
)$sim_df

results <- calc_pallares_results(sim_df3, 1000, 50)
```

Below is a sample of the code that could be used in order to generate the distribution of minor allele frequencies.

```
# sample 5K of the sites. This number could be increased or decreased
# note that the distinct call is necessary because there are a very small number
# of duplicated entries
alt <- fread('data/29Jun20_merged_alt_counts_allCHR.txt') %>%
  distinct(site, .keep_all = TRUE) %>%
  sample_n(5000)
both <- fread('data/29Jun20_merged_both_counts_allCHR.txt') %>%
  filter(site %in% alt$site) %>%
  distinct(site, .keep_all = TRUE)
```

```
alt_counts <- alt %>% select(-site) %>% rowSums()
both_counts <- both %>% select(-site) %>% rowSums()

alt_count_df <- data.frame(site=alt$site, alt_count = alt_counts)
both_count_df <- data.frame(site=both$site, total_count = both_counts)

count_df <- alt_count_df %>% inner_join(both_count_df, by = c("site"))
count_df <- count_df %>% mutate(alt_freq = alt_count / total_count)
# now, count_df$alt_freq contains a distribution over alt allele frequencies
```

Below is a sample of the code that could be used to generate the negative binomial parameters required for determining the number of reads.

```
data_colnames <- colnames(both)
variants <- data_colnames[data_colnames != 'site']
nb_params_list <- list()
variant_ind <- 1

for(variant in variants) {

  variant_reads <- both[[variant]]
  variant_fit <- MASS::fitdistr(
    x = variant_reads, densfun = "negative binomial", lower = c(1, 0)
  )
  variant_params_list <- list(
    t = variant_fit$estimate["size"], p = variant_fit$estimate["mu"]
  )
  nb_params_list[[variant_ind]] <- variant_params_list
  variant_ind <- variant_ind + 1

}

# now, nb_params_list has a set of parameters to be sampled from
```