Dissertation

Sergio Pérez-Limón

3/10/2023

Table of contents

Preface **1**

1 Introduction **3**

I MAGIC **5**

Running Code 7

2 Estimate mosaic metrics MAGIC **9**

3 What is the size of the recombination chunks in the MEMA population? **21**

4 What is the average size of the "same genotype" chunks in the parental magic haplotype? **27**

II CML **45**

Running Code 47

5 2023_Highland_experiment_design **49**

6 PT **63**

6.1 Running Code 63

7 AMF **65**

7.1 Running Code 65

8 Summary **67**

References **69**

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

Chapter 1

Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

Part I

MAGIC

Sergio Pérez-Limón

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

You can add options to executable code like this

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).

Chapter 2

Estimate mosaic metrics MAGIC

There are a few questions that we can make out of the MAGIC population:

- Is there any residual populations structure in the MAGIC population?
- Can we improve the parenta haplotype call in the MAGIC population if we increae the number of markers?
  - How does the choromosomic mosaic looks in the MAGIC population?
  - What's the median and maximum chromosomic chunk size in the MAGIC population?
  - What's the median and maximum "same genotype" chunk in a pairwise comparison of the MAGIC parents?
  - What is the minimal number of markers that we need to obtain information similar to the total of 8.2 M markers?

Logic: We hypothesize that most of the NAs or mistakes in the call of the parent in the MAGIC populations are because {qtl2} can't tell apart one parent from the other with enough confidence (with $\alpha = 0.5$) because the marker density is not enough to tell a parent from the other. In this case, increasing the number of markers between each chunk will help us only if the maximun size of the chunks with "same genotype" in the parents information is smaller than the recombination chunk size in the MAGIC population. IN other words, we're adding more relevant information in each of these chunks that would allow us to differentiate one parent from the other.

Loading libraries

```r
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
v ggplot2 3.4.1      v purrr   1.0.1
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.3.0      v stringr 1.5.0
v readr   2.1.3      v forcats 0.5.2
```

```
Warning: package 'ggplot2' was built under R version 4.2.2

Warning: package 'tidyr' was built under R version 4.2.2

Warning: package 'readr' was built under R version 4.2.2

Warning: package 'purrr' was built under R version 4.2.2

Warning: package 'dplyr' was built under R version 4.2.2

Warning: package 'stringr' was built under R version 4.2.2

Warning: package 'forcats' was built under R version 4.2.2
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```r
library(furrr)
```

```
Warning: package 'furrr' was built under R version 4.2.2

Loading required package: future

Warning: package 'future' was built under R version 4.2.2
```

```r
plan(multisession)
library(arrow)
```

```
Warning: package 'arrow' was built under R version 4.2.2

Attaching package: 'arrow'


The following object is masked from 'package:utils':

    timestamp
```

```r
library(tidymodels)
```

```
-- Attaching packages -------------------------------------- tidymodels 1.0.0 --
v broom        1.0.3          v rsample      1.1.1
v dials        1.1.0          v tune         1.0.1
v infer        1.0.4          v workflows    1.1.2
v modeldata    1.1.0          v workflowsets 1.0.0
v parsnip      1.0.3.9000     v yardstick    1.1.0
```

```
v recipes      1.0.4

Warning: package 'broom' was built under R version 4.2.2

Warning: package 'dials' was built under R version 4.2.2

Warning: package 'infer' was built under R version 4.2.2

Warning: package 'modeldata' was built under R version 4.2.2

Warning: package 'recipes' was built under R version 4.2.2

Warning: package 'rsample' was built under R version 4.2.2

Warning: package 'tune' was built under R version 4.2.2

Warning: package 'workflows' was built under R version 4.2.2

Warning: package 'yardstick' was built under R version 4.2.2

-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
* Use suppressPackageStartupMessages() to eliminate package startup messages
```

```r
library(qtl2)
```

```
Warning: package 'qtl2' was built under R version 4.2.2

Attaching package: 'qtl2'

The following object is masked from 'package:readr':

    read_csv
```

```r
library(tidymodels)
library(finetune)
```

```
Warning: package 'finetune' was built under R version 4.2.2
```

```r
library(reshape2)
```

```
Attaching package: 'reshape2'

The following object is masked from 'package:tidyr':
    smiths
```

```
library(tidymodels)
```

Clean CrossObject: Remove weid markers and individuals based on their total number of recombinations and LODerror

```
MEMA_DATA <-
  read_cross2(
    "C:/Users/sergi/Documents/SAWERS LAB/MAGIC/genetic_mapping/MEMA_CTRL_wo_weird.ya
    )
```

```
Warning in drop_incomplete_markers(output): Omitting 4738 markers that are not
in both genotypes and maps
```

```
Warning in check_cross2(output): Physical map out of order on chr 1, 2, 3, 4,
5, 6, 7, 8, 9, 10
```

```
snp.info.founders <- readr::read_csv(
  "C:/Users/sergi/Documents/SAWERS LAB/MAGIC/genetic_mapping/snp.info.founders.csv"
  ) %>%
  filter(!is.na(snp))
```

```
Rows: 19654 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): snp
dbl (3): chr, pos, sdp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# recombinations
#########################################################################################

genmap_MEMA <- MEMA_DATA$gmap

# Estimate genotype probabilities
set.seed(100)
genoprob_MEMA <-
  calc_genoprob(
    cross = MEMA_DATA,
    map = genmap_MEMA,
    error_prob=0.002) %>%
  clean_genoprob ()
```

```
# Estimate the genotypes with maximum marginal probabilities
set.seed(100)
geno_maxprob_MEMA <-
  maxmarg(
    probs = genoprob_MEMA,
    minprob = 0.95
    ) #assigns founder code to each marker/pos

# Estimate the genotypes with maximum marginal probabilities with letters
set.seed(100)
geno_maxprob_letters_MEMA <- maxmarg(
  probs= genoprob_MEMA,
  minprob = 0.95,
  return_char = TRUE
  )
```

Estimate the error lod probability for any marker for any family. Positive LOD scores suggest that there might be a mistake in the genotype probability. I'm estimating the proportion of genotype calls in each marker whose errorLOD > 0 and estimate the top 5 percentile values of the proportion distribution and remove the markers whose proportion belong to this group.

```
# Calculate genotyping error LOD scores
set.seed(100)
errorlod_MEMA <-
  calc_errorlod(
    cross = MEMA_DATA,
    probs = genoprob_MEMA
    )

errorlod_MEMA <- do.call("cbind", errorlod_MEMA)

# matrix to df
errorlod_MEMA_df <-
  errorlod_MEMA %>%
  as_tibble(rownames = "family")

# Estimate the proportion of LOD error > 0 in every marker
errorlod_MEMA_prop_marker <-
  errorlod_MEMA_df %>%
  pivot_longer(-family) %>%
  filter(!is.infinite(value)) %>%
  group_by(name) %>%
  summarise(prop = sum(value >= 0)/dim(errorlod_MEMA_df)[1])
```

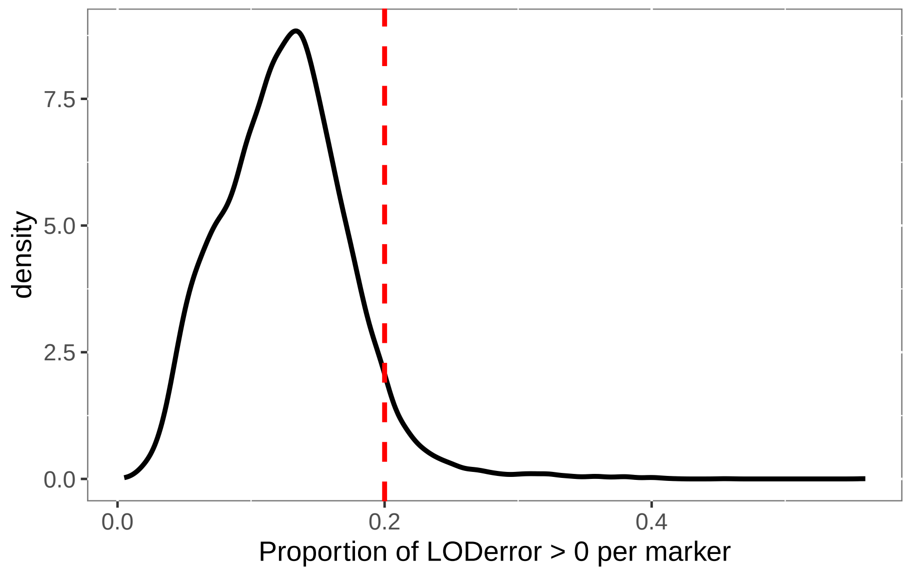
```
errorlod_MEMA_prop_marker
```

```
# A tibble: 14,176 x 2
   name         prop
   <chr>       <dbl>
 1 1_100088401 0.125
 2 1_100224800 0.115
 3 1_100225275 0.135
 4 1_100229005 0.11
 5 1_100344399 0.13
 6 1_100344420 0.13
 7 1_100344944 0.145
 8 1_100384962 0.13
 9 1_100385294 0.16
10 1_100388059 0.135
# ... with 14,166 more rows
```

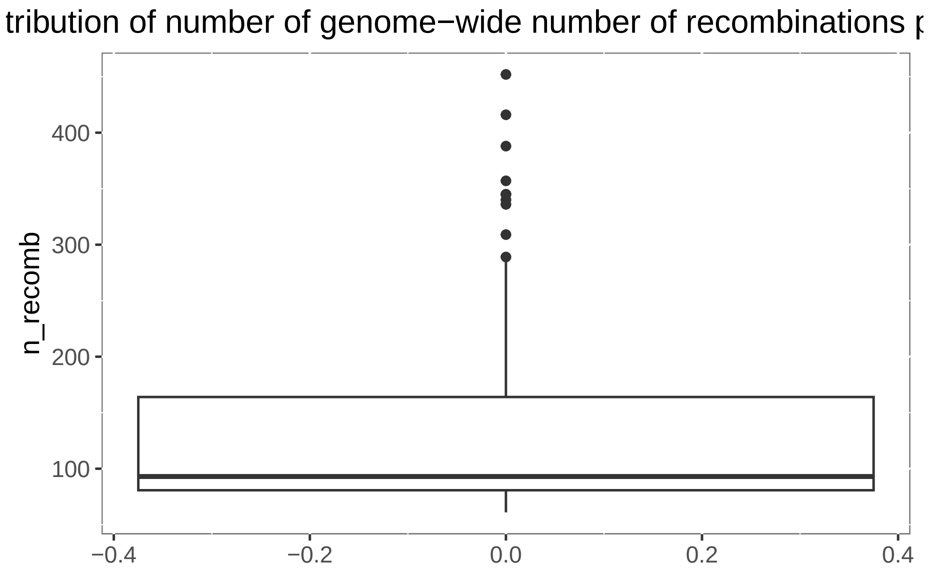
```
# Estimate the 95 percentile of the LODerror proportion distribution
errorlod_95_perc <- quantile(errorlod_MEMA_prop_marker$prop, 0.95)

errorlod_95_perc
```

```
95%
0.2
```

```
# Remove all the markers with prop >=0.2

errorlod_MEMA_prop_marker %>%
  ggplot(data =., aes(x = prop)) +
  geom_density(linewidth = 1) +
  xlab("Proportion of LODerror > 0 per marker")  +
  geom_vline(
    aes(xintercept = errorlod_95_perc),
    color = "red",
    linetype = "dashed",
    linewidth = 1
    ) +
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    legend.position = "top"
    )
```



```
# Select markers in the 95 percentile to remove
errorlod_marker_remove <- errorlod_MEMA_prop_marker %>%
  filter(prop >= errorlod_95_perc) %>%
  .$name

#############################################
# Count number of recombinations per family #
#############################################

# Estimate the total number of recombinations per family
set.seed(100)
n_recomb_family_MEMA <-
  count_xo(geno = geno_maxprob_MEMA) %>%
  as_tibble(rownames = "family") %>%
  pivot_longer(-family) %>%
  group_by(family) %>%
  summarise(n_recomb = sum(value)) %>%
  arrange(desc(n_recomb))

# Identify outliers in the distribution of genomewide recombination events
n_recomb_family_MEMA %>%
  ggplot(data =., aes(y = n_recomb)) +
  geom_boxplot() +
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    plot.title = element_text(hjust = 0.5)
    ) +
  ggtitle("Distribution of number of genome-wide number of recombinations per famil
```



```
# Outliers total number of recomb
outliers_lots_recomb_MEMA <-
  boxplot(n_recomb_family_MEMA$n_recomb, plot = F)$out

# non outlier families
family_normal_recomb <-
  n_recomb_family_MEMA %>%
  filter(! n_recomb %in% outliers_lots_recomb_MEMA) %>%
  .$family

MEMA_DATA_wo_weird <-
  MEMA_DATA %>%
  drop_markers(errorlod_marker_remove) %>%
  subset(x =., ind = family_normal_recomb)
```

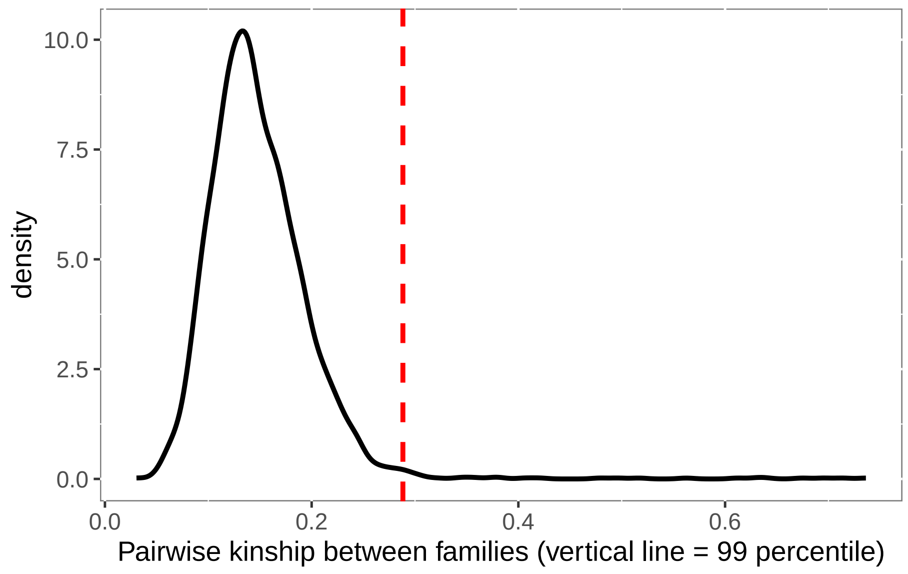
Is there any population structure in the MEMA population?

Approach: I'm going to estimate the kinship matrix using the overall option and a decomposition of eigenvalues of the same matrix and detect population structure.

```r
# Estimate genotype probabilities
set.seed(100)
genoprob_wo_weird_MEMA <-
  calc_genoprob(
    cross = MEMA_DATA_wo_weird,
    map = genmap_MEMA,
    error_prob=0.002) %>%
  clean_genoprob ()

# Estimate Allele probabilities
set.seed(100)
allele_prob_MEMA <-
  genoprob_to_alleleprob(genoprob_wo_weird_MEMA)

# Estimate kinship matrix
set.seed(100)
kinship_wo_weird_MEMA <- calc_kinship(
  probs = allele_prob_MEMA,
  type = "overall"
  )

# Estimate the eigenvalue decomposition of the kinship matrix
set.seed(100)
eigen_kinship_MEMA <-
  decomp_kinship(kinship_wo_weird_MEMA)

# correlation matrix of the kinship matrix
set.seed(100)
corr_kindhip_MEMA <-
  scale_kinship (kinship_wo_weird_MEMA)

# Pairwise correlation of families
pairwise_geno_corr_MEMA <- corr_kindhip_MEMA %>%
  as_tibble(rownames = "family") %>%
  pivot_longer(-family) %>%
  filter(family != name) %>%
  mutate(code = map2_chr(
    .x = family,
    .y = name,
    .f = ~ paste0(.x, .y) %>% str_split("") %>% unlist() %>% sort() %>% paste(collapse = "")
    )) %>%
  group_by(code) %>%
  filter(row_number() == 1) %>%
  ungroup() %>%
  select(-code)

# Estimate 99 percentile of pairwise correlation
geno_corr_99_perc <- quantile(pairwise_geno_corr_MEMA$value, 0.99)

geno_corr_99_perc
```

```
      99%
0.2882856
```

```
# plot the distribution of the pairwise correlation between families
pairwise_geno_corr_MEMA %>%
  ggplot(data =., aes(x = value)) +
  geom_density(linewidth = 1) +
  xlab("Pairwise kinship between families (vertical line = 99 percentile)")  +
  geom_vline(
    aes(xintercept = geno_corr_99_perc),
    color = "red",
    linetype = "dashed",
    linewidth = 1
    ) +
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    legend.position = "top"
    )
```



```r
# Identify family pairwise correlation values in the 99 percentile
familyes_high_corr <-
  pairwise_geno_corr_MEMA %>%
  filter(value >= geno_corr_99_perc) %>%
  arrange(desc(value))

familyes_high_corr
```

```
# A tibble: 27 x 3
   family  name    value
   <chr>   <chr>   <dbl>
 1 MEMA210 MEMA211 0.736
 2 MEMA245 MEMA244 0.714
 3 MEMA162 MEMA163 0.695
 4 MEMA128 MEMA127 0.675
 5 MEMA307 MEMA308 0.638
 6 MEMA033 MEMA034 0.631
 7 MEMA133 MEMA132 0.612
 8 MEMA055 MEMA054 0.563
 9 MEMA252 MEMA253 0.517
10 MEMA168 MEMA078 0.496
# ... with 17 more rows
# Select families to drop that are in the 99th percentile but doesn't have a phenoty

genotyped_fam_MEMA <-
  MEMA_DATA_wo_weird$geno$`1` %>% rownames()

families_no_pheno <-
  MEMA_DATA_wo_weird$pheno %>%
  rownames() %>%
  setdiff(genotyped_fam_MEMA, .)

families_no_pheno_high_corr <-
  familyes_high_corr %>%
  rowid_to_column() %>%
  rename(cor = value) %>%
  pivot_longer(c(family, name)) %>%
  count(value) %>%
  arrange(desc(n)) %>%
  filter(value %in% families_no_pheno) %>%
  .$value

# Drop families with high correlation and not phenotyped: c("MEMA163", "MEMA210", "M

MEMA_DATA_wo_weird2 <- MEMA_DATA_wo_weird %>%
  subset(ind = setdiff(genotyped_fam_MEMA, families_no_pheno_high_corr))
```

In the absence of population structure we can expect a correlation between families of 1/8. Any significant deviation from this number would imply that there is residual population structure in the population. The distribution of the pairwise correlation values for the MEMA family is right skewed with a median value of ~ 0.14, close to 0.125 (1/8) expected. This value is inflated a little bit by weird families that have a lot of correlation. We hypothesize that the high correlation values for these families (for instance families MEMA210-MEMA211; cor = 0.736) was the result of either mistakes in the development of families: Someone mistakenly grabbed seed from an incorrect envelope that was planted and generated a very closely relted family, or it could also be the result of mistakes in collecting tissue from plants for genotyping. Nevertheless, the results show that the median pairwise correlation doesn't deviate that much from the expected under the hypothesis of no population structure, so we can conclude that there is a reduced population structure in the MEMA pop. We're going to remove highly correlated individuals for whom we don't have phenotypic data.

Chapter 3

What is the size of the recombination chunks in the MEMA population?

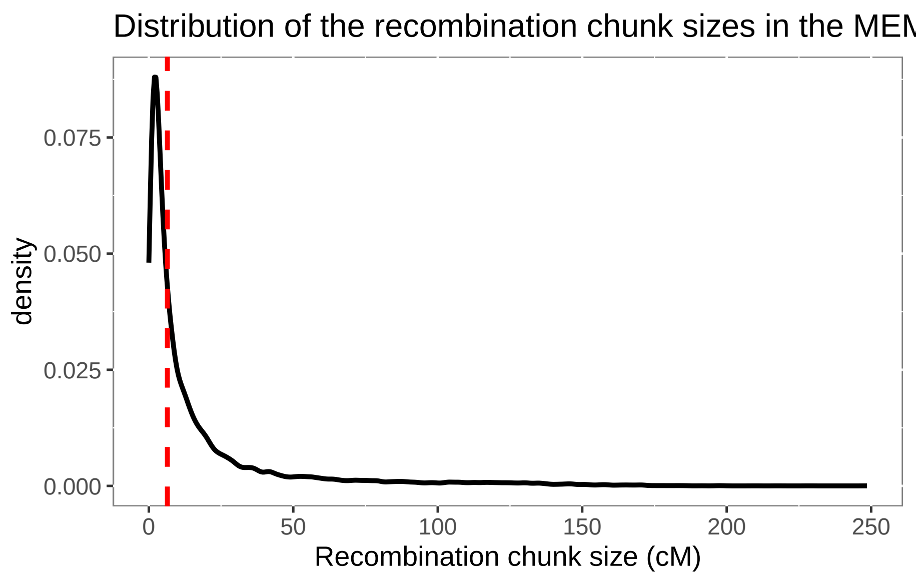
```
# Estimate genotype probabilities
set.seed(100)
genoprob_wo_weird2_MEMA <-
  calc_genoprob(
    cross = MEMA_DATA_wo_weird2,
    map = genmap_MEMA,
    error_prob=0.002) %>%
  clean_genoprob ()

# Estimate Allele probabilities
set.seed(100)
geno__wo_weird2_MEMA <-
  maxmarg(
    probs = genoprob_wo_weird2_MEMA,
    minprob = 0.95
    )

# Guess the phase of each family in each chromosome
phase_MEMA <- guess_phase(MEMA_DATA_wo_weird2, geno__wo_weird2_MEMA)

# genetic map
genmap_MEMA <-
  MEMA_DATA_wo_weird2$gmap %>%
  map_df(.x = ., .f = ~ .x %>% enframe(name = "marker", value = "g_pos"))

# markers at the end of each chromosome
genmap_chr_end <-
  genmap_MEMA %>%
  separate(marker, into = c("chr", "p_pos"), sep = "_", remove = F) %>%
  mutate(
    chr = as.integer(chr),
    p_pos = as.integer(p_pos)
    ) %>%
  arrange(chr, p_pos) %>%
  group_by(chr) %>%
  filter(row_number() == max(row_number())) %>%
  mutate(p_pos = p_pos/1e6) %>%
  ungroup() %>%
  rename(chr_end = p_pos)


# Identify the recombination breakpoints in the MEMA population
set.seed(100)
recomb_location_MEMA <-
  locate_xo(phase_MEMA, map = MEMA_DATA_wo_weird2$gmap) %>%
  map_df(
    .x = .,
    .f = ~ map_df(
      .x = .,
      .f = ~ enframe(.x, name = NULL, value = "location"),
      .id = "family"
      ),
    .id = "chr"
  ) %>%
  mutate(marker = find_marker(MEMA_DATA$gmap, chr = chr, pos = location)) %>%
  mutate(chr = as.integer(chr))

# Estimate the size of the recombination chunks in the MEMA population
recomb_chunk_size_MEMA <-
  recomb_location_MEMA  %>%
  group_by(family, chr) %>%
  nest() %>%
  mutate(data = map(
    .x = data,
    .f = ~ .x %>% add_row(location = NA, marker = NA)
    )) %>%
  ungroup() %>%
  unnest(c(data), keep_empty = T) %>%
  group_by(family, chr) %>%
  mutate(
    pos = gsub("^\\d{1,2}_", "", marker) %>% as.integer() %>% "/"(1e6),
    chr = as.integer(chr)
    ) %>%
  mutate(
    start = lag(pos, default = 0),
    end = pos
    ) %>%
  left_join(genmap_chr_end %>% select(-marker), by = "chr") %>%
  mutate(end = ifelse(is.na(pos), chr_end, end))  %>%
  select(family, chr, location, marker, start, end) %>%
  ungroup() %>%
  mutate(chunk_size = end - start)  %>%
  mutate(
    p_pos_start = find_marker(MEMA_DATA$gmap, chr = chr, pos = start),
    p_pos_end = find_marker(MEMA_DATA$gmap, chr = chr, pos = end)) %>%
  mutate(
    across(contains("p_pos"), ~ gsub("\\d{1,2}_", "", .) %>% as.integer() %>% "/"(1e6))
  ) %>%
  mutate(p_chunk_size = p_pos_end - p_pos_start) %>%
  filter(chunk_size > 0 & p_chunk_size > 0)

recomb_chunk_size_MEMA
```

```
# A tibble: 22,217 x 10
   family    chr location marker     start   end chunk~1 p_pos~2 p_pos~3 p_chu~4
   <chr>   <int>    <dbl> <chr>      <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1 MEMA156     1     14.0 1_7987929   0     7.99   7.99     1.74    4.91   3.17
 2 MEMA156     1     34.1 1_18969916  7.99 19.0   11.0      4.91   11.3    6.40
 3 MEMA156     1     65.8 1_42129633 19.0  42.1   23.2     11.3    23.6   12.3
 4 MEMA156     1     97.8 1_71985242 42.1  72.0   29.9     23.6    50.5   26.9
 5 MEMA156     1     98.6 1_72903601 72.0  72.9    0.918   50.5    53.8    3.24
 6 MEMA156     1     99.4 1_74159005 72.9  74.2    1.26    53.8    54.8    1.01
 7 MEMA156     1    101.  1_78116605 74.2  78.1    3.96    54.8    57.4    2.69
 8 MEMA156     1    104.  1_80559358 78.1  80.6    2.44    57.4    58.3    0.826
 9 MEMA156     1    105.  1_81673169 80.6  81.7    1.11    58.3    59.0    0.751
10 MEMA156     1    105.  1_82472265 81.7  82.5    0.799   59.0    59.4    0.420
# ... with 22,207 more rows, and abbreviated variable names 1: chunk_size,
#   2: p_pos_start, 3: p_pos_end, 4: p_chunk_size
```

```
recomb_med_max_size <-
  recomb_chunk_size_MEMA %>%
  summarise(
    min_chunk_size = min(chunk_size),
    median_chunk_size = median(chunk_size),
    max_chunk_size = max(chunk_size)
    )

recomb_med_max_size
```

```
# A tibble: 1 x 3
  min_chunk_size median_chunk_size max_chunk_size
           <dbl>             <dbl>          <dbl>
1         0.0180              6.43           249.
```

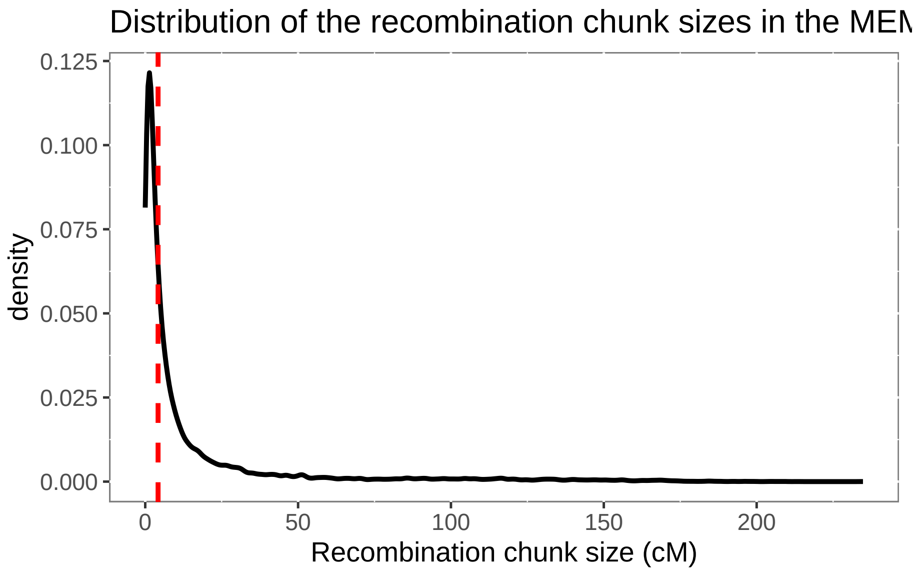
```
recomb_chunk_size_MEMA %>%
  ggplot(data =., aes(x = chunk_size)) +
  geom_density(linewidth = 1) +
  xlab("Recombination chunk size (cM)")  +
  geom_vline(
    data = recomb_med_max_size,
    aes(xintercept = median_chunk_size),
    color = "red",
    linetype = "dashed",
    linewidth = 1
    ) +
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    legend.position = "top"
    ) +
  ggtitle("Distribution of the recombination chunk sizes in the MEMA population")
```



```
recomb_med_max_p_size <-
  recomb_chunk_size_MEMA %>%
  summarise(
    min_chunk_size_p = min(p_chunk_size),
    median_chunk_size_p = median(p_chunk_size),
    max_chunk_size_p = max(p_chunk_size)
    )

recomb_med_max_p_size
```

```
# A tibble: 1 x 3
  min_chunk_size_p median_chunk_size_p max_chunk_size_p
             <dbl>               <dbl>            <dbl>
1       0.00000800                4.20             235.
```

```
recomb_chunk_size_MEMA %>%
  ggplot(data =., aes(x = p_chunk_size)) +
  geom_density(linewidth = 1) +
  xlab("Recombination chunk size (cM)")  +
  geom_vline(
    data = recomb_med_max_p_size,
    aes(xintercept = median_chunk_size_p),
    color = "red",
    linetype = "dashed",
    linewidth = 1
    ) +
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    legend.position = "top"
    ) +
  ggtitle("Distribution of the recombination chunk sizes in the MEMA population")
```



The median recombination chunk size is ~ 4.20 MB, but they can go from almos 0 to >200 MB. It is kinda weird. The most probable thing is that those very small chunks are just mistakes in the haplotype calling and therefore the chunks are very small. Is also worth to mention that the missing information in the parentall haplotype calls seems to be located when one haplotype ends to when the other starts, so it seems that with this marker density (~ 14K markers), rqtl2 can't confidently assign a parental haplotype in these transition regions.

Chapter 4

What is the average size of the “same genotype” chunks in the parental magic haplotype?

We hypothesize that rqtl2 is having trouble in calling parental haplotypes with confidence because there are chromosomic chunks where two or more parents have the same genotype, and with the actual marker density rqtl2 cannot tell apart one from the other. So we want to assess if including a greater number of markers (that come from WGS) can reduce the size of this chunks, therefore becoming easier to identify one parent from the other. In the other hand, it might be the case that even with a greater marker density the chunks stay about the same size, so including a genotype strategy to increase the marker density in the MEMA families will not be helpfull at all.

```
# Import CHIP data

mema_par_hap_raw <-
  read_delim(
    "C:/Users/sergi/Documents/SAWERS LAB/MAGIC/mosaic_estimates/gen_mosaic_estimates/magic_par
    delim = "\t",
    escape_double = FALSE,
    trim_ws = TRUE
    )
```

```
Rows: 23645 Columns: 19
-- Column specification --------------------------------------------------------
```

```
Delimiter: "\t"
chr (11): rs#, alleles, strand, m_zc, m_nt, m_rv, m_gd, m_tb, m_mu, m_ja, m_pt
dbl  (2): chrom, pos
lgl  (6): assembly#, center, protLSID, assayLSID, panelLSID, QCcode

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
colnames_WGS_data <- arrow::open_dataset(
  sources = "C:/Users/sergi/Documents/SAWERS LAB/MAGIC/mosaic_estimates/gen_mosaic_e
  )$schema$names

parent_chip_MEMA <-  c("GD", "JL", "MS", "NT", "PT", "RV", "TB", "ZC")


mema_par_hap <-
  mema_par_hap_raw %>%
  select(marker = `rs#`, chrom, pos, contains("m_")) %>%
  select(marker:pos, sort(names(.))) %>%
  rename_with(.cols = contains("m_"), ~ c("GD", "JL", "MS", "NT", "PT", "RV", "TB",
  pivot_longer(-c(marker:pos), names_to = "parental", values_to = "geno")

chip_parent_comp_mema <-
  crossing(p1 = parent_chip_MEMA, p2 = parent_chip_MEMA) %>%
  filter(p1 != p2) %>%
  mutate(code = map2_chr(
    .x = p1,
    .y = p2,
    .f = ~ paste0(.x, .y) %>% str_split("") %>% unlist() %>% sort() %>% paste(collap
    )) %>%
  group_by(code) %>%
  filter(row_number() == 1) %>%
  ungroup() %>%
  select(-code) %>%
  left_join(mema_par_hap %>% rename(geno1 = geno), by = c("p1" = "parental")) %>%
  left_join(mema_par_hap %>% rename(geno2 = geno), by = c("p2" = "parental", "marker

chip_parent_chunk_info <-
  chip_parent_comp_mema %>%
  mutate(comp_group = paste(p1, p2, sep ="-")) %>%
  select(-c(p1, p2)) %>%
  group_by(comp_group) %>%
  nest() %>%
  mutate(chunk_info = future_map(
    .x = data,
    .f = ~ .x %>%
      group_by(chrom) %>%
      rowid_to_column() %>%
      filter(geno1 == geno2) %>%
      mutate(
        dif_inicio = lead(rowid) - rowid,
        dif_final =  lag(dif_inicio)
        ) %>%
      filter(dif_inicio == 1 | dif_final == 1) %>%
      mutate(block = ifelse(dif_inicio == 1, NA, rowid)) %>%
      fill(block, .direction = "up") %>%
      group_by(block) %>%
      mutate(chunck_n_marker = n()) %>%
      filter(rowid == min(rowid) | rowid == max(rowid)) %>%
      group_by(chrom, block) %>%
      summarise(
        chunck_n_marker = first(chunck_n_marker),
        chunk_size = diff(pos),
        chunck_name = paste(first(rowid), last(rowid), sep = "-"),
        spans = paste(round(first(pos)/1e6, 2), round(last(pos)/1e6, 2), sep ="-")
        ) %>%
      ungroup()
  ))
```

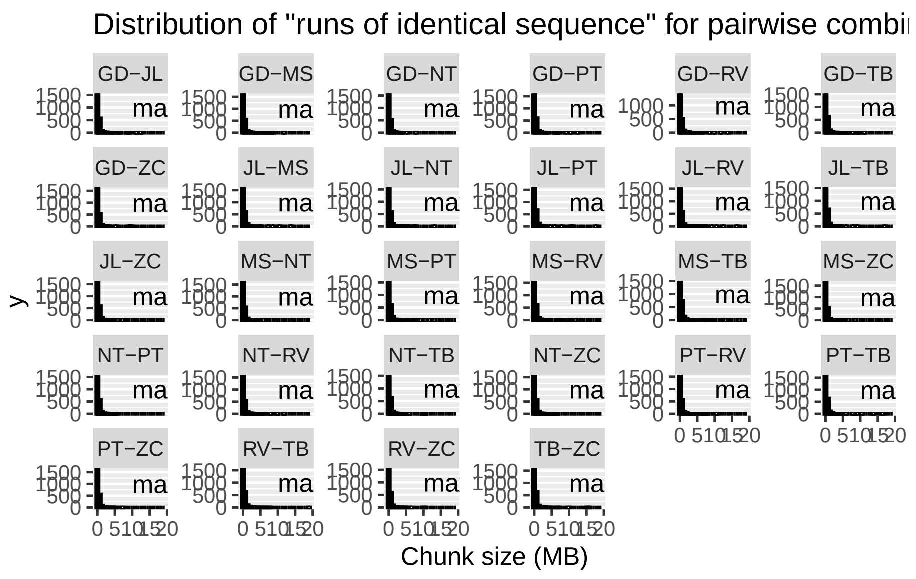
```
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
```

```
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
`summarise()` has grouped output by 'chrom', 'block'. You can override using
the `.groups` argument.
```

```r
chip_parent_chunk_info_pre <-
  chip_parent_chunk_info  %>%
  select(comp_group, chunk_info) %>%
  ungroup() %>%
  unnest(chunk_info) %>%
  mutate(chunk_size = round(chunk_size/1e6, 2)) %>%
  filter(chunk_size != 0)

chip_parent_chunk_info_pre %>%
  group_by(comp_group) %>%
  summarise(
    med_size = median(chunk_size),
    max_size = max(chunk_size)
    ) %>%
  summarise(
    med_size = median(med_size),
    max_size = median(max_size)
  )
```

```
# A tibble: 1 x 2
  med_size max_size
     <dbl>    <dbl>
1     0.19     12.1
```

```
  chip_parent_chunk_mema_label <- chip_parent_chunk_info_pre %>%
    group_by(comp_group) %>%
    summarise(
      label = sapply(c("min", "max", "median"), do.call, list(x = chunk_size))  %>%
        round(., 2) %>%
        paste(names(.), ., sep = " = ") %>%
        paste(., collapse = "\n"),
      y = 1000,
      chunk_size = 10
    )

  chip_parent_chunk_info_pre %>%
    ggplot(data =., aes(x = chunk_size, fill = comp_group )) +
    geom_histogram(color = "black", linewidth = 0.75, bins = 30) +
    geom_vline(
      data = . %>% group_by(comp_group ) %>% summarise(chunk_size = median(chunk_size)),
      aes(xintercept = chunk_size),
      linetype = "dashed",
      linewidth = 0.75
    ) +
    geom_text(
      data = chip_parent_chunk_mema_label,
      aes(label = label, x = chunk_size, y = y, hjust = "bottom"),
    ) +
    facet_wrap(. ~ comp_group, scales = "free_y") +
    xlab("Chunk size (MB)") +
    theme(
      legend.position = "none",
      panel.grid.major.x = element_blank(),
      panel.grid.minor.x = element_blank(),
      ) +
    ggtitle('Distribution of "runs of identical sequence" for pairwise combination of
```



In general the maximum size of the “same genotype” chunks seems to be 8-16 MB depending on the pairwise cross that is being analyzed. We expect to see a reduction in the size by using WGS (higher marker density)

```
wgs_mema_parquet_source <-
  "C:/Users/sergi/Documents/SAWERS LAB/MAGIC/mosaic_estimates/gen_mosaic_estimates/p

colnames_WGS_data <-
  arrow::open_dataset(sources = wgs_mema_parquet_source)$schema$names

WGS_parents_raw <-
  colnames_WGS_data[-c(1:11)] %>%
  crossing(p1 =., p2 = .) %>%
  filter(p1 != p2) %>%
  mutate(code = map2_chr(
    .x = p1,
    .y = p2,
    .f = ~ paste0(.x, .y) %>% str_split("") %>% unlist() %>% sort() %>% paste(collap
    )) %>%
  group_by(code) %>%
  filter(row_number() == 1) %>%
  ungroup() %>%
  select(-code) %>%
  mutate(
    cols = map2(
      .x = p1,
      .y = p2,
      .f = ~ c(colnames_WGS_data[c(1, 3, 4)], .x, .y)
    )) %>%
  mutate(comp_group_label = paste0(p1, "_", p2))


downsize_scheme <- tibble(
  n_marker = c(5000, 10000, 20000, 50000, 100000, 500000, 1e6, 3e6, 5e6, 8125930),
  code = c("5k", "10k", "20k", "50K", "100K", "500K", "1M", "3M", "5M", "8.2M")
  ) %>%
  mutate(prop = n_marker/8125930)

# for (j in 10) {
#
#   for (i in 1:dim(WGS_parents_raw)[1]) {
#
#     wgs_data_pre <- read_parquet(file = "parents_WGS_hap.parquet", col_select = all_of(WGS_p
#       rename_with(~c("marker", "chrom", "pos", "geno1", "geno2"))
#
#     if (j == dim(downsize_scheme)[1]) {
#
#       wgs_magic_chunks_data_pre <- wgs_data_pre
#
#     } else {
#
#       set.seed(100)
#       wgs_data_split <- initial_split(wgs_data_pre, prop = downsize_scheme$prop[j], strata =
#       wgs_magic_chunks_data_pre <- training(wgs_data_split)
#
#     }
#
#     wgs_magic_chunks <-
#       wgs_magic_chunks_data_pre %>%
#       group_by(chrom)  %>%
#       rowid_to_column() %>%
#       filter(geno1 == geno2) %>%
#       mutate(
#         dif_inicio = lead(rowid) - rowid,
#         dif_final =  lag(dif_inicio)
#         ) %>%
#       filter(dif_inicio == 1 | dif_final == 1) %>%
#       mutate(block = ifelse(dif_inicio == 1, NA, rowid)) %>%
#       fill(block, .direction = "up") %>%
#       group_by(block) %>%
#       mutate(chunck_n_marker = n()) %>%
#       filter(rowid == min(rowid) | rowid == max(rowid)) %>%
#       group_by(chrom, block) %>%
#       summarise(
#         chunck_n_marker = first(chunck_n_marker),
#         chunk_size = diff(pos),
#         chunck_name = paste(first(rowid), last(rowid), sep = "-"),
#         spans = paste(round(first(pos)/1e6, 2), round(last(pos)/1e6, 2), sep ="-")
#         ) %>%
#       ungroup() %>%
#     mutate(chunk_size = round(chunk_size/1e6, 2)) %>%
#     filter(chunk_size > 0)
#
#   SINK <- paste0(WGS_parents_raw$comp_group_label[i], "_", downsize_scheme$code[j]
#
#   write_parquet(wgs_magic_chunks, sink = SINK)
#
#   rm(wgs_data_pre, wgs_data_split, wgs_magic_chunks_data_pre, wgs_magic_chunks, SI
#   gc()
#
#   }
# }

parents_mema_gws_all_data <-
  WGS_parents_raw %>%
  select(comp_group_label) %>%
  mutate(ds = list(downsize_scheme)) %>%
  unnest(ds) %>%
  mutate(file = paste0(
    "C:/Users/sergi/Documents/SAWERS LAB/MAGIC/mosaic_estimates/gen_mosaic_estimates
    comp_group_label,
    "_", code,
    "_chunk_info.parquet")) %>%
  mutate(data = map(
    .x = file,
    .f = ~ read_parquet(.x)
  ))
```

```
WGS_parents_chunk_sizes <-
  parents_mema_gws_all_data %>%
  unnest(data) %>%
  group_by(comp_group_label, n_marker) %>%
  summarise(
    min_chunk_size = min(chunk_size),
    med_chunk_size = median(chunk_size),
    max_chunk_size = max(chunk_size),
    code = first(code)
    )
```

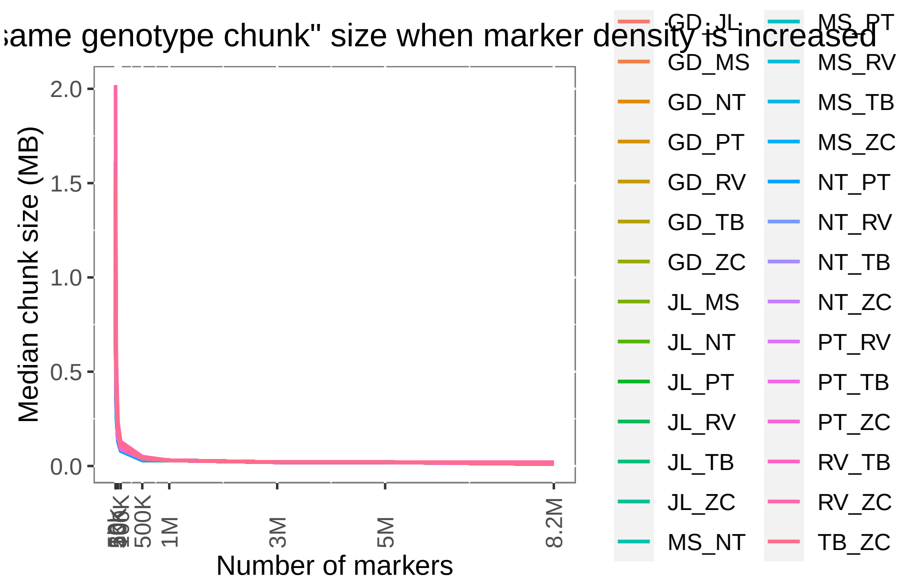
```
`summarise()` has grouped output by 'comp_group_label'. You can override using
the `.groups` argument.
```

```
WGS_parents_chunk_sizes %>%
  group_by(n_marker, code) %>%
  pivot_longer(contains("chunk_size")) %>%
  ungroup() %>%
  filter(name != "min_chunk_size") %>%
  group_by(code, name) %>%
  filter(value == min(value) | value == max(value)) %>%
  ungroup() %>%
  select(-comp_group_label) %>%
  distinct() %>%
  arrange(n_marker, name) %>%
  group_by(n_marker, code, name) %>%
  summarise(interval = paste(value, collapse = " - ")) %>%
  pivot_wider(names_from = name, values_from = interval)
```

```
`summarise()` has grouped output by 'n_marker', 'code'. You can override using
the `.groups` argument.

# A tibble: 10 x 4
# Groups:   n_marker, code [10]
   n_marker code  max_chunk_size med_chunk_size
      <dbl> <chr> <chr>          <chr>
 1     5000 5k    28.76 - 66.82  0.93 - 2.02
 2    10000 10k   21.82 - 56.12  0.51 - 1.01
 3    20000 20k   10.53 - 41.29  0.26 - 0.55
 4    50000 50K   7.57 - 22.21   0.13 - 0.23
 5   100000 100K  5.35 - 19.09   0.08 - 0.13
 6   500000 500K  2.46 - 7.87    0.03 - 0.05
 7  1000000 1M    2.38 - 5.83    0.03
 8  3000000 3M    5.49 - 1.19    0.02
 9  5000000 5M    5.33 - 1       0.02
```

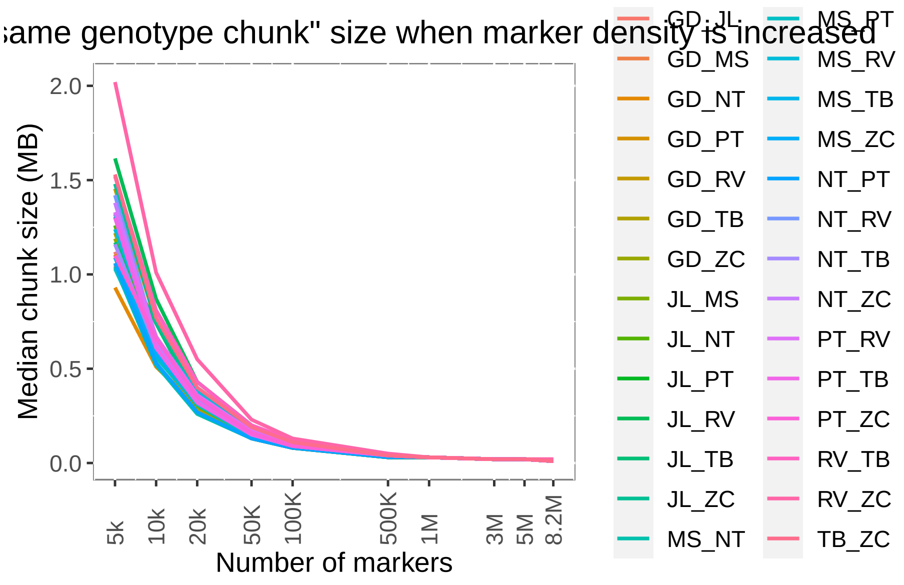
```
10  8125930 8.2M  0.56 - 4.13    0.01 - 0.02
```

```r
WGS_parents_med_chunk_sizes_plot <-
  WGS_parents_chunk_sizes %>%
  mutate(
    code = as_factor(code)) %>%
  ggplot(
    data =.,
    aes(x = n_marker, y = med_chunk_size, color = comp_group_label, group = comp_gro
  ) +
  geom_line(linewidth = 0.75) +
  scale_x_continuous(
    breaks = unique(WGS_parents_chunk_sizes$n_marker),
    labels = unique(WGS_parents_chunk_sizes$code)
  ) +
  xlab("Number of markers") +
  ylab("Median chunk size (MB)") +
  ggtitle('Reduction in "same genotype chunk" size when marker density is increased'
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 0),
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)
    )

WGS_parents_med_chunk_sizes_plot
```

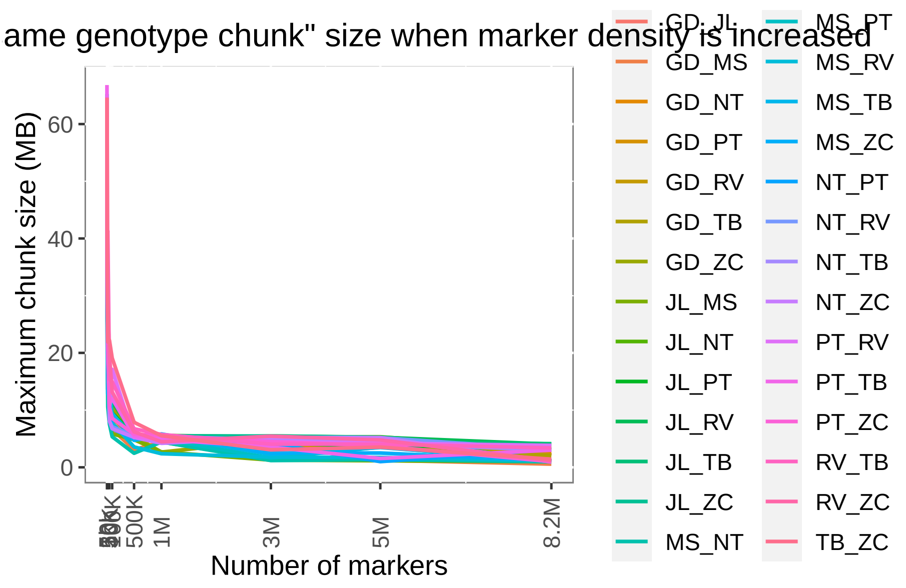


```r
WGS_parents_med_chunk_sizes_plot +
  scale_x_log10(
    breaks = unique(WGS_parents_chunk_sizes$n_marker),
    labels = unique(WGS_parents_chunk_sizes$code)
  )
```

```
Scale for x is already present.
Adding another scale for x, which will replace the existing scale.
```

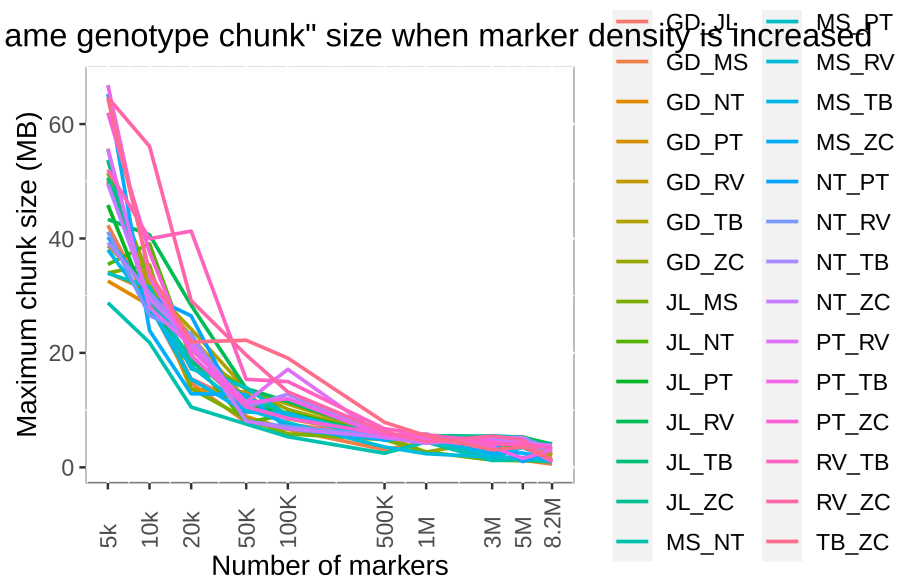


```
WGS_parents_max_chunk_sizes_plot <-
  WGS_parents_chunk_sizes %>%
  mutate(
    code = as_factor(code)) %>%
  ggplot(
    data =.,
    aes(x = n_marker, y = max_chunk_size, color = comp_group_label, group = comp_gro
  ) +
  geom_line(linewidth = 0.75) +
  scale_x_continuous(
    breaks = unique(WGS_parents_chunk_sizes$n_marker),
    labels = unique(WGS_parents_chunk_sizes$code)
  ) +
  xlab("Number of markers") +
  ylab("Maximum chunk size (MB)") +
  ggtitle('Reduction in "same genotype chunk" size when marker density is increased'
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 0),
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)
    )

WGS_parents_max_chunk_sizes_plot
```

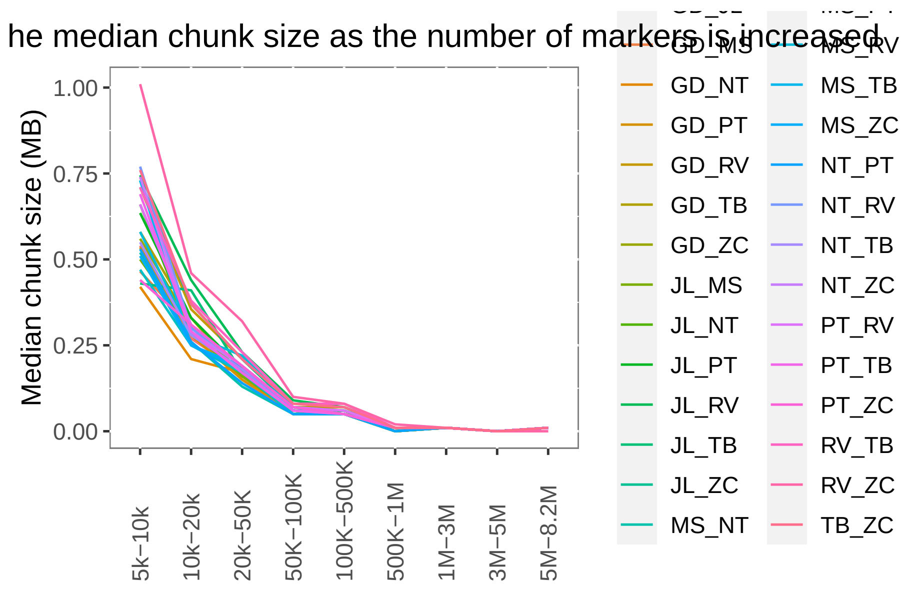


```r
WGS_parents_max_chunk_sizes_plot +
  scale_x_log10(
    breaks = unique(WGS_parents_chunk_sizes$n_marker),
    labels = unique(WGS_parents_chunk_sizes$code)
  )
```

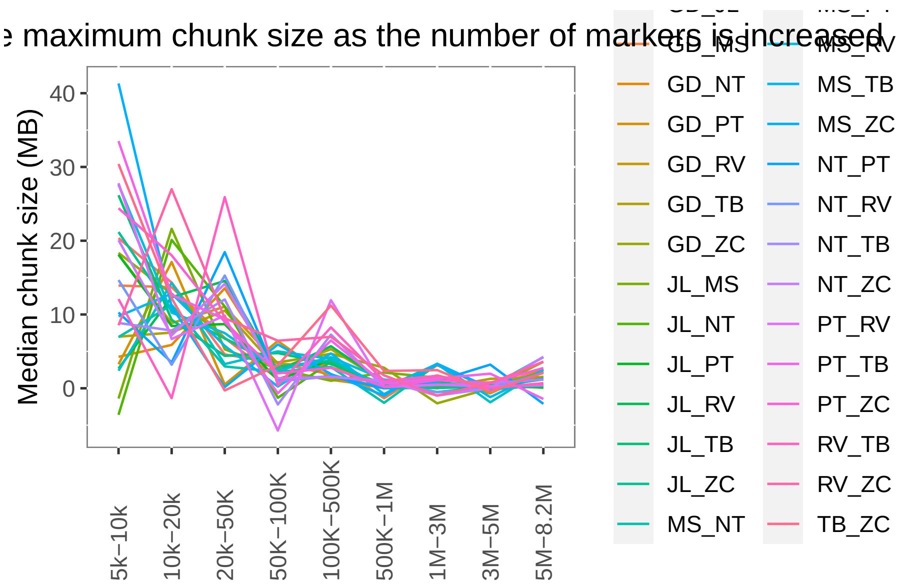
```
Scale for x is already present.
Adding another scale for x, which will replace the existing scale.
```



```
dif_chunk_size_par <-
  WGS_parents_chunk_sizes %>%
  group_by(comp_group_label) %>%
  mutate(
    dif_med = med_chunk_size - lead(med_chunk_size),
    dif_max = max_chunk_size - lead(max_chunk_size),
    code2 = lead(code),
    code2 = paste0(code, "-", code2)
    ) %>%
  filter(!is.na(dif_med)) %>%
  mutate(code2 = as_factor(code2))

dif_chunk_size_par %>%
  ggplot(
    data =.,
    aes(x = code2, y = dif_med, color = comp_group_label, group = comp_group_label))
  geom_line()  +
  xlab(NULL) +
  ylab("Median chunk size (MB)") +
  ggtitle('Difference in the median chunk size as the number of markers is increased
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 0),
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)
    )
```



```
dif_chunk_size_par %>%
  ggplot(
    data =.,
    aes(x = code2, y = dif_max, color = comp_group_label, group = comp_group_label)) +
  geom_line()  +
  xlab(NULL) +
  ylab("Median chunk size (MB)") +
  ggtitle('Difference in the maximum chunk size as the number of markers is increased') +
  theme(
    panel.background = element_rect(fill = "white", colour = "grey50"),
    text = element_text(size = 12),
    axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 0),
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5)
    )
```



We hypothesize that the big amount of missing information in the assignment of a parental haplotypes for markers in the MEMA population is because at the actual marker density (~ 15K markers) rqtl2 cannot differentiate between any parent with a 95% of confidence, thus generating missing information. This leads us to think that is because there are genomic regions in the parents that have "identical genotype (IBD?)", making difficult or impossible for qtl2 to differentiate one parent from the other. So we think that if we can reduce the size of this "same genotype" regions by identifying polymorphic markers across each pair of parents, we can give more information to rqtl2 to assign a parental haplotype and reduce the number of misscalling.

The question here is: how big are this regions? and If we increase the genome-wide marker density, can we reduce the size of this regions? Can we add more information with more markers? How many markers do we need to add more information and reduce the size of the "same genotype chunks"?

To answer these questions we assess the size of the "same genotype" regions in each pairwise combination of the parents of the MEMA population by using the 50K CHIP marker data (~ 15K SNPs) and WGS data (~8.2M SNPs). We use a "down-scale" simulation with the WGS data, where we randomly sampled markers across the genome with the same proportion of markers per chromosome as the original dataset, to assess if we can observe a gradient of how much information we acquire (in terms of reduction of the size of the same genotype chunks) as we increase the marker density. We selected 5000, 10,000, 20,000, 100,000, 1,000,000, 3,000,000, 5,000,000 and 8,200,000 markers to answer this question.

For the CHIP dataset, the same chunk genotype median and max size can go from 0.17-0.24 MB and 5.09-19 MB respectively. For WGS data, we can observe that an increase of the number of markers leads to the reduction of the median and max same genotype chunk size, for instance, with 50,000 markers we get a median chunk size from 0.13 - 0.23 MB and max size of 7.57 - 22.21 MB, and as we increase the marker density we observe a reduction in these values. So we can conclude that more dense marker data increases the information on the parents of the MEMA population. So it is worth to find a genotyping way that can increase the density of markers, like skim sequencing + imputation.

Part II

CML

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

You can add options to executable code like this

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).

Chapter 5

2023_Highland_experiment_design

Sergio Perez-Limon

Here I use the FieldHub app to design the field evaluation of the CML F2:3 population, for the Highlands we're using a p-rep design (more info: 10.1198/108571106X154443). In total there are going to be 210 genotypes of the HI73 and HI93 populations, 320 from the HI79 and Hermes as a CHECK evaluated in two locations. For this experimental design, 70 genotypes of each biparental population are going to be replicated, and the rest (140, 140 and 250 respectively) are going as single reps. The families are selected based on the phc BLUP value. It is important to note that we're only removing a few families based on this, so there is not an important bias/selection towards the best genotypes, but it might help us to select families that are going to survive. We select the families that are going to be repeated randomly. Check is repeated 50 times. The ecperiment is designed for a 25 columns x 40 rows experiment.

There is an augmented block design for Ameca, a possible "no stress" environment where we can have a baseline to compare the performance of the highland site. Here, the experiment is composed of a 53 block per repetition, and 2 replications, with the same genotypes as the p-rep design. Each genotype is present only once per replication and is not repeated across block either.

Loading libraries

```r
library(FielDHub)
library(googlesheets4)
```

```
Warning: package 'googlesheets4' was built under R version 4.2.2
```

```r
library(googledrive)
```

```
Attaching package: 'googledrive'
```

```
The following objects are masked from 'package:googlesheets4':

    request_generate, request_make
```

```r
library(tidyverse)
```

```
v ggplot2 3.4.1      v purrr   1.0.1
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.3.0      v stringr 1.5.0
v readr   2.1.3      v forcats 0.5.2
```

```
Warning: package 'ggplot2' was built under R version 4.2.2
```

```
Warning: package 'tidyr' was built under R version 4.2.2
```

```
Warning: package 'readr' was built under R version 4.2.2
```

```
Warning: package 'purrr' was built under R version 4.2.2
```

```
Warning: package 'dplyr' was built under R version 4.2.2
```

```
Warning: package 'stringr' was built under R version 4.2.2
```

```
Warning: package 'forcats' was built under R version 4.2.2
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter()                 masks stats::filter()
x dplyr::lag()                    masks stats::lag()
x googledrive::request_generate() masks googlesheets4::request_generate()
x googledrive::request_make()     masks googlesheets4::request_make()
```

```r
library(lme4)
```

```
Warning: package 'lme4' was built under R version 4.2.2
```

```
Loading required package: Matrix
```

```
Warning: package 'Matrix' was built under R version 4.2.2
```

```
Attaching package: 'Matrix'
```

```
The following objects are masked from 'package:tidyr':

    expand, pack, unpack
```

```
gs4_auth("checo.spl@gmail.com")
```

Import available genotypes and genotype information

```
# raw data for genotypes
data_url <- "https://docs.google.com/spreadsheets/d/14yHP2kqQZQ-wp0IdiUgeUeQIeqorQed3-m-xcwFu_

genotypes_data_raw <-
  data_url %>%
  as_id() %>%
  range_read(sheet = "Genetic Stocks")
```

```
v Reading from "CML457_459_Populations".
```

```
v Range ''Genetic Stocks''.
```

```
families_url <- "https://docs.google.com/spreadsheets/d/1Mix1vFbJkeLhC3psHaOxxJscgPXsiizt-unjr

seed_availability_url <- "https://docs.google.com/spreadsheets/d/14yHP2kqQZQ-wp0IdiUgeUeQIeqor

# Import raw data for families
families_data_raw <- families_url %>%
  as_id %>%
  range_read(ss =., sheet = "UNISEM21A", skip = 1)
```

```
v Reading from "21_NCS_PSU_LANGEBIO_FIELDS".
```

```
v Range ''UNISEM21A'!2:10000000'.
```

```
New names:
* `` -> `...16`
* `` -> `...18`
```

```
seed_available <- seed_availability_url %>%
  as_id() %>%
  range_read(ss =., sheet = "Genetic Stocks")
```

```
v Reading from "CML457_459_Populations".
v Range ''Genetic Stocks''.
```

```
packing_seed_info_url <- "https://docs.google.com/spreadsheets/d/14yHP2kqQZQ-wp0IdiUgeUeQIeqor

# packing seed info

packing_seed_info <-
```

```
    packing_seed_info_url %>%
    as_id() %>%
    range_read(ss =., sheet = "seed_packing_2022")
```

```
v Reading from "CML457_459_Populations".
v Range ''seed_packing_2022''.
```

```
  packing_seed_info_unique <-
    packing_seed_info %>%
    select(familia, F2_Parent, sobre_UNISEM) %>%
    unique() %>%
    rename(`Female genotype` = familia, `Female parent` = sobre_UNISEM)
```

Highland site p-rep design

```
  # Genotypes available for planting

  available_genos <-
    genotypes_data_raw %>%
    select(family = Family_ID, pop = Population_ID, use = `Use_23 (798)`) %>%
    filter(use == 1)

  # select families based on the best BLUP values
  cml_pops_raw_data <-
    data_url %>%
    as_id() %>%
    range_read("22_Highland_Raw_data_tidy") %>%
    mutate(phc = as.double(phc))
```

```
v Reading from "CML457_459_Populations".

v Range ''22_Highland_Raw_data_tidy''.
```

```
  cml_phc_example <-
    cml_pops_raw_data %>%
    select(location, family, phc, nblock) %>%
    mutate(across(c(location, family), ~ as_factor(.))) %>%
    mutate(phc = ifelse(phc > 2 | phc == 0, NA_real_, phc)) %>%
    filter(family != "HERMES")

  set.seed(100)
  phc_BLUP_example <-
    lmer(
      phc ~ location + (1|family) + (1|location:nblock), data = cml_phc_example
      ) %>%
    ranef() %>%
    .$family %>%
    as_tibble(rownames = "family") %>%
    rename(BLUP = `(Intercept)`)

  # Best families available based on the BLUP value
  prospect_families <-
    phc_BLUP_example %>%
    semi_join(available_genos) %>%
    mutate(pop = gsub("-\\d+$", "", family)) %>%
    arrange(pop, desc(BLUP)) %>%
    group_by(pop) %>%
    mutate(id = row_number()) %>%
    filter(
      (pop == "HI73" & id %in% c(1:210)) |
        (pop == "HI93" & id %in% c(1:210)) |
        (pop == "HI79" & id %in% c(1:320))
    ) %>%
    ungroup() %>%
    rename(phc_BLUP = BLUP)
```

```
Joining, by = "family"
```

```
# p-rep eperimental design using FieldHub

highland_2023_prep_design <-
  partially_replicated(
    nrows = 40,
    ncols = 25,
    repGens = c(530, 210, 1),
    repUnits = c(1, 2, 50),
    planter = "serpentine",
    l = 2,
    seed = 100,
    locationNames = c("site1", "site2")
    )
```

```
Warning message:
 Since plotNumber was missing, it was set up to default value of:  1001 2001
```

```r
# Extracting the fieldbook from the experimental design
high23_fieldbook <-
  highland_2023_prep_design$fieldBook %>%
  as_tibble() %>%
  rename_with(~tolower(.))

# Selecting rep families by random and randomizing families within each pop
set.seed(100)
high_exp_data <-
  prospect_families %>%
  group_by(pop) %>%
  slice_sample(prop = 1) %>%
  mutate(id2 = row_number()) %>%
  ungroup() %>%
  select(family, pop, id2) %>%
  add_row(family = "CHECK", id2 = NA, .before = 1) %>%
  mutate(
    rep = case_when(
      is.na(id2) ~ 50,
      id2 %in% c(1:70) ~ 2,
      T ~ 1
    )) %>%
  arrange(desc(rep), pop, id2) %>%
  mutate(id = row_number())

treatment_family <-
  high23_fieldbook %>%
  count(location, treatment) %>%
  mutate(id = gsub("G", "", treatment) %>% as.integer()) %>%
  rename(rep = n) %>%
  arrange(id) %>%
  left_join(high_exp_data) %>%
  select(treatment, family, pop) %>%
  distinct() %>%
  mutate(pop = ifelse(is.na(pop), "CHECK", pop))
```

```
Joining, by = c("rep", "id")
```

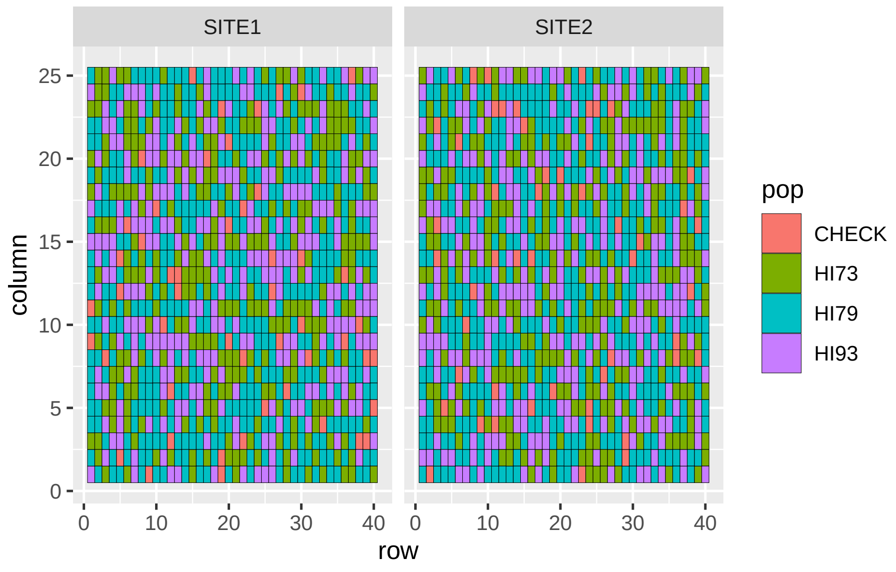
```r
high23_fieldbook_family <-
  high23_fieldbook %>%
  left_join(treatment_family)
```

```
Joining, by = "treatment"
```

```r
high23_fieldbook_family
```

```
# A tibble: 2,000 x 12
      id expt  location year   plot   row column checks entry treatment family
   <int> <chr> <chr>    <chr> <dbl> <int>  <int> <chr>  <dbl> <chr>     <chr>
 1     1 Expt1 SITE1    2023   1001     1      1 186      186 G186      HI93-271
 2     2 Expt1 SITE1    2023   1002     1      2 0        574 G574      HI79-370
 3     3 Expt1 SITE1    2023   1003     1      3 0        218 G218      HI73-028
 4     4 Expt1 SITE1    2023   1004     1      4 0        508 G508      HI79-280
 5     5 Expt1 SITE1    2023   1005     1      5 0        567 G567      HI79-306
 6     6 Expt1 SITE1    2023   1006     1      6 115      115 G115      HI79-286
 7     7 Expt1 SITE1    2023   1007     1      7 0        444 G444      HI79-321
 8     8 Expt1 SITE1    2023   1008     1      8 0        384 G384      HI79-191
 9     9 Expt1 SITE1    2023   1009     1      9 1          1 G1        CHECK
10    10 Expt1 SITE1    2023   1010     1     10 0        482 G482      HI79-104
# ... with 1,990 more rows, and 1 more variable: pop <chr>
```

```r
high23_fieldbook_family %>%
  ggplot(data =., aes(x = row, y = column)) +
  geom_tile(
    aes(fill = pop),
    color = "black") +
  facet_grid(. ~ location)
```



```
data_spreadsheet <-
  high23_fieldbook_family %>%
  select(
    `CML23-` = plot, Description = pop, `Female genotype` = family,
    row_design = row, column_design = column, checks) %>%
  mutate(
    `Origin (Package)` = "LANGEBIO",
    `Packed?` = " ",
    `Who/What` = "AL/JL",
    `Male parent` = "x sib",
    `Male genotype` = "x sib",
    `Number/Selection` = "14K",
    rep = case_when(
      checks == 0 ~ 1,
      Description == "CHECK" ~ 50,
      T ~ 2
    )
  ) %>%
  left_join(packing_seed_info_unique) %>%
  select(
    `CML23-`, `Origin (Package)`:`Who/What`, Description, `Female parent`,
    `Male parent`, `Female genotype`,  `Male genotype`, `Number/Selection`,
    row_design, column_design, rep
    ) %>%
  mutate(across(contains("Female"), ~ ifelse(is.na(.), "HERMES", .)))
```

```
Joining, by = "Female genotype"
```

```
write_csv(data_spreadsheet, "23_highland_experiment_spreadsheet.csv")
```

Ameca augmented complete block design

```
prospect_families %>%
  count(pop) %>%
  mutate(prop = n/sum(n)) %>%
  mutate(a = prop*24)
```

```
# A tibble: 3 x 4
  pop       n  prop     a
  <chr> <int> <dbl> <dbl>
1 HI73    210 0.284  6.81
2 HI79    320 0.432 10.4
3 HI93    210 0.284  6.81
```

```
prospect_families %>% dim %>% .[[1]]
```

```
[1] 740
```

```
Ameca_RCBD <- RCBD_augmented(
  lines = prospect_families %>% dim %>% .[[1]],
  planter = "serpentine",
  checks = 1,
  b = 53,
  repsExpt = 2,
  l = 1,
  random = TRUE,
  locationNames = c("Ameca"),
  seed = 100,
  )

# 57, 53, 50, 47, 44

Ameca_fieldbook <- Ameca_RCBD$fieldBook %>%
  as_tibble() %>%
  rename_with( ~ tolower(.))

Ameca_fieldbook %>%
  filter(treatment != "CH1") %>%
  count(block) %>%
  mutate(nn = n/2) %>%
  count(nn)
```

```
# A tibble: 1 x 2
     nn     n
  <dbl> <int>
1    14    53
```

```
prospect_families %>%
  count(pop) %>%
  mutate(prop = n/sum(n)) %>%
  mutate(a = prop*14)
```

```
# A tibble: 3 x 4
  pop       n  prop     a
  <chr> <int> <dbl> <dbl>
1 HI73    210 0.284  3.97
2 HI79    320 0.432  6.05
3 HI93    210 0.284  3.97
```

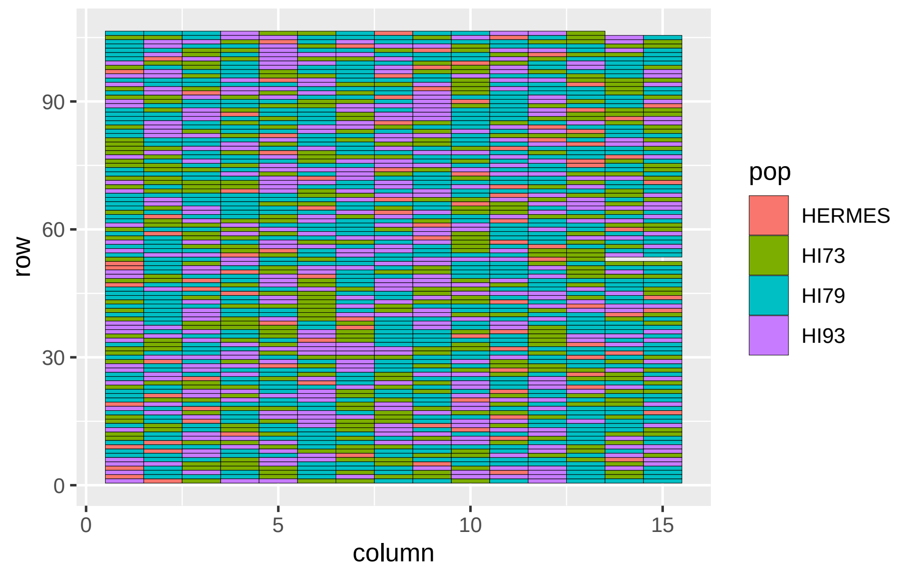
```
# Per block: 4 HI73/HI93; 6 HI79

set.seed(100)
Ameca_fieldbook_pops <-
  Ameca_fieldbook %>%
  mutate(rep = ifelse(id < 796, 1, 2)) %>%
  group_by(rep, block) %>%
  mutate(rowid = row_number()) %>%
  slice_sample(prop = 1) %>%
  mutate(
    type = ifelse(treatment == "CH1", "check", "family"),
    type = factor(type, levels = c("check", "family"))
    ) %>%
  ungroup() %>%
  arrange(rep, block, type) %>%
  mutate(what = rep(c("HERMES", rep("HI73", 4), rep("HI79", 6), rep("HI93", 4)), 106
  filter(treatment != "Filler")

Ameca_fieldbook_pops %>%
  count(rep, what)
```

```
# A tibble: 8 x 3
    rep what       n
  <dbl> <chr>  <int>
1     1 HERMES    53
2     1 HI73     211
3     1 HI79     317
4     1 HI93     212
5     2 HERMES    53
6     2 HI73     211
7     2 HI79     318
8     2 HI93     211
```

```
# Need 420 HI73/HI93; 640 HI79

set.seed(100)
extra_family_replacement <-
  Ameca_fieldbook_pops %>%
  filter(what %in% c("HI93", "HI73")) %>%
  group_by(rep, what) %>%
  slice_sample(n = 3) %>%
  filter(
    (rep == 1 & what == "HI73" & row_number() == 1) |
      (rep == 1 & what == "HI93" & row_number() %in% c(1:2)) |
      (rep == 2 & what == "HI73" & row_number() %in% c(1)) |
      (rep == 2 & what == "HI93" & row_number() %in% c(1))
  ) %>%
  mutate(what = "HI79")


set.seed(100)
Ameca_fieldbook_pops_family <- Ameca_fieldbook_pops %>%
  anti_join(extra_family_replacement, by = c("rep", "plot")) %>%
  bind_rows(extra_family_replacement) %>%
  arrange(rep, what) %>%
  group_by(rep, what) %>%
  slice_sample(prop = 1) %>%
  mutate(id2 = row_number()) %>%
  left_join(prospect_families %>% select(-phc_BLUP), by = c("what" = "pop", "id2" = "id")) %>%
  mutate(family = ifelse(what == "HERMES", "HERMES", family)) %>%
  ungroup()

Ameca_fieldbook_pops_family %>%
  arrange(id) %>%
  select(id, plot:block, rep, treatment, pop = what, family)
```

```
# A tibble: 1,586 x 10
      id  plot   row column checks block   rep treatment pop    family
   <int> <dbl> <int>  <int> <chr>  <dbl> <dbl> <chr>     <chr>  <chr>
 1     1   101     1      1 0          1     1 G658      HI93   HI93-179
 2     2   102     1      2 1          1     1 CH1       HERMES HERMES
 3     3   103     1      3 0          1     1 G205      HI73   HI73-092
 4     4   104     1      4 0          1     1 G133      HI93   HI93-233
 5     5   105     1      5 0          1     1 G641      HI93   HI93-108
 6     6   106     1      6 0          1     1 G321      HI73   HI73-137
 7     7   107     1      7 0          1     1 G418      HI73   HI73-220
 8     8   108     1      8 0          1     1 G44       HI79   HI79-162
 9     9   109     1      9 0          1     1 G636      HI79   HI79-268
10    10   110     1     10 0          1     1 G391      HI73   HI73-248
# ... with 1,576 more rows
```

```
Ameca_fieldbook_pops_family %>%
  select(id, plot:block, rep, treatment, pop = what, family) %>%
  ggplot(data =., aes(x = column, y = row, fill = pop)) +
  geom_tile(color = "black")
```



```r
Ameca_data_spreadsheet <-
  Ameca_fieldbook_pops_family %>%
  arrange(id) %>%
  mutate(`CML23-` = id + 4000) %>%
  select(
    `CML23-`, Description = what, `Female genotype` = family,
    row_design = row, column_design = column, checks, rep, block) %>%
  mutate(
    `Origin (Package)` = "LANGEBIO",
    `Packed?` = " ",
    `Who/What` = "AL/JL",
    `Male parent` = "x sib",
    `Male genotype` = "x sib",
    `Number/Selection` = "14K"
    ) %>%
  left_join(packing_seed_info_unique) %>%
  select(
    `CML23-`, `Origin (Package)`:`Who/What`, Description, `Female parent`,
    `Male parent`, `Female genotype`,  `Male genotype`, `Number/Selection`,
    row_design, column_design, rep, block
    ) %>%
  mutate(across(contains("Female"), ~ ifelse(is.na(.), "HERMES", .)))
```

```
Joining, by = "Female genotype"
```

```r
write_csv(Ameca_data_spreadsheet, "Ameca_data_spreadsheet.csv")
```

Chapter 6

PT

Sergio Pérez-Limón

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

6.1 Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

You can add options to executable code like this

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).

Chapter 7

AMF

Sergio Pérez-Limón

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

7.1 Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

You can add options to executable code like this

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).

Chapter 8

Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.