Next-generation sequencing bulk segregant analysis with QTLseqr

Ben N. Mansfeld and Rebecca Grumet

2018-07-07

Contents

Current version: Development - 0.7.0 **1**

Introduction **1**
Citations . . . 1
Quick Start . . . 2

Standard workflow **3**
Installation . . . 3
Input data . . . 3
Loaded data frame . . . 6
Filtering SNPs . . . 6
Running the analysis . . . 9
Plotting the data . . . 12
Extracting QTL data . . . 15

Summary **17**

Session info **17**

Current version: Development - 0.7.0

Introduction

Citations

If you use QTLseqr in published research, please cite:

> Mansfeld B.N. and Grumet R, QTLseqr: An R package for bulk segregant analysis with next-generation sequencing *The Plant Genome* doi:10.3835/plantgenome2018.01.0006

We also recommend citing the paper for the corresponding method you work with.

QTL-seq method:

> Takagi, H., Abe, A., Yoshida, K., Kosugi, S., Natsume, S., Mitsuoka, C., Uemura, A., Utsushi, H., Tamiru, M., Takuno, S., Innan, H., Cano, L. M., Kamoun, S. and Terauchi, R. (2013), QTL-seq: rapid mapping of quantitative trait loci in rice by whole genome resequencing of DNA from two bulked populations. *Plant J*, 74: 174–183. doi:10.1111/tpj.12105

G prime method:

> Magwene PM, Willis JH, Kelly JK (2011) The Statistics of Bulk Segregant Analysis Using Next Generation Sequencing. *PLOS Computational Biology* 7(11): e1002255. doi.org/10.1371/journal.pcbi.1002255

Quick Start

Here are the basic steps required to run and plot QTLseq and G' analysis

```r
# download and install the devtools package
install.packages("devtools")

# download the QTLseqr package from GitHub
devtools::install_github("bmansfeld/QTLseqr")
```

```r
#load the package
library("QTLseqr")

#Set sample and file names
HighBulk <- "SRR834931"
LowBulk <- "SRR834927"
file <- "SNPs_from_GATK.table"

#Choose which chromosomes will be included in the analysis (i.e. exclude smaller contigs)
Chroms <- paste0(rep("Chr", 12), 1:12)

#Import SNP data from file
df <-
    importFromGATK(
        file = file,
        highBulk = HighBulk,
        lowBulk = LowBulk,
        chromList = Chroms
    )

#Filter SNPs based on some criteria
df_filt <-
    filterSNPs(
        SNPset = df,
        refAlleleFreq = 0.20,
        minTotalDepth = 100,
        maxTotalDepth = 400,
        minSampleDepth = 40,
        minGQ = 99
    )

#Run G' analysis
df_filt <- runGprimeAnalysis(SNPset = df_filt,
                             windowSize = 1e6,
                             outlierFilter = "deltaSNP")

#Run QTLseq analysis
df_filt <- runQTLseqAnalysis(
    SNPset = df_filt,
    windowSize = 1e6,
    popStruc = "F2",
    bulkSize = 300,
    replications = 10000,
    intervals = c(95, 99)
)
```

```r
#Plot
plotQTLStats(
    SNPset = df_filt,
    var = "Gprime",
    plotThreshold = TRUE,
    q = 0.01
)

plotQTLStats(
    SNPset = df_filt,
    var = "deltaSNP",
    plotIntervals = TRUE)

#export summary CSV
getQTLTable(
    SNPset = df_filt,
    alpha = 0.01,
    export = TRUE,
    fileName = "my_BSA_QTL.csv"
)
```

Standard workflow

Installation

Let's install and load the QTLseqr package:

```r
#Install step if you have not done so yet:
#install.packages("devtools")

devtools::install_github("bmansfeld/QTLseqr")
library("QTLseqr")
```

Great! We now need to load some data. QTLseqr has with a sister package Yang2013data (derived from Yang et al. (2013)], that has some trial data you can play with.

Input data

QTLseqr supports importing data either from table file exported from the VariantsToTable function built in to GATK or a delimited file containing allele read depths for each bulk. The two available functions are `importFromGATK` and `importFromTable`.

Both functions import the SNP data and perform some preliminary calculations to find the following:

$$Reference\ allele\ frequency = \frac{Ref\ allele\ depth_{HighBulk} + Ref\ allele\ depth_{LowBulk}}{Total\ read\ depth\ for\ both\ bulks}$$

$$SNP\text{-}index_{per\ bulk} = \frac{Alternate\ allele\ depth}{Total\ read\ depth}$$

$$\Delta(SNP\text{-}index) = SNP\text{-}index_{HighBulk} - SNP\text{-}index_{LowBulk}$$

To demonstrate the use of the import functions we will first load the Yang et al. (2013) data file. We first need to download the package that contains the data from github.

```
#download and load data package (~50Mb)
devtools::install_github("bmansfeld/Yang2013data")
library("Yang2013data")

#Import the data
rawData <- system.file(
    "extdata",
    "Yang_et_al_2013.table",
    package = "Yang2013data",
    mustWork = TRUE)
```

If you have your own data you can simply refer to it directly:

```
rawData <- "C:/PATH/TO/MY/DIR/My_BSA_data.table"
```

We define the sample name for each of the bulks. We also can define a vector of the chromosomes to be included in the analysis (i.e. exclude smaller contigs), In this case, Chr1, Chr2 . . . Chr12.

```
HighBulk <- "SRR834931"
LowBulk <- "SRR834927"
Chroms <- paste0(rep("Chr", 12), 1:12)
```

Importing SNPs from GATK

Working directly with the GATK best practices guide for whole genome sequence should result in a VCF that is compatible with QTLseqr. In general the workflow suggested by GATK is per-sample variant calling followed by joint genotyping across samples. This will produce a VCF file that includes **BOTH** bulks, each with a different sample name (here SRR834927 and SRR834931), one SNP for example:

CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SRR834927	SRR834931
Chr1	31071	.	A	G	1390.44	PASS	..*...	GT:AD:DP:GQ:PL	0/1:34,36:70:99:897,0,855	0/1:26,22:48:99:522,0,698

**info column removed for brevity*

GATK have provided a fast VCF parser, the VariantsToTable tool, that extracts the necessary fields for easy use in downstream analysis.

We highly recommend reading What is a VCF and how should I interpret it? for more information on GATK VCF Fields and Genotype Fields

Though the use of GATK's VariantsToTable function is out of the scope of this vignette, the syntax for use with QTLseqr should look something like this:

```
java -jar GenomeAnalysisTK.jar \
-T VariantsToTable \
-R ${REF} \
-V ${NAME} \
-F CHROM -F POS -F REF -F ALT \
-GF AD -GF DP -GF GQ -GF PL \
-o ${NAME}.table
```

Where `${REF}` is the reference genome file and `${NAME}` is VCF file you wish to parse.

To run QTLseqr successfully, the required VCF fields `(-F)` are CHROM (Chromosome) and POS (Position). the required Genotype fields `(-GF)` are AD (Allele Depth), DP (Depth). Recommended fields are REF (Reference allele) and ALT (Alternative allele) Recommended Genotype fields are PL (Phred-scaled likelihoods) and GQ (Genotype Quality).

The `ImportFromGATK` function

The `importFromGATK` function imports SNP data from the output of the VariantsToTable function in GATK. After importing the data, the function then calculates total reference allele frequency for both bulks together, the SNP index for each bulk, and the $\Delta(SNP\text{-}index)$.

We then use the `importFromGATK` function to import the raw data. After importing the data, the function then calculates total reference allele frequency for both bulks together, the $SNP\text{-}index$ for each SNP in each bulk and the $\Delta(SNP\text{-}index)$ and returns a data frame.

Let's import:

```
#import data
df <-
    importFromGATK(
        file = rawData,
        highBulk = HighBulk,
        lowBulk = LowBulk,
        chromList = Chroms
     )
```

Importing from a delimited file

You can also import SNPs from a csv, tsv or any other delimited file. Your file must include some necessary columns: CHROM (Chromosome names) and POS (the SNP position) as well as the reference and alternate allele depths (number of reads supporting each allele). The allele depths should be in columns named in this format: *AD_<ALT/REF>.<sampleName>*. For example, the column for alternate allele depth for a high bulk sample named "sample1", should be "AD_ALT.sample1". Any other columns describing the SNPs are allowed, ie the actual allele calls, or a quality score. If the column is Bulk specific, It should be named columnName.sampleName, i.e "QUAL.sample1".

Here is the header of an example file:

CHROM	POS	REF	ALT	AD_REF.SRR834927	AD_ALT.SRR834927	GQ.SRR834927	AD_REF.SRR834931	AD_ALT.SRR834931	GQ.SRR834931
Chr1	31071	A	G	34	36	99	26	22	99
Chr1	31478	C	T	34	52	99	40	34	99
Chr1	33667	A	G	20	48	99	24	29	99
Chr1	34057	C	T	38	40	99	29	26	99
Chr1	35239	A	C	25	36	99	40	60	99

The `importFromTable` function

To load the data from a delimited file we use the `importFromTable` function. in our case it is a csv so it is comma delimited. The columns are denoted by the sample names for the bulks (SRR834931 and SRR834927). These names will be renamed to "HIGH" and "LOW". If you want you can set the column names as such in advance as the defaults for `highBulk` and `lowBulk` arguments are "HIGH" and "LOW", respectively. As in the `importFromGATK` function the `chromList` argument should be a string vector of the chromosomes to be used in the analysis. Useful for filtering out unwanted contigs etc.

```
df <- importFromTable(file = "Yang2013.csv",
                      highBulk = HighBulk,
```

```
                        lowBulk = LowBulk,
                        chromList = Chroms)
```

Loaded data frame

The loaded data frame should look like this. This data frame has other information about the genotype calls from GATK:

```
head(df)
```

```
##   CHROM   POS REF ALT AD_REF.LOW AD_ALT.LOW DP.LOW GQ.LOW     PL.LOW
## 1  Chr1 31071   A   G         34         36     70     99  897,0,855
## 2  Chr1 31478   C   T         34         52     86     99 1363,0,844
## 3  Chr1 33667   A   G         20         48     68     99 1331,0,438
## 4  Chr1 34057   C   T         38         40     78     99 1059,0,996
## 5  Chr1 35239   A   C         25         36     61     99  987,0,630
## 6  Chr1 38389   T   C         36         42     78     99 1066,0,906
##   SNPindex.LOW AD_REF.HIGH AD_ALT.HIGH DP.HIGH GQ.HIGH     PL.HIGH
## 1    0.5142857          26          22      48      99   522,0,698
## 2    0.6046512          40          34      74      99  848,0,1099
## 3    0.7058824          24          29      53      99   765,0,599
## 4    0.5128205          29          26      55      99   673,0,772
## 5    0.5901639          40          60     100      99 1632,0,1015
## 6    0.5384615          42          40      82      99  984,0,1105
##   SNPindex.HIGH   REF_FRQ     deltaSNP
## 1     0.4583333 0.5084746 -0.055952381
## 2     0.4594595 0.4625000 -0.145191703
## 3     0.5471698 0.3636364 -0.158712542
## 4     0.4727273 0.5037594 -0.040093240
## 5     0.6000000 0.4037267  0.009836066
## 6     0.4878049 0.4875000 -0.050656660
```

Let's review the column headers:

- CHROM - The chromosome this SNP is in
- POS - The position on the chromosome in nt
- REF - The reference allele at that position
- ALT - The alternate allele
- DP.HIGH - The read depth at that position in the high bulk
- AD_REF.HIGH - The allele depth of the reference allele in the high bulk
- AD_ALT.HIGH - The alternate allele depth in the the high bulk
- GQ.HIGH - The genotype quality score, (how confident we are in the genotyping)
- SNPindex.HIGH - The calculated SNP-index for the high bulk
- Same as above for the low bulk
- REF_FRQ - The reference allele frequency as defined above
- deltaSNP - The $\Delta(SNP\text{-}index)$ as defined above

Filtering SNPs

Now that we have loaded the data into R we can start cleaning it up by filtering some of the low confidence SNPs. While GATK has its own filtering tools, QTLseqr offers some options for filtering that may help reduce noise and improve results. Filtering is mainly based on read depth for each SNP, such that we can try to eliminate SNPs with low confidence, due to low coverage, and SNPs that may be in repetitive regions and

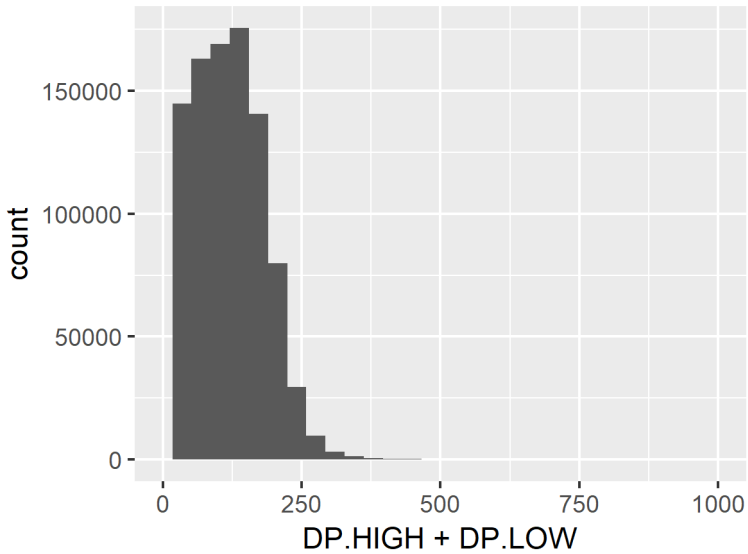
thus have inflated read depth. If you have your own quality columns you can always use R's base subsetting functions to filter out any SNPs you don't want.

Read depth histograms

One way to assess filtering thresholds is by plotting histograms of the read depths. We can get an idea of where to draw our thresholds. We'll use the ggplot2 package for this purpose, but you could use base R to plot as well.

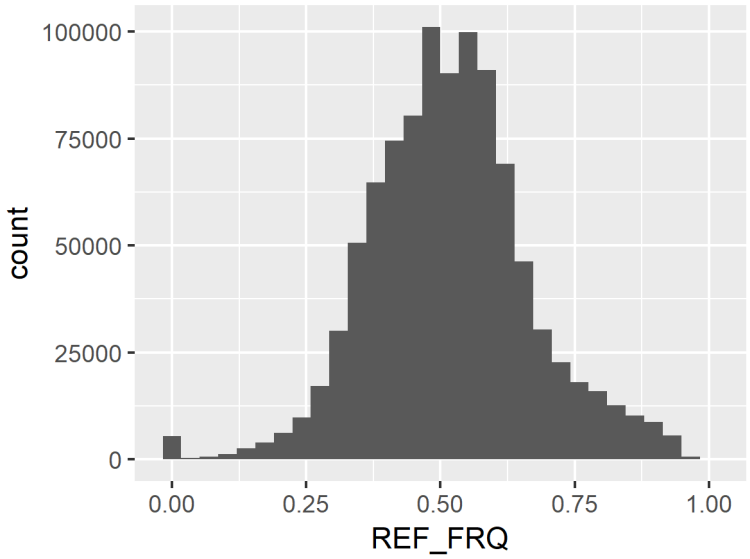
Lets look at total read depth for example:

```
library("ggplot2")
ggplot(data = df) +
    geom_histogram(aes(x = DP.HIGH + DP.LOW)) +
    xlim(0,1000)
```



...or look at total reference allele frequency:

```
ggplot(data = df) +
    geom_histogram(aes(x = REF_FRQ))
```



Using the filterSNPs function

Now that we have an idea about our read depth distribution we can filter out low confidence SNPS. In general we recommend filtering extremely low and high coverage SNPs, either in both bulks (`minTotalDepth/maxTotalDepth`) and/or in each bulk separately (`minSampleDepth`). We have the option to filter based on reference allele frequency (`refAlleleFreq`), this removes SNPs that for some reason are over- or under-represented in *BOTH* bulks. We can also use the GATK GQ score (Genotype Quality) to filter out low confidence SNPs. If the `verbose` parameter is set to `TRUE` (default) the function will report the numbers of SNPs filtered in each step.

```
df_filt <-
    filterSNPs(
        SNPset = df,
        refAlleleFreq = 0.20,
        minTotalDepth = 100,
        maxTotalDepth = 400,
        minSampleDepth = 40,
        minGQ = 99,
        verbose = TRUE
    )
```

```
## Filtering by reference allele frequency: 0.2 <= REF_FRQ <= 0.8
## ...Filtered 59754 SNPs
## Filtering by total sample read depth: Total DP >= 100
## ...Filtered 378814 SNPs
## Filtering by total sample read depth: Total DP <= 400
## ...Filtered 744 SNPs
## Filtering by per sample read depth: DP >= 40
## ...Filtered 4729 SNPs
## Filtering by Genotype Quality: GQ >= 99
## ...Filtered 6677 SNPs
## Original SNP number: 969487, Filtered: 450718, Remaining: 518769
```

This step is quick and we can go back and plot some histograms to see if we are happy with the results, and we can quickly re-run the filtering step if not.

Running the analysis

The analysis in QTLseqr is an implementation of both pipelines for bulk segregant analysis, G' and $\Delta(SNP\text{-}index)$, described by Magwene et al. (2011) and Takagi et al. (2013), respectively. We recommend reading both papers to fully understand the considerations and math behind the analysis.

There are two main analysis functions: 1. `runGprimeAnalysis` - performs Magwene et al type G' analysis 1. `runQTLseqAnalysis` - performs Takagi et al type QTLseq analysis

QTLseq analysis

Takagi et al. (2013) developed the method for QTLseq type NGS-BSA. The analysis is based on calculating the allele frequency differences, or $\Delta(SNP\text{-}index)$, from the allele depths at each SNP. To determine regions of the genome that significantly differ from the expected $\Delta(SNP\text{-}index)$ of 0, a simulation approach is used. Briefly, at each read depth, simulated SNP frequencies are bootstrapped, and the extreme quantiles are used as simulated confidence intervals. The true data are averaged over a sliding window and regions that surpass the CI are putative QTL.

When the analysis is run the following steps are performed:

1. First the number of SNPs within the sliding window are counted.
2. A tricube-smoothed $\Delta(SNP\text{-}index)$ is calculated within the set window size.
3. The minimum read depth at each position is calculated and the tricube-smoothed depth is calculated for the window.
4. The simulation is performed for data derived read depths (can be set by the user): Alternate allele frequency is calculated per bulk based on the population type and size (F2 or RIL) $\Delta(SNP\text{-}index)$ is simulated over several replications (default = 10000) for each bulk. The quantiles from the simulations are used to estimate the confidence intervals. Say for example the 99th quantile of 10000 $\Delta(SNP\text{-}index)$ simulations represents the 99% confidence interval for the true data.
5. Confidence intervals are matched with the relevant window depth at each SNP.

Here is an example for running the analysis for an F2 population, with a bulk size of 300 individuals in each bulk. The simulation is bootstrapped 10000 times and the two-sided 95 and 99% confidence intervals are calculated:

```
df_filt <- runQTLseqAnalysis(df_filt,
    windowSize = 1e6,
    popStruc = "F2",
    bulkSize = 300,
    replications = 10000,
    intervals = c(95, 99)
    )
```

```
## Counting SNPs in each window...
## Calculating tricube smoothed delta SNP index...
## Returning the following two sided confidence intervals: 95, 99
## Variable 'depth' not defined, using min and max depth from data: 40-197
## Assuming bulks selected from F2 population, with 300 individuals per bulk.
## Simulating 10000 SNPs with reads at each depth: 40-197
## Keeping SNPs with >= 0.3 SNP-index in both simulated bulks
## Joining, by = "tricubeDP"
```

G' analysis

An alternate approach to determine statistical significance of QTL from NGS-BSA was proposed by Magwene et al. (2011) – calculating a modified G statistic for each SNP based on the observed and expected allele depths and smoothing this value using a tricube smoothing kernel. Using the smoothed G statistic, or G', Magwene et al. allow for noise reduction while also addressing linkage disequilibrium between SNPs. Furthermore, as G' is close to being log normally distributed, p-values can be estimated for each SNP using non-parametric estimation of the null distribution of G'. This provides a clear and easy-to-interpret result as well as the option for multiple testing corrections.

Here, we will briefly summarize the steps performed by the main analysis function, `runGprimeAnalysis`.

The following steps are performed:

1. First the number of SNPs within the sliding window are counted.
2. A tricube-smoothed $\Delta(SNP\text{-}index)$ is calculated within the set window size.
3. Genome-wide G statistics are calculated by `getG`. G is defined by the equation:

$$G = 2 * \sum n_i * ln(\frac{obs(n_i)}{exp(n_i)})$$

   Where for each SNP, n_i from i = 1 to 4 corresponds to the reference and alternate allele depths for each bulk, as described in the following table:

Allele	High Bulk	Low Bulk
Reference	n_1	n_2
Alternate	n_3	n_4

   ...and $obs(n_i)$ are the observed allele depths as described in the data frame. `getG` calculates the G statistic using expected values assuming read depth is equal for all alleles in both bulks:

$$exp(n_1) = \frac{(n_1 + n_2) * (n_1 + n_3)}{(n_1 + n_2 + n_3 + n_4)}$$

$$exp(n_2) = \frac{(n_2 + n_1) * (n_2 + n_4)}{(n_1 + n_2 + n_3 + n_4)}$$

$$exp(n_3) = \frac{(n_3 + n_1) * (n_3 + n_4)}{(n_1 + n_2 + n_3 + n_4)}$$

$$exp(n_4) = \frac{(n_4 + n_2) * (n_4 + n_3)}{(n_1 + n_2 + n_3 + n_4)}$$

4. G' - A tricube-smoothed G statistic is predicted by constant local regression within each chromosome using the `tricubeStat` function. This works as a weighted average across neighboring SNPs that accounts for Linkage disequilibrium (LD) while minimizing noise attributed to SNP calling errors. G values for neighboring SNPs within the window are weighted by physical distance from the focal SNP.
5. P-values are estimated based using the non-parametric method described by Magwene et al. 2011 with the function `getPvals`. Briefly, using the natural log of G' a median absolute deviation (MAD) is calculated. The G' set is trimmed to exclude outlier regions (i.e. QTL) based on Hampel's rule. An alternate method for filtering out QTL that we propose is using absolute $\Delta(SNP\text{-}index)$ values greater than a set threshold (default = 0.1) to filter out potential QTL. An estimation of the mode of the trimmed set is calculated using the `mlv` function from the package `modeest`. Finally, the mean and variance of the set are estimated using the median and mode and p-values are estimated from a log normal distribution.

6. Negative Log10- and Benjamini-Hochberg adjusted p-values are calculated using `p.adjust`.

Let's run the function:

```
df_filt <- runGprimeAnalysis(df_filt,
    windowSize = 1e6,
    outlierFilter = "deltaSNP",
    filterThreshold = 0.1)
```

```
## Counting SNPs in each window...
## Calculating tricube smoothed delta SNP index...
## Calculating G and G' statistics...
## Using deltaSNP-index to filter outlier regions with a threshold of 0.1
## Estimating the mode of a trimmed G prime set using the 'modeest' package...
## Calculating p-values...
```

As this is window is using a tricube-smoothing kernel the window size *can* be much larger than you might expect. We however choose a window size of 1Mb for the sliding window analysis, for a discussion about window size we recommend reading Magwene et al. (2011). In general larger windows will produced smoother data. The functions making these calculations are rather fast, so we recommend testing several window sizes for your data, and deciding on the optimal size.

Some additional columns are added to the filtered data frame:

```
head(df_filt)
```

```
##   CHROM   POS REF ALT AD_REF.LOW AD_ALT.LOW DP.LOW GQ.LOW     PL.LOW
## 1  Chr1 31071   A   G         34         36     70     99  897,0,855
## 2  Chr1 31478   C   T         34         52     86     99 1363,0,844
## 3  Chr1 33667   A   G         20         48     68     99 1331,0,438
## 4  Chr1 34057   C   T         38         40     78     99 1059,0,996
## 5  Chr1 35239   A   C         25         36     61     99  987,0,630
## 6  Chr1 38389   T   C         36         42     78     99 1066,0,906
##   SNPindex.LOW AD_REF.HIGH AD_ALT.HIGH DP.HIGH GQ.HIGH     PL.HIGH
## 1    0.5142857          26          22      48      99   522,0,698
## 2    0.6046512          40          34      74      99  848,0,1099
## 3    0.7058824          24          29      53      99   765,0,599
## 4    0.5128205          29          26      55      99   673,0,772
## 5    0.5901639          40          60     100      99 1632,0,1015
## 6    0.5384615          42          40      82      99  984,0,1105
##   SNPindex.HIGH   REF_FRQ     deltaSNP nSNPs tricubeDeltaSNP minDP
## 1     0.4583333 0.5084746 -0.055952381   881     -0.07365553    48
## 2     0.4594595 0.4625000 -0.145191703   881     -0.07365807    74
## 3     0.5471698 0.3636364 -0.158712542   883     -0.07367173    53
## 4     0.4727273 0.5037594 -0.040093240   883     -0.07367416    55
## 5     0.6000000 0.4037267  0.009836066   883     -0.07368154    61
## 6     0.4878049 0.4875000 -0.050656660   883     -0.07370120    78
##   tricubeDP      CI_95      CI_99          G   Gprime    pvalue
## 1        70 -0.1714286 -0.2285714 0.35697092 1.919586 0.4670188
## 2        70 -0.1714286 -0.2285714 3.38161778 1.919650 0.4669994
## 3        70 -0.1714286 -0.2285714 3.23692853 1.919994 0.4668952
## 4        70 -0.1714286 -0.2285714 0.20748641 1.920056 0.4668766
## 5        70 -0.1714286 -0.2285714 0.01521766 1.920242 0.4668204
## 6        70 -0.1714286 -0.2285714 0.41076961 1.920737 0.4666705
##   negLog10Pval    qvalue
## 1    0.3306657 0.6683592
## 2    0.3306837 0.6683425
```

```
## 3    0.3307806 0.6682994
## 4    0.3307979 0.6682904
## 5    0.3308502 0.6682467
## 6    0.3309896 0.6681446
```

- nSNPs - the number of SNPs bracketing the focal SNP within the set sliding window
- tricubeDeltaSNP - the tricube-smoothed $\Delta(SNP\text{-}index)$
- G - the G value for the SNP
- Gprime - the tricube-smoothed G value
- pvalue - the p-value calculated by non-parametric estimation
- negLog10Pval - the $-log_{10}(p\text{-}value)$
- qvalue - Benjamini-Hochberg adjusted p-values

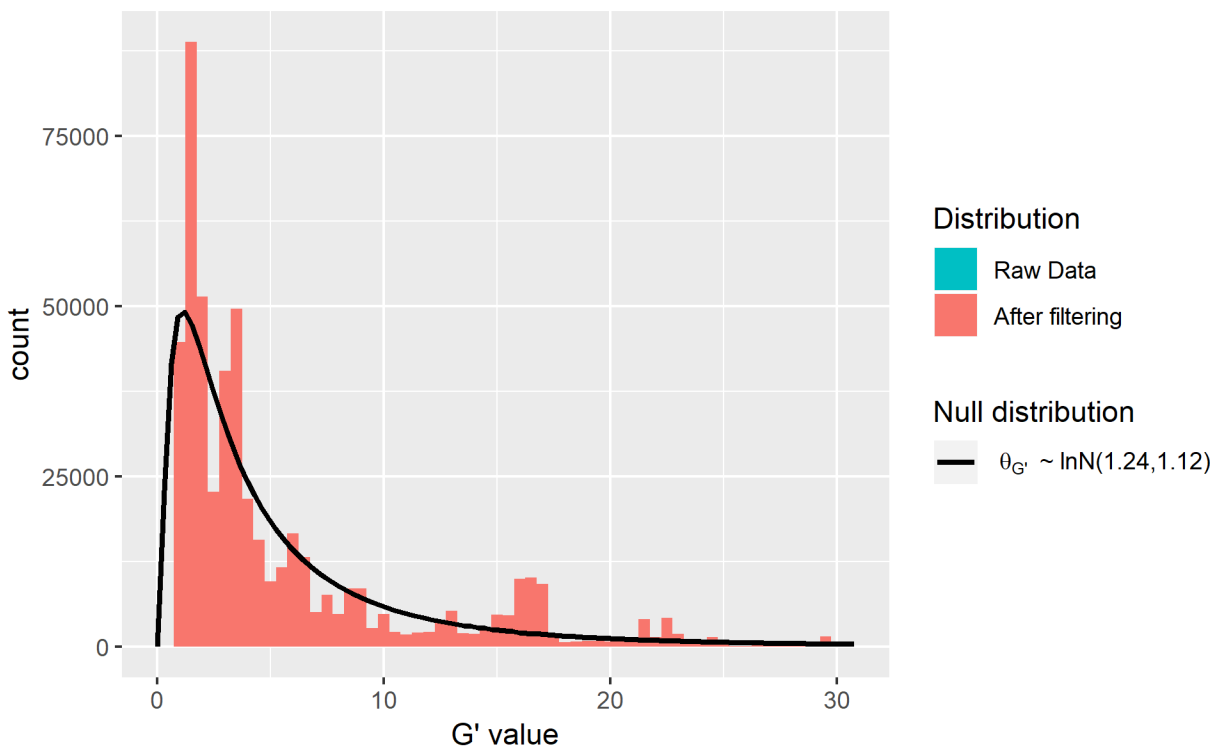
Plotting the data

QTLseqr offers two main plotting functions to check the validity of the G' analysis and to plot genome-wide or chromosome specific QTL analysis plots.

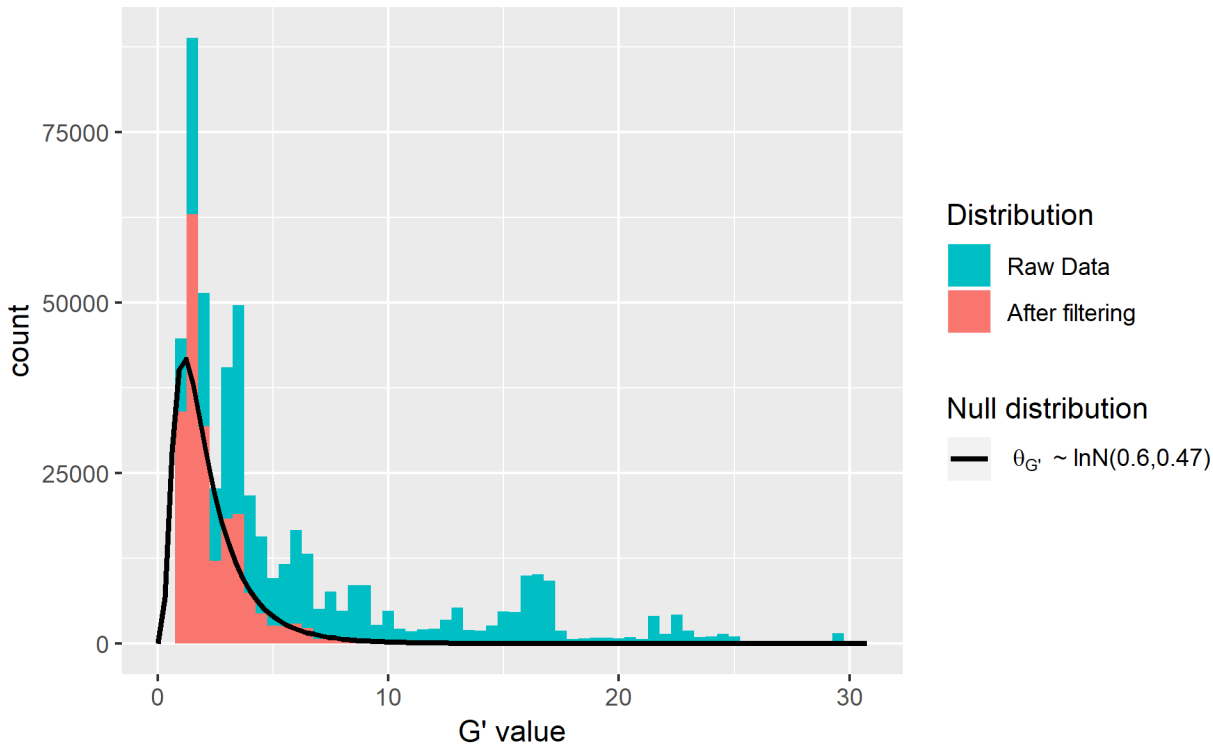
G' distribution plots

Due to the fact that p-values are estimated from the null distribution of G', an important check is to see if the null distribution of G' values is close to log normally distributed. For this purpose we use the `plotGprimeDist` function, which plots the G' histograms of both raw and filtered G' sets (see P-value calculation above) alongside the log-normal null distribution (which is reported in the legend). We can also use this to test which filtering method (Hampel or DeltaSNP) estimates a more accurate null distribution. If you use the `"deltaSNP"` method plotting G' distributions with different filter thresholds might also help reveal a better G' null distribution.

```
plotGprimeDist(SNPset = df_filt, outlierFilter = "Hampel")
```



```
plotGprimeDist(SNPset =df_filt, outlierFilter = "deltaSNP", filterThreshold = 0.1)
```

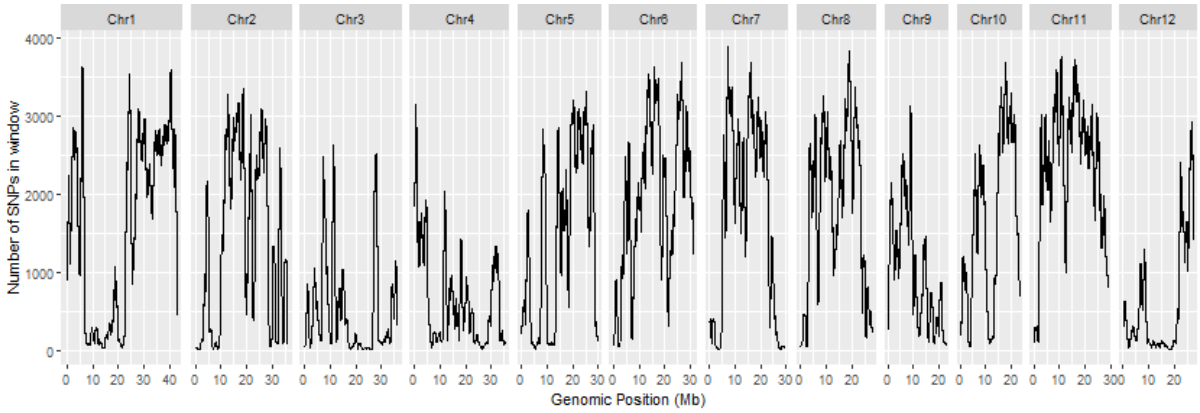


QTL analysis plots

Now that we are happy with our filtered data and it seems that the G' distribution is close to log-normal, we can finally plot some genome-wide figures and try to identify QTL.

Let's start by plotting the SNP/window distribution:

```
p1 <- plotQTLStats(SNPset = df_filt, var = "nSNPs")
p1
```

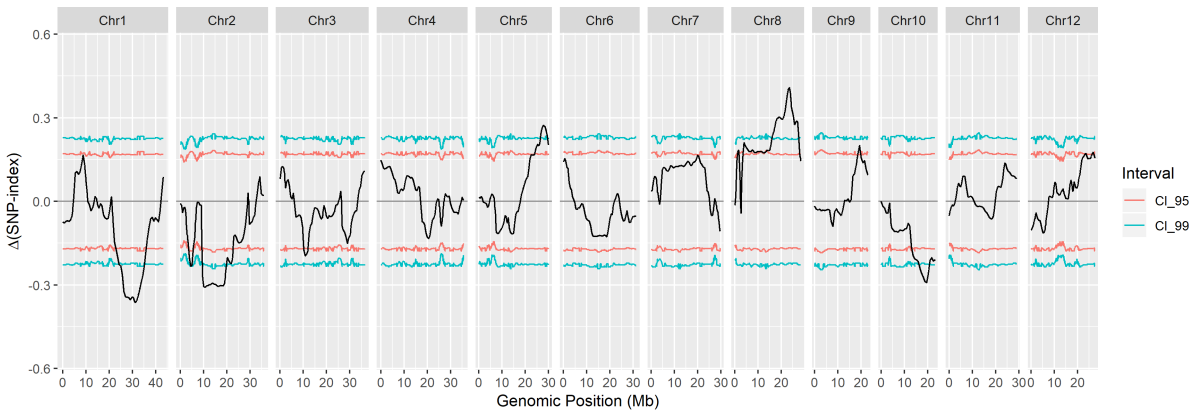


This is informative as we can assess if there are regions with extremely low SNP density.

More importantly lets identify some QTL by plotting the smoothed $\Delta(SNP\text{-}index)$ and G' values. If we've

performed QTLseq analysis we can also set `plotIntervals` to `TRUE` and plot the confidence intervals to identify QTL using that method.

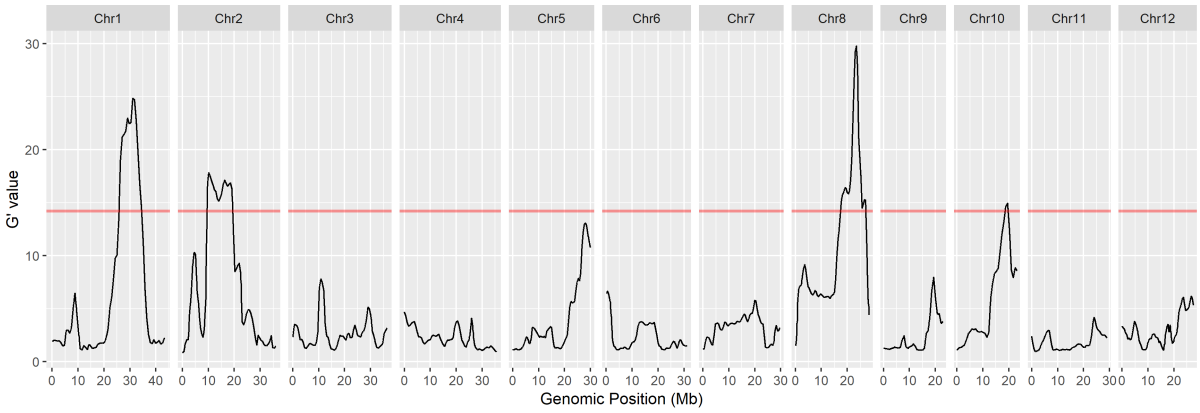
```
p2 <- plotQTLStats(SNPset = df_filt, var = "deltaSNP", plotIntervals = TRUE)
p2
```



We can see that there are some regions that have $\Delta(SNP\text{-}index)$ that pass the confidence interval thresholds, and are putative QTL. The directionality of the $\Delta(SNP\text{-}index)$ is also important for G' analysis. If the allele contributing to the trait is from the reference parent the $\Delta(SNP\text{-}index)$ should be less than 0. However, if the $\Delta(SNP\text{-}index) > 0$ then the contributing parent is the one with the alternate alleles.

Let's look at the G' values to see if these regions are significant and pass the FDR (q) of 0.01.

```
p3 <- plotQTLStats(SNPset = df_filt, var = "Gprime", plotThreshold = TRUE, q = 0.01)
p3
```

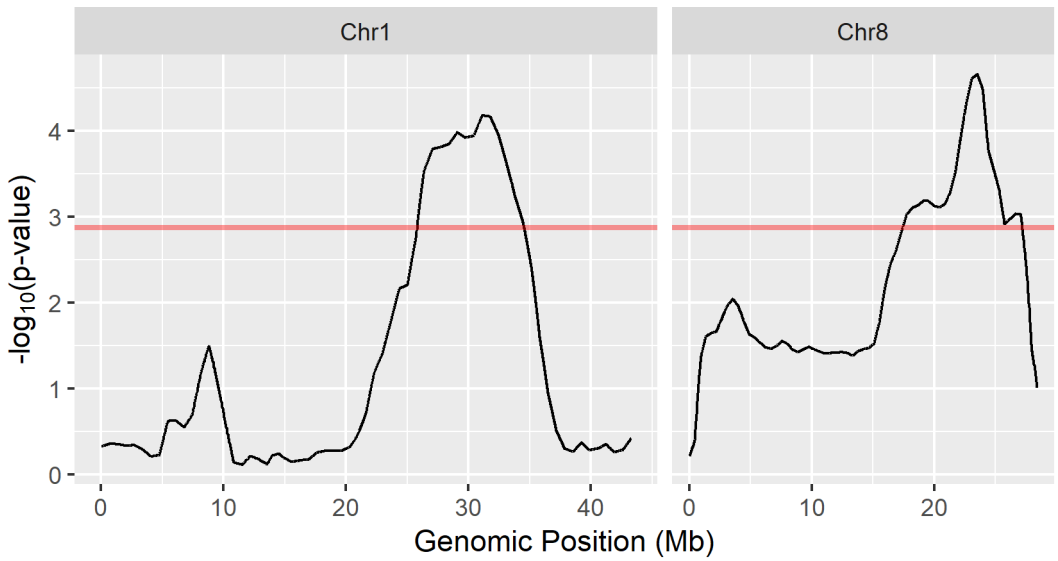


Great! It looks like there are QTL identified on Chromosomes 1, 2, 5, 8 and 10. Based on the $\Delta(SNP\text{-}index)$ and G' plots the QTL from Chr1 originates from the reference parent (Nipponbare rice, in this case) and the QTL on Chr8 was contributed by the other parent, for example.

We can also use the `plotQTLStats` function to the $-log_{10}(p\text{-}value)$. While this number is a direct derivative of G' it can be more self explanatory for some. We can use the subset parameter to plot one or a few of the chromosomes, say for a close up figure of a QTL of interest. Here we look at the $-log_{10}(p\text{-}value)$ plots of Chromosomes 1 and 8:

```
QTLplots <- plotQTLStats(
    SNPset = df_filt,
```

```
    var = "negLog10Pval",
    plotThreshold = TRUE,
    q = 0.01,
    subset = c("Chr1", "Chr8")
    )
QTLplots
```



Extracting QTL data

Now that we've plotted and identified some putative QTL we can extract the data using two functions `getSigRegions` and `getQTLTable`.

Extracting significant regions

The `getSigRegions` function will produce a list in which each element represents a QTL region. The elements are subsets from the original data frame you supplied. Any contiguous region above with an adjusted p-value above the set alpha will be returned. If there is a dip below the alpha this region will be split to two elements.

Let's examine the `head` of the first QTL:

```
QTL <- getSigRegions(SNPset = df_filt, alpha = 0.01)
head(QTL[[1]])
```

```
##   qtl CHROM      POS REF ALT AD_REF.LOW AD_ALT.LOW DP.LOW GQ.LOW
## 1   1  Chr1 25842439   T   C         21         59     80     99
## 2   1  Chr1 25844940   A   G         20         59     79     99
## 3   1  Chr1 25846683   A   G         16         47     63     99
## 4   1  Chr1 25847043   G   A         26         61     87     99
## 5   1  Chr1 25849237   G   A         12         46     58     99
## 6   1  Chr1 25851646   A   T         12         41     53     99
##       PL.LOW SNPindex.LOW AD_REF.HIGH AD_ALT.HIGH DP.HIGH GQ.HIGH
## 1 1651,0,439    0.7375000          58          47     105      99
## 2 1675,0,399    0.7468354          74          52     126      99
## 3 1342,0,326    0.7460317          48          30      78      99
```

```
## 4 1679,0,563    0.7011494          90          90     180      99
## 5 1311,0,222    0.7931034          30          20      50      99
## 6 1150,0,227    0.7735849          35          26      61      99
##       PL.HIGH SNPindex.HIGH   REF_FRQ   deltaSNP nSNPs tricubeDeltaSNP
## 1 1189,0,1546     0.4476190 0.4270270 -0.2898810  1068      -0.2762683
## 2 1270,0,1984     0.4126984 0.4585366 -0.3341370  1065      -0.2764951
## 3  768,0,1329     0.3846154 0.4539007 -0.3614164  1055      -0.2766532
## 4 2369,0,2351     0.5000000 0.4344569 -0.2011494  1055      -0.2766858
## 5   499,0,826     0.4000000 0.3888889 -0.3931034  1054      -0.2768848
## 6   650,0,915     0.4262295 0.4122807 -0.3473554  1052      -0.2771032
##   minDP tricubeDP      CI_95      CI_99         G   Gprime      pvalue
## 1    80        71 -0.1690141 -0.2253521 15.998496 14.52061 0.001185737
## 2    79        71 -0.1690141 -0.2253521 22.572856 14.54205 0.001177278
## 3    63        71 -0.1690141 -0.2253521 18.929521 14.55700 0.001171423
## 4    87        71 -0.1690141 -0.2253521  9.885982 14.56008 0.001170218
## 5    50        71 -0.1690141 -0.2253521 17.901883 14.57889 0.001162903
## 6    53        71 -0.1690141 -0.2253521 14.579060 14.59955 0.001154930
##   negLog10Pval      qvalue
## 1     2.926012 0.009145189
## 2     2.929121 0.009096584
## 3     2.931286 0.009060519
## 4     2.931733 0.009052934
## 5     2.934457 0.009008602
## 6     2.937444 0.008962478
```

Output QTL summary

While `getSigRegions` is useful for examining every SNP within each QTL and perhaps for some downstream analysis, the `getQTLTable` will summarize those results and can output a CSV by setting `export = TRUE` and `fileName = "MyQTLsummary.csv"`. We can set `method` as either `"Gprime"` or `"QTLseq"` depending on the type of analysis; `"Gprime"` will use `alpha` as FDR threshold and `"QTLseq"` will use the `interval` parameter, which should match one of the intervals calculated above.

Here is the summary for significant regions with a FDR of 0.01:

```r
results <- getQTLTable(SNPset = df_filt, method = "Gprime",alpha = 0.01, export = FALSE)
results
```

```
##   CHROM qtl    start      end  length nSNPs avgSNPs_Mb peakDeltaSNP
## 1  Chr1   1 25842439 34556923 8714484 20263       2325   -0.3631140
## 2  Chr2   1  9575918 19411590 9835672 24267       2467   -0.3081058
## 3  Chr8   1 17427057 27196250 9769193 20529       2101    0.4082090
## 4 Chr10   1 18619373 19792840 1173467  3590       3059   -0.2922478
##   avgDeltaSNP maxGprime meanGprime  sdGprime      AUCaT     meanPval
## 1  -0.3348544  24.83453   21.29574 2.6768852 60704185.7 0.0002362947
## 2  -0.3007383  17.82461   16.44684 0.6348595 22256829.6 0.0006525797
## 3   0.3215896  29.75614   18.89438 4.6368929 46533829.4 0.0005309317
## 4  -0.2888448  14.95028   14.68470 0.2098474   577793.3 0.0011258224
##      meanQval
## 1 0.005050808
## 2 0.007346565
## 3 0.006606070
## 4 0.008877309
```

The columns are:

- chromosome - The chromosome on which the region was identified
- qtl - the QTL identification number in this chromosome
- start - the start position on that chromosome, i.e. the position of the first SNP that passes the FDR threshold
- end - the end position
- length - the length in base pairs from start to end of the region
- nSNPs - the number of SNPs in the region
- avgSNPs_Mb - the average number of SNPs/Mb within that region
- peakDeltaSNP - the $\Delta(SNP\text{-}index)$ value at the peak summit
- maxGprime - the max G' score in the region
- meanGprime - the average G' score of that region
- sdGprime - the standard deviation of G' within the region
- AUCaT - the **A**rea **U**nder the **C**urve but **a**bove the **T**hreshold line, an indicator of how significant or wide the peak is
- meanPval - the average p-value in the region
- meanQval - the average adjusted p-value in the region

Summary

We've reviewed how to load SNP data from GATK and filter the data to contain high confidence SNPs. We then performed $\Delta(SNP\text{-}index)$ and G' analysis and calculate p-values and q-values based on the tricube-smoothed G' values. The QTL regions that pass our defined threshold can be stored as a list for further analysis or summarized as a table for publication.

Session info

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17134)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.0.0      bindrcpp_0.2.2     Yang2013data_0.1.0
## [4] kableExtra_0.9.0   QTLseqr_0.7.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.17      git2r_0.21.0      pillar_1.2.3
##  [4] compiler_3.5.1    plyr_1.8.4        bindr_0.1.1
##  [7] tools_3.5.1       digest_0.6.15     viridisLite_0.3.0
## [10] evaluate_0.10.1   memoise_1.1.0     tibble_1.4.2
```

```
## [13] gtable_0.2.0      lattice_0.20-35   pkgconfig_2.0.1
## [16] rlang_0.2.1       rstudioapi_0.7    commonmark_1.5
## [19] curl_3.2          yaml_2.1.19       httr_1.3.1
## [22] knitr_1.20        withr_2.1.2       dplyr_0.7.6
## [25] stringr_1.3.1     roxygen2_6.0.1    xml2_1.2.0
## [28] desc_1.2.0        gtools_3.8.1      devtools_1.13.6
## [31] hms_0.4.2         rprojroot_1.3-2   locfit_1.5-9.1
## [34] grid_3.5.1        modeest_2.1       tidyselect_0.2.4
## [37] glue_1.2.0        R6_2.2.2          rmarkdown_1.10
## [40] reshape2_1.4.3    purrr_0.2.5       readr_1.1.1
## [43] tidyr_0.8.1       magrittr_1.5      codetools_0.2-15
## [46] htmltools_0.3.6   backports_1.1.2   scales_0.5.0
## [49] rvest_0.3.2       assertthat_0.2.0  colorspace_1.3-2
## [52] labeling_0.3      stringi_1.2.3     lazyeval_0.2.1
## [55] munsell_0.5.0     crayon_1.3.4
```