# *De novo* linkage maps using R and R/qtl

*John T. Lovell*

*2017-02-23*

## Contents



---

email: johntlovell@gmail.com – website: lovelleeb.weebly.com – github: github.com/jtlovell/qtlTools

---

# 1 Introduction

The following is a tutorial to take a large genotype matrix and produce a linkage map. This task represents a major hurdle for researchers using NGS data to conduct QTL mapping. Perhaps the most important step here is to effectively cull the genotype matrix to a set of markers that accurately represent the underlying genotypes. There are a couple reasons that we need to cull the markers:

1) If recombination is absent or very rare — like between two very similar markers — genotyping errors will look like recombination events and cause marker miss-ordering.
2) The speed of map construction scales exponentially with the number of markers (since we operate on an $n$Marker - by - $n$Marker matrix), so any unnecessary markers slow the computation. For example, on a machine with 4Gb of RAM, computation of a recombination fraction matrix from 8k markers can take several hours (in a 4-way cross); such a matrix cannot be calculated for >12k markers.

We start the tutorial assuming that the researcher has a genotype matrix that has relatively few missing data points, such as the raw data from RAD-seq or GBS technology (albeit possible very large... i.e. >> 10k markers). If the genotype matrix was generated by shallow resequencing technology (e.g. Andolfatto et al. 2011) there will likely be a lot of missing data. Prior to working through the steps presented here, missing data from such approaches should be (at least partially) imputed. I usually do not pass markers with >20% missing data into a method for genetic map construction.

## 1.1 The R/qtl and qtlTools package

The R/qtl package is the premier statistical software for QTL mapping. The authors have also developed several tools to assist in genetic map construction. The pipeline below employs several functions within R/qtl:

Table 1: R/qtl functions called by qtlTools map building protocols

| | |
|---|---|
| est.map | Estimate genetic map |
| est.rf | Estimate pairwise recombination fractions |
| formLinkageGroups | Partition markers into linkage groups |
| ripple | Check for locally better marker orders |
| drop.markers | Remove a list of markers from a cross |
| geno.table | Test for segregation distortion |
| droponemarker | Look for markers that expand the genetic map |

The `qtlTools` package expands upon these functions and permits automated application on large genotype matrices with (potentially) anonymous markers. In particular, we discuss several protocols:

Table 2: qtlTools functions to build genetic maps from NGS data

| | |
|---|---|
| geno2cross | Convert a genotype matrix to R/qtl cross format |
| newLG | Re-make a cross object, ordering markers according to a list |
| dropSimilarMarkers | Choose the best marker from a set of very similar markers |
| tspOrder | find the marker order that limits the total recombination fraction |
| repPickMarkerSubset | pick the best marker in a window across the genome |
| repRipple | run the ripple method to find the best marker order on all chromosomes |
| reDoCross | a pipeline that runs repPickMarkerSubset and repRipple |
| matchMarkerOrder | flip linkage groups are reversed relative to an annotation |

# 2 Setup

## 2.1 Overview

We will employ a set of tools developed by Karl Broman and published in the `qtl` package. The `qtlTools` package, available on github (https://github.com/jtlovell/qtlTools), contains the necessary functions to order and cull markers. We will also use `ggplot2` to visualize some patterns, `TSP` to order markers and `devtools` to update the qtlTools package.

## 2.2 Install qtlTools

Before we get started, make sure you have all these packages: qtl, qtlTools, TSP, ggplot2, devtools. All except qtlTools can be installed from CRAN. To get qtlTools:

```
library(devtools)
install_github("jtlovell/qtlTools")
library(qtlTools)
```

## 2.3 Load all the packages

```
libs2load<-c("qtl","qtlTools","TSP","ggplot2")
suppressMessages(sapply(libs2load, require, character.only = TRUE))
```

```
##      qtl qtlTools      TSP  ggplot2
##     TRUE     TRUE     TRUE     TRUE
```

*Loads each package in a single line - equvialent to running* `require()` *or* `library()` *for each package separately*

## 2.4 Load the genotype data

See appendix 1 for the script to generate the simulated genotype matrix

# 3 Converting a genotype matrix to an R/qtl cross object

## 3.1 Overview

This tutorial assumes that a genotype matrix has been generated and alleles have been coded following R/qtl specifications. The matrix must contain one individual/row and one marker/column. For details see `?read.cross`. Here, we analyze a simulated mapping population with a genotype matrix called `genomat`.

Table 3: The first 5 individuals and 12 markers in the simulated dataset. **Note** that we ordered the simulated the data agnostic to the original chromosome. However, we stored the known positions in the names of markers

| | 1_728 | 3_563 | 1_194 | 2_59 | 2_75 | 3_644 | 1_517 | 2_84 | 3_731 | 3_462 | 2_202 | 1_632 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | B | B | B | B | B | B | A | B | B | A | A | B |
| 2 | B | B | B | A | A | B | B | A | B | B | A | B |
| 3 | A | A | B | A | A | A | B | A | A | A | B | A |
| 4 | B | B | B | A | A | B | B | A | A | B | A | A |
| 5 | B | B | A | B | B | B | A | B | B | B | B | A |

## 3.2 Marker Names

It is helpful to store individual ids in the `rownames` and marker IDs in the `colnames` of the matrix. If physical positions are known about the markers, renames them as chr_position (e.g. marker @ Chr1 and 150kb might be '1_150'). Here, extract the chr and position information from the marker names and the RIL line IDs from the rownames using qtlTools `splitText`, which takes a character string and splits it by a character. It is a wrapper for `strsplit` and is similar to excel's `text to columns`.

```
chr<-splitText(colnames(genomat), sep = "_", num = 1)
pos<-splitText(colnames(genomat), sep = "_", num = 2)
ids<-rownames(genomat)
```

## 3.3 Build an R/qtl cross file

R/qtl takes a very specific genotype matrix type; `geno2cross` helps make this happen.

```
# fake chromosome, since we are pretending we don't know the chromsome identity
chr = rep(1,ncol(genomat))
pos=1:ncol(genomat)/10 # fake position too.
geno2cross(genomat = genomat, chr=chr,  pos=pos,  id=ids,
           crossfile = "~/Downloads/cross.csv")
```

## 3.4 Read the genotype matrix into R/qtl

```
cross<-read.cross("csv", file = "~/Downloads/cross.csv", crosstype = "riself",
                  genotypes = c("A","B"), na.strings = "NA")
```

```
##  --Read the following data:
##   250  individuals
##   2500  markers
##   1  phenotypes
##  --Cross type: riself
```

# 4 Cull and process the genotype matrix in R/qtl

## 4.1 Overview

For all downstream calculations, it is important that each linkage group contains no more markers than is necessary. The number needed depends on the number of individuals in your mapping population and the length/n crossovers (cM) for each chromosome. A general rule of thumb is that you want no more than 1 marker for each cM and and each 100 individuals. As such, each recombination event is covered by one marker. You can get this, using the following equation:

`nmar = (nind(cross)/100)*totlen(cross)`

We can also think about this in terms of the fraction of individuals that recombine between any two markers: the 'recombination fraction'. The maximum recombination fraction that gives us all possible recombination events is

`1/n.individuals` or `1/nind(cross)`

And the minimum mapping distance between any marker pairs can then be determined as: `100/nind(cross)`

*It is important to note that if the error rate of genotyping is high, it is much better to have fewer high-confidence markers than a dense map ... At least for the purposes of constructing a genetic map.* **See appendix 2 for an explanation and simulation.**

```
print(nind(cross)) # number of individuals
```

```
## [1] 250
```

```
print(min.rf<-1/nind(cross)) # minimum recombination fraction
```

```
## [1] 0.004
```

```
print(min.cm<-100/nind(cross)) # minimum distance between any two markers
```

```
## [1] 0.4
```

## 4.2 Drop markers that are too close

To drop close markers, we use the function `dropSimilarMarkers`. We set the minimum recombination fraction at the threshold determined above. For the purposes of this simulation, we will use a minimum recombination fraction = 0.02. `dropSimilarMarkers` calculates a pairwise recombination fraction matrix, which is very memory intensive for large genotype matrices. Therefore we first run chromosome-by-chromosome or block-by-block. Here, we are pretending to not know the physical positions of the markers - they are placed in a single linkage group and not ordered therein. As such, we run blocks of 500 markers to get the matrix down to a reasonable size, then run for the whole matrix.

```
cross2<-dropSimilarMarkers(cross, blockSize = 500, byChr=F,
                           rf.threshold = .02,
                           runFullMatrix = T)
```

```
## initial n markers = 2500
## parsing cross object into blocks
## running on 5 blocks of 500 markers ...
## block: 1 2 3 4 5
## n markers after block-wise culling: 984
## running for the whole matrix of 984 markers
## n markers after whole-map culling: 273
```

# 5 Place markers into linkage groups

We are now ready to estimate recombination fractions for the entire matrix and bin markers into linkage groups.

## 5.1 Overview

Even if we are pretty sure that all markers are assigned to their correct linkage groups (chromosome), this step is still important - if a marker is incorrectly placed it will cause major problems. For example, if a marker is thought to be on Chr1, but is actually on Chr2, it will be uncorrelated with all markers on Chr1. This will cause a huge observed fraction of individuals that recombine and will expand the map, making inference of QTL much more difficult and imprecise.

## 5.2 Group markers into arbitrary linkage groups

The R/qtl function `formLinkageGroups` uses the recombination fraction among markers and the strength of the LOD score of recombination fractions to determine linkage groups. Both the maximum rfs and minimum LOD scores can be adjusted to get the markers into good groups. This step might be more of an art than science and requires adjustment of both parameters. Depending on the population, the researcher may adjust the max.rf to form the number of linkage groups that are appropriate for the study organism.

```
lgmar<-formLinkageGroups(cross2, max.rf=.23, min.lod=3)
```

## 5.3 Rename linkage groups

In a typical reference-based genotyping approach, we have a pretty good idea of the chromosome on which a marker resides, however, we might be wrong some small (or large) % of the time. If the markers are completely anonymous, we can skip this step. Remember the name of the original chromosome is stored in the name of the marker.

```
lgmar$origchr<-splitText(rownames(lgmar)) # extract chromosome ids from the marker names
tab<-table(lgmar) # tabulate the original and new marker identity
```

We can visualize the overlap by looking at the tabulation of the new linkage groups and the original chromosomes.

Table 4: The original and inferred linkage group identity of each marker (tabulated). **Note** that all markers from the original linkage groups cluster together, however the names are not the same.

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 92 | 0 |
| 2 | 119 | 0 | 0 |
| 3 | 0 | 0 | 62 |

## 5.4 Rename linkage group names based on maximum overlap

For each row in the tabulation of linkage groups `tab`, we ask, 'which new linkage groups is best represented?'.

```
newnames<-sapply(1:nchr(map),
                 function(x) rownames(tab)[which.max(tab[,x])])
```

## 5.5 Reassign markers to each linkage group

We then assign the markers in each linkage group to the corresponding name in the original map.

```
marlist<-lapply(1:nchr(map),
                function(x) rownames(lgmar)[lgmar$LG==x])
```

For the next step, it is important to produce a list of markers where each list element is named as the ordered linkage group ID and the markers are in order within each list element. Here we produce such an object.

```
names(marlist)<-newnames
marlist<-marlist[order(as.numeric(names(marlist)))]
```

## 5.6 Reorganize the cross with new linkage group names and marker groupings

The function `newLG` takes the cross object and the markerlist generated above to reorganize markers.

```
cross2<-newLG(cross = cross2, markerList = marlist)
```

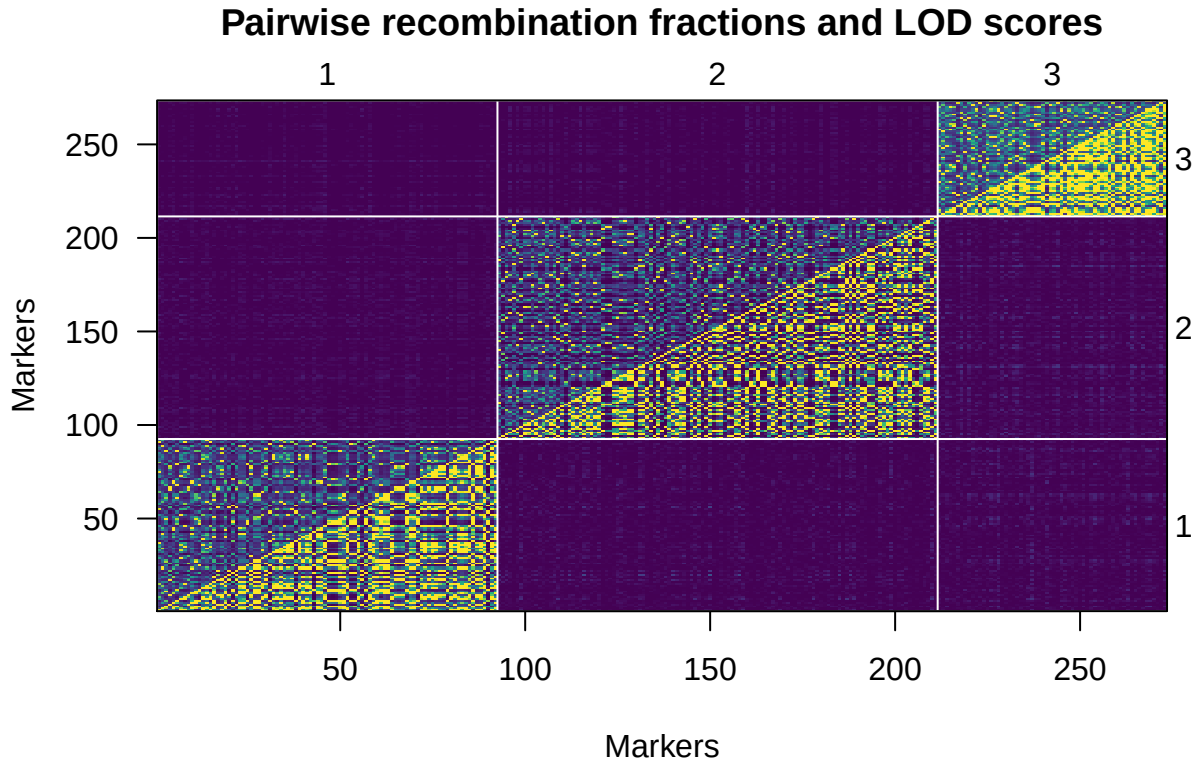


Figure 1: Pairwise recombination fraction plot following linkage group construction, but before marker ordering. Note that recombination fractions are always high between chromosomes (blue). All pairs of highly correlated markers (low recombination fractions, yellow) exist on the same chromosome. Since marker order is random, it is not surprising that close positions on a chromosome do not show low recombination fractions

# 6 Reorder markers

## 6.1 Overview

There are many methods to order markers. While joinmap4 remains the industry standard, it has drawbacks - it's slow and is not free. Recently several other methods have been proposed using graph theory. MSTmap is a good option, but it can only handle populations with 2-genotype markers (BC/RIL/DH/etc), not F2, 4-way etc. mapping populations. To overcome these issues, it is optimal to assess marker orders through evaluation of the recombination fraction matrix only.

## 6.2 Marker ordering using TSP solvers

Here, I build upon the `TSPmap` protocol and employ a travelling salesperson problem solver to find the shortest path through the recombination fraction matrix. The optimal method is `concorde`, however, this requires a separate installation of the concorde program. See `?tspOrder` for details on how to do this.

```
cross3<-tspOrder(cross = cross2,hamiltonian = TRUE,
                 method="concorde",concorde_path = "/Users/John/Documents/concorde/TSP")
```

## 6.3 Look at marker order

```
cross3<-est.rf(cross3)
```

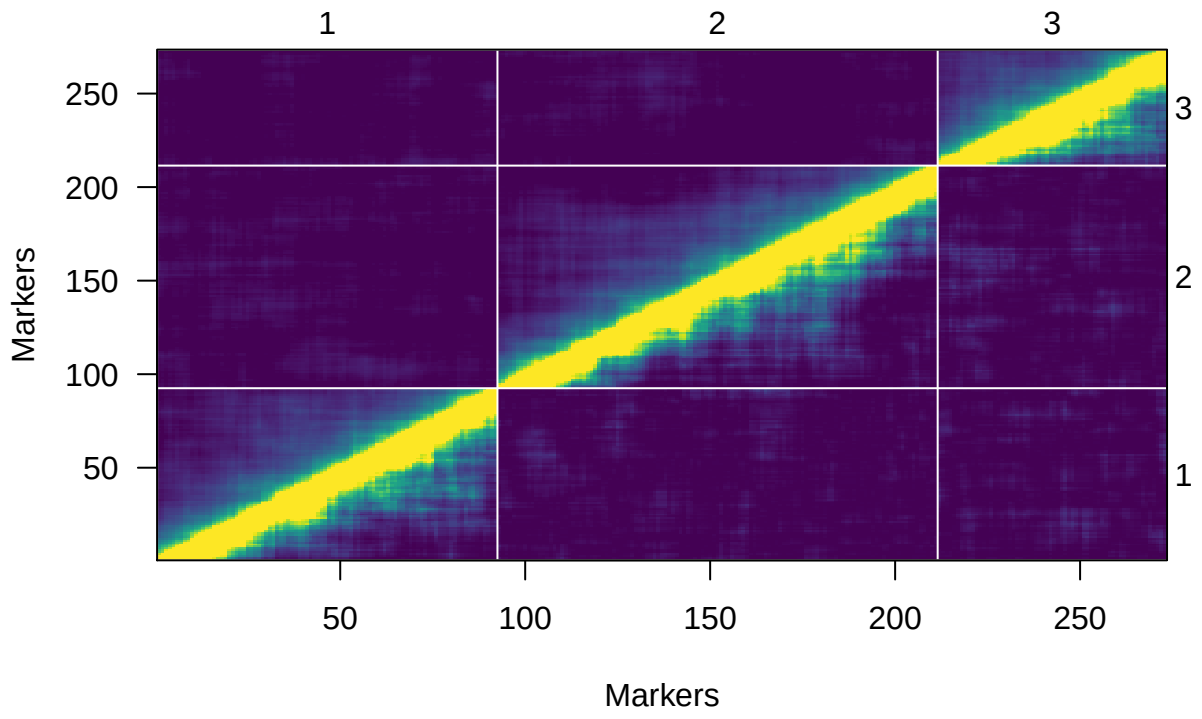


Figure 2: Pairwise recombination fraction plot following marker ordering. Note that following the marker reordering, there is a decay in correlations between markers from the diagonal. This is an indication of a high-quality map, since distant markers are not correlated.

# 7 Finalize the genetic map

## 7.1 Overview

TSP and other marker ordering problem solvers (e.g. MST, joinMap4 etc.) make mistakes. While the data used here is simulated, there can be small marker order problems. Sometimes a few problematic markers need to be removed, then marker ordering will work perfectly. Other times, we just need to fine-tune the marker order.

## 7.2 Refining the genetic map

The first thing to do, once marker order is established is to estimate the genetic map with `est.map`. Then, with cM distances calculated, cull the map to markers that are evenly spaced with `repPickMarkerSubset`. Finally, we can check for better marker orders using `ripple`.

```
cross3<-est.map(cross3, map.function = "kosambi")
rest.ril3<-repPickMarkerSubset(cross3, min.distance = 2)
rest.ril3<-repRipple(cross3, window = 4)
```

## 7.3 One-step map refining

I have piped these functions into a single call: `reDoCross`.

```
cross4<-reDoCross(cross3, min.distance = 2, window = 4)
```

```
## initial estimation of map
## drop close markers --- tossed 123 markers
## ripple marker order --- reduced map size by 7.741689
## drop close markers --- tossed 1 markers
## final map estimation
```

## 7.4 Look for and drop markers that have more recombination events than would be expected

Sometimes we observe markers that seem to be placed in the right position, but cause map expansions, or show more potentially erroneous calls than others. We can use the `qtl` function `droponemarker` to check for markers that, when excluded produce a more streamlined map. Since these data are simulated, we do not find any

```
dropone<-droponemarker(cross4,
                       map.function = "kosambi",
                       sex.sp = F,    verbose = F)
```

Using this output, we can screen for aberrant markers using `dropByDropone`. Since these data are simulated, we do not expect to find any markers to drop, but often, in real data this helps improve the map.

```
cross4<-dropByDropone(cross=cross4,
                      droponeRes = dropone,
                      midMarkerThresh = 1,
                      endMarkerThresh = 20)
```

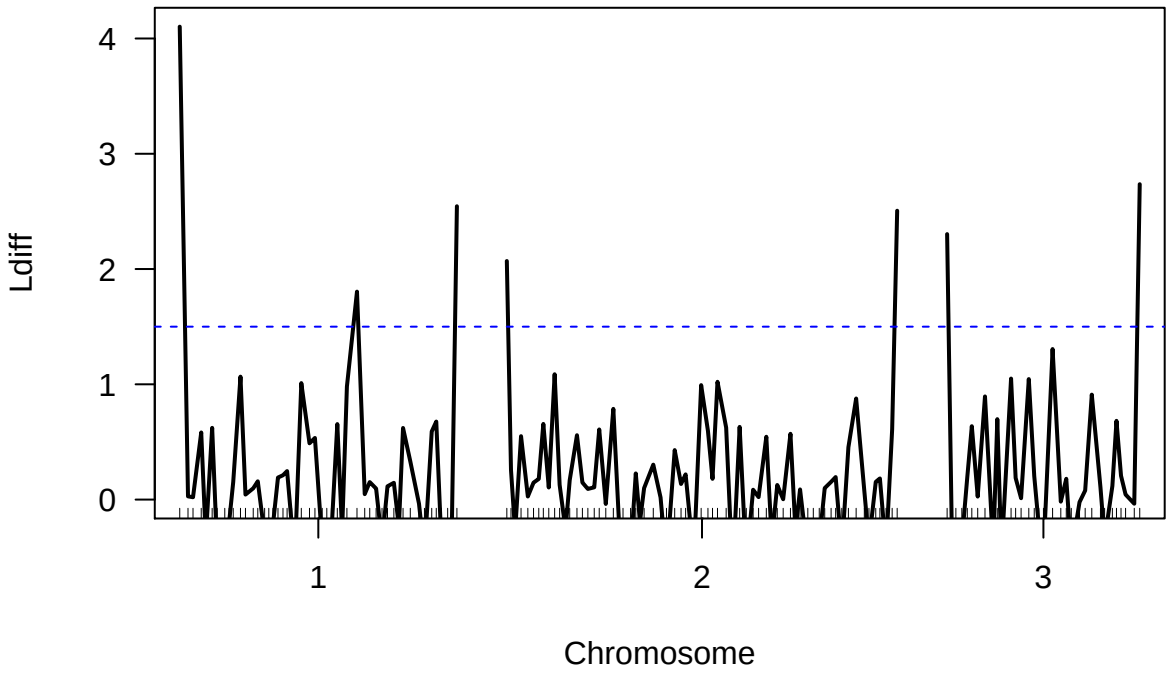


Figure 3: Plot of the relative map expansion when each marker is included. Markers on the ends of chromosomes tend to effect this since removing them reduces the coverage on a given chromosome. Note however that there is a marker on the interior of Chr1 with a value >1.

### 7.5 Find and drop markers with strong segregation distortion

Some markers may be biased towards one allele or another - while many of these may have been culled previously (`dropSimilarMarkers` and `repPickMarkerSubset` can both choose the marker with the least segregation distortion), some may remain. It is important to qualify that segregation distortion can be a biological reality, due to selection during population development or reproductive incompatibility between alleles (e.g. DM incompatibilities). In this case, we expect an gradual increase, then decline in the extent of segregation distortion around a selected marker. However, segregation distortion can also be caused by genotyping errors, due read-mapping bias or similar. In this case, we may expect one or a few markers to show segregation distortion while surround markers do not. This distinction is important because the first - selection induced SD - is unavoidable and an attribute of the population. However the latter can and should be culled. Doing so will reduce the map size and improve the efficacy of QTL mapping.

So, we should cull the highly distorted markers, but be careful to only remove those that we believe are due to genotype errors. Here is one way to do this:

```
gt<-geno.table(cross4, scanone.output=T)
toDrop<-rownames(gt)[gt$AA>.58 | gt$BB >.58]
cross5<-drop.markers(cross4, toDrop)
```

```
toDrop<-rownames(gt)[gt$AA>.58 | gt$BB >.58]
cross5<-drop.markers(cross4, toDrop)
```

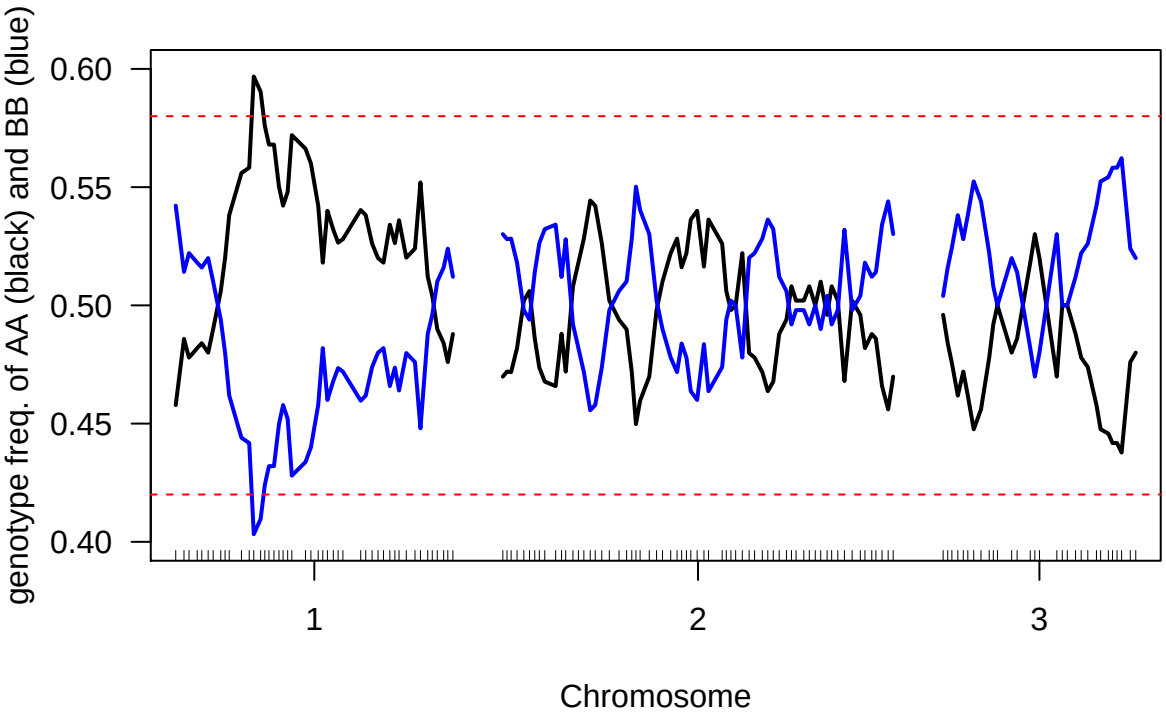


Figure 4: Segregation distortion plot prior to filtering. Threshold have been arbitrarily set to a minor allele frequency $> 0.42$

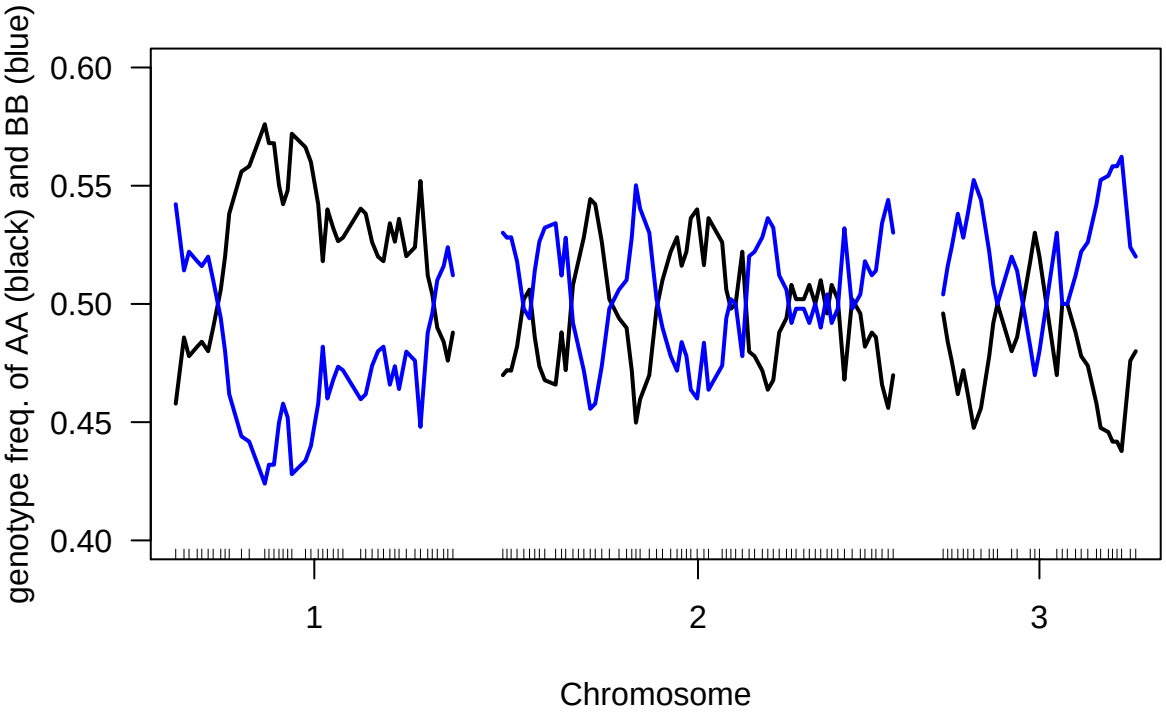


Figure 5: Segregation distortion plot after filtering markers with a major allele frequences $> 0.58$

# 8 Match the order of markers to known orders (e.g. annotation)

We often have a pretty good idea of the general order of markers. For example, what were the positions of the markers on the reference genome? Or where do the reads BLAST to on a related genome? When generating a genetic map, we often will want to make sure the map reflects know marker orders. To do this, we need to have some information about the chromosome and positions of marker - e.g. the position of the markers on some reference.

```
cross5<-matchMarkerOrder(cross4, plotit=F)
```

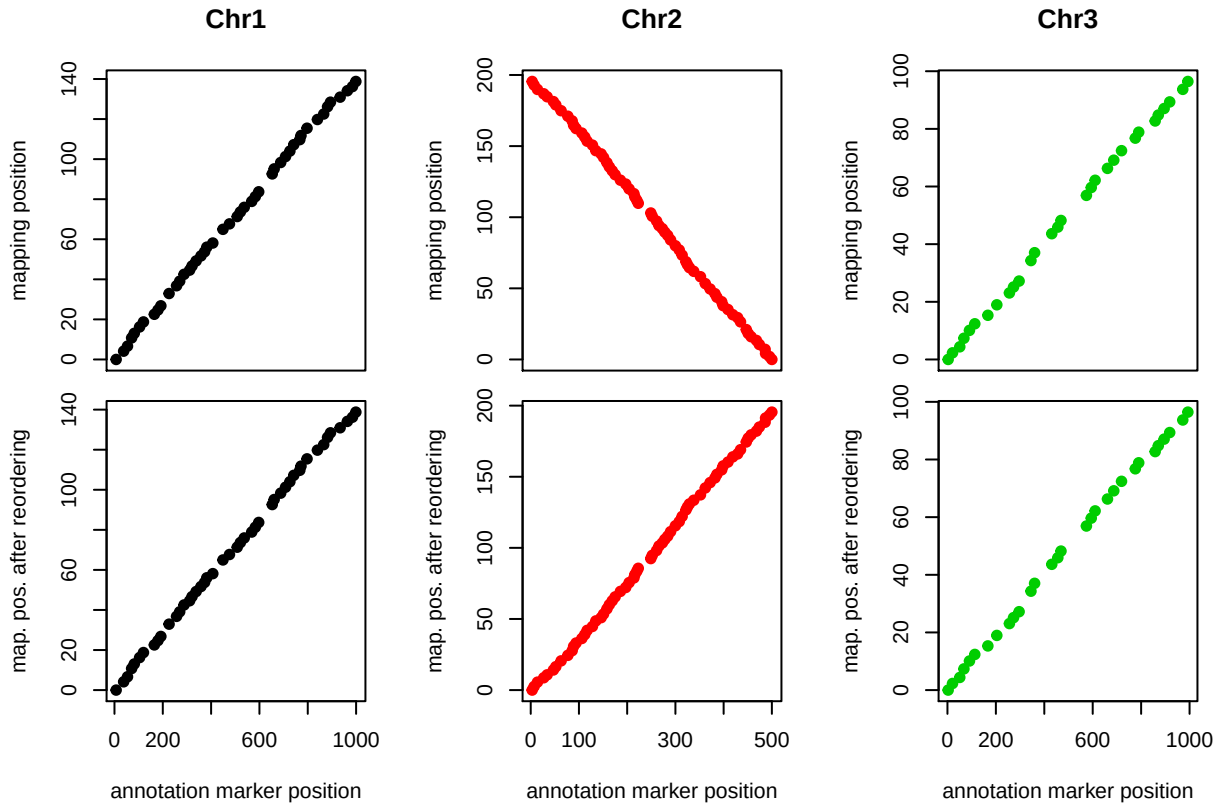


Figure 6: original (top) and new (bottom) marker order. Note that the initial slopes in the first plot may be negative (i.e. the marker order is inverted relative to the reference); however, after matchMarkerOrder the slopes are always positive

# 9 Add phenotypes to the cross and run QTL mapping

Typically you will now want to run QTL mapping using the cross. It is simple to add phenotypes. Here I will just simulate some data - all you need is a column with the genotype identifier (`id` here) and columns with the phenotype data.

```
phe<-data.frame(id = getid(cross5),
                phe = rnorm(nind(cross5)),
                stringsAsFactors=F)
```

Make sure the phenotype ids match the cross ids

```
phe<-phe[match(getid(cross5), phe$id),]
```

Add the phenotype data to the cross - now you are ready to do QTL mapping.

```
cross5$pheno<-data.frame(cross5$pheno, phe)
```

# 10 Other considerations

This is where I usually stop. If you have done a good job of pre-filtering markers (e.g. vcfTools), the map should be in publishable shape. However, sometimes you may want to add a few markers to the telomeres of chromosomes, or fill in gaps. In this case, I go back to the original matrix and add the markers there. If they add information to the map, they will be retained.

Aside from QTL mapping, genetic maps can be used to inform the physical positions and orientation of scaffolds when building genome assemblies. There are a few ways to accomplish this goal. There will be a tutorial on this coming in the next year or so.

Good luck.

# 11 Appendix 1: Make simulated data

## 11.1 Make some fake data that mimics what a real genotype matrix might look like

```
set.seed(42)
# make a map with 3 chromosomes
map <- sim.map(len=c(150,200,100),
               n.mar=c(1000,500,1000),
               include.x =FALSE, sex.sp =F)
# change the names to chromosome _ position.
for(i in 1:length(map))
  names(map[[i]])<-sapply(names(map[[i]]),
                          function(x) paste0(substr(x,2,2),
                                             "_",
                                             substr(x,4,nchar(x))))
# simulate 250 line RIL w/ random positions of markers and unknown chromosomes
cross <- sim.cross(map, type="riself", n.ind=250, map.function = "kosambi",
                   error.prob = 0, keep.qtlgeno = F,missing.prob=0.01)
# randomize the positions of markers
cross<-newLG(cross, markerList = list("a" = sample(markernames(cross))))
# code it as characters
genomat<-apply(pull.geno(cross),2,function(x) ifelse(is.na(x),NA,ifelse(x ==1 ,"A","B")))
# add information to the row names
rownames(genomat)<-1:nrow(genomat)
```

# 12 Appendix 2: Understanding the effect of genotyping errors.

Take the case of two markers. The probability of a genotyping error `P(E)` for each is identical. The observed proportion of recombinant individuals (cM/100) between markers is `d`, which can be due to genotyping error or true crossover events. We can then calculate the probability of true crossovers `P(XO)` as:

P(XO) = (1 - P(E)) * d

and the probability of false crossovers, which are caused by errors as:

P(XOe) = P(E) * (1 - d).

Therefore, the proportion of true crossover individuals as:

P(XOt) = P(XO) / (P(XOe) + P(XO)).

To illustrate this, lets compare distance between markers and error rates.

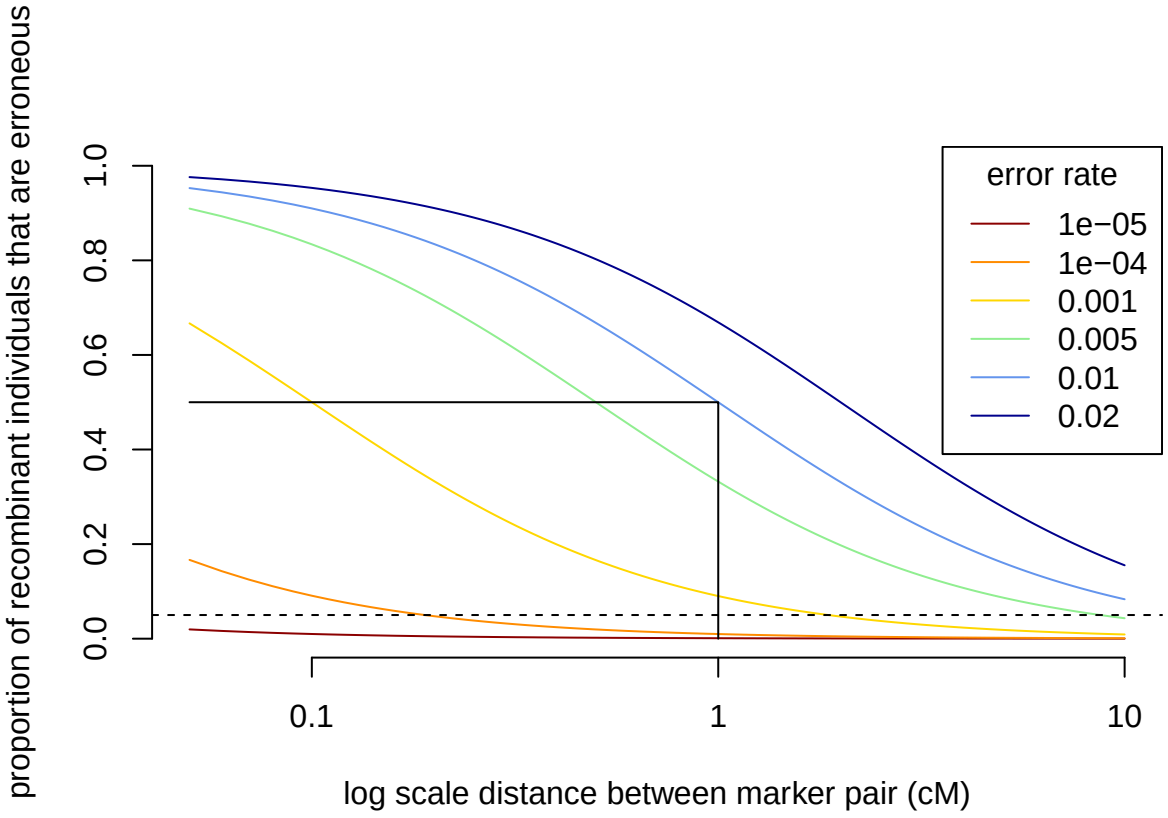


Figure 7: The effect of marker density and genotype error rate on the likelihood of calling an erroneous cross-over event. Here, we plot the likelihood that an observed cross-over event is actually a genotyping error between any two pairs of markers. The black line indicates that for an error rate of 1%, 1/2 of all crossover events between markers 1cM apart are erroneous.