Make Linkage Maps from NGS

John T. Lovell

2017-10-23

Contents

---

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

---

1 Part 1: Overview

For this tutorial, we will be using some real data, collected from an F2 mapping population of *P. hallii*. To get the data, install the `qtlToolsTutorials` and `qtlTools` packages:

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

data(markerData)
data("makeGeneticMapTutorialTmpData")
```

Table 1: example of 7 libraries and 10 markers from the markerData dataset. Each value represents the proportion of reads mapping to the reference genome vs. the alternative genome for a given marker 'HAL2' and library (column names)

	HAL2	chr_HAL2	pos_HAL2	SCGC	SCGG	SCGH	SGUT	SNNB	SNNC	SOAU
1	PhHAL.1G000100	Chr01	14202	1.00	0.00	0.32	1.00	0.98	0.31	0.99
11	PhHAL.1G002700	Chr01	230654	1.00	0.00	0.40	1.00	0.99	0.38	1.00
21	PhHAL.1G004300	Chr01	324119	1.00	0.01	0.48	0.99	0.98	0.50	1.00
31	PhHAL.1G005700	Chr01	412281	1.00	0.00	0.60	1.00	1.00	0.00	1.00
41	PhHAL.1G007100	Chr01	492795	1.00	0.00	0.45	0.99	1.00	0.49	0.98
51	PhHAL.1G010000	Chr01	710712	1.00	0.03	0.51	1.00	0.96	0.44	1.00
61	PhHAL.1G011700	Chr01	782252	0.99	0.00	0.59	1.00	0.99	0.57	0.99
71	PhHAL.1G012900	Chr01	841054	1.00	0.00	0.50	1.00	0.99	0.49	1.00
81	PhHAL.1G015000	Chr01	938411	1.00	0.01	0.57	1.00	0.99	0.54	1.00
91	PhHAL.1G016200	Chr01	1033921	1.00	0.01	0.50	1.00	0.98	0.51	0.99

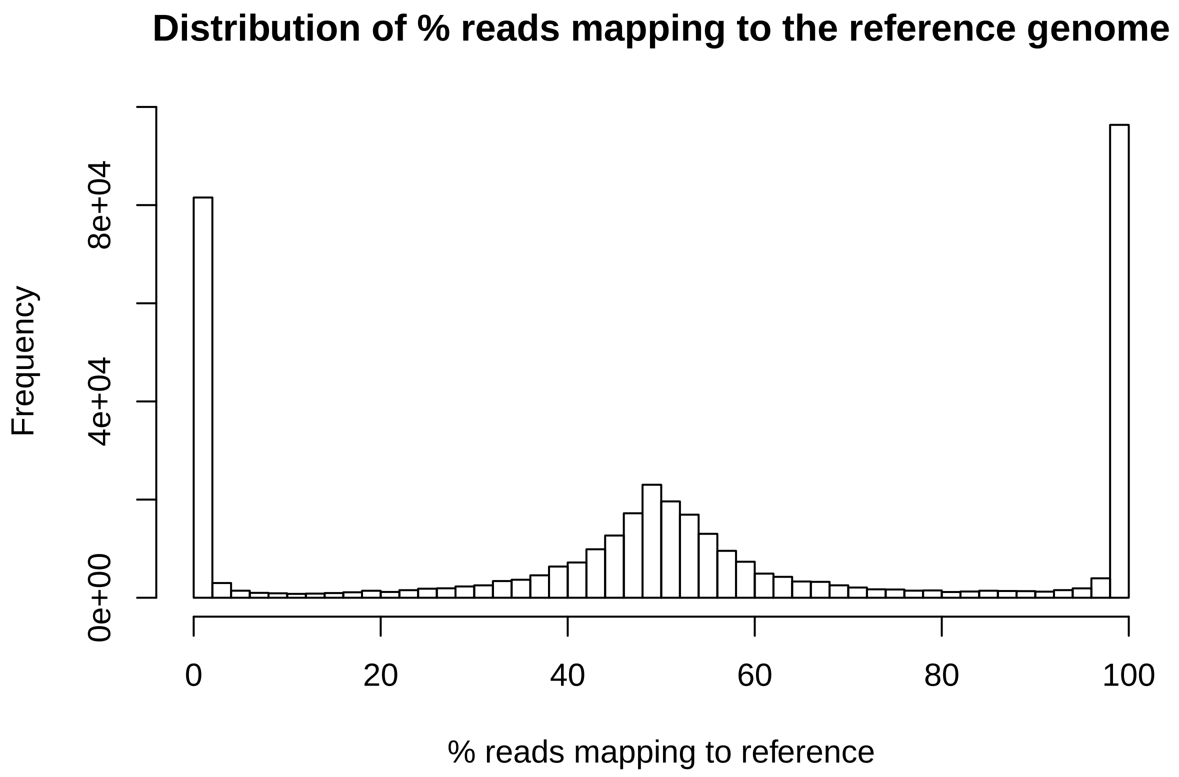


Figure 1: distribution of reads mapping to the HAL2 P. hallii reference genome. Since it is an F2, We expect that most loci have a 1:2:1 AA:AB:BB ratio.

2 Part 2: Make genotype calls

The hardest part of building a genetic (linkage) map is getting high-quality markers in a dense and consistent grid across the genome. NGS data can produce erroneous single marker calls, or results that look erroneous, but are actually OK. For example, it is difficult on a single-marker basis to distinguish between mapping bias and segregation distortion.

A good way to get around missing / erroneous data is to call markers in windows (or using imputations). Here, we use a qtlTools function `swGenotype`, where markers are binned and concensus genotypes are called. This is a nice approach if you have a good idea of the physical position of markers. However, if the markers are entirely ambiguous, other, more complicated approaches are needed.

`swGenotype` takes three vectors (chromosome id, physical position and marker id) and a matrix of the proportion of reference alleles with rows that match the chr, pos and marker.id vectors.

```
mar.chr = markerData$chr_HAL2
mar.pos = markerData$pos_HAL2
mar.id = markerData$HAL2
prop.ref = markerData[,-c(1:3)]
```

```
sw<-swGenotype(reference.prop = prop.ref,
               chr = mar.chr,
               pos = mar.pos,
               marker.id = mar.id)
```

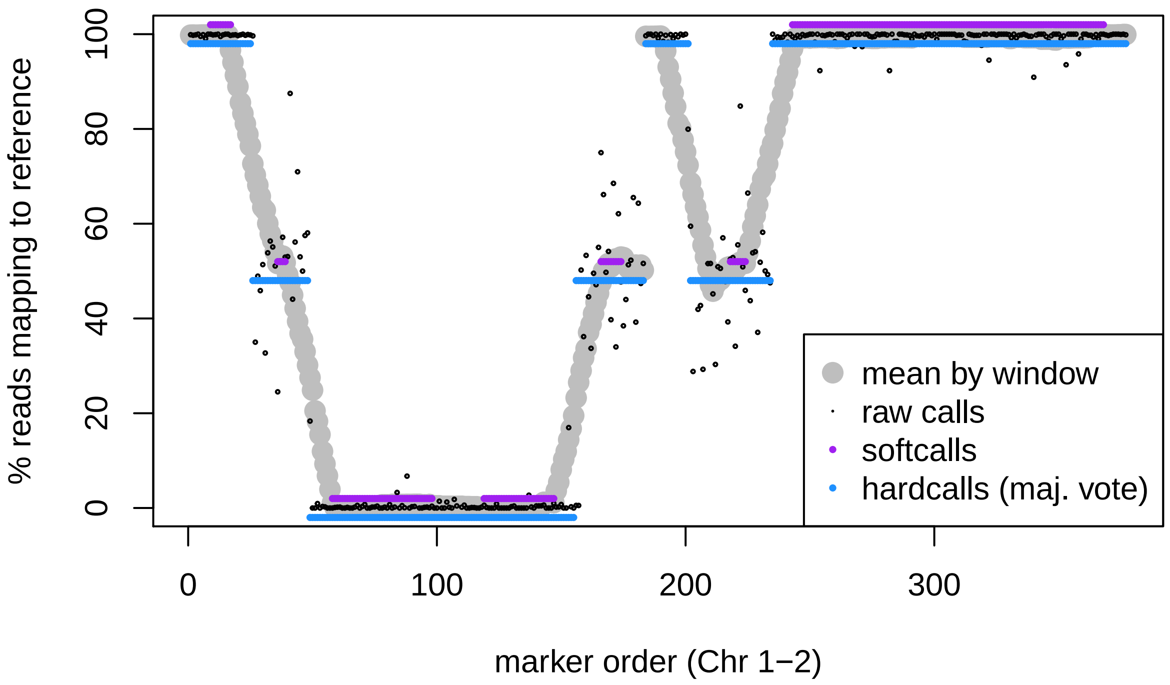


Figure 2: Distribution of marker calls across the first two P. hallii chromosomes. We retain the hard calls (blue), which are the majority vote for any 20-marker window.

3 Part 3: Reading markers into R/qtl

3.1 Overview . . .

We now have a very complete, highly accurate genotype matrix. With markers coded as the proportion of reads (0, .5, or 1) mapping to the reference. We now want to convert these to 0/1/2 coding and make sure that all markers are unique (there is no use for identical markers in linkage map construction).

We also want to store the known location of each marker in the rownames of the calls. It is best to score the names as chr_pos. This allows us to access the physical positions later.

3.2 Extract calls, rename rows and drop metadata . . .

```
hard.calls<-sw$hardCalls
rownames(hard.calls)<-paste0(gsub("Chr0","",hard.calls$chr),"_",hard.calls$pos)
hard.calls<-hard.calls[,-c(1:3)]
```

3.3 Drop identical markers and transpose the matrix . . .

```
hard.calls<-hard.calls[!duplicated(hard.calls),]
hard.calls<-t(hard.calls)
```

3.4 Recode the markers as 0/1/2 and save the matrix . . .

geno2cross writes the genotype matrix into a R/qtl cross object, and parses the marker names into chromosome and position vectors.

```
hard.calls[hard.calls==1]<-2
hard.calls[hard.calls==0.5]<-1
hard.calls[hard.calls==0]<-0

geno2cross(hard.calls)
```

```
## cross file written to cross.csv
```

3.5 Write into R/qtl format . . .

For the purposes of this tutorial, we want to pretend we don't know the physical location of the markers . . . like if we were working with a species that lacks a complete *de novo* reference genome. To do this, we will tell the geno2cross function to ignore the specification of chr_position in the marker names and instead provide random chromosome and position data.

```
geno2cross(hard.calls,crossfile = "~/Desktop/cross.csv",
           chr = sample(1:9, replace = T, size = ncol(hard.calls)),
           pos = runif(ncol(hard.calls), min = 0, max = 100))
```

```
## cross file written to ~/Desktop/cross.csv
```

3.6 Read the cross file into R/qtl . . .

```
cross<-read.cross("csv", file="~/Desktop/cross.csv",
                  genotypes=c(0,1,2), crosstype="f2")
```

```
##  --Read the following data:
##   264  individuals
##   1242  markers
##   1  phenotypes
##  --Cross type: f2
```

```
summary(cross)
```

```
##     F2 intercross
##
##     No. individuals:    264
##
##     No. phenotypes:     1
##     Percent phenotyped: 100
##
##     No. chromosomes:    9
##         Autosomes:      1 2 3 4 5 6 7 8 9
##
##     Total markers:      1242
##     No. markers:        144 143 146 140 120 137 145 143 124
##     Percent genotyped:  100
##     Genotypes (%):      AA:21.6  AB:51.3  BB:27.1  not BB:0.0  not AA:0.0
```

4 Part 4: Making the genetic map

4.1 Examine genotype frequencies by individual

Before we actually build the map, we want to make sure that the individuals in our population all look like recombinants. Our results will be biased if some individuals are identical to the parents . . . these individuals also won't contribute to QTL detection power, and should be dropped.

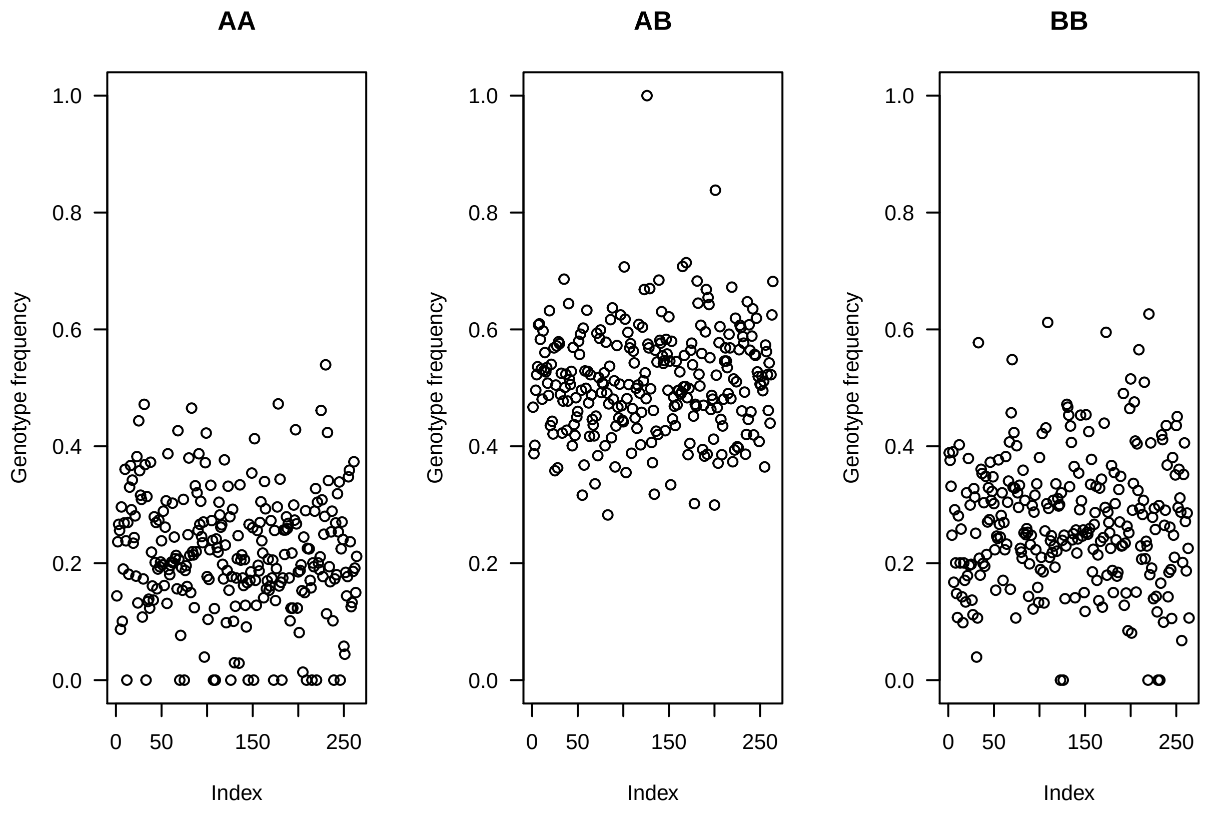


Figure 3: genotype frequecies for each individual. Note that many individuals have 0 frequencies of either the AA or BB. These should be excluded.

Drop non-recombinant or highly heterozygous individuals

```r
cross<-subset(cross,
              ind = gfreq[2,]<.8 & # <80% heterozygosity
                gfreq[1,]>0.02 & # >2% AA homozygote
                gfreq[3,]>0.02) # >2% BB homozygote
```

4.2 Drop markers that are too similar . . .

The main issue in making a genetic map (besides missing/erroneous data) is the need to calculate all pairwise recombination fractions among markers. Since this problem scales exponentially with the number of markers, and NGS marker sets usually number in the 1000's, we need to find a way to reduce our marker set to a managable number. One way to do this to split the markers up into blocks and drop very similar markers. This reduces the size of the problem, then lets us calculate the entire matrix recombination fraction.

```r
cross.sub<-dropSimilarMarkers(cross, blockSize = 500,
                              rf.threshold = 0.02, runFullMatrix = T,
                              byChr = F)
```

```r
summary(cross.sub)
```

```
##     F2 intercross
##
##     No. individuals:    242
##
##     No. phenotypes:     1
##     Percent phenotyped: 100
##
##     No. chromosomes:    9
##         Autosomes:      1 2 3 4 5 6 7 8 9
##
##     Total markers:      276
##     No. markers:        21 35 38 31 31 22 25 33 40
##     Percent genotyped:  100
##     Genotypes (%):      AA:22.8  AB:50.9  BB:26.3  not BB:0.0  not AA:0.0
```

```r
par(mfrow = c(1,1))
plot.rf(cross.sub)
```

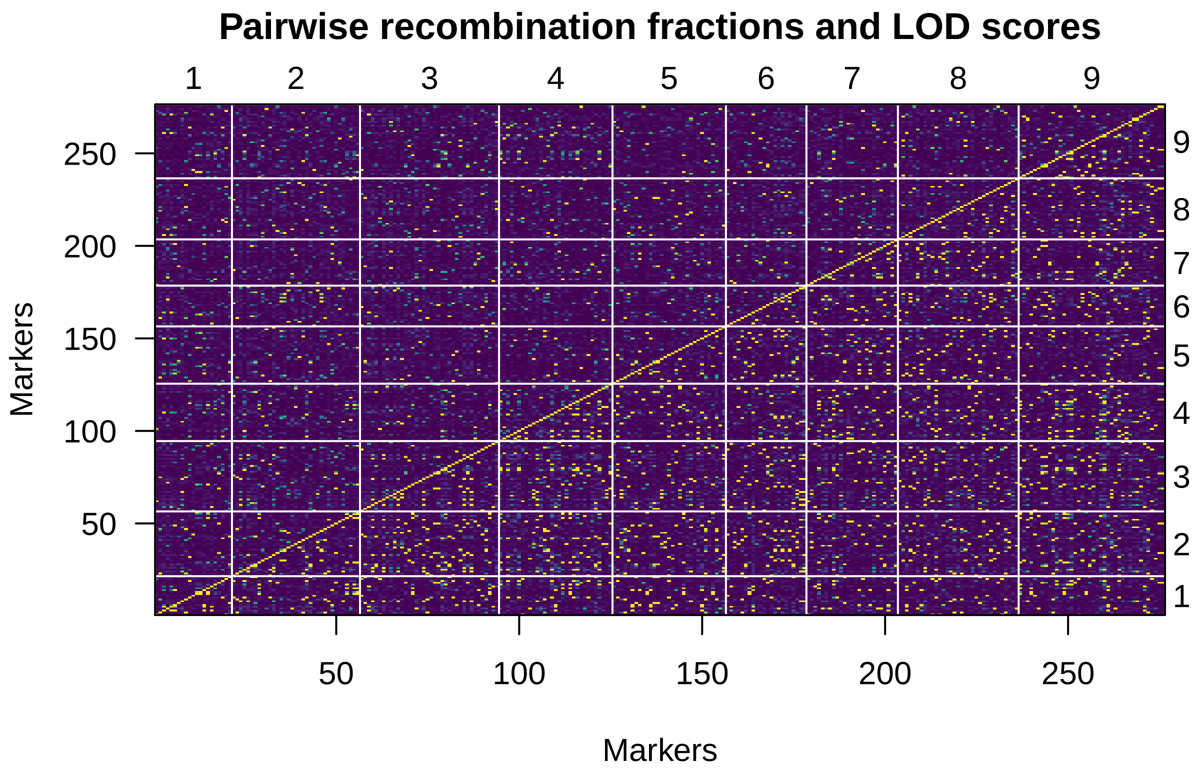


Figure 4: recombination fraction among markers. Warmer colors have lower recombination fractions (more similar), darkblue colors indicate marker pairs that are uncorrelated. Since we randomized the position of markers, it is not surprising that there is no association between position and recombination fraction

4.3 Form linkage groups . . .

The second step in making a genetic map is to bin markers into chromosomes. With all pairwise recombination fractions in hand, this is simple.

```
lgmar<-formLinkageGroups(cross.sub, reorgMarkers=F, max.rf = .23,verbose = F)
head(lgmar)
```

```
##             origchr LG
## 8_39145537        1  9
## 3_4210063         1  3
## 6_5751071         1  6
## 4_7989001         1  8
## 6_4819843         1  6
## 6_35850614        1  6
```

4.4 Rename linkage groups . . .

We then need to re-name the marker groupings so that they match their original chromosomes.

```
lgmar$true.chr<-splitText(rownames(lgmar))
lgmar$true.pos<-as.numeric(splitText(rownames(lgmar), num = 2))
```

```
marlist<-split(rownames(lgmar), as.factor(lgmar$true.chr))
cross2<-newLG(cross.sub, marlist, keep.rf = T)
```

```
plot.rf(cross2)
```

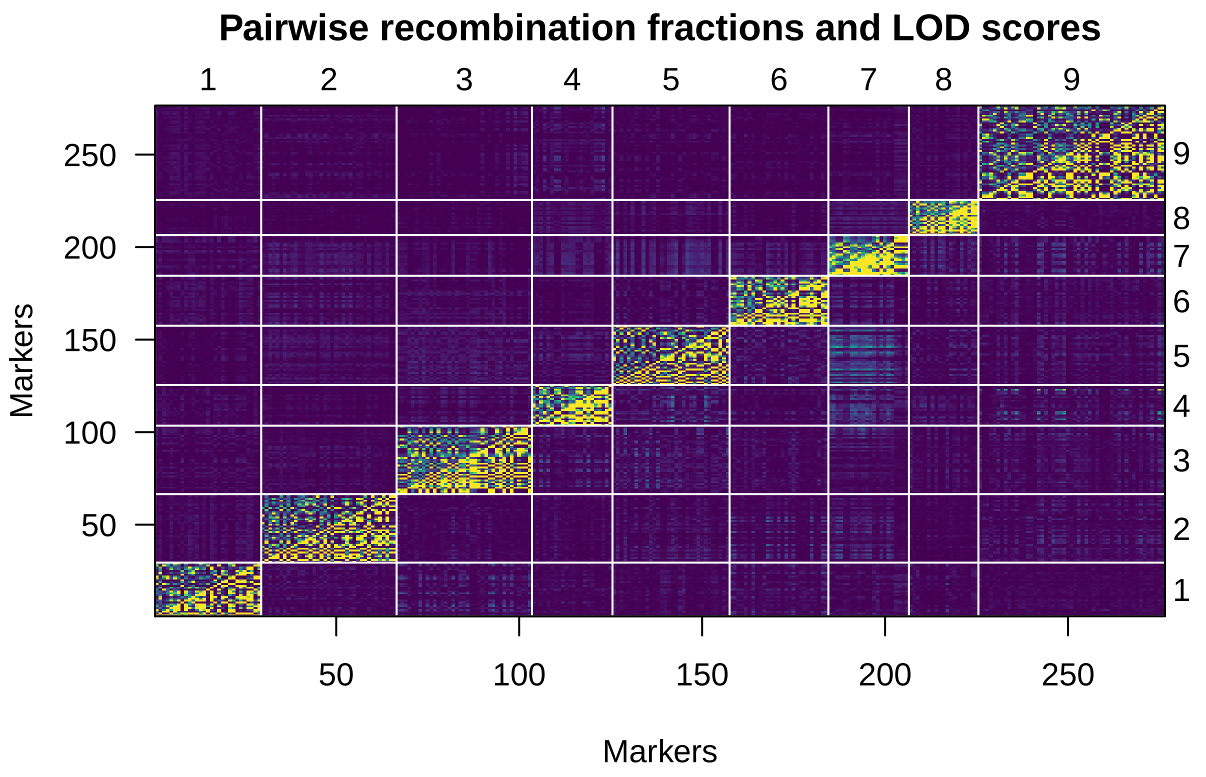


Figure 5: recombination fractions of markers, now the correlations are higher within chromosomes

4.5 Order markers within linkage groups . . .

Now we need to order markers within chromosomes. Historically, this is a difficult, error-prone and time intensive step. However, there are now a few approaches that make this super simple. To order markers, we will use a Traveling Salesperson Problem Solver. TSP solvers find the shortest distance to connect all points. We operate on the recombination fraction matrix to do so, using the program `concorde`. We call this program using the R package `TSP` and the qtlTools function `tspOrder`.

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

```
plot.rf(cross3)
```

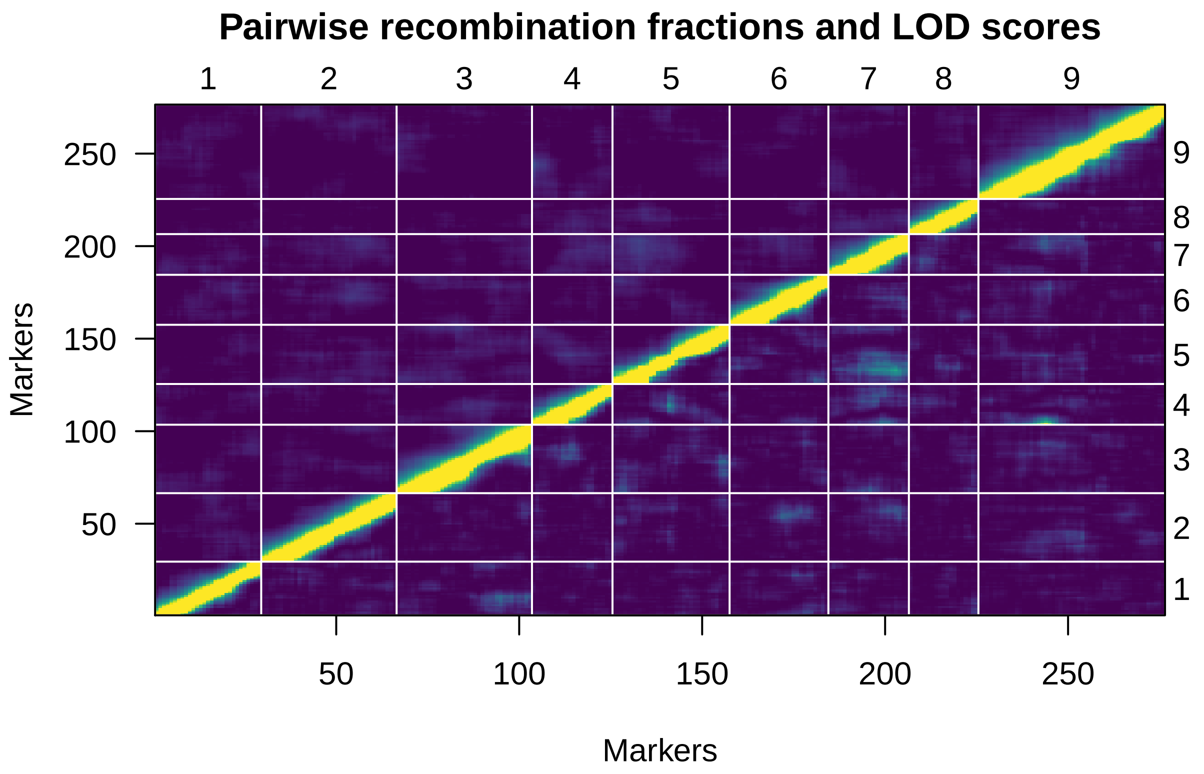


Figure 6: recombination fractions of markers, now proximate markers have the lowest recombination fraction (highest correlation)

4.6 Finalize the markers . . .

We can prune the cross, dropping markers that are still too close and re-order markers that seem to have better orders than the one found by TSP. In our case, there is nothing to be done tho.

```
cross4<-reDoCross(cross3, window = 3, min.distance = 2,
                  initialEstMap = T, verbose = T)
```

4.7 Orient chromosomes to match reference . . .

Finally, we may want to ensure that the chromosomes are oriented correctly. To do this, we go chromosome-by-chromosome and check whether the physical order matches the mapping order. We need to do this because the TSP solver performs agnostic to the input order of markers.

```
map<-pullMap(cross4)
map$phys.chr<-splitText(map$marker.name)
map$phys.pos<-as.numeric(splitText(map$marker.name, num = 2))/1e6
kable(head(map))
```

marker.name	chr	pos	phys.chr	phys.pos
1_56683837	1	0.000000	1	56.68384
1_54661016	1	5.583246	1	54.66102
1_53703843	1	8.748052	1	53.70384
1_53097662	1	11.054488	1	53.09766
1_52347245	1	15.085338	1	52.34725
1_52092080	1	17.602653	1	52.09208

```
cross5<-matchMarkerOrder(cross4)
```

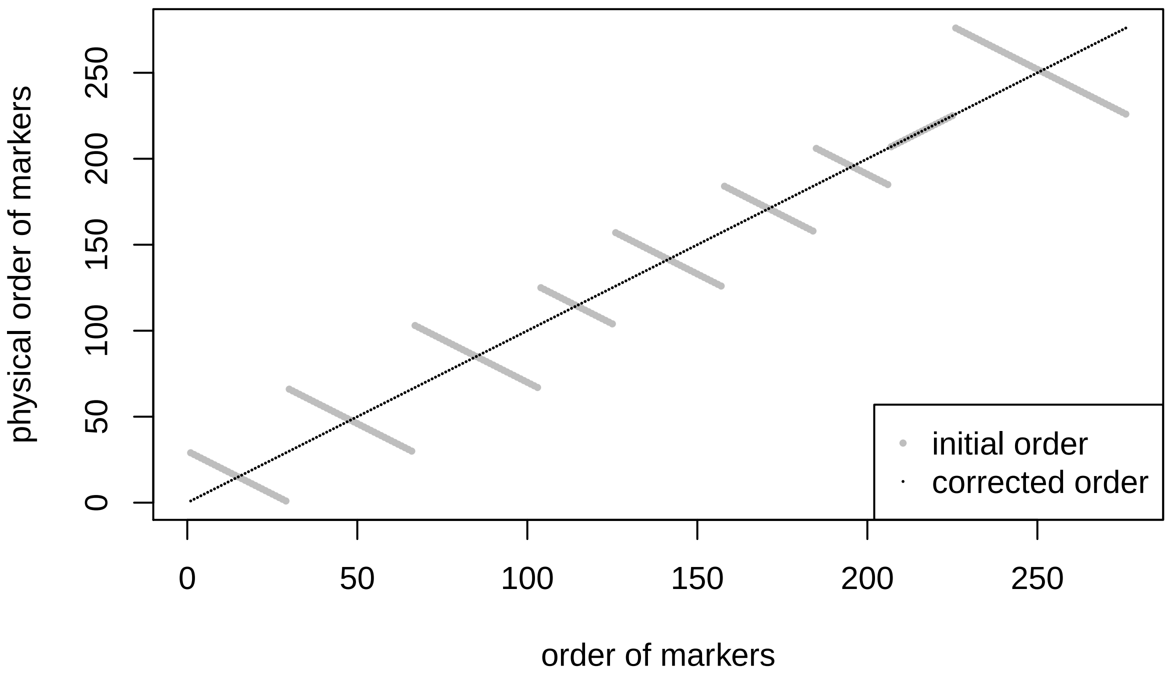


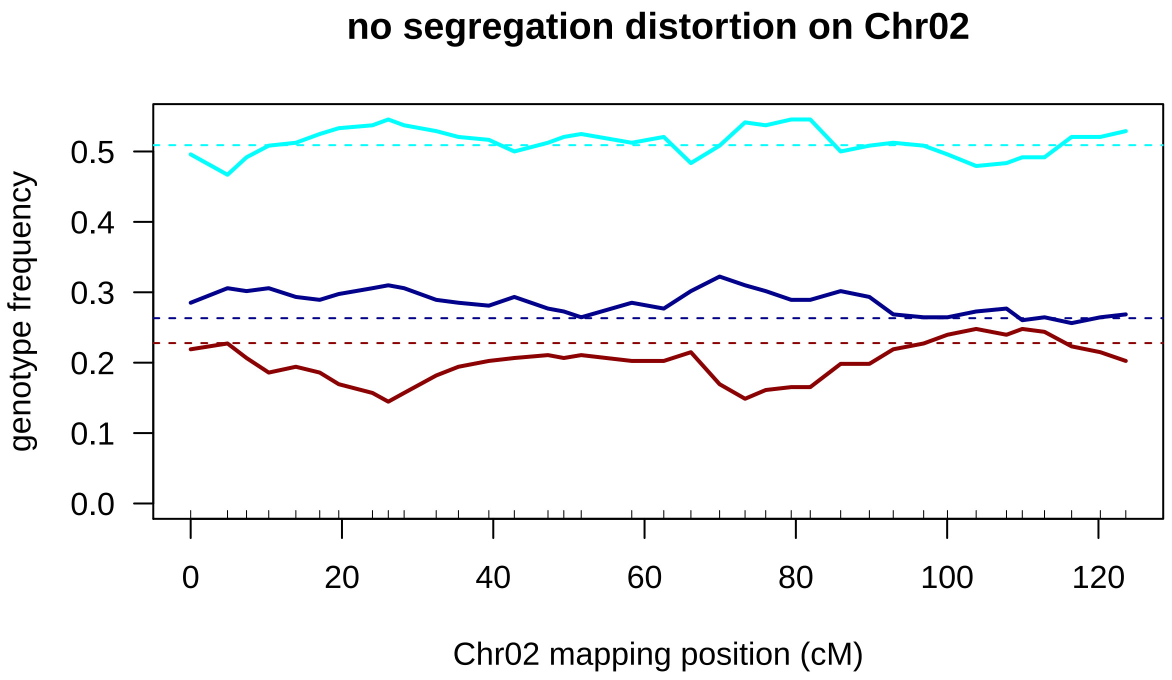
Figure 7: order of markers before (grey) and after (black) matching to physical orientiation of chromosomes.

5 Part 5: Some other considerations.

5.1 Segregation distortion...

Segregation distortion (the deviation of genotype frequecies from expected ratios) can be indicative of mapping bias, bad markers or erroneous genotype calls. Such errors make building a genetic map difficult ... However, often segregation distortion is real.

Take for example the P. hallii F2 population.



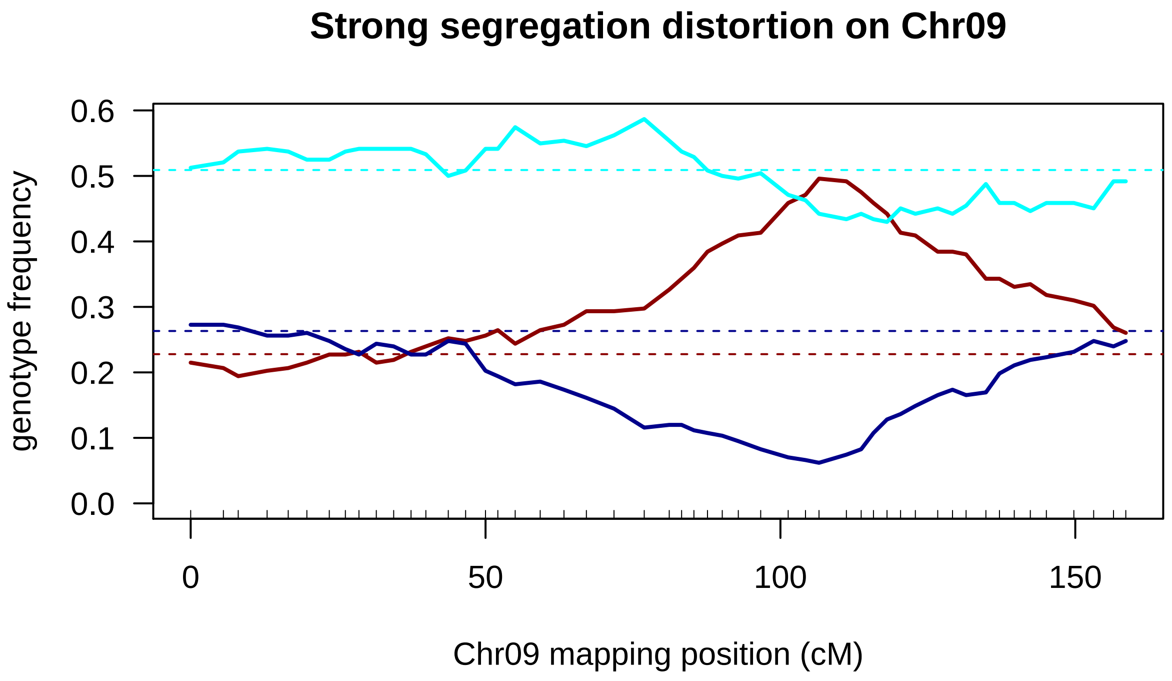


Figure 8: The frequency of the three genotypes, A/A (Red), B/B (Blue), A/B (Cyan), are plotted. Note some global bias towards the B/B genotype. This is not a problem, at least at this level of bias. On the right arm of Chr09 there is a near lack of B/B genotypes. Originally we dropped these distorted markers, which forced Chr09 to be split into two chromosomes. However, we now know that this distortion is real, caused by some genetic factor that when in the B/B state is selected against.

5.2 Erroneous data

Often we cannot control our error rate. For example, low coverage short read data may produce bad genotype calls ~0.1% of the time. Since we are dealing with a 276x242 genotype matrix, this error rate would produce ~70 genotyping errors.

We can easily deal with genotyping errors during QTL Mapping (see the tutorial); however, if genotyping errors are common they may affect our genetic map construction. It may be appropriate to make the genetic map, then subsequently drop suspicious marker-individual combinations.

We can do this as follows (this is slow, so just doing Chr08 for the example): First, calculate the LOD score for genotyping errors and pull out those errors above some threshold (2 is very conservative, I usually use 3 or higher).

```
el<-calc.errorlod(subset(cross5, chr = 8), error.prob=0.001,
                  map.function="kosambi")
tel<- top.errorlod(el, cutoff = 2)
```

Table 3: The markers with an error LOD score >2 on Chr08

chr	id	marker	errorlod
8	XZWG	8_6400141	5.459352
8	WZOT	8_651199	2.601151
8	XUCT	8_651199	2.601151
8	XBTT	8_1356304	2.356387
8	SSPU	8_1959371	2.300520

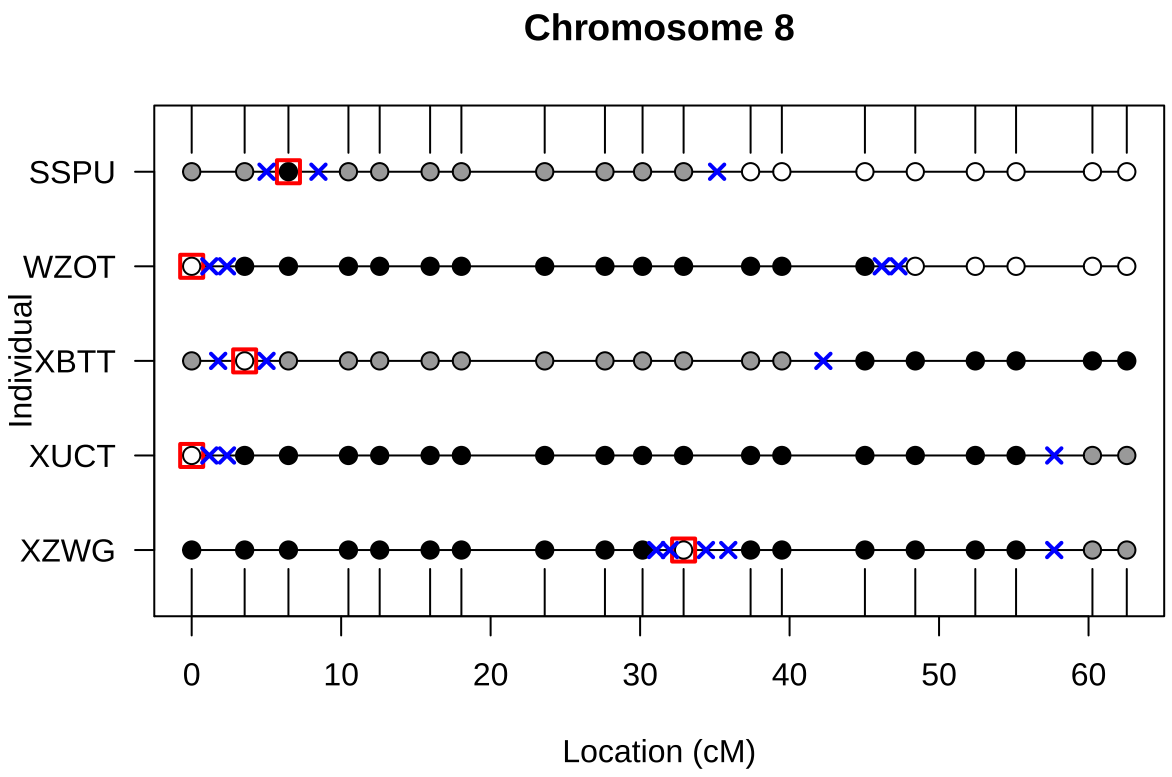


Figure 9: position of crossover events (x) and the distribtution of each genotype on Chr08

We then go through this data.frame and drop all marker-by-individual combinations that appear erroneous.

```r
cross.clean<-cross5
for(i in 1:nrow(tel)) {
  chr <- tel$chr[i]
  id <- tel$id[i]
  mar <- tel$marker[i]
  cross.clean$geno[[chr]]$data[cross5$pheno$id==id, mar] <- NA
}
cross6<-cross.clean
map1<-est.map(cross5, chr = 8, error.prob = 0.001, map.function = "kosambi")
map2<-est.map(cross6, chr = 8, error.prob = 0.001, map.function = "kosambi")
chrlen(map1) # original chr08 length
```

```
##        8
## 62.10223
```

```r
chrlen(map2) # new chr08 length
```

```
##        8
## 60.83308
```