De novo linkage maps using R and R/qtl

JT Lovell

2017-02-01

Introduction

The following is a tutorial to take a large genotype matrix and produce a linkage map. The important elements we will cover include:

1) Formatting a genotype matrix for R/qtl
2) Culling the matrix to evenly spaced markers
3) Making linkage groups
4) Ordering markers within linkage groups
5) Ordering the linkage groups to match known chromosome names and orders
6) Fine tuning and documentation

Step 2 is particularly important for genotype matrices generated from whole genome sequencing, RAD or other NGS technology. These techniques produce markers with a random distribution across the genome and have a relatively high error rate. If recombination is absent or very rare, genotyping errors will look like recombination events and cause marker miss-ordering. Also, the speed of map construction scales exponentially with the number of markers (since we operate on an nMarker * nMarker matrix). Retaining only the best and most informative markers is therefore pretty important.

Setup

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.

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)
```

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
```

Load the genotype data

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

Part 1: From a genotype matrix to a cross object

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 an F2 (intercross) mapping population with a genotype matrix called `genmat` that looks like this:

```
##   1_0 1_0.487 1_0.536 1_0.835 1_1.251 1_7.178 1_7.242 1_7.457
## 1 "h" "b"     "h"     "b"     "a"     "h"     NA      "h"
## 2 "a" "a"     "a"     "a"     "a"     "h"     "a"     "h"
## 3 "b" NA      "a"     NA      NA      NA      "a"     NA
## 4 "a" "a"     "b"     "a"     "a"     NA      NA      NA
## 5 "a" NA      "b"     NA      NA      "a"     "a"     "a"
## 6 "a" "h"     "b"     "h"     "h"     "h"     NA      "h"
## 7 "h" "a"     "h"     "a"     "a"     "a"     "a"     "h"
## 8 "a" NA      "a"     NA      NA      "h"     NA      "h"
```

Marker Names

I recommend storing individual identifiers in the `rownames` of the matrix and marker IDs in the `colnames`. If physical positions are known about the markers, it is really useful to name the markers as chr_position (e.g. marker @ Chr1 and 150kb might be '1_150'). Here, I extract the chr and position information from the marker names and the F2 line IDs from the rownames. I use qtlTools `splitText` function 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(genmat), sep = "_", num = 1)
pos<-splitText(colnames(genmat), sep = "_", num = 2)
ids<-rownames(genmat)
```

Build an R/qtl cross file

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

```
geno2cross(genomat = genmat,
           chr=chr, pos=pos, id=ids,
           crossfile = "~/Downloads/cross.csv")
```

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

Read the genotype matrix into R/qtl

```
cross<-read.cross("csv", file = "~/Downloads/cross.csv", crosstype = "f2",
                  genotypes = c("a","h","b"))
```

```
##  --Read the following data:
##   200  individuals
##   455  markers
##   1  phenotypes
##  --Cross type: f2
```

Part 2: Cull and process the genotype matrix in R/qtl

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)*chrlen(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 envents 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)`

A note on marker density and error probability: *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. See appendix 2 for an explanation and simulation*

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

```
## [1] 200
```

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

```
## [1] 0.005
```

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

```
## [1] 0.5
```

Drop markers by chromosome

To drop very close markers, we use the function "dropSimilarMarkers". We set the minimum recombination fraction at the threshold determined above. This requires a pairwise recombination fraction matrix, which is very memory intensive for large genotype matrices. Therefore we first run chromosome-by-chromosome. If we do not know the physical chromosomes of each marker, this step will not help much.

```
goodMars<-lapply(chrnames(cross), function(x){
  cr<-subset(cross, chr = x)
  cr<-est.rf(cr)
  cr<-dropSimilarMarkers(cr, rf.threshold = 0.01, verbose = F)
  return(markernames(cr))
})
toKeep<-unlist(goodMars)
toDrop<-markernames(cross)[!markernames(cross) %in% toKeep]
```

This produces a vector of markers that were too close to other markers and had more missing data and/or worse segregation distortion than its closest marker.

```
print(length(toDrop))
```

```
## [1] 48
```

```
cross<-drop.markers(cross, markers = toDrop)
```

Drop markers across the whole matrix

This will only help if some markers were given physical positions that are not on the correct linkage group.

```
cross<-est.rf(cross)
cross<-dropSimilarMarkers(cross, rf.threshold = 0.01)
```

```
## initial n markers:  407
## final n markers:  397
```

Part 3: Assign markers to linkage groups

Overview

Even if we are pretty sure that all markers are assigned to their correct linkage group (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 observerd fraction of individuals that recombine and will expand the map, making inference of QTL much more difficult and imprecise.

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 require adjustment of both parameters.

```
lgmar<-formLinkageGroups(cross, reorgMarkers=F)
```

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 annonymous, once can skip this step. Remember the name of the original chromosome is stored in the name of the marker.

```
origchr<-as.numeric(splitText(markernames(cross), sep = "_", num = 1))
lgmar$origchr<-origchr
tab<-table(lgmar)
```

We can visualize the overlap by looking at the tabulation of the new linkage groups and the original chromosomes. Notice near, but not complete overlap

```
print(tab)
```

```
##        LG
## origchr  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
##       1  0  0 27  1  0  0  1  1  0  0  0  0  0  0  0  0  0  0  1
##       2  0  0  2  0 20  0  0  0  2  0  0  1  0  1  1  0  0  0  0
##       3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0  0  0
##       4  0  0  0  0  0  0  0  0  0 16  0  0  0  1  0  0  1  0  0
##       5  0 41  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
##       6  0  0  0  0  0 21  0  0  1  0  0  0  0  0  0  0  0  0  0
##       7  0  0  1  0  0  0  0  0  0  0  0 16  0  0  0  0  0  0  0
##       8  0  0  0  0  0  0 18  0  0  0  0  0  0  0  0  0  0  0  0
##       9  0  0  0 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
##      10  0  1  1  0  0  0  0  1  0  0  0  0 17  0  0  0  0  0  0
##      11  0  0  1  0  1  0  0 14  0  0  0  0  0  0  0  1  0  0  0
##      12  0  0  0  0  0  0  0  0 14  0  0  0  0  0  0  0  0  0  1
##      13 48  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##      14  0  0  0  0  0  0  0  1  0  1  1  0  0  0  0  0 12  0  0
##      15  0  0  0  0  0  0  0  1  0  0  0  0  0  0 12  0  0  0  0
##      16  1  0  0  0  0  0  0  0  0  0 17  0  0  0  0  0  0  0  0
##      17  0  0  0  0  1  0  0  0  0  0  0  0  0 13  0  0  0  0  0
##      18  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8
##      19  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  1  0 13  0
```

Rename linkage group names based on maximum overlap

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

Reassign markers to each linkage group

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.

```
marlist<-lapply(1:nchr(cross), function(x) rownames(lgmar)[lgmar$LG==x])
names(marlist)<-newnames
marlist<-marlist[order(as.numeric(names(marlist)))]
```

For Chrs 1 and 2, it looks like this:

```
marlist[1:2]
```

```
## $`1`
##  [1] "1_0.487"   "1_0.835"   "1_1.251"   "1_7.178"   "1_7.242"
##  [6] "1_7.457"   "1_7.619"   "1_10.782"  "1_10.937"  "1_10.95"
## [11] "1_12.319"  "1_14.638"  "1_18.681"  "1_38.705"  "1_45.215"
## [16] "1_52.083"  "1_54.798"  "1_70.382"  "1_72.474"  "1_72.709"
## [21] "1_76.781"  "1_119.808" "1_121.103" "1_136.182" "1_146.287"
## [26] "1_149.583" "1_151.725" "2_0.478"   "2_41.876"  "7_25.568"
## [31] "10_9.879"  "11_76.976"
##
## $`2`
##  [1] "2_0.089"   "2_0.314"   "2_1.09"    "2_11.905"  "2_19.944"
##  [6] "2_21.038"  "2_22.782"  "2_25.503"  "2_26.207"  "2_30.019"
## [11] "2_39.614"  "2_45.862"  "2_47.665"  "2_48.941"  "2_65.158"
## [16] "2_65.468"  "2_68.621"  "2_71.279"  "2_71.544"  "2_83.496"
## [21] "11_80.453" "13_44.188" "17_0.861"
```

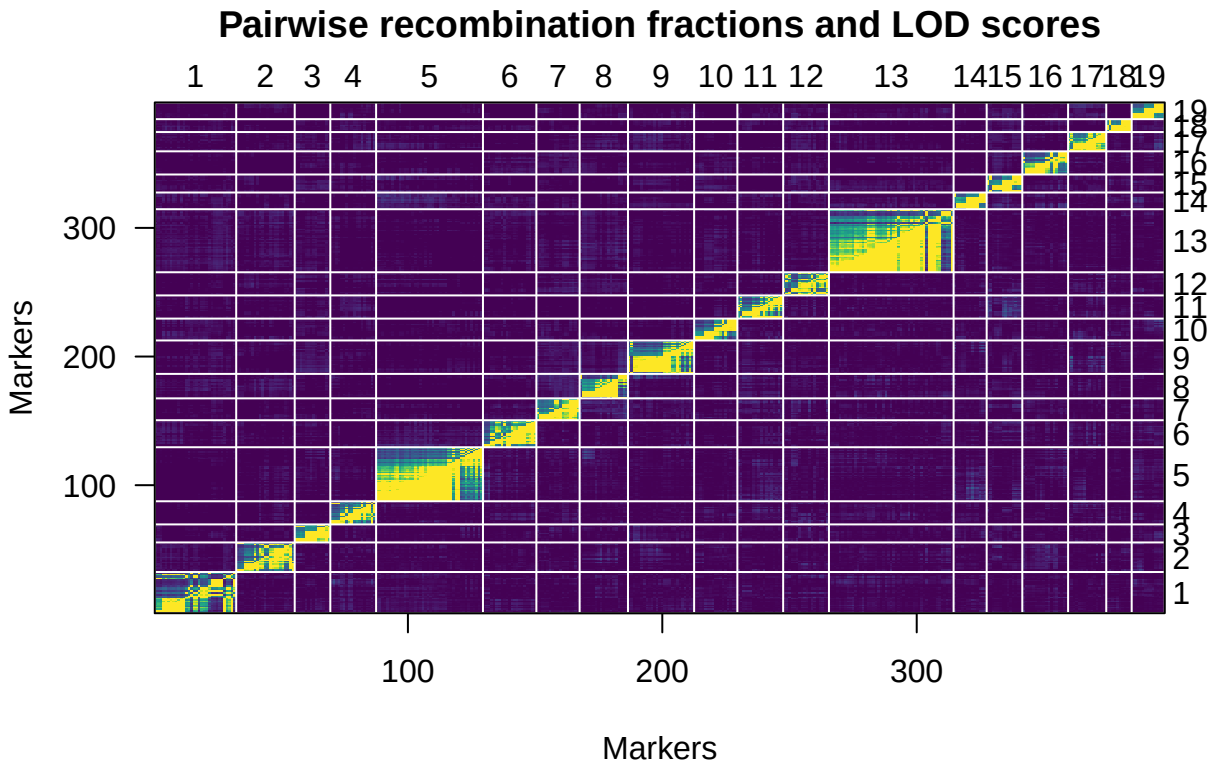
Reorganize the cross with new linkage group names and marker groupings

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

```
cross.newlg<-newLG(cross = cross, markerList = marlist)
cross.newlg<-est.rf(cross.newlg)
```

Note that markers now have high correlations within linkage groups (yellow), and low correlation among groups (blue), but the order within chromosomes is not necessarily correct.

```
plot.rf(cross.newlg)
```



Part 4: Reorder markers

Overview

There are many methods to order markers. While joinmap4 remains the industry standard, it has a bunch of drawbacks - it's slow, costly and is point-and-click. 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.

Marker ordering using TSP solvers

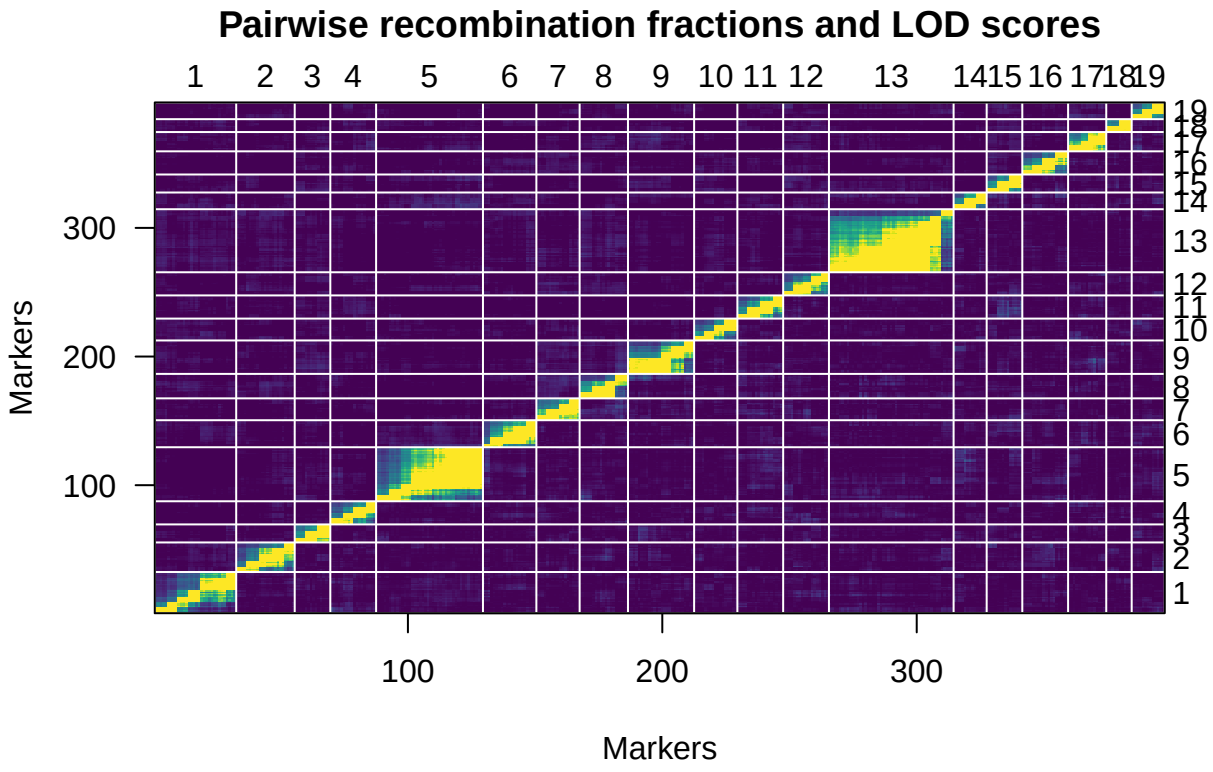
Here, I build upon Grey Monroe's `TSPmap` protocol and employ a travelling salesperson problem solver to find the shortest path through the recombination fraction matrix. The optimal method is `concorde`, however, this requires a separate installation of the concorde program. See `?tspOrder` for details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

```
cross.ordered<-tspOrder(cross = cross.newlg,
                  hamiltonian = TRUE,
                  method="concorde",
                  concorde_path = "/Users/John/Documents/concorde/TSP",
                  return = "cross")
```

Look at marker order

Note that following the marker reordering, there is a decay in correlations between markers from the diagonal.

```
cross.ordered<-est.rf(cross.ordered)
plot.rf(cross.ordered) # overall
```



Refine marker order

TSP is not perfect and sometimes chooses orders that result in a few more crossovers than necessary. Check for these errors and fix them using `ripple`.

```
cross.rip<-repRipple(cross.ordered, window = 4, re.est.map = F, verbose = F)
```

Estimate the genetic map

Now that we have an ordered genetic map, we need to estimate distances between markers.

```
map<-est.map(cross.rip, map.function = "kosambi")
cross.rip<-replace.map(cross.rip, map)
```

Drop markers that are too close to eachother

Like we did before with recombination fractions, we need to drop close markers. Here we use a sliding window and choose the best marker within a minimum distance window. We then re-estimate the genetic map and replace the old map with the new one.

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

```
## [1] 0.5
```

```
cross.sub<-repPickMarkerSubset(cross = cross.rip, verbose=T, min.distance = min.cm)
```

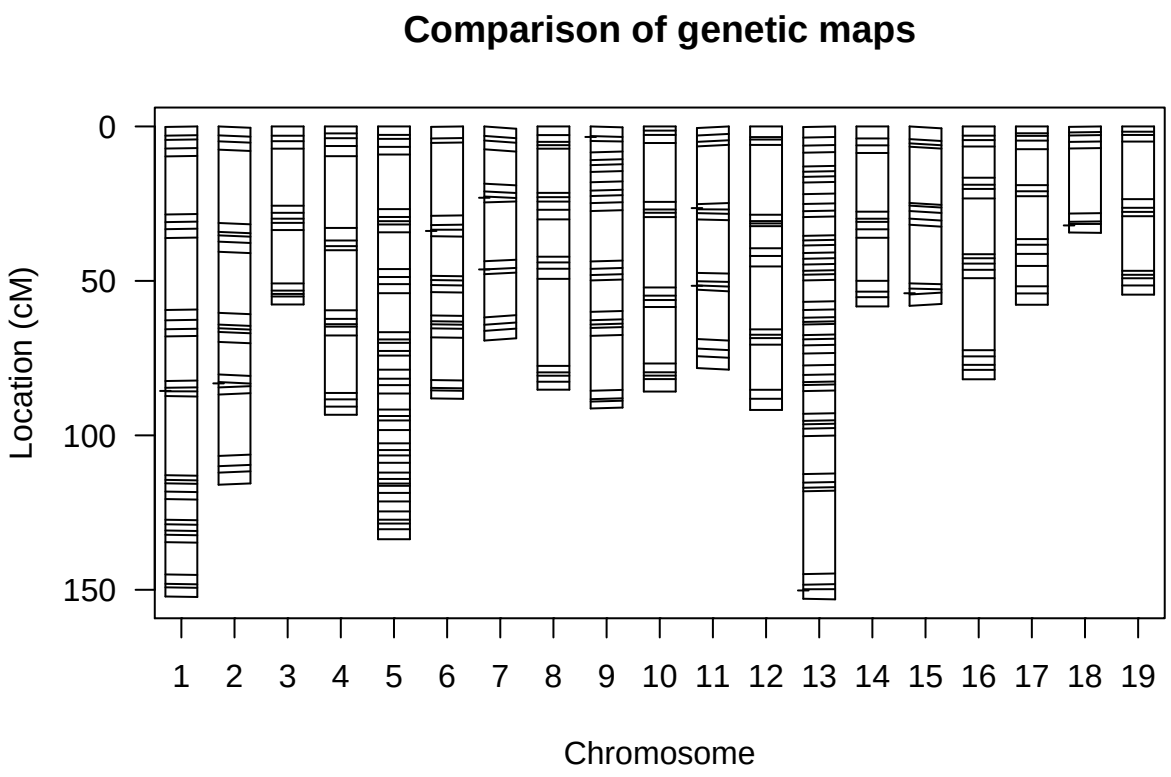
```
## dropped  11  markers
```

```
map<-est.map(cross.sub, map.function = "kosambi")
cross.sub<-replace.map(cross.sub, map)
```

Compare maps

We can see how we did by comparing genetic maps. For example, befoe and after dropping a few close markers.

```
plot.map(cross.rip, cross.sub)
```



Part 5: Compile final map

Overview

To produce a publishable map, we need to orient the markers according to previously published work.

Look at how marker orders in the new map compare to published orders

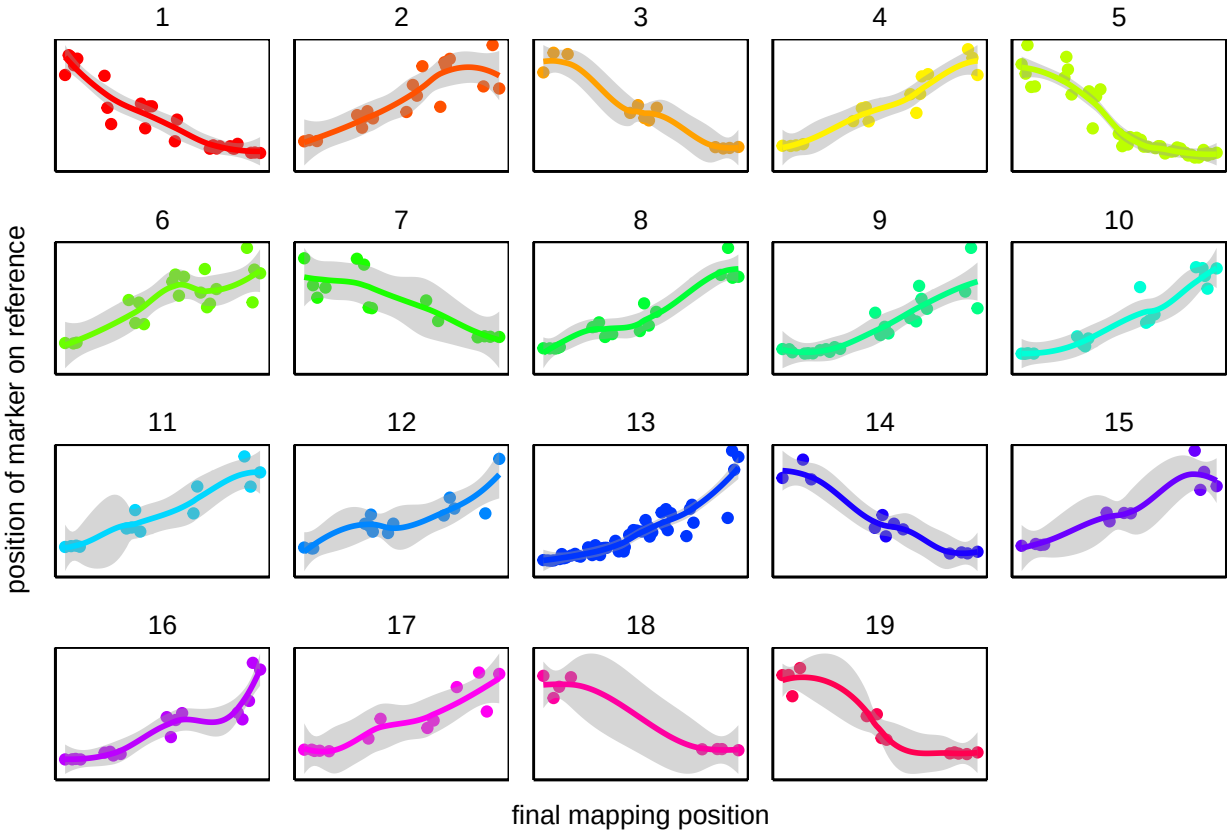
Make sure the orientation of each chromosome matches that of previously known positions

```
map<-pullMap(cross.sub)
map$chr.orig<-as.numeric(splitText(markernames(cross.sub), sep = "_", num = 1))
map$pos.orig<-as.numeric(splitText(markernames(cross.sub), sep = "_", num = 2))

map<-map[map$chr == map$chr.orig,]

ggplot(map, aes(x = pos, y = pos.orig, col = chr))+
  geom_point()+stat_smooth(method = "loess")+
  scale_color_manual(values = rainbow(19), guide = F)+
  theme_jtl()+
  theme(axis.text=element_blank(),
        axis.ticks=element_blank())+
  facet_wrap(~chr, scale = "free")+
```

```
  labs(x = "final mapping position", y = "position of marker on reference",
       main = "order of markers")
```



Find marker orders in chromosomes that are opposite to published orders

To do this, we extract the slope of a linear model for each chromosome. If negative, we note it.

```
flipIt<-sapply(unique(map$chr), function(i){
  out<-lm(pos ~ pos.orig, data = map[map$chr==i,])$coefficients["pos.orig"]
  names(out)<-i
  ifelse(out>0,FALSE,TRUE)
})

print(toflip<-names(flipIt)[flipIt])
```

```
## [1] "1"  "3"  "5"  "7"  "14" "18" "19"
```

Flip the order of markers that have reversed order relative to the reference

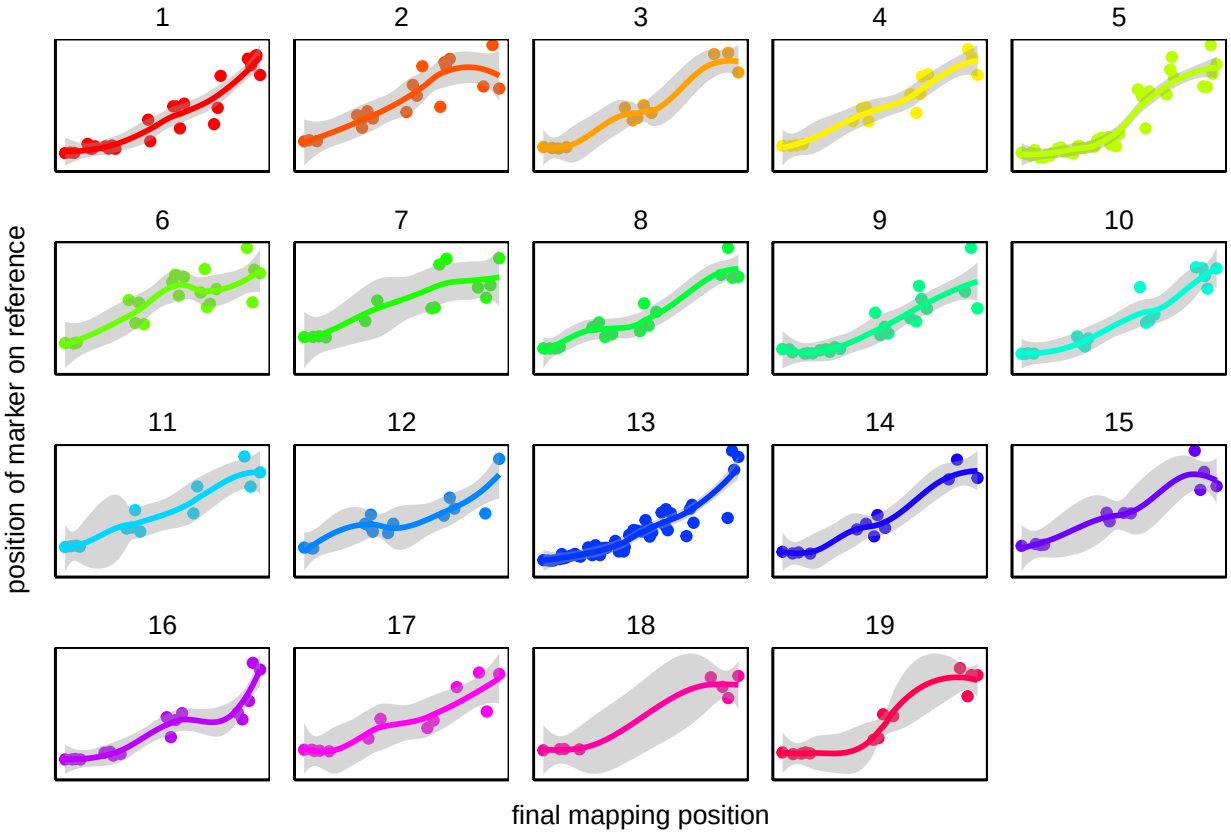
```
cross.final<-cross.sub
for(i in toflip) cross.final<-flip.order(cross.final, chr = i)
```

Make sure this worked

```
map<-pullMap(cross.final)
map$chr.orig<-as.numeric(splitText(markernames(cross.final), sep = "_", num = 1))
map$pos.orig<-as.numeric(splitText(markernames(cross.final), sep = "_", num = 2))
```

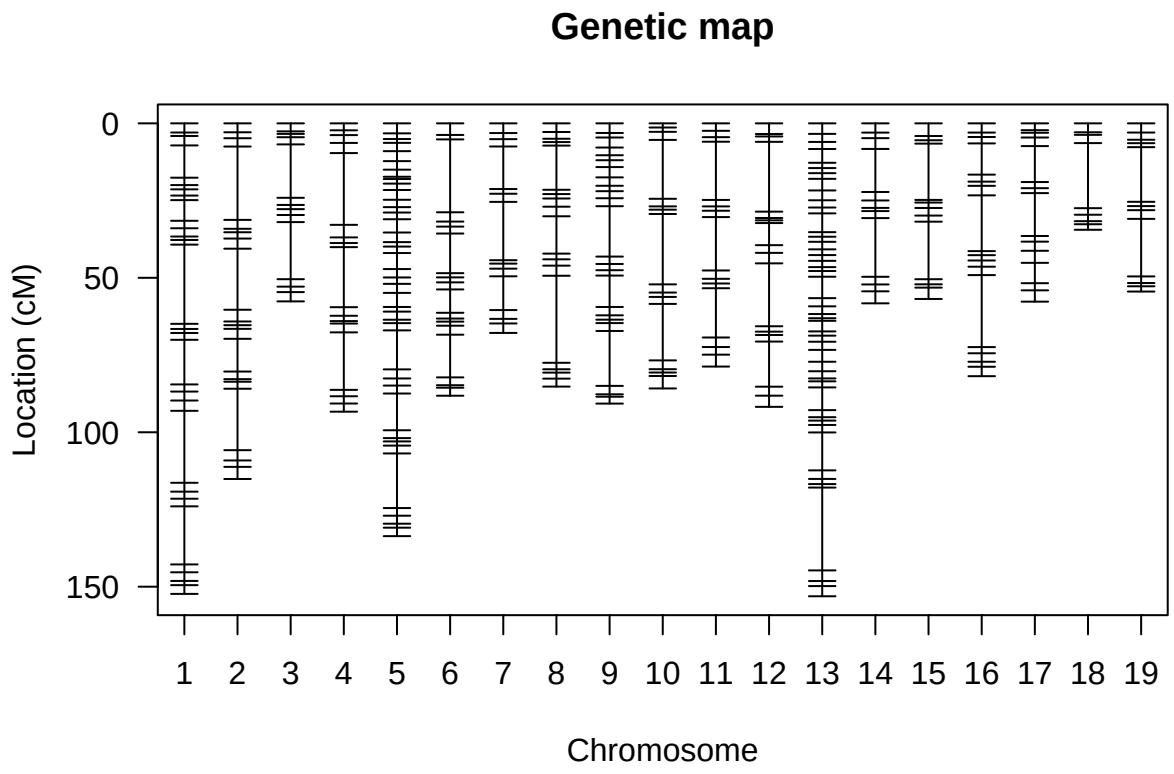
```
map<-map[map$chr == map$chr.orig,]

ggplot(map, aes(x = pos, y = pos.orig, col = chr))+
  geom_point()+stat_smooth(method = "loess")+
  scale_color_manual(values = rainbow(19), guide = F)+
  theme_jtl()+
  theme(axis.text=element_blank(),
        axis.ticks=element_blank())+
  facet_wrap(~chr, scale = "free")+
  labs(x = "final mapping position", y = "position of marker on reference",
       main = "order of markers after flipping")
```

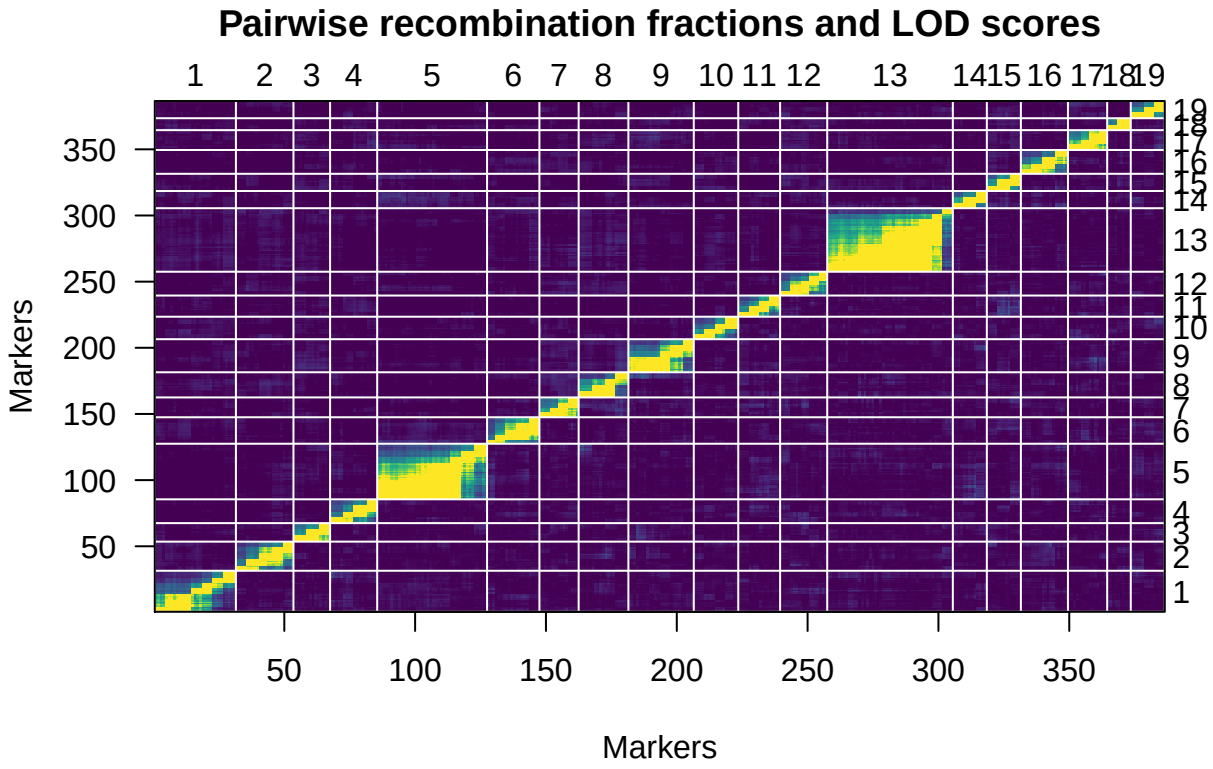


Make some final diagnostic plots

```
plot.map(cross.final)
```




```
cross.final<-est.rf(cross.final)
plot.rf(cross.final)
```



Write the cross to file

```
write.cross(cross.final, format = "csv", filestem = "~/Downloads/crossfinal")
```

Appendix 1: Make simulated data

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

```
# Grab genotype data from qtl::fake.f2 and recode a/h/b, drop X chromosome
data(fake.f2)
genmat<-pull.geno(fake.f2)
genmat<-apply(genmat, 2, as.character)
genmat<-apply(genmat, 2, function(x)
  ifelse(is.na(x),NA,
         ifelse(x == "1", "a",
                ifelse(x == "2", "h","b"))))
genmat<-genmat[,-grep("X", colnames(genmat))]

# Add in a bunch of markers that have just a few differences from the original data
newmars1<-genmat; substr(colnames(newmars1),1,1)<-"a"
newmars2<-genmat; substr(colnames(newmars2),1,1)<-"b"
newmars3<-genmat; substr(colnames(newmars3),1,1)<-"c"
newmar<-cbind(newmars1, newmars2, newmars3)
for(i in colnames(newmar)) {
  s<-sample(1:nrow(newmar),10,replace = F)
  for(j in s){
    newmar[s,i]<-sample(c("a","h","b"), 1)
  }
}
genmat<-cbind(genmat, newmar)
# Randomize positions of markers
chr<-sample(1:19, ncol(genmat), replace = T)
pos<-runif(n = ncol(genmat), min = 0, max = 100)
```

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

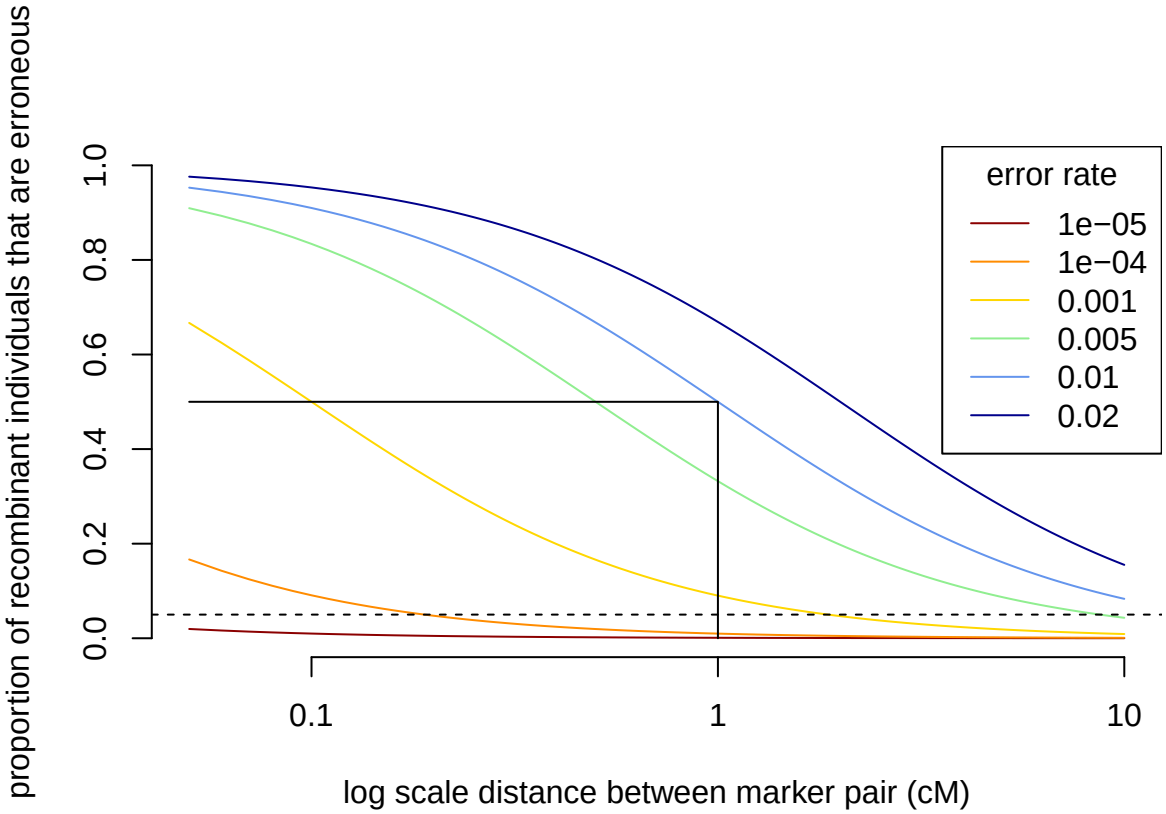
and the probability of false crossovers, which are caused by errors as:

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

Therefore, the proporition of true crossover individuals as:

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

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


proportion of recombinant individuals that are erroneous
0.0
0.2
0.4
0.6
0.8
1.0
0.1
1
10
log scale distance between marker pair (cM)
error rate
1e−05
1e−04
0.001
0.005
0.01
0.02