Fantasio

Isuru HAUPE & Marie MICHEL

2018-08-01

Contents

Introduction . . . 1
1. Getting started . . . 3
2. Running Fantasio . . . 6
3. Step by step usage of the package Fantasio . . . 7
4. Parallelism with the package . . . 12
5. Plotting . . . 12

Introduction

Fantasio is composed of several functions. Its goals are:

- For population genetic studies: estimating and detecting inbreeding on individuals without known genealogy, estimating the population proportion of mating types and the individual probability to be offspring of different mating types
- For rare disease studies: performing homozygosity mapping with heterogeneity
- For multifactorial disease studies: HBD-GWAS strategy

Fantasio implements the creation of several random sparse submaps on genome-wide data (to remove linkage disequilibrium). It also provides graphical outputs to facilitate interpretations of homozygosity mapping results and plots.

In this vignette, we illustrate how to use the package, using the data set HGDP-CEPH, a ship which contains 1043 individuals and 660918 markers. In order to access the data, you will need to download the HGDP.CEPH package (see below how) and load it.

Not all options of the functions are described here, but rather their basic usage. The reader is advised to look at the manual page of the function for details.

Principal concepts

Fantasio implements a maximum likelihood method that uses a hidden Markov chain to model the dependencies along the genome between the (observed) marker genotypes of an individual, and its (unobserved) homozygous by descent (HBD) status. The emission probabilities of this hidden Markov model (HMM) depend on the allele frequencies. The transition probabilities depend on the genetic distance between two adjacent markers. This model allows estimating the inbreeding coefficient f of an individual, and a parameter a, where af is the instantaneous rate of change per unit map length (here cM) from no HBD to HBD. Both HBD and non-HBD segment lengths are assumed to be distributed exponentially with mean lengths $\frac{1}{a(1-f)}$ and $\frac{1}{af}$, respectively.

THe method requires the markers to be in minimal linkage disequilibrium (LD). Otherwise biased estimations of f are produced. A strategy consisting of generating multiple random sparse genome maps (submaps) has been proposed to avoid this bias (Leutenegger et al. 2011). When several submaps are considered, f is estimated as the median value on all the f estimation obtained on the different maps after removing submaps with $a > 1$ (having an average HBD segment length of 1 cM is unlikely to be detected with a SNP density of 1 per 0.5 cM). This strategy has the advantage of not requiring any LD computation on the sample and of minimizing loss of information, as compared with a strategy that based on a single map of markers in minimal LD.

Fantasio statistical framework allows fixing HMM parameters to compute the likelihood of a mating type. These likelihoods can be used for:

- Inferring an individual as inbred by comparing the maximized likelihood with the one to be outbred with a likelihood ratio test
- Estimating the population proportion of mating types
- Estimating the individual probability to be into different mating types

When multiple submaps are used, the median p-values/probabilities are considered. See Leutenegger et al. 2011 for more details on the calculations.

Homozygosity mapping (Lander and Botstein 1987) consists in focusing on inbred affected individuals and searching for a region of the genome of shared homozygosity. Original homozygosity mapping requires that the genealogy of patients be known so that inbred patients can be identified and their respective f estimated. Leutenegger et al. (Leutenegger et al. 2006) proposed using the f estimated on genome-wide genetic data to compute a FLOD score, similar to Mortons LOD score (Morton 1955).

Genin et al. (Genin et al. 2012) adapted the FLOD formula for multiple submaps. FLOD(i)(m,s) is computed for each individual i, each marker m on each submap s, using the equation:

$$FLOD^{(i)}(m,s) = log_{10}\frac{P\left(Y_{m,s}^{(i)}|H_1\right)}{P\left(Y_{m,s}^{(i)}|H_0\right)} = log_{10}\frac{P\left(X_{m,s}^{(i)}=1|Y_{m,s}^{(i)}\right)+q.P\left(X_{m,s}^{(i)}=0|Y_{m,s}^{(i)}\right)}{\hat{f}_s^{(i)}+q.\left(1-\hat{f}_s^{(i)}\right)}$$

With the following parameters :

- $Y_{m,s}^{(i)}$ the observed genotype of individual i at marker m on submap s
- H_1 the hypothesis where marker m is linked to the disease, and H_0 the one where it is not
- $X_{m,s}^{(i)}$ the HBD status of individual i at marker m on submap s that is estimated together with the inbreeding coefficient using the HMM of the package
- $\hat{f}_s^{(i)}$ the estimated inbreeding coefficient of individual i on submap s
- q the assumed frequency of the mutation involved in the disease for this individual.

Results are then averaged over the different submaps to obtain a single $FLOD^{(i)}(m)$ at each marker m.

Genin et al. (Genin et al. 2012) proposed to detect fully penetrant rare recessive variants by performing homozygosity mapping on inbred cases from Genome-Wide Association Study (GWAS) data. Linkage evidence is then evaluated over the entire set I of inbred cases by computing a FLOD score, HFLOD(m,α), at each marker m, with a heterogeneity parameter α, that takes into account the possibility that only a fraction of the inbred affected individuals carry diseases causing mutations:

$$HFLOD(m,\alpha) = \sum log_{10}\left[\alpha.\frac{P\left(Y_{m,s}^{(i)}|H_1\right)}{P\left(Y_{m,s}^{(i)}|H_0\right)}+(1-\alpha)\right] = \sum log_{10}\left[\alpha.exp\left(FLOD^{(i)}(m)*log(10)\right)+(1-\alpha)\right]$$

This heterogeneity score is then maximized over α to evaluate the evidence of linkage at marker m where α is the estimate of the proportion of cases linked to this locus:

$$HFLOD(m) = max_{\alpha}(HFLOD(m,\alpha))$$

1. Getting started

The first thing you should know is that the package Fantasio depends on package gaston, please make sure to have it installed.

Please refer to the vignette of this package for more information.

Since we explained the concept behind the package let's make a usage example of it. For this we will use the data provided in the package HGDP-CEPH.

1.1 Installation

First and faremost install the package with the following command :

```r
install.packages("Fantasio")
```

After doing that we will need to run the following commands :

```r
require(Fantasio)
```

```
## Loading required package: Fantasio

## Loading required package: methods

## Loading required package: parallel

## Loading required package: gaston

## Loading required package: Rcpp

## Loading required package: RcppParallel

##
## Attaching package: 'RcppParallel'

## The following object is masked from 'package:Rcpp':
##
##     LdFlags

## Gaston set number of threads to 8. Use setThreadOptions() to modify this.

##
## Attaching package: 'gaston'

## The following object is masked from 'package:stats':
##
##     sigma

## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

1.2 Input HGDP-CEPH data file

```r
install.packages("HGDP.CEPH", repos="https://genostats.github.io/R/")
```

```r
require(HGDP.CEPH)
```

```
## Loading required package: HGDP.CEPH
```

From now on, we can use the package.

```r
filepath <-system.file("extdata", "hgdp_ceph.bed", package="HGDP.CEPH")
```

1.3 Creation of the bed matrix

Let us first create a bed.matrix object (see gaston package for details) for the data file we loaded from the package HGDP.CEPH with this command :

```r
x <- read.bed.matrix(filepath)
```

```
## Reading /ext/home/haupe/R/x86_64-pc-linux-gnu-library/3.4/HGDP.CEPH/extdata/hgdp_ceph.rds
## Reading /ext/home/haupe/R/x86_64-pc-linux-gnu-library/3.4/HGDP.CEPH/extdata/hgdp_ceph.bed
```

This command returns an updated 'bed.matrix' object (refer to gaston vignette for more informations and function documentation) :

```r
x <- set.stats(x)
```

```
## ped stats and snps stats have been set.
## 'p' has been set.
## 'mu' and 'sigma' have been set.
```

Here we only want to work on the Bedouin's population, so we selected this population with the following command :

```r
x.me <- select.inds(x, population == "Bedouin")
```

Please refer to the manual function of read.bed.matrix, set.stats and select.inds if needed (package gaston).

Please make sure that you have atleast some individuals with a phenotype of 2 (sick), the package only computes HBD, FLOD scores and HFLOD scores with attained individuals :

You can insure that your bed.matrix object is created and have the data needed with :

```r
str(x.me)
```

```
## Formal class 'bed.matrix' [package "gaston"] with 8 slots
##   ..@ ped                 :'data.frame': 48 obs. of  34 variables:
##   .. ..$ famid      : chr [1:48] "HGDP00607" "HGDP00608" "HGDP00609" "HGDP00610" ...
##   .. ..$ id         : chr [1:48] "HGDP00607" "HGDP00608" "HGDP00609" "HGDP00610" ...
##   .. ..$ father     : int [1:48] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ mother     : int [1:48] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ sex        : int [1:48] 2 1 1 1 1 2 2 2 2 1 ...
##   .. ..$ pheno      : int [1:48] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ population : Factor w/ 57 levels "Adygei","Balochi",..: 10 10 10 10 10 10 10 10 10 10 ...
##   .. ..$ region     : Factor w/ 27 levels "Algeria (Mzab)",..: 12 12 12 12 12 12 12 12 12 12 ...
##   .. ..$ region7    : Factor w/ 7 levels "Africa","America",..: 6 6 6 6 6 6 6 6 6 6 ...
##   .. ..$ H952       : logi [1:48] TRUE TRUE TRUE TRUE TRUE TRUE ...
##   .. ..$ N0         : int [1:48] 56890 58576 58375 60261 53403 61180 54121 55903 61857 62151 ...
##   .. ..$ N1         : int [1:48] 192705 189349 190265 186828 203585 183119 199168 196327 183012 18074
##   .. ..$ N2         : int [1:48] 394377 395838 394286 396726 386823 399508 390610 391298 398496 40115
##   .. ..$ NAs        : int [1:48] 301 510 1347 458 462 466 374 745 908 222 ...
##   .. ..$ N0.x       : int [1:48] 1514 3766 4313 4160 4434 1476 1491 1540 2261 4107 ...
##   .. ..$ N1.x       : int [1:48] 4936 0 0 0 0 4847 5354 5112 3243 0 ...
##   .. ..$ N2.x       : int [1:48] 10017 12688 12136 12289 12008 10140 9599 9810 10953 12361 ...
##   .. ..$ NAs.x      : int [1:48] 5 18 23 23 30 9 28 10 15 4 ...
##   .. ..$ N0.y       : int [1:48] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ N1.y       : int [1:48] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ N2.y       : int [1:48] 0 10 10 10 10 0 0 0 0 10 ...
##   .. ..$ NAs.y      : int [1:48] 10 0 0 0 0 10 10 10 10 0 ...
##   .. ..$ N0.mt      : int [1:48] 4 9 19 5 9 9 4 5 19 1 ...
##   .. ..$ N1.mt      : int [1:48] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ N2.mt      : int [1:48] 157 143 141 157 153 151 157 153 140 161 ...
##   .. ..$ NAs.mt     : int [1:48] 2 11 3 1 1 3 2 5 4 1 ...
##   .. ..$ callrate   : num [1:48] 1 0.999 0.998 0.999 0.999 ...
##   .. ..$ hz         : num [1:48] 0.299 0.294 0.296 0.29 0.316 ...
##   .. ..$ callrate.x : num [1:48] 1 0.999 0.999 0.999 0.998 ...
##   .. ..$ hz.x       : num [1:48] 0.3 0 0 0 0 ...
##   .. ..$ callrate.y : num [1:48] 0 1 1 1 1 0 0 0 0 1 ...
##   .. ..$ hz.y       : num [1:48] NaN 0 0 0 0 NaN NaN NaN NaN 0 ...
##   .. ..$ callrate.mt: num [1:48] 0.988 0.933 0.982 0.994 0.994 ...
##   .. ..$ hz.mt      : num [1:48] 0 0 0 0 0 0 0 0 0 0 ...
##   ..@ snps                :'data.frame': 660918 obs. of  17 variables:
##   .. ..$ chr     : int [1:660918] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ id      : chr [1:660918] "rs3094315" "rs12562034" "rs3934834" "rs9442372" ...
##   .. ..$ dist    : num [1:660918] 0.0916 0.0992 0.4963 0.5039 0.508 ...
##   .. ..$ pos     : int [1:660918] 742429 758311 995669 1008567 1011278 1011521 1011558 1020428 10214
##   .. ..$ A1      : chr [1:660918] "C" "A" "T" "A" ...
##   .. ..$ A2      : chr [1:660918] "T" "G" "C" "G" ...
##   .. ..$ N0      : int [1:660918] 3 4 6 16 3 0 3 5 8 3 ...
##   .. ..$ N1      : int [1:660918] 23 15 15 21 16 1 21 13 18 10 ...
##   .. ..$ N2      : int [1:660918] 22 29 27 11 29 47 24 30 22 35 ...
##   .. ..$ NAs     : int [1:660918] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..$ N0.f    : int [1:660918] NA NA NA NA NA NA NA NA NA NA ...
##   .. ..$ N1.f    : int [1:660918] NA NA NA NA NA NA NA NA NA NA ...
##   .. ..$ N2.f    : int [1:660918] NA NA NA NA NA NA NA NA NA NA ...
##   .. ..$ NAs.f   : int [1:660918] NA NA NA NA NA NA NA NA NA NA ...
##   .. ..$ callrate: num [1:660918] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ maf     : num [1:660918] 0.302 0.24 0.281 0.448 0.229 ...
##   .. ..$ hz      : num [1:660918] 0.479 0.312 0.312 0.438 0.333 ...
##   ..@ bed                 :<externalptr>
##   ..@ p                   : num [1:660918] 0.698 0.76 0.719 0.448 0.771 ...
##   ..@ mu                  : num [1:660918] 1.396 1.521 1.438 0.896 1.542 ...
##   ..@ sigma               : num [1:660918] 0.61 0.652 0.712 0.751 0.617 ...
##   ..@ standardize_p       : logi FALSE
##   ..@ standardize_mu_sigma: logi FALSE
```

This object contains two slots :

- ped : which gives you information about all the individuals in the data
- snps : which gives you information about the snps themselves

More information in the vignette of the gaston package.

2. Running Fantasio

We created a wrapper to make the usage of the package more simple.

We implemented in the package two differents methods in order to create n submaps :

- By "Hotspots" : with this method we use a file of recombination hotspots (downloaded from the HapMap website in hg17 (*)) to segment the genome. Segments should contain at least number_of_marker markers. Markers are then randomly selected within each segment along the genome. By doing this process we obtain a submap (a list of marker). The recombination hotspots have been converted to other buid (hg18, hg19) using hgLiftOver. The default recombination hotspots file used is in hg19. (*) http://hapmap.ncbi.nlm.nih.gov/downloads/recombination/2006-10_rel21_phaseI+II/hotspots
- By "Distance" : with this method we use a fix step based on genetic or physiscal distance (0.5 cM by default) to pick a marker randomly along the genome. More technically, segments are created whenever there is a gap larger than the step (0.5 cM) between adjacent markers. Each segment is then subdivided in several mini-segments. By default we create 20 mini-segments, each containing at least 50 markers. If this is not possible (not enough markers), we do not create mini-segments. After this process is done, we loop over the mini-segments, pick a random marker and walk through the mini-segments by picking the nearest marker after taking a step (default 0.5 cM) downstream and upstream the mini-segments.

The wrapper calls two different functions : `createSegmentsListBySnps` and `createSegmentsListBySnps`. The first function `createSegmentsListBySnps` is used to create a list of segments though the genome. The second function `makeAllSubmapsBySnps` or `makeAllSubsmapsbyHotspots` is used to create the submaps.

2.1 By Hotspots (Default)

By default, the submaps are created using the file of recombination hotspots and summarizing the results for each snp.

```r
submaps2 <- Fantasio(bedmatrix=x.me, segments="Hotspots", n=5, verbose=FALSE, list.id = "all")
```

We require that at least n.consecutive.marker markers are HBD before calling a HBD segment. Default value for n.consecutive.marker=5.

Here we need to use argument list.id = "all" because we do not have phenotype information for the HGDP-CEPH individuals. By default, Fantasio focuses on affected individuals only (status = 2).

2.2 By Hotspots by Segments

For the hotspots option, results can be summarized globally for each segment (recap.by.segments=TRUE). For this reason, n.consecutive.marker should be set to 1.

```r
submaps3 <- Fantasio(bedmatrix=x.me, segments="Hotspots", n=5, recap.by.segments=TRUE, verbose=FALSE, n
```

2.3 By Distance

```r
submaps4 <- Fantasio(bedmatrix=x.me, segments="Distance", n=5, verbose=FALSE, list.id = "all")
```

2.4 How to use the segment.option argument

In order to use the `segment.option` argument you need to pass a list of arguments, each variable names in the list must be an argument name in the function. The function that will be called is either `createsSegmentsListBySnps` if `segments` argument is equal to "Distance" or `createSegmentsListByHotspots` if `segments` argument is equal to "Hotspots" and the arguments list will be passed to it.

```r
l <- list(number_of_marker=50) #default is 0
submaps5 <- Fantasio(bedmatrix=x, segments="Hotspots", segment.options=l, n=5, recap.by.segments=TRUE, 
```

In the case of "Hotspots", by default, we do not require to have a minimum number of markers in each hotspots segments (number_of_marker = 0). With the above command line, we impose to have at least 50 markers in each segment.

3. Step by step usage of the package Fantasio

3.1 "By hotspots" method

3.1.1 Creation of the segments list

We will now create segments, which will be use to create the submaps later, further explication below, for now use this command :

```r
s <- createSegmentsListByHotspots(x.me)
```

```
## You are currently using version hg19 of hotspot
## Gathering all hotspots for the genome : ......................
## Gathering all the genome's markers : ......................
## Finding which markers are between two hotspots : ......................
```

This function creates a list of chromosomes, in each, you have a list of several segments created thanks to the hotspots file given in argument (files are given with the package), in each segments you have SNPs index.

You can watch a summary of what was done with :

```r
segmentsListSummary(s)
```

```
##    chromosome number_of_segments number_of_markers
## 1           1                904             48532
## 2           2                895             52722
## 3           3                750             43507
## 4           4                702             39040
## 5           5               721              39933
## 6           6               719              42184
## 7           7               564              34754
## 8           8               589              36417
## 9           9               563              30276
## 10         10               669              33427
## 11         11               537              31255
## 12         12               601              31039
## 13         13               475              24595
## 14         14               401              20926
## 15         15               383              19098
## 16         16               429              19028
## 17         17               374              16052
## 18         18               442              19496
## 19         19               217              10397
## 20         20               384              16228
## 21         21               225               9301
## 22         22               207               9379
```

This function creates a dataframe with three colums :

- chromosome
- number_of_segments
- number_of_marker

3.1.2 Creation of the submaps and computation

We will now head toward the creation of submaps using the following commands :

```
submaps <- makeAllSubmapsByHotspots(x.me, 5, s, verbose=FALSE, list.id = "all")
```

For the sake of clarity we have only created 5 submaps, but generally we do 100.

This function will creates 5 submaps, all the parameters can be modified (use args(makeAllSubmapsByHotspots) for more informations).

The variable submaps becomes an list.submaps object, you can watch the different elements of it with :

```
str(submaps) #careful it can become huge depending on your data sizes
```

3.1.3 Descrition of the object submaps

This object contains all the results of the different computation executed during the process of creating n submaps. Here is a complete description of each structure in this object :

- segments_list : the object Segments created previously

```
str(submaps@segments_list) #careful it can become huge depending on your data sizes
```

- atlas : the list of all the submaps created during the process. Each element of the list is an S4 object. Depending on the method you used the object can be either an `snsp.matrix` or an `hotspots.matrix` (here we use the hotspots method). Each submaps contains 15 slots :

- submap : the index of each marker picked (index from the bed.matrix object)
- ncol : the total number of marker picked
- nrow : the number of individuals
- ped : a dataframe with all the individuals genotype
- map : a dataframe with all the SNPS informations
- epsilon : genotyping error rate
- delta.dist : distance between each marker in cM/bp
- log.emiss : log of all the emission probabilities of the hidden Markov model
- a : a matrix with all the a's estimation
- f : a matrix with all the f's estimation
- likelihood0 : a matrix with all the likehood under the null hypothesis ($f = 0$)
- likelihood1 : a matrix with all the likehood under the inbred hypothesis ($f = 1$)
- p.lrt : p value of the likelihood ratio test
- HBD.prob : a matrix with all the HBD probabilities computed for each individual
- FLOD : a matrix with all the FLOD score computed

```
str(submaps@atlas)
```

- likelihood_summary : a dataframe with all the likelihood0 and likelihood1 computed over the submaps.

```
str(submaps@likelihood_summary)
```

- estimation_summary : a dataframe with all the a and f computed over the submaps

```
str(submaps@estimation_summary)
```

- marker_summary : a dataframe, which gives the number of marker and the number of times it has been picked,
  - number_of_time_picked
  - number_of_marker

```
str(submaps@marker_summary)
```

- submaps_summary : a dataframe which gives several informations about the a and f computed over the submaps. The dataframe contains 13 columns:
- FID: family identifier
- IID: individual identifier
- STATUS: status (1 non-affected, 2 affected, 0 unknown)
- SUBMAPS: number of submaps used
- QUALITY: percentage of valid submaps (i.e. submaps with $a < 1$)
- F_MIN: minimum f on valid submaps
- F_MAX: maximum f on valid submaps
- F_MEAN: mean f on valid submaps
- F_MEDIAN: median f on valid submaps (recommended to estimate f)
- A_MEDIAN: median a on valid submaps (recommended to estimate a)
- pLRT_MEDIAN: median p-value of LRT tests on valid submaps
- INBRED: a flag indicating if the individual is inbred (pLRT_MEDIAN < 0.05) or not
- pLRT_<0.05: number of valid submaps with a LRT (likelihood ratio test) having a p-value below 0.05

```
head(submaps@submap_summary)
```

```
##         FID       IID STATUS SUBMAPS QUALITY      F_MIN      F_MAX
## 1 HGDP00607 HGDP00607      1   5 / 5     100 0.02165396 0.02436498
## 2 HGDP00608 HGDP00608      1   5 / 5     100 0.03648315 0.04206039
## 3 HGDP00609 HGDP00609      1   5 / 5     100 0.03967886 0.04917616
## 4 HGDP00610 HGDP00610      1   5 / 5     100 0.04837743 0.06811777
## 5 HGDP00611 HGDP00611      1   1 / 5      20 0.00000000 0.00000000
## 6 HGDP00612 HGDP00612      1   5 / 5     100 0.05194160 0.06719188
##       F_MEAN   F_MEDIAN   A_MEDIAN  pLRT_MEDIAN INBRED pLRT_inf_0.05
## 1 0.02254044 0.02243875 0.12716348 3.361193e-25   TRUE             5
## 2 0.03880289 0.03849207 0.06624365 1.813744e-58   TRUE             5
## 3 0.04252860 0.04166841 0.12035708 7.875958e-50   TRUE             5
## 4 0.05727817 0.05600129 0.18104006 6.937032e-61   TRUE             5
## 5 0.00000000 0.00000000 0.01000000 1.000000e+00  FALSE             0
## 6 0.06054110 0.05979376 0.31707174 2.046046e-44   TRUE             5
```

- HBD_recap : a dataframe, which contains a mean of all the HBD inferences for an individual and a given marker.

```
submaps@HBD_recap[1:10, 1:10] # an individual * marker matrix
```

- FLOD_recap : a dataframe, which contains a mean of all the FLOD scores for an individual and a given marker.

```
submaps@FLOD_recap[1:10, 1:10] # an individual * marker matrix
```

- HBD_segments : a list of dataframe, each datafram is for an individuals, each dataframe contain a list of segment which will be used for plotting

```
str(submaps@HBD_segments[[1]])
```

- HFLOD : a dataframe with the value of HFLOD scores for every markers through all submaps.

```
str(submaps@HFLOD)
```

- bedmatrix : the bedmatrix object

```
str(submaps@bedmatrix)
```

- bySegments : a boolean indicating wheater the creation of summary statistics was made by segments or not
- unit : the unit of the marker (cM or Bp).
- gap : the value of the gap used to pick marker when doing submaps by Distance.

3.2 "By Hotspot" "by segments" method

We implemented a second inner method of the "By Hotspots" method. The only paramater that changes is `recap.by.segments`, it is put to TRUE. In the default "Hotspot" method the HBD probabilities and FLOD scores are computed for each marker randomly selected on each segment for the n submaps.

With the "Hotspot by segment" method HBD probabilities and FLOD scores correspond to the mean of HBD probabilities and FLOD score of each marker randomly selected on a segment for the n submaps in such a way that there is only one value for a segment.

3.2.1 Creation of the segments list

We use the same segment list that is used before (s).

3.2.2 Creation of the submaps and computation

As said before the only argument that changes is "recap.by.segments", it is put to TRUE.

```
submaps0 <- makeAllSubmapsByHotspots(x.me, 5, s, verbose=FALSE, recap.by.segments = TRUE, list.id = "al
```

3.3 "By Distance" method

3.3.1 Creation of the segments list

We will now create segments, which will be use to create the submaps later :

```
s1 <- createSegmentsListBySnps(x.me)
```

```
## Finding segments for the genome : .......................
## Finding which markers are between two segments: .......................
## Finding mini segments .......................
```

This function creates a list of chromosomes, in each, you have a list of several segments created thanks to the the gaps between markers, the value of the gap is given in argument, in each segments you have SNPS index. The function creates an object which will contains three slots :

- gap : the value of the gap
- unit : the unit of the markers ("cM" or "Bp")
- snps.segments : the list of segments

You can watch a summary of what was done with :

```
segmentsListSummary(s1)
```

This function creates a dataframe with three colums :

- chromosome
- number_of_segments
- number_of_marker

3.3.2 Creation of the submaps and computation

We will now head toward the creation of submaps using the following commands :

```r
submaps1 <- makeAllSubmapsBySnps(x.me, 5, s1, verbose=FALSE, list.id = "all")
```

The variable submaps becomes an list.submaps object, you can watch the different elements of it with :

```r
str(submaps1) #careful it can become huge depending on your data sizes
```

4. Parallelism with the package

We implemented a paralellism method to make the creation of the submaps more efficient (we paralellised the creation of the submaps, that is to say, the selection of markers). Make sure to have a Linux environment or one that can support the usage of multiple CPU.

In order to use it, use the `n.cores` argument, i.e : the number of CPU that will be used to make the differents submaps in the following functions :

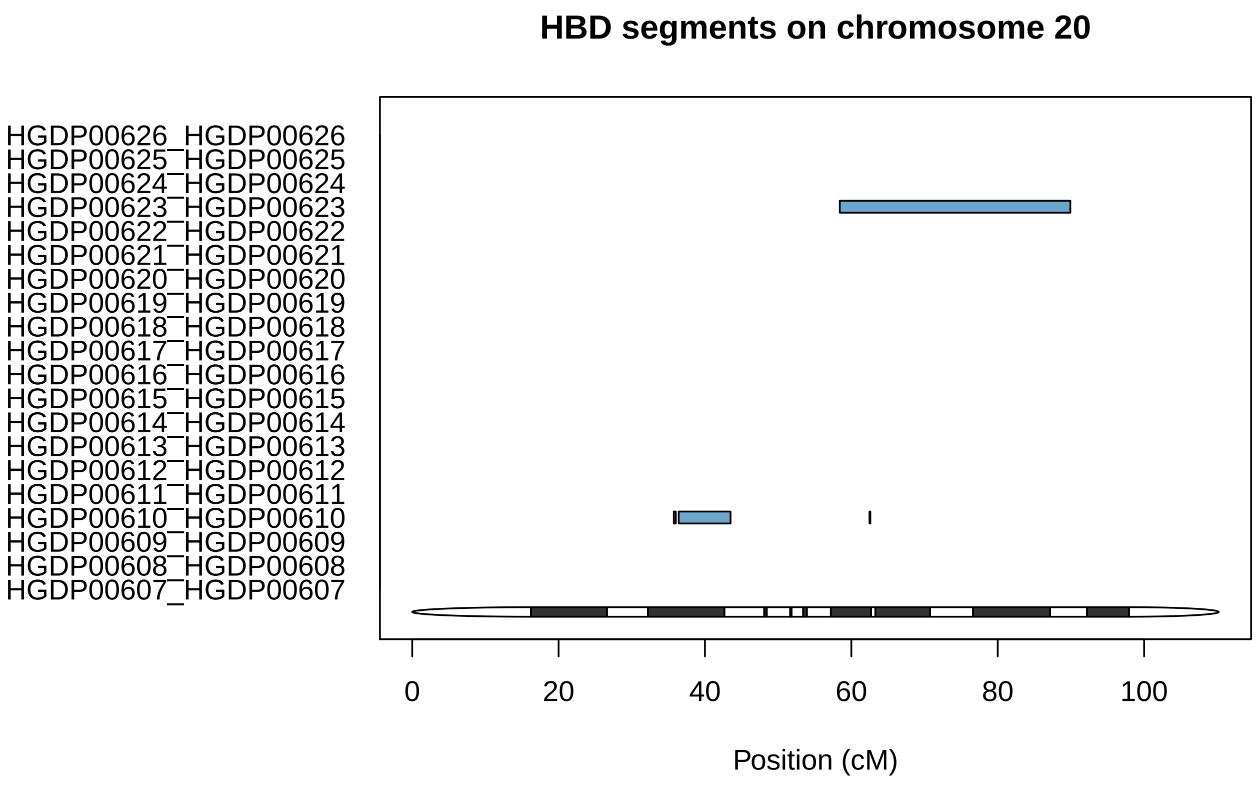
- Fantasio
- makeAllSubmapsByHotspots
- makeAllSubmapsBySnps

```r
submaps6 <- Fantasio(bedmatrix=x.me, segments="Hotspot", n=5, verbose=FALSE, n.cores=10, list.id = "all
```

5. Plotting

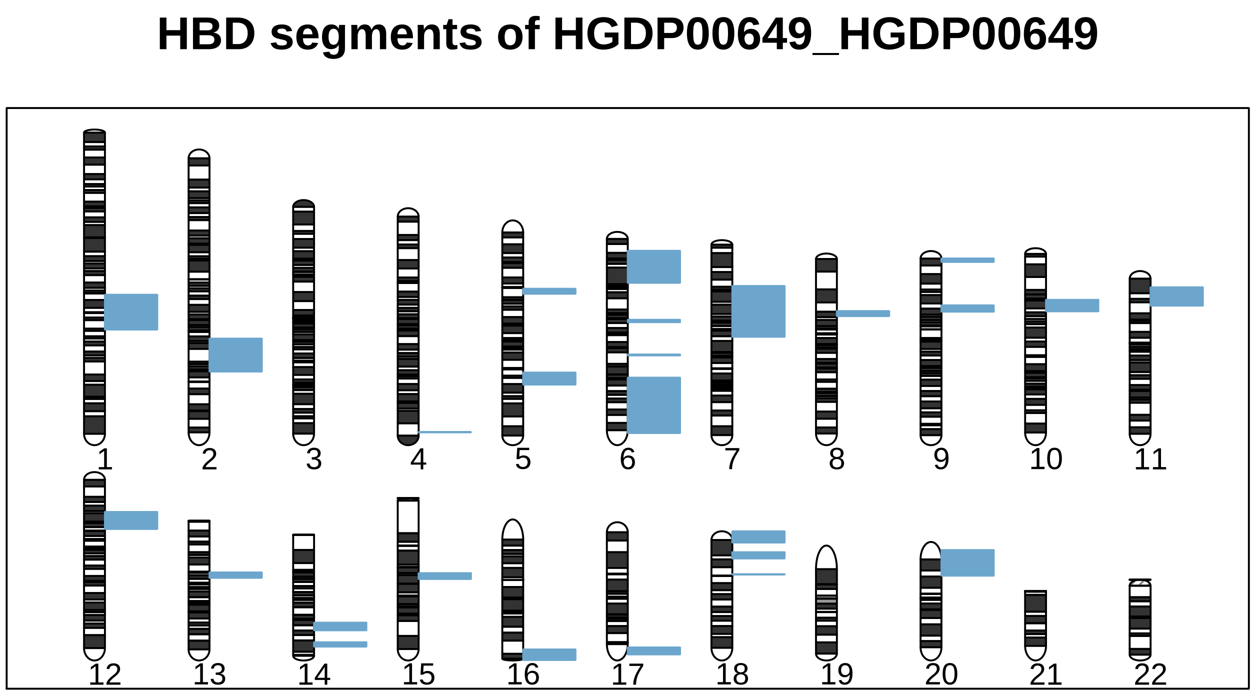
5.1 HBD plot for a chromosome

```r
HBD.plot.chr(submaps, chr=20)
```



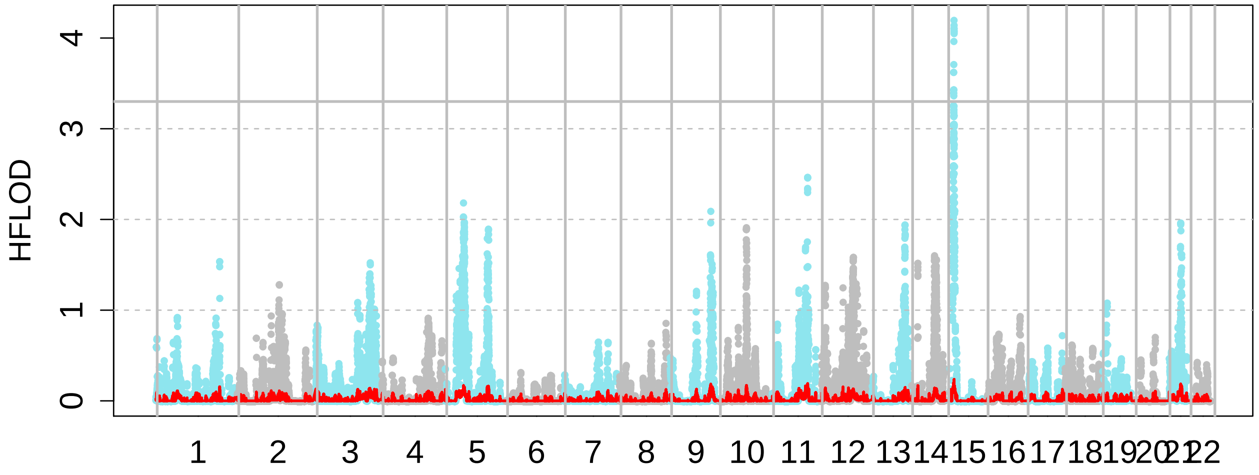
5.2 HBD plot for an individual

```r
HBD.plot.id(submaps, individual.id = "HGDP00649", family.id = "HGDP00649")
```



5.3 HFLOD manhattan plot

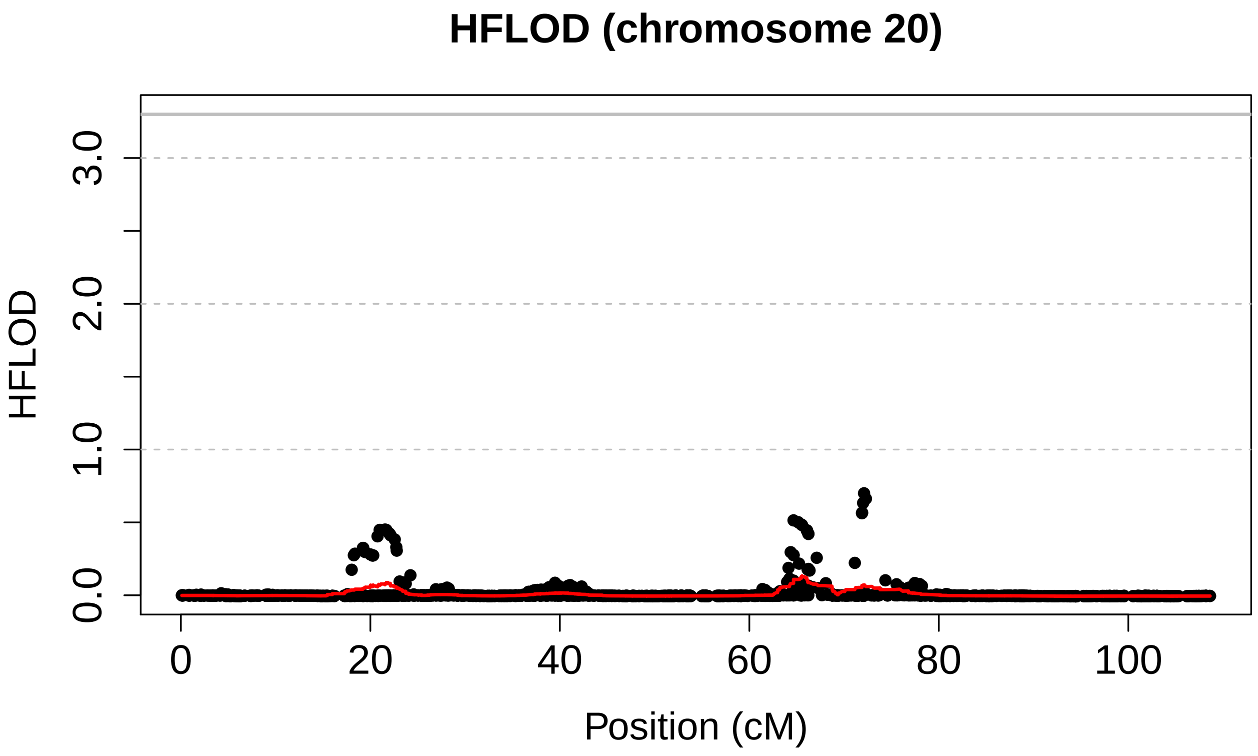
```r
HFLOD.manhattan.plot(submaps)
```



- The red lines that you see is the value of alpha for the markers.

5.4 HFLOD for a chromosome

```
HFLOD.plot.chr(submaps, chr=20)
```



- As you can see you have a red line plotted, it gives the moving average of the HFLOD, calculated on moving windows of 50 markers with the rollmean function of the R package zoo. This allows checking the consistency of HFLOD calculations (i.e. checking the fact that a high FLOD score is not due to one submap only). A moving average is computed to remove the impact of a submap with a false positive signal.