cgmisc

Package containing various functions useful in computational genetics, especially in genome-wide association studies

Marcin Kierczak

August 18, 2014

Introduction

Synopsis

Package cgmisc contains miscellaneous functions, hopefully useful for extending genome-wide association study (GWAS) analyses.

Purpose of this document

This document aims at presenting how to use functions provided in the `cgmisc` package in a typical GWAS data analyses workflow. It is, however, not aiming to be a standalone GWAS tutorial as such.

Conventions

Here, we present typographic conventions used throughout the text in order to facilitate following this document.

- All R commands are written in terminal type: `myfun(foo=T, bar=54)`
- In the above example: *myfun* is a function and both *foo* and *bar* are its arguments

Getting help

Like every other R function, the functions provided in this package are documented in the standard R-help (Rd) format and can be easily accessed by issuing **help**() or its shorter version, **?** function. For instance, if you want to get more information on how to use the clump.markers() function, type either `help(clumpmarkers)` or `?clump.markers` and press return/enter. To see this document from within R you type `vignette('cgmisc')`.

Working with `cgmisc`

Installation

In order to install `cgmisc`, you either use one of the R GUIs (native R GUI, RStudio etc.) or type the following command:

```
install.packages("cgmisc", repos="")
```

Functions in the `cgmisc` package often complement or use `GenABEL` package functions and data structures. `GenABEL` is an excellent and widely-used R package for performing genome-wide association studies and much more... Therefore `GenABEL` will be loaded automagically when loading cgmisc. You can load `cgmisc` package as follows:

```
require("cgmisc")
```

Loading and preparing example data

First, we load the data, in this case internal cgmisc data.

```
data(cgmisc_data)
```

And run some quality checks. It is important not to exclude any markers due to their low minor allele frequency as we will be looking for stretches of homozygosity.

```
qc <- check.marker(data, callrate = 0.95, perid.call = 0.95, maf = -1)
data.clean <- data[qc$idok, qc$snpok]
```

Once data is clean and neat, we can look at genomic kinship and try to determine population structure:

```
gkin <- ibs(data.clean, weight = "freq")
gkin.dist <- as.dist(0.5 - gkin)
gkin.mds <- cmdscale(gkin.dist)
plot(gkin.mds, xlab = "MDS1", ylab = "MDS2", las = 1, bty = "n")
grid()
```

So, it seems there is some population structure in here. Just by eyeballing, we can spot two distinct clusters of individuals.[1] Let's run a simple k-means clustering to determine which individuals belong to which cluster.

```
km <- kmeans(gkin.mds, centers = 2)
pop <- km$cluster
plot(gkin.mds, xlab = "MDS1", ylab = "MDS2", col = pop, las = 1, bty = "n")
grid()
```

[1] Well, OK, we actually did a scree test to see there is two clusters here. Eyeballing is not very scientific.

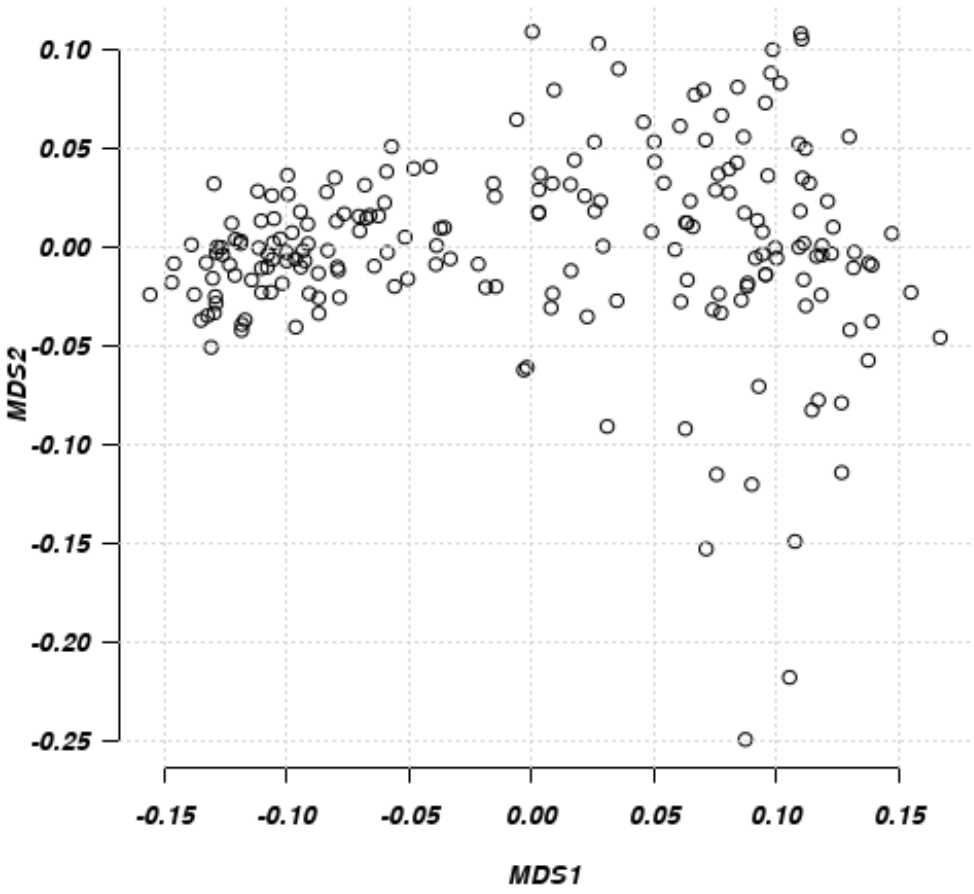


Figure 1: An MDS projection of genomic kinship.

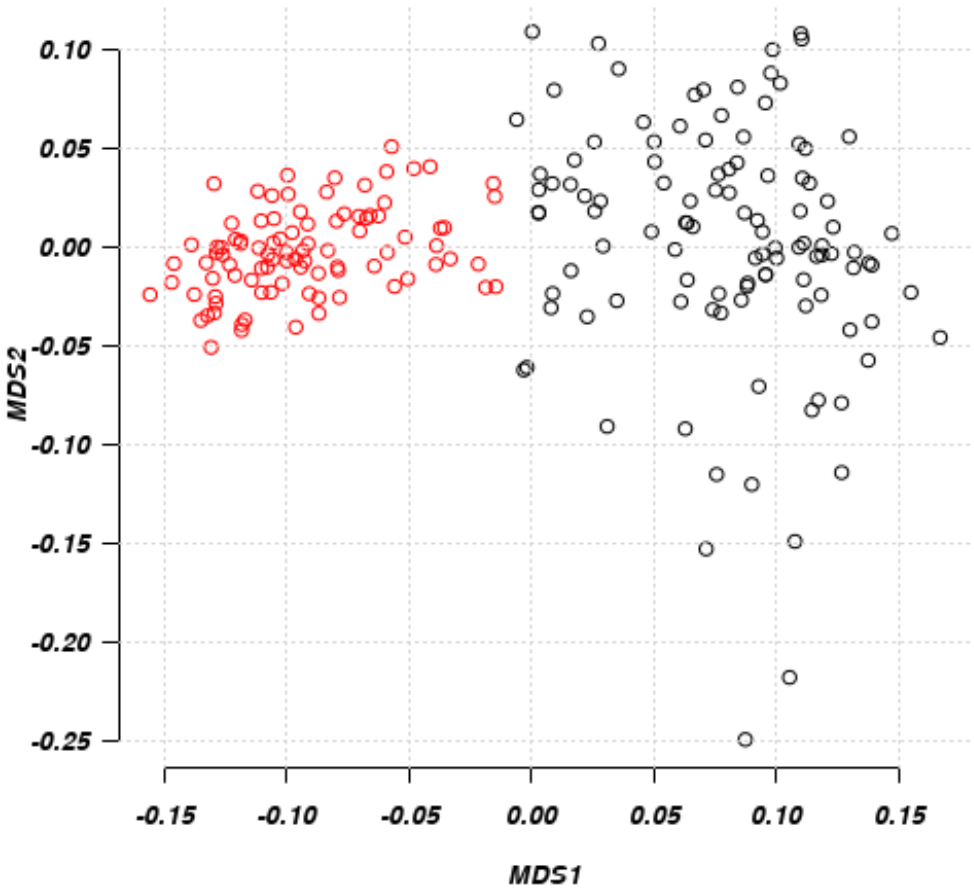


Figure 2: An MDS projection of genomic kinship.

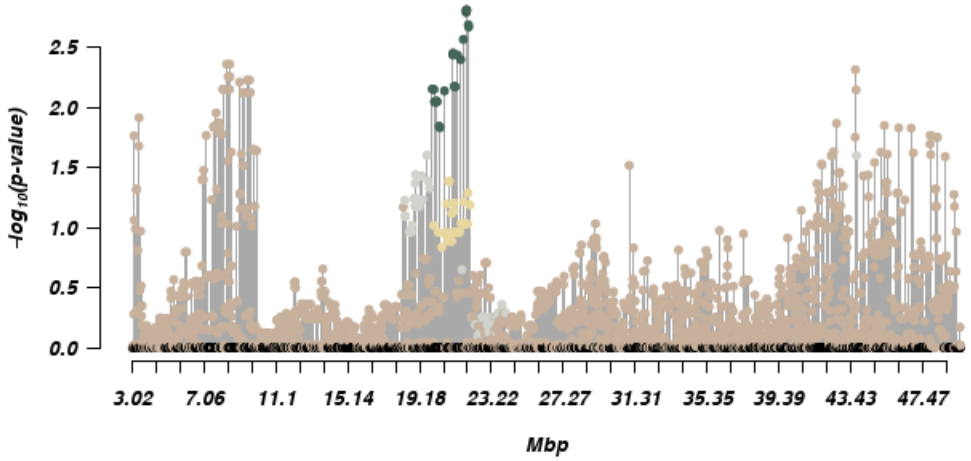
Comparing allele counts between populations

To compare allele counts between two populations, you first need to call `pop.allele.counts` function. You supply data (in our case only chromosome 27) and pop vector which assigns each individual to a population. Populations have to be encoded as 1 and 2. In addition, there is an option `progress` which turns on and off displaying of the progress bar.

```
dat <- data.clean[, data.clean@gtdata@chromosome == "27"]
allele.counts <- pop.allele.counts(data = dat, pops = pop, progress = F)
```

Now, when the tests are finished, you may want to plot the result. If you are plotting single chromosome, you may want to turn on coloring markers by LD (still experimental option).

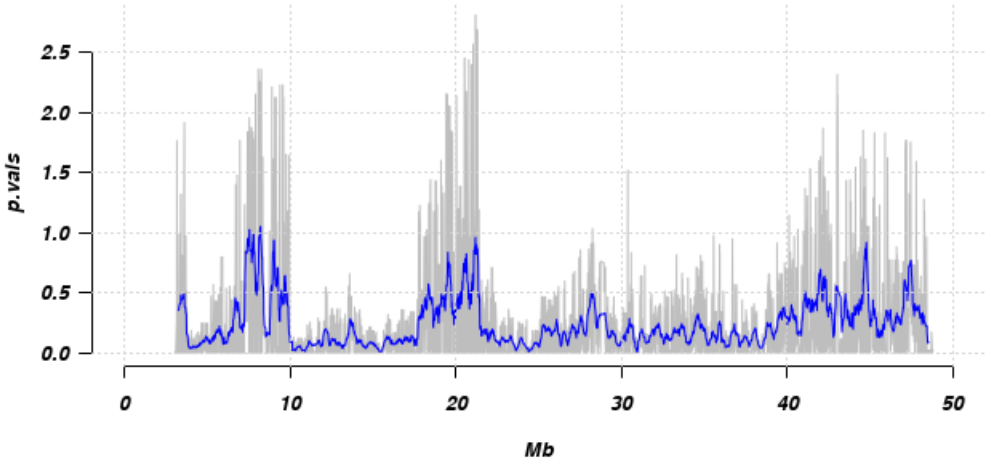
```
plot.pac(data = dat, allele.cnt = allele.counts, plot.LD = T)

## Loading required package:  wesanderson
```



But what if you want to plot the result in a custom way, say applying some window-based smoothing? The simplest type of window will be based on a number of markers, say we want to apply smoothing based on a sliding window of 21 markers:

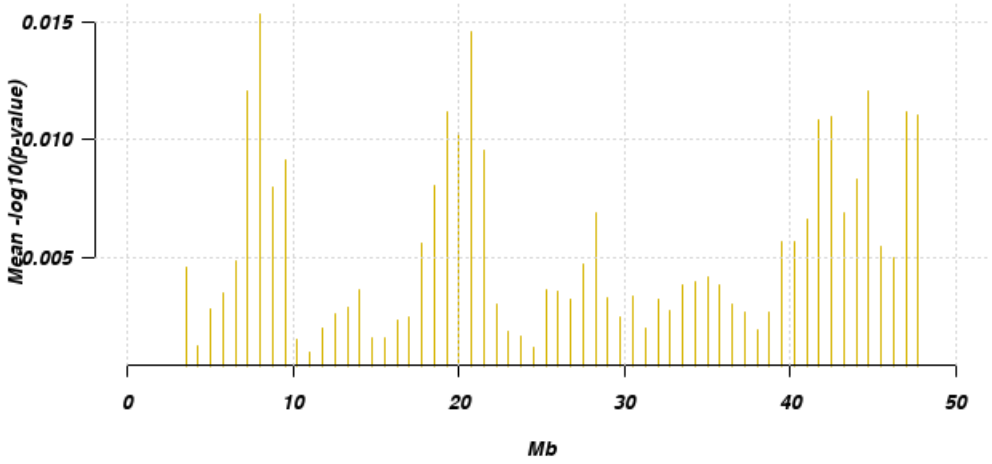
```
coords <- allele.counts@map
p.vals <- -log10(allele.counts@p.values)
# Handle NAs for the nice look
p.vals[is.na(p.vals)] <- 0
filter21 <- rep(1/21, 21)
smooth <- filter(p.vals, filter21, sides=2)
```

```
plot(coords/1e6, p.vals, xlab="Mb", type='l', col="grey",
     bty='n', xlim=c(0,50), las=1)
lines(coords/1e6, smooth, col='blue')
grid()
```



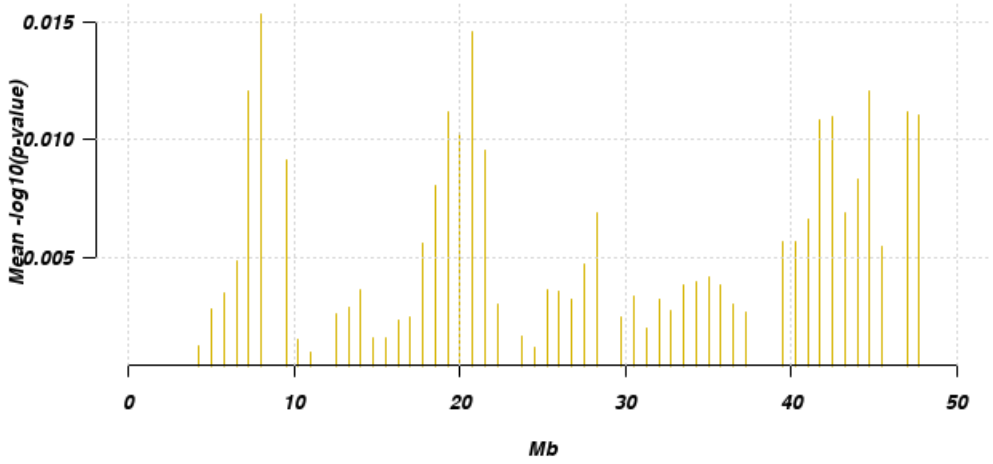
And what if you want the windows to span certain number of base pairs?

```
# Prepare a matrix of overlapping windows
W <- get.overlapping.windows(data = dat, chr = 27, size = 1e6,
                             overlap = 25e4)
# Get mean values per window
PW <- apply(W[[2]], MARGIN = 1, FUN = function(x) { mean(x * p.vals) } )
# Do plotting
plot(W[[1]]$midpoint/1e6, PW, type='h',
    col=wes.palette(2,name = "Cavalcanti")[1],
    xlim=c(0,50),
    las=1, bty='n', xlab="Mb", ylab="Mean -log10(p-value)")
grid()
```



It is also quite common to discard windows which contain just a few markers. Here, we set the threshold to 65 markers in a window. It is probably way too high, but we do it just as an example of how one can do filtering.

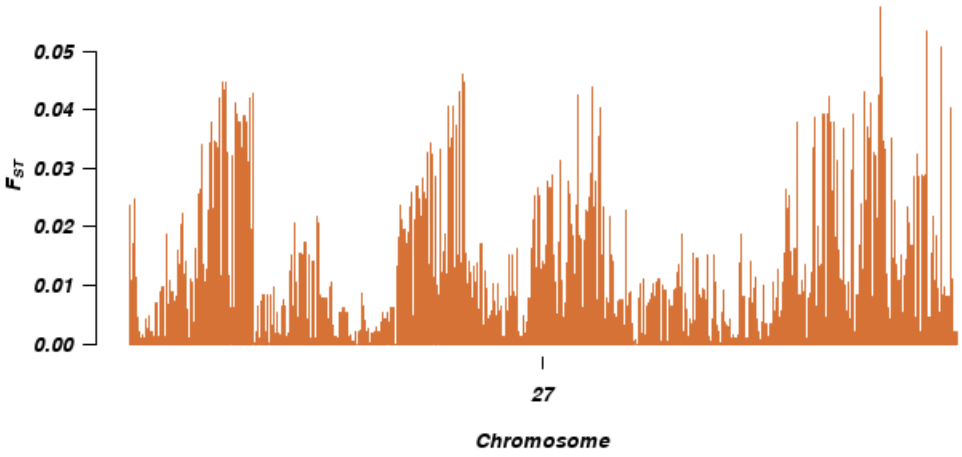
```
window.len <- apply(W[[2]], MARGIN = 1, sum)
threshold <- 65
W.filtered <- list(W[[1]][window.len >= threshold,],
                   W[[2]][window.len >= threshold,])
PW.filtered <- apply(W.filtered[[2]],
                     MARGIN = 1,
                     FUN = function(x) { mean(x * p.vals) } )
plot(W.filtered[[1]]$midpoint/1e6, PW.filtered, type='h',
    col=wes.palette(2,name = "Cavalcanti")[1],
    xlim=c(0,50),
    las=1, bty='n', xlab="Mb", ylab="Mean -log10(p-value)")
grid()
```



Computing and plotting F_{ST}

The fixation index (F_{ST}) is a widely used measure of population differentiation due to genetic structure. `cgmisc` package provides a function `compute.Fstats` that, along with `plot.Fstats` function, enable computation and visualization of F_{ST} for a pair of populations.

```
fst <- compute.Fstats(data = dat, pops = pop)
plot.fstats(data = dat, fstats = fst)
```



The `computeFstats` function returns an object of `fstats.result` class which contains two *slots*: global statistics, statistics for populations. Global statistics is a *data frame* and can be accessed by using either `glob(fst)` or `fst@glob` notation. Let's try to see global statistics for the 5 first markers:

```
fst@glob[1:5, ]
```

```
##    p.bar    q.bar       HI       HS       HT       FIS      FST       FIT
## 1 1.0000 0.000000 0.000000 0.000000 0.000000       NaN      NaN       NaN
## 2 1.0000 0.000000 0.000000 0.000000 0.000000       NaN      NaN       NaN
## 3 0.9975 0.002463 0.004926 0.004909 0.004914 -0.003517 0.001044 -0.002469
## 4 1.0000 0.000000 0.000000 0.000000 0.000000       NaN      NaN       NaN
## 5 0.9951 0.004926 0.009852 0.009783 0.009804 -0.007058 0.002093 -0.004950
```

Here, we can see 8 different columns for every marker. The meaning of these is:

- p.bar and q.bar – allele frequencies based on all individuals,
- HI, HS and HT – global heterozygosity indices based on the observed allele frequencies, allele frequencies expected under the *null* and allele frequencies computed using all individuals, respectively.
- FIS, FST and FIT – inbreeding coefficient individual/subpopulation (F_{IS}), subpopulation effect (F_{ST}) and inbreeding coefficient individual/total population (F_{IT})

Now, let us examine statistics obtained for the first 5 markers in subpopulation 2:

```
fst@pops[[2]][1:5, ]
```

```
##                AA Aa aa   N      p        q exp.AA exp.Aa   exp.aa
## TIGRP2P347125 110  0  0 110 1.0000 0.000000    110 0.0000 0.000000
## BICF2P1061825 110  0  0 110 1.0000 0.000000    110 0.0000 0.000000
## BICF2P522661  109  1  0 110 0.9955 0.004545    109 0.9955 0.002273
## BICF2P577353  110  0  0 110 1.0000 0.000000    110 0.0000 0.000000
## BICF2P467643  108  2  0 110 0.9909 0.009091    108 1.9818 0.009091
##                obs.het exp.het    dev.het        Fs
## TIGRP2P347125 0.000000 0.00000  0.000e+00       NaN
## BICF2P1061825 0.000000 0.00000  0.000e+00       NaN
## BICF2P522661  0.009091 0.00905 -4.132e-05 -0.004566
## BICF2P577353  0.000000 0.00000  0.000e+00       NaN
## BICF2P467643  0.018182 0.01802 -1.653e-04 -0.009174
```

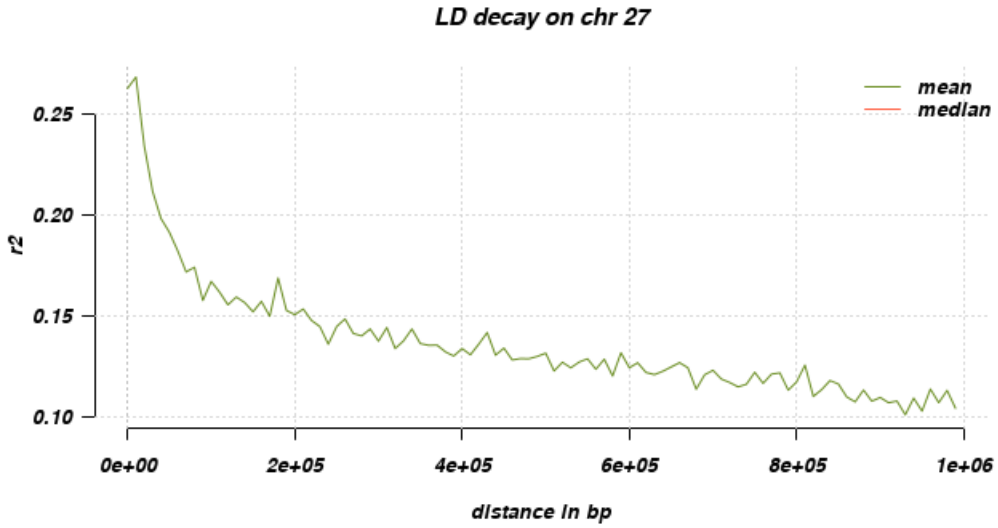
Here, we have the following columns per each marker:

- AA, Aa and aa – observed genotype counts,
- p, q – observed allele frequencies,
- exp.AA, exp.Aa, exp.aa – genotype frequencies expected under the *null*,
- obs.het, exp.het, dev.het – observed heterozygosity, heterozygosity expected under the *null* and difference of the two above,
- Fs – F-statistics value

Visualizing LD-decay

To visualize average rate of LD-decay with the distance between markers, one can use the `plot.LD.decay` function. It lets one specify the minimal and the maximal distance considered as well as the number of bins used in binning LD values. The more bins, the more "jumpy" the curve will be, the less bins the "smoother" the curve but at the same time the lower resolution. So choose something aroud 100 bins as a good compromise.

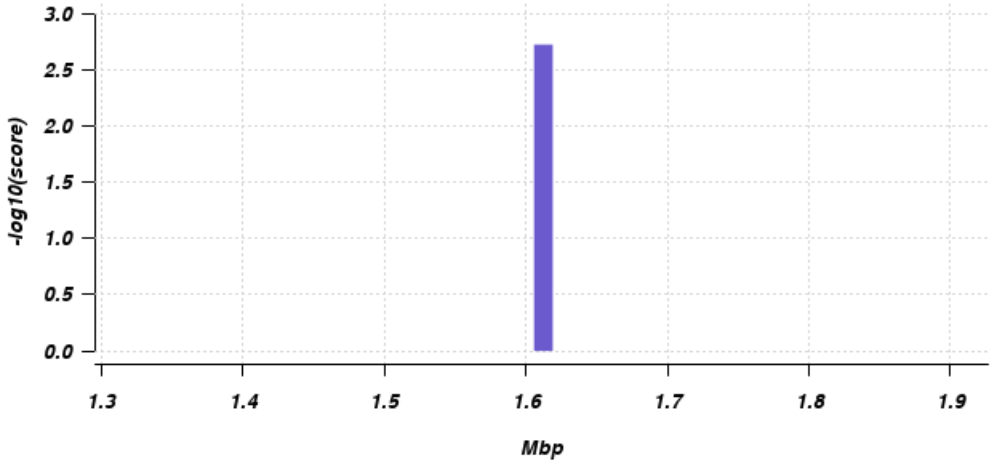
```
plot.LD.decay(data=dat, N=100, dmin=0, dmax=1e6,
              main="LD decay on chr 27")
```



Endogenous retroviral sequences

Given a set of coordinates in canFam3, returns $-log(p - value)$ of hits reflecting the likelihood that a particular region is an endogenous retroviral sequence (ERV).

```
plot.erv("chr16", coords = c(1500000, 1700000))
```



Association Analysis

Some of `cgmisc` functions use data which are the result of GWAS analyses. Let's perform GWAS on our data to obtain `GenABEL` `scan.gwaa-class` object :

```
an0 <- qtscore(response ~ sex, data = data)

## Warning:  1 observations deleted due to missingness
```

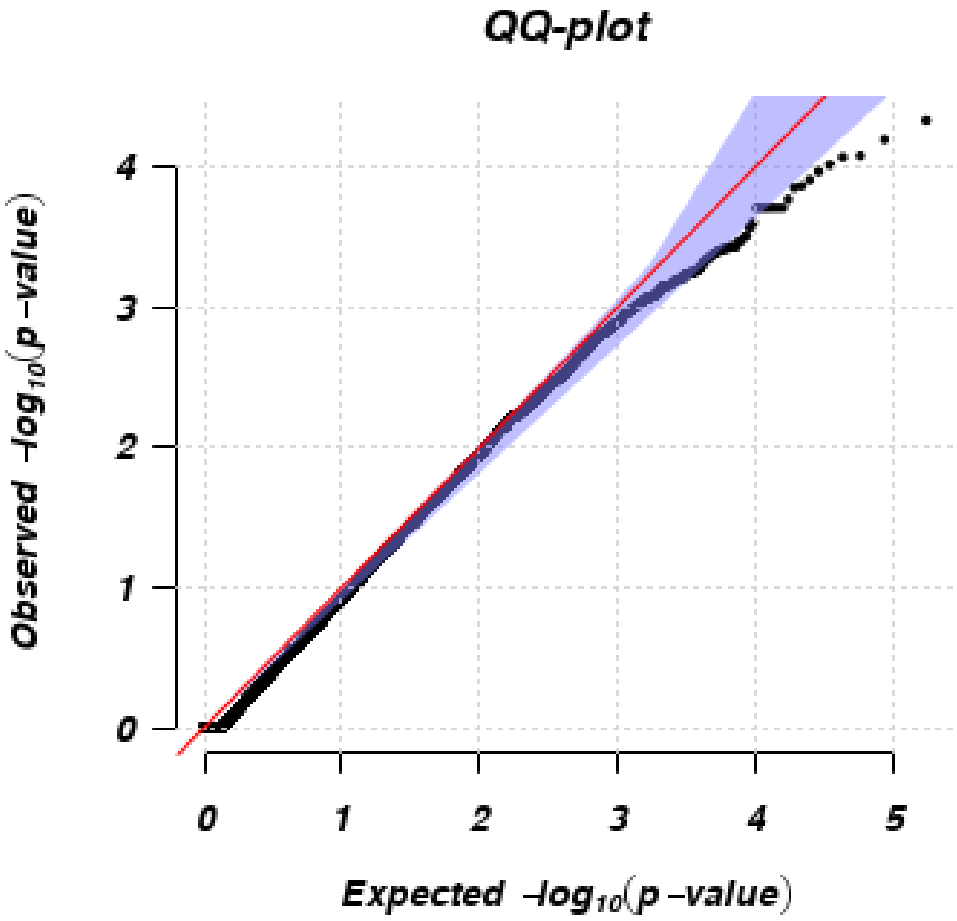
And have a look at top 5 markers

```
summary(an0, top = 5)

## Summary for top 5 results, sorted by P1df
##                 Chromosome Position Strand A1 A2   N    effB se_effB
## BICF2P1063345           34 40399702      u  T  G 206 -0.2834 0.06954
## BICF2P682714             1 15399848      u  A  G 189  0.5887 0.14714
## BICF2G630569243          6 80458945      u  C  A 206  0.3071 0.07797
## BICF2S2366791            6 70667322      u  C  T 206  0.2628 0.06682
## BICF2G630450144         34 40416964      u  A  G 206 -0.2705 0.06933
##                 chi2.1df      P1df   effAB   effBB chi2.2df      P2df
## BICF2P1063345      16.60 4.609e-05 -0.3644 -0.5354    17.51 1.579e-04
## BICF2P682714       16.01 6.309e-05  0.5912  1.1515    16.01 3.339e-04
```

```
## BICF2G630569243    15.51 8.216e-05  0.2904  0.6358    15.57 4.167e-04
## BICF2S2366791      15.46 8.410e-05  0.4558  0.4732    20.15 4.202e-05
## BICF2G630450144    15.22 9.573e-05 -0.3339 -0.5161    15.77 3.756e-04
##                       Pc1df
## BICF2P1063345   7.357e-05
## BICF2P682714    9.911e-05
## BICF2G630569243 1.274e-04
## BICF2S2366791   1.302e-04
## BICF2G630450144 1.472e-04
```

Plotting qunatile-quantile plot of p-values

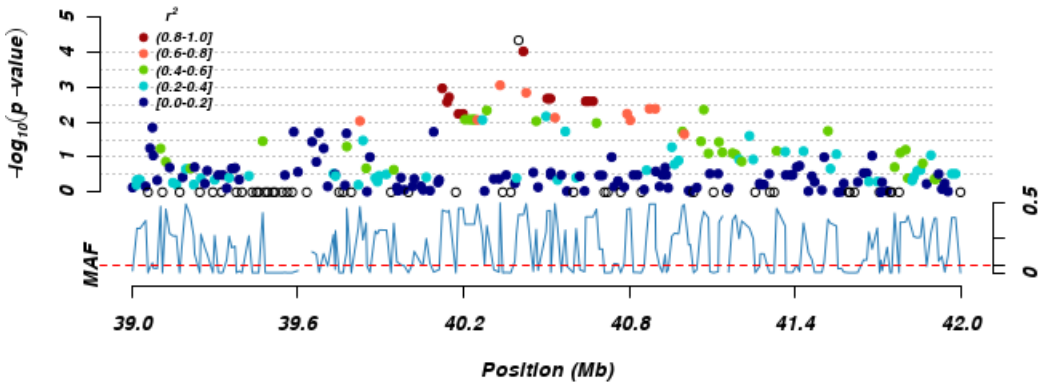
```
plot.qq(pvals = an0@results$P1df, conf = c(0.05, 0.95), step = 100)
```



Plot.Manhattan.LD

The `plot.Manhattan.LD` function allows you to visualize the LD pattern in a genome fragment on an enchanced Manhattan plot. You select one marker, typically the one with the strongest association to the analysed trait and all other markers in the region are coloured according to the degree of linkage disequilibrium with this index marker.

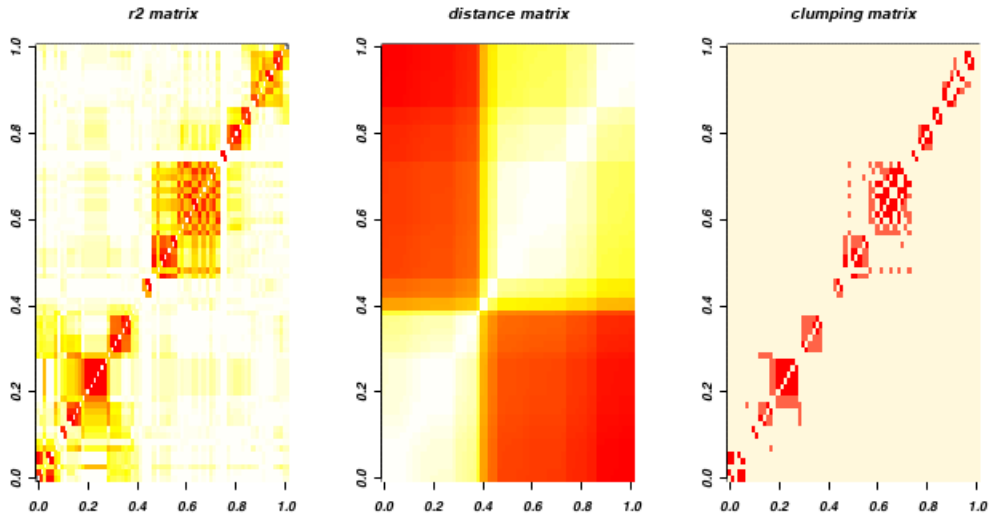
```
plot.manhattan.LD(data, an0, chr=34, region=c(39e6,42e6),
                  index.snp = 'BICF2P1063345', bonferroni = F)
```



Clumping

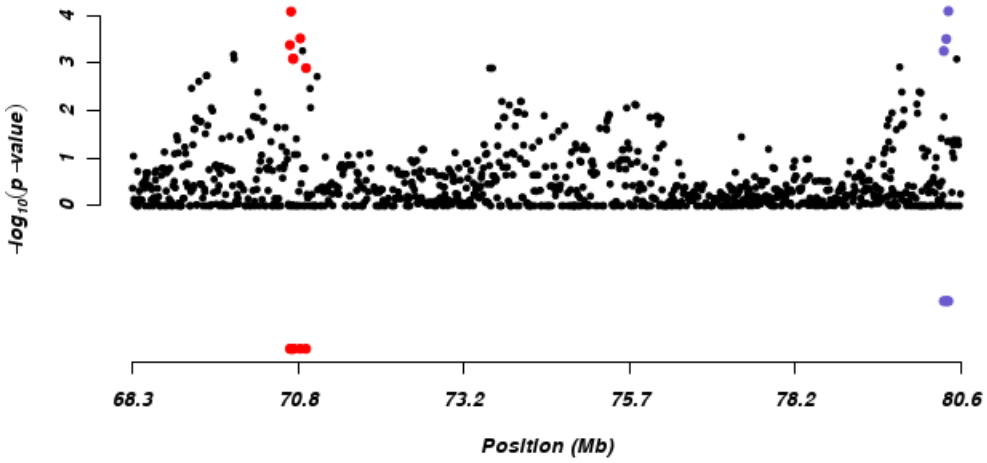
`clump.markers` function implements clumping procedure described in PLINK documentation. Clumping is based on linkage disequilibrium. The function returns list of clumps which can be used for further analyses or plotted using `plot.clumps` function included in our package.

```
clumps <- clump.markers(data, gwas.result = an0, chr = 6, bp.dist = 250000,
    p1 = 1e-04, p2 = 0.01, r2 = 0.5, image = T)
```



Now, one can plot clumps using the `plot.clumps()` function:

```
plot.clumps(gwas.result = an0, clumps = clumps,
            chr = 6, region = c(68.3e6, 81.6e6))
```



Data conversion functions

gwaa2bed

gwaa2bigRR

gwaa2PHASE

gwaa2vgwas

phase2fasta

phase2haploview

Convenience functions

open.region.UCSC

get.overlapping.windows

get.erv

get.adjacent.markers

get.chr.midpoints

get.LD.colors.r

create.haploview.info

choose.top.snps

Species-specific functions

Xfix.canfam