

cgmisc: package tutorial

*Marcin Kierczak, Jagoda Jabłońska, Simon Forsberg, Matteo Bianchi, Katarina Tengvall,
Mats Pettersson, Jennifer Meadows, Patric Jern, Orjan Carlborg, Kerstin Lindblad-Toh*

22 April 2015

Contents

Package installation	1
Loading data	2
Example analyses	2
Comparing subpopulations	4
Visualization and analyses of linkage structure	6
Detecting runs of homozygosity	9
Computing average heterozygosity using overlapping windows approach.	9
Phenotype boxplots	10
Endogenous retroviral sequences (ERV)	11
Conversion functions	12
Improved quantile-quantile plots	13
Dog-specific utilities	13
Interacting with the UCSC genome browser	14
Simple epistasis scan	14
List of aliases	15

cgmisc is an R package for enhanced genome-wide association studies (GWAS) and visualisation. This document aims at guiding you through the installation process and to demonstrate package capabilities in a series of practical examples based on an example data included in the package.

Package installation

The **cgmisc** package is available on GitHub repository and can be easily installed with the help of `devtools` package:

```
install.packages('devtools') # Install devotees if not installed
library(devtools)
install_github('mk-lab/cgmisc')
```

After the package has been installed, to use the package, it is necessary to load it into environment:

```
library("cgmisc")
```

Loading data

Whenever possible, the **cgmisc** package uses data structures used by the GenABEL (Aulchenko et al., 2007) package. In particular, the **gwaa.data-class** and the **gwaa.scan-class** structures are used. The package is shipped with an example dataset called **cgmisc_data** that contains genotyping data (Illumina, canFam2) for N=207 German shepherds originally collected for the project described in (Tengvall et al., 2012). The phenotypes, though, have been simulated in order to be able to illustrate various features of **cgmisc**. To load the example dataset, use the following command:

```
data("data")
```

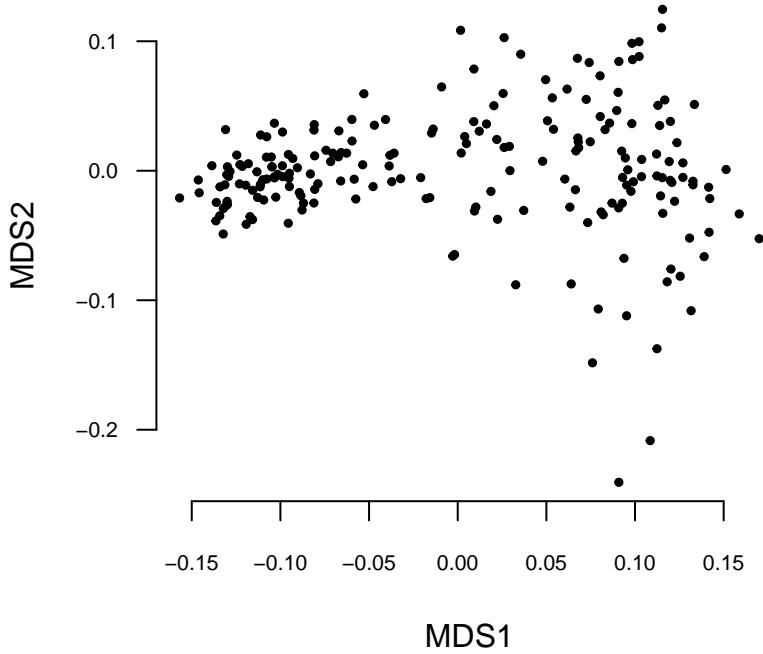
Example analyses

In order to illustrate how to use particular functions, we will perform a very much simplified GWAS analysis. We begin by initial quality control where we prune the data with per marker of per individual call rates below 95%. Based on 2000 randomly selected markers, we remove one (with lower call rate) from each pair of too similar (more than 95% similarity) individuals. We also set very low (10^{-3}) threshold for pruning on minor allele frequency (in practise only the monomorphic markers will be removed) and turn off checks based on the departure form Hardy-Weinberg equilibrium (p.level=10e-18)

```
qc1 <- check.marker(data, callrate = .95, perid.call = .95, ibs.threshold = .95,
                      ibs.mrk=2000, ibs.exclude="lower", p.level=10e-18, maf=1e-3)
data.qc1 <- data[qc1$idok, qc1$snpok]
```

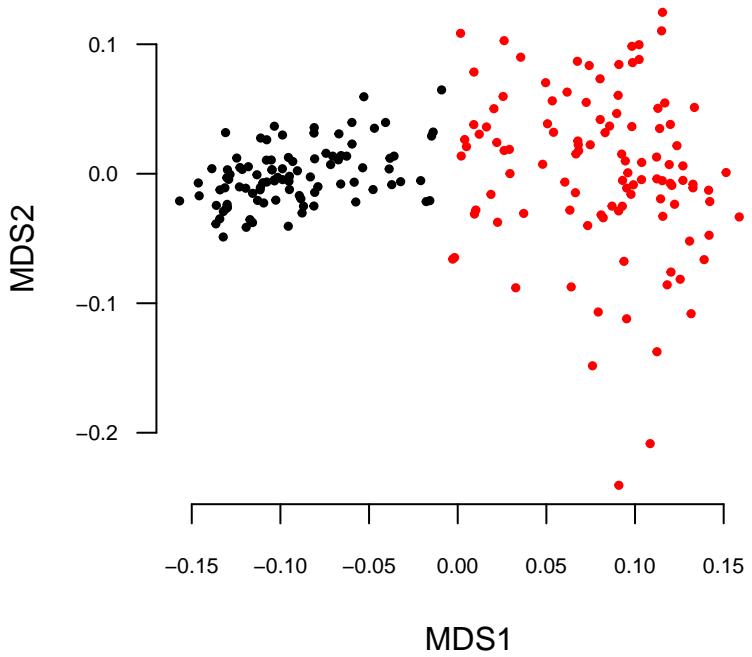
Next, we analyse population structure by means of genomic-kinship. Using all autosomal chromosomes, we calculate the Identity By State (IBS) matrix. In order to visualize the population structure, we then reduce the matrix to two dimensions by multidimensional scaling.

```
autosomal <- which(data.qc1@gtdata@chromosome != 39)
data.qc1.gkin <- ibs(data.qc1, snpsubset = autosomal, weight = 'freq')
data.qc1.dist <- as.dist(0.5 - data.qc1.gkin)
data.qc1.mds <- cmdscale(data.qc1.dist)
plot(data.qc1.mds, pch=19, cex=.5, las=1, xlab="MDS1", ylab="MDS2", cex.axis=.7, bty='n')
```



We can see that there is possible population structure here. We should investigate this further, but for our purposes, let's just run simple K-means clustering with the number of clusters *a priori* set to $K = 2$

```
kclust <- kmeans(data.qc1.mds, centers = 2)
plot(data.qc1.mds, pch=19, cex=.5, las=1, xlab="MDS1", ylab="MDS2", cex.axis=.7,
      bty='n', col=kclust$cluster)
```

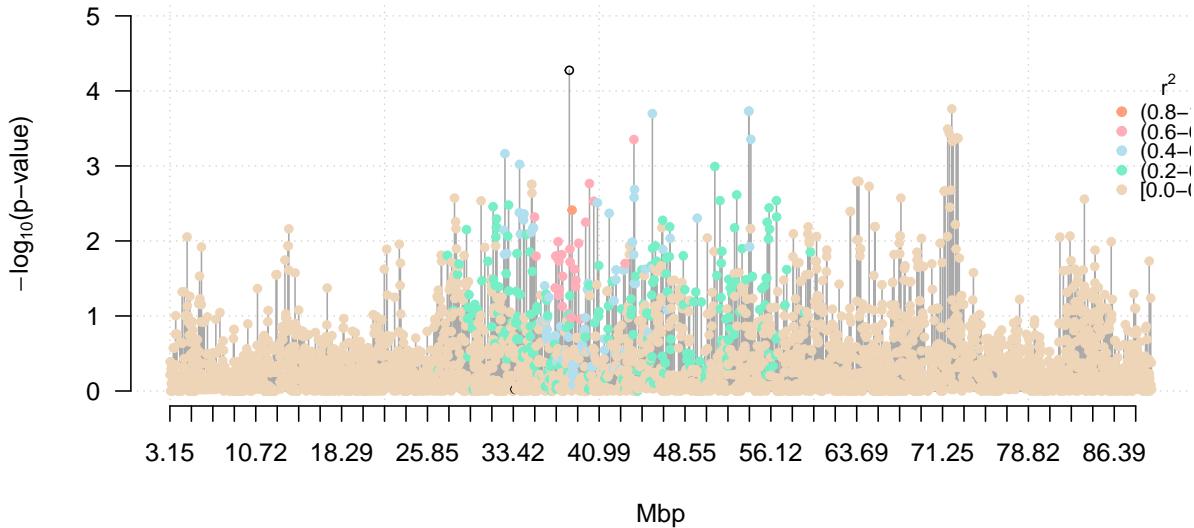


```
# Now, we can assign our individuals to calculated clusters which represent subpopulations.
pop <- kclust$cluster
```

We can compare subpopulations looking at the differences in the reference allele frequency using the `pop.allele.counts` and the `plot.pac` functions. Here, we just focus on chromosome 2.

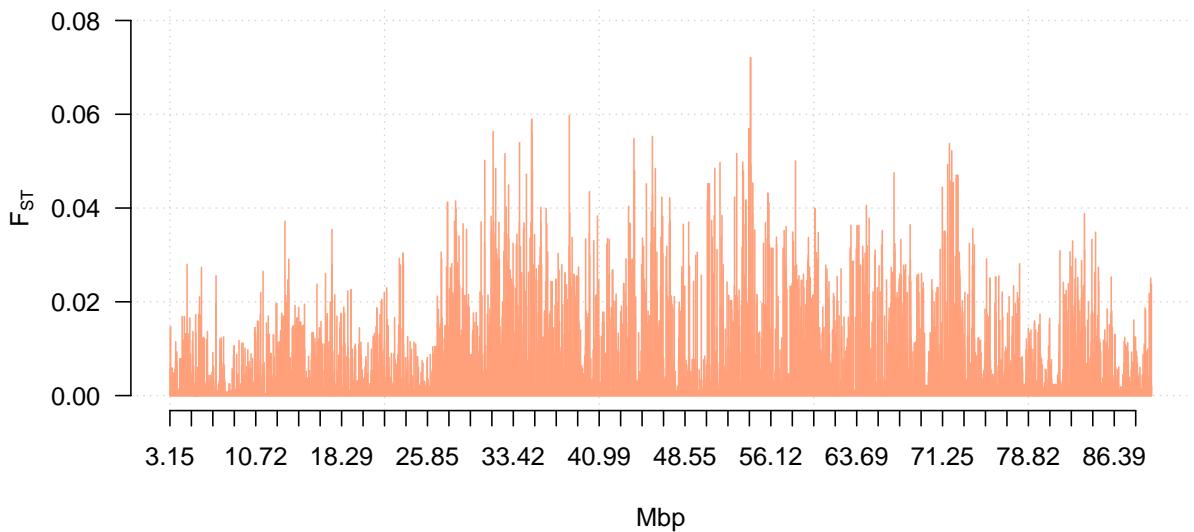
Comparing subpopulations

```
pac <- pop.allele.counts(data.qc1[,data.qc1@gtdata@chromosome==2], pop, progress=F)
plot.pac(data.qc1[,data.qc1@gtdata@chromosome==2], allele.cnt = pac, plot.LD = T)
```



In a similar way, we can compute and plot fixation index F_{ST} :

```
fst <- compute.fstats(data.qc1[,data.qc1@gtdata@chromosome==2], pop)
plot.fstats(data.qc1[,data.qc1@gtdata@chromosome==2], fst)
```

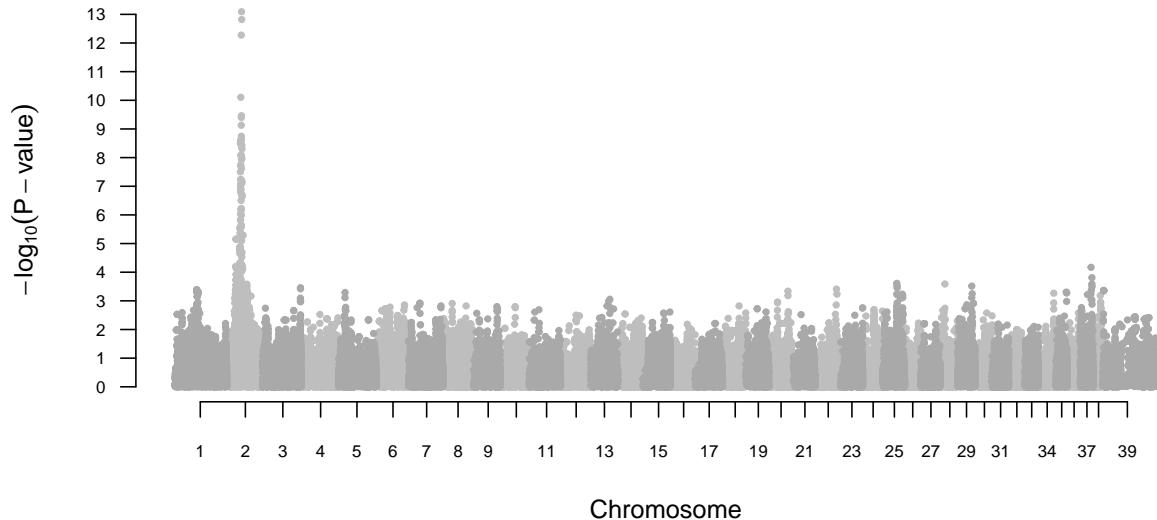


Having defined subpopulations, we can proceed to association analyses using mixed model with genomic kinship as random effect.

```
h2h <- polygenic_hglm(formula = ct ~ sex, data.qc1.gkin, data.qc1)
mm <- mmscore(h2h, data.qc1, strata = pop)
```

```
par(las=1, cex.axis=.7) # Tweak graphics
plot(mm, cex=.5, pch=19, col=c("darkgrey","grey"))
```

mmscore(h2h, data.qc1, pop)



As we can see, there is a very strong association signal on chromosome 2. We can examine it a bit closer using the `plot.manhattan.ld` function.

Visualization and analyses of linkage structure

Say, we would like to zoom in on chromosome 2 and visualise LD to the top-associated marker. First, we need the name and coordinates of the marker:

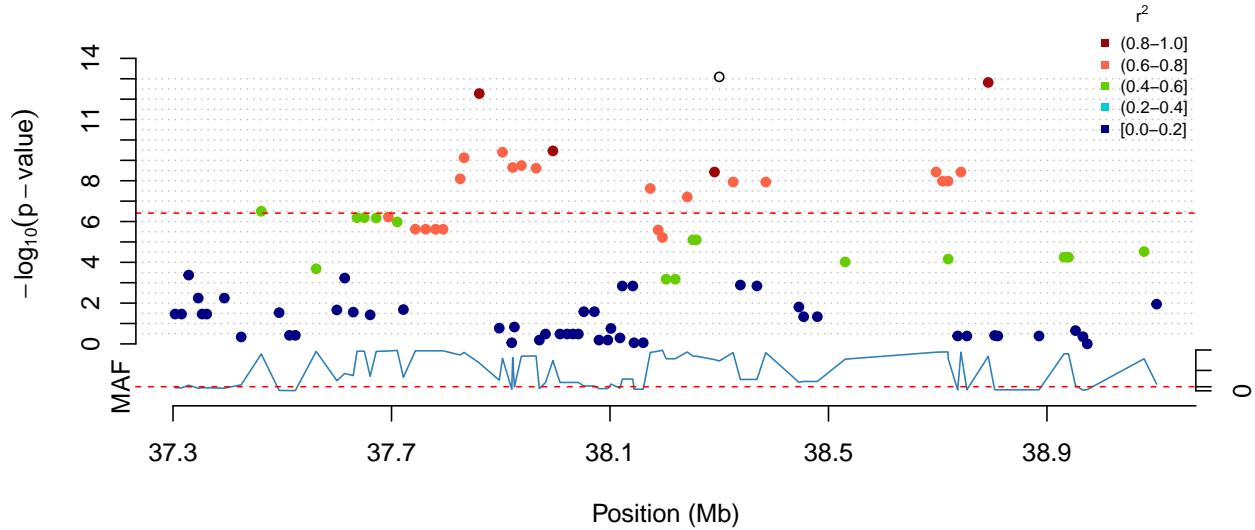
```
summary(mm, top=1)
```

```
## Summary for top 1 results, sorted by P1df

##          Chromosome Position Strand A1 A2   N      effB    se_effB
## BICF2S2365880      2 38256927      u  T  C 205 1.095257 0.1466398
##          chi2.1df      P1df      Pc1df effAB effBB chi2.2df P2df
## BICF2S2365880 55.78642 8.078765e-14 3.540639e-13     NA     NA      0    NA
```

We see that the top-associated marker is **BICF2S2365880** and its position is **38256927bp**. We will zoom in on a 2Mbp region centered on the marker using `plot.manhattan.ld`. Aforementioned function produces a standard so-called Manhattan plot with color-coded linkage disequilibrium (LD) to a specific reference marker measured by r2. Given genotyping data (as GenABEL gwaa.data-class object) and GWAS result in the form of p-values vector together with genetic coordinates of a region to be plotted (up to entire chromosome), `plot.manhattan.ld` produces a plot with genomic coordinates on the x-axis and $-\log_{10}(p\text{-value})$ on the Y axis. Linkage disequilibrium to a reference marker is represented by specific colours. By default, colours represent 5 discrete intervals of r2: [1.0, 0.8]; [0.8, 0.6]; [0.6, 0.4]; [0.4, 0.2]; [0.2, 0], where: [a, b[= {x ??? | b ??? x > b}. These intervals can be altered using function parameters.

```
plot.manhattan.LD(data = data.qc1, gwas.result = mm, chr = 2, region = c(37256927,39256927),
                   index.snp = "BICF2S2365880", legend.pos = "topright")
```



To extract markers from a given genomic region that are in high LD to a given index marker the `choose.top.snps` function can be used.

```
top.snps <- choose.top.snps(data = data.qc1, chr = 2, region = c(37256927,39256927),
                               index.snp = "BICF2S2365880")

print (top.snps[1:4,])
```

```
##                         marker      coord
## BICF2S2365880      INDEX SNP 38256927
## BICF2P462003    0.916279689260684 37817567
## BICF2P425207    0.866030555369713 38248275
## BICF2P612394    0.843123203444902 37952464
```

To extract markers within the user-defined (in bp) neighborhood of a given marker, one can use the `get.adjacent.markers` function.

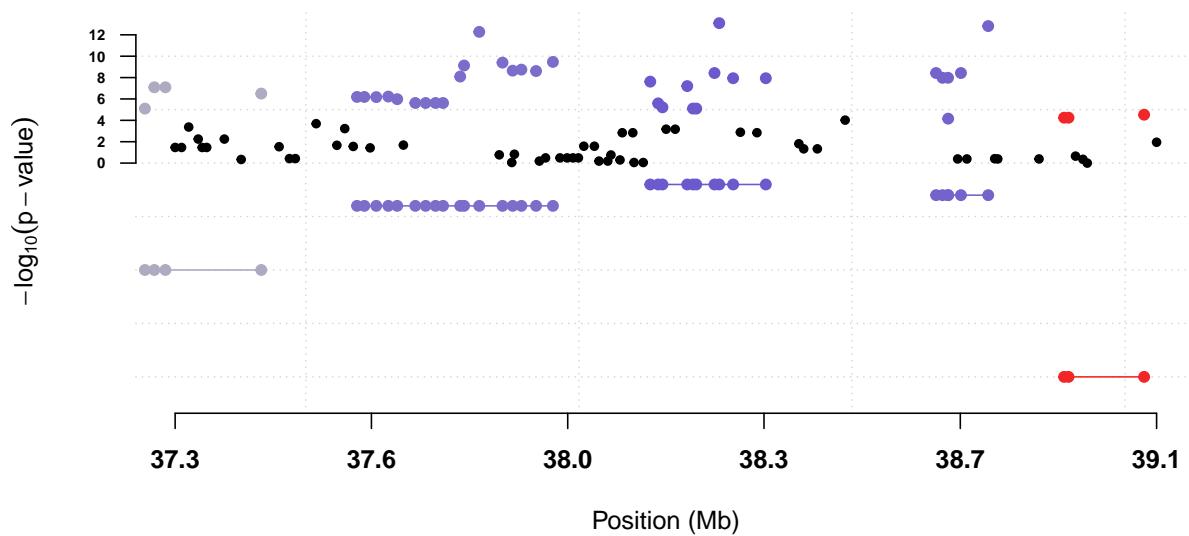
```
adjacent <- get.adjacent.markers(data = data.qc1, marker = "BICF2S2365880" , size.bp=1e4)

print(adjacent[1:4,])
```

```
##             BICF2P425207 BICF2S2365880
## dog224           1           1
## dog225           0           0
## dog226           1           1
## dog227           1           1
```

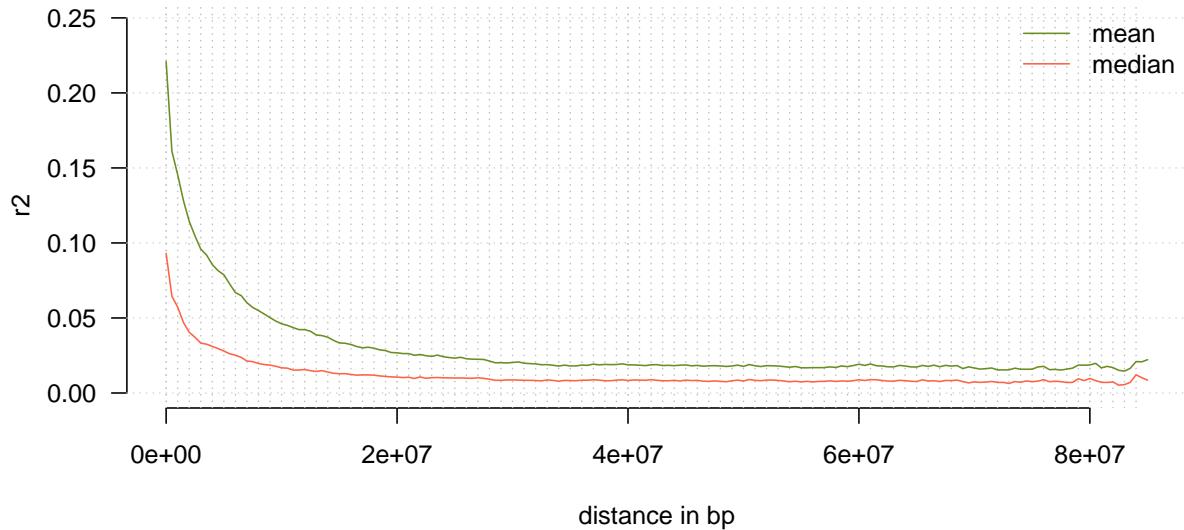
We can also use the clumping procedure as outlined in PLINK documentation to single out regions of interest. As we can see, there are two clumps represented by grey and red points respectively. The clumps are shown on both the standard Manhattan plot (upper panel) and, for improved clarity, also on a dedicated clump panel (lower panel). In short: a clump contains markers in high LD that are also significantly associated with the examined trait.

```
clumps <- clump.markers(data, mm, chr = 2)
plot.clumps(mm, clumps, 2, c(37256927,39256927))
```



To visualise LD decay on chromosome 2, one can call the `plot.ld.decay` function.

```
plot.LD.decay(data.qc1[,data.qc1@gtdata@chromosome==2])
```



Detecting runs of homozygosity

The `get.overlapping.windows` function divides the selected chromosome (or the whole genome) into overlapping chunks of given size and overlap. The function returns a list containing window coordinates along with a logical matrix where each window is represented by a row and the logical value per-marker is set to *true* if the marker is contained within the window and to *false* otherwise. One can specify (in bp) size of a window as well as overlap between windows.

We will divide chromosome 2 into overlapping windows and then use them to calculate mean heterozygosity and identify runs of homozygosity.

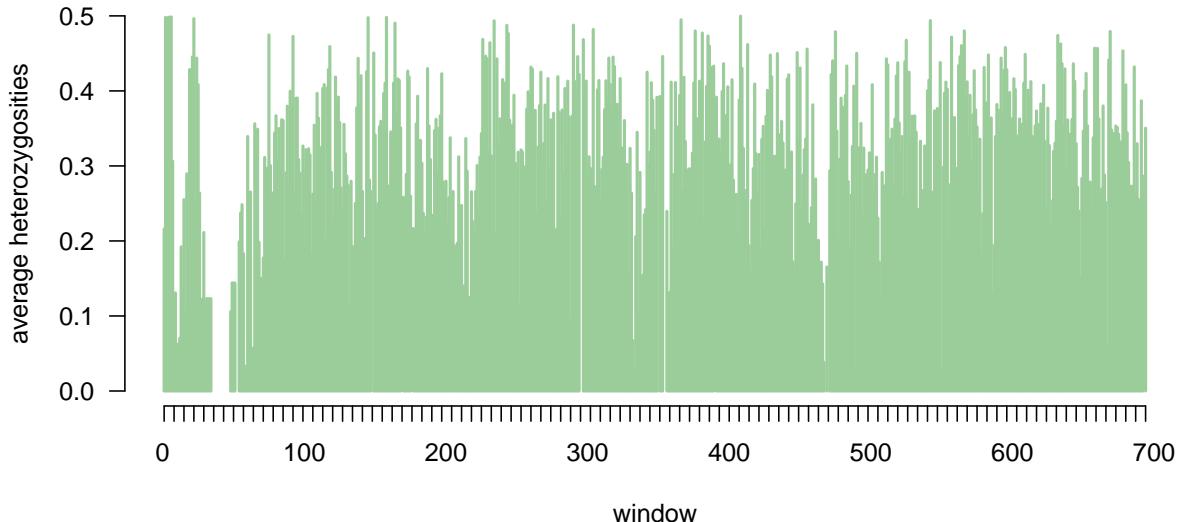
Computing average heterozygosity using overlapping windows approach.

In order to calculate the average heterozygosity in the set of overlapping windows, we can use the `het.overlap.windows` function. Heterozygosity is evaluated based on allelic frequencies of markers in particular overlapping windows and the basic Hardy-Weinberg theorem equation. The values range from 0 to 1 and correspond to the probability that the given set of loci, represented by the window, is heterozygous.

```
LW <- get.overlap.windows(data = data.qc1, chr = 2, size = 125e3, overlap = 2500)
het.windows <- het.overlap.wind(data.qc1, LW, F)
```

Now, having calculated average heterozygosity we can visualize them with `plot.overlap`

```
plot.overlap(LW,het.windows)
```



We can use calculated heterozygosity to examine runs of homozygosity across selected chromosome. Let's use `get.roh` and check if we have any stretches of reduced heterozygosity on chromosome 2. Below a given threshold, all windows will be treated as homozygous.

```
get.roh(data = data.qc1, chr=2, LW=LW, hetero.zyg=het.windows, threshold = 0.30, strict = TRUE)

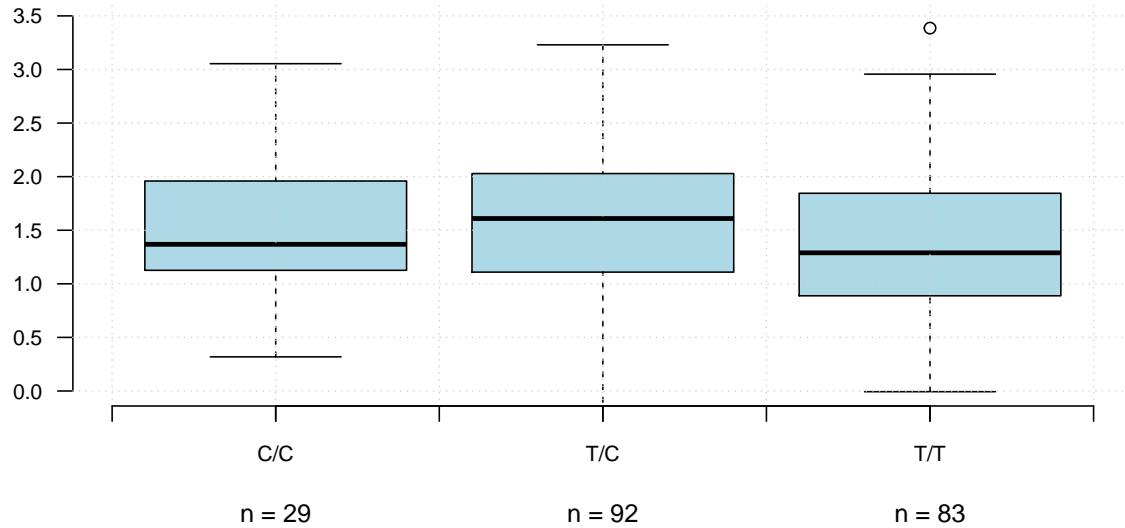
##      window      begin        end length
## [1,]     8 4010290 5360290     87
## [2,]    197 27162790 28022790     54
## [3,]    316 41740290 42477790     52
## [4,]    436 56440290 57667790     77
```

As a result we get a matrix with runs coordinates, length (in windows) and first window that starts a stretch.

Phenotype boxplots

For quantitative traits, the `boxplot.snp` function can be used to visually examine allele or genotype effect by plotting phenotype boxplots for the individuals in every genotypic class. The function works for both outbreed (three boxes) and inbred (two boxes) data.

```
marker <- 'BICF2S2365880'
trait.name <- 'response'
boxplot.snp(data = data.qc1, marker = marker, trait = trait.name, recode = F, font = 2)
```



Endogenous retroviral sequences (ERV)

The `get.erv` returns information about endogenous retroviral sequences (ERV) in an analysed region. At first, we need to obtain the list of ERV sequences from defined region of genome using `get.erv`. In the `cgmisc` package, we provide collection of canine ERVs identified in canFam3.1 assembly but you can also supply any other database.

Let's search for ERVs on chromosome 2:

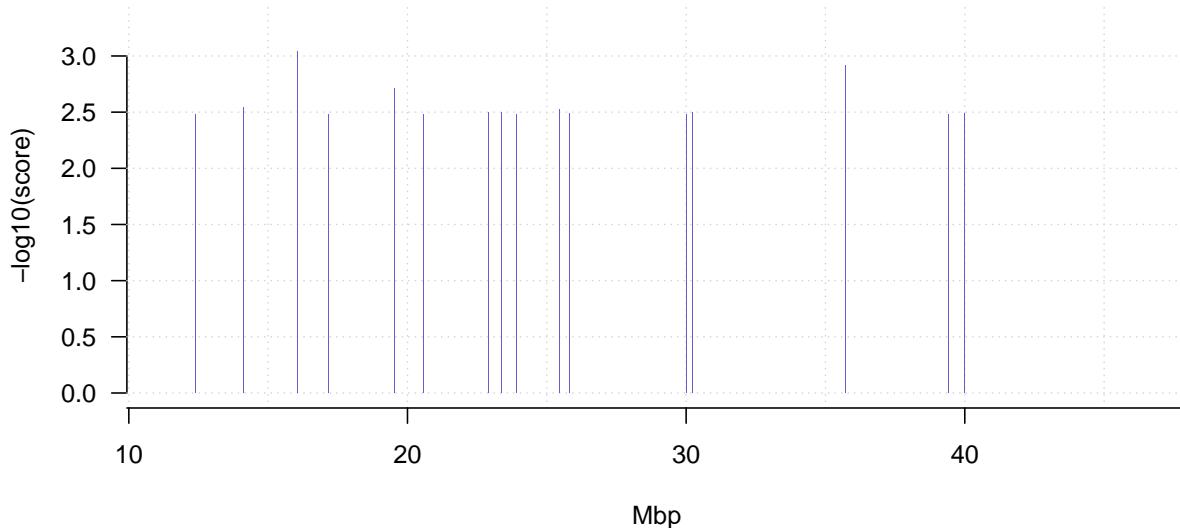
```
ervs <- get.erv(chr = "chr2", coords=c(10e6, 40e6))

print(ervs[1:4,])
```

```
##      id chromosome strand    start      end length score
## 1 2651       chr2      S 10015031 10006437   8594   309
## 2 2653       chr2      S 10128297 10123555   4742   847
## 3 2659       chr2      S 11537301 11525069  12232   319
## 4 2667       chr2      P 13840823 13856123  15300   324
##
##             subgenes
## 1 5LTR PBS MA CA NC Prot IN TM PPT 3LTR
## 2          5LTR PBS MA NC Prot RT 3LTR
## 3          5LTR PBS CA Prot IN TM PPT 3LTR
## 4          5LTR CA Prot RT IN PPT 3LTR
```

Having returned a list of ERV sequences, we are able to plot them with `plot.erv` function using the same region:

```
plot.erv(chr = "chr10", coords = c(10e6, 40e6))
```



Conversion functions

Our package provides a number of functions which enable switching between various data formats such as PHASE or FASTA. You can export `gwaa.data-class` used by `cgmisc` to i.e haplotype analysis software as well as convert external data format to the one used by `cgmisc`.

`gwaa.to.bigrr` The bigRR package implements a computationally-efficient generalized ridge regression (RR) algorithm for fitting models with a large number of parameters (Shen et al. 2013). `Gwaa.to.bigrr` function exports gwaa.data-class object to bigRR-compatible input structure.

`gwaa.to.vgwas` In a standard approach to GWAS, associations are detected based on differences in mean phenotypic values across genotype classes. Shen et al. (2012) have proposed an extended approach where the associations are detected based on differences in variances, not means only. They have implemented the approach in vGWAS [vGWAS] R package. `Gwaa.to.vgwas` converts gwaa.data-class object to a format readable by the vGWAS package.

`gwaa.to.phase` lets user convert a GenABEL gwaa.data-class object to the PHASE input format.

`phase.to.fasta` PHASE is a software for haplotype reconstruction `phase.to.fasta` function can be used to parse a part of PHASE output to a customised FASTA format:

```
'>haplo_1_count_176'
'TCGGGCTC'
```

In the above example, the first line contains the name of the haplotype along with its count, the second line provides haplotype sequence.

`phase.to.haplovview` Converts PHASE output format to Haplovview - a popular software which facilitates haplotype and LD analyses.

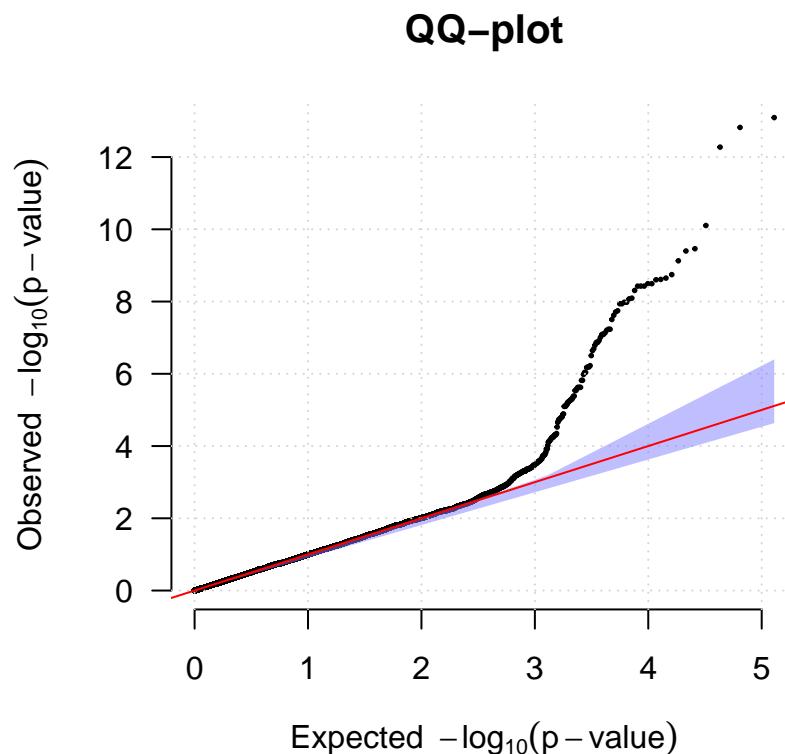
`gwaa.to.bed` produces a BED file containing p-values from predefined region of dataset that can be further used to visualize GWAS p-values as UCSC Genome Browser custom track.

Improved quantile-quantile plots

A quantile-quantile plot (QQ plots) is a graphical way of comparing two probability distributions. In GWAS studies, they are commonly used to compare computed per marker p-values with the ones expected under the null hypothesis. This comparison is then used to compute genomic-inflation factor λ which is a good measure of the degree of confounding caused by population structure. The **cgmisc** package contains two functions for better visualization and improved interpretation of standard QQ plots: `plot.qq` and `qq.emp`. The first function plots expected vs. observed distribution of p-values and shows theoretical confidence interval computed using approach outlined in Casella & Berger 2002). The `qq.emp` function, apart from showing theoretical confidence interval, also performs a number of randomization tests (shuffling the phenotypic value) to determine empirical confidence interval which, due to LD, is often narrower than the theoretical one.

//JJ NOTE: Permutation tests included in qq.emp?

```
plot.qq(mm[, 'P1df'], conf=c(0.05, 0.95), conf.type="shade", conf.col=rgb(.5,.5,1,.5), step=100)
```



Dog-specific utilities

Current approaches to finding genome-wide associations in diploid organisms quite often encounter difficulties when analysing non-autosomal parts of the genome, e.g. sex chromosomes [Young et al., 2008]. In **cgmisc**, the `chr.x.fix.canfam` partially addresses this issue by creating an artificial pseudo-autosome which consists of pseudo-autosomal regions of the X chromosome. Currently, the function is specific to data coming from domestic dog (*Canis familiaris*, assemblies canFam2 and canFam3.1).

```
chr.x.fix.canfam (data = data.qc1, assembly="canFam3")
```

Interacting with the UCSC genome browser

Sometimes, one may wish to display p-values from a genome-wide association scan directly in UCSC Genome Browser (Kent et al., 2002) to easily align annotations with the signals of interest. This can be done with `gwaa.to.bed` function that exports coordinates and p-values from `gwaa.data-scan` into bed file that can easily be used to set a custom path in the UCSC Genome Browser.

Using the `open.region.ucsc` function, it is possible to automatically open an Internet browser window containing UCSC Genome Browser result for a set of predefined genomic coordinates and assembly of interest.

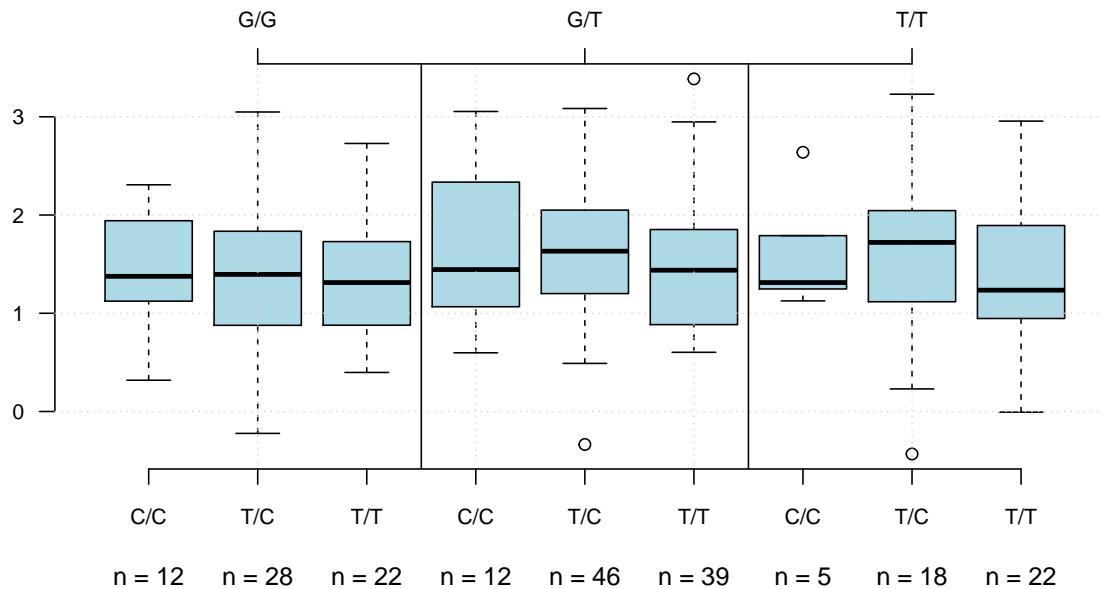
```
open.region.ucsc(chr = 2, coords = c(37256927,39256927), assembly = "canFam3")
```

Simple epistasis scan

To gain further understanding of the genetic architecture underlying our trait, we might want to search for potential epistasis between pairs of SNPs. In the function `epistasis.scan`, we implement a simple way of doing this by fitting linear models including two SNPs: $y = \beta_0 + \beta_1 * SNP + \beta_i * SNP_i + \beta_{i,int} * SNP * SNP_i + e$, where y is the phenotype, SNP and SNP_i are the genotypes at the two SNPs, and e is the residual. $i = 1...n$, where n is the number of SNPs in the input to the function. The function takes a `SNP`, phenotypes, and a `gwaa.data-class` object as input. It then fits linear models between `SNP` and all markers/SNPs in the `gwaa.data-class` object. To visualize a potential two-SNP interaction, one can use the function `boxplot.snp.twoWay`. This function draws a two locus genotype-phenotype map by plotting phenotype boxplots for the individuals in every genotypic class, as defined by the two SNPs jointly. This gives nine or four boxes, for outbred and inbred data respectively.

```
es <- epistasis.scan(data = data.qc1[,data.qc1@gtdata@chromosome==2], SNP = 'BICF2S2365880',
                      trait.name = 'response')
```

```
boxplot.snp.twoWay(data.qc1, marker1 = 'BICF2S2365880', marker2 = 'BICF2S2337600',
                     trait = 'response')
```



List of aliases

List of functions aliases:

plot.manhattan.ld: plot.manhattan.LD
plot.ld.decay: plot.LD.decay
phase.to.haplovview: phase2haplovview, PHASE.to.Haplovview, PHASE2Haplovview
phase.to.fasta: phase2fasta, PHASE.to.FASTA, PHASE2FASTA
open.region.ucsc: open.region.UCSC
gwaa.to.vgwas : gwaa2vgwas, gwaa.to.vGWAS, gwaa2vGWAS
gwaa.to.phase: gwaa2phase, gwaa.to.PHASE, gwaa2PHASE
gwaa.to.bigrr: gwaa2bigrr, gwaa.to.bigRR, gwaa2bigrr
gwaa.to.bed: gwaa2bed
get.ld.colors: get.LD.colors
plot.fstats: plot.Fst
compute.fstats: compute.Fstats
create.haplovview.info: create.Haplovview.info