Summarization and quantitative trait analysis of CNV ranges

The *CNVRanger* package implements a comprehensive tool suite for the analysis of copy number variation (CNV). This includes functionality for summarizing individual CNV calls across a population, assessing overlap with functional genomic regions, and association analysis with gene expression and quantitative phenotypes.

# Introduction

Copy number variation (CNV) is a frequently observed deviation from the diploid state due to duplication or deletion of genomic regions. CNVs can be experimentally detected based on comparative genomic hybridization, and computationally inferred from SNP-arrays or next-generation sequencing data. These technologies for CNV detection have in common that they report, for each sample under study, genomic regions that are duplicated or deleted with respect to a reference. Such regions are denoted as *CNV calls* in the following and will be considered the starting point for analysis with the [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) package. However, as the coordinates of *CNV calls* as well as their respective copy number are requested, any external software used to detect CNVs needs to be able to provide such an information.

The key parts of the functionality implemented in CNVRanger were developed, described, and applied in several previous studies:

* Genome-wide detection of CNVs and their association with meat tenderness in Nelore cattle [da Silva et al., 2016](https://doi.org/10.1371/journal.pone.0157711)
* Widespread modulation of gene expression by copy number variation in skeletal muscle [Geistlinger et al., 2018](https://doi.org/10.1038/s41598-018-19782-4)
* CNVs are associated with genomic architecture in a songbird [da Silva et al., 2018](https://doi.org/10.1186/s12864-018-4577-1)

The [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) package implements three frequently used approaches for summarizing CNV calls:

1. The [CNVRuler](http://www.ircgp.com/CNVRuler) procedure that trims region margins based on regional density [Kim et al., 2012](https://doi.org/10.1093/bioinformatics/bts239),
2. the reciprocal overlap (RO) procedure that requires calls to sufficiently overlap with one another [Conrad et al., 2010](https://doi.org/10.1038/nature08516), and
3. the [GISTIC](http://www.broadinstitute.org/cancer/cga/gistic) procedure that identifies recurrent CNV regions [Beroukhim et al., 2007](https://doi.org/10.1073/pnas.0710052104).

In addition, [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) provides functionality for the analysis of the overlap of CNVs with functional genomic regions such as genes, promoters, and enhancers. The package also implements RNA-seq expression Quantitative Trait Loci (eQTL) analysis for CNVs by interfacing with the [*edgeR*](https://bioconductor.org/packages/3.9/edgeR) package with convenient options for common analyses including restriction by genomic regions and cis-eQTLs. Similarly, [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) also interfaces with [PLINK](http://zzz.bwh.harvard.edu/plink), thereby enabling traditional genome-wide association studies (GWAS) between CNVs and quantitative phenotypes. It is important to note that the [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) package assumes its input dataset to contain CNV calls previously filtered by quality within the external software that was used for CNV calling, or by a custom implementation chosen by the user. [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) intends to provide downstream summarization and association analysis for CNV calls, but it currently do not implements any function to deal with CNV calling or filtering by quality. Therefore, the user needs to consider the false negative-positive rate present in a CNV dataset in order to correctly interpret the results. Moreover, in the current version, CNVRanger is not able to generate proper results for polyploid species.

# Reading and accessing CNV data

The [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) package uses Bioconductor core data structures implemented in the [*GenomicRanges*](https://bioconductor.org/packages/3.9/GenomicRanges) and [*RaggedExperiment*](https://bioconductor.org/packages/3.9/RaggedExperiment) packages to represent, access, and manipulate CNV data.

We start by loading the package.

library(CNVRanger)

## Input data format

For demonstration, we consider CNV calls as obtained with [PennCNV](http://penncnv.openbioinformatics.org) from SNP-chip data in a Brazilian cattle breed ([da Silva et al., 2016](https://doi.org/10.1371/journal.pone.0157711)).

Here, we use a data subset and only consider CNV calls on chromosome 1 and 2, for which there are roughly 3000 CNV calls as obtained for 711 samples.

data.dir <- system.file("extdata", package="CNVRanger")  
call.file <- file.path(data.dir, "Silva16\_PONE\_CNV\_calls.csv")  
calls <- read.csv(call.file, as.is=TRUE)  
nrow(calls)

## [1] 3000

head(calls)

## chr start end NE\_id state  
## 1 chr1 16947 45013 NE001423 3  
## 2 chr1 36337 67130 NE001426 3  
## 3 chr1 16947 36337 NE001428 3  
## 4 chr1 36337 105963 NE001519 3  
## 5 chr1 36337 83412 NE001534 3  
## 6 chr1 36337 83412 NE001648 3

length(unique(calls[,"NE\_id"]))

## [1] 711

## Representation as a GRangesList

We group the calls by sample ID, resulting in a GRangesList. Each element of the list corresponds to a sample, and contains the genomic coordinates of the CNV calls for this sample (along with the copy number state in the State metadata column).

grl <- makeGRangesListFromDataFrame(calls,   
 split.field="NE\_id", keep.extra.columns=TRUE)  
grl

## GRangesList object of length 711:  
## $NE001357   
## GRanges object with 5 ranges and 1 metadata column:  
## seqnames ranges strand | state  
## <Rle> <IRanges> <Rle> | <integer>  
## [1] chr1 4569526-4577608 \* | 3  
## [2] chr1 15984544-15996851 \* | 1  
## [3] chr1 38306432-38330161 \* | 3  
## [4] chr1 93730576-93819471 \* | 0  
## [5] chr2 40529044-40540747 \* | 3  
##   
## $NE001358   
## GRanges object with 1 range and 1 metadata column:  
## seqnames ranges strand | state  
## [1] chr1 105042452-105233446 \* | 1  
##   
## $NE001359   
## GRanges object with 4 ranges and 1 metadata column:  
## seqnames ranges strand | state  
## [1] chr1 4569526-4577608 \* | 3  
## [2] chr1 31686841-31695808 \* | 0  
## [3] chr1 93730576-93819471 \* | 0  
## [4] chr2 2527718-2535261 \* | 0  
##   
## ...  
## <708 more elements>  
## -------  
## seqinfo: 2 sequences from an unspecified genome; no seqlengths

The advantage of representing the CNV calls as a GRangesList is that it allows to leverage the comprehensive set of operations on genomic regions implemented in the [*GenomicRanges*](https://bioconductor.org/packages/3.9/GenomicRanges) packages - for instance, sorting of the calls according to their genomic coordinates.

grl <- sort(grl)  
grl

## GRangesList object of length 711:  
## $NE001357   
## GRanges object with 5 ranges and 1 metadata column:  
## seqnames ranges strand | state  
## <Rle> <IRanges> <Rle> | <integer>  
## [1] chr1 4569526-4577608 \* | 3  
## [2] chr1 15984544-15996851 \* | 1  
## [3] chr1 38306432-38330161 \* | 3  
## [4] chr1 93730576-93819471 \* | 0  
## [5] chr2 40529044-40540747 \* | 3  
##   
## $NE001358   
## GRanges object with 1 range and 1 metadata column:  
## seqnames ranges strand | state  
## [1] chr1 105042452-105233446 \* | 1  
##   
## $NE001359   
## GRanges object with 4 ranges and 1 metadata column:  
## seqnames ranges strand | state  
## [1] chr1 4569526-4577608 \* | 3  
## [2] chr1 31686841-31695808 \* | 0  
## [3] chr1 93730576-93819471 \* | 0  
## [4] chr2 2527718-2535261 \* | 0  
##   
## ...  
## <708 more elements>  
## -------  
## seqinfo: 2 sequences from an unspecified genome; no seqlengths

The input file containg CNV calls should necessarily have at least five columns, which will indicate for each call its (i) chromosome, (ii) downstream breakpoint start, (iii) upstream breakpoint, (iv) sample name and the (v) copy number state. Independently of the format of CNV calls (i.e. tab or space delimited), the R object containing these calls should be converted to a GRangesList, as described above, in order to be used as input in [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) functions.

## Representation as a RaggedExperiment

An alternative matrix-like representation of the CNV calls can be obtained with the [*RaggedExperiment*](https://bioconductor.org/packages/3.9/RaggedExperiment) data class. It resembles in many aspects the [*SummarizedExperiment*](https://bioconductor.org/packages/3.9/SummarizedExperiment) data class for storing gene expression data as e.g. obtained with RNA-seq.

ra <- RaggedExperiment(grl)  
ra

## class: RaggedExperiment   
## dim: 3000 711   
## assays(1): state  
## rownames: NULL  
## colnames(711): NE001357 NE001358 ... NE003967 NE003968  
## colData names(0):

As apparent from the dim slot of the object, it stores the CNV calls in the rows and the samples in the columns. Note that the CN state is now represented as an assay matrix which can be easily accessed and subsetted.

assay(ra[1:5,1:5])

## NE001357 NE001358 NE001359 NE001360 NE001361  
## chr1:4569526-4577608 3 NA NA NA NA  
## chr1:15984544-15996851 1 NA NA NA NA  
## chr1:38306432-38330161 3 NA NA NA NA  
## chr1:93730576-93819471 0 NA NA NA NA  
## chr2:40529044-40540747 3 NA NA NA NA

As with [*SummarizedExperiment*](https://bioconductor.org/packages/3.9/SummarizedExperiment) objects, additional information for the samples are annotated in the colData slot. For example, we annotate the steer weight and its feed conversion ratio (FCR) using simulated data. Feed conversion ratio is the ratio of dry matter intake to live-weight gain. A typical range of feed conversion ratios is 4.5-7.5 with a lower number being more desirable as it would indicate that a steer required less feed per pound of gain.

weight <- rnorm(ncol(ra), mean=1100, sd=100)  
fcr <- rnorm(ncol(ra), mean=6, sd=1.5)  
colData(ra)$weight <- round(weight, digits=2)  
colData(ra)$fcr <- round(fcr, digits=2)  
colData(ra)

## DataFrame with 711 rows and 2 columns  
## weight fcr  
## <numeric> <numeric>  
## NE001357 1328.7 4.38  
## NE001358 1092 7.92  
## NE001359 978.48 5.24  
## NE001360 1126.47 7.93  
## NE001361 1041.18 4.41  
## ... ... ...  
## NE003962 1002.91 1.12  
## NE003963 997.46 6.76  
## NE003966 1120.24 4.63  
## NE003967 1057.42 3.62  
## NE003968 1135.58 7.46

# Summarizing individual CNV calls across a population

In CNV analysis, it is often of interest to summarize individual calls across the population, (i.e. to define CNV regions), for subsequent association analysis with expression and phenotype data. In the simplest case, this just merges overlapping individual calls into summarized regions. However, this typically inflates CNV region size, and more appropriate approaches have been developed for this purpose.

## Trimming low-density areas

Here, we use the approach from [CNVRuler](http://www.ircgp.com/CNVRuler) to summarize CNV calls to CNV regions (see [Figure 1](https://academic.oup.com/view-large/figure/83392426/bts239f1.jpeg) in [Kim et al., 2012](https://doi.org/10.1093/bioinformatics/bts239) for an illustration of the approach). This trims low-density areas as defined by the density argument, which is set here to <10% of the number of calls within a summarized region.

cnvrs <- populationRanges(grl, density=0.1)  
cnvrs

## GRanges object with 303 ranges and 2 metadata columns:  
## seqnames ranges strand | freq type  
## <Rle> <IRanges> <Rle> | <numeric> <character>  
## [1] chr1 16947-111645 \* | 103 gain  
## [2] chr1 1419261-1630187 \* | 18 gain  
## [3] chr1 1896112-2004603 \* | 218 gain  
## [4] chr1 4139727-4203274 \* | 1 gain  
## [5] chr1 4554832-4577608 \* | 23 gain  
## ... ... ... ... . ... ...  
## [299] chr2 136310067-136322489 \* | 2 loss  
## [300] chr2 136375337-136386940 \* | 1 loss  
## [301] chr2 136455546-136466040 \* | 1 loss  
## [302] chr2 136749793-136802493 \* | 39 both  
## [303] chr2 139194749-139665914 \* | 58 both  
## -------  
## seqinfo: 2 sequences from an unspecified genome; no seqlengths

Note that CNV frequency (number of samples overlapping each region) and CNV type (gain, loss, or both) have also been annotated in the columns freq and type, respectively.

## Reciprocal overlap

We also provide an implementation of the *Reciprocal Overlap (RO)* procedure that requires calls to sufficiently overlap with one another as e.g. used by [Conrad et al., 2010](https://doi.org/10.1038/nature08516). This merges calls with an RO above a threshold as given by the ro.thresh argument. For example, an RO of 0.51 between two genomic regions *A* and *B* requires that *B* overlaps at least 51% of *A*, *and* that *A* also overlaps at least 51% of *B*.

ro.cnvrs <- populationRanges(grl[1:100], mode="RO", ro.thresh=0.51)  
ro.cnvrs

## GRanges object with 85 ranges and 2 metadata columns:  
## seqnames ranges strand | freq type  
## <Rle> <IRanges> <Rle> | <numeric> <character>  
## [1] chr1 16947-45013 \* | 6 gain  
## [2] chr1 36337-67130 \* | 6 gain  
## [3] chr1 36337-105963 \* | 6 gain  
## [4] chr1 1419261-1506862 \* | 3 gain  
## [5] chr1 1539361-1625471 \* | 3 gain  
## ... ... ... ... . ... ...  
## [81] chr2 136215094-136232653 \* | 2 loss  
## [82] chr2 136749793-136776410 \* | 1 gain  
## [83] chr2 138738929-139004086 \* | 1 loss  
## [84] chr2 139194749-139274355 \* | 1 gain  
## [85] chr2 139324752-139665914 \* | 3 loss  
## -------  
## seqinfo: 2 sequences from an unspecified genome; no seqlengths

## Identifying recurrent regions

In particular in cancer, it is important to distinguish driver from passenger mutations, i.e. to distinguish meaningful events from random background aberrations. The [GISTIC](http://www.broadinstitute.org/cancer/cga/gistic) method identifies those regions of the genome that are aberrant more often than would be expected by chance, with greater weight given to high amplitude events (high-level copy-number gains or homozygous deletions) that are less likely to represent random aberrations ([Beroukhim et al., 2007](https://doi.org/10.1073/pnas.0710052104)).

By setting est.recur=TRUE, we deploy a GISTIC-like significance estimation

cnvrs <- populationRanges(grl, density=0.1, est.recur=TRUE)  
cnvrs

## GRanges object with 303 ranges and 3 metadata columns:  
## seqnames ranges strand | freq type  
## <Rle> <IRanges> <Rle> | <numeric> <character>  
## [1] chr1 16947-111645 \* | 103 gain  
## [2] chr1 1419261-1630187 \* | 18 gain  
## [3] chr1 1896112-2004603 \* | 218 gain  
## [4] chr1 4139727-4203274 \* | 1 gain  
## [5] chr1 4554832-4577608 \* | 23 gain  
## ... ... ... ... . ... ...  
## [299] chr2 136310067-136322489 \* | 2 loss  
## [300] chr2 136375337-136386940 \* | 1 loss  
## [301] chr2 136455546-136466040 \* | 1 loss  
## [302] chr2 136749793-136802493 \* | 39 both  
## [303] chr2 139194749-139665914 \* | 58 both  
## pvalue  
## <numeric>  
## [1] 0.00980392156862742  
## [2] 0.107843137254902  
## [3] 0  
## [4] 0.558823529411765  
## [5] 0.0882352941176471  
## ... ...  
## [299] 0.236111111111111  
## [300] 0.421296296296296  
## [301] 0.421296296296296  
## [302] 0.0588235294117647  
## [303] 0.0392156862745098  
## -------  
## seqinfo: 2 sequences from an unspecified genome; no seqlengths

and filter for recurrent CNVs that exceed a significance threshold of 0.05.

cnvrs[cnvrs$pvalue < 0.05]

## GRanges object with 17 ranges and 3 metadata columns:  
## seqnames ranges strand | freq type  
## <Rle> <IRanges> <Rle> | <numeric> <character>  
## [1] chr1 16947-111645 \* | 103 gain  
## [2] chr1 1896112-2004603 \* | 218 gain  
## [3] chr1 15984544-15996851 \* | 116 loss  
## [4] chr1 31686841-31695808 \* | 274 loss  
## [5] chr1 69205418-69219486 \* | 46 loss  
## ... ... ... ... . ... ...  
## [13] chr2 97882695-97896935 \* | 80 loss  
## [14] chr2 124330343-124398570 \* | 39 loss  
## [15] chr2 135096060-135271140 \* | 84 gain  
## [16] chr2 135290754-135553033 \* | 83 gain  
## [17] chr2 139194749-139665914 \* | 58 both  
## pvalue  
## <numeric>  
## [1] 0.00980392156862742  
## [2] 0  
## [3] 0.0185185185185185  
## [4] 0.00462962962962965  
## [5] 0.0416666666666666  
## ... ...  
## [13] 0.0231481481481481  
## [14] 0.0462962962962963  
## [15] 0.0196078431372549  
## [16] 0.0294117647058824  
## [17] 0.0392156862745098  
## -------  
## seqinfo: 2 sequences from an unspecified genome; no seqlengths

# Overlap analysis of CNVs with functional genomic regions

Once individual CNV calls have been summarized across the population, it is typically of interest whether the resulting CNV regions overlap with functional genomic regions such as genes, promoters, or enhancers. As a certain amount of overlap can be expected just by chance, an assessment of statistical significance is needed to decide whether the observed overlap is greater (enrichment) or less (depletion) than expected by chance.

The [*regioneR*](https://bioconductor.org/packages/3.9/regioneR) package implements a general framework for testing overlaps of genomic regions based on permutation sampling. This allows to repeatedly sample random regions from the genome, matching size and chromosomal distribution of the region set under study (here: the CNV regions). By recomputing the overlap with the functional features in each permutation, statistical significance of the observed overlap can be assessed.

We demonstrate in the following how this strategy can be used to assess the overlap between the detected CNV regions and protein-coding regions in the cattle genome. We expect to find a depletion as protein-coding regions are highly conserved and rarely subject to long-range structural variation such as CNV. Hence, is the overlap between CNVs and protein-coding genes less than expected by chance?

To obtain the location of protein-coding genes, we query available *Bos taurus* annotation from Ensembl

library(AnnotationHub)  
ah <- AnnotationHub()

## snapshotDate(): 2019-05-02

ahDb <- query(ah, pattern = c("Bos taurus", "EnsDb"))  
ahDb

## AnnotationHub with 10 records  
## # snapshotDate(): 2019-05-02   
## # $dataprovider: Ensembl  
## # $species: Bos taurus  
## # $rdataclass: EnsDb  
## # additional mcols(): taxonomyid, genome, description,  
## # coordinate\_1\_based, maintainer, rdatadateadded, preparerclass,  
## # tags, rdatapath, sourceurl, sourcetype   
## # retrieve records with, e.g., 'object[["AH53189"]]'   
##   
## title   
## AH53189 | Ensembl 87 EnsDb for Bos Taurus  
## AH53693 | Ensembl 88 EnsDb for Bos Taurus  
## AH56658 | Ensembl 89 EnsDb for Bos Taurus  
## AH57731 | Ensembl 90 EnsDb for Bos Taurus  
## AH60745 | Ensembl 91 EnsDb for Bos Taurus  
## AH60948 | Ensembl 92 EnsDb for Bos Taurus  
## AH64416 | Ensembl 93 EnsDb for Bos Taurus  
## AH64886 | Ensembl 94 EnsDb for Bos taurus  
## AH67912 | Ensembl 95 EnsDb for Bos taurus  
## AH69141 | Ensembl 96 EnsDb for Bos taurus

and retrieve gene coordinates in the UMD3.1 assembly (Ensembl 92).

ahEdb <- ahDb[["AH60948"]]

## downloading 0 resources

## loading from cache   
## 'AH60948 : 67694'

bt.genes <- genes(ahEdb)  
seqlevels(bt.genes) <- paste0("chr", seqlevels(bt.genes))  
bt.genes

## GRanges object with 24616 ranges and 8 metadata columns:  
## seqnames ranges strand |  
## <Rle> <IRanges> <Rle> |  
## ENSBTAG00000046619 chr1 19774-19899 - |  
## ENSBTAG00000006858 chr1 34627-35558 + |  
## ENSBTAG00000039257 chr1 69695-71121 - |  
## ENSBTAG00000035349 chr1 83323-84281 - |  
## ENSBTAG00000001753 chr1 124849-179713 - |  
## ... ... ... ... .  
## ENSBTAG00000025951 chrX 148526584-148535857 + |  
## ENSBTAG00000029592 chrX 148538917-148539037 - |  
## ENSBTAG00000016989 chrX 148576705-148582356 - |  
## ENSBTAG00000025952 chrX 148774930-148780901 - |  
## ENSBTAG00000047839 chrX 148804071-148805135 + |  
## gene\_id gene\_name gene\_biotype  
## <character> <character> <character>  
## ENSBTAG00000046619 ENSBTAG00000046619 RF00001 rRNA  
## ENSBTAG00000006858 ENSBTAG00000006858 pseudogene  
## ENSBTAG00000039257 ENSBTAG00000039257 protein\_coding  
## ENSBTAG00000035349 ENSBTAG00000035349 pseudogene  
## ENSBTAG00000001753 ENSBTAG00000001753 protein\_coding  
## ... ... ... ...  
## ENSBTAG00000025951 ENSBTAG00000025951 protein\_coding  
## ENSBTAG00000029592 ENSBTAG00000029592 RF00001 rRNA  
## ENSBTAG00000016989 ENSBTAG00000016989 protein\_coding  
## ENSBTAG00000025952 ENSBTAG00000025952 protein\_coding  
## ENSBTAG00000047839 ENSBTAG00000047839 P2RY8 protein\_coding  
## seq\_coord\_system  
## <character>  
## ENSBTAG00000046619 chromosome  
## ENSBTAG00000006858 chromosome  
## ENSBTAG00000039257 chromosome  
## ENSBTAG00000035349 chromosome  
## ENSBTAG00000001753 chromosome  
## ... ...  
## ENSBTAG00000025951 chromosome  
## ENSBTAG00000029592 chromosome  
## ENSBTAG00000016989 chromosome  
## ENSBTAG00000025952 chromosome  
## ENSBTAG00000047839 chromosome  
## description  
## <character>  
## ENSBTAG00000046619 NULL  
## ENSBTAG00000006858 NULL  
## ENSBTAG00000039257 NULL  
## ENSBTAG00000035349 NULL  
## ENSBTAG00000001753 NULL  
## ... ...  
## ENSBTAG00000025951 NULL  
## ENSBTAG00000029592 NULL  
## ENSBTAG00000016989 NULL  
## ENSBTAG00000025952 NULL  
## ENSBTAG00000047839 P2Y receptor family member 8 [Source:VGNC Symbol;Acc:VGNC:32531]  
## gene\_id\_version symbol entrezid  
## <character> <character> <list>  
## ENSBTAG00000046619 ENSBTAG00000046619.1 RF00001 NA  
## ENSBTAG00000006858 ENSBTAG00000006858.5 NA  
## ENSBTAG00000039257 ENSBTAG00000039257.2 NA  
## ENSBTAG00000035349 ENSBTAG00000035349.3 NA  
## ENSBTAG00000001753 ENSBTAG00000001753.4 507243  
## ... ... ... ...  
## ENSBTAG00000025951 ENSBTAG00000025951.4 NA  
## ENSBTAG00000029592 ENSBTAG00000029592.1 RF00001 NA  
## ENSBTAG00000016989 ENSBTAG00000016989.5 NA  
## ENSBTAG00000025952 ENSBTAG00000025952.3 785083  
## ENSBTAG00000047839 ENSBTAG00000047839.1 P2RY8 100299937  
## -------  
## seqinfo: 48 sequences from UMD3.1 genome

To speed up the example, we restrict analysis to chromosomes 1 and 2.

sel.genes <- bt.genes[seqnames(bt.genes) %in% c("chr1", "chr2")]  
sel.genes <- sel.genes[sel.genes$gene\_biotype == "protein\_coding"]  
sel.cnvrs <- cnvrs[seqnames(cnvrs) %in% c("chr1", "chr2")]

Now, we are applying an overlap permutation test with 100 permutations (ntimes=100), while maintaining chromosomal distribution of the CNV region set (per.chromosome=TRUE). Furthermore, we use the option count.once=TRUE to count an overlapping CNV region only once, even if it overlaps with 2 or more genes. We also allow random regions to be sampled from the entire genome (mask=NA), although in certain scenarios masking certain regions such as telomeres and centromeres is advisable. Also note that we use 100 permutations for demonstration only. To draw robust conclusions a minimum of 1000 permutations should be carried out.

library(regioneR)  
library(BSgenome.Btaurus.UCSC.bosTau6.masked)  
res <- suppressWarnings(overlapPermTest(A=sel.cnvrs, B=sel.genes, ntimes=100,   
 genome="bosTau6", mask=NA, per.chromosome=TRUE, count.once=TRUE))  
res

## $numOverlaps  
## P-value: 0.0099009900990099  
## Z-score: -2.4097  
## Number of iterations: 100  
## Alternative: less  
## Evaluation of the original region set: 97  
## Evaluation function: numOverlaps  
## Randomization function: randomizeRegions  
##   
## attr(,"class")  
## [1] "permTestResultsList"

summary(res[[1]]$permuted)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 100.0 108.8 113.0 114.3 118.2 133.0

The resulting permutation *p*-value indicates a significant depletion. Out of the 303 CNV regions, 97 overlap with at least one gene. In contrast, when repeatedly drawing random regions matching the CNV regions in size and chromosomal distribution, the mean number of overlapping regions across permutations was 114.3 7.2.

This finding is consistent with our observations across the whole genome ([da Silva et al., 2016](https://doi.org/10.1371/journal.pone.0157711)) and findings from the 1000 Genomes Poject ([Sudmant et al., 2015](https://www.nature.com/articles/nature15394)).

Note: the function regioneR::permTest allows to incorporate user-defined functions for randomizing regions and evaluating additional measures of overlap such as total genomic size in bp.

# CNV-expression association analysis

Studies of expression quantitative trait loci (eQTLs) aim at the discovery of genetic variants that explain variation in gene expression levels ([Nica and Dermitzakis, 2013](https://www.ncbi.nlm.nih.gov/pubmed/23650636)). Mainly applied in the context of SNPs, the concept also naturally extends to the analysis of CNVs.

The [*CNVRanger*](https://bioconductor.org/packages/3.9/CNVRanger) package implements association testing between CNV regions and RNA-seq read counts using [*edgeR*](https://bioconductor.org/packages/3.9/edgeR), which applies generalized linear models based on the negative-binomial distribution while incorporating normalization factors for different library sizes.

In the case of only one CN state deviating from 2n for a CNV region under investigation, this reduces to the classical 2-group comparison. For more than two states (e.g. 0n, 1n, 2n), edgeR’s ANOVA-like test is applied to test all deviating groups for significant expression differences relative to 2n.

## Dealing with individual CNV and RNA-seq assays

We demonstrate the functionality by loading RNA-seq read count data from skeletal muscle samples for 183 Nelore cattle steers, which we analyzed for CNV-expression effects as previously described ([Geistlinger et al., 2018](https://doi.org/10.1038/s41598-018-19782-4)).

rseq.file <- file.path(data.dir, "counts\_cleaned.txt")  
rcounts <- read.delim(rseq.file, row.names=1, stringsAsFactors=FALSE)  
rcounts <- as.matrix(rcounts)  
dim(rcounts)

## [1] 939 183

rcounts[1:5, 1:5]

## NE001407 NE001408 NE001424 NE001439 NE001448  
## ENSBTAG00000000088 64 65 233 135 134  
## ENSBTAG00000000160 20 28 50 13 18  
## ENSBTAG00000000176 279 373 679 223 417  
## ENSBTAG00000000201 252 271 544 155 334  
## ENSBTAG00000000210 268 379 486 172 443

For demonstration, we restrict analysis to the 939 genes on chromosome 1 and 2, and store the RNA-seq expression data in a [*SummarizedExperiment*](https://bioconductor.org/packages/3.9/SummarizedExperiment).

library(SummarizedExperiment)  
rse <- SummarizedExperiment(assays=list(rcounts=rcounts),   
 rowRanges=granges(sel.genes)[rownames(rcounts)])  
rse

## class: RangedSummarizedExperiment   
## dim: 939 183   
## metadata(0):  
## assays(1): rcounts  
## rownames(939): ENSBTAG00000000088 ENSBTAG00000000160 ...  
## ENSBTAG00000048210 ENSBTAG00000048228  
## rowData names(0):  
## colnames(183): NE001407 NE001408 ... NE003840 NE003843  
## colData names(0):

Assuming distinct modes of action, effects observed in the CNV-expression analysis are typically divided into (i) local effects (*cis*), where expression changes coincide with CNVs in the respective genes, and (ii) distal effects (*trans*), where CNVs supposedly affect trans-acting regulators such as transcription factors.

However, due to power considerations and to avoid detection of spurious effects, stringent filtering of (i) not sufficiently expressed genes, and (ii) CNV regions with insufficient sample size in groups deviating from 2n, should be carried out when testing for distal effects. Local effects have a clear spatial indication and the number of genes locating in or close to a CNV region of interest is typically small; testing for differential expression between CN states is thus generally better powered for local effects and less stringent filter criteria can be applied.

In the following, we carry out CNV-expression association analysis by providing the CNV regions to test (cnvrs), the individual CNV calls (grl) to determine per-sample CN state in each CNV region, the RNA-seq read counts (rse), and the size of the genomic window around each CNV region (window). The window argument thereby determines which genes are considered for testing for each CNV region and is set here to 1 Mbp.

Further, use the min.cpm and min.samples arguments to exclude from the analysis (i) genes with fewer than min.cpm reads per million reads mapped (cpm, counts per million), and (ii) CNV regions with fewer than min.samples samples in a group deviating from 2n.

res <- cnvEQTL(cnvrs, grl, rse, window="1Mbp", verbose=TRUE)

## Restricting analysis to 179 intersecting samples

## Preprocessing RNA-seq data ...

## Summarizing per-sample CN state in each CNV region

## Excluding 286 cnvrs not satisfying min.samples threshold

## Analyzing 12 regions with >=1 gene in the given window

## 1 of 12

## 2 of 12

## 3 of 12

## 4 of 12

## 5 of 12

## 6 of 12

## 7 of 12

## 8 of 12

## 9 of 12

## 10 of 12

## 11 of 12

## 12 of 12

res

## DataFrame with 89 rows and 5 columns  
## CNVR Gene logFC  
## <character> <character> <numeric>  
## 1 chr1:16947-111645 ENSBTAG00000018278 0.194859708204693  
## 2 chr1:16947-111645 ENSBTAG00000021997 -0.0812496004002579  
## 3 chr1:16947-111645 ENSBTAG00000020035 0.0745028563584966  
## 4 chr1:16947-111645 ENSBTAG00000011528 0.0118392582964246  
## 5 chr1:16947-111645 ENSBTAG00000012594 -0.00661188778069866  
## ... ... ... ...  
## 85 chr2:135290754-135553033 ENSBTAG00000030340 -0.0310183382350032  
## 86 chr2:135290754-135553033 ENSBTAG00000008314 -0.0148596021181995  
## 87 chr2:135290754-135553033 ENSBTAG00000008309 0.0172997593964661  
## 88 chr2:135290754-135553033 ENSBTAG00000003403 -0.00336281264078722  
## 89 chr2:135290754-135553033 ENSBTAG00000013282 -0.000223673376098313  
## PValue AdjPValue  
## <numeric> <numeric>  
## 1 0.0073151463593773 0.400665393142863  
## 2 0.179616021014383 0.84156465759887  
## 3 0.740325113668282 0.997718543468309  
## 4 0.915565410373996 0.997718543468309  
## 5 0.943057146391022 0.997718543468309  
## ... ... ...  
## 85 0.850536851523271 0.997718543468309  
## 86 0.869368789972092 0.997718543468309  
## 87 0.88551436276196 0.997718543468309  
## 88 0.987950415161084 0.997718543468309  
## 89 0.997718543468309 0.997718543468309

The resulting DataFrame contains in the first column the CNV regions tested. The second column contains the genes tested in the genomic window around each CNV region, and the other columns report (i) log2 fold change with respect to the 2n group, (ii) edgeR’s DE *p*-value, and (iii) the (per default) Benjamini-Hochberg adjusted *p*-value.

## Dealing with TCGA data stored in a MultiAssayExperiment

In the previous section, we individually prepared the CNV and RNA-seq data for CNV-expression association analysis. In the following, we demonstrate how to perform an integrated preparation of the two assays when stored in a [*MultiAssayExperiment*](https://bioconductor.org/packages/3.9/MultiAssayExperiment). We therefore consider glioblastoma [GBM](https://cancergenome.nih.gov/cancersselected/glioblastomamultiforme) data from The Cancer Genome Atlas [TCGA](https://cancergenome.nih.gov), which can conveniently be accessed with the [*curatedTCGAData*](https://bioconductor.org/packages/3.9/curatedTCGAData) package.

library(curatedTCGAData)  
suppressMessages(  
 gbm <- curatedTCGAData("GBM",   
 assays=c("GISTIC\_Peaks", "CNVSNP", "RNASeq2GeneNorm"),   
 dry.run=FALSE)  
)

gbm

## A MultiAssayExperiment object of 3 listed  
## experiments with user-defined names and respective classes.   
## Containing an ExperimentList class object of length 3:   
## [1] GBM\_CNVSNP-20160128: RaggedExperiment with 146852 rows and 1104 columns   
## [2] GBM\_RNASeq2GeneNorm-20160128: SummarizedExperiment with 20501 rows and 166 columns   
## [3] GBM\_GISTIC\_Peaks-20160128: RangedSummarizedExperiment with 68 rows and 577 columns   
## Features:   
## experiments() - obtain the ExperimentList instance   
## colData() - the primary/phenotype DataFrame   
## sampleMap() - the sample availability DataFrame   
## `$`, `[`, `[[` - extract colData columns, subset, or experiment   
## \*Format() - convert into a long or wide DataFrame   
## assays() - convert ExperimentList to a SimpleList of matrices

The returned MultiAssayExperiment contains three assays:

* the SNP-based CNV calls stored in a RaggedExperiment (GBM\_CNVSNP),
* the recurrent CNV regions summarized across the population using the [GISTIC](http://www.broadinstitute.org/cancer/cga/gistic) method (GBM\_GISTIC\_Peaks), and
* the normalized RNA-seq gene expression values in a SummarizedExperiment (GBM\_RNASeq2GeneNorm).

To annotate the genomic coordinates of the genes measured in the RNA-seq assay, we use the function symbolsToRanges from the [*TCGAutils*](https://bioconductor.org/packages/3.9/TCGAutils) package. For demonstration, we restrict the analysis to chromosome 1 and 2.

library(TCGAutils)  
gbm <- suppressMessages(symbolsToRanges(gbm, unmapped=FALSE))  
for(i in 1:3)   
{  
 genome(rowRanges(gbm[[i]])) <- "hg19"  
 seqlevelsStyle(rowRanges(gbm[[i]])) <- "UCSC"  
 ind <- as.character(seqnames(rowRanges(gbm[[i]]))) %in% c("chr1", "chr2")  
 gbm[[i]] <- gbm[[i]][ind,]  
}  
gbm

## A MultiAssayExperiment object of 3 listed  
## experiments with user-defined names and respective classes.   
## Containing an ExperimentList class object of length 3:   
## [1] GBM\_CNVSNP-20160128: RaggedExperiment with 17818 rows and 1104 columns   
## [2] GBM\_GISTIC\_Peaks-20160128: RangedSummarizedExperiment with 12 rows and 577 columns   
## [3] GBM\_RNASeq2GeneNorm-20160128\_ranged: RangedSummarizedExperiment with 2939 rows and 166 columns   
## Features:   
## experiments() - obtain the ExperimentList instance   
## colData() - the primary/phenotype DataFrame   
## sampleMap() - the sample availability DataFrame   
## `$`, `[`, `[[` - extract colData columns, subset, or experiment   
## \*Format() - convert into a long or wide DataFrame   
## assays() - convert ExperimentList to a SimpleList of matrices

We now restrict the analysis to intersecting patients of the three assays using MultiAssayExperiment’s intersectColumns function, and select *Primary Solid Tumor* samples using the splitAssays function from the [*TCGAutils*](https://bioconductor.org/packages/3.9/TCGAutils) package.

gbm <- intersectColumns(gbm)  
sampleTables(gbm)

## $`GBM\_CNVSNP-20160128`  
##   
## 01 02 10 11   
## 154 13 146 1   
##   
## $`GBM\_GISTIC\_Peaks-20160128`  
##   
## 01   
## 154   
##   
## $`GBM\_RNASeq2GeneNorm-20160128\_ranged`  
##   
## 01 02   
## 147 13

data(sampleTypes)  
sampleTypes

## Code Definition  
## 1 01 Primary Solid Tumor  
## 2 02 Recurrent Solid Tumor  
## 3 03 Primary Blood Derived Cancer - Peripheral Blood  
## 4 04 Recurrent Blood Derived Cancer - Bone Marrow  
## 5 05 Additional - New Primary  
## 6 06 Metastatic  
## 7 07 Additional Metastatic  
## 8 08 Human Tumor Original Cells  
## 9 09 Primary Blood Derived Cancer - Bone Marrow  
## 10 10 Blood Derived Normal  
## 11 11 Solid Tissue Normal  
## 12 12 Buccal Cell Normal  
## 13 13 EBV Immortalized Normal  
## 14 14 Bone Marrow Normal  
## 15 15 sample type 15  
## 16 16 sample type 16  
## 17 20 Control Analyte  
## 18 40 Recurrent Blood Derived Cancer - Peripheral Blood  
## 19 50 Cell Lines  
## 20 60 Primary Xenograft Tissue  
## 21 61 Cell Line Derived Xenograft Tissue  
## 22 99 sample type 99  
## Short.Letter.Code  
## 1 TP  
## 2 TR  
## 3 TB  
## 4 TRBM  
## 5 TAP  
## 6 TM  
## 7 TAM  
## 8 THOC  
## 9 TBM  
## 10 NB  
## 11 NT  
## 12 NBC  
## 13 NEBV  
## 14 NBM  
## 15 15SH  
## 16 16SH  
## 17 CELLC  
## 18 TRB  
## 19 CELL  
## 20 XP  
## 21 XCL  
## 22 99SH

gbm <- splitAssays(gbm, sampleCodes="01")  
gbm

## A MultiAssayExperiment object of 3 listed  
## experiments with user-defined names and respective classes.   
## Containing an ExperimentList class object of length 3:   
## [1] 01\_GBM\_CNVSNP-20160128: RaggedExperiment with 17818 rows and 154 columns   
## [2] 01\_GBM\_GISTIC\_Peaks-20160128: RangedSummarizedExperiment with 12 rows and 154 columns   
## [3] 01\_GBM\_RNASeq2GeneNorm-20160128\_ranged: RangedSummarizedExperiment with 2939 rows and 147 columns   
## Features:   
## experiments() - obtain the ExperimentList instance   
## colData() - the primary/phenotype DataFrame   
## sampleMap() - the sample availability DataFrame   
## `$`, `[`, `[[` - extract colData columns, subset, or experiment   
## \*Format() - convert into a long or wide DataFrame   
## assays() - convert ExperimentList to a SimpleList of matrices

We transform CN estimates into CN states where states are encoded as:

* 0: homozygous deletion (2-copy loss)
* 1: heterozygous deletion (1-copy loss)
* 2: normal diploid state
* 3: 1-copy gain
* 4: amplification (>= 2-copy gain)

ind <- grep("CNVSNP", names(gbm))  
state <- round(mcols(gbm[[ind]])$Segment\_Mean)  
state[state > 2] <- 2  
state[state < -2] <- -2  
mcols(gbm[[ind]])$State <- state + 2  
mcols(gbm[[ind]]) <- mcols(gbm[[ind]])[,3:1]  
table(mcols(gbm[[ind]])$State)

##   
## 0 1 2 3 4   
## 3270 3025 10223 856 444

The data is now ready for CNV-expression association analysis, where we find only two CNV regions with sufficient sample size for testing using the default value of 10 for the minSamples argument.

res <- cnvEQTL(cnvrs="01\_GBM\_GISTIC\_Peaks-20160128",   
 calls="01\_GBM\_CNVSNP-20160128",   
 rcounts="01\_GBM\_RNASeq2GeneNorm-20160128\_ranged",   
 data=gbm, window="1Mbp", verbose=TRUE)

## harmonizing input:  
## removing 154 sampleMap rows not in names(experiments)

## Preprocessing RNA-seq data ...

## Summarizing per-sample CN state in each CNV region

## Excluding 10 cnvrs not satisfying min.samples threshold

## Analyzing 2 regions with >=1 gene in the given window

## 1 of 2

## 2 of 2

res

## DataFrame with 39 rows and 5 columns  
## CNVR Gene logFC  
## <character> <character> <numeric>  
## 1 chr1:3394251-6475685 RPL22 -0.725652140752693  
## 2 chr1:3394251-6475685 CAMTA1 -0.710092209593344  
## 3 chr1:3394251-6475685 KLHL21 -0.804994378329488  
## 4 chr1:3394251-6475685 DNAJC11 -0.639331827106962  
## 5 chr1:3394251-6475685 PHF13 -0.644451457052169  
## ... ... ... ...  
## 35 chr1:7908902-8336254 VAMP3 -0.380592228642377  
## 36 chr1:7908902-8336254 H6PD -0.487860462635182  
## 37 chr1:7908902-8336254 PER3 -0.496046999162807  
## 38 chr1:7908902-8336254 ERRFI1 -0.503783986501104  
## 39 chr1:7908902-8336254 SLC2A5 -0.445492206932205  
## PValue AdjPValue  
## <numeric> <numeric>  
## 1 9.61335285501132e-09 1.87460380672721e-07  
## 2 2.57086167012869e-07 2.11153045530697e-06  
## 3 5.49625845091602e-06 3.57256799309541e-05  
## 4 6.45920365909795e-06 3.59869918149743e-05  
## 5 1.15203955333185e-05 4.99217139777135e-05  
## ... ... ...  
## 35 0.00155925324719434 0.0035771103906223  
## 36 0.00420206699725758 0.00819403064465228  
## 37 0.0280814360824601 0.0456323336339976  
## 38 0.0500141322286792 0.0722717568907921  
## 39 0.064396891327881 0.0866027159237021

# CNV-phenotype association analysis

For CNV calls inferred from SNP-chip or sequencing data, we additionally provide functionality to carry out a probe-level genome-wide association study (GWAS) between CNVs and quantitative phenotypes as previously described ([da Silva et al., 2016](https://doi.org/10.1371/journal.pone.0157711)).

This treats common CN polymorphisms (CNPs, allele frequency >1%) as SNPs of equal frequency and carries out a GWAS as implemented in [PLINK](http://zzz.bwh.harvard.edu/plink/gvar.shtml). The current version of cnvGWAS fucntion runs the CNV-phenotype in the background with [PLINK](http://zzz.bwh.harvard.edu/plink/gvar.shtml), import the results and assign SNP probe *p*-values to their respective CNV segments.

For demonstration, we use CNV data of a wild population of songbirds ([da Silva et al., 2018](https://doi.org/10.1186/s12864-018-4577-1)).

As before we read in the CNV calls and store them in a GRangesList.

cnv.loc <- file.path(data.dir, "CNVOut.txt")   
cnv.calls <- read.delim(cnv.loc, as.is=TRUE)   
cnv.calls <- makeGRangesListFromDataFrame(cnv.calls,   
 split.field="sample.id", keep.extra.columns=TRUE)  
cnv.calls

## GRangesList object of length 10:  
## $112   
## GRanges object with 2 ranges and 4 metadata columns:  
## seqnames ranges strand | state num.snps  
## <Rle> <IRanges> <Rle> | <integer> <integer>  
## [1] 1 100727703-100730748 \* | 0 8  
## [2] 10 19062731-19096193 \* | 3 9  
## start.probe end.probe  
## <character> <character>  
## [1] AX-100388724 AX-100765659  
## [2] AX-100271359 AX-100147230  
##   
## $175   
## GRanges object with 2 ranges and 4 metadata columns:  
## seqnames ranges strand | state num.snps start.probe  
## [1] 27 2299391-2308228 \* | 3 6 AX-100610990  
## [2] 8 4122253-4193189 \* | 3 62 AX-100097083  
## end.probe  
## [1] AX-100178489  
## [2] AX-100912769  
##   
## $356   
## GRanges object with 1 range and 4 metadata columns:  
## seqnames ranges strand | state num.snps start.probe  
## [1] 1 100728444-100730748 \* | 0 6 AX-100700982  
## end.probe  
## [1] AX-100765659  
##   
## ...  
## <7 more elements>  
## -------  
## seqinfo: 10 sequences from an unspecified genome; no seqlengths

Here, we use genomic estimated breeding values (GEBVs) for breeding time (BT) as the quantitative phenotype, and accordingly analyze for each CNV region whether change in copy number is associated with change in the genetic potential for breeding time.

## Setting up a CNV-GWAS

The function setupCnvGWAS imports CNV calls, phenotype information, and the probe map (if available). The information required for analysis is then stored in the resulting phen.info list:

## GEBV values  
phen.loc <- file.path(data.dir, "Pheno.txt")  
  
## Genomic positions of the probes used in the CNV call  
map.loc <- file.path(data.dir, "MapPenn.txt")  
  
phen.info <- setupCnvGWAS("example", phen.loc, cnv.calls, map.loc)

## Phenotypes and CNVs can be also be imported as a RaggedExperiment object  
phen.df <- read.delim(phen.loc)  
colDat <- DataFrame(phen.df)  
re <- RaggedExperiment(cnv.calls, colData=colDat)  
re

## class: RaggedExperiment   
## dim: 19 10   
## assays(4): state num.snps start.probe end.probe  
## rownames: NULL  
## colnames(10): 112 175 ... 1334 2391  
## colData names(4): sample.id fam sex BT

phen.info <- setupCnvGWAS("example", cnv.out.loc=re, map.loc=map.loc)  
phen.info

## $samplesPhen  
## [1] "911" "622" "1195" "112" "175" "2391" "1068" "546" "356" "1334"  
##   
## $phenotypes  
## [1] "BT"  
##   
## $phenotypesdf  
## BT  
## 1 -2.842842  
## 2 -2.884186  
## 3 -3.062731  
## 4 -3.161435  
## 5 -3.597768  
## 6 3.623262  
## 7 3.216123  
## 8 3.129881  
## 9 3.106459  
## 10 3.004740  
##   
## $phenotypesSam  
## samplesPhen BT  
## 1 911 -2.842842  
## 2 622 -2.884186  
## 3 1195 -3.062731  
## 4 112 -3.161435  
## 5 175 -3.597768  
## 6 2391 3.623262  
## 7 1068 3.216123  
## 8 546 3.129881  
## 9 356 3.106459  
## 10 1334 3.004740  
##   
## $FamID  
## samplesPhen V2  
## 1 911 -9  
## 2 622 -9  
## 3 1195 622  
## 4 112 -9  
## 5 175 -9  
## 6 2391 -9  
## 7 1068 -9  
## 8 546 -9  
## 9 356 -9  
## 10 1334 -9  
##   
## $SexIds  
## samplesPhen V2  
## 1 911 2  
## 2 622 2  
## 3 1195 2  
## 4 112 2  
## 5 175 2  
## 6 2391 2  
## 7 1068 2  
## 8 546 2  
## 9 356 2  
## 10 1334 2  
##   
## $all.paths  
## Inputs   
## "C:\\Users\\vinic\\AppData\\Local\\CNVRanger\\CNVRanger/example/Inputs"   
## PLINK   
## "C:/Users/vinic/AppData/Local/CNVRanger/CNVRanger/example/PLINK/plink-1.07-dos"   
## Results   
## "C:\\Users\\vinic\\AppData\\Local\\CNVRanger\\CNVRanger/example/Results"

The last item of the list displays the working directory:

all.paths <- phen.info$all.paths  
all.paths

## Inputs   
## "C:\\Users\\vinic\\AppData\\Local\\CNVRanger\\CNVRanger/example/Inputs"   
## PLINK   
## "C:/Users/vinic/AppData/Local/CNVRanger/CNVRanger/example/PLINK/plink-1.07-dos"   
## Results   
## "C:\\Users\\vinic\\AppData\\Local\\CNVRanger\\CNVRanger/example/Results"

For the GWAS, chromosome names are assumed to be integer (i.e. 1, 2, 3, ...). Non-integer chromosome names can be encoded by providing a data.frame that describes the mapping from character names to corresponding integers.

For the example data, chromosomes *1A*, *4A*, *25LG1*, *25LG2*, and *LGE22* are correspondingly encoded via

# Define chr correspondence to numeric  
chr.code.name<- data.frame(   
 V1=c(16, 25, 29:31),   
 V2=c("1A", "4A", "25LG1", "25LG2", "LGE22"))

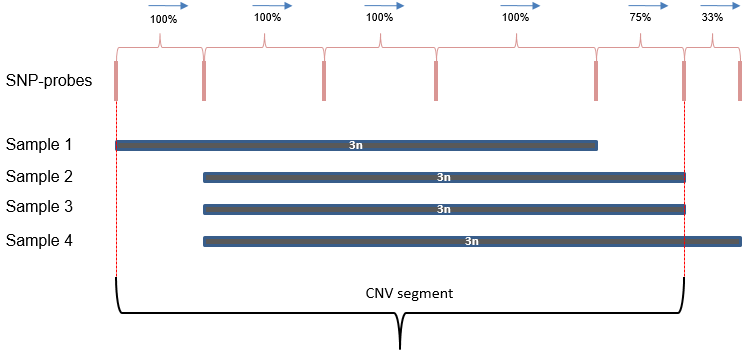
## Running a CNV-GWAS

We can then run the actual CNV-GWAS, here without correction for multiple testing which is done *for demonstration only*. In real analyses, multiple testing correction is recommended to avoid inflation of false positive findings.

segs.pvalue.gr <- cnvGWAS(phen.info, chr.code.name=chr.code.name, method.m.test="none")  
segs.pvalue.gr

## GRanges object with 16 ranges and 6 metadata columns:  
## seqnames ranges strand | SegName MinPvalue  
## <Rle> <IRanges> <Rle> | <integer> <numeric>  
## [1] 1 98171563-98184039 \* | 2 0.0323  
## [2] 8 4121283-4188293 \* | 7 0.03494  
## [3] 8 4193189 \* | 8 0.1124  
## [4] 1 98186123-98186543 \* | 3 0.1202  
## [5] 1 98147555-98171009 \* | 1 0.1977  
## ... ... ... ... . ... ...  
## [12] 18 1278467-1295371 \* | 13 0.3856  
## [13] 11 18840662 \* | 12 0.3926  
## [14] 21 3326720-3329134 \* | 14 0.3926  
## [15] 11 18836038-18839377 \* | 11 0.9688  
## [16] 1 100728444-100730326 \* | 4 0.9722  
## NameProbe Frequency MinPvalueAdjusted Phenotype  
## <character> <character> <numeric> <character>  
## [1] AX-100075281 3 0.0323 BT  
## [2] AX-100015058 3 0.03494 BT  
## [3] AX-100912769 2 0.1124 BT  
## [4] AX-100011247 2 0.1202 BT  
## [5] AX-100171012 4 0.1977 BT  
## ... ... ... ... ...  
## [12] AX-100116173 1 0.3856 BT  
## [13] AX-100673859 1 0.3926 BT  
## [14] AX-100389358 1 0.3926 BT  
## [15] AX-100337682 2 0.9688 BT  
## [16] AX-100108194 2 0.9722 BT  
## -------  
## seqinfo: 10 sequences from an unspecified genome; no seqlengths

The CNV-GWAS uses the concept of CNV segments to define CNV loci.



Definition of CNV segments

This procedure was originally proposed in our previous work in Nelore cattle ([da Silva et al., 2016](https://doi.org/10.1371/journal.pone.0157711)) and defines CNV segments based on CNV genotype similarity of subsequent SNP probes.

The default is min.sim=0.95, which will continuously add probe positions to a given CNV segment until the pairwise genotype similarity drops below 95%. An example of detailed up-down CNV genotype concordance that is used for the construction of CNV segments is given in S12 Table in [da Silva et al., 2016](https://doi.org/10.1371/journal.pone.0157711).

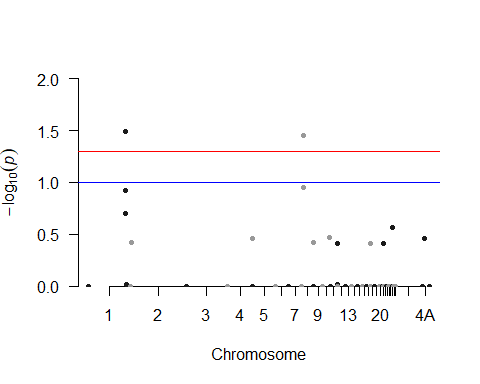
As PLINK returns a *p*-value for each probe, only one of the *p*-values of the probes contained in a CNV segment is chosen as the segment *p*-value.

This is similar to a common approach used in differential expression (DE) analysis of microarray gene expression data, where typically the most significant DE probe is chosen in case of multiple probes mapping to the same gene.

Here, the representative probe for the CNV segment can be chosen to be the probe with lowest *p*-value (assign.probe="min.pvalue", default) or the one with highest CNV frequency (assign.probe="high.freq").

Multiple testing correction based on the number of CNV segments tested is carried out using the FDR approach (default). Results can then be displayed as for regular GWAS via a Manhattan plot (which can optionally be exported to a pdf file).

## Define the chromosome order in the plot  
order.chrs <- c(1:24, "25LG1", "25LG2", 27:28, "LGE22", "1A", "4A")  
  
## Chromosome sizes  
chr.size.file <- file.path(data.dir, "Parus\_major\_chr\_sizes.txt")  
chr.sizes <- scan(chr.size.file)  
chr.size.order <- data.frame(chr=order.chrs, sizes=chr.sizes, stringsAsFactors=FALSE)  
  
## Plot a pdf file with a manhatthan within the 'Results' workfolder  
plotManhattan(all.paths, segs.pvalue.gr, chr.size.order, plot.pdf=FALSE)



## Producing a GDS file in advance

To support an efficient memory management in genotype analysis, the genomic data structure (GDS) file format for array-oriented data was previously implemented in the [*gdsfmt*](https://bioconductor.org/packages/3.9/gdsfmt) package. Thus, to make use of this efficiency, CNV genotypes constructed and associated with the cnvGWAS function are stored in a CNV.gds file, which is automatically produced and placed in the Inputs folder (i.e. all.paths[1]).

Therefore, running a GWAS implies that any GDS file produced by previous analysis will be overwritten. Use produce.gds=FALSE to avoid overwriting in the GWAS run.

For convenience, a GDS file can be produced before the GWAS analysis with the generateGDS function. This additionally returns a GRanges object containing the genomic position, name and, frequency of each probe used to construct the CNV segments for the GWAS analysis.

Note that probes.cnv.gr object contains the integer chromosome names (as the GDS file on disk). Only the segs.pvalue.gr, which stores the GWAS results, has the character chromosome names.

## Create a GDS file in disk and export the SNP probe positions  
probes.cnv.gr <- generateGDS(phen.info, chr.code.name=chr.code.name)  
probes.cnv.gr

## GRanges object with 189 ranges and 3 metadata columns:  
## seqnames ranges strand | Name freq snp.id  
## <Rle> <IRanges> <Rle> | <character> <integer> <integer>  
## [1] 1 98147555 \* | AX-100939600 2 1  
## [2] 1 98148072 \* | AX-100088448 2 2  
## [3] 1 98150537 \* | AX-100954037 2 3  
## [4] 1 98151270 \* | AX-100836117 2 4  
## [5] 1 98151959 \* | AX-100027637 2 5  
## ... ... ... ... . ... ... ...  
## [185] 25 6471766 \* | AX-100066308 1 185  
## [186] 25 6473449 \* | AX-100023435 1 186  
## [187] 25 6474550 \* | AX-100397956 1 187  
## [188] 25 6475943 \* | AX-100363929 1 188  
## [189] 27 2308228 \* | AX-100178489 1 189  
## -------  
## seqinfo: 15 sequences from an unspecified genome; no seqlengths

## Run GWAS with existent GDS file  
segs.pvalue.gr <- cnvGWAS(phen.info, chr.code.name=chr.code.name, produce.gds=FALSE)

## Using relative signal intensities

CNV detection using SNP-chip intensities and allele frequencies can produce biased CNV frequencies ([da Silva et al., 2018](https://doi.org/10.1186/s12864-018-4577-1)). Therefore, we also provide the option to carry out the GWAS based on the SNP-wise relative signal intensities (log R ratios, LRRs), which reflect at certain extend the number of copies on the SNP region and are used as input for several CNV callers such as [PennCNV](http://penncnv.openbioinformatics.org) and [Birdsuit](https://www.broadinstitute.org/birdsuite/birdsuite). Directly using LRR values for the GWAS thereby facilitates the identification of CNV segments associated with the phenotype, disregarding putatively biased frequency estimates from a strict CNV calling procedure. Although sometimes interesting as an auxiliar analysis, using LRR values directly can generate false positive results as higher noise is expected in comparison with a CNV dataset detected and properly filtered by an external CNV caller software.

To perform the GWAS using LRR values, import the LRR/BAF values and set run.lrr=TRUE in the cnvGWAS function:

# List files to import LRR/BAF   
files <- list.files(data.dir, pattern = "\\.cnv.txt.adjusted$")  
samples <- sub(".cnv.txt.adjusted$", "", files)  
samples <- sub("^GT","", samples)  
sample.files <- data.frame(file.names=files, sample.names=samples)  
   
# All missing samples will have LRR = '0' and BAF = '0.5' in all SNPs listed in the GDS file  
importLrrBaf(all.paths, data.dir, sample.files, verbose=FALSE)  
  
# Read the GDS to check if the LRR/BAF nodes were added  
cnv.gds <- file.path(all.paths[1], "CNV.gds")  
(genofile <- SNPRelate::snpgdsOpen(cnv.gds, allow.fork=TRUE, readonly=FALSE))

## File: C:\Users\vinic\AppData\Local\CNVRanger\CNVRanger\example\Inputs\CNV.gds (49.8K)  
## + [ ] \*  
## |--+ sample.id { Str8 10 ZIP\_ra(122.7%), 61B }  
## |--+ snp.id { Str8 189 ZIP\_ra(45.4%), 301B }  
## |--+ snp.rs.id { Str8 189 ZIP\_ra(31.9%), 791B }  
## |--+ snp.position { Int32 189 ZIP\_ra(86.2%), 659B }  
## |--+ snp.chromosome { Str8 189 ZIP\_ra(11.8%), 56B }  
## |--+ genotype { Bit2 189x10, 473B } \*  
## |--+ CNVgenotype { Float64 189x10, 14.8K }  
## |--+ phenotype [ data.frame ] \*  
## | |--+ samplesPhen { Str8 10, 44B }  
## | \--+ BT { Float64 10, 80B }  
## |--+ FamID { Str8 10, 31B }  
## |--+ Sex { Str8 10, 20B }  
## |--+ Chr.names [ data.frame ] \*  
## | |--+ V1 { Float64 5, 40B }  
## | \--+ V2 { Int32,factor 5, 20B } \*  
## |--+ LRR { Float64 189x10, 14.8K }  
## \--+ BAF { Float64 189x10, 14.8K }

gdsfmt::closefn.gds(genofile)  
  
# Run the CNV-GWAS with existent GDS  
segs.pvalue.gr <- cnvGWAS(phen.info, chr.code.name=chr.code.name, produce.gds=FALSE, run.lrr=TRUE)

# Session info

sessionInfo()

## R version 3.6.0 (2019-04-26)  
## Platform: x86\_64-w64-mingw32/x64 (64-bit)  
## Running under: Windows >= 8 x64 (build 9200)  
##   
## Matrix products: default  
##   
## Random number generation:  
## RNG: Mersenne-Twister   
## Normal: Inversion   
## Sample: Rounding   
##   
## locale:  
## [1] LC\_COLLATE=English\_United States.1252   
## [2] LC\_CTYPE=English\_United States.1252   
## [3] LC\_MONETARY=English\_United States.1252  
## [4] LC\_NUMERIC=C   
## [5] LC\_TIME=English\_United States.1252   
##   
## attached base packages:  
## [1] parallel stats4 stats graphics grDevices utils datasets   
## [8] methods base   
##   
## other attached packages:  
## [1] ensembldb\_2.8.0   
## [2] AnnotationFilter\_1.8.0   
## [3] GenomicFeatures\_1.36.1   
## [4] AnnotationDbi\_1.46.0   
## [5] TCGAutils\_1.4.0   
## [6] curatedTCGAData\_1.6.0   
## [7] MultiAssayExperiment\_1.10.4   
## [8] SummarizedExperiment\_1.14.0   
## [9] DelayedArray\_0.10.0   
## [10] BiocParallel\_1.17.18   
## [11] matrixStats\_0.54.0   
## [12] Biobase\_2.44.0   
## [13] BSgenome.Btaurus.UCSC.bosTau6.masked\_1.3.99  
## [14] BSgenome.Btaurus.UCSC.bosTau6\_1.4.0   
## [15] BSgenome\_1.52.0   
## [16] rtracklayer\_1.44.0   
## [17] Biostrings\_2.52.0   
## [18] XVector\_0.24.0   
## [19] regioneR\_1.16.2   
## [20] AnnotationHub\_2.16.0   
## [21] BiocFileCache\_1.8.0   
## [22] dbplyr\_1.4.2   
## [23] BiocStyle\_2.12.0   
## [24] usethis\_1.5.0   
## [25] devtools\_2.0.2   
## [26] CNVRanger\_1.1.0   
## [27] RaggedExperiment\_1.8.0   
## [28] GenomicRanges\_1.36.0   
## [29] GenomeInfoDb\_1.20.0   
## [30] IRanges\_2.18.1   
## [31] S4Vectors\_0.22.0   
## [32] BiocGenerics\_0.30.0   
##   
## loaded via a namespace (and not attached):  
## [1] rprojroot\_1.3-2   
## [2] qqman\_0.1.4   
## [3] fs\_1.3.1   
## [4] rstudioapi\_0.10   
## [5] roxygen2\_6.1.1   
## [6] remotes\_2.0.4   
## [7] bit64\_0.9-7   
## [8] interactiveDisplayBase\_1.22.0   
## [9] xml2\_1.2.0   
## [10] splines\_3.6.0   
## [11] knitr\_1.23   
## [12] pkgload\_1.0.2   
## [13] GDSArray\_1.4.2   
## [14] jsonlite\_1.6   
## [15] Rsamtools\_2.0.0   
## [16] shiny\_1.3.2   
## [17] BiocManager\_1.30.4   
## [18] readr\_1.3.1   
## [19] compiler\_3.6.0   
## [20] httr\_1.4.0   
## [21] backports\_1.1.4   
## [22] lazyeval\_0.2.2   
## [23] assertthat\_0.2.1   
## [24] Matrix\_1.2-17   
## [25] TxDb.Hsapiens.UCSC.hg19.knownGene\_3.2.2  
## [26] limma\_3.40.2   
## [27] cli\_1.1.0   
## [28] later\_0.8.0   
## [29] htmltools\_0.3.6   
## [30] prettyunits\_1.0.2   
## [31] tools\_3.6.0   
## [32] glue\_1.3.1   
## [33] GenomeInfoDbData\_1.2.1   
## [34] reshape2\_1.4.3   
## [35] dplyr\_0.8.1   
## [36] rappdirs\_0.3.1   
## [37] Rcpp\_1.0.1   
## [38] ExperimentHub\_1.10.0   
## [39] xfun\_0.7   
## [40] stringr\_1.4.0   
## [41] ps\_1.3.0   
## [42] rvest\_0.3.4   
## [43] mime\_0.7   
## [44] statmod\_1.4.32   
## [45] XML\_3.98-1.20   
## [46] SNPRelate\_1.18.0   
## [47] org.Hs.eg.db\_3.8.2   
## [48] edgeR\_3.26.4   
## [49] zlibbioc\_1.30.0   
## [50] ProtGenerics\_1.16.0   
## [51] hms\_0.4.2   
## [52] promises\_1.0.1   
## [53] gdsfmt\_1.20.0   
## [54] yaml\_2.2.0   
## [55] curl\_3.3   
## [56] memoise\_1.1.0   
## [57] biomaRt\_2.40.0   
## [58] calibrate\_1.7.2   
## [59] stringi\_1.4.3   
## [60] RSQLite\_2.1.1   
## [61] highr\_0.8   
## [62] desc\_1.2.0   
## [63] GenomicDataCommons\_1.8.0   
## [64] pkgbuild\_1.0.3   
## [65] rlang\_0.3.4   
## [66] pkgconfig\_2.0.2   
## [67] commonmark\_1.7   
## [68] bitops\_1.0-6   
## [69] SeqArray\_1.24.1   
## [70] evaluate\_0.14   
## [71] lattice\_0.20-38   
## [72] purrr\_0.3.2   
## [73] GenomicAlignments\_1.20.0   
## [74] bit\_1.1-14   
## [75] processx\_3.3.1   
## [76] tidyselect\_0.2.5   
## [77] plyr\_1.8.4   
## [78] magrittr\_1.5   
## [79] R6\_2.4.0   
## [80] DBI\_1.0.0   
## [81] pillar\_1.4.1   
## [82] withr\_2.1.2   
## [83] RCurl\_1.95-4.12   
## [84] tibble\_2.1.3   
## [85] crayon\_1.3.4   
## [86] rmarkdown\_1.13   
## [87] progress\_1.2.2   
## [88] locfit\_1.5-9.1   
## [89] grid\_3.6.0   
## [90] data.table\_1.12.2   
## [91] blob\_1.1.1   
## [92] callr\_3.2.0   
## [93] digest\_0.6.19   
## [94] xtable\_1.8-4   
## [95] httpuv\_1.5.1   
## [96] sessioninfo\_1.1.1