

CNVrd2: A package for measuring gene copy number, identifying SNPs tagging copy number variants, and detecting copy number polymorphic genomic regions

Hoang Tan Nguyen^{1,2}, Tony R Merriman¹ and Michael A Black¹

¹Department of Biochemistry, University of Otago

²Department of Mathematics and Statistics, University of Otago

April 15, 2025

Contents

1	Introduction	2
2	Getting started	3
2.1	Measuring FCGR3B CN	6
2.1.1	CNVrd2 object	6
2.1.2	Count reads in windows	7
2.1.3	Segmentation	7
2.1.4	Obtain copy-number count	8
2.1.5	Plots	12
2.2	Identifying tag SNPs/INDELs for FCGR3B CNVs	12
3	Working with complex loci	14
3.1	Measuring CCL3L1 CN	16
3.2	Identifying tag-SNPs for CCL3L1 CNVs	22
4	Identifying polymorphic regions	25
5	Note	28
6	Session information	28

1 Introduction

The *CNVrd2* package¹ (Nguyen *et al.*, 2014) utilizes next-generation sequencing (NGS) data to measure human-gene copy number (CN) and identify single-nucleotide polymorphisms (SNPs), and insertions and deletions (INDELs) that are in linkage disequilibrium with a gene of interest. Typically, the data being used are low- or medium-coverage whole genome sequence (WGS) data from multiple individuals in a population. Such data comprise collections of sequence reads that have been aligned (or "mapped") to an appropriate reference genome. Changes in read depth (i.e., the number of reads aligned to a specific region of the genome) can indicate changes in DNA copy number in this region (i.e., deletions or duplications of specific portions of DNA). If this region encompasses a gene, then changes in copy number may also be reflected by changes in gene activity - such changes have been shown to be associated with altered risk of disease in human populations, and altered trait distributions in agricultural settings.

To measure gene CN, *CNVrd2* firstly divides a region (usually at least 1Mb) flanking a gene of interest into constant-sized windows, and counts reads mapped in these windows. Next, these read-count windows are transformed and standardized. After that, the *DNAcopy* package (Venkatraman and Olshen, 2007) is used to join the per-window standardized counts into regions (or "segments") of similar values. The package then refines the segmentation step and outputs segmentation results, namely segmentation scores (SS), for each sample. A function in the *CNVrd2* package is then used to group SSs into copy-number groups.

To calculate linkage disequilibrium (LD) between gene CNVs and SNPs/INDELs nearby, SNPs/INDELs are coded into numeric values (0, 1, 2) and Fisher's Exact Test is used to assess associations between SNPs/INDELs and copy number. *CNVrd2* is designed to identify SNPs/INDELs that can be used as a surrogate marker for CNVs, therefore multiple samples are needed to obtain reliable results. The package also uses distribution quantiles to identify highly polymorphic regions of the genome (within a collection of samples) and can identify regions with variable polymorphism between populations. The BAM format (Li *et al.*, 2009) for aligned-NGS data and VCF format (Danecek *et al.*, 2011) for structural variant information are used as the main forms of input for the package.

¹CNVrd2 is an improved version of the pipeline *CNVrd* used to identify tagSNPs of *FCGR3A/B* CNV

2 Getting started

First, we load the package in our R session. Note that the *rjags* package ([Plummer, 2013](#)) requires the associated JAGS application to be installed.

```
library('CNVrd2')

## Loading required package: VariantAnnotation
## Loading required package: BiocGenerics
## Loading required package: generics
##
## Attaching package: 'generics'
## The following objects are masked from 'package:base':
##
##      as.difftime, as.factor, as.ordered, intersect, is.element,
##      setdiff, setequal, union
##
## Attaching package: 'BiocGenerics'
## The following objects are masked from 'package:stats':
##
##      IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##      Filter, Find, Map, Position, Reduce, anyDuplicated, aperm,
##      append, as.data.frame, basename, cbind, colnames, dirname,
##      do.call, duplicated, eval, evalq, get, grep, grepl,
##      is.unsorted, lapply, mapply, match, mget, order, paste,
##      pmax, pmax.int, pmin, pmin.int, rank, rbind, rownames,
##      sapply, saveRDS, table, tapply, unique, unsplit, which.max,
##      which.min
## Loading required package: MatrixGenerics
## Loading required package: matrixStats
##
## Attaching package: 'MatrixGenerics'
## The following objects are masked from 'package:matrixStats':
##
```

```

##      colAlls, colAnyNAs, colAnys, colAugsPerRowSet, colCollapse,
##      colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##      colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##      colMads, colMaxs, colMeans2, colMedians, colMins,
##      colOrderStats, colProds, colQuantiles, colRanges, colRanks,
##      colSdDiffs, colSds, colSums2, colTabulates, colVarDiffs,
##      colVars, colWeightedMads, colWeightedMeans,
##      colWeightedMedians, colWeightedSds, colWeightedVars,
##      rowAlls, rowAnyNAs, rowAnys, rowAugsPerColSet, rowCollapse,
##      rowCounts, rowCummaxs, rowCummins, rowCumprods, rowCumsums,
##      rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps, rowMadDiffs,
##      rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##      rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##      rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs,
##      rowVars, rowWeightedMads, rowWeightedMeans,
##      rowWeightedMedians, rowWeightedSds, rowWeightedVars

## Loading required package: GenomeInfoDb
## Loading required package: S4Vectors
## Loading required package: stats4
##
## Attaching package: 'S4Vectors'
## The following object is masked from 'package:utils':
##
##      findMatches
## The following objects are masked from 'package:base':
##
##      I, expand.grid, unname
## Loading required package: IRanges
##
## Attaching package: 'IRanges'
## The following object is masked from 'package:grDevices':
##
##      windows
## Loading required package: GenomicRanges
## Loading required package: SummarizedExperiment

```

```

## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages
##     'citation("pkgname")'.
##
## Attaching package: 'Biobase'
## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians
## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians
## Loading required package: Rsamtools
## Loading required package: Biostrings
## Loading required package: XVector
##
## Attaching package: 'Biostrings'
## The following object is masked from 'package:base':
##
##     strsplit
##
## Attaching package: 'VariantAnnotation'
## The following object is masked from 'package:base':
##
##     tabulate
## Loading required package: parallel
## Loading required package: rjags
## Loading required package: coda
## Linked to JAGS 4.3.0
## Loaded modules: basemod, bugs
## Loading required package: ggplot2
## Loading required package: gridExtra

```

```
##
## Attaching package: 'gridExtra'
## The following object is masked from 'package:Biobase':
##
##      combine
## The following object is masked from 'package:BiocGenerics':
##
##      combine
```

Working with BAM and VCF files.

The following section describes the workflow of the *CNVrd2* package in reading BAM and VCF files into R. The 58 MXL-sample BAM files (chr1:161100000-162100000) were downloaded from the 1000 Genomes Project to measure copy number counts of *FCGR3B* gene (chr1:161592986-161601753). Users can download the file *MXLexample.zip* on

<http://code.google.com/p/cnvrdfortagsnps/downloads/list> and unzip it into a directory.

Alternatively, to run the example without downloading the associated BAM files, users can skip to section 2.1.3 to load a pre-processed version of the same data.

2.1 Measuring FCGR3B CN

2.1.1 CNVrd2 object

We need to make an object of class *CNVrd2* to define a region we want to investigate (regions sized > 1Mb tend to work well - multiple genes can be included by specifying the start and end positions of each). Here, we choose 1000bp-constant windows. We also need to supply a directory that consists of BAM files including only mapped reads. Users who have **not downloaded the BAM files**, should skip to section 2.1.3

```
objectCNVrd2 <- new("CNVrd2", windows = 1000, chr = "chr1",
  st = 161100001, en = 162100000,
  dirBamFile = "BamMXL",
  genes = c(161592986, 161601753),
  geneNames = "3B")
```

2.1.2 Count reads in windows

Use the function *countReadInWindow* to read the BAM files into R and count the number of reads in each of the windows.

```
readCountMatrix <- countReadInWindow(Object = objectCNVrd2, correctGC = TRUE)
```

If GC-content correction is selected (*correctGC=TRUE*) then a reference genome must be supplied. The default reference genome is the human reference genome (UCSC version hg19). A full list of reference genomes available through Bioconductor can be obtained from:

<http://www.bioconductor.org/packages/release/bioc/html/BSgenome.html>

2.1.3 Segmentation

Use the function *segmentSamples* to segment and obtain segmentation scores for the *FCGR3B* gene (Figure 1):

```
##Obtain segmentation scores
resultSegment <- segmentSamples(Object = objectCNVrd2, stdCntMatrix = readCountMatrix)
```

Instead of reading BAM files directly, we can use a matrix of read counts for the function *segmentSamples*. Here, we obtain a read-count matrix from data in the *CNVrd2* package.

```
##Load data into R
data(fcgr3bMXL)
##Reload readCountMatrix
readCountMatrix <- resultSegment$stdCntMatrix
##Take a quick look the data
readCountMatrix[1:2, 1:2]
##Make a CNVrd2 object
objectCNVrd2 <- new("CNVrd2", windows = 1000, chr = "chr1",
                    st = 161100001, en = 162100000,
                    dirBamFile = "BamMXL",
                    genes = c(161592986, 161601753),
                    geneNames = "3B")
##Obtain segmentation scores
resultSegment <- segmentSamples(Object = objectCNVrd2, stdCntMatrix = readCountMatrix)
```

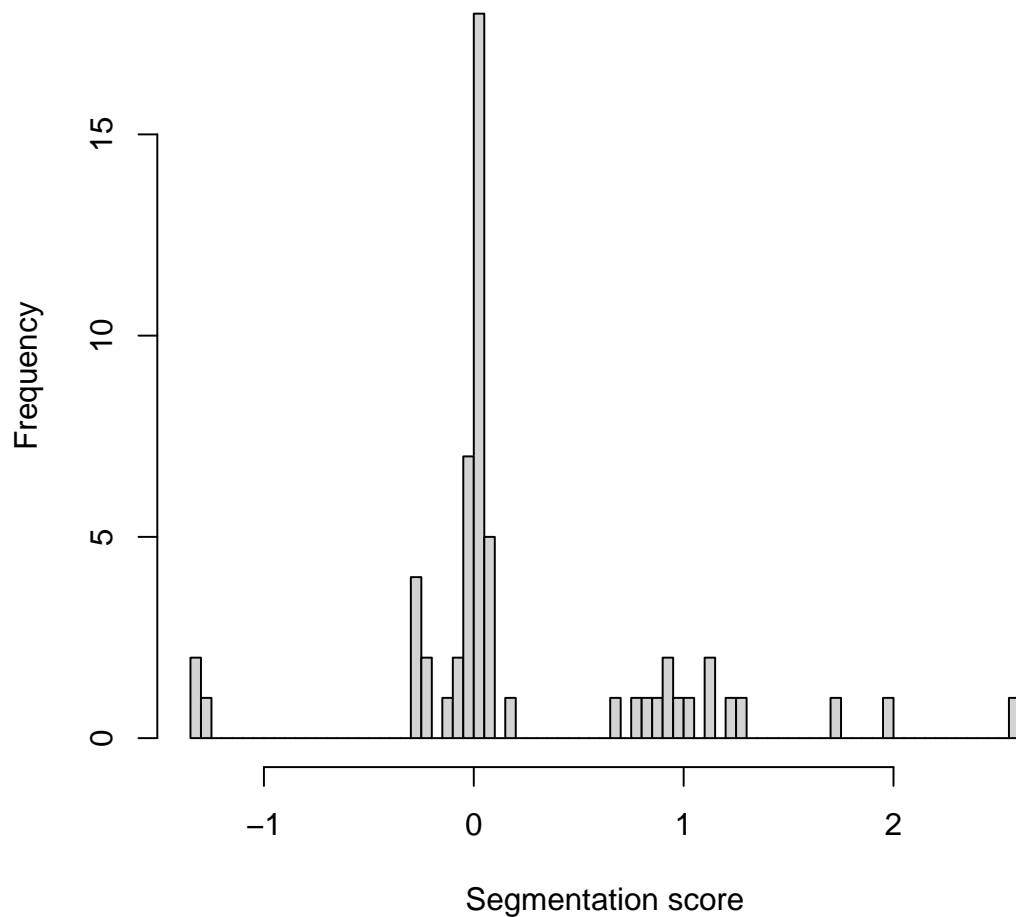


Figure 1: FCGR3B segmentation score.

```
##View these segmentation results
sS <- resultSegment$segmentationScores
hist(sS[, 1], 100, xlab = 'Segmentation score', main = '')
```

2.1.4 Obtain copy-number count

The data in Figure 1 suggest four distinct groups of segmentation scores, likely related to four different copy number genotypes. The function *groupCNVs* uses a normal mixture model to cluster SSs into groups. Unequal variances are assumed by default (*EV = FALSE*), however, if there are relatively few SS values in one group then we can set *EV = TRUE* (see the *groupCNVs* manual

page for additional details).

```
objectCluster <- new("clusteringCNVs",
                     x = resultSegment$segmentationScores[, 1], k = 4, EV = TRUE)
#Cluster into 4 groups
copynumberGroups <- groupCNVs(Object = objectCluster)

## =====
## 7 iterations
## 4 components with Equal variances
## m:  -1.30876666666667-0.02580184504113590.9838838408971832.08487018121448
## p:  0.05172413793153550.6896606864511990.2069007456105260.0517144300067392
## sigma:  0.1469858562965880.1469858562965880.1469858562965880.146985856296588
## =====
```

Clustering results are shown in Figure 2, and the group assignments for the samples are contained in the *allGroups* object. For example, the NA19648 sample is assigned to the second group because the probability associated with membership of this group is higher than that of the other groups (nearly 1).

```
copynumberGroups$allGroups[1:3, ]

##              Name Classification      Group1
## NA19648.MXL.bam NA19648.MXL.bam        2 1.784492e-18
## NA19649.MXL.bam NA19649.MXL.bam        2 3.742821e-17
## NA19651.MXL.bam NA19651.MXL.bam        3 5.012828e-61
##              Group2      Group3      Group4      score
## NA19648.MXL.bam 1.000000e+00 2.472136e-11 3.368400e-46 -0.0195
## NA19649.MXL.bam 1.000000e+00 2.250428e-12 2.248390e-48 -0.0710
## NA19651.MXL.bam 1.540821e-13 1.000000e+00 4.647376e-10  1.1393
```

If we would like to force outliers into the lowest or highest CN genotype groups (e.g., dividing the data into three groups: deletions, normal CN, duplications) then we can use options *rightLimit* (Figure 3) or *leftLimit* or both.

```
#Set right limit = 1.5 to make values > 1.5 be into the largest group.
objectCluster <- new("clusteringCNVs",
                     x = resultSegment$segmentationScores[, 1], k = 3, EV = TRUE)
```



Figure 2: FCGR3B CN groups.

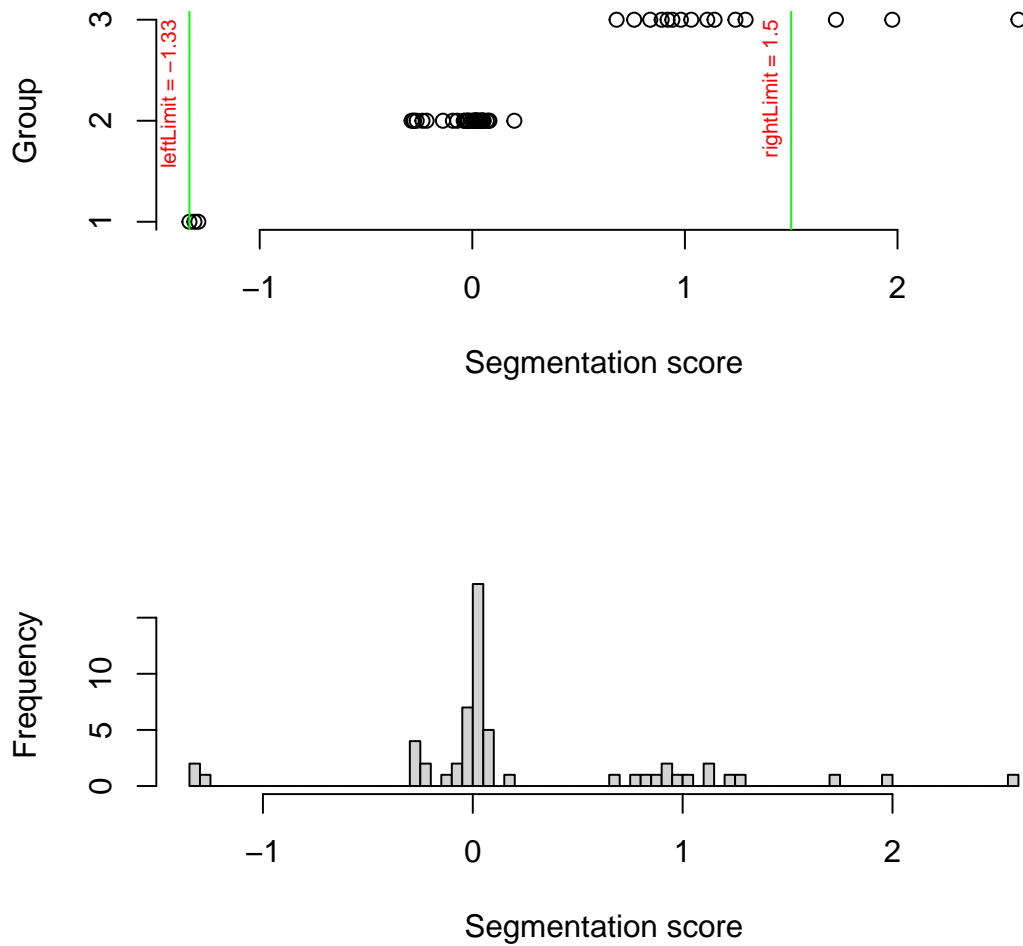


Figure 3: FCGR3B CN groups (rightLimit = 1.5).

```
copynumberGroups <- groupCNVs(Object = objectCluster, rightLimit = 1.5)

## =====
## 5 iterations
## 3 components with Equal variances
## m: -1.30876666666667-0.02580733715714130.983841901058517
## p: 0.05454545454545460.7272728952233420.218181650231204
## sigma: 0.1254981622689890.1254981622689890.125498162268989
## =====
```

2.1.5 Plots

The function *plotCNVrd2* can plot multiple samples. Trace plots of some of the samples exhibiting duplications at the FCGR locus are shown in Figure 4. Here, based on information from the literature, we assume that a copy number of two is the most common CN genotype.

```
allGroups <- copynumberGroups$allGroups
###Obtain names of duplicate samples
duplicatedSamples <- rownames(allGroups[allGroups[, 2] > 2,])
###Plot 6 duplicate samples
par(mfrow = c(3, 2))
for (ii in duplicatedSamples[1:6])
  plotCNVrd2(object = objectCNVrd2,

             segmentObject = resultSegment,

             sampleName = ii)
```

2.2 Identifying tag SNPs/INDELs for FCGR3B CNVs

The function *calculateLDSNPandCNV* is used to calculate LD between CNVs and SNPs/INDELs. This function will read a VCF file into R and transform phased/unphased values (00, 01, 10, 11) into numeric values (0, 1, 2 or 0, 1). For a large VCF file (e.g., ≥ 1 Mb), we generally use the option *nChunkForVcf=50* to break the file into 50 chunks for reading into R.

```
##Obtain VCF-file information in CNVrd2 package
vcfFile <- system.file(package="CNVrd2", "extdata",
                      "chr1.161600000.161611000.vcf.gz")
##Make a data frame named sampleCNV including samples, CNs, population names
sampleCNV <- data.frame(copynumberGroups$allGroups[, c(1,2) ],rep("MXL", dim(copynumberGroups$allGroups)[1]),
rownames(sampleCNV) <- substr(sampleCNV[, 1], 1, 7)
sampleCNV[, 1] <- rownames(sampleCNV)
##The first column must be the sample names and some samples should be in the vcf file
tagSNPandINDELofMXL <- calculateLDSNPandCNV(sampleCNV = sampleCNV,
                                             vcfFile = vcfFile, cnvColumn = 2,
                                             population = "MXL", popColumn = 3,
```

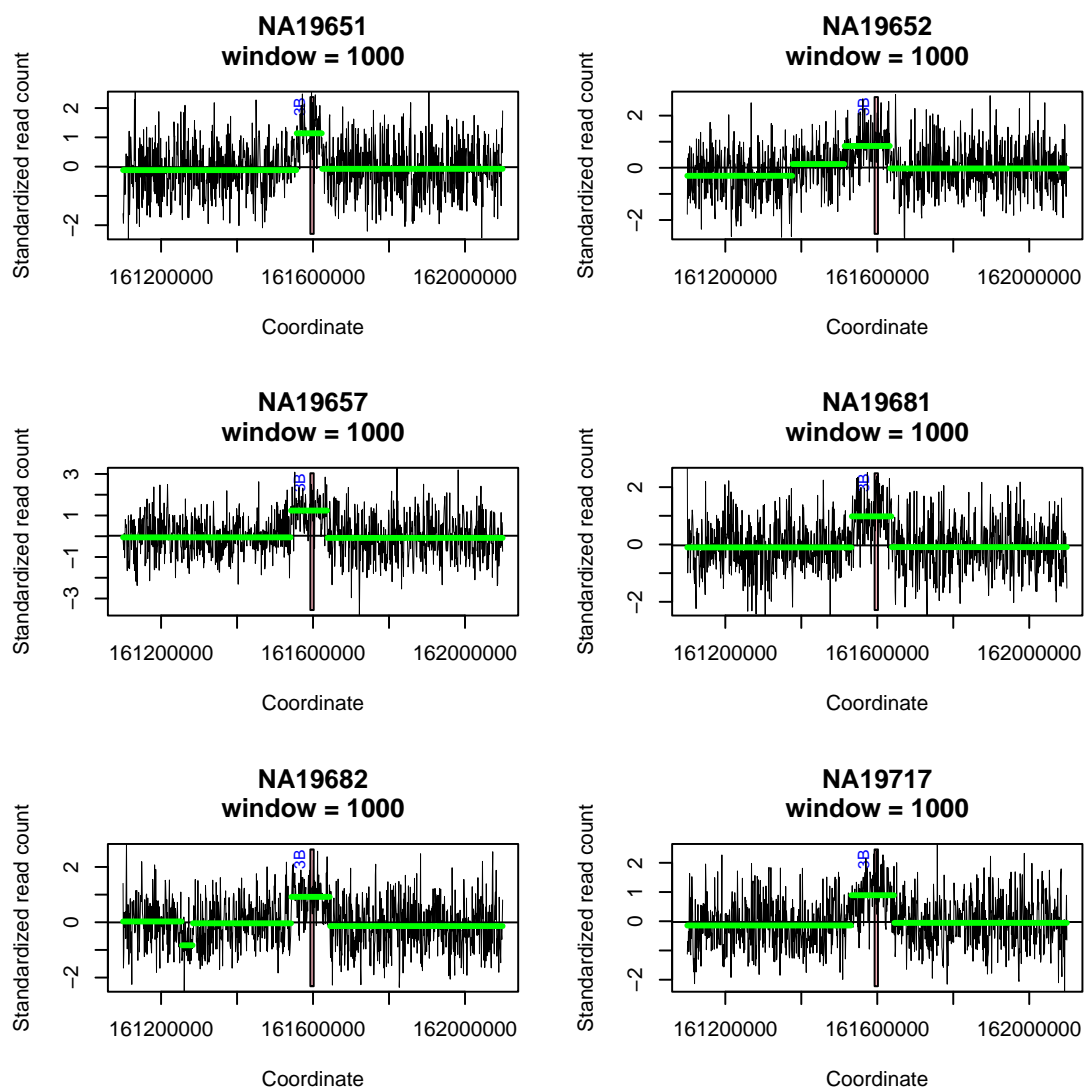


Figure 4: MXL duplicated samples.

```

nChunkForVcf = 5, chr = "1",
st = 161600001, en = 161611000,
codeSNP= "Three", codeCNV = "ThreeGroup")

## Reading the VCF file 1:161600001-161611000 with 5 blocks each
## VCF file: 161600001 to 161602751
## VCF file: 161602752 to 161605500
## VCF file: 161605501 to 161608250
## VCF file: 161608251 to 161611000
## Calculating p and r2 values for MXL population.

```

```

head(tagSNPandINDELofMXL)

##          1CN_(n=3) 2CN_(n=36) 3CN_(n=15) p.values      r
## rs117435514      0.00      2.78      86.67  2.1e-09  0.81
## rs185696163      0.00      2.78      86.67  2.1e-09  0.81
## rs34015117       0.00      2.78      80.00  2.3e-08  0.77
## rs76736176       0.00      2.78      66.67  1.5e-06  0.68
## esv2661911      66.67      0.00      0.00  2.1e-03 -0.39
## rs72704050      33.33     52.78     93.33  5.1e-03  0.34
##          p.valuesAdjusted    r2 POP
## rs117435514      1.3e-07 0.66 MXL
## rs185696163      1.3e-07 0.66 MXL
## rs34015117       9.6e-07 0.59 MXL
## rs76736176       4.7e-05 0.46 MXL
## esv2661911       5.4e-02 0.15 MXL
## rs72704050       1.1e-01 0.11 MXL

```

From the results of the LD analysis, *rs117435514* is the best tagSNP for duplications: 0%, 2.78% and 86.7% of deleted, normal and duplicated samples have this SNP (adjusted p-value = 7.1e-08, $r^2 = 0.66$).

3 Working with complex loci

CNVrd2 can also be used to measure multiallelic copy-number polymorphisms. For loci having high CN, users should use the function *segmentSamplesUsingPopInformation* to adjust the seg-

mentation process across populations. An xample of a gene exhibiting this type of complex CN polymorphism is *CCL3L1*. Below we use the package to measure *CCL3L1* CN and identify tag SNPs/INDELs for *CCL3L1* CNVs.

The data set used here includes 1,917 samples of five large populations European, East Asian, West African, South Asian ancestry and Americas with a total of 26 small populations as in the table below:

Large Pop	Small Pop	Sample size
Americas	CLM	65
Americas	MXL	59
Americas	PEL	60
Americas	PUR	74
East Asian	CDX	88
East Asian	CHB	83
East Asian	CHS	104
East Asian	JPT	82
East Asian	KHV	78
European	CEU	96
European	FIN	78
European	GBR	77
European	IBS	77
European	TSI	100
South Asian	BEB	50
South Asian	GIH	81
South Asian	ITU	39
South Asian	PJL	37
South Asian	STU	49
African	ACB	74
African	ASW	50
African	ESN	64
African	GWD	105
African	LWK	90
African	MSL	68
African	YRI	89

3.1 Measuring CCL3L1 CN

The *ccl3l1data* data includes 1917 samples downloaded from the 1000 Genomes Project in October 2012 and March 2013, their corresponding populations, segmentation scores and CNs. The segmentation scores were obtained by using the function *segmentSamplesUsingPopInformation* for a 1Mb region (chr17:33670000-34670000) with 500bp-constant windows.

```
##Load data into R:
data(ccl3l1data)
head(ccl3l1data)

##      Name Pop      SS CN
## 1 HG00096 GBR -0.6932975  1
## 2 HG00100 GBR -0.2308212  2
## 3 HG00103 GBR -0.3511010  2
## 4 HG00106 GBR -0.8012343  1
## 5 HG00108 GBR -0.4119802  2
## 6 HG00111 GBR -0.2879718  2

hist(ccl3l1data$SS, 100)
```

As can be seen in Figure 5, the data is multimodal and there are not clear clusters on the right. Therefore, we can use a single population which has clear clusters to obtain prior information for the clustering process into CN groups. Here, we used the large European-ancestry population to obtain prior information.

```
xyEuro <- ccl3l1data[grep("CEU|TSI|IBS|GBR|FIN", ccl3l1data[, 2]), ]
yEuro <- xyEuro[, 3]
names(yEuro) <- xyEuro[, 1]
hist(yEuro, 100, xlab = '', main = '')
```

As can be seen from Figure 6, the European-ancestry data exhibit relatively clear clusters, allowing us to classify the samples into different CN groups.

Note: if we use the option *autoDetermineGroup* = *TRUE* in the function *groupC-*



Figure 5: CCL3L1 segmentation score.

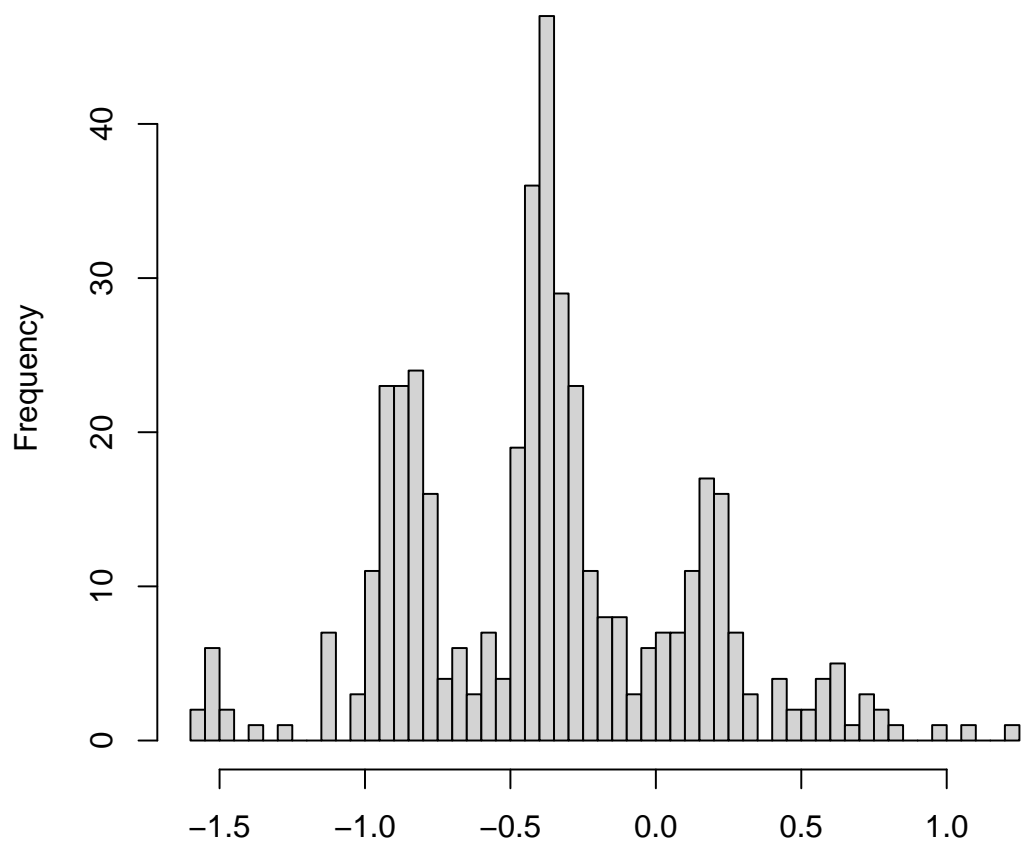


Figure 6: European-ancestry segmentation score.

NVs then the Bayesian information criterion (BIC) will be used to choose a suitable number of components (See [Schwarz \(1978\)](#)).

```
##Clustering European segmentation
##scores into group: 5 groups were chosen

objectClusterEuroCCL3L1 <- new("clusteringCNVs", x = yEuro, k = 5)

europeanCCL3L1Groups <- groupCNVs(Object = objectClusterEuroCCL3L1)

## =====
## 7 iterations
## 5 components with Unequal variances
## m:  -1.51402924373199-0.871863641754189-0.3596050190158570.1442265197789390.611056959363024
## p:  0.02569476547601550.281378789950390.4465042675825080.1787783814870730.0676437955040135
## sigma:  0.04985566125899970.1132002471605970.1008864849220440.1154674632491250.230716323036497
## =====
```

Next, we use these results to infer *CCL3L1* CN in all populations. The following code collects information about the means, standard deviations and proportions of the mixture components from the European population.

```
##Means
lambda0 <- as.numeric(europeanCCL3L1Groups$m)

##SD
sdEM <- as.numeric(europeanCCL3L1Groups$sigma)

##Proportions
pEM <- as.numeric(europeanCCL3L1Groups$p)
```

Take a look these results:

```
lambda0

## [1] -1.5140292 -0.8718636 -0.3596050  0.1442265  0.6110570

sdEM

## [1] 0.04985566 0.11320025 0.10088648 0.11546746 0.23071632
```



Figure 7: Clustering results of European-ancestry sample sets.

```

pEM

## [1] 0.02569477 0.28137879 0.44650427 0.17877838 0.06764380

###Calculate the distances between groups
for (ii in 2:5){print(lambda0[ii] - lambda0[ii-1])}

## [1] 0.6421656
## [1] 0.5122586
## [1] 0.5038315
## [1] 0.4668304

###All segmentation scores
ccl3l1X <- ccl3l1data$SS
names(ccl3l1X) <- as.character(ccl3l1data$Name)
range(ccl3l1X)

## [1] -1.675230 4.257877

```

The information above is then used by the function *groupBayesianCNVs* to cluster the segmentation scores for the combined set of European-ancestry cohorts into different CN groups. There is a high value in the full SS set (Figure 5), which we eliminate in the following steps by setting *rightLimit=4* so that it is automatically allocated into the highest CN group. Using the other values, combined with locus-specific information from the literature, we set the number of groups to be 10. All prior information was used in our work to obtain CN for CCL3L1 gene (the four group).

```

##Set prior information:
#prior for the sd of the means of groups:
#5 was set for the third group = 2 CN
sd <- c(1, 1, 5, 1, 1)
ccl3l1X <- sort(ccl3l1X)

###Data
xData <- ccl3l1X

###Number of groups

```

```

nGroups <- 10

###prior for means of groups

lambda0 <- lambda0

###Prior for mixing proportions

alpha0 <- c(3, 29, 44, 18, 7, 5, rep(2, nGroups -length(pEM) -1))

##Prior for the distances between groups

distanceBetweenGroups = 0.485

sdEM = sdEM

```

The final (5th) group of the results of the European-ancestry sample sets has a large standard deviation as a result of the scattering of values on the right (Figure 6), therefore, we can set this value to equal the standard deviation of the fourth group to avoid an overly wide mixture component.

```

##Adjust standard deviation for the fifth group

sdEM[5] <- sdEM[4]

```

Run the *groupBayesianCNVs* function to obtain CN groups.

```

set.seed(123)

groupCCL3L1allPops <- groupBayesianCNVs(xData = xData, nGroups = nGroups,
                                       lambda0 = lambda0,
                                       sd0 = sdEM, alpha0 = alpha0,
                                       distanceBetweenGroups = distanceBetweenGroups,
                                       sd0ftau = sd,
                                       rightLimit = 4)

```

These results would be similar to the results of the fourth column in the data of the package.

3.2 Identifying tag-SNPs for CCL3L1 CNVs

We can obtain tag-SNPs/INDELs for multiple populations simultaneously. Below, we reuse the CCL3L1 data to obtain tag-SNPs for some populations.

```
rownames(ccl3l1data) <- ccl3l1data[, 1]
```

Load VCF file into R and choose populations which we would like to find tagSNPs/INDELs.

```
##Obtain vcf-file information in CNVrd2
vcfFileCCL3L1 <- system.file(package="CNVrd2", "extdata",
                              "chr17.34800000.34830000.vcf.gz")

##Set populations we would like to identify tagSNPs
allPops <- c("TSI", "CEU", "GBR", "FIN", "IBS")

##Identify tag SNPs/INDELs
tagSNPandINDELofCCL3L1 <- calculateLDSNPandCNV(sampleCNV = ccl3l1data,
                                              vcfFile = vcfFileCCL3L1, cnvColumn = 4,
                                              population = allPops, popColumn = 2,
                                              nChunkForVcf = 5, chr = "17",
                                              st = 34800000, en = 34830000 )

## Reading the VCF file 17:34800000-34830000 with 5 blocks each
## VCF file: 34800000 to 34807500
## VCF file: 34807501 to 34815000
## VCF file: 34815001 to 34822500
## VCF file: 34822501 to 34830000

## Calculating p and r2 values for TSI population.
## Calculating p and r2 values for CEU population.
## Calculating p and r2 values for GBR population.
## Calculating p and r2 values for FIN population.
## Calculating p and r2 values for IBS population.
```

Take a quick look some significant results (multiple populations: the return value of *calculateLDSNPandCNV* is a list of populations).

```
lapply(tagSNPandINDELofCCL3L1, head)

## [[1]]
##          OCN_(n=5) 1CN_(n=32) 2CN_(n=41) 3CN_(n=16) 4CN_(n=4)
## rs8064426         80      87.50    12.20    18.75      0
## rs113877493        60      71.88     2.44     6.25      0
## rs11316723         80      84.38    12.20    18.75      0
## rs8072769          80      78.12    12.20    12.50      0
## rs9911791          80      87.50    17.07    18.75     25
## rs113435750        80      78.12    12.20    12.50     25
```

```

##          p.values      r p.valuesAdjusted  r2 POP
## rs8064426  1.1e-11 -0.64          2.0e-09 0.40 TSI
## rs113877493 4.0e-11 -0.63          3.7e-09 0.39 TSI
## rs11316723  8.7e-11 -0.62          5.4e-09 0.38 TSI
## rs8072769  1.2e-09 -0.61          4.6e-08 0.37 TSI
## rs9911791  5.2e-10 -0.58          2.4e-08 0.34 TSI
## rs113435750 3.0e-09 -0.57          7.9e-08 0.33 TSI
##
## [[2]]
##          OCN_(n=2) 1CN_(n=20) 2CN_(n=39) 3CN_(n=14) 4CN_(n=3)
## rs8072769          0          75          2.56          0.00          0
## rs113435750          0          75          5.13          0.00          0
## rs138153523          0          75          5.13          0.00          0
## rs11316723          0          80          5.13          7.14          0
## rs8064426          0          80          5.13          7.14          0
## rs9911791          0          85          10.26          7.14          0
##
##          5CN_(n=1) p.values      r p.valuesAdjusted  r2 POP
## rs8072769          0 7.9e-10 -0.61          1.5e-07 0.37 CEU
## rs113435750          0 6.9e-09 -0.60          2.6e-07 0.35 CEU
## rs138153523          0 6.9e-09 -0.60          2.6e-07 0.35 CEU
## rs11316723          0 5.1e-09 -0.57          2.6e-07 0.33 CEU
## rs8064426          0 5.1e-09 -0.57          2.6e-07 0.33 CEU
## rs9911791          0 1.9e-08 -0.58          5.8e-07 0.33 CEU
##
## [[3]]
##          1CN_(n=18) 2CN_(n=37) 3CN_(n=11) 4CN_(n=4) p.values      r
## rs11316723          72.22          2.70          0          0 1.2e-08 -0.65
## rs8064426          72.22          2.70          0          0 1.2e-08 -0.65
## rs9911791          72.22          8.11          0          0 5.5e-07 -0.62
## rs113877493          50.00          0.00          0          0 3.8e-06 -0.54
## rs8072769          50.00          2.70          0          0 5.1e-05 -0.52
## rs113435750          50.00          2.70          0          0 5.1e-05 -0.52
##
##          p.valuesAdjusted  r2 POP
## rs11316723          1.1e-06 0.43 GBR
## rs8064426          1.1e-06 0.43 GBR
## rs9911791          3.4e-05 0.38 GBR
## rs113877493          1.8e-04 0.29 GBR
## rs8072769          1.6e-03 0.27 GBR
## rs113435750          1.6e-03 0.27 GBR
##
## [[4]]
##          OCN_(n=2) 1CN_(n=19) 2CN_(n=28) 3CN_(n=18) 4CN_(n=6)
## rs113877493          100          73.88          7.14          0.00          0
## rs6607368          100          84.21          21.43          5.56          0
## rs8067765          100          63.16          7.14          0.00          0
## rs60952743          100          63.16          7.14          0.00          0
## rs8070238          100          63.16          7.14          0.00          0
## rs8072238          100          63.16          7.14          0.00          0
##
##          5CN_(n=1) p.values      r p.valuesAdjusted  r2 POP
## rs113877493          0 5.0e-09 -0.67          9.2e-07 0.45 FIN
## rs6607368          0 1.2e-07 -0.66          4.8e-06 0.43 FIN
## rs8067765          0 2.6e-07 -0.61          4.8e-06 0.38 FIN
## rs60952743          0 2.6e-07 -0.61          4.8e-06 0.38 FIN
## rs8070238          0 2.6e-07 -0.61          4.8e-06 0.38 FIN
## rs8072238          0 2.6e-07 -0.61          4.8e-06 0.38 FIN
##
## [[5]]
##          1CN_(n=2) 2CN_(n=1) 3CN_(n=1) 4CN_(n=2) p.values      r
## rs4796217          100          100          100          0 2.0e-01 -0.85
## rs28856610          0          0          0          100 2.0e-01 0.85
## rs11651338          100          0          100          0 2.0e-01 -0.70
## rs138347191          0          0          0          50 1.0e+00 0.54
## rs4796216          50          100          100          100 1.0e+00 0.54
## rs60520102          0          0          0          50 1.0e+00 0.54

```



```
##
## rs4796217      1e+00 0.73 IBS
## rs28856610     1e+00 0.73 IBS
## rs11651338     1e+00 0.49 IBS
## rs138347191    1e+00 0.29 IBS
## rs4796216      1e+00 0.29 IBS
## rs60520102     1e+00 0.29 IBS
```

The output above provides evidence that *rs113877493* may be a tagSNP for *CCL3L1* deletions in the FIN ($p = 1.1\text{e-}07$, $r^2 = 0.44$) and GBR ($p = 2.6\text{e-}04$, $r^2 = 0.27$) populations.

```
##Notice: these results are in a list
for (ii in 1:length(allPops))
  write.table(tagSNPandINDELofCCL3L1[[ii]],
             paste("TagSNPforPop", ii, ".csv", sep = ""),
             quote = FALSE, sep = ",")
```

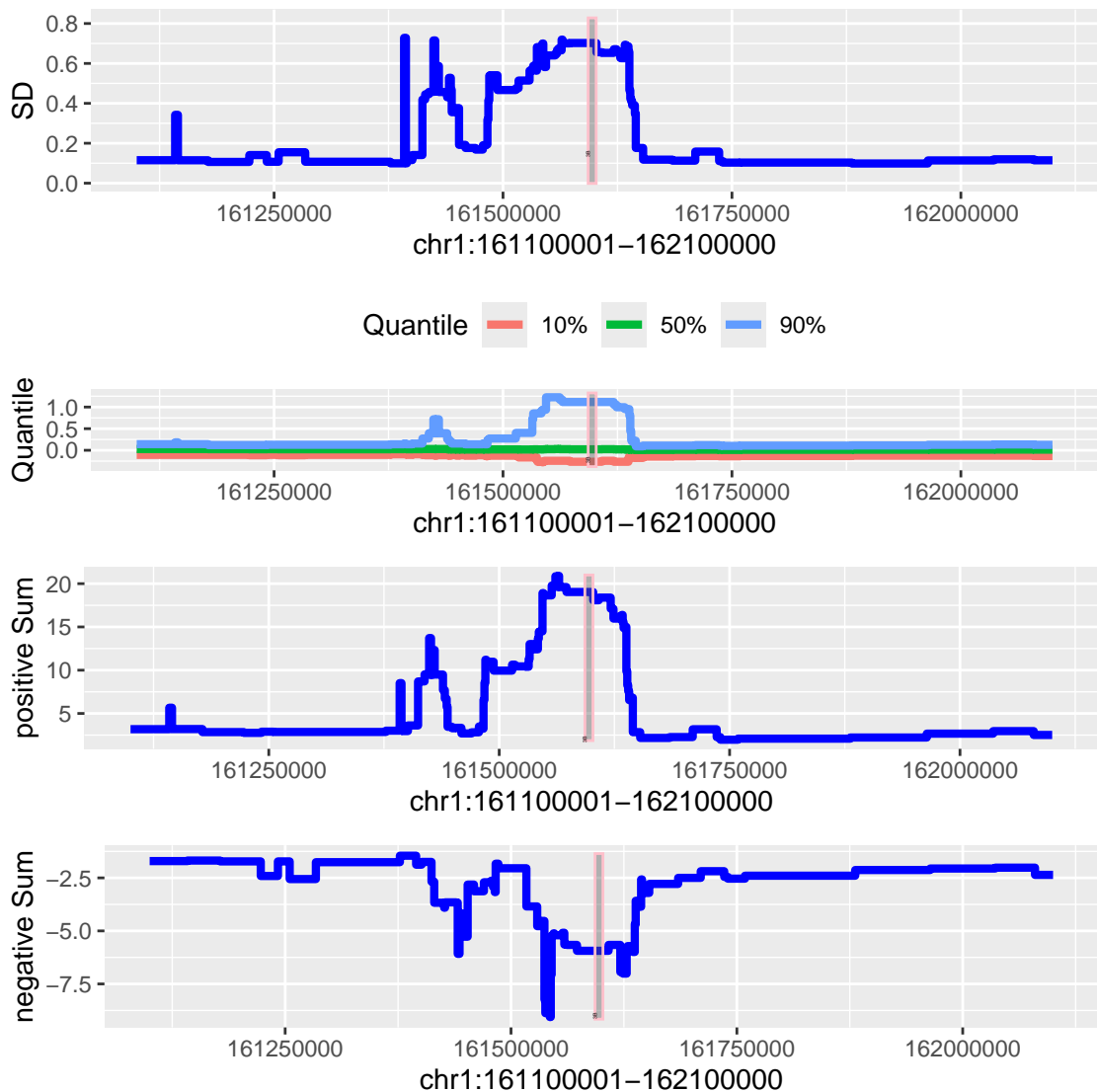
Note: All TagSNP results for DEFB103A, CCL3L1 and FCGR3B CNV will be uploaded on <https://github.com/hoangtn/CNVrd2>

4 Identifying polymorphic regions

CNVrd2 can also be used to identify CN polymorphic regions and the putative boundaries of these regions. We reuse the data from the FCGR3 locus to investigate the polymorphic region around the two genes.

```
fcgr3PolymorphicRegion <- identifyPolymorphicRegion(Object = objectCNVrd2,
                                                    segmentObject = resultSegment,
                                                    plotLegend = FALSE)

## Calculate segmentation scores for polymorphic regions
```



To plot a small region around the gene, we use the function *plotPolymorphicRegion*. Users can change *typePlot* to only plot SDs or percentiles.

```
plotPolymorphicRegion(Object = objectCNVrd2, polymorphicRegionObject = fcgr3PolymorphicRegion,
  xlim = c(161300000, 161800000), drawThresholds = TRUE,
  typePlot = "SD")
```

```
## IRanges object with 5 ranges and 0 metadata columns:
```

```
##      start      end      width
##      <integer> <integer> <integer>
## [1] 161392001 161394000      2000
## [2] 161424001 161426000      2000
```

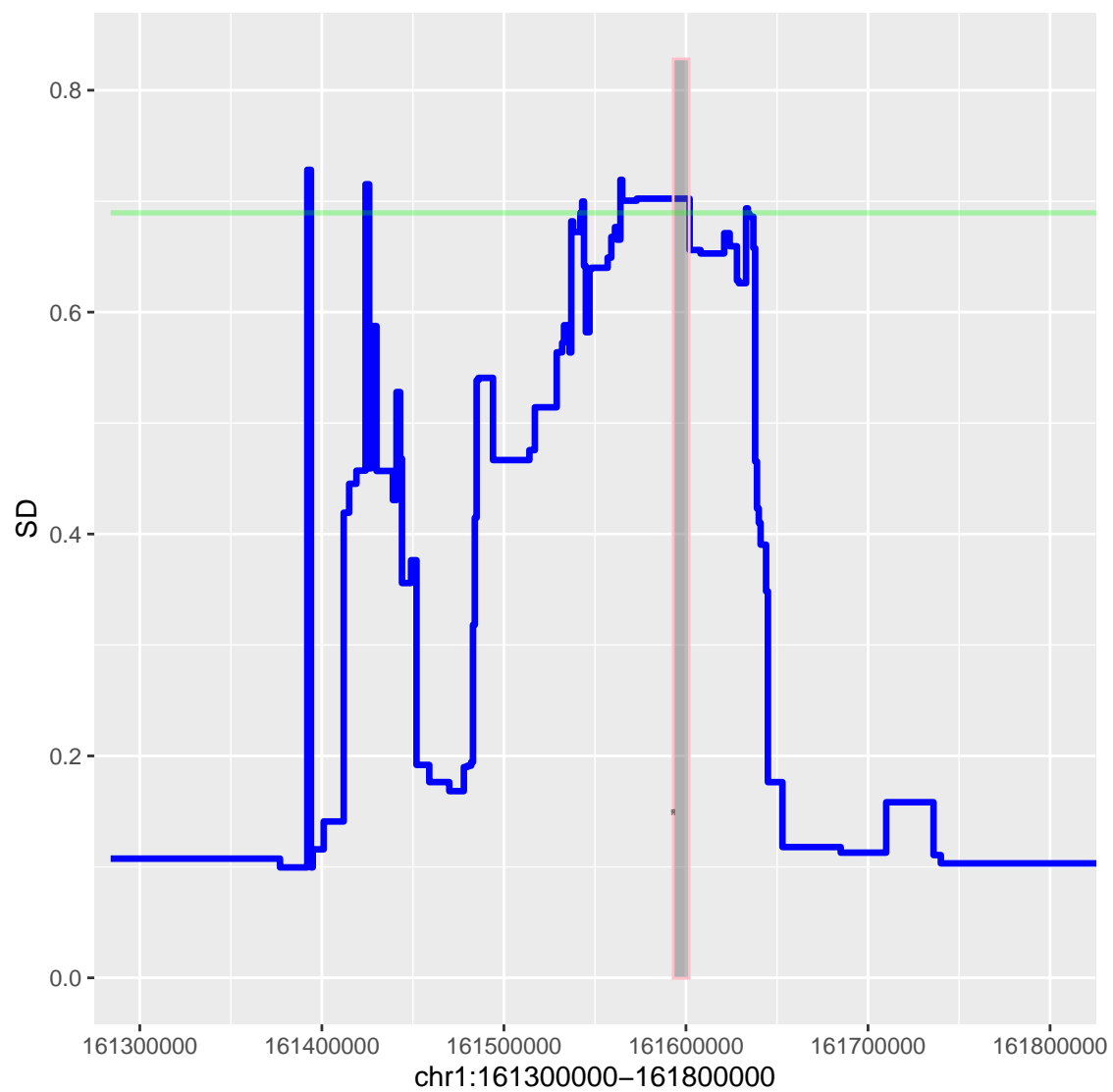


Figure 8: CN polymorphic region at FCGR3 locus, represented by quantiles of the distribution of segmentation scores across samples.

```
## [3] 161542001 161544000      2000
## [4] 161564001 161602000     38000
## [5] 161633001 161634000      1000
```

```
plotPolymorphicRegion(Object = objectCNVrd2, polymorphicRegionObject = fcgr3PolymorphicRegion,
                      xlim = c(161300000, 161800000), sdThreshold = 0.05, drawThresholds = TRUE,
                      typePlot = "SD")
```

```
## IRanges object with 4 ranges and 0 metadata columns:
```

```
##           start           end           width
##      <integer> <integer> <integer>
## [1] 161392001 161394000           2000
## [2] 161424001 161426000           2000
## [3] 161564001 161565000           1000
## [4] 161573001 161602000          29000
```

Here, we are using standard deviations to identify polymorphic regions. Therefore, the putative boundaries of these regions rely on the parameter *sdThreshold*. Alternatively, users can use different percentiles to identify these regions. In that case, two parameters *quantileValue* and *thresholdForPolymorphicRegions* can be used to adjust the boundaries of regions. For example, here we can set small *sdThreshold* values to obtain only high-polymorphic regions (e.g., *CCL3L1*), but it can omit some medium-polymorphic regions (e.g., *FCGR3A/3B*). Figure 8 and 9 depicts two different thresholds resulting in different polymorphic regions.

In the function *identifyPolymorphicRegion*, if we would like to obtain only polymorphic regions which differentiate between populations (e.g., to detect evidence of selection) then we can use the option *VstTest=TRUE*. This option will calculate the Vst statistics (Redon *et al.*, 2006). Users have to supply a vector which includes population information in *popName*. The returned putative boundaries will be the intersection of polymorphic regions and regions having $\max Vst \geq \text{thresholdVST}$.

5 Note

If we use the option *entireGene = TRUE* in the step *segmentation* then the pipeline will not refine the segmentation results (the results will be the same as the pipeline used in Nguyen *et al.* (2013)).

6 Session information

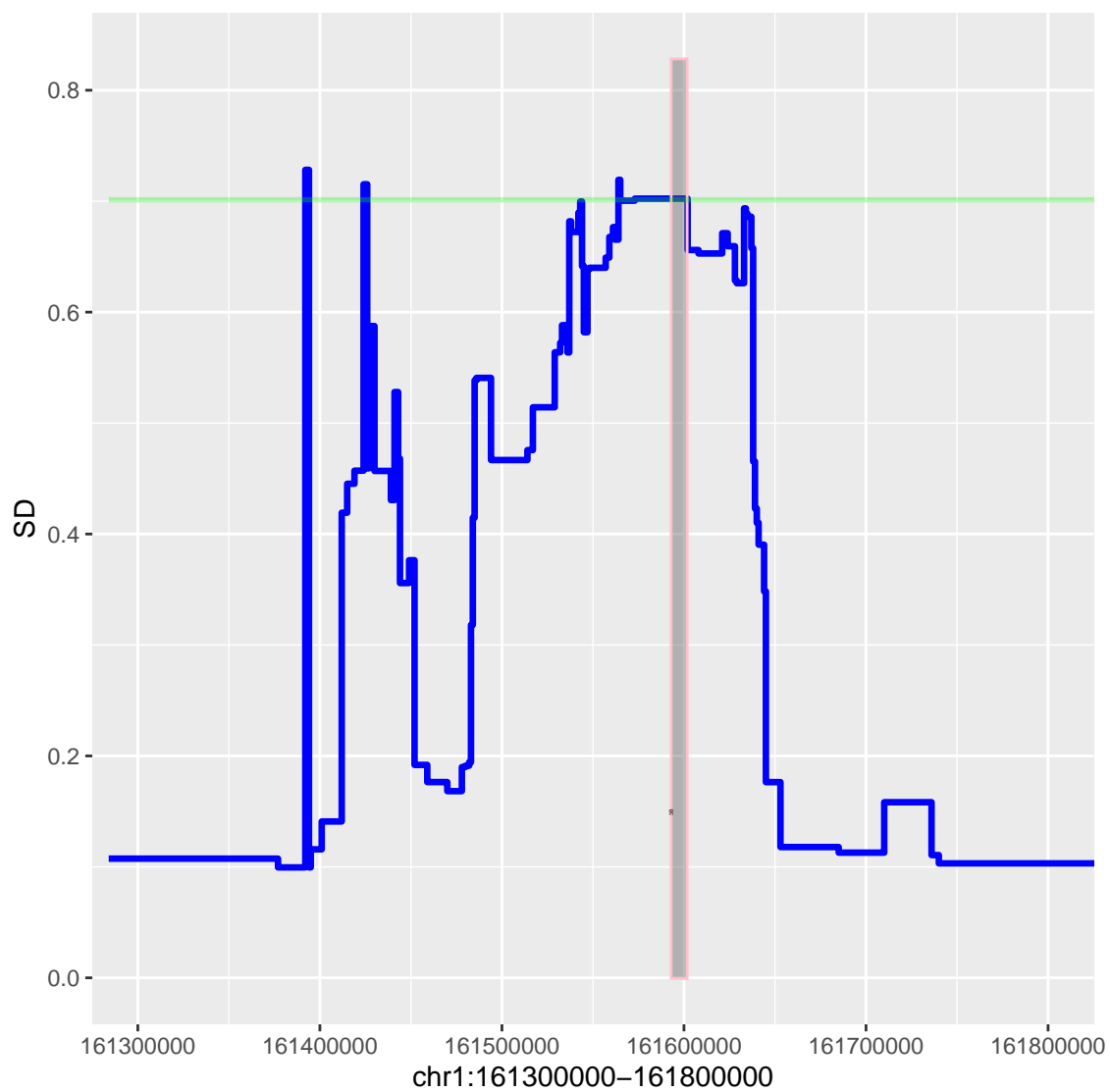


Figure 9: CN polymorphic region at FCGR3 locus, represented by quantiles of the distribution of segmentation scores across samples.

```

sessionInfo()

## R version 4.5.0 RC (2025-04-04 r88126 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows Server 2022 x64 (build 20348)
##
## Matrix products: default
##   LAPACK version 3.12.1
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] parallel stats4 stats graphics grDevices utils
## [7] datasets methods base
##
## other attached packages:
## [1] CNVrd2_1.46.0 gridExtra_2.3
## [3] ggplot2_3.5.2 rjags_4-17
## [5] coda_0.19-4.1 VariantAnnotation_1.54.0
## [7] Rsamtools_2.24.0 Biostrings_2.76.0
## [9] XVector_0.48.0 SummarizedExperiment_1.38.0
## [11] Biobase_2.68.0 GenomicRanges_1.60.0
## [13] GenomeInfoDb_1.44.0 IRanges_2.42.0
## [15] S4Vectors_0.46.0 MatrixGenerics_1.20.0
## [17] matrixStats_1.5.0 BiocGenerics_0.54.0
## [19] generics_0.1.3 knitr_1.50
##
## loaded via a namespace (and not attached):

```

```
## [1] KEGGREST_1.48.0      gtable_0.3.6
## [3] rjson_0.2.23          xfun_0.52
## [5] lattice_0.22-7        vctrs_0.6.5
## [7] tools_4.5.0           bitops_1.0-9
## [9] curl_6.2.2            tibble_3.2.1
## [11] AnnotationDbi_1.70.0  RSQLite_2.3.9
## [13] highr_0.11            blob_1.2.4
## [15] pkgconfig_2.0.3       Matrix_1.7-3
## [17] BSgenome_1.76.0       lifecycle_1.0.4
## [19] GenomeInfoDbData_1.2.14 farver_2.1.2
## [21] compiler_4.5.0        munsell_0.5.1
## [23] codetools_0.2-20      RCurl_1.98-1.17
## [25] yaml_2.3.10           pillar_1.10.2
## [27] crayon_1.5.3          BiocParallel_1.42.0
## [29] DelayedArray_0.34.0   cachem_1.1.0
## [31] abind_1.4-8           tidyselect_1.2.1
## [33] dplyr_1.1.4           restfulr_0.0.15
## [35] labeling_0.4.3        fastmap_1.2.0
## [37] grid_4.5.0            colorspace_2.1-1
## [39] cli_3.6.4             SparseArray_1.8.0
## [41] magrittr_2.0.3        S4Arrays_1.8.0
## [43] GenomicFeatures_1.60.0 XML_3.99-0.18
## [45] withr_3.0.2           UCSC.utils_1.4.0
## [47] scales_1.3.0          bit64_4.6.0-1
## [49] httr_1.4.7            bit_4.6.0
## [51] png_0.1-8             DNACopy_1.82.0
## [53] memoise_2.0.1         evaluate_1.0.3
## [55] BiocIO_1.18.0         rtracklayer_1.68.0
## [57] rlang_1.1.6           glue_1.8.0
## [59] DBI_1.2.3             jsonlite_2.0.0
## [61] R6_2.6.1              GenomicAlignments_1.44.0
```

References

- Danecek P, Auton A, Abecasis G, Albers C, Banks E, DePristo M, Handsaker R, Lunter G, Marth G, Sherry S, *et al.* (2011). “The variant call format and VCFtools.” *Bioinformatics*, **27**(15), 2156–2158.
- Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, *et al.* (2009). “The sequence alignment/map format and SAMtools.” *Bioinformatics*, **25**(16), 2078–2079.
- Nguyen HT, Merriman TR, Black MA (2013). “CNVrd, a Read-Depth Algorithm for Assigning Copy-Number at the FCGR Locus: Population-Specific Tagging of Copy Number Variation at FCGR3B.” *PLOS ONE*, **8**(4), e63219.
- Nguyen HT, Merriman TR, Black MA (2014). “The CNVrd2 package: measurement of copy number at complex loci using high-throughput sequencing data.” *Frontiers in Genetics*, **5**, 248.
- Plummer M (2013). *rjags: Bayesian graphical models using MCMC*. R package version 3-10, URL <http://CRAN.R-project.org/package=rjags>.
- Redon R, Ishikawa S, Fitch KR, Feuk L, Perry GH, Andrews TD, Fiegler H, Shapero MH, Carson AR, Chen W, *et al.* (2006). “Global variation in copy number in the human genome.” *nature*, **444**(7118), 444–454.
- Schwarz G (1978). “Estimating the dimension of a model.” *The annals of statistics*, **6**(2), 461–464.
- Venkatraman E, Olshen AB (2007). “A faster circular binary segmentation algorithm for the analysis of array CGH data.” *Bioinformatics*, **23**(6), 657–663.