

The chromPlot user's guide

Karen Y. Oróstica and Ricardo A. Verdugo

April 30, 2024

Contents

1	Introduction	4
2	Creating a plot with genomic coordinates	4
3	Input data	7
4	Types of data visualization	10
4.1	Chromosomes banding	10
4.1.1	Plotting G banding	10
4.1.2	Genomic elements	13
4.1.3	Assigning different colors	15
4.1.4	Grouping elements by category	18
4.1.5	Synteny	20
4.2	Histograms	22

4.2.1	Single histogram	22
4.2.2	Stacked histograms: multiple files	24
4.2.3	Stacked histograms: single file	29
4.3	XY plots	31
4.3.1	Using points	32
4.3.2	Using connected lines	33
4.3.3	Coloring by datapoints exceeding a threshold	34
4.3.4	Plotting LOD curves	37
4.3.5	Plotting a map with IDs	38
4.4	Segments	40
4.4.1	Large stacked segments	40
4.4.2	Large stacked segments grouped by two categories	43
4.4.3	Large non-overlapping segments	47
4.5	Multiple data types	49
5	Graphics settings	52
5.1	Choosing side	52
5.2	Choosing colors	54
5.3	Placement of legends	54
6	Acknowledgments	56

1 Introduction

Visualization is an important step in data analysis workflows for genomic data. Here, we introduce the use of **chromPlot**, an R package for global visualization of genome-wide data. **chromPlot** is suitable for any organism with linear chromosomes. Data is visualized along chromosomes in a variety of formats such as segments, histograms, points and lines. One plot may include multiple tracks of data, which can be placed inside or on either side of the chromosome body representation.

The package has proven to be useful in a variety of applications, for instance, detecting chromosomal clustering of differentially expressed genes, combining diverse information such as genetic linkage to phenotypes and gene expression, quality controlling genome resequencing experiments, visualizing results from genome-wide scans for positive selection, synteny between two species, among others.

2 Creating a plot with genomic coordinates

The **gaps** argument is used to tell **chromPlot** what system of coordinates to use. The information is provided as a table following the format for the ‘Gap’ track in the Table Browser of the UCSC website¹. From this table, **chromPlot** extracts the number of chromosomes, chromosomes names and lengths, and the position of centromeres (shown as solid circles). The tables for the latest genome build of human and mouse are provided with package (**hg_gap** and **mm10_gap**) and are loaded by **data()**. The user can use tables downloaded from the UCSC Table Browser for other genomes. If no data is provided to **gaps**, plotting is still possible as long as one of **annot1**, **bands** or **org** arguments is provided. The information will be taken from those objects, in that preference order, except for centromeres which will not be plotted.

¹<https://genome.ucsc.edu/>

In this example, we will plot the chromosomes in the hg19 human genome. `chromPlot` returns some messages when doing calculations. Here, it just retrieves the number of bases in each chromosomes. Messages will be omitted in next examples.

```
> library("chromPlot")
```

```
> data(hg_gap)
```

```
> head(hg_gap)
```

	Chrom	Start	End	Name
1	1	124535434	142535434	heterochromatin
2	1	121535434	124535434	centromere
3	1	3845268	3995268	contig
4	1	13219912	13319912	contig
5	1	17125658	17175658	clone
6	1	29878082	30028082	contig

```
> chromPlot(gaps=hg_gap)
```

Chrom 1 : 249250621 bp

Chrom 2 : 243199373 bp

Chrom 3 : 198022430 bp

Chrom 4 : 191154276 bp

Chrom 5 : 180915260 bp

Chrom 6 : 171115067 bp

Chrom 7 : 159138663 bp

Chrom 8 : 146364022 bp

Chrom 9 : 141213431 bp

Chrom 10 : 135534747 bp

Chrom 11 : 135006516 bp

Chrom 12 : 133851895 bp

Chrom 13 : 115169878 bp

Chrom 14 : 107349540 bp

Chrom 15 : 102531392 bp

Chrom 16 : 90354753 bp

Chrom 17 : 79759049 bp

Chrom 18 : 78077248 bp

Chrom 19 : 59128983 bp

Chrom 20 : 63025520 bp
Chrom 21 : 48129895 bp
Chrom 22 : 51304566 bp
Chrom X : 155270560 bp
Chrom Y : 59373566 bp



3 Input data

`chromPlot` has 8 arguments that can take objects with genomic data: (`annot1`, `annot2`, `annot3`, `annot4`, `segment`, `segment2`, `stat` and `stat2`). Data provided to these arguments are internally converted to data tracks that can be plotted. These arguments take their input in any of these formats:

1. A string with a filename or URL
2. A data frame

3. A GRanges object (`GenomicRanges` package)

Additionally, the user may obtain a list of all ensemble genes by providing and organism name to the `org` argument (ignored if data is provided to `annot1`).

The data provided as objects of class `data.frame` must follow the BED format in order to be used as tracks by `chromPlot`². However, as opposed to the files in BED format, track must have column names. The columns `Chrom` (character class), `Start` (integer class) and `End` (integer class) are mandatory. `chromPlot` can work with categorical or quantitative data. The categorical data must have a column called `Group` (character class), which represents the categorical variable to classify each genomic element. In the case of quantitative data, the user must indicate the column name with the score when calling `chromPlot()` by setting the `statCol` parameter.

Examples of different data tables will be shown throughout this tutorial. All data used in this vignette are included in `chromPlot` (`inst/extdata` folder). In order to keep the package size small, we have included only a few chromosomes in each file. We use mostly public data obtained from the UCSC Genome Browser³ or from The 1000 Genomes Selection Browser 1.0⁴, i.e. the `iHS`, `Fst` and `xpehh` tables shown below.

In the following example code, an annotation package from Bioconductor to display the density of all transcripts in the genome. We load a `TxDb` object (inherit class from `AnnotationDb`) with all known gene transcripts in the hg19 human genome. We extract the transcripts for this gene definition and plot them genome-wide. The transcripts object (`txgr`) has `GRanges` class, from *GenomicRanges* package. The *GenomeFeatures* package is required to extract the transcripts from the annotation object.

```
> library("TxDb.Hsapiens.UCSC.hg19.knownGene")
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> library(GenomeFeatures)
```

²<https://genome.ucsc.edu/FAQ/FAQformat.html#format1>

³<http://genome.ucsc.edu/>

⁴<http://hsb.upf.edu/>


```
> txgr <- transcripts(txdb)
> txgr
```

GRanges object with 82960 ranges and 2 metadata columns:

	seqnames	ranges	strand	tx_id	tx_name
	<Rle>	<IRanges>	<Rle>	<integer>	<character>
[1]	chr1	11874-14409	+	1	uc001aaa.3
[2]	chr1	11874-14409	+	2	uc010nxq.1
[3]	chr1	11874-14409	+	3	uc010nxr.1
[4]	chr1	69091-70008	+	4	uc001aal.1
[5]	chr1	321084-321115	+	5	uc001aaq.2
...
[82956]	chrUn_gl000237	1-2686	-	82956	uc011mgu.1
[82957]	chrUn_gl000241	20433-36875	-	82957	uc011mgv.2
[82958]	chrUn_gl000243	11501-11530	+	82958	uc011mgw.1
[82959]	chrUn_gl000243	13608-13637	+	82959	uc022brq.1
[82960]	chrUn_gl000247	5787-5816	-	82960	uc022brr.1

seqinfo: 93 sequences (1 circular) from hg19 genome

```
> chromPlot(gaps=hg_gap, annot1=txgr)
```



4 Types of data visualization

4.1 Chromosomes banding

4.1.1 Plotting G banding

The `chromPlot` package can create idiograms by providing a 'cytoBandIdeo' table taken from the Table Browser at the UCSC Genome Browser website. These tables are provided with the package for human and mouse (`hg_cytoBandIdeo` and `mm10_cytoBandIdeo`).

In the next code, we show how to obtain an idiogram with a subset of chromosomes for human:

```
> data(hg_cytoBandIdeo)
> head(hg_cytoBandIdeo)
```

	Chrom	Start	End	Name	gieStain
1	1	0	2300000	p36.33	gneg
2	1	2300000	5400000	p36.32	gpos25
3	1	5400000	7200000	p36.31	gneg
4	1	7200000	9200000	p36.23	gpos25
5	1	9200000	12700000	p36.22	gneg
6	1	12700000	16200000	p36.21	gpos50

You can choose chromosomes using `chr` parameter, which receives a vector with the name of the chromosomes.

```
> chromPlot(bands=hg_cytoBandIdeo, gaps=hg_gap, chr=c("1", "2", "3", "4", "5",  
+ "6"), figCols=6)
```



4.1.2 Genomic elements

`chromplot` can plot the location of genomic elements in the chromosomal body. For this example, we will use a table of refSeq genes taken from the UCSC Genome Browser. The file included in the package contains only chromosomes 19 to 21 to keep the package's size small.

```
> data_file1      <- system.file("extdata", "hg19_refGeneChr19-21.txt",
+ package = "chromPlot")
> refGeneHg       <- read.table(data_file1, sep="\t", header=TRUE,
+ stringsAsFactors=FALSE)
> refGeneHg$Colors <- "red"
> head(refGeneHg)
```

	Chrom	Start	End	Name	Colors
1	chr19	41937222	41945843	NM_018035	red
2	chr19	41937222	41945481	NM_001167867	red
3	chr19	41937222	41945843	NM_001167869	red
4	chr19	41937222	41945481	NM_001167868	red
5	chr19	58694355	58724928	NM_016324	red
6	chr19	50321535	50340237	NM_030973	red

```
> chromPlot(gaps=hg_gap, bands=refGeneHg, chr=c(19, 20, 21), figCols=3)
```



4.1.3 Assigning different colors

It is possible to use different colors for each genomic element. However, you should keep in mind that humans can only distinguish a limited number of colors in a plot. Therefore, for continuous variables, it is useful to create bins of data and assign colors to each bin.

```
> data_file2 <- system.file("extdata", "Fst_CEU-YRI-W200Chr19-21.bed", package
+ = "chromPlot")
> fst <- read.table(data_file2, sep="\t", stringsAsFactors=FALSE, header=TRUE)
> head(fst)
```

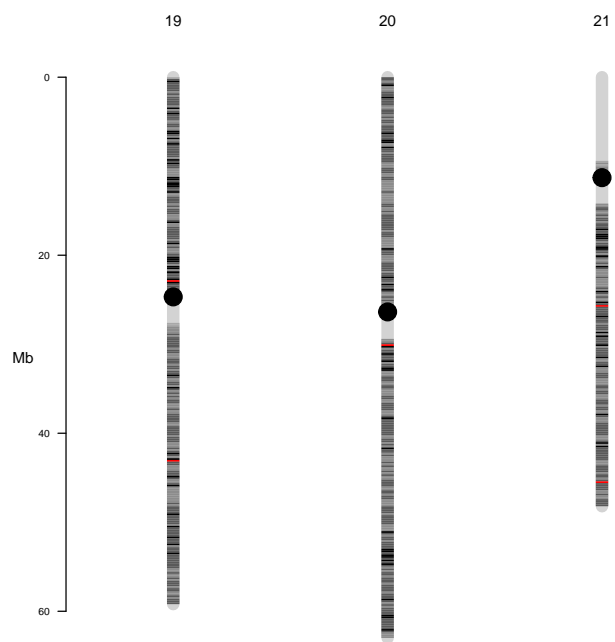
	Chrom	Start	End	win.n	win.FST	win.max
1	19	10000001	10200000	1788	0.05867522	0.6810
2	19	1000001	1200000	2022	0.05720885	0.6590
3	19	10200001	10400000	1425	0.03499754	0.3584
4	19	10400001	10600000	1377	0.04107502	0.4172
5	19	10600001	10800000	1435	0.04324279	0.3513
6	19	10800001	11000000	1289	0.03154461	0.5857

```
> fst$Colors <-
+ ifelse(fst$win.FST >= 0      & fst$win.FST < 0.025, "gray66",
+ ifelse(fst$win.FST >= 0.025 & fst$win.FST < 0.05,  "grey55",
+ ifelse(fst$win.FST >= 0.05  & fst$win.FST < 0.075, "grey35",
+ ifelse(fst$win.FST >= 0.075 & fst$win.FST < 0.1,   "black",
+ ifelse(fst$win.FST >= 0.1   & fst$win.FST < 1,     "red", "red")))))
> head(fst)
```

	Chrom	Start	End	win.n	win.FST	win.max	Colors
1	19	10000001	10200000	1788	0.05867522	0.6810	grey35
2	19	1000001	1200000	2022	0.05720885	0.6590	grey35
3	19	10200001	10400000	1425	0.03499754	0.3584	grey55
4	19	10400001	10600000	1377	0.04107502	0.4172	grey55

5	19	10600001	10800000	1435	0.04324279	0.3513	grey55
6	19	10800001	11000000	1289	0.03154461	0.5857	grey55


```
> chromPlot(gaps=hg_gap, chr=c(19, 20, 21), bands=fst, figCols=3)
```



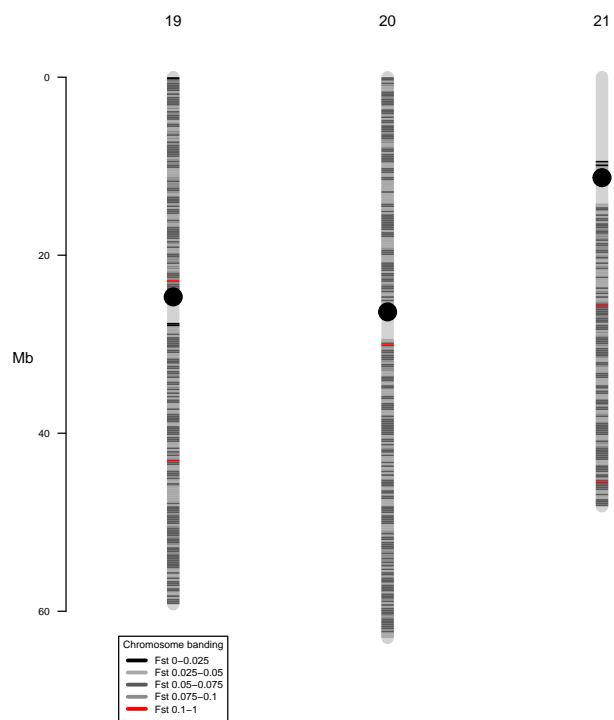
4.1.4 Grouping elements by category

If elements are assigned to categories in the Group column of the track, `chromplot` creates a legend. If the Colors column is available, it will use custom colors, otherwise it assigns arbitrary colors.

```
> fst$Group <-  
+ ifelse(fst$win.FST >= 0      & fst$win.FST < 0.025, "Fst 0-0.025",  
+ ifelse(fst$win.FST >= 0.025 & fst$win.FST < 0.05,  "Fst 0.025-0.05",  
+ ifelse(fst$win.FST >= 0.05  & fst$win.FST < 0.075, "Fst 0.05-0.075",  
+ ifelse(fst$win.FST >= 0.075 & fst$win.FST < 0.1,   "Fst 0.075-0.1",  
+ ifelse(fst$win.FST >= 0.1   & fst$win.FST < 1,     "Fst 0.1-1", "na"))))  
> head(fst)
```

	Chrom	Start	End	win.n	win.FST	win.max	Colors	Group
1	19	10000001	10200000	1788	0.05867522	0.6810	grey35 Fst	0.05-0.075
2	19	1000001	1200000	2022	0.05720885	0.6590	grey35 Fst	0.05-0.075
3	19	10200001	10400000	1425	0.03499754	0.3584	grey55 Fst	0.025-0.05
4	19	10400001	10600000	1377	0.04107502	0.4172	grey55 Fst	0.025-0.05
5	19	10600001	10800000	1435	0.04324279	0.3513	grey55 Fst	0.025-0.05
6	19	10800001	11000000	1289	0.03154461	0.5857	grey55 Fst	0.025-0.05

```
> chromPlot(gaps=hg_gap, chr=c(19, 20, 21), bands=fst, figCols=3)
```



4.1.5 Synteny

This package is able of represent genomic regions that are conserved between two species. `chromplot` can work with AXT alignment files ⁵. Each alignment block in an AXT file contains three lines: a summary line (alignment information) and 2 sequence lines:

```
0 chr19 3001012 3001075 chr11 70568380 70568443 - 3500
TCAGCTCATAAATCACCTCCTGCCACAAGCCTGGCCTGGTCCCAGGAGAGTGTCCAGGCTCAGA
TCTGTTTCATAAACCACCTGCCATGACAAGCCTGGCCTGTTCCCAAGACAATGTCCAGGCTCAGA
```

```
1 chr19 3008279 3008357 chr11 70573976 70574054 - 3900
CACAATCTTCACATTGAGATCCTGAGTTGCTGATCAGAATGGAAGGCTGAGCTAAGATGAGCGA
CACAGTCTTCACATTGAGGTACCAAGTTGTGGATCAGAATGGAAAGCTAGGCTATGATGAGGGA
```

Moreover, `chromplot` is able to work with BED format. In the next example, we show how to graph sinteny between human and mouse from BED file.

```
> data_file3 <- system.file("extdata", "sinteny_Hg-mm10Chr19-21.txt", package =
+ "chromPlot")
> sinteny <- read.table(data_file3, sep="\t", stringsAsFactors=FALSE,
+ header=TRUE)
> head(sinteny)
```

	Chrom	Start	End	Group
1	chr19	60014	60661	chr6
2	chr19	60662	62424	chr6
3	chr19	64350	65036	chr6
4	chr19	65068	65395	chr6
5	chr19	65918	68409	chr6
6	chr19	69198	69857	chr17

⁵<https://genome.ucsc.edu/goldenPath/help/axt.html>

```
> chromPlot(gaps=hg_gap, bands=sinteny, chr=c(19:21), figCols=3)
```



4.2 Histograms

4.2.1 Single histogram

The user can generate a histogram for any of the following tracks: `annot1`, `annot2`, `annot3`, `annot4`, `segment`, and `segment2`. Histograms are created when the number of genomic elements in a track exceeds a maximum set by the `maxSegs` argument (200 by default) or the maximum size of the elements is $<$ `bin` size (1 Mb by default). Histograms can be plotted on either side of each chromosome. The side can be set for each track independently (see section 5.1).

The following example represents all annotated genes in the human genome⁶. You can also use BiomaRt package⁷ to get annotated information remotely.

```
> refGeneHg$Colors <- NULL
> head(refGeneHg)
```

	Chrom	Start	End	Name
1	chr19	41937222	41945843	NM_018035
2	chr19	41937222	41945481	NM_001167867
3	chr19	41937222	41945843	NM_001167869
4	chr19	41937222	41945481	NM_001167868
5	chr19	58694355	58724928	NM_016324
6	chr19	50321535	50340237	NM_030973

⁶<https://genome.ucsc.edu/cgi-bin/hgTables>

⁷<http://bioconductor.org/packages/2.3/bioc/html/biomaRt.html>

```
> chromPlot(gaps=hg_gap, bands=hg_cytoBandIdeo, annot1=refGeneHg, chr=c(19:21),
+ figCols=3)
```



Using biomaRt package:

```
> chromPlot(bands=hg_cytoBandIdeo, gaps=hg_gap, org="hsapiens")
```

(Same figure as above).

4.2.2 Stacked histograms: multiple files

It is possible to superimpose multiple histograms. This feature can be useful to represent processed data, obtained after of several stages of filtering or selection. For example, in microarray experiments, different colors of each histogram bar can represent the total number of genes (red), genes represented on the array (yellow), differentially over-expressed genes (green) and differentially sub-expressed genes (blue) in that order. The `annot3` and `annot4` parameters receive filtered and selected subsets of data array respectively. Given that both `annot4` and `annot3` contain information that has been 'selected' and 'filtered', the resulting histogram is quite small compared to gene density (red histogram).

```
> data_file4 <-system.file("extdata", "mm10_refGeneChr2-11-17-19.txt", package= "chromPlot")
> ref_mm10    <-read.table(data_file4, sep="\t", stringsAsFactors=FALSE, header
+ =TRUE)
> data_file5 <- system.file("extdata", "arrayChr17-19.txt", package = "chromPlot")
> array      <- read.table(data_file5, sep="\t", header=TRUE, stringsAsFactors=FALSE)
> head(ref_mm10)
```

	Chrom	Start	End	Name
1	chr2	50296809	50365000	NR_040361
2	chr2	50296809	50433967	NR_040362
3	chr2	40596772	42653598	NM_053011
4	chr2	58567333	58792971	NM_001289660
5	chr2	92184181	92364666	NM_138755
6	chr2	92221561	92364666	NM_001109690

```
> head(array, 4)
```

	Chrom	Start	End	Name
1	chr17	37399677	37400607	Olfr98
2	chr18	77996305	78006519	Haus1

3	chr19	5273920	5295455	Sf3b2
4	chr17	48526209	48549145	Nfya

Now, we will load the GenesDE object, and then we will obtain a subset of them, that it will contain over-expressed (nivel column equal to +) and sub-expressed (nivel column equal to -) genes.

```
> data(mm10_gap)
> data_file6 <- system.file("extdata", "GenesDEChr17-19.bed", package =
+ "chromPlot")
> GenesDE <- read.table(data_file6, sep="\t", header=TRUE,
+ stringsAsFactors=FALSE)
> head(GenesDE)
```

	Chrom	Start	End	Name	DE	nivel
1	chr18	74216566	74216635	mMA032457	-0.75	-
2	chr17	33778407	33778476	mMA032872	-0.63	-
3	chr17	69287649	69287718	mMA035704	0.77	+
4	chr17	31531186	31531255	mMC000870	0.72	+
5	chr18	84879549	84879618	mMC000964	0.62	+
6	chr19	45578791	45578860	mMC001997	0.60	+

```
> DEpos <- subset(GenesDE, nivel%in%"+")
> DEneg <- subset(GenesDE, nivel%in%"-")
> head(DEpos, 4)
```

	Chrom	Start	End	Name	DE	nivel
3	chr17	69287649	69287718	mMA035704	0.77	+
4	chr17	31531186	31531255	mMC000870	0.72	+
5	chr18	84879549	84879618	mMC000964	0.62	+
6	chr19	45578791	45578860	mMC001997	0.60	+

```
> head(DEneg, 4)
```

	Chrom	Start	End	Name	DE	nivel
1	chr18	74216566	74216635	mMA032457	-0.75	-

2	chr17	33778407	33778476	mMA032872	-0.63	-
7	chr19	5753674	5753743	mMC005778	-0.61	-
11	chr17	14404313	14404382	mMC011279	-0.84	-

```

> chromPlot(gaps=mm10_gap, bands=mm10_cytoBandIdeo, annot1=ref_mm10,
+ annot2=array, annot3=DEneg, annot4=DEpos, chr=c( "17", "18", "19"), figCols=3,
+ chrSide=c(-1, -1, -1, 1, -1, 1, -1, 1), noHist=FALSE)

```



4.2.3 Stacked histograms: single file

`chromplot` can also show stacked histograms from a `data.frame` with a 'Group' column containing category for each genomic elements. The `segment` and `segment2` arguments can take this type of input. As an example, we will plot differentially expressed genes classified by monocytes subtypes (Classical-noClassical and intermediate) on the right side of the chromosome, and a histogram of refSeq genes on the left side.

```
> data_file7 <- system.file("extdata", "monocitosDEChr19-21.txt", package =  
+ "chromPlot")  
> monocytes <- read.table(data_file7, sep="\t", header=TRUE,  
+ stringsAsFactors=FALSE)  
> head(monocytes)
```

	Chrom	Start	End	Group
1	chr19	18368098	18368147	Intermediate
2	chr19	17972951	17973000	Intermediate
3	chr19	46056289	46056338	Intermediate
4	chr20	30252463	30252512	Intermediate
5	chr21	32492542	32492591	Intermediate
6	chr19	39405989	39406038	Intermediate

```
> chromPlot(gaps=hg_gap, bands=hg_cytoBandIdeo, annot1=refGeneHg,
+ segment=monocytes, chrSide=c(-1,1,1,1,1,1,1,1,1), figCols=3, chr=c(19:21))
```



4.3 XY plots

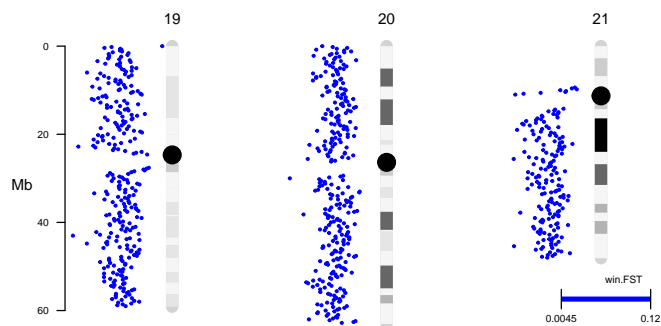
The arguments **stat** and **stat2** can take tracks of genomic elements associated with numeric values. The user can choose between lines or points for representing each data point along chromosomes by using the **statTyp** parameter (p = point, l = line). The **statCol** parameter must contain the name of the column containing continuous values in **stat** (use **statCol2** for **stat2**). It is possible to apply a statistical function (mean, median, sum etc) to the data using **statSumm** parameters ('none' by default). If the value is 'none', chromPlot will not apply any statistical function.

```
> head(fst)
```

	Chrom	Start	End	win.n	win.FST	win.max	Colors	Group
1	19	10000001	10200000	1788	0.05867522	0.6810	grey35 Fst	0.05-0.075
2	19	1000001	1200000	2022	0.05720885	0.6590	grey35 Fst	0.05-0.075
3	19	10200001	10400000	1425	0.03499754	0.3584	grey55 Fst	0.025-0.05
4	19	10400001	10600000	1377	0.04107502	0.4172	grey55 Fst	0.025-0.05
5	19	10600001	10800000	1435	0.04324279	0.3513	grey55 Fst	0.025-0.05
6	19	10800001	11000000	1289	0.03154461	0.5857	grey55 Fst	0.025-0.05

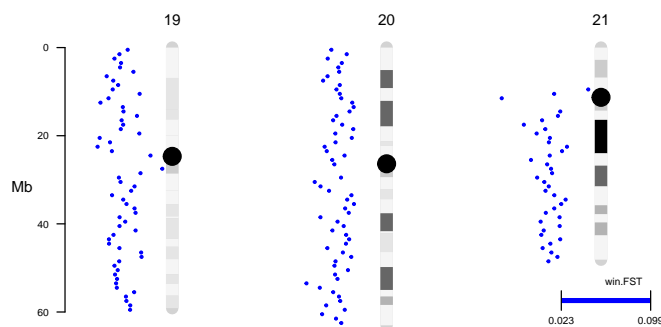
4.3.1 Using points

```
> chromPlot(bands=hg_cytoBandIdeo, gaps=hg_gap, stat=fst, statCol="win.FST",
+ statName="win.FST", statTyp="p", chr=c(19:21), figCols=3, scex=0.7, spty=20,
+ statSumm="none")
```



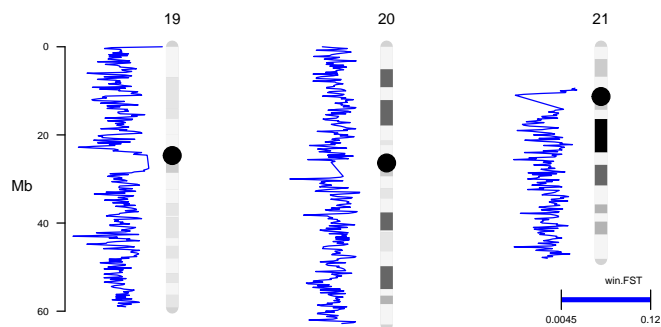
or calculating a mean of each value per bin by giving setting `statSumm="mean"`.

```
> chromPlot(bands=hg_cytoBandIdeo, gaps=hg_gap, stat=fst, statCol="win.FST",
+ statName="win.FST", statTyp="p", chr=c(19:21), figCols=3, scex=0.7, spty=20,
+ statSumm="mean")
```



4.3.2 Using connected lines

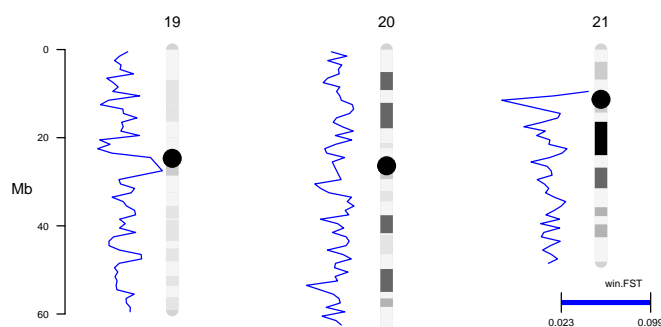
```
> chromPlot( bands=hg_cytoBandIdeo, gaps=hg_gap, stat=fst, statCol="win.FST",  
+ statName="win.FST", statTyp="l", chr=c(19:21), figCols=3, statSumm="none")
```



Here, we can smooth

the graph by using a mean per bin:

```
> chromPlot( bands=hg_cytoBandIdeo, gaps=hg_gap, stat=fst, statCol="win.FST",  
+ statName="win.FST", statTyp="l", chr=c(19:21), figCols=3, statSumm="mean")
```



Note that the `statSumm` argument can receive any function name ("none" is the default). No sanity check is performed, and thus the user is responsible to

make sure that using that function makes sense for the data at hand.

4.3.3 Coloring by datapoints exceeding a threshold

We will plot to two tracks of data with continuous values simultaneously using the (`stat` and `stat2` arguments. A third one will be shown on the chromosomal body after being categorized in arbitrary bins (see section 4.1.4). The values on both tracks of continuous data will be colored according to a threshold provided by the user in the `statThreshold` and `statThreshold2` parameters, which are applied for the `stat` and `stat2` tracks, respectively.

```
> data_file8 <- system.file("extdata", "iHS_CEUChr19-21", package = "chromPlot")
> ihs <- read.table(data_file8, sep="\t", stringsAsFactors=FALSE, header=TRUE)
> head(ihs)
```

	Chrom	Start	End	iHS	Name
1	19	52501632	52501633	1.4914346	rs8103812
2	19	11095063	11095064	0.9520553	rs112825147
3	20	51172436	51172437	1.4262380	rs4268981
4	21	18842550	18842551	0.3136856	rs77147477
5	21	26240760	26240761	0.4122098	rs2226391
6	20	52752592	52752593	2.3400389	rs6013901

```
> data_file9 <-system.file("extdata", "XPEHH_CEU-YRIChr19-21", package="chromPlot")
> xpehh <-read.table(data_file9, sep="\t", stringsAsFactors=FALSE, header=TRUE)
> head(xpehh)
```

	Chrom	Start	End	XP	Name
1	20	15849464	15849465	1.51707487	rs183441159
2	21	32430761	32430762	0.54250598	rs148400564
3	20	59957644	59957645	0.35507696	rs6121418
4	20	61887895	61887896	0.54328659	rs910892

```

5    20 50429216 50429217 0.27747208 rs73273526
6    20 45208887 45208888 -0.06347502 rs144014837

```

We can label any data point by providing an 'ID' column with labels. ID values of NA, NULL, or empty ("") are ignored. Here, we will only label single data point with the maximum XP value.

```

> xpehh$ID <- ""
> xpehh[which.max(xpehh$XP), "ID"] <- xpehh[which.max(xpehh$XP), "Name"]
> head(xpehh)

```

	Chrom	Start	End	XP	Name	ID
1	20	15849464	15849465	1.51707487	rs183441159	
2	21	32430761	32430762	0.54250598	rs148400564	
3	20	59957644	59957645	0.35507696	rs6121418	
4	20	61887895	61887896	0.54328659	rs910892	
5	20	50429216	50429217	0.27747208	rs73273526	
6	20	45208887	45208888	-0.06347502	rs144014837	

```

> chromPlot(gaps=hg_gap, bands=fst, stat=ihs, stat2=xpehh, statCol="iHS",
+ statCol2="XP", statName="iHS", statName2="normxpehh", colStat="red", colStat2="blue", st
+ bin=1e6, figCols=3, cex=0.7, statSumm="none", legChrom=19, stack=FALSE)

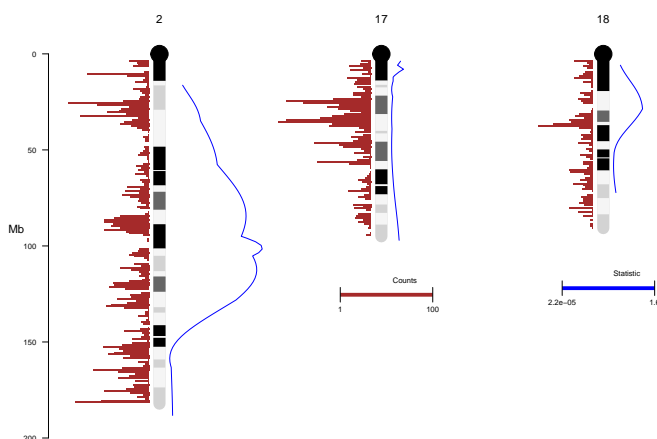
```



4.3.4 Plotting LOD curves

A potential use of connected lines is plotting the results from QTL mapping. Here we show a simple example of how to plot the LOD curves from a QTL mapping experiment in mice along a histogram of gene density. For demonstration purposes, we use a simple formula for converting cM to bp. A per-chromosome map or an appropriate online tool (<http://cgd.jax.org/mousemapconverter/>) should be used in real applications.

```
> library(qtl)
> data(hyper)
> hyper <- calc.genoprob(hyper, step=1)
> hyper <- scanone(hyper)
> QTLs <- hyper
> colnames(QTLs) <- c("Chrom", "cM", "LOD")
> QTLs$Start <- 1732273 + QTLs$cM * 1895417
> chromPlot(gaps=mm10_gap, bands=mm10_cytoBandIdeo, annot1=ref_mm10, stat=QTLs,
+ statCol="LOD", chrSide=c(-1,1,1,1,1,1,1,1), statTyp="l", chr=c(2,17:18), figCols=3)
```



4.3.5 Plotting a map with IDs

In the previous section, we used an ID to highlight one point from a track with continuous values. However, `chromPlot` can display many IDs, while trying to avoid overlapping of text labels. Points are ordered by position and the overlapping labels are moved downwards. This is useful for displaying maps, e.g. genetic or physical maps of genetic markers. For this the user must ensure that the table contains the ID column. The values in that column will be plotted as labels next to the data point.

In the following example we show the IDs of a small panel of 150 SNPs. We will use a different color for known (rs) and novel (non rs) SNPs. By setting `statType="n"` we avoid plotting the actual data point.

```
> data_file10 <- system.file("extdata",  
+ "CLG_AIMs_150_chr_hg19_v2_SNP_rs_rn.csv",  
+ package = "chromPlot")  
> AIMS <- read.csv(data_file10, sep=",")  
> head(AIMS)
```

	Chrom	Start	End	ID	Colors
1	4	31841506	31841507	rn131966	darkgreen
2	7	61540368	61540369	rn243926	darkgreen
3	12	109427241	109427242	rn381459	darkgreen
4	4	100673238	100673239	rn145426	darkgreen
5	8	122220756	122220757	rn286585	darkgreen
6	10	56881122	56881123	rn322283	darkgreen

```
> chromPlot(gaps=hg_gap, bands=hg_cytoBandIdeo, stat=AIMS, statCol="Value",
+ statName="Value", noHist=TRUE, figCols=4, cex=0.7, chr=c(1:8), statTyp="n",
+ chrSide=c(1,1,1,1,1,1,-1,1))
```



4.4 Segments

4.4.1 Large stacked segments

`chromplot` allows for the user represent large segments as vertical bars on either side of the chromosomal bodies. If the maximum segment size of segments is smaller than `bin` (1 Mb by default), or there are more segments than `maxSegs` (200 by default), they will be plotted as a histogram. However, the user can change this behavior by setting the `noHist` parameter to `TRUE`. If a 'Group' column is present in the table of segments, it is used as a category variable and different colors are used for segments in each category. The user can set the colors to be used in the `colSegments` and `colSegments2` arguments.

This type of graph is useful for displaying, for instance, QTLs (quantitative trait locus), due to the fact that they cover large genomic regions. Here we show how to graph segments on the side of the chromosomal body. By setting `stack=TRUE` (default), drawing space is saved by plotting all nonoverlapping segments at the minimum possible distance from the chromosome. Otherwise, they are plotted at increasing distance from the chromosome, regardless of whether they overlap or not.

```
> data_file12 <-system.file("extdata", "QTL.csv", package = "chromPlot")
> qtl          <-read.table(data_file12, sep="," , header =TRUE,
+ stringsAsFactors=FALSE)
> head(qtl)
```

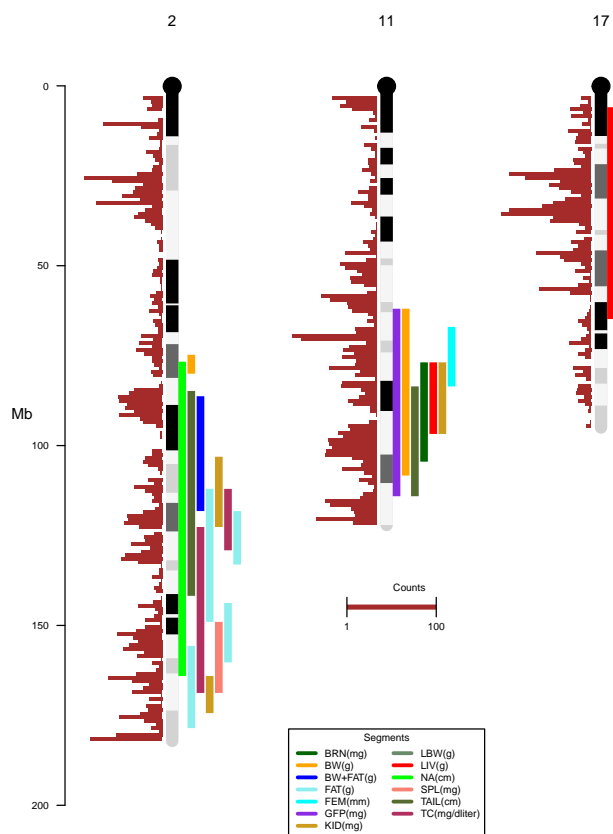
	Chrom	Start	End	Group	Name
1	2	112034866	149008061	FAT(g)	Fatq1
2	2	155693206	178535307	FAT(g)	Fatq2
3	2	149008061	168761938	SPL(mg)	Swq6
4	2	103105582	122639899	KID(mg)	Kwq7
5	2	164060872	174372769	KID(mg)	Kwq8

6 2 84814041 141777153 TAIL(cm) Tailq7

```

> chromPlot(gaps=mm10_gap, segment=qt1, noHist=TRUE, annot1=ref_mm10,
+ chrSide=c(-1,1,1,1,1,1,1,1,1), chr=c(2,11,17), stack=TRUE, figCol=3,
+ bands=mm10_cytoBandIdeo)

```



4.4.2 Large stacked segments grouped by two categories

When the segments have more than one category (up to two supported), they are differentiated by a combination of color and shape for a point plotted in the middle of the segment. The segment itself is shown in gray. The first category is taken from the 'Group' column and establishes the color of the symbol. The second category is taken from the 'Group2' column and determines the symbol shape.

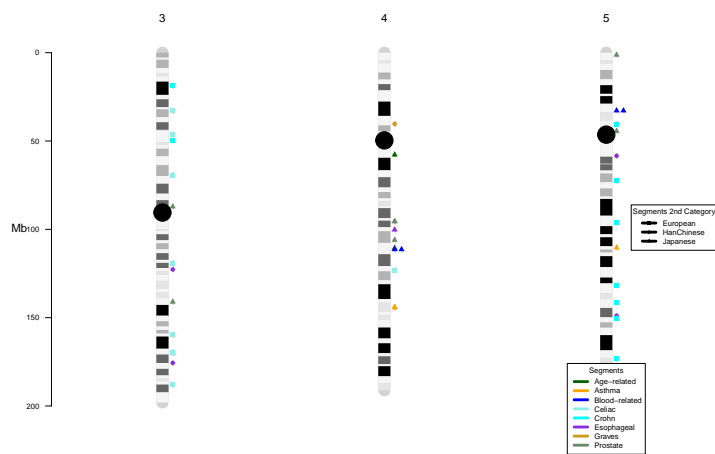
In the following example, we use data for SNPs associated with phenotypes and ethnicity, taken from phenoGram website ⁸.

```
> data_file11 <- system.file("extdata", "phenogram-ancestry-sample.txt",  
+ package = "chromPlot")  
> pheno_ancestry <- read.csv(data_file11, sep="\t", header=TRUE)  
> head(pheno_ancestry)
```

	Chrom	Start	End	Name	Group	Group2
1	1	10796866	10796867	rs880315	Blood-related	Japanese
2	1	10796866	10796867	rs880315	Blood-related	Japanese
3	1	113190807	113190808	rs17030613	Blood-related	Japanese
4	1	113190807	113190808	rs17030613	Blood-related	Japanese
5	1	196646176	196646177	rs1329424	Age-related	Japanese
6	1	196679455	196679456	rs10737680	Age-related	Japanese

⁸<http://visualization.ritchielab.psu.edu/phenograms/examples>

```
> chromPlot(bands=hg_cytoBandIdeo, gaps=hg_gap, segment=pheno_ancestry,
+ noHist=TRUE, chr=c(3:5), figCols=3, legChrom=5)
```



Since the data contain SNPs positions, the segments are only 1 bp long and the resulting lines are too small to be seen. For display purposes, we will increase the segments' sizes by adding a 500Kb pad to either side of each SNP.

```
> pheno_ancestry$Start<-pheno_ancestry$Start-5e6
> pheno_ancestry$End<-pheno_ancestry$End+5e6
> head(pheno_ancestry)
```

	Chrom	Start	End	Name	Group	Group2
1	1	5796866	15796867	rs880315	Blood-related	Japanese
2	1	5796866	15796867	rs880315	Blood-related	Japanese
3	1	108190807	118190808	rs17030613	Blood-related	Japanese
4	1	108190807	118190808	rs17030613	Blood-related	Japanese
5	1	191646176	201646177	rs1329424	Age-related	Japanese
6	1	191679455	201679456	rs10737680	Age-related	Japanese

```
> chromPlot(bands=hg_cytoBandIdeo, gaps=hg_gap, segment=pheno_ancestry,
+ noHist=TRUE, chr=c(3:5), figCols=3, legChrom=5)
```



4.4.3 Large non-overlapping segments

`chromplot` can categorize genomic regions (Group column) and then represent them with different colors. Also the package is capable of showing non-overlapping regions along the chromosome. The following example shows the ancestry of each chromosomal region. The user can obtain the annotation data updated through the `biomaRt` package.

```
> data_file13 <- system.file("extdata", "ancestry_humanChr19-21.txt", package =  
+ "chromPlot")  
> ancestry <- read.table(data_file13, sep="\t", stringsAsFactors=FALSE,  
+ header=TRUE)  
> head(ancestry)
```

	Chrom	Start	End	Group	Strand
1	19	261033	327323	AMR	+
2	19	865406	1175396	AMR	+
3	19	1364306	1642507	AMR	+
4	19	1882762	1973732	AMR	+
5	19	2491586	2728577	AMR	+
6	19	2906475	2997897	AMR	+

```

> chromPlot(gaps=hg_gap, bands=hg_cytoBandIdeo, chrSide=c(-1,1,1,1,1,1,1,1),
+ noHist=TRUE, annot1=refGeneHg, figCols=3, segment=ancestry, colAnnot1="blue",
+ chr=c(19:21), legChrom=21)

```



4.5 Multiple data types

The `chromPlot` package is able to plot diverse types of tracks simultaneously.

```
> chromPlot(stat=fst, statCol="win.FST", statName="win.FST", gaps=hg_gap,  
+ bands=hg_cytoBandIdeo, statTyp="l", noHist=TRUE, annot1=refGeneHg,  
+ chrSide=c(-1, 1, 1, 1, 1, 1, 1, 1), chr = c(19:21), figCols=3, cex=1)
```



Here we show a figure from in Verdugo et al. (2010), to represent the association between the genetic divergence regions (darkred regions in the body of the chromosomes), the QTLs (color bars on the right of the chromosome), and the absence of association with gene density shown (histogram on the left side of the chromosomes).

```
> options(stringsAsFactors = FALSE);
> data_file14<-system.file("extdata", "donor_regions.csv", package = "chromPlot")
> region<-read.csv(data_file14, sep=",")
> region$Colors    <- "darkred"
> head(region)
```

	Chrom	Start	End	Group	Colors
1	chr2	74903477	180989506	donor region	darkred
2	chr11	61609496	114085002	donor region	darkred
3	chr17	5936872	86128472	donor region	darkred

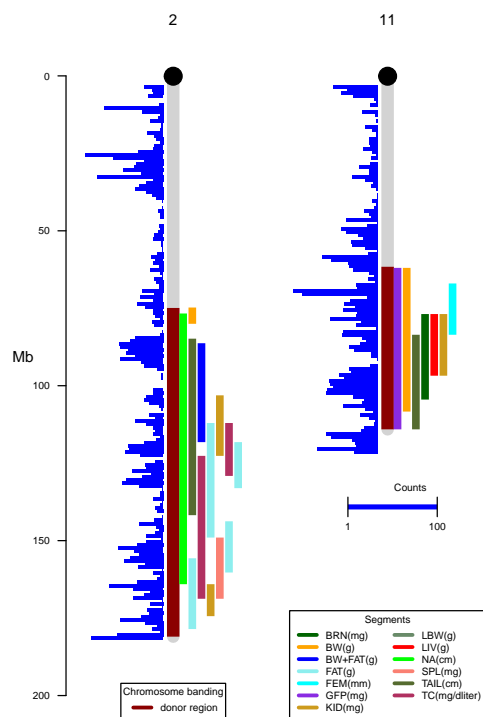
```
> head(qtl)
```

	Chrom	Start	End	Group	Name
1	2	112034866	149008061	FAT(g)	Fatq1
2	2	155693206	178535307	FAT(g)	Fatq2
3	2	149008061	168761938	SPL(mg)	Swq6
4	2	103105582	122639899	KID(mg)	Kwq7
5	2	164060872	174372769	KID(mg)	Kwq8
6	2	84814041	141777153	TAIL(cm)	Tailq7

```

> chromPlot(gaps=mm10_gap, segment=qt1, noHist=TRUE, annot1=ref_mm10,
+ chrSide=c(-1,1,1,1,1,1,1,1,1), chr=c(2,11,17), stack=TRUE, figCol=3,
+ bands=region, colAnnot1="blue")

```



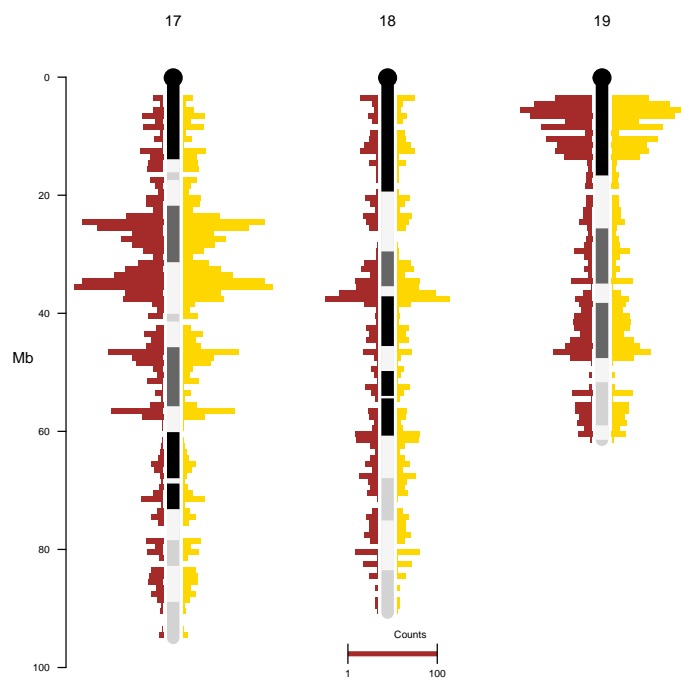
5 Graphics settings

5.1 Choosing side

The user can choose a chromosome side for any track of data, except if given to the `bands` argument, in which case it is plotten on the body of the chromosome. The `chrSide` parameter receives a vector with values 1 or -1 for each genomic tracks (`annot1`, `annot2`, `annot3`, `annot4`, `segment`, `segment2`, `stat` and `stat2`) placing them to the right (if -1) or to the left (if 1) of the chromosomes.

For demonstration, here we show the same track of data on two different sides.

```
> chromPlot(gaps=mm10_gap, bands=mm10_cytoBandIdeo, annot1=ref_mm10,  
+ annot2=ref_mm10, chrSide=c(-1, 1, 1, 1, 1, 1, 1, 1), chr=c(17:19), figCols=3)
```



5.2 Choosing colors

For each parameter that received a `data.frame`, the user can specify a color for plotting. If the data will be plotted as segments, the user can specify a vector of colors. The color will be assigned in the order provided to each level of a category (when a `Group` column is present in the data table). The color parameters and their respective data tracks are as follows:

1. `colAnnot1:` `annot1`
2. `colAnnot2:` `annot2`
3. `colAnnot3:` `annot3`
4. `colAnnot4:` `annot4`
5. `colSegments:` `segment`
6. `colSegments2:` `segment2`
7. `colStat:` `stat`
8. `colStat2:` `stat2`

For data that are plotted individually, i.e. bands, segments, points in XY, or data labels, it is possible to set an arbitrary color for each element by providing a color name in a column called “Colors” in the data table. Setting a value in this way overrides any color provided in the above arguments for a given track. The user is responsible for providing color names that R understands. No check is done by `chromPlot`, but R will complain if a wrong name is used. For an example of this use, see section 4.1.3.

5.3 Placement of legends

`chromPlot` places the legends under the smallest or second smallest chromosome, depending on the number of legends needed. The legend for the second

category of a segments track is placed in the middle-right of the plotting area of the smallest chromosome. These choices were made because they worked in most cases that we tested. However, the placement of legends in R not easily automated to produce optimal results in all situations. Depending on the particular conditions of a plot such as data density, chromosomes chosen, font size and the size of the plotting device, the the legend by block viewing some data.

When not pleased with the result of `chromPlot`'s placing of legends, the user has two options:

1. setting the `legChrom` argument to an arbitrary chromosome name. The legend will be placed under that chromosome. If more than one legend is needed the first one will be placed under the chromosome before the chosen chromosome, unless only one chromosome is plotted.
2. setting the `legChrom` to `NA` to omit plotting a legend. The user can use the `legend()` function to create a custom legend and can choose the best location by trial and error.

6 Acknowledgments

This work was funded by a the FONDECYT Grant 11121666 by CONICYT, Government of Chile. We acknowledge all the users who have tested the software and provided valuable feedback. Particularly, Alejandro Blanco and Paloma Contreras.

7 REFERENCES

Verdugo, Ricardo A., Charles R. Farber, Craig H. Warden, and Juan F. Medrano. 2010. “Serious Limitations of the QTL/Microarray Approach for QTL Gene Discovery.” *BMC Biology* 8 (1): 96. doi:10.1186/1741-7007-8-96.