An introduction of drawing genomic figures with *circlize*

Zuguang Gu <z.gu@dkfz.de>

June 3, 2014

1 Introduction

Since circos plots are mostly used in genomics research, the *circlize* package especially provides functions which focus on genomic plots. These functions are synonymous to the basic circos graphical functions but expect special format of input data:

- `circos.genomicTrackPlotRegion`: create a new track and add graphics.
- `circos.genomicPoints`: low-level function, add points
- `circos.genomicLines`: low-level function, add lines
- `circos.genomicRect`: low-level function, add rectangles
- `circos.genomicText`: low-level function, add text
- `circos.genomicLink`: add links

The genomic functions are implemented by basic circos functions (e.g. `circos.trackPlotRegion`, `circos.points`), thus, you can customize your own plots by both genomic functions and basic circos functions.

2 Input data

circlize expects input data as a data frame or a list of data frames in which there are at least three columns. The first column is genomic category (e.g. chromosome), the second column is the start position in the genomic category and the third column is the end position. Following columns are optional where numeric values or other related values are stored. Such data structure is known as *BED* format and is broadly used in genomic research.

circlize provides a simple function `generateRandomBed` which can generate random genomic data. Positions are uniformly generated from human geonme. In the function, `nr` and `nc` are number of rows and numeric columns that users want. Please note `nr` are not always equal to the number of rows which are returned by the function. `fun` is a self-defined function to generate random values.

```
> bed = generateRandomBed()
> bed = generateRandomBed(nr = 200, nc = 4)
> bed = generateRandomBed(fun = function(k) runif(k))
```

3 Initialize with cytoband data

Similar as general circos plots, the first step is to initialize the plot with genomic categories. In most situations, genomic categories are measured by chromosomes. The easiest way is to used `circos.initializeWithIdeogram`:

```
> circos.initializeWithIdeogram()
```

By default, the function will initialize the plot with cytoband data of hg19. You can also use your own cytoband data by specifying the path of your cytoband file or providing your cytoband data as a data frame.

```
> cytoband.file = paste(system.file(package = "circlize"),
+     "/extdata/cytoBand.txt", sep = "")
> circos.initializeWithIdeogram(cytoband.file)
> cytoband.df = read.table(cytoband.file, colClasses = c("character", "numeric",
+     "numeric", "character", "character"), sep = "\t")
> circos.initializeWithIdeogram(cytoband.df)
```

If you want to read cytoband data from file, please explicitly specify `colClasses` arguments and set the class of position columns as `numeric`. The reason is since positions are represented as integers, `read.table` would treat those numbers as `integer` by default. In initialization of circos plot, *circlize* needs to calculate the summation of all chromosome lengths. The sumation of such large integers would throw error of data overflow.

For simple use, users can also specify abbreviation of the species and the function will download cytoband file from UCSC server automatically.

```
> circos.initializeWithIdeogram(species = "hg18")
> circos.initializeWithIdeogram(species = "mm10")
```

By default, the function will use all chromosomes which are available in cytoband data to initialize the circos plot. Users can also choose a subset of chromosomes by specifying `chromosome.index`. Please note this argument is only used for subsetting but not for ordering.

```
> circos.initializeWithIdeogram(chromosome.index = c("chr1", "chr2"))
```

Initialization step is important for circos plot. It controls the order of chromosomes which is going to be shown on the circle. There are several ways to control the order. If `cytoband` is provided as a data frame, and if the first column is a factor, the order of chromosomes would be `levels(cytoband[[1]])`. If the first column is not a factor, the order of chromosomes would be `unique(cytoband[[1]])`. If `sort.chr` is set to `TRUE`, chromosomes will be sorted.

```
> cytoband = read.table(cytoband.file)
> circos.initializeWithIdeogram(cytoband, sort.chr = FALSE)
> cytoband[[1]] = factor(cytoband[[1]], levels = paste("chr", c(22:1, "X", "Y")))
> circos.initializeWithIdeogram(cytoband, sort.chr = FALSE)
> cytoband = read.table(cytoband.file)
> circos.initializeWithIdeogram(cytoband, sort.chr = TRUE)
```

If `cytoband` is specified as a file path, or `species` is specified, the order of chromosomes depends on the original order in the source file.

circlize provides a function `read.cytoband` which can read/download and process cytoband data. In fact, `circos.initializeWithIdeogram` calls `read.cytoband` internally. Please refer to the help page of the function for more details.

```
> cytoband = read.cytoband()
> cytoband = read.cytoband(file)
> cytoband = read.cytoband(df)
> cytoband = read.cytoband(species)
```

After the intialization of the circos plot, the function will additionally create a track where there are genomic axis and chromosome names, and create another track where there is an ideogram. `plotType` can be used to control which graphics need to be plotted.

```
> circos.initializeWithIdeogram(plotType = c("axis", "labels"))
> circos.initializeWithIdeogram(plotType = NULL)
```

Similar as general circos plot, the layout of circos plot can be controlled by `circos.par`

```
> circos.par("start.degree" = 90)
> circos.initializeWithIdeogram()
> circos.par("gap.degree" = rep(c(2, 4), 11))
> circos.initializeWithIdeogram()
```

Please refer to figure 1 for examples of different ways to initialize genomic circos plot.

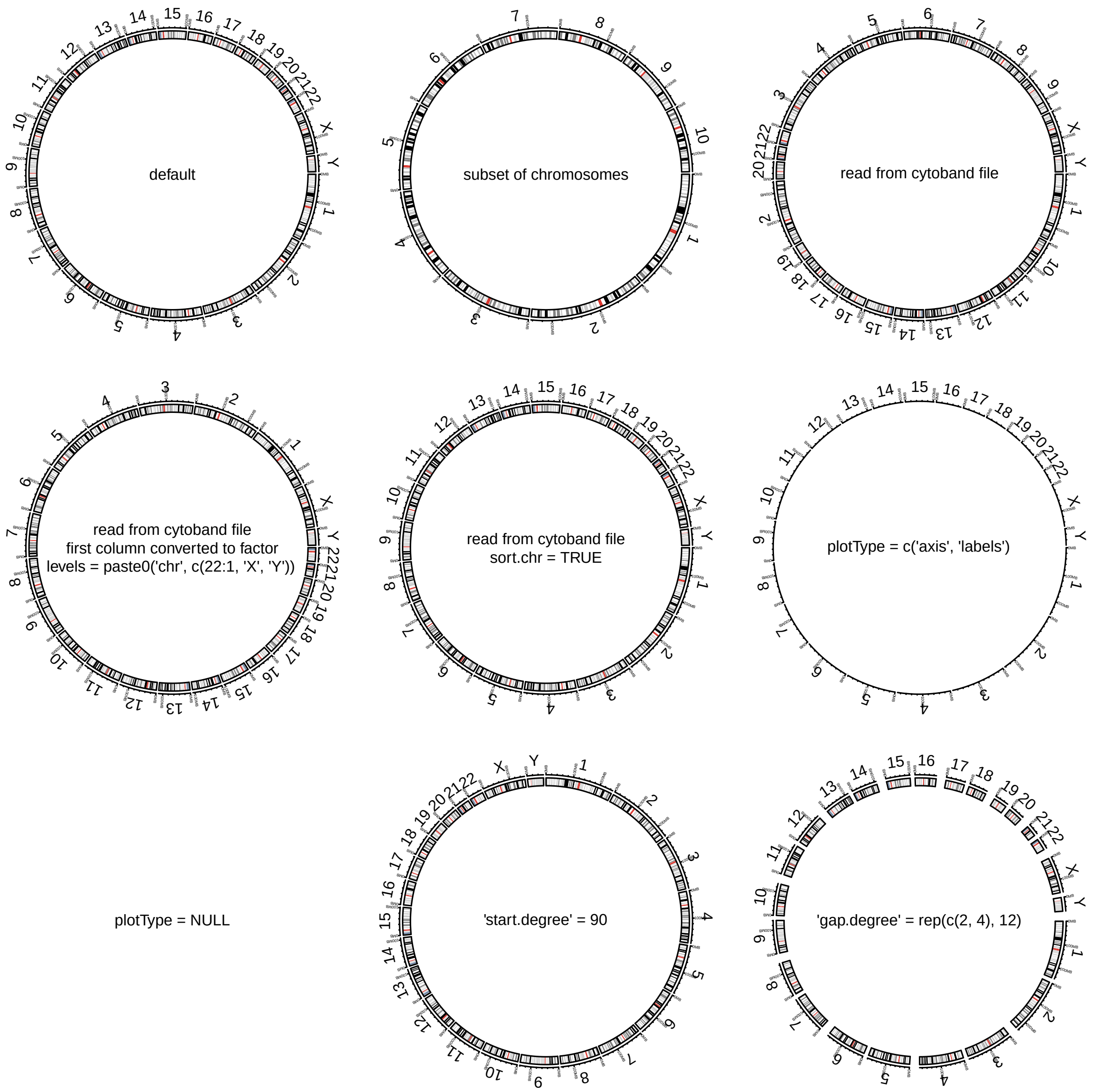

Figure 1: Different ways to initialize genomic circos plot

4 Initialize with general genomic category

Cytoband data is just a special case of genomic category. `circos.genomicInitialize` can initialize circos plot with any kind of genomic categories. In fact, `circos.initializeWithIdeogram` is implemented by `circos.genomicInitialize`. The input data for the function is a data frame with at least three columns. The first column is genomic category (for cytoband data, it is chromosome name), and the next two columns are genomic positions in each genomic category. The range of each category will be inferred from the minimun position and the maximum position in corresponding category. In the following example, a circos plot is initialized with three genes.

```
> df = data.frame(
+     name  = c("TP53",  "TP63",    "TP73"),
+     start = c(7565097, 189349205, 3569084),
+     end   = c(7590856, 189615068, 3652765))
> circos.genomicInitialize(df)
```

Note it is not necessary that the record for each gene is one row.

As explained in previous section, the order of genomic categies is controlled by the first column of `df` which depends whether it is a factor or a simple vector. Alternative names can be assigned to each category and the order of names correspond to the order of genomic categories.

```
> circos.genomicInitialize(df, sector.names = c("tp53", "tp63", "tp73"))
```

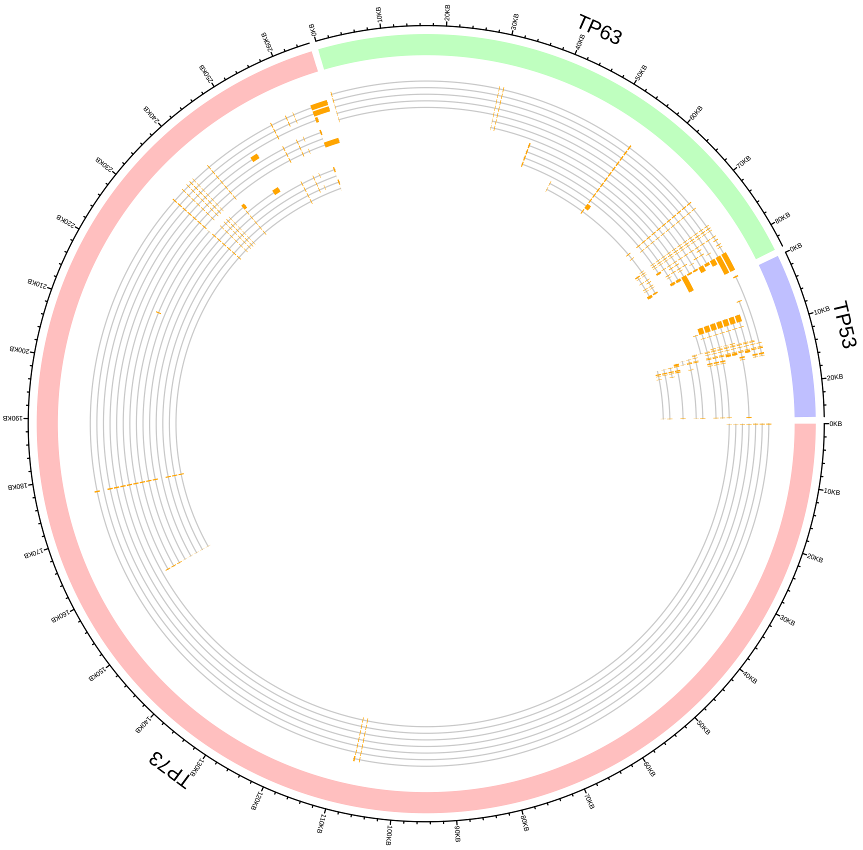

Figure 2: A circos plot with three genes

Figure 2 initializes a circos plot with three genes and plots all alternative transcripts. The transcripts are drawn by `circos.genomicRect` which will be explained in following sections.

5 Create plotting regions

In following sections, chromosome will be used as the type of genomic category.

Similar as `circos.trackPlotRegion`, `circos.genomicTrackPlotRegion` also accepts a self-defined function `panel.fun` which is applied in every cell but with different form.

```
> circos.genomicTrackPlotRegion(data, panel.fun = function(region, value, ...) {
+     circos.genomicPoints(region, value, ...)
+ })
```

Inside `panel.fun`, users can use low-level genomic graphical functions to add basic graphics in each cell. `panel.fun` expects two arguments `region` and `value`. `region` is a data frame containing start position and end position in the current chromosome which is extracted from `data`. `value` is also a data frame which contains other columns in `data`. There should be a third arguments `...` which is mandatory and is used to pass user-invisible variables to inner functions.

Since circos.genomicTrackPlotRegion will create a new track, it needs values to calculate range of y-values to arrange data points. Users can either specify the index of numeric columns in data by numeric.column or set ylim. If none of them are set, the function will try to look for all numeric columns in data (of cource, excluding the first three columns), and set them as numeric.column.

```
> circos.genomicTrackPlotRegion(data, ylim = c(0, 1),
+     panel.fun = function(region, value, ...) {
+         circos.genomicPoints(region, value, ...)
+ })
> circos.genomicTrackPlotRegion(data, numeric.column,
+     panel.fun = function(region, value, ...) {
+         circos.genomicPoints(region, value, ...)
+ })
```

5.1 Points

circos.genomicPoints is similar as circos.points. The difference is circos.genomicPoints expects a data frame containing genomic regions and a data frame containing values. The data column for plotting should be indicated by numeric.column. If the function is called inside circos.genomicTrackPlotRegion and users have been already set numeric.column in circos.genomicTrackPlotRegion, proper value of numeric.column will be passed to circos.genomicPoints through ... in panel.fun. Which means, you need to add ... as the final argument in circos.genomicPoints to pass such informatioin into it. If numeric.column is not set, circos.genomicPoints will use all numeric columns detected in value.

```
> circos.genomicPoints(region, value, ...)
> circos.genomicPoints(region, value, numeric.column = c(1, 2))
> circos.genomicPoints(region, value, cex, pch)
> circos.genomicPoints(region, value, sector.index, track.index)
```

If there is only one numeric column, graphical parameters such as pch, cex can be of length one or number of rows of region. If there are more than one numeric columns specified, points for each numeric column will be added iteratively, and the graphical parameters should be either length one or number of numeric columns specified.

5.2 Lines

circos.genomicLines is similar as circos.lines. The setting of graphical paramters is similar as circos.genomicPoints.

```
> circos.genomicLines(region, value, ...)
> circos.genomicLines(region, value, numeric.column = c(1, 2))
> circos.genomicLines(region, value, lwd, lty = "segment")
> circos.genomicLines(region, value, area, area.baseline, border)
> circos.genomicLines(region, value, sector.index, track.index)
```

For lty, we additionally provide a new option segment by which each genomic interval will represent as a 'horizontal' line at corresponding value in value.

5.3 Text

For circos.genomicText, the position of text can be specified either by numeric.column or a seperated vector y. The labels of text can be specified either by labels.column or a vector labels.

```
> circos.genomicText(region, value, ...)
> circos.genomicText(region, value, y = 1, labels = "gene")
> circos.genomicText(region, value, direction, adj)
> circos.genomicText(region, value, sector.index, track.index)
```

5.4 Rectangle

For `circos.genomicRect`, the positions of top and bottom of the rectangles can be specified by `ytop`, `ybottom` or `ytop.column`, `ybottom.column`.

```
> circos.genomicRect(region, value, ytop = 1, ybottom = 0)
> circos.genomicRect(region, value, col, border)
```

One of the usage of `circos.genomicRect` is to plot heatmap on the circle. *circlize* provides a simple function `colorRamp2` which wraps `colorRamp` to interpolate colors. The arguments of `colorRamp2` are break points and colors which correspond to the the break points. `colorRamp2` returnes a new function which can be used to generate new colors.

```
> col_fun = colorRamp2(breaks = c(-1, 0, 1), colors = c("green", "black", "red"))
> col_fun(c(-2, -1, -0.5, 0, 0.5, 1, 2))
```

```
[1] "#00FF00" "#00FF00" "#007F00" "#000000" "#7F0000" "#FF0000" "#FF0000"
```

5.5 More details on `circos.genomicTrackPlotRegion`

The behavior of `circos.genomicTrackPlotRegion` and `panel.fun` will be different according to different input data and different settings.

5.5.1 Normal mode

If input data is a simple data frame, `region` in `panel.fun` would be a data frame containing start position and end position in the current chromosome which is extracted from input data. `value` is also a data frame which contains columns in input data excluding the first three columns. Index of proper numeric columns will be passed by `...`. So if users want to use such information, they need to pass `...` to low-level genomic function such as `circos.genoimcPoints` as well.

```
> bed = generateRandomBed(nc = 2)
> circos.genomicTrackPlotRegion(bed, panel.fun = function(region, value, ...) {
+     circos.genomicPoints(region, value, ...)
+     circos.genomicPoints(region, value)
+     circos.genomicPoints(region, value, numeric.column = 1)
+ })
```

If input data is a list of data frames, `panel.fun` is applied on each data frame iteratively. Under such situation, `region` and `value` will contain corresponding data in the current data frame. The numeric index for the current data frame can be get by `getI(...)`.

```
> bedlist = list(generateRandomBed(), generateRandomBed())
> circos.genomicTrackPlotRegion(bedlist,
+     panel.fun = function(region, value, ...) {
+         i = getI(...)
+         circos.genomicPoints(region, value, col = i, ...)
+ })
```

5.5.2 `stack` mode

`circos.genomicTrackPlotRegion` also support a `stack` mode. Under `stack` mode, `ylim` is re-defined inside the function. The y-axis will be splitted into several parts with equal height and graphics will be drawn on each 'horizontal' lines (y = 1, 2, ...).

Under `stack` mode, when input data is a single data frame containing one or more numeric columns, each numeric column defined in `numeric.column` will be treated as a single unit. `ylim` is re-defined to `c(0.5, n+0.5)` in which `n` is number of numeric columns. `panel.fun` will be applied iteratively on each numeric column. In each iteration, in `panel.fun`, `region` is still the genomic regions in current genomic category, but `value` only contains current numeric column plus all non-numeric columns. All low-level genomic graphical functions will be drawn on the 'horizontal line' `y = i` in which `i` is the index of current numeric column and the value of `i` can be obtained by `getI`.

```
> bed = generateRandomBed(nc = 2)
> circos.genomicTrackPlotRegion(bed, stack = TRUE,
+     panel.fun = function(region, value, ...) {
+         i = getI(...)
+         circos.genomicPoints(region, value, col = i, ...)
+ })
```

When input data is a list containing data frames, each data frame will be treated as a single unit. The situation is quite similar as described previously. `ylim` is re-defined to `c(0.5, n+0.5)` in which `n` is number of data frames. `panel.fun` will be applied iteratively on each data frame. In each iteration, in `panel.fun`, `region` is still the genomic regions in current chromosome, and `value` contains columns in current data frame excluding the first three columns.

```
> bedlist = list(generateRandomBed(), generateRandomBed())
> circos.genomicTrackPlotRegion(bedlist, stack = TRUE,
+     panel.fun = function(region, value, ...) {
+         i = getI(...)
+         circos.genomicPoints(region, value, ...)
+ })
```

Please see figure 3, for examples of using different settings in `circos.genomicTrackPlotRegion`.

5.5.3 Mixed use of general circos functions

`panel.fun` is applied on each cell, which means, besides genomic functions, you can also use general circos functions to add more graphics. For example, some horizontal lines and texts need to be added to each cell and axis need to put on top of each cell:

```
> circos.genomicTrackPlotRegion(bed, ylim = c(-1, 1),
+     panel.fun = function(region, value, ...) {
+         circos.genomicPoints(region, value, ...)
+         cell.xlim = get.cell.meta.data("cell.xlim")
+         for(h in c(-1, -0.5, 0, 0.5, 1)) {
+             circos.lines(cell.xlim, c(0, 0), lty = 2, col = "grey")
+         }
+         circos.text(x, y, labels)
+         circos.axis("top")
+ })
```

5.6 links

`circos.genomicLink` expects two data frames and it will add links from genomic intervals in the first data frame to corresponding genomic intervals in the second data frame (figure 4).

```
> circos.genomicLink(bed1, bed2)
> circos.genomicLink(bed1, bed2, col)
```

5.7 Highlight chromosomes

`highlight.chromosome` provides a simple to highlight chromosomes. Just remember to use transparent filled colors. The position of the highlighted region and be fine-tuned by `padding` argument which are percentages of corresponding height and width in the highlighted region. (figure 5)

```
> highlight.chromosome("chr1")
> highlight.chromosome("chr1", track.index = c(2, 3))
> highlight.chromosome("chr1", col = NA, border = "red")
> highlight.chromosome("chr1", padding = c(0.1, 0.1, 0.1, 0.1))
```

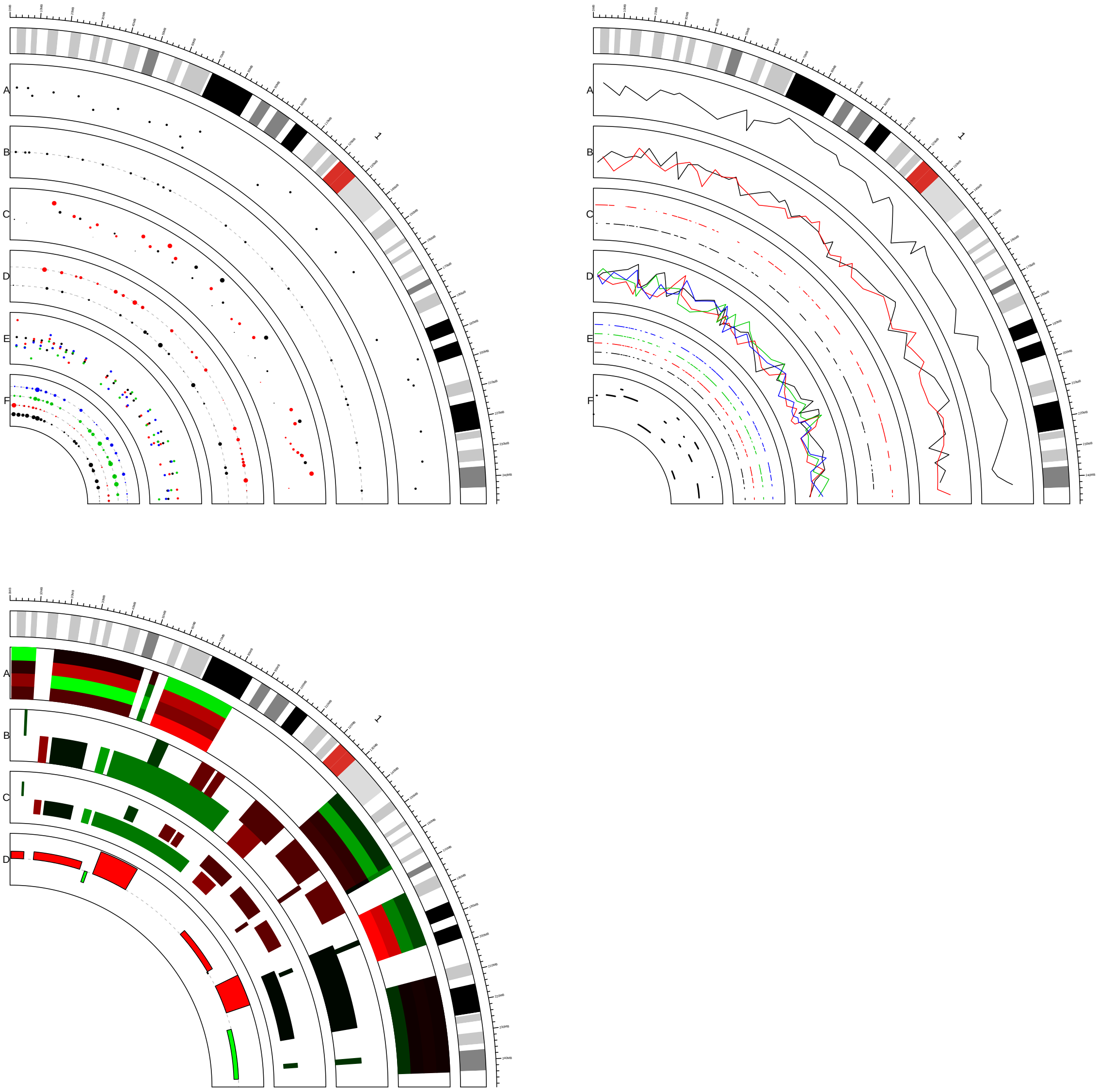

Figure 3: Topleft: Plotting points from A) a data frame with one numeric column; B) a data frame with one numeric column and under `stack` mode; C) a list of two data frames D) a list of two data frames under `stack` mode; E) a data frame with four numeric column; F) a data frame with four numeric column and under `stack` mode. Topright: Plotting lines in differnet ways. Plotting lines from A) a data frame with one numeric column; B) a list of two data frames C) a list of two data frames under `stack` mode; D) a data frame with four numeric column; E) a data frame with four numeric column and under `stack` mode. F) a data frame with one numeric column and `lty` is set to `segment`. Bottomleft: Plotting rectangles in differnet ways. Plotting lines from A) a data frame with four numeric column and under `stack` mode. B) a list of two data frames under `stack` mode; C) and D) adding rectangles with self-defined `panel.fun`.

6 High-level genomic functions

circlize implements several high-level functions which may help to visualize genomic data.

6.1 Position transformation

There is one representative situation when genomic position transformation needs to be applied. For example, there are two sets of regions in a chromosome in which regions in one set regions are quite

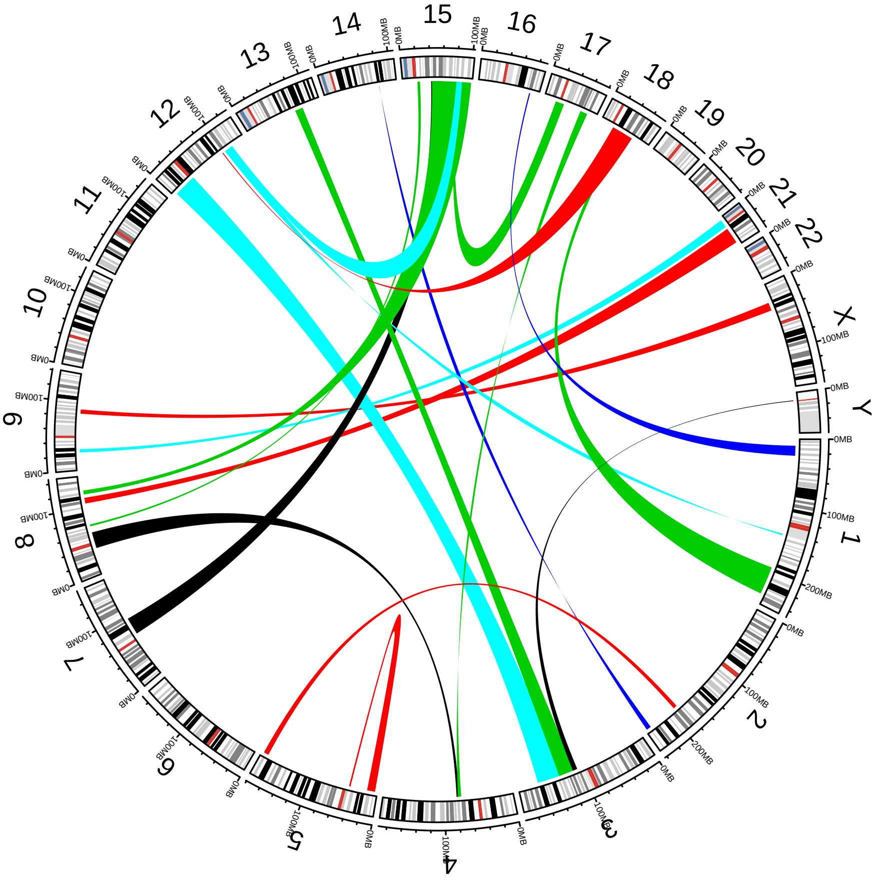

Figure 4: Add links from two sets of regions.

densely to each other and regions in other set are far from others. Heatmap or text is going to be drawn on the next track. If there is no position transformtion, heatmap or text for those dense regions would be overlapped and hard to identify, also ugly to visualize. Thus, a way to transform original positions to new positions would help for the visualization.

Low-level genomic functions such as `circos.genomicPoints` all accept an argument `posTransform` to apply user-defined position transformation. Value for `posTransform` is a self-defined function which only accepts one argument: a data frame with two columns (start position and end position). There is only one requirement for position transformation: Number of rows of regions should be the same before and after the transformation. In *circlize*, there already provides a position transformation function `posTransform.default` which distributes positions uniformly in current chromosome.

To do the transformation:

```
> circos.genomicTrackPlotRegion(data, panel.fun = function(region, value, ...) {
+     circos.genomicPoints(region, value, posTransform = posTransform.default, ...)
+ })
```

There is also a `circos.genomicPosTransformLines` which add a line from untransformed regions to transformed regions.

```
> circos.genomicPosTransformLines(data, posTransform = posTransform.default)
> circos.genomicPosTransformLines(data, posTransform = posTransform.default,
+     horizontalLine = "top")
> circos.genomicPosTransformLines(data, posTransform = posTransform.default,
+     type = "reverse")
```

Please see figure 6 for examples.

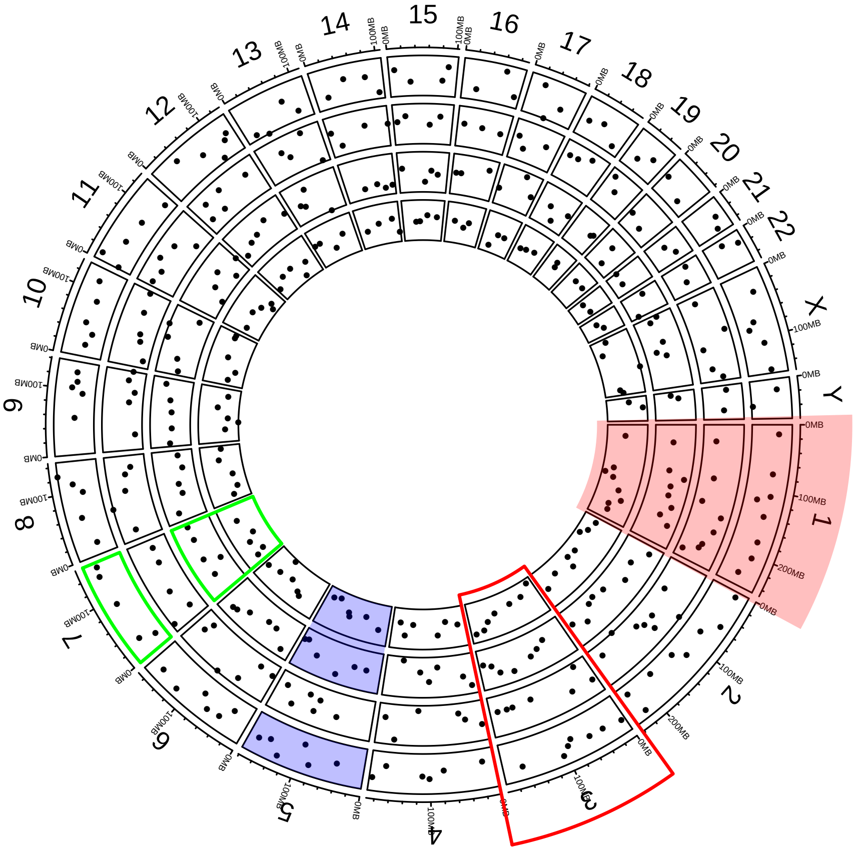

Figure 5: Highlight chromosomes.

6.2 Genomic density and Railfall plot

`circos.genomicDensity` calculate how much a genomic window is covered by regions in `bed`. The input data can be a single data frame or a list of data frames.

```
> circos.genomicDensity(bed)
> circos.genomicDensity(bed, window.size = 1e6)
> circos.genomicDensity(bedlist)
```

Rainfall plot can be used to visualize distribution of regions. On the plot, y-axis corresponds to the distance to neighbour regions (log10-based). So if there is a drop-down on the plot, it means there is a cluster of regions at that area (figure 7). The input data can be a single data frame or a list of data frames.

```
> circos.genoimcRainfall(bed)
> circos.genoimcRainfall(bedlist, col = c("red", "green"))
```

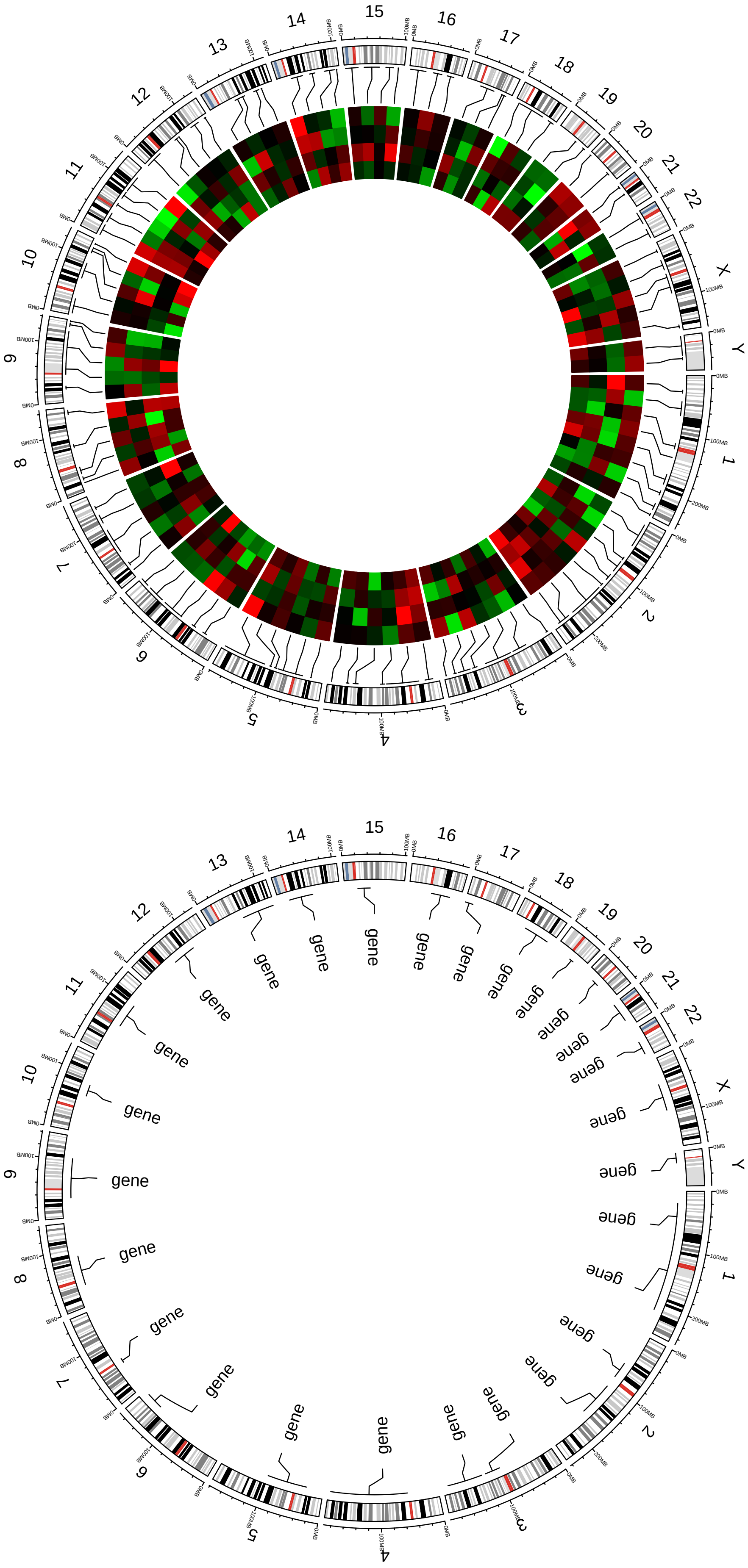

Figure 6: Transformation of genomic positions

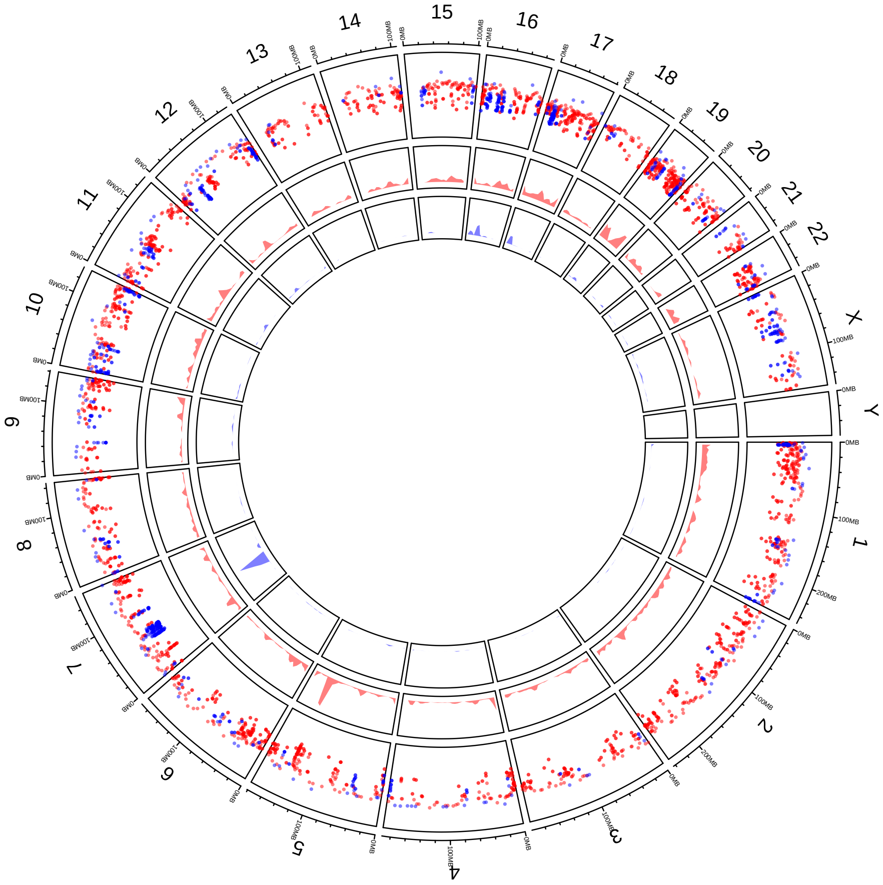

Figure 7: Rainfall plot and genomic density