How to draw Chord Diagram

Zuguang Gu <z.gu@dkfz.de>

October 12, 2014

One unique feature of circular layout is the link (or connector) to represent relations between elements (http://circos.ca/intro/tabular_visualization/). The name of such plot is sometimes called Chord diagram (http://en.wikipedia.org/wiki/Chord_diagram). In `circlize`, it is easy to plot it in a straightforward or customized way.

1 Start from ground

Normally, the relationship can be represented as a matrix in which value in i^{th} row and j^{th} column is kind of strength for the relationship. We first generate an example data. In the example data, letters in upper case is one measurement and letters in lower case is another measurement. The number in the table is the amount of observations in the intersection of two measurements.

```
> set.seed(123)
> mat = matrix(sample(1:100, 18, replace = TRUE), 3, 6)
> rownames(mat) = letters[1:3]
> colnames(mat) = LETTERS[1:6]
> rn = rownames(mat)
> cn = colnames(mat)
> mat
```

```
   A  B  C  D  E  F
a 29 89 53 46 68 90
b 79 95 90 96 58 25
c 41  5 56 46 11  5
```

Sector names in circos plot correspond to the union of row names and column names in the matrix. First let's construct the `factors` variable and calculate `xlim`. Since row names and columns are different, the data range is simply summation in rows or columns respectively.

```
> factors = c(letters[1:3], rev(LETTERS[1:6]))
> factors = factor(factors, levels = factors)
> col_sum = apply(mat, 2, sum)
> row_sum = apply(mat, 1, sum)
> xlim = cbind(rep(0, 9), c(row_sum, rev(col_sum)))
```

Then initialize the circular layout for this table (figure 1). We specify the width of gaps between two measurements by `gap.degree` in `circos.par`. The colors for different measurements are specified by `bg.col` in `circos.trackPlotRegion`.

```
> par(mar = c(1, 1, 1, 1))
> circos.par(cell.padding = c(0, 0, 0, 0),
+     gap.degree = c(2, 2, 10, 2, 2, 2, 2, 2, 10), start.degree = 10/2)
> circos.initialize(factors = factors, xlim = xlim)
> circos.trackPlotRegion(factors = factors, ylim = c(0, 1), bg.border = NA,
+     bg.col = c("red", "green", "blue", rep("grey", 6)), track.height = 0.05,
+     panel.fun = function(x, y) {
+         sector.name = get.cell.meta.data("sector.index")
+         xlim = get.cell.meta.data("xlim")
+         circos.text(mean(xlim), 1.5, sector.name, adj = c(0.5, 0))
+ })
```

Finally the links are added to the plot.

```
> col = c("#FF000020", "#00FF0020", "#0000FF20")
> for(i in seq_len(nrow(mat))) {
+     for(j in rev(seq_len(ncol(mat)))) {
+         circos.link(rn[i], c(sum(mat[i, seq_len(j-1)]), sum(mat[i, seq_len(j)])),
+             cn[j], c(sum(mat[seq_len(i-1), j]), sum(mat[seq_len(i), j])),
+             col = col[i], border = "white")
+     }
+ }
> circos.clear()
```

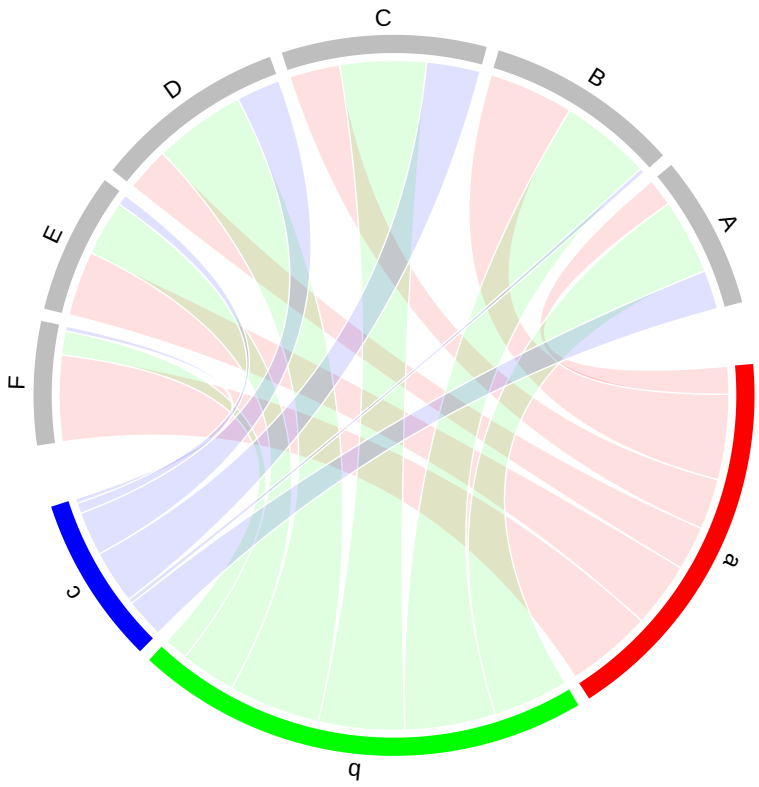

Figure 1: Table in circular layout

2 The pre-defined `chordDiagram` function

A general function `chordDiagram` is already defined in the package.

2.1 Basic usage

We will still use `mat` object in previous section to demonstrate the usage of `chordDiagram`. The most simple usage is just calling `chordDiagram` with `mat` (figure 2, top left).

```
> chordDiagram(mat)
```

The default Chord diagram consists of a track of labels, a track of grids, and links. Under default settings, the grid colors are randomly generated. The link colors are same as colors for grids which correspond to rows. The order of sectors is the order of `union(rownames(mat), colnames(mat))`.

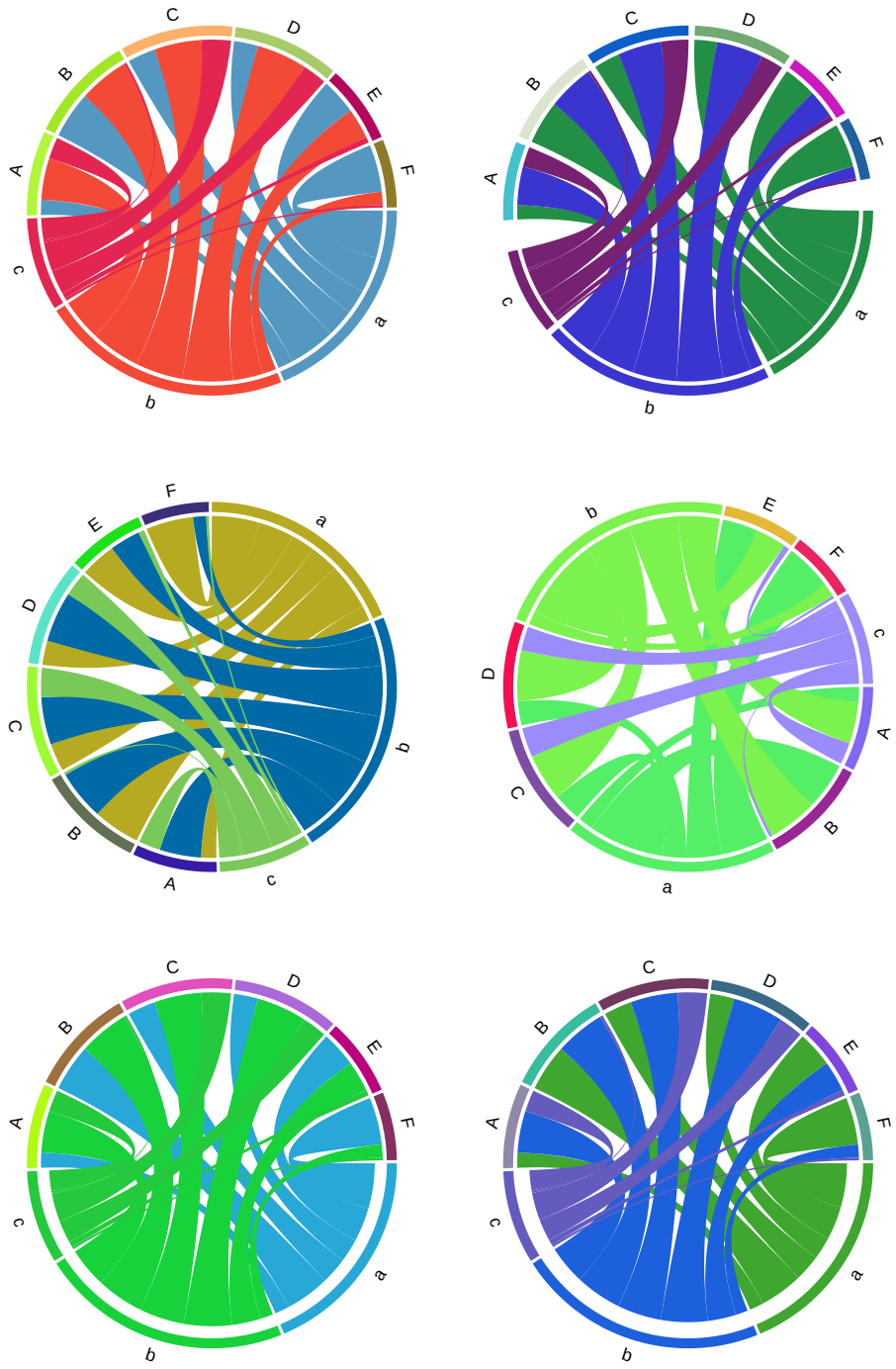

Figure 2: Basic settings for plotting Chord diagram. Top left: default style; Top right: set `gap.degree`; Middle left: set `start.degree`; Middle right: set orders of sectors; Bottom left: set `directional`; Bottom right: set `directional` and `directionGridHeight`

The gaps between sectors can be set through `circos.par` (figure 2, top right). It is useful when rows and columns are different measurements (as in `mat`). Please note since you change default circos graphical settings, you need to use `circos.clear` in the end to reset it.

```
> circos.par(gap.degree = c(rep(2, nrow(mat)-1), 10, rep(2, ncol(mat)-1), 10))
> chordDiagram(mat)
> circos.clear()
```

Similarly, the start degree of the first sector can also be set through `circos.par` (figure 2, middle left).

```
> circos.par(start.degree = 90)
> chordDiagram(mat)
> circos.clear()
```

The order of sectors can be controlled by `order` argument (figure 2, middle right).

```
> chordDiagram(mat, order = c("A", "B", "a", "C", "D", "b", "E", "F", "c"))
```

In some cases, rows and columns represent information of direction. Argument `directional` can be set to illustrate such direction. If `directional` is set to `TRUE`, the links will have unequal height of root (figure 2, bottom). The type of direction can be set through `fromRows` and the difference between the unequal root can be set through `directionGridHeight`.

```
> chordDiagram(mat, directional = TRUE)
> chordDiagram(mat, directional = TRUE, directionGridHeight = 0.06)
> chordDiagram(mat, directional = TRUE, fromRows = FALSE, directionGridHeight = 0.06)
```

2.2 Color settings

Setting colors is also flexible. Colors for grids can be set through `grid.col`. Values of `grid.col` should be a named vector of which names correspond to sector names. If `grid.col` has no name index, the order of `grid.col` corresponds to the order of sector names. As explained before, the default link colors are same as colors for grids which correspond to rows (figure 3, top left).

```
> grid.col = NULL  # just create the variable
> grid.col[letters[1:3]] = c("red", "green", "blue")
> grid.col[LETTERS[1:6]] = "grey"
> chordDiagram(mat, grid.col = grid.col)
```

Transparency of link colors can be set through `transparency` (figure 3, top middle). The value should between 0 to 1 in which 0 means no transparency and 1 means full transparency.

```
> chordDiagram(mat, grid.col = grid.col, transparency = 0.5)
```

Colors for links can be customized by providing a matrix of colors which correspond to `mat`. In the following example, `col_mat` already contains transparency, `transparency` will be disabled if it is set (figure 3, top right).

```
> rand_color = function(n, alpha = 1) {
+     return(rgb(runif(n), runif(n), runif(n), alpha = alpha))
+ }
> col_mat = rand_color(length(mat), alpha = 0.5)
> dim(col_mat) = dim(mat)
> chordDiagram(mat, grid.col = grid.col, col = col_mat)
```

`col` argument can also be a self-defined function which maps values to colors. Here we use `colorRamp2` which is available in this package to generate a function with a list of break points and corresponding colors (figure 3, middle left).

```
> col_fun = colorRamp2(quantile(mat, seq(0, 1, by = 0.1)), rev(heat.colors(11)))
> chordDiagram(mat, grid.col = grid.col, col = col_fun, transparency = 0.5)
```

Sometimes you don't need to generate the whole color matrix. You can just provide colors which correspond to rows or columns so that links from a same row/column will have the same color (figure 3, middle middle/right).

```
> chordDiagram(mat, grid.col = grid.col, row.col = 1:3, transparency = 0.5)
> chordDiagram(mat, grid.col = grid.col, column.col = 1:6, transparency = 0.5)
```

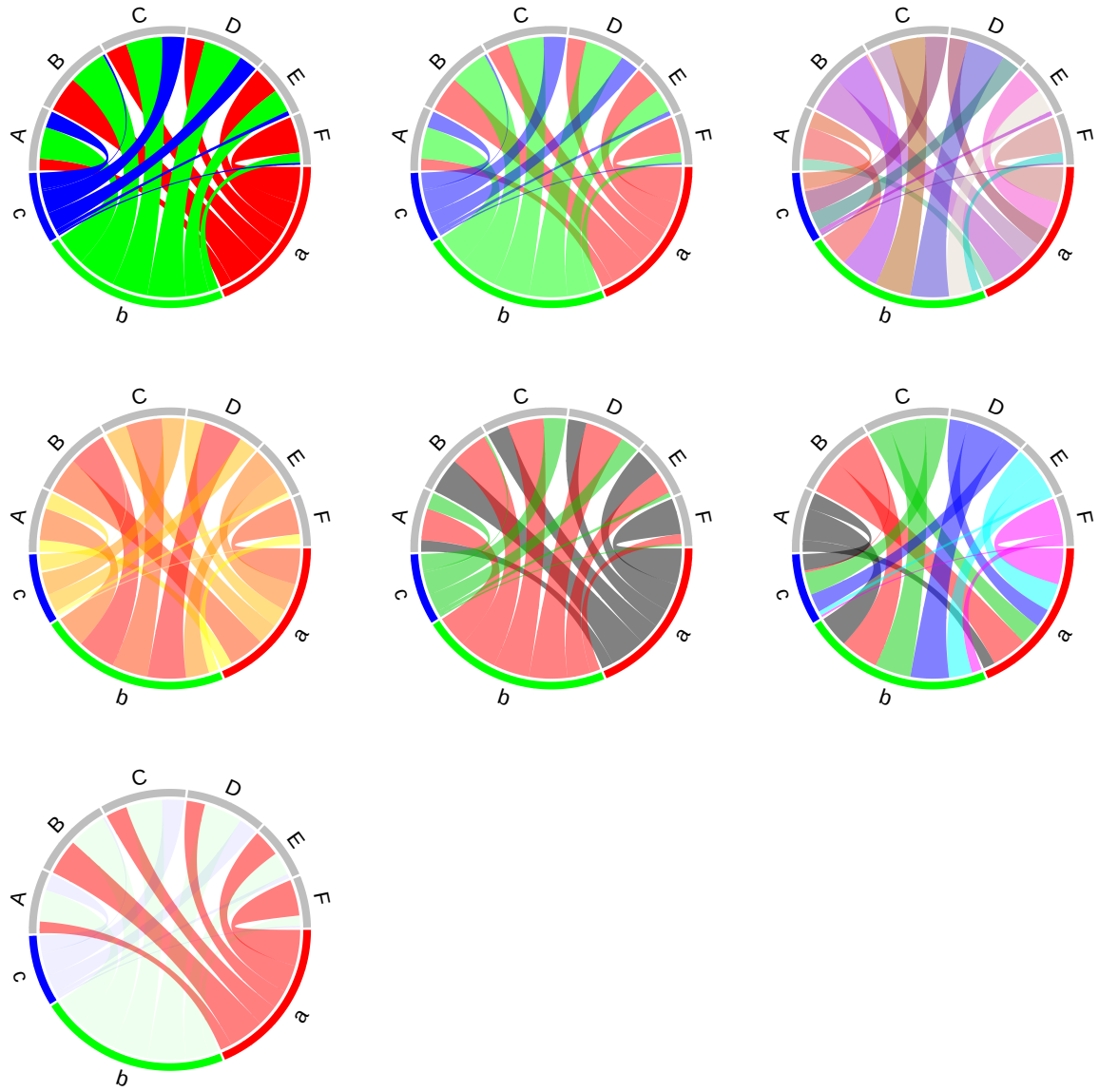

Figure 3: Color settings for plotting Chord diagram. Top left: set `grid.col`; Top middle: set `transparency`; Top right: set `col` as a matrix; Middle left: set `col` as a function; Middle middle: set `row.col`; Middle right: set `column.col`; Bottom left: set `row.col` to highlight links from one row.

With setting `row.col` for example, we can highlight links from one specific sector (figure 3, bottom left).

```
> chordDiagram(mat, grid.col = grid.col, row.col = c("#FF000080", "#00FF0010", "#0000FF10"))
```

Again, if transparency is already set in `col` or `row.col` or `column.col`, `transparency` argument will be disabled if it is set.

2.3 Advanced usage

Although `chordDiagram` provides default style which is enough for most visualization tasks, still you can have more fine-tune on the plot.

By default, there is a track for labels and a track for grids. These two tracks can be controlled by `annotationTrack`. Available values for this argument are `grid` and `name` (figure 4, top left).

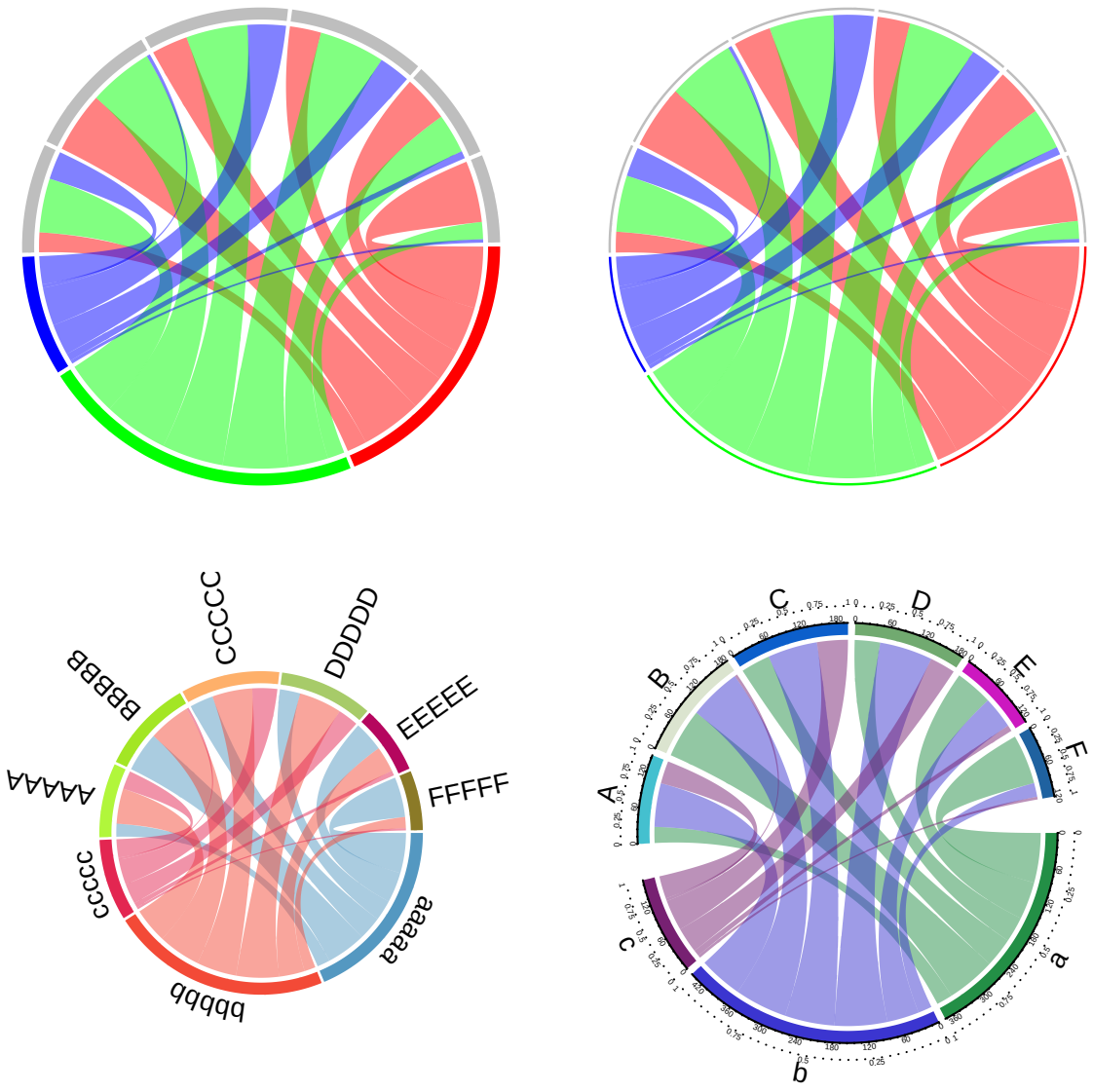

Figure 4: Advanced Chord diagram plotting

```
> chordDiagram(mat, grid.col = grid.col, annotationTrack = "grid", transparency = 0.5)
> circos.info()

All your sectors:
[1] "a" "b" "c" "A" "B" "C" "D" "E" "F"

All your tracks:
[1] 1

Your current sector.index is F
Your current track.index is 1
```

The height of annotation tracks can be set through `annotationTrackHeight` which corresponds to values in `annotationTrack` (figure 4, top right). The value in `annotationTrackHeight` is the percentage to the radius of unit circle.

```
> chordDiagram(mat, grid.col = grid.col, annotationTrack = "grid",
+     annotationTrackHeight = 0.01, transparency = 0.5)
```

```
> chordDiagram(mat, grid.col = grid.col, annotationTrack = c("name", "grid"),
+     annotationTrackHeight = c(0.03, 0.01), transparency = 0.5)
```

Several blank tracks can be allocated before Chord diagram is plotted. Then self-defined graphics can be added to these blank tracks afterwards. The number of pre-allocated tracks can be set through `preAllocateTracks`.

```
> chordDiagram(mat, annotationTrack = NULL, preAllocateTracks = 3)
> circos.info()

All your sectors:
[1] "a" "b" "c" "A" "B" "C" "D" "E" "F"

All your tracks:
[1] 1 2 3

Your current sector.index is F
Your current track.index is 3
```

The default settings for pre-allocated tracks are:

```
> list(ylim = c(0, 1),
+     track.height = circos.par("track.height"),
+     bg.col = NA,
+     bg.border = NA,
+     bg.lty = par("lty"),
+     bg.lwd = par("lwd"))
```

The default settings for pre-allocated tracks can be overwritten by specifying `preAllocateTracks` as a list.

```
> chordDiagram(mat, annotationTrack = NULL,
+     preAllocateTracks = list(track.height = 0.3))
```

If more than one tracks need to be pre-allocated, just specify `preAllocateTracks` as a list which contains settings for each track:

```
> chordDiagram(mat, annotationTrack = NULL,
+     preAllocateTracks = list(list(track.height = 0.1),
+                              list(bg.border = "black")))
```

In `chordDiagram`, there is no argument to control the style of labels/sector names. But this can be done by first pre-allocating a blank track and customizing the labels in it later. In the following example, one track is firstly allocated and a Chord diagram is plotted without label track. Later, the first track is updated with setting `facing` of labels (figure 4, bottom left).

```
> chordDiagram(mat, annotationTrack = "grid", preAllocateTracks = list(track.height = 0.3))
> circos.trackPlotRegion(track.index = 1, panel.fun = function(x, y) {
+     xlim = get.cell.meta.data("xlim")
+     ylim = get.cell.meta.data("ylim")
+     sector.name = get.cell.meta.data("sector.index")
+         label = paste0(rep(sector.name, 5), collapse="")
+     if(sector.name %in% rn) {
+         circos.text(mean(xlim), ylim[1], label, facing = "bending", adj = c(0.5, 0))
+     }
+     if(sector.name %in% cn) {
+         circos.text(mean(xlim), ylim[1], label, facing = "clockwise", adj = c(0, 0.5))
+     }
+ }, bg.border = NA)
```

Since how to add graphics in the blank tracks is determined by users, it would be flexible to customize the Chord diagram with different styles. In figure 4 (bottom right), two axes are added in every sector.

2.4 Visualization of other matrix

Rows and columns in `mat` can also overlap. In this case, direction of the link is quite important in visualization (figure 5, top left).

```
> set.seed(123)
> mat = matrix(sample(100, 25), 5)
> rownames(mat) = letters[1:5]
> colnames(mat) = letters[1:5]
> mat
```

```
   a  b  c  d  e
a 29  5 87 77 72
b 79 50 98 21 55
c 41 83 60  4 90
d 86 51 94 27 85
e 91 42  9 78 81
```

```
> chordDiagram(mat, directional = TRUE, row.col = 1:5, transparency = 0.5)
```

`chordDiagram` can also be used to visualize symmetric matrix. If `symmetric` is set to `TRUE`, only lower triangular matrix without the diagonal will be used (figure 5, top right). Of course, your matrix should be symmetric.

```
> chordDiagram(cor(mat), symmetric = TRUE,
+     col = colorRamp2(c(-1, 0, 1), c("green", "white", "red")), transparency = 0.5)
```

If you don't need self-loop for which two roots of a link is in a same sector, just set corresponding values to 0 in `mat` (figure 5, bottom).

```
> for(cn in intersect(rownames(mat), colnames(mat))) {
+         mat[cn, cn] = 0
+ }
> mat
```

```
   a  b  c  d  e
a  0  5 87 77 72
b 79  0 98 21 55
c 41 83  0  4 90
d 86 51 94  0 85
e 91 42  9 78  0
```

```
> chordDiagram(mat, directional = TRUE, row.col = 1:5, transparency = 0.5)
```

2.5 Compare two Chord Diagrams

Normally, values in `mat` are normalized to the summation, and each value is put to the circle according to its percentage, which means the width for each link only represents kind of relative value. However, when comparing two Chord Diagrams, it is necessary that unit width of links should represent in a same scale. This problem can be solved by adding more blank gaps to the Chord Diagram which has smaller summation.

First, let's plot a Chord Diagram. In this Chord Diagram, we set larger gaps between rows and columns for better visualization. Axis on the grid illustrates scale of the values.

```
> mat1 = matrix(sample(20, 25, replace = TRUE), 5)
> gap.degree = c(rep(2, 4), 10, rep(2, 4), 10)
> circos.par(gap.degree = gap.degree, start.degree = -10/2)
> chordDiagram(mat1, directional = TRUE, grid.col = rep(1:5, 2), transparency = 0.5)
> for(si in get.all.sector.index()) {
+     circos.axis(labels.cex = 0.5, sector.index = si, track.index = 2)
+ }
> circos.clear()
```

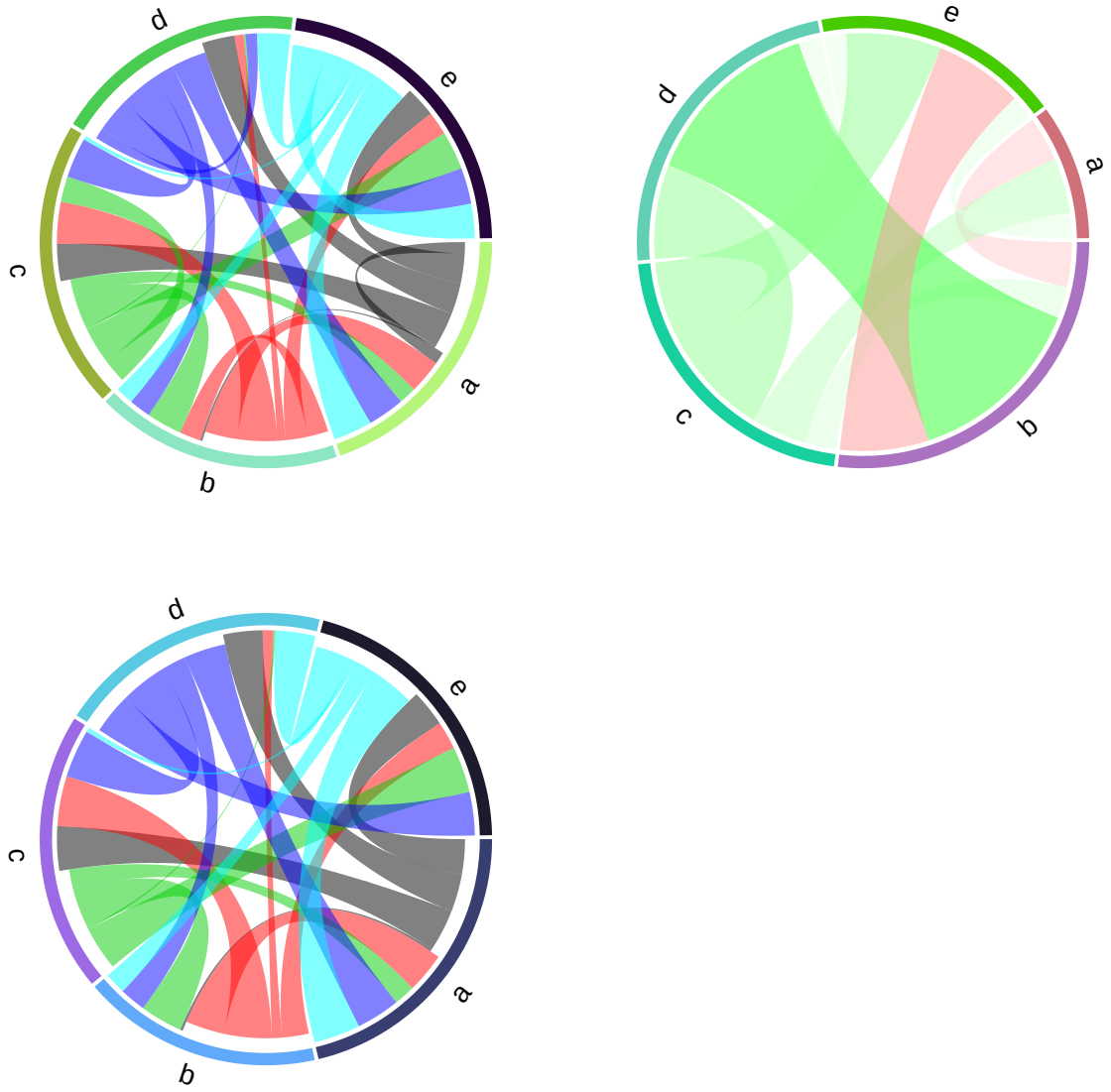

Figure 5: Chord diagram for other types of matrix. Top left: with directions; Top right: symmetric correlation matrix; Top bottom: without self-loop

The second matrix only has half the values in `mat1`.

```
> mat2 = mat1 / 2
```

If the second Chord Diagram is plotted in the way as the first one, the two diagrams will looks the same which makes the comparison not straightforward. Now what we want to compare is also the absolute values. For example, if the matrix contains the amount of transmission from one state to another state, then the interest is to see which diagram has more transmissions.

First we calculate the percentage of `mat2` in `mat1`. And then calculate the degree which corresponds to the summation difference. In the following code, `360 - sum(gap.degree)` is the degree for summation of `mat1` and `blank_degree` corresponds the summation difference.

```
> percent = sum(mat2) / sum(mat1)
> blank_degree = (360 - sum(gap.degree)) * (1 - percent)
```

Since we have the additional blank gap, we can set it to `circos.par` and plot the second Chord Diagram.

```
> gap.degree = c(rep(2, 4), blank_degree/2 + 10, rep(2, 4), blank_degree/2 + 10)
> circos.par(gap.degree = gap.degree, start.degree = -(blank_degree/2 + 10)/2)
> chordDiagram(mat2, directional = TRUE, grid.col = rep(1:5, 2), transparency = 0.5)
> for(si in get.all.sector.index()) {
+     circos.axis(labels.cex = 0.5, sector.index = si, track.index = 2)
+ }
> circos.clear()
```

Now the scale of the two Chord Diagrams (figure 6) are the same if you look at the scale of axes in the two diagrams.

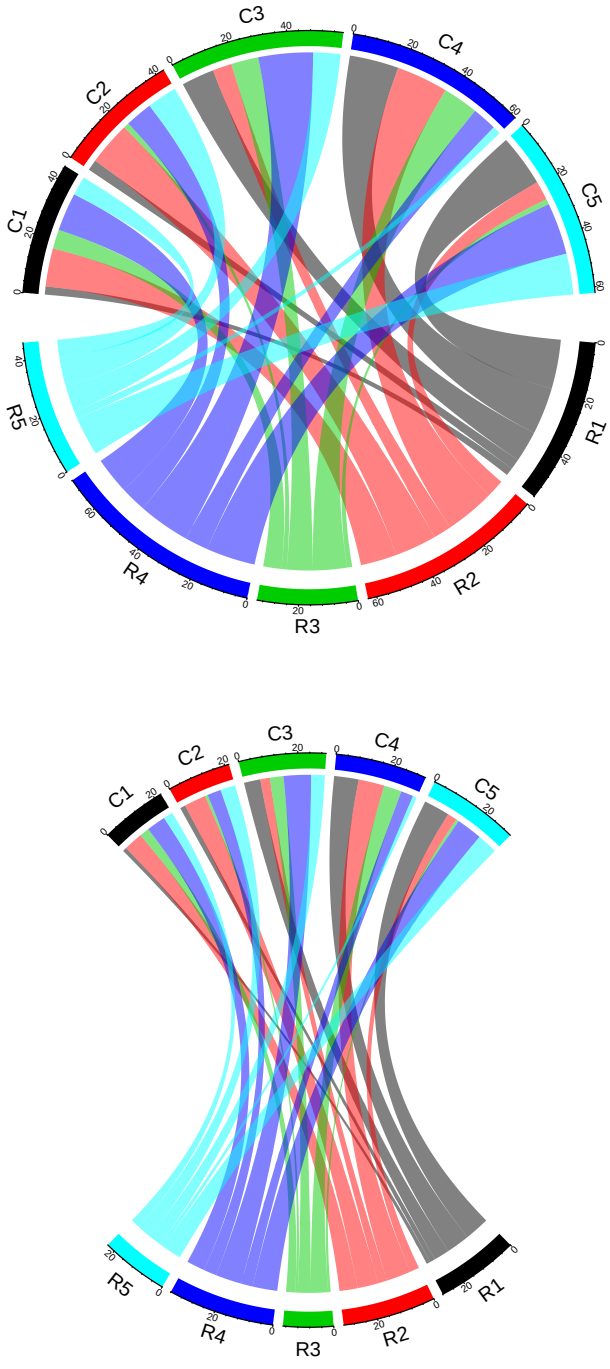

Figure 6: Compare two Chord Diagrams and make them in same scale