How to make Chord Diagram

Zuguang Gu <z.gu@dkfz.de>

December 8, 2014

One unique feature of circular layout is the links (or connectors) 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 values in i^{th} row and j^{th} column are some kind of strength for the relationship. Let's assume we have a transition matrix in which rows correspond to starting states and columns correspond to ending states. Numbers in the matrix are the amount of observations that transit between two states.

```
library(circlize)

mat = matrix(1:18, 3, 6)
rownames(mat) = paste0("S", 1:3)
colnames(mat) = paste0("E", 1:6)

mat
```

```
##    E1 E2 E3 E4 E5 E6
## S1  1  4  7 10 13 16
## S2  2  5  8 11 14 17
## S3  3  6  9 12 15 18
```

Sector names in the circos plot correspond to the union of row names and column names from the matrix. To initialize the circos plot, we need to construct `factors` and `xlim`. Since here row names and columns are different, the data range is simply the summation in rows or columns respectively. Just be careful with the order of `factors` and `xlim`.

```
rn = rownames(mat)
cn = colnames(mat)

factors = c(rn, cn)
factors = factor(factors, levels = factors)

col_sum = apply(mat, 2, sum)
row_sum = apply(mat, 1, sum)
xlim = cbind(rep(0, length(factors)), c(row_sum, col_sum))
```

The next step is to create a new track to put the grids which represent states and the names for the states. In following code, colors for different states are specified by `bg.col` in `circos.trackPlotRegion`. You may meet several warning messages saying there are points out of plotting regions, but it is totally fine because we create a new track and fill in with colors for the whole plotting regions then add text outside each region.

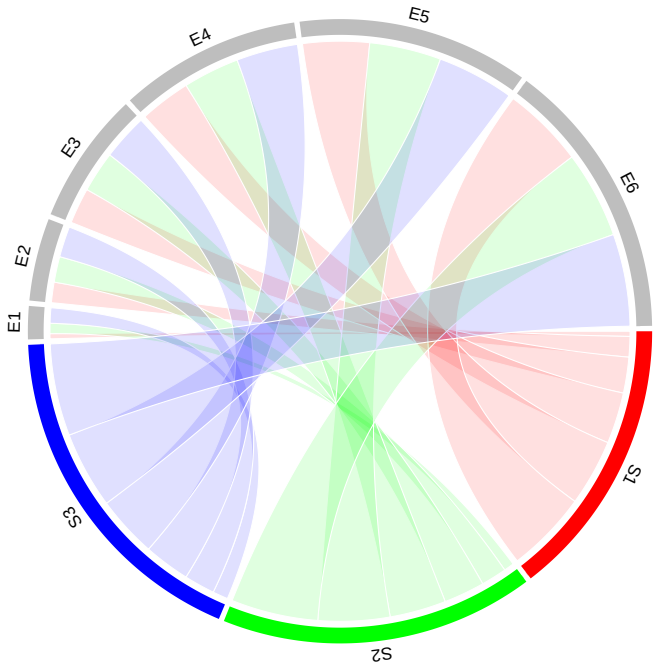

Figure 1: Matrix in circular layout.

```
par(mar = c(1, 1, 1, 1))
circos.par(cell.padding = c(0, 0, 0, 0))
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 add links to the plot (figure 1).

```
col = c("#FF000020", "#00FF0020", "#0000FF20")
for(i in seq_len(nrow(mat))) {
    for(j in 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()
```

Above codes show the basic idea to implement Chord Diagram in **circlize**. Of course you can customize the plot in your own way such as changing the order for ending states 1 to 6 so that the links are not over-twisted.

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 A).

```
set.seed(999)
chordDiagram(mat)
circos.clear()
```

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 with 50% transparency, that is why we add `set.seed` to make sure that users can make same colors with the same code. By default, the link colors are same as grid colors which correspond to rows. The order of sectors is the order of `union(rownames(mat), colnames(mat))`.

Since Chord Diagram is implemented by **circlize** package, like normal circos plot, there are a lot of settings that can be tuned.

The gaps between sectors can be set through `circos.par` (figure 2 B). 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 for the first sector can also be set through `circos.par` (figure 2 C).

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

The order of sectors can be controlled by `order` argument (figure 2 D). Of course, the length of `order` vector should be same as the number of sectors.

```
chordDiagram(mat, order = c("S1", "E1", "E2", "S2", "E3", "E4", "S3", "E5", "E6"))
```

2.2 Colors

Setting colors is 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 grids which correspond to rows (figure 3 A).

```
grid.col = NULL  # just create the variable
grid.col[rn] = c("red", "green", "blue")
grid.col[cn] = "grey"
chordDiagram(mat, grid.col = grid.col)
```

Transparency of link colors can be set through `transparency` (figure 3 B). The value should between 0 to 1 in which 0 means no transparency and 1 means full transparency. Default transparency is 0.5.

```
chordDiagram(mat, grid.col = grid.col, transparency = 0)
```

Colors for links can be customized by providing a matrix of colors which correspond to `mat`. In the following example, we use `rand_color` which is shipped by **circlize** package to generate a random color matrix. Note since `col_mat` already contains transparency, `transparency` will be ignored if it is set (figure 3 C).

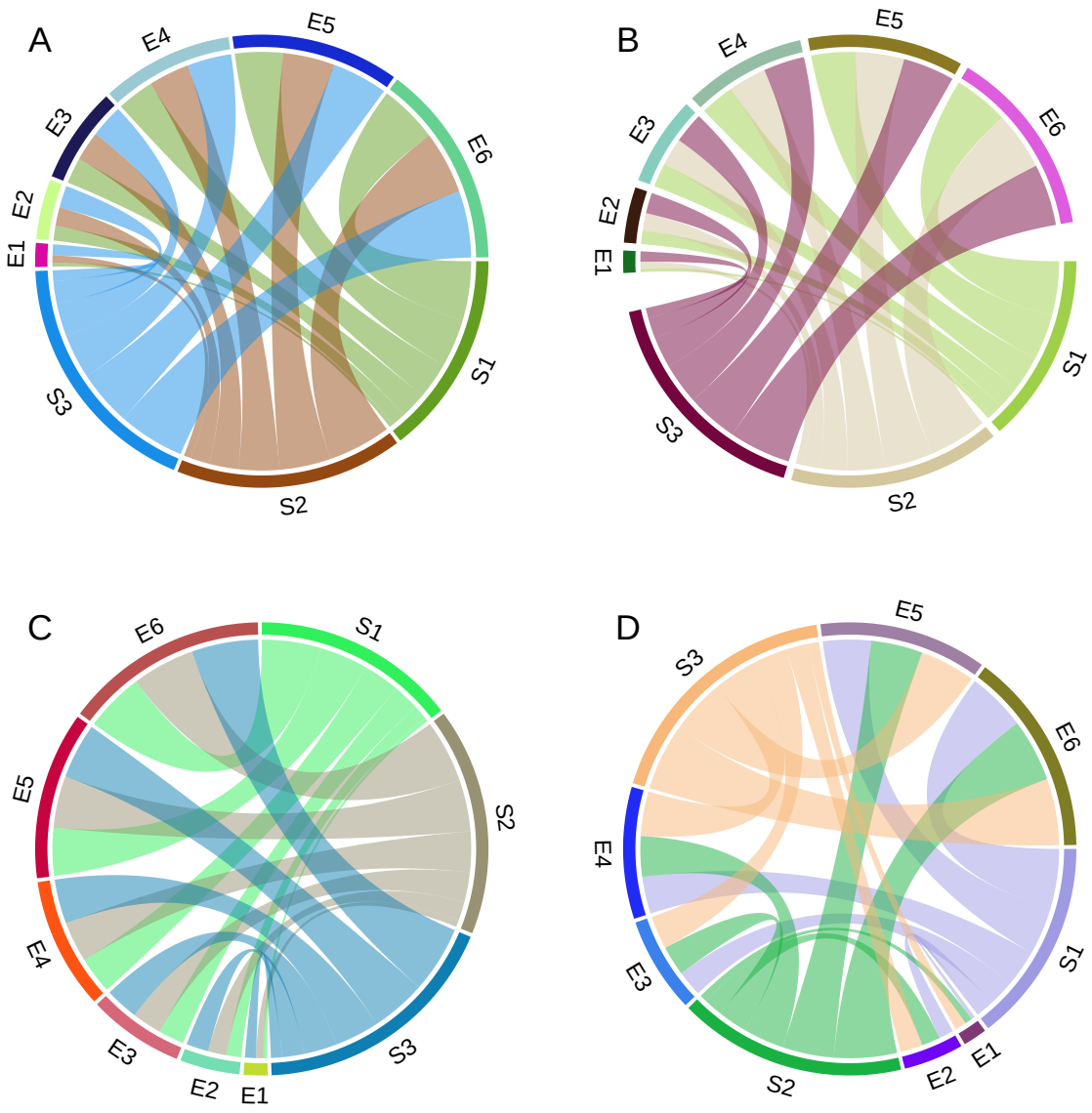

Figure 2: Basic usages of `chordDiagram`. A) default style; B) set `gap.degree`; C) set `start.degree`; D) set orders of sectors.

```
col_mat = rand_color(length(mat), transparency = 0.5)
dim(col_mat) = dim(mat)  # to make sure it is a matrix
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 **circlize** to generate a function with a list of break points and corresponding colors (figure 3 D).

```
col_fun = colorRamp2(quantile(mat, seq(0, 1, length.out = 18)), rev(rainbow(18)))
chordDiagram(mat, grid.col = grid.col, col = col_fun)
```

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 E, F). Here note values of colors can be set as numbers, color names or hex code, same as in the traditional R graphics.

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

To emphasize again, transparency of links can be included in `col` or `row.col` or `column.col`, if transparency is already set there, `transparency` argument will be ignored if it is set.

2.3 Other Style of links

`link.lwd`, `link.lty` and `link.border` controls the style of links. All these three parameters can be set either a single scalar or a matrix with names.

If it is set as a single scale, it means all links share the same style (figure 4 A).

```
chordDiagram(mat, grid.col = grid.col, link.lwd = 2, link.lty = 2, link.border = "black")
```

If it is set as a matrix, it should correspond to rows and columns in `mat` (figure 4 B).

```
lwd_mat = matrix(1, nrow = nrow(mat), ncol = ncol(mat))
rownames(lwd_mat) = rownames(mat)
colnames(lwd_mat) = colnames(mat)
lwd_mat[mat > 12] = 2

border_mat = matrix(NA, nrow = nrow(mat), ncol = ncol(mat))
rownames(border_mat) = rownames(mat)
colnames(border_mat) = colnames(mat)
border_mat[mat > 12] = "black"

chordDiagram(mat, grid.col = grid.col, link.lwd = lwd_mat, link.border = border_mat)
```

The matrix is not necessary to have same number of rows and columns as in `mat`. It can also be a sub matrix (figure 4 C). For rows or columns of which the corresponding values are not specified in the matrix, default values are filled in. But whenever it is a full matrix or a sub matrix, it must have row names and column names so that the settings can be mapped to the correct links.

```
border_mat2 = matrix("black", nrow = 1, ncol = ncol(mat))
rownames(border_mat2) = rownames(mat)[2]
colnames(border_mat2) = colnames(mat)

chordDiagram(mat, grid.col = grid.col, link.lwd = 2, link.border = border_mat2)
```

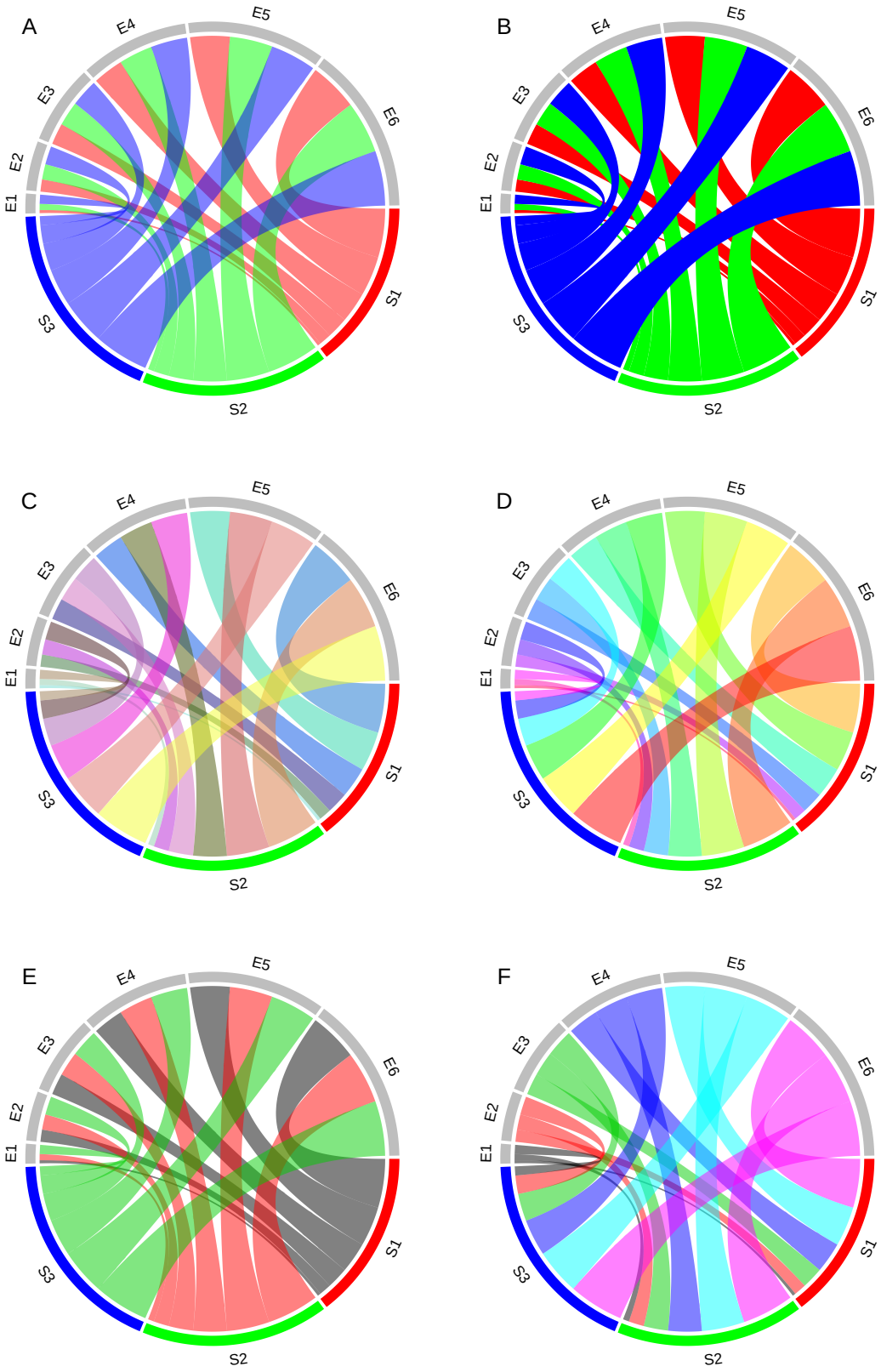

Figure 3: Color settings in `chordDiagram`. A) set `grid.col`; B) set `transparency`; C) set `col` as a matrix; D) set `col` as a function; E) set `row.col`; F) set `column.col`.

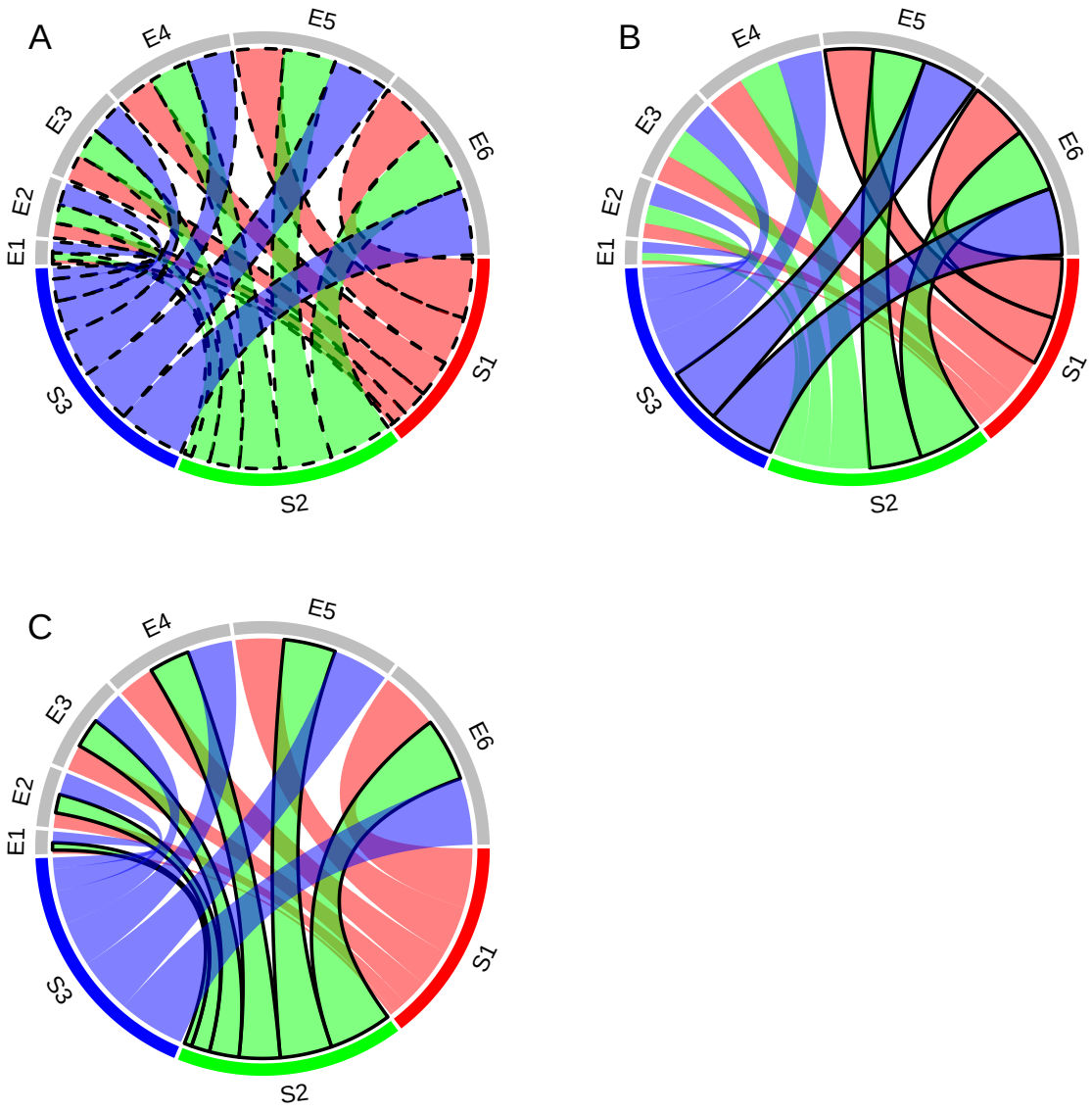

Figure 4: Link style settings in `chordDiagram`. A) set as scalar; B) set as matrix; C) set as sub matrix.

2.4 Highlight links

Sometimes we want to highlight some links to emphasize the importance of such connections. Highlighting by different link styles are introduced in previous section and here we focus on highlighting by colors.

The key point for highlighting by colors is to set different colors for different links. For example, if we want to highlight links which correspond to state "S1", we can set `row.col` with different transparency (figure 5 A).

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

We can also highlight links with values larger than a cutoff (figure 5 B, C). There are at least two ways to do it. First, construct a color matrix and set links with small values to full transparency.

```
col_mat[mat < 12] = "#00000000"
chordDiagram(mat, grid.col = grid.col, col = col_mat)
```

Second, use a color function to generate different colors with differnet values.

```
col_fun = function(x) ifelse(x < 12, "#00000000", "#FF000080")
chordDiagram(mat, grid.col = grid.col, col = col_fun)
```

Just remember if you want to highlight links by colors, make sure to set high or full transparency to the links that you want to ignore.

2.5 Directional matrix

In some cases, rows and columns represent information of direction. Argument `directional` is used to illustrate such direction. If `directional` is set to `TRUE`, the links will have unequal height of root (figure 6 A). The type of direction can be set through `fromRows` and the difference between the unequal root can be set through `diffHeight` (figure 6 B, C). In `chordDiagram`, the starting foot of the link is shorter than the ending foot to give people the feeling that the link is moving out.

```
chordDiagram(mat, directional = TRUE)
chordDiagram(mat, directional = TRUE, diffHeight = 0.08)
chordDiagram(mat, directional = TRUE, fromRows = FALSE)
```

Row names and column names in `mat` can also overlap. In this case, showing directions of the link is quite important in visualization (figure 6 D).

```
mat2 = matrix(sample(100, 35), nrow = 5)
rownames(mat2) = letters[1:5]
colnames(mat2) = letters[1:7]
mat2
```

```
##    a  b  c  d  e  f  g
## a 84 96 83 68 99 34 93
## b 41 98 44 57 63 10  9
## c 11 20 36  4 13 40 27
## d 43 33  5 79 87 62 15
## e 52 21 73 22 69 61 85
```

```
chordDiagram(mat2, directional = TRUE, row.col = 1:5)
```

If you don't need self-loop for which two roots of a link are in a same sector, just set corresponding values to 0 in `mat2` (figure 6 E).

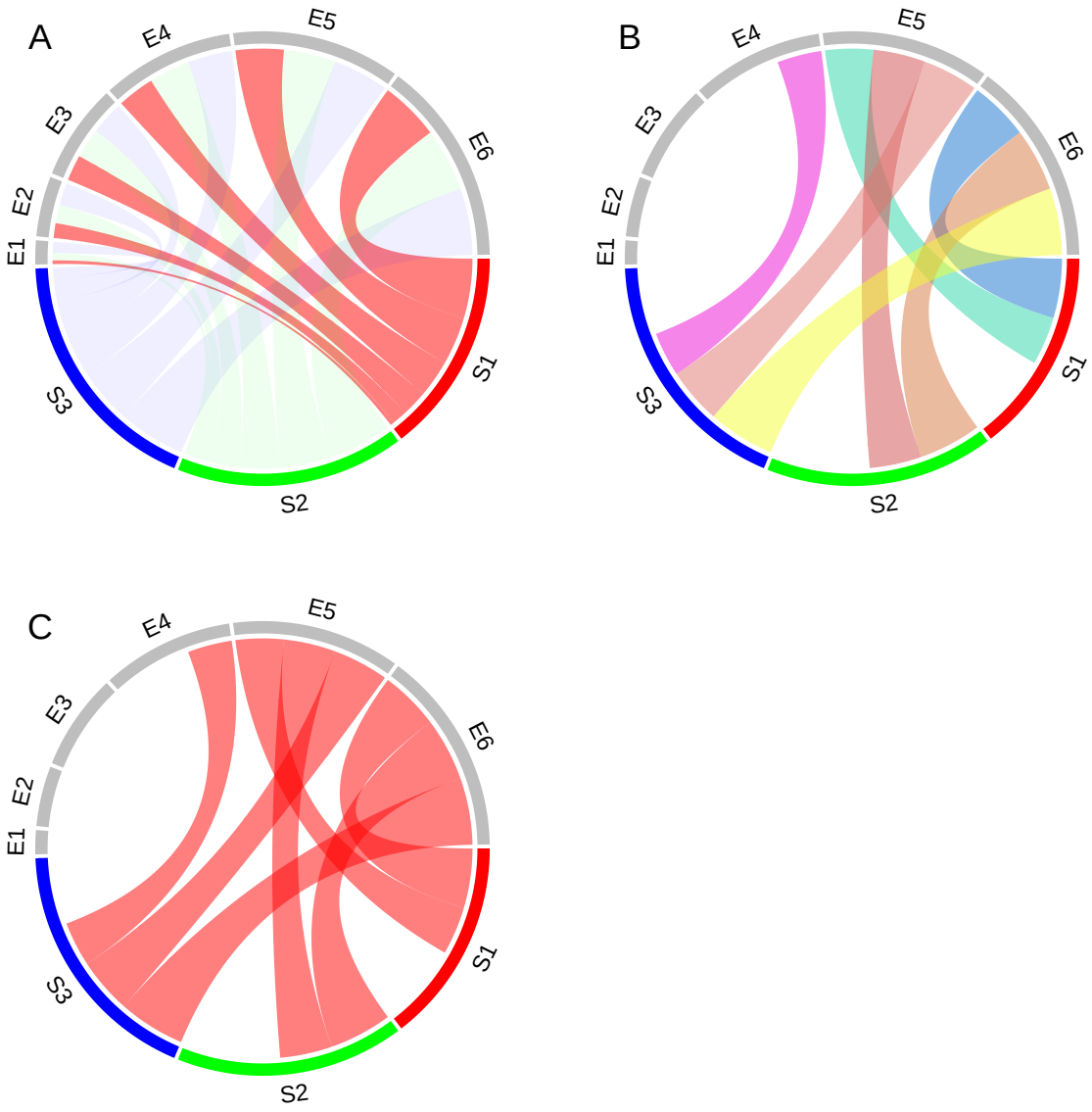

Figure 5: Highlight links by colors. A) set `row.col`; B) set by matrix; C) set by color function.

```
mat3 = mat2
for(cn in intersect(rownames(mat3), colnames(mat3))) {
    mat3[cn, cn] = 0
}
mat3
```

```
##    a  b  c  d  e  f  g
## a  0 96 83 68 99 34 93
## b 41  0 44 57 63 10  9
## c 11 20  0  4 13 40 27
## d 43 33  5  0 87 62 15
## e 52 21 73 22  0 61 85
```

```
chordDiagram(mat3, directional = TRUE, row.col = 1:5)
```

2.6 Symmetric matrix

`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. Of course, your matrix should be symmetric. In figure 7, you can see the difference with specifying `symmetric` or not when visualizing a symmetric matrix.

```
mat3 = matrix(rnorm(100), 10)
colnames(mat3) = letters[1:10]
chordDiagram(cor(mat3), symmetric = TRUE,
    col = colorRamp2(c(-1, 0, 1), c("green", "white", "red")))
```

3 Advanced usage

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

3.1 Organization of tracks

By default, `chordDiagram` which utilizes circos graphics functions will create two tracks, one track for labels and one track for grids.

```
chordDiagram(mat)
circos.info()
```

```
## All your sectors:
## [1] "S1" "S2" "S3" "E1" "E2" "E3" "E4" "E5" "E6"
##
## All your tracks:
## [1] 1 2
##
## Your current sector.index is E6
## Your current track.index is 2
```

These two tracks can be controlled by `annotationTrack`. Available values for this argument are `grid` and `name`. The height of annotation tracks can be set through `annotationTrackHeight` which corresponds to values in `annotationTrack` (figure 8 A, B, C). The value in `annotationTrackHeight` is the percentage to the radius of unit circle.

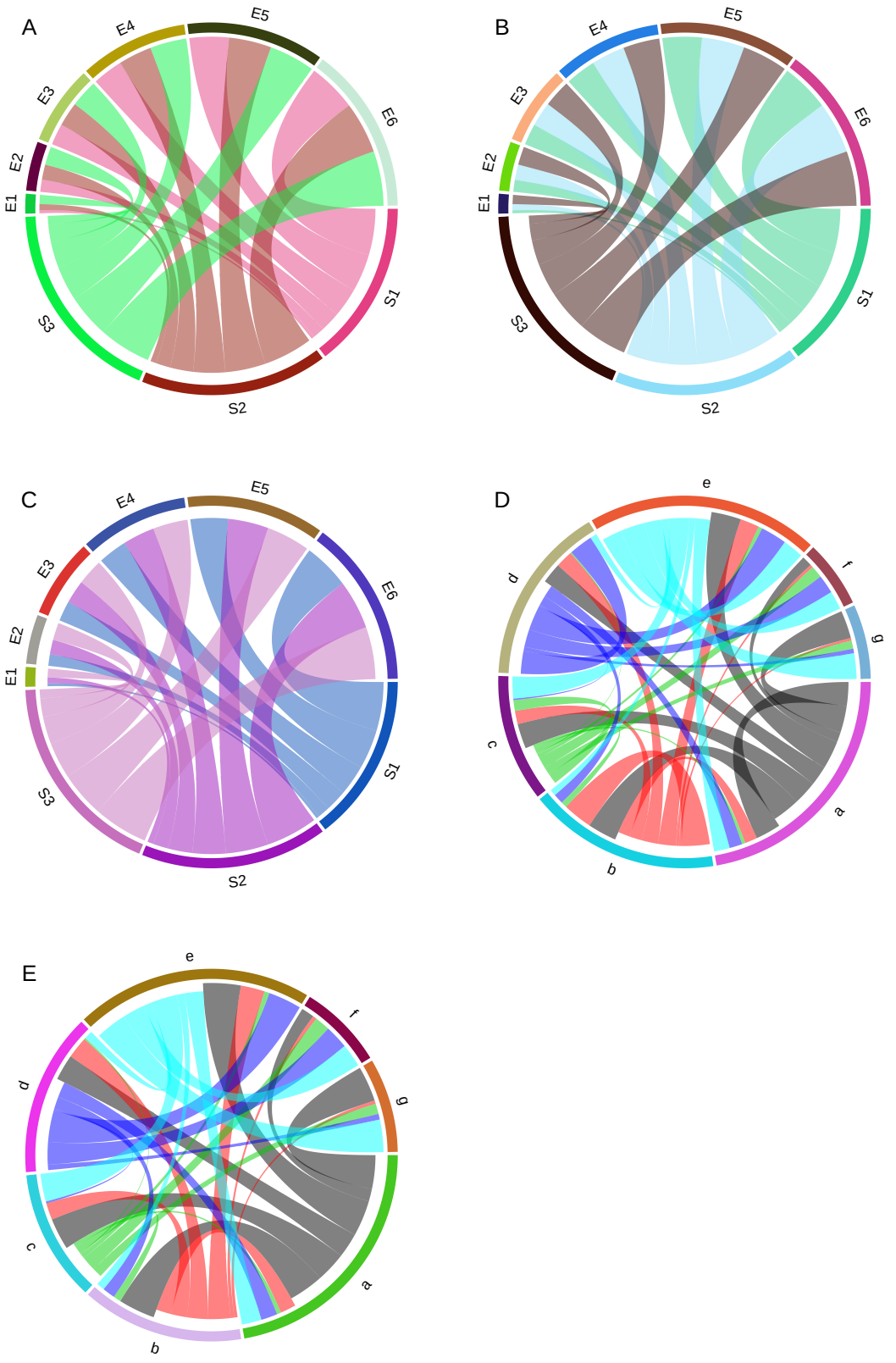

Figure 6: Visualization of directional matrix. A) with default settings; B) set difference of two feet of links; C) set the starting feet; D) row names and column names have overlaps; E) remove self-loop.

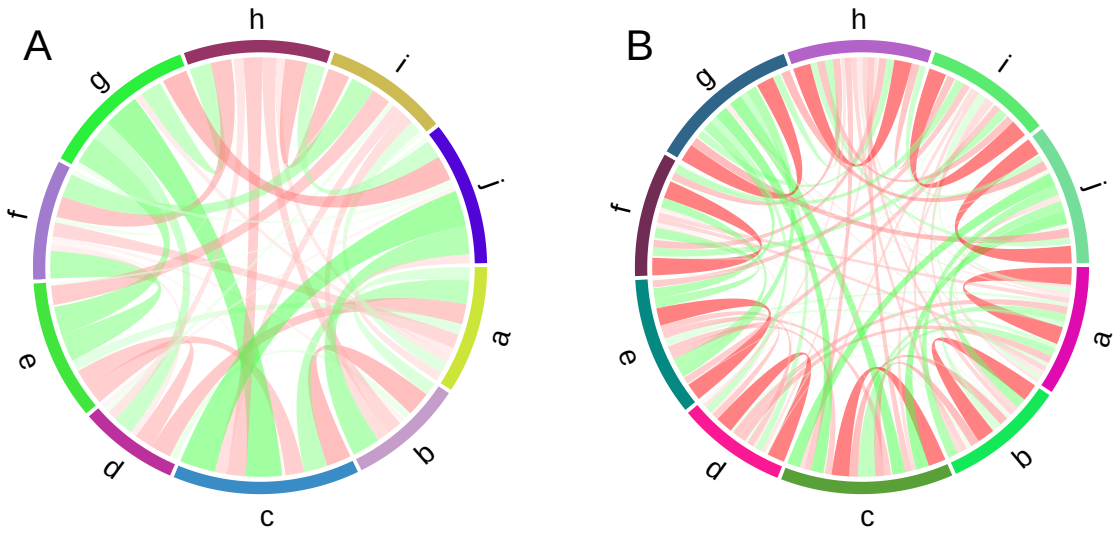

Figure 7: Visualization of symmetric matrix. A) set `symmetric` to `TRUE`; B) set `symmetric` to `FALSE`.

```
chordDiagram(mat, grid.col = grid.col, annotationTrack = "grid")
chordDiagram(mat, grid.col = grid.col, annotationTrack = c("name", "grid"),
    annotationTrackHeight = c(0.03, 0.01))
chordDiagram(mat, annotationTrack = NULL)
```

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

```
chordDiagram(mat, preAllocateTracks = 2)
circos.info()

## All your sectors:
## [1] "S1" "S2" "S3" "E1" "E2" "E3" "E4" "E5" "E6"
##
## All your tracks:
## [1] 1 2 3 4
##
## Your current sector.index is E6
## Your current track.index is 4
```

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))
circos.info(sector.index = "S1", track.index = 1)
```

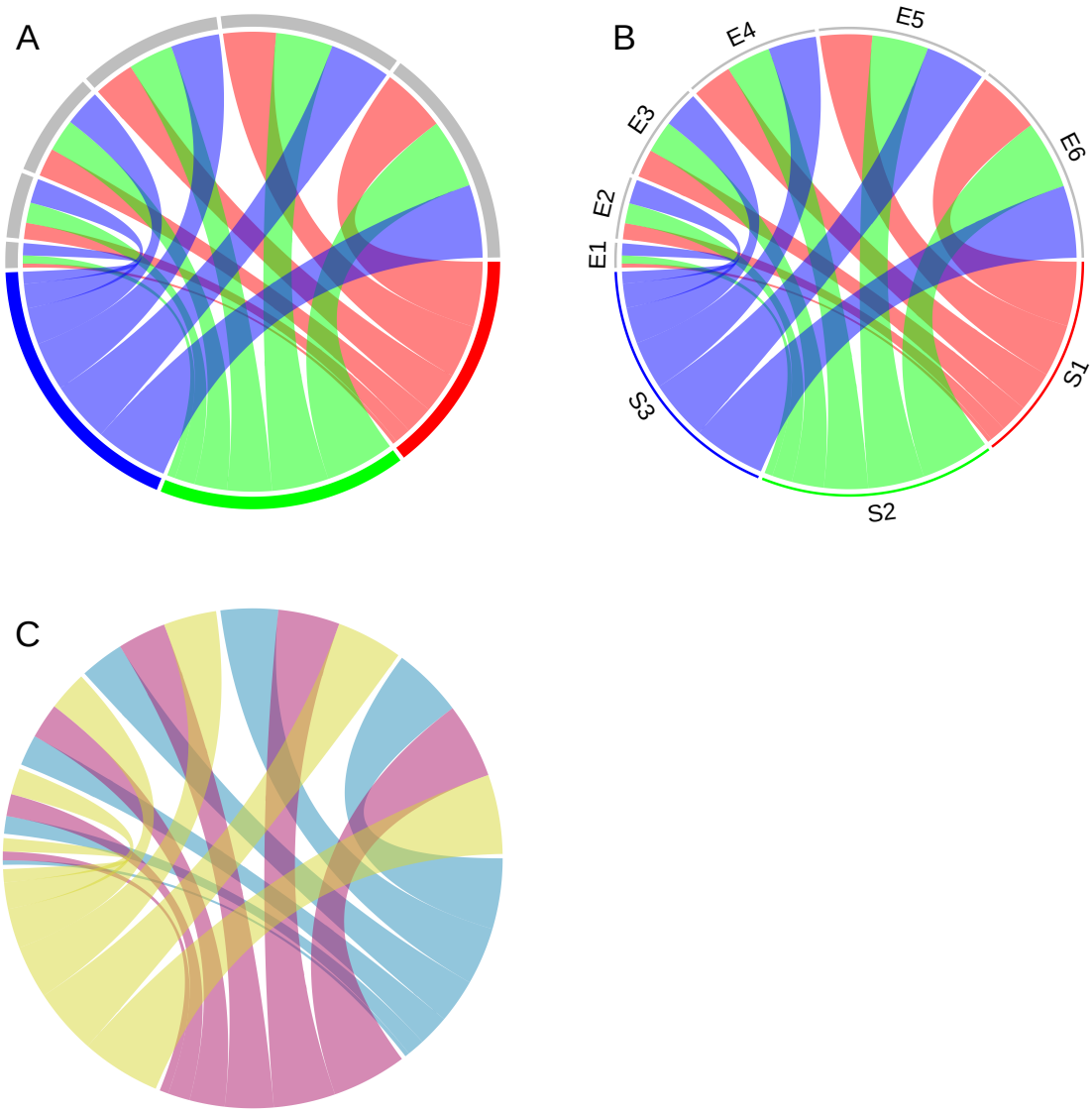

Figure 8: Track organization in `chordDiagram`. A) only show the grid track; B) set label track and grid track with heights; C) do not add label track or grid track.

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")))
```

By default `chordDiagram` provides poor support for styles of sector labels and axis, but with `preAllocateTracks` it is rather easy to customize them. Such customization will be introduced in next section.

3.2 Customize sector labels

In `chordDiagram`, there is no argument to control the style of sector names. But this can be done by first pre-allocating an empty track and customizing the labels in it later. In the following example, one track is firstly allocated and a Chord Diagram is added without label track. Later, the first track is updated with setting `facing` of labels (figure 9 A).

```
chordDiagram(mat, annotationTrack = "grid", preAllocateTracks = list(track.height = 0.3))
# we go back to the first track and customize sector labels
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")
    circos.text(mean(xlim), ylim[1], sector.name, facing = "clockwise",
        niceFacing = TRUE, adj = c(0, 0.5))
}, bg.border = NA)
```

In the following example, the labels are put inside the grids (9 B). Please note `get.cell.meta.data` and `circos.text` are used outside of `panel.fun`, so `track.index` and `sector.index` should be specified explicitly.

```
chordDiagram(mat, annotationTrack = "grid", annotationTrackHeight = 0.15)
for(si in get.all.sector.index()) {
    xlim = get.cell.meta.data("xlim", sector.index = si, track.index = 1)
    ylim = get.cell.meta.data("ylim", sector.index = si, track.index = 1)
    circos.text(mean(xlim), mean(ylim), si, sector.index = si, track.index = 1,
        facing = "bending", col = "white")
}
```

For the last example, we add the sector labels with different style. If the width of the sector is less than 20 degree, the labels are added in the radical direction.

```
chordDiagram(mat, annotationTrack = "grid", preAllocateTracks = list(track.height = 0.1))
circos.trackPlotRegion(track.index = 1, panel.fun = function(x, y) {
    xlim = get.cell.meta.data("xlim")
    xplot = get.cell.meta.data("xplot")
    ylim = get.cell.meta.data("ylim")
    sector.name = get.cell.meta.data("sector.index")

    if(abs(xplot[2] - xplot[1]) < 20) {
        circos.text(mean(xlim), ylim[1], sector.name, facing = "clockwise",
            niceFacing = TRUE, adj = c(0, 0.5))
    } else {
        circos.text(mean(xlim), ylim[1], sector.name, facing = "inside",
            niceFacing = TRUE, adj = c(0.5, 0))
    }
}, bg.border = NA)
```

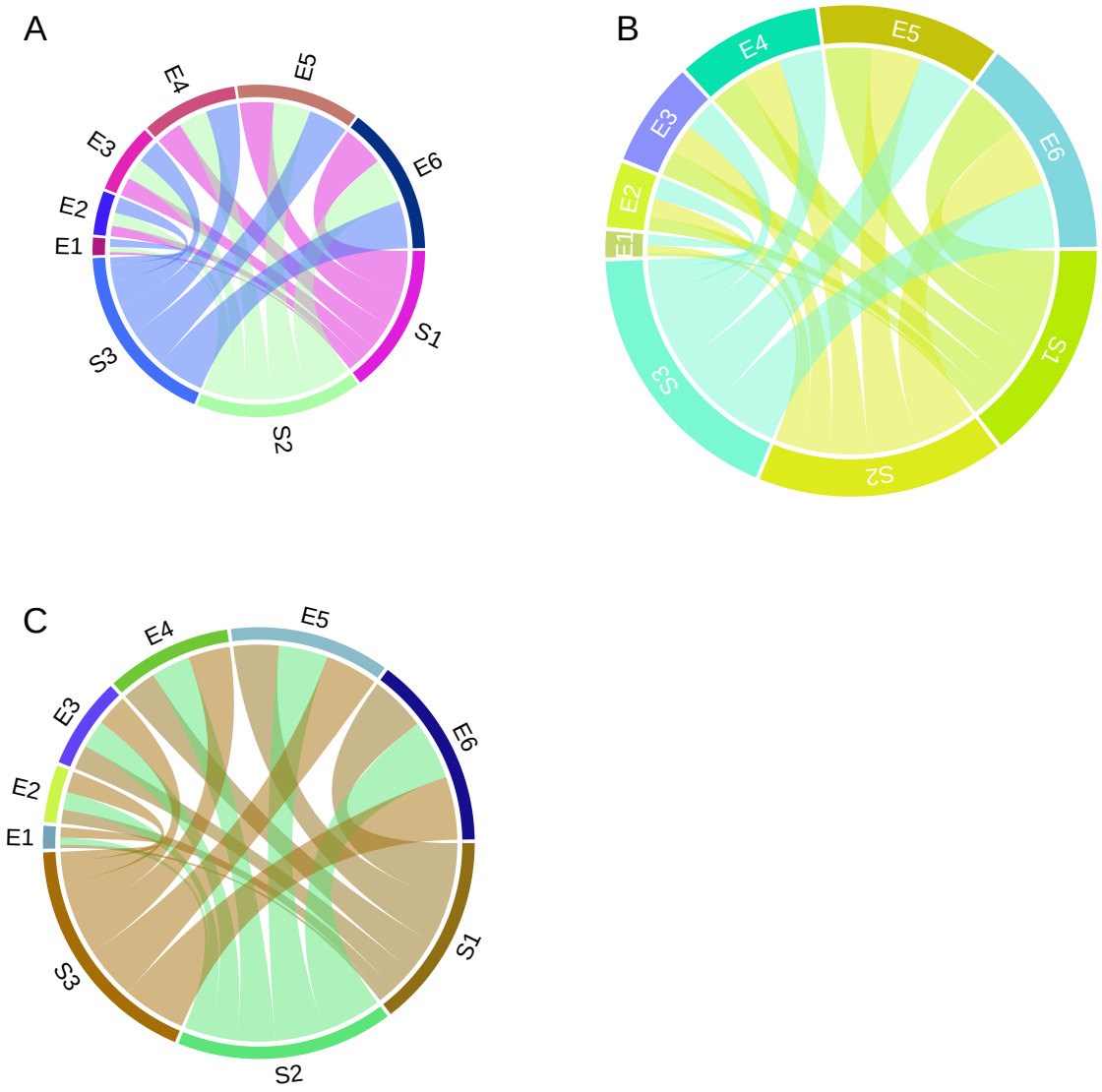

Figure 9: Customize sector labels. A) put sector labels in radical direction; B) sector labels are put inside grids; C) sector labels are put in different direction according the width of sectors.

One last thing, when you set direction of sector labels as radical, if the labels are too long and exceed your figure region, you can either decrease the size of the font or set `canvas.xlim` and `canvas.ylim` to wider intervals.

3.3 Customize sector axes

Axes are helpful to visualize the absolute values of links. By default `chordDiagram` does not provide argument to add axes. But it is easy with self-defined code.

Since there is already a grid track, we can add the axes directly to this track (figure 10 A). Here since the default height for the grid track is small, thus the default height for the axes ticks are also small, we manually set it to a larger value by `major.tick.percentage`. The value of `major.tick.percentage` is the ratio to the track height where axes are plotted.

```
chordDiagram(mat, grid.col = grid.col)
for(si in get.all.sector.index()) {
    # here the index for the grid track is 2
    circos.axis(h = "top", labels.cex = 0.3, major.tick.percentage = 0.2,
    sector.index = si, track.index = 2)
}
```

Another type of axes which show relative percent is also helpful for visualizing Chord Diagram. Here we pre-allocate an empty track by `preAllocateTracks` and come back to this track to add axes later. In following example, a major tick is put every 25% in each sector. And the axes are only added if the sector width is larger than 20 degree (figure 10 B).

```
# similar as the previous example, but we only plot the grid track
chordDiagram(mat, annotationTrack = "grid", preAllocateTracks = list(track.height = 0.1))
for(si in get.all.sector.index()) {
    circos.axis(h = "top", labels.cex = 0.3, major.tick.percentage = 0.2,
        sector.index = si, track.index = 2)
}

# the second axis as well as the sector labels are added in this track
circos.trackPlotRegion(track.index = 1, panel.fun = function(x, y) {
    xlim = get.cell.meta.data("xlim")
    xplot = get.cell.meta.data("xplot")
    ylim = get.cell.meta.data("ylim")
    sector.name = get.cell.meta.data("sector.index")

    if(abs(xplot[2] - xplot[1]) > 20) {
        circos.lines(xlim, c(mean(ylim), mean(ylim)), lty = 3) # dotted line
        for(p in seq(0.2, 1, by = 0.2)) {
            circos.text(p*(xlim[2] - xlim[1]) + xlim[1], mean(ylim) + 0.1,
                p, cex = 0.3, adj = c(0.5, 0), niceFacing = TRUE)
        }
    }
    circos.text(mean(xlim), 1, sector.name, niceFacing = TRUE, adj = c(0.5, 0))
}, bg.border = NA)
circos.clear()
```

3.4 Compare two Chord Diagrams

Normally, in Chord Diagram, 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 in the two plots should be represented in a same scale. This problem can be solved by adding more blank gaps to the Chord Diagram which has smaller values.

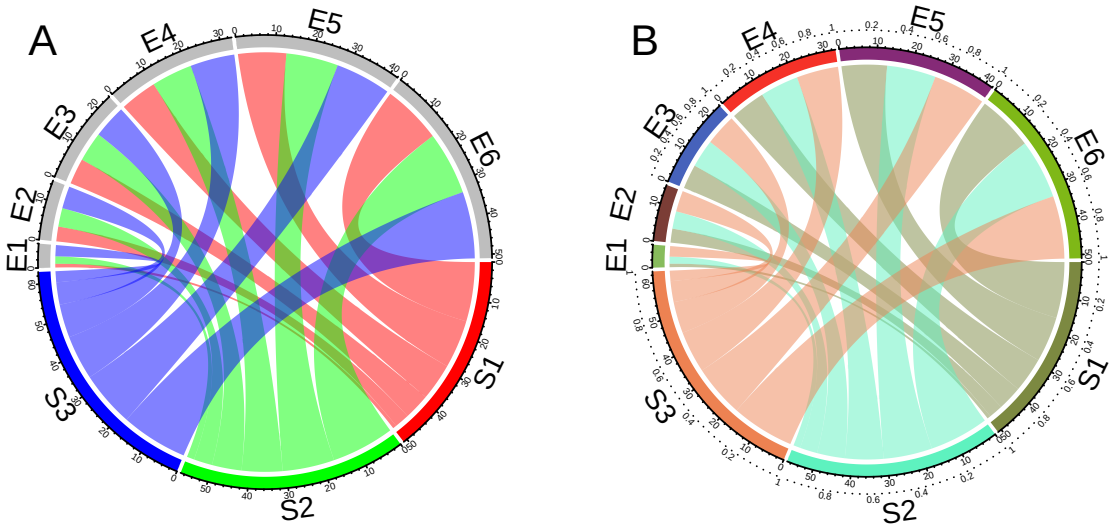

Figure 10: Customize sector axes. A) add axes to the grid track; B) add another percentage axes

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.clear()
circos.par(gap.degree = gap.degree, start.degree = -10/2)
chordDiagram(mat1, directional = TRUE, grid.col = rep(1:5, 2))
for(si in get.all.sector.index()) {
    circos.axis(labels.cex = 0.3, major.tick.percentage = 0.2,
        sector.index = si, track.index = 2)
}
circos.clear()
```

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 exactly the same which makes the comparison impossible. What we want to compare between two diagrams is the absolute values. For example, if the matrix contains the amount of transitions from one state to another state, then the interest is to see which diagram has more transitions.

First we calculate the percentage of `mat2` in `mat1`. And then we calculate the degree which corresponds to the difference. In the following code, `360 - sum(gap.degree)` is the total degree for values in `mat1` (excluding the gaps) and `blank.degree` corresponds the difference between `mat1` and `mat2`.

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

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

```
big.gap = (blank.degree - sum(rep(2, 8)))/2
gap.degree = c(rep(2, 4), big.gap, rep(2, 4), big.gap)
circos.par(gap.degree = gap.degree, start.degree = -big.gap/2)
```

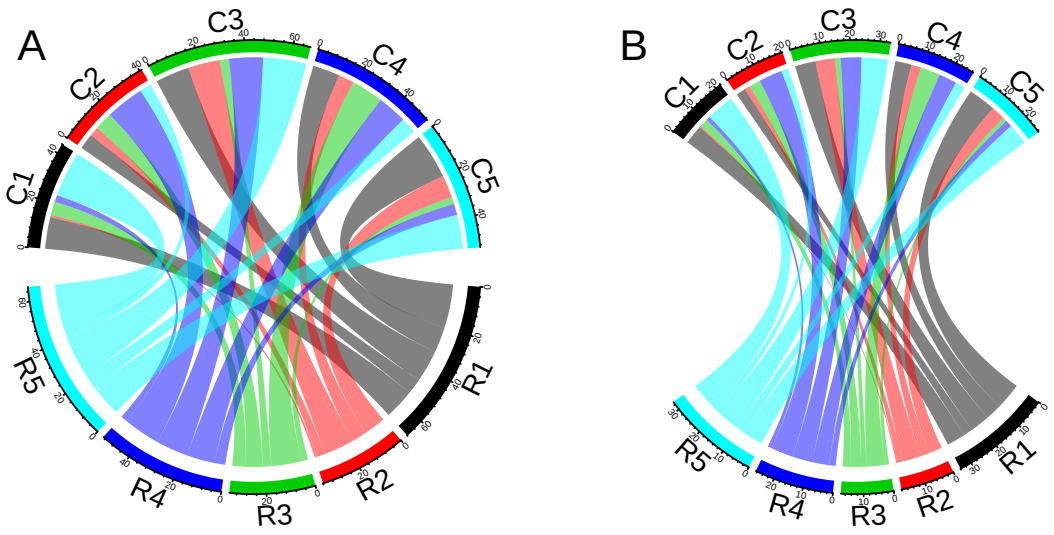

Figure 11: Compare two Chord Diagrams and make them in same scale. bottom matrix has half the values as in the upper matrix.

```
chordDiagram(mat2, directional = TRUE, grid.col = rep(1:5, 2), transparency = 0.5)
for(si in get.all.sector.index()) {
    circos.axis(labels.cex = 0.3, major.tick.percentage = 0.2,
        sector.index = si, track.index = 2)
}
circos.clear()
```

Now the scale of the two Chord Diagrams (figure 11) are the same if you look at the scale of axes in the two diagrams.

4 Misc

If a sector in Chord Diagram is too small, it will be removed from the original matrix. In the following matrix, the second row and third column contain tiny numbers.

```
mat = matrix(rnorm(36), 6, 6)
rownames(mat) = paste0("R", 1:6)
colnames(mat) = paste0("C", 1:6)
mat[2, ] = 1e-10
mat[, 3] = 1e-10
```

In the Chord Diagram, categories corresponding to the second row and the third column will be removed (figure 12 A).

```
chordDiagram(mat)
```

If you set `row.col`, `column.col` or `col` to specify the colors of the links, colors corresponding to the second row and the third column will also be removed (figure 12 B).

```
chordDiagram(mat, row.col = rep(c("red", "blue"), 3))
```

`grid.col` is reduced if it is set as a vector which has the same length as categories which are from the unreduced matrix (figure 12 C).

```
chordDiagram(mat, grid.col = rep(c("red", "blue"), 6))
circos.clear()
```

`circos.par("gap.degree")` will be reduced as well (figure 12 D).

```
circos.par("gap.degree" = rep(c(2, 10), 6))
chordDiagram(mat)
circos.clear()
```

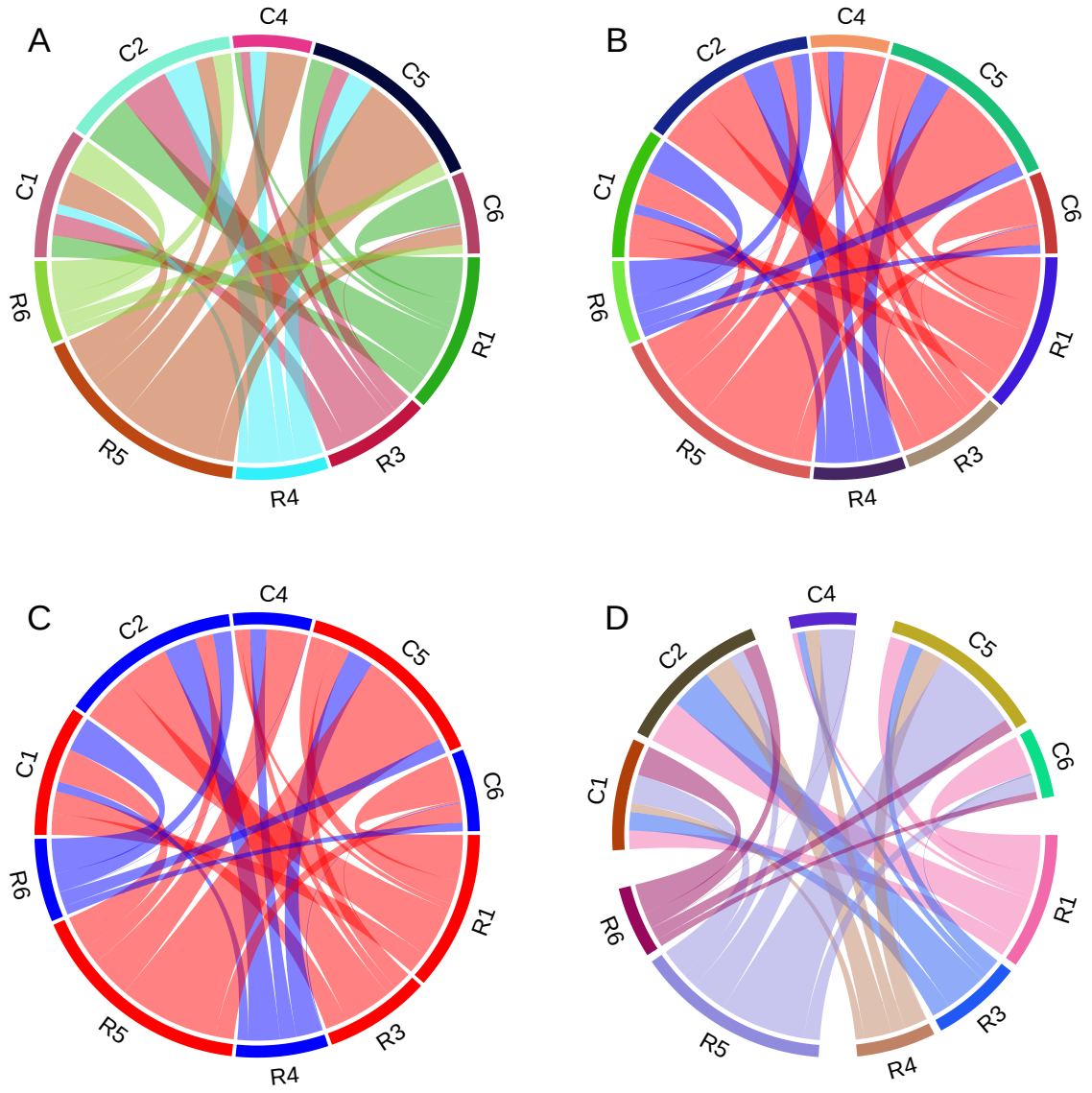

Figure 12: Reduced Chord Diagram with removing tiny sectors. A) notice how sector labels are reduced; B) notice how link colors are reduced; C) notice how grid colors are reduced; D) notice how gap degrees are reduced.