How to draw Chord diagram

Zuguang Gu <z.gu@dkfz.de>

July 11, 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. 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 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], 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, col_sum))
```

Then initialize the circular layout of 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 = 5)
> 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))
+
+         # plot white border in the grids
+         if(sector.name %in% rn) {
+             for(i in seq_len(ncol(mat))) {
+                 circos.lines(rep(sum(mat[sector.name, seq_len(i)]), 2), c(0, 1),
+                     col = "white")
+             }
+         } else if(sector.name %in% cn) {
+             for(i in rev(seq_len(nrow(mat)))) {
+                 circos.lines(rep(sum(mat[seq_len(i), sector.name]), 2), c(0, 1),
+                    col = "white")
+             }
+         }
+ })
```

Finally the links are added to the plot.

```
> 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()
```

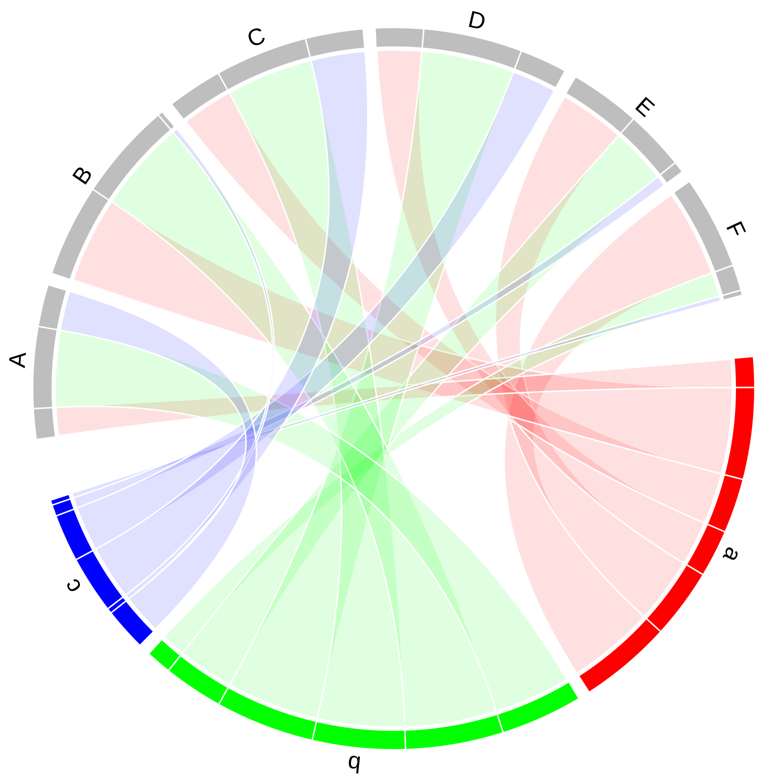


Figure 1: Table in circular layout

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 usage of `chordDiagram`. The most simple usage is just calling `chordDiagram` with `mat` (figure 2, top left).

```
> chordDiagram(mat)
> circos.clear()
```

The default Chord diagram consists with 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 grid which correspond to rows. The order of sectors is the order of `union(rownames(mat), colnames(mat))`.

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`).

```
> 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 plot 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 difference between the unequal root can be set through `directionGridHeight`.

```
> chordDiagram(mat, directional = TRUE)
> chordDiagram(mat, directional = TRUE, 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` must be a named vector of which names correspond to sector names. As explained before, the default link colors are same as colors for grid which correspond to rows (figure 3, top left).

```
> grid.col = NULL
> 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 complete 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).

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

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.

2.3 Advanced usage

Although `chordDiagram` provides default style which is enough for most visualization tasks. But still you can have more fine-tune of 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`.

```
> chordDiagram(mat, grid.col = grid.col, annotationTrack = "grid")
> 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
```

Several blank tracks can be allocated before Chord diagram is plotted. Then self-defined graphics can be added to these blank tracks later. 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("default.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 on 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, top right).

```
> 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")
+     if(sector.name %in% rn) {
+         label = paste0(rep(sector.name, 5), collapse="")
+         circos.text(mean(xlim), ylim[1], label, facing = "bending", adj = c(0.5, 0))
+     }
+     if(sector.name %in% cn) {
+         label = paste0(rep(sector.name, 5), collapse="")
+         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, 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).

```
> 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, bottom).

```
> 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.

```
> 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)
```

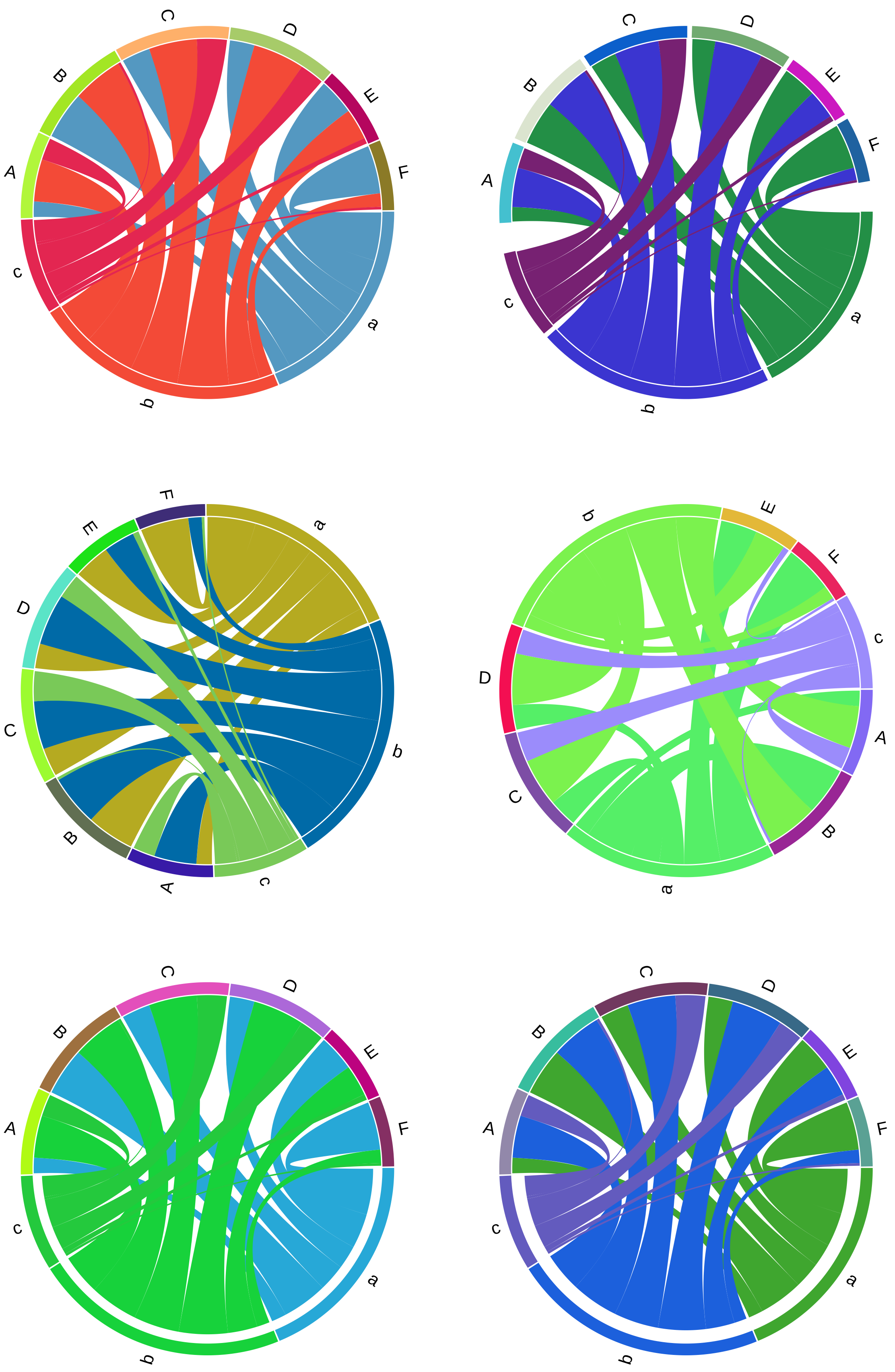


Figure 2: Basic settings for plotting Chord diagram

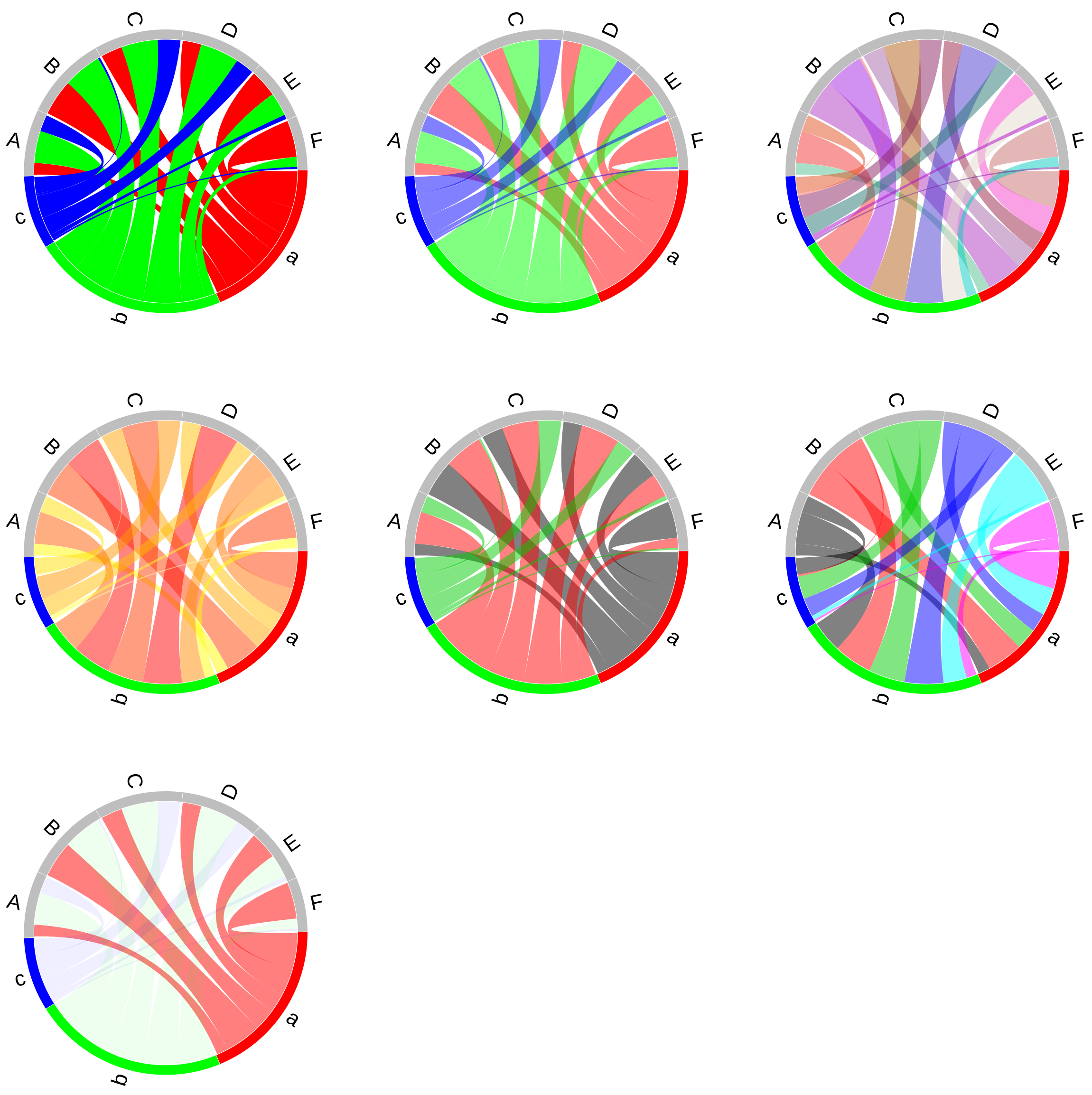


Figure 3: Color settings for plotting Chord diagram

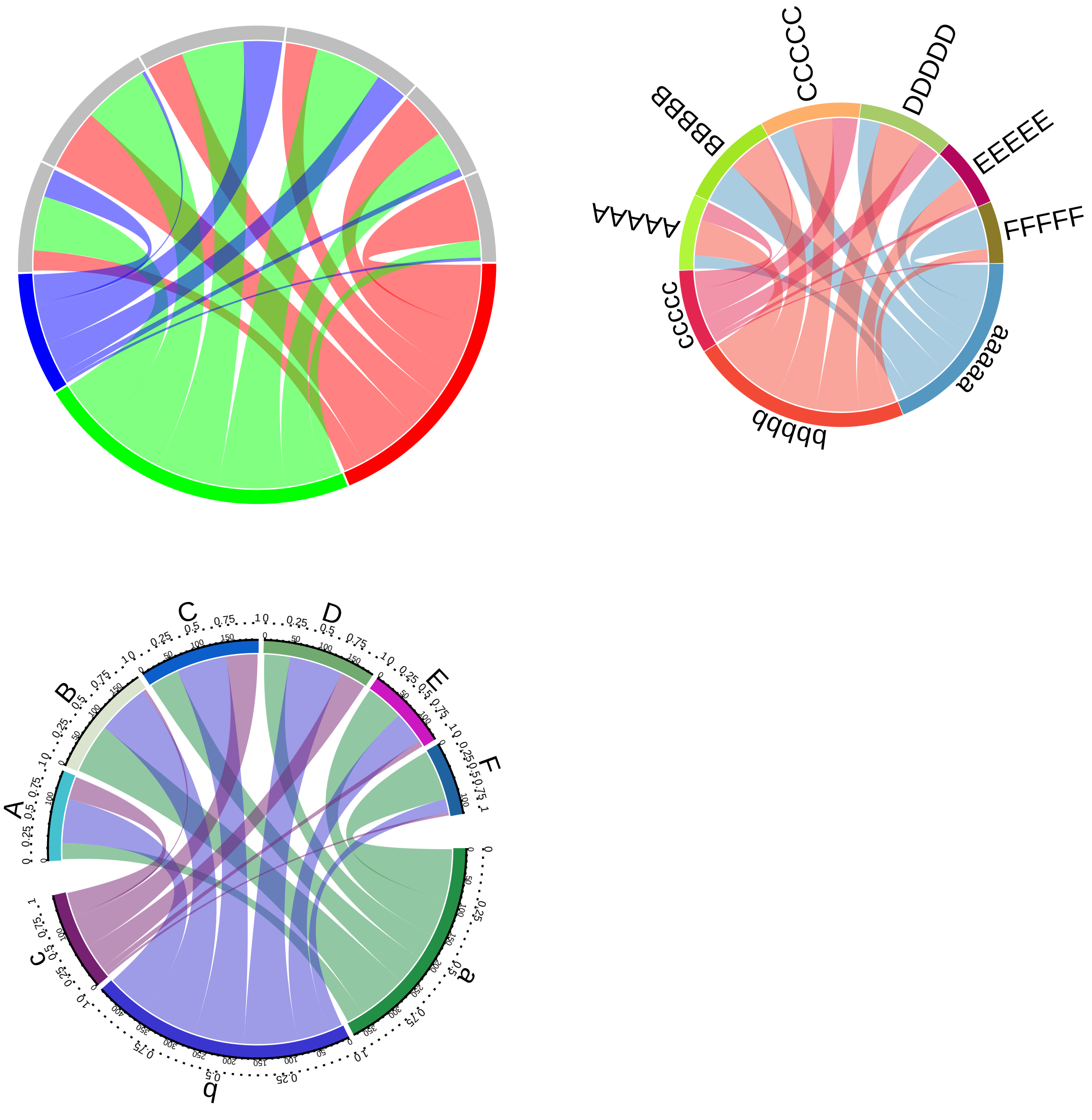
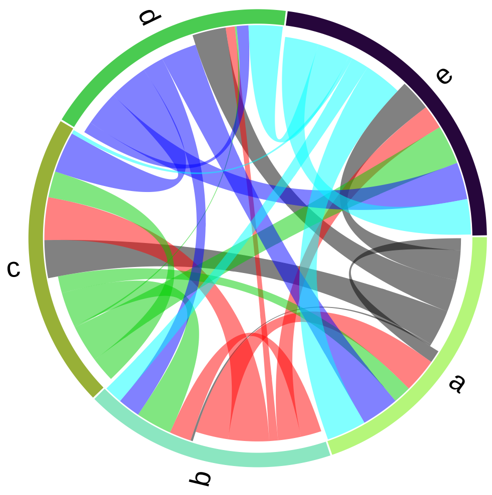
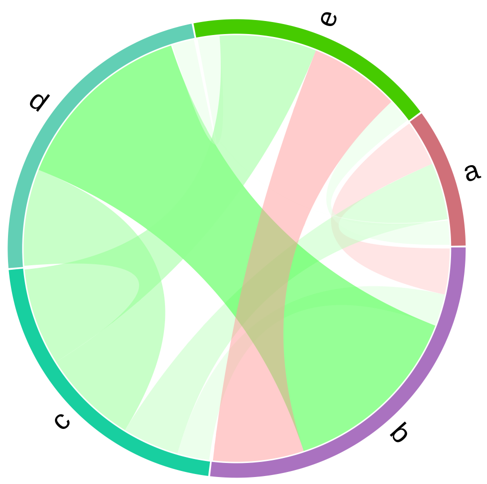


Figure 4: Advanced Chord diagram plotting





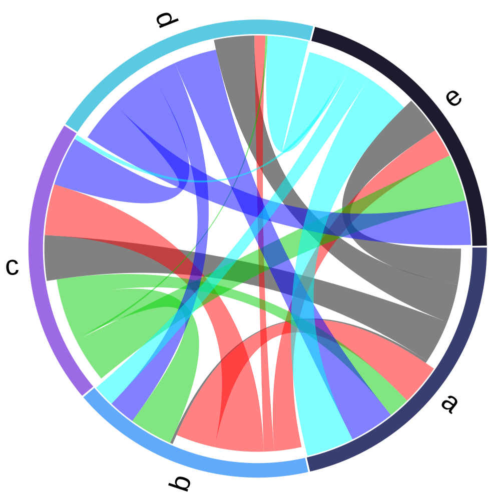


Figure 5: Chord diagram for other matrix