Investigating graphical representations of origin-destination spatial flow data

Supplementary material

2021-12-16

Contents

This document provides the **R** codes used to reproduce the results included in the paper *Investigating graphical representations of origin-destination spatial flow data.*

Several packages and plenty of lines of code are required for plotting the different figures we present in this paper. To help potential users to represent these figures, we create a unique function called `plot_flows()`, available on Github, which simplifies considerably the syntax code. However, the function still depends on the following R packages: **arcdiagram**, **circlize**, **classInt**, **colorspace**, **ggalluvial**, **gplots**, **igraph**, **sf**, **tidyverse**, **viridis**.

Interested users can install the required packages from CRAN or Github as follows:

```
install.packages(c(
  "circlize", # circular flows charts
  "classInt", # discretization of numeric variable
  "colorspace", # color palettes
  "devtools", # necessary to install Github packages
  "ggalluvial", # Sankey diagram
  "ggridges",  # ridges plot
  "gplots", # heat maps,
  "igraph", # useful to construct adjencency matrix
  "plot.matrix", # heat maps
  "sf", # spatial norm
  "tidyverse", # tidyverse universe
  "viridis" # color palette for heat map
  ))
```

```
devtools::install_github('gastonstat/arcdiagram')   # Arc diagram plot
devtools::install_github("LukeCe/spflow")           # flows data
```

Users can now load our function from Github as follows:

```
source("codes/plot_flows.R")
```

The mandatory input arguments of the function `plot_flows()` are:

- `y` is the flow vector of size N;
- `index_o`, respectively `index_d`, is the vector of size N containing the identifiers of the origins, respectively destinations;
- `xy_sf` is a spatial object **sf** which contains the coordinates of the S sites;
- `type_plot` is a character giving the name of the methods among `"Sankey"`, `"circular"`, `"arc"`, `"flow_map"`, `"griffith"`, `"heatmap"`

1 Illustration used in our article

Importing the data (the data were originally presented in Laurent *et al.*, 2020):

```
load("data/data_long.RData")
```

For reproducing the different graphics we present with any source of flows data, user needs to define the following objects :

- `y` is the vector of flows of size N:

```
y <- data_long[, "y"]
N <- length(y)
```

- `index_o` resp. `index_d` is the vector of size N with the indices of the origin resp. destination:

```
index_o <- data_long[, "cont_o"]
index_d <- substr(data_long[, "cont_d"], 1, nchar(data_long[, "cont_d"]) - 1)
```

- Finally, we need to compute the geographical coordinates as POINT, for each site $s \in S$; it may be stored in a **sf** object (**centroid_sf** hereafter) so that it can be transformed easily in another CRS. The id of the observations may be stored in `S` variable and must coincide with the notation used for the flows. In our example, each geographical region is not defined as an administrative area; thus, we use the centroid of an arbitrary country which belongs to the considered zones:

```
library(sf)
centroid_sf <- st_centroid(world[
  sapply(c("Canada", "Mexico", "Brazil", "Bahamas",
        "Switzerland", "France", "Poland", "Kazakhstan",
        "Jordan", "Algeria", "Cameroon",
        "India", "Thailand", "China", "Australia"),
        function(x) grep(x, world$name_long)), ])
```

Then we define the abbreviate and full names of the geographical region:

```
levels_S <- c("N.America", "C.America", "S.America", "Caribbean", "Eu-oth",
                   "UE-bef-2004", "UE-aft-2004", "ex-URSS", "M.East", "N.Africa",
                   "Africa", "S.Asia", "SE.Asia", "E.Asia", "Pacific")
labels_S <- c("North America", "Central America",
              "South America", "Caribbean", "Other European countries",
              "European Union (before 2004 exp)",
              "European Union (after 2004 exp)",
```

```
             "ex URSS", "Middle East", "North Africa",
             "Sub-Saharan Africa", "South Asia", "South East Asia", "East Asia", "Pacific")
```

We associate the names to the variable `S`:

```
centroid_sf$S <- levels_S
```

The CRS must be defined; in our case it is WGS 84:

```
st_crs(centroid_sf) <- 4326
st_crs(world) <- 4326
```

- Find the most appropriated CRS: in our case, we will use two different representations: the Mollweide projection ("+proj=moll") on one hand which preserves area relationships and "epsg:3035" on other hand which use a projection more convenient for plotting the flows. User must choose this criteria with respect to the locations of the data (see for instance http://magrit.cnrs.fr/docs/projection_list_fr.html). We apply the transformation on the boundaries of the world countries (that does not correspond to our spatial unit but it covers the same territory and it is helpful to visualize them) and the centroid of the sites $s \in S$

```
poly_sf <- st_transform(world[-160, ], "+proj=moll")
xy_sf <- st_transform(centroid_sf, "+proj=moll")
```

The colors have been chosen with package **colorspace**, one unique color for each $s \in S$:

```
library("colorspace")
q4 <- qualitative_hcl(length(levels_S), palette = "Dark 3")
names(q4) <- levels_S
```

Version of the input parameters including the full names:

```
index_o_l <- factor(index_o, levels = levels_S, labels = labels_S)
index_d_l <- factor(index_d, levels = levels_S, labels = labels_S)
xy_sf_l <- xy_sf
xy_sf_l$S <- factor(xy_sf_l$S, levels = levels_S, labels = labels_S)
q4_l <- qualitative_hcl(length(labels_S), palette = "Dark 3")
names(q4_l) <- labels_S
```

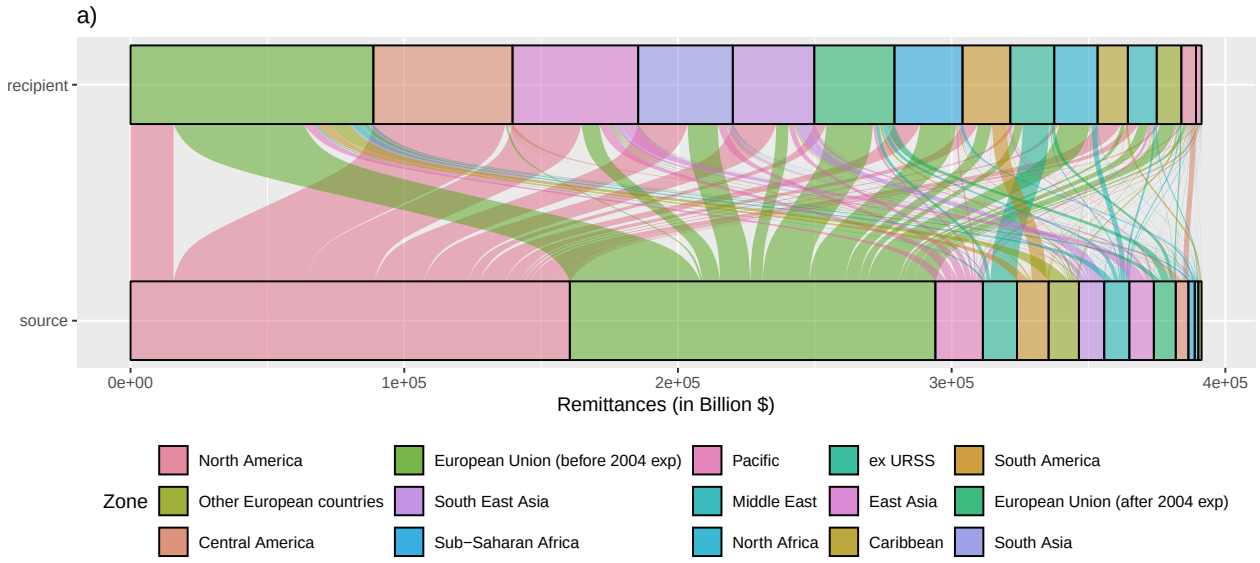
1.1 Sankey Diagram (Figure 2 in the article)

To plot a Sankey diagram, our function `plot_flows()` calls the **ggalluvial** package.

When building Sankey diagrams (circular diagrams, arc diagram and clustering heat map), the ordering of the entities is crucial in order to optimize the visualization properties of the graphs. In our open source software tool, we allow the different ordering methods presented in Figure 1 in the article. Users can define the ordering with the argument `ordering` among one of the following possibilities: `"descending"`, `"clustering"`, `"distance"`, `"longitude"`, `"latitude"`, `"none"`. We recommend to use by default the method based on the geographical distance between the sites. Optional arguments allow users to customize their graphs. Nevertheless, the default parameters enable users to obtain already nice figures.

To get Figure 2a), we use the `"descending"` option :

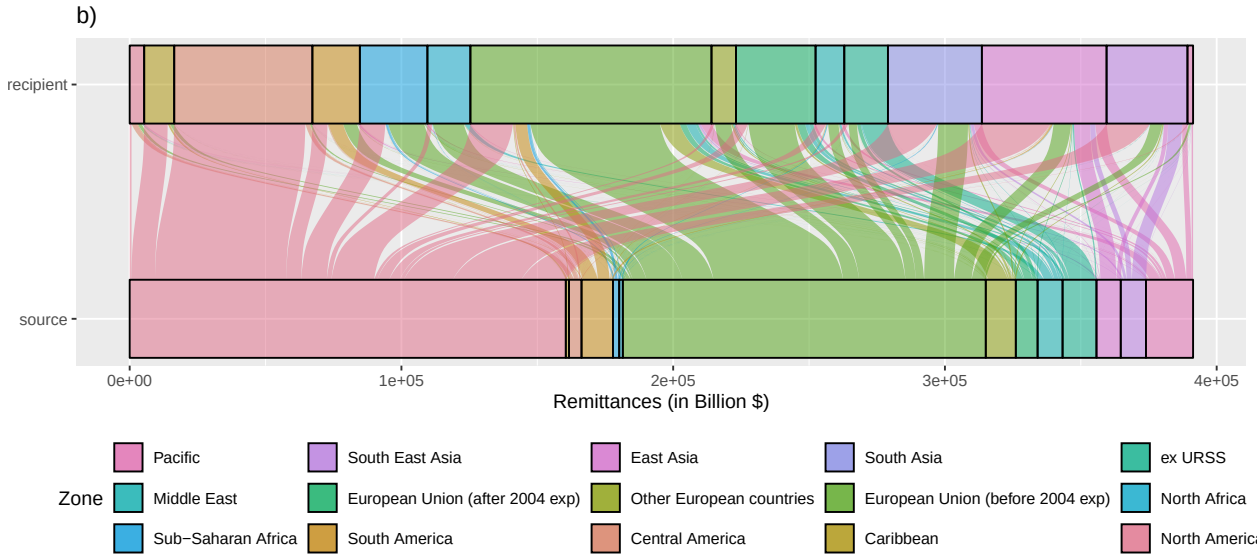
```
plot_flows(y, index_o_l, index_d_l, type_plot = "Sankey", xy_sf = xy_sf_l,
           ordering = "descending", color = q4_l, sankey.options = list(
             labels_od = c("source", "recipient"), nrows = 3,
             title = "a)", ylab = "Remittances (in Billion $)"))
```




```
# ggsave(filename = "figures/remitt_a.pdf", width = 10, height = 4.5)
```

To get Figure 2b), we use the `"distance"` option. Specifically, for the Sankey diagram, the optional parameters are included in the argument `sankey.options` as a list object. They allow to customize the legend and the labels of the diagram:

```
plot_flows(y, index_o_l, index_d_l, type_plot = "Sankey", xy_sf = xy_sf_l,
           ordering = "distance", color = q4_l, sankey.options = list(
             labels_od = c("source", "recipient"), nrows = 3,
             title = "b)", ylab = "Remittances (in Billion $)"))
```




```
# ggsave(filename = "./figures/remitt_b.pdf", width = 10, height = 4.5)
```

1.2 Circular flow charts (Figure 3)

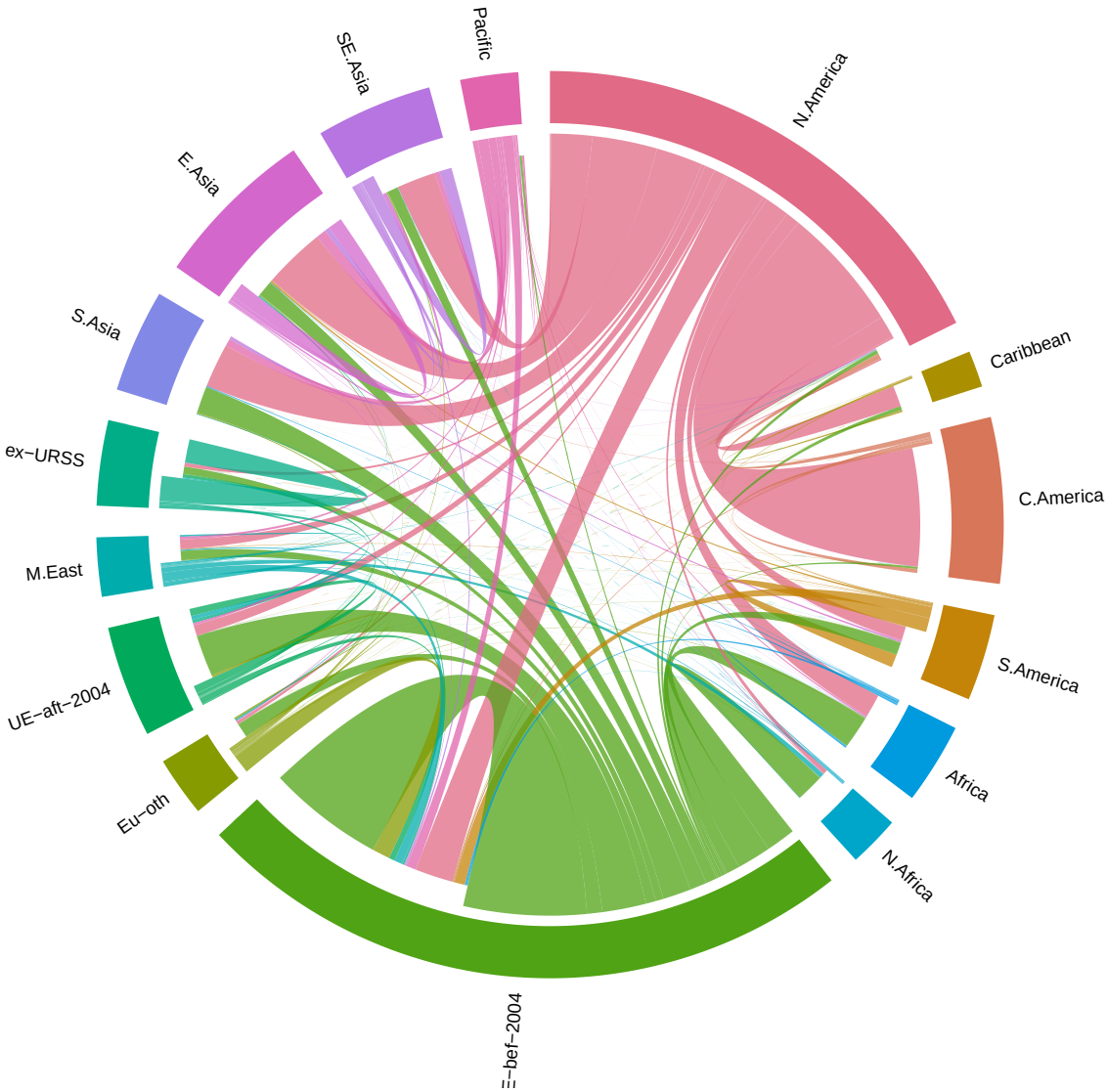
To print a circular flow chart, our function `plot_flows()` calls different functions included in the **circlize** package

data must be first stored into a matrix. Each cell (i, j) represents the flow value of origin i and destination j.

Row names and column names might be given.

To get figure 3, we can use different functions included in the **circlize** package

```
# pdf(file = "figures/circsize.pdf", width = 6, height = 6)
par(mar = c(0, 0, 0, 0), oma = c(0, 0, 0, 0), cex = 0.6, xpd = T)
plot_flows(y, index_o, index_d, color = q4, type_plot = "circular", xy_sf = xy_sf)
```

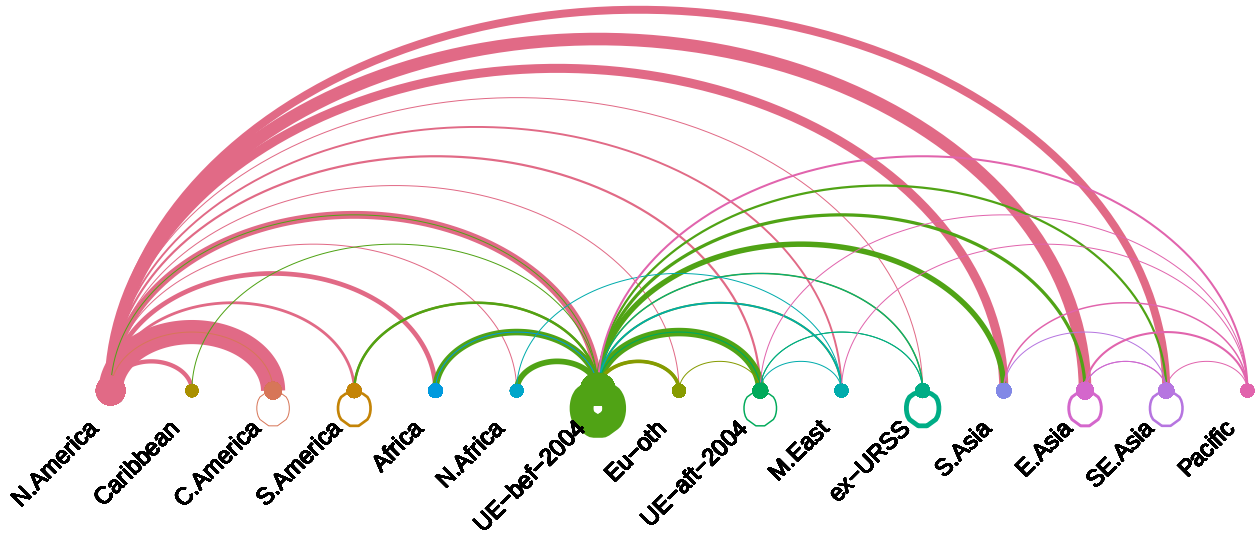



```
# dev.off()
```

1.3 Arc diagram (Figure 4)

To print an Arc diagram, our function `plot_flows()` uses the **igraph** package norm. In addition, we modify the **R** function `arcdiagram()`. We have introduced two changes to `arcdiagram()`. First, we add the possibility to print the intra-regional flows. Second, instead of plotting all the flows (including those that have very small values and which may create noise in the readability of the graph), we allow users to plot only the largest flows. To do so, we introduce the additional optional argument, `alpha.q`, that gives the value of the highest quantile to represent (it is equal to 75% by default). Two additional parameters allow to modify the maximum size of the nodes and the width of the arc:

```
# pdf("figures/arcplot.pdf", width = 12, height = 5)
plot_flows(y, index_o, index_d, type_plot = "arc",
           xy_sf = xy_sf, color = q4,
   arc.options = list(maxcex = 1, maxlwd = 1, alpha.q = 0.75))
```




```
# dev.off()
```

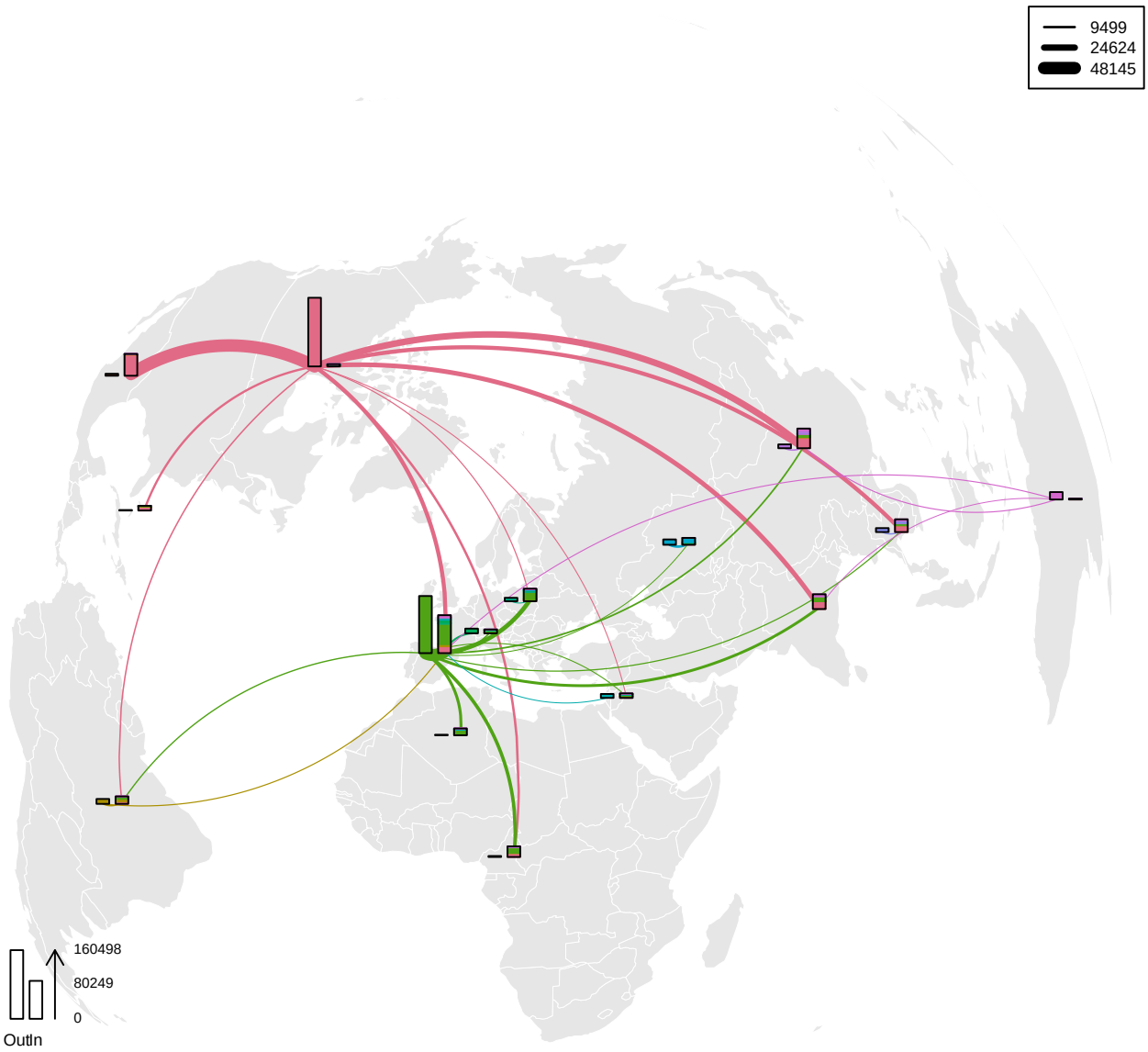
1.4 Map flows (figure 6)

To plot a map flow, there are several optional parameters to modify. Among them: `alpha.q` gives the values of the higher quantile to represent; `col_geometry` gives the color of the polygons of the entities; `border_geometry` gives the color of the border.

If necessary, one can first change the CRS:

```
poly_sf <- st_transform(world[-160, ], 3035)
xy_sf <- st_transform(centroid_sf, 3035)
```

```
# pdf("figures/map.pdf", width = 8, height = 6.5)
par(mar = c(0, 0, 0, 0))
plot_flows(y, index_o, index_d, type_plot = "flow_map",
           xy_sf = xy_sf, col = q4, contours_map = poly_sf,
           flow_map.options = list(alpha.q = 0.85,
                                   col_geometry = rgb(0.9, 0.9, 0.9),
                                   border_geometry = "white"))
```




```
# dev.off()
```

1.5 Map flow à la Griffith (figure 7)

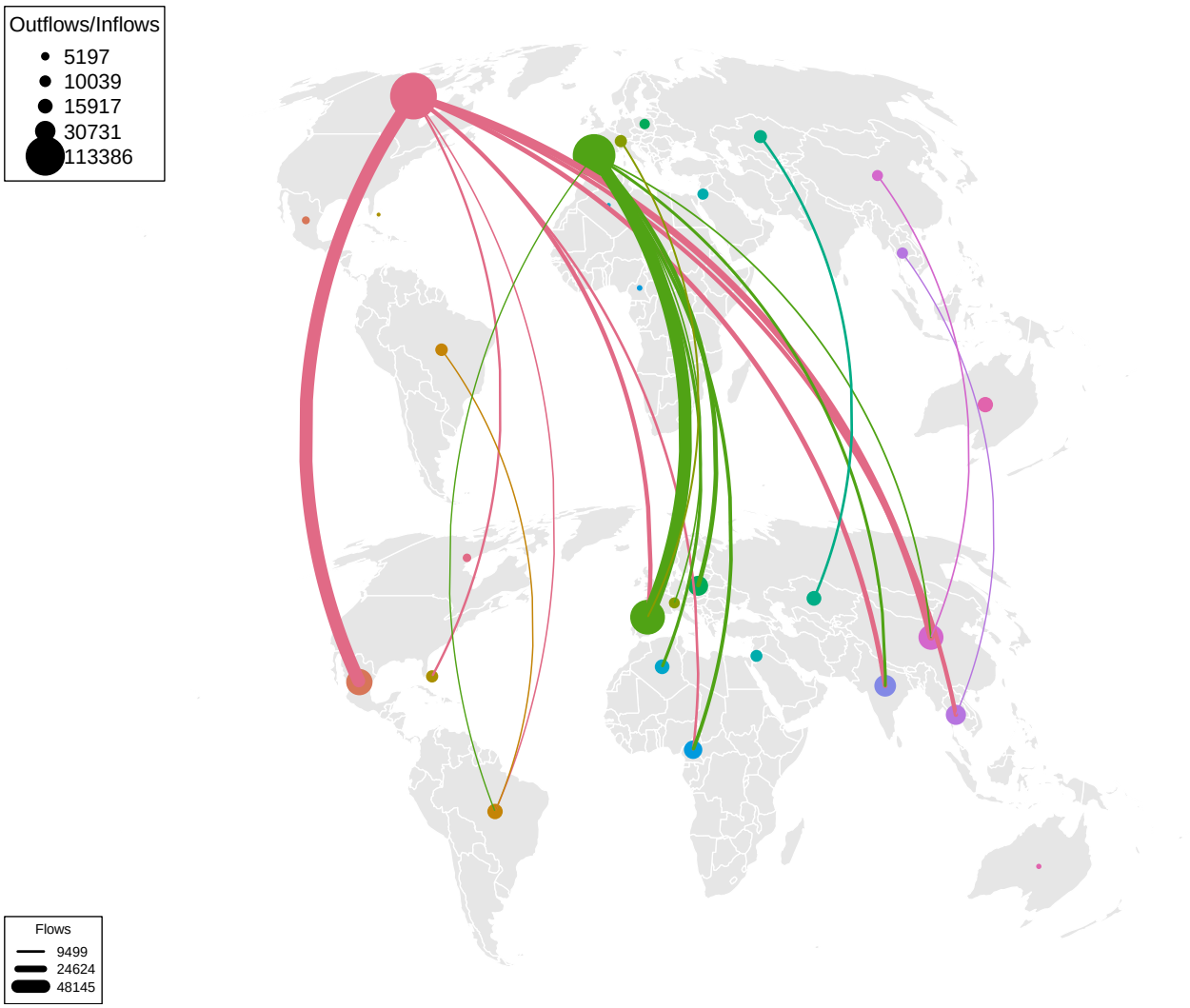
To represent a map flow à la Griffith, one can change the position of the destination map with respect to a scaling factor given by the parameter `scale_xy`. The remaining parameters are identical to the map flow function.

If necessary, one can first change the CRS:

```
poly_sf <- st_transform(world[-160, ], "+proj=moll")
xy_sf <- st_transform(centroid_sf, "+proj=moll")
```

```
# pdf("figures/map_2.pdf", width = 10, height = 8)
par(mar = c(0, 0, 0, 0))
plot_flows(y, index_o, index_d, type_plot = "griffith", xy_sf = xy_sf,
           contours_map = poly_sf, col = q4,
           griffith.options = list(
             scale_xy = c(1/20, - 1),
```

```
            alpha.q = 0.9,
            col_geometry_o = rgb(0.9, 0.9, 0.9),
            col_geometry_d = rgb(0.9, 0.9, 0.9)))
```



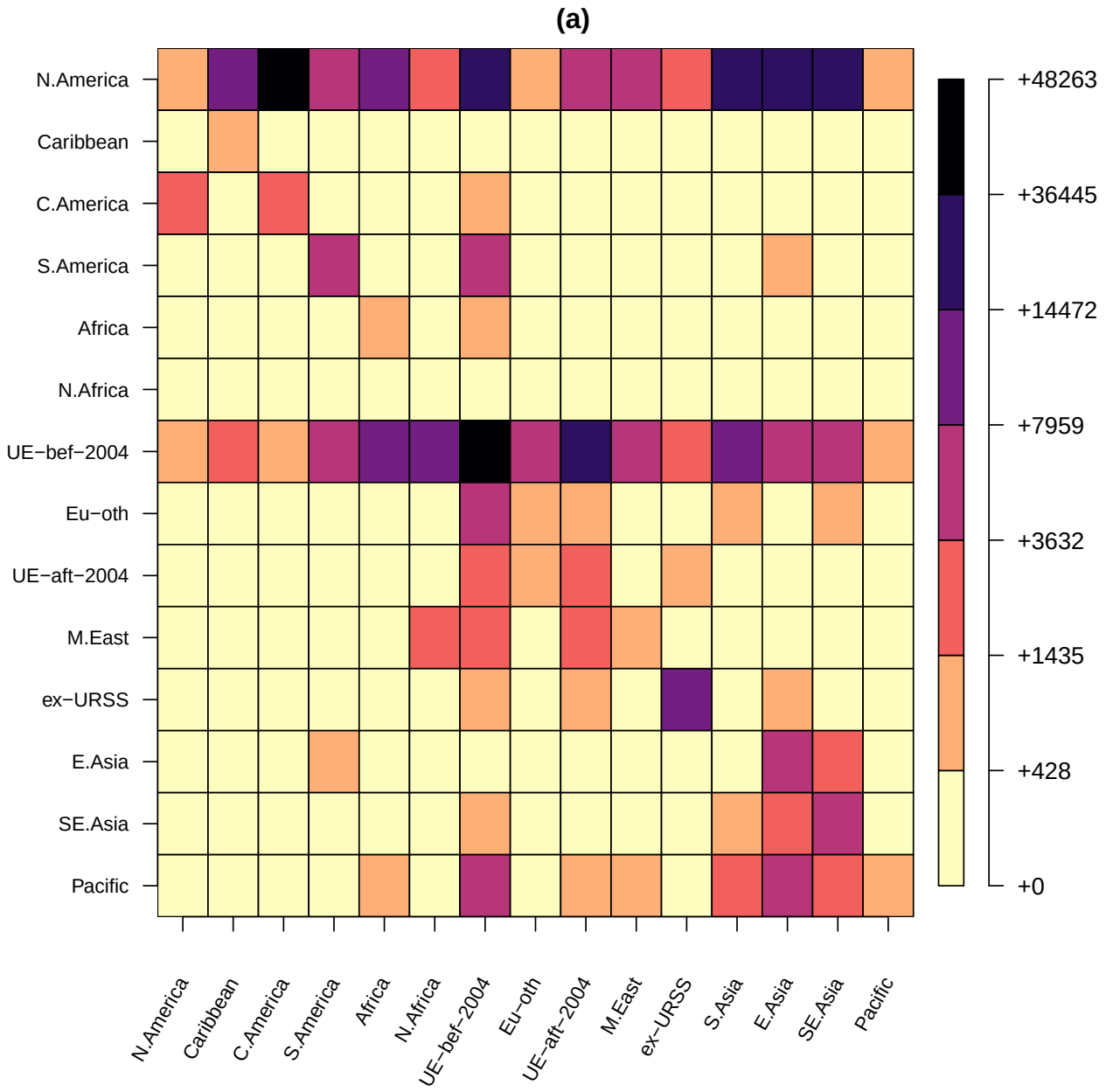

```
# dev.off()
```

1.6 Clustering heat map (figure 8)

Finally, to represent the clustering heat map, we need to define the number of classes and the method that allows to discretize the Y variable

- To get Figure 9 a)

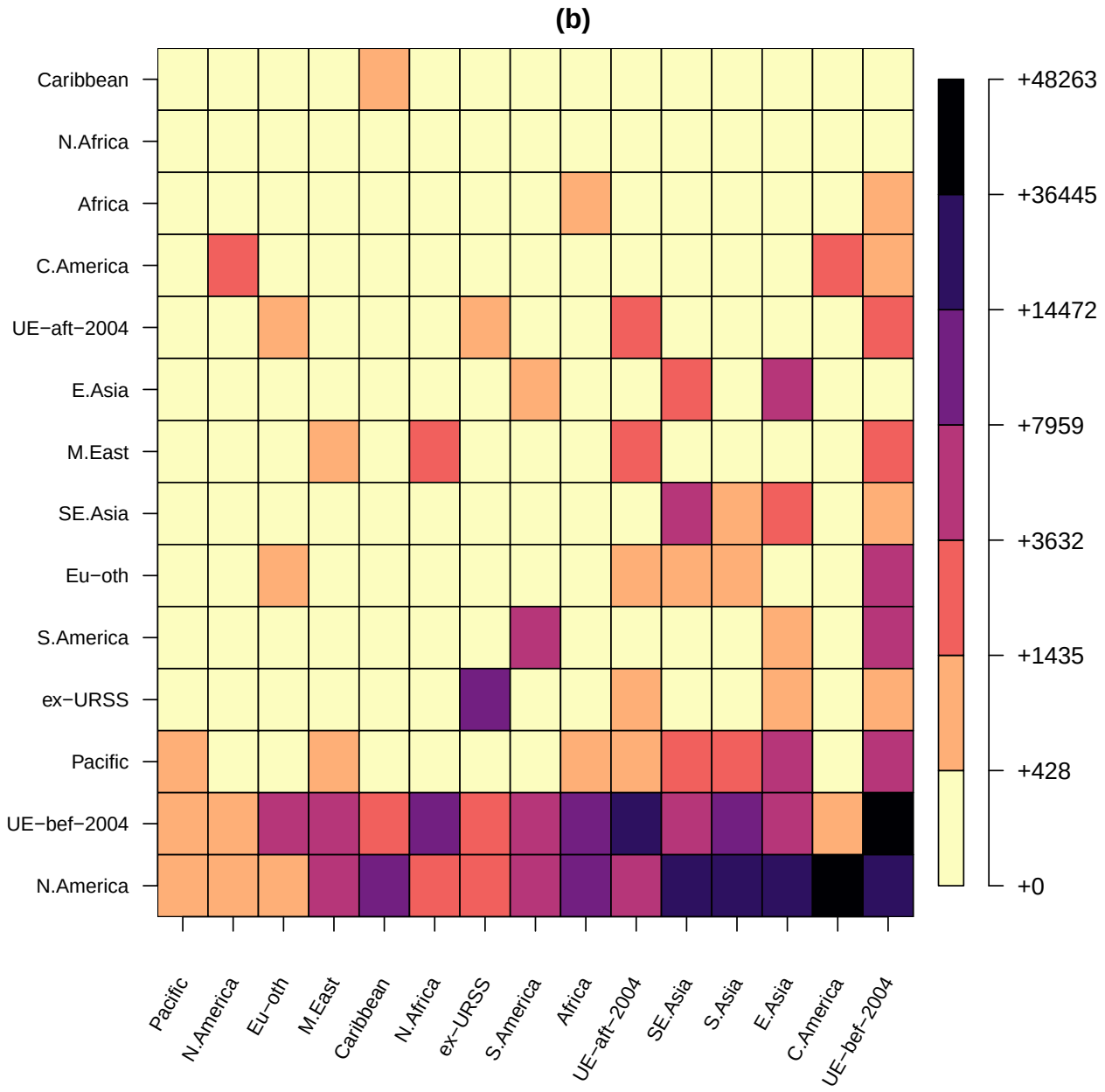
```
# pdf(file = "figures/matrix_flows.pdf", width = 8, height = 8)
par(mar=c(7.5, 6.8, 2.1, 4.1), las = 1, lheight = .6) # adapt margins
plot_flows(y, index_o, index_d, ordering = "distance",
           type_plot = "heatmap", xy_sf = xy_sf,
           heatmap.options = list(style_class = "kmeans",
                                  n.class = 7,
                                  title = "(a)"))
```




```
# dev.off()
```

- To get Figure 9 b)

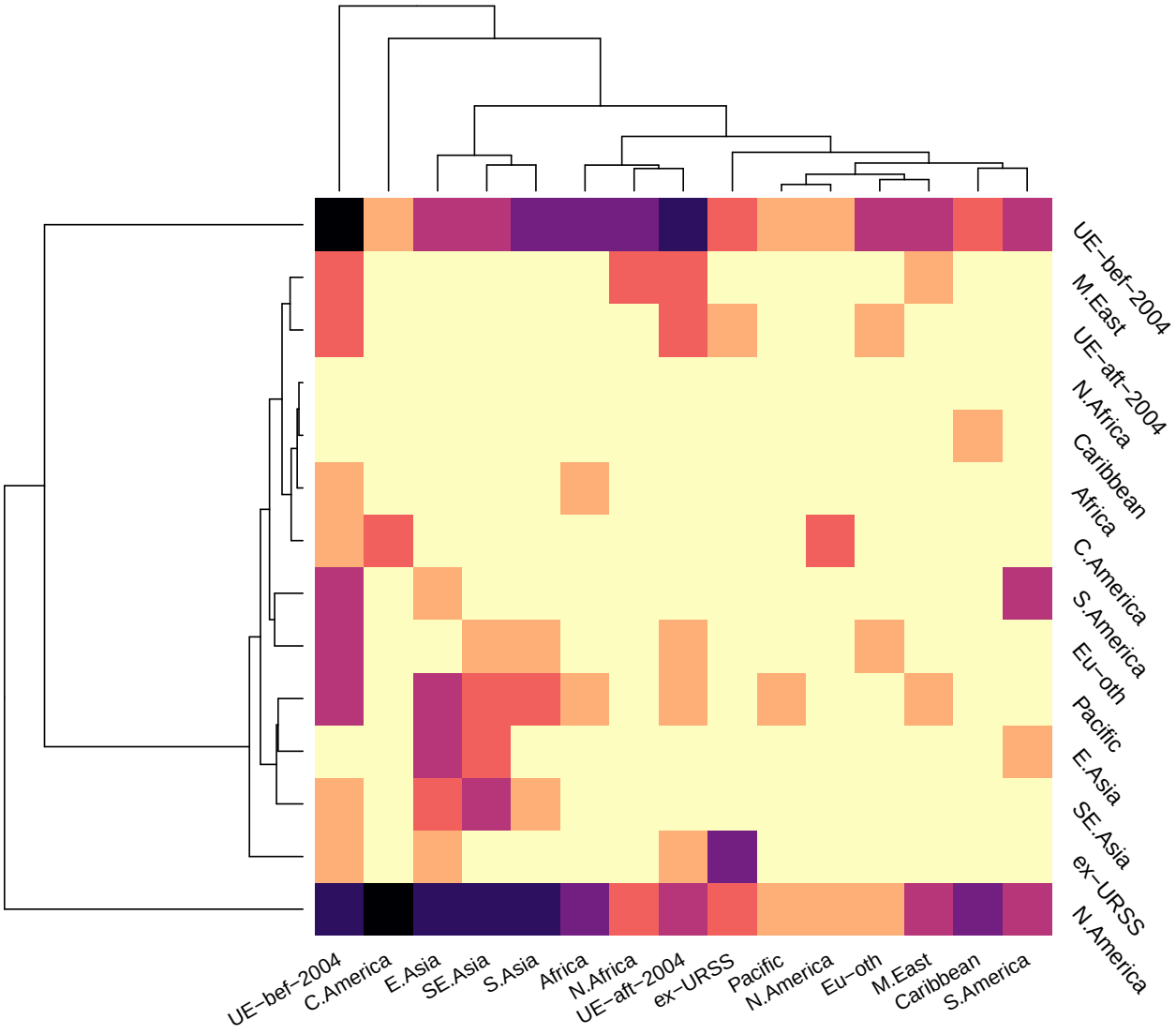
```
# pdf(file = "figures/matrix_flows_2.pdf", width = 8, height = 8)
par(mar=c(7.5, 6.8, 2.1, 4.1), las = 1, lheight = .6) # adapt margins
plot_flows(y, index_o, index_d, ordering = "descending",
           type_plot = "heatmap", xy_sf = xy_sf,
           heatmap.options = list(style_class = "kmeans",
                                  n.class = 7,
                                  title = "(b)"))
```




```
# dev.off()
```

- To get the heat map with the clustering methods presented in Figure 9 c), our function `plot_flows()` calls the package **gplots**

```
# pdf(file = "figures/matrix_flows_3.pdf", width = 8, height = 8)
par(mar=c(7.5, 6.8, 2.1, 4.1), las = 1, lheight = .6) # adapt margins
plot_flows(y, index_o, index_d, ordering = "clustering",
           type_plot = "heatmap", xy_sf = xy_sf,
           heatmap.options = list(style_class = "kmeans",
                                  n.class = 7,
                                  title = "(c)"))
```




```
# dev.off()
```

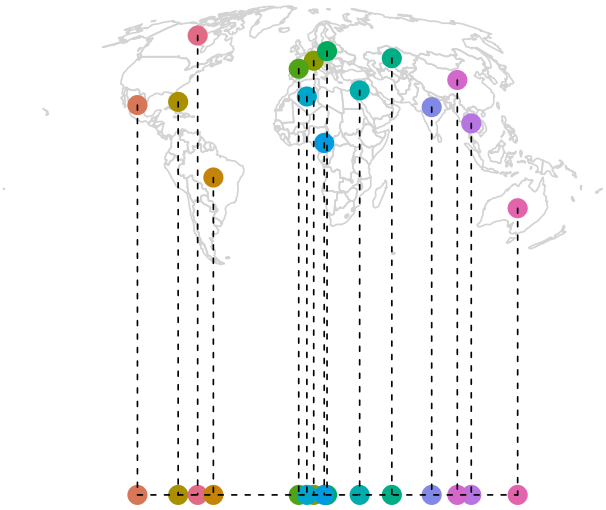
1.7 How to choose an appropriate ordering ?

Here are the codes used to print Figure 1 in the article:

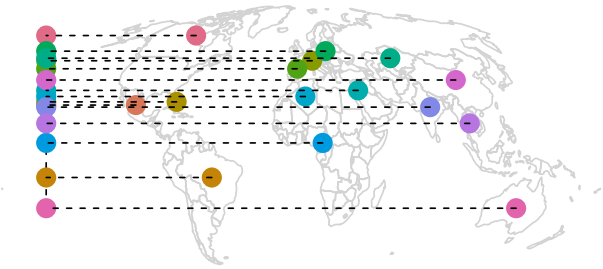
```
# pdf("figures/projection.pdf", width = 12, height = 8)
par(oma = c(0, 0, 0, 0), mar = c(0, 0, 0, 0), mfrow = c(2, 2))
# 1st plot
plot(st_geometry(poly_sf), border = "lightgrey")
plot(st_geometry(xy_sf), add = T, cex = 2, pch = 16, col = q4)
x1 <- st_coordinates(xy_sf)[, 1]
x2 <- rep(par()$yaxp[1], 15)
points(x1, x2, cex = 2, pch = 16, col = q4)
segments(st_coordinates(xy_sf)[, 1], st_coordinates(xy_sf)[, 2], x1, x2, lty = 2)
segments(min(x1), par()$yaxp[1], max(x1), par()$yaxp[1], lty = 2)
text(par()$xaxp[1], par()$yaxp[2], "a)", pos = 1, cex = 2)
```

```r
# 2nd plot
plot(st_geometry(poly_sf), border = "lightgrey")
plot(st_geometry(xy_sf), add = T, cex = 2, pch = 16, col = q4)
x1 <- (rep(par()$xaxp[1], 15))
x2 <- st_coordinates(xy_sf)[, 2]
segments(par()$xaxp[1], min(x2), par()$xaxp[1], max(x2), lty = 2)
points(x1, x2, cex = 2, pch = 16, col = q4)
segments(st_coordinates(xy_sf)[, 1], st_coordinates(xy_sf)[, 2],
         x1, x2, lty = 2)
text(par()$xaxp[1], par()$yaxp[2], "b)", pos = 1, cex = 2)

# 3rd plot
plot(st_geometry(poly_sf), border = "lightgrey")
plot(st_geometry(xy_sf), add = T, cex = 2, pch = 16, col = q4)
mat_distance <- st_distance(xy_sf)

my_index <- 1
for(k in 1:15) {
  my_id <- 2
  my_order <- order(mat_distance[my_index[k], ])
  my_choice <- my_order[my_id]
  while(my_choice %in% my_index) {
    my_id <- my_id + 1
    my_choice <- my_order[my_id]
}
  segments(st_coordinates(xy_sf)[my_index[k], 1], st_coordinates(xy_sf)[my_index[k], 2],
           st_coordinates(xy_sf)[my_choice, 1],
           st_coordinates(xy_sf)[my_choice, 2], lty = 2)
  my_index <- c(my_index, my_choice)
}

text(par()$xaxp[1], par()$yaxp[2], "c)", pos = 1, cex = 2)
# dev.off()
```

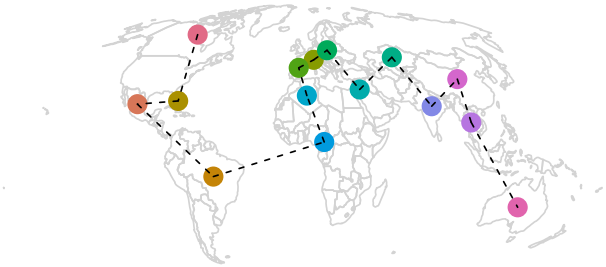
a)



b)



c)



2 Trainning with another data sets

2.1 Commuting flows

This dataset is included in the **R** package **spflow** (Dargel and Laurent, 2021). It consists of commuting flows in the 20 districts in the city of Paris.

Preparation of the data:

```
library(spflow)
library(sf)
data("paris10km_commuteflows")
data("paris10km_municipalities")
```

```
# vector of the flows :
# Select the main arrondissements
paris_intra_o <- substr(paris10km_commuteflows$ID_ORIG, 1 , 2) == "75"
paris_intra_d <- substr(paris10km_commuteflows$ID_DEST, 1 , 2) == "75"
paris_intra_S <- substr(paris10km_municipalities$ID_MUN, 1, 2) == "75"
# create the vector of flows
y <- paris10km_commuteflows[paris_intra_o & paris_intra_d, "COMMUTE_FLOW"]
# create the indices of origin and destination
index_o <- paris10km_commuteflows[paris_intra_o & paris_intra_d, "ID_ORIG"]
index_d <- paris10km_commuteflows[paris_intra_o & paris_intra_d, "ID_DEST"]
# create the spatial objects with the coordinates of the sites
xy_sf <- st_centroid(paris10km_municipalities[paris_intra_S, ])
```

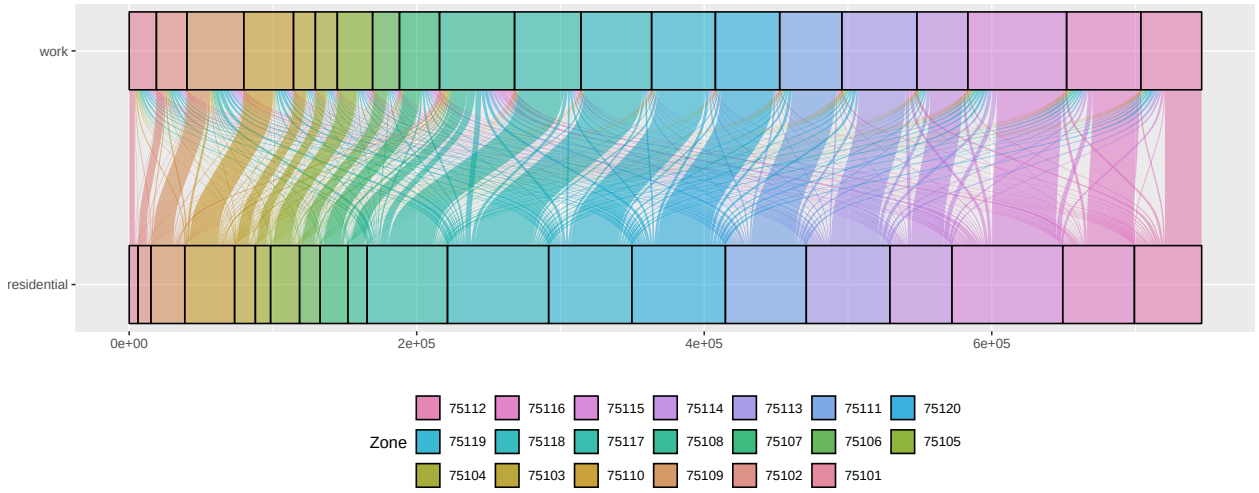
```
## Warning in st_centroid.sf(paris10km_municipalities[paris_intra_S, ]):
## st_centroid assumes attributes are constant over geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for longitude/
## latitude data
```

```
names(xy_sf)[1] <- "S"
# create the spatial objects with the polygons of the site
contours_map <- paris10km_municipalities[paris_intra_S, ]
```

- Sankey plot :

```
plot_flows(y, index_o, index_d, type_plot = "Sankey", xy_sf = xy_sf,
           sankey.options = list(labels_od = c("residential", "work")))
```




```
# ggsave(filename = "figures/paris/remitt.pdf", width = 12, height = 5)
```

- Circular plot :

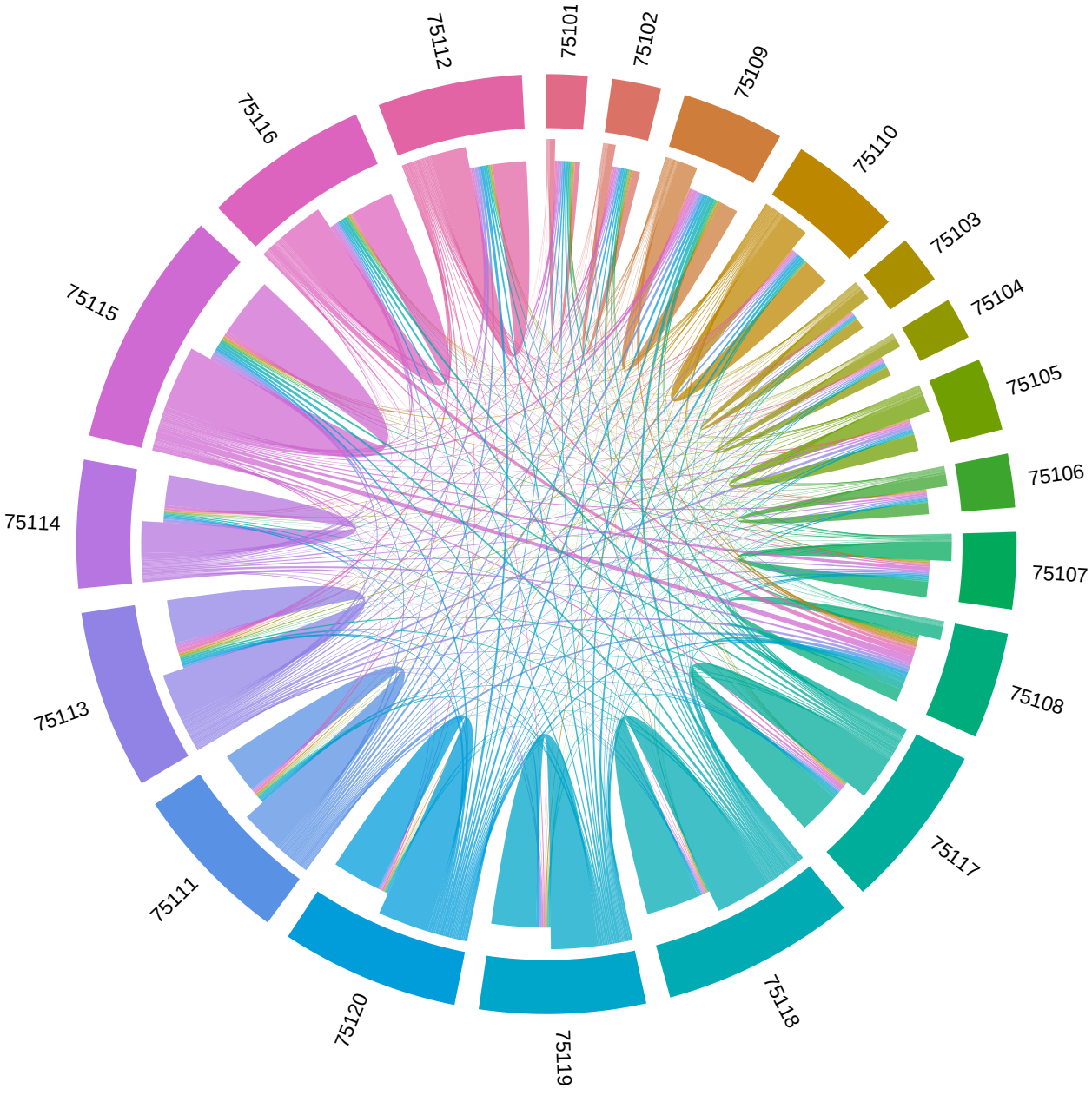
```
# pdf(file = "figures/paris/circular.pdf", width = 8, height = 8)
plot_flows(y, index_o, index_d, type_plot = "circular", xy_sf = xy_sf)
```

```
## Note: The second link end is drawn out of sector '75103'.
```

```
## Note: The second link end is drawn out of sector '75105'.
```

```
## Note: The second link end is drawn out of sector '75107'.
```

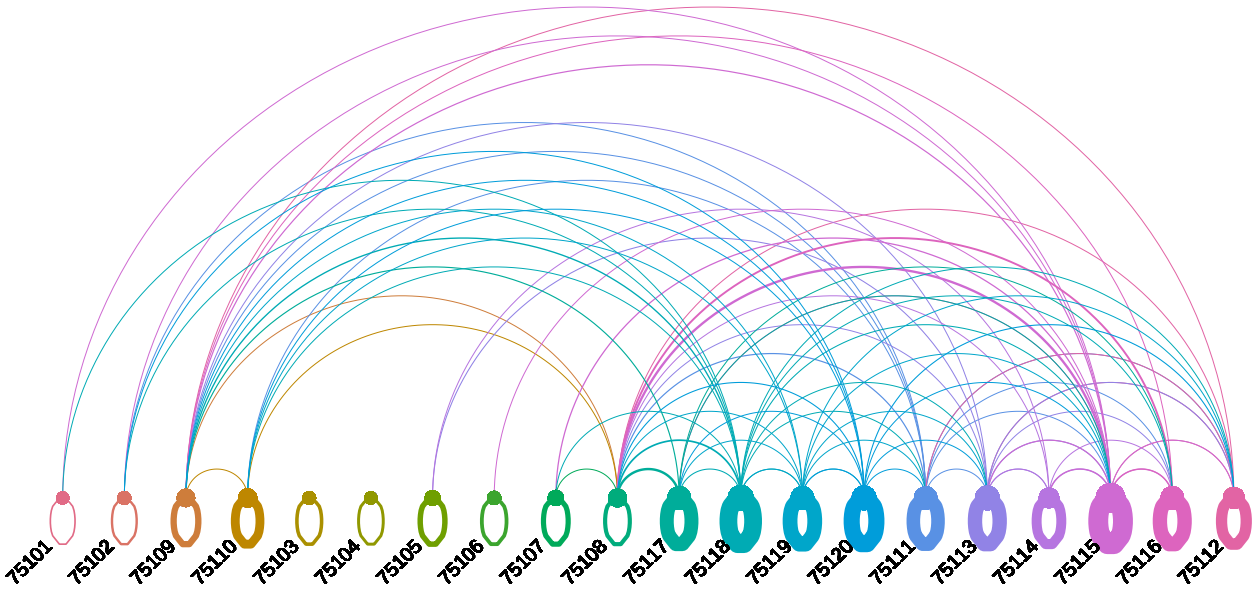
```
## Note: The second link end is drawn out of sector '75108'.
## Note: The second link end is drawn out of sector '75109'.
## Note: The second link end is drawn out of sector '75111'.
## Note: The second link end is drawn out of sector '75114'.
## Note: The second link end is drawn out of sector '75119'.
```




```
# dev.off()
```

- Arc diagram :

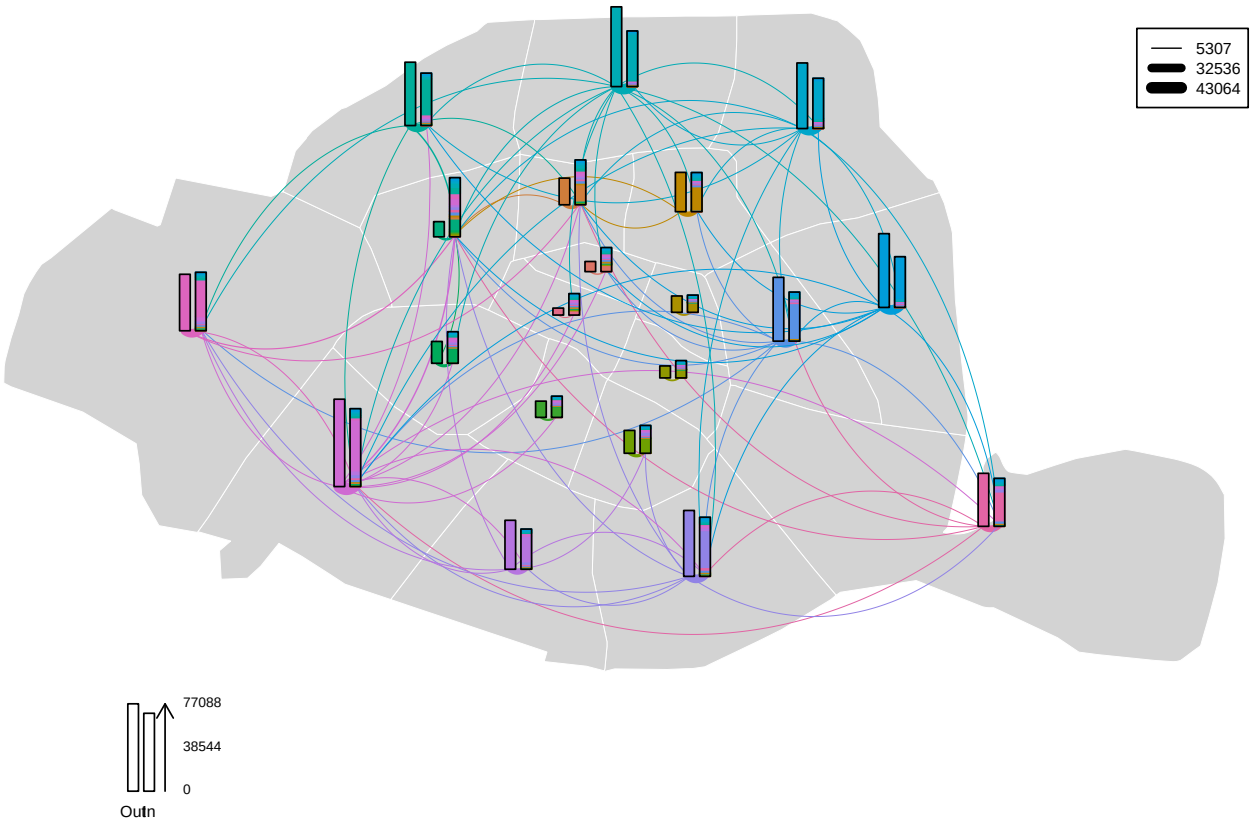
```
# pdf(file = "figures/paris/arc.pdf", width = 12, height = 6)
plot_flows(y, index_o, index_d, type_plot = "arc", xy_sf = xy_sf)
```




```
# dev.off()
```

- Flow map :

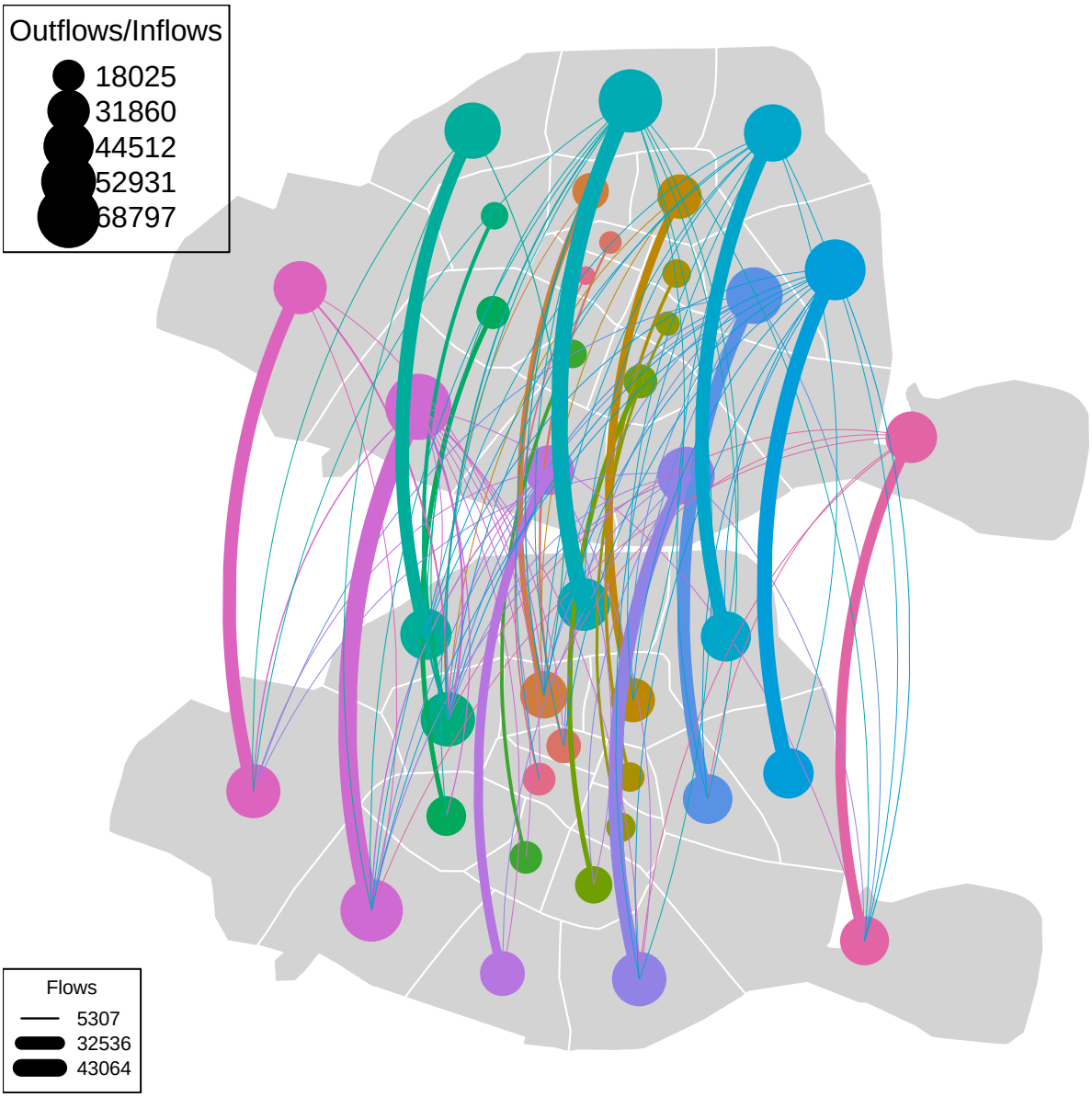
```
# pdf(file = "figures/paris/flow_map.pdf", width = 12, height = 6)
par(oma = c(0, 0, 0, 0), mar = c(0, 0, 0, 0))
plot_flows(y, index_o, index_d, type_plot = "flow_map", xy_sf = xy_sf,
           contours_map = contours_map
)
```



```
# dev.off()
```

- Griffith :

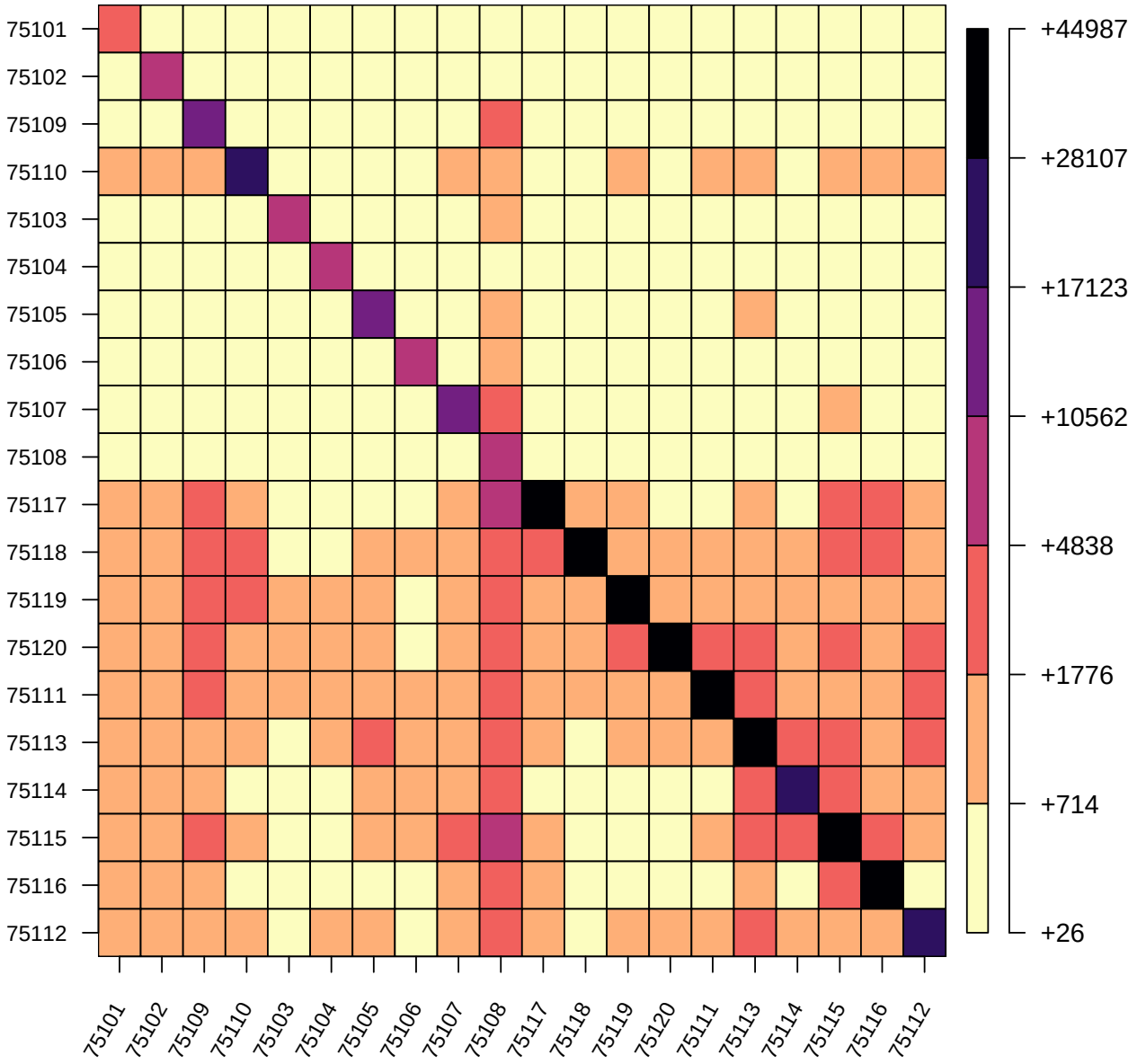
```
# pdf(file = "figures/paris/griffith.pdf", width = 10, height = 12)
plot_flows(y, index_o, index_d, type_plot = "griffith", xy_sf = xy_sf,
           contours_map = contours_map)
```




```
# dev.off()
```

- Heat map :

```
# pdf(file = "figures/paris/heatmap.pdf", width = 8, height = 8)
par(mar=c(7.5, 6.8, 2.1, 4.1), las = 1, lheight = .6) # adapt margins
plot_flows(y, index_o, index_d, type_plot = "heatmap", xy_sf = xy_sf)
```




```
# dev.off()
```

2.2 Migration data

The second data is from Abel and Sander (2014). The authors present data on bilateral migration flows between 196 countries from 1990 through 2010.

To conclude, we exhibit the map flow, the Sankey diagram, the circular diagram, the clustering heat map and the arc diagram for the migration application. The Figure depicts the 1% most important migration flows, which represent 11893 migration flows, 191 origins, 192 destinations and 196 unique sites over the five year period between 2005-2010. Although the number of origin-destination flows is considerably large, the various diagrams permit to identify the salient features of the migration data at the country level.

The world administrative boundaries are available on https://public.opendatasoft.com/explore/dataset/world-administrative-boundaries/export/

We import them:

```
world_sf <- read_sf("data/migration/world-administrative-boundaries.shp")
```

We use an appropriate CRS to better visualize the flows data:

```
poly_sf <- st_transform(world_sf, 3035)
```

Abel and Sander (2014) present data on bilateral migration flows between 196 countries from 1990 through 2010. The migrations flows are available at https://www.science.org/doi/abs/10.1126/science.1248676?sitei d=sci&keytype=ref&ijkey=ypit4%2Fxi7wo4M&

```
mig <- readxl::read_xlsx("data/migration/abel-database-s2.xlsx", sheet = 4,
                         skip = 2)
```

We prepare the flows data:

```
y <- NULL
index_o <- NULL
index_d <- NULL

for(i in 1:(nrow(mig) - 1)) {
  for (j in 4:(ncol(mig) - 1)) {
    if (mig[i, j] != 0) {
      index_o <- c(index_o, as.character(mig[i, 3]))
      index_d <- c(index_d, as.character(colnames(mig)[j]))
      y <- c(y, as.numeric(mig[i, j]))
    }
  }
}
```

We drop the Channel Islands which are not included in the world boundaries data base

```
good_index <- !(index_o == "CHI" | index_d == "CHI")
y <- y[good_index]
index_o <- index_o[good_index]
index_d <- index_d[good_index]
O <- unique(index_o)
D <- unique(index_d)
S <- union(O, D)
```

Note that $n_o = 190$, $n_d = 191$, $n = 195$

We prepare the centroids of the world countries' boundaries:

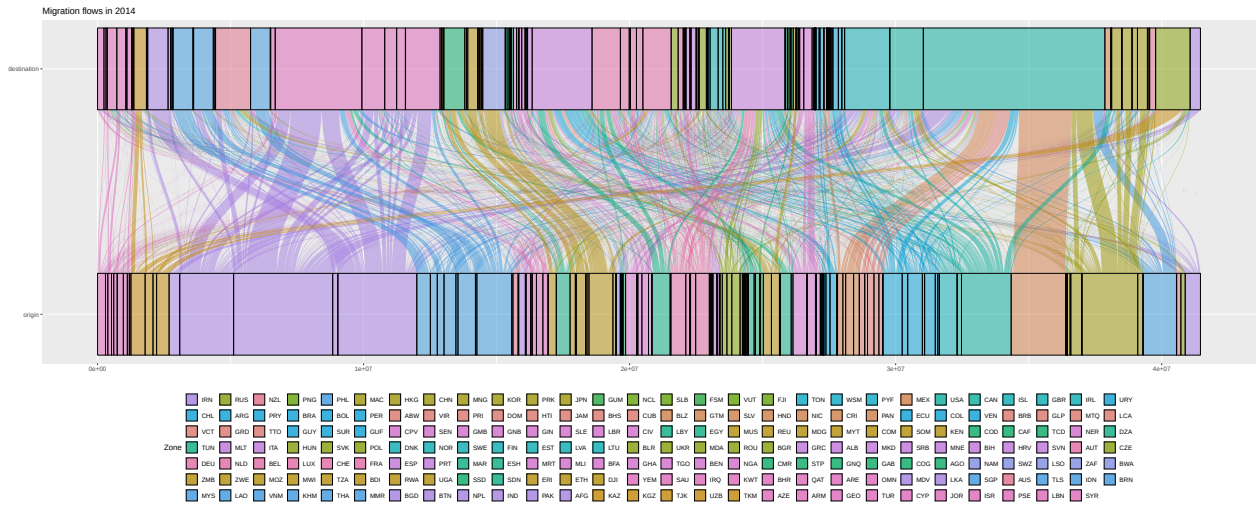
```
# create the spatial objects with the coordinates of the sites
xy_sf <- st_centroid(poly_sf[poly_sf$iso3 %in% S, ])
# drop the doublons
xy_sf <- xy_sf[-c(48, 70), ]
xy_sf$S <- xy_sf$iso3
xy <- st_coordinates(xy_sf)
rownames(xy) <- xy_sf$iso3
```

We define the colors with respect to the macro-geographial zones :

```
zones <- unique(xy_sf$region)
col_pal <- colorspace::qualitative_hcl(length(zones), palette = "Dark 3")
q4 <- col_pal[factor(xy_sf$region)]
names(q4) <- factor(xy_sf$S)
```

- Sankey plot :

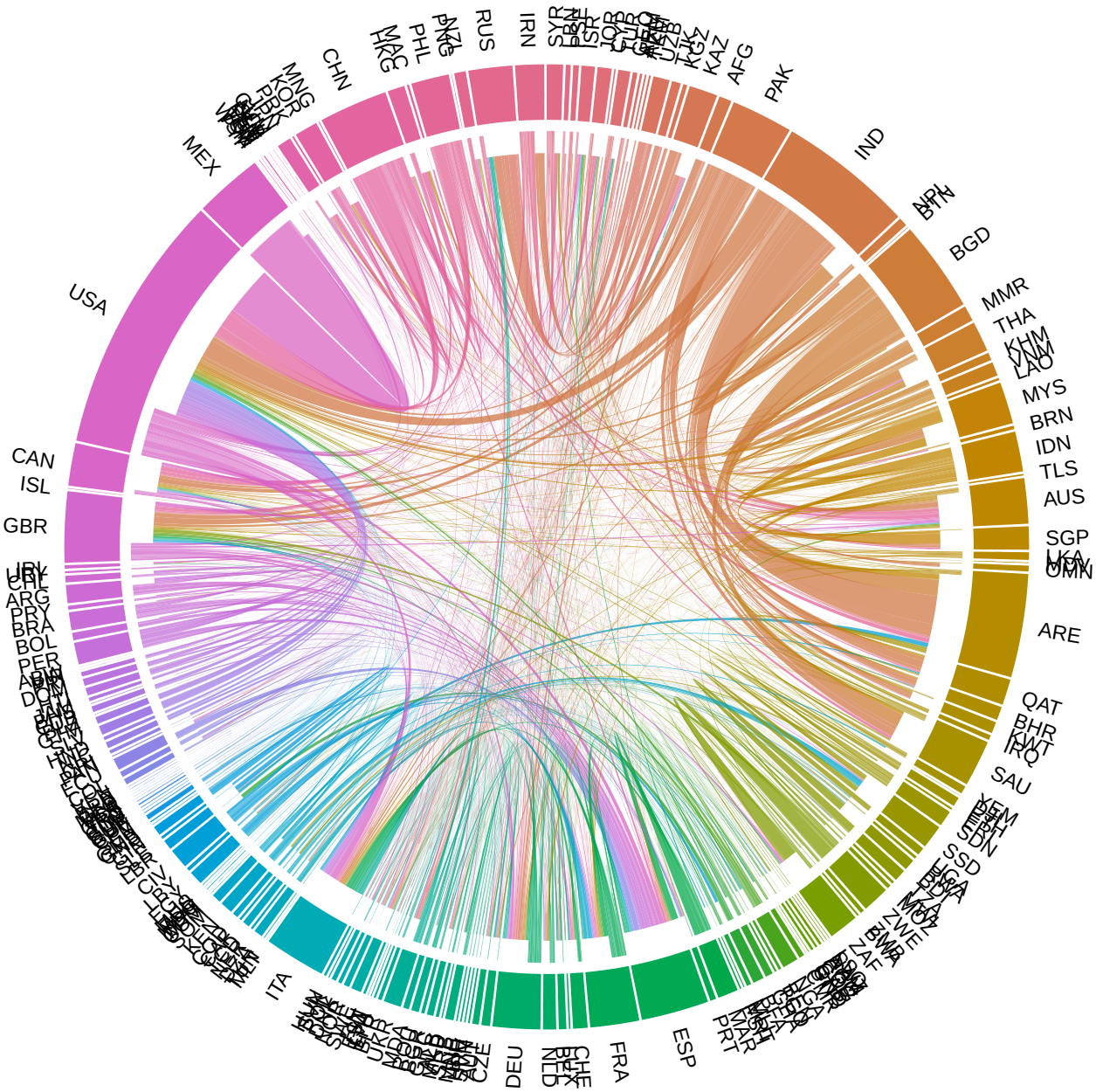
```
plot_flows(y, index_o, index_d, type_plot = "Sankey", xy_sf = xy_sf,
           color = q4,
           sankey.options = list(labels_od = c("origin", "destination"),
                                 nrows = 7,
                                 title = "Migration flows in 2014"))
```




```
# ggsave(filename = "figures/migration/remitt_mig.pdf",
#       width = 25, height = 10)
```

- Circular plot :

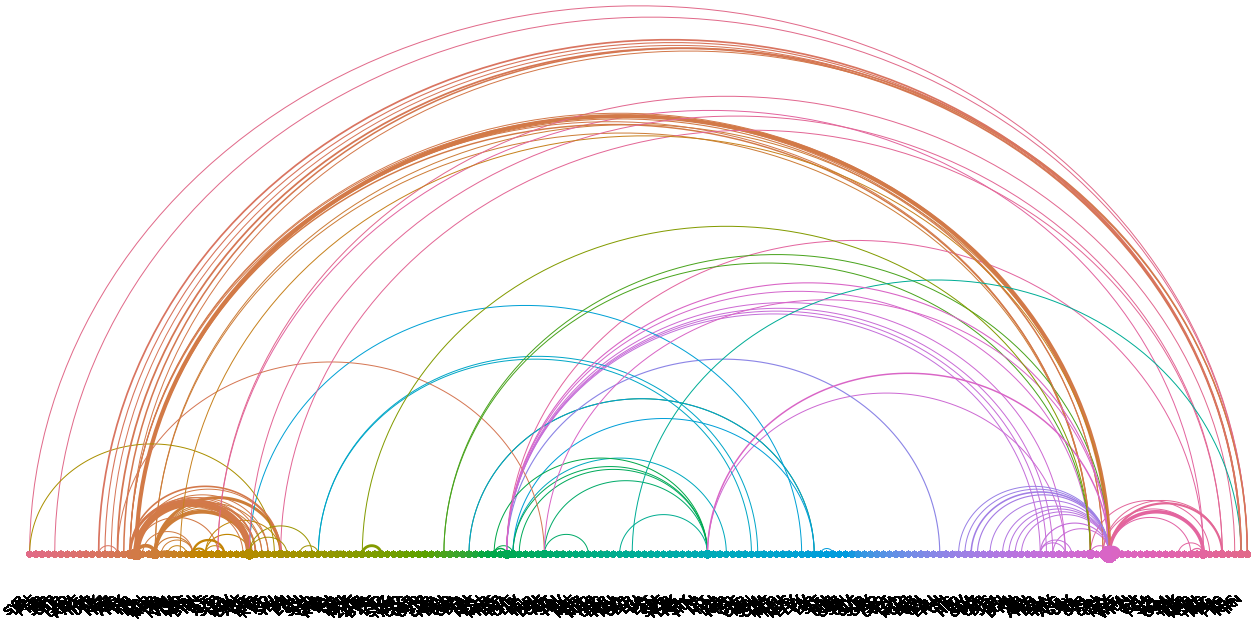
```
# pdf(file = "figures/migration/circular_mig.pdf", width = 20, height = 20)
plot_flows(y, index_o, index_d, type_plot = "circular", xy_sf = xy_sf)
```




```
# dev.off()
```

- Arc diagram :

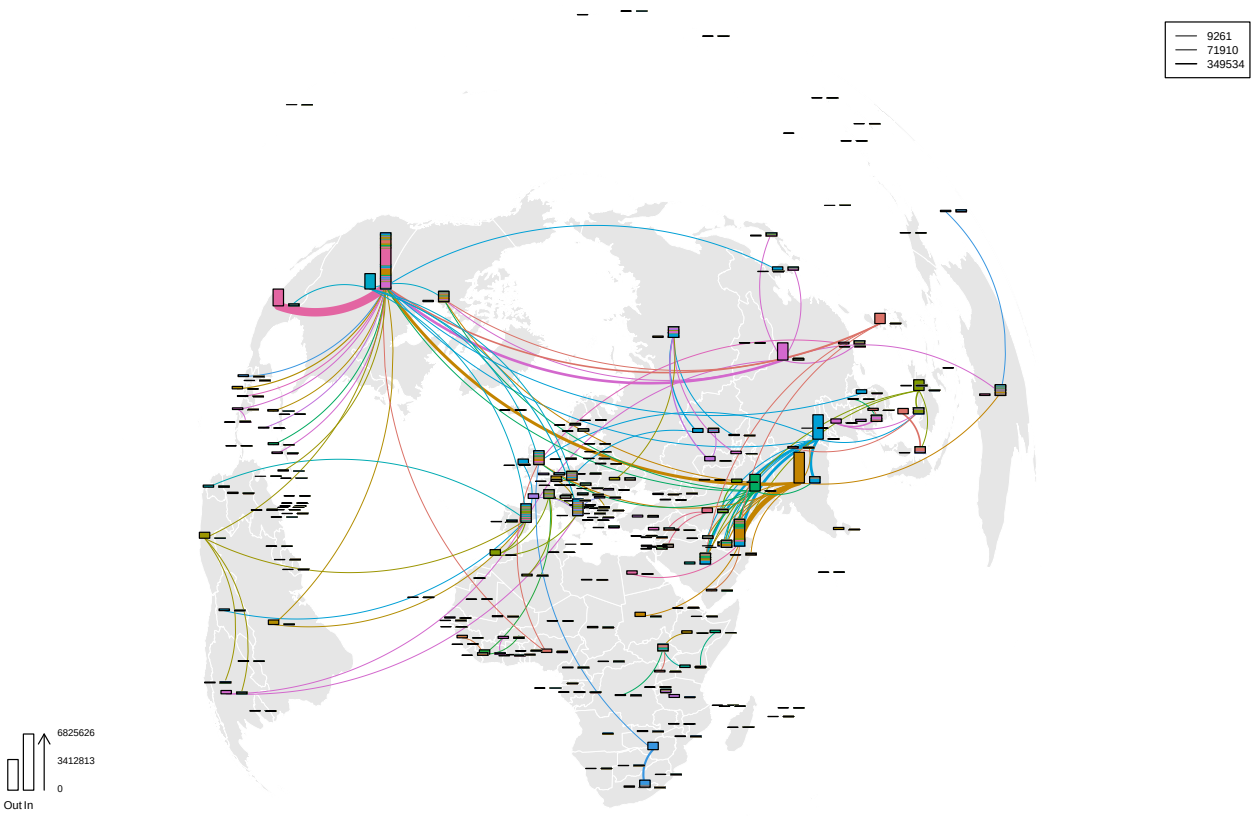
```
# pdf(file = "figures/migration/arc_mig.pdf", width = 20, height = 10)
plot_flows(y, index_o, index_d, ordering = "distance",
             type_plot = "arc", xy_sf = xy_sf,
           arc.options = list(maxcex = 1, maxlwd = 1, alpha.q = 0.99))
```



```
# dev.off()
```

- Flow map :

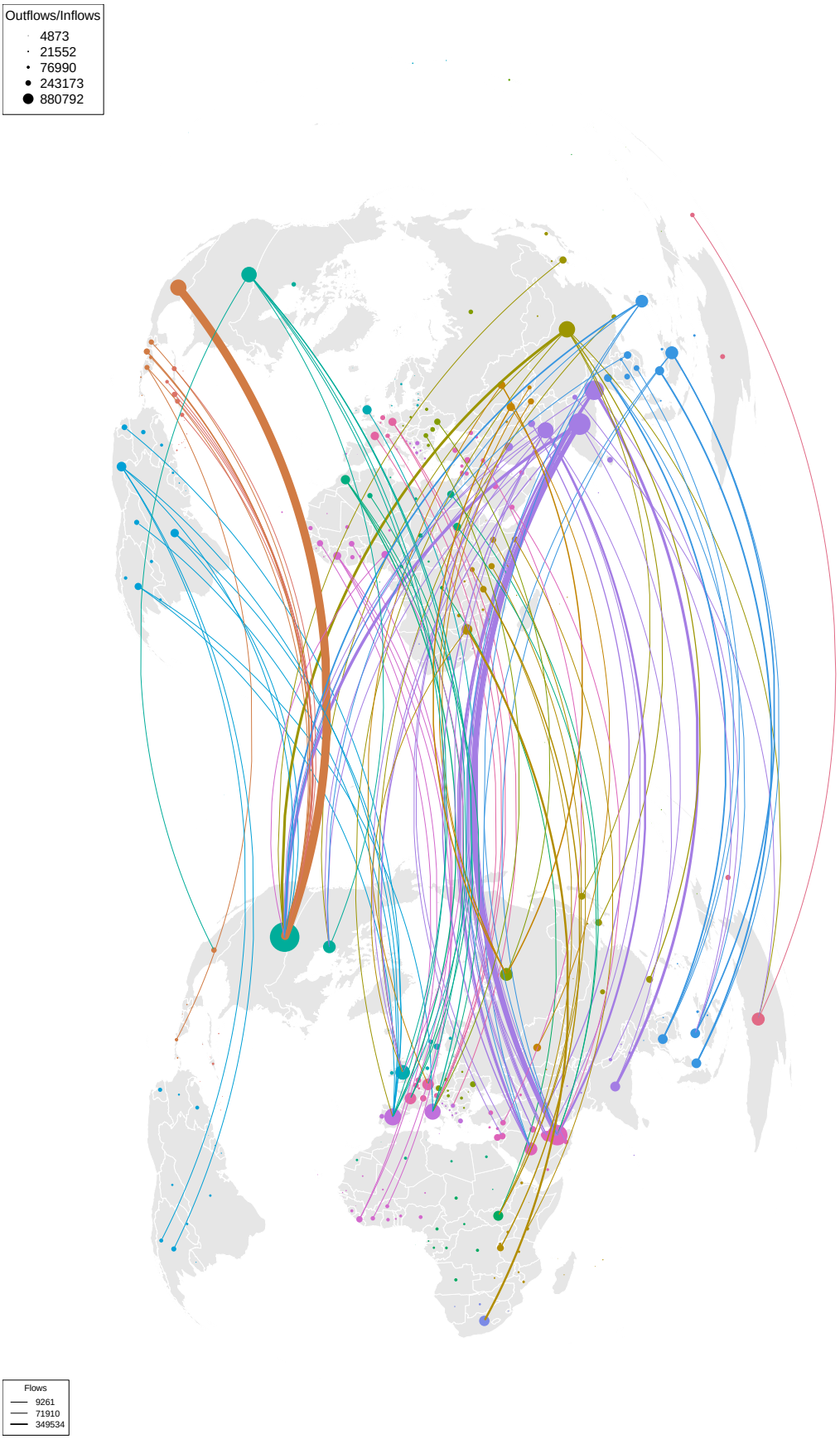
```
# pdf("figures/migration/flow_map_mig.pdf")
par(oma = c(0, 0, 0, 0), mar = c(0, 0, 0, 0))
plot_flows(y, index_o, index_d, type_plot = "flow_map", xy_sf = xy_sf,
           col = q4, contours_map = poly_sf,
           flow_map.options = list(alpha.q = 0.99,
                                   col_geometry = rgb(0.9, 0.9, 0.9),
                                   border_geometry = "white"))
```




```
# dev.off()
```

- Griffith :

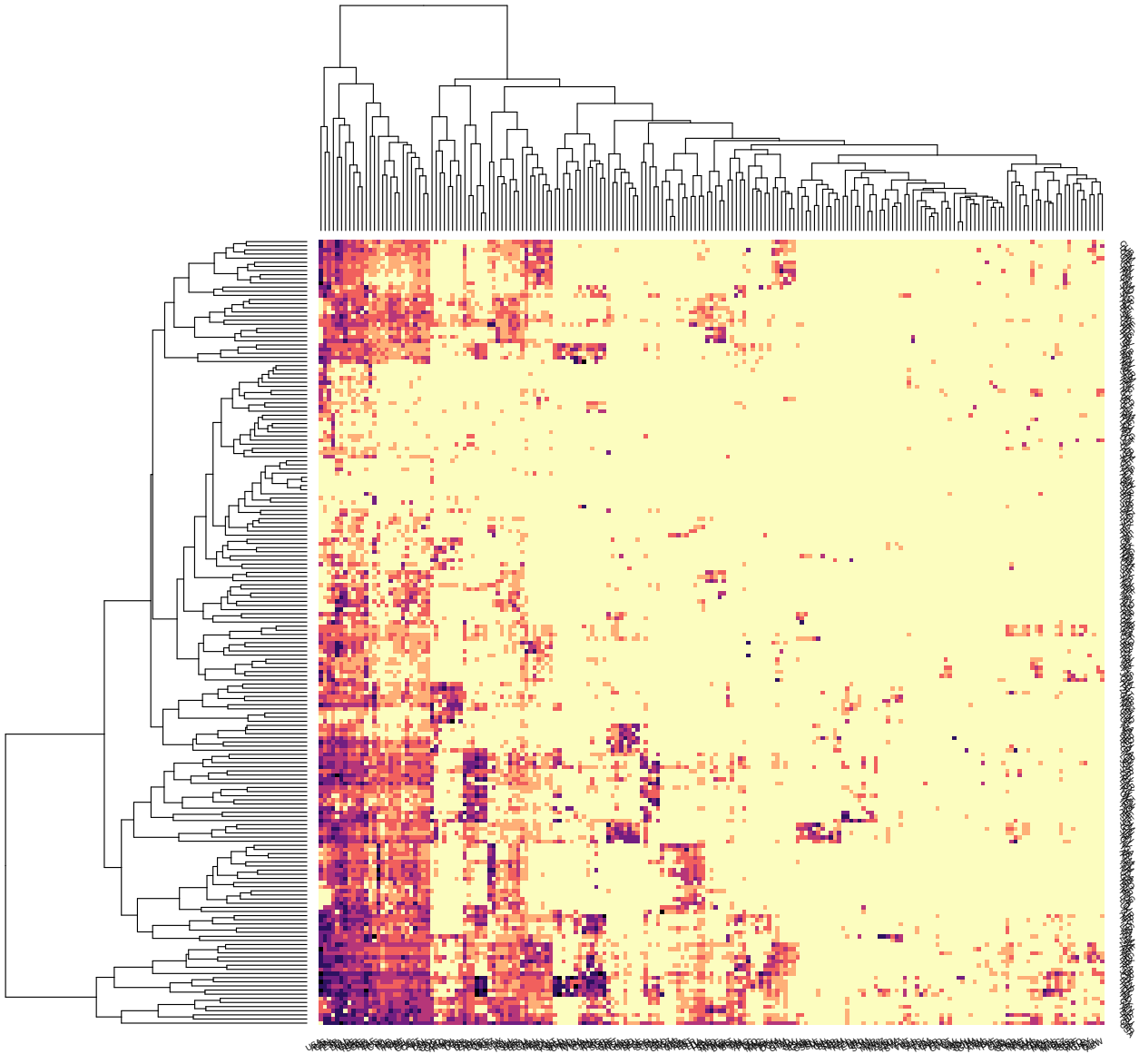
```
# pdf("figures/migration/griffith_mig.pdf", width = 12, height = 20)
plot_flows(y, index_o, index_d, type_plot = "griffith", xy_sf = xy_sf,
           contours_map = poly_sf, col = q4,
           griffith.options = list(
             scale_xy = c(1/20, - 1),
             alpha.q = 0.99,
             col_geometry_o = rgb(0.9, 0.9, 0.9),
             col_geometry_d = rgb(0.9, 0.9, 0.9)))
```


Outflows/Inflows
4873
21552
76990
243173
880792
Flows
9261
71910
349534

```
# dev.off()
```

- Heat map :

```
# pdf(file = "figures/migration/mig_heatmap.pdf", width = 20, height = 20)
par(mar=c(7.5, 6.8, 2.1, 4.1), las = 1, lheight = .6) # adapt margins
plot_flows(log(y), index_o, index_d, ordering = "clustering",
           type_plot = "heatmap", xy_sf = xy_sf,
           heatmap.options = list(style_class = "equal",
                                  n.class = 7,
                                  title = ""))
```



```
# dev.off()
```

3 Bibliography

Abel, G. J. and Sander, N. (2014). Quantifying global international migration flows. Science, 343(6178):1520–1522

Dargel, L. and Laurent, T. (2021). **spflow**: Spatial Econometric Interaction Models. **R** package version 0.1.0

Laurent T., Margaretic P. and Thomas-Agnan C. (2021). *Bilateral remittances and network effects: Do neighboring countries matter?*, *TSE WP*.