Spatial Flows Modelling with **R**

T. Laurent, P. Margaretic, C. Thomas-Agnan

April 19, 2019

Contents

Packages needed:

```r
require("arcdiagram") # representation of flows
require("cartography")# representation of spatial data
require("haven")      # import stata files
require("Matrix")     # sparse matrix
require("rgdal")      # import spatial data
require("spdep")      # spatial econometrics modelling
require("tidyverse")  # tidyverse data
require("maptools")   # spatial
```

How to present the data in practice

Simulated example

Let us consider the example used in Thomas-Agnan and LeSage (2014) in section 83.5.1. First, we define the variables used in the space of the n spatial regions.

Spatial flows data storage when $N = n^2$

When the number of flows N is equal to n^2 where n is the number of spatial regions, users can simplify the presentations of the data.

We consider $n = 8$ regions noted $R_1, R_2, ..., R_8$:

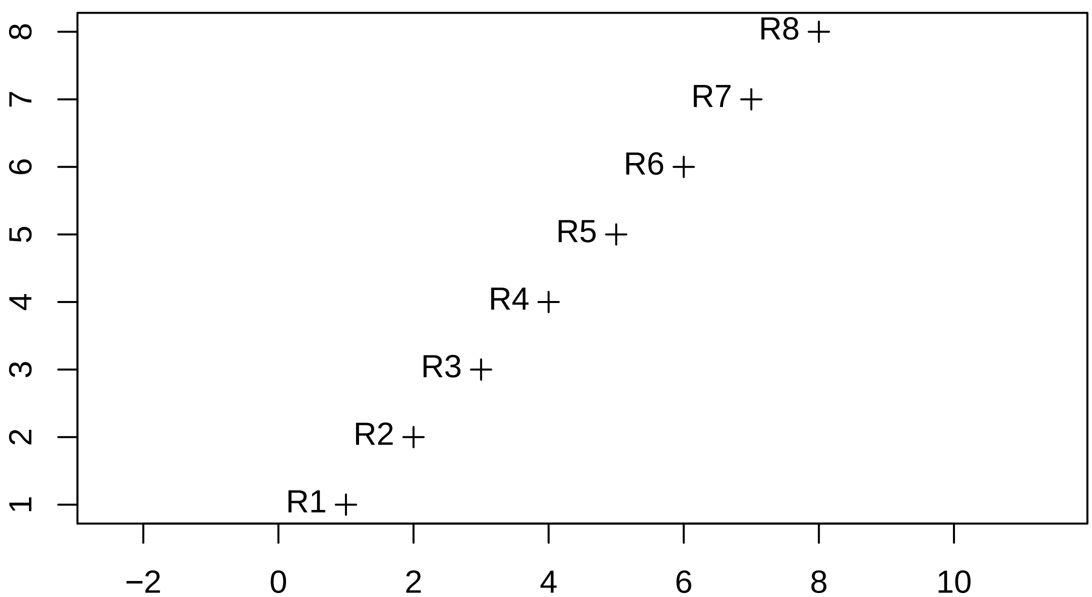
```r
n <- 8
id_region <- paste0("R", 1:8)
```

In Thomas-Agnan and LeSage (2014), the single vector $x' = (40, 30, 20, 10, 7, 10, 15, 25)$ is used. Hereby, we use a second variable to get a more general example, so we have the case where $R = 2$. Here, we store the explanatory variables in a **data.frame** which is the standard form for collecting the data with **R**.

```r
set.seed(123)
x <- data.frame(id = id_region,
                x1 = c(40, 30, 20, 10, 7, 10, 15, 25),
                x2 = runif(n),
                row.names = "id")
R <- ncol(x)
```

A set of n latitude and longitude coordinates (both equal to $1, 2, ..., 8$) is used. We used the **Spatial** norm proposed by Pebesma and Bivand (2005).

```r
x$long <- 1:8
x$lat <- 1:8
coordinates(x) <- ~ long + lat
plot(x, axes = T)
text(coordinates(x), id_region, pos = 2)
```



We define the associated spatial weight matrix W of size $n \times n$ based on two nearest neighbors. Note that when the number of flows $N = n^2$, we do not need to build the spatial weight matrices W_o, W_d, W_w of size $n^2 \times n^2$. The computations will be done by using the properties of the Kronecker products.

```r
w <- 0.5 * matrix(c(0, 1, 1, 0, 0, 0, 0, 0,
                    1, 0, 1, 0, 0, 0, 0, 0,
                    0, 1, 0, 1, 0, 0, 0, 0,
                    0, 0, 1, 0, 1, 0, 0, 0,
                    0, 0, 0, 1, 0, 1, 0, 0,
                    0, 0, 0, 0, 1, 0, 1, 0,
                    0, 0, 0, 0, 0, 1, 0, 1,
                    0, 0, 0, 0, 0, 1, 1, 0),
                  8, 8, byrow = T)
```

Besides, when $N = n^2$, the explanatory variables can be presented in the dimension $n \times p$ rather than the full dimension $n^2 \times p$ of flows.

Finally, the distances between flows can be presented in a matrix of size $n \times n$.

```
G <- as.matrix(dist(coordinates(x)))
```

We will see that the dependent variables Y will be presented also in a $n \times n$ matrix format.

Spatial flows data storage when $N \leq n^2$

When the number of flows N is lower than n^2, we have to present the data differently than in the previous section. This can happen when users decide to drop some observations which are badly informed or are extreme values.

Explanatory variables

We define the explanatory variables in the space of the flows $N \times p$. We use the **data.frame** format which contains one column noted **origin** and one column noted **dest**. The data are presented as in Lesage and Pace (2009) (see formula (2)).

```
flows_data <- data.frame(origin = rep(id_region, each = n),
                         dest = rep(id_region, n))
```

Then, we have to transform the explanatory variables from the spatial regions to the spatial flows. For doing this, we use the function *merge()* successively two times: the first time for defining the **origin** variable and the second time for defining the **dest** variable:

```
flows_data <- merge(flows_data, x@data, by.x = "dest", by.y = "row.names")
flows_data <- merge(flows_data, x@data, by.x = "origin", by.y = "row.names")
names(flows_data)[3:ncol(flows_data)] <- paste0(names(x@data),
  c(rep("_d", ncol(x@data)), rep("_o", ncol(x@data))))
```

Important step: the step of merging has disordered the observations. The idea is to keep the data ordered firstly by origin and secondly by destination. Let us print the first 6 flows :

```
flows_data <- flows_data[order(flows_data$origin, flows_data$dest), ]
head(flows_data)
```

```
##   origin dest x1_d      x2_d x1_o      x2_o
## 1     R1   R1   40 0.2875775   40 0.2875775
## 2     R1   R2   30 0.7883051   40 0.2875775
## 4     R1   R3   20 0.4089769   40 0.2875775
## 5     R1   R4   10 0.8830174   40 0.2875775
## 8     R1   R5    7 0.9404673   40 0.2875775
## 6     R1   R6   10 0.0455565   40 0.2875775
```

To produce an n^2 vector of distances g:

```
flows_data$g <- as.vector(G)
G_dot <- flows_data$g  - mean(G)
```

Spatial weight matrices W_o, W_d, W_w

To define the matrices W_o, W_d, W_w, we use the Kronecker product and the specificity of sparse matrices (functions *kronecker()* and *Diagonal()* in package **Matrix**):

```r
W_d <- kronecker(Diagonal(n), w)
W_o <- kronecker(w, Diagonal(n))
W_w <- kronecker(as(w, "Matrix"), w)
```

DGP of the Y variables

We simulate 9 different spatial autoregressive interaction models:

$y_9 = (I_N - \rho_o W_o - \rho_d W_d + \rho_w W_w)^{-1}(Z\delta + \epsilon)$,

$y_8 = (I_N - \rho_o W_o - \rho_d W_d + \rho_d \rho_o W_w)^{-1}(Z\delta + \epsilon)$,

$y_7 = (I_N - \rho_o W_o - \rho_d W_d)^{-1}(Z\delta + \epsilon)$,

$y_6 = (I_N - \rho_{odw}(W_o + W_d + W_w)/3)^{-1}(Z\delta + \epsilon)$,

$y_5 = (I_N - \rho_{od}(W_o + W_d)/2)^{-1}(Z\delta + \epsilon)$,

$y_4 = (I_N - \rho_w W_w)^{-1}(Z\delta + \epsilon)$,

$y_3 = (I_N - \rho_o W_o)^{-1}(Z\delta + \epsilon)$,

$y_2 = (I_N - \rho_d W_d)^{-1}(Z\delta + \epsilon)$,

$y_1 = (Z\delta + \epsilon)$,

with $Z = (1_N, X_o, X_d, g)$ and $\delta = (\alpha, \beta_o, \beta_d, \gamma)$. We generate a set of flows Y with $\alpha = 0$, $\beta_d = (0.5, 1)$, $\beta_o = (1.5, 2)$, $\gamma = -0.5$, $\rho_d = 0.4$, $\rho_o = 0.4$, and $\rho_w = 0.2$.

```r
N <- n^2
delta <- c(0, 0.5, 1, 1.5, 2, -0.5)
id_x_d <- substr(names(flows_data), nchar(names(flows_data)) - 1,
                 nchar(names(flows_data))) == "_d"
x_d <- as(flows_data[ , id_x_d], "matrix")
id_x_o <- substr(names(flows_data), nchar(names(flows_data)) - 1,
                 nchar(names(flows_data))) == "_o"
x_o <- as(flows_data[ , id_x_o], "matrix")
Z <- cbind(rep(1, N), x_d, x_o, flows_data$g)
rho_d <- 0.4
rho_o <- 0.3
rho_w <- -0.4
```

To simulate the data:

```r
set.seed(123)
z_delta <- Z %*% delta + rnorm(N)
```

Note that the Y variable is presented as an $n \times n$ matrix when the number of flows is equal to $N = n^2$, otherwise we will present Y as a vector of size N.

```r
flows_data$y_9 <- as.numeric(solve(diag(N) - rho_d * W_d - rho_o * W_o - rho_w * W_w,
                         z_delta))
Y_9 <- matrix(flows_data$y_9, n, n)
```

```r
flows_data$y_8 <- as.numeric(solve(diag(N) - rho_d * W_d - rho_o * W_o + rho_d * rho_o * W_w,
                         z_delta))
Y_8 <- matrix(flows_data$y_8, n, n)
```

```r
flows_data$y_7 <- as.numeric(solve(diag(N) - rho_d * W_d - rho_o * W_o,
                             z_delta))
Y_7 <- matrix(flows_data$y_7, n, n)
```

```r
flows_data$y_6 <- as.numeric(solve(diag(N) - rho_d * (W_d + W_o + W_w)/3,
                             z_delta))
Y_6 <- matrix(flows_data$y_6, n, n)
```

```r
flows_data$y_5 <- as.numeric(solve(diag(N) - rho_d * (W_d + W_o)/2,
                             z_delta))
Y_5 <- matrix(flows_data$y_5, n, n)
```

```r
flows_data$y_4 <- as.numeric(solve(diag(N) - rho_w * W_w,
                             z_delta))
Y_4 <- matrix(flows_data$y_4, n, n)
```

```r
flows_data$y_3 <- as.numeric(solve(diag(N) - rho_o * W_o,
                             z_delta))
Y_3 <- matrix(flows_data$y_3, n, n)
```

```r
flows_data$y_2 <- as.numeric(solve(diag(N) - rho_d * W_d,
                             z_delta))
Y_2 <- matrix(flows_data$y_2, n, n)
```

```r
flows_data$y_1 <- as.numeric(z_delta)
Y_1 <- matrix(z_delta, n, n)
```

Finally, the data set corresponding to the flows is presented in that form :

```r
head(flows_data)
```

```
##   origin dest x1_d      x2_d x1_o      x2_o        g       y_9      y_8
## 1     R1   R1   40 0.2875775   40 0.2875775 0.000000 119.71806 167.7971
## 2     R1   R2   30 0.7883051   40 0.2875775 1.414214 112.74466 161.9849
## 4     R1   R3   20 0.4089769   40 0.2875775 2.828427 106.89864 152.6888
## 5     R1   R4   10 0.8830174   40 0.2875775 4.242641  97.43629 140.4047
## 8     R1   R5    7 0.9404673   40 0.2875775 5.656854  94.30298 135.0307
## 6     R1   R6   10 0.0455565   40 0.2875775 7.071068  95.92412 136.1662
##        y_7       y_6       y_5      y_4       y_3      y_2      y_1
## 1 213.3235 118.49049 121.40082 65.79857 104.86232 128.8365 80.30226
## 2 207.8994 113.61274 115.97802 59.69747  97.88771 124.7731 75.42618
## 4 196.6074 106.70133 109.01826 57.76874  92.21086 117.8983 71.12863
## 5 182.0858  97.46131  99.43396 51.55295  82.86082 109.0750 64.40736
## 8 174.9610  93.73032  95.69547 50.49977  79.71650 105.4401 62.31648
## 6 175.4944  95.02291  97.06055 52.29089  81.36999 106.5429 63.80024
```

Real data

This dataset was found here: https://www.wto.org/english/res_e/booksp_e/advancedwtounctad2016_e.pdf

p.39: "The primary source of information for aggregated (country-level) bilateral trade flows is the International Monetary Fund (IMF)'s Direction of Trade Statistics (DOTS). The database covers 184 countries. Annual data are available from 1947, while monthly and quarterly data start from 1960. Data are reported in US dollars. Relying on DOTS and other national sources of data, Barbieri and Keshk have created a database (Correlates of War Project) that tracks total national trade and bilateral trade flows (imports and exports) between states from 1870-2009 in current US dollars."

p.40 : “In all the applications presented in this chapter, the results are obtained from the same balanced panel data covering the aggregate manufacturing sector of 69 countries over the period 1986-2006. The sample combines data from several sources. Most importantly, it includes consistently constructed international and intra-national trade flows data, which were assembled and provided by Thomas Zylkin. The original sources for the international trade data are the UN COMTRADE database and the CEPII TradeProd database. COMTRADE is the primary data source and TradeProd is used for instances when it includes positive flows for observations when no trade flows are reported in COMTRADE. Intra-national trade for each country is constructed as the difference between total manufacturing production and total manufacturing exports. Importantly, both of these variables are reported on a gross basis, which ensures consistency between intra-national and international trade. Three sources are used to construct the production data: the UN UNIDO INDSTAT database, the CEPII TradeProd database, and the World Bank’s TPP database. The data on RTAs were taken from Mario Larch’s Regional Trade Agreements Database. Finally, all standard gravity variables including distance, contiguous bor-=ders, common language, and colonial ties are from the CEPII GeoDist database. An important advantage of the GeoDist database is that the weighted-average methods used to construct distance ensure consistency between the measures of intra-national and international distance, because each method uses population-weighted distances across the major economic centres within or across countries”

To import and prepare the data:

```r
source("./R/trade_data.R")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/laurent/Documents/paula/Paula Flows/data/World WGS84/Pays_WGS84.shp", layer: "Pays_WGS
## with 251 features
## It has 1 fields
```

The flows are presented in the object **wto**. The number of flows is equal to 4761. The variable of interest is the variable **trade** and the distances between origin and destination are given in the variable **DIST**.

```r
head(wto)
```

```
## # A tibble: 6 x 9
##   exporter importer pair_id  year   trade   DIST  CNTG  LANG  CLNY
##   <chr>    <chr>      <dbl> <dbl>   <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 ARG      ARG        12339  2006 32313.      0     0     0     0
## 2 ARG      AUS            1  2006   108.  12045.     0     0     0
## 3 ARG      AUT            2  2006    25.2 11751.     0     0     0
## 4 ARG      BEL            4  2006   189.  11305.     0     0     0
## 5 ARG      BGR            3  2006    17.0 12116.     0     0     0
## 6 ARG      BOL            6  2006   392.   1866.     1     1     0
```

```r
dim(wto)
```

```
## [1] 4761    9
```

The spatial units (the countries) are presented in the object **wto_spatial**. The number of countries is equal to 69. We define the variable **GDP** which is the GDP given by the United Nations in 2017 (https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)).

```r
dim(wto_spatial)
```

```
## [1] 69  2
```

```r
head(wto_spatial@data)
```

```
##     NOM     GDP
## 245 ARG  637486
## 30  AUS 1408675
```

```
## 242 AUT  416835
## 23  BEL  494763
## 45  BGR   58222
## 207 BOL   37508
```

Case of a square flow matrix $N = n^2$

The data consists in $N = 4761$ flows observed on $n = 69$ countries. As $N = n^2$, we first present the flows and the distances between origin/destination as matrices Y and G of size 69×69.

```r
Y_wto <- matrix(wto$trade, 69, 69)
G_wto <- matrix(wto$DIST, 69, 69)
```

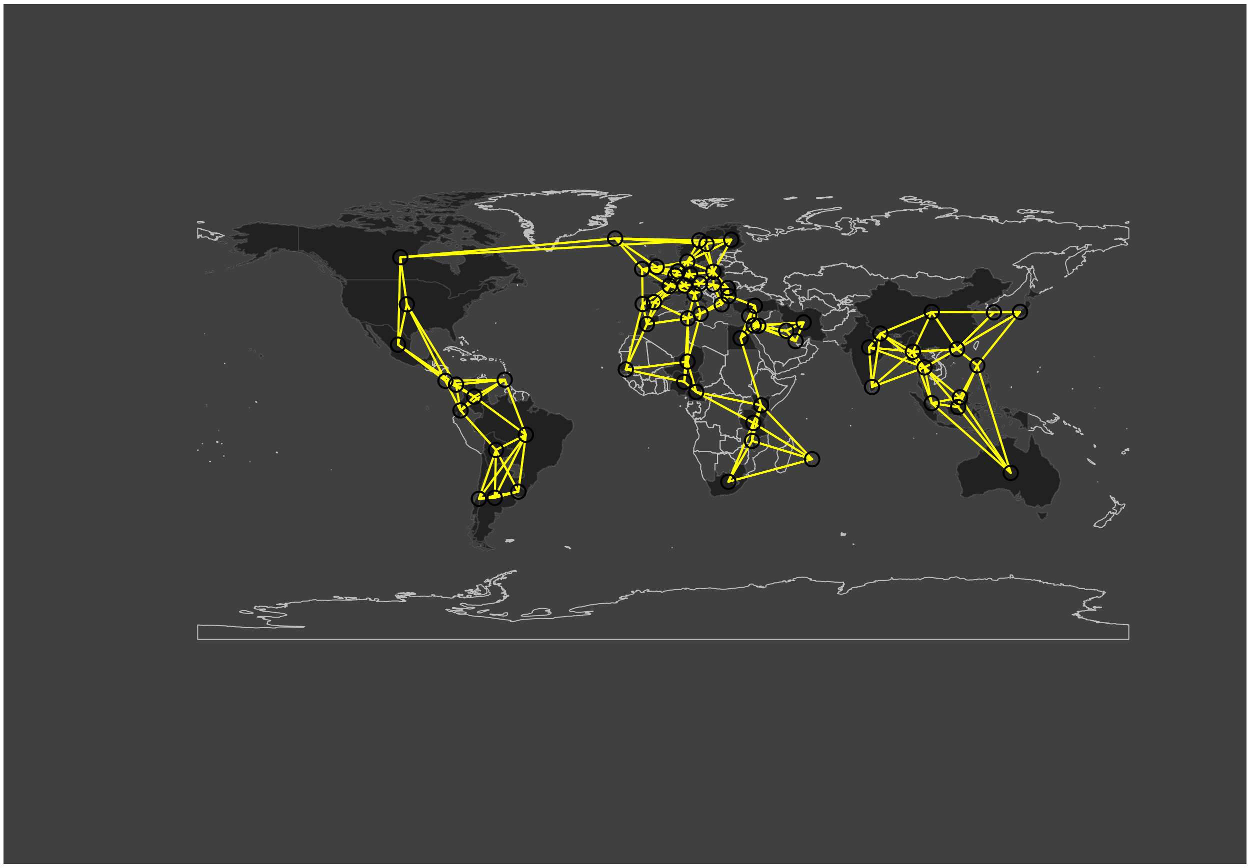
Spatial weight matrix

We compute the spatial weight matrix based on the 4 nearest neighbours:

```r
ppv1 <- knn2nb(knearneigh(coordinates(wto_spatial), k = 4, longlat = T), sym = T)
wto_spatial_weight <- nb2listw(ppv1)
wto_W <- listw2mat(wto_spatial_weight)
```

We represent the links on the map:

```r
par(bg = "grey25")
plot(world, border = "grey", lwd = 0.5)
plot(wto_spatial, col = "grey13", border = "grey25",
     bg = "grey25", lwd = 0.5, add = T)
plot(ppv1, coordinates(wto_spatial), add = T, col = "yellow")
```



Case of a non-square flow matrix $N < n^2$

If we decide to filter the flows and only keep trades which are higher than 0, we have to present the data differently.

First, we add the GDP to the data observed at the flows level:

```
wto$DGP_d <- wto_spatial@data$GDP
wto$DGP_o <- rep(wto_spatial@data$GDP, each = 69)
```

Then, we identify the ids of the flows we want to drop:

```
ind_filter <- wto$exporter != wto$importer & wto$trade > 0
wto_filter <- wto[ind_filter, ]
```

Spatial weight matrices W_o, W_d, W_w

To define the matrices W_o, W_d, W_w, we use the Kronecker product by using the specificity of sparse matrices:

```
wto_W_d <- kronecker(Diagonal(69), wto_W)
wto_W_o <- kronecker(wto_W, Diagonal(69))
wto_W_w <- kronecker(as(wto_W, "Matrix"), wto_W)
```

Then, we only select the flows that we are interested in:

```
wto_W_d_filter <- wto_W_d[ind_filter, ind_filter]
wto_W_o_filter <- wto_W_o[ind_filter, ind_filter]
wto_W_w_filter <- wto_W_w[ind_filter, ind_filter]
```

Vizualisation

We present in this section some **R** packages which permit to visualize spatial flows data.

Simulated example

We use the package **arcdiagram** (see https://github.com/gastonstat/arcdiagram) which is convenient for our simulated data because our spatial flows can be assimilated to a graph structure. Indeed, the distances between two consecutives geographical observations are exactly the same and thus we can represent them as nodes.

In the graph structure, one node can not be linked with itself. In other term, for plotting the intra flow, we will plot a node with a size proportionnal to the value of the intra flows. For doing this, we first need to define the minimum and maximum cex for representing the nodes :

```
max_symbol_size <- 4
min_symbol_size <- 1
```

And then, compute the cex of each node:

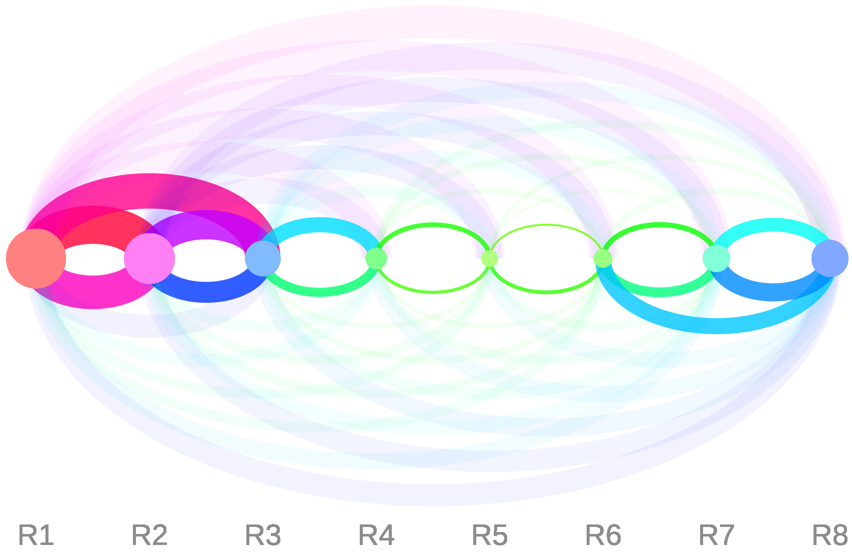
```
plotvar <- diag(Y_8)
symbol_size <- ((plotvar - min(plotvar))/(max(plotvar) - min(plotvar))*
                  (max_symbol_size - min_symbol_size) + min_symbol_size)
```

Then, we want to represent the flows with a width proportionnal to the value of the flows. Thus, we do something similar:

```
max_lwd <- 20
min_lwd <- 1
flows_lwd <- ((flows_data[, "y_8"] - min(flows_data[, "y_8"]))/
                (max(flows_data[, "y_8"]) - min(flows_data[, "y_8"]))*
                  (max_lwd - min_lwd) + min_lwd)
```

Finally, we represent the flows such that the flows above the axis correspond to the flows from a left node to a right node and the flows below the axis correspond to the flows from a right node to a left node. We decide to plot with a deep color the flows whose origin and destination are neighbors.

```
arcplot(as(flows_data[, 1:2], "matrix"), labels = id_region, las = 1,
        show.nodes = TRUE,
        above = as.character(flows_data[, 1]) <= as.character(flows_data[, 2]),
        cex.nodes = symbol_size, lwd.arcs = flows_lwd,
        col.arcs = hsv(flows_data[, "y_8"]/max(flows_data[, "y_8"]),
                       alpha = ifelse(as.vector(t(w)) > 0, 0.8, 0.05)),
        col.nodes = hsv(plotvar/max(plotvar), 0.5))
```



Real data

We use here the package **Cartography** for representing the flows.

```
mtq_mob <- getLinkLayer(
  x = wto_spatial,
  xid = "NOM",
  df = wto,
  dfid = c("exporter","importer")
)
```

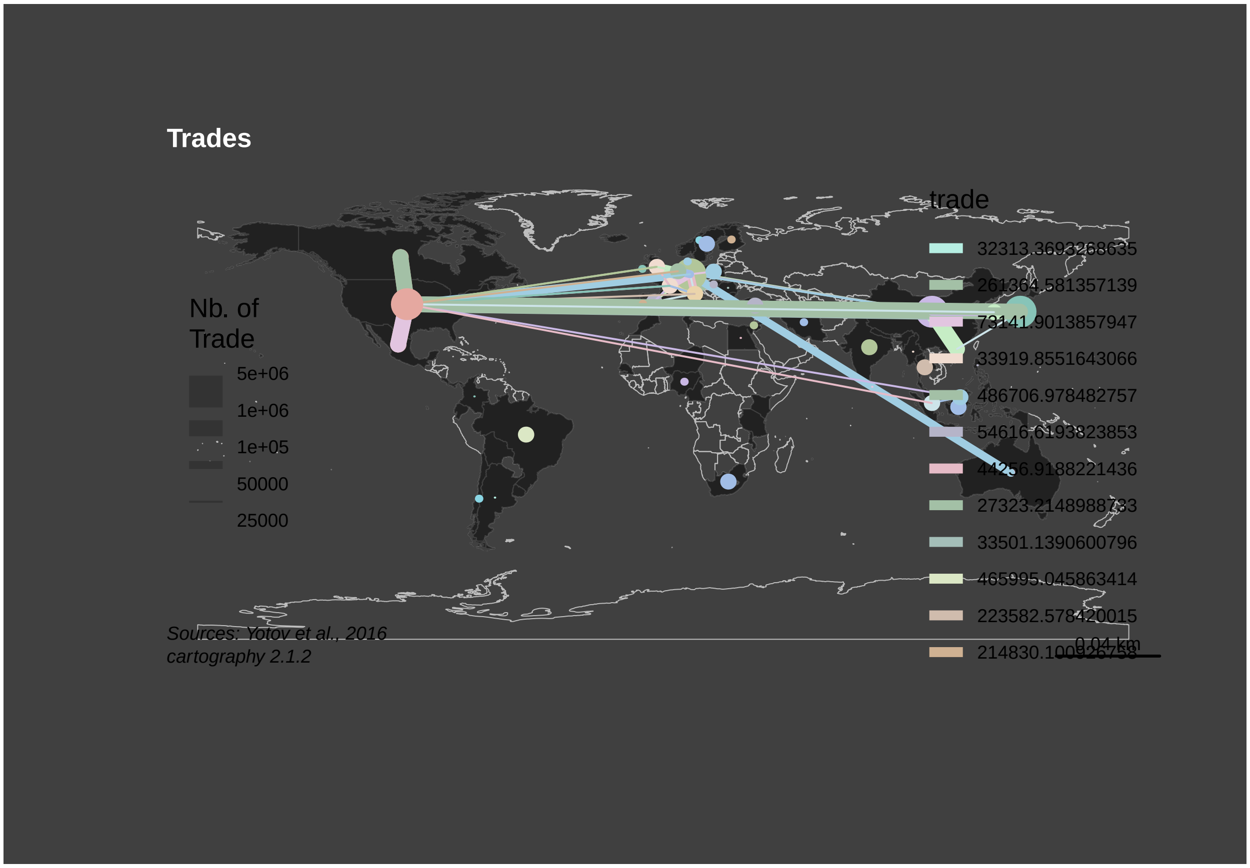
```
## Warning in st_centroid.sfc(x = sf::st_geometry(x), of_largest_polygon
## = max(sf::st_is(sf::st_as_sf(x), : st_centroid does not give correct
## centroids for longitude/latitude data
```

```
## Linking to GEOS 3.5.1, GDAL 2.2.2, PROJ 4.9.2
```

```
par(bg = "grey25")
# plot municipalities
plot(world, border = "grey", lwd = 0.5)
plot(wto_spatial, col = "grey13", border = "grey25",
     bg = "grey25", lwd = 0.5, add = T)
# plot graduated links
gradLinkTypoLayer(
  x = mtq_mob,
  xid = c("exporter","importer"),
  df = wto,
  dfid = c("exporter","importer"),
  var = "trade",
  breaks = c(25000, 50000, 100000, 1000000, 5000000),
  lwd = c(1, 4, 8, 16),
  var2 = "trade",
  legend.var.pos = "left",
  legend.var.title.txt = "Nb. of\nTrade",
)

# map layout
```

```r
layoutLayer(title = "Trades",
            sources = "Sources: Yotov et al., 2016",
            author = paste0("cartography ", packageVersion("cartography")),
            frame = FALSE, col = "grey25", coltitle = "white",
            tabtitle = TRUE)
```



Non spatial modelling

Gravity model

The form of the gravity model is $Y = \alpha 1_N + X_o\beta_o + X_d\beta_d + \gamma g + \epsilon$. To fit this model with **R**, we propose two options:

- We implement the formulas proposed by LeSage and Pace (2008) which avoid to store the full vectors for the explanatory variables by using the kronecker properties.
- Use the function *lm()* applied to the vectorized form of the data set.

LeSage and Pace (2008) estimation

LeSage and Pace (2008) show that we can avoid to store the flows data by using the property of the Kroneker product in the space of the regions data. In their paper, they consider the case where x is centered which permits some further simplifications in the resolution of the problem. We call this function *gravity_model()*. It takes as input arguments :

- **x**, a **data.frame** or a matrix with explanatory variable observed on the n geographical sites.
- **Y**, the matrix of flows of size $n \times n$,
- **G**, the matrix of distance of size $n \times n$,
- **ind_d**, the indices of the variables in **x** which will be used at the destination,
- **ind_o**, the indices of the variables in **x** of the variables used at the origin.

```
source("./R/gravity_model.R")
```

Applications

With the toy data

```
gravity_model(x = x@data, Y = Y_1, G = G, ind_d = 1, ind_o = c(1, 2))
```

```
##                    [,1]
## (Intercept) 41.0014536
## x1_d         0.4768285
## x1_o         1.5009268
## x2_o         2.1677060
## g           -0.4791414
```

We compare the results with the ones obtained with the function *lm()* :

```
gravity_flows <- lm(y_1 ~ x1_d + x1_o + x2_o + g, data = flows_data)
```

The function *summary()* gives also the standard errors of the estimated coefficients and the results of the t-test. It also gives the values of the R^2.

```
summary(gravity_flows)
```

```
##
## Call:
## lm(formula = y_1 ~ x1_d + x1_o + x2_o + g, data = flows_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.75706 -0.55469  0.02289  0.59376  2.31347
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.89431    0.41638   2.148   0.0358 *
## x1_d         0.47683    0.01115  42.761  < 2e-16 ***
## x1_o         1.50093    0.01130 132.863  < 2e-16 ***
## x2_o         2.16771    0.37856   5.726 3.66e-07 ***
## g           -0.47914    0.04592 -10.435 5.09e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9233 on 59 degrees of freedom
## Multiple R-squared:  0.9971, Adjusted R-squared:  0.9969
## F-statistic:  5061 on 4 and 59 DF,  p-value: < 2.2e-16
```

Remark 1: Unsurprisingly, we find the same values of the estimates for β_o, β_d and γ than those obtained with the function *gravity_model()*. The estimate of the intercept is different because the data have been centered.

Remark 2: to compute the estimates with the *lm()* function, the user needs to work with the full matrix of size $N \times 2p$ where p is the number of explanatory variable.

With the real data

```
gravity_model(x = matrix(wto_spatial@data[, "GDP"], dimnames = list(1:69, "GDP")), Y = Y_wto, G = G_wto
```

```
##                      [,1]
## (Intercept)  2.047246e+04
## GDP_d        4.412317e-03
## GDP_o        4.340411e-03
## g           -1.998171e+00
```

We compare the results with the ones obtained with the *lm()* function :

```
gravity_flows_wto <- lm(trade ~ DGP_d + DGP_o + DIST, data = wto)
summary(gravity_flows_wto)
```

```
##
## Call:
## lm(formula = trade ~ DGP_d + DGP_o + DIST, data = wto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -89851   -9090   -1785    5632 4051633
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.125e+04  2.402e+03   4.686 2.87e-06 ***
## DGP_d        4.412e-03  4.372e-04  10.093  < 2e-16 ***
## DGP_o        4.340e-03  4.372e-04   9.929  < 2e-16 ***
## DIST        -1.998e+00  2.704e-01  -7.389 1.74e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 83840 on 4757 degrees of freedom
## Multiple R-squared:  0.04759,    Adjusted R-squared:  0.04699
## F-statistic: 79.23 on 3 and 4757 DF,  p-value: < 2.2e-16
```

Remark: in the case we want to work with positive flows only, we cannot use the function *gravity_model()* because the formula is adapted to the case where $N = n^2$. For example :

```
gravity_flows_wto_filter <- lm(trade ~ DGP_d + DGP_o + DIST, data = wto_filter)
summary(gravity_flows_wto_filter)
```

```
##
## Call:
## lm(formula = trade ~ DGP_d + DGP_o + DIST, data = wto_filter)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -19413  -1546   -394    754 213766
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  2.097e+03  2.423e+02   8.658   <2e-16 ***
## DGP_d        9.532e-04  4.320e-05  22.065   <2e-16 ***
## DGP_o        8.726e-04  4.300e-05  20.290   <2e-16 ***
## DIST        -3.197e-01  2.737e-02 -11.682   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8165 on 4550 degrees of freedom
## Multiple R-squared:  0.1735, Adjusted R-squared:  0.1729
## F-statistic: 318.3 on 3 and 4550 DF,  p-value: < 2.2e-16
```

Spatial Modelling

When the number of flows N is equal to n^2, this implies some simplifications in the computations. However, this is not always the case in practice. That is why we consider the two options :

- $N = n^2$
- $N < n^2$

Spatial Autoregressive Interaction Models when $N = n \times n$

We use the Bayesian SAR method for estimating the parameters. Before estimating the parameters in the SAR flows model, we have to create intermediate functions :

- *ftrace1()*, which computes the trace of the spatial weight matrices W, W^2, W^3, $\ldots$, W^{miter},
- *fodet1()*, which computes the jacobian matrix in the case of the model (9), i.e. the model with the 3 spatial weight matrices W_o, W_d, W_w
- *lndetmc()*, which computes the jacobian matrix in the case of a model with only one spatial weight matrix,
- *c_sarf()*, which computes the log likelihood conditionnally to ρ.

Traces of the spatial weight matrix

First we code the function **ftrace1()** which computes the traces of W, W^2, W^3, $\ldots$, W^{miter}. The two possible methods are **"exact"** (based on the computation of W, W^2, W^3, $\ldots$, W^{miter}) or **"approx"** (based on an MCMC approximation, Barry and Pace, 99). The argument **miter** corresponds to the desired maximum order trace and **riter** the maximum number of iterations used to estimate the trace.

```
ftrace1(w, method = "exact", miter = 10, riter = 50)
```

Example: we compute the traces on the 10 first powers of the spatial weigh matrix. Here we do not use an approximation because the size of the matrix is small

```
(traces <- ftrace1(w))
```

```
##  [1] 0.0000000 3.5000000 0.7500000 2.3750000 0.9375000 1.9062500 0.9843750
##  [8] 1.6484375 0.9960938 1.4785156
```

By using the algorithm proposed by Barry and Pace (1999), the approximated traces are equal to:

```
(traces_approx <- ftrace1(w, method = "approx", miter = 10, riter = 50000))
```

```
##  [1] 0.0000000 3.5000000 0.7481150 2.3685700 0.9359388 1.9001875 0.9824009
##  [8] 1.6426837 0.9935809 1.4730344
```

Computation of the determinant

General case with W_o, W_d, W_w

To compute the log determinant in the case of the full model (model 9), we code the function **fodet1()**. The input arguments are:

- **parms**, a **numeric** vector containing ρ_1, ρ_2, ρ_3,
- **traces**, a **numeric** vector containing the estimated traces of W, W^2, $\ldots$,W^{miter},
- **n**, the sample size.

```
fodet1(parms, traces, n)
```

```
source("./R/fodet1.R")
```

Case with only one spatial weight matrix (W_o, W_d or W_w)

In the particular case where there is a single spatial weight matrix (model 2 to 6 in Lesage and Pace, 2008), the algorithm is much simpler because the computation of $Ln|I_N - \rho W_S|$ where $S = o, d, w, o + d, o + d + w$ can be expressed directly as a function of the Jacobian of W. First, the user has to compute the trace of the matrix W by using the function *ftrace1()* and then compute the log determinant by using the function *lndetmc()*.

The function takes as input arguments:

- **parms**, a scalar usually corresponding to the value of ρ,
- **traces** a vector of numeric corresponding to the eigen values of spatial weight matrix W,
- **n**, an integer, the size of the sample.

```
lndetmc(parms, traces, n)
```

```
source("./R/lndetmc.R")
```

In the case of a small matrix, we use the exact values of the traces of W, W^2, W^3, $\ldots$

```
lndetmc(0.25, traces, n)
```

```
## [1] -0.9269884
```

In the case of a larger matrix, one can use the approximation:

```
lndetmc(0.25, traces_approx, n)
```

```
## [1] -0.926855
```

Case of two spatial weight matrices

One can useformula (29) of Lesage and Pace (2008) to sum the log determinants of the two spatial weight matrices. For example, if $\rho_o = 0.4$ and $\rho_d = 0.2$, then the log determinant is equal to:

```
lndetmc(0.4, traces_approx, n) + lndetmc(0.2, traces_approx, n)
```

```
## [1] -3.102132
```

Evaluation of the log likelihood conditionnally to ρ

The function *c_sarf()* takes as arguments:

- **rho**, a vector containing the estimated values of ρ_d, ρ_d, ρ_w,
- **sige**, the value of σ^2

- **Q**, cross-product matrix of the various component residuals
- **traces** a vector of numeric corresponding to the eigenvalues of the spatial weight matrix W,
- **n**, an integer, the sample size.

```
c_sarf(rho, sige, Q, traces, n, nvars)
```

```
source("./R/c_sarf.R")
```

Function *sar_flow()* model

It takes as input arguments :

- **x**, a **data.frame** or a matrix with explanatory variables observed on the n geographical sites.
- **Y**, the matrix of flows of size $n \times n$,
- **G**, the matrix of distances of size $n \times n$,
- **w**, the spatial weight matrix of size $n \times n$,
- **ind_d**, the indices of the variables in **x** which will be used at the destination,
- **ind_o**, the indices of the variables in **x** of the variables used at the origin.

```
sar_flow(x, Y, G, w, ind_d = NULL, ind_o = NULL, model = "model_9")
```

Application to the toy data

Model 2

We evaluate model 2 when Y corresponds to the DGP used with model 2. We choose to include characteristics x_1 only at destination and x_2 only at origin.

```
system.time(sar_simu_2 <- sar_flow(x = x@data,
                         Y = Y_2,
                         G = G,
                         w = w,
                         ind_d = c(1, 2),
                         ind_o = c(1, 2),
                         model = "model_2")
)
```

```
## ===========================================================================

##    user  system elapsed
##   7.688   1.680   7.399
```

```
sar_simu_2
```

```
##                   mean    lower_05   lower_95    t_stat
## rho_d        0.3992659  0.32487805  0.4691650  8.907723
## (intercept) 39.2656906 34.76685716 44.0094687 13.710997
## x1_d         0.4805888  0.44696255  0.5158584 22.663798
## x2_d         0.6876071  0.02068526  1.3726127  1.669396
## x1_o         1.5026036  1.32504031  1.6902776 13.283114
## x2_o         2.1790225  1.44603527  2.9037203  4.818383
## g           -0.4807810 -0.58301389 -0.3830206 -7.958400
```

We compare the results with the *lagsarlm()* function and we remark that we obtain similar results :

```
result_lagsarlm<- lagsarlm(y_2 ~ x1_d + x2_d + x1_o + x2_o + g,
                           data = flows_data,
```

```r
                            mat2listw(W_d))
summary(result_lagsarlm, correlation = TRUE)
```

```
##
## Call:
## lagsarlm(formula = y_2 ~ x1_d + x2_d + x1_o + x2_o + g, data = flows_data,
##     listw = mat2listw(W_d))
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -1.700763 -0.483141 -0.088983  0.645908  2.134618
##
## Type: lag
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept)  0.352291   0.569137  0.6190   0.53592
## x1_d         0.476617   0.018379 25.9321 < 2.2e-16
## x2_d         0.685150   0.352562  1.9433   0.05197
## x1_o         1.477928   0.102202 14.4608 < 2.2e-16
## x2_o         2.134293   0.381139  5.5998 2.146e-08
## g           -0.473644   0.052124 -9.0869 < 2.2e-16
##
## Rho: 0.40924, LR test value: 60.229, p-value: 8.4377e-15
## Asymptotic standard error: 0.040874
##     z-value: 10.012, p-value: < 2.22e-16
## Wald statistic: 100.25, p-value: < 2.22e-16
##
## Log likelihood: -83.76652 for lag model
## ML residual variance (sigma squared): 0.73865, (sigma: 0.85945)
## Number of observations: 64
## Number of parameters estimated: 8
## AIC: 183.53, (AIC for lm: 241.76)
## LM test for residual autocorrelation
## test value: 0.19862, p-value: 0.65583
##
##  Correlation of coefficients
##             sigma rho   (Intercept) x1_d  x2_d  x1_o  x2_o
## rho         -0.10
## (Intercept)  0.06 -0.60
## x1_d         0.08 -0.82  0.27
## x2_d         0.00  0.03 -0.44        0.07
## x1_o         0.10 -0.99  0.55        0.82 -0.03
## x2_o         0.04 -0.38 -0.17        0.31 -0.01  0.39
## g           -0.06  0.57 -0.41       -0.59  0.00 -0.59 -0.23
```

However, we remark that we canno't use S2SLS method because of inversion problems.

```r
result_s2sls<- stsls(y_2 ~ x1_d + x2_d + x1_o + x2_o + g,
                            data = flows_data,
                            mat2listw(W_d),
                        robust = F,
                        legacy = T,
                        HC = "HC1")
summary(result_lagsarlm, correlation = TRUE)
```

I started to code the S2SLS for a general spatial model flow (model 9). The codes are in the *s2sls()* function.

Model 9

We evaluate model 9 when Y corresponds to the DGP used with model 9. We choose to include characteristics x_1 only at destination and x_2 only at origin.

```
system.time(sar_simu_9 <- sar_flow(x = x@data,
                         Y = Y_9,
                         G = G,
                         w = w,
                         ind_d = 1,
                         ind_o = 2,
                         model = "model_9")
)
sar_simu_9
```

Application to the real data

With the real data, we do not know which model should we use. We try here some possibilities.

Model 3

```
system.time(sar_true_data_3 <- sar_flow(x = wto_country[, "GDP"],
                         Y = Y_wto,
                         G = G_wto,
                         w = wto_W,
                         model = "model_3")
)
sar_true_data_3
```

Spatial Autoregressive Interaction Models when $N < n^2$

In that case, the number of flows is lower than $n \times n$. Users have to present the data in vectorized form.

Function *sar_flow_2()* model

It takes as input arguments :

- **x**, a **data.frame** or a matrix with explanatory variable observed on the N flows.
- **Y**, the vector of flows of size N,
- **g**, the vector of distances of size N,
- **W_d**, spatial weight matrix of size $N \times N$,
- **W_o**, spatial weight matrix of size $N \times N$,
- **W_w**, spatial weight matrix of size $N \times N$,
- **ind_d**, the indices of the variables in **x** which will be used at the destination,
- **ind_o**, the indices of the variables in **x** used at the origin.

```
sar_flow_2(x, y, g, W_d, W_o, W_w,
           ind_d = NULL, ind_o = NULL, model = "")
```

Application to the real data

Model 3

```
system.time(sar_mod_true_3 <- sar_flow_2(x = wto_filter[, c("DGP_d", "DGP_o")],
           y = wto_filter$trade,
           g = wto_filter$DIST,
           W_d = wto_W_o_filter) )
sar_mod_true_3
```

Interpreting the results

We compute the matrix $A(W) = (I_{N \times N} - \rho_o W_o - \rho_d W_d - \rho_w W_W)^{-1}$. We use the funtion *powerWeights()* which computes the power of matrix.

```
powerWeights(W, rho, order = 250, X,
   tol = .Machine$double.eps^(3/5))
```

Application on the Model 2 (simulated data)

```
W_hat <- sar_simu_2$mean[1] * W_d
AW <- powerWeights(W_hat, 1, order = 250, Diagonal(nrow(W_d)),
                   tol = .Machine$double.eps^(3/5))
```

Then, we compute the formula given in theorem 1. We first compute the left part of the equation:

```
res_o1 <- 0

for (k in 1:n) {
 res_o1 <- res_o1 + sum(AW[((1:n)[(!(1:n) %in% k)]) + n * (k - 1), (1:n) + n * (k - 1)])
}

(res_o1 <- sar_simu_2$mean[5:6] * res_o1 / N)
```

```
## [1] 2.188619 3.173858
```

We then compute the second part of the equation:

```
res_o2 <- 0

for(k in 1:n) {
 res_o2 <- res_o2 + sum(AW[((1:n)[(!(1:n) %in% k)]) + n * (k - 1),
                          seq(k, by = n, length.out = n)])
}
(res_o2 <- sar_simu_2$mean[3:4] * res_o2 / N)
```

```
## [1] 0.03477231 0.04975082
```

References

- LeSage J.P. and Pace R.K. (2008). Spatial econometric modeling of origin-destination flows. Journal of Regional Science, 48(5), 941—967.
- Pebesma E.J. and Bivand R.S. (2005). Classes and methods for spatial data in **R**, R News, 5(2), 9–13.
- Thomas-Agnan C. and LeSage J.P. (2014). Spatial Econometric OD-Flow Models. In: Fischer M., Nijkamp P. (eds) Handbook of Regional Science. Springer, Berlin, Heidelberg.