Spatial Flows Modelling with **R**

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

August 21, 2019

Contents

Packages needed:

```
install.packages("devtools")
install.packages(c("cartography", "Matrix", "rgdal",
                   "spdep", "tidyverse", "maptools", "spatialreg"))
```

```
require("cartography")# representation of spatial data
require("Matrix")     # sparse matrix
require("rgdal")      # import spatial data
require("spdep")      # spatial econometrics modelling
require("tidyverse")  # tidyverse data
require("maptools")   # spatial
```

Data preparation

LeSage and Pace (2008) present the spatial interaction model specification (eq. 20) for modelling origin destination flows :

$$(I_{N\times N} - \rho_d W_d y - \rho_o W_o y - \rho_w W_w y)Y = \alpha \iota_N + X_d\beta_d + X_o\beta_o + XL_d\delta_d + XL_o\delta_o + \gamma G + \epsilon$$

One example of application is the analysis of home to work commuting flows. In the spatial econometrics litterature, the origin and destination locations coincide. In our paper, we propose to extend this model to the case where the locations at origin and destinations do not coincide. This can happen in geomarketing applications where the locations of the origins are the customers and the locations at destinations are the spatial coordinates of the states.

Data storage

Let n_o be the number of geographical sites at the origin and n_d the number of geographical sites at destination. We denote by Y_{ij} the flow which represents a quantity moving from a geographical site i ($i = i_1, ..., i_{n_o}$) towards a geographical site j ($j = j_1...j_{n_d}$). Let $N = n_o n_d$. We also denote by G_{ij} the distance between site i and site j. Usually, the list of origins and destinations coincide which simplifies the notation because in that case $i_1 = j_1 = 1, \ldots, i_{n_o} = j_{n_d} = n$. In that particular case, we observe the same characteristics x at origin and destination. When the list of origins and destinations are not the same, this complicates the notation because the variables observed at origin and destination may be different. Thus, we should note x the variables observed at the origin and z the variables observed at destination.

Dependent and distance variable

To store the dependent variable and the distances, the user has two options:

- Flows and distances are stored into matrices of size $n_o \times n_d$,
- Flows and distances are stored into vectors of size N.

Explanatory variables

To store the explanatory variables, the user also has two options:

- Explanatory variables x observed at origin (respectively z at destination) are stored into a **data.frame** of size $n_o \times R_o$ (resp. $n_d \times R_d$) where R_o (resp. R_d) are the respective numbers of explanatory variables.
- Explanatory variables observed at origin and destination are stored into a **data.frame** of size $n_o n_d \times (R_o + R_d)$. To obtain this form, the user has to use a kronecker product applied to x and z.

Spatial weight matrices

We denote by OW (resp. DW) the spatial weight matrix of size $n_o \times n_o$ (resp. $n_d \times n_d$) which determine if two locations at origin (resp. destination) are neighbours.

To build spatial matrices W_o, W_d or W_w of size $N \times N$ the user can use the properties of kronecker products and the spatial weight matrices OW and DW. He can also use the properties of sparse matrices.

Sparse simulated data example

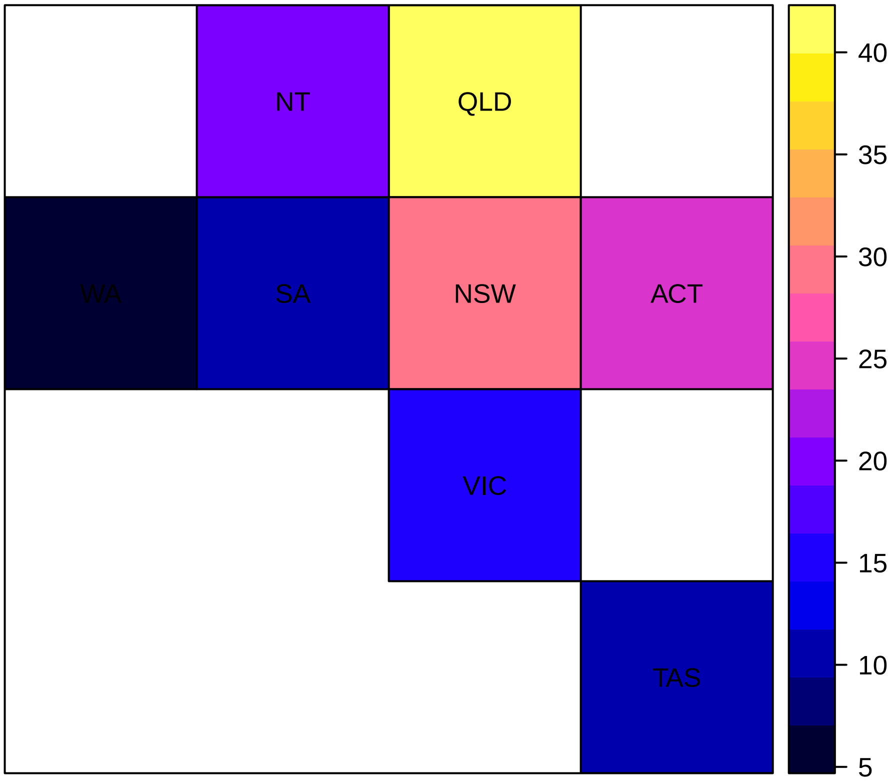
We will consider two cases: the case where the list of origins and destinations coincide and the case where it does not coincide.

List of origins and destinations coincide

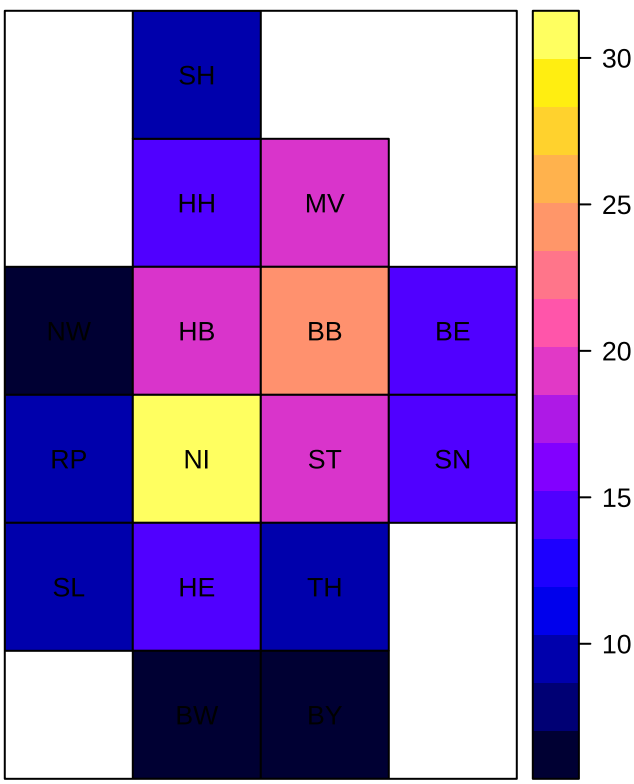
We consider three examples with different numbers of observations. We use examples from https://ialab.it.monash.edu/~dwyer/papers/maptrix.pdf. We define the polygons by using the function *create_grid()* inspired by the example given by R. Bivand in https://stat.ethz.ch/pipermail/r-sig-geo/2009-December/007163.html. We consider one explanatory variable for each data set. The programs to obtain these simulated examples are in "simulated_examples.R. For Australia, we also used real data obtained from the Australian Bureau of Statistics (https://itt.abs.gov.au/itt/r.jsp?databyregion).

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

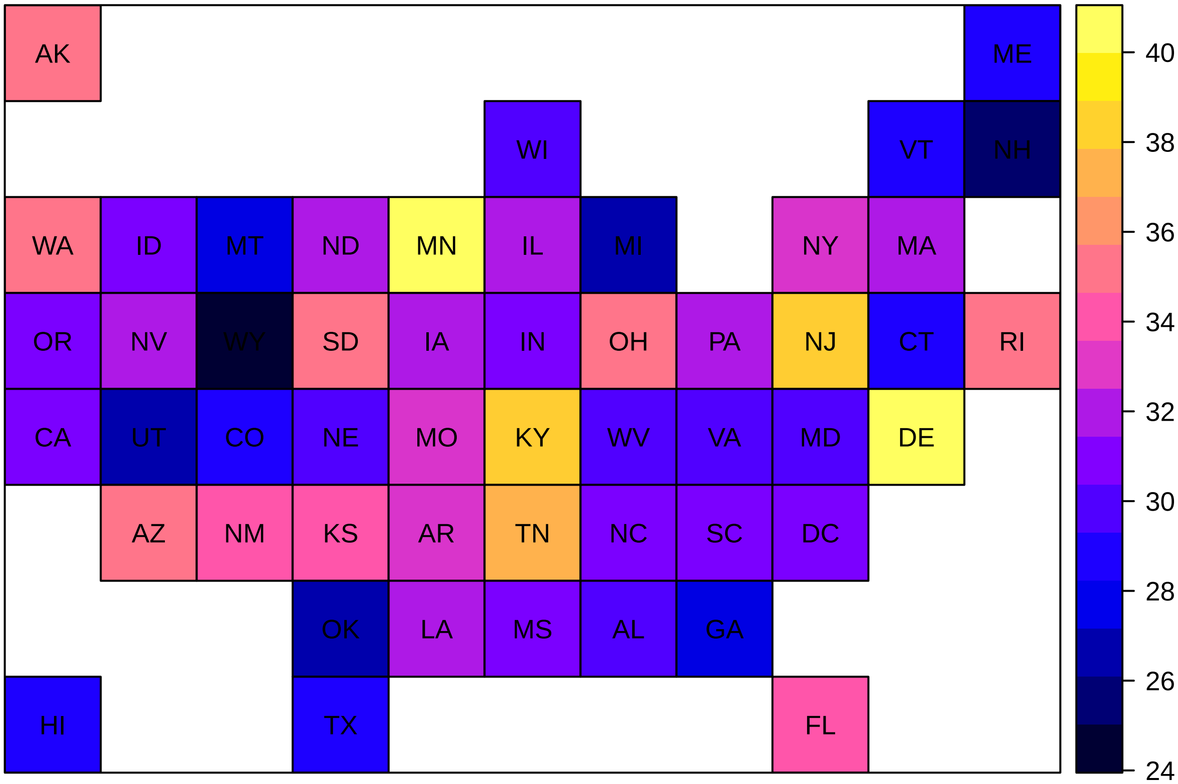
- The first example is a simplification of the 8 main regions of Australia.



- The second example is a simplification of the 16 main regions of Germany.



- The third example is a simplification of the 51 main regions of the USA:



Storing the variables into origin-destination format

If user wants to store the explanatory variables in a matrix of size $N \times R$, he has to use the kronecker product. We present the results for the Australian regions. First, we create the **origin** and **dest** variables:

```
n_au <- nrow(spdf_au)
flows_au <- data.frame(origin = rep(row.names(spdf_au),
                                    each = n_au),
                       dest = rep(row.names(spdf_au), n_au))
```

We create the destination data:

```
flows_au_d <- sapply(spdf_au@data[, c("x", "wage", "pop", "age",
                                      "ln_wage", "ln_pop", "ln_age")],
                     function (x) kronecker(rep(1, n_au), x))
colnames(flows_au_d) <- paste0(colnames(flows_au_d), "_d")
```

We create the origin data:

```
flows_au_o <- sapply(spdf_au@data[, c("x", "wage", "pop", "age",
                                      "ln_wage", "ln_pop", "ln_age")],
                     function (x) kronecker(x, rep(1, n_au)))
colnames(flows_au_o) <- paste0(colnames(flows_au_o), "_o")
```

We merge the data:

```
flows_au <- cbind(flows_au, flows_au_d, flows_au_o)
head(flows_au)
```

```
##   origin dest x_d wage_d   pop_d age_d ln_wage_d ln_pop_d ln_age_d x_o
## 1     NT   NT  20  56783  247327  32.6  10.94699 12.41847 3.484312  20
```

```
## 2     NT  QLD  40  47177 5011216  37.1  10.76166 15.42719 3.613617  20
## 3     NT   WA   7  52691 2595192  36.6  10.87220 14.76917 3.600048  20
## 4     NT   SA  10  46619 1736422  40.0  10.74976 14.36734 3.688879  20
## 5     NT  NSW  30  49256 4988241  37.5  10.80479 15.42259 3.624341  20
## 6     NT  ACT  25  64901  420960  35.0  11.08062 12.95029 3.555348  20
##   wage_o  pop_o age_o ln_wage_o ln_pop_o ln_age_o
## 1  56783 247327  32.6  10.94699 12.41847 3.484312
## 2  56783 247327  32.6  10.94699 12.41847 3.484312
## 3  56783 247327  32.6  10.94699 12.41847 3.484312
## 4  56783 247327  32.6  10.94699 12.41847 3.484312
## 5  56783 247327  32.6  10.94699 12.41847 3.484312
## 6  56783 247327  32.6  10.94699 12.41847 3.484312
```

If user wants to store the dependant variable from a matrix of size $n \times n$ to a vector of size N, he has to use the function *as.vector()*. He has to take care of the ordering of the observations. For example in the simple case of 3 origins and 2 destinations where the columns of the matrix Y represent the origins and the rows correspond to the destinations, to obtain the vectorized form where the first n_d elements of y represent flows from origin 1 to all n_d destinations:

```
Y <- matrix(1:6, nrow = 2, ncol = 3, byrow = T,
            dimnames = list(paste0("D", 1:2), paste0("O", 1:3)))
y <- as.vector(Y)
```

If the columns of the matrix Y represent the destinations and the rows correspond to the origin, to obtain the vectorized form, user has to transpose the matrix before using the function *as.vector()*.

```
au_flows <- au_flows[id_region_au, id_region_au]
flows_au$real_Y <- as.vector(au_flows)
```

Same has been done for Germany and USA.

Distances between locations

The distances between origins and destinations can be stored in a matrix of size $n \times n$.

```
G_au <- as.matrix(log(1 + dist(coordinates(spdf_au))))
```

It can also be added to the **data.frame** which presents the data in vectorized form.

```
flows_au$g <- as.vector(G_au)
```

Same has been done for Germany, USA and the simulated grid.

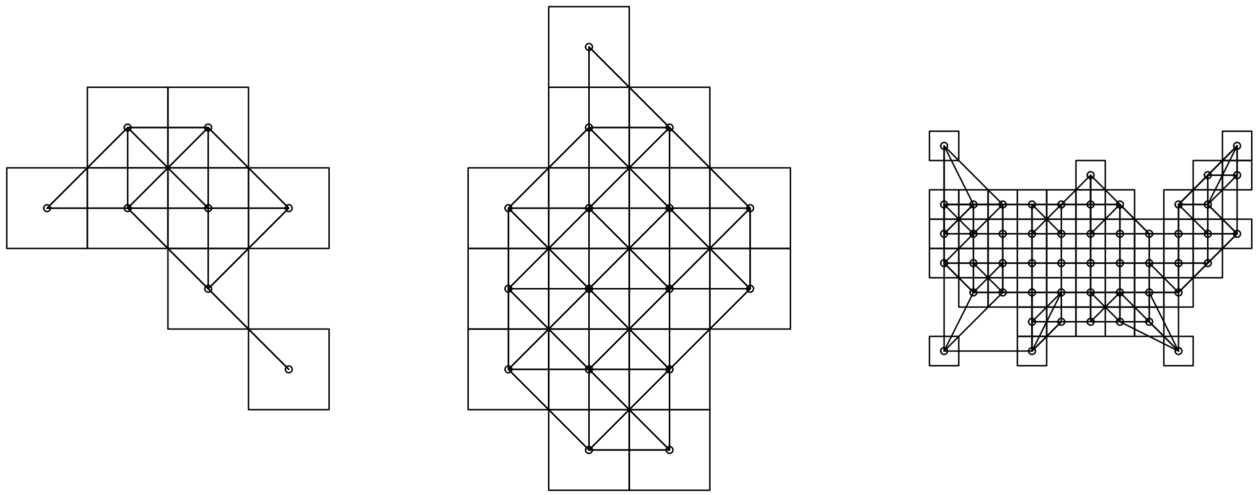
Construct the spatial weight matrices

To define the spatial weight matrices for our geographical sites, we use the contiguity properties for Australia and Germany. Because some states in USA have no neighbours when using this method, we use the 4 nearest neighbours method for USA. All these methods have been implemented in package **sp** (Bivand et al., 2013).

```
w_au_nb <- poly2nb(spdf_au)
w_au <- listw2mat(nb2listw(w_au_nb))
w_ge_nb <- poly2nb(spdf_ge)
w_ge <- listw2mat(nb2listw(w_ge_nb))
w_usa_nb <- knn2nb(knearneigh(coordinates(spdf_usa), k = 4))
w_usa <- listw2mat(nb2listw(w_usa_nb))
```

We represent the spatial links between the observations:

```r
#pdf("figures/spdf_neighbors.pdf", width = 10, height = 5)
par(mfrow = c(1, 3))
plot(spdf_au)
plot(w_au_nb, coordinates(spdf_au), add = T)
plot(spdf_ge)
plot(w_ge_nb, coordinates(spdf_ge), add = T)
plot(spdf_usa)
plot(w_usa_nb, coordinates(spdf_usa), add = T)
```



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

To build the spatial weight matrices W_o, W_d and W_w, the user has to use Kronecker products. It can be interesting to use the properties of sparse matrices as well to avoid to store too much data. For example we compare the memory needed to store W_o with or without using the sparse properties:

```r
object.size(kronecker(diag(n_au), w_au))
```

```
## 32984 bytes
```

```r
object.size(kronecker(Diagonal(n_au), w_au))
```

```
## 5080 bytes
```

```r
W_au_d <- kronecker(Diagonal(n_au), w_au)
W_au_o <- kronecker(w_au, Diagonal(n_au))
W_au_w <- kronecker(as(w_au, "Matrix"), w_au)
```

We prepare the lagged explanatory variables:

```r
flows_au$lagged_x_d <- as.matrix(W_au_d) %*% flows_au$x_d
flows_au[, c("lagged_wage_d", "lagged_pop_d", "lagged_age_d",
             "lagged_ln_wage_d", "lagged_ln_pop_d", "lagged_ln_age_d")] <-
  as.matrix(W_au_d) %*%
  as.matrix(flows_au[, c("wage_d", "pop_d", "age_d",
                              "ln_wage_d", "ln_pop_d", "ln_age_d")])
flows_au$lagged_x_o <- as.matrix(W_au_o) %*% flows_au$x_o
flows_au[, c("lagged_wage_o", "lagged_pop_o", "lagged_age_o",
             "lagged_ln_wage_o", "lagged_ln_pop_o", "lagged_ln_age_o")] <-
  as.matrix(W_au_o) %*%
  as.matrix(flows_au[, c("wage_o", "pop_o", "age_o",
                              "ln_wage_o", "ln_pop_o", "ln_age_o")])
```

Same has been done for Germany and USA.

List of origins and destinations do not coincide

We consider the migration flows from countries of Africa towards countries in Europe. The data are extracted from https://journals.sagepub.com/doi/suppl/10.1177/0022002718823907

To load the data:

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

```
## Loading required package: haven

## Reading layer `Pays_WGS84' from data source `/home/laurent/Documents/paula/Paula Flows/data/World WGS
## Simple feature collection with 251 features and 1 field
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -180 ymin: -89.9 xmax: 180 ymax: 83.6236
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

Origins and destinations spatial objects are stored in the **sf** norm (Pebezma, 2018). To plot the data:

```
# pdf("figures/grid.pdf", width = 7, height = 8)
plot(st_geometry(europe), xlim = c(-40, 60), ylim = c(-40, 70))
plot(st_geometry(africa), add = T)
```

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

Origin data contains 51 observations and 3 variables:

```r
head(dplyr::select(as.data.frame(africa), -geometry))
```

```
##            NOM iso3_o   popul_o        gdp_o civilconflict_o
## 1      Algeria    DZA 38934.336 214000000000               1
## 2       Angola    AGO 24227.523 127000000000               0
## 3        Benin    BEN 10598.482   9707432016               0
## 4     Botswana    BWA  2219.937  15880203581               0
## 5      Burindi    BDI 10816.860   3093647227               0
## 6 Burkina Faso    BFA 17589.197  12257141798               0
```

Destination data contains 21 observations and 2 variables:

```r
head(dplyr::select(as.data.frame(europe), -geometry))
```

```
##              NOM iso3_d   popul_d    gdp_d
## 1        Austria    AUT  8516.916 4.38e+11
## 2        Belgium    BEL 11226.322 5.32e+11
## 3 Czech Republic    CZE 10542.666 2.08e+11
## 4        Denmark    DNK  5646.899 3.46e+11
## 5        Finland    FIN  5479.660 2.72e+11
## 6         France    FRA 64121.250 2.84e+12
```

The dependent variable is tored in the matrix **Y_mig** of size 21×51.

Transform the explanatory variables to origin-destination format

To store the explanatory variables in a matrix of size $N \times R$, the user has to use Kronecker products separetly for origins and for destinations:

```r
n_mig_o <- nrow(africa)
n_mig_d <- nrow(europe)
x_mig_origin <- kronecker(as.matrix(africa %>%
                  dplyr::select(popul_o, gdp_o, civilconflict_o)  %>%
                  st_drop_geometry()), rep(1, n_mig_d))
colnames(x_mig_origin) <- c("popul_o", "gdp_o", "civilconflict_o")
x_mig_dest <- kronecker(rep(1, n_mig_o), as.matrix(europe %>%
                  dplyr::select(popul_d, gdp_d)  %>%
                  st_drop_geometry()))
colnames(x_mig_dest) <- c("popul_d", "gdp_d")
flows_mig <- data.frame(origin = rep(africa$NOM,
                                     each = n_mig_d),
                         dest = rep(europe$NOM, n_mig_o),
                         x_mig_origin,
                        x_mig_dest)
```

Distances between locations

The distances between flows can be presented in a matrix of size $n_d \times n_o$. We use the function *st_distance()* from **sf** package (Pebesma, 2018). We could have also used the function *gDistance()* from package **rgeos** (Bivand and Rundel, 2019).

```r
G_mig <- t(as(st_distance(st_centroid(africa),
                      st_centroid(europe)), "matrix"))
```

It can be also added to the data.frame which presents the data in vectorized form

```r
flows_mig$g <- as.vector(G_mig)
```

We also add the dependent variable

```r
flows_mig$y <- as.vector(Y_mig)
```

Finally, the vectyorized form looks like:

```r
head(flows_mig)
```

```
##    origin           dest  popul_o    gdp_o civilconflict_o   popul_d
## 1 Algeria        Austria 38934.34 2.14e+11               1  8516.916
## 2 Algeria        Belgium 38934.34 2.14e+11               1 11226.322
## 3 Algeria Czech Republic 38934.34 2.14e+11               1 10542.666
## 4 Algeria        Denmark 38934.34 2.14e+11               1  5646.899
## 5 Algeria        Finland 38934.34 2.14e+11               1  5479.660
## 6 Algeria         France 38934.34 2.14e+11               1 64121.250
##      gdp_d       g     y
## 1 4.38e+11 2376138   307
## 2 5.32e+11 2502128   740
## 3 2.08e+11 2628304    46
## 4 3.46e+11 3146837    17
## 5 2.72e+11 4373682    44
## 6 2.84e+12 2042010 24108
```

Construct the spatial weight matrices

To define the spatial weight matrix on our geographical sites, we use the 4-nearest neighbours properties for origins and destinations.

```r
w_mig_o_nb <- knn2nb(knearneigh(as(st_centroid(africa), "Spatial"), k = 4))
w_mig_o <- listw2mat(nb2listw(w_mig_o_nb))
w_mig_d_nb <- knn2nb(knearneigh(as(st_centroid(europe), "Spatial"), k = 4))
w_mig_d <- listw2mat(nb2listw(w_mig_d_nb))
```

To build the spatial weight matrices W_o, W_d and W_w, the user has to use Kronecker products.

```r
W_mig_d <- kronecker(Diagonal(n_mig_o), w_mig_d)
W_mig_o <- kronecker(w_mig_o, Diagonal(n_mig_d))
W_mig_w <- W_mig_o %*% W_mig_d
```

We prepare the lagged explanatory variables:

```r
flows_mig[, c("lagged_ln_popul_d", "lagged_ln_gdp_d")] <-
  as.matrix(W_mig_d) %*% as.matrix(log(1 + flows_mig[, c("popul_d", "gdp_d")]))
flows_mig[, c("lagged_ln_popul_o", "lagged_ln_gdp_o", "lagged_civilconflict_o")] <-
  as.matrix(W_mig_o) %*% as.matrix(cbind(log(1 + flows_mig[, c("popul_o", "gdp_o")]),
                                         flows_mig$civilconflict_o))
```

DGP of the Y variables

Intraregional modelling

Laurent, Margaretic and Thomas-Agnan (2019) proposes the spatial interaction model specification for modelling origin destination flows which take into account the intraregionnal variation in flows :

$$(I_{N\times N}-\rho_d W_d y-\rho_o W_o y-\rho_w W_w y)Y=\iota_N\alpha+vec(I_n)\alpha_i+X_d\beta_d+X_o\beta_o+X_i\beta_i+XL_d\delta_d+XL_o\delta_o+XL_i\delta_i+\gamma G+\epsilon$$

The previously described procedure aims at allowing the coefficients associated with the matrices X_d, X_o to more accurately reflect the variation in interregional flows, and those associated with the matrix X_i to capture intraregional variation in flows.

Same has been done for Germany and USA.

```
flows_au_od <- flows_au
flows_au$vec_In <- as.vector(diag(n_au))

for (var in c("x", "wage", "pop", "age",
              "lagged_x", "lagged_wage", "lagged_pop", "lagged_age",
              "ln_wage", "ln_pop", "ln_age",
              "lagged_ln_wage", "lagged_ln_pop", "lagged_ln_age")) {
  flows_au[, paste0(var, "_i")] <- flows_au[, paste0(var, "_o")]
  flows_au[flows_au$origin != flows_au$dest, paste0(var, "_i")] <- 0
  flows_au[flows_au$origin == flows_au$dest, paste0(var, "_o")] <- 0
  flows_au[flows_au$origin == flows_au$dest, paste0(var, "_d")] <- 0
}
```

We simulate 9 different SDM 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 $Z = (1_N, vec(I_n), X_d, X_o, X_i, XL_d, XL_o, XL_i, G)$, $\delta = (\alpha, \beta_d, \beta_o, \beta_i, \delta_d, \delta_o, \delta_i, \gamma)$. We generate a set of flows Y with $\alpha = 1$, $\alpha_i = 0.5$, $\beta_d = 1$, $\beta_o = 0.5$, $\beta_i = 2$, $\delta_d = 0.25$, $\delta_o = 0.1$, $\delta_i = 0.5$, $\gamma = -2.0$, $\rho_d = 0.4$, $\rho_o = 0.4$, and $\rho_w = -0.16$.

```
delta <- c(2, 1.5, 1, 0.5, 2.5, 0.25, 0.1, 0.5, -2)
rho <- c(0.4, 0.4, -0.16)
```

We consider one simulation for the model for the case "intra":

```
Z_au <- cbind(1, flows_au$vec_In,
              flows_au$x_d, flows_au$x_o, flows_au$x_i,
              flows_au$lagged_x_d, flows_au$lagged_x_o, flows_au$lagged_x_i,
              flows_au$g)
```

We consider another simulation for the model for the case without "intra". Because the origin and destination are different in the intra case, we have to consider antother matrix:

```
Z_au_od <- cbind(1, flows_au_od$x_d, flows_au_od$x_o,
              flows_au_od$lagged_x_d, flows_au_od$lagged_x_o,
              flows_au_od$g)
```

Flows can be presented in vectorized format. For this, we use the function *DGP_flow_sdm()* which allows to simulate flows data.

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

The function *DGP_flow_sdm()* takes as arguments:

```
DGP_flow_sdm(z, delta, rho, W_d, W_o, W_w,
             seed = NULL, sigma = 1, message = F)
```

- **z**, the matrix containing the explanatory variables,
- **delta**, the vector of parameters,
- **rho**, the vector with (ρ_d, ρ_o, ρ_w)
- **W_o**, **W_d**, **W_w** the spatial weight matrices of size $N \times N$
- **seed**, a vector of integer values for the seed. NULL by default
- **sigma**, the variance of the residuals
- **message**, print a message to indicate the ratio of sd(noise)/sd(signal).
- **model**, a character.

```
flows_au[, "y_9"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho,
               W_d = as.matrix(W_au_d),
               W_o = as.matrix(W_au_o),
               W_w = as.matrix(W_au_w),
               seed = 123, sigma = 1, message = T,
               model = "model_9")
```

```
## sd(noise)/sd(signal):
## 0.03565514
```

```
flows_au[, "y_8"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1:2],
               W_d = as.matrix(W_au_d),
               W_o = as.matrix(W_au_o),
               W_w = as.matrix(W_au_w),
               seed = 123, sigma = 1, message = T,
               model = "model_8")
```

```
## sd(noise)/sd(signal):
## 0.03412139
```

```
flows_au[, "y_7"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1:2],
               W_d = as.matrix(W_au_d),
               W_o = as.matrix(W_au_o),
               seed = 123, sigma = 1, message = T,
               model = "model_7")
```

```
## sd(noise)/sd(signal):
## 0.03319209
```

```
flows_au[, "y_6"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1],
               W_d = as.matrix(W_au_d),
               W_o = as.matrix(W_au_o),
               W_w = as.matrix(W_au_w),
               seed = 123, sigma = 1, message = T,
               model = "model_6")
```

```
## sd(noise)/sd(signal):
## 0.04368685
```

```r
flows_au[, "y_5"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1],
               W_d = as.matrix(W_au_d),
               W_o = as.matrix(W_au_o),
               seed = 123, sigma = 1, message = T,
               model = "model_5")
```

```
## sd(noise)/sd(signal):
## 0.04241537
```

```r
flows_au[, "y_4"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1],
               W_w = as.matrix(W_au_w),
               seed = 123, sigma = 1, message = T,
               model = "model_4")
```

```
## sd(noise)/sd(signal):
## 0.04689977
```

```r
flows_au[, "y_3"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1],
               W_o = as.matrix(W_au_o),
               seed = 123, sigma = 1, message = T,
               model = "model_3")
```

```
## sd(noise)/sd(signal):
## 0.03995532
```

```r
flows_au[, "y_2"] <- DGP_flow_sdm(z = Z_au, delta = delta,  rho = rho[1],
               W_d = as.matrix(W_au_d),
               seed = 123, sigma = 1, message = T,
               model = "model_2")
```

```
## sd(noise)/sd(signal):
## 0.04588884
```

```r
flows_au[, "y_1"] <- DGP_flow_sdm(z = Z_au, delta = delta,
               seed = 123, sigma = 1, message = T,
               model = "model_1")
```

```
## sd(noise)/sd(signal):
## 0.05136486
```

```r
flows_au_od[, "y_1"] <- DGP_flow_sdm(z = Z_au_od, delta = delta[-c(2, 5, 8)],
               seed = 123, sigma = 1, message = T,
               model = "model_1")
```

```
## sd(noise)/sd(signal):
## 0.06965243
```

The data set corresponding to the vectorized flows is presented in that form :

```r
head(flows_au)
```

```
##   origin dest x_d wage_d   pop_d age_d ln_wage_d ln_pop_d ln_age_d x_o
## 1     NT   NT   0      0       0   0.0   0.00000  0.00000 0.000000   0
## 2     NT  QLD  40  47177 5011216  37.1  10.76166 15.42719 3.613617  20
## 3     NT   WA   7  52691 2595192  36.6  10.87220 14.76917 3.600048  20
## 4     NT   SA  10  46619 1736422  40.0  10.74976 14.36734 3.688879  20
## 5     NT  NSW  30  49256 4988241  37.5  10.80479 15.42259 3.624341  20
## 6     NT  ACT  25  64901  420960  35.0  11.08062 12.95029 3.555348  20
##   wage_o  pop_o age_o ln_wage_o ln_pop_o ln_age_o real_Y         g
```

```
## 1      0      0   0.0   0.00000  0.00000 0.000000  23045 0.0000000
## 2  56783 247327  32.6  10.94699 12.41847 3.484312   5253 0.6931472
## 3  56783 247327  32.6  10.94699 12.41847 3.484312   2469 0.8813736
## 4  56783 247327  32.6  10.94699 12.41847 3.484312   2664 0.6931472
## 5  56783 247327  32.6  10.94699 12.41847 3.484312   2832 0.8813736
## 6  56783 247327  32.6  10.94699 12.41847 3.484312    478 1.1743590
##   lagged_x_d lagged_wage_d lagged_pop_d lagged_age_d lagged_ln_wage_d
## 1    0.00000          0.00          0.0      0.00000          0.00000
## 2   21.25000      54389.75    1848237.5     36.27500         10.89554
## 3   15.00000      51701.00     991874.5     36.30000         10.84838
## 4   22.40000      50957.00    3561065.0     35.88000         10.83654
## 5   22.00000      52871.60    2475854.8     36.06000         10.86722
## 6   28.33333      48437.00    4987602.0     36.73333         10.78784
##   lagged_ln_pop_d lagged_ln_age_d lagged_x_o lagged_wage_o lagged_pop_o
## 1         0.00000        0.000000       0.00          0.00            0
## 2        13.78967        3.588220      21.75      48935.75      3582768
## 3        13.39290        3.586596      21.75      48935.75      3582768
## 4        14.69100        3.578933      21.75      48935.75      3582768
## 5        14.11618        3.582900      21.75      48935.75      3582768
## 6        15.42246        3.603435      21.75      48935.75      3582768
##   lagged_age_o lagged_ln_wage_o lagged_ln_pop_o lagged_ln_age_o vec_In x_i
## 1          0.0           0.0000         0.00000        0.000000      1  20
## 2         37.8          10.7971        14.99657        3.631721      0   0
## 3         37.8          10.7971        14.99657        3.631721      0   0
## 4         37.8          10.7971        14.99657        3.631721      0   0
## 5         37.8          10.7971        14.99657        3.631721      0   0
## 6         37.8          10.7971        14.99657        3.631721      0   0
##   wage_i  pop_i age_i lagged_x_i lagged_wage_i lagged_pop_i lagged_age_i
## 1  56783 247327  32.6      21.75      48935.75      3582768         37.8
## 2      0      0   0.0       0.00          0.00            0          0.0
## 3      0      0   0.0       0.00          0.00            0          0.0
## 4      0      0   0.0       0.00          0.00            0          0.0
## 5      0      0   0.0       0.00          0.00            0          0.0
## 6      0      0   0.0       0.00          0.00            0          0.0
##   ln_wage_i ln_pop_i ln_age_i lagged_ln_wage_i lagged_ln_pop_i
## 1  10.94699 12.41847 3.484312          10.7971        14.99657
## 2   0.00000  0.00000 0.000000           0.0000         0.00000
## 3   0.00000  0.00000 0.000000           0.0000         0.00000
## 4   0.00000  0.00000 0.000000           0.0000         0.00000
## 5   0.00000  0.00000 0.000000           0.0000         0.00000
## 6   0.00000  0.00000 0.000000           0.0000         0.00000
##   lagged_ln_age_i       y_9      y_8      y_7      y_6      y_5      y_4
## 1        3.631721 138.78343 183.1790 236.4074 92.27365 91.52001 94.21432
## 2        0.000000 152.59191 197.6044 251.3662 91.45915 93.77262 87.24268
## 3        0.000000  90.13894 132.7049 184.6835 50.14653 49.41979 50.65792
## 4        0.000000  99.58167 143.7356 196.7668 56.30037 55.29903 57.99046
## 5        0.000000 134.76710 179.4888 233.0052 79.62194 80.83499 77.28274
## 6        0.000000 128.10481 174.4394 229.0117 77.21580 76.65724 78.49089
##         y_3      y_2      y_1
## 1  91.07546 91.67665 63.81452
## 2 102.46821 88.16239 57.87103
## 3  42.46592 54.66826 24.72096
## 4  50.41088 58.05983 28.45921
## 5  85.37387 77.73074 48.04154
```

```
## 6  76.67647 75.44639 45.62468
```

Flows can be also be presented in matrix format.

```
Y_au_9 <- matrix(flows_au$y_9, n_au, n_au)
Y_au_8 <- matrix(flows_au$y_8, n_au, n_au)
Y_au_7 <- matrix(flows_au$y_7, n_au, n_au)
Y_au_6 <- matrix(flows_au$y_6, n_au, n_au)
Y_au_5 <- matrix(flows_au$y_5, n_au, n_au)
Y_au_4 <- matrix(flows_au$y_4, n_au, n_au)
Y_au_3 <- matrix(flows_au$y_3, n_au, n_au)
Y_au_2 <- matrix(flows_au$y_2, n_au, n_au)
Y_au_1 <- matrix(flows_au$y_1, n_au, n_au)
Y_au_od_1 <- matrix(flows_au_od$y_1, n_au, n_au)
```

Same has been done for Germany, USA and the grid examples.

```
Z_usa <- cbind(1, flows_ge$vec_In,
               flows_usa$x_d, flows_usa$x_o, flows_usa$x_i,
              flows_usa$lagged_x_d, flows_usa$lagged_x_o, flows_usa$lagged_x_i,
              flows_usa$g)
```

```
## Warning in cbind(1, flows_ge$vec_In, flows_usa$x_d, flows_usa$x_o,
## flows_usa$x_i, : number of rows of result is not a multiple of vector
## length (arg 2)
```

```
Z_usa_od <- cbind(1, flows_usa_od$x_d, flows_usa_od$x_o, flows_usa_od$x_i,
              flows_usa_od$lagged_x_d, flows_usa_od$lagged_x_o, flows_usa_od$lagged_x_i,
              flows_usa_od$g)


flows_usa[, "y_9"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho,
               W_d = as.matrix(W_usa_d),
               W_o = as.matrix(W_usa_o),
               W_w = as.matrix(W_usa_w),
               seed = 123, sigma = 1, message = F,
               model = "model_9")

flows_usa[, "y_8"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1:2],
               W_d = as.matrix(W_usa_d),
               W_o = as.matrix(W_usa_o),
               W_w = as.matrix(W_usa_w),
               seed = 123, sigma = 1, message = F,
               model = "model_8")

flows_usa[, "y_7"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1:2],
               W_d = as.matrix(W_usa_d),
               W_o = as.matrix(W_usa_o),
               seed = 123, sigma = 1, message = F,
               model = "model_7")

flows_usa[, "y_6"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1],
               W_d = as.matrix(W_usa_d),
               W_o = as.matrix(W_usa_o),
               W_w = as.matrix(W_usa_w),
               seed = 123, sigma = 1, message = F,
```

```r
                model = "model_6")

flows_usa[, "y_5"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1],
                W_d = as.matrix(W_usa_d),
                W_o = as.matrix(W_usa_o),
                seed = 123, sigma = 1, message = F,
                model = "model_5")

flows_usa[, "y_4"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1],
                W_w = as.matrix(W_usa_w),
                seed = 123, sigma = 1, message = F,
                model = "model_4")

flows_usa[, "y_3"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1],
                W_o = as.matrix(W_usa_o),
                seed = 123, sigma = 1, message = F,
                model = "model_3")

flows_usa[, "y_2"] <- DGP_flow_sdm(z = Z_usa, delta = delta,  rho = rho[1],
                W_d = as.matrix(W_usa_d),
                seed = 123, sigma = 1, message = F,
                model = "model_2")

flows_usa[, "y_1"] <- DGP_flow_sdm(z = Z_usa, delta = delta,
                seed = 123, sigma = 1, message = F,
                model = "model_1")

flows_usa_od[, "y_1"] <- DGP_flow_sdm(z = Z_usa_od, delta = delta[-c(2, 5, 8)],
                seed = 123, sigma = 1, message = F,
                model = "model_1")
```

Data Vizualization

List of origins and destinations coincide

```r
mtq_mob <- getLinkLayer(
  x = spdf_au,
  xid = "NOM",
  df = flows_au,
  dfid = c("origin","dest")
)
```

We discretize the variable *real_Y*:

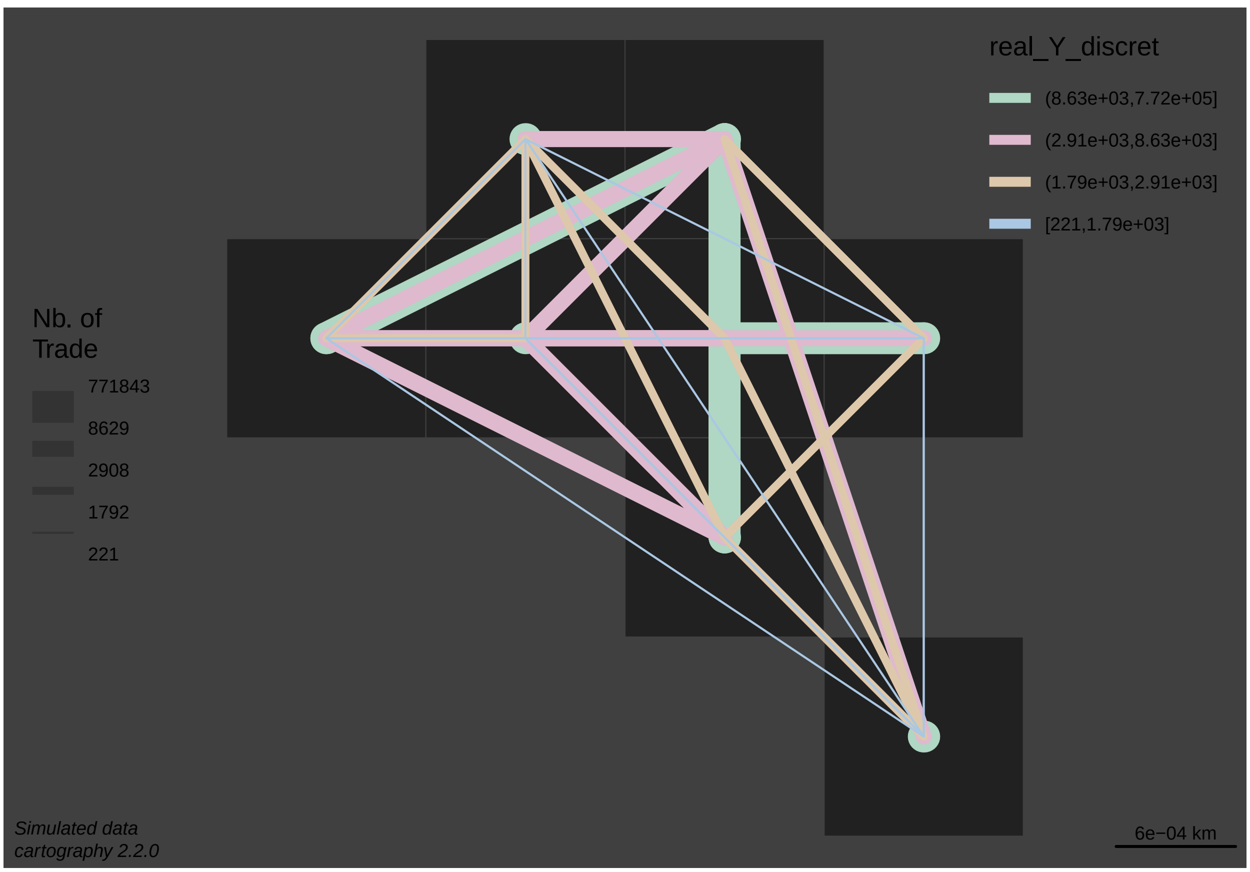
```r
breaks <- quantile(flows_au$real_Y)
flows_au$real_Y_discret <- cut(flows_au$real_Y, breaks,
                            include.lowest = T)
```

We plot all the flows.

```r
# pdf("figures/flows.pdf", width = 6, height = 6)
par(bg = "grey25", oma = c(0, 0, 0, 0),
    mar = c(0, 0, 0, 0), mai = c(0, 0, 0, 0))
```

```r
# plot municipalities
plot(spdf_au, col = "grey13", border = "grey25",
     bg = "grey25", lwd = 0.5)
# plot graduated links
gradLinkTypoLayer(
  x = mtq_mob,
  xid = c("origin","dest"),
  df = flows_au,
  dfid = c("origin","dest"),
  var = "real_Y",
  breaks = breaks,
  lwd = c(1, 4, 8, 16),
  var2 = "real_Y_discret",
  legend.var.pos = "left",
  legend.var.title.txt = "Nb. of\nTrade",
)

# map layout
layoutLayer(title = "Trades",
            sources = "Simulated data",
            author = paste0("cartography ", packageVersion("cartography")),
            frame = FALSE, col = "grey25", coltitle = "white",
            tabtitle = TRUE)
```



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

List of origins and destinations do not coincide

We first have to create a Spatial object containing both origin and destinations spatial units.

```r
sf_mig_o_d <- rbind(africa[, "NOM"],
                    europe[, "NOM"])
```

```r
mtq_mob <- getLinkLayer(
  x = sf_mig_o_d,
  xid = "NOM",
  df = flows_mig,
  dfid = c("origin","dest")
)
```

```
## 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
```

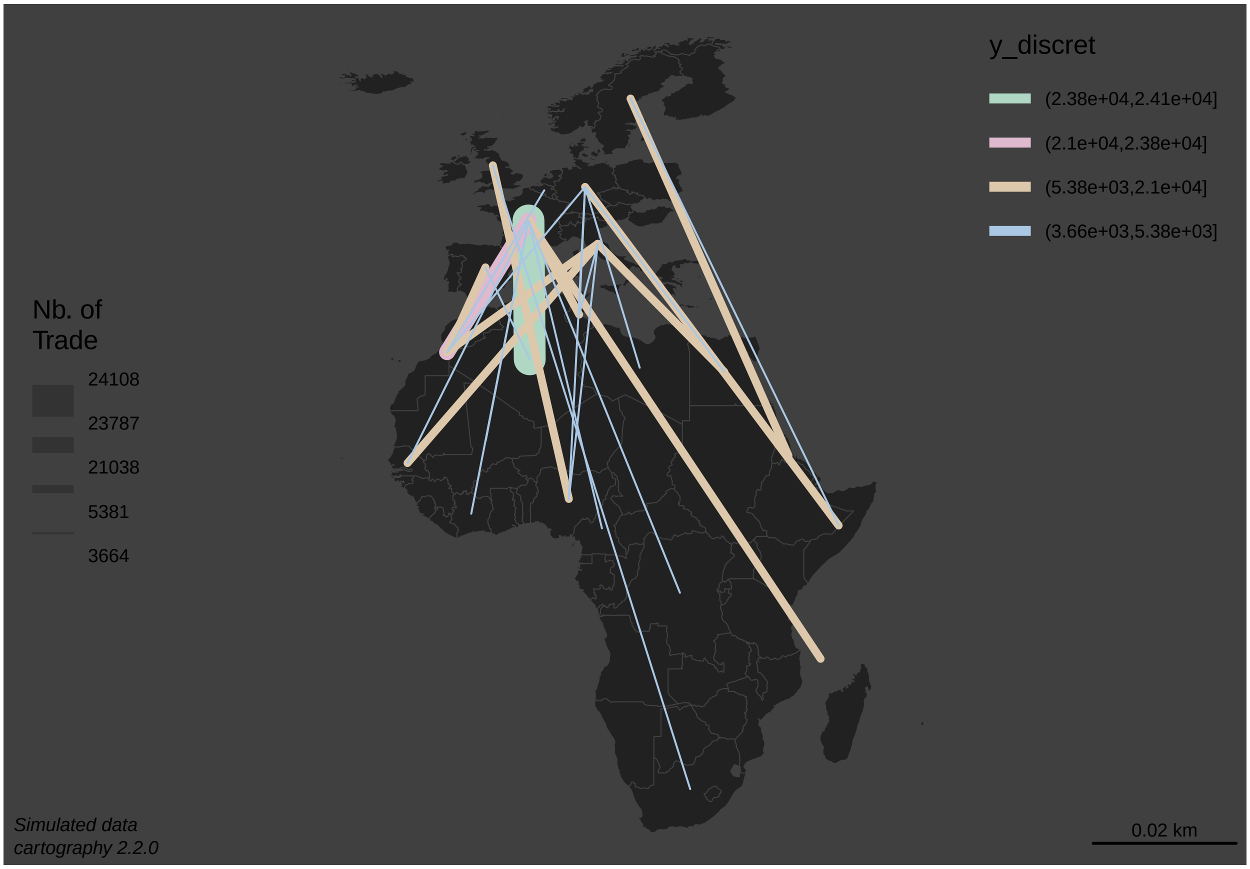
We discretize the variable y_9:

```r
breaks <- quantile(flows_mig$y,
                   c(0, 0.975, 0.99, 0.999, 0.9999, 1))
flows_mig$y_discret <- cut(flows_mig$y, breaks,
                           include.lowest = T)
```

We plot the largest flows only.

```r
par(bg = "grey25", oma = c(0, 0, 0, 0),
    mar = c(0, 0, 0, 0), mai = c(0, 0, 0, 0))
# plot municipalities
plot(st_geometry(sf_mig_o_d), col = "grey13", border = "grey25",
     bg = "grey25", lwd = 0.5)
# plot graduated links
gradLinkTypoLayer(
  x = mtq_mob,
  xid = c("origin","dest"),
  df = flows_mig,
  dfid = c("origin","dest"),
  var = "y",
  breaks = breaks[2:6],
  lwd = c(1, 4, 8, 16),
  var2 = "y_discret",
  legend.var.pos = "left",
  legend.var.title.txt = "Nb. of\nTrade",
)

# map layout
layoutLayer(title = "Trades",
            sources = "Simulated data",
            author = paste0("cartography ", packageVersion("cartography")),
            frame = FALSE, col = "grey25", coltitle = "white",
            tabtitle = TRUE)
```



Non spatial modelling

Gravity model

The form of the classical 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 product properties.
- Use the function *lm()* applied to the vectorized form of the data set.

We also consider the model which includes the lagged variables (option **lagged = T**) :

$Y = \alpha 1_N + X_o\beta_o + X_d\beta_d + XL_o\delta_o + XL_d\delta_d + \gamma g + \epsilon.$

Finally, we also consider the intraregional variation in flows (option **intra_x = T**): $Y = \alpha 1_N + X_o\beta_o + X_d\beta_d + X_i\beta_i + \gamma g + \epsilon$. In that case, we remind that X_o and X_d have 0 values for the intraregional flows.

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 allows some further simplifications in the resolution of the problem. In our function, we propose to use or not this siplification. We call this function *gravity_model()* which consider the case where the list of origin and destination coincide. It takes as input arguments :

```
gravity_model(x, Y, G, ind_d = NULL, ind_o = NULL,
                          lagged = F, centered = F, w = NULL,
                          intra_x = F)
```

- **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 $n \times n$,
- **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,
- **lagged**, a boolean which includes if **T** the lagged explanatory variables,
- **centered**, a boolean which centered the explanatory variables and uses the simplified formulas,
- **w**, the spatial weight matrix to use when the lagged explanatory variables are used,
- **intra_x**, a boolean which includes X_i.

```
source("./R/gravity_model_2.R")
```

Applications

With the Australian simulated data

```
gravity_model(x = as.matrix(spdf_au@data$x), Y = Y_au_od_1,
              G = G_au, lagged = T, centered = T, w = w_au,
              intra_x = F)
```

```
##                       Estimate   t value
## (Intercept)          1.1720308        NA
## Dest_x_1             0.9769370 80.495591
## Dest_lagged_x_1      0.2896625  9.039242
## Origin_x_1           0.5193237 42.790140
## Origin_lagged_x_1    0.1019835  3.182508
## Distance            -1.9061557 -7.031534
```

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

```
gravity_flows <- lm(y_1 ~ x_d + lagged_x_d + x_o + lagged_x_o + g,
                    data = flows_au_od)
```

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 ~ x_d + lagged_x_d + x_o + lagged_x_o + g, data = flows_au_od)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.54031 -0.57884 -0.04272  0.63090  1.69673
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.17203    0.90523   1.295  0.20054
## x_d          0.97694    0.01214  80.496  < 2e-16 ***
## lagged_x_d   0.28966    0.03204   9.039 1.14e-12 ***
```

```
## x_o          0.51932    0.01214  42.790  < 2e-16 ***
## lagged_x_o   0.10198    0.03204   3.183  0.00235 **
## g           -1.90616    0.27109  -7.032 2.59e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8798 on 58 degrees of freedom
## Multiple R-squared:  0.9957, Adjusted R-squared:  0.9953
## F-statistic:  2676 on 5 and 58 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()*.

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 Australian simulated data with intra effect

```
gravity_model(x = as.matrix(spdf_au@data$x), Y = Y_au_1,
              G = G_au, lagged = T, centered = F, w = w_au,
                intra_x = T)
```

```
##                       Estimate    t value
## (Intercept)          1.3033153  0.9910409
## c_i                  2.9486244  1.4057186
## Dest_x_1             0.9751649 73.7831007
## Dest_lagged_x_1      0.2982068  8.3869809
## Origin_x_1           0.5175517 39.1590861
## Origin_lagged_x_1    0.1105278  3.1085621
## Intra_x_1            2.4999213 72.7319742
## Intra_lagged_x_1     0.4515568  4.9986009
## Distance            -2.2838836 -4.5442838
```

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

```
gravity_flows <- lm(y_1 ~ vec_In + x_d + lagged_x_d +
                      x_o + lagged_x_o +
                      lagged_x_i + x_i + g,
                    data = flows_au)
summary(gravity_flows)
```

```
##
## Call:
## lm(formula = y_1 ~ vec_In + x_d + lagged_x_d + x_o + lagged_x_o +
##     lagged_x_i + x_i + g, data = flows_au)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.46150 -0.67034 -0.04866  0.62828  1.60896
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.30332    1.31510   0.991  0.32601
## vec_In       2.94862    2.09759   1.406  0.16543
## x_d          0.97516    0.01322  73.783  < 2e-16 ***
```

```
## lagged_x_d   0.29821    0.03556   8.387 2.05e-11 ***
## x_o          0.51755    0.01322  39.159  < 2e-16 ***
## lagged_x_o   0.11053    0.03556   3.109  0.00297 **
## lagged_x_i   0.45156    0.09034   4.999 6.22e-06 ***
## x_i          2.49992    0.03437  72.732  < 2e-16 ***
## g           -2.28388    0.50258  -4.544 3.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8821 on 55 degrees of freedom
## Multiple R-squared:  0.9978, Adjusted R-squared:  0.9974
## F-statistic:  3049 on 8 and 55 DF,  p-value: < 2.2e-16
```

With the Australian real data

```
gravity_model(x = log(as.matrix(spdf_au@data[,
                      c("wage", "pop", "age")])),
              Y = log(au_flows), ind_d = c(2, 3),
              ind_o = c(1, 2),
              G = G_au, lagged = T, centered = F, w = w_au,
              intra_x = F)
```

```
##                         Estimate       t value
## (Intercept)        -5.051673e+01  -0.523040792
## Dest_pop            1.251048e-01   0.691395278
## Dest_age            2.187303e+00   0.726579577
## Dest_lagged_pop    -6.147041e-01  -2.079579205
## Dest_lagged_age     5.683966e-01   0.074591769
## Origin_wage        -1.167183e+00  -1.039328769
## Origin_pop          5.785819e-04   0.002628818
## Origin_lagged_wage  7.138924e+00   0.918653433
## Origin_lagged_pop  -3.963546e-01  -1.438307362
## Distance           -3.168656e+00 -13.254195081
```

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

```
gravity_real_flows <- lm(log(real_Y) ~ log(pop_d) + log(age_d) +
                          lagged_ln_pop_d + lagged_ln_age_d +
                           log(wage_o) + log(pop_o) +
                           lagged_ln_wage_o + lagged_ln_pop_o + g,
                         data = flows_au_od)
summary(gravity_real_flows)
```

```
##
## Call:
## lm(formula = log(real_Y) ~ log(pop_d) + log(age_d) + lagged_ln_pop_d +
##     lagged_ln_age_d + log(wage_o) + log(pop_o) + lagged_ln_wage_o +
##     lagged_ln_pop_o + g, data = flows_au_od)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.89055 -0.48742  0.05391  0.51197  1.47402
##
## Coefficients:
```

```
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -5.052e+01  9.658e+01  -0.523   0.6031
## log(pop_d)        1.251e-01  1.809e-01   0.691   0.4923
## log(age_d)        2.187e+00  3.010e+00   0.727   0.4706
## lagged_ln_pop_d  -6.147e-01  2.956e-01  -2.080   0.0423 *
## lagged_ln_age_d   5.684e-01  7.620e+00   0.075   0.9408
## log(wage_o)      -1.167e+00  1.123e+00  -1.039   0.3033
## log(pop_o)        5.786e-04  2.201e-01   0.003   0.9979
## lagged_ln_wage_o  7.139e+00  7.771e+00   0.919   0.3624
## lagged_ln_pop_o  -3.964e-01  2.756e-01  -1.438   0.1561
## g                -3.169e+00  2.391e-01 -13.254   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7585 on 54 degrees of freedom
## Multiple R-squared:  0.8518, Adjusted R-squared:  0.8271
## F-statistic: 34.49 on 9 and 54 DF,  p-value: < 2.2e-16
```

Intraregionnal model:

```
gravity_model(x = log(as.matrix(spdf_au@data[,
                      c("wage", "pop")])),
              Y = log(au_flows), G = G_au, lagged = T, w = w_au,
              intra_x = T)
```

```
##                         Estimate      t value
## (Intercept)        -217.56906471 -2.442340937
## c_i                 239.41145351  1.353111906
## Dest_wage             0.07843001  0.105610089
## Dest_pop              0.26593507  1.782219863
## Dest_lagged_wage      9.32553198  1.841599157
## Dest_lagged_pop      -0.18027685 -0.968432411
## Origin_wage          -0.03483535 -0.046907608
## Origin_pop            0.21637189  1.450061756
## Origin_lagged_wage   11.29466535  2.230462167
## Origin_lagged_pop    -0.11444915 -0.614811426
## Intra_wage           -2.18561245 -1.145659208
## Intra_pop             0.92857459  2.497180844
## Intra_lagged_wage     0.01437802  0.001085805
## Intra_lagged_pop      0.04205367  0.090251833
## Distance             -0.93071136 -3.285626145
```

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

```
gravity_real_flows <- lm(log(real_Y) ~ vec_In +
                            ln_wage_d + ln_pop_d  +
                            lagged_ln_wage_d + lagged_ln_pop_d  +
                            ln_wage_o + ln_pop_o +
                            lagged_ln_wage_o + lagged_ln_pop_o +
                            ln_wage_i + ln_pop_i +
                            lagged_ln_wage_i + lagged_ln_pop_i + g,
                          data = flows_au)
summary(gravity_real_flows)
```

```
##
## Call:
```

```
## lm(formula = log(real_Y) ~ vec_In + ln_wage_d + ln_pop_d + lagged_ln_wage_d +
##     lagged_ln_pop_d + ln_wage_o + ln_pop_o + lagged_ln_wage_o +
##     lagged_ln_pop_o + ln_wage_i + ln_pop_i + lagged_ln_wage_i +
##     lagged_ln_pop_i + g, data = flows_au)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.94845 -0.23826  0.02689  0.27302  1.16890
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -217.56906   89.08218  -2.442  0.01825 *
## vec_In            239.41145  176.93396   1.353  0.18223
## ln_wage_d           0.07843    0.74264   0.106  0.91632
## ln_pop_d            0.26594    0.14922   1.782  0.08091 .
## lagged_ln_wage_d    9.32553    5.06382   1.842  0.07159 .
## lagged_ln_pop_d    -0.18028    0.18615  -0.968  0.33758
## ln_wage_o          -0.03484    0.74264  -0.047  0.96278
## ln_pop_o            0.21637    0.14922   1.450  0.15341
## lagged_ln_wage_o   11.29467    5.06382   2.230  0.03033 *
## lagged_ln_pop_o    -0.11445    0.18615  -0.615  0.54152
## ln_wage_i          -2.18561    1.90773  -1.146  0.25750
## ln_pop_i            0.92857    0.37185   2.497  0.01593 *
## lagged_ln_wage_i    0.01438   13.24180   0.001  0.99914
## lagged_ln_pop_i     0.04205    0.46596   0.090  0.92846
## g                  -0.93071    0.28327  -3.286  0.00188 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4571 on 49 degrees of freedom
## Multiple R-squared:  0.9512, Adjusted R-squared:  0.9372
## F-statistic: 68.18 on 14 and 49 DF,  p-value: < 2.2e-16
```

Comparison of computationnal time

```r
microbenchmark::microbenchmark(
  # australia
  gravity_model(x = as.matrix(spdf_au@data$x),
                Y = Y_au_od_1, G = G_au, centered = T),
  gravity_model(x = as.matrix(spdf_au@data$x),
                Y = Y_au_od_1, G = G_au),
  lm(y_1 ~ x_d + x_o + g, data = flows_au_od),
  gravity_model(x = as.matrix(spdf_au@data$x),
                Y = Y_au_1, G = G_au, intra_x = T),
  # germany
  gravity_model(x = as.matrix(spdf_ge@data$x),
                 Y = Y_ge_od_1, G = G_ge, centered = T),
  gravity_model(x = as.matrix(spdf_ge@data$x),
                 Y = Y_ge_od_1, G = G_ge),
  lm(y_1 ~ x_d + x_o + g, data = flows_ge),
  gravity_model(x = as.matrix(spdf_ge@data$x),
                 Y = Y_ge_od_1, G = G_ge, intra_x = T),
  # usa
```

```r
  gravity_model(x = as.matrix(spdf_usa@data$x),
                Y = Y_usa_od_1, G = G_usa, centered = T),
  gravity_model(x = as.matrix(spdf_usa@data$x),
                Y = Y_usa_od_1, G = G_usa),
  lm(y_1 ~ x_d + x_o + g, data = flows_usa),
  gravity_model(x = as.matrix(spdf_usa@data$x), intra_x = T,
                Y = Y_usa_od_1, G = G_usa),
  times = 1000L
)
```

```
## Unit: microseconds
##                                                                                    expr
##     gravity_model(x = as.matrix(spdf_au@data$x), Y = Y_au_od_1, G = G_au,      centered = T)
##                           gravity_model(x = as.matrix(spdf_au@data$x), Y = Y_au_od_1, G = G_au)
##                                                  lm(y_1 ~ x_d + x_o + g, data = flows_au_od)
##         gravity_model(x = as.matrix(spdf_au@data$x), Y = Y_au_1, G = G_au,      intra_x = T)
##     gravity_model(x = as.matrix(spdf_ge@data$x), Y = Y_ge_od_1, G = G_ge,      centered = T)
##                           gravity_model(x = as.matrix(spdf_ge@data$x), Y = Y_ge_od_1, G = G_ge)
##                                                     lm(y_1 ~ x_d + x_o + g, data = flows_ge)
##      gravity_model(x = as.matrix(spdf_ge@data$x), Y = Y_ge_od_1, G = G_ge,      intra_x = T)
##   gravity_model(x = as.matrix(spdf_usa@data$x), Y = Y_usa_od_1,      G = G_usa, centered = T)
##                 gravity_model(x = as.matrix(spdf_usa@data$x), Y = Y_usa_od_1,      G = G_usa)
##                                                    lm(y_1 ~ x_d + x_o + g, data = flows_usa)
##    gravity_model(x = as.matrix(spdf_usa@data$x), intra_x = T, Y = Y_usa_od_1,      G = G_usa)
##       min        lq      mean    median        uq       max neval
##    534.566  575.0730  643.9429  596.3045  639.1175 12019.406  1000
##    557.962  596.3400  671.8970  619.4925  663.2130 14228.556  1000
##    794.026  859.7815  936.1527  887.5780  952.0420 12221.736  1000
##    648.616  695.7940  763.9454  720.9370  772.0960 12438.244  1000
##    566.692  607.3050  644.7051  630.4575  669.3240   841.238  1000
##    585.829  631.7850  699.3453  658.2890  700.6135 14450.372  1000
##    818.819  883.9810  979.9978  911.3935  971.4570 12124.935  1000
##    695.969  742.4830  823.3441  769.2325  822.5210 12428.885  1000
##    778.591  822.5910  902.5733  852.4125  909.4030 15081.108  1000
##    820.006  863.2735  931.2615  897.0070  952.1815 12287.736  1000
##   1082.959 1163.4510 1243.4164 1197.1845 1273.6255 12631.285  1000
##    996.286 1047.5845 1153.9971 1091.7940 1168.6190 12837.806  1000
```

Remarks:

- *gravity_model()* is faster than *lm()* function.
- when centering the explanatory variables, the computational time is slightly faster.

Origin and destination differ

```r
source("./R/gravity_model_3.R")
```

In that case, we use another function:

```
gravity_model_geo(Y, G, x_d, x_o,
                  ind_d = NULL, ind_o = NULL,
                  lagged = F, centered = F,
                  DW = NULL, OW = NULL)
```

- $\mathbf{Y}$, the matrix of flows of size $n_d \times n_o$,

- **G**, the matrix of distances of size $n_d \times n_o$,
- **x_d**, a **data.frame** or a matrix with explanatory variables observed on the n_d geographical sites.
- **x_o**, a **data.frame** or a matrix with explanatory variables observed on the n_o geographical sites.
- **ind_d**, the indices of the variables in **x_d** which will be used at the destination,
- **ind_o**, the indices of the variables in **x_o** of the variables used at the origin,
- **lagged**, a boolean which includes if **T** the lagged explanatory variables,
- **centered**, a boolean which centered the explanatory variables and uses the simplified formulas,
- **DW**, the spatial weight matrix for x_d,
- **OW**, the spatial weight matrix for x_o.

```
gravity_model_geo(Y = log(1 + Y_mig), G = log(1 + G_mig),
                  x_d = log(1 + mig_data_dest[, c("popul_d", "gdp_d")]),
                  x_o = log(1 + mig_data_origin[, c("gdp_o", "civilconflict_o")]),
                  lagged = T, OW = w_mig_o, DW = w_mig_d)
```

```
##                            Estimate   t value
## (Intercept)             -46.9489524 -4.906028
## popul_d                  -0.9102603 -6.533500
## gdp_d                     1.8756220 10.996898
## lagged_popul_d            0.3442942  1.329881
## lagged_gdp_d              0.6985628  2.299576
## gdp_o                     0.3600043  8.721082
## civilconflict_o           0.9782239  4.039618
## lagged_gdp_o             -0.3130075 -4.072969
## lagged_civilconflict_o    1.4441002  2.870804
## Distance                 -1.0190182 -5.508375
```

**Comparaison with *lm()* function:

```
summary(lm(log(1 + y) ~ log(1 + popul_d) + log(1 + gdp_d) +
             lagged_ln_popul_d + lagged_ln_gdp_d +
             log(1 + gdp_o) + civilconflict_o +
             lagged_ln_gdp_o + lagged_civilconflict_o +
             log(1 + g), data = flows_mig)
)
```

```
##
## Call:
## lm(formula = log(1 + y) ~ log(1 + popul_d) + log(1 + gdp_d) +
##     lagged_ln_popul_d + lagged_ln_gdp_d + log(1 + gdp_o) + civilconflict_o +
##     lagged_ln_gdp_o + lagged_civilconflict_o + log(1 + g), data = flows_mig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8481 -1.2546 -0.0444  1.1924  6.4681
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -46.94895    9.56965  -4.906 1.07e-06 ***
## log(1 + popul_d)        -0.91026    0.13932  -6.533 9.95e-11 ***
## log(1 + gdp_d)           1.87562    0.17056  10.997  < 2e-16 ***
## lagged_ln_popul_d        0.34429    0.25889   1.330  0.18384
## lagged_ln_gdp_d          0.69856    0.30378   2.300  0.02167 *
## log(1 + gdp_o)           0.36000    0.04128   8.721  < 2e-16 ***
## civilconflict_o          0.67805    0.16785   4.040 5.74e-05 ***
## lagged_ln_gdp_o         -0.31301    0.07685  -4.073 4.99e-05 ***
```

```
## lagged_civilconflict_o   1.00097    0.34867   2.871  0.00418 **
## log(1 + g)              -1.01902    0.18499  -5.508 4.54e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.958 on 1061 degrees of freedom
## Multiple R-squared:  0.3554, Adjusted R-squared:   0.35
## F-statistic:    65 on 9 and 1061 DF,  p-value: < 2.2e-16
```

Spatial Modelling

When the number of flows N is equal to $n_o \times n_d$, this implies some simplifications in the computations. However, this is not always the case in practice. That is why we consider the two options :

- $N = n_o \times n_d$
- $N < n_o \times n_d$

Spatial Autoregressive Interaction Models when $N = n_o \times n_d$

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

- *ftrace1()*, which computes the traces of the spatial weight matrices W, W^2, W^3, ..., W^{miter},
- *fodet1()*, which computes the jacobian matrix in the case of 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 a single spatial weight matrix,
- *c_sarf()*, which computes the log likelihood conditionnally to ρ.

Traces of the spatial weight matrix

We first code the function **ftrace1()** which computes the traces of W, W^2, W^3, ..., W^{miter}. The two possible methods are **"exact"** (based on the computation of W, W^2, W^3, ..., 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.

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

```
(traces <- ftrace1(w_au))
```

```
##  [1] 0.0000000 2.2216667 0.6800000 1.3862347 0.8506111 1.1482334 0.9287809
##  [8] 1.0616054 0.9661001 1.0268552
```

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

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

```
##  [1] 0.0000000 2.2216667 0.6775452 1.3862026 0.8491984 1.1483120 0.9279157
##  [8] 1.0615429 0.9655098 1.0266705
```

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:

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

- **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.

```
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:

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

- **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.

```
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_au)
```

```
## [1] -0.5963679
```

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

```
lndetmc(0.25, traces_approx, n_au)
```

```
## [1] -0.5962631
```

Case of two spatial weight matrices

One can use formula (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_au) + lndetmc(0.2, traces_approx, n_au)
```

```
## [1] -2.00631
```

Evaluation of the log likelihood conditionnally to ρ

The function *c_sarf()* takes as arguments:

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

- **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.

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

Function *sar_flow()* model

It takes as input arguments :

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

- **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.

Function *sar_flow_2()* model

In the case when $N \leq n^2$, users have to present the data in vectorized form. It is also possible to use this function when $N = n^2$,

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 toy data

Model 2

Bayesian estimation

We evaluate model 2 when Y corresponds to the DGP used with model 2.

We compare the estimates obtained when we used the method $N = n^2$ and when we used the method $N \leq n^2$.

```
system.time(sar_simu_2_method1 <- sar_flow(x = matrix(au_df$x,
                                  nrow = n_au, dimnames = list(1:n_au, "x")),
                                  Y = Y_au_2,
                                  G = G_au,
```

```r
                                    w = w_au,
                                    model = "model_2",
                                    lagged = T)
)
sar_simu_2_method1
```

```
##                   mean   lower_05   lower_95    t_stat
## rho_d        0.4623762  0.3091899  0.6100935  5.031816
## (intercept) 38.4598025 27.4516349 49.9175758  5.609155
## x_d          0.9705691  0.9431801  0.9973142 59.020454
## lagged_x_d   0.4833250  0.3368945  0.6337902  5.313244
## x_o          0.4649870  0.3348140  0.6002513  5.757190
## lagged_x_o   0.2207421  0.1229031  0.3166776  3.758288
## g           -1.8332084 -2.3579427 -1.3009484 -5.716980
```

Both methods give approximatly the same estimates. The first one is obtained in 6.5s when the second is obtained in 54s.

```r
system.time(sar_simu_2_method2 <- sar_flow_2(x = flows_au[, c("x_d", "W_dx_d",
                                                            "x_o", "W_ox_o")],
                                             y = flows_au$y_2,
                                             g = flows_au$g,
                                             W_d = W_au_d,
                                             model = "model_2", centered = F)
)
sar_simu_2_method2
```

```
##                   mean   lower_05   lower_95     t_stat
## rho_d        0.4656506  0.3181005  0.6107132  5.2486684
## (intercept) -2.6479208 -7.0264660  1.7447900 -0.9965869
## x_d          0.9700295  0.9420244  0.9975006 57.2890893
## W_dx_d       0.4803916  0.3367637  0.6239925  5.4966765
## x_o          0.4628306  0.3349231  0.5918894  5.9665638
## W_ox_o       0.2186749  0.1228022  0.3153432  3.7171283
## g           -1.8261854 -2.3552201 -1.2899456 -5.6536965
```

Computationnal time of the Spatial Interaction Durbin Model 2 according to the size of the samples:

```
##     matrix  vector
## au   6.522  67.286
## ge   6.042  80.966
## usa  7.341 440.591
```

Comparaison with the log-likelihood estimation

We compare the results with the *lagsarlm()* function and we remark that we obtain similar results :(results are obtained in less than 1s).

```r
result_lagsarlm <- lagsarlm(y_2 ~ x_d + lagged_x_d + x_o + x_i +  lagged_x_o + g,
                            data = flows_au,
                            mat2listw(W_au_d))
```

```
## Warning: Function lagsarlm moved to the spatialreg package
```

```r
summary(result_lagsarlm)
```

```
##
## Call:spatialreg::lagsarlm(formula = formula, data = data, listw = listw,
```

```
##     na.action = na.action, Durbin = Durbin, type = type, method = method,
##     quiet = quiet, zero.policy = zero.policy, interval = interval,
##     tol.solve = tol.solve, trs = trs, control = control)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.51189 -0.76377 -0.28608  0.57234  3.42982
##
## Type: lag
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept)  6.166896   2.027862  3.0411  0.002357
## x_d          0.972057   0.018210 53.3790 < 2.2e-16
## lagged_x_d   0.079652   0.055891  1.4251  0.154118
## x_o          0.429677   0.051957  8.2699 2.220e-16
## x_i          2.565482   0.063786 40.2201 < 2.2e-16
## lagged_x_o  -0.048057   0.039900 -1.2045  0.228414
## g           -3.274083   0.817888 -4.0031 6.252e-05
##
## Rho: 0.48284, LR test value: 65.741, p-value: 5.5511e-16
## Asymptotic standard error: 0.042187
##     z-value: 11.445, p-value: < 2.22e-16
## Wald statistic: 131, p-value: < 2.22e-16
##
## Log likelihood: -105.4957 for lag model
## ML residual variance (sigma squared): 1.4637, (sigma: 1.2098)
## Number of observations: 64
## Number of parameters estimated: 9
## AIC: 228.99, (AIC for lm: 292.73)
## LM test for residual autocorrelation
## test value: 0.44473, p-value: 0.50485
```

Computational time with respect to the size of the samples:

```
## Warning: Function lagsarlm moved to the spatialreg package

## Warning: Function lagsarlm moved to the spatialreg package

## Warning: Function lagsarlm moved to the spatialreg package

##     vector
## au   0.358
## ge   0.339
## usa  6.392
```

Comparaison with the S2SLS estimation

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

```
result_s2sls<- stsls(y_2 ~ x_d + lagged_x_d + x_o + lagged_x_o + g,
                           data = flows_au,
                           mat2listw(W_au_d))
summary(result_s2sls)
```

We coded the S2SLS for a general spatial model flow (model 9). The codes are in the *s2sls_flow()* function which takes as input arguments :

```
s2sls_flow(x_d, x_o, y, g, W_d = NULL, W_o = NULL, W_w = NULL)
```

- **x__d**, a **data.frame** or a matrix with the destination explanatory variables observed on the N flows.
- **x__o**, a **data.frame** or a matrix with the origin explanatory variables observed on the N flows.
- **y**, the vector of flows of size N,
- **g**, the vector of distances of size N,
- **W__d**, the spatial weight destination matrix of size $N \times N$,
- **W__o**, the spatial weight origin matrix of size $N \times N$,
- **W__w**, the spatial weight matrix of size $N \times N$,

```
(sar_simu_2_sls <- s2sls_flow(x_d = flows_au[, c("x_d", "lagged_x_d")],
                              x_o = flows_au[, c("x_o", "lagged_x_o")],
                              y = flows_au$y_2,
                              g = flows_au$g,
                              instru_x_d = c(F, T),
                              instru_x_o = c(F, F),
                              W_d = W_au_d))
```

```
## $res_beta
## (intercept)         x_d  lagged_x_d         x_o  lagged_x_o           g
##  73.4272089   1.0006591  -0.6354743   0.7602675  -0.9768320 -22.5135575
##
## $RHO
##     rho_d
## 0.1877035
##
## $SIGMA
## 1 x 1 Matrix of class "dgeMatrix"
##          [,1]
## [1,] 149.2104
##
## $sd_beta
## (intercept)         x_d  lagged_x_d         x_o  lagged_x_o           g
##   1.1125780   5.1712365  -1.1134779   0.6952664  -2.0365634  -1.0171380
##
## $sd_rho
##     rho_d
## 0.1891958
```

Computational time with respect to the size of the samples:

```
##     vector
## au   0.006
## ge   0.016
## usa  0.135
```

Model 9

Bayesian estimation

We evaluate model 9 when Y corresponds to the DGP used with model 9 by using the full matrix. Computation was done in 376s by using the full matrix.

```
##                     mean    lower_05   lower_95      t_stat
## rho_d        0.406902180  0.218740009  0.5919997  3.50518778
```

```
## rho_o        0.233830826  0.008705707  0.4566817  1.70080403
## rho_w        0.009674402 -0.257909444  0.2722803  0.06062362
## (intercept) 41.348042875 23.873345120 59.6141991  3.82064656
## x_d          1.228495772  0.875828502  1.5876867  5.62514008
## lagged_x_d   0.533468572  0.308407285  0.7654556  3.84399674
## x_o          0.511842773  0.356113650  0.6686257  5.25435586
## lagged_x_o   0.244174395  0.119732665  0.3750072  3.11619189
## g           -1.960153706 -2.558843466 -1.3679442 -5.41762874
```

We evaluate model 9 when Y corresponds to the DGP used with model 9 by using the second method. Computation was done in 335s.

```r
system.time(sar_simu_9_method2 <- sar_flow_2(x = flows_au[,
                                        c("x_d", "lagged_x_d",
                                     y = flows_au$y_9,
                                     g = flows_au$g,
                                     W_d = W_au_d, W_o = W_au_o, W_w = W_au_w,
                                     model = "model_9", centered = F)
)
```

```
##                    mean     lower_05   lower_95     t_stat
## rho_d        0.42562687   0.25320346  0.5932288  4.0250772
## rho_o        0.25834110   0.01177208  0.4769866  1.8286354
## rho_w       -0.02813266  -0.26963156  0.2203537 -0.1899264
## (intercept) -6.30340566 -15.55306672  3.5534915 -1.0834542
## x_d          1.19015984   0.83975651  1.5850129  5.3057276
## W_dx_d       0.52419109   0.29146390  0.7577167  3.6979791
## x_o          0.49669842   0.35538186  0.6435418  5.6086749
## W_ox_o       0.23835135   0.10662290  0.3666120  3.0256707
## g           -1.93417000  -2.52985770 -1.3433499 -5.3655075
```

Computationnal time of the Spatial Interaction Durbin Model 2 according to the size of the samples:

```r
(time_bayesian_sdm <- data.frame(matrix = c(time_au_bayes_1[3], time_ge_bayes_1[3],
                                        time_usa_bayes_1[3]),
                           vector = c(time_au_bayes_2[3], time_ge_bayes_2[3],
                                        time_usa_bayes_2[3]),
                           row.names = c("au", "ge", "usa")))
```

```
##      matrix    vector
## au  405.647   294.659
## ge  407.003   489.030
## usa 404.491 22451.615
```

S2SLS estimation

```r
(sar_simu_9_sls <- s2sls_flow(x_d = flows_au[, c("x_d", "lagged_x_d")],
                              x_o = flows_au[, c("x_o", "lagged_x_o")],
                              y = flows_au$y_9,
                              g = flows_au$g,
                              instru_x_d = c(F, T),
                              instru_x_o = c(F, F),
                              W_o = W_au_o,
                              W_d = W_au_d,
                              W_w = W_au_w))
```

```
## $res_beta
```

```
##  (intercept)          x_d   lagged_x_d          x_o   lagged_x_o
## -119.7741817   -0.2777263   -1.4760173   -0.3323978   -0.5598469
##            g
##   27.9684298
##
## $RHO
##      rho_d      rho_o      rho_w
## 1.12581789 0.99570604 0.07443029
##
## $SIGMA
## 1 x 1 Matrix of class "dgeMatrix"
##         [,1]
## [1,] 22.1904
##
## $sd_beta
## (intercept)         x_d  lagged_x_d         x_o  lagged_x_o           g
##  -5.4651321  -0.9810884  -6.5131071  -1.5150154  -3.6102864   5.3324818
##
## $sd_rho
##     rho_d     rho_o     rho_w
## 5.9029705 6.4398340 0.3862299
```

Interpreting the results

Understanding the decomposition of impacts

With the bayesian estimates obtained below in the model 9, one could obtain the predictions by using for example the IC formula (Goulard et al, 2017).

```
delta_estimates <- sar_simu_9_method1[c("(intercept)", "x_d", "x_o",
                                "lagged_x_d", "lagged_x_o", "g"), "mean" ]
#delta_estimates <- c(0, 1, 0.5, 0, 0, -0.5 )
rho_estimates <- sar_simu_9_method1[c("rho_d", "rho_o", "rho_w"), "mean" ]
#rho_estimates <- c(0.4, 0.4, -0.16)
A_W <- solve(diag(n_au ^ 2) - rho_estimates[1] * W_au_d -
                   rho_estimates[2] * W_au_o - rho_estimates[3] * W_au_w)
Y_predict <- A_W %*% Z_au_od %*% delta_estimates
```

Lesage and Pace (2004) illustrate the concept of spillovers in the general case a SAR model. They look the effect on the predictions when they increase by one unit the explanatory variable for one observation. Here we increase the variable x by one unit in the observation R3. For doing that we create a function which permits to transform the data.frame.

```
epsilon_when_change_one_unit <- function (which_unit, x, g, W_d, W_o, A_W,
                                          delta_estimates, Y_predict,
                                          change_xo = T, change_xd = T) {
  n <- length(x)
  N <- n * n
  add_1 <- numeric(n)
  add_1[which_unit] <- 1
  x_changed <- x + add_1
  if (change_xo) {
    x_o <- kronecker(x_changed, rep(1, n))
```

```
  } else {
    x_o <- kronecker(x, rep(1, n))
  }
  if (change_xd) {
    x_d <- kronecker(rep(1, n), x_changed)
  } else {
    x_d <- kronecker(rep(1, n), x)
  }
  Z_changed <- cbind(rep(1, N), x_d, x_o, W_d %*% x_d, W_o %*% x_o, g)
  Y_predict_changed <- A_W %*% Z_changed %*% delta_estimates
  return(as.numeric(Y_predict_changed - Y_predict))
}
```

Now, we look at the differences obtained between the predictions and we can observe that when changing only observation R3, it impacted all the flows.

```
matrix(epsilon_when_change_one_unit(3, au_df$x, flows_au$g,
                                    as.matrix(W_au_d), as.matrix(W_au_o),
                                    A_W, delta_estimates, Y_predict), 8, 8, byrow = T)
```

```
##           [,1]      [,2]     [,3]      [,4]      [,5]       [,6]      [,7]
## [1,] 0.7583940 0.3903010 2.096858 0.6723578 0.3706631 0.30137492 0.3254686
## [2,] 0.5782279 0.2101349 1.916692 0.4921917 0.1904970 0.12120882 0.1453025
## [3,] 1.4781591 1.1100661 2.816623 1.3921229 1.0904282 1.02114001 1.0452337
## [4,] 0.7180803 0.3499873 2.056544 0.6320441 0.3303494 0.26106123 0.2851549
## [5,] 0.5712131 0.2031202 1.909677 0.4851769 0.1834822 0.11419408 0.1382878
## [6,] 0.5454420 0.1773491 1.883906 0.4594058 0.1577111 0.08842297 0.1125166
## [7,] 0.5558398 0.1877468 1.894304 0.4698036 0.1681089 0.09882073 0.1229144
## [8,] 0.5402742 0.1721812 1.878738 0.4542380 0.1525433 0.08325513 0.1073488
##            [,8]
## [1,] 0.28143388
## [2,] 0.10126778
## [3,] 1.00119897
## [4,] 0.24112020
## [5,] 0.09425304
## [6,] 0.06848193
## [7,] 0.07887969
## [8,] 0.06331409
```

Lesage and Thomas-Agnan (2014) propose to summarize the impacts into 4 main groups :

- The OD which consists in summing up all the flows which have $R3$ as origin (and excluding the intra) which correspond to the 3rd row,
- The DE which consists in summing up all the flows which have $R3$ as destination (and excluding the intra) which correspond to the 3rd column,
- The intra which consists in the intra flow $R3$ (3rd row, 3rd column)
- The NE which consists in the rest of the flows

Then, to have an overview of all the impacts, one can change from one unit all the observations, and then summarize the impacts as seen previously :

```
res <- matrix(0, 5, 4)
for (i in 1:3) {

  if (i == 1 | i == 2) {
    change_xo = T
```

```
    if (i == 1)
      change_xd = T
    else
      change_xd = F
  }

  if (i == 3) {
    change_xo = F
    change_xd = T
  }

OE <- matrix(0, 8, 8)
DE <- matrix(0, 8, 8)
NE <- matrix(0, 8, 8)
intra <- matrix(0, 8, 8)
total <- matrix(0, 8, 8)
for (k in 1:8) {
  change_Rk <- matrix(epsilon_when_change_one_unit(k, au_df$x, flows_au$g,
                                                   as.matrix(W_au_d), as.matrix(W_au_o),
                                                   A_W, delta_estimates, Y_predict,
                                                   change_xo = change_xo,
                                                   change_xd = change_xd),
                      8, 8, byrow = T)
  intra[k, k] <- intra[k, k] + change_Rk[k, k]
  OE[k, ] <- change_Rk[k, ]
  OE[k, k] <- 0
  DE[, k] <- change_Rk[, k]
  DE[k, k] <- 0
  NE[!((1:8) %in% k), !((1:8) %in% k)] <- NE[!((1:8) %in% k), !((1:8) %in% k)] + change_Rk[!((1:8) %in%
}

# to obtain the final results
res[1:4, i] <- c(mean(OE), mean(DE), mean(intra), mean(NE))
}
res[, 4] <- apply(res[, 2:3], 1, sum)
res[5, ] <- apply(res, 2, sum)
rownames(res) <- c("Origin", "Destination", "Intra",
                   "Network", "Total")
colnames(res) <- c("delta_x", "delta_xo", "delta_xd", "delta_xo + delta_xd")
```

Finally, we obtain that table :

```
res
```

```
##               delta_x  delta_xo  delta_xd delta_xo + delta_xd
## Origin      1.2380738 0.8529943 0.3850796           1.2380738
## Destination 1.8629516 0.1484645 1.7144872           1.8629516
## Intra       0.3667831 0.1218563 0.2449267           0.3667831
## Network     3.7348084 1.0392513 2.6955571           3.7348084
## Total       7.2026169 2.1625663 5.0400506           7.2026169
```

Computation

Herby, we try to simplify the computations of the impacts.

Computation of $A(W)$

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)

We separate origin and destination estimates :

```
hat_beta_d <- sar_simu_9_method1["x_d", "mean"]
#hat_beta_d <- 1
names(hat_beta_d) <- "x_d"
hat_beta_o <- sar_simu_9_method1["x_o", "mean"]
#hat_beta_o <- 0.5
names(hat_beta_o) <- "x_o"
hat_delta_d <- sar_simu_9_method1["lagged_x_d", "mean"]
#hat_delta_d <- 0
names(hat_delta_d) <- "Wd_xd"
hat_delta_o <- sar_simu_9_method1["lagged_x_o", "mean"]
#hat_delta_o <- 0
names(hat_delta_o) <- "Wo_xo"
```

We compute $A(W)$:

```
W_hat <- rho_estimates[1] * W_au_d + rho_estimates[2] * W_au_o +
  rho_estimates[3] * W_au_w
AW <- powerWeights(W_hat, 1, order = 250, Diagonal(nrow(W_hat)),
                   tol = .Machine$double.eps^(3/5))
```

We may need to compute $A(W) \times W$ in the case of the SDM model:

```
AW_Wo <- AW %*% W_au_o
AW_Wd <- AW %*% W_au_d
```

We need to identify each origin and destination:

```
all_dest <- flows_au[, "dest"]
all_origin <- flows_au[, "origin"]
```

We coded the function *OE_impact()*, *DE_impact()*, *NE_impact()*, *intra_impact()* which take as argument :

- **AW**, the matrix $A(W)$,
- **AW_W**, the matrix $A(W) \times W$ if the model is spatial Durbin,
- **all_dest**, a vector which contains the id of the destination in $A(W)$,
- **all_origin**, a vector which contains the id of the origin in $A(W)$,

Origin effect

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

To get the origin effect:

```r
origin_effect <- OE_impact(AW, AW_Wd = AW_Wd, AW_Wo = AW_Wo,
                           all_dest, all_origin)

(OE <- (origin_effect[[1]] * hat_beta_o + origin_effect[[2]] * hat_beta_d +
       origin_effect[[3]] * hat_delta_o + origin_effect[[4]] * hat_delta_d  ) / n_au^2)
```

```
##      x_o
## 1.238074
```

Destination effect

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

To get the destination effect:

```r
destination_effect <- DE_impact(AW, AW_Wd = AW_Wd, AW_Wo = AW_Wo,
                           all_dest, all_origin)

(DE <- (destination_effect[[1]] * hat_beta_o + destination_effect[[2]] * hat_beta_d +
         destination_effect[[3]] * hat_delta_o + destination_effect[[4]] * hat_delta_d  ) / n_au^2)
```

```
##      x_o
## 1.862952
```

NE effect

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

To get the NE effect:

```r
ne_effect <- NE_impact(AW, AW_Wd = AW_Wd, AW_Wo = AW_Wo,
                           all_dest, all_origin)

(NE <- (ne_effect[[1]] * hat_beta_o + ne_effect[[2]] * hat_beta_d +
         ne_effect[[3]] * hat_delta_o + ne_effect[[4]] * hat_delta_d  ) / n_au^2)
```

```
##      x_o
## 3.734808
```

intra effect

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

To get the intra effect:

```r
intra_effect <- intra_impact(AW, AW_Wd = AW_Wd, AW_Wo = AW_Wo,
                           all_dest, all_origin)

(intra <- (intra_effect[[1]] * hat_beta_o + intra_effect[[2]] * hat_beta_d +
         intra_effect[[3]] * hat_delta_o + intra_effect[[4]] * hat_delta_d  ) / n_au^2)
```

```
##       x_o
## 0.3634416
```

Summarise

```
(impacts_mod2 <- cbind(OE, DE, NE, intra))
```

```
##            OE       DE       NE     intra
## x_o 1.238074 1.862952 3.734808 0.3634416
```

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.