Spatial Flows Modelling with **R**

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

July 5, 2019

Contents

Packages needed:

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

```r
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
```

1 Data preparation

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

$$y = \rho_d W_d y + \rho_o W_o y + \rho_w W_w y + \alpha \iota_N + X_d \beta_d + X_o \beta_o + \gamma g + \epsilon$$

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.

1.1 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.

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 Sparse simulated data example

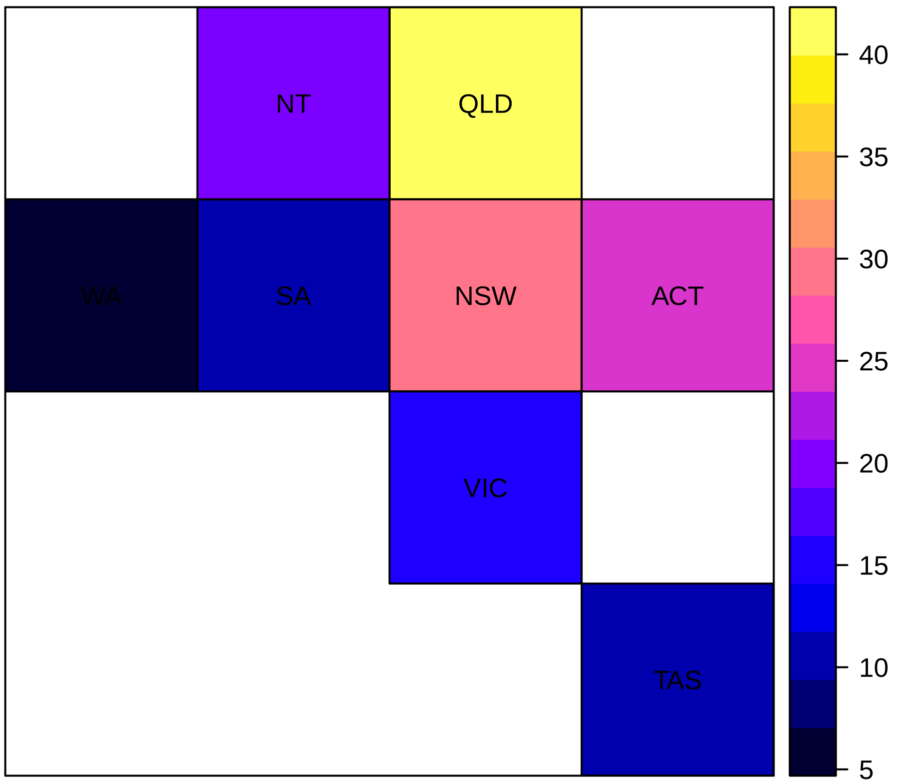
We will consider two cases: the case where the list of origins and destinations coincide and the case where it does not coincide.

1.1.4.1 List of origins and destinations coincide

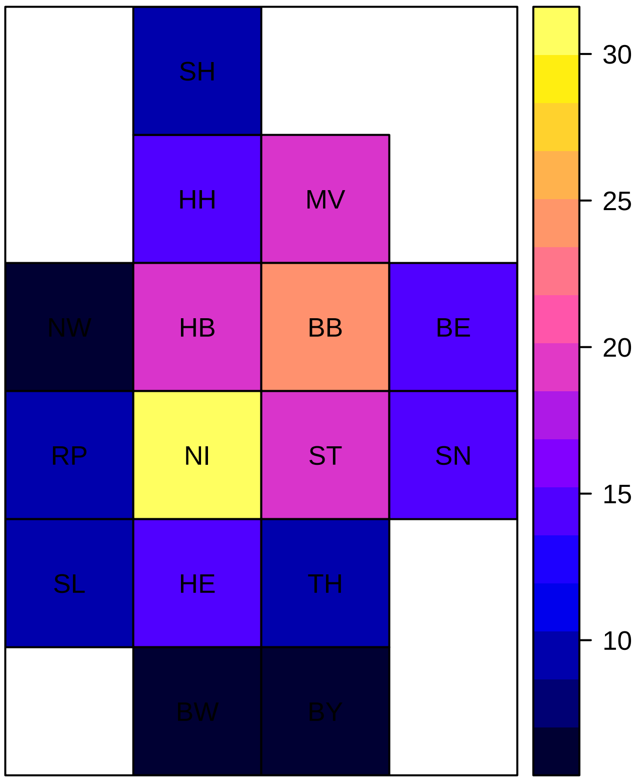
We consider three examples with different numbers of observations. We use examples from https://ialab.it.monash.edu/~dwyer/papers/maptrix.pdf. We define the polygons by using the function *create_grid()* inspired by the example given by R. Bivand in https://stat.ethz.ch/pipermail/r-sig-geo/2009-December/007163.html. We consider one explanatory variable for each data set. The programs to obtain these simulated examples are in "simulated_examples.R.

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

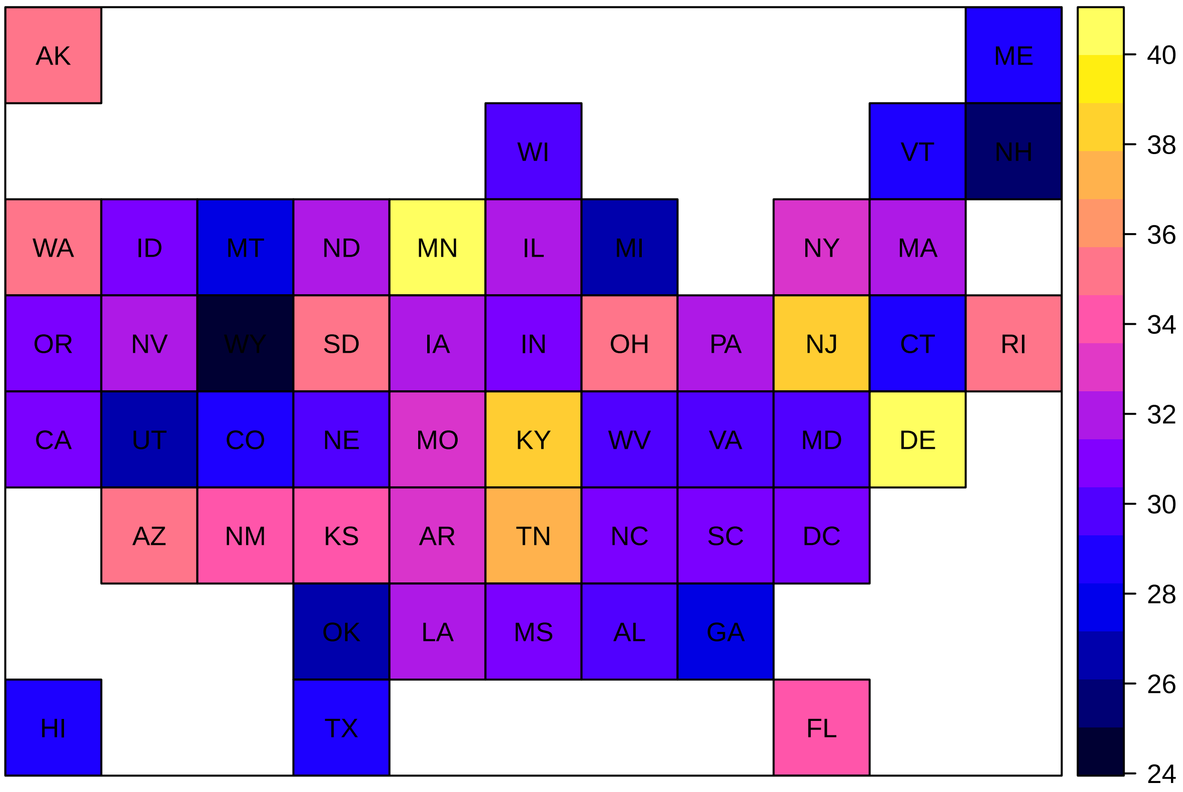
- The first example is a simplification of the 8 main regions of Australia.



- 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:



1.1.4.1.1 Storing the explanatory 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. Same has been done for Germany and USA:

```
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),
                      x_o = kronecker(spdf_au$x, rep(1, n_au)),
                      x_d = kronecker(rep(1, n_au), spdf_au$x))
head(flows_au)
```

```
##   origin dest x_o x_d
## 1     NT   NT  20  20
## 2     NT  QLD  20  40
## 3     NT   WA  20   7
## 4     NT   SA  20  10
## 5     NT  NSW  20  30
## 6     NT  ACT  20  25
```

1.1.4.1.2 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.

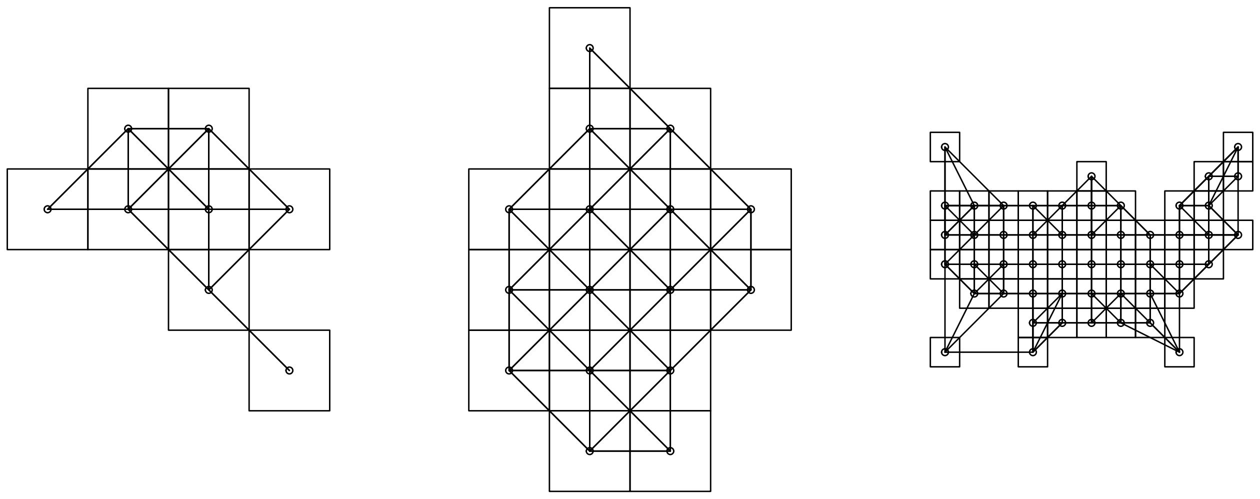
1.1.4.1.3 Construct the spatial weight matrices

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:

```
#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)
```



```
#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 aw 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:

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

```
## 32984 bytes
```

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

```
## 5080 bytes
```

```
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:

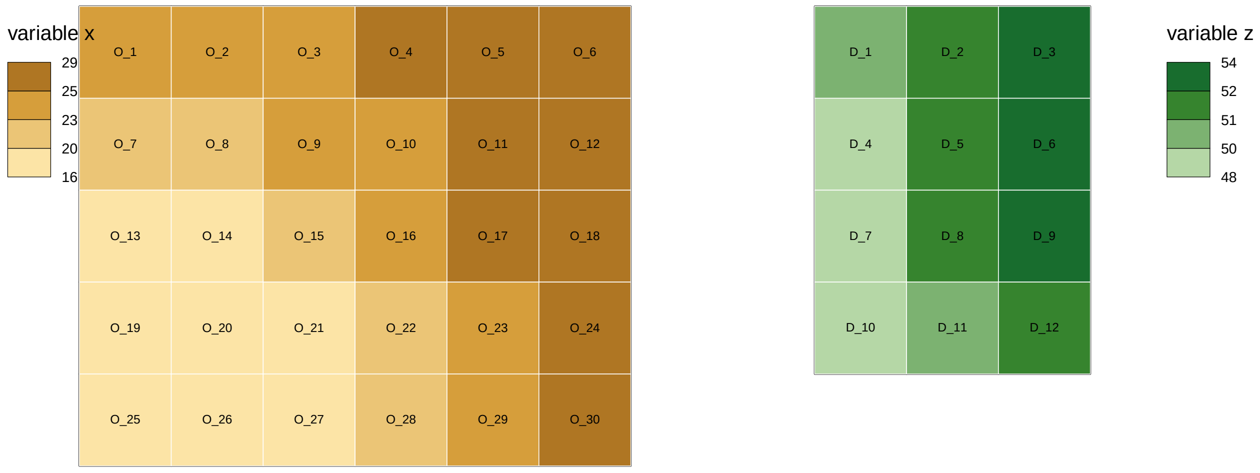
```
flows_au$W_dx_d <- as.matrix(W_au_d) %*% flows_au$x_d
flows_au$W_ox_o <- as.matrix(W_au_o) %*% flows_au$x_o
```

Same has been done for Germany and USA.

1.1.4.2 List of origins and destinations do not coincide

We prepare a grid of origins and a grid of destinations. The explanatory variables at origin and at destination are not necessarly the same.

```
# pdf("figures/grid.pdf", width = 10, height = 6)
plot(spdf_grid_o, xlim = c(-1, 12), ylim = c(0, 4))
choroLayer(spdf = spdf_grid_o, var = "x", nclass = 4,
           col = carto.pal(pal1 = "sand.pal", n1 = 4),
           border = "white", lwd = 0.5, legend.pos = "topleft",
           legend.title.txt = "variable x", add = TRUE
)
text(coordinates(spdf_grid_o), row.names(spdf_grid_o), cex = 0.5)
plot(spdf_grid_d, add = T)
choroLayer(spdf = spdf_grid_d, var = "z", nclass = 4,
           col = carto.pal(pal1 = "green.pal", n1 = 4),
           border = "white", lwd = 0.5, legend.pos = "topright",
           legend.title.txt = "variable z", add = TRUE
)
text(coordinates(spdf_grid_d), row.names(spdf_grid_d), cex = 0.5)
```



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

1.1.4.2.1 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 :

```
n_grid_o <- nrow(sp_grid_o)
n_grid_d <- nrow(sp_grid_d)
flows_grid <- data.frame(origin = rep(row.names(sp_grid_o),
                                      each = n_grid_d),
                         dest = rep(row.names(sp_grid_d), n_grid_o),
                       x_o = kronecker(sp_grid_o$x, rep(1, n_grid_d)),
                       z_d = kronecker(rep(1, n_grid_o), sp_grid_d$z))
head(flows_grid)
```

```
##   origin dest x_o z_d
## 1    O_1  D_1  23  50
## 2    O_1  D_2  23  52
## 3    O_1  D_3  23  54
## 4    O_1  D_4  23  49
## 5    O_1  D_5  23  51
## 6    O_1  D_6  23  53
```

1.1.4.2.2 Distances between locations

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

```
G_grid <- rgeos::gDistance(sp_grid_o, sp_grid_d, byid = T)
```

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

```
flows_grid$g <- as.vector(G_grid)
```

1.1.4.2.3 Construct the spatial weight matrices

To define the spatial weight matrix on our geographical sites, we use the contiguity properties for origins and destinations.

```
w_grid_o_nb <- poly2nb(spdf_grid_o)
w_grid_o <- listw2mat(nb2listw(w_grid_o_nb))
w_grid_d_nb <- poly2nb(spdf_grid_d)
w_grid_d <- listw2mat(nb2listw(w_grid_d_nb))
```

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

```
W_grid_d <- kronecker(Diagonal(n_grid_o), w_grid_d)
W_grid_o <- kronecker(w_grid_o, Diagonal(n_grid_d))
W_grid_w <- W_grid_o %*% W_grid_d
```

We prepare the lagged explanatory variables:

```
flows_grid$W_dz_d <- as.matrix(W_grid_d) %*% flows_grid$z_d
flows_grid$W_ox_o <- as.matrix(W_grid_o) %*% flows_grid$x_o
```

1.1.5 DGP of the Y variables

We simulate 9 different SDM interaction models:

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

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

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

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

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

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

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

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

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

$y_o = (Z\delta_{gravity} + \epsilon),$

with $Z = (1_N, X_d, X_o, W_d X_d, W_o X_o, g)$, $\delta = (\alpha, \beta_d, \beta_o, \delta_d, \delta_o, \gamma)$ and $\delta_{gravity} = (\alpha, \beta_d, \beta_o, 0, 0, \gamma)$. We generate a set of flows Y with $\alpha = 0$, $\beta_d = 1$, $\beta_o = 0.5$, $\delta_d = 0.5$, $\delta_o = 0.25$, $\gamma = -2.0$, $\rho_d = 0.4$, $\rho_o = 0.4$, and $\rho_w = -0.16$.

```
delta <- c(0, 1, 0.5, 0.5, 0.25, -2)
rho <- c(0.4, 0.4, -0.16)
```

```
Z_au <- cbind(1, flows_au$x_d, flows_au$x_o,
              as.matrix(W_au_d) %*% flows_au$x_d,
              as.matrix(W_au_o) %*% flows_au$x_o,
              flows_au$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**, an integer value for the seed
- **sigma**, the variance of the residuals
- **message**, print a message to indicate the ratio of sd(noise)/sd(signal).

```
flows_au[, c("y_9", "y_8", "y_7", "y_6", "y_5",
             "y_4", "y_3", "y_2", "y_1", "y_0")] <-
  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)
```

```
## sd(noise)/sd(signal):
## y9:  0.04088024
## y8:  0.03674443
## y7:  0.03778429
## y6:  0.05276569
## y5:  0.05054783
## y4:  0.06883898
## y3:  0.04640652
## y2:  0.056281
## y1:  0.06582714
## y0:  0.07322045
```

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

```
head(flows_au)
```

```
##   origin dest x_o x_d         g   W_dx_d W_ox_o       y_9      y_8
## 1     NT   NT  20  20 0.0000000 21.75000  21.75 127.07957 339.6184
## 2     NT  QLD  20  40 0.6931472 21.25000  21.75 160.09508 375.1901
## 3     NT   WA  20   7 0.8813736 15.00000  21.75  97.00443 305.8809
## 4     NT   SA  20  10 0.6931472 22.40000  21.75 110.71052 323.0426
## 5     NT  NSW  20  30 0.8813736 22.00000  21.75 144.20394 358.6642
## 6     NT  ACT  20  25 1.1743590 28.33333  21.75 145.26238 362.6524
##        y_7      y_6       y_5      y_4       y_3      y_2      y_1
## 1 230.1419 76.37259  76.25340 39.31789  75.87818 76.41378 45.75202
## 2 264.8655 98.72720 100.33954 57.90878 107.36944 96.04838 64.44603
## 3 197.5005 56.01984  54.64739 24.24246  49.44641 58.23100 29.73346
## 4 213.6519 64.59723  63.62979 28.85749  59.41629 66.07393 35.32171
## 5 248.5627 87.59216  88.37243 48.26058  91.72855 86.26428 54.80404
## 6 251.7071 88.16446  88.54845 46.57147  89.64987 87.27157 53.97051
##        y_0
## 1 29.43952
## 2 48.38353
## 3 16.79596
## 4 18.68421
## 5 38.36654
## 6 34.36635
```

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$y_1, n_au, n_au)
Y_au_0 <- matrix(flows_au$y_0, n_au, n_au)
```

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

```
Z_ge <- cbind(1, flows_ge$x_d, flows_ge$x_o,
              as.matrix(W_ge_d) %*% flows_ge$x_d,
              as.matrix(W_ge_o) %*% flows_ge$x_o,
              flows_ge$g)
flows_ge[, c("y_9", "y_8", "y_7", "y_6", "y_5",
             "y_4", "y_3", "y_2", "y_1", "y_0")] <-
  DGP_flow_sdm(z = Z_ge, delta = delta, rho = rho,
               W_d = as.matrix(W_ge_d),
               W_o = as.matrix(W_ge_o),
               W_w = as.matrix(W_ge_w),
               seed = 1234, sigma = 1, message = T)
```

```
## sd(noise)/sd(signal):
## y9:  0.07350916
## y8:  0.05973944
## y7:  0.06405818
## y6:  0.09715192
## y5:  0.09357275
## y4:  0.127838
## y3:  0.08319032
## y2:  0.1043098
## y1:  0.1226104
## y0:  0.1332832
```

```r
Z_usa <- cbind(1, flows_usa$x_d, flows_usa$x_o,
              as.matrix(W_usa_d) %*% flows_usa$x_d,
              as.matrix(W_usa_o) %*% flows_usa$x_o,
              flows_usa$g)
flows_usa[, c("y_9", "y_8", "y_7", "y_6", "y_5",
             "y_4", "y_3", "y_2", "y_1", "y_0")] <-
  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 = 1234, sigma = 1, message = T)
```

```
## sd(noise)/sd(signal):
## y9:  0.1448589
## y8:  0.1053475
## y7:  0.1201223
## y6:  0.1947265
## y5:  0.1872503
## y4:  0.2550643
## y3:  0.1678062
## y2:  0.2036249
## y1:  0.2467603
## y0:  0.2548274
```

```r
Z_grid <- cbind(1, flows_grid$z_d, flows_grid$x_o,
              as.numeric(W_grid_d %*% as(flows_grid$z_d, "sparseVector")),
              as.numeric(W_grid_o %*% as(flows_grid$x_o, "sparseVector")),
              flows_grid$g)
flows_grid[, c("y_9", "y_8", "y_7", "y_6", "y_5",
             "y_4", "y_3", "y_2", "y_1", "y_0")] <-
  DGP_flow_sdm(z = Z_grid, delta = delta, rho = rho,
               W_d = W_grid_d,
               W_o = W_grid_o,
               W_w = W_grid_w,
               seed = 123, sigma = 1, message = T)
```

```
## sd(noise)/sd(signal):
## y9:  0.07472864
## y8:  0.03895307
## y7:  0.04927908
## y6:  0.1127862
## y5:  0.111616
## y4:  0.1972354
## y3:  0.1186835
## y2:  0.1101645
## y1:  0.1746036
## y0:  0.1991334
```

2 Data Vizualization

2.1 List of origins and destinations coincide

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

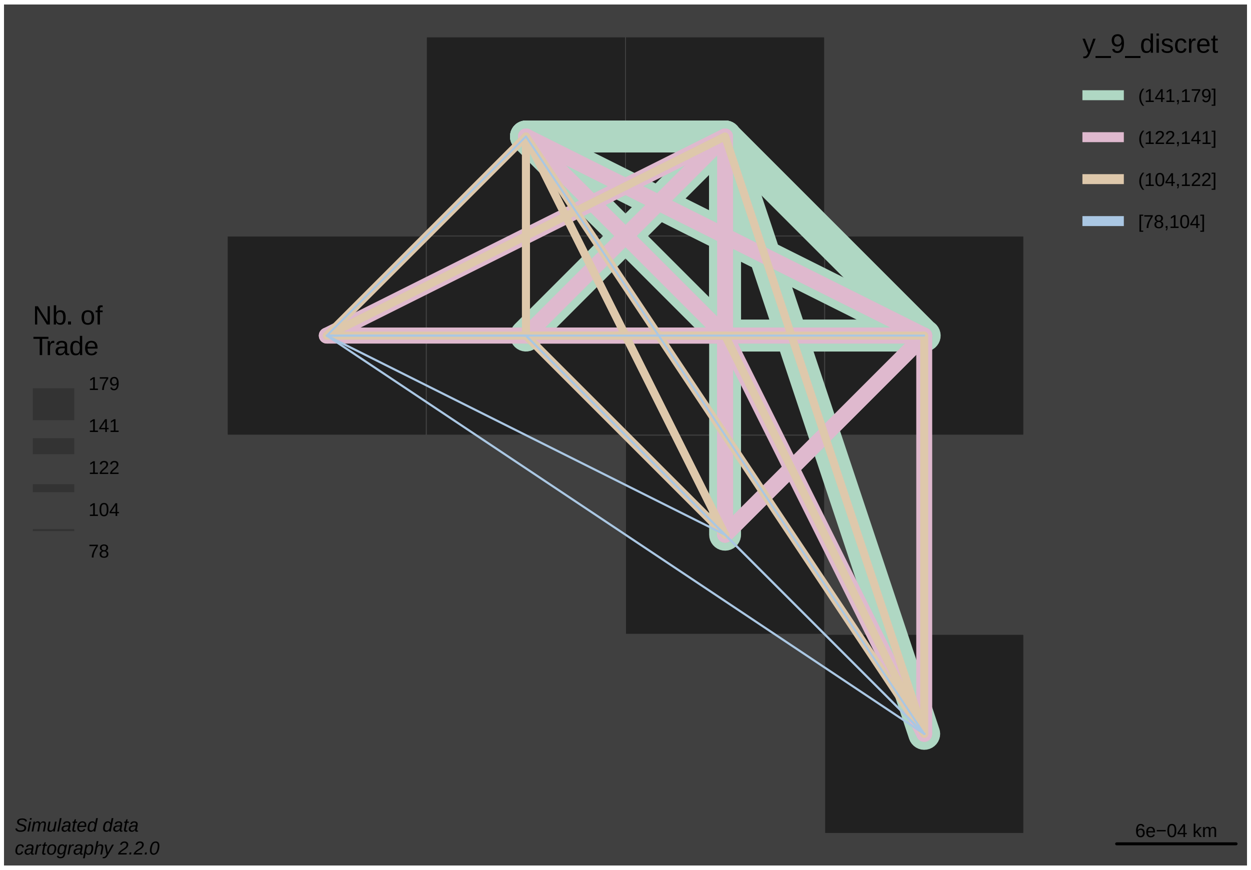
We discretize the variable y_9:

```
breaks <- quantile(flows_au$y_9)
flows_au$y_9_discret <- cut(flows_au$y_9, breaks,
                            include.lowest = T)
```

We plot all the flows.

```
# 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))
# 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 = "y_9",
  breaks = breaks,
  lwd = c(1, 4, 8, 16),
  var2 = "y_9_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)
```



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

2.2 List of origins and destinations do not coincide

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

```
spdf_grid_o_d <- spRbind(spdf_grid_o[, "NOM"], spdf_grid_d[, "NOM"])
```

```
mtq_mob <- getLinkLayer(
  x = spdf_grid_o_d,
  xid = "NOM",
  df = flows_grid,
  dfid = c("origin","dest")
)
```

We discretize the variable y_9:

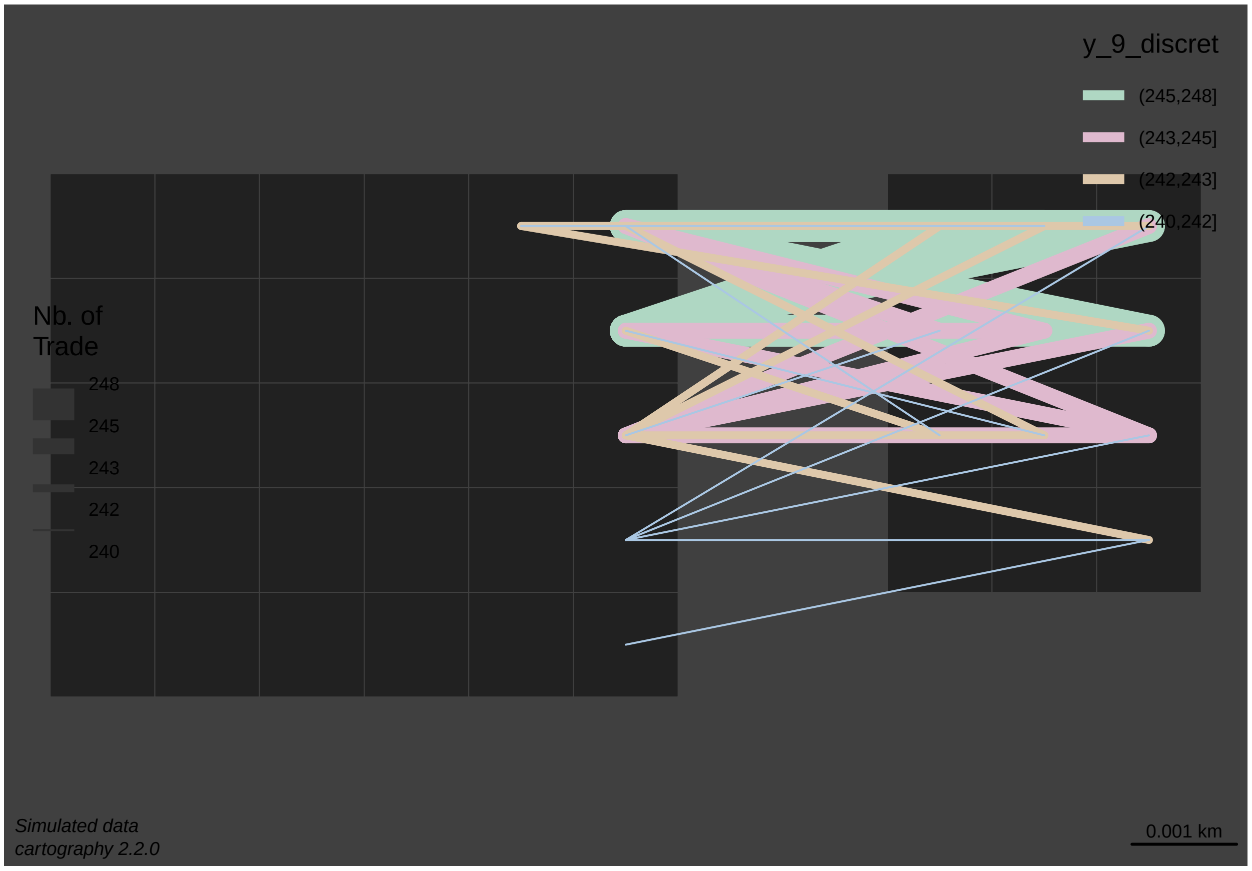
```
breaks <- quantile(flows_grid$y_9, c(0, 0.9, 0.925, 0.95, 0.975, 1))
flows_grid$y_9_discret <- cut(flows_grid$y_9, breaks,
                             include.lowest = T)
```

We plot the largest flows only.

```
par(bg = "grey25", oma = c(0, 0, 0, 0),
    mar = c(0, 0, 0, 0), mai = c(0, 0, 0, 0))
# plot municipalities
plot(spdf_grid_o_d, col = "grey13", border = "grey25",
```

```r
     bg = "grey25", lwd = 0.5)
# plot graduated links
gradLinkTypoLayer(
  x = mtq_mob,
  xid = c("origin","dest"),
  df = flows_grid,
  dfid = c("origin","dest"),
  var = "y_9",
  breaks = breaks[2:6],
  lwd = c(1, 4, 8, 16),
  var2 = "y_9_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 #

2.3 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.

2.4 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. We call this function *gravity_model()*. It takes as input arguments :

```
gravity_model(x, Y, G, ind_d = NULL, ind_o = NULL)
```

- **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.

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

2.5 Applications

2.5.1 With the Australian simulated data

```
gravity_model(x = as.matrix(spdf_au@data$x), Y = Y_au_0, G = G_au)
```

```
##                   [,1]
## (Intercept) 29.4261068
## _d           0.9847713
## _o           0.5195392
## g           -1.9439991
```

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

```
gravity_flows <- lm(y_0 ~ x_d + x_o + g, data = flows_au)
```

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_0 ~ x_d + x_o + g, data = flows_au)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.62892 -0.63796  0.00185  0.59954  1.86723
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.09599    0.41965  -0.229     0.82
## x_d          0.98477    0.01031  95.510  < 2e-16 ***
## x_o          0.51954    0.01031  50.389  < 2e-16 ***
## g           -1.94400    0.26686  -7.285 8.17e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8764 on 60 degrees of freedom
## Multiple R-squared:  0.9951, Adjusted R-squared:  0.9949
## F-statistic:  4060 on 3 and 60 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.

2.5.2 Comparison of computationnal times according to size

```
##     matrix vector
## au   0.001  0.001
## ge   0.001  0.002
## usa  0.001  0.002
```

3 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 :

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

3.1 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, $\ldots$, 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.

3.1.1 Traces of the spatial weight matrix

We first 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.

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 0.6785355 1.3853101 0.8496334 1.1475253 0.9279852
##  [8] 1.0609386 0.9653729 1.0261889
```

3.1.2 Computation of the determinant

3.1.2.1 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, ...,W^{miter},
- **n**, the sample size.

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

3.1.2.2 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, ...

```
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] -0.5962978
```

3.1.2.3 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] -2.006454
```

3.1.3 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")
```

3.1.4 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.

3.1.5 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 = "")
```

3.1.6 Application to the toy data

3.1.6.1 Model 2

3.1.6.1.1 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,
                                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.

```
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
```

3.1.6.1.2 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).

```
result_lagsarlm <- lagsarlm(y_2 ~ x_d + W_dx_d + x_o + W_ox_o + g,
                            data = flows_au,
                            mat2listw(W_au_d))
```

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

```
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
## -1.636859 -0.528004 -0.054146  0.584378  1.799559
##
## Type: lag
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value  Pr(>|z|)
## (Intercept) -3.666198   2.359996 -1.5535    0.1203
## x_d          0.966462   0.013636 70.8754 < 2.2e-16
## W_dx_d       0.447323   0.074688  5.9892 2.108e-09
## x_o          0.430871   0.068250  6.3131 2.734e-10
## W_ox_o       0.199988   0.045733  4.3730 1.226e-05
## g           -1.780904   0.263705 -6.7534 1.444e-11
##
## Rho: 0.50241, LR test value: 26.667, p-value: 2.4172e-07
## Asymptotic standard error: 0.076827
##     z-value: 6.5396, p-value: 6.1688e-11
## Wald statistic: 42.766, p-value: 6.1688e-11
##
## Log likelihood: -80.35319 for lag model
## ML residual variance (sigma squared): 0.66214, (sigma: 0.81372)
## Number of observations: 64
## Number of parameters estimated: 8
## AIC: 176.71, (AIC for lm: 201.37)
## LM test for residual autocorrelation
## test value: 3.4995, p-value: 0.061387
```

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   0.281
## ge   0.395
## usa  7.654
```

3.1.6.1.3 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 + W_dx_d + x_o + W_ox_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", "W_dx_d")],
                              x_o = flows_au[, c("x_o", "W_ox_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      W_dx_d         x_o      W_ox_o           g
##  -7.8128579   0.9511585   0.3124142   0.3016425   0.1240235  -1.5979115
##
## $RHO
##     rho_d
## 0.6520422
##
## $SIGMA
## 1 x 1 Matrix of class "dgeMatrix"
##           [,1]
## [1,] 0.6494528
##
## $sd_beta
## (intercept)         x_d      W_dx_d         x_o      W_ox_o           g
##   -2.618070   61.979502    3.195325    3.350657    2.061499   -5.733580
##
## $sd_rho
##    rho_d
## 6.303734
```

Computational time with respect to the size of the samples:

```
##     vector
## au   0.006
## ge   0.012
## usa  0.123
```

3.1.6.2 Model 9

3.1.6.2.1 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", "W_dx_d",
                                                                "x_o", "W_ox_o")],
                                     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
```

3.1.6.2.2 S2SLS estimation

```
(sar_simu_9_sls <- s2sls_flow(x_d = flows_au[, c("x_d", "W_dx_d")],
                              x_o = flows_au[, c("x_o", "W_ox_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      W_dx_d         x_o      W_ox_o           g
##  3.87995455  0.25504144  0.14255695  0.10409143  0.03816427 -1.17880828
##
## $RHO
##      rho_d      rho_o      rho_w
##  0.8880304  0.8456573 -0.8453386
##
## $SIGMA
## 1 x 1 Matrix of class "dgeMatrix"
##          [,1]
## [1,] 0.6822863
##
## $sd_beta
## (intercept)         x_d      W_dx_d         x_o      W_ox_o           g
## 236.7001213   0.5419036   0.7038231   0.7001330   0.3823234  -3.2274324
##
## $sd_rho
##     rho_d     rho_o     rho_w
##  4.895752  2.866186 -4.654285
```

4 Interpreting the results

4.1 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 %*% 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 :

```r
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 :

```r
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
```

4.2 Computation

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

4.2.1 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))
```

4.3 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 $A(W)$,

4.3.1 Origin effect

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

To get the origin effect:

```
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
```

4.3.2 Destination effect

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

To get the destination effect:

```
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
```

4.3.3 NE effect

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

To get the NE effect:

```
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
```

4.3.4 intra effect

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

To get the intra effect:

```
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
```

4.3.5 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.