Modelling spatial spread I

Andrea Parisi

Main contact for this notebook: Andrea Parisi (andrea.parisi@warwick.ac.uk)

Course context

Purpose and scope of the course

This material has been developed as part of the *GeMVi* project: *NIHR Global Health Research Group on the Application of Genomics and Modelling to the Control of Virus Pathogens* in East Africa and the University of Warwick.

In these workshops you will be introduced to some common techniques used in infectious disease modelling. The topics covered will include the implementation of deterministic and stochastic compartmental models, the use of maximum likelihood estimation to analyse super-spreading behaviour in novel disease outbreaks, modelling of contact patterns, and optimisation techniques for fitting epidemic models to real data.

Contributors

The contributors are all part of the GeMVi project:

- Prof. James Nokes (JNokes@kemri-wellcome.org)
- Prof. Matt Keeling (m.j.keeling@warwick.ac.uk)
- Dr. Joe Hilton (j.hilton@warwick.ac.uk)
- Dr. Rabia Aziza (rabia.aziza@warwick.ac.uk)
- Dr. Samuel Brand (s.brand@warwick.ac.uk)
- Dr. Andrea Parisi (andrea.parisi@warwick.ac.uk)

Helpful references for the course

- Anderson, R. M., & May, R. M. (1992). Infectious Diseases of Humans: Dynamics and Control.
- Bjørnstad, O. N. (2018). Epidemics, models and data using R. https://doi.org/10.1007/978-3-319-97487-3
- Diekmann, O., & Heesterbeek, J. A. P. (2000). Mathematical epidemiology of infectious diseases: model building, analysis and interpretation. 104: John Wiley and Sons.
- Keeling, Matt J., & Pejman Rohani. Modeling Infectious Diseases in Humans and Animals. Princeton University Press, 2008.
- Martcheva, M., 2015. An Introduction to Mathematical Epidemiology, Texts in Applied Mathematics. Springer US, Boston, MA. https://doi.org/10.1007/978-1-4899-7612-3
- Vynnycky, Emilia, & White, Richard G. An Introduction to Infectious Disease Modelling

Requirements

The following packages are required for this tutorial. Please have them installed before the start.

1. pracma - *Practical Numerical Math Routines* implements a number of mathematical functions, a couple of which are used in this tutorial.
2. deSolve - *Differential Equation solver* implements methods to integrate initial value differential equations
3. viridis - *Viridis colour palette* implements the viridis colour palettes, a set of colour palettes that provide perceptually uniform colours, are appropriate for color blindness and render properly in black and white. Particularly useful for visualising maps.
4. sf - *Simple Features* is a package useful to handle GIS (Geographic Information Systems) datasets.
5. lwgeom [optional] - This is an optional package and is not required for this tutorial. However, it may be needed to handle some vector maps.

The packages may be installed using (uncomment as needed and copy-paste into the console):

```
#install.packages("pracma", "deSolve", "viridis", "sf")
#install.packages("lwgeom")
```

Introduction

Infectious diseases spread within a population by transmission from infected to susceptible individuals. Once an individual is infected, he/she will carry the disease around and transmit it to other individuals while moving around in the neighbourhood, in the wider community, to other countries.

Different diseases transmit through different transmission routes. The spread of sexually transmitted diseases is driven by transmission between sexual partners, ebola is spread through contact with body fluids, cholera through contaminated food or water typically associated with sanitation issues, influenza by airborne transmission in close proximity or hand-to-nose/mouth transmission. In this workshop we will focus on infections which transmit through direct contact such as influenza, ebola, coronavirus, or tubercolosis.

For infectious diseases transmitted by direct contact, spatial spread is driven by the movement of individuals, hence understanding how people move is key to understanding how infectious diseases, and how they can be controlled. Although spatial spread is also relevant to vector-borne diseases and those which spread through contaminated food and water, the impact of spatial structure in these cases is more complicated than in the examples we will consider here.

Metapopulation models

Metapopulation models partition a population of size N into a set of interacting sub-populations, each of size N_j where $\sum_j N_j = N$. Each sub-population may represent a different county, district, or disease-specific risk area; the way a population is partitioned into sub-populations in a model is typically driven by data availability. Suppose we wish to describe the epidemiological evolution of a disease in each subpopulation. We can build a model for each subpopulation and then couple all subpopulations. There are different ways to achieve this goal depending on the level of detail required. A simple approach is to assume that a fraction of each subpopulation, due to travel, enters into contact with other subpopulation at a rate that reflects how individuals travel to different locations. For the full population of size N with per-capita birth/death rate μ (recall that with both rates equal, the population size is constant), the SIR model can be written down as:

$$\begin{aligned}\frac{dS}{dt} &= \mu N - \beta\frac{S(t)I(t)}{N} - \mu S(t)\\ \frac{dI}{dt} &= \beta\frac{S(t)I(t)}{N} - \gamma I(t) - \mu I(t).\end{aligned}$$

We can modify these equations to get the dynamics of the ith subpopulation as follows:

$$\begin{aligned}\frac{dS_i}{dt} &= \mu(N_i - S_i(t)) - \beta_i\sum_j \rho_{ij}\frac{S_i(t)I_j(t)}{N_i}\\ \frac{dI_i}{dt} &= \beta_i\sum_j \rho_{ij}\frac{S_i(t)I_j(t)}{N_i} - (\gamma+\mu)I_i(t).\end{aligned} \tag{1}$$

The parameter β_i is the person-to-person transmission rate within subpopulation i, whereas the *spatial contact matrix* ρ_{ij} represents the coupling *to* subpopulation i *from* subpopulation j. The underlying assumption here is that individuals from j will visit i and participate to the local dynamics. The simplest interpretation of ρ_{ij} is that it is the fraction of the population in j that will visit i. It follows that ρ_{jj} is the fraction of the population in j that will stay in location j, and thus $\sum_i \rho_{ij} = 1$. In practice, the contact matrix should also into account the duration of stay, since visitors who stay in a subpopulation for a longer duration have more chance to generate infections. For instance, if $\bar{\rho}_{ij}$ is the number individuals daily commuting from work from j to i, and τ is the duration of stay in daily fractions, then $\rho_{ij} = \bar{\rho}_{ij}\tau \quad \forall j \neq i$. We will discuss this in more detail later when dealing with a practical example.

In order to build a model that can be integrated efficiently, we write the equations in eq. (1) using vector notation. First, we introduce the force of infection in subpopulation i defined as:

$$\lambda_i(t) = \beta_i\sum_j \rho_{ij}I_j(t)$$

so that the set of equations in (1) takes the form:

$$\begin{aligned}\frac{dS_i}{dt} &= \mu(N_i - S_i(t)) - \lambda_i(t)\frac{S_i(t)}{N_i} \\ \frac{dI_i}{dt} &= \lambda_i(t)\frac{S_i(t)}{N_i} - (\gamma+\mu)I_i(t)\end{aligned}$$

We now use the following notation: we denote elementwise multiplication with $\otimes$, and elementwise division with $\oslash$. Using this notation, the force of infection takes the form $\vec{\lambda} = \vec{\beta} \otimes \boldsymbol{\rho} \cdot \vec{I}$, and it becomes straightforward to write down the model as:

$$\begin{aligned}\frac{d\vec{S}}{dt} &= \mu(\vec{N} - \vec{S}(t)) - \vec{\lambda}(t) \otimes \vec{S}(t) \oslash \vec{N} \\ \frac{d\vec{I}}{dt} &= \vec{\lambda}(t) \otimes \vec{S}(t) \oslash \vec{N} - (\gamma+\mu)\vec{I}(t)\end{aligned} \tag{2}$$

Example model

In this example we will build a generic metapopulation model using R. We first load the set of required libraries:

```r
library(pracma)
library(deSolve)
```

```
##
## Attaching package: 'deSolve'

## The following object is masked from 'package:pracma':
##
##     rk4
```

```r
library(viridis)
```

```
## Loading required package: viridisLite
```

We use the ode function from the `deSolve` library to integrate the metapopulation model. The function takes 5 arguments: 1) an initial condition, 2) a set of times on which we want the integrated model to be evaluated, 3) the ode model to be integrated, 4) a set of parameters that may be used in the evaluation of (3), and finally 5) an integration method.

```
res <- deSolve::ode( xstart, times, sir.model, params, method=deSolve::rk4 )
```

The core of the model is a function that takes as arguments a sequence of times, a set of variables (susceptibles and infectives in each sub-population or `patch`), and a list of parameters, and evaluates the right-hand side of equation (2).

```r
sir.model <- function( t, x, params )  {
    patches <- params[[1]]    # This is the number of sub-populations
    mu <- params[[2]]         # Birth/death rate
    gamma <- params[[3]]      # Recovery rate
    beta <- params[[4]]       # Common transmission rate
    rho <- params[[5]]        # Contact matrix
    NN  <- params[[6]]        # Sub-population sizes

    # Extract the susceptibles and infectives
    pop.sus <- x[1:patches]
    pop.inf <- x[(patches+1):(2*patches)]
    lambda <- beta * rho %*% pop.inf  # Matrix multiplication

    # Find rhs of (1) using vector calculus
    ds <- mu*(NN-pop.sus)-lambda*pop.sus/NN
    di <- +lambda*pop.sus/NN - (mu+gamma)*pop.inf
    res <- c(ds,di)
    return(list(res))
}
```

Now we can build an instance of the metapopulation model: we use `patches` subpopulations, each patch `jj` having a population size `NN[jj]`. For this example we assume 16 patches of 10,000 individuals each. Thus:

```r
patches <- 16
NN <- rep(10000, patches)
```

Let us use a set of parameters typical of flu.

```r
mu <- 1.0/(70*365)
gamma <- 1./2.2
beta  <- rep(1.66, patches)
```

Note that we defined `beta` as a vector following eq. (2). However, if all β_i have the same value, one could define `beta` as a single number instead of a vector, and the formula inside the `sir.model` R function would still work correct. Thus, from now on, whenever the β_i have all the same value, we will assign to `beta` a single value.

We need now to build the matrix describing the coupling between patches. For the moment, we build a random matrix, constrained to have $\sum_i \rho_{ij} = 1$.

```r
rho <- matrix(2*runif(patches*patches), nrow=patches, ncol=patches) / patches
# Diagonal elements represent individuals not moving
# and are set by applying the constraint on rho
for (jj in 1:nrow(rho))  {
    rho[jj,jj] <- 0
}
```

```r
# Make sure that sum_i rho_ij = 1
for (jj in 1:ncol(rho))  {
    if (sum(rho[,jj]) > 1)  {
        rho[,jj] <- rho[,jj] / sum(rho[,jj])
    }
    rho[jj,jj] <- 1-sum(rho[,jj])
}
```

We can now group all parameters in the list params:

```r
params <- list(patches, mu, gamma, beta, rho, NN)
```

We are almost done: we need a sequence of times and an initial condition (ten infectives in patch 1).

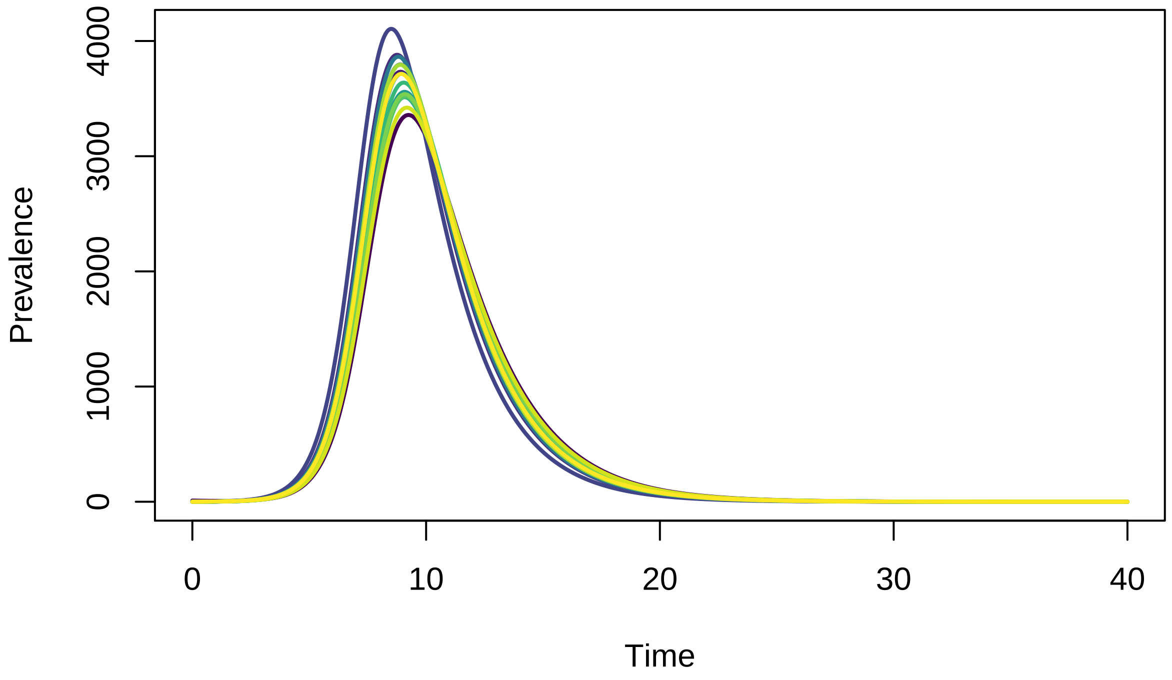
```r
times <- seq( from = 0, to = 40, by = 0.05)
start.I0 <- rep(0, patches)
start.I0[1] <- 10
start.S0 <- NN-start.I0
xstart <- c( start.S0, start.I0 )
```

We are ready now to integrate the model

```r
res <- deSolve::ode( xstart, times, sir.model, params, method=deSolve::rk4 )
```

Visualization of results: here we focus on infectives in each patch. We introduce the function showOutput that plots the number of infectives as a function of time in all patches. We will use it here and later on.

```r
output <- data.frame( res )

showOutput <- function(out, nregions)  {
    x.min <- min(out$time)
    x.max <- max(out$time)
    y.min <- 0
    y.max <- max(out[,(2+nregions):ncol(out)])
    plot( c(x.min, x.max), c(y.min, y.max), type="n", col=3, lwd=2,
          ylim=c(0,y.max), xlab="Time", ylab="Prevalence" )
    for (ii in 1:nregions)  {
        lines( out$time, out[,(1+nregions+ii)], lwd=2, col=viridis(nregions)[ii] )
    }
}

showOutput(output, patches)
```



It is evident that there is some variability in the epidemic between distinct patches. It might not seem much, but this is a simplistic model: same population size in every patch and, most of all, contact matrix completely random. This means that even if one connection is weak, there are other similar connections that will compensate.

Exercises

- *Exercise 1*: Increase the number of patches and observe how the infection curves will tend to overlap when such number becomes large. Can you explain why?
- *Exercise 2*: Increase the population size of each patch by multiplying them by the same factor. What happens to the infection curves? Why?

The contact matrix

Is it realistic to use a random contact matrix? Certainly not. To clarify, let us add a bit more realism to the model. Let us assume that each of the patches introduced above represents an administrative subdivision of a country. Let us imagine the country of Squareland which has the shape of a square, and whose territory is divided in the above $4^2 = 16$ patches that have the same population of 10,000 inhabitants. Just to make things clear, let us associate to each patch a set

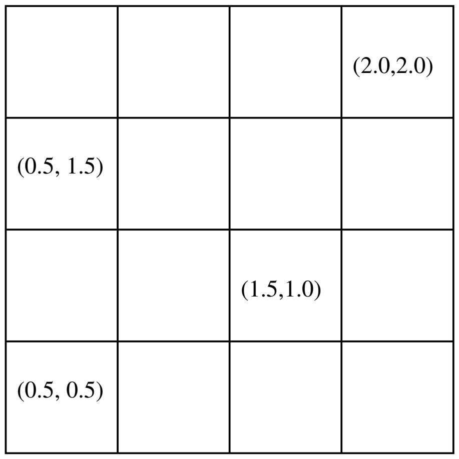


Figure 1: A representation of the country of Squareland: each subpopulation is located in space with a corresponding latitude and longitude

of coordinates (like latitude and longitude, see Fig.1). Our country (by chance) lives just about the Equator, off the south coast of West Africa, over 2 degrees of latitude and 2 of longitude:

```
side <- 4
# Latitude, longitude for each patch
coordinates <- matrix(0, nrow=patches, ncol=2)
entry <- 1
for (lat in 1:side)  {
    for (lon in 1:side)  {
        coordinates[entry,] <- c(2.0*lat/side, 2.0*lon/side)
        entry <- entry+1
    }
}
```

Let us focus on one of the subpopulations, and check what the couplings with the other patches are. There are 16 entries for each patch:

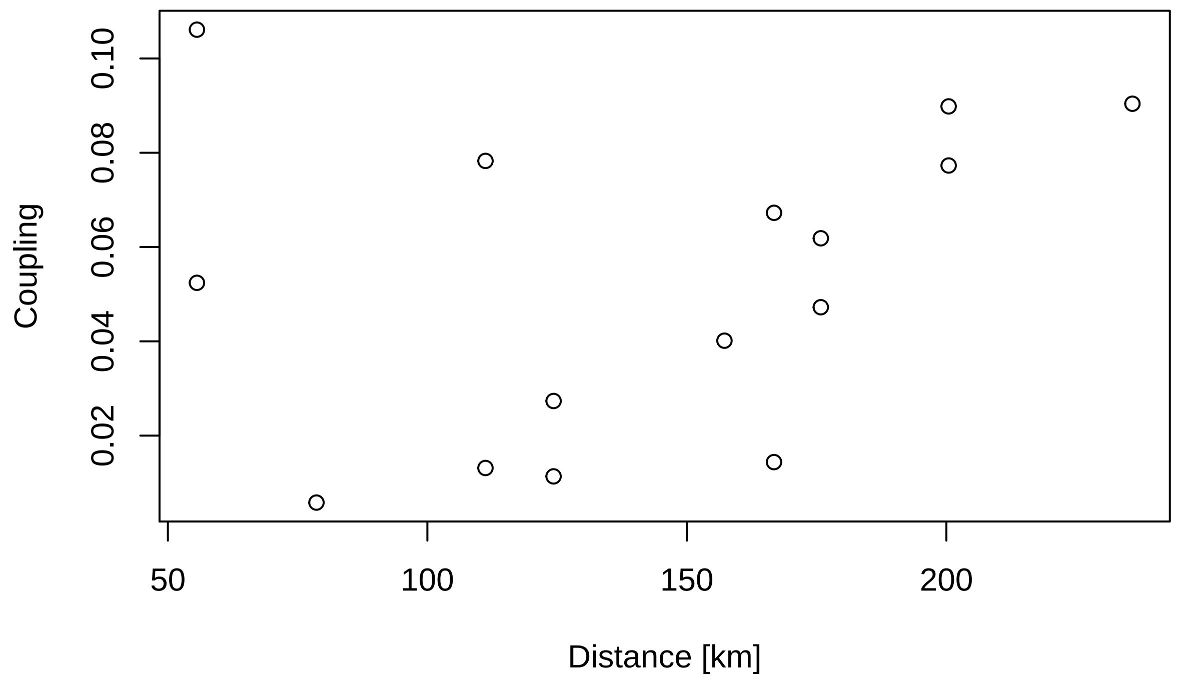
```
rho[1,]
```

```
##  [1] 0.000000000 0.106105620 0.013135670 0.014386046 0.052415095
##  [6] 0.005800891 0.011357920 0.047237250 0.078270062 0.027345532
## [11] 0.040135866 0.089832724 0.067262673 0.061856784 0.077296593
## [16] 0.090395776
```

Let us now visualize the relationship between the coupling strength and distances. We calculate distances using the haversine function from the `pracma` package, which calculates the distance between two points on the surface of Earth, given their latitudes and longitudes:

```
dists <- apply(coordinates, 1, function(x) {haversine(coordinates[1,], x)})
# Only consider moving between distinct sub-populations
dists <- dists[ 2:length(dists) ]
```

```r
coupl <- rho[1,2:nrow(rho)]
plot( dists, coupl, ty='p', ylab="Coupling", xlab = "Distance [km]")
```



It is evident that distance plays no role here. However, this is unreasonable, as people will rarely move far away from their home residence. Indeed, how people move has been the subject of intense research during the last 10-15 years, and we will see some basic features of human mobility soon. Before discussing this however, let us try a simple model where we put some limits on the movement of individual. Let us now assume that nearest neighbour patches have some modest coupling, whereas coupling between more distant patches is negligible (Fig 2). This simple approach has been one of the first attempts to model spatial transmission.

Our coupling matrix will be as follows:

```r
entry <- 1
rho3 <- zeros(patches) # Generates a patches x patches matrix of zeros
for (dst in 1:patches)  {
    for (src in 1:patches)  {
        if (dst == src+side || dst == src-side ||
                    dst == src+1 || dst == src-1)  {
            rho3[dst,src] <- 0.01
        }
    }
```

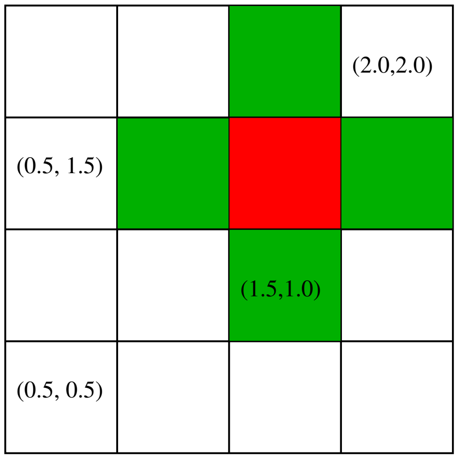
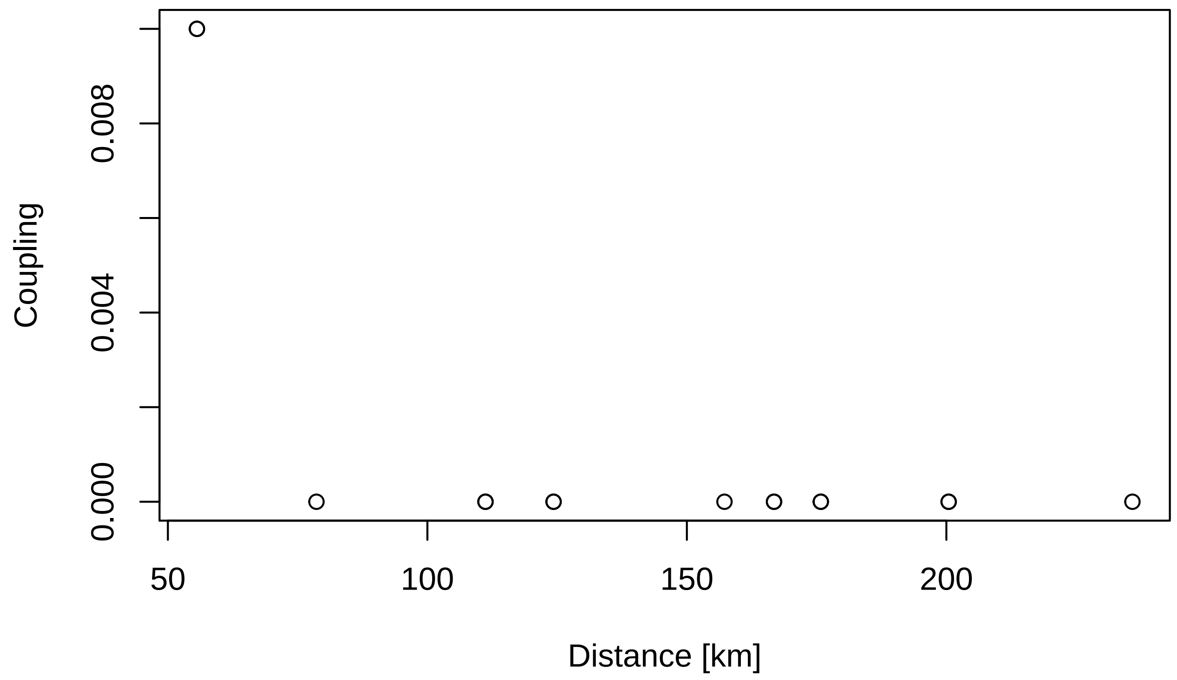


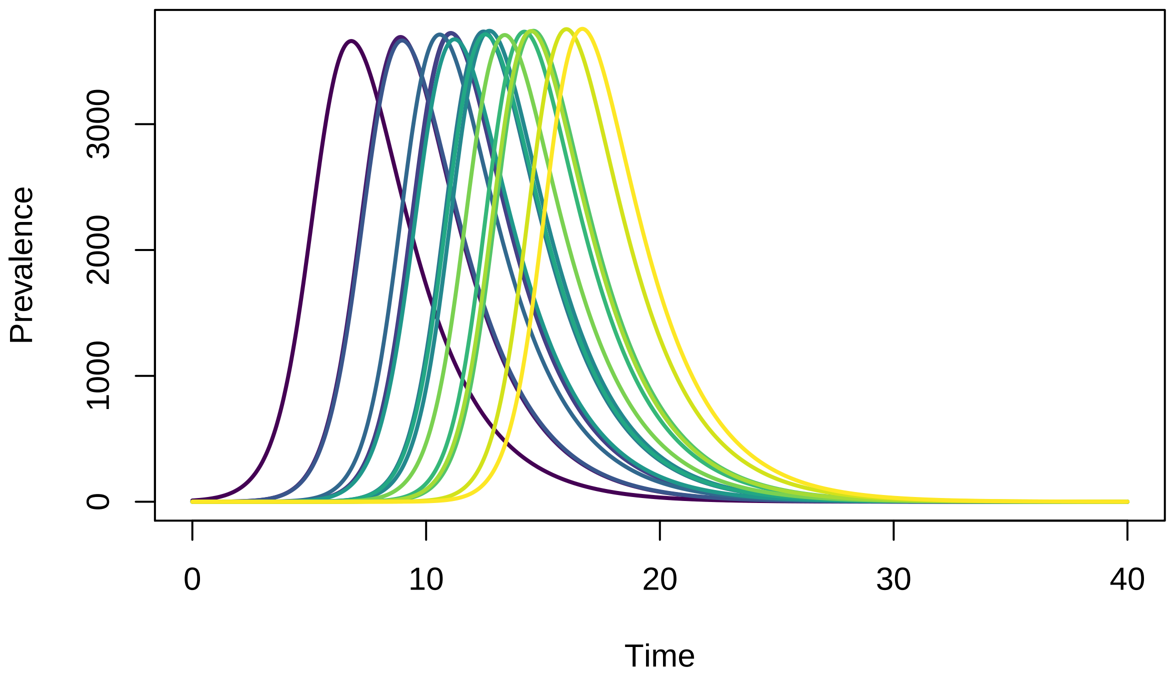
Figure 2: Simple model for mobility: a small coupling between nearest-neighbouring subpopulations and zero elsewhere.

```
}
for (jj in 1:patches)  {
    rho3[jj,jj] = 1-sum(rho3[,jj])
}


dists <- apply(coordinates, 1, function(x) {haversine(coordinates[1,], x)})
# Only consider commuting, exclude those that do not move
dists <- dists[ 2:length(dists) ]
coupl <- rho3[1,2:nrow(rho)]
plot( dists, coupl, ty='p', ylab="Coupling", xlab = "Distance [km]")
```



The consequence of this drastic approach are shown here:

```
beta <- 1.66
times <- seq( from = 0, to = 40, by = 0.05)
start.I0 <- rep(0, patches)
start.I0[1] <- 10
start.S0 <- NN-start.I0
xstart <- c( start.S0, start.I0 )

params <- list(patches, mu, gamma, beta, rho3, NN)
res <- ode( xstart, times, sir.model, params, method=deSolve::rk4 )
output <- data.frame(res)
showOutput(output, patches)
```



Because transmission occurs only through nearest neighbours counties, transmission between distant patches is delayed and the epidemic peaks in different counties at different times.

Exercises

- *Exercise 3* - What happens if a small next nearest-neighbours interaction is added? (By next nearest-neighbours interaction, we mean a small amount of contact/transmission between populations which have a neighbour in common, but which are not directly connected)

An example using real data

In the examples so far we have used illustrative "toy" data. We will turn to a realistic example. We will use data from the US which is readily available. The major difficulty we face here is that data typically does not come in a format ready for use for our purposes. In this case, for instance, we will have to deal with three data sets: a first data set containing the shapes of the counties we wish to work with, a data set detailing the population size of each county in the US, and a data set detailing data on commuting between counties for the whole US. We will need to connect these three data bases and then use them to inform our model.

We start by loading the relevant libraries. The sf library provides tools for visualization of shapefiles (a vector data format for GIS software [Geographic Information System]). The library `lwgeom` is required to handle some shapefiles: it will not be needed here. The viridis palette is with us again!

```r
library(sf)
```

```
## Linking to GEOS 3.6.2, GDAL 2.2.3, PROJ 4.9.3
```

```r
#library(lwgeom)  # Required by some shapefiles
library(viridis)
```

Let us load a map of California counties, in the US

```r
counties <- st_read("Data/CA_Counties/CA_Counties_TIGER2016.shp")
```

```
## Reading layer `CA_Counties_TIGER2016' from data source `/home/parisia/Cabinet/Work/Te
## Simple feature collection with 58 features and 17 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -13857270 ymin: 3832931 xmax: -12705030 ymax: 5162404
## CRS:            3857
```

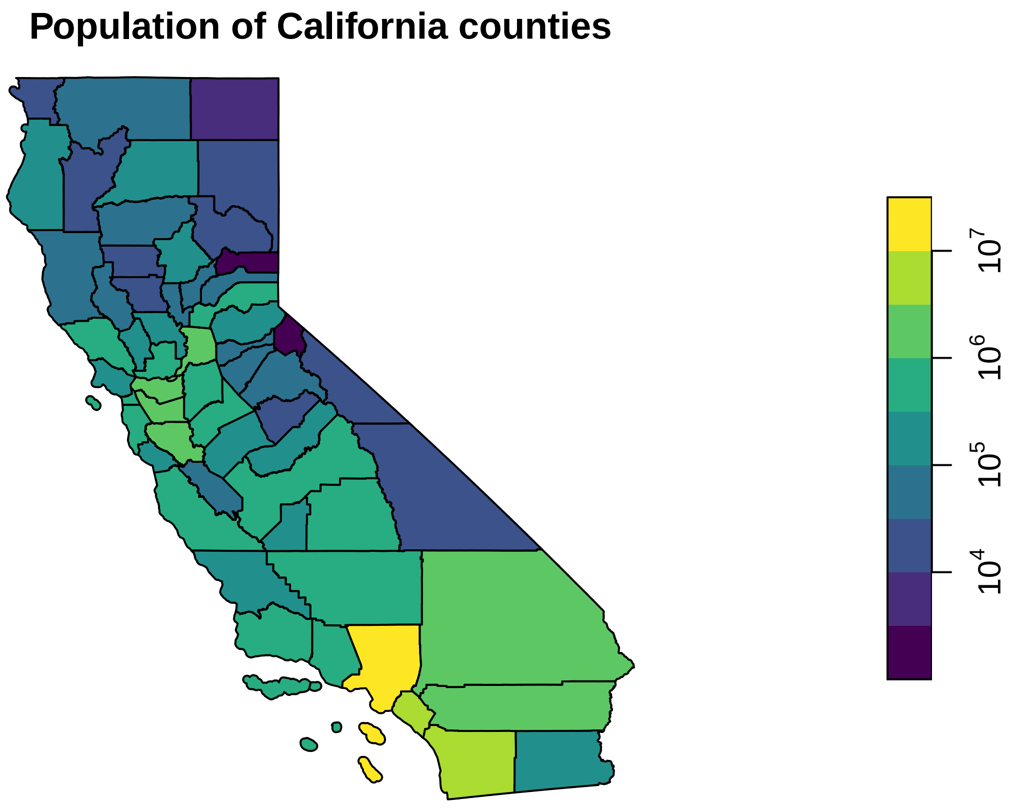
Maps in GIS formats are stored using a projection/coordinates reference system (CRS) chosen by the author of the map. There are thousands of different system of coordinates, so it is always useful to move to a selected coordinate reference system on which we know how to work. A good and simple system of coordinates is the one that uses latitude and longitude, we thus transform the map into this coordinate reference system with the command:

```r
counties.wgs84 <- st_transform( counties, crs="+proj=longlat +datum=WGS84 +no_defs")
```

The WGS84 acronym refers to a standard geodetic curvature: the Earth is not spherical, so this acronym tells the transformation routine how to correctly describe its curvature. Finally, the `no_defs` option instructs the library that performs the transformation not to overwrite map features with default values that the library assumes. Let us now read the population size of the counties. The file that is going to be loaded has the population size of all counties in the US, so we select those of California, and add the data to our map `data.frame`. We then visualize it using a logarithmic scaling

```r
counties.pop <- read.csv("Data/CA_counties_population.csv", header=TRUE)
# Include only California counties. STNAME holds the 50 state names
counties.pop <- counties.pop[ counties.pop$STNAME == "California", ]
# Exclude California itself which is NOT a county. CTYNAME holds County names
counties.pop <- counties.pop[ counties.pop$CTYNAME != "California", ]
#edit(counties.pop)  # To inspect what is inside

# Simplify the data.frame to just 2 columns
counties.pop <- data.frame( CTYNAME=counties.pop$CTYNAME,
                            pop=as.numeric(counties.pop$POPESTIMATE2016) )
```

```r
#edit(counties.pop)

# Merge population size to the main data.frame.
# NAMELSAD holds county names in the main data base
counties.wgs84 <- merge(counties.wgs84, counties.pop, by.x = 'NAMELSAD',
                        by.y="CTYNAME", all.x=FALSE, all.y=FALSE)
plot(counties.wgs84['pop'], pal=viridis(9), logz=TRUE,
     main="Population of California counties")
```



In order to deal with commuting data and connect it to the counties of the map above, it is useful to associate an id to each county (1,2,. . .). First, we force an ordering of counties that can be applied to all our data sets; then we associate an id, and store the resulting mapping in an ad-hoc data frame that can be used as a look-up dictionary. We do the same for the population data set.

```r
counties.wgs84 <- counties.wgs84[ order(counties.wgs84$NAMELSAD), ]
counties.wgs84$id <- seq(1,nrow(counties.wgs84))

# We add the same IDs to the population data.frame
counties.pop <- counties.pop[ order(counties.pop$CTYNAME), ]
counties.pop$id <- seq(1,nrow(counties.pop))
#print(counties.pop)
```

```r
# We can also build a mapping between IDs and county names
counties.mapper <- data.frame( name=counties.wgs84$NAME, id=counties.wgs84$id )
```

Now we load the commuting data and reference the entries by the `id` just introduced. The place of residence will get a `src` id, and the place of work a `dst` id:

```r
comm <- read.csv("Data/Vital_Signs__Commute_Patterns_-_Bay_Area.csv",
                 header=TRUE)
#edit(comm)
comm <- merge(comm, counties.mapper, by.x = 'Res_Geo_Short', by.y = 'name')
comm$src <- comm$id
comm$id <- NULL
comm <- merge(comm, counties.mapper, by.x = 'Work_Geo_Short', by.y = 'name')
comm$dst <- comm$id
comm$id <- NULL
```

The advantage of this is that we can now identify vector and matrix entries in eq.(2) with the `id` variable.

We are now ready to implement this more realistic model. First let us build the coupling matrix ρ_{ij}. To do this we scan the `comm` data frame and set the values of ρ_{ij} appropriately. Typically there will be missing entries: if no entry is available for the number of commuters between location j and i, that means that there were none. We introduce a function to do the job, that takes as argument the column name from which we wish to extract the values to place in ρ_{ij}. This will allow us to build contact matrices from other sources later on. In this approach we also take into account how long individuals are going to spend at their destination. Since these are home-work commuting data, individuals will only spend a certain amount of time at their destination. We are thus taking into account the average amount of work hours spent at destination.

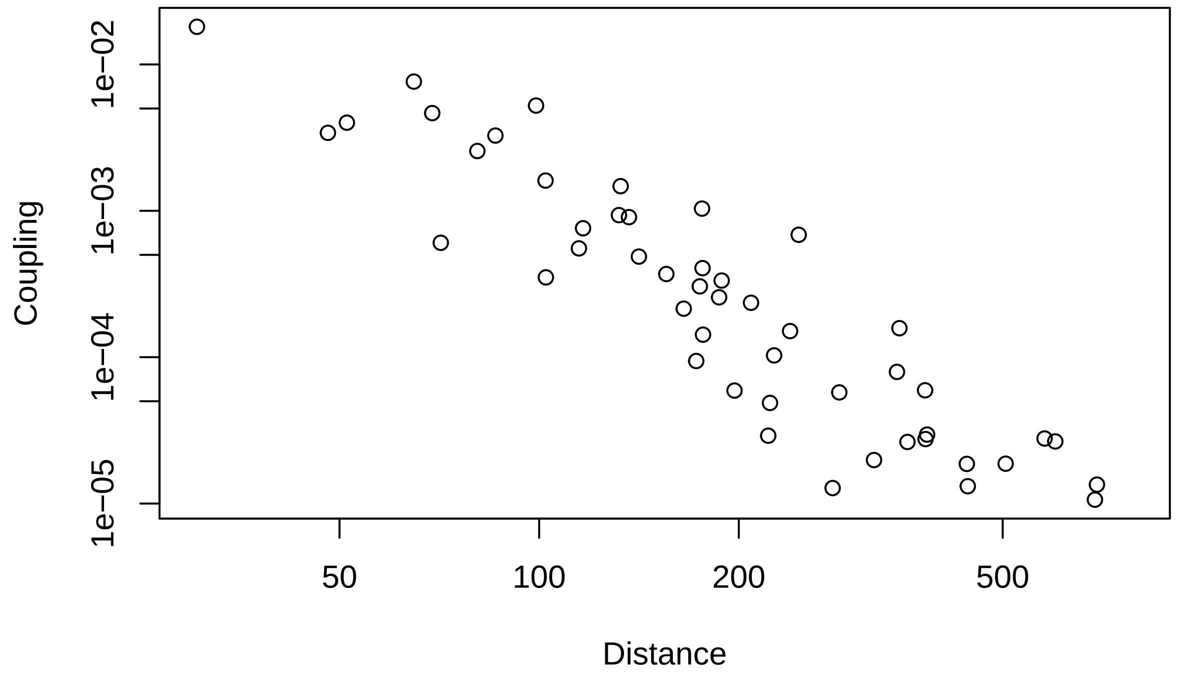
```r
build.contact.matrix <- function( map, comm, colname, spenthours_per_week = 40.0)  {
    npatches <- nrow(map)
    mm <- zeros(npatches) # Generates a patches x patches matrix of zeros
    for (dst in 1:npatches)  {
        for (src in 1:npatches)  {
            if (dst == src)  {
                # Entries rho_{jj} will be handled separately
                next
            } else {
                hasentry <- nrow(comm[ comm$dst == dst & comm$src == src, ])
                if (hasentry > 1)  {
                    print("Too many entries for src-dst data")
                    quit()
                } else if (hasentry)  {
                    entry <- comm[ comm$dst == dst & comm$src == src, ]
                    mm[dst,src] <- entry[,colname]
                    mm[dst,src] <- mm[dst,src] / map[ map$id == src, ]$pop
                    mm[dst,src] <- mm[dst,src]*spenthours_per_week/(7*24.0)
                }
            }
        }
    }
    for (src in 1:npatches)  {
        tot <- sum(mm[,src])
        mm[src,src] <- 1 - tot
    }

    ## Here a sanity check:
    if (min(mm) < 0)  {
        stop( "Sanity check on rho did not work!")
    }
    return (mm)
}

rho <- build.contact.matrix( counties.wgs84, comm, "Total", 34 )
patches <- nrow(rho)
```

We can plot the coupling as a function of distance and compare with the case of the random matrix. The main data.frame already stores latitude and longitude of the centroids of each county, so we do not need to calculate them (the `sf` package provides the `st_centroid()` function for this purpose). For some reason, R misinterprets the values for latitude and longitude, so we remove the initial + from the values provided to let it correctly interpret the data. We calculate distances using the haversine function

```
# Latitude, longitude for each patch
coordinates <- matrix(0, nrow=patches, ncol=2)
entry <- 1
for (from in counties.wgs84$id)  {
    coordinates[entry,] <- c(
        as.numeric(gsub( "+", "",
                    counties.wgs84[ counties.wgs84$id == from, ]$INTPTLAT )),
        as.numeric(gsub( "+", "",
                    counties.wgs84[ counties.wgs84$id == from, ]$INTPTLON ))
    )
    entry <- entry+1
}
dists <- apply(coordinates, 1, function(x) {haversine(coordinates[1,], x)})

# Only consider commuting, exclude those that do not move
coupl <- rho[1,2:nrow(rho)]
dists <- dists[ 2:nrow(rho) ]
# Equivalently: dists <- dists[ dists > 0 ]
```

```r
#     "         dists <- dists[ 2:length(dists) ]
plot(dists, coupl, log="xy", ylab="Coupling", xlab="Distance")
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 8 y values <= 0 omitted
## from logarithmic plot
```



Here we see something very interesting: the dependence of the spatial coupling with the distance is roughly linear in log-log scale. This kind of dependence is called *power-law* (as $\rho_{ij} \propto d_{ij}^{-\alpha}$ with $\alpha > 0$: the coupling is proportional to some negative power of the distance). We can also measure the power exponent, but we need to exclude from the calculation the zeros using a trick:

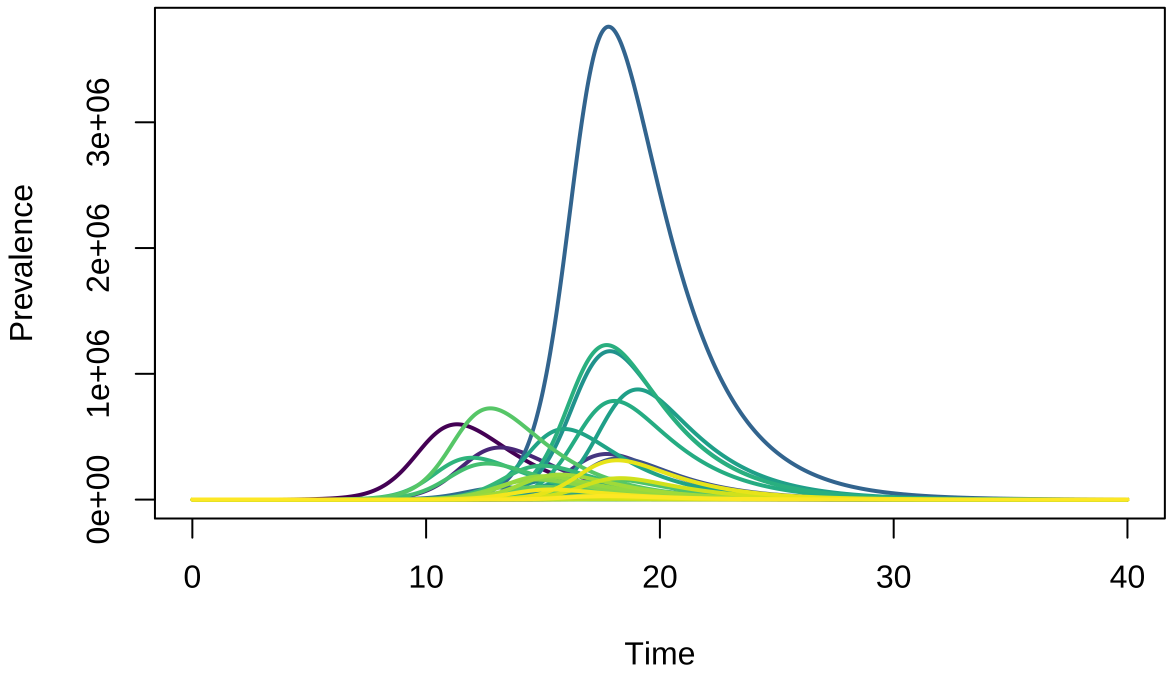
```r
coupl[ coupl == 0 ] <- NA
lm( log(coupl) ~ log(dists), na.action = "na.omit" )
```

```
##
## Call:
## lm(formula = log(coupl) ~ log(dists), na.action = "na.omit")
##
## Coefficients:
## (Intercept)   log(dists)
##       4.592       -2.482
```

The appearence of power-laws in human mobility and infectious disease data has important consequences: in this specific case it is showing that there is no intrinsic scale for human travel, and trips occurring at all distances carry a relevant contribution. Consequently, relevant transmission of disease occurs on multiple spatial scales at the same time. This kind of super-diffusion explains the rapid spread of infectious diseases from the moment they appear within a population and the difficulty involved in their containment.

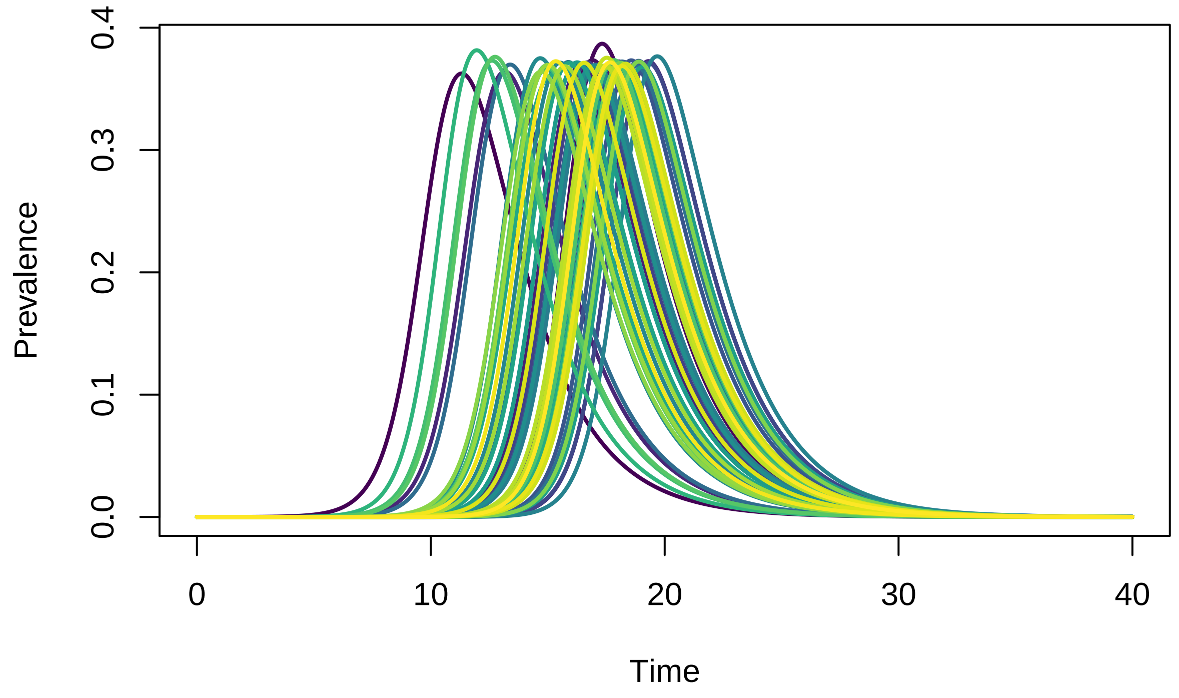
We can now integrate the model to study the spread of flu in California from a few initial cases.

```r
NN <- counties.wgs84$pop
beta <- 1.66
times <- seq( from = 0, to = 40, by = 0.05)
start.I0 <- rep(0, patches)
start.I0[1] <- 10
start.S0 <- NN-start.I0
xstart <- c( start.S0, start.I0 )

params <- list(patches, mu, gamma, beta, rho, NN)
res <- ode( xstart, times, sir.model, params, method=deSolve::rk4 )
output <- data.frame(res)
showOutput(output, patches)
```



We can also plot the daily fraction of infected individuals in each patch. This allows for comparison between patches of differen population size

```r
res2 <- t( apply(res, 1, function(x) {x/c(1,NN,NN)}) )
output <- as.data.frame( res2 )
showOutput(output, patches)
```



Exercises

- *Exercise 4* - What would be the outcome if the initial infective were in top-right county (Modoc county)? What would be the outcome if the initial infective were in the Los Angeles county? Could you comment on the outcomes?

Concluding

Before finishing, we save the modified data sets so that we will be able to use them in the next workshop

```r
if (!dir.exists("Data/Local"))  {
    dir.create("Data/Local")
}
counties.wgs84$ALAND <- as.character(counties.wgs84$ALAND)
counties.wgs84$AWATER <- as.character(counties.wgs84$AWATER)
write_sf(counties.wgs84, dsn="Data/Local/map.shp", delete_layer=TRUE)
write.table(counties.pop, file="Data/Local/population.dat",
            col.names=TRUE, row.names=FALSE)
write.table(coordinates, file="Data/Local/coordinates.dat",
            col.names=TRUE, row.names=FALSE)
write.table(counties.mapper, file="Data/Local/mapper.dat",
            col.names=TRUE, row.names=FALSE)
write.table(comm, file="Data/Local/comm.dat",
            col.names=TRUE, row.names=FALSE)
```