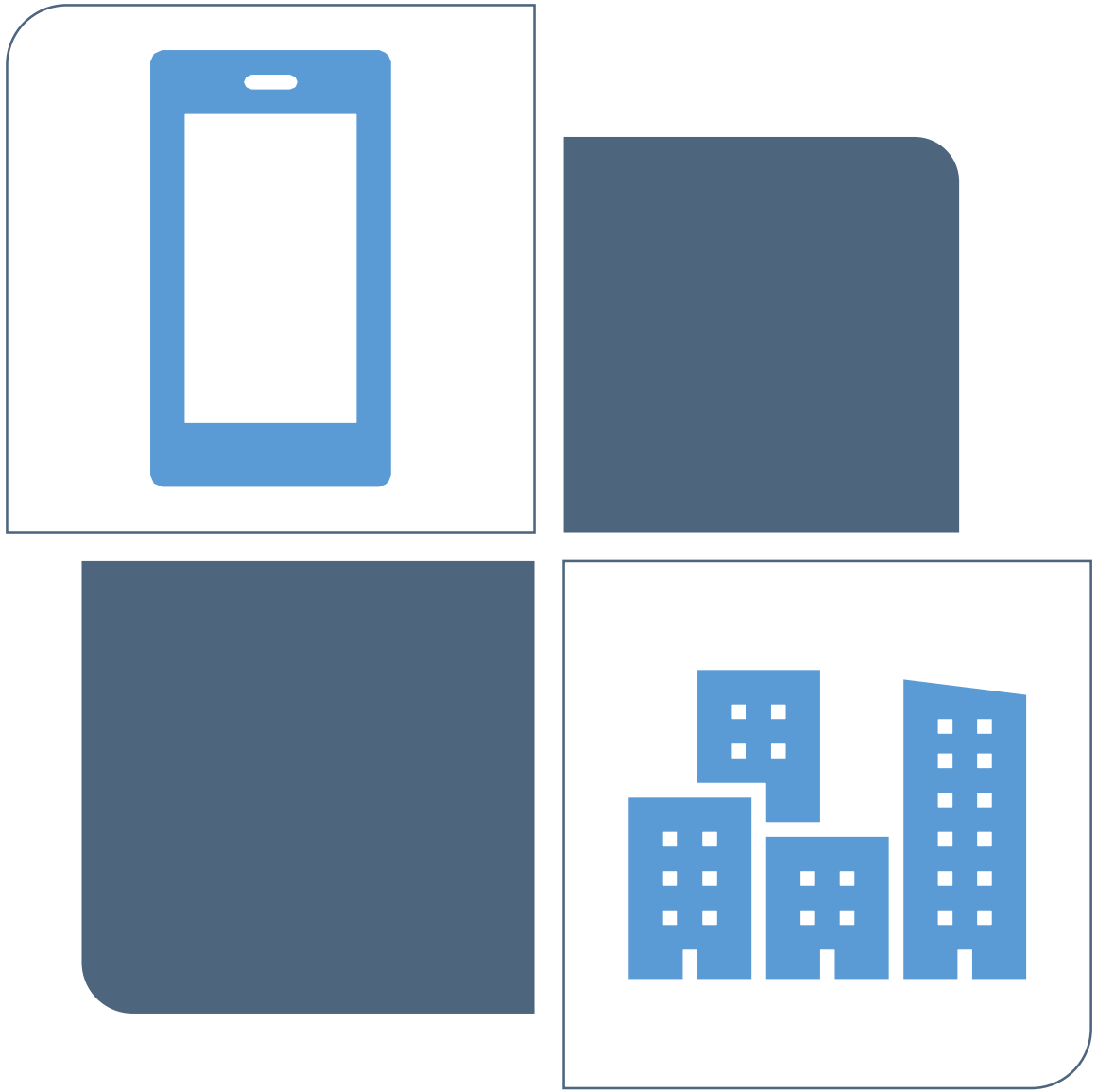
pestim - an R package to compute population estimations using mobile phone data

@uRos2018

Bogdan OANCEA
David SALGADO
Antoniade Ciprian ALEXANDRU
Luis SANGUIAO



Content of the presentation

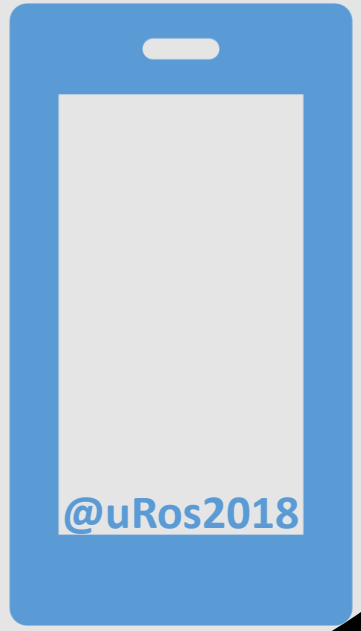
Introduction

Population estimation using *pestim*

Implementation details

A performance study of *pestim*

Conclusions and future developments



Introduction

- Integration of the mobile phone data sets in the production of the official statistics was one of the main goals of the ESSnet Big Data project;
- An important part of the work during this project was to develop an R package which we called *pestim* to implement the methodology designed to estimate the population counts of different territorial cells;
- The theoretical model implemented by *pestim* package was based on a hierarchical model borrowed from the ecological sampling techniques;

Introduction

- The theoretical model can be summarized as:

$$N_i^{MNO} \simeq Bin(N_i, p_i), \quad N_i^{MNO} \perp N_j^{MNO}, \quad i \neq j = 1,2,\dots,I$$
$$N_i \simeq Po(\lambda_i), N_i \perp N_j, \quad i \neq j = 1,2,\dots,I$$
$$p_i \simeq Beta(\alpha_i, \beta_i), \quad p_i \perp p_j, i \neq j = 1,2,\dots,I$$
$$(\alpha_i, \beta_i) \simeq \frac{f_1\left(\frac{\alpha_i}{\alpha_i+\beta_i}; N^{REG}, z\right) * f_2(\alpha_i+\beta_i; N^{REG}, z)}{\alpha_i+\beta_i}, (\alpha_i, \beta_i) \perp (\alpha_j, \beta_j), i \neq j = 1,2,\dots,I$$
$$\lambda_i \simeq f_3(\lambda_i; N^{REG}, z), \quad (\lambda_i>0, \lambda_i \perp \lambda_j), i \neq j = 1,2,\dots,I$$

- The prior information are incorporated in the probability distributions f_1, f_2 and f_3.

@uRos2018

Introduction

- The posterior distribution **P**(N|N^{MNO}; N^{REG}) is given by:

$$P(N|N^{MNO}, N^{REG}) \propto \int_0^{\infty} d\lambda \, P(\lambda|\, N^{MNO}; N^{REG}) * Po(N; \lambda)$$

- The integral from the RHS is computed using a Monte Carlo technique using stratified importance sampling and relies on computing the product of a ratio of two Beta functions and the confluent hypergeometric function (${}_1F_1$);

@uRos2018

Introduction

- Why R?
  - Freely available;
  - It seems to be the most used software inside the statistics community (at least in EU countries);
  - Portable: there are R distributions for all major operating systems currently in use in the official statistics community (the famous slogan "*wRite once Run anywhere*" is perfectly valid for the R environment too);

Introduction

- ***pestim*** is freely available under the GPL3 and EUPL licenses at the following address:

  https://github.com/MobilePhoneESSnetBigData/pestim

- It requires at least R version 3.3.0, but upgrading R to the newest version is highly recommended;
- At this moment, we recommend to install it from sources (requires compilation)
  - For Windows users, Rtools should be installed;
  - For Linux and Mac OS X a proper C++ compiler should be available (gcc or LLVM);
- Installing the package is simple:

```
library(devtools)
install_github("MobilePhoneESSnetBigData/pestim", build_vignettes=TRUE)
```

@uRos2018

Introduction

- We also provide binaries for Windows and Mac OS X
  - for Windows : **https://github.com/MobilePhoneESSnetBigData/Estimation_Population/blob/master/pestim_0.1.0.zip**
  - for Mac OS X : **https://github.com/MobilePhoneESSnetBigData/Estimation_Population/blob/master/pestim_0.1.0.tgz**
- Some *minimal* hardware requirements (to run the example simulations):
  - a computer with *at least 8GB of RAM memory and an Intel I7 processor with 4 physical cores* (although Intel I5 with 2 cores works, but with the corresponding increase in the running time);

Introduction

- Documentation of the package is available as:
  - A package vignette;
  - A Reference Manual, available at: pestim/doc/pestim_Reference_Manual.pdf
  - Usual R Help for each function included in this package callable from R console with:
    - `?help(pestim)`
    - `?function_name`

@uRos2018

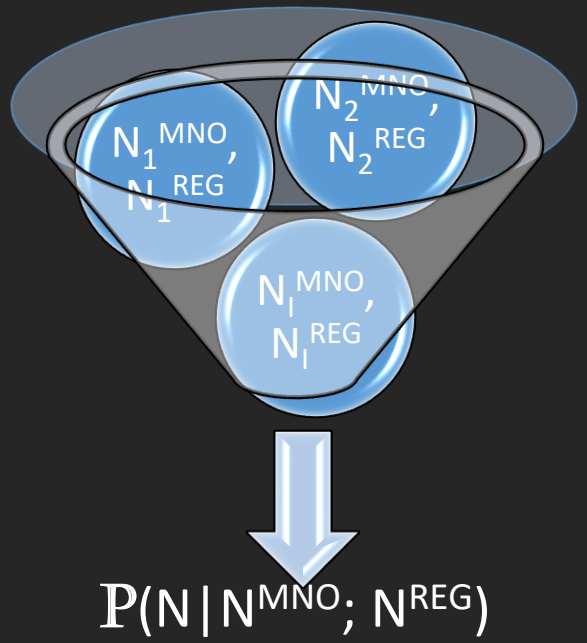
Population estimation using *pestim*

@uRos2018

- The model implemented in ***pestim*** package is based on two prior assumptions:
  - Given that mobile phone data and official data operate at different time scales, we assume that there is an *initial time instant* in which we can equate population figures from both sources;
  - The mobility patterns of individuals *do not depend* on the mobile network operator which they are subscribed to;

Population estimation using *pestim*

- The process diagram of computing population estimation using mobile phone and official population data is depicted below:



(N_1^{MNO}, N_1^{REG}) … (N_I^{MNO}, N_I^{REG}) are the population counts reported by the MNO in territorial cells and $\mathbb{P}(N \mid N^{MNO}; N^{REG})$ is the posterior probability distribution that can be used to assess the uncertainty in the output estimates;

@uRos2018

Population estimation using *pestim*

- We show how ***pestim*** can be used to produce estimation at the *initial time instant* t_0 for a number of cells;
- Then, we will generalize for successive time instants;

@uRos2018

Population estimation using *pestim*

- The process of estimating the population:

1. Set the values for N^{MNO} and N^{REG}
2. Generate values using the prior distributions for the hyperparameters
3. Estimate the population counts using a predefined number of simulations
4. Compute some statistics (mean, median, mode) of the estimated population counts
5. Visualize the results

@uRos2018

Population estimation using *pestim*

- We used simulated data sets for N=24 territorial cells;
- We loaded the initial data sources into the data.frame *N_0* which have the columns:
  - *N_0$pop_official* storing the data for N^{Reg} ;
  - *N_0$pop_phone* for N^{MNO}
- We assumed some weakly informative prior distributions for f_1 and f_2 using the uniform distribution for them and a gamma distribution for f_3.

@uRos2018

Population estimation using *pestim*

```
f1<-list()
f2<-list()
f3<-list()
N = 24
sc = 100
cv_lambda <- 0.3
alpha <- 1 / cv_lambda**2 - 1
for(i in 1:N) {
  f1[[i]] <- list('unif', xMin = 0.3, xMax = 0.5)
  f2[[i]]<- list('unif', xMin = 1, xMax =
N_0$pop_official[i]/sc+10)
  f3[[i]]<-list('gamma', shape = 1+alpha, scale =
N_0$pop_official[i]/sc/alpha)
}

estim_t0 <- postN0(nMNO = N_0$pop_phone/sc, nReg =
N_0$pop_official/sc,fu = f1, fv = f2, flambda = f3,
scale=sc, nThreads = 8)
```

@uRos2018

Population estimation using *pestim*

- The actual computation is performed by the function *postN0*();
- This function generates *n* random values according to the posterior distribution $P(N|N^{MNO};N^{REG})$ that are used to compute the mean/mode/median and accuracy indicators;
- We used the scaling parameter *sc=100* in order to avoid a numerical overflow during the computations;
- We also used 8 threads to make use of the full computation power of our computer;

@uRos2018

Population estimation using *pestim*

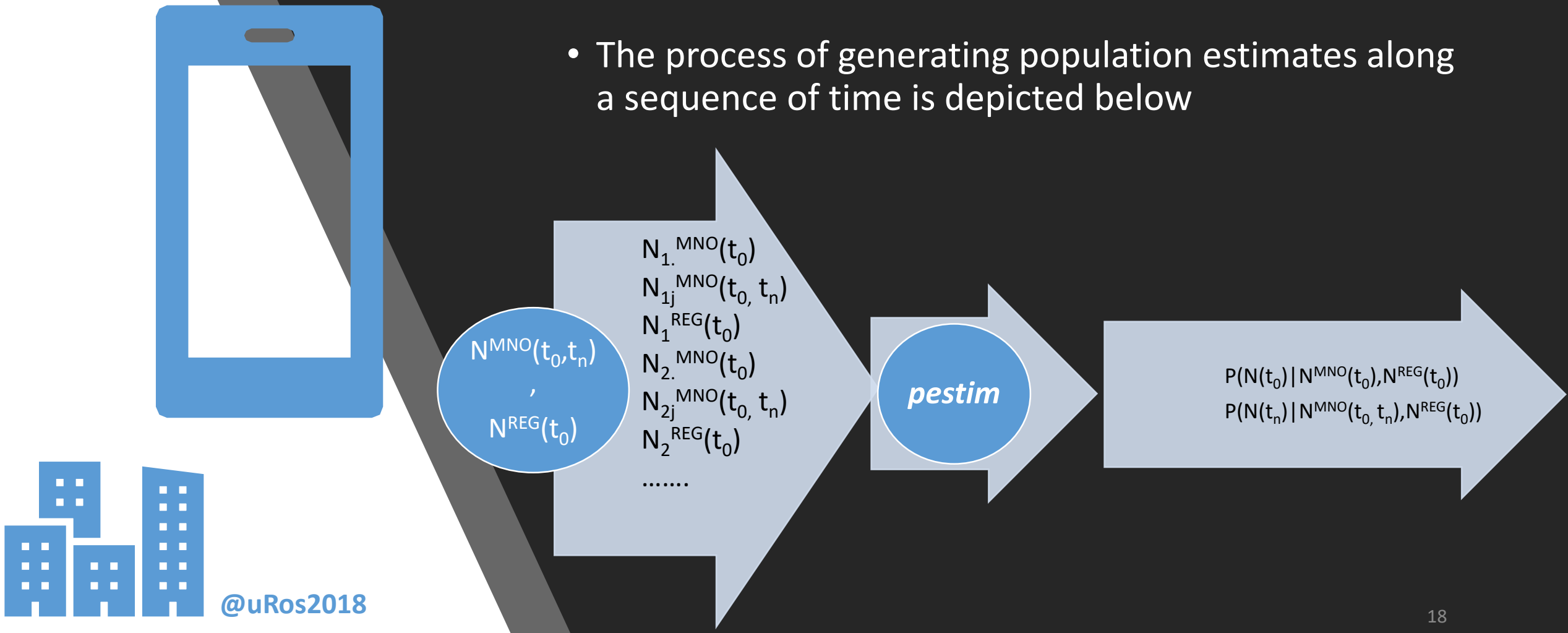
- The results look like this:

	Mean	StDev	CV	Median	Median_CI_LB	Median_CI_UB	MedianQuantileCV	Mode	Mode_CI_LB	Mode_CI_UB	ModeQuantileCV
[1]	6028	1580.19	26.21	5778	5687.675	5879.550	35.32	5852	3411	9356	34.87
[2]	8200	1773.30	21.63	8064	7949.725	8170.475	27.53	6815	5323	12143	32.58
[3]	2779	718.26	25.84	2710	2659	2744	34.85	2967	1449	4238	31.83
...											
[23]	4401	1664.99	37.83	4160	4037.525	4242.900	54.43	3954	1546	7636	57.26
[24]	5127	1266.18	24.69	4996	4901.525	5100.475	32.66	5325	2835	7678	30.64

@uRos2018

Population estimation using *pestim*

- The process of generating population estimates along a sequence of time is depicted below



@uRos2018

Population estimation using *pestim*

- Estimation for a sequence of time instants supposes to compute the transition probability from one cell to another using MNO data;
- We present a code snippet that implements the estimations for 40 successive time instants and the same 24 territorial cells;
- *transitions_all* variable stores the number of people $N_{ij}^{MNO}(t_0,t_n)$ moving for a cell *i* to cell *j* in the time interval (t_0,t_n) as the MNO detects them;
- We used the same scaling factor as in the previous example, the uniform distribution for f_1 and f_2 and a gamma distribution for f_3;

@uRos2018

Population estimation using *pestim*

```
sc = 100
nReg = N_0$pop_official

# List of priors for f2
v0 <- nReg /sc
cv_v0 <- 0.10
fv <- lapply(v0, function(u){
  umin <- max(0, u - cv_v0 * u)
  umax <- u + cv_v0 * u
  output <- list('unif', xMin =
umin, xMax = umax)
  return(output)
})

# List of priors for f3 (flambda)
cv_lambda <- 0.6
alpha <- 1 / cv_lambda**2 - 1
f3 <- lapply(v0,
function(v){list('gamma', shape =
1 + alpha, scale =  v / alpha)})


# Names and parameters of priors
for the #transition probabilities
distNames <- rep('unif', 24)
variation <- rep(list(list(cv =
0.20)), 24)

for(i in 1:40) {
    nMNOmat = transitions_all[[i]]
    u0 <- rowSums(nMNOmat) /nReg
    cv_u0 <- 0.15
    f1 <- lapply(u0, function(u){
      umin <- max(0, u - cv_u0 *
u)
      umax <- min(1, u + cv_u0 *
u)
      output <- list('unif', xMin
= umin, xMax = umax)
      return(output)
    })
postNt(nMNOmat/sc, nReg/sc, fu =
f1, fv=f2, flambda=f3, distNames,
variation, scale = sc, nThreads =
8)
}
```

@uRos2018

Implementation details

pestim software package contains basically three types of functions:

High level functions

Provides computation for population estimates.

postN0, postNt, postNtcondN0

Functions implementing different statistical distributions

Generate random values according to different statistical distributions for priors, posteriors, and also for parameter specifications for these distributions.

dtriang, rtriang, ptriang, qtriang, dlambda, rlambda, rmatProb, rN0, rNt, rNtcondN0, rg, rp, alphaPrior, genAlpha, genUV.

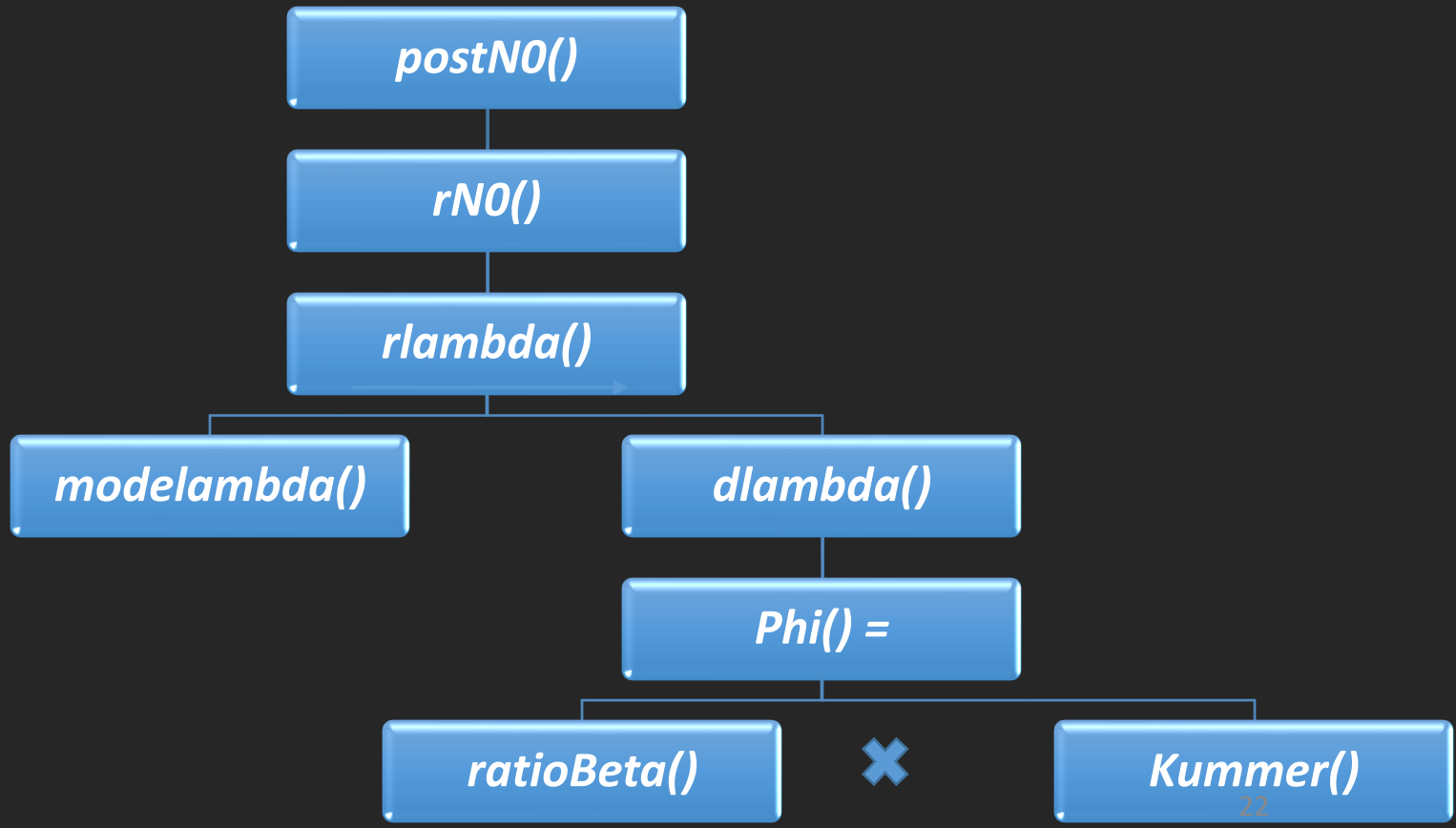
Utility functions

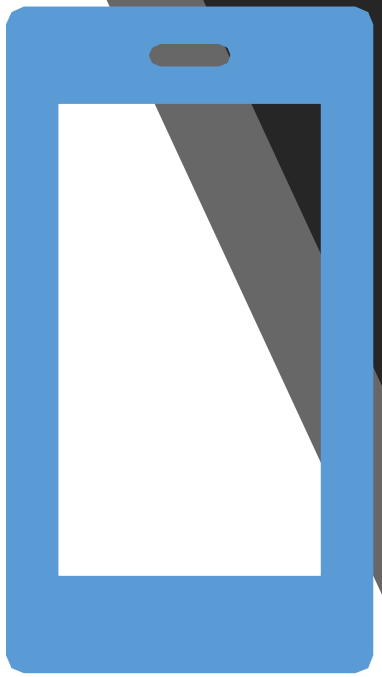
Provide implementation of some mathematical functions such as the ratio of two beta functions, the confluent hypergeometric function, etc.

ratioBeta, kummer, Phi, modeLambda

Implementation details

- Implementation details for the *postN0*() function that estimates the population counts for a set of territorial cells at the initial time;
- The complete calling tree of this function:





@uRos2018

Implementation details

@uRos2018

- ***postN0()*** generates ***n*** random values for population according to the posterior distribution by calling ***rN0()*** which in turn calls ***rlambda(),*** the posterior distribution being a Poisson distribution;
- ***rlambda()*** computes the mode for the posterior distribution of λ using ***modeLambda()*** and then apply the acception-rejection method to generate the random values;
- ***modeLambda()*** uses the posterior density function of the parameter λ from the hierarchical model, implemented by ***dlambda()***;
- ***dlambda()*** computes the unnormalized posterior density function of the λ parameter:

$$f(\lambda|N^{MNO};N^{REG}) \propto f(\lambda)*\text{dPois}(N^{MNO};\lambda)*\text{S}(\lambda;N^{MNO},N^{REG})$$

Implementation details

where S is given by:

$$S(\lambda, N^{MNO}, N^{REG}) = \int_0^{\infty} dv f_2(v) \int_0^1 du\, f_1(u)\Phi(uv, (1-u)v; \lambda,$$
$$N^{MNO}, N^{REG}) = \int_0^{\infty} dv f_2(v) \int_0^1 du\, f_1(u)\overline{\Phi}\,(u, v; \lambda, N^{MNO}, N^{REG})$$

- It is computed using a Monte Carlo technique and ϕ is computed as a ratio of two Beta functions (***ratioBeta()*** function) multiplied by the confluent hypergeometric function ${}_1F_1$ (***kummer()***);
- ***kummer()*** is one of the most computationally demanding functions in ***pestim*** package and we implemented it using C++ language and Rcpp and RcppParallel packages;

@uRos2018

Implementation details

$$ {}_1F_1(z;a;b) \approx S_N = \sum_{j=0}^{N} \frac{(a)_j z^j}{(b)_j j!} = \sum_{j=0}^{N} A_j $$

where $(a)_j$ is the Pochhammer symbol

- We used a Taylor series expansion to compute this function:

```
A = 1, S = A
for(j=0, A / S< tol; j=j+1)
    A = A * (a+j)/(b+j) z/(j+1)
    S = S + A
endfor
```

@uRos2018

Implementation details

- We found that calling the C++ ***kummer(z, a, b)*** function from R for each tuple of parameters (z,a,b) is inefficient;
- We transformed the function to receive vectors as parameters, to reduce the number of function calls;
- The computation for a tuple (z_i, a_i, b_i) is independent from the computation for (z_j, a_j, b_j) so we can parallelize the computations:

```
1 divide vectors z, a, b in equal chunks
2 for(each chunk z_c, a_c, b_c) do in parallel
      kummer(z_c, a_c, b_c)
```

- The parallel version of the algorithm was implemented using RccpParallel

@uRos2018

Implementation details

- *rN0*() generates the random values according to the posterior distribution needed to compute the estimations for the territorial cells;
- The estimation for one cell is independent from the estimations for the other cells -> this is the most natural point to introduce further parallelization of the code;
- We used *foreach* package to compute the estimations for each cell in parallel using *doParallel* package as a parallel backend:

```
makecluster(nThreads)
for each(i = 1 to nCells) do in parallel {
    random_points[i] = rN0(...)
}
```

@uRos2018

A performance study of *pestim*

- Scalability is an important feature of the *pestim* package;
- 3 experimental setups:

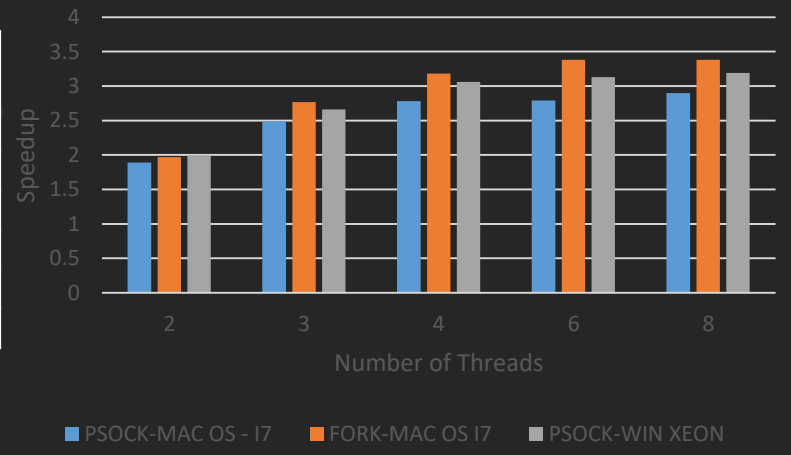
Processor	Intel I7 Skylake (6820HQ), 2.7Ghz, Turbo Boost 3.6Ghz, L1 32k x 32k per core, L2 256k per core, L3 8 MB
RAM	16 GB, 2133MHz, LPDDR3 SDRAM
OS	Mac OS X High Sierra 10.13.4
R	R version 3.4.1 64 bit
C++ compiler	C++ Compiler: Apple LLVM version 9.1.0
Cluster type	PSOCK for experimental setup 1 / FORK for experimental setup 2

Processor	Intel XEON E1246 v3, 3.5 Ghz, Turbo Boost 3.9 GHz, L1 32k x 32k per core, L2 256k per core, L3 8MB
RAM	16 GB 1600MHz DDR3
OS	Windows 8.1
R	R version 3.3.2 64 bit
C++ compiler	GNU C ver. 4.9.3 (min_gw)
Cluster type	PSOCK

A performance study of the *pestim*

- We tested the performances of *postN0()* function to estimate the population counts for a number of 24 territorial cells with the number of computing threads taking values from 1 to 8;

No. threads	2	3	4	6	8
Experimental setup 1 (PSOCK-MAC OS-I7)					
Speedup	1.89	2.49	2.78	2.79	2.90
Experimental setup 2 (FORK-MAC OS-I7)					
Speedup	1.97	2.77	3.18	3.38	3.38
Experimental setup 3 (PSOCK-WIN XEON)					
Speedup	1.99	2.66	3.06	3.13	3.19

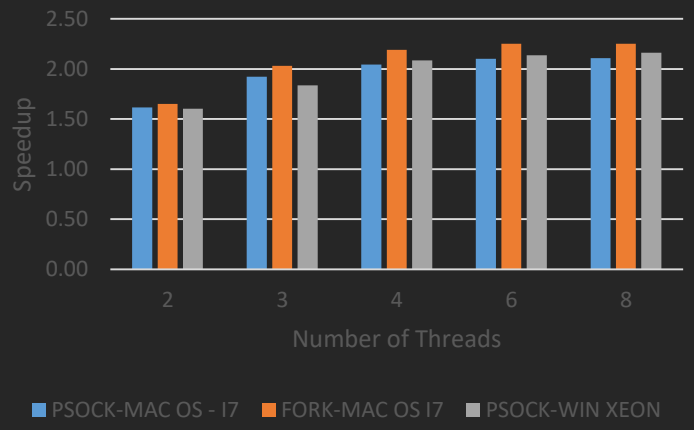


- A FORK based cluster under a Unix compatible operating system performs slightly better than a PSOCK based cluster;
- The speedup become saturated after nThreads = 4 ;

A performance study of the *pestim*

- We performed the same test for estimating the population counts for a number of 40 time instants using *postNt*()

No. threads	2	3	4	6	8
Experimental setup 1 (PSOCK-MAC OS-I7)					
Speedup	1.62	1.92	2.05	2.10	2.11
Experimental setup 2 (FORK-MAC OS-I7)					
Speedup	1.65	2.03	2.19	2.25	2.25
Experimental setup 3 (PSOCK-WIN XEON)					
Speedup	1.60	1.84	2.08	2.14	2.16



- The results are similar to the pervious experiment, showing that a FORK cluster performs better than a PSOCK one;

Conclusions and future developments

- *pestim* R package combines aggregated mobile phone data with another official data source to estimate the population counts in a number of territorial cells both at an initial moment and at a sequence of successive time instants;
- *pestim* shows a good scalability and an easy to use high level interface;
- We intend to further improve the performance of the computations:
  - more internal functions - to be implemented in C++;
  - explore the possibility of using GPU computations;
- We also intend to add a visualization layer.