Introducing spatial microsimulation with R: a practical

Lovelace, Robin
r.lovelace@leeds.ac.uk

April 29, 2014

Contents

1 Foundations

1.1 Prerequisites for this practical

This practical is about spatial microsimulation in the software R. We suggest you install and take a look at this powerful program before getting started. We recommend using R within RStudio, which makes using R much easier. Instructions to install both R and RStudio can be found online: rstudio.com/ide/download/desktop.

The other prerequisite for the course is downloading the example data. These can be downloaded in a single zip file which can be found on the course's GitHub repository: github.com/Robinlovelace/smsim-course. Click on the "Download ZIP" button to the right of this page and extract the folder into your desktop or other suitable place on your computer.

Once the folder has been successfully extracted open it in your browser and take a look around. You will find a number of files and two sub-folders entitled 'data', and 'figures'. When you get to the sections that use R code, it is useful for R to operate from within the smsim-course-master folder. Probably the best way to set this up is to open the file 'smsim-course.Rproj' from within RStudio (fig. 1). Try this now and click on the 'Files' tab in the bottom right hand window of RStudio. Before using the power of R in RStudio it's worth understanding a bit about 'IPF', the algorithm we will use to generate the synthetic population or 'spatial microdata' (fig. 2).

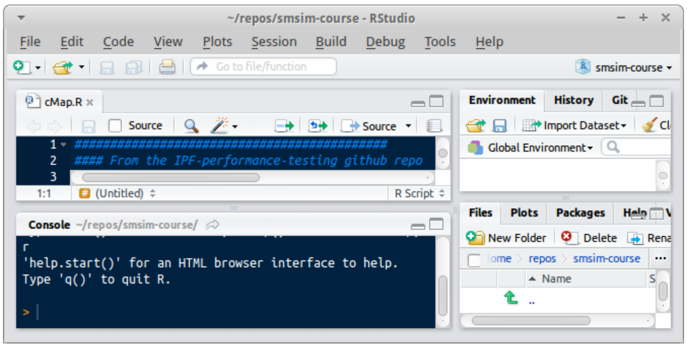


Figure 1: The RStudio user interface. Note the project title 'smsim-course' in the top right and the 'Files' tab at the top of the left hand window.

1.2 Learning by doing

As Kabacoff (2011, xxii) put it regarding R, "the best way to learn is to experiment" and the same applies to spatial microsimulation. We believe you will learn the technique/art best not by reading about it, but by doing it. Mistakes are inevitable in any challenging task and should not be discouraged. In fact it is by making blunders, identifying and then correcting them that many people learn best. Think of someone learning to skate: no one ever picks up a skateboard for the first time being able to 'surf the sidewalks'. It takes time, patience and plenty of falls before you master the art. The same applies to spatial microsimulation.

One of the tricky things about spatial microsimulation for newcomers is its use of specialist language. It is important to know exactly what is meant by 'special microdata' and other technical terms. To this end we have created a glossary that provide succinct definitions (see section 5). Any term that is *italicised* in the text has a glossary entry.

Spatial microsimulation works by taking *microdata* at the individual level and using aggregate-level constraints to allocate these individuals to zones. The two main methods are *deterministic reweighting* and *combinatorial optimisation*. This practical takes the former approach using a process called *iterative proportional fitting* (IPF). IPF is used to increase the weights of individuals who are representative of the target area and reduce the weights of individuals who are relatively rare (Lovelace and Ballas, 2013). The output is a *spatial microdataset*.

A schematic of the process is shown in fig. 2. Take a look at the image and think about the process. But we don't want to get bogged down in theory or applications in this course: we want to 'get our hands dirty'. Next we will do just that, by applying the process to some example data.

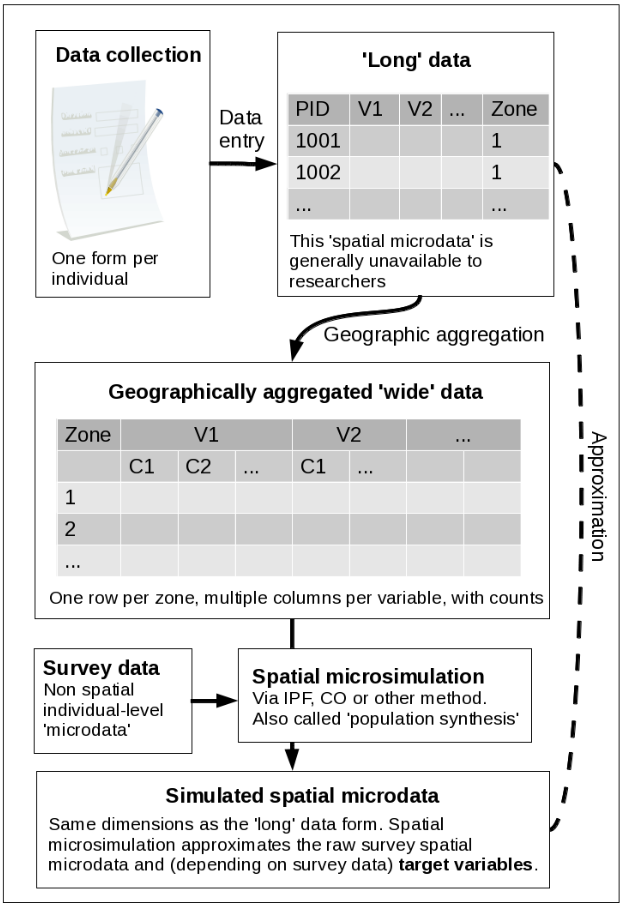


Figure 2: Schema of iterative proportional fitting (IPF) and combinatorial optimisation in the wider context of the availability of different data formats and spatial microsimulation.

1.3 Some input data

Let's start with a very basic dataset. To aid understanding, we will first do the reweighting by hand to understand the process before automating it on the computer later. As outlined in Figure 2, there are two input files required to perform spatial microsimulation:

- Survey data - information about individuals
- Geographically aggregated ('wide') data - typically aggregated Census tables

Table 1 shows 5 individuals, who are defined by two constraint variables: age and sex. This is analguous to the survey data shown in Figure 2. Table 2 presents this same data in aggregated form, whose margin totals can be used in the IPF procedure described shortly.

Table 3 contains data for a hypothetical area. This is analguous to the geographically aggregated 'wide' data shown in Figure 2.

Table 1: A hypothetical input microdata set ('Survey data')

Individual	Age	Sex	Weight
1	59	Male	1
2	54	Male	1
3	35	Male	**1**
4	73	Female	1
5	49	Female	1

Table 2: The aggregated results of the weighted microdata set ($m(1)$). Note, these values depend on the current weight so change after each iteration; this table is for the initial weight.

Marginal totals		j		
	Age/sex	Male	Female	T
i	Under-50	**1**	1	**2**
	Over-50	2	1	3
	T	3	2	5

Table 3: Small area constraints (s) (Geographically aggregated 'wide' data).

Constraint ⇒	i		j	
Category ⇒	i_1	i_2	j_1	j_2
Area ⇓	Under-50	Over-50	Male	Female
1	**8**	4	6	6

Table 4 illustrates this aggregate level table in a different form, to show our ignorance of interaction between age and sex for this geography.

Table 4: Small area constraints expressed as marginal totals, and the cell values to be estimated.

Marginal totals		j		
	Age/sex	Male	Female	T
i	Under-50	**?**	?	**8**
	Over-50	?	?	4
	T	6	6	12

With our input data in place, we can begin our first iteration.

1.4 The IPF equation

Using the tables about we readjust the weights of the individuals so that:

- If there are exactly the same number of
- Individuals who are rare in the geography of interest are given a smaller weight (we need fewer of them).
- Individuals who are common in the geography of interest are given a larger weight (we need more of them).
- Their sum equals the total population of the geography of interest (12 in our example).

For each constraint (age or sex for example), the current weight is multiplied by the respective total from the geographically aggregated table (Table 3) and divided by the corresponding marginal total of the survey data (see table 2). This is done one constraint at a time, as described in eq. (1) and more precisely by eq. (2) for constraint i (age in this case):

$$Weight_{New} = Weight_{Current} \times \frac{Survey_{Constraint\,Total}}{Census_{Constraint\,Total}} \tag{1}$$

$$w(n+1)_{ij} = \frac{w(n)_{ij} \times sT_i}{mT(n)_i} \tag{2}$$

where $w(n+1)_{ij}$ is the new weight for individuals with characteristics i (age, in this case), and j (sex), $w(n)_{ij}$ is the original weight for individuals with these characteristics, sT_i is element marginal total of the small area constraint, s (Table 3) and $mT(n)_i$ is the marginal total of category j of the aggregated results of the weighted microdata, m (Table 2). n represents the iteration number.

After creating $Weight_{New}$, the individual level data is recreated before the next weight can be calculated using the next constraint.

Do not worry about understanding the above equation for now. More important is implementing it. Follow the emboldened values in tables 1 to 4 to see how the new weight of individual 3 is calculated for the age constraint. Table 5 illustrates the outcome. Notice that the sum of the weights is equal to the total population, from the constraint variables.

Table 5: Reweighting the hypothetical microdataset in order to fit Table 3.

Individual	Sex	age-group	Weight	New weight, w(2)
1	Male	Over-50	1	$1 \times 4/3 = \frac{4}{3}$
2	Male	Over-50	1	$1 \times 4/3 = \frac{4}{3}$
3	Male	Under-50	1	$\mathbf{1} \times \mathbf{8/2} = 4$
4	Female	Over-50	1	$1 \times 4/3 = \frac{4}{3}$
5	Female	Under-50	1	$1 \times 8/2 = 4$
Total			5	12

After the individual level data have been re-aggregated (table 6), the next stage is to repeat eq. (2) for the age constraint to generate a third set of weights, by replacing the i in sT_i and $mT(n)_i$ with j and incrementing the value of n:

$$w(3)_{ij} = \frac{w(2)_{ij} \times sT_j}{mT(2)_j} \tag{3}$$

1.5 Test your understanding

To test your understanding of IPF, try to apply eq. (3) to the information above and that presented in table 6. This should result in the following vector of new weights, for individuals 1 to 5. Calculate the correct values and pencil them in in place of the question marks.

$$w(3) = \left(\frac{6 \times \frac{4}{3}}{6\frac{2}{3}}, \frac{6 \times \frac{4}{3}}{?}, \frac{\mathbf{4 \times 6}}{\mathbf{6\frac{2}{3}}}, \frac{?}{?}, \frac{4 \times 6}{5\frac{1}{3}}\right) \tag{4}$$

After simplifying these fractions, the results are as follows. One 'sanity' check of your method here is whether the sum of these weights is still equal to the area's total population of twelve. Test this is true:

$$w(3) = (\frac{6}{5}, \frac{6}{5}, \frac{\mathbf{18}}{\mathbf{5}}, \frac{3}{2}, \frac{9}{2}) \quad (5)$$

What do the weights in w(3) actually mean? They indicate how representative each individual is of the target zone after one iteration of IPF, constraining by age and sex. Individual number 5 has the highest weight because there is only one young female in the survey dataset yet seemingly many in the area in question.

Notice also that after each iteration the fit between the marginal totals of m and s improves.[1]

Table 6: The aggregated results of the weighted microdata set after constraining for age ($m(2)$).

Marginal totals		i		
	Age/sex	Male	Female	T
j	Under-50	**4**	4	8
	Over-50	$\frac{8}{3}$	$\frac{4}{3}$	4
	T	$\mathbf{6\frac{2}{3}}$	$5\frac{1}{3}$	12

1.6 IPF in a spreadsheet

To ensure smooth transition between the IPF process described in mathematics above and its implementation in R this section provides an intermediary stage: the speed of computation and the visual support of a graphical user interface (GUI).

1.7 IPF in R: a simple example

So far we have implemented IPF by hand and in a spreadsheet. This section explains how the IPF *algorithm* described above is implemented in R, using the same input data.[2] The code snippets are taken from the 'simple.R' script file (just a plain text document, like all R scripts) so you can copy and paste. However, it is highly recommended to not look at this file until you need to, towards the end of section. This is 'leaning by typing'!

The data presented in the above worked example are saved in the 'simple' sub-folder of 'data' as .csv files. To load them in R, we use the following commands:

Listing 1: Loading the input data in R

```
ind <- read.csv("data/simple/ind.csv") # load the individual level data
cons <- read.csv("data/simple/cons.csv") # load aggregate constraints
```

To check that the *data frames* have loaded correctly, simply ask R to print their contents by entering the object's name. To display the individual level data, for example, simply type the following:[3]

[1]This can be checked by comparing the aggregated weighted individuals with the small area constraints. Total absolute error (TAE), defined as the sum of all differences between simulated and observed marginal totals, improves between $m(1)$ to $m(2)$, falling from 14 to 6 in table 2 and table 6 above. TAE for $m(3)$ (not shown, but calculated by aggregating $w(3)$) improves even more, to 1.3. This number would eventually converge to 0 through subsequent iterations, a defining feature of IPF.

[2]A fuller tutorial is available from Rpubs, a site dedicated to publishing R analyses that are reproducible. This tutorial uses the RMarkdown mark-up language, which enables R code to be run and presented within documents. See http://rpubs.com/RobinLovelace/5089 .

[3]An alternative way to do this is to click on the object's name in the 'Environment' tab in RStudio's top right window.

Listing 2: Checking the contents of the individual level data frame

```
ind # Show the data frame in R
##   id age sex
##    1  59   m
##    2  54   m
##    3  35   m
##    4  73   f
##    5  49   f
```

Note that the input data is identical to the tables illustrated above for consistency. Instead of constraining for just one zone as we did in the example above, we will fit the data to six zones here. To subset a dataset, add square brackets to the end of the object's name in R. Entering `cons[1:2,]`, for example, should output the first two rows of the constraint variable. Within the square brackets the number before the comma refers to columns (blank means all columns) and the numbers after the comma refer to rows. If the output of this command looks like the text below, congratulations are in order: you have successfully loaded the constraint dataset.

Listing 3: Printing a subset of the constraint data in R

```
cons[1:2,]
##     X16.49 X50. m f
##   1      8    4 6 6
##   2      2    8 4 6
```

Note that the top row is identical to table 3 — we can therefore compare the results of doing IPF on the computer with the results obtained by hand.

To make the individual level data comparable to the aggregate level we must first categorise it and count the number in each category. This job is performed by the R *script* categorise.R, contained in the 'simple' data folder. Run it by entering `source("data/simple/categorise.R"))`. This results in the creation of a new object called ind.cat, which converts the variables into a matrix of 0s and 1s with same number of columns as the constraint data. Note that the 'categories.R' script must be modified each time it is used with a different dataset. To check that the script has worked properly, lets count the number of individuals it contains:

Listing 4: Output from ind.cat

```
colSums(ind.cat)
##     a16.49    a50.       m       f
##          2       3       3       2
```

The sum of both age and sex variables is 5 (the total number of individuals): it worked! Now the data is in the same form as the constraint variables and we have checked the data makes sense, we can create the 'weight' matrix and begin reweighting the individuals to zones:[4]

With all of these objects in place, we are ready to begin allocating new weights to the individuals via IPF. Below is written in code the IPF formula presented in section 1.4.

The above code creates the weight matrix that allocates individuals to zones. Aggregating this data to zones allows us to update 'ind.agg' with new values that will be closer to the constraint

[4] The above code will probably seem daunting and not make sense unless you are an experienced R user. Do not worry about this for now, you can always check what is going on later by checking R's documentation (e.g. try entering '?rep') or by pressing the tab key when entering arguments for R commands. For the time being, it is best to press-on with the example and understand the concepts — the details can be looked at later.

Listing 5: Creating the weight and aggregated individual level matrices

```
weights <- array(1, dim=c(nrow(ind),nrow(cons)))
ind.agg <- matrix(rep(colSums(ind.cat), nrow(cons)), nrow(cons))
```

Listing 6: Creating new weights for the individuals based on constraint 1 (age) via IPF

```
for (j in 1:nrow(cons)){
  for(i in 1:ncol(con1)){
 weights[which(ind.cat[,i] == 1),j] <- con1[j,i] / ind.agg[j,i]}}
```

variables. This is verified in the 'simple.R' script file with a line of code that calculates the Total Absolute Error after each iteration — `sum(sqrt((ind.agg-cons)^2))`. Check the error decreases with each iteration.

At this stage you should switch to running the code in 'simple.R'. In RStudio to run a line of code from the top left window, simply press `Ctl-Enter`. Use this technique to constrain by the second constraint, sex. Notice that we have just done in code what was previously done by hand. Thus we can verify the output of the R implementation against that obtained through mental arithmetic. Compare the following listing with eq. (5): are the weights the same?

Listing 7: W(3) after constraining by age and sex in R. Compare with eq. (5)

```
weights3[,1] # check the weights allocated for zone 1
##    [1] 1.2 1.2 3.6 1.5 4.5
```

1.8 Iterations

In the above example, we have seen the 'bare bones' of spatial microsimulation using IPF to generate weights from which sample populations can be created. To perform multiple iterations of the same model, we have prepared a slightly more complex script than 'simple.R' called 'etsim.R' that can be found in the 'data/simple/' folder. Th etsim script generates weights for as many iterations as you please (just change the value of `num.its`). This happens by calling another script called 'e2.R' which is simply run repeatedly for each iteration.

There is great scope for taking the analysis further: some further tests and plots are presented on the on-line versions of this section. The simplest case is contained in Rpubs document rpubs.com/RobinLovelace/6193 and a more complex case (with three constraints) can be found in Rpubs document 5089. For now, however, we progress to a more complex example, CakeMap.

2 CakeMap: A worked example

In this section we will process real data to arrive at an important result: an estimate of the amount of cake eaten in different wards in West Yorkshire. The example is deliberately rather absurd: hopefully this will make the technique more memorable. An attempt has been made to present the method in a generalisable was as possible, allowing users to apply the technique described here to their own data.

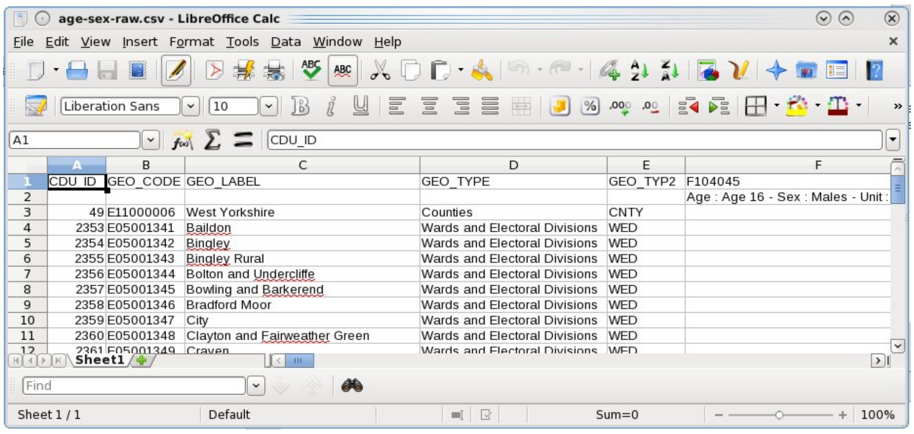
The code needed to run the main part of the example is contained within 'cakeMap.R'. Note that this script makes frequent reference to files contained in the 'cakeMap' folder 'data'. `read.csv("data/cakeMap/ind.csv")`, for example, is used to read the individual level data into R.

Because this example uses real world data (albeit in anonymised form), we will see how important the process of 'data cleaning' is. Rarely are datasets downloaded in exactly the right form for them to be pumped directly into a spatial microsimulation model. For that reason we describe the process of loading and preparing the input data first: similar steps will be needed to use your own data as an input into a spatial microsimulation model.

2.1 Loading and preparing the input data

The raw constraint variables for CakeMap were downloaded from the Infuse website (infuse.mimas.ac.uk/). These, logically enough, are stored in the 'cakeMap/data/' directory as .csv files and contain the word 'raw' so it is clear which files contain the raw data. The file 'age-sex-raw.csv', for example is the raw age and sex data that was downloaded. As the screenshot in fig. 3 shows, it was provided in a rather verbose form and needed to be processed before it could be usefully used as an input in a spatial microsimulation model. 'con1.csv' stands for 'constraint 1' and represents this age/sex data after it has been processed.

To ensure reproducibility in the process of converting the raw data into this 'spatial microsimulation ready' dataset, all of the steps used to clean and rearrange the data have been saved. Take a look at the R script file 'process-age.R' — these are the kinds of steps that the researcher will need to undertake before performing a spatial microsimulation model. Here is not the place to delve into the details of data reformatting; there are resources dedicated to that (**?**Kabacoff, 2011). However, it is worth taking a look at the 'process-age.R' script and the other 'process*' files to gain insight into how R can be used to quickly and effectively be used to process complex raw data into a form that is ready for use in modelling work.



	A	B	C	D	E	F
1	CDU_ID	GEO_CODE	GEO_LABEL	GEO_TYPE	GEO_TYP2	F104045
2						Age : Age 16 - Sex : Males - Unit :
3	49	E11000006	West Yorkshire	Counties	CNTY	
4	2353	E05001341	Baildon	Wards and Electoral Divisions	WED	
5	2354	E05001342	Bingley	Wards and Electoral Divisions	WED	
6	2355	E05001343	Bingley Rural	Wards and Electoral Divisions	WED	
7	2356	E05001344	Bolton and Undercliffe	Wards and Electoral Divisions	WED	
8	2357	E05001345	Bowling and Barkerend	Wards and Electoral Divisions	WED	
9	2358	E05001346	Bradford Moor	Wards and Electoral Divisions	WED	
10	2359	E05001347	City	Wards and Electoral Divisions	WED	
11	2360	E05001348	Clayton and Fairweather Green	Wards and Electoral Divisions	WED	
12	2361	E05001349	Craven	Wards and Electoral Divisions	WED	

Figure 3: The raw input data for the CakeMap model, downloaded from the InFuse website.

The input data generated through this process of data preparation are named 'con1.csv' to 'con3.csv'. For simplicity, all these were merged into a single dataset called 'cons.csv'. All the input data for this section are loaded with the following commands:

Listing 8: Loading the input data for CakeMap (see 'cMap.R')

```
ind <- read.csv("data/cakeMap/ind.csv")
cons <- read.csv("data/cakeMap/cons.csv")
```

Take a look at these input data using the techniques learned in the previous section. To test your understanding, try to answer the following questions: what are the constraint variables? How many individuals are in the survey microdataset? How many zones will we generate spatial microdata for?

For bonus points that will test your R skills as well as your practical knowledge of spatial microsimulation, try constructing queries in R that will automatically answer these questions.

It is vital to understand the input datasets before trying to model them, so take some time exploring the input. Only when satisfied with your understanding of the datasets (a pen and paper can help here, as well as R!) is it time to move on to generate the spatial microdata using IPF.

2.2 Performing IPF on the CakeMap data

2.3 Processing the output

2.4 Mapping the results

3 Applying the technique in the real world

Here we will continue with the cakeMap model we ran in the previous section.

3.1 Model checking and validation

To make an analogy with food safety standards, openness about mistakes is conducive to high standards (Powell et al., 2011). Transparency in model verification is desirable for similar reasons. The two main strategies are 1) comparing the model results with knowledge of how it *should* perform *a-priori* (model checking) and 2) comparison between the model results and empirical data (validation).

A proven method of checking that data analysis and processing is working is wide ranging and continual visual exploration of its output (Janert, 2010).

Beyond typos or simple conceptual errors in model code, more fundamental questions should be asked of spatial microsimulation models. The validity of the assumptions on which they are built, and the confidence one should have in the results are important.

3.2 Visualising the results

Visualisation is an important part of communicating quantitative data, especially so when the datasets are large and complex so not conducive to description with tables or words.

Because we have generated spatial data, it is useful to create a map of the results, to see how it varies from place to place. The code used to do this found in 'cMapPlot.R'. A vital function within this script is 'merge', which is used to add the simulated cake data to the geographic dataframe:[5]

Listing 9: The merge function for joining the spatial microsimulation results with geographic data. Compare with eq. (5)

```
merge(wardsF, wards@data, by = "id")
```

The above line of code by default selects all the data contained in the first object ('wardsF') and adds to it new variables from the second object based on the linking variable (in this case "id"). Also in that script file you will encounter the function `fortify`, the purpose of which is to convert the spatial data object into a data frame. More on this process is described in **?**. The final map result of 'cakeMapPlot.R' is illustrated below (fig. 4).

3.3 Analysis

Once a spatial microdataset has been generated that we are happy with, we will probably want to analyse it further. This means exploring it — its main features, variability and links with other datasets. To illustrate this process we will load an additional dataset and compare it with the estimates of cake consumption per person generated in the previous section at the ward level.

The hypothesis we would like to test is that cake consumption is linked to deprivation: More deprived people will eat unheathily and cake is a relatively cheap 'comfort food'. Assuming our simulated data is correct — a questionable assumption but lets roll with it for now — we can explore this at the ward level thanks to a dataset on modelled income from neighbourhood statistics.

Because the income dataset was produced for old ward boundaries (they were slightly modified for the 2011 census), we cannot merge with the spatial dataset based on the new zone codes. Instead we rely on the name of the wards. The code below provides a snapshot of these names and demonstrates how they can be joined using the 'join' function.

The above code first joins the two datasets together and then checks the result by seeing how many matches names there are. In practice the fit between old names and new names is quite poor: only 71 out of 124. In a proper analysis we would have to solve this problem (e.g. via the

[5] 'join' is an alternative to merge from the 'plyr' package also used in the 'cMapPlot.R' script that performs the same task. Assuming 'plyr' is loaded — 'library(plyr)' you can read more about join by entering '?join' in R.

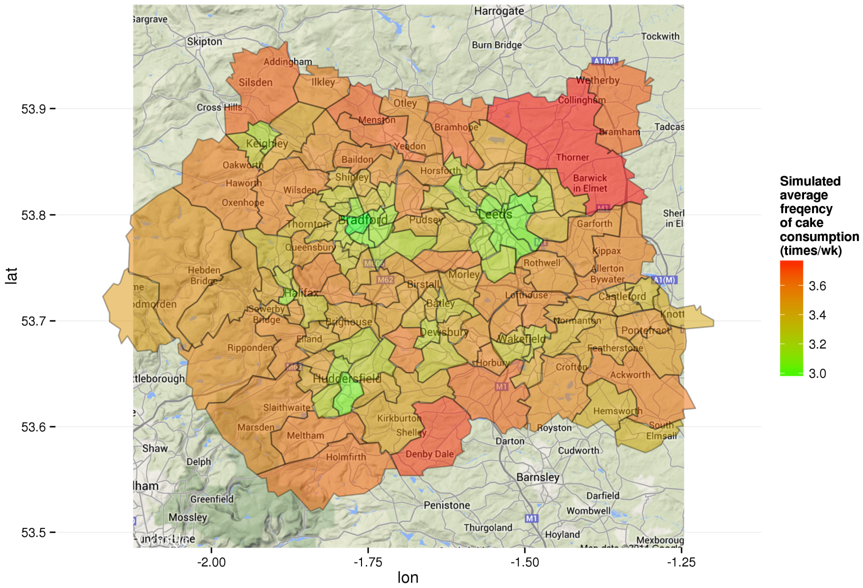


Figure 4: Choropleth map of the spatial distribution of average frequency of cake consumption in West Yorkshire, based on simulated data.

Listing 10: The merge function for joining the spatial microsimulation results with geographic data. Compare with eq. (5)

```
wards@data <- join(wards@data, imd)
summary(imd$NAME %in% wards$NAME)
##       Mode    FALSE     TRUE     NA's
##    logical       55       71        0
```

command `pmatch` whhich stands for partial match). For the purposes of this excercise we will simply plot income against simulated cake consumption to gain a feeling what it tells us about the relationship between cake consumption and wealth (fig. 5).

The question raised by this finding is: why? Not why is cake consumption higher in wealthy areas (this has not been established) but: why has the model resulted in this correlation? To explore this question we need to go back and look at the individual level data. The most relevant constraint variable for income was class. When we look at the relationship between social class and cake consumption in the Dental Health Survey, we find that there is indeed a link: individuals in the highest three classes (1.1, 1.2, 2) have an average cake intake of 3.9 cakes per week whereas the three lowest classes have an average intake of 3.7. This is a relatively modest difference but, when averaging over large areas, it helps explain the result. The class dependence of cake consumption in the Dental Health Survey is illustrated in fig. 6.

4 Discussion

What have we learned during the course of this practical tutorial? Given that it was geared towards beginners, from one perspective one could say ‘not a lot’ relative to the many potential applications. The emphasis on understanding of the basics was deliberate: it is only on strong foundations that large and long-lived buildings can be constructed and the same applies to

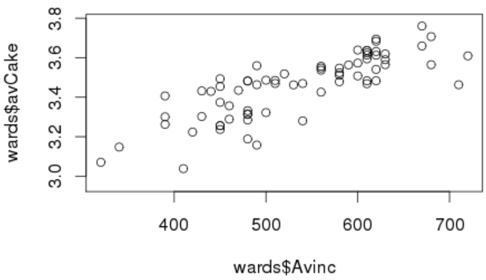


Figure 5: Relationship between modelled average ward income and simulated number of cakes eaten per person per week.

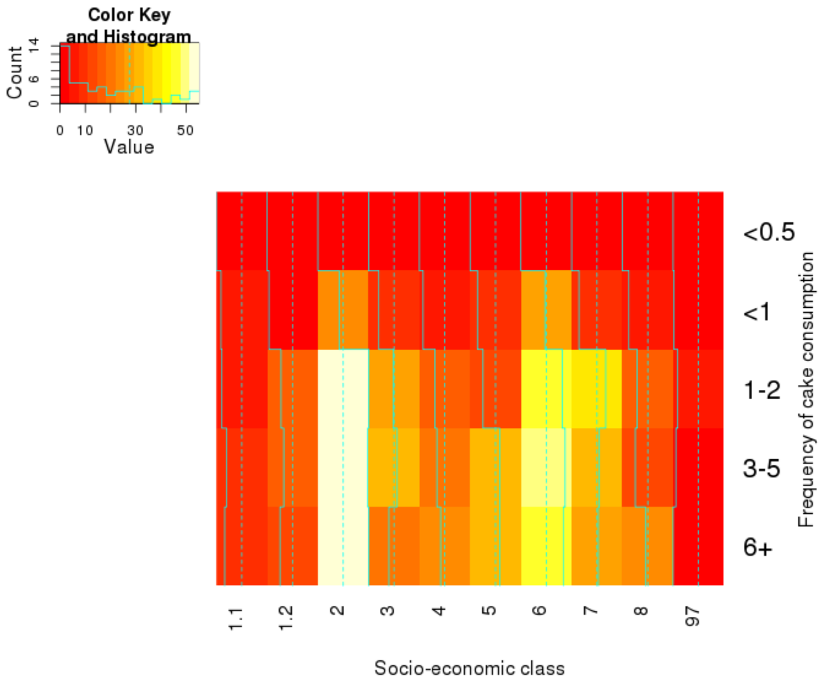


Figure 6: Relationship between social class and cake consumption in the individual level data.

geographical research.

The focus on strong foundations is also sensible because the number of people actively involved in spatial microsimulation research, let alone development of new software and methods, is very small. Specialist skills and understanding is needed to join this research area and a central outcome of this course has been to provide these. More specifically, we have learned how to:

- Perform iterative proportional fitting, one of the commonly used techniques for spatial microsimulation, by hand, in a spreadsheet and in R.
- Iterate the process any number of times using pre-made R code.
- Use simple queries and visuals to check the process has worked as expected.
- Process the spatial microdata to create estimates of target variables and join this with output with geographics zones.
- Map the result and begin to look for explanation in the dataset.

One important thing that the cakeMap example should have taught you is about the dangers and limitations of using spatial microsimulation. Most of the people who read articles based on the spatial microsimulation will not understand the technical details of the method. Therefore, great care must be taken not to mislead.

It is, for example, untrue to say that we have demonstrated cake consumption to be higher in wealthy areas of Leeds. We have done nothing of the sort! We have merely created a synthetic individual-level dataset for Leeds based on 3 constraint variables and allocated these individuals to zones. This spatial microdataset is not 'real'. We can have a high level of confidence that the values for the constraint variables are correct for this spatial microdataset but the joint distribution (e.g. the cross tabulation between age and car ownership) will tend towards the average contained within the survey microdataset and is likely to mask regional variability. In an ward where the proportion of young drivers is very high, for example, our model result is likely to provide underestimates of the numbers of young people owning cars whilst overestimating the number of older people owning cars.

The estimates generated for the target variable — cake consumption — are likely to be even further from reality. They simply reflect the relationship between cake consumption and a selection of constraint variables at the national level. Do we have good evidence to suggest that age/sex, socio-economic class and household car ownership are good proxies of cake consumption across different parts of the country? No. So we should treat the results as what they are: an amusing example of what you can do with spatial microsimulation, not a reliable description of the real world. To validate this example we would need to conduct some kind of randomised survey in the target wards to identify whether cake consumption really does vary in the ways described by the model. Most likely we would find that it does not.

This brings us nicely on to the final point of discussion: how you use spatial microsimulation in your own research. This of course is up to you, but during the process of this course we hope that you have picked up some ideas about best practice in the field. These include:

- Ensure, to the extent possible, **reproducibility in your method and findings**. This includes, at a minimum, clear description of the input data, explanation of the method used and the software needed. As illustrated with the cakeMap example, it is now relatively easy to ensure complete reproducibility even in complex analyses. This will not always be possible due to confidentiality of input data. However, the creation of an example dataset and provision of code should always be possible. This is highly recommended as it will greatly help others reproduce your findings (improving the scientific credibility of your research), provide a learning opportunity for yourself and others and increase the probability of other academics citing your work.

- Use spatial microsimulation only when it is the **most appropriate tool for the job**. This means that if there are alternatives such as geographically weighted regression analysis or analysis of large secondary datasets, these should be considered beforehand. Generating an entirely new individual-level dataset is not to be taken lightly and risks distracting from more grounded research. Thus it should be seen as an *addition to* rather than a *replacement for* more established methods.

- When spatial microsimulation is used, **be transparent about its underlying assumptions and limitations**. Simply forgetting to include the word 'simulated' in the caption of fig. 4, for example, could lead the reader to believe it is actual cake consumption that is being described. In the case of a cakeMap this may not matter but in areas of public health and the environment the consequences of such oversight could be deadly.

Underlying each of these points is a wider responsibility: to communicate one's research with clarity and transparency. Too much modelling research is shrouded in a cloud of jargon, unstated assumptions and verbose English. If nothing else, this tutorial should provide guidance on how to improve standards in the field and move towards best practice for reproducibility (see (**?**))

Spatial microsimulation is a powerful tool. Like any powertool, it can acheive very useful results for its user but can also cause great harm if used incorrectly. Think of a pneumatic drill: this could be used to build new public infrastructure such as bicycle paths. It could also, in clumsy hands, be used to destroy existing infrastructure. The same applies to spatial microsimulation: at best it can greatly help out with complex research questions such as the distributional impacts of new transport policies (Lovelace et al., 2014). At worst, it can waste valuable research time, create misleading results and shroud academic research behind an impenetrable wall of jargon.

We have little doubt that the vast majority of people will the former option. This course has hopefully equipped its students with the tools to pursue this lofty aim.

5 Glossary

- **Algorithm**: a series of computer commands which are executed in a well defined order. Algorithms process input data and produce an output.

- **Combinatorial optimisation** is an approach to spatial microsimulation that generates spatial microdata by randomly selecting individuals from a survey dataset and measuring the fit between the simulated output and the constraint variables. If the fit improves after any particular change, the change is kept. Williamson (2007) provides a practical user manual for implementing the technique in code.

- **Data frame**: a type of object (formally referred to as a class) in R, data frames are square tables composed of rows and columns of information. As with many things in R, the best way to understand dataframes is to create them and experiment. The following creates a dataframe with two variables: name and height:

  ```
  data.frame(name = c("Robin", "Phil"), height.cm = c(172, 174))
  ```

  Note that each new variable is entered using the command `c()`. This is how R creates objects with the *vector* data class — a one dimensional matrix — and that text data must be entered in quote marks.

- **Deterministic reweighting** is an approach to generating spatial microdata that allocates fractional weights to individuals based on how representative they are of the target area. It differs from combinatorial optimisation approaches in that it requires no random numbers. The most frequently used method of deterministic reweighting is IPF.

- **Iteration**: one instance of a process that is repeated many times until a predefined end point, often within an *algorithm*.
- **Iterative proportional fitting** (IPF): an iterative process implemented in mathematics and algorithms to find the maximum likelihood of cells that are constrained by multiple sets of marginal totals. To make this abstract definition even more confusing, there are multiple terms which refer to the process, including 'biproportional fitting' and 'matrix raking'. In plain English, IPF in the context of spatial microsimulation can be defined as *a statistical technique for allocating weights to individuals depending on how representative they are of different zones*. IPF is a type of deterministic reweighting, meaning that random numbers are not needed to generate the result and that the output weights are real (not integer) numbers.

6 References

Janert, P.K., 2010. Data analysis with open source tools. O'Reilly Media.

Kabacoff, R., 2011. R in Action. Manning Publications Co.

Lovelace, R., Ballas, D., 2013. Truncate, replicate, sample: A method for creating integer weights for spatial microsimulation. Computers, Environment and Urban Systems 41, 1–11.

Lovelace, R., Ballas, D., Watson, M., 2014. A spatial microsimulation approach for the analysis of commuter patterns: from individual to regional levels. Journal of Transport Geography 34, 282–296.

Powell, D.a., Jacob, C.J., Chapman, B.J., 2011. Enhancing food safety culture to reduce rates of foodborne illness. Food Control 22, 817–822.

Williamson, P., 2007. CO Instruction Manual: Working Paper 2007/1 (v. 07.06.25). Technical Report June. University of Liverpool.