User Guide to Rgeoprofile (version 1.2)

Welcome to the tutorial of Rgeoprofile, an R package for locating the sources of disease outbreaks, alien species invasions, animal foragers or whatever else you would like! The method is explained fully in a paper in press at *Methods in Ecology and Evolution* – in the meantime, a preprint is available from Mark Stevenson m.stevenson@qmul.ac.uk or from Steve Le Comber at s.c.lecomber@qmul.ac.uk.

This document is a rough walkthrough of the key functions that you can use to go from loading in your data to producing lovely geoprofiles layered over Google maps.

To get you started, there are simulated 'crime sites' and 'suspect sites' on the Le Comber Lab website (http://evolve.sbcs.qmul.ac.uk/lecomber/sample-page/geographic-profiling/geographic-profiling-in-r/). The suspect sites relate to Buckingham Palace, 10 Downing Street, the old GLC and the Mayor of London's office.

There are 10 simple steps to using this package:

1) Download and install the package

2) Import your data

3) Run LoadData()

4) Run ModelParameters()

5) Run GraphicParameters()

6) Run CreateMaps()

7) Run RunMCMC()

8) Run ThinandAnalyse()

9) Run PlotGP()

10) OPTIONAL: Run Reporthitscores()

Now this may seem like a lot but only two of these functions take any time at all. Steps 1+2+3 ensure the package is installed and your data loaded in the correct format. Steps 4+5 set up the parameters to be used by the model and the plotting functions. Step 6 is where you can have a look at your data on a map to check it and where the prior on source emergence is calculated. Step 7 is where the work takes place and is a more time intensive step. Step 8 thins the MCMC samples to remove

autocorrelation and makes the final geoprofile; again this step can take a little bit of time. Step 9 is the fun bit, where lovely maps with nice looking contours are created. Feel free to re-run step 9 as many times as you like to get perfect maps. Finally, step 10 is optional. If you have imported suspect or source location data, then you can generate a list of hit scores of these sources (how high up on the surface they are).

1) Download and install the package

Go to our website, the page Geographic Profiling in R in the Le Comber Lab tab. Find the link to Rgeoprofile and download it. The website is here:

http://evolve.sbcs.qmul.ac.uk/lecomber/sample-page/geographic-profiling/geographic-profiling-in-r/

Once you have downloaded the package file, place it in your working directory. If you don't know where that is, use:

```
getwd()
```

Once you have the package file in your working directory, run the following code to install the package:

```
install.packages("~/Rgeoprofile_1.2.tar.gz", repos = NULL, type = "source")
```

Then load the package into your library and install the other dependent packages:

```
library(Rgeoprofile)
```

You should receive a few warnings from the other packages that install…but most importantly none from our package! That completes the installation phase.

2) Import your data

The next step involves reading in some data into R for the model to run on. If you are a moderately frequent R user, you should be familiar with the various ways you can do this, but I will quickly review one way of doing this.

There are some important steps in creating the data file so that it contains the correct type of data. Our package uses Longitude and Latitude data as the raw input. The data file should be a text tab delimited file, with two columns only. The first column should have longitude values and the second column should have latitude values. **THEY MUST BE IN THIS ORDER**. You do not need column headings (but it is OK to have them) and there should be no other information in the file. The longitude and latitude data should both be in decimal format.

Save this file somewhere and read it into R. I prefer to use the following to do this:

```
mydata<-read.table(file.choose(), header=F)
```

This should open a window allowing you to choose your file. Then it will be loaded into R as the object `mydata`. That's your data loaded in, you can at this point also load in some source data. The same rules apply to source data, long then lat in two columns only with no other information. Load it in using the same function:

```
mysources<-read.table(file.choose(), header=F)
```

At this point you are good to go. You have your data loaded in and optionally you also have some source data.

3) Run `LoadData()`

The next step is to run the function `LoadData()`. This function reads your data objects and reads off some important information (such as data length, number of source points etc.) to pass to other functions later. We also used this point to load in a custom colour scheme for the MCMC. **THIS IS AN ESSENTIAL STEP AND CANNOT BE SKIPPED**.

There are a few options for `LoadData()`:

```
LoadData(Data=mydata, Sources= "NULL")
```

You have two arguments to pass to `LoadData()`, `Data` and `Sources`. `Data` is the data file that you have just loaded in. Sources is a multi-use argument, if left as the default "NULL" then `LoadData` will assumed that you are not loading in source data. If you assign a matrix or data frame to this argument then these will become the source data. For example:

```
LoadData(Data=mydata, Sources= mysources)
```

Once you have done this, you have successfully run the first essential function, and you are ready to move on!

4) Run `ModelParameters()`

This next step sets the parameters for the model and the MCMC that will be passed to other later functions. You also have the option to plot the prior on the parameter sigma so that you can examine it and make sure it is sensible. THIS IS ANOTHER ESSENTIAL STEP AND CANNOT BE SKIPPED. `ModelParameters()` is built with a wide range of default values and can be run as follows:

```
ModelParameters()
```

But it is important for the user to be clear about what all these parameters are, and to change them so that they are sensible. For a full description of the model and the parameters in it please see (Verity et al. in press). The full list of arguments for this function is:

```
ModelParameters(sigma = 1, tau = "DEFAULT", minburnin = 100,
maxburnin = 1000, chains = 5, Samples = 10000)
```

The arguments are as follows:

a) `sigma`: A critical parameter which sets the parameter sigma. This default value may have very little relevance to the problem being approached so needs to be altered to give a sensible prior on sigma. The units of this expectation are in decimal degrees. For example, for human movement in urban settings, a value of 0.01, which corresponds to a standard deviation of the bivariate normal distribution around the sources of about 900m in London, is probably appropriate.

b) `tau`: the parameter that sets the prior on source emergence. Basically should always be set to DEFAULT, that way the prior is determined from the spread of the data points. This means that we will expect sources within two standard deviations of a normal distribution centred on the data – essentially that our sources will be most likely near the data points.

c) `minburnin`: An MCMC parameter, sets the minimum number of iterations for the MCMC burn-in once it has hit the accepted convergence value of having a shrink factor of less than 1.1. Defaults to 100, which is pretty reasonable, can be set to any positive value, best somewhere in the 100-500 range.

d) `maxburnin`: An MCMC parameter, sets the maximum number of iterations for the MCMC burn-in. Defaults to 1000, which is pretty reasonable, can be set to any positive value, best somewhere in the 500-3000 range. If you are not achieving convergence, then this value should be increased; you might also consider using more chains.

e) `chains`: An MCMC parameter, sets the number of chains you release into the MCMC to sample with. Defaults to 5, and so far I haven't had to increase this number. If you struggle to find convergence and feel that your samples are being stuck in local optima you may need to increase this value. Best in the 5-10 range.

f) `Samples`: Another MCMC parameter and one of the most important. Sets the number of samples that you take. This will determine how long it takes you to run RunMCMC(), if this number is large RunMCMC may take a long time to run. Defaults to 10000, which will produce a good surface but may take a while to run. If you want a quick and dirty surface to have a look at, you could set this to as low as 1000. If the number of samples is too low, beware of autocorrelation.

So by this point you have loaded in your data, extracted some key measures and set your model and MCMC parameters. Time to move on to the next step.

5) Run `GraphicParameters()`

Another critical function, this one sets the parameters for plotting and the creation of Google maps. These parameters are again passed to other functions to be used later on.

`GraphicParameters()` is very similar in design to `ModelParameters()`, having a large amount of default values loaded in. This means you can just run it with:

```
GraphicParameters()
```

But again it is important for you to be familiar with all of the inputs to this function as they will determine the style of many of the nice images you may well want to make. The full breakdown with all of the arguments is here:

```
GraphicParameters(Guardrail = 0.05, nring = 20, transp = 0.4,
gridsize = 640, gridsize2 = 300, MapType = "roadmap", Location =
getwd(), pointcol = "black")
```

The arguments are as follows:

a) `Guardrail`: A parameter that sets the search area and size of plotting windows of Google maps. Essentially it is the % extra distance to leave around a window the size of the data points. If set to 0.05 it will leave 5% extra space around the data points in your models surface and in the maps produced. This defaults to 0.05 but can easily be changed. Values more than 1 will produce very zoomed out maps. MUST BE POSITIVE.

b) `nring:` The number of rings on contours and filled contours on surfaces produced. Defaults to 20 which produces a nice number of contours on most maps. Generally if you have a large zoomed out map you will need more contours and if you have a zoomed in one you will need fewer. You can change this parameter on the fly later when plotting so that your image looks great.

c) `transp`: The transparency of image layers to be overlaid on Google maps. 0 is completely transparent and 1 is completely opaque. Defaults to 0.4 which works nicely on 'roadmap' style maps, may need to be reduced to 0.2/0.3 for other map types such as 'hybrid' or 'satellite'.

d) `gridsize`: The gridsize of the Google maps, the maximum set by Google is 640*640. This value defaults to 640. It is not recommended that this value is lowered as it will reduce the resolution of the resulting maps.

e) `gridsize2`: The resolution of the geoprofile produced by the MCMC. Defaults to 300. If this parameter is increased it will substantially increase the time taken for the later function `ThinandAnalyse()` to run. This value works well at 300, but if you want to produce really nice high resolution profiles you may need to increase this parameter.

f) `MapType:` Another key parameter, it sets the format of the Google maps produced. Available options are: "roadmap", "mobile", "satellite", "terrain" or "hybrid". Roadmap looks nice in urban environments and satellite may be better in rural/agricultural and wilderness environments.

g) `Location:` The filepath that Google Maps, surfaces and outputs will be saved. Defaults to the working directory. Needs to be working for map generation to take place, check your computer has given R access permissions (Windows machines in particular can be very annoying).

h) `pointcol:` The colours of the data points on Google map images produced by `PlotGP` and `CreateMaps`. This value defaults to "black" you can use any colour argument e.g. "blue", 2, "lightgreen" etc. Works well with black, note that blue can cause problems as the default colour for source points is blue.

Now you are done setting all your parameters, time to get on with the important task of using them! The next stage is to plot your data and get a feel for it.

6) Run `CreateMaps()`

The next function to use will both let you plot your data and look at it on a lovely shiny Google Map and also do a load of other useful calculations in the background, like getting all the map data and creating the prior surface on the source locations. Just run:

```
CreateMaps(PlotPrior = T)
```

This function only has one argument:

a) `PlotPrior:` Comes in `TRUE` or `FALSE` form only, if `TRUE`, will plot the filled contour of the prior surface overlaid on your map, turn this off if you just want to look at your data on the map.

Great, this is nice and easy, and feel free to use this as many times as you like at this point. Go back and change the graphic parameters to alter the plots (to look at satellite or hybrid maps for example).

It is very important to do a sense check at this point, and make sure your data are appearing in the right place in the world, and that it is not upside down or back to front! Furthermore, the end result is a normal plot window, so you can add extra points or objects to the plot as you choose.

A good extra thing to do at this point might be to plot your sources on this map (if you have them). Can be done with some extra freestyle code like:

```
points(mysources , pch = 15 , col = "blue")
```

Once you are happy with your data and have saved any of the plots that you need you can move on to the next step. And it's a big one!

7) Run `RunMCMC()`

This is the big daddy step, so before you run it make sure everything is correct! Check your parameters objects and make sure you are happy with your data. This step runs the MCMC burn-in, sampling and produces diagnostic plots of convergence and autocorrelation.

There are no options to pass to this function: it already has all the information it needs, so just go ahead and run it.

To go through quickly what it is doing:

a) Integration over prior on concentration parameter alpha. This will be handily pointed out to you on screen. Sit back and wait while this happens, it shouldn't take too long (unless you have a giant dataset).

b) Run the MCMC burn-in for the specified length of time. The convergence of chains is shown at this point using the Gelman-Rubin diagnostic statistic, evaluated on the log-likelihood of the model. Once convergence falls under 1.1 and the minimum number of iterations has been reached then the burn-in will terminate. Basically, the chains are all reaching similar positions and will eventually get to the point where the model has enough samples to move onto the next step.

c) Take MCMC samples as defined by `Samples` set in `ModelParameters()`. The plots will show the group assignment of the data points every hundred iterations. This is the fun bit; you can watch the grouping in the plots as the points dance around in different colours. This is the model slowly improving its estimate of the grouping.

d) Combine information across all integrated surfaces and produce diagnostic plots. The two plots produced should be: the auto-correlation of the MCMC and the realised number of sources created by the model. The auto-correlation is important and should determine the degree of thinning used in the next step. You should look to see how often the line crosses up and down through the x axis: if it is roughly every 100 times you must thin your sample by 100, and if it is every 1000 then you

should thin it by 1000. The realised number of sources is the model's best guess at the number of source locations. **NOTE**: this plot is quite interesting and you may want to save it now as it is difficult to get back.

8) Run `ThinandAnalyse()`

This is a very important next step (aren't they all!) that thins the MCMC samples and analyses them to construct the geoprofile. This function takes the posteriors obtained by the MCMC and thins them to remove autocorrelation as determined by the user. The plots produced by RunMCMC include and auto-correlation plot. This can be used to assess the degree of thinning needed. A value of thinning = 100 would keep one value for every 100 samples. After this step, the remaining samples are used to construct a geoprofile. This is determined by the single argument thinning, which defaults to 100. Once you have had a look at the auto-correlation plot, you can set thinning to the appropriate level and run:

```
ThinandAnalyse(thinning = 100) ## Or whatever is correct ##
```

This function will rapidly thin the dataset, then go about constructing the geoprofile from the remaining posterior samples.

NOTE: if you remove too many samples when thinning the number of samples left to construct the geoprofile will be small, introducing further error. If autocorrelation is large it is recommended that more samples are taken when using `RunMCMC()`.

Once this is done, you will have created your geoprofile and are finally ready to plot it on a map.

9) Run `PlotGP()`

This is the final mandatory step to view your geoprofile! This function has a few options, and many of these can be reset by changing the options set in `GraphicParameters()` earlier. You can just run `PlotGP()` but it is again best to understand what the arguments are so you can configure your maps correctly:

```
PlotGP(Window = "DEFAULT" , Legend = T)
```

The arguments are as follows:

a) `Window`: This decides between three specific types of plots. The first option is "DEFAULT" which plots a default sized map given the guardrail around the data points. The size of this guardrail was set earlier with `GraphicParameters()`, so feel free to go back and change this now. Window can also be set as a list of x and y limits what you wish the plot window to be. Pass a list to the argument in the format: `list(c(minimum x axis, maximum x axis),c(minimum y axis, maximum y axis))`. The final option (and also the coolest) is "ZOOM". "ZOOM" produces a map of the default size and then allows the use of `locator(2)` clicks to define a zoomed in window: a new map will then be drawn given the size set by `locator(2)`. This lets you highlight sections of the map and then produce detailed zoomed in sections. The Google map will automatically rescale and provide greater resolution as you zoom in. This is easier to use than it probably sounds: simply run PlotGP (Window = "ZOOM") to draw the full profile, then select the window and click in two places to set the area of the zoomed map. This will then extract the relevant data from Google Maps and redraw the relevant part of the profile.

b) `Legend:` A simple TRUE or FALSE option. If TRUE it will plot a simple legend with the parameters, points (and sources if you have them) and the hit score bandings. We felt we should include this option as the bandings can be hard to calculate each time. This will make the plot window containing the geoprofile unavailable for further calls to `plot` or `points`. Set to `Legend` to FALSE to remove the call to `split.screen` that is used to create the legend.

Using these two settings, you can have six options to plot:

1) plot a map with exact x and y limits defined using a list

2) plot a map with a default area defined by a guardrail.

3) Plot a map that can be zoomed in using two clicks.

All three of the above options can be plotted either with or without a legend, leading to six options total. On top of this you can change all the options in `GraphicParameters()` to give hundreds

of options in plotting your maps. If you are still not content, disable the `Legend` and simply use further calls to functions like `points`, `legend` and `objects` to add things to your plots.

Admire your beautiful geoprofile, hopefully it shows something interesting…

10) Run `Reporthitscores()`

This is an optional final step. This step will only work if you have loaded in source data, otherwise you will get the following message: `"No sources present"`. This function outputs a table with the hit scores of each of your sources. It doesn't have any arguments but needs the full final outputs of `ThinandAnalyse()` to be able to work. Simply run:

```
Reporthitscores()
```

Returns and object called `hit_output`, a data frame of three columns and a number of rows equal to the length of sources. The rows are: the longitude of the sources, the latitude of the sources and the hit score of the source in question.

For a discussion of hit score values and their use in geographic profiling in Biology see: Stevenson et al. (2012), where this method of assessing search strategies is defined and discussed.

So there you have it, a full guide to using the package Rgeoprofile. If you have any problems or questions (or praise… well, you never know) please get in touch with me: Mark Stevenson m.stevenson@qmul.ac.uk , as I am the main architect and software maintainer. Have fun, happy profiling and may your chains always converge!

References

Stevenson, M. D., Rossmo, D. K., Knell, R. J., & Le Comber, S. C. (2012). Geographic profiling as a novel spatial tool for targeting the control of invasive species. *Ecography*, *35*(8), 704-715.

Verity, R., Stevenson, M. D., Rossmo, D. K., Nichols, R. A., & Le Comber, S. C.(2014). Spatial targeting of infectious disease control: identifying multiple, unknown sources. *Methods in Ecology and Evolution* (in press).