**Using R as a GIS for spatial analysis**

*Daniel Chapman, CEH Edinburgh*

In this session we will illustrate some of R’s potential as a GIS and for spatial analysis by modelling the effect of distance to water on human population density in the UK. Obviously the point is to illustrate R’s potential rather than try to produce a sensible model for the determinants of human population density.

Some of the R functionality that will be covered includes:

1. Reading in GIS files (ESRI shapefiles and rasters).
2. Working with coordinate reference systems or projections in R
3. Basic visualisation of spatial data
4. Geoprocessing of rasters and vector data
5. Regression modelling of spatially autocorrelated data

The input files are contained in the folder ‘GIS\_files’ available from the GitHub page. They are:

1. GBR\_adm2.shp – shapefile of UK administrative boundary polygons
2. GBR\_water\_areas\_dcw.shp – shapefile of large water bodies in the UK, from Digital Chart of the World
3. gbr\_msk\_pop.grd – raster of UK population density, in R’s native raster .grd format

This document explains the main steps in the analysis. The workflow is contained in the R script accompanying this document (R\_as\_a\_GIS.R), which has been tested on R x64 3.3.1 and 3.3.2. It is recommended to run the script from the basic R GUI rather than RStudio, as the latter can become unstable with spatial datasets.

The workflow can be summarised as:

1. Generate a random stratified sample of locations in the UK, for which we obtain data to model
2. Calculate the distance to water bodies at the sampling locations
3. Estimate the human population density at the sampling locations
4. Use a Generalised Least Squares (GLS) model to test the association between human populations density and distance water, while accounting for spatial autocorrelation in the model errors.

Before working through the document, please see the R script for instructions on:

1. Installing and loading the required R libraries.
2. Setting the random number seed (optional to hopefully allow replication of the exact results in this document).
3. Setting the working directory to the directory in which where you have saved the folder ‘GIS\_files’. Please do not set the working directory to ‘GIS\_files’ itself as this makes loading the shapefiles difficult (see below).

**Step 1 - produce random sampling locations across the UK, stratified by administrative area**

We first load the administrative polygons shapefile:

# List GIS layers that can be read from the folder 'GIS\_files'

ogrListLayers**(**dsn**=**"GIS\_files"**)**

# read in UK administrative polygons shapefile as a SpatialPolygonsDataFrame object

UK\_wgs84 **=** readOGR**(**dsn**=**"GIS\_files", layer**=**"GBR\_adm2"**)**

UK\_wgs84 # print a summary of the object

plot**(**UK\_wgs84**)**

Next, we need to re-project the polygons to the British National Grid (BNG) coordinate reference system. To do this, we need to know the Proj4 string for the BNG. To find this, visit <http://spatialreference.org/> and enter ‘OSGB’ in the search box at the top right of the page. From the results, select ‘EPSG:27700: OSGB 1936 / British National Grid’ and click the link to ‘Proj4’. You should see the result ‘+proj=tmerc +lat\_0=49 +lon\_0=-2 +k=0.9996012717 +x\_0=400000 +y\_0=-100000 +ellps=airy +datum=OSGB36 +units=m +no\_defs’ which we can use in R.

# Convert coordinate system from WGS84 (long/lat) to OSGB 1936 (British National Grid)

# You can find the Proj4 string for OSGB1936 from www.spatialreference.org

UK **=** spTransform**(**UK\_wgs84, CRSobj**=**"+proj=tmerc +lat\_0=49 +lon\_0=-2 +k=0.9996012717 +x\_0=400000 +y\_0=-100000 +ellps=airy +datum=OSGB36 +units=m +no\_defs"**)**

UK # note that the coord. ref has changed

plot**(**UK**)**

SpatialPolygonDataFrame objects contain both the geometry of the polygons and an ‘attribute table’ stored as a data.frame with information on each individual polygon. The data.frame is stored as a ‘slot’ called ‘data’, so can be accessed in full using the @ operator. Here we print a summary of the data.frame:

# examine the data.frame (attribute table)

summary**(**UK@data**)**

The data.frame can also be manipulated similarly to a standard data.frame. Here, we use the gArea function to calculate the area of each polygon, use the $ operator to add it as a column of the attribute table, and then plot a histogram of the results:

# Add polygon area in km2 as a field to the data.frame.

# gArea calculates the area in m2 as this are the units of the coord. ref,

# so we convert to km2

UK**$**area **=** gArea**(**UK, byid**=TRUE)** **/** **(**1000**\***1000**)**

summary**(**UK@data**)** # note that area is a new field of the data.frame

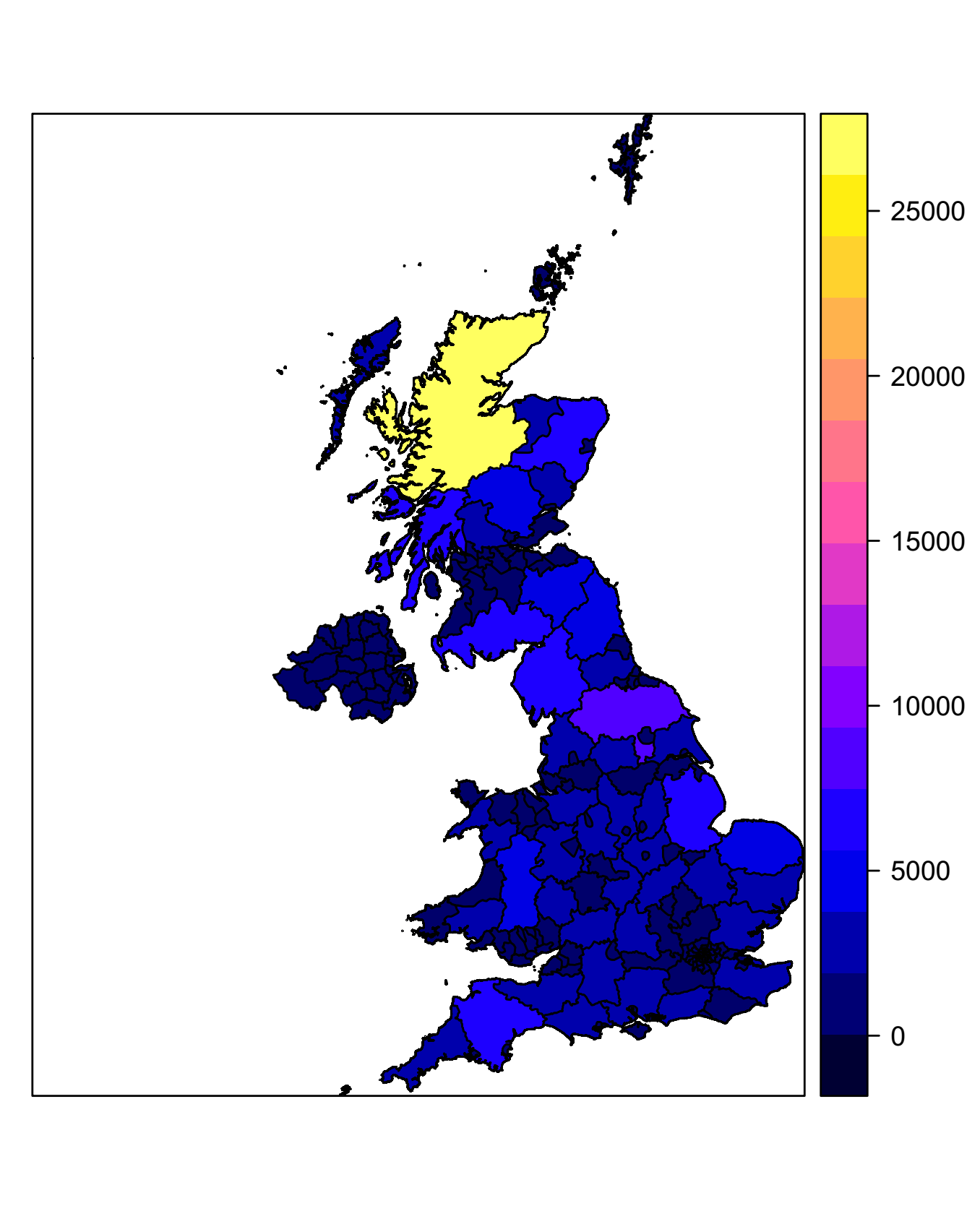
# access the field of the data.frame for plotting a histogram

hist**(**UK**$**area, br**=**100, col**=**"grey"**)**

To visualise the Spatial\* and Raster\* objects, the spplot function is very useful. Here we use spplot to shade the polygons by their area. Note that this relies on a field in the data.frame being called area:

# Make chloropleth map of UK, shaded by area

spplot**(**UK, "area"**)**



To improve the speed of processing using the UK polygon, we can use the gUnaryUnion function to ‘dissolve’ the individual polygons into the UK outline:

# Dissolve the UK multi-part polygons to a single UK outline (takes ~10 seconds on my machine)

UK\_out **=** gUnaryUnion**(**UK**)**

plot**(**UK\_out**)**

To improve visualisation of the UK polygon, we can use the gSimplify function to smooth the outline:

# Simplify the outline polygon to make plotting cleaner

UK\_out\_simple **=** gSimplify**(**UK\_out, tol**=**5000, topologyPreserve**=FALSE)**

plot**(**UK\_out\_simple**)**

# compare sizes of the polygon objects to show the effects of dissolving and simplifying...

format**(**object.size**(**UK**)**, units**=**"Mb"**)**

format**(**object.size**(**UK\_out**)**, units**=**"Mb"**)**

format**(**object.size**(**UK\_out\_simple**)**, units**=**"Mb"**)**

The next step is to generate a set of sampling locations for modelling, stratified by administrative area. This will be done by creating 4 random points within each administrative area. A major advantage of R is the ability to apply geoprocessing operations to individual elements of the Spatial\* object, in this case individual polygons. Here, the sapply function is used to operate on each polygon in turn. Individual polygons are selected using standard row number indexing, and the spsample function is used to produce random points within the selected polygon. The bind function is them used to convert the output from a list to a single SpatialPoints object:

# Use sapply and indexing to create a stratified sample of random

# locations in the UK, with 4 points per administrative area...

rndPtsList **=** sapply**(**1**:**length**(**UK**)**, **function(**idx**){**

P **=** UK**[**idx,**]** # select the polygon of interest

PT **=** spsample**(**P, n**=**4, type**=**"random"**)** # random sample 4 points in the polygon

return**(**PT**)** # return the SpatialPoints object

**})** # returns a list of SpatialPoints

class**(**rndPtsList**)**

# 'Bind' the list of SpatialPoints into a single SpatialPoints object...

rndPts **=** do.call**(**bind, rndPtsList**)**

rndPts

plot**(**UK\_out\_simple**)**

points**(**rndPts, pch**=**16, cex**=**0.5, col**=**"red"**)**

**Step 2 - calculate the distance from the random sampling points to major waterbodies**

First, we load in the major waterbodies shapefile as a SpatialPolygonsDataFrame and re-project to the British National Grid:

# Read in the UK major waterbodies polygons

water\_wgs84 **=** readOGR**(**dsn**=**"GIS\_files", layer**=**"GBR\_water\_areas\_dcw"**)**

water\_wgs84

# Re-project OSGB 1936.

# This time we can use the projection of the UK polygons in the code

water **=** spTransform**(**water\_wgs84, CRSobj**=**UK@proj4string**)**

water

# plot the waterbodies...

plot**(**UK\_out\_simple**)**

plot**(**water, col**=**"blue", border**=**"blue", add**=TRUE)**

Now we can estimate the distances from the sampling points to their nearest waterbody. Here, we illustrate two options for this – rasterization and direct calculation from the Spatial\* vectors.

To estimate by rasterization, we first create a template raster at 1x1 km resolution covering the whole of the UK and snapped to the British National Grid:

# First, create a bounding box for the UK as an extent object

myExtent **=** extent**(**c**(**0,66,0,122**)\***10000**)**

plot**(**myExtent, add**=TRUE**, col**=**"green4"**)**

# Create an empty raster in OSGB1936, with the desired extent and

# 1x1 km resolution

rTemplate **=** raster**(**ext**=**myExtent, res**=**1000, crs**=**projection**(**UK**))**

rTemplate # print summary to screen

We can now use the rasterize function to convert the waterbody polygons to a RasterLayer object aligned to the template raster. The initial result is the % cover of the waterbody polygons in each grid cell. We convert that to a raster with values of 1 for grid cells containing a waterbody and NoData (NA) for grid cells with no water body. This makes use of R’s useful abilities to apply functions and operators directly to values in the raster (here the > and == operators) and to access data values of the raster similarly to a vector or matrix with the square brackets:

# Convert the water polygons to a raster, using the template.

# Setting 'field=1' means the resulting raster has value of 1 where the

# cell contains water

rWater **=** rasterize**(**x**=**water, y**=**rTemplate, getCover**=TRUE)**

rWater

plot**(**rWater**)**

# Convert to binary (contains water or does not)

rWater **=** **(**rWater **>** 0**)**

# Convert 0 values to NA

rWater**[**rWater **==** 0**]** **=** **NA**

# Display the final raster...

plot**(**rWater, col**=**"blue"**)**

plot**(**UK\_out\_simple, add**=TRUE)**

We can now use the distance function to make a new RasterLayer containing the distance from each grid cell to the nearest grid cell containing a waterbody:

# Calculate raster of closest distance to the waterbodies in km

waterDist **=** distance**(**rWater**)** **/** 1000

plot**(**waterDist**)**

plot**(**UK\_out\_simple, add**=TRUE)**

To aid visualisation, will use the mask function to ‘clip’ the raster to the UK outline and apply a log transformation on the fly:

# Mask the distance to water by the UK outline

waterDist\_masked **=** mask**(**waterDist, UK\_out**)**

plot**(**waterDist\_masked, col**=**bpy.colors**(**100**))**

plot**(**UK\_out\_simple, add**=TRUE)**

# Display a histogram of the distances to water...

hist**(**waterDist\_masked**)**

hist**(**log1p**(**waterDist\_masked**))** # a ln(x+1) transform might look nicer

plot**(**log1p**(**waterDist\_masked**)**, col**=**bpy.colors**(**100**))**

plot**(**UK\_out\_simple, add**=TRUE)**

Finally, the extract function is used to obtain the values of the distance raster at the sampling points:

# extract the distance values at the random points

ptDist1 **=** extract**(**waterDist, rndPts**)**

It is also possible to estimate the distance to nearest water body directly from the Spatial\* objects using the gDistance function. This should give more accurate results:

# Alternatively, we can calculate the distance to the water polygons

# more accurately and more simply using vectors

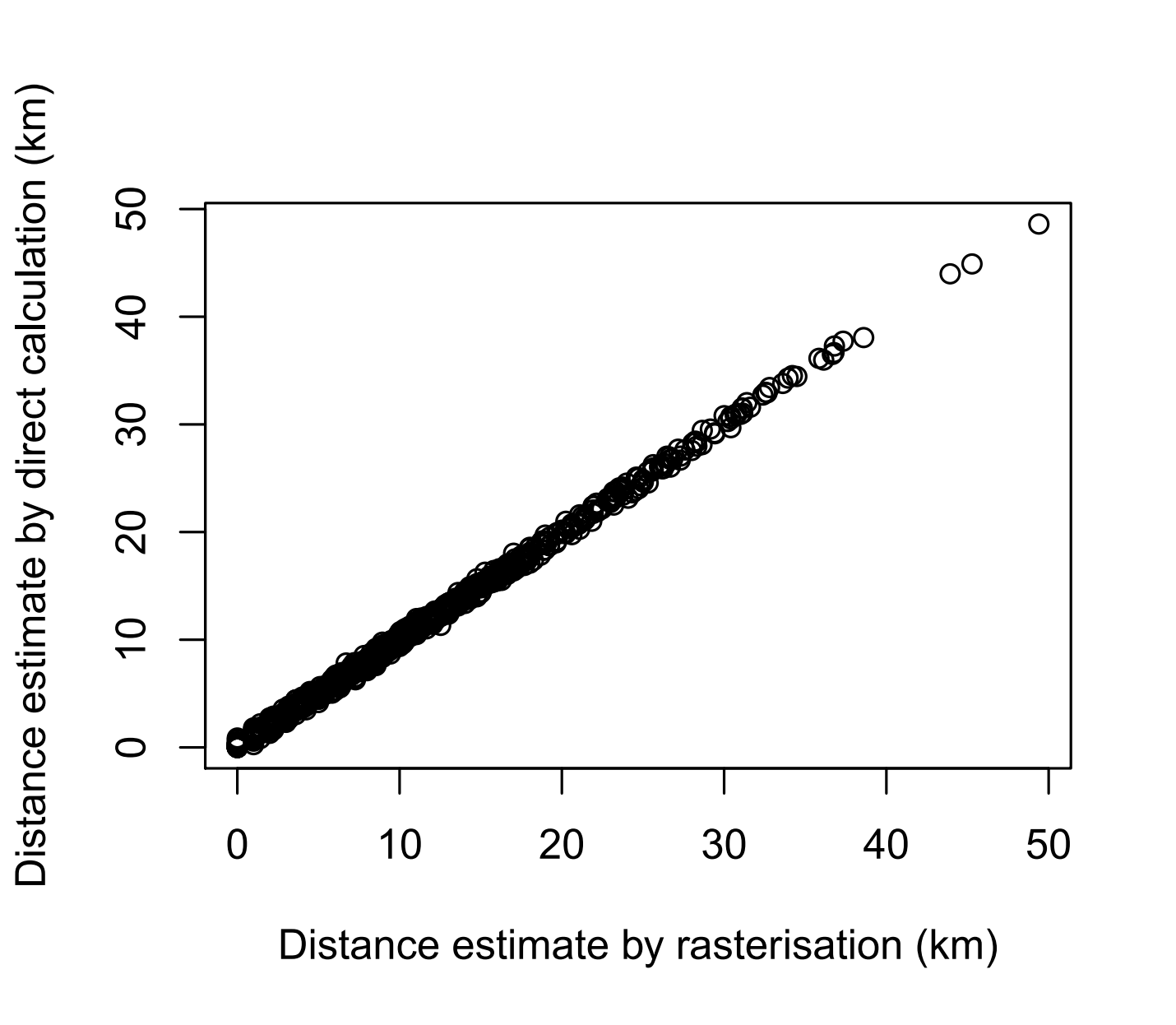
allDists **=** gDistance**(**spgeom1**=**water, spgeom2**=**rndPts, byid**=TRUE)** **/** 1000

dim**(**allDists**)** # returns a matrix of the distance from all points to all polygons

ptDist2 **=** apply**(**allDists, 1, min**)** # find the minimum distance for each point

plot**(**ptDist1, ptDist2, xlab**=**"Distance estimate by rasterisation (km)",

ylab**=**"Distance estimate by direct calculation (km)"**)** # the results are very similar



**Step 3 - estimate the human population density at each random sampling point**

Human population density is available as a raster, which can be loaded using the raster function:

# Read in the population density raster in 'grd' format...

rPop\_wgs84 **=** raster**(**"GIS\_files/gbr\_msk\_pop.grd"**)**

rPop\_wgs84

plot**(**log**(**rPop\_wgs84**)**, col**=**bpy.colors**(**100**))** # plot on the log scale

plot**(**UK\_wgs84, add**=TRUE)**

Some of the sampling points may be very close to the coast and beyond the coverage of the raster. To avoid having NAs in the extracted data, we need to replace the NoData values (NAs) in the raster with some values. For simplicity, here we replace them with the minimum value in the raster (2), though there are probably more sensible ways to do it. To do the replacement, we make use of the getValues function to obtain all the values on the grid:

# Because some sampling points are close to the coast and not on the population raster

# coverage, we will replace the NAs with lowest value in the raster

minVal **=** min**(**getValues**(**rPop\_wgs84**)**, na.rm**=TRUE)** # find the minimum value

rPop\_wgs84**[**is.na**(**rPop\_wgs84**)]** **=** minVal # assign the minimum value where the raster has an NA

plot**(**log**(**rPop\_wgs84**)**, col**=**bpy.colors**(**100**))** # plot on the log scale

plot**(**UK\_wgs84, add**=TRUE)**

The raster values at the points can be obtained directly using the extract function. Here, the points and the raster are in different projections, so R re-projects the points on the fly and gives a warning:

# Extract population density values at the random points by bilinear interpolation.

# Note the warning that they are not in the same projection,

# so the points were re-projected on the fly

ptPop **=** extract**(**rPop\_wgs84, rndPts, method**=**"bilinear"**)**

summary**(**ptPop**)**

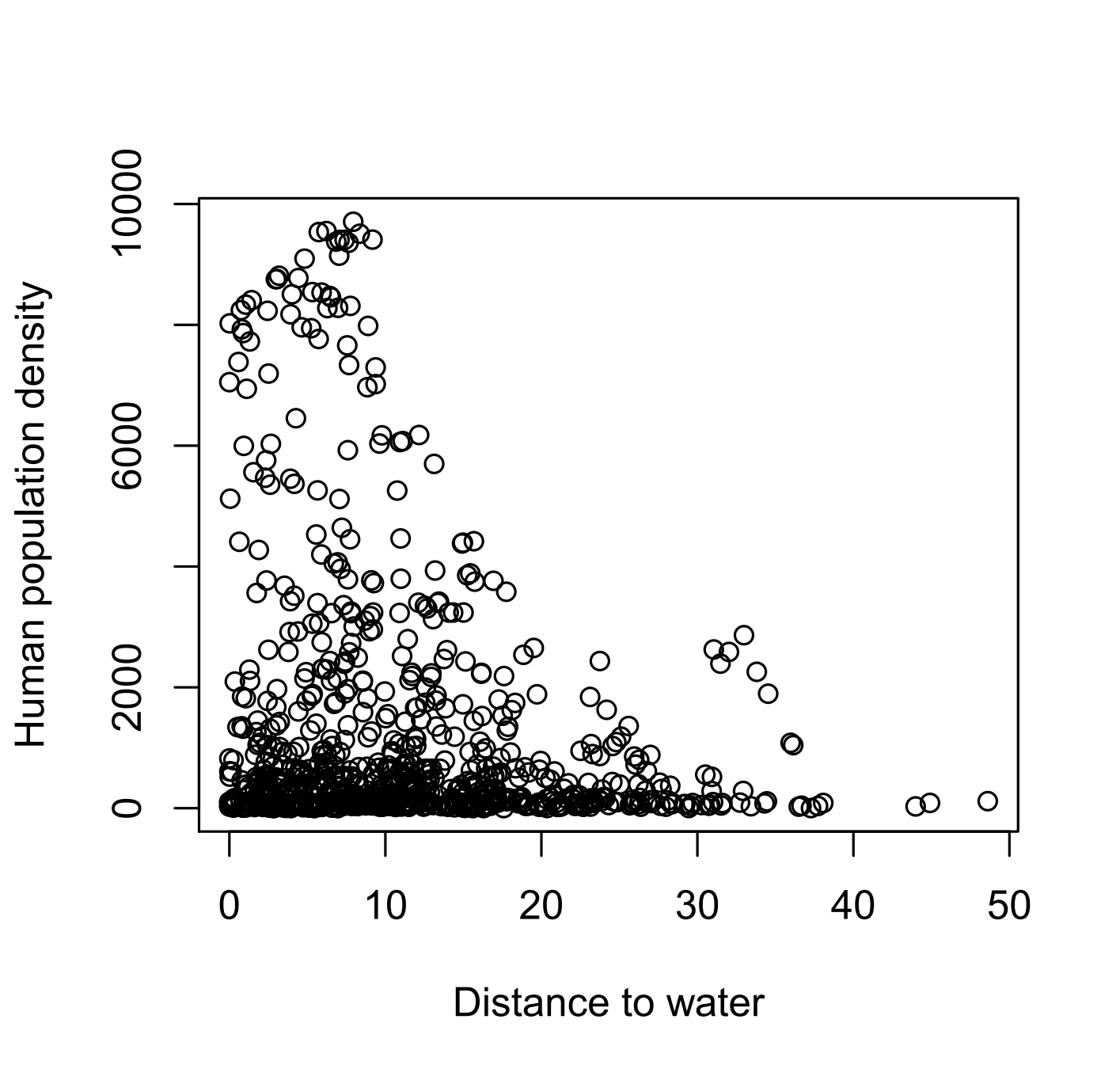
**Step 4 - Model the effect of distance to water on human population density at each random, accounting for spatial autocorrelation**

Now the GIS processing has been done, it is possible to test whether distance from water is correlated to human population density. Simple plotting of the data suggests the possibility of a negative relationship:

# Display the relationship between distance and human population

# at the random sampling points

plot**(**ptDist2, ptPop, xlab**=**"Distance to water", ylab**=**"Human population density"**)**



Histograms of the raw data are right-skewed, suggesting we should use log-transformation before analysis:

# Examine the distribution of the variables - suggests log transformations

hist**(**ptDist2**)**

hist**(**log1p**(**ptDist2**))**

hist**(**ptPop**)**

hist**(**log**(**ptPop**))**

Now all the required variables can be combined into a data.frame for modelling. The point coordinates are expressed as the British National Grid eastings and northings in 100s of km. The purpose of this is to make sure that the coordinates are on a similar scale as the data, which aids fitting of the spatial statistical model:

# Combine the required variables in a data.frame for input to the models

X **=** data.frame**(**

x **=** coordinates**(**rndPts**)[**,1**]/(**1000**\***100**)**, # easting in 100s of km

y **=** coordinates**(**rndPts**)[**,2**]/(**1000**\***100**)**, # northing in 100s of km

lnPopDens **=** log**(**ptPop**)**, # log-transformed population density

lnWaterDist **=** log1p**(**ptDist2**)** # log(x+1) transformed distance to water

**)**

# Note we use coordinates in 100s of km so their variance is similar to that

# of the predictors. This aids model fitting.

summary**(**X**)**

apply**(**X, 2, sd**)**

To test whether the model variables are spatially autocorrelated (i.e. values close in space are more similar than would be expected) we apply a global Moran’s test. To do this, we calculate a pairwise distance matrix using the spDists function and use its reciprocal as inverse distance weights. The test statistic (Moran’s I) and its P value is obtained using the Moran.I function from the ape library. See <https://en.wikipedia.org/wiki/Moran%27s_I> for more details on the test. There is highly significant autocorrelation in both predictors:

# Test whether the the predictor variables are spatially autocorrelated

# using a global Moran's I test

# First, create pairwise distance weights...

ptDM **=** spDists**(**rndPts**)** # creates a pairwise distance matrix between all points

ptWM **=** 1**/**ptDM # convert to an inverse distance weight matrix

diag**(**ptWM**)** **=** 0 # set 0 weights for the diagonal (i.e. pairs of the same points)

# Now, run the tests...

Moran.I**(**x**=**X**$**lnPopDens, weight**=**ptWM**)**

Moran.I**(**x**=**X**$**lnWaterDist, weight**=**ptWM**)**

The first step in the analysis is to fit a standard linear regression by ordinary least squares (OLS), which suggests a significant negative effect of distance from water on human density:

# fit a basic linear regression (ordinary least squares, OLS)

L **=** lm**(**lnPopDens **~** lnWaterDist, data **=** X**)**

summary**(**L**)**

To accommodate spatial autocorrelation, we need to re-fit the regression model using generalised least squares (GLS), which can be done with the gls function from the nlme package. See <https://en.wikipedia.org/wiki/Generalized_least_squares> for more details on GLS. The basic model is equivalent to the OLS regression:

# re-fit using generalised least squares (GLS), with no spatial autocorrelation

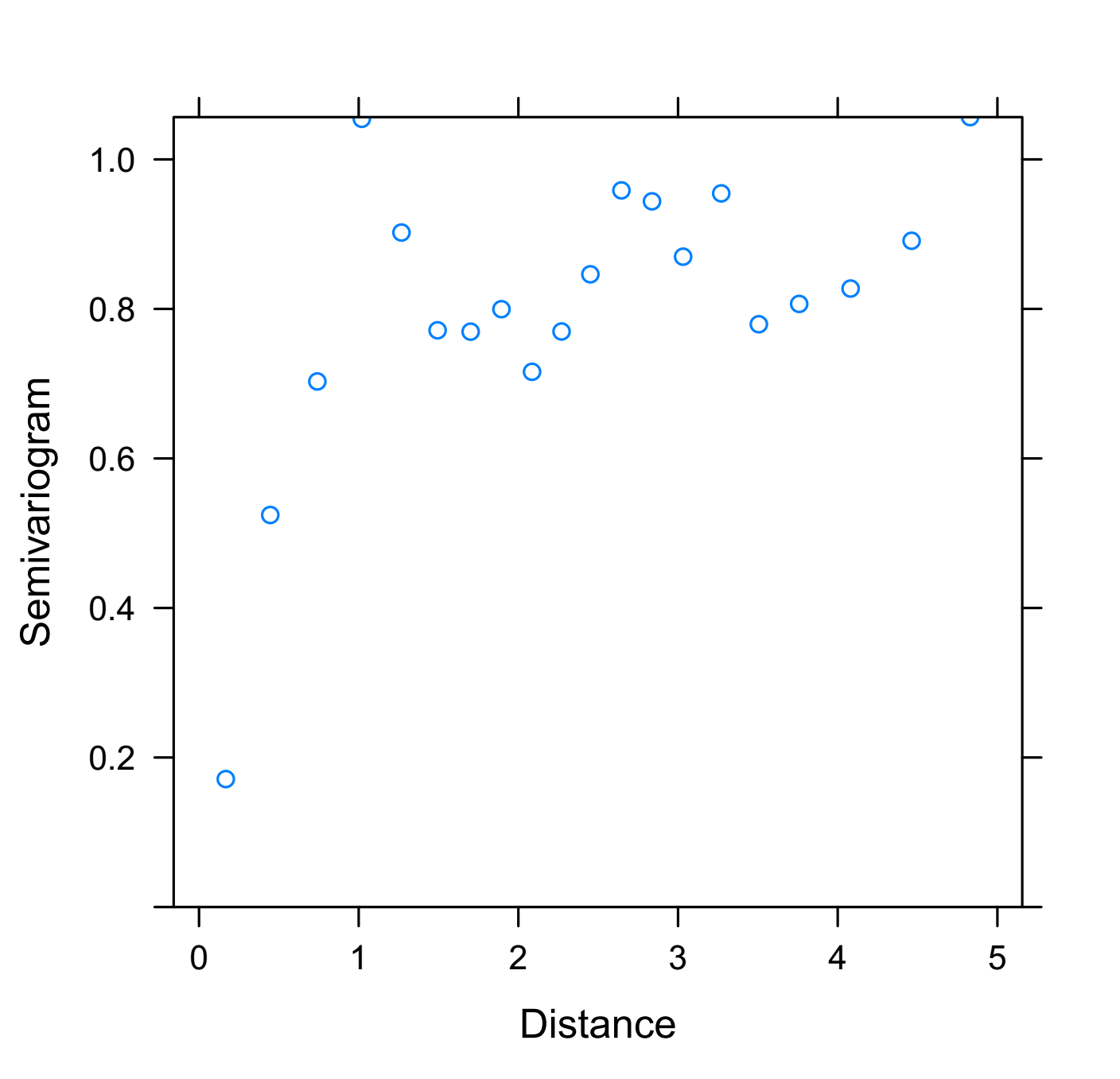
G1 **=** gls**(**lnPopDens **~** lnWaterDist, data **=** X, method**=**"ML"**)**

summary**(**G1**)**

These basic models assume the residual errors are independent of one another. If the residuals exhibit spatial autocorrelation then the P values will likely be underestimated, i.e. the statistical significance of the model effects is inflated. To examine the residual autocorrelation, three useful approaches can be employed:

1. Visualising the semi-variogram using the Variogram function from nlme. The variogram shows the variance among data values separated by different distance (see <https://en.wikipedia.org/wiki/Variogram>). If the variance is lower at close distances then spatial autocorrelation is indicated.

plot**(**Variogram**(**G1, form**=~**x**+**y, resType**=**"pearson", maxDist**=**5**)**, smooth**=FALSE)**



1. The Moran test can be used as above to test for significant positive autocorrelation:

Moran.I**(**x**=**residuals**(**G1**)**, weight**=**ptWM**)**

1. The spline.correlog function of the ncf package can be used to plot a smoothed correlogram, i.e. a function describing the decay in correlation between residuals separated by different distances.

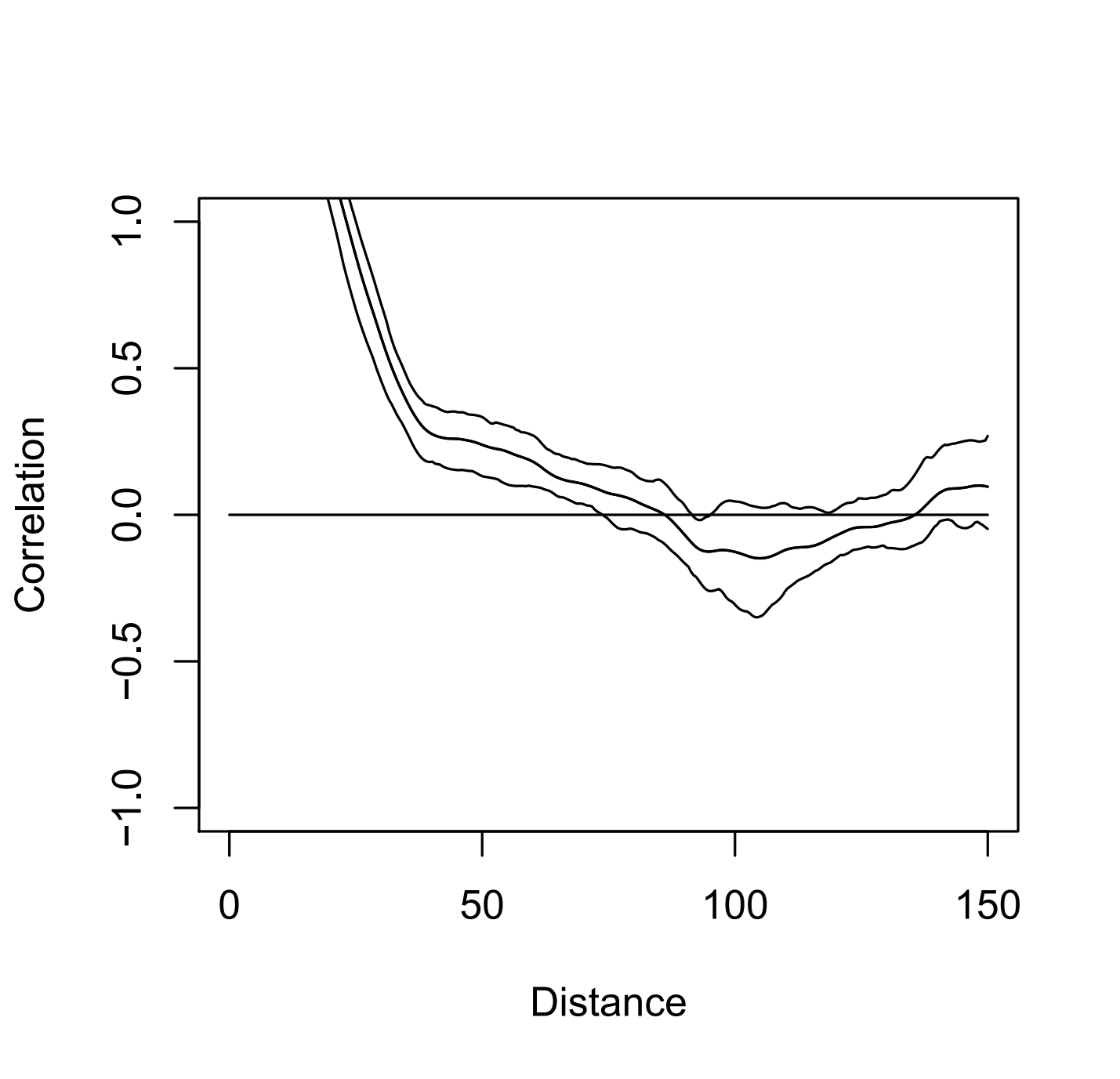
# 3. Spline correlogram up to 150 km, with bootstrap confidence intervals -

# at what distance does the autocorrelation reach 0?

plot**(**spline.correlog**(**x**=**coordinates**(**rndPts**)[**,1**]/**1000,

y**=**coordinates**(**rndPts**)[**,2**]/**1000,

z**=**residuals**(**G1**)**, resamp **=** 100, xmax**=**150**))**



Since there is clear evidence of residual autocorrelation, we need to re-fit the GLS model including a model for spatial correlation among the model errors. The parameters of the correlation function will be estimated along with the fixed terms of the model. Several different correlation structures can be used (see ?corSpatial for details). For this example we will use a spherical correlation function with two parameters – a range parameter (describing the distance at which autocorrelation occurs) and a nugget (i.e. the variance at 0 distance, which can be attributed to sampling error). When spatial autocorrelation is modelled the significance of the effect disappears, suggesting there is no real correlation. The anova function can be used to compare the fits of the spatial model and the basic model based on information criteria and likelihood ratio tests:

# Re-fit the GLS model with spatial autocorrelation in residuals.

# We will fit a 'spherical' semivariogram function with a

# nugget parameter (variance at 0 distance)

G2 **=** gls**(**lnPopDens **~** lnWaterDist,

correlation **=** corSpher**(**form**=~**x**+**y, nugget**=TRUE)**,

data **=** X, method**=**"ML"**)**

summary**(**G2**)**

# Accounting for the autocorrelation, the relationship is not statistically significant

# Note the estimated range of autocorrelation is ~211 km

# Is the model with spatial autocorrelation a better fit?

anova**(**G1, G2**)**

Often the fitting of the autocorrelation parameters can get stuck in local optima. Therefore, it is useful to choose starting values for the correlation model parameters that are close to the optima. Fortunately, the GLS models can be implemented quickly with fixed, user-defined correlation parameters, so it is possible to screen a large number of autocorrelation parameters. Below, GLS models are fitted for a variety of fixed range parameters to compare log-likelihoods. The best range parameter is used as the starting value of the full model. Fortunately the present analysis seems to have found the best correlation structure:

# The choice of starting parameter for the range of autocorrelation

# can be very important, as the optimisation algorithm is not always great

# and can get stuck in local optima.

# We can find a good starting value by fitting the model with different fixed

# ranges of autocorrelation and finding the one with the highest fit...

rangeTest **=** sapply**(**seq**(**0.3,5,0.1**)**, **function(**r**){**

message**(**r**\***100, " km"**)** # print the fixed range parameter to screen

G **=** gls**(**lnPopDens **~** lnWaterDist,

correlation **=** corSpher**(**form**=~**x**+**y, nugget**=TRUE**, value**=**c**(**r,0.01**)**, fixed**=TRUE)**,

data **=** X, method**=**"ML"**)** # fit the GLS with fixed autocorrelation

c**(**r, logLik**(**G**))** # return the range and the model log-likelihood

**})**

plot**(**t**(**rangeTest**)**, type**=**"b", pch**=**16,

xlab**=**"Range of autocorrelation", ylab**=**"Log-likelihood"**)**

bestRange **=** rangeTest**[**1, which.max**(**rangeTest**[**2,**])]**

# re-fit the model starting at the best range we found...

G3 **=** gls**(**lnPopDens **~** lnWaterDist,

correlation **=** corSpher**(**form**=~**x**+**y, nugget**=TRUE**, value**=**c**(**bestRange,0.01**))**,

data **=** X, method**=**"ML"**)**

summary**(**G3**)**

anova**(**G1, G3**)**

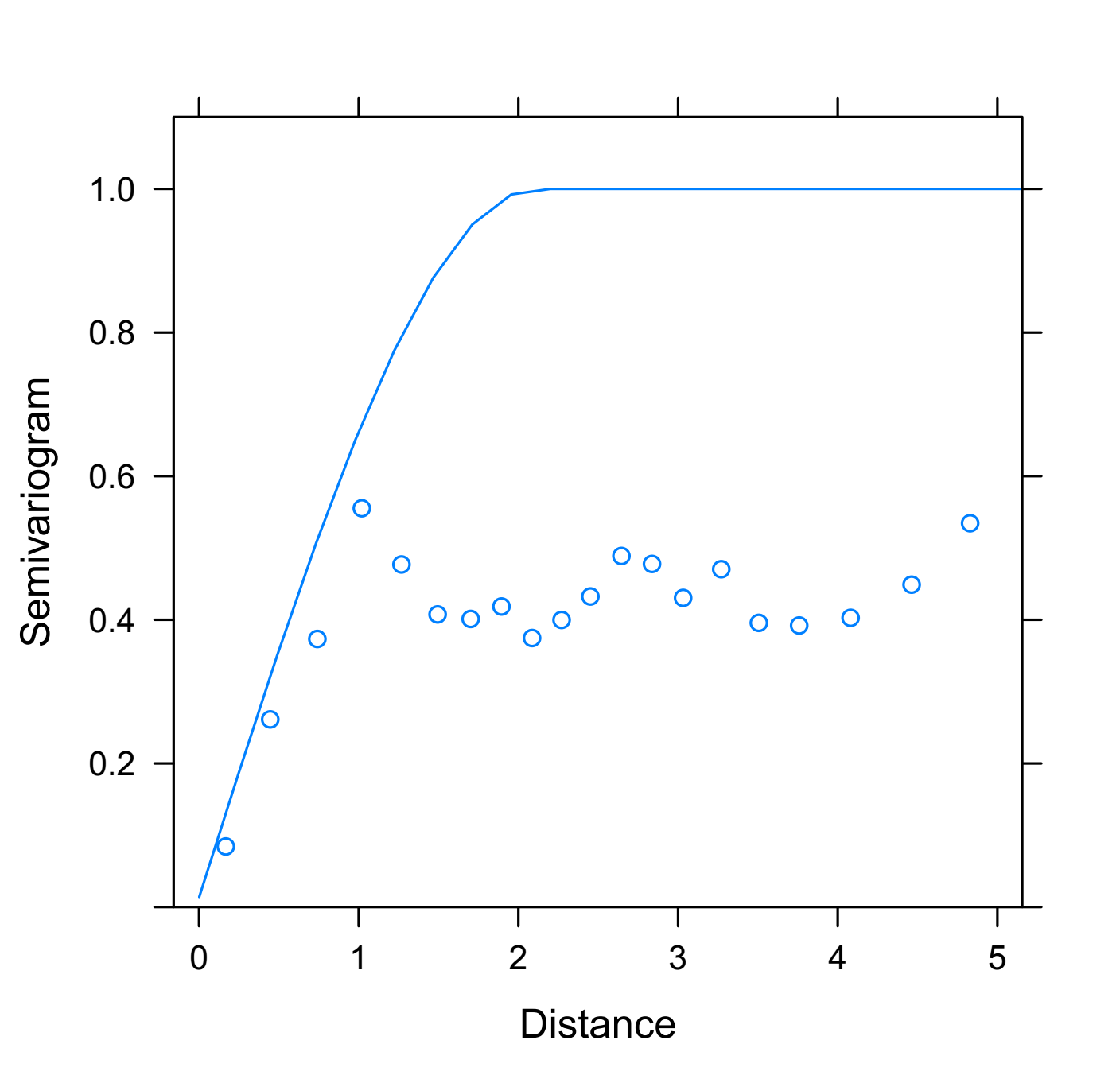
anova**(**G2, G3**)**

# G3 finds the same result as G2!

The fitted autocorrelation function can be displayed using the variogram function on the Pearson residuals. These residuals still exhibit the spatial autocorrelation that we have tried to model. The model seems to fit well for the variance at short distances:

# What autocorrelation function is fitted

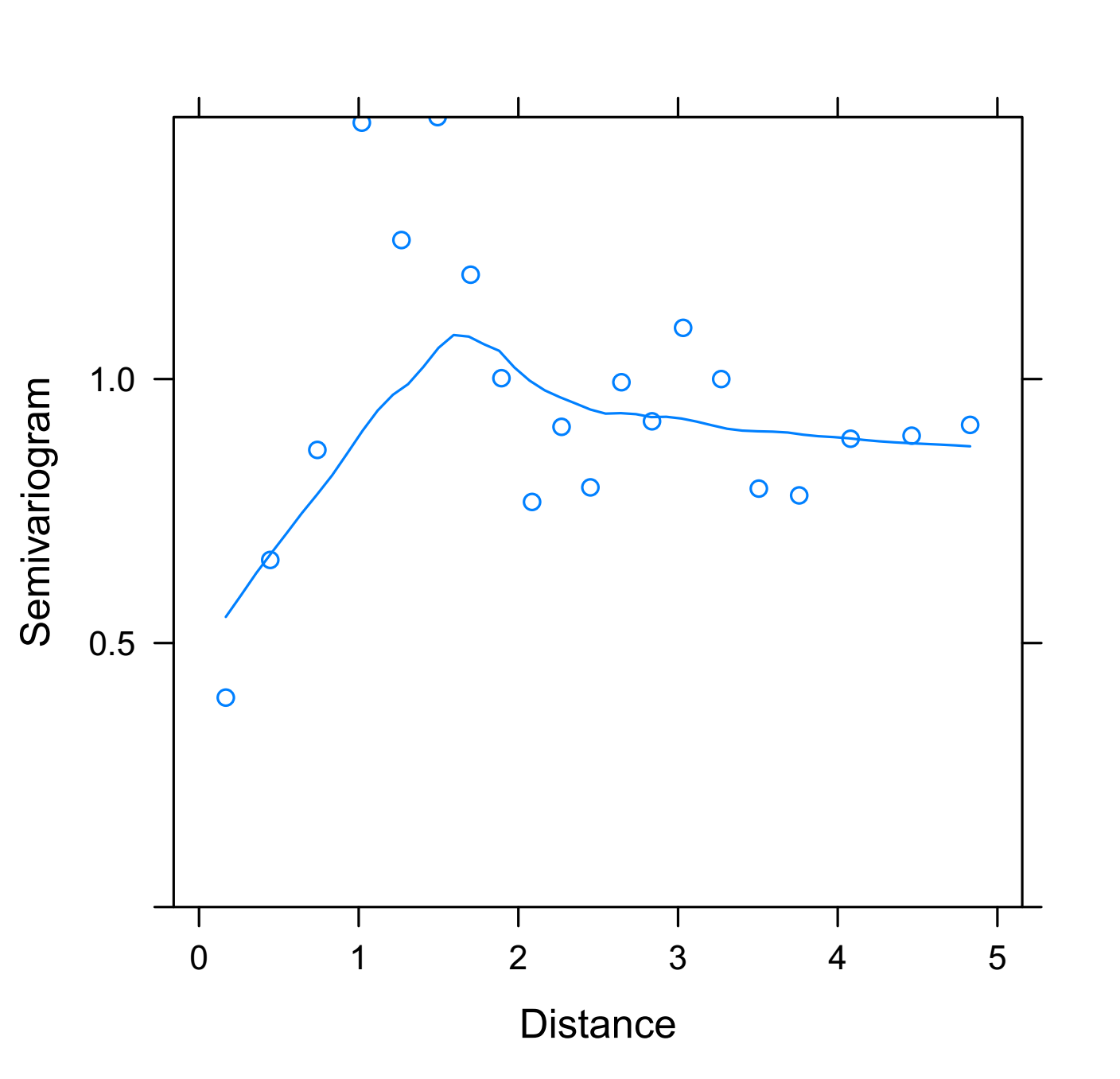
plot**(**Variogram**(**G3, form**=~**x**+**y, resType**=**"pearson", maxDist**=**5**))**



The residual spatial autocorrelation additional to that accounted for by the spherical model can be assessed based on the normalised residuals of the model. There still looks to be some trend, but the situation is better than the variogram for the first model:

# What residual autocorrelation remains...

plot**(**Variogram**(**G3, form**=~**x**+**y, resType**=**"normalized", maxDist**=**5**))**



Moran’s test on the normalised residuals shows that the autocorrelation has been reduced to a very low level:

# Check the reduction in residual autocorrelation, compared to the first model

Moran.I**(**x**=**residuals**(**G1, type**=**"pearson"**)**, weight**=**ptWM**)**

Moran.I**(**x**=**residuals**(**G3, type**=**"normalized"**)**, weight**=**ptWM**)**

# Significant autocorrelation remains, but it is very weak and unlikely to affect the conclusion.

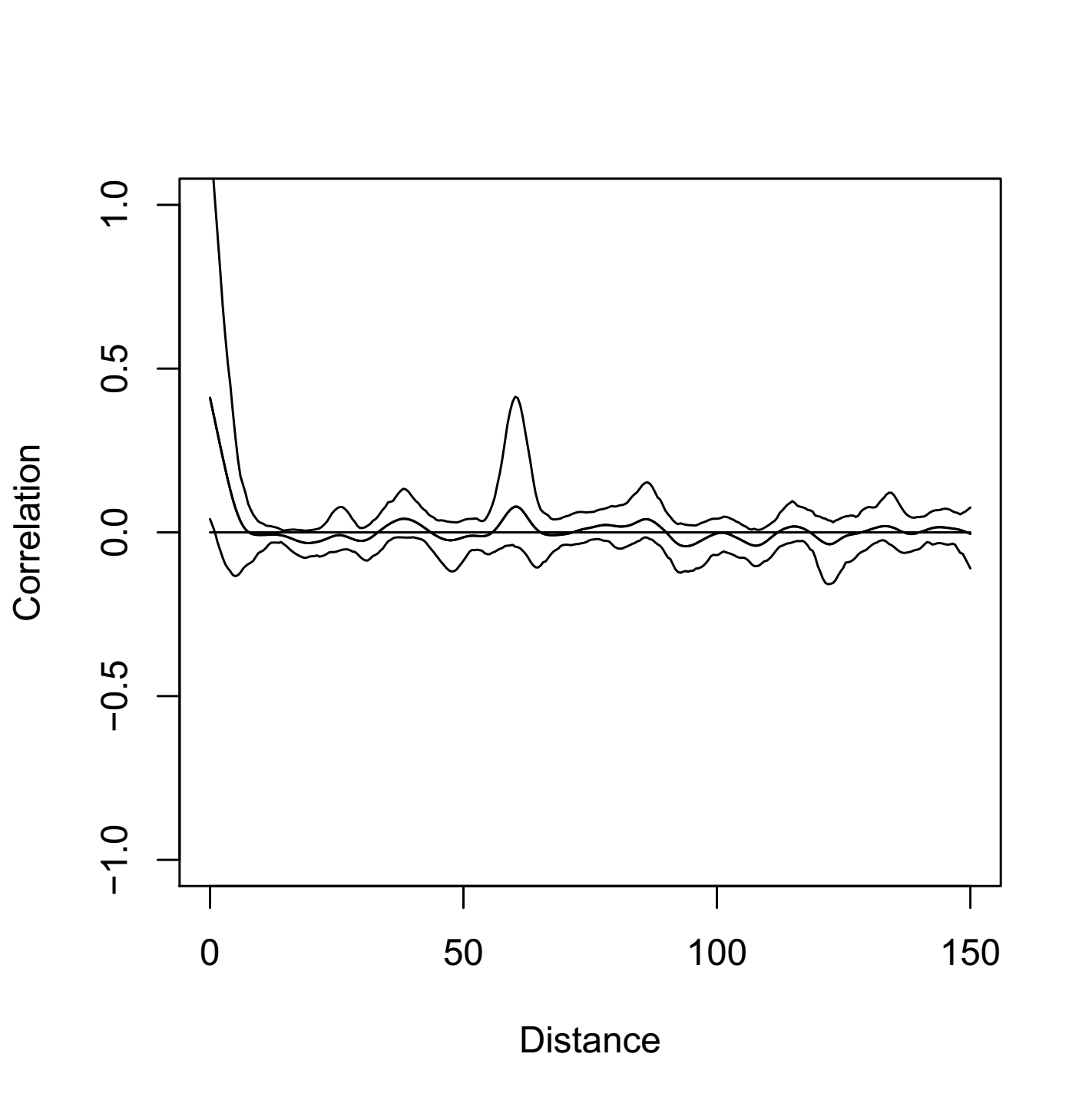
Finally, this is confirmed by the spline correlogram:

# Spline correlogram on residuals...

plot**(**spline.correlog**(**x**=**coordinates**(**rndPts**)[**,1**]/**1000,

y**=**coordinates**(**rndPts**)[**,2**]/**1000,

z**=**residuals**(**G3, type**=**"normalized"**)**, resamp **=** 100, xmax**=**150**))**



Thus we conclude that the spatial GLS model is appropriate and shows no effect of distance to water on human population density