3

Software (R+GIS+GE)

This chapter will introduce you to five main packages that we will later on use in various exercises from chapter 5 to 11: R, SAGA, GRASS, ILWIS and Google Earth (GE). All these are available as open source or as freeware and no licenses are needed to use them. By combining the capabilities of the five software packages we can operationalize preparation, processing and the visualization of the generated maps. In this handbook, ILWIS GIS will be primarily used for basic editing and to process and prepare vector and raster maps; SAGA/GRASS GIS will be used to run analysis on DEMs, but also for geostatistical interpolations; R + packages will be used for various types of statistical and geostatistical analysis, but also for data processing automation; Google Earth will be used for visualization and interpretation of results.

In all cases we will use R to control all processes, so that each exercise will culminate in a single R script ('*R on top*'; Fig. 3.1). In subsequent section, we will refer to the R + Open Source Desktop GIS combo of applications that combine geographical and statistical analysis and visualization as R+GIS+GE.

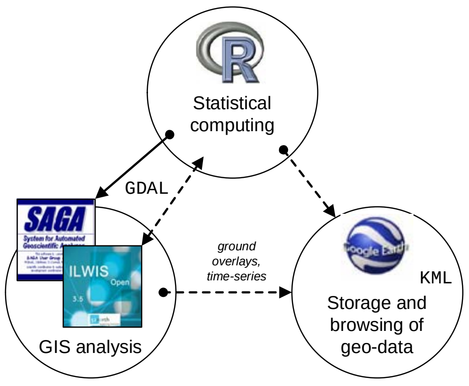

Statistical computing
GDAL
ground overlays, time-series
KML
Storage and browsing of geo-data
GIS analysis

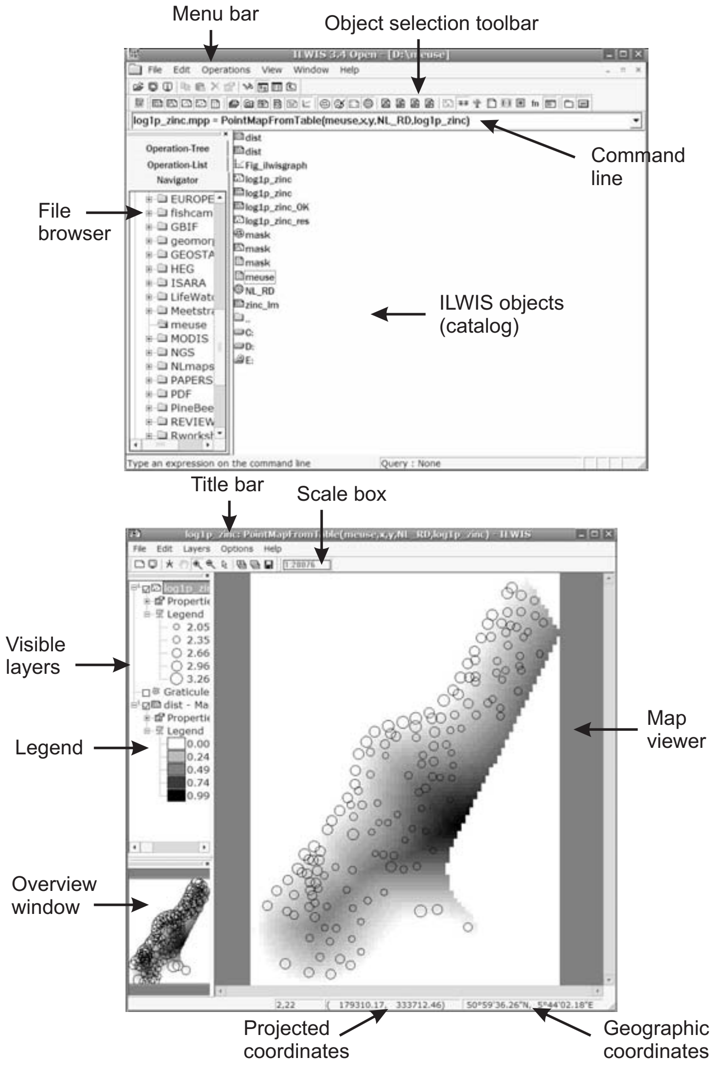
Fig. 3.1: The software triangle.

This chapter is meant to serve as a sort of a mini-manual that should help you to quickly obtain and install software, take first steps, and start doing some initial analysis. However, many details about the installation and processing steps are missing. To find more info about the algorithms and functionality of the software, please refer to the provided URLs and/or documentation listed at the end of the chapter. Note also that the instruction provided in this and following chapters basically refer to Window OS.

3.1 Geographical analysis: desktop GIS

3.1.1 ILWIS

ILWIS (**Integrated Land and Water Information System**) is a stand-alone integrated GIS package developed at the International Institute of Geo-Information Science and Earth Observations (ITC), Enschede, Netherlands. ILWIS was originally built for educational purposes and low-cost applications in developing countries. Its development started in 1984 and the first version (DOS version 1.0) was released in 1988. ILWIS 2.0 for Windows was released at the end of 1996, and a more compact and stable version 3.0 (WIN 95) was released by mid 2001. From 2004, ILWIS was distributed solely by ITC as shareware at a nominal price, and from July 2007, ILWIS shifted to open source. ILWIS is now freely available ('as-is' and free of charge) as open source software (binaries and source code) under the 52°North initiative.



log1p_zinc.mpp = PointMapFromTable(meuse,x,y,NL_RD,log1p_zinc)

Fig. 3.2: ILWIS main window (above) and map window (below).

The most recent version of ILWIS (3.6) offers a range of image processing, vector, raster, geostatistical, statistical, database and similar operations (Unit Geo Software Development, 2001). In addition, a user can create new scripts, adjust the operation menus and even build Visual Basic, Delphi, or C++ applications that will run on top of ILWIS and use its internal functions. In principle, the biggest advantage of ILWIS is that it is a compact package with a diverse vector and raster-based GIS functionality; the biggest disadvantages are bugs and instabilities and necessity to import data to ILWIS format from other more popular GIS packages.

To install ILWIS, download[1] and run the MS Windows installation. In the installation folder, you will find the main executable for ILWIS. Double click this file to start ILWIS. You will first see the main program window, which can be compared to the ArcGIS catalog (Fig. 3.2). The main program window is, in fact, a file browser which lists all ILWIS operations, objects and supplementary files within a working directory. The ILWIS Main window consists of a Menu bar, a Standard toolbar, an Object selection toolbar, a Command line, a Catalog, a Status bar and an Operations/Navigator pane with an Operation-tree, an Operation-list and a Navigator. The left pane (Operations/Navigator) is used to browse available operations and directories and the right menu shows available spatial objects and supplementary files (Fig. 3.2). GIS layers in different formats will not be visible in the catalog until we define the external file extension.

An advantage of ILWIS is that, every time a user runs an command from the menu bar or operation tree, ILWIS will record the operation in ILWIS command language. For example, you can run ordinary kriging using: from the main menu select *Operations* $\mapsto$ *Interpolation* $\mapsto$ *Point interpolation* $\mapsto$ *kriging*, which will be shown as:

```
ILWIS: log1p_zinc_OK.mpr = MapKrigingOrdinary(log1p_zinc, dist.grf,
+         Exponential(0.000,0.770,449.000), 3000, plane, 0, 20, 60, no)
```

where `log1p_zinc_OK.mpr` is the output map, `MapKrigingOrdinary` is the interpolation function, `log1p_zinc` is the attribute point map with values of the target variable, `dist.grf` is the grid definition (georeference) and `Exponential(0.000,0.770,449.000)` are the variogram parameters (see also section 5.3.2). This means that you can now edit this command and run it directly from the command line, instead of manually selecting the operations from the menu bar. In addition, you can copy such commands into an ILWIS script to enable automation of data analysis. ILWIS script can use up to nine script parameters, which can be either spatial objects, values or textual strings.

The new versions of ILWIS (>3.5) are developed as MS Visual 2008 project. The ILWIS user interface and ILWIS analytical functionality have now been completely separated making it easier to write server side

[1] `https://52north.org/download/Ilwis/`

applications for ILWIS. This allows us to control ILWIS also from R, e.g. by setting the location of ILWIS on your machine:

```
> ILWIS <- "C:\\Progra~1\\N52\\Ilwis35\\IlwisClient.exe -C"
```

To combine ILWIS and R commands in R, we use:

```
> shell(cmd=paste(ILWIS, "open log1p_zinc_OK.mpr -noask"), wait=F)
```

ILWIS has a number of built-in statistical and geostatistical functions. With respect to interpolation possibilities, it can be used to prepare a variogram, to analyze the anisotropy in the data (including the variogram surface), to run ordinary kriging and co-kriging (with one co-variable), to implement universal kriging with coordinates[2] as predictors and to run linear regression. ILWIS has also a number of original geostatistical algorithms. For example, it offers direct kriging from raster data (which can be used e.g. to filter the missing values in a raster map), and also does direct calculation of variograms from raster data. ILWIS is also suitable to run some basic statistical analysis on multiple raster layers (map lists): it offers principal component analysis on rasters, correlation analysis between rasters and multi-layer map statistics (min, max, average and standard deviation).

Although ILWIS cannot be used to run regression-kriging as defined in §2.1.5, it can be used to run a similar type of analysis. For example, a table can be imported and converted to a point map using the *Table to PointMap* operation. The point map can then be overlaid over raster maps to analyze if the two variables are correlated. This can be done by combining the table calculation and the `MapValue` function. In the same table, you can then derive a simple linear regression model e.g. `log1p_zinc = b0 + b1 * dist`. By fitting a least square fit using a polynomial, you will get: `b0=67.985` and `b1=-4.429`. This means that the zinc concentration decreases with an increase of `dist` — distance from the river (Fig. 3.3). Note that, in ILWIS, you cannot derive the Generalized Least Squares (GLS) regression coefficients (Eq.2.1.3) but only the OLS coefficients, which is statistically suboptimal method, because the residuals are possibly auto-correlated (see §2.1). In fact, regression modeling in ILWIS is so limited that the best advice is to always export the table data to R and then run statistical analysis using R packages. After estimating the regression coefficients, you can produce a map of `zinc` content (deterministic part of variation) by running a map calculation:

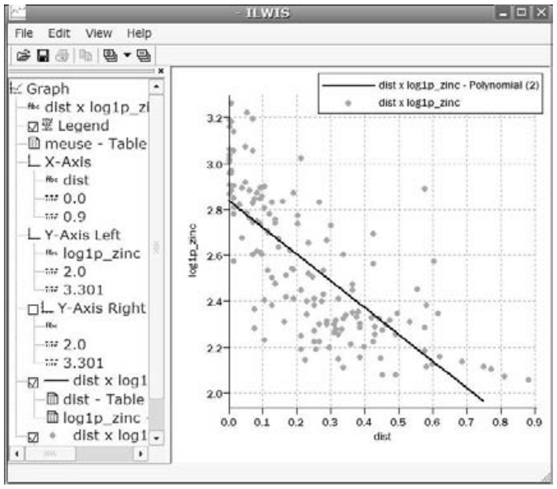


Fig. 3.3: Correlation plot `dist` vs `log1p_zinc`.

```
ILWIS: zinc_lm = 2.838 -1.169 * dist
```

Now you can estimate the residuals at sampled locations using table calculation:

```
ILWIS: log1p_zinc_res = log1p_zinc - MapValue(zinc_lm, coord(X,Y,NL_RD))
```

You can create a point map for residuals and derive a variogram of residuals by using operations *Statistics* ↦ *Spatial correlation* from the main menu. If you use a lag spacing of 100 m, you will get a variogram that can be fitted[3] with an exponential variogram model (`C0=0.008`, `C1=0.056`, `R=295`). The residuals can now be interpolated using ordinary kriging, which produces a typical kriging pattern. The fitted trend and residuals can then be added back together using:

[2]In ILWIS, the term *Universal kriging* is used exclusively for interpolation of point data using transforms of the coordinates.

[3]ILWIS does not support automated variogram fitting.

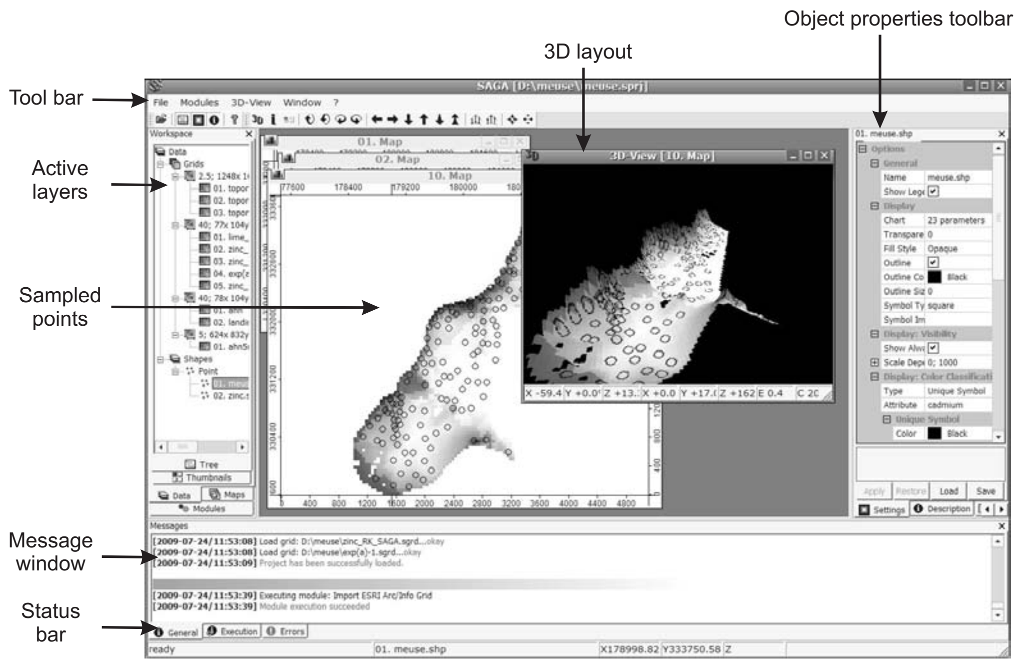
```
ILWIS: zinc_res_OK.mpr = MapKrigingOrdinary(log1p_zinc_res, dist.grf,
+           Exponential(0.008,0.056,295.000), 3000, plane, 0, 20, 60, no)
ILWIS: log1p_zinc_RK = zinc_lm + zinc_res_OK
ILWIS: zinc_rk = pow(10, log1p_zinc_RK)-1
```

which gives regression-kriging predictions. Note that, because a complete RK algorithm with GLS estimation of regression is not implemented in ILWIS (§2.1.5), we are not able to derive a map of the prediction variance (Eq.2.1.5). For these reasons, regression-kriging in ILWIS is not really encouraged and you should consider using more sophisticated geostatistical packages such as gstat and/or geoR.

Finally, raster maps from ILWIS can be exported to other packages. You can always export them to ArcInfo ASCII (.ASC) format. If the georeference in ILWIS has been set as center of the corner pixels, then you might need to manually edit the *.asc header[4]. Otherwise, you will not be able to import such maps to ArcGIS (8 or higher) or e.g. Idrisi. The pending ILWIS v3.7 will be even more compatible with the OGC simple features, WPS query features and similar. At the moment, the fastest and most efficient solution to read/write ILWIS rasters to other supported GDAL formats is FWTools[5].

3.1.2 SAGA

SAGA[6] (**System for Automated Geoscientific Analyzes**) is an open source GIS that has been developed since 2001 at the University of Göttingen[7], Germany, with the aim to simplify the implementation of new algorithms for spatial data analysis (Conrad, 2006, 2007). It is a full-fledged GIS with support for raster and vector data. SAGA includes a large set of geoscientific algorithms, and is especially powerful for the analysis of DEMs. With the release of version 2.0 in 2005, SAGA runs under both Windows and Linux operating systems. SAGA is an open-source package, which makes it especially attractive to users that would like to extend or improve its existing functionality.



```
[2009-07-24/11:53:08] Load grid: D:\meuse\zinc_RK_SAGA.sgrd...okay
[2009-07-24/11:53:08] Load grid: D:\meuse\exp(a)-1.sgrd...okay
[2009-07-24/11:53:09] Project has been successfully loaded.

[2009-07-24/11:53:39] Executing module: Import ESRI Arc/Info Grid
[2009-07-24/11:53:39] Module execution succeeded
```

```
ready    01. meuse.shp    X178998.82 Y333750.58 Z
```

Fig. 3.4: The SAGA GUI elements and displays.

SAGA handles tables, vector and raster data and natively supports at least one file format for each data type. Currently SAGA (2.0.4) provides about 48 free module libraries with >300 modules, most of them

[4]Simply replace in the header of the file `xllcenter` and `yllcenter` with `xllcorner` and `yllcorner`.
[5]`http://fwtools.maptools.org`
[6]`http://saga-gis.org`
[7]The group recently collectively moved to the Institut für Geographie, University of Hamburg.

published under the GPL. The modules cover geo–statistics, geomorphometric analysis, image processing, cartographic projections, and various tools for vector and raster data manipulation. Modules can be executed directly by using their associated parameters window. After you have imported all maps to SAGA, you can also save the whole project so that all associated maps and visualization settings are retained. The most comprehensive modules in SAGA are connected with hydrologic, morphometric and climatic analysis of DEMs.

To install SAGA, download and unzip the compiled binaries[8] to some default local directory. Then run the `saga_gui.exe` and you will get a GUI as shown in Fig. 3.4. Likewise, to install SAGA on Linux machines, you can also download the compiled binaries (e.g. `saga_2.0.4_bin_linux.tar.gz`), then run some basic configuration:

```
> ./configure --enable-unicode --with-gnu-ld=yes
```

Note that there are two possibilities to compile the software under Linux: (a) either a Non-Unicode or (b) a Unicode version of SAGA. Building the Unicode version is straight forward and recommended. Have also in mind that, under Linux, wxWidgets, PROJ 4, GDAL, JASPER, TIFF and GCC packages need to be obtained and configured separately before you can start using SAGA (for the unicode compilation, `wx.` configure does not check for JASPER libraries!). For a correct installation, you should follow the instructions provided via the SAGA WIKI[9]. SAGA is unfortunately still not available for Mac OS X.

In addition to the GUI, a second user front end, the SAGA command line interpreter can be used to execute modules. Library RSAGA[10] provides access to geocomputing and terrain analysis functions of SAGA from within R by running the command line version of SAGA (Brenning, 2008)[11]. RSAGA provides also several R functions for handling and manipulating ASCII grids, including a flexible framework for applying local functions or focal functions to multiple grids (Brenning, 2008). It is important to emphasize that RSAGA package is used mainly to control SAGA operations from R. To be able to run RSAGA, you need to have SAGA installed locally on your machine. SAGA GIS does NOT come in the RSAGA library.

To find what SAGA libraries[12] and modules do and require, you should use the `rsaga.get.modules` and `rsaga.get.usage` commands. For example, to see which parameters are needed to generate a DEM from a shapefile type:

```
> rsaga.env()

 $workspace
 [1] "."

 $cmd
 [1] "saga_cmd.exe"

 $path
 [1] "C:/Progra~1/saga_vc"

 $modules
 [1] "C:/Progra~1/saga_vc/modules"


> rsaga.get.modules("grid_spline")

 $grid_spline
   code                                     name interactive
 1    0                   Thin Plate Spline (Global)       FALSE
 2    1                    Thin Plate Spline (Local)       FALSE
 3    2                      Thin Plate Spline (TIN)       FALSE
 4    3                       B-Spline Approximation       FALSE
 5    4              Multilevel B-Spline Interpolation       FALSE
 6    5 Multilevel B-Spline Interpolation (from Grid)       FALSE
 7   NA                                         <NA>       FALSE
 8   NA                                         <NA>       FALSE
```

[8]http://sourceforge.net/projects/saga-gis/files/

[9]http://sourceforge.net/apps/trac/saga-gis/wiki/CompilingaLinuxUnicodeversion

[10]http://cran.r-project.org/web/packages/RSAGA/

[11]RPyGeo package can be used to control ArcGIS geoprocessor in a similar way.

[12]We also advise you to open SAGA and then first run processing manually (point–and–click) processing. The names of the SAGA libraries can be obtained by browsing the `/modules/` directory.

```
> rsaga.get.usage("grid_spline", 1)

 SAGA CMD 2.0.4
 library path:       C:/Progra~1/saga_vc/modules
 library name:       grid_spline
 module name :       Thin Plate Spline (Local)
 Usage: 1 [-GRID <str>] -SHAPES <str> [-FIELD <num>] [-TARGET <num>] [-REGUL <str>]
  [-RADIUS <str>] [-SELECT <num>] [-MAXPOINTS <num>] [-USER_CELL_SIZE <str>]
  [-USER_FIT_EXTENT] [-USER_X_EXTENT_MIN <str>] [-USER_X_EXTENT_MAX <str>]
  [-USER_Y_EXTENT_MIN <str>] [-USER_Y_EXTENT_MAX <str>] [-SYSTEM_SYSTEM_NX <num>]
  [-SYSTEM_SYSTEM_NY <num>] [-SYSTEM_SYSTEM_X <str>] [-SYSTEM_SYSTEM_Y <str>]
  [-SYSTEM_SYSTEM_D <str>] [-GRID_GRID <str>]
   -GRID:<str>                Grid
         Data Object (optional output)
   -SHAPES:<str>              Points
         Shapes (input)
   -FIELD:<num>               Attribute
         Table field
   -TARGET:<num>              Target Grid
         Choice
         Available Choices:
         [0] user defined
         [1] grid system
         [2] grid
   -REGUL:<str>               Regularisation
         Floating point
   -RADIUS:<str>              Search Radius
         Floating point
   -SELECT:<num>              Points Selection
         Choice
         Available Choices:
         [0] all points in search radius
         [1] maximum number of points
   -MAXPOINTS:<num>           Maximum Number of Points
         Integer
         Minimum: 1.000000
   -USER_CELL_SIZE:<str>      Grid Size
         Floating point
         Minimum: 0.000000
   -USER_FIT_EXTENT           Fit Extent
         Boolean
   -USER_X_EXTENT_MIN:<str>   X-Extent
         Value range
   -USER_X_EXTENT_MAX:<str>   X-Extent
         Value range
   -USER_Y_EXTENT_MIN:<str>   Y-Extent
         Value range
   -USER_Y_EXTENT_MAX:<str>   Y-Extent
         Value range
   -SYSTEM_SYSTEM_NX:<num>    Grid System
         Grid system
   -SYSTEM_SYSTEM_NY:<num>    Grid System
         Grid system
   -SYSTEM_SYSTEM_X:<str>     Grid System
         Grid system
   -SYSTEM_SYSTEM_Y:<str>     Grid System
         Grid system
   -SYSTEM_SYSTEM_D:<str>     Grid System
         Grid system
   -GRID_GRID:<str>           Grid
         Grid (input)
```

Most SAGA modules — with the exception of a few that can only be executed in interactive mode — can be run from within R using the `rsaga.geoprocessor` function, RSAGA's low-level workhorse. However, RSAGA also provides R wrapper functions and associated help pages for many SAGA modules. As an example, a slope raster can be calculated from a digital elevation model with SAGA's local morphometry module, which can be accessed with the `rsaga.local.morphometry` function or more specifically with `rsaga.slope` (Brenning, 2008).

SAGA can read directly ESRI shapefiles and table data. Grids need to be converted to the native SAGA grid format (`*.sgrd`). This raster format is now supported by GDAL (starting with GDAL 1.7.0) and can be read directly via the `readGDAL` method. Alternatively, you can convert some GDAL-supported formats to SAGA grids and back by using:

```
# write to SAGA grid:
> rsaga.esri.to.sgrd(in.grids="meuse_soil.asc", out.sgrd="meuse_soil.sgrd",
+       in.path=getwd())
# read SAGA grid:
> rsaga.sgrd.to.esri(in.sgrd="meuse_soil.sgrd", out.grids="meuse_soil.asc",
+       out.path=getwd())
```

SAGA grid comprises tree files:

(1.) `*.sgrd` — the header file with name, data format, XLL, YLL, rows columns, cell size, z-factor and no data value;

(2.) `*.sdat` - the raw data file;

(3.) `*.hgrd` - the history file;

In some cases, you might consider reading directly the raw data and header data to R, which can be done by using e.g.:

```
# read SAGA grid format:
> sgrd <- matrix((unlist(strsplit(readLines(file("meuse_soil.sgrd")), split="\t= "))),
+        ncol=2, byrow=T)
> sgrd

       [,1]              [,2]
  [1,] "NAME"            "meuse_soil"
  [2,] "DESCRIPTION"     "UNIT"
  [3,] "DATAFILE_OFFSET" "0"
  [4,] "DATAFORMAT"      "FLOAT"
  [5,] "BYTEORDER_BIG"   "FALSE"
  [6,] "POSITION_XMIN"   "178460.0000000000"
  [7,] "POSITION_YMIN"   "329620.0000000000"
  [8,] "CELLCOUNT_X"     "78"
  [9,] "CELLCOUNT_Y"     "104"
 [10,] "CELLSIZE"        "40.0000000000"
 [11,] "Z_FACTOR"        "1.000000"
 [12,] "NODATA_VALUE"    "-9999.000000"
 [13,] "TOPTOBOTTOM"     "FALSE"

# read the raw data: 4bit, numeric (FLOAT), byte order small;
> sdat <- readBin("meuse_soil.sdat", what="numeric", size=4,
+      n=as.integer(sgrd[8,2])*as.integer(sgrd[9,2]))
> sdat.sp <- as.im(list(x=seq(from=as.integer(sgrd[6,2]),
+      length.out=as.integer(sgrd[8,2]), by=as.integer(sgrd[10,2])),
+      y=seq(from=as.integer(sgrd[7,2]), length.out=as.integer(sgrd[9,2]),
+      by=as.integer(sgrd[10,2])), z=matrix(sdat, nrow=as.integer(sgrd[8,2]),
+      ncol=as.integer(sgrd[9,2]))))
> sdat.sp <- as(sdat.sp, "SpatialGridDataFrame")
# replace the mask value with NA's:
> sdat.sp@data[[1]] <- ifelse(sdat.sp@data[[1]]==as.integer(sgrd[12,2]), NA,
+      sdat.sp@data[[1]])
> spplot(sdat.sp)
```

Another possibility to read SAGA grids directly to R is via the `read.sgrd` wrapper function (this uses `rsaga.sgrd.to.esri` method to write to a `tempfile()`), and then `read.ascii.grid` to import data to R):

```
> gridmaps <- readGDAL("meuse_soil.asc")
> gridmaps$soil <- as.vector(t(read.sgrd("meuse_soil.sgrd", return.header=FALSE)))
```

which reads raw data as a vector to an existing `SpatialGridDataFrame`.

SAGA offers limited capabilities for geostatistical analysis, but in a very user-friendly environment. Note that many commands in SAGA are available only by right-clicking the specific data layers. For example, you can make a correlation plot between two grids by right-clicking a map of interest, then select *Show Scatterplot* and you will receive a module execution window where you can select the second grid (or a shapefile) that you would like to correlate with your grid of interest. This will plot all grid-pairs and display the regression model and its R-square (see further Fig. 10.8). The setting of the Scatterplot options can be modified by selecting *Scatterplot* from the main menu. Here you can adjust the regression formula, obtain the regression details, and adjust the graphical settings of the scatterplot.

Under the module *Geostatistics*, three groups of operations can be found: (a) *Grid* (various operations on grids); (b) *Points* (derivation of semivariances) and (c) *Kriging* (ordinary and universal kriging). Under the group *Grid*, several modules can be run: *Multiple Regression Analysis* (relates point data with rasters), *Radius of Variance* (detects a minimum radius to reach a particular variance value in a given neighborhood), *Representativeness* (derives variance in a local neighborhood), *Residual Analysis* (derives local mean value, local difference from mean, local variance, range and percentile), *Statistics for Grids* (derives mean, range and standard deviation for a list of rasters) and *Zonal Grid Statistics* (derives statistical measures for various zones and based on multiple grids). For the purpose of geostatistical mapping, we are especially interested in correlating points with rasters (see §1.3.2), which can be done via the *Multiple Regression Analysis* module. Initiating this module opens a parameter setting window (Fig. 3.5).

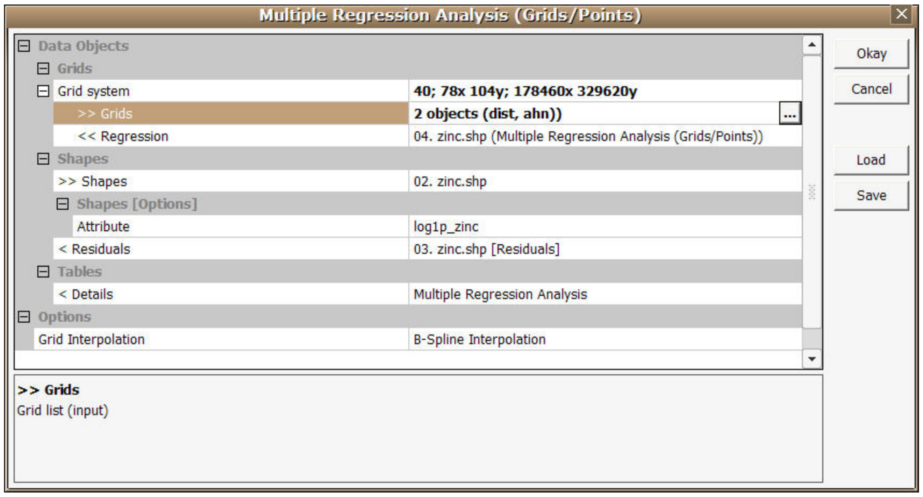
In the *Multiple Regression Analysis* module, SAGA will estimate the values of points at grids, run the regression analysis and predict the values at each location. You will also get a textual output (message window) that will show the regression model, and a list of the predictors according to their importance e.g.:

```
Executing module: Multiple Regression Analysis (Grids/Points)
Parameters
Grid system: 40; 78x 104y; 178460x 329620y
Grids: 2 objects (dist, ahn))
Shapes: zinc.shp
Attribute: log1p_zinc
Details: Multiple Regression Analysis
Residuals: zinc.shp [Residuals]
Regression: zinc.shp (Multiple Regression Analysis (Grids/Points))
Grid Interpolation: B-Spline Interpolation


Regression:
 Y = 6.651112 -2.474725*[dist] -0.116471*[ahn]

Correlation:
1: R2 = 54.695052% [54.695052%] -> dist
2: R2 = 55.268255% [0.573202%] -> ahn
```

in this case the most significant predictor is `dist`; the second predictor explains <1% of the variability in `log1p_zinc` (see further Fig. 5.6). The model explains 55.3% of the total variation.

When selecting the multiple regression analysis options, you can also opt to derive the residuals and fit the variogram of residuals. These will be written as a shapefile that can then be used to derive semivariances. Select *Geostatistics* $\mapsto$ *Points* $\mapsto$ *Semivariogram* and specify the distance increment (lag) and maximum distance. The variogram can be displayed by again right clicking a table and selecting *Show Scatterplot* option. Presently, the variogram (regression) models in SAGA are limited to linear, exponential and logarithmic models. In general, fitting and use of variograms in SAGA is discouraged[13].

[13]Exceptionally, you should use the logarithmic model which will estimate something close to the exponential variogram model (Eq.1.3.8).



Data Objects	
Grids	
Grid system	40; 78x 104y; 178460x 329620y
>> Grids	2 objects (dist, ahn))
<< Regression	04. zinc.shp (Multiple Regression Analysis (Grids/Points))
Shapes	
>> Shapes	02. zinc.shp
Shapes [Options]	
Attribute	log1p_zinc
< Residuals	03. zinc.shp [Residuals]
Tables	
< Details	Multiple Regression Analysis
Options	
Grid Interpolation	B-Spline Interpolation

>> Grids
Grid list (input)

Fig. 3.5: Running predictions by using regression analysis in SAGA GIS: parameter settings window. The "Grid Interpolation" setting indicates the way SAGA will estimate values of grids at calibration points. This should not be confused with other gridding techniques available in SAGA.

Once the regression model and the variogram of the residuals have been estimated, a user can also run regression-kriging, which is available in SAGA under the module *Geostatistics* $\mapsto$ *Universal kriging*. Global and local (search radius) version of the Universal kriging are available. Use of local Universal kriging with a small search radius ($\ll$100 points) is not recommended because it over-simplifies the technique, and can lead to artefacts[14]. Note also that, in SAGA, you can select as many predictors as you wish, as long as they are all in the same grid system. The final results can be visualized in both 2D and 3D spaces.

Another advantage of SAGA is the ability to use script files for the automation of complex work-flows, which can then be applied to different data projects. Scripting of SAGA modules is now possible in two ways:

(1.) Using the command line interpreter (`saga_cmd.exe`) with DOS batch scripts. Some instructions on how to generate batch files can be found in Conrad (2006, 2007).

(2.) A much more flexible way of scripting utilizes the Python interface to the SAGA Application Programming Interface (SAGA-API).

In addition to scripting possibilities, SAGA allows you to save SAGA parameter files (`*.sprm`) that contain all inputs and output parameters set using the module execution window. These parameter files can be edited in an ASCII editor, which can be quite useful to automate processing.

In summary, SAGA GIS has many attractive features for both geographical and statistical analysis of spatial data: (1) it has a large library of modules, especially to parameterize geomorphometric features, (2) it can generate maps from points and rasters by using multiple linear regression and regression-kriging, and (3) it is an open source GIS with a popular GUI. Compared to `gstat`, SAGA is not able to run geostatistical simulations, GLS estimation nor stratified or co-kriging. However, it is capable of running regression-kriging in a statistically sound way (unlike ILWIS). The advantage of SAGA over R is that it can load and process relatively large maps (not recommended in R for example) and that it can be used to visualize the input and output maps in 2D and 2.5D (see further section 5.5.2).

3.1.3 GRASS GIS

GRASS[15] (**Geographic Resources Analysis Support System**) is a general-purpose Geographic Information System (GIS) for the management, processing, analysis, modeling and visualization of many types of geo-referenced data. It is Open Source software released under GNU General Public License and is available on

[14] Neither local variograms nor local regression models are estimated. See §2.2 for a detailed discussion.

[15] `http://grass.itc.it`

the three major platforms (Microsfot Windows, Mac OS X and Linux). The main component of the development and software maintenance is built on top of highly automated web-based infrastructure sponsored by ITC-irst (Centre for Scientific and Technological Research) in Trento, Italy with numerous worldwide mirror sites. GRASS includes functions to process raster maps, including derivation of descriptive statistics for maps, histograms, but also generation of statistics for time series. There are also several unique interpolation techniques. For example the Regularized Spline with Tension (RST) interpolation, which has been quoted as one of the most sophisticated methods to generate smooth surfaces from point data (Mitasova et al., 2005).

In version 5.0 of GRASS, several basic geostatistical functionalities existed including ordinary kriging and variogram plotting, however, developers of GRASS ultimately concluded that there was no need to build geostatistical functionality from scratch when a complete open source package already existed. The current philosophy (v 6.5) focuses on making GRASS functions also available in R, so that both GIS and statistical operations can be integrated in a single command line. A complete overview of the Geostatistics and spatial data analysis functionality can be found via the GRASS website[16]. Certainly, if you are a Linux user and already familiar with GRASS, you will probably not encounter many problems in installing GRASS and using the syntax.

Unlike SAGA, GRASS requires that you set some initial '*environmental*' parameters, i.e. initial setting that describe your project. There are three initial environmental parameters: `DATABASE` — a directory (folder) on disk to contain all GRASS maps and data; `LOCATION` — the name of a geographic location (defined by a co-ordinate system and a rectangular boundary), and `MAPSET` — a rectangular REGION and a set of maps (Neteler and Mitasova, 2008). Every `LOCATION` contains at least a `MAPSET` called `PERMANENT`, which is readable by all sessions. GRASS locations are actually powerful abstractions that do resemble the way in which workflows were/are set up in larger multi-user projects. The mapsets parameter is used to distinguish users, and `PERMANENT` was privileged with regard to who could change it — often the database/location/mapset tree components can be on different physical file systems. On single-user systems or projects, this construction seems irrelevant, but it isn't when many users work collaborating on the same location.

GRASS can be controlled from R thanks to the spgrass6[17] package (Bivand, 2005; Bivand et al., 2008): `initGRASS` can be used to define the environmental parameters; description of each GRASS module can be obtained by using the `parseGRASS` method. The recommended reference manual for GRASS is the "*GRASS book*" (Neteler and Mitasova, 2008); a complete list of the modules can be found in the GRASS reference manual[18]. Some examples of how to use GRASS via R are shown in §10.6.2. Another powerful combo of applications similar to the one shown in Fig. 3.1 is the QGIS+GRASS+R triangle. In this case, a GUI (QGIS) stands on top of GRASS (which stands *on top* of R), so that this combination is worth checking for users that prefer GUI's.

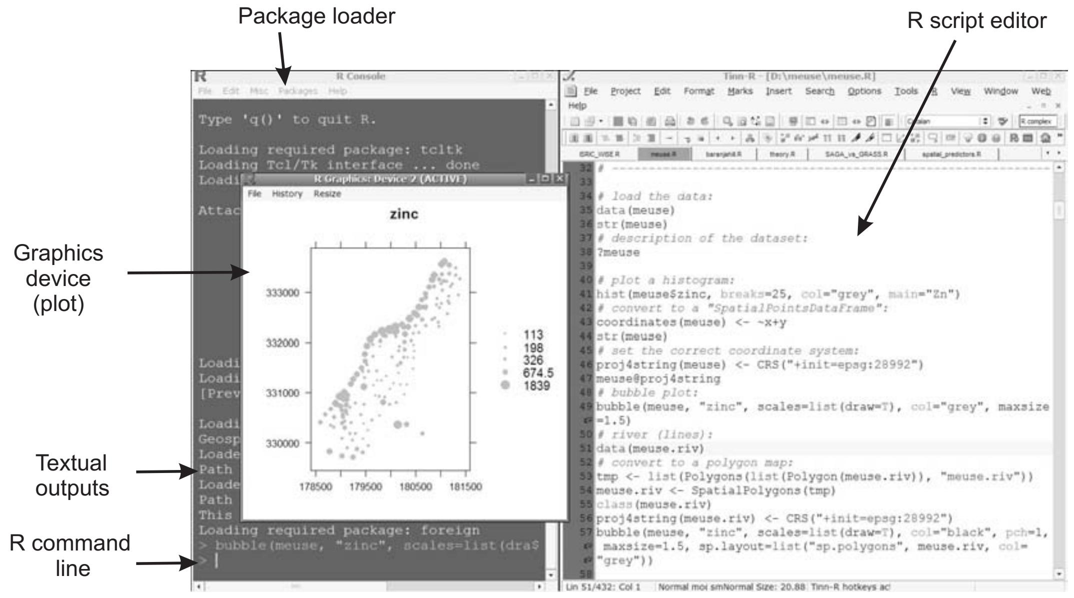
3.2 Statistical computing: R

R[19] is the open source implementation of the S language for statistical computing (R Development Core Team, 2009). Apparently, the name "R" was selected for two reasons: (1) precedence — "R" is a letter before "S", and (2) coincidence — both of the creators' names start with a letter "R". The S language provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering,. . .) and graphical techniques, and is highly extensible (Chambers and Hastie, 1992; Venables and Ripley, 2002). It has often been the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

Although much of the R code is always under development, a large part of the code is usable, portable and extendible. This makes R one of the most suitable coding environments for academic societies. Although it typically takes a lot of time for non-computer scientists to learn the R syntax, the benefits are worth the time investment.

To install R under Windows, download and run an installation executable file from the R-project homepage. This will install R for Windows with a GUI. After starting R, you will first need to set-up the working directory and install additional packages. To run geostatistical analysis in R, you will need to add the following R packages: gstat (gtat in R), rgdal (GDAL import of GIS layers in R), sp (support for spatial objects in R),

[16]`http://grass.itc.it/statsgrass/`
[17]`http://cran.r-project.org/web/packages/spgrass6/`
[18]See your local installation `file:///C:/GRASS/docs/html/full_index.html`.
[19]`http://www.r-project.org`


Package loader
R script editor
Graphics device (plot)
Textual outputs
R command line
R Console
File Edit Misc Packages Help
R Graphics: Device 2 (ACTIVE)
File History Resize
zinc
333000
332000
331000
330000
178500 179500 180500 181500
113
198
326
674.5
1839
Tinn-R - [D:\meuse\meuse.R]
File Project Edit Format Marks Insert Search Options Tools R View Window Web Help
Lin 51/432: Col 1 Normal moi smNormal Size: 20.88 Tinn-R hotkeys ac

```
Type 'q()' to quit R.

Loading required package: tcltk
Loading Tcl/Tk interface ... done
Loadi

Attac



Loadi
Loadi
[Prev

Loadi
Geosp
Loade
Path
Loade
Path
This
Loading required package: foreign
> bubble(meuse, "zinc", scales=list(dra$
> |
```

```
# -----------------------------------------------------------

# load the data:
data(meuse)
str(meuse)
# description of the dataset:
?meuse

# plot a histogram:
hist(meuse$zinc, breaks=25, col="grey", main="Zn")
# convert to a "SpatialPointsDataFrame":
coordinates(meuse) <- ~x+y
str(meuse)
# set the correct coordinate system:
proj4string(meuse) <- CRS("+init=epsg:28992")
meuse@proj4string
# bubble plot:
bubble(meuse, "zinc", scales=list(draw=T), col="grey", maxsize
=1.5)
# river (lines):
data(meuse.riv)
# convert to a polygon map:
tmp <- list(Polygons(list(Polygon(meuse.riv)), "meuse.riv"))
meuse.riv <- SpatialPolygons(tmp)
class(meuse.riv)
proj4string(meuse.riv) <- CRS("+init=epsg:28992")
bubble(meuse, "zinc", scales=list(draw=T), col="black", pch=1,
 maxsize=1.5, sp.layout=list("sp.polygons", meuse.riv, col=
"grey"))
```

Fig. 3.6: The basic GUI of R under Windows (controlled using **Tinn-R**) and a typical plot produced using the sp package.

spatstat (spatial statistics in R) and maptools.

To install these packages you should do the following. First start the R GUI, then select the *Packages* ↦ *Load package* from the main menu. Note that, if you wish to install a package on the fly, you will need to select a suitable CRAN mirror from which it will download and unpack a package. Another quick way to get all packages used in R to do spatial analysis[20] (as explained in Bivand et al. (2008)) is to install the ctv package and then execute the command:

```
> install.packages("ctv")
> library(ctv)
> install.views("Spatial")
```

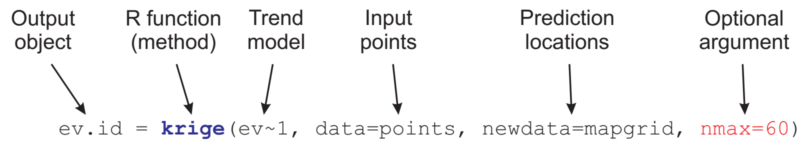
After you install a package, you will still need to load it into your workspace (every time you start R) before you can use its functionality. A package is commonly loaded using e.g.:

```
> library(gstat)
```

Much of information about some package can be found in the package installation directory in sub-directory `html`, or can be called directly from the R session. For example, important information on how to add more rgdal drivers can be found in the attached documentation:

```
> file.show(system.file("README.windows", package="rgdal"))
```

R is an object-based, functional language. Typically, a function in R consists of three items: its arguments, its body and its environment. A function is *invoked* by a name, followed by its arguments. Arguments themselves can be positional or named and can have defaults in which case they can be omitted (see also p.32):



```
ev.id = krige(ev~1, data=points, newdata=mapgrid, nmax=60)
```

[20] `http://cran.r-project.org/web/views/Spatial.html`

Functions typically return their result as their value, not via an argument. In fact, if the body of a function changes an argument it is only changing a local copy of the argument and the calling program does not get the changed result.

R is widely recognized as one of the fastest growing and most comprehensive statistical computing tools[21]. It is estimated that the current number of active R users (Google trends service) is about 430k, but this number is constantly growing. R practically offers statistical analysis and visualization of unlimited sophistication. A user is not restricted to a small set of procedures or options, and because of the contributed packages, users are not limited to one method of accomplishing a given computation or graphical presentation. As we will see later, R became attractive for geostatistical mapping mainly due to the recent integration of the geostatistical tools (gstat, geoR) and tools that allow R computations with spatial data layers (sp, maptools, raster and similar).

Note that in R, the user must type commands to enter data, do analyzes, and plot graphs. This might seem inefficient to users familiar with MS Excel and similar intuitive, point-and-click packages. If a single argument in the command is incorrect, inappropriate or mistyped, you will get an error message indicating where the problem might be. If the error message is not helpful, then try receiving more help about some operation. Many very useful introductory notes and books, including translations of manuals into other languages than English, are available from the documentation section[22]. Another very useful source of information is the R News[23] newsletter, which often offers many practical examples of data processing. Vol. 1/2 of R News, for example, is completely dedicated to spatial statistics in R; see also Pebesma and Bivand (2005) for an overview of classes and methods for spatial data in R. The '*Spatial*' packages can be nicely combined with e.g. the '*Environmentrics*'[24] packages. The interactive graphics[25] in R is also increasingly powerful (Urbanek and Theus, 2008). To really get an idea about the recent developments, and to get support with using spatial packages, you should register with the special interest group R-sig-Geo[26].

Although there are several packages in R to do geostatistical analysis and mapping, many recognize R+gstat/geoR as the only complete and fully-operational packages, especially if you wish to run regression-kriging, multivariate analysis, geostatistical simulations and block predictions (Hengl et al., 2007a; Rossiter, 2007). To allow extension of R functionalities to operations with spatial data, the developer of gstat, with the support of colleagues, has developed the sp[27] package (Pebesma and Bivand, 2005; Bivand et al., 2008). Now, users are able to load GIS layers directly into R, run geostatistical analysis on grid and points and display spatial layers as in a standard GIS package. In addition to sp, two important spatial data protocols have also been recently integrated into R: (1) **GIS data exchange protocols** (GDAL — Geospatial Data Abstraction Library, and OGR[28] — OpenGIS Simple Features Reference Implementation), and (2) **map projection protocols** (PROJ.4[29] — Cartographic Projections Library). These allow R users to import/export raster and vector maps, run raster/vector based operations and combine them with statistical computing functionality of various packages. The development of GIS and graphical functionalities within R has already caused a small revolution and many GIS analysts are seriously thinking about completely shifting to R.

3.2.1 gstat

gstat[30] is a stand-alone package for geostatistical analysis developed by Edzer Pebesma during his PhD studies at the University of Utrecht in the Netherlands in 1997. As of 2003, the gstat functionality is also available as an S extension, either as R package or S-Plus library. Current development focuses mainly on the R/S extensions, although the stand alone version can still be used for many applications. To install gstat (the stand-alone version) under Windows, download the `gstat.exe` and `gstatw.exe` (variogram modeling with GUI) files from the gstat.org website and put them in your system directory[31]. Then, you can always run gstat from the Windows start menu. The `gstat.exe` runs as a DOS application, which means that there is no GUI.

[21] The article in the New Your Times by Vance (2009) has caused much attention.
[22] `http://www.r-project.org/doc/bib/R-books.html`
[23] `http://cran.r-project.org/doc/Rnews/`; now superseded by R Journal.
[24] `http://cran.r-project.org/web/views/Environmetrics.html`
[25] `http://www.interactivegraphics.org`
[26] `https://stat.ethz.ch/mailman/listinfo/r-sig-geo`
[27] `http://r-spatial.sourceforge.net`
[28] `http://www.gdal.org/ogr/` — for Mac OS X users, there is no binary package available from CRAN.
[29] `http://proj.maptools.org`
[30] `http://www.gstat.org`
[31] E.g. `C:\Windows\system32\`

A user controls the processing by editing the command files.

gstat is possibly the most complete, and certainly the most accessible geostatistical package in the World. It can be used to calculate sample variograms, fit valid models, plot variograms, calculate (pseudo) cross variograms, and calculate and fit directional variograms and variogram models (anisotropy coefficients are not fitted automatically). Kriging and (sequential) conditional simulation can be done under (simplifications of) the universal co-kriging model. Any number of variables may be spatially cross-correlated. Each variable may have its own number of trend functions specified (being coordinates, or so-called external drift variables). Simplifications of this model include ordinary and simple kriging, ordinary or simple co-kriging, universal kriging, external drift kriging, Gaussian conditional or unconditional simulation or cosimulation. In addition, variables may share trend coefficients (e.g. for collocated co-kriging). To learn about capabilities of gstat, a user is advised to read the gstat User's manual[32], which is still by far the most complete documentation of the gstat.

A complete overview of gstat functions and examples of R commands are given in Pebesma (2004). A more recent update of the functionality can be found in Bivand et al. (2008, §8); gstat development tree is from 2010 also available from a public SVN[33]. The most widely used gstat functions in R include:

- `variogram` — calculates sample (experimental) variograms;
- `plot.variogram` — plots an experimental variogram with automatic detection of lag spacing and maximum distance;
- `fit.variogram` — iteratively fits an experimental variogram using reweighted least squares estimation;
- `krige` — a generic function to make predictions by inverse distance interpolation, ordinary kriging, OLS regression, regression-kriging and co-kriging;
- `krige.cv` — runs krige with cross-validation using the *n*-fold or *leave-one-out* method;

R offers much more flexibility than the stand-alone version of gstat, because users can extend the optional arguments and combine them with outputs or functions derived from other R packages. For example, instead of using a trend model with a constant (intercept), one could use outputs of a linear model fitting, which allows even more compact scripting.

3.2.2 The stand-alone version of gstat

As mentioned previously, gstat can be run as a stand-alone application, or as a R package. In the stand-alone version of the gstat, everything is done via compact scripts or command files. The best approach to prepare the command files is to learn from the list of example command files that can be found in the gstat User's manual[34]. Preparing the command files for gstat is rather simple and fast. For example, to run inverse distance interpolation the command file would look like this:

```
#  Inverse distance interpolation on a mask map

data(zinc): 'meuse.eas', x=1, y=2, v=3;
mask: 'dist.asc'; # the prediction locations
predictions(zinc): 'zinc_idw.asc'; # result map
```

where the first line defines the input point data set (`points.eas` — an input table in the GeoEAS[35] format), the coordinate columns (x, y) are the first and the second column in this table, and the variable of interest is in the third column; the prediction locations are the grid nodes of the map `dist.asc`[36] and the results of interpolation will be written to a raster map `zinc_idw.asc`.

To extend the predictions to regression-kriging, the command file needs to include the auxiliary maps and the variogram model for the residuals:

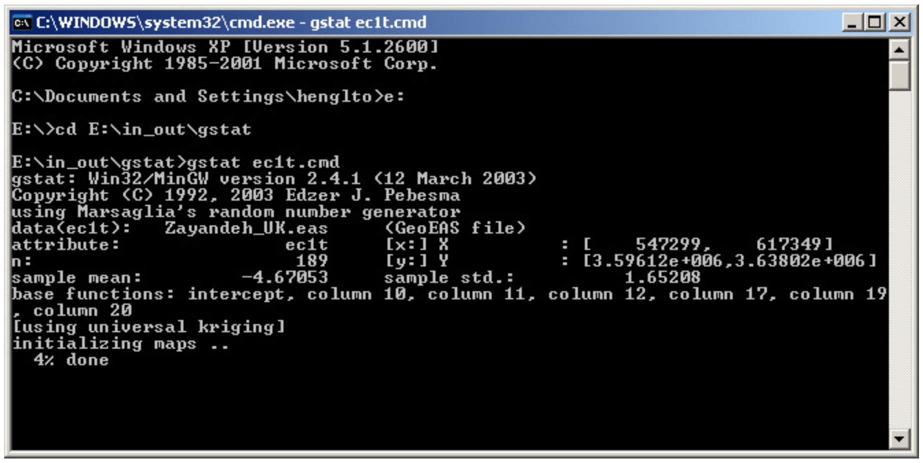
[32]http://gstat.org/manual/
[33]https://52north.org/svn/geostatistics/
[34]http://gstat.org/manual/node30.html
[35]http://www.epa.gov/ada/csmos/models/geoeas.html
[36]Typically ArcInfo ASCII format for raster maps.

```
#  Regression-kriging using two auxiliary maps
#  Target variable is log-transformed

data(ln_zinc): 'meuse.eas', x=1, y=2, v=3, X=4,5, log;
variogram(ln_zinc): 0.055 Nug(0) + 0.156 Exp(374);
mask: 'dist.asc', 'ahn.asc'; # the predictors
predictions(ev): 'ln_zinc_rk.asc'; # result map
```

where `X` defines the auxiliary predictors, `0.055 Nug(0) + 0.156 Exp(374)` is the variogram of residuals and `dist.asc` and `ahn.asc` are the auxiliary predictors. All auxiliary maps need to have the same grid definition and need to be available also in the input table. In addition, the predictors need to be sorted in the same order in both the first and the third line. Note that there are many optional arguments that can be included in the command file: a search radius can be set using "`max=50`"; switching from predictions to simulations can be done using "`method: gs`"; bloc kriging can be initiated using "`blocksize: dx=100`".

To run a command file start DOS prompt by typing: `> cmd`, then move to the active directory by typing: e.g. `> cd c:\gstat`; to run spatial predictions or simulations run the gstat program together with a specific gstat command file from the Windows cmd console (Fig. 3.7):

```
cmd gstat.exe ec1t.cmd
```



```
C:\WINDOWS\system32\cmd.exe - gstat ec1t.cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\hengl to>e:

E:\>cd E:\in_out\gstat

E:\in_out\gstat>gstat ec1t.cmd
gstat: Win32/MinGW version 2.4.1 (12 March 2003)
Copyright (C) 1992, 2003 Edzer J. Pebesma
using Marsaglia's random number generator
data(ec1t):   Zayandeh_UK.eas     (GeoEAS file)
attribute:                ec1t     [x:] X           : [     547299,     617349]
n:                         189     [y:] Y           : [3.59612e+006,3.63802e+006]
sample mean:          -4.67053     sample std.:           1.65208
base functions: intercept, column 10, column 11, column 12, column 17, column 19
, column 20
[using universal kriging]
initializing maps ..
  4% done
```

Fig. 3.7: Running interpolation using the gstat stand-alone: the DOS command prompt.

gstat can also automatically fit a variogram by using:

```
data(ev): 'points.eas', x=1,y=2,v=3;

# set an initial variogram:
variogram(ev): 1 Nug(0) + 5 Exp(1000);
# fit the variogram using standard weights:
method: semivariogram;
set fit=7;

# write the fitted variogram model and the corresponding gnuplot
set output= 'vgm_ev.cmd';
set plotfile= 'vgm_ev.plt';
```

where set `fit=7` defines the fitting method (weights=$N_j/\mathbf{h}_j^2$), `vgm_ev.cmd` is the text file where the fitted parameters will be written. Once you fitted a variogram, you can then view it using the wgnuplot[37]. Note that, for automated modeling of variogram, you will need to define the fitting method and an initial variogram, which is then iteratively fitted against the sampled values. A reasonable initial exponential variogram can be

[37]`http://www.gnuplot.info`

produced by setting nugget parameter = measurement error, sill parameter = sampled variance, and range parameter = 10% of the spatial extent of the data (or two times the mean distance to the nearest neighbor). This can be termed a **standard initial variogram model**. Although the true variogram can be quite different, it is important to have a good idea of what the variogram *should* look like.

There are many advantages to using the stand-alone version of gstat. The biggest ones are that it takes little time to prepare a script and that it can work with large maps (unlike R that often faces memory problems). In addition, the results of interpolation are directly saved in a GIS format and can be loaded to ILWIS or SAGA. However, for regression-kriging, we need to estimate the regression model first, then derive the residuals and estimate their variogram model, which cannot be automated in gstat so we must in any case load the data to some statistical package before we can prepare the command file. Hence, the stand-alone version of gstat, as with SAGA, can be used for geostatistical mapping only after regression modeling and variogram fitting have been completed.

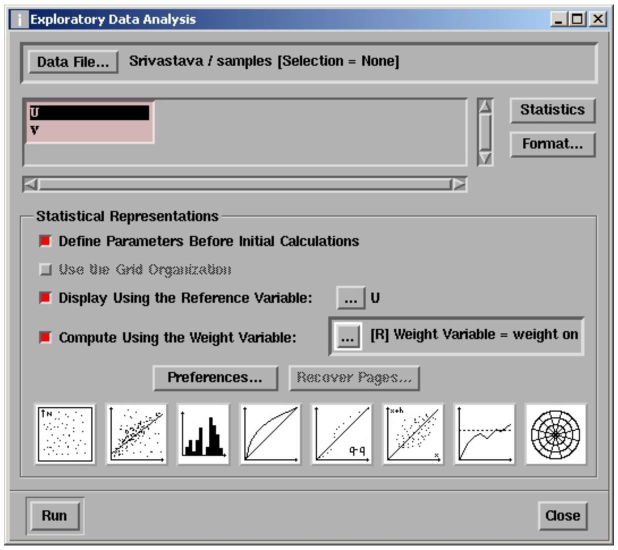
3.2.3 geoR

An equally comprehensive package for geostatistical analysis is geoR, a package extensively described by Diggle and Ribeiro Jr (2007) and Ribeiro Jr et al. (2003); a series of tutorials can be found on the package homepage[38]. In principle, a large part of the functionality of gstat and geoR overlap[39]; on the other hand, geoR has many original methods, including an original format for spatial data (called geodata). geoR can be in general considered to be more suited for variogram model estimation (including interactive visual fitting), modeling of non-normal variables and simulations. A short demo of what geoR can do considering standard geostatistical analysis is given in section 5.5.3.

geoR also allows for Bayesian kriging; its extension — the package geoRglm — can work with binomial and Poisson processes (Ribeiro Jr et al., 2003). In comparison, fitting a Generalized Linear Geostatistical Model (GLGM) can be more conclusive than fitting simple linear models in gstat since we can model the geographical and regression terms more objectively (Diggle and Ribeiro Jr, 2007). This was, for example, the original motivation for the geoRglm and spBayes packages (Ribeiro Jr et al., 2003). However, GLGMs are not yet ready for operational mapping, and R code will need to be adapted.

3.2.4 Isatis

Isatis[40] is probably the most expensive geostatistical package (>10K €) available on the market today. However, it is widely regarded as one of the most professional packages for environmental sciences. Isatis was originally built for Unix, but there are MS Windows and Linux versions also. From the launch of the package in 1993, >1000 licences have been purchased worldwide. Standard Isatis clients are Oil and Gas companies, consultancy teams, mining corporations and environmental agencies. The software will be here shortly presented for informative purposes. We will not use Isatis in the exercises, but it is worth mentioning it.



Data File... Srivastava / samples [Selection = None]

U
V

Statistical Representations

- [✓] Define Parameters Before Initial Calculations
- [ ] Use the Grid Organization
- [✓] Display Using the Reference Variable: ... U
- [✓] Compute Using the Weight Variable: ... [R] Weight Variable = weight on

Fig. 3.8: Exploratory data analysis possibilities in Isatis.

Isatis offers a wide range of geostatistical functions ranging from 2D/3D isotropic and directional variogram modeling, univariate and multivariate kriging, punctual and block estimation, drift estimation, universal kriging, col-

[38] `http://www.leg.ufpr.br/geoR/geoRdoc/tutorials.html`

[39] Other comparable packages with geostatistical analysis are fields, spatial, sgeostat and RandomFields, but this book for practical reasons focuses only on gstat and geoR.

[40] `http://www.geovariances.com` — the name "Isatis" is not an abbreviation. Apparently, the creators of Isatis were passionate climbers so they name their package after one climbing site in France.

located co-kriging, kriging with external drift, kriging with inequalities (introduce localized constraints to bound the model), factorial kriging, disjunctive kriging etc. Isatis especially excels with respect to interactivity of exploratory analysis, variogram modeling, detection of local outliers and anisotropy (Fig. 3.8).

Regression-kriging in Isatis can be run by selecting *Interpolation* ↦ *Estimation* ↦ *External Drift (Co)-kriging*. Here you will need to select the target variable (point map), predictors and the variogram model for residuals. You can import the point and raster maps as shapefiles and raster maps as ArcView ASCII grids (importing/exporting options are limited to standard GIS formats). Note that, before you can do any analysis, you first need to define the project name and working directory using the data file manager. After you import the two maps, you can visualize them using the display launcher.

Note that KED in Isatis is limited to only one (three when scripts are used) auxiliary raster map (called *background variable* in Isatis). Isatis justifies limitation of number of auxiliary predictors by computational efficiency. In any case, a user can first run factor analysis on multiple predictors and then select the most significant component, or simply use the regression estimates as the auxiliary map. Isatis offers a variety of options for the automated fitting of variograms. You can also edit the *Model Parameter File* where the characteristics of the model you wish to apply for kriging are stored.

3.3 Geographical visualization: Google Earth (GE)

Google Earth[41], Google's geographical browser, is increasingly popular in the research community. Google Earth was developed by Keyhole, Inc., a company acquired by Google in 2004. The product was renamed Google Earth in 2005 and is currently available for use on personal computers running Microsoft Windows 2000, XP or Vista, Mac OS X and Linux. All displays in Google Earth are controlled by KML files, which are written in the **Keyhole Markup Language**[42] developed by Keyhole, Inc. KML is an XML-based language for managing the display of three-dimensional geospatial data, and is used in several geographical browsers (Google Maps, Google Mobile, ArcGIS Explorer and World Wind). The KML file specifies a set of standard features (placemarks, images, polygons, 3D models, textual descriptions, etc.) for display in Google Earth. Each place always has a longitude and a latitude. Other data can make the view more specific, such as tilt, heading, altitude, which together define a *camera view*. The KML data sets can be edited using an ASCII editor (as with HTML), but they can be edited also directly in Google Earth. KML files are very often distributed as KMZ files, which are zipped KML files with a `.kmz` extension. Google has recently '*given away*' KML to the general public, i.e. it has been registered as an **OGC standard**[43].

To install Google Earth, run the installer that you can obtain from Google's website. To start a KML file, just double-click it and the map will be displayed using the default settings. Other standard background layers, such as roads, borders, places and similar geographic features, can be turned on or off using the Layers panel. There are also commercial Plus and Pro versions of Google Earth, but for purpose of our exercises, the free version has all necessary functionality.

The rapid emergence and uptake of Google Earth may be considered evidence for a trend towards a more visual approach to spatial data handling. Google Earth's sophisticated spatial indexing of very large data sets combined with an open architecture for integrating and customizing new data is having a radical effect on many Geographic Information Systems (Wood, 2008; Craglia et al., 2008). If we extrapolate these trends, we could foresee that in 10 to 20 years time Google Earth will contain near to real-time global satellite imagery of photographic quality, all cars and transportation vehicles (even people) will be GPS-tagged, almost every data will have spatio-temporal reference, and the Virtual Globe will be a digital 4D replica of Earth in the scale 1:1 (regardless of how 'scary' that seems).

One of biggest impacts of Google Earth and Maps is that they have opened up the exploration of spatial data to a much wider community of non-expert users. Google Earth is a ground braking software in at least five categories:

Availability — It is a free browser that is available to everyone. Likewise, users can upload their own geographic data and share it with anybody (or with selected users only).

High quality background maps — The background maps (remotely sensed images, roads, administrative units, topography and other layers) are constantly updated and improved. At the moment, almost

[41] `http://earth.google.com`
[42] `http://earth.google.com/kml/kml_tut.html` / `http://code.google.com/apis/kml/`
[43] `http://www.opengeospatial.org/standards/kml/`

30% of the world is available in high resolution (2 m IKONOS images[44]). All these layers have been georeferenced at relatively high accuracy (horizontal RMSE of 20–30 m or better) and can be used to validate the spatial accuracy of moderate-resolution remote sensing products and similar GIS layers (Potere, 2008). Overlaying GIS layers of unknown accuracy on GE can be revealing. Google has recently agreed to license imagery for their mapping products from the GeoEye-1 satellite. In the near future, Google plans to update 50 cm resolution imagery with near to global coverage on monthly basis.

A single coordinate system — The geographic data in Google Earth is visualized using a 3D model (central projection) rather than a projected 2D system. This practically eliminates all the headaches associated with understanding projection systems and merging maps from different projection systems. However, always have in mind that any printed Google Earth display, although it might appear to be 2D, will always show distortions due to Earth's curvature (or due to relief displacements). At very detailed scales (blocks of the buildings), these distortions can be ignored so that distances on the screen correspond closely to distances on the ground.

Web-based data sharing — Google Earth data is located on internet servers so that the users do not need to download or install any data locally. Rasters are distributed through a tiling system: by zooming in or out of the map, only local tiles at different scales (19 zoom levels) will be downloaded from the server.

Popular interface — Google Earth, as with many other Google products, is completely user-oriented. What makes Google Earth especially popular is the impression of literarily flying over Earth's surface and interactively exploring the content of various spatial layers.

API services — A variety of geo-services are available that can be used via Java programming interface or similar. By using the Google Maps API one can geocode the addresses, downloading various static maps and attached information. Google Maps API service in fact allow mash-ups that often exceed what the creators of software had originally in mind.

There are several competitors to Google Earth (NASA's World Wind[45], ArcGIS Explorer[46], 3D Weather Globe[47], Capaware[48]), although none of them are equivalent to Google Earth in all of the above-listed aspects. On the other hand, Google Earth poses some copyright limitations, so you should first read the *Terms and Conditions*[49] before you decide to use it for your own projects. For example, Google welcomes you to use any of the multimedia produced using Google tools as long as you preserve the copyrights and attributions including the Google logo attribution. However, you cannot sell these to others, provide them as part of a service, or use them in a commercial product such as a book or TV show without first getting a rights clearance from Google.

While at present Google Earth is primarily used as a geo-browser for exploring spatially referenced data, its functionality can be integrated with geostatistical tools, which can stimulate sharing of environmental data between international agencies and research groups. Although Google Earth does not really offer much standard GIS functionality, it can be used also to add content, such as points or lines to existing maps, measure areas and distances, derive UTM coordinates and eventually load GPS data. Still, the main use of Google Earth depends on its visualization capabilities that cannot be compared to any desktop GIS. The base maps in Google Earth are extensive and of high quality, both considering spatial accuracy and content. In that sense, Google Earth is a GIS that exceeds any existing public GIS in the world.

To load your own GIS data to Google Earth, there are several possibilities. First, you need to understand that there is a difference between loading vector and raster maps into Google Earth. Typically, it is relatively easy to load vector data such as points or lines into Google Earth, and somewhat more complicated to do the same with raster maps. Note also that, because Google Earth works exclusively with Latitude/Longitude projection system (WGS84[50] ellipsoid), all vector/raster maps need to be first reprojected before they can be

[44]Microsoft recently released the Bing Maps 3D (`http://www.bing.com/maps/`) service that also has an impressive map/image coverage of the globe.

[45]`http://worldwind.arc.nasa.gov/`

[46]`http://www.esri.com/software/arcgis/explorer/`

[47]`http://www.mackiev.com/3d_globe.html`

[48]`http://www.capaware.org`

[49]`http://www.google.com/permissions/geoguidelines.html`

[50]`http://earth-info.nga.mil/GandG/wgs84/`

exported to KML format. More about importing the data to Google Earth can be found via the Google Earth User Guide[51].

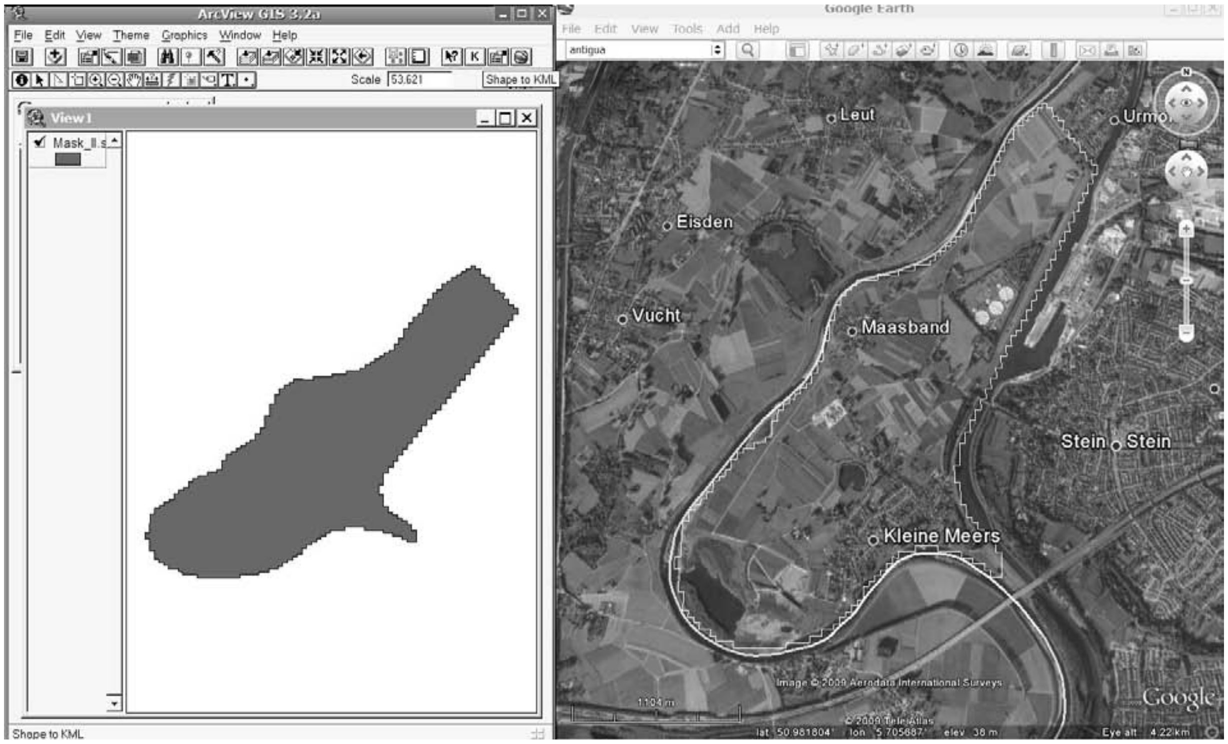


Fig. 3.9: Exporting ESRI shapefiles to KML using the SHAPE 2 KML ESRI script in ArcView 3.2. Note that the vector maps need to be first reprojected to LatLon WGS84 system.

3.3.1 Exporting vector maps to KML

Vector maps can be loaded by using various plugins/scripts in packages such as ArcView, MapWindow and R. Shapefiles can be directly transported to KML format by using ArcView's SHAPE 2 KML[52] script, courtesy of Domenico Ciavarella. To install this script, download it, unzip it and copy the two files to your ArcView 3.2 program directory:

- `..\ARCVIEW\EXT32\shape2KML.avx`
- `..\ARCVIEW\ETC\shp2kmlSource.apr`

This will install an extension that can be easily started from the main program menu (Fig. 3.9). Now you can open a layer that you wish to convert to KML and then click on the button to enter some additional parameters. There is also a commercial plugin for ArcGIS called Arc2Earth[53], which offers various export options. An alternative way to export shapefiles to KML is the Shape2Earth plugin[54] for the open-source GIS MapWindow. Although MapWindow is an open-source GIS, the Shape2Earth plugin is shareware so you might need to purchase it.

To export point or line features to KML in R, you can use the `writeOGR` method available in rgdal package. Export can be achieved in three steps, e.g.:

```
# 1. Load the rgdal package for GIS data exchange:
> require(c("rgdal","gstat","lattice","RASAGA","maptools","akima"))
```

[51]http://earth.google.com/userguide/v4/ug_importdata.html
[52]http://arcscripts.esri.com/details.asp?dbid=14254
[53]http://www.arc2earth.com
[54]http://www.mapwindow.org/download.php?show_details=29

```
# 2. Reproject the original map from local coordinates:
> data(meuse)
> coordinates(meuse) <- ∼ x+y
> proj4string(meuse) <- CRS("+init=epsg:28992")
> meuse.ll <- spTransform(meuse, CRS("+proj=longlat +datum=WGS84"))
# 3. Export the point map using the "KML" OGR driver:
> writeOGR(meuse.ll["lead"], "meuse_lead.kml", "lead", "KML")
```

See further p.119 for instructions on how to correctly set-up the coordinate system for the meuse case 1
study. A more sophisticated way to generate a KML is to directly write to a KML file using loops. This way 2
one has a full control of the visualization parameters. For example, to produce a bubble-type of plot (compare 3
with Fig. 5.2) in Google Earth with actual numbers attached as labels to a point map, we can do: 4

```
> varname <- "lead"  # variable name
> maxvar <- max(meuse.ll[varname]@data)  # maximum value
> filename <- file(paste(varname, "_bubble.kml", sep=""), "w")
> write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>", filename)
> write("<kml xmlns=\"http://earth.google.com/kml/2.2\">", filename, append = TRUE)
> write("<Document>", filename, append = TRUE)
> write(paste("<name>", varname, "</name>", sep=" "), filename, append = TRUE)
> write("<open>1</open>", filename, append = TRUE)
# Write points in a loop:
> for (i in 1:length(meuse.ll@data[[1]])) {
>   write(paste('  <Style id="','pnt', i,'">',sep=""), filename, append = TRUE)
>   write("    <LabelStyle>", filename, append = TRUE)
>   write("     <scale>0.7</scale>", filename, append = TRUE)
>   write("    </LabelStyle>", filename, append = TRUE)
>   write("               <IconStyle>", filename, append = TRUE)
>   write("  <color>ff0000ff</color>", filename, append = TRUE)
>   write(paste("        <scale>", meuse.ll[i,varname]@data[[1]]/maxvar*2+0.3,
+     "</scale>", sep=""), filename, append = TRUE)
>   write("                       <Icon>", filename, append = TRUE)
# Icon type:
>   write("   <href>http://maps.google.com/mapfiles/kml/shapes/donut.png</href>",
+     filename, append = TRUE)
>   write("    </Icon>", filename, append = TRUE)
>   write("          </IconStyle>", filename, append = TRUE)
>   write(" </Style>", filename, append = TRUE)
> }
> write("<Folder>", filename, append = TRUE)
> write(paste("<name>Donut icon for", varname,"</name>"), filename, append = TRUE)
# Write placemark style in a loop:
> for (i in 1:length(meuse.ll@data[[1]])) {
>   write("  <Placemark>", filename, append = TRUE)
>   write(paste("  <name>", meuse.ll[i,varname]@data[[1]],"</name>", sep=""),
+  filename, append = TRUE)
>   write(paste("  <styleUrl>#pnt",i,"</styleUrl>", sep=""), filename, append=TRUE)
>   write("    <Point>", filename, append = TRUE)
>   write(paste("      <coordinates>",coordinates(meuse.ll)[[i,1]],",",
+ coordinates(meuse.ll)[[i,2]],",10</coordinates>", sep=""), filename, append=TRUE)
>   write("    </Point>", filename, append = TRUE)
>   write("    </Placemark>", filename, append = TRUE)
> }
> write("</Folder>", filename, append = TRUE)
> write("</Document>", filename, append = TRUE)
> write("</kml>", filename, append = TRUE)
> close(filename)
# To zip the file use the 7z program:
# system(paste("7za a -tzip ", varname, "_bubble.kmz ", varname, "_bubble.kml", sep=""))
# unlink(paste(varname, "_bubble.kml", sep=""))
```

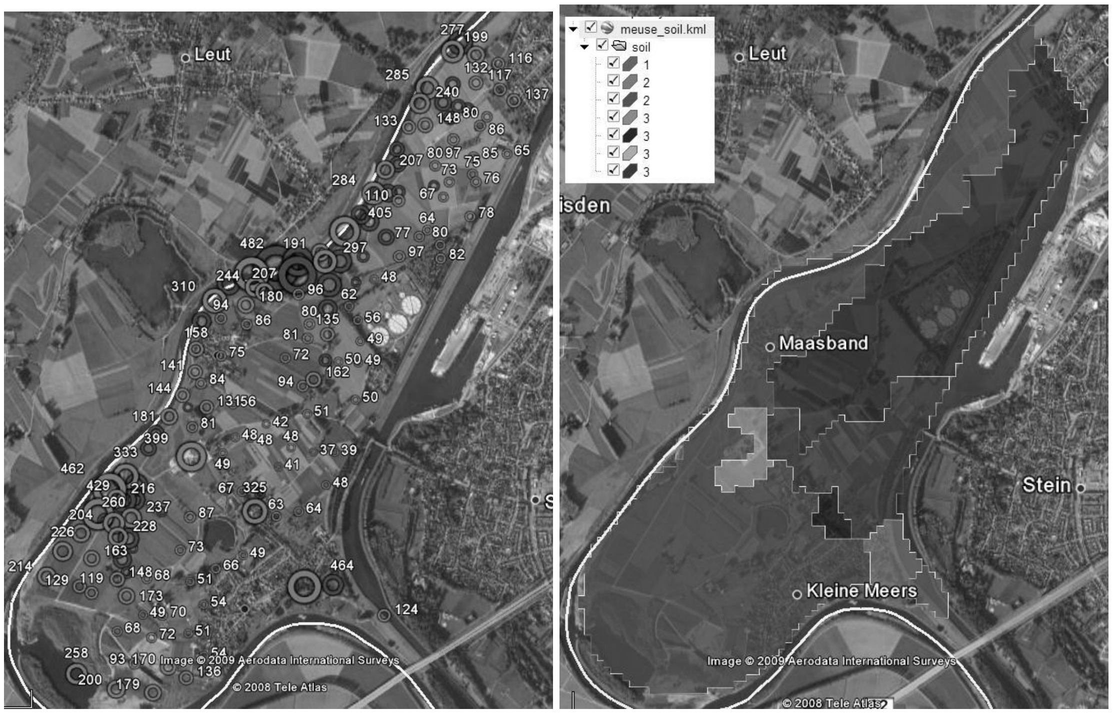


Fig. 3.10: Zinc values visualized using the bubble-type of plot in Google Earth (left). Polygon map (soil types) exported to KML and colored using random colors with transparency (right).

which will produce a plot shown in Fig. 3.10. Note that one can also output a multiline file by using e.g. `cat("ABCDEF", pi, "XYZ", file = "myfile.txt")`, rather than outputting each line separately (see also the `sep=` and `append=` arguments to `cat`).

Polygon maps can also be exported using the `writeOGR` command, as implemented in the package rgdal. In the case of the meuse data set, we first need to prepare a polygon map:

```
> data(meuse.grid)
> coordinates(meuse.grid) <- ~ x+y
> gridded(meuse.grid) <- TRUE
> proj4string(meuse.grid) <- CRS("+init=epsg:28992")
# raster to polygon conversion;
> write.asciigrid(meuse.grid["soil"], "meuse_soil.asc")
> rsaga.esri.to.sgrd(in.grids="meuse_soil.asc", out.sgrd="meuse_soil.sgrd",
+    in.path=getwd())
> rsaga.geoprocessor(lib="shapes_grid", module=6,
+    param=list(GRID="meuse_soil.sgrd", SHAPES="meuse_soil.shp", CLASS_ALL=1))
> soil <- readShapePoly("meuse_soil.shp", proj4string=CRS("+init=epsg:28992"),
+    force_ring=T)
> soil.ll <- spTransform(soil, CRS("+proj=longlat +datum=WGS84"))
> writeOGR(soil.ll["NAME"], "meuse_soil.kml", "soil", "KML")
```

The result can be seen in Fig. 3.10 (right). Also in this case we could have manually written the KML files to achieve the best effect.

3.3.2 Exporting raster maps (images) to KML

Rasters cannot be exported to KML as easily as vector maps. Google Earth does not allow import of GIS raster formats, but only input of images that can then be draped over a terrain (**ground overlay**). The images need to be exclusively in one of the following formats: JPG, BMP, GIF, TIFF, TGA and PNG. Typically, export of raster maps to KML follows these steps:

(1.) Determine the grid system of the map in the LatLonWGS84 system. You need to determine five parameters: southern edge (south), northern edge (north), western edge (west), eastern edge (east) and cellsize in arcdegrees (Fig. 3.11).

(2.) Reproject the original raster map using the new LatLonWGS84 grid.

(3.) Export the raster map using a graphical format (e.g. TIFF), and optionally the corresponding legend.

(4.) Prepare a KML file that includes a JPG of the map (Ground Overlay[55]), legend of the map (Screen Overlay) and description of how the map was produced. The JPG images you can locate on some server and then refer to an URL.

For data sets in geographical coordinates, a cell size correction factor can be estimated as a function of the latitude and spacing at the equator (Hengl and Reuter, 2008, p.34):

$$\Delta x_{\text{metric}} = F \cdot \cos(\varphi) \cdot \Delta x^0_{\text{degree}} \tag{3.3.1}$$

where Δx_{metric} is the East/West grid spacing estimated for a given latitude (φ), $\Delta x^0_{\text{degree}}$ is the grid spacing in degrees at equator, and F is the empirical constant used to convert from degrees to meters (Fig. 3.11).

Once you have resampled the map, you can then export it as an image and copy to some server. Examples how to generate KML ground overlays in R are further discussed in sections 5.6.2 and 10.6.3. A KML file that can be used to visualize a result of geostatistical mapping has approximately the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
<Document>
    <name>Raster map example</name>
    <GroundOverlay>
        <name>Map name</name>
        <description>Description of how was map produced.</description>
        <Icon>
                <href>exported_map.jpg</href>
            </Icon>
            <LatLonBox>
                <north>51.591667</north>
                <south>51.504167</south>
                <east>10.151389</east>
                <west>10.010972</west>
            </LatLonBox>
        </GroundOverlay>
        <ScreenOverlay>
            <name>Legend</name>
            <Icon>
                <href>map_legend.jpg</href>
            </Icon>
            <overlayXY x="0" y="1" xunits="fraction" yunits="fraction"/>
            <screenXY x="0" y="1" xunits="fraction" yunits="fraction"/>
            <rotationXY x="0" y="0" xunits="fraction" yunits="fraction"/>
            <size x="0" y="0" xunits="fraction" yunits="fraction"/>
        </ScreenOverlay>
</Document>
</kml>
```

The output map and the associated legend can be both placed directly on a server. An example of ground overlay can be seen in Fig. 5.20 and Fig. 10.12. Once you open this map in Google Earth, you can edit it, modify the transparency, change the icons used and combine it with other vector layers. Ground Overlays can also be added directly in Google Earth by using commands *Add* ↦ *Image Overlay*, then enter the correct

[55] http://code.google.com/apis/kml/documentation/kml_tut.html

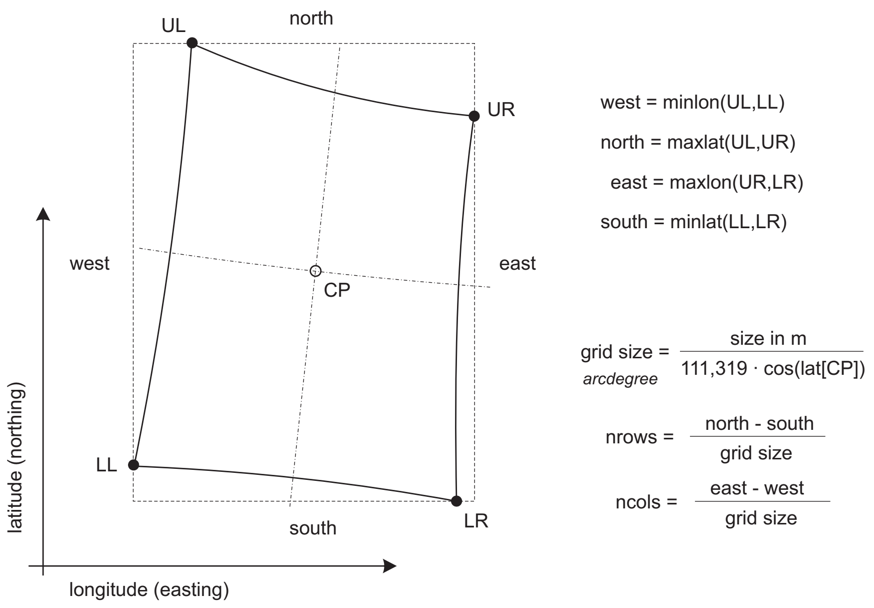
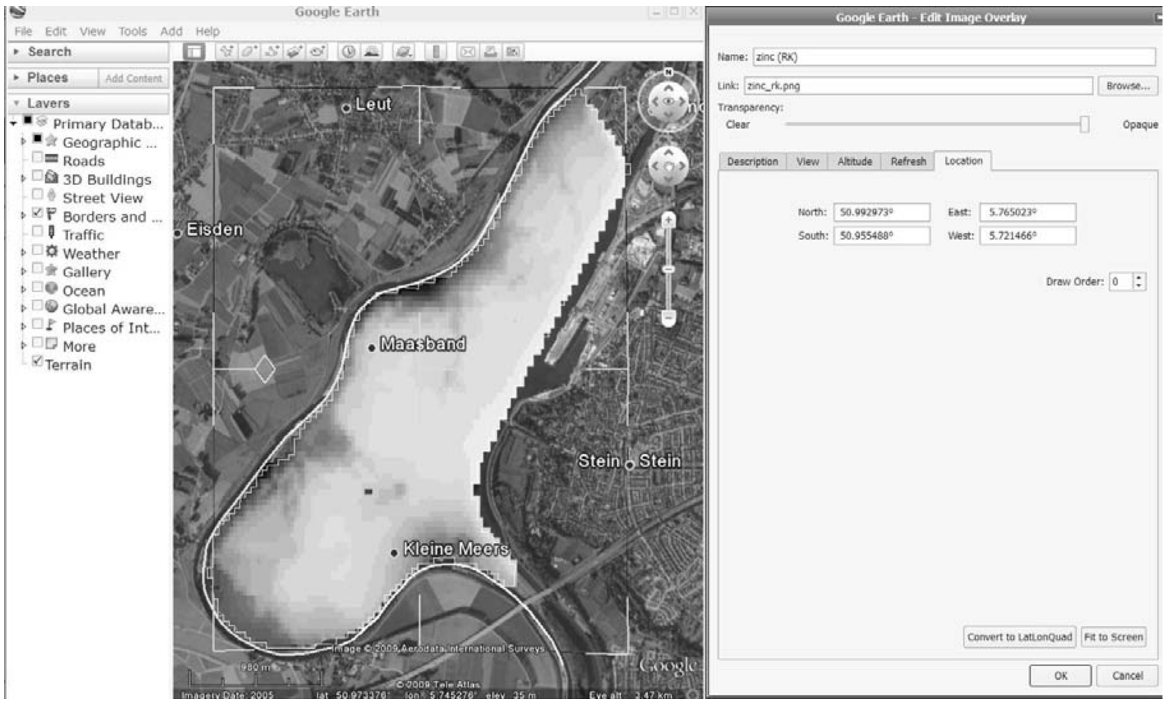


Fig. 3.11: Determination of the bounding coordinates and cell size in the `LatLonWGS84` geographic projection system using an existing Cartesian system. For large areas (continents), it is advisable to visually validate the estimated values.



Name: zinc (RK)
Link: zinc_rk.png
Transparency: Clear — Opaque

North: 50.992973° East: 5.765023°
South: 50.955488° West: 5.721466°

Draw Order: 0

Fig. 3.12: Preparation of the image ground overlays using the Google Earth menu.

bounding coordinates and location of the image file (Fig. 3.12). Because the image is located on some server, it can also be automatically refreshed and/or linked to a Web Mapping Service (WMS). For a more sophisticated use of Google interfaces see for example the interactive **KML sampler**[56], that will give you some good ideas about what is possible in Google Earth. Another interactive KML creator that plots various (geographical) CIA World Factbook, World Resources Institute EarthTrends and UN Data is the KML FactBook[57].

Another possibility to export the gridded maps to R (without resampling grids) is to use the vector structure of the grid, i.e. to export each grid node as a small squared polygon[58]. First, we can convert the grids to polygons using the maptools package and reproject them to geographic coordinates (Bivand et al., 2008):

```
# generate predictions e.g.:
> zinc.rk <- krige(log1p(zinc) ~ dist+ahn, data=meuse, newdata=meuse.grid,
+           model=vgm(psill=0.151, "Exp", range=374, nugget=0.055))
> meuse.grid$zinc.rk <- expm1(zinc.rk$var1.pred)
# convert grids to pixels (mask missing areas):
> meuse.pix <- as(meuse.grid["zinc.rk"], "SpatialPixelsDataFrame")
# convert grids to polygons:
> grd.poly <- as.SpatialPolygons.SpatialPixels(meuse.pix)
# The function is not suitable for high-resolution grids!!
> proj4string(grd.poly) <- CRS("+init=epsg:28992")
> grd.poly.ll <- spTransform(grd.poly, CRS("+proj=longlat +datum=WGS84"))
> grd.spoly.ll <- SpatialPolygonsDataFrame(grd.poly.ll,
+       data.frame(meuse.pix$zinc.rk), match.ID=FALSE)
```

Next, we need to estimate the Google codes for colors for each polygon. The easiest way to achieve this is to generate an RGB image in R, then reformat the values following the KML tutorial:

```
> tiff(file="zinc_rk.tif", width=meuse.grid@grid@cells.dim[1],
+       height=meuse.grid@grid@cells.dim[2], bg="transparent")
> par(mar=c(0,0,0,0), xaxs="i", yaxs="i", xaxt="n", yaxt="n")
> image(as.image.SpatialGridDataFrame(meuse.grid["zinc.rk"]), col=bpy.colors())
> dev.off()
# read RGB layer back into R:
> myTile <- readGDAL("zinc_rk.tif", silent=TRUE)
> i.colors <- myTile@data[meuse.pix@grid.index,]
> i.colors[1:3,]

     band1 band2 band3
 69     72     0   255
 146    94     0   255
 147    51     0   255


> i.colors$B <- round(i.colors$band3/255*100, 0)
> i.colors$G <- round(i.colors$band2/255*100, 0)
> i.colors$R <- round(i.colors$band1/255*100, 0)
# generate Google colors:
> i.colors$FBGR <- paste("9d", ifelse(i.colors$B<10, paste("0", i.colors$B, sep=""),
+     ifelse(i.colors$B==100, "ff", i.colors$B)),
+     ifelse(i.colors$G<10, paste("0", i.colors$G, sep=""),
+     ifelse(i.colors$G==100, "ff", i.colors$G)),
+     ifelse(i.colors$R<10, paste("0", i.colors$R, sep=""),
+     ifelse(i.colors$R==100, "ff", i.colors$R)), sep="")
> i.colors$FBGR[1:3]

 [1] "9dff0028" "9dff0037" "9dff0020"
```

and we can write Polygons to KML with color attached in R:

[56] `http://kml-samples.googlecode.com/svn/trunk/interactive/index.html`
[57] `http://www.kmlfactbook.org/`
[58] This is really recommended only for fairly small grid, e.g. with $\ll 10^6$ grid nodes.

```
> varname <- "zinc_rk"  # variable name
> filename <- file(paste(varname, "_poly.kml", sep=""), "w")
> write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>", filename)
> write("<kml xmlns=\"http://earth.google.com/kml/2.2\">", filename, append = TRUE)
> write("<Document>", filename, append = TRUE)
> write(paste("<name>", varname, "</name>", sep=" "), filename, append = TRUE)
> write("<open>1</open>", filename, append = TRUE)
> for (i in 1:length(grd.spoly.ll@data[[1]])) {
>   write(paste(' <Style id="','poly', i,'">',sep=""), filename, append = TRUE)
>   write("   <LineStyle>", filename, append = TRUE)
>   write("   <width>0.5</width>", filename, append = TRUE)
>   write("   </LineStyle>", filename, append = TRUE)
>   write("                <PolyStyle>", filename, append = TRUE)
>   write(paste('      <color>', i.colors$FBGR[i], '</color>', sep=""),
+         filename, append = TRUE)
>   write("          </PolyStyle>", filename, append = TRUE)
>   write(" </Style>", filename, append = TRUE)
> }
> write("<Folder>", filename, append = TRUE)
> write(paste("<name>Poly ID", varname,"</name>"), filename, append = TRUE)
> for (i in 1:length(grd.spoly.ll@data[[1]])) {
>   write("  <Placemark>", filename, append = TRUE)
>   write(paste("  <name>", grd.spoly.ll@polygons[[i]]@ID, "</name>", sep=""),
+         filename, append = TRUE)
>   write("  <visibility>1</visibility>", filename, append = TRUE)
>   write(paste("  <styleUrl>#poly", i, "</styleUrl>", sep=""), filename, append = TRUE)
>   write("    <Polygon>", filename, append = TRUE)
>   write("    <tessellate>1</tessellate>", filename, append = TRUE)
>   write("      <altitudeMode>extruded</altitudeMode>", filename, append = TRUE)
>   write("      <outerBoundaryIs>", filename, append = TRUE)
>   write("        <LinearRing>", filename, append = TRUE)
>   write("      <coordinates>", filename, append = TRUE)
>   write(paste("          ", grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[1,1], ",",
+        grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[1,2], ",1", sep=""),
+        filename, append = TRUE)
>   write(paste("          ", grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[2,1], ",",
+        grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[2,2], ",1", sep=""),
+        filename, append = TRUE)
>   write(paste("          ", grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[3,1], ",",
+        grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[3,2], ",1", sep=""), ",",
+        filename, append = TRUE)
>   write(paste("          ", grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[4,1], ",",
+        grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[4,2], ",1", sep=""),
+        filename, append = TRUE)
>   write(paste("          ", grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[5,1], ",",
+        grd.spoly.ll@polygons[[i]]@Polygons[[1]]@coords[5,2], ",1", sep=""),
+        filename, append = TRUE)
>   write("      </coordinates>", filename, append = TRUE)
>   write("        </LinearRing>", filename, append = TRUE)
>   write("      </outerBoundaryIs>", filename, append = TRUE)
>   write("    </Polygon>", filename, append = TRUE)
>   write("    </Placemark>", filename, append = TRUE)
> }
> write("</Folder>", filename, append = TRUE)
> write("</Document>", filename, append = TRUE)
> write("</kml>", filename, append = TRUE)
> close(filename)
```

In similar fashion, time-series of maps (range of maps covering the same geographic domain, but referring to a different time period) can be exported and explored in Google Earth using **time-slider**. Note also that a list of remotely sensed image bands or maps produced by geostatistical simulations can be exported as a time-series, and then visualized as animation. The maptools package is not able to generate GE KMLs using space-time data[59], but we can instead write directly a KML file from R. An example is further shown in section 11.5.3.

Users of MatLab can explore possibilities for exporting the results of geostatistical analysis to Google Earth by using the Google Earth Toolbox[60]. This toolbox serves not only to export maps (vectors, ground overlays), but also as a friendly tool to export the associated legends, generate 3D surfaces, contours from isometric maps, wind barbs and 3D vector objects. For example, a gridded map produced using some spatial prediction technique can be converted to a KML format using e.g.:

```
MATLAB: examplemap = ge_groundoverlay(N,E,S,W,... 'imageURL','mapimage.png');
MATLAB: ge_output('examplemap.kml',kmlStr);
```

where `N,E,S,W` are the bounding coordinates that can be determined automatically or set by the user.

3.3.3 Reading KML files to R

In principle, everything we export from R to KML we can also read back into R. GDAL has an OGR KML driver that should work for vector data also on Windows OS with the standard rgdal binary. The GDAL website[61] indicates that KML reading is only available if GDAL/OGR is built with the Expat XML Parser, otherwise only KML writing will be supported. Supported OGR geometry types are: `Point`, `Linestring`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` and `MultiGeometry`. Unfortunately, reading of more complex KML files is still a cumbersome. Reading of KML files can not easily be automated because the code often requires editing, for the simple reason that KML does not (yet) have standard spatial data formats.

The most recent version of the package maptools contains methods (e.g. `getKMLcoordinates`) to read KML files to R and coerce them to the sp formats. KML could carry a lot of non-spatial information, or spatial information that is not supported by most GIS (e.g. viewing angle, transparency, description tags, style definition etc.). In addition, you could have more maps within the same KML file, which makes it difficult to automate import of KML files to a GIS.

To read the point map `meuse_lead.kml` we exported previously using the `writeOGR` method, we can always use the XML[62] package:

```
> library(XML)
> meuse_lead.kml <- xmlTreeParse("meuse_lead.kml")
> lengthp <- length(meuse_lead.kml$doc[[1]][[1]][[1]])-1
> lead_sp <- data.frame(Longitude=rep(0,lengthp), Latitude=rep(0,lengthp),
+    Var=rep(0,lengthp))
> for(j in 1:lengthp) {
>   LatLon <- unclass(meuse_lead.kml$doc[[1]][[1]][[1]][j+1][[1]][2][[1]][[1]][[1]])$value
>   Var <- unclass(meuse_lead.kml$doc[[1]][[1]][[1]][j+1][[1]][1][[1]][[1]])$value
>   lead_sp$Longitude[[j]] <- as.numeric(matrix(unlist(strsplit(LatLon,
+      split=",")), ncol=2)[1])
>   lead_sp$Latitude[[j]] <- as.numeric(matrix(unlist(strsplit(LatLon,
+      split=",")), ncol=2)[2])
>   lead_sp$Var[[j]] <- as.numeric(matrix(unlist(strsplit(strsplit(Var,
+      split="<i>")[[1]][2], split="</i>")), ncol=2)[1])
> }
> coordinates(lead_sp) <- ~ Longitude+Latitude
> proj4string(lead_sp) <- CRS("+proj=longlat +ellps=WGS84")
> bubble(lead_sp, "Var")
```

Note that it will take time until you actually locate where in the KML file the coordinates of points and attribute values are located (note long lines of sub-lists). After that it is relatively easy to automate creation of a `SpatialPointsDataFrame`. This code could be shorten by using the `xmlGetAttr()`, `xmlChildren()` and `xmlValue()` methods. You might also consider using the KML2SHP[63] converter (ArcView script) to read KML files (points, lines, polygons) and generate shapefiles directly from KML files.

[59] The new stpp package (see R-forge) is expected to bridge this gap.

[60] http://www.mathworks.com/matlabcentral/fileexchange/12954

[61] http://www.gdal.org/ogr/drv_kml.html

[62] http://cran.r-project.org/web/packages/XML/

[63] http://arcscripts.esri.com/details.asp?dbid=14988

3.4 Summary points

3.4.1 Strengths and limitations of geostatistical software

Both open source and proprietary software packages have strong and weak points. A comparison of different aspects of geostatistical packages listed at the AI-Geostats website[64] and several well-known GIS packages can be seen in Table 3.1. Although universal kriging (using coordinates) is available in most geostatistical packages, kriging with external drift with multiple auxiliary maps can be run in only a limited number of packages. From all software listed in Table 3.1, only Isatis, SAGA, and gstat/geoR (as stand-alone application or integrated into R) offer the possibility to interpolate a variable using (multiple) auxiliary maps (Hengl et al., 2007a). Note that the comparison of packages is not trivial because many proprietary packages rely on plugins/toolboxes that are either distributed separately or are valid for certain software versions only. For example, ArcGIS has excellent capabilities for writing and reading KML files, but the user needs to obtain the Arc2Earth package, which is sold separately.

Hengl et al. (2007a) tested regression-kriging in variety of packages to discover that RK in Isatis is limited to use of a single (three in script mode) auxiliary maps (Bleines et al., 2004). In gstat, both RK predictions and simulations (predictors as base maps) at both point and block support can be run by defining short scripts, which can help automatize interpolation of large amounts of data. However, gstat implements the algorithm with extended matrix (KED), which means that both the values of predictors and of target variable are used to estimate the values at each new location, which for large data sets can be time-consuming or can lead to computational problems (Leopold et al., 2005). geoR stands out as a package with the most sophisticated approaches to model estimation, but it is not really operational for use with large data sets. It was not designed to work with common spatial classes and data formats and some manual work is needed to get the maps out (see further section 5.5.3).

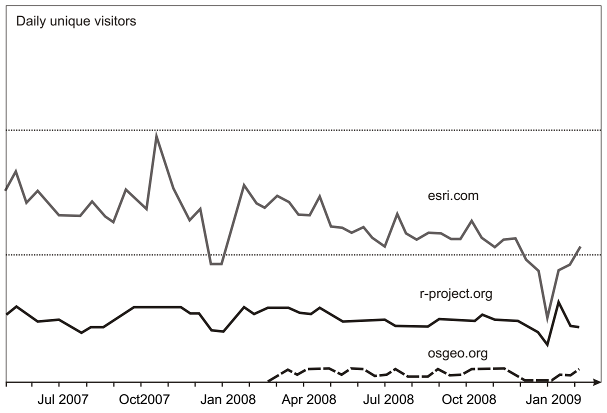


Fig. 3.13: Trends in the web-traffic for esri.com, r-project.org and osgeo.org (following the trends.google.com statistics).

Setting RK in gstat or SAGA to a smaller window search can lead to termination of the program due to singular matrix problems[65]. In fact, local RK with a global variogram model is not entirely valid because the regression model will differ locally, hence the algorithm should also estimate the variogram model for residuals for each local neighborhood (as mentioned previously in §2.2). The singular matrix problem will happen especially when indicator variables are used as predictors or if the two predictor maps are highly correlated. Another issue is the computational effort. Interpolation of $\gg 10^3$ points over 1M of pixels can last up to several hours on a standard PC. To run simulations in R+gstat with the same settings will take even more time. This proves that, although the KED procedure is mathematically elegant, it might be more effective for real-life applications to fit the trend and residuals separately (the regression-kriging approach). A limitation of `gstat.exe` is that it is a stand-alone application and the algorithms cannot be adjusted easily. Unlike the gstat package in R that can be extended and then uploaded by anybody. A limitation of R, however, is that it can reach memory use problems if larger rasters or larger quantities of rasters are loaded into R. Visualization and visual exploration of large maps in R is also not recommended.

So in summary, if you wish to fit your data in a statistically optimal way and with no limitations on the number of predictors and statistical operations — R and packages gstat and geoR are the most professional choice. If you do not feel confident about using software environment without an interface, then you could try running global Universal kriging in SAGA. However, SAGA does not provide professional variogram modeling capabilities, so a combination R+GIS+GE is probably the best idea. In fact, the computation is a bit faster in SAGA than in R and there are no memory limit problems (see further section 5.5.2). However, in SAGA you

[64] `http://ai-geostats.org`

[65] You can prevent gstat from breaking predictions by setting the `krige` option `set=list(cn_max=1e10, debug=0)`.

Table 3.1: Comparison of spatio-temporal data analysis capabilities of some popular statistical and GIS packages (versions in year 2009): ★ — full capability, ⋆ — possible but with many limitations, − — not possible in this package. Commercial price category: I — > 1000 EUR; II — 500-1000 EUR; III — < 500 EUR; IV — open source or freeware. Main application: A — statistical analysis and data mining; B — interpolation of point data; C — processing / preparation of input maps; E — visualization and visual exploration. After Hengl et al. (2007a).

Aspect	S-PLUS	R+gstat	R+geoR	MatLab	SURFER	ISATIS	GEOEas	GSLIB	GRASS	PC Raster	ILWIS	IDRISI	ArcGIS	SAGA
Commercial price category	II	IV	IV	I	III	I	IV	IV	IV	III	IV	II	I	IV
Main application	A, B	A, B	A, B	A, E	B, E	B	B	B	B, C	C	B, C	B, C	B, E	B, C
User-friendly environment to non-expert	★	−	−	★	★	★	−	−	⋆	−	★	★	★	★
Quality of support and description of algorithms	⋆	★	★	★	★	★	★	★	⋆	⋆	★	★	⋆	⋆
Standard GIS capabilities	−	⋆	−	⋆	⋆	−	−	−	★	⋆	★	★	★	★
GDAL support	−	★	⋆	⋆	−	⋆	−	−	★	⋆	⋆	★	★	⋆
Standard descriptive statistical analysis	★	★	★	★	−	★	⋆	⋆	⋆	−	★	★	★	⋆
Image processing tools (orthorectification, filtering, land surface analysis)	−	−	−	⋆	−	−	−	−	★	−	★	★	⋆	★
Comprehensive regression analysis (regression trees, GLM)	★	★	★	★	−	⋆	−	−	−	−	−	⋆	−	−
Interactive (automated) variogram modeling	−	⋆	★	−	−	★	−	⋆	−	−	−	★	★	−
Regression-kriging with auxiliary maps	−	★	⋆	−	−	⋆	−	−	★	⋆	⋆	★	−	★
Dynamic modeling (simulations, spatial iterations, propagation, animations)	−	⋆	⋆	⋆	−	−	−	−	⋆	★	⋆	⋆	⋆	⋆
Processing of large data sets	★	⋆	−	⋆	⋆	★	−	−	★	⋆	⋆	★	★	★
Export of maps to Google Earth (KML)	−	★	⋆	★	−	−	−	−	⋆	−	−	⋆	★	−

will not be able to objectively estimate the variogram of residuals or GLS model for the deterministic part of variation.

The R+SAGA/GRASS+GE combo of applications allows full GIS + statistics integration and can support practically 80% of processing/visualization capabilities available in proprietary packages such as ArcInfo/Map or Idrisi. The advantage of combining R with open source GIS is that you will be able to process and visualize even large data sets, because R is not really suited for processing large data volumes, and it was never meant to be used for visual exploration or editing of spatial data. On the other hand, packages such as ILWIS and SAGA allow you to input, edit and visually explore geographical data, before and after the actual statistical analysis. Note also that ILWIS, SAGA and GRASS extend the image processing functionality (especially image filtering, resampling, geomorphometric analysis and similar) of R that is, at the moment, limited to only few experimental packages (e.g. biOps, rimage; raster[66]). An alternative for GIS+R integration is QGIS[67], which has among its main characteristics a python console, and a very elaborate way of adding Python plugins, which is already in use used for an efficient R plugin (manageR).

In principle, we will only use open source software to run the exercises in this book, and there are several good reasons. Since the 1980's, the GIS research community has been primarily influenced by the (proprietary) software licence practices that limited sharing of ideas and user-controlled development of new functionality (Steiniger and Bocher, 2009). With the initiation of the **Open Source Geospatial Foundation** (OSGeo), a new era began: the development and use of open source GIS has experienced a boom over the last few years; the enthusiasm to share code, experiences, and to collaborate on projects is growing (Bivand, 2006).

Steiniger and Bocher (2009) recognize four indicators of this trend: (1) increasing number of projects run using the open source GIS, (2) increasing financial support by government agencies, (3) increasing download rates, and (4) increasing number of use-cases. By comparing the web-traffic for proprietary and open source GIS (Fig. 3.13) one can notice that OSGeo has indeed an increasing role in the world market of GIS. Young and senior researchers are slowly considering switching from using proprietary software such as ESRI's ArcGIS and/or Mathworks' MatLab to R+SAGA, but experience (Windows *vs* Linux) teaches us that it would be over-optimistic to expect that this shift will go fast and without resistance.

3.4.2 Getting addicted to R

From the previously-discussed software tools, one software needs to be especially emphasized, and that is R (R Development Core Team, 2009). Many R users believe that there is not much in statistics that R cannot do[68] (Zuur et al., 2009). Certainly, the number of packages is increasing everyday, and so is the community. There are at least five good (objective) reasons why you should get deeper into R (Rossiter, 2009):

It is of high quality — It is a non-proprietary product of international collaboration between top statisticians.

It helps you think critically — It stimulates critical thinking about problem-solving rather than a *push the button* mentality.

It is an open source software — Source code is published, so you can see the exact algorithms being used; expert statisticians can make sure the code is correct.

It allows automation — Repetitive procedures can easily be automated by user-written scripts or functions.

It helps you document your work — By scripting in R, anybody is able to reproduce your work (processing metadata). You can record steps taken using history mechanism even without scripting, e.g. by using the `savehistory()` command.

It can handle and generate maps — R now also provides rich facilities for interpolation and statistical analysis of spatial data, including export to GIS packages and Google Earth.

The main problem with R is that each step must be run via a command line, which means that the analyst must really be an R expert. Although one can criticize R for a lack of an user-friendly interface, in fact, most power users in statistics never use a GUI. GUI's are fine for baby-steps and getting started, but not for

[66]`http://r-forge.r-project.org/projects/raster/` — raster is possibly the most active R spatial project at the moment.

[67]`http://qgis.org/`

[68]This is probably somewhat biased statement. For example, R is not (yet) operational for processing of large images (filter analysis, map iterations etc.), and many other standard geographical analysis steps are lacking.

production work. The whole point is that one can develop a script or program that, once it is right, can be re-run and will produce exactly the same results each time (excluding simulations of course).

Here are some useful tips on how to get addicted to R. First, you should note that you can edit the R scripts in user-friendly script editors such as Tinn-R[69] or use the package R commander (Rcmdr[70]), which has an user-friendly graphical interface. Second, you should take small steps before you get into really sophisticated script development. Start with some simple examples and then try to do the same exercises with your own data. The best way to learn R is to look at existing scripts. For example, a French colleague, Romain François, maintains a gallery of R graphics[71] that is dedicated to the noble goal of getting you addicted to R. A similar-purpose website is the **R Graphical Manual**[72]. Robert I. Kabacoff maintains a website called **Quick-R**[73] that gives an overview of R philosophy and functionality. John Verzani maintains a website with **simple examples in R** that will get you going[74]. If you love cookbooks, R has those too[75]. In fact, you might make the following package the first you try. Type the following command into R to (1) download a tutorial package; (2) load the package in R; and (3) tell R to open a webpage with more information about the tutorial (Verzani, 2004):

```
> install.packages("UsingR")
> library(UsingR)
> UsingR("exercises")
```

Third, if your R script does not work, do not despair, try to obtain some specialized literature. Reading specialized literature from the experts is possibly the best way to learn script writing in R. Start with Zuur et al. (2009), then continue with classics such as Chambers and Hastie (1992), Venables and Ripley (2002) and/or Murrell (2006). Other excellent and freely available introductions to R are Kuhnert and Venables (2005), Burns (2009) and Rossiter (2009). If you are interested in running spatial analysis or geostatistics in R, then books by Bivand et al. (2008), Baddeley (2008) Reimann et al. (2008), and/or Diggle and Ribeiro Jr (2007) are a must.

Getting help

Because the R community is large, chances are good that the problems you encounter have already been solved or at least discussed, i.e. it is already *there*. The issue is how to find this information. There are several sources where you should look:

(1.) the help files locally installed on your machine;

(2.) mailing/discussion lists;

(3.) various website and tutorials, and

(4.) books distributed by commercial publishers;

You should really start searching in this order — from your machine to a bookstore — although it is always a good idea to cross-check all possible sources and compare alternatives. Please also bear in mind that (a) not everything that you grab from www is correct and up-to-date; and (b) it is also possible that you have an original problem that nobody else has experienced.

Imagine that you would like to find out which packages on your machine can run interpolation by kriging. You can quickly find out this by running the method `help.search`, which will give something like this:

```
> help.search("kriging")

 Help files with alias or concept or title matching 'kriging' using fuzzy matching:

 image.kriging(geoR)          Image or Perspective Plot with Kriging Results
 krige.bayes(geoR)            Bayesian Analysis for Gaussian Geostatistical Models
```

[69] http://www.sciviews.org/Tinn-R/
[70] http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/
[71] http://addictedtor.free.fr
[72] http://bm2.genes.nig.ac.jp/RGM2/
[73] http://www.statmethods.net
[74] http://www.math.csi.cuny.edu/Statistics/R/simpleR/
[75] http://www.r-cookbook.com

```
krige.conv(geoR)             Spatial Prediction -- Conventional Kriging
krweights(geoR)              Computes kriging weights
ksline(geoR)                 Spatial Prediction -- Conventional Kriging
legend.krige(geoR)           Add a legend to a image with kriging results
wo(geoR)                     Kriging example data from Webster and Oliver
xvalid(geoR)                 Cross-validation by kriging
krige(gstat)                 Simple, Ordinary or Universal, global or local,
                             Point or Block Kriging, or simulation.
krige.cv(gstat)              (co)kriging cross validation, n-fold or leave-one-out
ossfim(gstat)                Kriging standard errors as function of grid spacing
                             and block size
krige(sgeostat)              Kriging
prmat(spatial)               Evaluate Kriging Surface over a Grid
semat(spatial)               Evaluate Kriging Standard Error of Prediction over a Grid

Type 'help(FOO, package = PKG)' to inspect entry 'FOO(PKG) TITLE'.
```

This shows that kriging (and its variants) is implemented in (at least) four packages. We can now display the help for the method "krige" that is available in the package gstat:

```
> help(krige, package=gstat)
```

The archives of the mailing lists are available via the servers in Zürich. They are fairly extensive and the best way to find something useful is to search them. The fastest way to search all R mailing lists is to use the RSiteSearch method. For example, imagine that you are trying to run kriging and then the console gives you the following error message e.g.:

```
"Error : dimensions do not match: locations XXXX and data YYYY"
```

Based on the error message we can list at least 3–5 keywords that will help us search the mailing list, e.g.:

```
> RSiteSearch("krige {dimensions do not match}")
```

This will give over 15 messages[76] with a thread matching exactly your error message. This means that other people also had this problem, so now you only need to locate the right solution. You should sort the messages by date and then start from the most recent message. The answer to your problem will be in one of the replies submitted by the mailing list subscribers. You can quickly check if this is a solution that you need by making a small script and then testing it.

Of course, you can at any time Google the key words of interest. However, you might instead consider using the Rseek.org[77] search engine maintained by Sasha Goodman. The advantage of using Rseek over e.g. general Google is that it focuses only on R publications, mailing lists, vignettes, tutorials etc. The result of the search is sorted in categories, which makes it easier to locate the right source.

If you are eventually not able to find a solution yourself, you can try sending the description of your problem to a mailing list, i.e. asking the R *gurus*. Note that there are MANY R mailing lists[78], so you first have to be sure to find the right one. Sending a right message to a wrong mailing list will still leave you without an answer. Also have in mind that everything you send to a mailing list is public/archived, so better cross-check your message before you send it. When asking for a help from a mailing list, use the existing pre-installed data sets to describe your problem[79].

Do's:

- If you have not done so already, **read the R posting guide**[80]!
- **Use the existing pre-installed data sets** (come together with a certain package) **to describe your problem**. You can list all available data sets on your machine by typing `data()`. This way you do not have to attach your original data or waste time on trying to explain your case study.

[76]Unfortunately, `RSiteSearch()` no longer searches R-sig-geo — the full archive of messages is now on Nabble.
[77]`http://rseek.org`
[78]There are several specific **Special Interest Group** mailing lists; see `http://www.r-project.org/mail.html`.
[79]Then you only need to communicate the problem and not the specifics of a data set; there is also no need to share your data.
[80]`http://www.r-project.org/posting-guide.html`

- If your problem is completely specific to your data set, then **upload it an put it on some web-directory** so that somebody can access it and see what really goes on.
- **Link your problem to some previously described problems**; put it in some actual context (to really understand what I mean here, you should consider attending the Use R conferences).
- **Acknowledge the work (time spent) other people do to help you**.
- You can submit not only the problems you discover but also the **information that you think is interesting for the community**.

Don'ts:

- **Do not send poorly formulated questions**. Make sure you give technical description of your data, purpose of your analysis, even the details about your operating system, RAM etc. Try to put yourself in a position of a person that is interested to help — try to provide all needed information as if the person who is ready to help you would feel like sitting at your computer.
- **Do not send too much**. One message, one question (or better to say "*one message, one problem*"). Nobody reading R mailing lists has time to read long articles with multiple discussion points. Your problem should fit half the page; if somebody gets more interested, you can continue the discussion also off the list.
- R **comes with ABSOLUTELY NO WARRANTY**. If you loose data or get strange results, you are welcome to improve the code yourself (or consider obtaining some commercial software). **Complaining to a mailing list about what frustrates you about R makes no sense, because nobody is obliged to take any responsibility**.
- R is a community project (it is based on the solidarity between the users). **Think what you can do for the community and not what the community can do for you**.

Probably the worst thing that can happen to your question is that you do not get any reply (and this does not necessarily mean that nobody wants to help you or that nobody know the solution)! There are several possible reasons why this happened:

- **You have asked too much**! Some people post questions that could take weeks to investigate (maybe this justifies a project proposal?). Instead, you should always limit yourself to 1-2 concrete issues. Broader discussions about various more general topics and statistical theory are sometimes also welcome, but they should be connected with specific packages.
- **You did not introduce your question/topic properly**. If your question is very specific to your field and the subscribers cannot really understand what you are doing, you need to think of ways to introduce your field and describe the specific context. The only way to learn the language used by the R mailing lists is to browse the existing mails (archives).
- **You are requesting that somebody does a work for you that you could do yourself**! R and its packages are all open source, which allows YOU to double check the underlying algorithms and extend them where necessary. If you want other people to do programming for you, then you are at the wrong place (some commercial software companies do accept wish-lists and similar types of requests, but that's what they are paid for anyway).
- **Your question has been answered already few times and it is quite annoying that you did not do your homework to check this**.

Remember: everything you send to mailing lists reach large audiences (for example, R-sig-geo has +1000 subscribers), and it is archived on-line, so you should be more careful about what you post. If you develop a bad reputation of being ignorant and/or too sloppy, then people might start ignoring your questions even if they eventually start getting the right shape.

Tips for successful scripting in R

R is a command line based environment, but users do really write things directly to a command line. It is more common to first write using text editors (e.g. Tinn-R, JGR) and then "*send lines*" of code to R command line. When generating an R script, there are few useful tips that you might consider following (especially if you plan to share this script with a wider community):

- Document your code to explain what you are doing. Comments in R can be inserted after the "#" sign; **You can never put too many comments in your code!**
- **Add some meta-information about the script at the beginning of your script** — its authors, last update, purpose, inputs and outputs, reference where somebody can find more info (R scripts usually come as supplementary materials for project reports or articles) and difficulties one might experience. In many situations it is also advisable to mention the package version used to generate outputs. Because R is dynamic and many re-designs happen, some old scripts might become incompatible with the new packages.
- Once you tested your script and saw that it works, **tidy-up the code** — remove unnecessary lines, improve the code where needed, and test it using extreme cases (Burns (2009) provides some useful tips on how to improve scripting). In R, many equivalent operations can be run via different paths. In fact, the same techniques are commonly implemented in multiple packages. On the other hand, not all methods are equally efficient (speed, robustness), i.e. equally elegant, so that it is often worth investigating what might be the most elegant way to run some analysis. A good practice is to always write a number of smaller functions and then a function that does everything using the smaller functions. Note also that the variable names and list of input maps can be save and dealt with as with lists (see e.g. page166).
- **Place the data sets on-line** (this way you only need to distribute the script) and then call the data by using the `download.file` method in R;

All these things will make a life easier for your colleagues, but also to yourself if you decide to come back to your own script in few years (or few months). Another thing you might consider is to directly write the code and comments in Tinn-R using the Sweave[81] package. Note that you can still run this script from Tinn-R, you only need to specify where the R code begins (<<>>=) and ends (@). This way, you do not only distribute the code, but also all explanation, formulae etc. Building metadata for both the data sets and scripts you produce is becoming increasingly important.

Memory limit problems

Another important aspect to consider is R's ability to handle large data sets. Currently, many pharmaceutical organizations and financial institutions use R as their primary analytical engine. They crunch huge amounts of data, so it works on very large data sets. However loading, displaying and processing large geographic data sets like big raster maps ($\gg$1M pixels) can be problematic with R. Windows 32–bit OS can allocate a maximum of 2 GB of memory to an application[82]. Even if your computer has >2 GB of RAM, you will receive an error message like:

```
Error: cannot allocate vector of size 12.6 Mb
```

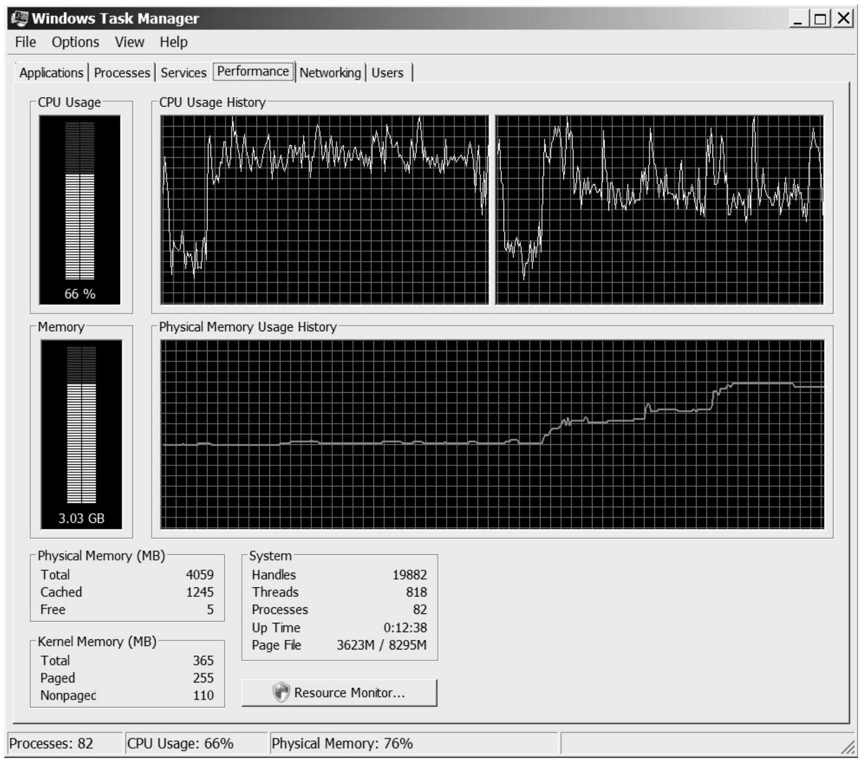
which might suggest to you that R has problems with surprisingly small data sets. This message actually means R has already allocated all of the 2 GB of memory available to it. Check the system's memory allocation by typing:

```
> gc()
```

and you will find R is already occupying a lot of RAM. The key issue is that R stores data in a temporary buffer, which can result in problems with memory availability. However, the benefit of this strategy is it allows maximum flexibility in data structures.. For Burns (2009) there are only two solutions to the problem: (1) Improve your code; (2) Get a bigger computer!

To increase your computing capacities, you can also consider doing one of the following:

[81] `http://www.stat.uni-muenchen.de/~leisch/Sweave/`
[82] Read more at: `http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.mspx`



CPU Usage: 66 %

Memory: 3.03 GB

Physical Memory (MB)
Total 4059
Cached 1245
Free 5

Kernel Memory (MB)
Total 365
Paged 255
Nonpagec 110

System
Handles 19882
Threads 818
Processes 82
Up Time 0:12:38
Page File 3623M / 8295M

Processes: 82 CPU Usage: 66% Physical Memory: 76%

Fig. 3.14: Windows task manager showing the CPU and memory usage. Once the computing in R comes close to 2 GB of physical memory, Windows will not allow R to use any more memory. The solution to this problem is to use a PC with an 64–bit OS.

- Reduce the grid resolution of your maps. If you reduce the grid cell size by half, the memory usage will be four times smaller.

- Consider splitting your data set into tiles. Load data tile by tile, write the results to physical memory or external database, remove temporary files, repeat the analysis until all tiles are finished. This is the so called "*database*" solution to memory handling (Burns, 2009).

- Obtain a new machine. Install a 64–bit OS with >10GB of RAM. A 64–bit OS will allow you to use more application memory.

- Consider obtaining a personal supercomputer[83] with a completely customizable OS. Supercomputer is about 200 times faster than your PC, mainly because it facilitates multicore operations. The price of a standard personal supercomputer is about 5–10 times that of a standard PC.

During the processing, you might try releasing some free memory by continuously using the `gc()` command. This will remove some temporary files and hence increase some free memory. If you are Windows OS user, you should closely monitor your Windows Task manager (Fig. 3.14), and then, when needed, use garbage collector (`gc()`) and/or remove (`rm()`) commands to increase free space. Another alternative approach is to combine R with other (preferably open-source) GIS packages, i.e. to run all excessive processing externally from R. It is also possible that you could run extensive calculations even with your limited PC. This is because processing is increasingly distributed. For example, colleagues from the Centre for e-Science in Lancaster have been recently developing an R package called MultiR[84] that should be able to significantly speed up R calculations by employing grid computing facilities (Grose et al., 2006).

[83] See e.g. `http://www.nvidia.com/object/personal_supercomputing.html`
[84] `http://cran.r-project.org/web/views/HighPerformanceComputing.html`

3.4.3 Further software developments

There are still many geostatistical operations that we are aware of, but have not been implemented and are not available to broader public (§2.10.3). What programmers might consider for future is the refinement of (local) regression-kriging in a moving window. This will allow users to visualize variation in regression (maps of R-square and regression coefficients) and variogram models (maps of variogram parameters). Note that the regression-kriging with moving window would need to be fully automated, which might not be an easy task considering the computational complexity. Also, unlike OK with a moving window (Walter et al., 2001), regression-kriging has much higher requirements considering the minimum number of observations (at least 10 per predictor, at least 50 to model variogram). In general, our impression is that many of the procedures (regression and variogram modeling) in regression-kriging can be automated and amount of data modeling definitions expanded (local or global modeling, transformations, selection of predictors, type of GLMs etc.), as long as the point data set is large and of high quality. Ideally, users should be able to easily test various combinations of input parameters and then (in real-time) select the one that produces the most satisfactory predictions.

Open-source packages open the door to analyzes of unlimited sophistication. However, they were not designed with a graphical user interfaces (GUI's), or wizards typical for proprietary GIS packages. Because of this, they are not easily used by non-experts. There is thus opportunity both for proprietary GIS to incorporate regression-kriging ideas and for open-source software to become more user-friendly.

3.4.4 Towards a system for automated mapping

Geostatistics provides a set of mathematical tools that have been used now over 50 years to generate maps from point observations and to model the associated uncertainty. It has proven to be an effective tool for a large number of applications ranging from mining and soil and vegetation mapping to environmental monitoring and climatic modeling. Several years ago, geostatistical analysis was considered to be impossible without the intervention of a spatial analyst, who would manually fit variograms, decide on the support size and elaborate on selection of the interpolation technique. Today, the heart of a mapping project can be the computer program that implements proven and widely accepted (geo)statistical prediction methods. This leads to a principle of **automated mapping** where the analyst focuses his work only on preparing the inputs and supervising the data processing[85]. This way, the time and resources required to go from field data to the final GIS product (geoinformation) are used more efficiently.

Automated mapping is still utopia for many mapping agencies. At the moment, environmental monitoring groups worldwide tend to run analyzes separately, often with incorrect techniques, frequently without making the right conclusions, and almost always without considering data and/or results of adjacent mapping groups. On one side, the amount of field and remotely sensed data in the world is rapidly increasing (see section 4); on the other side, we are not able to provide reliable information to decision makers in near real-time. It is increasingly necessary that we automate the production of maps (and models) that depict environmental information. In addition, there is an increasing need to bring international groups together and start "*piecing together a global jigsaw puzzle*"[86] to enable production of a global harmonized GIS of all environmental resources. All this proves that automated mapping is an emerging research field and will receive significant attention in geography and Earth sciences in general (Pebesma et al., 2009).

A group of collaborators, including the author of this book, have begun preliminary work to design, develop, and test a web-based automated mapping system called **auto-map.org**. This web-portal should allow the users to upload their point data and then: (a) produce the best linear predictions depending of the nature/-type of a target variable, (b) interpret the result of analysis through an intelligent report generation system, (c) allow interactive exploration of the uncertainty, and (d) suggest collection of additional samples — all at click of button. The analysis should be possible via a web-interface and through e.g. Google Earth plugin, so that various users can access outputs from various mapping projects. All outputs will be coded using the HTML and Google Earth (KML) language. Registered users will be able to update the existing inputs and re-run analysis or assess the quality of maps (Fig. 3.15). A protocol to convert and synchronize environmental variables coming from various countries/themes will need to be developed in parallel (based on GML/GeoSciML[87]).

[85] See for example outputs of the INTAMAP project; `http://www.intamap.org`.

[86] Ian Jackson of the British Geological Survey; see also the `http://www.onegeology.org` project.

[87] `http://www.cgi-iugs.org`

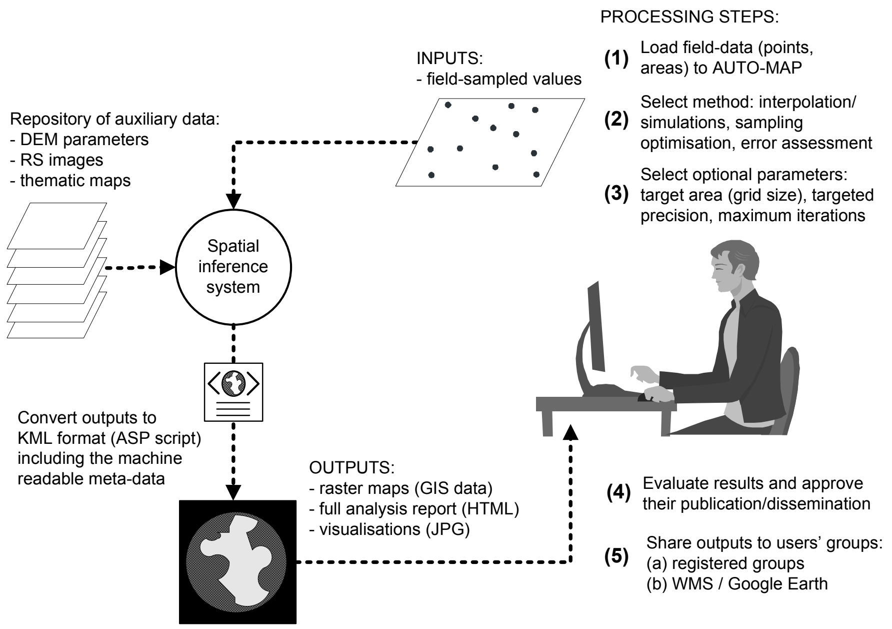

PROCESSING STEPS:
(1) Load field-data (points, areas) to AUTO-MAP
(2) Select method: interpolation/ simulations, sampling optimisation, error assessment
(3) Select optional parameters: target area (grid size), targeted precision, maximum iterations
INPUTS:
- field-sampled values
Repository of auxiliary data:
- DEM parameters
- RS images
- thematic maps
Spatial inference system
Convert outputs to KML format (ASP script) including the machine readable meta-data
OUTPUTS:
- raster maps (GIS data)
- full analysis report (HTML)
- visualisations (JPG)
(4) Evaluate results and approve their publication/dissemination
(5) Share outputs to users' groups:
(a) registered groups
(b) WMS / Google Earth

Fig. 3.15: A proposal for the flow of procedures in auto-map.org: a web-based system for automated predictive mapping using geostatistics. The initial fitting of the models should be completely automated; the user then evaluates the results and makes eventual revisions.

There would be many benefits of having a robust, near-realtime automated mapping tool with a friendly web-interface. Here are some important ones:

- the time spent on data-processing would be seriously reduced; the spatial predictions would be available in near real time;
- through a browsable GIS, such as Google Earth, various thematic groups can learn how to exchange their data and jointly organize sampling and interpolation;
- the cost-effectiveness of the mapping would increase:
  - budget of the new survey projects can be reduced by optimising the sampling designs;
  - a lower amount of samples is needed to achieve equally good predictions;

It is logical to assume that software for automated mapping will need to be *intelligent*. It will not only be able to detect anomalies, but also to communicate this information to users, autonomously make choices on whether to mask out parts of the data sets, use different weights to fit the models or run comparison for various alternatives. This also means that development of such a system will not be possible without a collaboration between geostatisticians, computer scientists and environmental engineers.

Many geostatisticians believe that map production should never be based on a *black-box* system. The author of this guide agrees with these views. Although data processing automation would be beneficial to all, analysts should at any time have the control to adjust the automated mapping system if needed. To do this, they should have full insight into algorithms used and be able to explore input data sets at any moment.

Further reading:

★ Bolstad, P., 2008. **GIS Fundamentals**, 3rd edition. Atlas books, Minnesota, 620 p.

★ Burns, P. 2009. **The R Inferno.** Burns statistics, London, 103 p.

★ Conrad, O. 2007. SAGA — program structure and current state of implementation. In: Böhner, J., Raymond, K., Strobl, J., (eds.) **SAGA — Analysis and modeling applications**, Göttinger Geographische abhandlungen, Göttingen, pp. 39-52.

★ Rossiter, D.G., 2007. **Introduction to the R Project for Statistical Computing for use at ITC**. International Institute for Geo-information Science & Earth Observation (ITC), Enschede, Netherlands, 136 p.

★ Venables, W. N. and Ripley, B. D., 2002. **Modern applied statistics with S**. Statistics and computing. Springer, New York, 481 p.

★ Zuur, A. F., Ieno, E. N., Meesters, E. H. W. G., 2009. **A Beginner's Guide to R**. Springer, Use R series, 228 p.

★ `http://www.52north.org` — 52° North initiative responsible for the distribution of ILWIS GIS.

★ `http://www.saga-gis.org` — homepage of the SAGA GIS project.

★ `http://cran.r-project.org/web/views/Spatial.html` — CRAN Task View: Analysis of Spatial Data maintained by Roger Bivand.