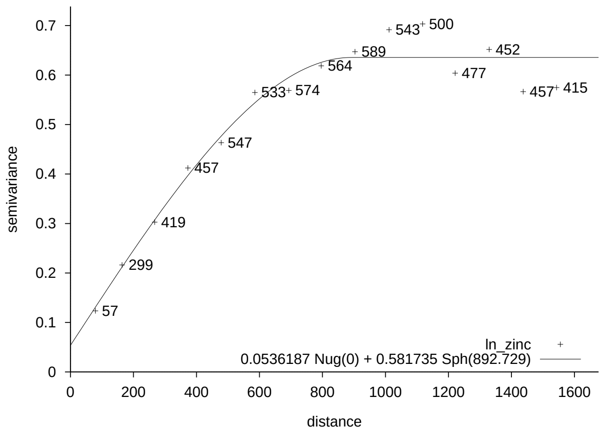
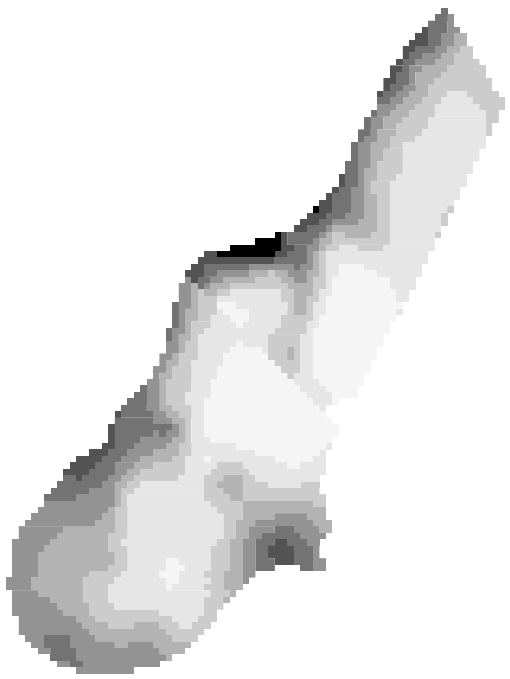
gstat user's manual





Edzer J. Pebesma
Dept. of Physical Geography, Utrecht University
P.O. Box 80.115, 3508 TC, Utrecht, The Netherlands

Relevant Internet locations:
Gstat: `http://www.geog.uu.nl/gstat/`
Meschach: `ftp://ftpmaths.anu.edu.au/pub/meschach/meschach.html`
Gnuplot 3.7: `ftp://cmpc1.phys.soton.ac.uk/`

Contents

Chapter 1

Introduction

Gstat is a program for the modelling, prediction and simulation of geostatistical data in one, two or three dimensions. Geostatistical data are data (measurements) collected at known locations in space, from a function (process) that has a value at every location in a certain (1, 2 or 3-D) domain. These data (or some transform of them) are modelled as the sum of a constant or varying trend and a spatially correlated residual. Given a model for the trend, and under some stationarity assumptions, geostatistical modelling involves the estimation of the spatial correlation. Geostatistical prediction ('kriging') is finding the best linear unbiased prediction (the expected value) with its prediction error for a variable at a location, given observations and a model for their spatial variation. Simulation of a spatial variable is the creation of randomly drawn realizations of a field given a model for the data, possibly conditioned on observations.

In gstat, geostatistical modelling comprises calculation of sample variograms and cross variograms (or covariograms) and fitting models to them. Sample (co-) variograms are calculated from ordinary, weighted or generalised least squares residuals. Nested models are fitted to sample (co-) variograms using weighted least squares, and during a fit each single parameter can be fixed. Restricted maximum likelyhood estimation of partial sills is also implemented. In the interactive variogram modelling user interface of gstat, variograms are plotted using the plotting program gnuplot.

Gstat provides prediction and estimation using a model that is the sum of a trend modelled as a linear function of polynomials of the coordinates or of user-defined base functions, and an independent or dependent, geostatistically modelled residual. This allows simple, ordinary and universal kriging, simple, ordinary and universal cokriging, standardised cokriging, kriging with external drift, block kriging and "kriging the trend", as well as uncorrelated, ordinary or weighted least squares regression prediction. Simulation in gstat comprises uni- or multivariable conditional or unconditional multi-Gaussian sequential simulation of point values or block averages, or (multi-) indicator sequential simulation.

Besides many of the common options found in other geostatistical software packages, gstat offers the unique combination of

- an interactive user interface for modelling variograms and generalised covariances (residual variograms), that uses the device-independent plotting program gnuplot for graphical display
- support for several ascii and binary data and grid map file formats for input and output
- a concise, intuitive and flexible command language
- user customization of program defaults
- no built-in limits
- free, portable ansi-c source code

The theory of geostatistics is not explained in this manual. Good texts on the subject are e.g. Journel and Huijbregts (1978) and Cressie (1993). The practice of geostatistical computation is explained only very briefly. Texts about practical and computational aspects are e.g. Isaaks and Strivastava (1989) and Deutsch and Journel (1992). This manual explains how things are done with gstat.

Chapter 2 explains the concepts behind gstat and its basic methods and prediction or simulation modes. Chapter 3 treats the simple, multiple, multivariable and stratified modes, and change of support (block kriging). Chapter 4 is a complete reference of the command file syntax. Chapter 5 explains how the program can be further controlled, for instance by using start-up files, command line options or environment variables. Finally, Chapter 6 lists a number of example command files that demonstrate most of the capabilities of gstat (these files are part of the program distribution). Note that the gstat home page provides the html versin of this manual, with command files fully hyperlinked to all input and output results, at

`http://www.geog.uu.nl/gstat/manual`

The appendices contain more technical details: equations for modelling and prediction (A) and error messages and help information (C). Suggestions for improvement of gstat or this manual are welcome—send them to

`gstat-info@geog.uu.nl`

Further reading

In *Computers and Geosciences* a paper written on gstat appeared Pebesma and Wesseling (1998). Beyond much of the information present in this manual, it discusses

- implementation and efficiency issues in gstat (computational and algorithmic aspects in geostatistics)
- managing large geostatistical projects
- technical issues (portability, numerical precision, implementation of the command file parser and interactive user interface, program limits)
- a comparison with GSLIB Deutsch and Journel (1992) and GeoEAS.

gstat-announce mailing list

A mailing lists for announces (version releases etc.) regarding gstat, exists:
`gstat-announce@geog.uu.nl`
or visit the gstat home page, `http://www.geog.uu.nl/gstat/`.

Chapter 2

Getting started

2.1 Invoking gstat

Gstat is started either by typing

`gstat` *command_file*

where *command_file* is the name of a file with gstat commands, or by typing

```
gstat -i
```

In the latter case, the interactive variogram modelling user interface is started. This interface can be used to specify data, calculate and plot sample variograms, fit variogram models and create variogram plot files. Within the interface, help is obtained by pressing 'H' or '?', and the program stops after pressing 'q' or 'x'. At the end of a variogram modelling session the program settings concerning data and (fitted) variogram models can be written to a gstat command file by pressing 'c'. Such a command file will look like:

```
#
# gstat command file
#
data(zinc): 'zinc.eas', x=1, y=2, v=3;
variogram(zinc): 0.0716914 Nug(0) + 0.563708 Sph(917.803);
```

The first three lines that start with a `#` are comment lines, the following lines are gstat commands. The first command specifies where data were read (the variable `zinc` was read from the file `zinc.eas`, having measured values on column 3, x-coordinates on column 1 and y-coordinates on column 2), the second command defines a variogram model for the variable `zinc` (the sum of a nugget and a spherical model). Suppose that these commands are held in a file called `zinc.gst`, then starting gstat with the command

```
gstat zinc.gst
```

will start the variogram modelling user interface after reading data and variogram model, and work can be continued at the point where it was saved before.

At a later stage, the data and variogram definitions in the command file can be used for kriging. This is accomplished by adding commands to the command file that specify where the kriging predictions should be made, and where the results should be written to. Consider the command file `zinc_pr.gst`:

```
#
# gstat command file
#
data(zinc): 'zinc.eas', x=1, y=2, v=3;
variogram(zinc): 0.0717 Nug(0) + 0.564 Sph(917.8);
mask:        'mask.map';
predictions(zinc): 'zinc_pr.map';
variances(zinc):  'zinc_var.map';
```

It says that prediction locations are the grid cell centres of the (non-missing valued) cells in the grid map `mask.map`, that ordinary kriging predictions should be written to the grid map `zinc_pr.map`, and ordinary kriging prediction variances to `zinc_var.map`, after starting gstat as:

```
gstat zinc_pr.gst
```

2.2 Data formats

Measurement data (measured values, their spatial coordinates, and optionally base function values) are read from ascii table or (simplified) GeoEAS table files, from grid map files (see Appendix D) or from Idrisi point (.vec) files. Table (or column) files are ascii files without a header, with each line holding one record, and fields on a line separated by blanks, commas or tabs. The simplified GeoEAS format is a table file with a header. The GeoEAS file header has text information on the first line; m, the number of variables in the file on the second line, and m variable names on lines 3 to $m + 2$.

Grid map files may have one of the following formats: Arc-Info grid (gridfloat or gridascii), Idrisi image (ascii, real binary, byte), PCRaster (all formats, Van Deursen and Wesseling (1992)), ER-Mapper, GMT, Surfer ascii. Output grid maps are written in the same format as the input (mask) map. Files in formats that use two files and assume fixed extensions (idrisi, gridfloat) should be denoted by the file base name only (omit the extension for gstat). A grid map conversion is built in gstat, see section 5.5 Technical information on data and grid map formats supported is found in appendix D. Gstat has a built-in map format conversion utility (convert, Section 5.5).

2.3 Default program action

Based on the information read in the command file, gstat decides what to do: variogram modelling, prediction or simulation, and, in case of prediction, which prediction method to use. The decision tree for the default program action is shown in Fig. 2.1.

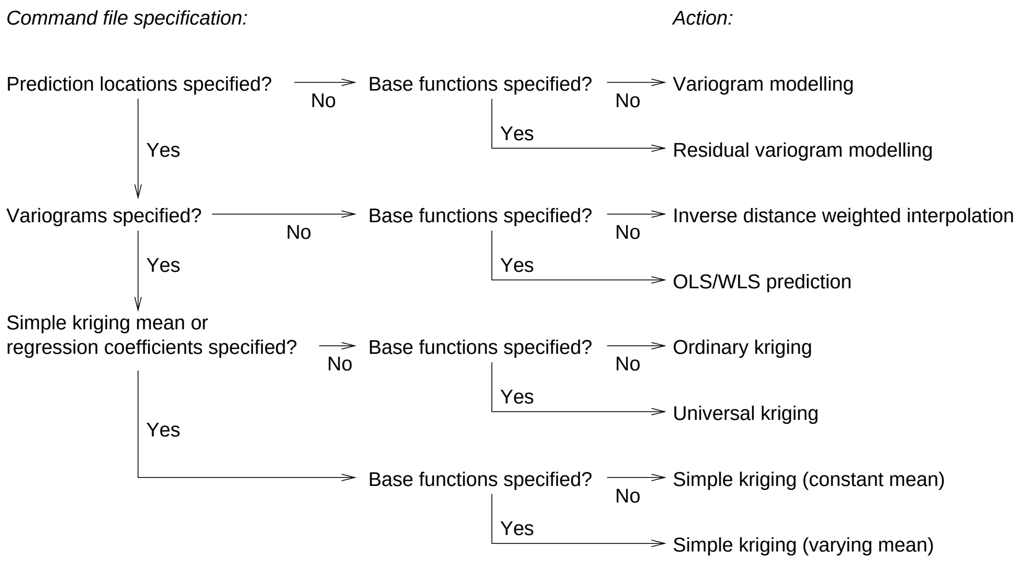

Command file specification:
Action:
Prediction locations specified?
No
Base functions specified?
No
Variogram modelling
Yes
Yes
Residual variogram modelling
Variograms specified?
No
Base functions specified?
No
Inverse distance weighted interpolation
Yes
Yes
OLS/WLS prediction
Simple kriging mean or regression coefficients specified?
No
Base functions specified?
No
Ordinary kriging
Yes
Universal kriging
Yes
Base functions specified?
No
Simple kriging (constant mean)
Yes
Simple kriging (varying mean)

Figure 2.1: Decision tree for default program action

When observations, variogram, and prediction locations are specified, the default action is ordinary kriging on mask map locations. When, in the final example in section 2.1, the line

`variogram(zinc):...;`

is left out, inverse distance weighted interpolation is the default action (kriging demands specification of the variogram). When the command

`mask: ... ;`

is left out, the default action is variogram modelling because prediction demands specification of prediction locations.

Sometimes it is necessary to override the default program action (e.g. to calculate variograms non-interactively, or to do simulation instead of prediction). Information about overriding the default program action is found in sections 2.4, 2.6 and 4.5.

2.4 Modelling spatial dependence

When no prediction locations are defined in the command file, gstat starts the interactive variogram modelling user interface (example [6.1], example [6.2]). Multiple variables are analyzed when they are specified with `data(`*id*`)` commands, each having a unique *id*. From this interface sample variograms, covariograms, cross variograms and cross covariograms can be calculated, viewed, and modelled (see Appendix A.1); variogram plots can be saved (e.g. as PostScript file, Fig. 2.2) and printed; and modified settings of data and fitted variograms can be saved as a gstat command file. The interface has several selection items and single-key options. Summary help is obtained by pressing 'H' (shift-h).

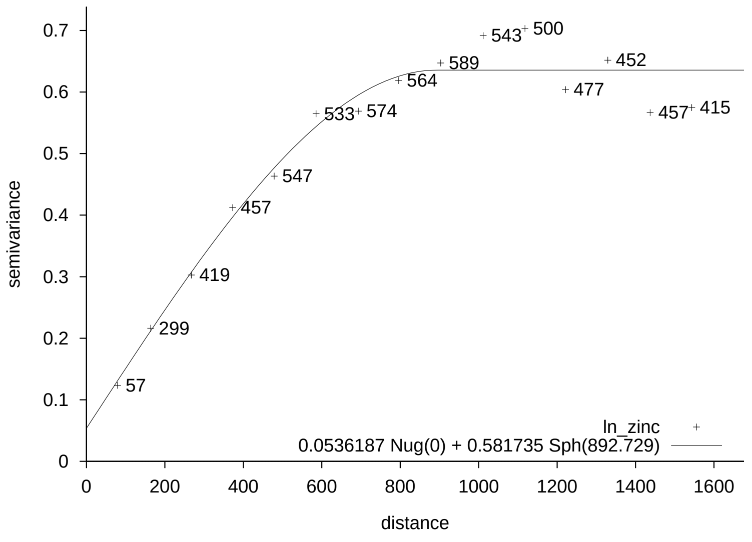


Figure 2.2: Variogram plot from gnuplot

Help on a specific user interface item is obtained by selecting the item with the cursor keys and pressing '?'. What follows is a brief description of the visible items in the user interface:

`enter/modify data` enter a new variable or modify (reload) a variable (allows only a few input options)

`choose variable` choose a variable, or, if the next field is on a cross (co-) variogram, a pair of variables

`calculate what` choose what to calculate (variogram, cross variogram, covariogram or cross covariogram) `cutoff, width` prompts for the cutoff (the maximum distance at which pairs of data points will be considered for inclusion in sample variogram estimates) and the lag width (the step size of distance intervals for sample variogram estimates). Non-even interval boundaries can be obtained by specifying `bounds` in the command file (section 4.1)

`direction` enter directional parameters (direction angle and direction tolerance: the maximum deviation from this direction tolerated for a pair of data points to be included in the sample variogram estimate)

`variogram model` enter a variogram model or change variogram model parameters (section 4.3)

`fit method` choose a variogram fit method (or no fit)

`show plot` show variogram and model (if present, and after optional fitting)

Variogram models can be fitted to the sample variogram using iterative reweighted least squares estimation Cressie (1985), or can be fitted directly to the sample data using REML estimation Kitanidis (1985). Appendix A.1 gives details on the calculation of sample (co-) variograms and model fitting. Non-linear least squares fitting is only guaranteed to work when good initial values are provided. Therefore, and more in general, visual examination of model fit is recommended.

Variogram plots can be saved as encapsulated PostScript file (Fig. 2.2) from gnuplot by pressing 'P' or as gif file by pressing 'G' (gif only when the gd library was linked to gnuplot). Plots can be customised (e.g. labels, legend, title) by first saving sample variogram estimates to a file ('e'), then saving the gnuplot commands to a file ('g'), then modifying this file and finally using gnuplot to create the PostScript (or other graphics) file.

By default, direct and cross variograms and covariograms are calculated from ordinary least squares residuals by using a linear model (as default only an intercept, section 2.7). Generalised least squares residuals are used when the command

```
set gls=1;
```

is added to the command file (sections 2.7, 4.4).

Non-interactive variogram modelling

Variograms can also be calculated non-interactively, by adding the command

```
method: semivariogram;
```

or

```
method: covariogram;
```

to the command file (section 4.5).

Sample variograms can be saved to a file, using for instance:

```
variogram(zinc): 'zinc.est';
```

For large data sets, it may be best to calculate sample variograms non-interactively and do the modelling afterwards. This is accomplished by first saving the sample variograms to file as described above, and then to load only the sample variograms in the user interface (not the data), which is done by defining dummy data:

```
data(zinc); # dummy data
```

and a valid sample variogram, as

```
variogram(zinc): 'zinc.est';
```

or, when a variogram model should be defined ahead of fitting:

```
variogram(zinc): 'zinc.est', 1 Nug() + 1 Sph(800);
```

Variogram maps

If in addition to one of these `method` commands a mask map is specified, then gstat calculates the variogram map Isaaks and Strivastava (1989) for the field specified by the mask, and writes this map to the output map `zincv.map`

```
variogram(zinc): 'zinc.map';
```

optionally, in addition the corresponding number of data pairs can be written to the output map `zincn.map` when specified as

```
variogram(zinc): 'zince.map', 'zincn.map';
```

(typically a variogram map is centred around (0,0) and has map dimension and cell size similar to cutoff and interval width values).

2.5 Prediction

If prediction locations are defined in the command file, gstat chooses a prediction method depending on the model defined by the complete set of commands in a command file.

When no variograms are specified, inverse distance weighted interpolation is the default action (Fig. 2.1, example [6.3]).

When variograms are specified the default prediction method is ordinary kriging Journel and Huijbregts (1978); Cressie (1993) (example [6.4] and example [6.8]).

Simple kriging is the default action when in addition for each variable the simple kriging mean (`sk_mean` or `b`) is set (section 4.2; example [6.5], universal kriging or uncorrelated linear model prediction is used when a model for the trend is defined ,section 2.7). Multiple prediction, multivariable prediction, and stratified prediction are described in section 3.1-3.4. Prediction of block averages is described in section 3.5.

If the prediction locations are specified as a mask map with the command

```
mask: 'file';
```

then predictions and prediction variances are written to output maps only when these maps are specified explicitly (section 4.1; example [6.5]).

As an alternative to prediction on grid map locations, prediction on non-gridded locations is the default action when these locations are specified with the

```
data(): ... ;
```

command (note the absence of an identifier between the parentheses). In this case, output is written in ascii table or simplified GeoEAS format to the file defined by the command `set output='file';` (example [6.4], or defined with the command line option `-o`, section 5.2).

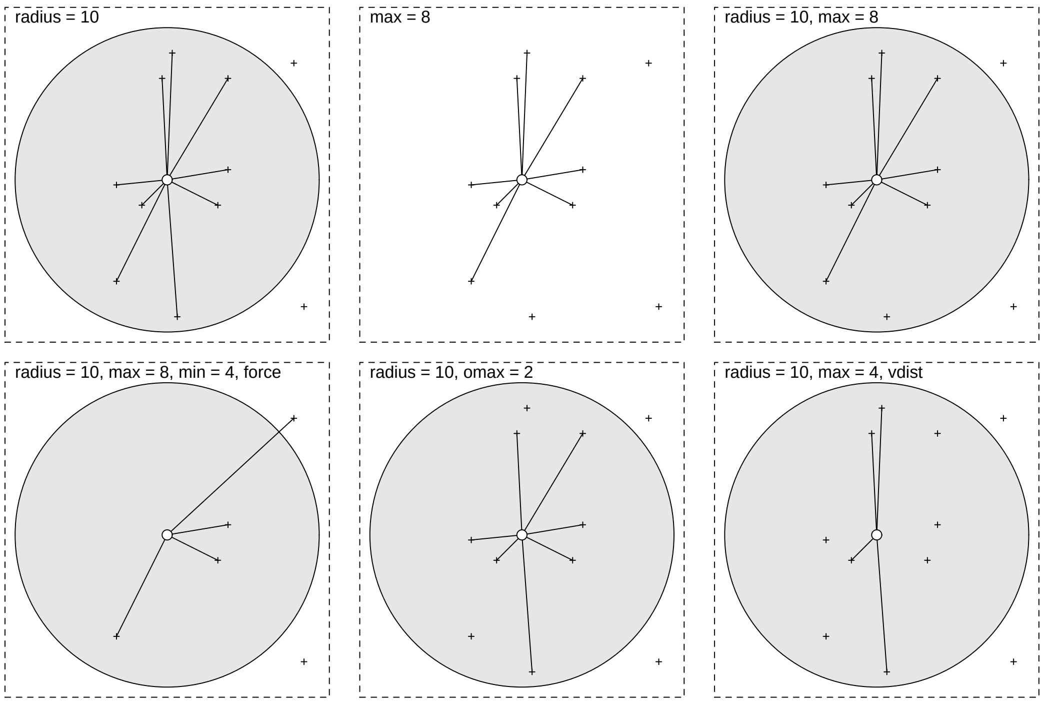

radius = 10
max = 8
radius = 10, max = 8
radius = 10, max = 8, min = 4, force
radius = 10, omax = 2
radius = 10, max = 4, vdist

Figure 2.3: Local neighbourhood selections. Lines indicate selected points (+). Lower right: variable with anisotropic variogram having a strong north-south correlation

Local Neighbourhoods

By default, gstat uses global prediction, meaning that for each prediction all data values are used. However, it is often desirable to use not all data values, but only a subset in a (spatial) neighbourhood around the prediction (simulation) location, for either computational reasons or the wish to assume first-order stationarity only locally. Gstat allows local neighbourhood selections to be based on distance (`radius`), number of data points (`max`, `min`), variogram distance (`vdist`), and number of data points per octant (3D) or quadrant (2D) (`omax`). The options are explained below (see also Fig. 2.3, section 4.2 examples in chapter 6).

The quadtree-based algorithm used to obtain data points in a local search neighbourhood is described in Hjaltason and Samet (1995), and is found at `http://www.cs.umd.edu/ brabec/quadtree/index.html` (Bucket PR Quadtree demo).

`radius = 10` select all data points within 10 (euclidian) distance units from the prediction location

`max = 8` select the 8 data points that are closest (in euclidian distance) to the prediction location (or take all data points if less than 8 are available)

Some options should be combined, and permitted combinations are explained below. (Combinations not mentioned might result in unexpected or undesired results.)

`radius = 10, max = 8` after selecting all data points at (euclidian) distances from the prediction location less or equal to 10, choose the 8 closest when more than 8 are found

`radius = 10, max = 8, min = 4` in addition to the previous selection, generate a missing value if less than 4 points are found within the search radius 10

`radius = 10, max = 8, min = 4, force` in addition to the previous selection, if less than 4 data points are found in the search radius, instead of generating a missing value, select (force) the 4 nearest (in euclidian distance) data points, regardless their distance

`radius = 10, omax = 2` after selecting all data points at distances less or equal to 10, choose the 2 closest data points in each octant (3D), quadrant (2D) or secant (1D) `radius = 10, vdist, ...` after the radius selection, decide what the *nearest* data points are on the base of point-to-point semivariance of the data variable instead of euclidian distances ("semivariance distance": in case of anisotropy this allows the prevalence of more correlated points over the, in the euclidian sense, nearest points)

Indicator kriging
Basically, indicator kriging is equivalent to simple or ordinary kriging of indicator-transformed data. However, resulting estimates of indicator values are not guaranteed to satisfy order relations. During indicator kriging, gstat will do order relation violation correction for independent, cumulative or categorical (disjunct) indicators only if the `order` is to one of the values in Table 2.1 in section 2.6, `order` in section 4.4 and Deutsch and Journel (1992); Goovaerts (1997).

2.6 Simulation

Simulation Davis (1987); Gómez-Hernández and Journel (1993); Myers (1989) is done by setting up a command file for simple kriging (section 2.5) and changing the default action to Gaussian simulation by adding the command

```
method: gs;
```

(example [6.6] and example [6.7]), or to indicator simulation by adding the command

```
method: is;
```

If valid data are present (i.e., data are available in the neighbourhoods defined), conditional simulation is done. Unconditional simulation is done when only dummy variables (`dummy`, section 4.2) or data outside every possible neighbourhood are defined.

The sequential simulation algorithm Gómez-Hernández and Journel (1993) is used for the simulation. This algorithm visits each simulation location, following a random path. After simulating a value (or set of values in the multivariable case) at the location, it is added to the conditioning data.

A few notes on the practice of (indicator or Gaussian) simulation with gstat are:

- Even for simulating small fields (e.g. 500 cells) it is strongly recommended to limit the kriging neighbourhood in order to get local kriging (section 2.5). Because every simulated point or block is added to the data, 'global' simulation would soon amount to large kriging systems, thus slowing down the simulation quickly.

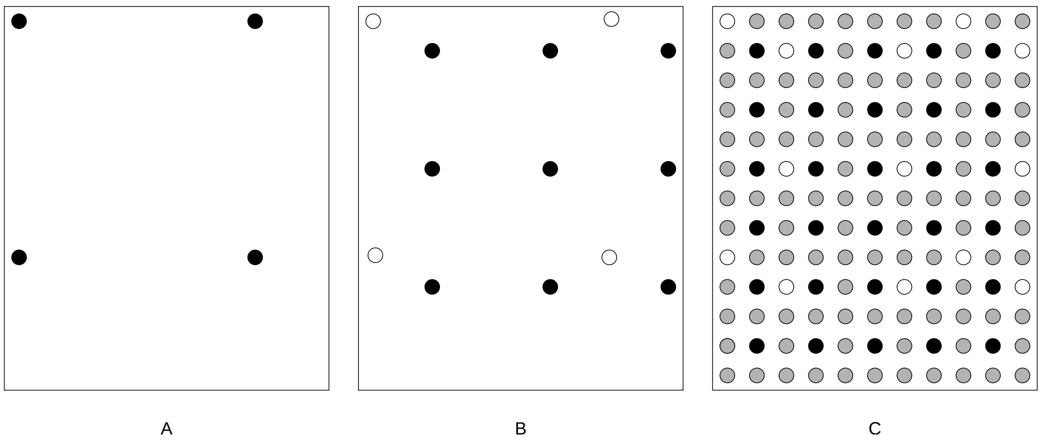


Figure 2.4: Recursively refining visiting sequence during simulation

- Gstat can create many simulations simultaneously in an efficient way: a single random path is followed and for each simulation location, the neighbourhood selection and the solution to the kriging system are reused for all subsequent simulations. When many simulations are required (e.g. for a Monte Carlo study) the time saving will be significant (see `set nsim`). (Note that the use of local approximations (local kriging) results in slightly dependent realizations when obtained by following a single random path.)

- When simulation is done on a regular grid (using a mask map), in order to reproduce the statistical properties up to reasonably large distances, a recursively refining random path ("multiple-steps simulation", Gómez-Hernández and Journel (1993)) is followed, shown in Fig. 2.4.

  A random path is started on a (randomly located) coarse grid (A: the coarsest (2 $\times$ 2) grid with a grid spacing that is a power of 2). The simulation grid is refined recursively: 4 $\times$ 4 (B), 8 $\times$ 8 (C, black dots) halving the grid spacing each step) until all grid locations are visited (C, grey dots). Neighbourhood definitions (see section 4.2, and also the `force` flag) may ensure that necessary conditioning data are used for reproducing the statistical properties sufficiently.

- Simple kriging results in a correct conditional distribution, and in Gaussian simulations that honour the specified variogram. Universal or ordinary kriging can be used if enough conditioning data are available, and leads to 'rougher' simulations, less honouring the variogram but better adjusted to a non-stationary mean ("major heterogeneities",

Indicator	order violation	correction	`set order`
independent	$\hat{p}_i < 0$	$\tilde{p}_i = 0$	`1-4`
independent	$\hat{p}_i > 1$	$\tilde{p}_i = 1$	`1-4`
categorical, open	$\sum_{i=1}^{n} \hat{p}_i > 1$	$\tilde{p}_i = \hat{p}_i / \sum_{i=1}^{n} \hat{p}_i$	`2`
categorical, closed	$\sum_{i=1}^{n} \hat{p}_i \neq 1$	$\tilde{p}_i = \hat{p}_i / \sum_{i=1}^{n} \hat{p}_i$	`3`
cumulative	$\hat{p}_i < \hat{p}_{i-1}$	$\tilde{p}_i - \tilde{p}_{i-1} = 0$	`4`

Table 2.1: Order relation corrections

Deutsch and Journel (1992)), when present in the conditioning data. The parameter `nXuk` (section 4.4) controls the choice between simple or universal (ordinary) kriging. Another way to obtain non-stationary simulations is to add a varying mean to a stationary (simple kriging) simulation afterwards.

Gaussian block simulation

Gstat simulates block averages when a non-zero block size is specified (section 3.5). The implementation of this is a rather inefficient one. Simulation will be faster when `nblockdiscr` is set to a low value (to 3 or 2, section 4.4), at the expense of the accuracy of point-to-block and block-to-block covariance calculations (see Appendix A.3).

Indicator simulation

From data definitions alone, gstat cannot decide whether it is working with indicator variables or not. In case of prediction this is not crucial—procedure-wise, indicator kriging is identical to simple or ordinary kriging. When indicator simulation is done for multiple variables, a number of different situations may occur, and for correct results, it should be specified explicitly if the set of indicator variables is (i) independent, (ii) cumulative or (iii) disjunct:

- if the set is *independent*, simulated indicator variable can take a value 1 or 0 independently from the other indicator variables
- if the ordered set of indicator variables $I_0(s), ..., I_{n-1}(s)$ is *cumulative*, then $I_j(s) = 1$ *implies* that $I_0(s), ...I_{j-1}(s)$ are all 1 (in gstat, the order of a set of indicator variables equals the order in which the variables appear in the command file)
- if a set of indicators is *disjunct*, then $I_j(s) = 1$ implies that $I_i(s) = 0$ for all $i \neq j$.

Independent indicators may represent independent variables. A set of cumulative indicators may represent the cumulative distribution function of a single continuous variable and a set of disjunct indicators can represent the categories of a categorical variable (see also the data command options and `Category`). Table 2.1 shows the corrections done for the different types of indicator variables and the value `order` should be set to to obtain the corrections (estimated probality $\hat{p}_i$, corrected estimate $\tilde{p}_i$). Cumulative indicators are corrected using the "upward-downward" approach, see Deutsch and Journel (1992, p. 80) or Goovaerts (1997, p. 324).

For multiple indicator simulation (no indicator cross variograms are specified), by default independent indicator simulation is done. The subsequent indicator variables are taken as cumulative indicators if the command

```
set order=4;
```

is added to the command file (Table 2.1). They will be treated as disjunct if `order` is set to 2 or 3 (see section 4.4).

2.7 Linear models in gstat

Defining a model

In ordinary and simple kriging each observation $z(s_i)$ (the value of variable z at location s_i) is represented by the model

$$Z(s_i) = m + e(s_i) \tag{2.1}$$

with, in case of ordinary kriging $Z(s)$ intrinsically stationary and m an unknown (locally) constant trend, or, in case of simple kriging, $Z(s)$ a second order stationary and m a known, constant trend.

A wider class of models is obtained when the observation $z(s_i)$ is modelled as the sum of a spatially non-constant (i.e. non-stationary) trend $m(s_i)$ and an intrinsically stationary error $e(s_i)$:

$$Z(s_i) = m(s_i) + e(s_i) \tag{2.2}$$

In the universal kriging model such a trend is modelled as a linear function in p known base functions $f_j(s) = (f_j(s_1), ..., f_j(s_n))'$ and p unknown constants β_i, which yields, for the observation at s_i

$$Z(s_i) = \sum_{j=1}^{p} f_j(s_i)\beta_j + e(s_i)$$

and for all observations

$$Z(s) = \sum_{j=1}^{p} f_j(s)\beta_j + e(s)$$

which can be written in matrix notation as

$$Z(s) = F\beta + e(s) \tag{2.3}$$

with $F = (f_1(s), ..., f_p(s))$ and $\beta = (\beta_1, ..., \beta_p)'$. Ordinary kriging (2.1) is the special case where this model has only an intercept ($p = 1$, $f_1(s) = 1, \forall x$ and $\beta_1 = m$).

Gstat calculates prediction under the multivariable universal kriging model Ver Hoef and Cressie (1993) when base functions $f_i(s)$ and variogram(s) for $e(s)$ are specified (see appendix A.2 for the prediction equations). An intercept (the constant value as in the ordinary kriging model) in (2.1) is assumed for each variable by default, and only non-intercept base functions need to be specified. Base functions can be polynomials of the coordinates (e.g. x, x^2, xy etc.) or user-defined.

Coordinate polynomial base functions

For a data variable in two dimensions, a first order linear trend in the coordinates is defined by

```
   data(x):  'file', x=1, y=2, v=3, X=x&y;
```

or, as a shorthand for this, the coordinate polynomial trend order degree can be specified:

```
   data(x):  'file', x=1, y=2, v=3, d=1;
```

(Note that `d=1` is equivalent to `X=x` for one-dimensional, `X=x&y` to for two-dimensional and to `X=x&y&z` for three-dimensional data.) Values of coordinate polynomial base functions at observation and prediction locations are obtained from the (standardised) location coordinates s_i (see also example [6.18]).

User-defined base functions

Non-coordinate polynomial, user-defined functions can also be specified as base functions. Because they are not known, they should be defined as column numbers in a data file (example [6.13]), like

```
   data(x):  'file', x=1, y=2, v=3, X=4&5;
```

User-defined and coordinate polynomial base functions may be intermixed.

When binary (e.g., 0/1) variables are used as base functions, and the sum of these functions coincides with an intercept (i.e., summed row-wise, the columns equal a column with a constant), the default intercept has to be overridden. This is done by specifying `-1` as the first column number of the base functions (example [6.14]).

Specification of the user-defined base function values at prediction locations is necessary, since they are needed in the prediction. For the prediction locations they are needed too, and for map prediction locations they are defined as a list of mask maps containing the base functions. For the `data()` prediction locations they are defined as the `X` column numbers in the corresponding file. In both cases the number of base functions thus specified and the order in which they appear should match the order in which the (non-intercept and non-coordinate polynomial) `X` columns appear in subsequent `data(`*id*`)` commands.

If more than one variable is defined and only direct variograms are specified, multiple universal kriging is done. If in addition to direct variograms cross variograms are specified, multivariable universal kriging is done Ver Hoef and Cressie (1993) (universal cokriging, section 3.3).

Ordinary and weighted least squares trend prediction

If base functions are specified but no variograms are specified, the default prediction method is the (multiple) regression prediction, (ordinary least squares, OLS) assuming that the $e(s)$ are independently identically distributed (IID). In this case the prediction variance is the classical regression prediction variance for a single observation (example [6.16] and example [6.18]), or for the mean value when the block size is non-zero (section 3.5).

If the errors are assumed to be independent with different variances, $v_i\sigma^2$ then specifying the constants v_i will result in weighted least squares (WLS) prediction. The values v_i are not variances but merely relate an individual residuals variance to σ^2. For instance, if an observation is an average of n_i measurements, then, assuming the variance of individual measurements is constant, v_i can be set to n_i^{-1}. In the unweighted case, all v_i are 1, and for prediction variances to make sense, the v_i should be related to this unity value (σ^2 should be "the" residual variance).

Generalised least squares trend prediction

If at prediction locations s_0, for some reason not the kriging prediction but the generalised least squares estimate (or BLUE, best linear unbiased estimate) of the trend $f(s_0)\hat{\beta}$ and its estimation variance are needed, then this is obtained by overriding the default method (ordinary or universal kriging) using

```
    method: trend;
```

Setting $f_i(s_0)$ to 1 and all other $f(s_0)$ to 0 yields the generalised least squares estimate (BLUE) of $\hat{\beta}_i$. See appendix A.2 for details on weighted or combined weighted and generalised least squares prediction.

Residual variograms

If base functions are specified but no prediction locations are specified, then the sample direct or cross variogram and covariogram is calculated from ordinary least squares (OLS) residuals, as obtained from the linear model with IID errors. If generalised least squares residuals are preferred to OLS residuals, the (initial) variograms should be set, and `gls` should be set to 1 (section 4.4).

Chapter 3

Prediction modes and change of support

3.1 Modes

When a command file holds more than one variable, each specified with a unique ***id***, then different prediction (or simulation) *modes* are possible, depending on the complete model specification: predictions can be made independently ('multiple' prediction), dependently ('multivariable' prediction), or variables may correspond to certain portions (categories) in the mask map or prediction location data ('stratified' prediction). The modes hold equally for prediction and simulation. When only one variable is defined, predictions (or simulations) are made for this variable at every prediction location ('simple' mode).

The decision tree gstat uses to determine the mode is shown in Fig. 3.1. The choice for a certain mode is always implicit, and is made after gstat read the command file and examined the data.

3.2 Multiple mode

When multiple variables are defined, and no variograms or only direct variograms are defined (no cross variograms), then *multiple* prediction (simulation) is the default action. See for instance example [6.10]. In the multiple mode, predictions or simulations are made for each variable independently. The advantage of using the multiple mode over using a command files for each variable, is that, besides being concise, if the variables have identical locations, each neighbourhood search is done only for the first variable (example [6.17]).

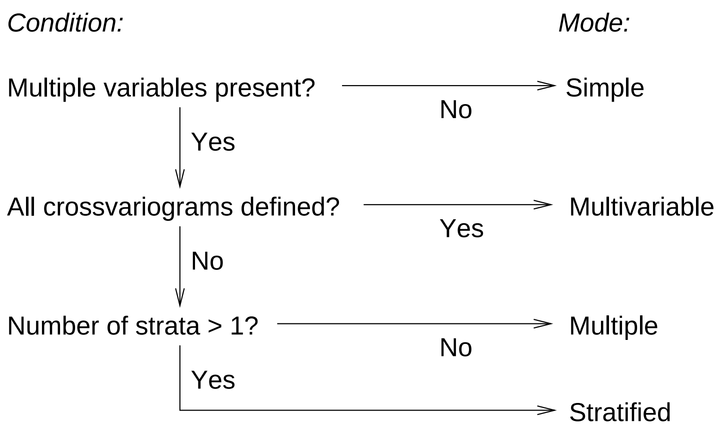


Figure 3.1: Prediction modes

3.3 Multivariable mode

If in addition to direct variograms the cross variograms are defined for all variable pairs, then the prediction mode becomes *multivariable* (i.e., cokriging or co-simulations). In case of multivariable prediction, prediction error covariances from multivariable prediction Ver Hoef and Cressie (1993) on a map can be specified per identifier pair with `covariances(`*`id1,id2`*`)` (example [6.11]). In gstat, multivariable prediction comprises simple cokriging, ordinary cokriging or universal cokriging (as well as standardised cokriging, multivariable indicator or Gaussian simulation).

When, for multiple variables a linear model is specified with independent errors (no variograms are defined), and one or more of the variables' regression parameters are defined as common parameters (with the command `merge`, section 4.1) then the prediction mode becomes multivariable as well (cf. analysis-of-covariance models refXXchristensen96).

3.4 Stratified mode

Stratified kriging or simulation (each variable with it's direct variogram applies to a specific area in the mask map) is the default action if the following conditions hold (example [6.12]):

- more than one data variable is defined
- only for the first data variable output maps (`predictions`, `variances`) are defined

- the (first) mask map defined has more than one category

Data variables are numbered in the order they appear in the command file, starting at 0. Let the minimum grid value of the (first) mask map be m, If at a specific grid cell the mask map has value j, then for that cell the stratified prediction map (and variance map) will have predictions for variable $j - m$ (rounded to the nearest integer). Thus, predictions at mask map cells having value m will get predictions for the first variable.

In case of universal kriging or least squares prediction with user-defined base functions, the maps with base function values should follow the category map: in the stratified mode the first mask map is the map with the categories.

3.5 Change of support (block prediction)

Average values for square, rectangular or arbitrarily shaped blocks can be predicted in gstat for all kriging variants, for OLS or WLS prediction or for inverse distance weighted interpolation, or they can be simulated using multi-Gaussian simulation.

The mean value of a block, the size of a grid cell, is obtained by adding the command

```
blocksize;
```

to the command file. Alternatively, block mean values for rectangular blocks with arbitrary size, centred at prediction locations are obtained when the block size is specified, in one dimension:

```
blocksize: dx = 1;
```

for a line element with length 1, in two dimensions

```
blocksize: dx = 1, dy = 2;
```

for a rectangular element with size 1×2 or in three dimensions

```
blocksize: dx = 1, dy = 2, dz = 3;
```

for a block with dimensions $1 \times 2 \times 3$, see example [6.4] and example [6.8].

Block averages are approximated by discretizing (“representing”) the block with a limited number of points Journel and Huijbregts (1978); Carr and Palmer (1993). Blocks with arbitrary shapes may be defined by specifying the points discretizing the block (see appendix A.3).

Chapter 4

Command file syntax

4.1 Commands

General conventions for gstat command files are:

- command files are ascii text files
- each command ends with a `;`
- command files start with one or more `data(`*`id`*`)` commands to define the data, where *`id`*, the identifier, is an unique one-word reminder for the variable defined
- regular file names are written between single or double quotes, as `'file.dat'` or `"file.dat"`, special file names include pipes, shell command output substitution or append-to files (section 5.4)
- white space (spaces, tabs, newlines) is ignored, except in file names
- comment is supported as follows: a `#` may appear anywhere in a line and gstat will ignore the rest of the line. It will not have this effect inside quoted strings (e.g. file names).

The commands that can appear in command files are listed below. Here, `id`, `id1` and `id2` refer to three distinct identifiers, and `file` is a valid file name. (Commands may be abbreviated to the first few characters that make them unique.)

4.1.1 General options

`data(id):` *body*`;` here, *body* defines the data to be read for variable `id` (section 4.2) `variogram(id):` *body*`;` here, *body* defines the variogram model of variable `id` (section 4.3)

`variogram(id1,id2):` *body*`;` here, *body* defines the cross variogram of variables `id1` and `id2` (section 4.3)

`method:` *body*`;` here, *body* specifies the method (section 4.5)

`set` *parameter*`=`*value*`;` assign *value* to *parameter* (section 4.4)

4.1.2 Variogram modelling options

`bounds:` *body*`;` *body* is either a list with (white-space or comma-separated) strictly increasing interval boundary values, or it is a file name, pointing to a file that contains such a list

4.1.3 Prediction or simulation options

`mask:` *body*; here, *body* defines the input mask map(s), with the locations where predictions will be made (the non-missing valued cells), and the values of the user-defined base functions at map prediction locations (for universal kriging or linear models) or the category number (for stratified prediction or simulation); for multiple mask maps *body* is a comma-separated list of file names.

`predictions(id): 'file';` here, `file` defines the output map with the predictions on variable `id`

`estimates(id): 'file';` synonymous to the `predictions` command

`variances(id): 'file';` here, `file` defines the output map that will hold the prediction error variances on `id`

`covariances(id1,id2): 'file';` here, `file` defines the output map that will hold the prediction error covariances on variables `id1` and `id2`

`data():` *body*`;` here, *body* defines the non-gridded prediction locations

`blocksize:` *body*`;` here, *body* defines the block size (default 0, see section 3.5)

`edges:` *body*`;` here, *body* is a comma-separated list with files containing open or closed polygons. Edges (boundaries) may be used in interpolation to further constrain a neighbourhood definition: only when a point is on the same side of an edge as the prediction location, it will be included for prediction (or simulation). See Appendix B for details and polygon file formats.

`area:` *body*`;` here, *body* defines the 'block' discretization points (section 3.5, Appendix A.3)

`merge id1(`*i*`) with id2(`*j*`);` In multivariable ordinary or universal kriging (or simulation), by default each variable has it's own set of parameters β_k. The merge command allows to define a *common* parameter for two or more variables. Suppose, $Z_1(x) = m + e_1(x)$ and $Z_2(x) = m + e_2(x)$, where m is the unknown *common* mean for both variables. Note: the variable numbers *i* and *j* start at 0; `merge id1 with id2` is the abbreviation of `merge id1(0) with id2(0)` (see also Appendix A.2).

4.2 'data'

The general form of the `data` command is

`data(`*identifier*`): 'file',` *options* `;`

The file name should refer to an existing file in ascii table form, simplified GeoEAS format or one of the supported grid map formats. Options can be single keywords like `log` or expressions like `x=2`. Column 0 means not defined (non-existent). The full list of options is [default values between square brackets]:

4.2.1 General options

`v=5` column 5 contains the data (measurement) variable [0, or obtained from grid map]

`x=1` column 1 contains the x-coordinate [0, or obtained from grid map]

`y=2` column 2 contains the y-coordinate [0, or obtained from grid map]

`z=3` column 3 contains the z-coordinate [0, or obtained from grid map]

`d=1` use a first order (polynomial) linear model in the coordinates as the trend; allowed order values are 0, 1, 2 and 3; see also `X`, `sk_mean` and `b` [0: only an intercept as trend]

`mv=-1` define missing value as the value -1 [the string `NA`, see also `set mv`] `average` average values with identical locations (i.e., their separation distance is less than `zero`) [noaverage]

`log` log transform the variable (natural logarithm) [no transform]

`I=5` transform the observation variable v to

$$I(v,5) = \begin{cases} 1 & \text{if } v \leq 5, \\ 0 & \text{otherwise} \end{cases}$$

[no transform]

`v=6, Category='sand'` transform the observation variable v to 1 if the string in column 6 equals the Category string `sand`, and to 0 in any other case. [no transform]

`ns='filename.out'` transform the observations to their normal score, and write the normal score table to `filename.out`. In this file, ach line contains the (sorted) original value and its normal score. Normal scores are computed as

$$n_j(x_i) = \Phi^{-1}((j + 0.5)/n)$$

with j the *rank* (1...n) of $z(x_i)$ and $\Phi(\cdot)$ the Gaussian cumulative density function. In case of ties (when multiple $z(x_i)$ have the same value), ranks are averaged for the tied data before normal scores are calculated, to avoid assignment of arbitrary values. Gstat does (currently) not provide any means for backtransformation.

`standard` standardise variable (to mean 0, variance 1) [do not standardise]

`X=8&9&x&y` apart from a default intercept, the values of the base functions at the data locations are the variables that are in columns 8, 9 and the x- and y-coordinate. (Polynomial coordinate base functions allowed are: `x3` for x^3, `y3` for y^3, `z3` for z^3, `x2` for x^2, `y2` for y^2, `z2` for z^2, `x` for x, `y` for y, `z` for z, `x2y` for x^2y, `xy2` for xy^2, `x2z` for x^2z, `xz2` for xz^2, `y2z` for y^2z, `yz2` for yz^2, `xy` for xy, `xz` for xz and `yz` for yz, provided that the corresponding coordinate is defined) [use only an intercept (mean) as trend]

`X=-1&8&9` the values of the base functions at the data locations are on columns 8, 9 and 10, with no intercept [use only an intercept (mean) as trend] `b=[2.4, 1.7, -3.9]` define the (known) regression coefficients, corresponding to the `X` entries given. See also `sk_mean`; `b` generalises the concept of a known constant mean to a known mean function. [undefined; regression coefficients are unkonwn]

`V=6` column 6 contains the proportionality factor v_i to the residual variance $v_i\sigma^2$ of the `v`-variable (i.e. the diagonal entries of matrix D, see *Ordinary and weighted least squares trend prediction* in section 2.7, and Appendix A.2). This will have an effect on the least squares residuals (thus affecting the sample covariogram and pseudo cross variogram), as well as on uncorrelated least squares prediction, kriging prediction, and trend prediction. [0: assuming identical variances or variances strictly derived from the variograms]

`every=10` For regular (systematic) sampling records from a data file, `every` is set to the step size. A value of 10 only samples data records $1, 11, 21, ...$ [1: don't sample but select all data]

`offset=1` Controls the starting sample element for regular sampling. Effective in combination with `every` only. When `every=10` and `offset=2`, then elements $2, 12, 22, ...$ are selected; if `every=20` and `offset=5`, elements $5, 25, 45, ...$ are selected. [1: start sampling at first data point].

`prob=0.1` Inclusion probability for random sampling data records from the file [1: all records are read].

4.2.2 Variogram modelling options

`noresidual` do not calculate OLS (or GLS, see `gls`) residuals for sample variogram or covariogram estimation (Appendix A.1). For sample variogram estimation in absence of base functions, setting `noresidual` will yield identical results, but will result in a modest gain in speed and memory saving. In other cases, it will result in the estimation of non-centred covariograms or pseudo-cross variograms [calculate residuals before variogram or covariogram estimation]

`dX=0.1` include a pair of data points $\{z(x_i),\ z(x_j)\}$ for sample variogram calculation only when $||f(x_i)-f(x_j)|| \leq 0.1$ with $f(x_i) = (f_1(x_i), ..., f_p(x_i))$ and $||u|| = \sqrt{u'u}$. This allows *pooled* estimation of within-strata variograms, or variograms of (near-)replicates in a linear model (for point pairs having similar values for regressors like depth, time, or a category variable) [do not evaluate]

4.2.3 Prediction or simulation options

`radius=4.5` select observations in a local neighbourhood when they are within a distance of 4.5 [large: select all] (see section 2.5)

`max=30` maximum number of observations in a local neighbourhood selection is 30 [large: no maximum] (see section 2.5)

`min=10` minimum number of observations in a local neighbourhood selection is 10 [0] (see section 2.5)

`omax=2` maximum number of observations per octant (3D), quadrant (2D) or secant (1D) is 2 (this only works in addition to a `radius` set) [0: don't evaluate] (see section 2.5, and also `method: nr;`, section 4.5)

`square` select points in a square (or block) neighbourhood, with square (block) sizes equal to 2 $\times$ radius [circular (spherical) neighbourhood]

`vdist` use variogram value as the distance criterium for `min`/`max`/`omax` neighbourhood selection (but define `radius` as euclidian distance) [use Euclidean distance] (see section 2.5)

`force` force neighbourhood selection to the minimum number of observations, disregarding the distance [unless simple kriging is used, generate a missing value if less than the required minimum number of observations are found within a distance defined by `radius`] (see section 2.5)

`s=7` define variable with the strata for `data()` locations [0, no strata]

`sk_mean=2.4` define the simple kriging mean to be 2.4 [not defined: an unknown mean (intercept) is assumed for each variable]. NOTE: the code `sk_mean=2.4` is equivalent to `b=[2.4]`

`dummy` define a dummy variable [require valid data to be read]

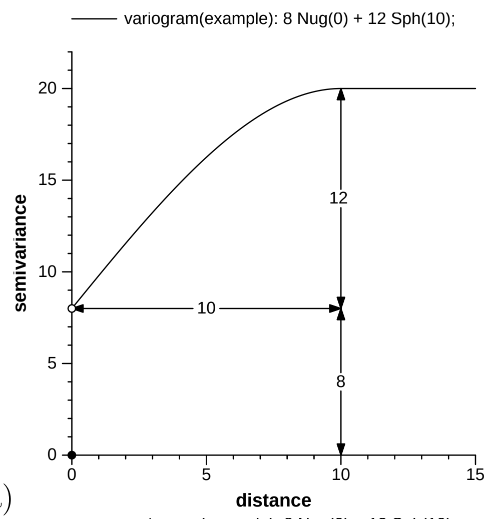
4.3 ‘variogram’

Variogram models are coded as the sum of one or more simple models (and optionally an anisotropy structure). A simple variogram model is denoted by

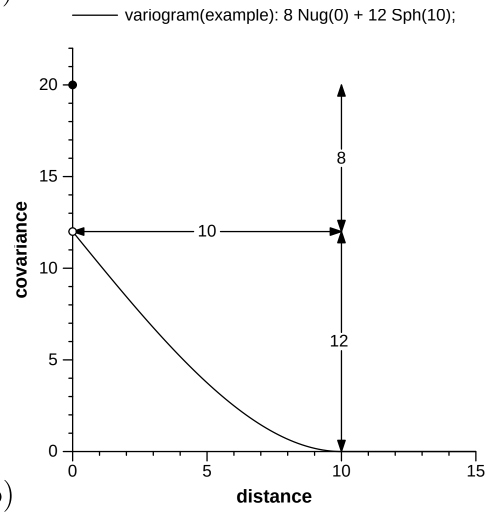
$c\,Mod(a)$

with c the vertical (variance) scaling factor, Mod the model type, and a the range (horizontal, distance scaling factor) of this simple model. If Mod is a transitive model (i.e. after some distance, or asymptotically as $h \to \infty$ the simple model reaches a certain maximum) then c is the partial sill of that model and the simple covariogram that corresponds to this variogram model is:

$c(1 - Mod(a))$



(a)



(b)

Figure 4.1: Example variogram `8 Nug() + 12 Sph(10)`. Semivariance representation (a) and covariance representation (b)

Fig. 4.1 gives an example of the variogram model `8 Nug() + 12 Sph(10)` as semivariance and covariance representation. Unit models available in gstat are listed in table 4.1.

All unit basic variogram models are shown in Fig. 4.2. Note that the

model	**syntax**	$\gamma(h)$	h range
Nugget	`1 Nug(0)`	0 1	$h = 0$ $h > 0$
Spherical	`1 Sph(a)`	$\frac{3h}{2a} - \frac{1}{2}(\frac{h}{a})^3$ 1	$0 \leq h \leq a$ $h > a$
Exponential	`1 Exp(a)`	$1 - \exp(\frac{-h}{a})$	$h \geq 0$
Linear	`1 Lin(0)`	h	$h \geq 0$
Linear-with-sill[1]	`1 Lin(a)`	$\frac{h}{a}$ 1	$0 \leq h \leq a$ $h > a$
Circular	`1 Cir(a)`	$\frac{2h}{\pi a}\sqrt{1 - (\frac{h}{a})^2} + \frac{2}{\pi} \arcsin \frac{h}{a}$ 1	$0 \leq h \leq a$ $h > a$
Pentaspherical	`1 Pen(a)`	$\frac{15h}{8a} - \frac{5}{4}(\frac{h}{a})^3 + \frac{3}{8}(\frac{h}{a})^5$ 1	$0 \leq h \leq a$ $h > a$
Gaussian	`1 Gau(a)`	$\gamma(h) = 1 - \exp(-(\frac{h}{a})^2)$	$h \geq 0$
Bessel[2]	`1 Bes(a)`	$1 - \frac{h}{a}K_1(\frac{h}{a})$	$h \geq 0$
Logarithmic	`1 Log(a)`	0 $\log(h + a)$	$h = 0$ $h > 0$
Power	`1 Pow(a)`	h^a	$h \geq 0, 0 < a \leq 2$
Periodic	`1 Per(a)`	$1 - \cos(\frac{2\pi h}{a})$	$h \geq 0$

Table 4.1: Simple variogram models in gstat: the building blocks for a variogram model

`Exp()`, `Gau()` and `Bes()` models reach their sill asymptotically (as $h \to \infty$). The 'effective range', is the distance where the variogram reaches 95% of its maximum, and this is 3`a` for `Exp(a)`, $\sqrt{3}$`a` for `Gau(a)` and 4`a` for `Bes(a)`. The logarithmic and power model are unbounded (and are therefore not suitable for covariance modelling or simple kriging).

Pseudo cross variograms may have a non-zero value for $h = 0$. An intercept can be defined as a constant added to the variogram model, e.g.

```
1.5 + 0.5 Nug() + 2.2 Sph(20)
```

or, equivalently

```
1.5 Int() + 0.5 Nug() + 2.2 Sph(20)
```

and can be fitted only when a sample variogram estimate is available at zero distance.

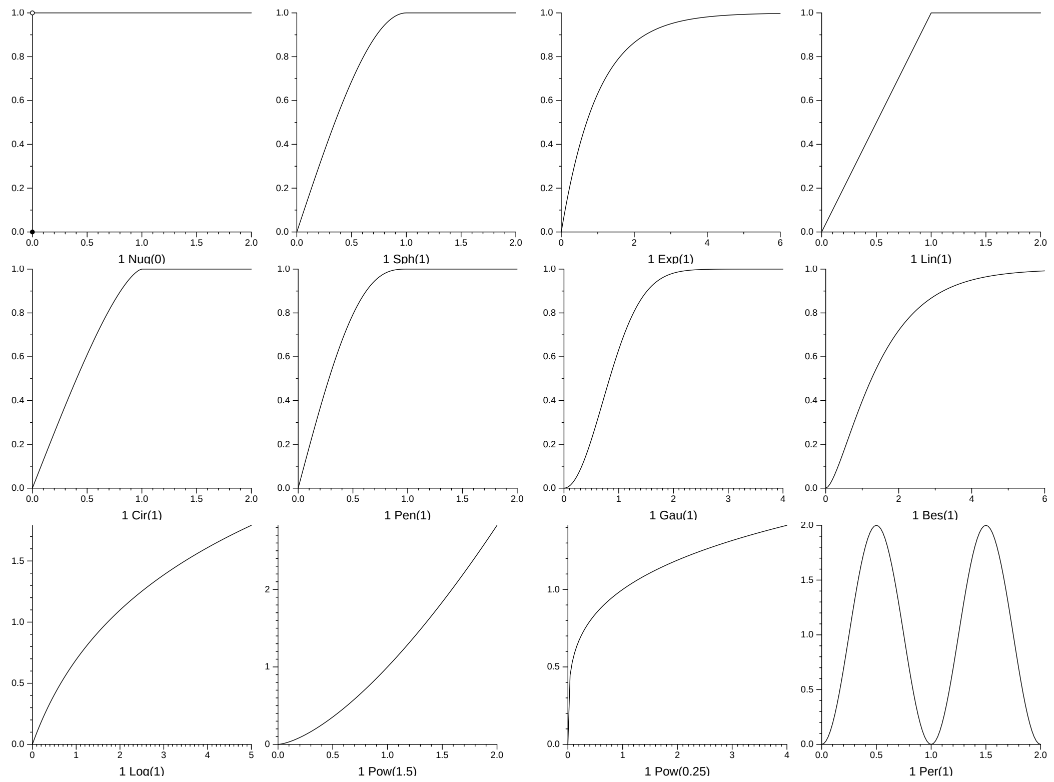


Figure 4.2: The unit basic variogram models

Geometric anisotropy

Geometric anisotropy can be modelled for each individual simple model by addition of two or five anisotropy parameters after the range, e.g.

$c\,Mod(a, p, s)$

for 2-d anisotropy, or, for 3-d anisotropy:

$c\,Mod(a, p, q, r, s, t)$

Using anisotropy, the variogram model range parameter (a) is the maximum range, that of the major direction of continuity (direction of spatial correlation at longest distances). The range in the direction perpendicular to the major direction is the minor range. The anisotropy ratio is the ratio between the minor range and the major range (a value between 0 and 1).

A two-dimensional range ellipse is defined by (a, p, s), a three-dimensional ellipsoid is defined by (a, p, q, r, s, t). In the two-dimensional case p is the angle for the principal direction of continuity (measured in degrees, clockwise from positive Y, north), and s is the anisotropy ratio. So, the range in the major direction (p) is a, and the range in the minor direction ($p + 90$) is as.

In three dimensions p is the angle for the principal direction of continuity

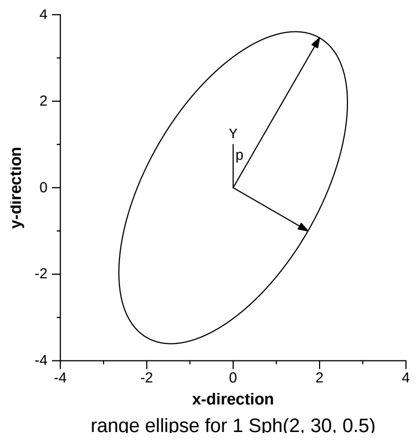


Figure 4.3: anisotropy ellipse

(measured in degrees, clockwise from Y), q is the dip angle for the principal direction of continuity (measured in positive degrees up from horizontal), r is the third rotation angle to rotate the two minor directions around the principal direction defined by p and q. A positive angle acts clockwise while looking in the principal direction. Anisotropy ratios s and t are the ratios between the major range and each of the two minor ranges. (Note that $c\,Mod(a, p, s)$ is equivalent to $c\,Mod(a, p, 0, 0, s, 1)$.)

Zonal anisotropy

Zonal anisotropy (anisotropy in the sill) can be obtained by defining geometric anisotropy with large anisotropy ratios. For instance, when the spatial working domain is not too large (the largest distance in the area considered does not exceed, let's say, 100), then the model

```
1 Sph(2e5, 90, 1e-5)
```

will have nearly zero values (and thus be practically absent) in direction 90 (east-west), whereas it will reach the sill (1) at distance 2 in direction 0 (north-south).

4.4 'set'

The general form of the `set` command is

`set` *parameter* = *value* ;

[1]only valid for one-dimensional data

[2]$K_1(\cdot)$ is the first order modified Bessel function of the second kind

The list of variables that can be 'set' [default value between brackets]:

4.4.1 General options

`set debug=2;` set debug level to 2 [1; options are listed in Appendix C.4]

`set cn_max=1.0e8;` check the condition number of matrices. If it is larger than `cn_max`, then generate a missing value and report a (near) singularity warning. Matrices checked are V and $F'V^{-1}F$ (or $F'D^{-1}F$). A suitable value for `cn_max` seems $1/\sqrt{\texttt{DBL\_EPSILON}}$, which is about 10^8. Condition numbers are estimated using LU factorization Stewart and Leyk (1994), and may be an order of magnitude wrong. Condition numbers are reported if `debug` is set to report covariance matrices. [not set: check for singularity only during variogram model fitting]

`set logfile='gstat.log';` set the file name where debug information is written to (see `set debug` and Appendix C.4) [stdout, debug information is written to the screen]

`set mv='MisVal';` define the default missing value string as `MisVal` [the string `NA`] (Note that numerical missing values can be defined with `mv` in a `data` command)

`set output='file';` write ascii output to `file` (e.g., variogram estimates, predictions and variance at non-gridded locations)

`set plotfile='file';` `file` defines the file name for gnuplot commands (not set, use temporary files). When set during ordinary or universal kriging, kriging weights are written as plot files for gnuplot (see `plotweights`). Affects file name usage for variogram plotting through gnuplot.

`set zero=1.0e-10;` specify the highest value absolote differences in distance and prediction variances may have to be considered equal to zero [$10 \times$ `DBL_EPSILON`, about 2^{-15}].

`set marginals=`*`list`*`;` List of values or maps with the mean and variance of the first variable, the second variable, ... See also section A.4.

4.4.2 Variogram modelling options

`set alpha=45.0;` directional sample (co-) variogram: set direction in $< x, y >$ plane, in positive degrees clockwise from positive y (North) [0.0] `set beta=30.0;` directional sample (co-) variogram: set direction in z, in positive degrees up from the $< x, y >$ plane [0.0]

`set Cressie=1;` (for sample variogram calculation) use Cressie's square-root variogram estimator [0]

`set cutoff=0.5;` set cutoff (max. dist. for sample variogram) at 0.5 [a fixed fraction of the maximum distance, see `set fraction`]

`set dots=1000;` change the number of plotting points at which gstat will let gnuplot switch from plotting points (+) with numbers of points of pairs, to plotting dots without numbers [500]

`set fit=1;` fit the variogram model to the experimental variogram, using weighted least squares fit. Values for `fit` are shown in table 4.2 [0, do not fit]

`fit`	fit by	weight
0	-	- (no fit)
1	gstat	N_j
2	gstat	$N_j/\{\gamma(h_j)\}^2$
3	gnuplot	N_j
4	gnuplot	$N_j/\{\gamma(h_j)\}^2$
5	gstat	REML
6	gstat	no weights (OLS)
7	gstat	N_j/h_j^2

Table 4.2: values for `fit`

`set fit_limit=1.0e-10;` set fit limit to 1.0e-10 (Appendix A.1) [1.0e-6]

`set format='%.3g';` the format used for real values in variograms, e.g. `%.3g` limits the number of significant digits shown to 3. A valid C-language format string for a double should be used, misspecification may result in unpredictable behaviour. [`%g` : use 6 significant digits]

`set fraction=0.25;` specify the default cutoff for sample variogram calculation as fraction of the length of the diagonal in the square or block spanning the data locations [0.333]

`set gnuplot='mygnuplot';` invoke the program `mygnuplot` as gnuplot (variogram display) [gnuplot, or wgnuplot for Win32] `set gnuplot35='gpt35';` invoke the program `gpt35` (gnuplot version 3.5) for variogram display only [use gnuplot, or the value of `set gnuplot`]

`set gpterm='latex';` set the gnuplot terminal specification and options (a string to follow the gnuplot "set term" command). This option will overrule the 'postscript' or 'gif' settings from the variogram modelling user interface, thus allowing plotting to other graphic file formats and modification of options (see gnuplot documentation). [for gif: `'gif transparent size 480, 360'`, for PostScript: `'postscript eps solid 17'`

`set intervals=20;` specify the default number of intervals for sample variogram calculation [15]

`set iter=20;` use not more than 20 iterations on iterative fit methods [50]

`set pager='less';` use 'less' as pager to be called from the variogram modeling interface [the value of the environment variable `PAGER` (if set), or else the program `more`]

`set secure=1;` prevent any calls to the functions `system()`, `popen()` or `remove()`, terminate program whenever one of the first two appear (once set, it cannot be set back) [0, not secure]

`set sym=1;` force directional sample cross covariance and pseudo cross semivariance to be symmetric [0, asymmetric]

`set tol_hor=45.0;` directional sample (co-)variogram: set horizontal tolerance angle in degrees [90.0]

`set tol_ver=20.0;` directional sample (co-) variogram: set vertical tolerance angle in degrees [90.0]

`set width=0.05;` set lag width to 0.05 (distance interval width for sample variogram) [`cutoff/intervals`]

`set gls=1;` use generalised least squares residuals instead of the default ordinary least squares (OLS) residuals for sample variograms or covariograms [0, use OLS or WLS residuals]

`set zero_dist=1;` determine what happens with variogram estimates at distance zero. Values are 1: include in first interval, 2: omit, 3: calculate separately [1 for variograms, 3 for covariograms]

4.4.3 Prediction or simulation options

`set idp=3.5;` set inverse distance power to 3.5 [2]

`set nblockdiscr=10;` use regular block discretization with 10 points in each dimension at non-zero block size (note: 10 in 3 dimensions results in 1000 discretizing points) [4, and use Gauss quadrature (see Appendix A.3)]

`set nsim=100;` create 100 independent simulations when following a single random path (output maps will get the simulation number attached to their names, therefore short names should be chosen in environments with file name restrictions) [1]

`set n_uk=40;` (for conditional simulation only) use universal (or ordinary) kriging instead of simple kriging when the number of data in a kriging setting is greater than or equal to 40. For multivariable prediction the neighbourhood size is summed over all variables, otherwise it is evaluated per variable. Setting `n_uk` to zero limits use of simple kriging to empty neighbourhoods only [very large: always use simple kriging]

`set order=2;` define the action when order relation violations occur during indicator simulation (section 2.6; table 2.1) or indicator kriging (section 2.5). Values are 0: no correction for indicator kriging, assure that estimated probabilities are in [0,1] before simulation; 1: as 0, but also for indicator kriging; 2: rescale the estimated probabilities if their sum is larger than 1; 3: rescale the estimated probabilities so that they sum up to 1; 4: do order relation correction for cumulative indicators (using the upward-downward averaging steps of GSLIB Deutsch and Journel (1992)). [0: do only basic order relation violation corrections for indicator simulation]

`set plotweights=10;` When `plotfile` is set, kriging weights will be plotted during ordinary or universal kriging. If `plotweights` is larger than 1, data point sizes are proportional to kriging weights using size intervals of $\frac{1}{\texttt{plotweights}}$. If not (default), kriging weights are plotted as data point labels (i.e., as text).

`set quantile=0.25;` when `method` was set to `med`, report p-quantile of local neighbourhood selection as prediction value, and $(1-p)$-value as prediction variance [0.5: the median]

`set rp=0;` follow regular, non-random path during sequential simulation [1, follow a random path] `set seed=1023;` set seed for random number generator [0: seed is read from the internal clock. If possible, microseconds are used. To check this, run gstat a few times with `debug` set to 2].

`set useed=4053341103U;` set seed for random number generator when outside the range of a signed integer; note the 'U' at the end of the number [see `seed`].

`set sparse=10;` Use sparse matrix routines for covariance matrix. The number of `sparse` should be a reasonable estimate of the number of non-zero columns in each row of the covariance matrix V. [only available when sparse matrix routines in meschach are linked in; 0: use dense matrices]

`set xvalid=1;` turn cross validation on (if prediction is possible) [0, no cross validation]

`set zmap=10.0;` set height of mask map(s) to 10 when observations are 3-D. [0.0]

`set lhs=1;` Apply Latin hypercube sampling to Gaussian simulations; see also `marginals`

`set nocheck=1;` Ignore error in case of an non-permitted coregionalisation (intrinsic correlation or linear model of coregionalisation)

4.5 '`method`'

Values for `method` that make gstat deviate from the default action are:

`gs` override the default kriging method to get Gaussian simulations (section 2.6)

`is` override the default kriging method to get indicator simulations (section 2.6)

`semivariogram` sample semivariance on output (see also `fit`, section 4.4)

`covariogram` sample covariogram on output

`trend` spatial trend estimation ($x_0\hat{\beta}$), using generalised least squares if variograms are specified, or else using ordinary or weighted least squares (see Appendix A.2) `map` report the value of m mask maps at the prediction location of `data()` as the output values, provided that n (dummy) input variable are defined, with $n \geq m/2$ (example [6.9])

`distance` report distance to nearest observation as the predicted value, and the distance to the most distant observation (in the neighbourhood selection, if defined) as prediction variance

`nr` report number of observations in neighbourhood as predicted value, and, if omax was set, the number of non-empty octants (or quadrants) as prediction variance

`div` diversity (the number of distinct values) in a local neighbourhood is reported as the predicted value, the range of the values (largest value − smallest value) is reported as prediction variance

`med` local median or quantile estimation, see `quantile` in section 4.4

`point-in-polygon` Provided a set of (closed) polygons is given with the `edges` command, the (first) polygon in which the prediction location is located is given in the output. Predicted value carries the file number of the polygon, prediction variance the polygon number in the file. See also Appendix B

Chapter 5

Further control

5.1 Start-up

When gstat is started, it first looks for an initialization file to load. If the environment variable `GSTATRC` specifies a file name, gstat will read the contents of this file, or else gstat will search for the file `$HOME/.gstatrc` and read it when found. If present, the initialization file should contain valid gstat commands. After reading the initialization file, command line options are processed, and finally the command file is processed. The processing order determines when previously set commands will be overridden.

Information on gnuplot customization is obtained by starting gnuplot and typing “help” or “help environment”. (If the help file is not found, the environment variable `GNUHELP` should be set to the gnuplot help file, `gnuplot.gih`.)

5.2 Command line options

Some options in gstat can be specified on the command line. Command line options appear between the gstat command and the command file name, options start with a `-`, and option arguments follow the option letter, possible separated by white space, e.g. the command line

```
gstat -l gstat.log -d 2 zinc.gst
```

has two options, ‘`l`’ having argument `gstat.log` and ‘`d`’ having argument 2. The final argument, `zinc.gst` is not part of an option, it is the main argument (the command file). A short list of command line options is obtained by typing

```
gstat -h
```

option	*meaning*	*keyword*
`-C`	print copyright notice	
`-W`	print no-warranty notice	
`-v`	print long version information	
`-i`	enter the interactive variogram modeling user interface (only available when `method` was not set previously)	
`-d` n	invoke debug level n, see table C.4	`debug`
`-s`	silent mode: print only warnings and errors	`debug`
`-o file`	specify (ascii) output file	`output`
`-p file`	specify gnuplot plot file	`plotfile`
`-l file`	specify (ascii) log file for debug information (default to screen)	`logfile`
`-r` n	use random number generator n (try -1; see section 5.3)	
`-S`	secure mode	`secure`
`-e` *action*	execute action *action* (see section 5.5)	

Table 5.1: command line options

Special file names (pipes, append files) are explained in section 5.4. The list of options is given in table 5.1.

5.3 Random number generators

For the simulation of random fields, gstat can use one of several random number generators (RNG). The default generator depends on the platform on which gstat was compiled:

- when the GNU Scientific Library (GSL) was present, the default rng is mt19937 in this library
- else, when drand48 is present on the platform, this random number generator is used by default
- else, Marsaglia's random number generator is used.

The default RNG can be controlled with the command line option `-r` (section 5.2), try

```
gstat -r -1.
```

for an overview of available RNG's.

The GSL provides 26 different generators, and the one used is controlled by setting the environment variable GSL_RNG_TYPE to one of the following values: `ranlux389 ranlux cmrg mrg mt19937 tt800 taus ran0 ran1 ran2 ran3 ranf rand48 ranmar zuf slatec r250 random minstd uni uni32 vax transputer rand random8 randu`. See the GSL documentation for details.

The seed of the RNG can be set by the variable `seed`. Default, the seed is read from the CPU clock (using microseconds, when the function `gettimeofday` is present, or else seconds). For the GSL, the environment variable GSL_RNG_SEED may be used to override the seed in gstat (set or default).

5.4 Special file names

Certain special file names are allowed in gstat. They enable filtering, appending to files, using standard input or output streams and command output substitution. (They will not have this effect for PCRaster file names.)

`'> file'` If `file` exists, then append the output to `file` (if it is an output file) instead of starting with a fresh file

`'| file'` Open `file` as a pipe, either for reading or for writing

`'-'` Use, instead of a disk file, for a reading process the stream `stdin`, or for a writing process the file `stdout`

`` `cmd` `` execute the shell command `cmd` and substitute its output for the file name

Using pipes is at the user's responsibility. Blindly executing command files that contain file names with pipes to harmful commands may result in damage (loss of files for instance). This also applies to the setting of the `gnuplot` command(s), see section 4.4. Potential damage from such situations is considered as "consumers risk", it is comparable to renaming harmful programs to often-used commands.

For safety reasons, the variable `secure` can be set to 1, which prohibits gstat from system calls, creating pipes or deleting temporary files. Unlike other variables, once `secure` is set, it cannot be set back (indeed, for security).

As an example of using a pipe as file name, the first variable in example [6.12] containing the data selection from `zinc_map.eas` with a zero in column 5, could have been defined directly in terms of `zinc_map.eas` as

```
data(zinc.at.0):
"| awk '{if(NR<11||$5==0){print $0}}' zinc_map.eas",
 x=1, y=2, v=3, log, min = 20, max = 40, radius = 1000;
```

provided that the `awk` program is available.

5.5 Execute (-e) actions

Several simple "special actions" are available in ("linked into") gstat. They are invoked as
`gstat -e` *action arguments*
and are strictly controlled by command line options and arguments. A list of available *action*s is obtained by calling
`gstat -e`
and usage for each *action* is printed after running
`gstat -e` *action*
without arguments. Following is a list of available *actions*:

`cover` *out_map in_map1 in_map2 ...* cover several input maps into the output map: substitute in *out_map* the missing values of *in_map1* with the first non-missing value found in subsequent input maps, on a cell-by-cell basis

`convert` Converts grid map file from one format to another (minimal converter)

`nominal` *out_map in_map0 in_map1 ... in_mapn* from multiple input grid map files write in each grid cell of *out_map* the number of the first map with a non-zero value in that grid cell (a value in $[0, ..., n]$)

`map2fig` convert point data or grid map data to fig (XFig/fig2dev) file. XFig is a vector graphics editor for X-Windows that supports exporting to numerous formats.

`map2gif` convert grid map file to gif file. The gif library used to generate gif's from maps is not distributed as part of the gstat distribution; see `http://www.boutell.com/gd/`

`map2png` See `map2gif`. Modern versions of the gd library only support PNG instead of GIF, because of the unisys license thing. If such a version of gd is found, gstat only supports map2png to create PNG's.

`semivariogram` calculate sample variogram, using command line interface `covariogram` calculate sample covariogram, using command line interface

`semivariance` print semivariance table for a variogram model

`covariance` print covariance table for a variogram model

`statistics` output summary statistics for values read from one or more files (for each file: min, max, median, quantiles, mean, standard deviation, n)

5.6 Compiling and installing gstat

In order to work with gstat, binary executables of gstat and gnuplot are needed. Both programs can be obtained in source form from the internet and they can be compiled to executables using an ANSI-C compiler. On Unix systems, typing in the gstat directory

```
./configure
```

followed by

```
make; make install
```

may be sufficient for compiling and installing gstat.

Similarly, for compiling gnuplot, typing

```
./configure; make; make install
```

in the gnuplot directory may be sufficient to compile an install gnuplot. Type

```
./configure --help
```

to obtain the command line options for non-default configuring (e.g. of install directories, libraries used). By default, for compatibility reasons the output files from flex and bison are used for compiling gstat. If you want to use your own lex or yacc version, remove the files lex.c, parse.c and parse.h from the src directory. From bash or ksh, this is also accomplished by:

```
LEXYACC=1 ./configure
```

To my knowledge gstat has been successfully compiled on HP-UX, IBM AIX, Dec OSF/Alpha, SunOS, SGI, Linux, MS-Windows (95, NT) and MS-DOS/DPMI. When modifications to the makefiles or source files are necessary to make the software run, please report them to `gstat-info@geog.uu.nl`

Gstat requires two external libraries: the Meschach matrix library Stewart and Leyk (1994) (available from netlib) and PCRaster grid map API (csf, version 2). Both are distributed as part of the gstat source code distribution. One other library is optional, and required for `map2gif`: the gd gif library (copyright 1994, 1995, Quest Protein Database Center, Cold Spring Harbor Labs, see `http://www.boutell.com/gd/index.html`).

Binary executable files should be installed in a directory in the search path for executables.

Chapter 6

Example command files

Following are the example command files, as distributed with gstat. The data (zinc concentrations of the top soil, Fig. 6.1a) are collected in a flood plain of the river Maas, not far from where the Maas entered the Netherlands (Borgharen, Itteren, about 3 to 5 km North of Maastricht). All coordinates are in metres, using the standard coordinates of Dutch topographical maps. Moving from the river, zinc concentrations tend to decrease(Fig. 6.1b).

For the universal kriging examples, let the function $D(x)$ be the function that is for every location x the (normalised) square root distance to the river. This function is physically stored for the observation locations in column 4 of `zinc_map.eas` (as obtained in example [6.9]), and for the prediction locations in the map `sqrtdist.map` (Fig. 6.1c). The prediction area is split in two separate sub-regions, A and B (Fig. 6.1d). Let the function $I_A(x)$ be 1 if $x \in A$ or else 0; and let the function $I_B(x)$ be 1 if $x \in B$ or else 0.

The functions $I_A(x)$ and $I_B(x)$ are physically stored for the observation locations in columns 5 and 6 of `zinc_map.eas`, and for the prediction locations in the maps `part_a.map` and `part_b.map`. (Note that the partitioning in A and B is arbitrary, it serves only illustrational purposes.)

The following command files, data and maps are distributed with gstat. The online (html) version of this manual has example command files with hyperlinks to all input data and (figures of) output maps.

These command files are purely for illustration and as such only suggest possible forms of analysis. Remind that everything from a `#` to the end of the line is comment.

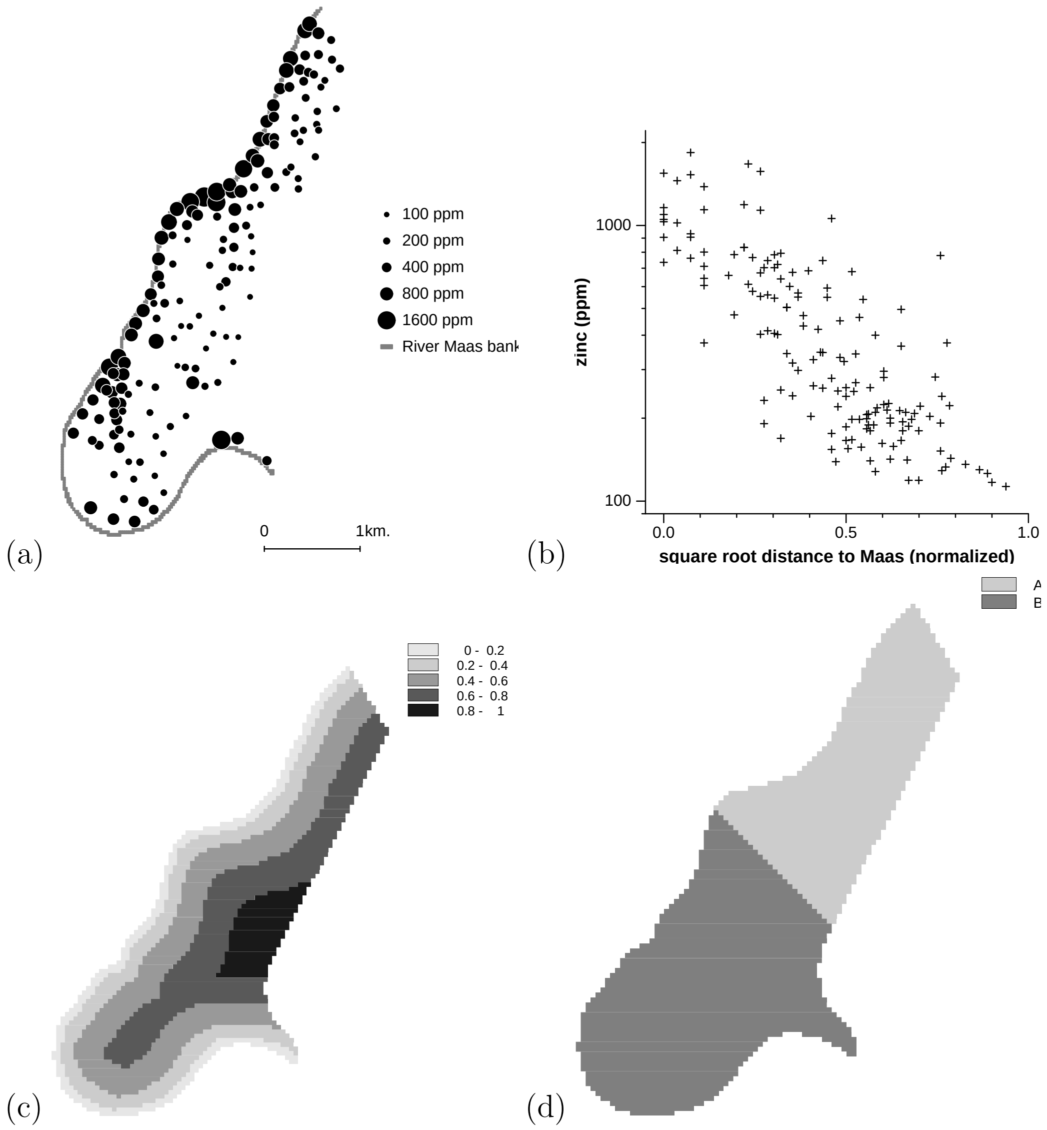


Figure 6.1: (a) map of zinc measurements in the top soil, (b) scatter plot of zinc measurements and distance to the river Maas, (c) map with normalised square root distance to the river Maas, (d) partitioning of the prediction area

6.1 Example 1. Variogram modelling

```
#
# One variable definition:
# to start the variogram modelling user interface.
#
data(zinc): 'zinc.eas', x=1, y=2, v=3;
```

6.2 Example 2. Variogram modelling of two variables

```
#
# Two variables with (initial estimates of) variograms,
# start the variogram modelling user interface
#
data(zinc): 'zinc.eas', x=1, y=2, v=3;
data(ln_zinc): 'zinc.eas', x=1, y=2, v=3, log;

variogram(zinc): 10000 Nug() + 140000 Sph(800);
variogram(ln_zinc): 1 Nug() + 1 Sph(800);
```

6.3 Example 3. Inverse distance interpolation

```
#
# Inverse distance interpolation on a mask map
#
data(zinc): 'zinc.eas', x=1, y=2, v=3;
mask: 'mask_map'; # the prediction locations
predictions(zinc): 'id_pr'; # result map
```

6.4 Example 4. Ordinary block kriging

```
#
# Local ordinary block kriging at non-gridded locations
```

```
#
data(zinc): 'zinc.eas', x=1, y=2, v=3,
min=20, max=40, radius=1000; # local neighbourhood
variogram(zinc): 2.42e+04 Nug(0) + 1.34e+05 Sph(800);
data(): 'locs.eas', x=1, y=2; # prediction locations
blocksize: dx=40, dy=40; # 40 × 40 block averages
set output = 'zincok.out'; # ascii output file
```

6.5 Example 5. Simple kriging on a mask map

```
#
# Local simple point kriging on a mask map
#
data(ln_zinc): 'zinc.eas', x=1, y=2, v=3, log,
min=20, max=40, radius=1000, sk_mean=5.9;
variogram(ln_zinc): 0.0554 Nug(0) + 0.581 Sph(900);
mask: 'mask_map';
predictions(ln_zinc): 'lzn_skpr';
variances(ln_zinc): 'lzn_skvr';
```

6.6 Example 6. Unconditional simulation

```
#
# Unconditional Gaussian simulation on a mask
# (local neigbourhoods, simple kriging)
#
# defines empty variable:
data(ln_zn_dummy): dummy, sk_mean=5.9, max=20;
variogram(ln_zn_dummy): 0.0554 Nug(0) + 0.581 Sph(900);
mask: 'mask_map';
method: gs; # Gaussian simulation instead of kriging
predictions(ln_zn_dummy): 'lzn_uspr';
```

6.7 Example 7. Conditional simulation

```
#
# Gaussian simulation, conditional upon data
# (local neighbourhoods, simple and ordinary kriging)
#
data(ln_zinc): 'zinc.eas', x=1, y=2, v=3, log,
sk_mean=5.9, max=20;
variogram(ln_zinc): 0.0554 Nug(0) + 0.581 Sph(900);
mask: 'mask_map';
method: gs;
predictions(ln_zinc): 'lzn_cspr';
set n_uk = 20;
# use ordinary kriging when ≥ 20 data in neighbouhood
# set nsim=10;
```

6.8 Example 8. Ordinary block kriging on a mask map

```
#
# Change of support: local ordinary block kriging on a mask
#
data(ln_zinc): 'zinc.eas', x=1, y=2, v=3, log,
min=20, max=40, radius=1000;
variogram(ln_zinc): 0.0554 Nug(0) + 0.581 Sph(900);
mask: 'mask_map';
predictions(ln_zinc): 'lzn_okbp';
variances(ln_zinc): 'lzn_okbv';
blocksize: dx=40, dy=40; # define block dimensions
```

6.9 Example 9. Map values at point locations

```
#
# Obtain map values at data() locations
# (Point-map overlay)
#
data(a): dummy; # define n dummy data variable (n=n_masks/2)
data(b):  dummy;
data():  'zinc.eas', x=1, y=2, v=3; # prediction locations
method:  map; # mapvalues as 'predictions'
masks:  'sqrtdist', 'part_a', 'part_b'; # the maps
set output = 'zincmap.eas'; # ascii output file.
```

6.10 Example 10. Multiple kriging

```
#
# Multiple kriging: prediction on more than one variable
# (ordinary kriging of two variables)
# (note that zinc_map.eas wass obtained through ex09.gst)
#
data(ln_zinc):  'zincmap.eas', x=1, y=2, v=3, log,
min=20, max=40, radius=1000;
data(sq_dist):  'zincmap.eas', x=1, y=2, v=4,
min=20, max=40, radius=1000;
variogram(ln_zinc):  0.0554 Nug(0) + 0.581 Sph(900);
variogram(sq_dist):  0.0631 Sph(900);
mask:  'mask_map';
predictions(ln_zinc):  'lzn_okpr';
variances(ln_zinc):  'lzn_okvr';
predictions(sq_dist):  'sqd_okpr';
variances(sq_dist):  'sqd_okvr';
```

6.11 Example 11. Multivariable kriging (cokriging)

```
#
# Multivariable kriging: ordinary local cokriging of two variables
#
data(ln_zinc):  'zincmap.eas', x=1, y=2, v=3, log,
min=20, max=40, radius=1000;
data(sq_dist):  'zincmap.eas', x=1, y=2, v=4,
min=20, max=40, radius=1000;
variogram(ln_zinc):  0.0554 Nug(0) + 0.581 Sph(900);
variogram(sq_dist):  0.0001 Nug(0) + 0.0631 Sph(900);
variogram(ln_zinc, sq_dist): 0 Nug(0)-0.156 Sph(900);
# NOTE: the 0 Nug(0)'s are added to make gstat recognize
# the Linear Model of Coregionalization
mask: 'mask_map';
predictions(ln_zinc): 'lzn_ckpr';
variances(ln_zinc): 'lzn_ckvr';
predictions(sq_dist): 'sqd_ckpr';
variances(sq_dist): 'sqd_ckvr';
# the next map holds the prediction error covariances:
covariances(sq_dist,ln_zinc): 'znsqdcov';
```

6.12 Example 12. Stratified ordinary kriging

```
#
# Stratified ordinary kriging (within-categorie ordinary kriging)
#
data(zinc.at.0): 'zincat0.eas', x=1, y=2, v=3, log,
min=20, max=40, radius=1000; # where part_a = 0
data(zinc.at.1): 'zincat1.eas', x=1, y=2, v=3, log,
min=20, max=40, radius=1000; # where part_a = 1
variogram(zinc.at.0): 0.0654 Nug(0) + 0.548 Sph(900);
variogram(zinc.at.1): 0.716 Sph(900);
# the mask map is 0 for zinc.at.0 locations, 1 for zinc.at.1
mask: 'part_a';
# stratified mode: one map holds predictions for all vars:
predictions: 'lzn_stpr';
# another the prediction variances for all vars:
variances: 'lzn_stvr';
```

6.13 Example 13. Universal kriging (a)

using the model $Z(x) = \beta\_1 + D(x)\beta\_2 + e(x)$:

```
#
# Local universal kriging, using one continuous variable
#
data(ln_zinc): 'zincmap.eas', x=1, y=2, v=3, log,
X=4, # sqrtdist values at data locations
```

```
min=20, max=40, radius=1000; # apply model locally
# the variogram should be that of the residual:
variogram(ln_zinc):  0.0674 Nug(0) + 0.149 Sph(700);
mask:  'sqrtdist'; # sqrtdist values at prediction locations
predictions(ln_zinc):  'lzn_ukpr';
variances(ln_zinc):  'lzn_ukvr';
```

6.14 Example 14. Universal kriging (b)

```
#
# Universal kriging, using one continuous and
# two binary variables.
#
data(ln_zinc):  'zincmap.eas', x=1, y=2, v=3, log,
X=-1&4&5&6;
# -1: no default intercept (col. 5 and 6 form an intercept)
# use global kriging: local kriging would lead to a singularity
# the variogram of e is:
variogram(ln_zinc):  0.0698 Nug(0) + 0.147 Sph(709);
# mask maps holding the independent variable values
# at prediction locations, their order corresponding
# that of the X-columns:
mask:  'sqrtdist', 'part_a', 'part_b';
predictions(ln_zinc):  'lzn_vkpr';
variances(ln_zinc):  'lzn_vkvr';
```

using the model $Z(x) = D(x)\beta\_1 + I\_A(x)\beta\_2 + I\_B(x)\beta\_3 + e(x)$:

6.15 Example 14a. Stratified universal kriging

using the model $Z\_i(x) = \beta\_i, 1 + D(x)\beta\_i, 2 + e\_i(x)$, i being the category number:

```
#
# Stratified universal kriging (within-categorie universal kriging)
#
data(zinc.at.0):  'zincat0.eas', x=1, y=2, v=3, X=4, log,
min=20, max=40, radius=1000; # where part_a = 0
```

```
data(zinc.at.1): 'zincat1.eas', x=1, y=2, v=3, X=4, log,
min=20, max=40, radius=1000; # where part_a = 1

# residual variograms:
variogram(zinc.at.0): 0.096572 Nug(0) + 0.226367 Sph(1069.33);
variogram(zinc.at.1): 0.115766 Sph(237.257);

mask: 'part_a', # 0 for zinc.at.0 locations, 1 for zinc.at.1 locs.
'sqrtdist', # predictor values corresp. to col. 4 for zinc.at.0
'sqrtdist'; # predictor values corresp. to col. 4 for zinc.at.1
predictions: 'lzn_stup';
variances: 'lzn_stuv';
```

6.16 Example 15. Linear model prediction (OLS)

using the model $Z(x) = \beta\_1 + D(x)\beta\_2 + e(x)$:

```
#
# Local linear model, using one continuous variable
#
data(ln_zinc): 'zincmap.eas', x=1, y=2, v=3, X=4, log,
min=20, max=40, radius=1000; # apply linear model locally
# no variogram definition: assume residual to be IID.
mask: 'sqrtdist';
predictions(ln_zinc): 'lzn_trpr';
# prediction variance for point locations:
variances(ln_zinc): 'lzn_trvr';
```

6.17 Example 16. Multivariable (cumulative) indicator simulation

```
#
# Multivariable indicator cosimulation
#
data(i200): 'zinc.eas', x=1, y=2, v=3, max=20, radius=1000,
I=200, sk_mean = 0.28;
```

```
data(i400): 'zinc.eas', x=1, y=2, v=3, max=20, radius=1000,
I=400, sk_mean = 0.56;
data(i800): 'zinc.eas', x=1, y=2, v=3, max=20, radius=1000,
I=800, sk_mean = 0.85;

# define an LMC:
variogram(i200): 0.0490637 Nug(0) + 0.182814 Exp(300);
variogram(i400): 0.0608225 Nug(0) + 0.21216 Exp(300);
variogram(i200, i400): 0 Nug() + 0.14806 Exp(300);
variogram(i800): 0.0550284 Nug(0) + 0.0842966 Exp(300);
variogram(i200, i800): 0 Nug() + 0.0525584 Exp(300);
variogram(i400, i800): 0 Nug() + 0.102852 Exp(300);

method: is;
mask: 'mask_map';
# apply order corrections for cumulative indicators:
set order = 4;

predictions(i200): 'i200pr';
predictions(i400): 'i400pr';
predictions(i800): 'i800pr';
# uncomment next line to get 5 simulations:
# set nsim = 5;
```

6.18 Example 17. Trend surface prediction

```
#
# global coordinate polynomial trend surfaces
# trend orders 0-3.
#
data(zinc.0): 'zinc.eas', x=1, y=2, v=3, d=0, log;
data(zinc.1): 'zinc.eas', x=1, y=2, v=3, d=1, log;
data(zinc.2): 'zinc.eas', x=1, y=2, v=3, d=2, log;
data(zinc.3): 'zinc.eas', x=1, y=2, v=3, d=3, log;
mask: 'mask_map';
# predict block averages for very small blocks:
blocksize: dx=1, dy=1;
# variances apply to mean values,
# not for single observations
```

```
predictions(zinc.0):  'lzn_tr0';
variances (zinc.0):  'lzn_vr0';
predictions(zinc.1):  'lzn_tr1';
variances (zinc.1):  'lzn_vr1';
predictions(zinc.2):  'lzn_tr2';
variances (zinc.2):  'lzn_vr2';
predictions(zinc.3):  'lzn_tr3';
variances (zinc.3):  'lzn_vr3';
```

Appendix A

Equations

A.1 Spatial dependence

Sample variogram and covariogram

All variograms and covariograms are calculated from predicted residuals $\hat{e}(s_i) = z(s_i) - \hat{m}(s_i)$, with $\hat{m}(s_i)$ the ordinary least square estimates of $m(s_i)$, fitted globally (all data of the variable are used in a linear model assuming IID errors), unless one of `dX`, `noresidual` or `gls` is set. The sample variogram is calculated from residuals from a single realization z for regular distance intervals $[h_j, h_j + \delta]$. by:

$$\hat{\gamma}(\bar{h}_j) = \frac{1}{2N_j} \sum_{i=1}^{N_j} (\hat{e}(s_i) - \hat{e}(s_i + h))^2, \quad \forall (s_i, s_i + h) : h \in [h_j, h_j + \delta]$$

with $\bar{h}_j$ the average of all N_j h's. A covariogram is modeled by fitting a model to the sample covariogram $\hat{C}(h)$ calculated by:

$$\hat{C}(\bar{h}_j) = \frac{1}{N_j} \sum_{i=1}^{N_j} \hat{e}(s_i)\hat{e}(s_i + h), \quad \forall (s_i, s_i + h) : h \in [h_j, h_j + \delta]$$

The sample cross variogram $\hat{\gamma}_{kl}(h)$ is calculated from sample data by:

$$\hat{\gamma}_{kl}(\bar{h}_j) = \frac{1}{2N_j} \sum_{i=1}^{N_j} (\hat{e}_k(s_i) - \hat{e}_k(s_i + h))(\hat{e}_l(s_i) - \hat{e}_l(s_i + h)),$$
$$\forall (s_i, s_i + h) : h \in [h_j, h_j + \delta]$$

The sample pseudo cross variogram $g_{kl}(h)$ is calculated from sample data by

$$\hat{g}_{kl}(\bar{h}_j) = \frac{1}{2N_j} \sum_{i=1}^{N_j} (\hat{e}_k(s_i) - \hat{e}_l(s_i + h))^2, \quad \forall (s_i, s_i + h) : h \in [h_j, h_j + \delta].$$

The sample cross covariogram $\hat{C}_{kl}(h)$ is calculated by the sample cross covariance

$$\hat{C}_{kl}(\bar{h}_j) = \frac{1}{N_j} \sum_{i=1}^{N_j} \hat{e}_k(s_i)\hat{e}_l(s_i+h), \qquad \forall (s_i, s_i+h) : h \in [h_j, h_j+\delta]$$

Gstat provides calculation of sample variogram, covariogram, cross variogram, pseudo cross variogram and cross covariogram, where width δ, number of intervals j and direction of h can be controlled. When some cross variogram is requested, gstat decides which one should be calculated: the first, 'classic' cross variogram is calculated when the two variables have the same number of observations and identical coordinates and order, in any other case the pseudo cross variogram is calculated, this information is written to the second line of the file with the sample variogram.

Estimation of variogram model parameters

Gstat provides several methods for estimating variogram model parameters. Fitting of a variogram model to the sample variogram is done by iteratively reweighted least squares (WLS, Cressie (1993)), minimizing

$$\sum_{j=1}^{n} w_j(\hat{\gamma}(\bar{h}_j) - \gamma(\bar{h}_j))^2$$

with w_j either equal to N_j or to $\gamma(\bar{h}_j)^{-2}N_j$.

Fixing parameters in a weighted least squares fit can be done by putting an `@` before the range or the sill parameter to fix: e.g. fitting the variogram `1 Sph(@ 0.2)` will only fit the sill parameter (1), fitting `@1 Sph(0.2)` will only fit the range parameter (0.2).

Within gstat, iterative fitting stops when the number of steps exceeds 50 (or the value set by `iter`) or when the fit has converged. A fit is considered as 'converged' when the change in the weighted sum of squares of differences between variogram model and sample variogram becomes less then $10^6\times$ the last value of this sum of squares (this number is controlled with `fit_limit`).

Both gstat and gnuplot fix the fitting weights during iteration. For this reason, when the fitted model strongly differs from the initial (starting) model, another fitting round may converge to a (substantially) different model, because the variogram model, and consequently the weights, changed. Reiterated weighted fitting may very well result in never converging cycles.

Gstat uses Gauss-Newton fitting with (mostly) analytical derivative functions; gnuplot uses Levenberg-Marquardt fitting with numerical derivatives.

Finally, gstat provides REML (restricted maximum likelyhood) estimation of partial sill parameters Kitanidis (1985); Christensen (1993) from sample data: this method is equivalent to a full weighted least squares fitting of the variogram (sill parameters) to the pairwise products of the observations (the covariogram cloud). REML estimation is shown to be equivalent to iterated MIVQUE (minimum variance quadratic unbiased estimation), and to iterated MINQUE (minimum norm quadratic unbiased estimation) with an Euclidean norm Christensen (1993). REML may be slow for moderate to large data sets (more than 100 observations).

A.2 Prediction equations

This appendix assumes some familiarity with the matrix notation introduced in Section 2.7. From the universal kriging equations, ordinary kriging can be derived as the special case where $p = 1$ and F and $f(s_0)$ contain only ones. Ordinary least squares prediction is a special case of uncorrelated weighted least squares prediction and estimation (constant weights).

Using the matrix notation of Section 2.7, the observations $z(s)$ are represented by the model

$$Z(s) = F\beta + e(s), \qquad \mathrm{E}(e(s)) = 0, \qquad \mathrm{Cov}(e(s)) = V \tag{A.1}$$

with $Z(s) = (Z(s_1), ..., Z(s_n))'$, $F = (f_1(s), ..., f_p(s))$ with $f_i(s) = (f_i(s_1), ..., f_i(s_n))'$, and $\beta = (\beta_1, ..., \beta_p)'$. Given this model (i.e., V is known), the best linear unbiased prediction (kriging predictor) of $Z(s_0)$ is

$$\hat{Z}(s_0) = f(s_0)\hat{\beta} + v_0'V^{-1}(z(s) - F\hat{\beta}), \tag{A.2}$$

with $f(s_0) = (f_1(s_0), ..., f_p(s_0))$ and $v_0 = (\mathrm{Cov}(e(s_1), e(s_0)), ..., \mathrm{Cov}(e(s_n), e(s_0)))'$, and where $\hat{\beta}$ is the best linear unbiased estimate of β:

$$\hat{\beta} = (F'V^{-1}F)^{-1}F'V^{-1}z(s) \tag{A.3}$$

The kriging prediction has prediction variance (kriging variance)

$$\begin{aligned}\mathrm{Var}(Z(s_0) - \hat{Z}(s_0)) &= \sigma^2_{Z(s_0)} - v_0'V^{-1}v_0 + \\ &\quad (f(s_0) - v_0'V^{-1}F)(F'V^{-1}F)^{-1}(f(s_0) - v_0'V^{-1}F)'\end{aligned} \tag{A.4}$$

with $\sigma^2_Z(s_0) = \mathrm{Var}(Z(s_0))$. When the trend contains an intercept (which will be true in most cases), it is enough to know generalised covariances, defined as $c - \gamma(h)$, for arbitrary c and with $\gamma(h)$ the variogram of $Z(s)$. Gstat chooses c as the sill of the variogram. When block predictions $\hat{Z}(B_0)$ are required, then for user-defined base functions the values for $f(B_0)$ should be given as input to gstat (in the mask map(s) or in the `X`-columns of the `data()` file), whereas point-to-block and block-to-block covariances (i.e., $\text{Cov}(e(s_i), e(B_0))$ and $\sigma^2_Z(B_0)$) are derived from the point-to-point (generalised) covariances, using Gaussian quadrature Carr and Palmer (1993) or, when either `nblockdiscr` or `area` is specified, using simple integration (regular or user-specified discretization, equally weighted).

Simple kriging

When β is known, simple kriging prediction is obtained. It only involves the prediction of $e(s_0)$:

$$\hat{Z}(s_0) = f(s_0)\beta + v_0 V^{-1}(z(s_0) - f(s_0)\beta)$$

having variance

$$\text{Var}(Z(s_0) - \hat{Z}(s_0)) = \sigma^2_{Z(s_0)} - v_0' V^{-1} v_0$$

Multivariable prediction

When s variables $Z_k(s)$, $k = 1, ..., m$ each follow a linear model $Z_k(s) = F_k\beta_k + e_k(s)$, and the $e_k(s)$ are correlated, then it makes sense to extend the weighted least squares model to allow multivariable prediction. Without loss of generality, assume $m = 2$. When $\mathbf{z}(s) = (z_1(s), z_2(s))'$ and $\mathbf{B} = (\beta_1, \beta_2)'$ are substituted for $z(s)$ and β, and when

$$\mathbf{f}(s_0) = \begin{bmatrix} f^1(s_0) & 0 \\ 0 & f^2(s_0) \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} F_1 & 0 \\ 0 & F_2 \end{bmatrix},$$
$$\mathbf{V} = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}, \quad \mathbf{v_0} = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix}$$

with $f^k(s_0)$ the $f(s_0)$ that corresponds to variable k, with

$$V_{21} = [\text{Cov}(e_2(s_i), e_1(s_j))],$$

$$v_{21} = (\text{Cov}(e_2(s_1), e_1(s_0)), ..., \text{Cov}(e_2(s_n), e_1(s_0)))',$$

and 0 a conforming zero matrix or vector, are substituted for $f(s_0)$, F, V and v_0, then the left-hand sides of both (A.2) and (A.4) yield the multivariable predictions: the left-hand side of (A.2) then becomes the prediction vector

$\hat{\mathbf{z}}(s_0) = (\hat{z}_1(s_0), \hat{z}_2(s_0))'$, and the left-hand side of (A.4) becomes the (2×2) matrix with prediction covariances.

The examples above assume that each variable $Z_k(s)$ has it's own unique parameter vector β_k. It is however possible to define common parameters for multiple variables (with `merge`, section 4.1). It is for instance possible to define a common mean (intercept) (see "standardised ordinary cokriging" in Deutsch and Journel (1992, p. 70), Isaaks and Strivastava (1989, p. 409–416), or Goovaerts (1997, p. 323), or "collocated cokriging" Goovaerts (1997, p. 326)), or to define a common regressor across different variables (cf. Analysis of covariance models, Christensen (1996, p. 193 and excercise 9.1)). In any case, $\mathbf{f}(s_0)$ and $\mathbf{F}$ loose their typical block structure. For instance, with $m = 2$, and the trend of both $Z_1(s)$ and $Z_2(s)$ consist of one single intercept (merging the intercept for both variables) leads to an $\mathbf{F}$ matrix with only one column, filled with ones.

Uncorrelated least squares prediction

When the residuals are not correlated and have unequal variance, i.e. under the model

$$Z(s) = F\beta + e(s), \quad \mathrm{E}(e(s)) = 0, \quad \mathrm{Cov}(e(s)) = \sigma^2 D$$

with D a known diagonal matrix, the uncorrelated least squares estimate for β is obtained by

$$\hat{\beta}_* = (F'D^{-1}F)^{-1}F'D^{-1}z(s)$$

having variance

$$\mathrm{Var}(\beta - \hat{\beta}_*) = (F'D^{-1}F)^{-1}\sigma^2$$

where σ^2 is estimated by

$$s^2 = z(s)'(I - FD^{-1}(F'D^{-1}F)^{-1}F'D^{-1})z(s)/(n-p)$$

Least squares predictions at location s_0 are obtained by

$$\hat{Z}(s_0) = f(s_0)\hat{\beta}_* \tag{A.5}$$

having variance

$$\mathrm{Var}(Z(s_0) - \hat{Z}(s_0)) = (1 + f(s_0)(F'D^{-1}F)^{-1}f(s_0)')\sigma^2 \tag{A.6}$$

or, when block prediction is involved

$$\mathrm{Var}(Z(B_0) - \hat{Z}(B_0)) = f(s_0)(F'D^{-1}F)^{-1}f(s_0)'\sigma^2 \tag{A.7}$$

When variograms are specified, trend prediction (using `method: trend;`) involves the calculation of $f(s_0)\hat{\beta}$, having variance $f(s_0)(F'V^{-1}F)^{-1}f(s_0)'$. When no variograms are specified, trend prediction involves the evaluation of (A.5) and for variances either (A.6) or, for blocks, (A.7). D is by default the identity matrix I, in which case ordinary least squares (OLS) is used, or else it has elements specified by the `V` column of the `data(id)` file concerned. Specifying `V` also results in weighted uncorrelated least squares estimation of residuals in case of variogram modelling.

When common parameters are defined with `merge`, and no variograms are specified, then estimation and prediction under the OLS and WLS models will be done, in which case the constant variance is assumed for the joint (multivariable) residual. In this case, known ratios in variances *between* different variables can still be set using the `V` field of each data variable.

Kriging data with known measurement errors

Known, constant measurement error can be defined in the variogram model. Suppose the variable y has an apparent nugget effect of 1 and a spatially correlated part of `1 exp(10)`, than the variogram model can be written as `1 Nug() + 1 Exp(10)`. This would yield the 'standard' kriging predictions, i.e. exact interpolation. If it is known that 75% of the nugget variance constitues of measurement error, and predictions are required for the measurement error free part of the variable, then the variogram of y can be defined as:

```
variogram(y): 0.75 Err() + 0.25 Nug() + 1 Exp(10)
```

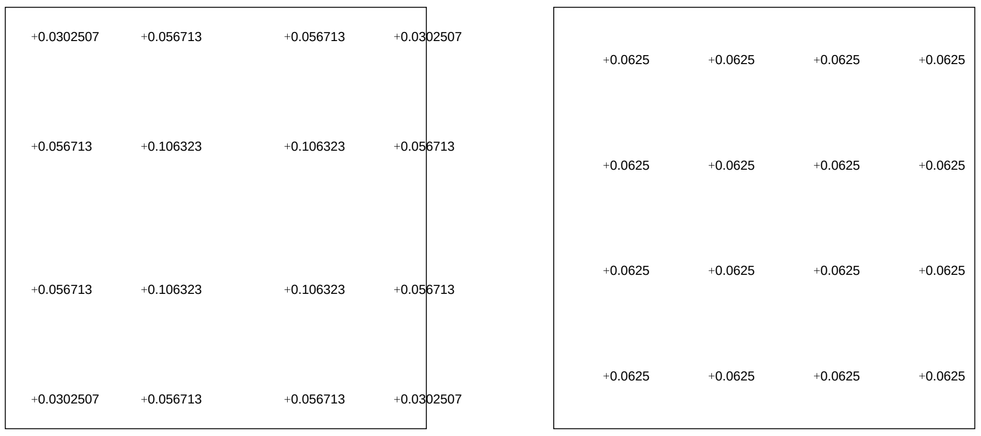
see for more information Cressie (1993).

Known, varying measurement errors can be defined (for data in column files) by specifying the `V` field. When variograms are specified and the goal is prediction or trend prediction, then the covariances $\mathrm{Cov}(e(s_i), e(s_i))$ are taken to be $c - \gamma(h) + \sigma^2_\epsilon(s_i)$, with $\sigma^2_\epsilon(s_i)$ the value of the `V`-field of record i, thus interpreting the variance field (`V`) as a known, location-specific measurement error Delhomme (1978); Pebesma (1996); Cressie (1993). Otherwise formulated: in this case $V + D$ is used as covariance matrix, instead of V only. Putting a constant in the `V` field should yield the same results as specifying this value in the `Err` term of the variogram.

A.3 Change of support: details

The average of a function $f(\cdot)$ over a block (or line or volume) B,

$$f(B) = |B|^{-1} \int_B f(s)ds$$



(a) (b)

Figure A.1: Block discretization: locations s_i (+) and weights w_i using (a) 4×4 Gaussian quadrature, (b) 4×4 regular

with $|B|$ the block area (or lenght or volume) is approximated by:

$$f(B) \approx \sum_{i=1}^{N} w_i f(s_i)$$

with $\sum_{i=1}^{N} w_i = 1$, and s_i the points that discretise the block B and where w_i are the weights for each point s_i.

Rectangular blocks

For rectangular blocks, gstat calculates block averages (for semivariances, (generalised) covariances, coordinate polynomials or inverse distance weighted interpolations) by using Gauss quadrature with 4 points in each block dimension (Fig. A.1, Carr and Palmer (1993)). Regular discretization ($w_i = N^{-1}$) is obtained by setting `nblockdiscr` to the number of points in each direction (section 4.4). Blocks are always centred at prediction locations.

Block-to-block covariances $C(B_1, B_2) = |B_1|^{-1}|B_2|^{-1} \int_{B_1} \int_{B_2} C(x) du dv$ are calculated as

$$C(B_1, B_2) \approx \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} w_i w_j C(s_i, s_j)$$

with s_i discretizing B_1 and s_j discretizing B_2. For block kriging, block-to-block (generalised) covariances $C(B_0, B_0)$ are calculated only once per variogram model. For Gaussian simulation of block averages though, this double sum is recalculated for each pair of (simulated) block averages in a kriging neighbourhood. This takes a while.

Non-rectangular blocks

Mean values for arbitrary shaped, e.g. non-rectangular ‘blocks’ are obtained when the `area:'file', ... ;` command is set (using the `data` syntax, section 4.2) to specify the points that describe the shape. This area should, depending on the dimensions, be centred around the location (0), (0,0), or (0,0,0) in order to obtain predictions centred around the prediction locations. Weights w_i of points discretizing the area are set to N^{-1}, or, if the `V` field is defined for the area, they are set to the values of that field.

If `area` is specified but *no* prediction locations are specified, then the area average of the (points discretizing) `area` itself will be calculated, and written to output. In this case, the area should not (necessarily) be centred around zero, but should be the actual area for which area-average predictions are required.

Base functions block averages

When base functions are used for the trend, gstat assumes that the user-defined base function values at the prediction location *are* block average values: gstat cannot average user-defined base functions over the prediction block (it does so for coordinate polynomial base-functions).

A.4 Latin hypercube sampling

Suppose we want to simulate the outcome of a single continuous random variable Z with known distribution F_Z, to study how the outcome of a model $g(Z)$ depends on the distribution of Z. We could do this by using simple random sampling, i.e. randomly drawing values from F_Z. However, if g is expensive to evaluate, other sampling strategies may be more efficient: it is for instance well known that stratified sampling can characterise the population equally well as simple random sampling with a smaller sample size. Stratified sampling works as follows: (i) the distribution of Z is divided into m segments, (ii) the distribution of n samples over these segments is proportional to the probabilities of Z falling in the segments, and (iii) each sample is drawn from its segment by simple random sampling. Maximal stratification takes place when the number of segments (strata) m equals the number of data n, and when Z has probability m^{-1} of falling in each segment, and this is the most efficient.

When the model depends on two variables, like $h(Z_1, Z_2)$, then values of both Z_1 and Z_2 can be drawn by simple random sampling or by stratified random sampling. The most efficient sample is maximally stratified for both

Z_1 and Z_2 simultaneously. This means that a sample of size n of pairs (z_1, z_2) is marginally $n-$stratified for both Z_1 and Z_2. Such a sample is called a Latin hypercube sample McKay et al. (1979).

When, in a spatial context, multiple Gaussian simulations are created, the simulations are independent in the sense that they are a *random* sample from the ensemble of all realisations that were possible under the model specified: at a specific location x_0 the subsequently realised values of the variable $Z(x_0)$, $(z^1(x_0), ..., z^n(x_0))$ are a simple random sample from the distribution of $Z(x_0)$. (The values of $Z^i(x)$ and $Z^j(x')$, $i \neq j, x \neq x'$ may be *spatially* dependent when both simulations were conditioned on a common data set.) When we want a stratified sample of $Z(x_0)$, then this could simply be drawn when $F_{Z(x_0)}$ were known. However, this is less trivial when $Z^i(x)$ and $Z^i(x')$, $x \neq x'$ still have to obey the prespecified spatial correlation. A procedure for obtaining a Latin hypercube sample in this context (multiple, spatially correlated variables) is given in Stein (1987) and this procedure has been implemented in gstat.

The procedure requires the marginal distribution of $Z(x_0)$. In case of unconditional simulations, this distribution will be constant everywhere, and when simple point kriging is used to obtain the Gaussian simulations, this distribution is $\mathcal{N}(\mu_{sk}, C(0))$ with μ_{sk} the simple kriging mean (`sk_mean`) and $C(0)$ the sill of the variogram used. If for instance 3 variables are defined in a command file, and they have marginal distributions of respectively $\mathcal{N}(4, 3)$, $\mathcal{N}(2, 5)$ and $\mathcal{N}(7, 2.5)$, then this is defined in a command file by the command

```
    marginals:  4, 3, 2, 5, 7, 2.5;
```

If no such command is given, marginal distributions for variable i will be taken as $\mathcal{N}(\mu_{sk,i}, C_i(0))$, a normal distribution with mean $\mu_{sk,i}$ and variance $C_i(0)$ (the sill of the variogram for variable i). If simulations are conditional to known point data, then the marginal distributions are location specific, and are obtained from simple kriging (prediction and prediction variance) of the conditioning data. If, for 2 variables, such marginals reside in the maps `pr1`, `var1`, `pr2` and `var2`, denoting marginal distributions (as maps) $\mathcal{N}$(`pr1`,`var1`) and $\mathcal{N}$(`pr2`, `var2`), then this is defined in a command file as

```
    marginals:  'pr1', 'var1', 'pr2', 'var2';
```

In case of stratified simulation, only the two maps with the (stratified) marginals have to be specified in the `marginals` command. Note that the `marginals` construct limits the definition of location specific marginal distributions (for *conditional* simulation with Latin hypercube sampling) to gridded simulation.

A Latin hypercube sample is constructed from a simple random sample if the command

```
    set lhs=1;
```

is set. The size of the (Latin hypercube) sample is set to 1000 by

```
   set nsim=1000;
```

In order to obtain correct results, the sample size (the number of simulations) must be large: in small samples the spatial correlation may be disturbed by the Latin hypercube procedure Pebesma and Heuvelink (1999).

Appendix B

Using polylines with gstat

(This chapter was written by Konstantin Malakhanov. Names and addresses of contributors to the code are found in the source code file polygon.c)

Often during interpolation one has to take into account boundaries (edges) between data and/or interpolation points. These boundaries can be natural, like rivers or geological faults, or man-made, like legal boundaries. The boundaries should be used at the selection of candidate data points, so only appropriate ones will be used for estimation.

For open boundaries, selected data points and the estimation point have to be at the same side of each boundary. For closed boundaries, data points and the estimation point all have to be either inside or outside of each boundary.

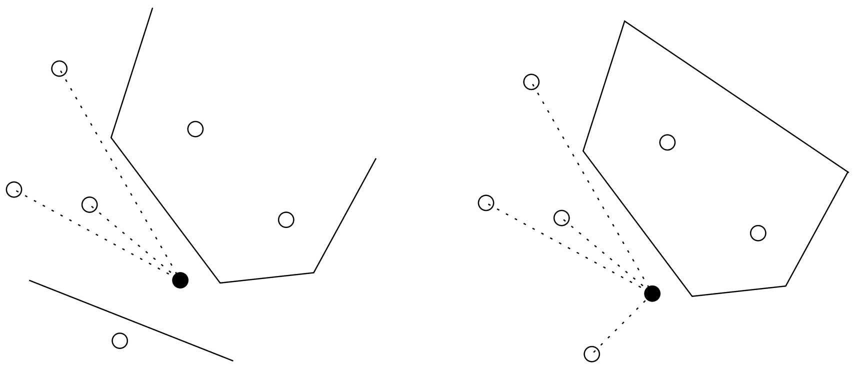


Figure B.1: Most often cases of open and closed boundaries (• estimation point, ∘ data point). Selected data points are connected.

Certainly one can imagine cases of more complicated topology, with open and closed boundaries mixed, or with connected boundaries etc. To see an example how boundaries can be used in estimation process, see Joyce et al. (1997). For general computational geometry questions, see a book of O'Rourke O'Rourke (1998) (with software available at http://cs.smith.edu/~orourke/books/ftp.html) and Computational Geometry FAQ O'Rourke.

To handle the "interpolation with boundaries" in gstat, a new keyword (`edges`) and a new method (`point-in-polygon`) are introduced. `edges` allows to take boundaries into account during estimation. `point-in-polygon` calculates which data points are inside of given polygons. The *point-in-polygon* test is useful, if you want to exclude some data points outside of given boundaries.

B.1 Implementation aspects

First we introduce the following definitions:

polyline - a line of multiple connected straight segments,

polygon - a closed polyline (first and last coordinate of the polyline are equal).

For testing if two points are at the same side of a given boundary, we use the *line-of-sight* test (fig. B.2).

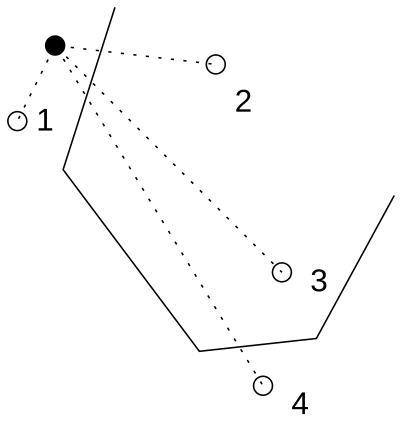


Figure B.2: Line-of-sight test

An (imaginary) segment goes from the estimation point to each data point in question. If the number of segment intersections is even, then both points are supposed to be at the same side of the boundary, if the number of intersections is odd, they are separated by the boundary.

In this example, the line •-1 has 0 intersections with the edge, the lines •-2 and •-3 have only one intersection and the line •-4 has two intersections with the edge. So the data points 1 and 4 are assumed to be at the same side of the edge as the estimation point.

There are some special cases in the test, like:

1. an estimation point or a data point lies on the edge
2. the segment goes exactly through a vertex of the edge (fig. B.3)

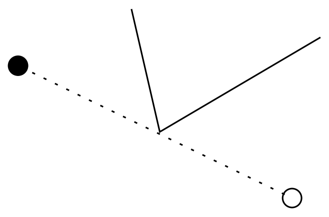


Figure B.3: Special case: the segment goes exactly through a vertex of the edge

In the first case, the edge is completely ignored. In the second case, the intersection of the segment does not count and testing will be continued.

The results of the test depend heavily on the topological connectivity of edges (cf. fig. B.4).

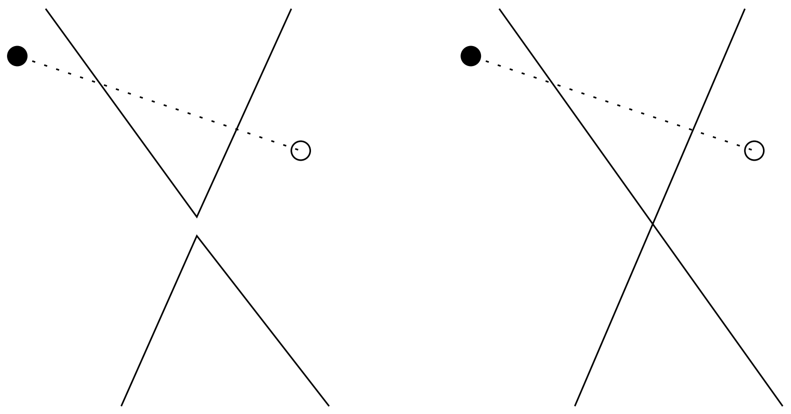


Figure B.4: Different results of line-of-sight test

Here at the left subfigure (boundaries are slightly shifted for better overview) both points are connected, whereas in the right subfigure, the points are disconnected by both line segments.

The *line-of-sight* test does not work for more complicated cases, for instance in case of spiral boundaries (fig. B.5).

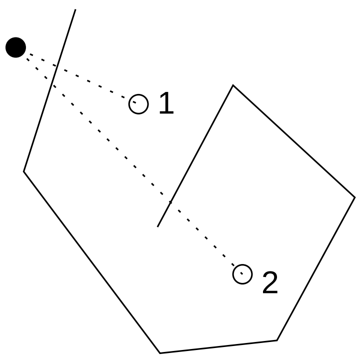


Figure B.5: Line-of-sight test does not work in case of spiral edges

Here the *line-of-sight* test gives wrong results. Point 1 is actually at the same side as the estimation point and point 2 is at another side.

Concerning the question of finding the shortest path between two points (see section B.6), we are looking for the complete solution of this problem. For now, the *line-of-sight* test should sufficiently work in most of the usual cases.

For *point-in-polygon* test of points we have used the InPoly routine, which is a part of the code in O'Rourke (1998).

> InPoly test is written by Joseph O'Rourke, contributions by Min Xu, June 1997.

For a given point it can define, if the point

- lies strictly inside or outside of a polygon,
- is a vertex of a polygon,
- lies exactly at the edge between vertexes.

B.2 Formats available for input

Boundaries can be read in two formats:

E00 format – this is supposed to be ASCII ARC/INFO coverage format. As this format is not officially documented by ESRI, no warranty can be given.

First and second lines of a file are:

```
EXP␣␣␣␣Name_of_coverage
ARC
```

Then every polyline (polygon) has the header

```
I_ok dummy_I dummy_I dummy_I dummy_I dummy_I I_np
```

x-y coordinates follow then, with either one coordinate (two numbers) or two coordinates (four numbers) per line. `I_np` is a number of coordinates in that polyline. A polyline/polygon will be read in, if `I_ok` > 0.

"plot" format – only polylines/polygons data without header.

For every polyline/polygon, the first line gives a number of points. x-y coordinates follow then, with either one coordinate (two numbers) or two coordinates (four numbers) in each line.

Coverage format is automatically recognized by `EXP` as the first word of a file.

B.3 New keywords

1. `edges`. To use like:

   ```
   edges: "file1","file2",...;
   ```

   Given boundary files will be read in and the boundaries will be used during neighborhood search. You can give both polylines and polygons.

2. `point-in-polygon`. To use like:

   ```
   method: point-in-polygon;
   ```

   With given point locations (through `data` statement), gstat will search which polygons the points are in. The output is a list of locations with a file number in the prediction column and a polygon number in the variance column of the output file. For points inside a polygon, this will be numbers counting from 0. Locations which are not inside of any polygon will have NA in both columns.

B.4 Using *point-in-polygon* test

To use *point-in-polygon* test, give through data statement the locations of points you want to test for being inside of given edges.

For this test, polylines do not have to be closed – they are treated as closed anyway. Points on the polyline or coincident with vertices are assumed to be inside of the polygon.

A data point gets the number of the first of given polygons it is in, or NA otherwise. You can parse the output with tools you have at the hand (grep, awk, Perl...) and select points you are interested in.

B.5 Using polylines with interpolation

Edges are used for neighborhood selection after testing for radius/maximum number of data points. The global selection will be changed as well.

Suppose we have an estimation point (to get an interpolated/simulated value at) and a data point. First, all relevant edges in the search neighborhood are found. This works by comparing the bounding boxes of all edges with the box region of the estimation point (fig. B.6).

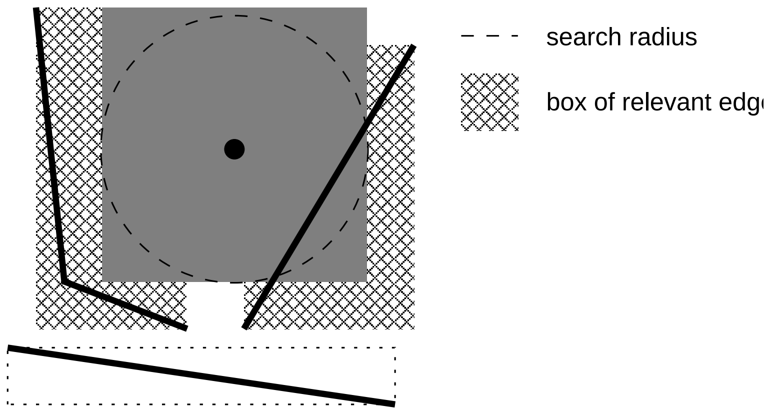


Figure B.6: Test of edges' extents

Then, depending on the type of the polyline (open/closed), the *line-of-sight* test or the *point-in-polygon* test is performed for the estimation point and all data points found so far.

During the testing, if the estimation point is found being on any edge, this edge is skipped for further testing for this estimation point for all data points. If a data point lies on any edge, this edge will not be tested for this data point anymore.

The edges test is repeated for all edges found. A data point will be used, if it passes the test for all edges.

As you can see, all testing is done by brute-force testing. So if you have a lot of edges, with all of them relevant for most of the data points, this will slow down interpolation/simulation a lot. Smarter edges searching is for sure possible, for example, using line quadtrees. Better testing can probably be implemented in connection with finding the shortest path between two points.

B.6 Distance calculation with boundaries

Both variogram calculation and interpolation depend on the calculation of distance between two points. Without edges, this distance is simply an Euclidean one. Taking edges into account, the situation will be more difficult (fig. B.7).

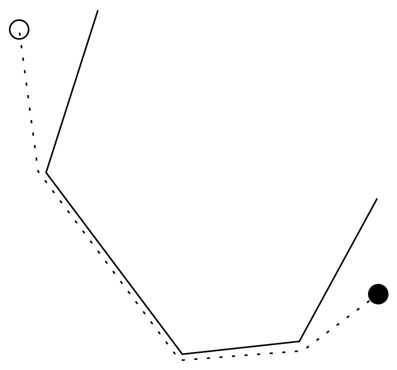


Figure B.7: Indirect path between two points

Depending on the boundary topology, a shortest path could be not so obvious at all.

There is a well-known solution for the problem of finding the shortest path between points separated by some polygons (i.e. building the connection graph and using Dijkstra's algorithm). We were not able to find any ready solution for the case of *open* polylines. So until the solution is found, the distance between two points will be calculated without taking boundaries into account. We suppose that the correct solution will also eliminate the problems of the *line-of-sight* test.

Appendix C

Trouble shooting

C.1 Error messages

C.1.1 From gstat

Errors can occur in the gstat code or during the matrix operations in the meschach matrix library. The cause of and possible solutions to the latter are explained in the second part of this section. Following is a list of the gstat error messages (with exit values).

`variable not set: ... (2)` the named variable should have been set in the command file.

`variable outside valid range: ... (3)` the named variable is outside its valid range (e.g. negative, where it should be positive).

`value not allowed for: ... (4)` the named variable got a value that was not allowed (e.g. outside the valid range, or in contradiction to other settings)

`no file name set: ... (5)` a file name was not set where it should.

`write failed on file '...' (6)` could not write the named file (e.g. no write permission, file system full,...).

`read failed on file '...' (7)` could not read the named file (e.g. file does not exist, no read permission,...).

`cannot read real value from '...' (8)` could not transform the named string into a real number.

`cannot read integer from '...' (9)` could not transform the named string into an integer.

`syntax error: ... (10)` a syntax error occured in the file, at the position pointed to.

`argument option error on '...' (11)` the named command line argument is erroneous.

`domain (math) error on '...' (12)` math domain error

`out of dynamic memory (13)` Memory resources exhausted.
Reduce problem size or increase computer resources (e.g. platform, memory, swap space).

`i/o error: ... (14)` interactive mode cannot be combined with redirected input or output streams.

`no command file (15)` no command file was specified.

`no user interface available (16)`

`error while writing to a pipe (17)`

`error while reading to a pipe (18)`

`operation not allowed in secure mode (19)`

`error in meschach matrix libary (20)` followed by further notification on what happened and hints on how this can be resolved

`general error: ... (-1)` and the next,

`NULL argument in function '...' (1)` should not occur, please report this type of error the author (use command line option -d2).

Other possible errors are:

`error during variogram fit (no exit from user interface)` If you specify for instance a variogram model:

```
variogram(a): 1sph(2) + 1sph(2);
```

then the two models are linearly dependent—fitting their sill will lead to a singularity. Also, if, at a nonlinear fit, the range of a model tends to infinity, the true model may have to be a linear model (having one parameter), but two parameters are being fit for it—they will then be linearly dependent and lead to a singularity during fit. Solution: simplify the variogram model.

C.1.2 From meschach

The two most frequently encountered errors from the meschach matrix library are:

```
   Matrix not positive definite ...
```

or

```
   Singular matrix in function ...
```

Apart from out-of-memory errors (see above) the meschach library terminates program execution when it encounters a matrix singularity.

These two error messages may occur:

- during simulation, when an observation falls *almost* exactly at a simulation location. Solution: increase the value of `zero`
- when two observations occur at identical location occur and `noaverage` or `average`=0 was defined in a `data` definition. Solution: remove the `noaverage`, or add `average`=1
- when a Gaussian variogram is used without a nugget effect. Solution: add a nugget effect.
- when universal kriging is used (`X` and/or `d` defined), and at some stage, usually in a local neighbourhood, one of the covariates (`X`-variables) is constant (and therefore no longer independent from the intercept), or dependent on another `X`-variable. Solution: increase the neighbourhood size and make sure that `X`-variables are *never* dependent.

C.2 Strange results

Very strange values

When two (or more) of the observations are at very close distant but not at the same location, the kriging system may become *ill conditioned* (i.e. unstable). Ill conditioned kriging systems may lead to exceptional answers (unrealistically high or low values) or to error messages. The solution to this is to replace the close observations with one new observation (e.g. their average) or to relocate them on the same location (so that gstat will, by default, replace them by their average).

Checking the kriging matrices for their condition number can be done by setting `cn_max` to an appropriate value. If the estimated condition number of a matrix exceeds this value, a warning message is printed and a missing value is generated.

Negative cokriging prediction variances

If arbitrary coregionalizations are defined (and the command `set nocheck=1;` is set to allow this), than you will probably have encountered the warning:

```
   Warning:  No Intrinsic Correlation or Linear Model of
Coregionalization
```

or

```
   Warning:  Cauchy-Schwartz violation:  ...
```

After the first warning, positive definiteness of the cokriging system cannot be guaranteed anymore and thus cokriging variances may become anything—positive or negative, even if the variograms pass the Cauchy-Schwartz check.

Unrecognised IC or LMC

IC (intrinsic correlation) or LMC (linear model of coregionalization) Journel and Huijbregts (1978); Goovaerts (1997) are two models for a set of variograms and cross variograms that guarantee non-negative prediction variance. Gstat only recognises them when the order in which basic variogram models appear in the variogram definition are identical. E.g.,

```
variogram(a):   1   nug() + 1   sph(2);
variogram(b):   2   nug() + 1   sph(2);
variogram(a,b): 0.5 nug() + 0.8 sph(2);
```

will be recognised as LMC, but

```
variogram(a):   1   nug() + 1  sph(2);
variogram(b):   1   sph(2)+ 2  nug(); # <- changed order
variogram(a,b): 0.5 nug() + 0.8sph(2);
```

will not be recognised by gstat as LMC, although it is one. Both definitions will produce identical output.

Simulation speed

Simulation may be slow. Speeding up simulations can be done by (1) a faster machine, or (2) tuning (reduce) the neighbourhood size, especially the radius, or (3) choosing another simulation program, or (4) modifying the source (let me know!).

C.3 The value of zero

At several places in gstat, a result of a calculation that is very close to zero should be treated as zero. Therefor, the value of `zero` is set to a small positive value, and a quantity a is treated as zero when $|a| < \epsilon$, with ϵ the value of `zero`. This applies to the following cases:

1. when `average` is set, data points are averaged when their separation distance is smaller than ϵ
2. when, in sequential simulation, the absolute value of the kriging variance is smaller than ϵ, it is treated as zero and ignored for simulation (i.e., the kriging prediction is returned as the simulated value).
3. for the calculation of difference between two maps, (-e mapdiff) cell values are said to be different if they differ more than ϵ.
4. for the calculation of point-block or block-block (generalized) covariances, a block discretization point is shifted over distance ϵ when it is closer than ϵ to another (discretizing) point.
5. for the filling of the point-point (generalized) covariance matrix during REML fitting of covariance models, an off-diagonal point-point distances in x, y and z direction will be set to SIGN(a)ϵ when the separation distance between point pairs is smaller then ϵ.

C.4 Debug information

Although the gstat error messages are intended to clarify what went wrong, sometimes more information is needed to solve the problem. Extra information on various subjects can be printed during program execution, by setting the debug level to a specific value, e.g. to 9 by the command

```
set debug = 9;
```

or by setting the equivalent command line option `-d 9` (section 5.2). All debug information ("help information") is written to the screen (stdout) but can be redirected to a log file by the command

```
set logfile = 'gstat.log';
```

or by the command line option `-l gstat.log`. Allowed values for the debug level, and their effect are listed in table C.4.

To combine options, their values are summed. For instance, setting the debug level to 3 invokes both levels 1 and 2; setting it to 1023 would invoke them all. (Note that in certain circumstances, the log file size can become huge.)

`debug`	*output*
`0`	suppres any output except warning and error messages
`1`	normal output (default): short data report, program action and mode, program progress in %, total execution time
`2`	print the value of all global variables, all files read and written, and include source file name and line number in error messages
`4`	print OLS and WLS fit diagnostics
`8`	print all data after reading them
`16`	print the neighbourhood selection for each prediction location
`32`	print (generalised) covariance matrices, design matrices, solutions, kriging weights, etc.
`64`	print variogram fit diagnostics (number of iterations and variogram model in each iteration step) and order relation violations (indicator kriging values before and after order relation correction)
`128`	print warning on forced neighbourhoods (see `force`, section 4.4)
`256`	print, instead of program progress in %, for gridded prediction or simulation the current row and column number, or else the current record number
`512`	print block (or area) discretization data for each prediction location

Table C.1: values for debug and their output

Plotting kriging weights

When gstat is used for kriging prediction and the plot file name is defined, with for instance the command:

```
set plotfile='plot.gp';
```

then kriging weights are printed to the plot file is such a way that they can, for each prediction location, be plotted with gnuplot, using

```
gnuplot plot.gp
```

The variable `plotweights` can be set to express kriging weights using different symbol sizes, its value expresses the range of sizes.

The steps of saving plot commands to file and starting gnuplot may be combined on operating systems that support pipes, using:

```
set plotfile='| gnuplot';
```

assuming that the gnuplot executable is in the current search path.

Appendix D

Grid map and data formats

D.1 PCRaster maps

Input PCRaster maps should be readable as REAL4 maps. This implies that any value scale, and all cell representations except REAL8, are allowed as input maps. Output maps are written as REAL4 scalar maps, except for maps resulting from indicator simulation (UINT1 scalar maps). Old PCRaster (version 1) maps are supported when gstat is compiled with `CSF_V1` defined, and linked to the version 1 csf library. (Support for version 1 may not be maintained in the future.)

D.2 Idrisi data and maps

Idrisi file names should be given without extensions: gstat assumes the extensions (.dvc, .vec, .doc, .img) to be file-type specific. For instance, when a mask is defined as

```
mask: 'maskmap';
```

then gstat assumes the files `maskmap.doc` and `maskmap.img` to be present and in the correct idrisi format.

Data can be read from idrisi point files (extensions `.dvc` and `.vec`). In the `.dvc` file, the field `id type` should be `real`, `file type` should be `ascii` and `object type` should be `point`.

Data or grid maps can be read from idrisi image files (extensions `.doc` and `.img`). In the `.doc` file, the field `data type` should be `real`, `byte` or `integer`, the field `file type` should be `ascii` or `binary`; the fields `colums`, `rows`, `min X`, `max Y` and `resolution` should all be set (`resolution` must be holding the cell size).

D.3 ArcInfo/Arcview grid maps

D.3.1 asciigrid

ArcInfo (or ArcView) Asciigrid maps are single ascii files, starting with a number of header lines, followed by the grid cell values (row-wise, from left-to-right). The header lines ave a field name and a value. Field names are pretty much self-explanatory, they are: `ncols`, `nrows`, `cellsize`, `xllcenter` or `xllcorner`, `yllcenter` or `yllcorner`, and (optional) `nodata_value`.

D.3.2 floatgrid

A floatgrid map `maskmap` consist of two files: one ascii file with the name `maskmap.hdr` that contains the grid map topology (as the asciigrid header), and a binary file named either `maskmap.flt` or `maskmap`, containing the cell values. Specify only the file name without the `.hdr` or `.flt` extension in gstat commands. Field names in the header file are: `ncols`, `nrows`, `cellsize`, `xllcenter` or `xllcorner`, `yllcenter` or `yllcorner`, `byteorder`, and (optional) `nodata_value`. The field `byteorder` should have value `lsbfirst` for byte order of little endian processors (least significant byte first, like INTEL), or else `msbfirst` (HP-PA and the like).

From version 2.1 on, gstat adds by default the `.flt` extension to the binary grids file name to facilitate importing in ArcView (with 3D or spatial analyst).

D.4 ER-Mapper maps

ER-Mapper support was contributed by Steve Joyce, who wrote the following about it on Fri, 2 Jan 1998:

ER-Mapper provides a c-function library for programmers to read and write datafiles in a standard way. Maybe you remember my first version used this library, but linking gstat together with ER-Mapper.lib was just an enormous pain in the ass. It turns out, ER-Mapper raster files are fairly straightforward anyway, with a separate ASCII header and binary data file. So the current version reads and writes the ER-Mapper files directly without using ermapper.lib. This may cause it to fail for future versions of ER-Mapper, but I can live with that.

ER-Mapper raster files can be multi-channel–I check the number of channels on input files and bail out if there is more than one. Maybe sometime we can set up the data specification syntax to include a channel as you talked about before.

ER-Mapper files can specify to skip a number of bytes in the binary file–I don't take care of this, but check the file size consistencey of the binary file.

ER-Mapper header binary files can have a different root name than the header–I don't take care of this, and force them to be the same.

ER-Mapper data types can be signed or unsigned chars, ints (16 or 32 bits), 4-byte real or 8-byte double. I read all formats and cast into 4-byte real for sorage in the gstat gridmap. I always write 4-byte real.

Byte order can be specified in ER-Mapper raster files–I check for it and reorder as necessary.

The ER-Mapper coordinate origin can be at an arbitrary fractional pixel position–I correct it to be the upper left corner of the upper left pixel in the grid.

ER-Mapper coordinates can be either Easting/Northing, Long/Lat, or Raw(X-Y). I do a strict check for 'EN' coordinates, because they match the definition used in gridmaps. RAW coordinates have positive 'y' going down (same as cell coordinates) whereas EN coordinates have positive 'y' going up.

D.5 GMT grid maps

GMT grid maps are basically netcdf files. Gstat only includes GMT support when it is linked with the netcdf library. This library is detected automatically by the configure script (section 5.6), or it is added when configure is invoked as

```
./configure --with-netcdf
```

GMT map support was contributed by Konstantin Malakhanov who wrote about it:

"And last but not least: I write here all limitations of using GMT grids with GSTAT :-

1. GMT grids can be centered at pixels or at nodes. GSTAT grids are centered at pixels, so node-wise GSTAT grids will be converted to pixel-centered. GMT grids from GSTAT are always pixel-centered. Convertation could be made in two ways: either decrease the number of rows and columns by one and set pixel values to mean (or median, or what you like at most) value of 4 nodes (this changes values, but preserves boundaries of grid) or extent grid limits to half cell size to west/east/north/south and use nodes as centers of new pixels (this preserves values, but slightly changes the limits of grids). I implemented

the second way, so if you have GMT node-centered grid as mask, then the extensions of result grid from gstat will be one cell size bigger in X- and Y-directions!

2. GMT grids can have multiplication factor and an add offset for z-values. As GSTAT grid definition does not allow for that, GMT grid values with factor different from 1.0 and/or value offset different from 0.0 will be accordingly transformed during the loading. (Comment: for reasons I cannot understand GMT grids have sometimes factor 0.0 which makes no sense. So factor==0.0 will be treated as 1.0). GSTAT grids in GMT format always have factor== 1.0 and value offset==0.0.

3. GMT system allows for rectangular coordinate system or for geographical projections, but there is no way to detect it from grid itself (in GMT commands , projection is almost always supplied as one of arguments by user). So the way GMT grids are treated is defined by user and not stored in a grid. That means that using GSTAT for grids which are supposed to have longitude/lattitude coordinates WILL give results, but these results are useless as spheroid of Earth is not taken into account and it means by no way that GSTAT can interpolate/simulate over sphere in geographical projections (if one needs such things, take a look at Spherekit at http://www.ncgia.ucsb.edu/pubs/spherekit). So I didn't follow this branch further.

4. GMT grid definition has fields for names of x-,y-,z-units. These fields are ignored at reading and will be set to " " in the result grid.

5. GMT grids can have complex z-values. This is neither checked for nor used!"

D.6 Surfer grid maps

Gstat now supports Surfer ascii (DSAA) grids. Missing values are stored as a value outside the data range (given in the file header). In gstat command files, grid map names should never have an extension (leave the `.grd` out).

Bibliography

J.R. Carr and J.A. Palmer. Revisiting the accurate calculation of block-sample covariances using gauss quadrature. *Mathematical Geology*, 25(5): 507–524, 1993.

R. Christensen. Quadratic covariance estimation and equivalence of predictions. *Mathematical Geology*, 25(5):541–558, 1993.

R. Christensen. *Plane answers to complex questions: the theory of linear models*. Springer-Verlag, New York, second edition, 1996.

N. Cressie. Fitting variogram models by weighted least squares. *Mathematical Geology*, 17(5):563–586, 1985.

N. Cressie. *Statistics for Spatial Data, Revised edition*. John Wiley and Sons, Inc., 1993.

M. Davis. Production of conditional simulation via the lu triangular decomposition of the covariance matrix. *Mathematical Geology*, 19(2):99–107, 1987.

J.P. Delhomme. Kriging in the hydrosciences. *Advances in Water Resources*, 1(5):251–266, 1978.

C. Deutsch and A.G. Journel. *GSLIB: Geostatistical Software Library and User's Guide*. Oxford University Press, New York, 1992.

J.J. Gómez-Hernández and A. G. Journel. Joint sequential simulation of multigaussian fields. In A. Soares, editor, *Geostatistics Troia '92, I.*, pages 85–94, 1993.

P. Goovaerts. *Geostatistics for natural resources evaluation*. Oxford University Press, 1997.

G. Hjaltason and H. Samet. Ranking in spatial databases. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases - 4th Symposium, SSD'95*, number 951 in Lecture Notes in Computer Science, pages 83–95. Springer-Verlag, Berlin, 1995.

E.H. Isaaks and R.M. Strivastava. *An introduction to Applied Geostatistics*. Oxford University Press, 1989.

A.G. Journel and Ch.J. Huijbregts. *Mining Geostatistics*. Academic Press, London, 1978.

Stephen Joyce, Jörgen Wallerman, and Håkan Olson. Edges and raster surfaces – a new mix of data structures for representing forestry information. In *18th ICA/ACI International Cartographic Conference*, Stockholm, Sweden, June 23-27 1997. Available at `http://www.resgeom.slu.se/~fjasj/iCC-97.htm`.

P. Kitanidis. Minimum-variance quadratic estimation of covariances of regionalized variables. *Mathematical Geology*, 17(2):195–208, 1985.

M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

D.E. Myers. Vector conditional simulation. In M. Armstrong (ed.), editor, *Geostatistics*, pages 282–293, 1989.

Joseph O'Rourke. Computational Geometry FAQ. `http://www.exaflop.org/docs/cgafaq`.

Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2 edition, September 1998. Software available at `http://cs.smith.edu/~orourke/books/ftp.html`.

E.J. Pebesma. *Mapping Groundwater Quality in the Netherlands*, volume 199 of *Netherlands Geographical Studies*. Utrecht University, 1996. (Ordering information available at `http://www.geog.uu.nl/ngs/ngs.html`).

E.J. Pebesma and G.B.M. Heuvelink. Latin hypercube sampling of multigaussian random fields. *Technometrics*, 41(4):303–312, 1999.

E.J. Pebesma and C.G. Wesseling. Gstat, a program for geostatistical modelling, prediction and simulation. *Computers and Geosciences*, 24(1):17–31, 1998.

M.L. Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

D.E. Stewart and Z. Leyk. *Meschach: Matrix Computations in C*. Proceedings of the Centre for Mathematics and its Applications, Volume 32. Australian National University, 1994. Online information available at `ftp://thrain.anu.edu.au/pub/meschach`.

W.P.A. Van Deursen and C.G. Wesseling. *The PC-RASTER Package*. Department of Physical Geography, University of Utrecht, 1992. Online information available at `http://www.geog.uu.nl/pcraster.html`.

J.M. Ver Hoef and N. Cressie. Multivariable spatial prediction. *Mathematical Geology*, 25(2):219–240, 1993.

Index