geoR : Package for Geostatistical Data Analysis
An illustrative session

Paulo J. Ribeiro Jr. & Peter J. Diggle

Last update: January 25, 2004

1 Introduction

The package **geoR** provides functions for geostatistical data analysis using the software R. This document illustrates some (but not all!) of the capabilities of the package.

The objective is to familiarise the reader with the **geoR**'s commands for data analysis and show some of the graphical outputs which can be produced.

The commands used here are just illustrative, providing basic examples of the package handling. We did not attempt to perform a definitive analysis of the data-set used throughout the exemples neither to cover all the details of the package capability.

In what follows:

- the R commands are shown in *`slanted typewriter fonts like this`*,
- the corresponding output, if any, is shown in `typewriter fonts like this`.

Typically, default arguments are used for the function calls and the user is encouraged to inspect other arguments of the functions using the `args` and `help` functions. For instance, to see all the arguments for the function `variog` type `args(variog)` and/or `help(variog)`.

The commands shown in this page are also available in the file `http://www.est.ufpr.br/geoR/geoRdoc/geoRintro.R`.

We refer to the **geoR** documentation (`http://www.est.ufpr.br/geoR/geoR.html#help`) for more details on the functions included in the package **geoR**.

2 Starting a Session and Loading Data

2.1 Loading the package

After starting an R session, load **geoR** with the command `library` (or `require`). If the package is loaded correctly a message will be displayed.

```
> library(geoR)
```

If the installation directory for the package is the default location for R packages, type:

```
> library(geoR, lib.loc="PATH_TO_geoR")
```

where '"PATH_TO_geoR"' is the path to the directory where **geoR** was installed.

2.2 Using data

Typically, data are stored as an object (a list) of the class `geodata`. An object of this class contains two obligatory elements: the coordinates of data locations as first element (`$coords`) and the data values as second element (`$data`), which can be a vector or a matrix. Objets of the class `geodata` may have other elements such as covariates and coordinates of the borders of the study area.

We refer to the documentation for the functions `as.geodata` and `read.geodata` for more information on how to import/convert data and on the definitions for the class `"geodata"`. Check `http://www.est.ufpr.br/geoR/importingASCII.html` for information on how to read data from an ASCII (text) file.

There a a few data-sets included in the package distribution. For the examples included in this document we use the data set `s100` included in the **geoR** distribution. To load this data type:

```
> data(s100)
```

The list of all data-sets included in the package is given by `data(package='geoR')`.

3 Exploratory Tools

A quick summary for the `geodata` object can be obtained using a method for `summary` which will return a information on the coordinates and data values.

```
> summary(s100)
```

```
$coords.summary
        Coord.X    Coord.Y
min 0.005638006 0.01091027
max 0.983920544 0.99124979

$distances.summary
        min         max
0.007640962 1.278175109

$data.summary
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.1680  0.2730  1.1050  0.9307  1.6100  2.8680
```

The elements of the list `$covariate`, `$borders` and/or `units.m` will be also summarized if present in the `geodata` object.

3.1 Plotting data locations and values

The function `plot.geodata` shows a 2×2 display with data locations (top plots) and data *versus* coordinates (bottom plots). For an object of the class `geodata` the command `plot(s100)` produce the output shown in Figure **??**.

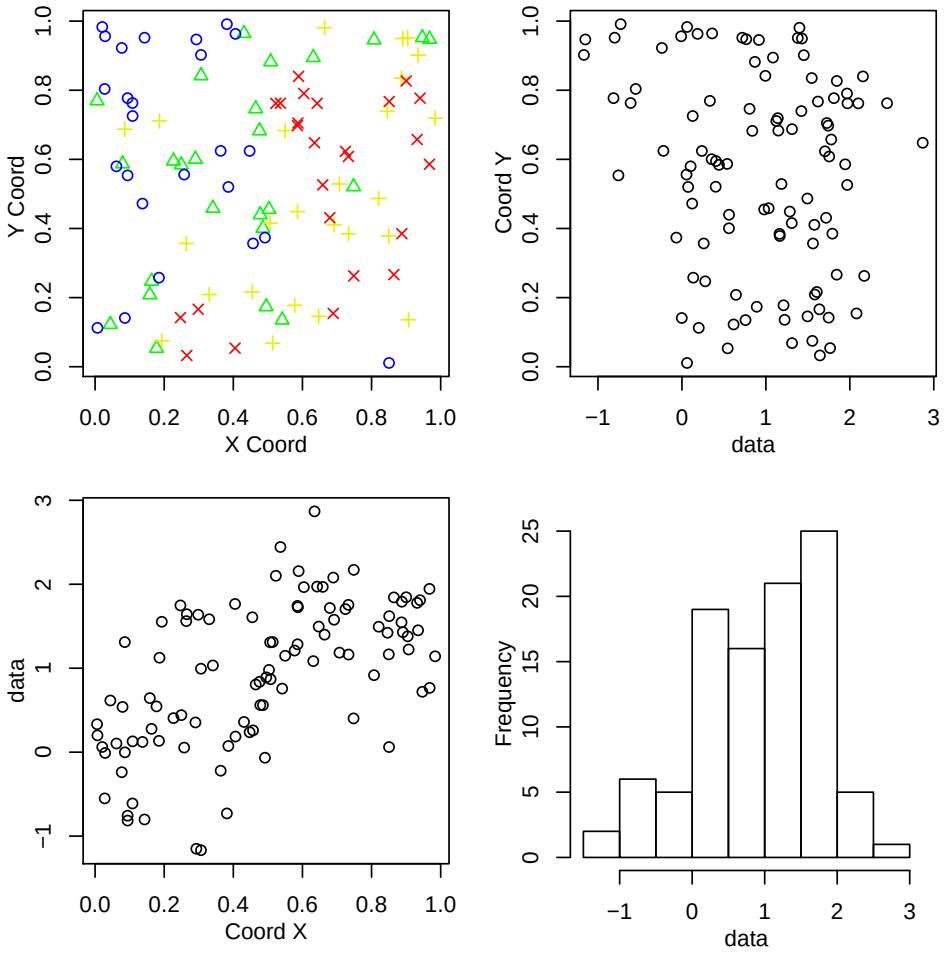

Figure 1: Plot produced by `plot.geodata`.

The function `points.geodata` produces a plot showing the data locations. Alternatively, points indicating the data locations can be added to a current plot. There are options to specify point sizes, patterns and colors, which can be set to be proportional to the data values or specified quantiles. Some examples of graphical outputs are illustrated by the commands below and corresponding plots as shown in Figure **??**.

```
> par(mfrow = c(2, 2))
> points(s100, xlab = "Coord X", ylab = "Coord Y")
> points(s100, xlab = "Coord X", ylab = "Coord Y", pt.divide = "rank.prop")
> points(s100, xlab = "Coord X", ylab = "Coord Y", cex.max = 1.7,
+     col = gray(seq(1, 0.1, l = 100)), pt.divide = "equal")
> points(s100, pt.divide = "quintile", xlab = "Coord X",
+     ylab = "Coord Y")
```

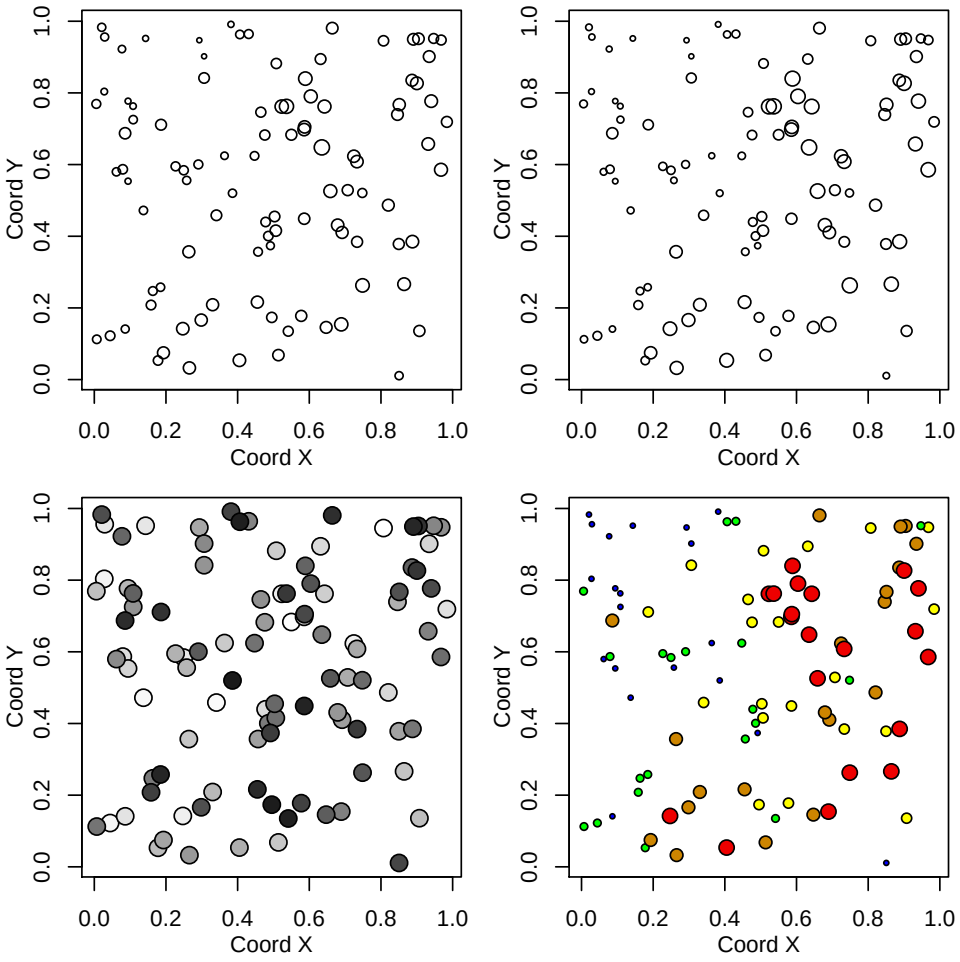

Figure 2: Plot produced by `points.geodata`.

3.2 Empirical variograms

Empirical variograms are calculated using the function `variog`. There are options for the *classical* or *modulus* estimator. Results can be returned as variogram clouds, binned or smoothed variograms. There are methods for `plot` to facilitate the display of the results as shown in Figure **??**.

```
> cloud1 <- variog(s100, option = "cloud", max.dist = 1)
> cloud2 <- variog(s100, option = "cloud", estimator.type = "modulus",
+     max.dist = 1)
> bin1 <- variog(s100, uvec = seq(0, 1, l = 11))
> bin2 <- variog(s100, uvec = seq(0, 1, l = 11), estimator.type = "modulus")
> par(mfrow = c(2, 2))
> plot(cloud1, main = "classical estimator")
> plot(cloud2, main = "modulus estimator")
> plot(bin1, main = "classical estimator")
> plot(bin2, main = "modulus estimator")
```

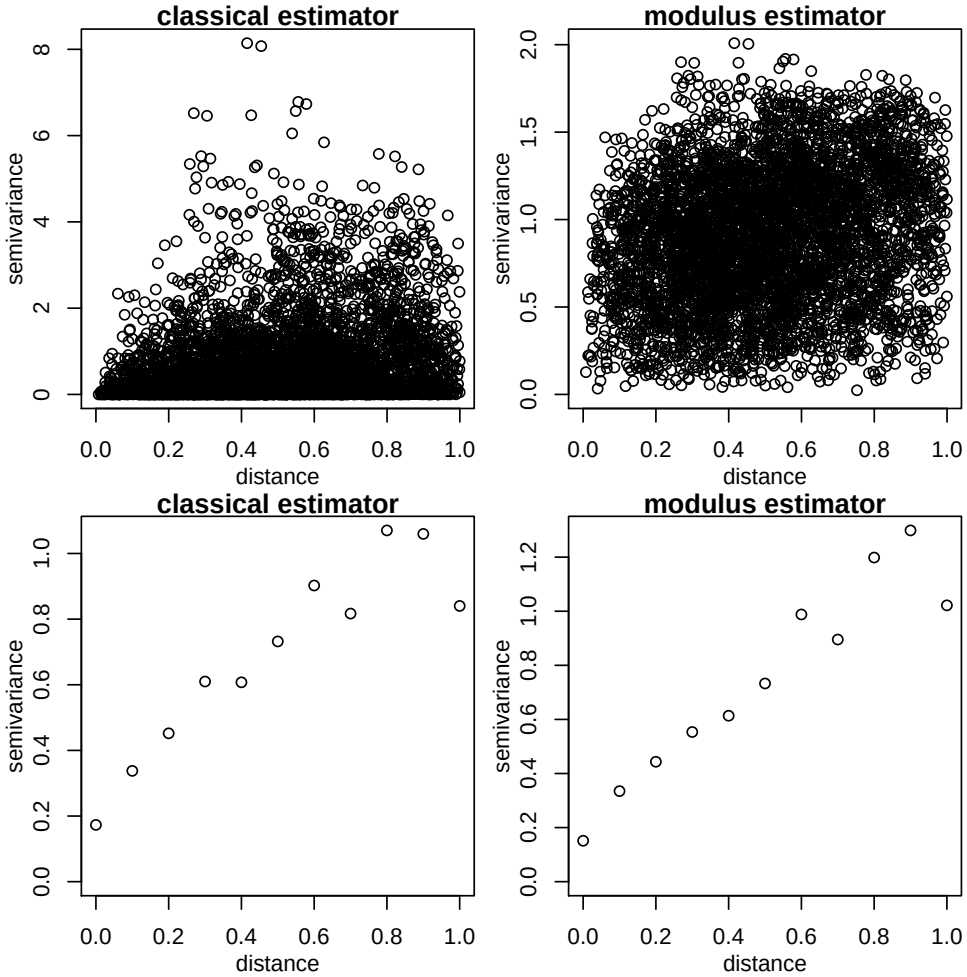

Figure 3: Ploting of `variog` results.

Several results are returned by the function `variog`. The first three are the more important ones and contains the distances, the estimated semivariance and the number of pairs for each bin.

```
> names(bin1)

 [1] "u"                "v"                "n"
 [4] "sd"               "bins.lim"         "ind.bin"
 [7] "var.mark"         "beta.ols"         "output.type"
[10] "max.dist"         "estimator.type"   "n.data"
[13] "lambda"           "trend"            "pairs.min"
[16] "nugget.tolerance" "direction"        "tolerance"
[19] "uvec"             "call"
```

Furthermore, the points of the variogram clouds can be grouped into classes of distances ("bins") and displayed with a box-plot for each bin as shown in Figure **??**. This can be used as an exploratory tool to access variogram results.

```
> bin1 <- variog(s100, uvec = seq(0, 1, l = 11), bin.cloud = T)
> bin2 <- variog(s100, uvec = seq(0, 1, l = 11), estimator.type = "modulus",
+     bin.cloud = T)
> par(mfrow = c(1, 2))
> plot(bin1, bin.cloud = T, main = "classical estimator")
> plot(bin2, bin.cloud = T, main = "modulus estimator")
```

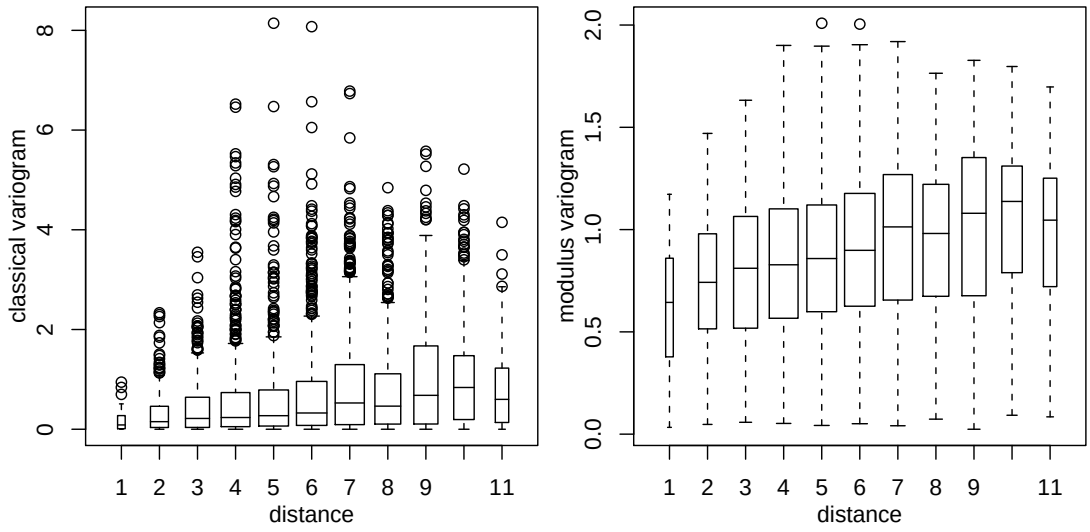

Figure 4: Box-plots for each of the variogram bins.

Directional variograms can also be computed by the function `variog` using the arguments '`direction`' and '`tolerance`'. For example, to compute a variogram for the direction 60 degrees with the default tolerance angle (22.5 degrees) the command would be:

```
> vario60 <- variog(s100, max.dist = 1, direction = pi/3)
```

For a quick computation in four directions we can use the function `variog4` which by default computes variogram for the direction angles 0^o, 45^o, 90^o and 135^o.

```
> vario.4 <- variog4(s100, max.dist = 1)
```

The Figure ?? show the directional variograms obtained with the commands above.

```
> par(mfrow = c(1, 2), mar = c(3, 3, 1.5, 0.5))
> plot(vario60)
> title(main = expression(paste("directional, angle = ",
+     60 * degree)))
> plot(vario.4, lwd = 2)
```

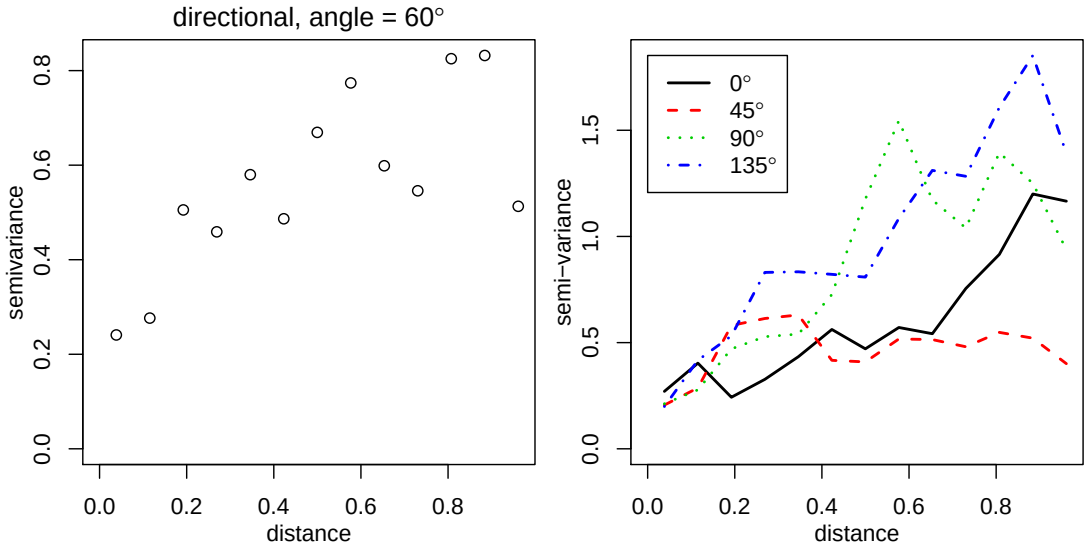

Figure 5: Directional variograms

4 Parameter Estimation

Theoretical and empirical variograms can be plotted and visually compared. For example, the left panel in Figure ?? shows the theoretical variogram model used to simulate the data `s100` and two estimated variograms.

```
> bin1 <- variog(s100, uvec = seq(0, 1, l = 11))
> plot(bin1)
> lines.variomodel(cov.model = "exp", cov.pars = c(1, 0.3),
+     nugget = 0, max.dist = 1, lwd = 3)
> smooth <- variog(s100, option = "smooth", max.dist = 1,
+     n.points = 100, kernel = "normal", band = 0.2)
> lines(smooth, type = "l", lty = 2)
```

```
> legend(0.4, 0.3, c("empirical", "exponential model",
+     "smoothed"), lty = c(1, 1, 2), lwd = c(1, 3, 1))
```

In practice we usually don't know the true parameters which have top be estimated by some method. In the package **geoR** the model parameters can be estimated:

1. *"by eye"*: trying different models over empirical variograms (using the function `lines.variomodel`),
2. *by least squares fit of empirical variograms*: with options for ordinary (OLS) and weighted (WLS) least squares (using the function `variofit`),
3. *by likelihood based methods*: with options for maximum likelihood (ML) and restricted maximum likelihood (REML) (using the function `likfit`),
4. *Bayesian methods*: are also implemented and will be presented in Section 5 (using the function `krige.bayes`).

Fitting "by eye" consists of drawing curves of theoretical variogram functions over an empirical variogram, changing the variogram model and/or its parameters and, at last, choosing one of them. The following commands show how to add a line with a variogram model to a variogram plot. Three different variogram models are used.

```
> plot(variog(s100, max.dist = 1))
> lines.variomodel(cov.model = "exp", cov.pars = c(1, 0.3),
+     nug = 0, max.dist = 1)
> lines.variomodel(cov.model = "mat", cov.pars = c(0.85,
+     0.2), nug = 0.1, kappa = 1, max.dist = 1, lty = 2)
> lines.variomodel(cov.model = "sph", cov.pars = c(0.8,
+     0.8), nug = 0.1, max.dist = 1, lwd = 2)
```

When using the parameter estimation functions `variofit` and `likfit` the nugget effect parameter can either be estimated or set to a fixed value. The same applies for smoothness, anisotropy and transformation parameters. Options for taking trends into account are also included. Trends can be specified as polynomial functions of the coordinates and/or linear functions of given covariates.

An example call to `likfit` is given below. Methods for `print()` and `summary()` have been written to summarize the resulting objects.

```
> ml <- likfit(s100, ini = c(1, 0.5))

---------------------------------------------------------------
likfit: likelihood maximisation using the function optim.
likfit: Use control() to pass additional
         arguments for the maximisation function.
        For further details see documentation for optim.
```

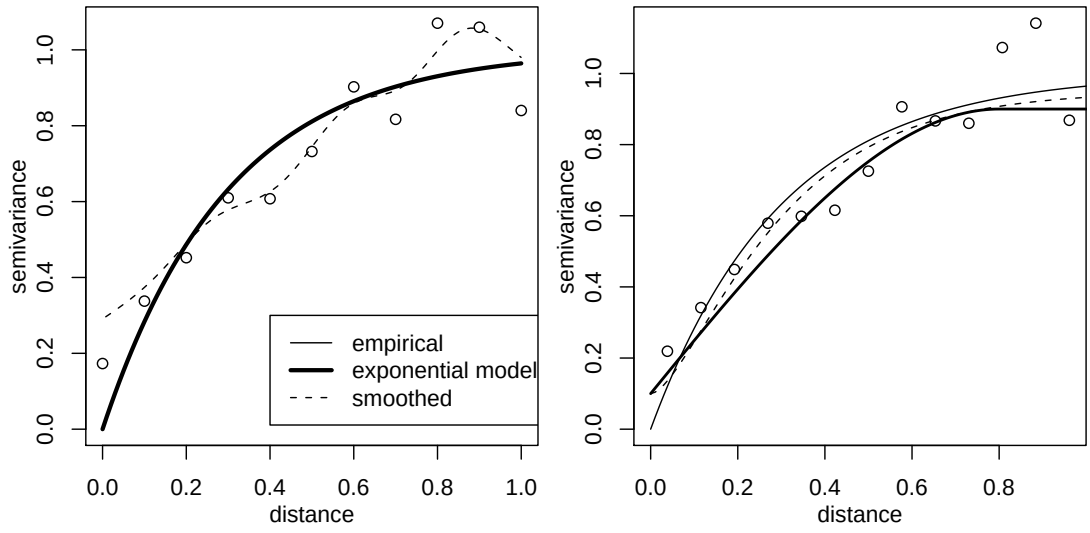

Figure 6: Theoretical variogram curves added to empirical variograms.

```
likfit: It is highly advisable to run this function several
        times with different initial values for the parameters.
likfit: WARNING: This step can be time demanding!
---------------------------------------------------------------
likfit: end of numerical maximisation.

> ml

likfit: estimated model parameters:
    beta    tausq  sigmasq      phi
"0.7766" "0.0000" "0.7517" "0.1827"

likfit: maximised log-likelihood = -83.57

> summary(ml)

Summary of the parameter estimation
-----------------------------------
Estimation method: maximum likelihood

Parameters of the mean component (trend):
  beta
0.7766

Parameters of the spatial component:
   correlation function: exponential
```

```
      (estimated) variance parameter sigmasq (partial sill) =  0.7517
      (estimated) cor. fct. parameter phi (range parameter)  =  0.1827
   anisotropy parameters:
      (fixed) anisotropy angle = 0  ( 0 degrees )
      (fixed) anisotropy ratio = 1

Parameter of the error component:
      (estimated) nugget =  0

Transformation parameter:
      (fixed) Box-Cox parameter = 1 (no transformation)

Maximised Likelihood:
   log.L n.params      AIC      BIC
"-83.57"      "4"  "175.1"  "185.6"

Call:
likfit(geodata = s100, ini.cov.pars = c(1, 0.5))
```

The commands below shows how to fit models by using different methods, with options for fixed or estimated nugget parameter. Notice there are other features not illustrated here such as estimation of trends, anisotropy, smoothness and Box-Cox transformation parameter. Notice in the call above that the functions show some messages while they are running — and we don't want to see them in the following calls. To prevent this we can set the argument `messages = FALSE` at each function call or, to set it globally for all functions use `options()` as follows.

```
> options(geoR.messages = FALSE)
```

- Fitting models with nugget fixed to zero

```
> ml <- likfit(s100, ini = c(1, 0.5), fix.nugget = T)
> reml <- likfit(s100, ini = c(1, 0.5), fix.nugget = T,
+     method = "RML")
> ols <- variofit(bin1, ini = c(1, 0.5), fix.nugget = T,
+     weights = "equal")
> wls <- variofit(bin1, ini = c(1, 0.5), fix.nugget = T)
```

- Fitting models with a fixed value for the nugget

```
> ml.fn <- likfit(s100, ini = c(1, 0.5), fix.nugget = T,
+     nugget = 0.15)
> reml.fn <- likfit(s100, ini = c(1, 0.5), fix.nugget = T,
+     nugget = 0.15, method = "RML")
> ols.fn <- variofit(bin1, ini = c(1, 0.5), fix.nugget = T,
```

```
+     nugget = 0.15, weights = "equal")
> wls.fn <- variofit(bin1, ini = c(1, 0.5), fix.nugget = T,
+     nugget = 0.15)
```

- Fitting models estimated nugget

```
> ml.n <- likfit(s100, ini = c(1, 0.5), nug = 0.5)
> reml.n <- likfit(s100, ini = c(1, 0.5), nug = 0.5, method = "RML")
> ols.n <- variofit(bin1, ini = c(1, 0.5), nugget = 0.5,
+     weights = "equal")
> wls.n <- variofit(bin1, ini = c(1, 0.5), nugget = 0.5)
```

Now, the comands for plotting fitted models against empirical variogram as show in Figure **??** are:

```
> par(mfrow = c(1, 3))
> plot(bin1, main = expression(paste("fixed ", tau^2 ==
+     0)))
> lines(ml, max.dist = 1)
> lines(reml, lwd = 2, max.dist = 1)
> lines(ols, lty = 2, max.dist = 1)
> lines(wls, lty = 2, lwd = 2, max.dist = 1)
> legend(0.5, 0.3, legend = c("ML", "REML", "OLS", "WLS"),
+     lty = c(1, 1, 2, 2), lwd = c(1, 2, 1, 2), cex = 0.7)
> plot(bin1, main = expression(paste("fixed ", tau^2 ==
+     0.15)))
> lines(ml.fn, max.dist = 1)
> lines(reml.fn, lwd = 2, max.dist = 1)
> lines(ols.fn, lty = 2, max.dist = 1)
> lines(wls.fn, lty = 2, lwd = 2, max.dist = 1)
> legend(0.5, 0.3, legend = c("ML", "REML", "OLS", "WLS"),
+     lty = c(1, 1, 2, 2), lwd = c(1, 2, 1, 2), cex = 0.7)
> plot(bin1, main = expression(paste("estimated  ", tau^2)))
> lines(ml.n, max.dist = 1)
> lines(reml.n, lwd = 2, max.dist = 1)
> lines(ols.n, lty = 2, max.dist = 1)
> lines(wls.n, lty = 2, lwd = 2, max.dist = 1)
> legend(0.5, 0.3, legend = c("ML", "REML", "OLS", "WLS"),
+     lty = c(1, 1, 2, 2), lwd = c(1, 2, 1, 2), cex = 0.7)
```

Two kinds of variogram *envelopes* computed by simulation are illustrated in the figure below.

The plot on the left-hand side shows an envelope based on permutations of the data values across the locations, i.e. envelopes built under the assumption of no spatial correlation. The envelopes shown on the right-hand side are based on simulations from a

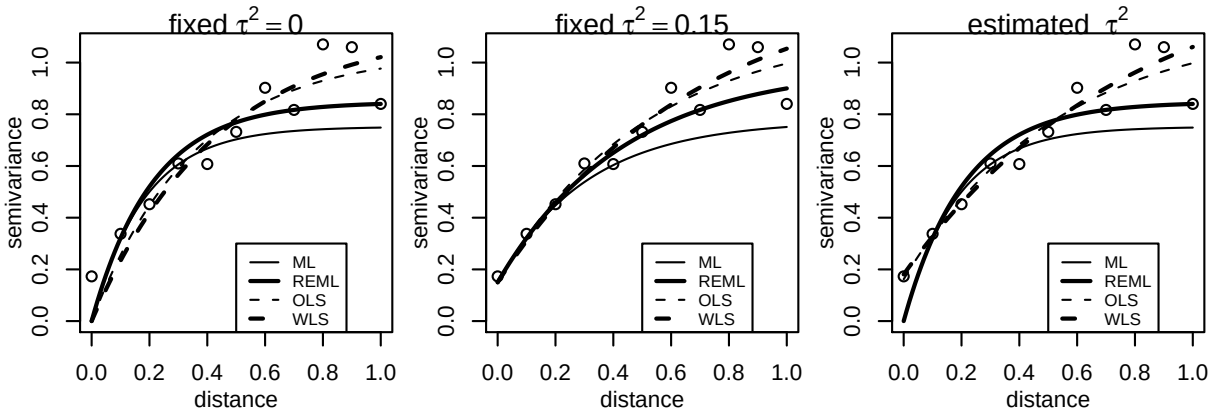

Figure 7: Empirical variogram and models fitted by different methods.

given set of model parameters, in this example the parameter estimates from the WLS variogram fit. This envelope shows the variability of the empirical variogram.

```
> env.mc <- variog.mc.env(s100, obj.var = bin1)
> env.model <- variog.model.env(s100, obj.var = bin1, model = wls)

> par(mfrow = c(1, 2))
> plot(bin1, envelope = env.mc)
> plot(bin1, envelope = env.model)
```

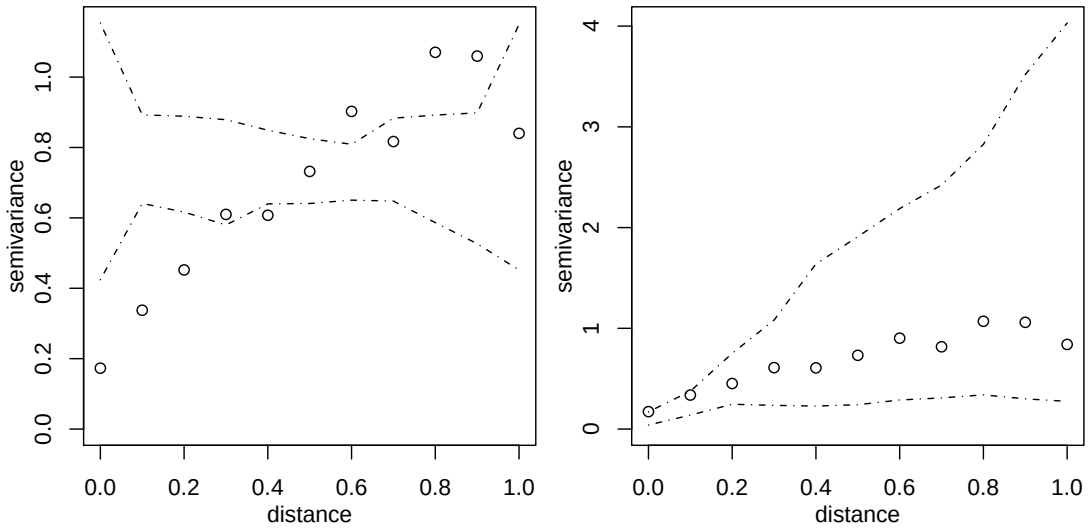

Figure 8: Variogram envelopes.

Profile likelihoods (1-D and 2-D) are computed by the function `proflik`. Here we show the profile likelihoods for the covariance parameters of the model without nugget effect

```
> prof <- proflik(ml, geodata = s100, sill.val = seq(0.48,
+     2, l = 11), range.val = seq(0.1, 0.52, l = 11), uni.only = FALSE)
> par(mfrow = c(1, 3))
> plot(prof, nlevels = 16)
```

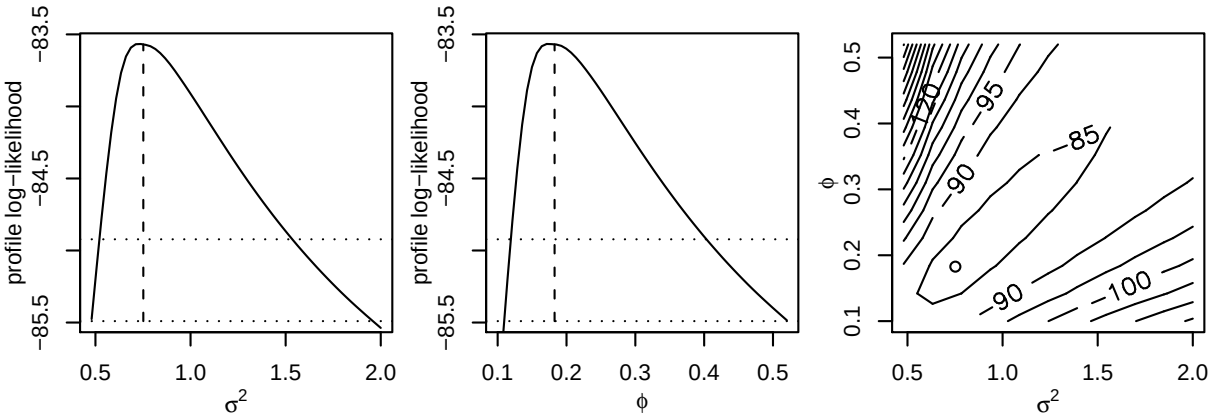

Figure 9: Profile likelihoods for the model parameters

previously fitted by `likfit`.
WARNING: RUNNING THE NEXT COMMAND CAN BE TIME-CONSUMING

5 Cross-Validation

The function `xvalid` performs cross-validation either using the *leaving-one-out* strategy or using a different set of locations provided by the user through the argument '`location.xvalid`'.

For the first strategy, data points are removed one by one and predicted by kriging using the remaining data. The commands below illustrates cross-validation for the models fitted by maximum likelihood and weighted least squares. In the following two calls the model parameters remains the same for the prediction at each location.

```
> xv.ml <- xvalid(s100, model = ml)
> xv.wls <- xvalid(s100, model = wls)
```

Graphical results are shown for the cross-validation results where the leaving-one-out strategy combined with the wls estimates for the parameters was used. Cross-validation residuals are obtained subtracting the observed data minus the predicted value. Standardised residuals are obtained dividing by the square root of the prediction variance ('kriging variance'). By default the 10 plots shown in the Figure **??** are produced but the user can restrict the choice using the function arguments.

```
> par(mfcol = c(5, 2), mar = c(3, 3, 1, 0.5), mgp = c(1.5,
```

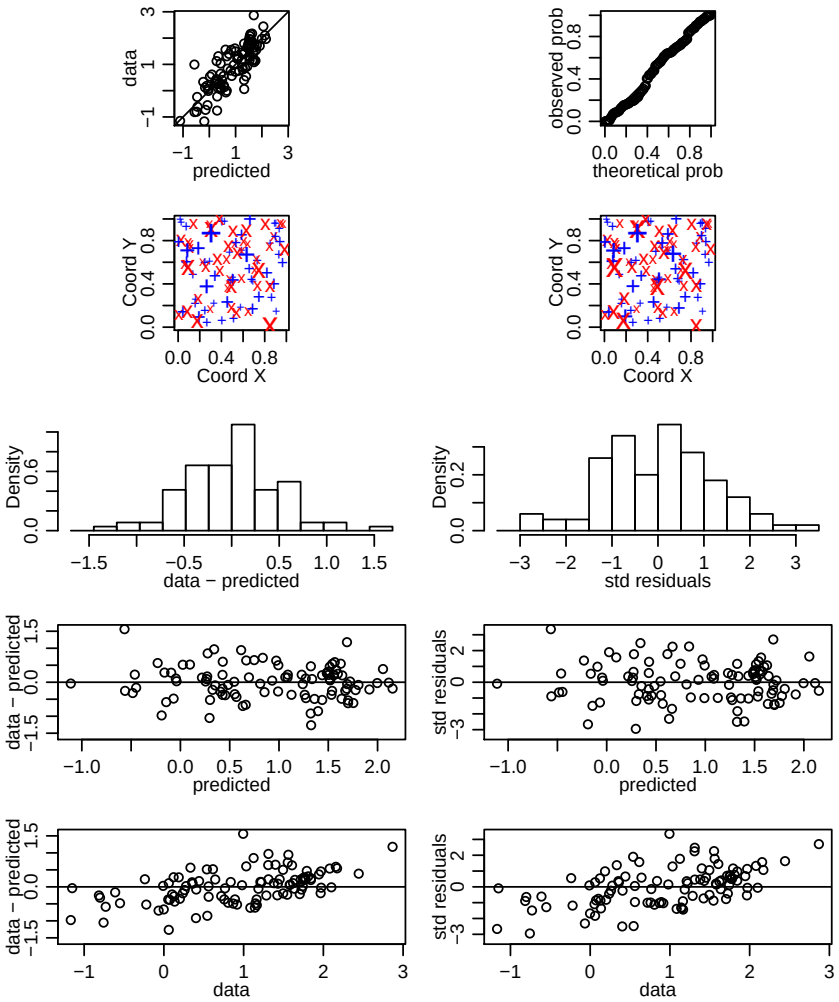

Figure 10: Cross-validations results

```
+     0.7, 0))
> plot(xv.wls)
```

A variation of this method is illustrated by the next two calls where the model parameters are re-estimated each time a point is removed from the data-set.
WARNING: RUNNING THE NEXT COMMAND CAN BE TIME-CONSUMING

```
> xvR.ml <- xvalid(s100, model = ml, reest = TRUE)
> xvR.wls <- xvalid(s100, model = wls, reest = TRUE, variog.obj = bin1)
```

6 Spatial Interpolation

Conventional geostatistical spatial interpolation (*kriging*) can be performed with options for:

1. *Simple kriging*
2. *Ordinary kriging*
3. *Trend (universal) kriging*
4. *External trend kriging*

There are additional options for Box-Cox transformation (and back transformation of the results) and anisotropic models. Simulations can be drawn from the resulting predictive distributions if requested.

As a first example consider the prediction at four locations labeled *1, 2, 3, 4* and indicated in the figure below.

```
> plot(s100$coords, xlim = c(0, 1.2), ylim = c(0, 1.2),
+     xlab = "Coord X", ylab = "Coord Y")
> loci <- matrix(c(0.2, 0.6, 0.2, 1.1, 0.2, 0.3, 1, 1.1),
+     ncol = 2)
> text(loci, as.character(1:4), col = "red")
> polygon(x = c(0, 1, 1, 0), y = c(0, 0, 1, 1), lty = 2)
```

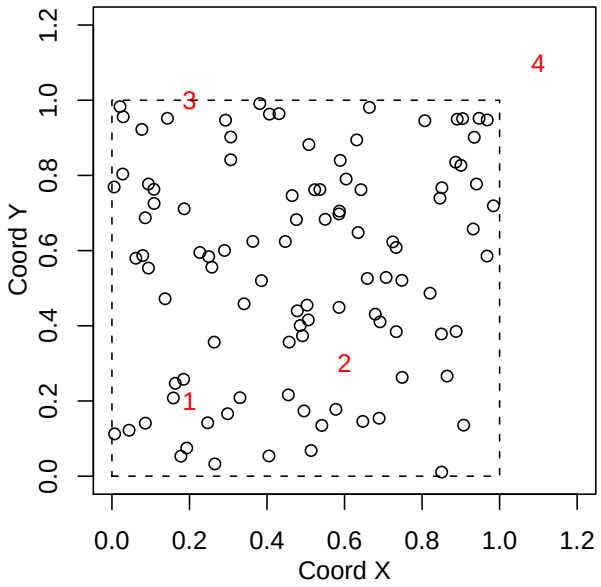

Figure 11: Data locations and points to be predicted

The command to perform *ordinary kriging* using the parameters estimated by weighted least squares with nugget fixed to zero would be:

```
> kc4 <- krige.conv(s100, locations = loci, krige = krige.control(obj.m = wls))
```

The output is a list including the predicted values (`kc4$predict`) and the kriging variances (`kc4$krige.var`).

Consider now a second example. The goal is to perform prediction on a grid covering the area and to display the results. Again, we use ordinary kriging. The commands commands below defines a grid of locations and performs the prediction at those locations.

```
> pred.grid <- expand.grid(seq(0, 1, l = 51), seq(0, 1,
+     l = 51))
> kc <- krige.conv(s100, loc = pred.grid, krige = krige.control(obj.m = ml))
```

A method for the function `image` can be used for displaying predicted values as shown in the next Figure, as well as other prediction results returned by `krige.conv`.

```
> image(kc, loc = pred.grid, col = gray(seq(1, 0.1, l = 30)),
+     xlab = "Coord X", ylab = "Coord Y")
```

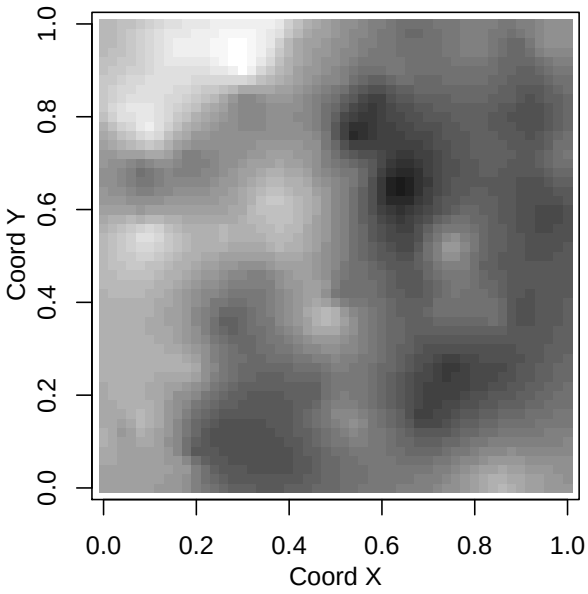

Figure 12: Map of the kriging estimates

7 BAYESIAN ANALYSIS

Bayesian analysis for Gaussian models is implemented by the function `krige.bayes`. It can be performed for different "degrees of uncertainty", meaning that model parameters can be treated as fixed or random.

As an example consider a model without nugget and including uncertainty in the mean, sill and range parameters. Prediction at the four locations indicated above is performed by typing a command like:

WARNING: RUNNING THE NEXT COMMAND CAN BE TIME-CONSUMING

```
> bsp4 <- krige.bayes(s100, loc = loci, prior = prior.control(phi.discrete = se
+     5, 1 = 101), phi.prior = "rec"), output = output.control(n.post = 5000))
```

Histograms showing posterior distribution for the model parameters can be plotted by typing: Using summaries of these posterior distributions (means, medians or modes) we

```
> par(mfrow = c(1, 3), mar = c(3, 3, 1, 0.5), mgp = c(2,
+     1, 0))
> hist(bsp4$posterior$sample$beta, main = "", xlab = expression(beta),
+     prob = T)
> hist(bsp4$posterior$sample$sigmasq, main = "", xlab = expression(sigma^2),
+     prob = T)
> hist(bsp4$posterior$sample$phi, main = "", xlab = expression(phi),
+     prob = T)
```

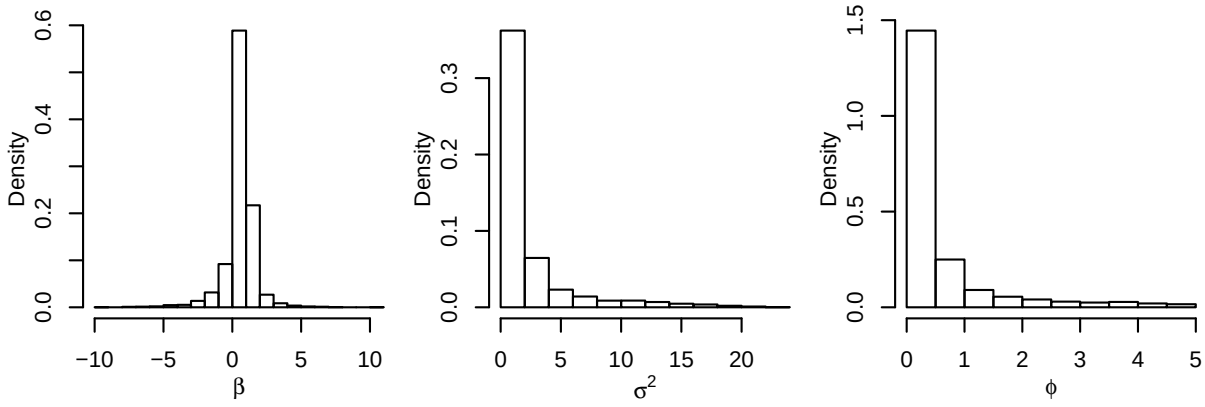

Figure 13: Histograms of samples from posterior distribution

can check the "estimated Bayesian variograms" against the empirical variogram, as shown in the next figure. Notice that it is also possible to compare these estimates with other fitted variograms such as the ones computed in Section 3.

The next figure shows predictive distributions at the four selected locations. Dashed lines show Gaussian distributions with mean and variance given by results of ordinary kriging obtained in Section 4. The full lines correspond to the Bayesian prediction. The plot shows results of density estimation using samples from the predictive distributions.

```
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+     kpx <- seq(kc4$pred[i] - 3 * sqrt(kc4$krige.var[i]),
+         kc4$pred[i] + 3 * sqrt(kc4$krige.var[i]), l = 100)
+     kpy <- dnorm(kpx, mean = kc4$pred[i], sd = sqrt(kc4$krige.var[i]))
+     bp <- density(bsp4$predic$sim[i, ])
+     rx <- range(c(kpx, bp$x))
```

```
> plot(bin1, ylim = c(0, 1.5))
> lines(bsp4, max.dist = 1.2, summ = mean)
> lines(bsp4, max.dist = 1.2, summ = median, lty = 2)
> lines(bsp4, max.dist = 1.2, summ = "mode", post = "par",
+     lwd = 2, lty = 2)
> legend(0.25, 0.4, legend = c("variogram posterior mean",
+     "variogram posterior median", "parameters posterior mode"),
+     lty = c(1, 2, 2), lwd = c(1, 1, 2), cex = 0.8)
```

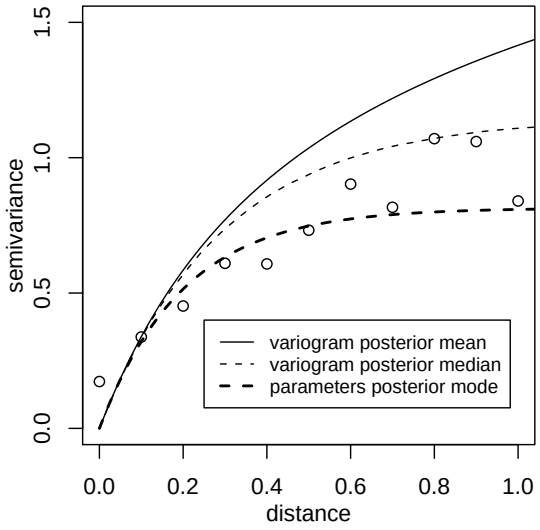

Figure 14: Variogram models based on the posterior distributions

```
+     ry <- range(c(kpy, bp$y))
+     plot(cbind(rx, ry), type = "n", xlab = paste("Location",
+         i), ylab = "density", xlim = c(-4, 4), ylim = c(0,
+         1.1))
+     lines(kpx, kpy, lty = 2)
+     lines(bp)
+ }
```

Consider now, under the same model assumptions, obtaining simulations from the predictive distributions on a grid of points covering the area. The commands to define the grid and perform Bayesian prediction are:

```
> pred.grid <- expand.grid(seq(0, 1, l = 31), seq(0, 1,
+     l = 31))
> bsp <- krige.bayes(s100, loc = pred.grid, prior = prior.control(phi.discrete
+     5, l = 51)), output = output.control(n.predictive = 2))
```

Maps with the summaries and simulations of the predictive distribution can be plotted as follows.

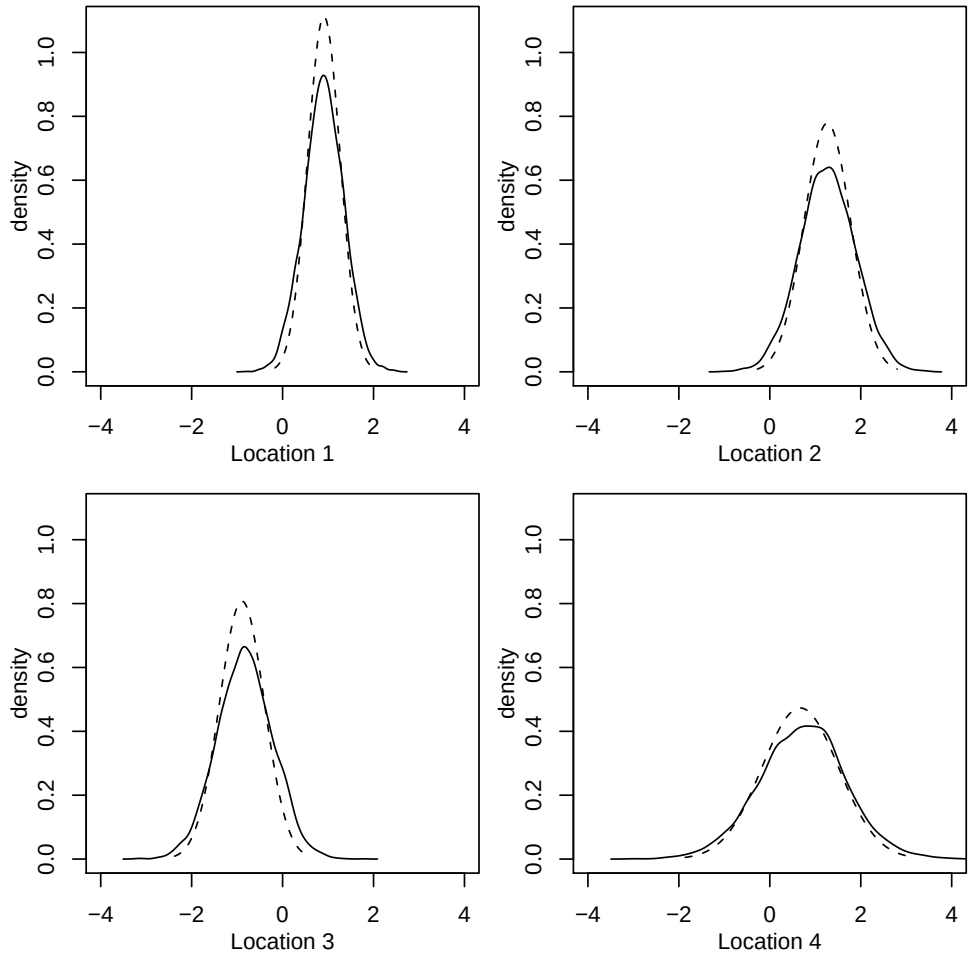

Figure 15: Predictive distributions at the four selected locations

```
> par(mfrow = c(2, 2), mar = c(3, 3, 1, 0.5), mgp = c(1.5,
+     0.7, 0))
> image(bsp, loc = pred.grid, main = "predicted", col = gray(seq(1,
+     0.1, l = 30)))
> image(bsp, val = "variance", loc = pred.grid, main = "prediction variance",
+     col = gray(seq(1, 0.1, l = 30)))
> image(bsp, val = "simulation", number.col = 1, loc = pred.grid,
+     main = "a simulation from\nthe predictive distribution",
+     col = gray(seq(1, 0.1, l = 30)))
> image(bsp, val = "simulation", number.col = 2, loc = pred.grid,
+     main = "another simulation from \n the predictive distribution",
+     col = gray(seq(1, 0.1, l = 30)))
```

Note: Further examples for the function `krige.bayes` are given in the file `http://www.est.ufpr.br/geoR/geoRdoc/examples.krige.bayes.R"` .

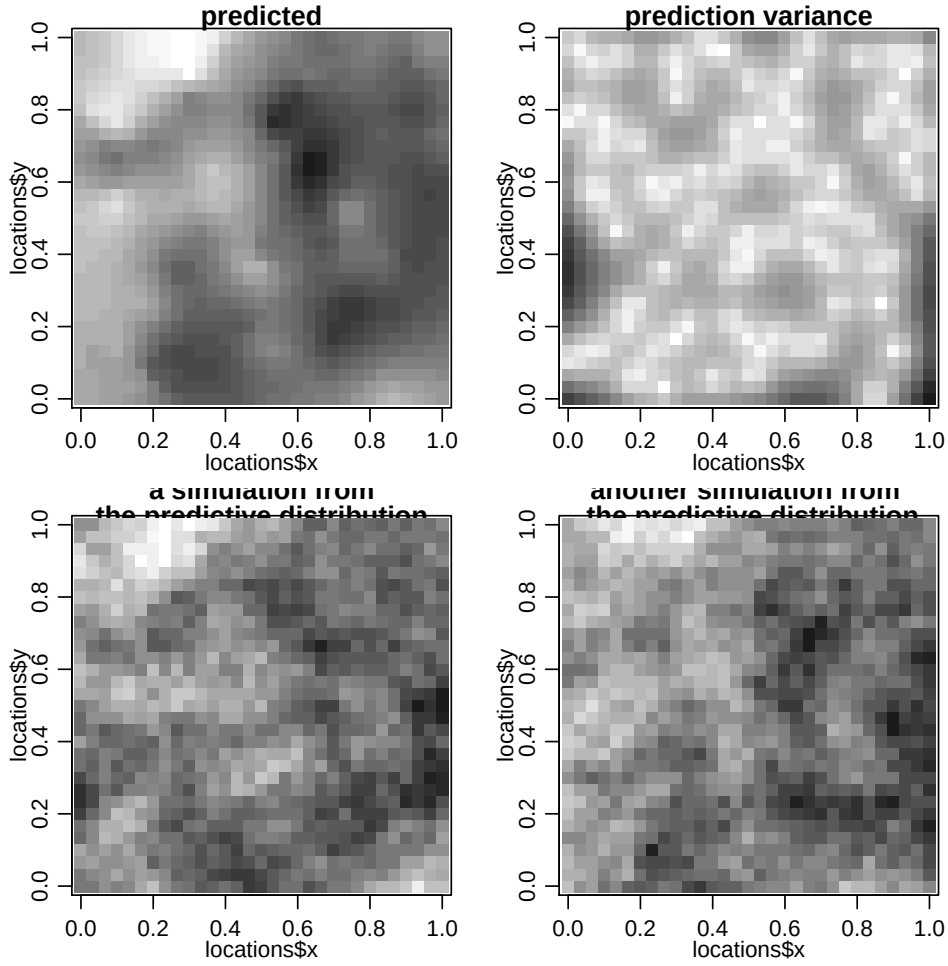

Figure 16: Maps obtained from the predictive distribution

8 Simulation of Gaussian Random Fields

The function `grf` generates simulations of Gaussian random fields on regular or irregular sets of locations. Some of its functionality is illustrated by the next commands.

NOTE: we recommend the package **RandomFields** (`http://cran.r-project.org/src/contrib/PACKAGES.html#RandomFields`) for a more comprehensive implementation for simulation of Gaussian Random Fields.

9 Citing geoR

The function `cite.geoR` shows information on how to cite geoR in publications, and we remember the function `citation` shows how to cite R.

```
> cite.geoR()
```

```
> par(mfrow = c(1, 2))
> sim1 <- grf(100, cov.pars = c(1, 0.25))
> points.geodata(sim1, main = "simulated locations and values")
> plot(sim1, max.dist = 1, main = "true and empirical variograms")
```

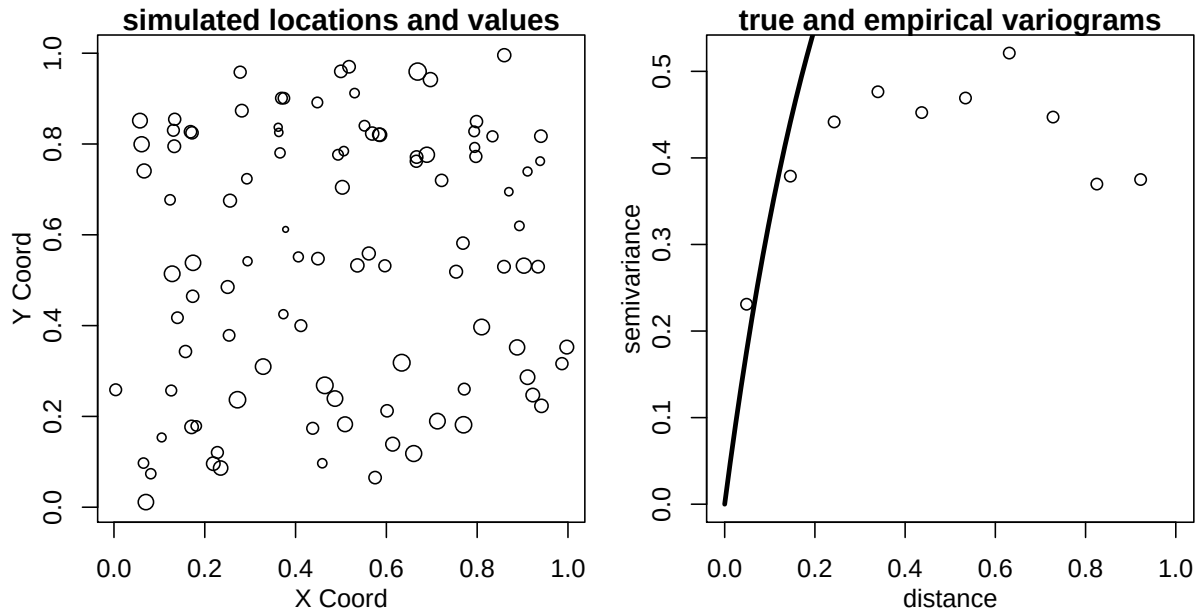

Figure 17: Simulation results produced with the function `grf`.

```
To cite geoR in publications, use

  RIBEIRO Jr., P.J. & DIGGLE, P.J. (2001) geoR: A package for
  geostatistical analysis. R-NEWS, Vol 1, No 2, 15-18. ISSN
  1609-3631.

Please cite geoR when using it for data analysis!

A BibTeX entry for LaTeX users is

  @Article{,
     title          = {{geoR}: a package for geostatistical analysis},
     author        = {Ribeiro Jr., P.J. and Diggle, P.J.},
     journal       = {R-NEWS},
     year           = {2001},
     volume          = {1},
     number          = {2},
     pages           = {15--18},
```

```
> par(mfrow = c(1, 2))
> sim2 <- grf(441, grid = "reg", cov.pars = c(1, 0.25))
> image(sim2, main = "a small-ish simulation", col = gray(seq(1,
+     0.1, l = 30)))
> sim3 <- grf(40401, grid = "reg", cov.pars = c(10, 0.2),
+     met = "circ")

 mtot=262144

> image(sim3, main = "simulation on a fine grid", col = gray(seq(1,
+     0.1, l = 30)))
```

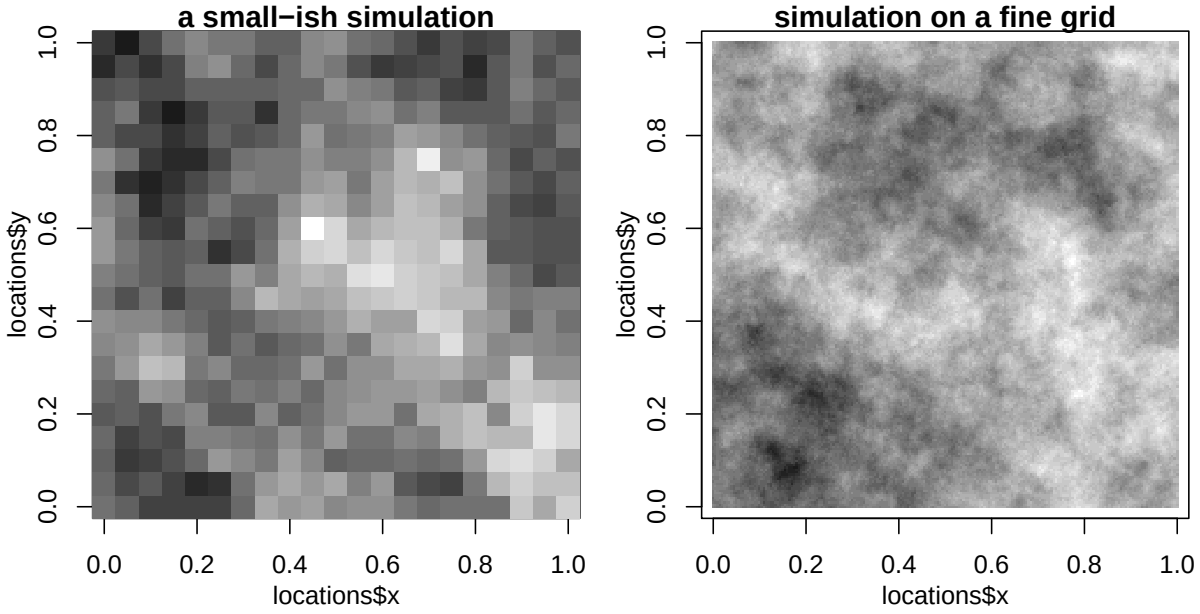

Figure 18: Simulations with different resolutions

```
    issn          = {1609-3631},
    url           = {http://cran.R-project.org/doc/Rnews}
  }

> citation()

To cite R in publications, use

  R Development Core Team (2004). R: A language and
  environment for statistical computing. R Foundation for
  Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3,
  URL http://www.R-project.org.
```

```
We have invested a lot of effort in creating R, please cite
it when using it for data analysis.

A BibTeX entry for LaTeX users is

  @Manual{,
     title        = {R: A language and environment for
                     statistical computing},
     author       = {{R Development Core Team}},
     organization = {R Foundation for Statistical Computing},
     address      = {Vienna, Austria},
     year         = 2004,
     note         = {ISBN 3-900051-00-3},
     url          = {http://www.R-project.org}
   }
```