detecZAC

DOCUMENTATION

R functions for the
detection of Zones of Abrupt Change
for spatial data

Edith Gabriel

16th October 2006

Acknowledgements

Most of this work was made possible by the Institut National de la Recherche Agronomique who funded a PhD grant on this subject. D. Allard and J-N. Bacro are co-authors of the method implemented here[1].
I thanks D. Allard who helped in programming.

---

1. The reference for the implemented method is Allard D., Gabriel E. and Bacro J-N. (2005), available at `www.maths.lancs.ac.uk/~gabriel/papers/AllardGabrielBacro.pdf`

2. `R` is a free software environment for statistical computing and graphics. `R` is available for free from `http://www.r-project.org`.

Contents

1 Introduction

`detecZAC` is a set of programs for detecting the zones where a spatially sampled variable, say Z, presents sharp variations. Such zones are called Zones of Abrupt Change (ZACs).

The method is based on the statistical properties of the estimated gradient of Z. The local gradient is first interpolated using kriging. Then, we test whether the estimated local gradient is above some critical threshold $t_{1-\alpha}$ computed under the null hypothesis of a constant mean. The locations where the local test is rejected define the potential ZACs, which are then tested globally.

The method requires the covariance function of Z, which in practice needs to be estimated. Because the method is run under the null hypothesis of absence of ZAC, in the presence of a discontinuity the variance of Z is overestimated leading to a loss of power of the method. To solve this difficulty, ZACs are obtained by a non-automatic iterative procedure. At each step of the procedure, the covariance function, the level α and the ZACs are estimated. Convergence is reached when the set of ZACs remains unchanged.

This method is specifically designed to detect local abrupt changes, it neither allows to detect smooth transitions and global trends, nor to be used as a classification method.

This is an instruction manual for downloading, installing and running `detecZAC`. It assumes that the reader has some knowledge of the method. Otherwise, for a detailed description of the implemented method, please read Allard D., Gabriel E. and Bacro J-N. (2005), available at

`www.maths.lancs.ac.uk/~gabriel/papers/AllardGabrielBacro.pdf`

To run `detecZAC`, you need:

- a `R` version $\geq$ 1.9.0,
- the `R` packages `Splancs` and `geoR`,
- a Fortran compiler.

2 Installing detecZAC

2.1 Download detecZAC

detecZAC is available for download from

www.maths.lancs.ac.uk/~gabriel

From there you can download the R source code and the Fortran functions. You can either compile manually Fortran functions or download the .so file (Linux/Unix users) or .dll file (Windows users). This file is not warranty so you may have to compile the .f.

2.2 Installation procedure

2.2.1 Linux/Unix users

1. Download the R source code "detecZAC.txt" from the website.
2. Create a directory for the Fortran functions, *e.g.*
   (a) Open a Terminal.
   (b) mkdir /home/.../MyFortran
3. Download the file "zac.so" in the directory created in 2,
   or, download and compile the Fortran function "zac.f" in the directory created in 2, *e.g.*
   (a) Open a Terminal.
   (b) Go to the directory containing the Fortran functions cd /home/.../MyFortran
   (c) Compile the function: R CMD SHLIB zac.f
   This creates a file named "zac.so". This file must be contained in the directory created in 2.
4. Run R.
5. Source the R code, *e.g.* source("/home/.../detecZAC.txt").
6. Load Splancs and geoR packages, *e.g.* library(splancs).
7. Define a path, named "myfortranpath", where the Fortran functions can be found, *e.g.* myfortranpath <- "/home/.../MyFortran/".

You are now ready to use the different programs contained in detecZAC.txt.

2.2.2 Windows users

1. Download the `R` source code "`detecZACwin.txt`" from the website.
2. Create a directory for the Fortran functions, *e.g.* named 'Myfortran'.
3. Download the file "`zac.dll`" in the directory created in 2,

   or download the Fortran function "`zac.f`" in the directory created in 2 and compile it to obtain the executable file "`zac.dll`". This file must be contained in the directory created in 2.
4. Run `R`.
5. Source the `R` code, *e.g.* `source("C:\\...\\detecZACwin.txt")`.
6. Load `Splancs` and `geoR` packages.
7. Define a path, named "`myfortranpath`", where the Fortran functions can be found, *e.g.* `myfortranpath <- "C:\\...\\MyFortran\\"`.

You are now ready to use the different programs contained in `deteczac.txt`.

3 Using detecZAC

3.1 Applying the method

ZACs are obtained by the non-automatic iterative procedure:

$\texttt{ZAC}^{(0)} \leftarrow \emptyset$; $k \leftarrow 0$; convergence <- F;
until (convergence) {
$k \leftarrow k+1$;
estimate $C_Z^{(k)}(h)$ using all pairs $\{Z(x_i), Z(x_j)\}$ such as $[x_i; x_j] \cap \texttt{ZAC}^{(k-1)} = \emptyset$;
find $\alpha^{(k)}$;
estimate $\texttt{ZAC}^{(k)}$ using $(C_Z^{(k)}(h); \alpha^{(k)})$
If ($\texttt{ZAC}^{(k)} = \texttt{ZAC}^{(k-1)}$) convergence <- T }

The covariance function is estimated using the variogram function, which is widely used in the geostatistical literature. For a stationary random field $Z(\mathbf{x})$, the variogram

$$\gamma(\mathbf{h}) = \frac{1}{2}\mathbb{E}\left[\{Z(\mathbf{x}) - Z(\mathbf{x}+\mathbf{h})\}^2\right]$$

is related to the covariance function:

$$\gamma(\mathbf{h}) = C_Z(\mathbf{0}) - C_Z(\mathbf{h}).$$

I chose to estimate the variogram by a (possibly weighted) least square fit of the experimental variogram (Cressie, 1993)[1]:

$$\hat{\gamma}(\mathbf{h}) = \frac{1}{2n_{\mathbf{h}}} \sum_{\mathbf{x}-\mathbf{x}'\sim\mathbf{h}} \left\{Z(\mathbf{x}) - Z(\mathbf{x}')\right\}^2,$$

where $n_{\mathbf{h}}$ is the number of pairs involved in the sum. This estimation is implemented in the package geoR.

Through this section, the iterative procedure is illustrated on an example. The file detecZAC.txt further contains:

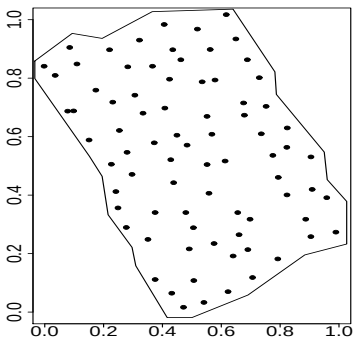
- ExData, a 3-columns data-frame containing the x,y-coordinates and the value of the sample z,
- Poly, a matrix containing the x, y-coordinates of the vertices of the polygon boundary of the study domain.

```
## Visualisation of the sample

x <- ExData$x
y <- ExData$y
z <- ExData$z

plot(x,y,xlim=c(0,1),ylim=c(0,1),pch=19,
     xlab="",ylab="")

polygon(Poly,lwd=2)
```



[1] Cressie (1993) Statistics for Spatial Data. New York: Wiley.

First step of the iterative procedure

We first estimate a global variogram, *i.e.* using all pairs of samples.

```
## visualisation of the variogram

geoD <- as.geodata(as.data.frame(cbind(x,y,z)))
v1 <- variog(geoD,max.dist=0.7,uvec=seq(0,0.7,length=12))
plot(v1$u,v1$v,ylim=c(0,max(v1$v)),xlim=c(0,max(v1$u)),xlab="",ylab="",cex=2,
     cex.axis=2)

## global estimation of the variogram

M <- variofit(v1,c(3000,0.04),fix.nugget=T,cov.model="exponential")
V.step1 <- as.vario.param(model="exponential",range=M$cov.pars[2],
                          sill=M$cov.pars[1])
```

Using these parameters, we estimate the local level α and a first set of ZACs.
Note that computing the level α is long (around one hour in this example). In Section 3.3, I propose an approximation of this level that is instantaneous to get.

```
## estimation of the level alpha

xygrid <- make.grid(60,60,poly=Poly)
alpha.step1 <- eval.alpha(x, y, xygrid, vario.param=V.step1, alphapval=0.05,
                          alphalim=c(0.00005,0.05), ni=100)

## detection and plot of ZACs

zac.step1 <- deteczac(x, y, z, vario.param=V.step1, nx=60, ny=60, poly=Poly,
                      alpha=alpha.step1, alphapval=0.05)

plot.zac(zac.step1)
```

kth step of the iterative procedure

In the presence of ZACs, the variogram is re-estimated by discarding all pairs $\{Z(\mathbf{x}_i), Z(\mathbf{x}_j)\}$ for which the segment $[\mathbf{x}_i, \mathbf{x}_j]$ intersects a ZAC. If the range of the variogram changes, the level α must be re-estimated. ZACs are then re-estimated using the new parameters.

```
## re-estimation of the variogram, e.g. here at the fourth step

v4 <- vario.zac(zac.step3, dist.seq = seq(0,0.7,length=12))
M <- variofit(v4,c(3000,0.2),fix.nugget=T,cov.model="exponential")
V.step4 <- as.vario.param(model="exponential",range=M$cov.pars[2],
                          sill=M$cov.pars[1])

## estimation of the level alpha

alpha.step4 <- eval.alpha(x, y, zac.step1$xygrid, vario.param=V.step4,
                          alphalim = c(0.00005,0.05), alphapval=0.05, ni=100)

## ZACs detection

zac.step4 <- deteczac(x, y, z, vario.param=V.step4, nx=60, ny=60, poly=Poly,
                      alpha=alpha.step4, alphapval=0.05)

## Comparison with the previous step

par(mfrow=c(1,2))
plot.zac(zac.step3)
plot.zac(zac.step4)
```

Convergence

The procedure is iterated until convergence occurs. Convergence is reached when the set of ZACs remains unchanged. Whilst a proof of the convergence does not exist, our results suggest that convergence occurs in less than 5 iterations. This can be explained by the fact that most of the discarded pairs are eliminated at the first re-estimation of the variogram; the parameters of the estimated covariance function are then almost identical in the subsequent iterations.
Note that if convergence occurs at the kth step, the parameters of the estimated covariance function at steps $k-1$ and k are not necessarily the same.

In our example, an exponential covariance was considered at each step. Table 1 contains the estimation of the parameters (range and sill) and Figure 1 illustrates the detected ZACs. These results show that the procedure converges at the third step (the detected ZACs being the same at step 3 and step 4).

Step	$\hat{b}$	$\hat{\sigma}^2$	α	s	ns	p-value	size
1	0.087	3522	$8.8\,10^{-4}$	1	4	$[6.4\,10^{-3}; 8.5\,10^{-1}]$	$[3.1\,10^{-4}; 5.6\,10^{-1}]$
2	0.130	3570	$1.5\,10^{-3}$	2	1	$[2.3\,10^{-4}; 9.3\,10^{-1}]$	$[3.7\,10^{-3}; 1.7\,10^{-2}]$
3	0.134	3144	$1.5\,10^{-3}$	3	0	$[2.2\,10^{-5}; 1.9\,10^{-2}]$	$[6.2\,10^{-3}; 2.2\,10^{-2}]$
4	0.154	3184	$9.9\,10^{-4}$	3	0	$[2.4\,10^{-5}; 1.6\,10^{-2}]$	$[6.2\,10^{-3}; 2.2\,10^{-2}]$

Table 1: *Parameters estimation (range $\hat{b}$ and sill $\hat{\sigma}^2$) of the exponential covariance functions and summary of the detected potential ZACs (number of significant and non-significant ZACs, s and ns respectively, range of the p-values and range of the size).*

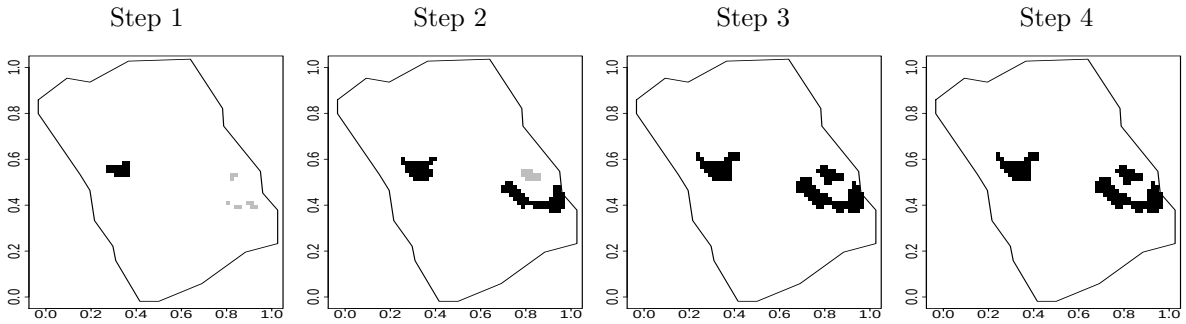


Figure 1: *Non-significant (grey) and significant (black) ZACs detected at the different steps of the iterative procedure.*

3.2 Interpreting the output

In practice, the method is run on a grid superimposed on the domain. On each grid node $\mathbf{x}_p$, the gradient $\partial Z(\mathbf{x}_p)$, the matrix $\mathrm{Var}\left(\partial Z(\mathbf{x}_p)\right)$ and the field $T(\mathbf{x}_p) = \partial Z(\mathbf{x}_p)' \mathrm{Var}\left(\partial Z(\mathbf{x}_p)\right) \partial Z(\mathbf{x}_p)$ are computed. Under the null hypothesis of absence of ZAC, the field $T(\cdot)$ has a chi-square distribution with two degrees of freedom. Then for the local level of confidence, α, the set of grid nodes whose statistic $T(\mathbf{x}_p)$ is above $t_{1-\alpha}$, the $(1-\alpha)$-quantile of the $\chi^2(2)$ distribution, define the potential ZACs.
We have a pointwise test for each point $\mathbf{x}$ of $\mathcal{D}$. These local tests are then aggregated in a global one in order to test globally the existence of a ZAC. For each potential ZAC, the p-value associated to the global test is then computed by using the distribution its size.

The function '`deteczac`' estimates the Zones of Abrupt Change at a level α for a given set of estimated parameters of the covariance function, *i.e.* it performs the local test of detection at each grid node and tests the significance of the detected ZACs.
This function returns a list which contains in particular:

- `zac`: matrix with zero values where there is no ZAC, 1 where there are non-significant potential ZACs, 2 where there are significant potential ZACs and `NaN` outside $\mathcal{D}$,
- `p.value`: vector of p-values associated to the test of significance of the detected ZACs. If the p-value is below the global level of confidence η, fixed by the user, the potential ZAC is considered as significant and defines a ZAC.
- `Smax`: vector of areas of the detected potential ZACs.

The detected ZACs can be plotted using the function '`plot.zac`'. The plot illustrates both the results of the local tests (colored pixels) and the global test (different colors for the significant and non-significant potential ZACs). This is illustrated on Figure 1, where the colored pixels represent the points $\mathbf{x}$ such that $T(\mathbf{x}) \geq t_{1-\alpha}$, grey pixels indicate non-significant potential ZACs and black pixels significant potential ZACs.

The width of the detected ZACs depends on the local sample density. Indeed, in densely sampled zones the gradient is estimated with more precision than in a less sampled zone. This leads to a better ability to delineate a (potential) discontinuity.
The estimated gradient in densely sampled zones may also be artificially large and may lead to statistically significant ZACs that are not physically meaningful and usually small. This is

also the case for outliers and the method could thus be used to detect them. In an unsampled zone, conversely, the estimated gradient is very low and almost constant, thus leading to the impossibility of estimating a ZAC. In view to resolving whether the absence of ZAC is due to a lack of sample points or to a stationary variable, I assessed the local power of the detection test[2]. The corresponding set of functions associated to the assessment of the power will be soon added to this software.

3.3 Improving computation time

Detecting ZACs is long because the local level of confidence α is long to compute. I propose a way for approximating α, that needs less computation. It is based on the integral range, A, of the covariance function $C_Z(\mathbf{h})$ of the variable Z.
The integral range (Lantuéjoul, 1991)[3] is the quantity:

$$A = \int_{\mathbb{R}^2} \frac{C_Z(\mathbf{h})}{C_Z(\mathbf{0})}\,\mathrm{d}\mathbf{h}.$$

Let us denote `D` the area of $\mathcal{D}$ and $\bar{Z}(\mathcal{D})$ the spatial average of $Z(\cdot)$ on $\mathcal{D}$. If $A \neq 0$ and `D`$>>A$, according to $\mathrm{Var}\big(\bar{Z}(\mathcal{D})\big) \approx \sigma^2/N$, with $N = \mathtt{D}/A$. Then, denoting η the global level of confidence, we have:

$$\eta = \mathbb{P}\Big[\cup_{\mathbf{x}\in\mathcal{D}}\big\{T(\mathbf{x}) \geq t_{1-\alpha}\big\}\Big] \approx 1 - \Big(\mathbb{P}\big[T(\mathbf{x}) \leq t_{1-\alpha}\big]\Big)^N,$$

leading to the following approximation for α, denoted α_G:

$$\alpha_G = 1 - (1-\eta)^{1/N}.$$

Using α_G provides similar results than using the correct level α (estimated by Monte-Carlo simulations) only if the discretization of the interpolation grid is sufficiently fine. You must be carefull in using it. Indeed, when the interpolation grid is not sufficiently fine, this approximation overestimates the level α and hence you could find ZACs which are actually non-existent.

In practice, the command to evaluate the level α (*e.g.* at the fourth step):

```
## estimation of the level alpha

alpha.step4 <- eval.alpha(x, y, zac.step1$xygrid, vario.param=V.step4,
                          alphalim = c(0.00005,0.05), alphapval=0.05, ni=100)
```

must be replaced by:

```
## approximation of the level alpha

alpha.step4 <- alphaG(vario.param=V.step4, alphapval=0.05, D=areapl(Poly))
```

I applied the iterative procedure to our exemple using α_G. The procedure still converges at the third iteration. At this step, the estimated parameters are: $\hat{b} = 0.154$ and $\sigma^2 = 3071$, which are

[2]Gabriel, E. and Allard D. (2006) Evaluating the sampling pattern when detecting Zones of Abrupt change. *Environmental and Ecological Statistics*, to appear

[3]Lantuéjoul, C. (1991) Ergodicity and integral range. *Journal of Microscopy*, **161**, 387–403.

close to the ones estimated when considering the estimation of α (that is also the case for all steps). The values of α_G obtained at the different steps are given in Table 2 (second row). The approximation provides greater values of α than its estimation. Figure 2 illustrates the detected

	Step 1	Step 2	Step 3	Step 4
estimation	0.00088	0.00151	0.00152	0.00099
approximation	0.00349	0.00741	0.01100	0.01001

Table 2: *Estimation and approximation of α.*

ZACs at the convergence using either the estimation of α (Figure 2*a*) or its approximation (Figure 2*b*). ZACs are detected at the same locations in both cases. The consequence of an overestimation of α is that ZACs are larger.

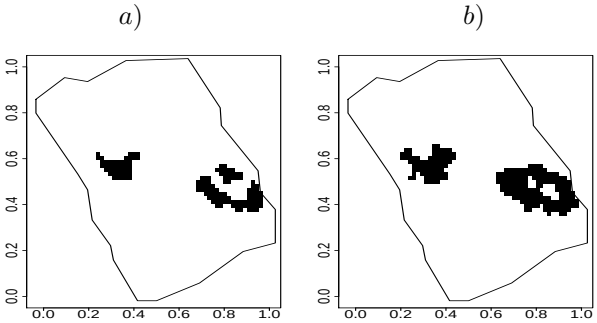


Figure 2: *ZACs detected at the convergence when using: a) the estimation of α, b) its approximation.*

Considering α_G instead of α can often be used as a first approximation for detecting ZACs. But you must check its values and do not use it if it appears too large (it is unreasonable to use it when it is greater than 0.05, because in that case the mathematical porpoerties on which the method is based do no longer hold). I do not advice to use α_G when the estimated range parameter is large, because in this case α_G will be too much larger than α.

3.4 Permitting a nugget effect in the variogram estimation

If the covariance function $C_Z(\mathbf{h})$ has a discontinuity at the origin (the so called nugget effect in the geostatistical literature), the kriging interpolator of $Z(\mathbf{x})$ is discontinuous at sample locations, but it is still continuous at any other location. The theory is thus applicable if there is some proportion of nugget effect. It is however not applicable if there is a pure nugget effect, because in this case the gradient of the kriging interpolator is null for all $\mathbf{x}$ in $\mathcal{D}$.

Let us go back to the example and consider the variogram $\gamma(\mathbf{h}) = c_0 + c_1 \{1 - \exp(-||\mathbf{h}||/b)\}$, where c_0 denotes the nugget effect and $\sigma^2 = c_0 + c_1$ is the sill. At the first iteration, the estimated parameters are $\hat{b} = 0.0776$ $\hat{c}_0 = 240.22$ and $\hat{c}_1 = 3238.82$ (i.e. $\hat{\sigma}^2 = 3479.04$), and at convergence (third iteration) we have $\hat{b} = 0.2029$, $\hat{c}_0 = 425.21$ and $\hat{c}_1 = 2924.66$ (i.e. $\hat{\sigma}^2 = 3349.87$).

```
## estimation of the variogram

geoD <- as.geodata(as.data.frame(cbind(x,y,z)))
v1 <- variog(geoD,max.dist=0.7,uvec=seq(0,0.7,length=10))
M<-variofit(v1,c(3000,0.04),cov.model="exponential")
V.pep1 <- as.vario.param(model="exponential", range=M$cov.pars[2],
                         sill=M$cov.pars[1], nugget=M$nugget)

## detection of ZACs

alpha.pep1 <- alphaG(vario.param=V.pep1, alphapval = 0.05, D = areapl(Poly))
zac.pep1 <- deteczac(x, y, z, vario.param=V.pep1, nx=60, ny=60, poly=Poly,
                     alpha = alpha.pep1, alphapval = 0.05, kriging = 2)

...

## re-estimation of the variogram

v3 <- vario.zac(zac.pep2, dist.seq = seq(0, 0.7, length=10))
M<-variofit(v3,c(2500,0.05),cov.model="exponential")
V.pep3 <- as.vario.param(model="exponential", range=M$cov.pars[2],
                         sill=M$cov.pars[1], nugget=M$nugget)

## detection of ZACs

alpha.pep3 <- alphaG(vario.param=V.pep3, alphapval = 0.05, D = areapl(Poly))
zac.pep3 <- deteczac(x, y, z, vario.param=V.pep3, nx=60, ny=60, poly=Poly,
                     alpha = alpha.pep3, alphapval = 0.05, kriging = 2)
```

The detected ZACs at the convergence (Figure 3*a*) are similar to the ones obtained when considering no nugget effect in the variogram (Figure 3*b*). This illustrates that the iterative algorithm is able to detect discontinuities in the expectation even when a nugget is permitted.

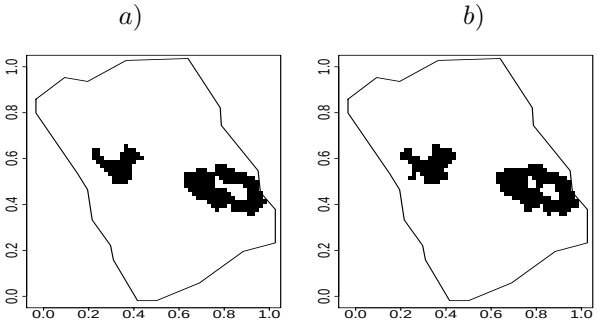


Figure 3: *ZACs detected at the convergence when using: a) a nugget effect b) no nugget effect in the variogram estimation.*

4 Help for the functions

4.1 alphaG

Description

This function approximates the local level α.

Usage

```
alphaG(vario.param, alphapval=0.05, D=1)
```

Arguments

vario.param	object of the class 'VarioParam'.
alphapval	global level at which p-values are compared.
D	area of the study domain.

Details

The local level α can be approximated by:

$$\alpha = 1 - (1 - \texttt{alphapval})^{1/N},$$

where $N = \texttt{D}/A$ represents the number of independent local tests that we can consider on the domain $\mathcal{D}$, D is the area of $\mathcal{D}$ and A denotes the integral range (Lantuéjoul, 1991) of the covariance function $C_Z(\mathbf{h})$ of the variable Z: $A = \int_{\mathbb{R}^2} \frac{C_Z(\mathbf{h})}{C_Z(\mathbf{0})} \, \mathrm{d}\mathbf{h}$.
This function is only implemented for exponential, Gaussian or spherical variogram models.

Value

alpha: local level of confidence.

Author(s)

Edith Gabriel, e.gabriel@lancaster.ac.uk.

Reference

Lantuéjoul, C. (1991) Ergodicity and integral range. *Journal of Microscopy*, **161**, 387–403.

See Also

as.vario.param.

Examples

```
vparam <- as.vario.param(model="exponential", range=0.2, sill=3)
alpha <- alphaG(vparam)
```

4.2 as.vario.param

Description

Create an object of class "VarioParam".

Usage

```
as.vario.param(model, range, sill, nugget=0)
```

Arguments

`model`: Variogram model, see details.
`range`: numeric; range of the variogram.
`sill`: numeric; sill of the variogram if `nugget=0`, sill-nugget otherwise.
`nugget`: numeric; nugget effect.

Details

The implemented variogram models are:

- `linear` for bounded linear
- `circular` for circular
- `spherical` for spherical
- `exponential` for exponential
- `pentaspherical` for pentaspherical
- `power` for power function
- `gaussian` for Gaussian
- `bessel` for Bessel function
- `hole bessel` for 2d hole effect with Bessel function
- `pepitic` for pepitic.

Value

`param` list of class "VarioParam" containing:

- `typmod` a numeric value defining the variogram model
- `range` and `sill` the parameters of the variogram.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

Examples

```
vario.param <- as.vario.param(model="exponential", range=0.2, sill=3)
```

4.3 buildcovvec, buildcovgrad, buildcovhess and buildcovmat.

Description

Compute the covariance vector at a given point, the gradient of a covariance function at a given point, the Hessian matrix of a covariance function at a given point, and the covariance matrix for a specified covariance model.

Usage

```
buildcovvec(x0, y0, x, y, typmod=4, ax=0.1, pal=1, path=myfortranpath)
buildcovgrad(x0, y0, x, y, typmod=4, ax=0.1, pal=1, path=myfortranpath)
buildcovhess(x0, y0, x, y, typmod=4, ax=0.1, pal=1, path=myfortranpath)
buildcovmat(x, y, typmod=4, ax=0.1, pal=1, path=myfortranpath)
```

Arguments

`x0, y0`: point where the variogram must be computed.
`x, y`: vectors of data locations.
`typmod`: numeric; model of covariance function.
`ax`: range of the covariance function.
`pal`: sill of the covariance function.
`path`: for the Fortran functions.

Details

The implemented covariance models are:
1 for bounded linear,
2 for circular,
3 for spherical,
4 for exponential,
5 for pentaspherical,
6 for power function,
7 for Gaussian,
8 for Bessel function,
9 for 2d hole effect with Bessel function,
10 for pepitic.

Value

`cc`: matrix or vector of the computations.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`. This code is based on Fortran routines developed by R. Webster and M. Goulard, and enhanced by D. Allard.

Examples

```
x <- rnorm(100); y <- rnorm(100)
x0 <- rnorm(1); y0 <- rnorm(1)

covvec <- buildcovvec(x0, y0, x, y, typmod=4, ax=0.2)
covgrad <- buildcovgrad(x0, y0, x, y, typmod=4, ax=0.2)
covhess <- buildcovhess(x0, y0, x, y, typmod=4, ax=0.2)
covmat <- buildcovmat(x, y, typmod=4, ax=0.2)
```

4.4 cc

Description

Determination of connected components for a given threshold.

Usage

```
cc(X2, talpha, path=myfortranpath)
```

Arguments

`X2`: matrix containing value of the chi-square field.
`talpha`: threshold,
`path`: for the Fortran functions.

Value

List containing:

- `ncc`: number of connected component,
- `lab`: mask of the connected components.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`. This code is based on Fortran routines developed by D. Allard.

Examples

```
x <- rnorm(100); y <- rnorm(100)
xygrid <- make.grid(nx=50,ny=50,poly=NULL)
vario.param <- as.vario.param(model="exponential",range=0.2,sill=1)
X2 <- sim.data(x, y, xygrid, vario.param)$X2
compco <- cc(X2,qchisq(0.95,2))
compco$ncc
cclab <- matrix(compco$lab,ncol=length(xygrid$y),byrow=T)
cclab[xygrid$mask==F] <- NaN
image(cclab,axes=F)
```

4.5 deteczac

Description

This function estimates the Zones of Abrupt Change at a level α.

Usage

```
deteczac(x, y, z, vario.param, nx=50, ny=50, poly=NULL, alpha=0.01,
         alphapval=0.05, kriging=2)
```

Arguments

`x, y, z`: values and data locations,
`vario.param`: object of class 'VarioParam',
`nx, ny`: dimension of the grid superimposed on the domain,
`poly`: matrix containing the x, y-coordinates of the vertices of the polygon boundary,
`alpha` : local level of confidence,
`alphapval`: global level at whici p-values are compared,
`kriging`: numerical value specifying the kriging interpolator: 1 for simple kriging and 2 for ordinary kriging.

Value

A list containing:

- `zac`: matrix with zero values where there is no ZAC, 1 where there are non-significant ZACs, 2 where there are significant ZACs and NaN outside `poly`,
- `X2`: matrix of values of the chi-square field,
- `zk`: matrix of values of the interpolated field,
- `detlambda`: matrix containing the determinant of the curvature of the chi-square field evaluated at each grid node,
- `p.value`: vector of p-values associated to the detected ZACs,
- `cclab`: matrix with zero values where there is no ZAC, 1 where there are non-significant ZACs, 2 where there are significant ZACs,
- `*max`: coordinates `Xmax,Ymax` of the maximum of the chi-square `Tmax` and area `Smax` of the corresponding ZAC,
- `xygrid`: result of the function '`make.grid`',
- `...`: copy of the arguments `x, y, z, vario.param, alpha, alphapval, poly` passed.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

See Also

`as.vario.param` and `make.grid`.

4.6 elim

Description

Remove the pairs of samples intersecting or belonging to ZACs.

Usage

```
elim(reszac, only.sign=T)
```

Arguments

`reszac`: result of the function '`deteczac`'.
`only.sign`: logical; indicate whether the removed pairs of samples intersect only significant ZACs (if `TRUE`) or both significant and non-significant ZACs (if `FALSE`).

Details

This function discards all pairs $\{Z(x_i, y_i), Z(x_j, y_j)\}$ for which the segment $[(x_i, y_i), (x_j, y_j)]$ intersects a ZAC.

Value

List containing:

- `X, Y, XX, YY`: vectors of coordinates of the non-discarded pairs of samples, corresponding to (x_k, y_k) and (x_l, y_l) such that the segment $[(x_k, y_k), (x_l, y_l)]$ does not intersect a ZAC,
- `Z, ZZ`: vectors of the corresponding data,
- `xelim, yelim, zelim`: coordinates and values of the discarded points,
- `PL`: list of convex envelops defining the ZACs.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

4.7 eval.alpha

Description

This function evaluates the local level α adequate to a sample under the hypothesis of stationarity.

Usage

```
eval.alpha(x, y, xygrid, vario.param, alphalim = c(0.00005,0.05),
           alphapval=0.05, ni=100, allgrid=F)
```

Arguments

`x, y`: sample locations,
`xygrid`: result of the function '`make.grid`',
`vario.param`: object of class "VarioParam",
`alphalim`: minimum and maximum values of α as starting points of the algorithm
`alphapval`: level at which p-values are compared,
`ni`: number of simulation to generate for finding alpha,
`allgrid`: logical; if `TRUE` ZACs are computed on the whole grid, otherwise inside the convex envelop containing the samples.

Details

The correct level alpha depends on `alphapval`, the covariance function, the discretisation of the interpolation grid and the sampling pattern. It is assessed by Monte-Carlo simulation: a set of K standard Gaussian random fields, corresponding to the null hypothesis of the absence of any ZAC, are simulated on the same sample locations with the estimated covariance function. The estimated local level α is such that ZACs are detected on K.`alphapval` simulations.

Note that time of computation can be long, e.g. in the example given below.

Value

`alpha`.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

See Also

`as.vario.param`, `make.grid`, `sim.data` and `nb.zac`.

Examples

```
x <- rnorm(100); y <- rnorm(100)
xygrid <- make.grid(nx=50,ny=50,poly=NULL)
vario.param <- as.vario.param(model="exponential",range=0.2,sill=1)
alpha <- eval.alpha(x, y, xygrid, vario.param)
```

4.8 intersection

Description

Test whether the two segments $[(x_1, y_1), (x_2, y_2)]$ and $[(x_3, y_3), (x_4, y_4)]$ are intersected.

Usage

```
intersection(x1, y1, x2, y2, x3, y3, x4, y4)
```

Arguments

`xi, yi, i=1,2,3,4`: coordinates of the points defining the two segments.

Value

Logical specifying whether or not the segments are intersected.

Author(s)

Edith Gabriel, e.gabriel@lancaster.ac.uk.

4.9 make.grid

Description

This function generates a rectangular grid of points in a polygon.

Usage

```
make.grid(nx, ny, poly=NULL)
```

Arguments

`nx, ny`: number of points in each row and each column of the rectangular grid,
`poly`: matrix containing the x, y-coordinates of the vertices of the polygon boundary.

Details

Let us denote $\mathcal{P}$ the polygon and (x_p, y_p) the points defining $\mathcal{P}$. The `nx`×`ny` grid is built according to the algorithm:

1. Compute the mesh size of the grid:

$$x_{inc} = (\max(x_p) - \min(x_p))/\texttt{nx} \quad ; \quad y_{inc} = (\max(y_p) - \min(y_p))/\texttt{ny}.$$

2. Define the grid nodes (x, y) by:
   a) $x_1 = \min(x_p) + x_{inc}/2; y_1 = \min(y_p) + y_i nc/2$
   b) $x_i = x_{i-1} + x_{inc}, i = 1, \ldots, \texttt{nx}$; $y_j = y_{j-1} + y_{inc}, j = 1, \ldots, \texttt{ny}$.

Value

A list containing:

- `x ,y`: numeric vectors giving the coordinates of the points of the rectangular grid,
- `xx, yy`: `nx`×`ny` matrix containing x and y coordinates respectively,
- `pts`: index of the grid points belonging to the polygon,
- `xinc, yinc`: mesh size of the grid,
- `mask`: `nx`×`ny` matrix of logicals. `TRUE` if the grid point belongs to the polygon.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

Examples

```
poly1 <- matrix(c(0,0.5,1,0.5,0.5,1,0.5,0),ncol=2,byrow=F)
xygrid <- make.grid(nx=50,ny=50,poly=poly1)
plot(poly1,type="n",axes=F,xlab="",ylab="")
polygon(poly1,lwd=2)
points(as.vector(xygrid$xx)[xygrid$pts],as.vector(xygrid$yy)[xygrid$pts],
       pch=19,col=4,cex=0.4)
```

4.10 na2zero

Description

This function replaces missing values by zero in a matrix.

Usage

```
na2zero(M)
```

Arguments

M: a numeric matrix with possibly NaN values.

Value

The matrix M.

Author(s)

Edith Gabriel, e.gabriel@lancaster.ac.uk.

4.11 nb.zac

Description

This function is used in the determination of the local level α to evaluate the number of simulations with significant ZACs when the hypothesis of absence of ZAC is true.

Usage

```
nb.zac(X2, Ldet, xygrid, alpha=0.01, alphapval=0.05)
```

Arguments

`X2`: `nx`×`ny`×`ni` table containing the values of the chi-square field for each of the `ni` simulations,
`Ldet`: `nx`×`ny`×`ni` containing the values of the determinant of $\mathbf{\Lambda}$, matrix of curvature of the chi-square field evaluated at each pixel of the interpolation grid,
`xygrid`: result of the function '`make.grid`',
`alpha`: local level of confidence,
`alphapval`: global level at which p-values are compared.

Value

`Nsim`: number of simulation with significant ZACs.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

See Also

`eval.alpha`, `make.grid` and `sim.data`.

4.12 plot.zac

Description

This function either plots the detected ZACs at the level α or represents the significant ZACs, estimated at the level α, to a greater level α_0.

Usage

```
plot.zac(reszac, alpha0=NULL, ...)
```

Arguments

`reszac`: result of the function '`deteczac`',
`alpha0`: level at which ZACs estimated at the level α must be plotted, $\alpha_0 > \alpha$.
If `NULL`, ZACs are represented to the level α,
`...`: graphical parameters can be given as arguments to image.

Value

No value is returned.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

See Also

`zac`.

Examples

```
v1 <- as.vario.param(model="exponential",range=0.4,sill=3)
zac1 <- zac(ExData$x, ExData$y, ExData$z, vario.param=v1, alpha = 0.002)
plot.zac(reszac=zac1,col=c("white","gray","black"),axes=F,xlab="",ylab="")
```

4.13 rmultnorm

Description

This function generates n Gaussian vectors for a given mean vector and a given covariance matrix.

Usage

```
rmultnorm(n, mu, vmat, tol=1e-07)
```

Arguments

`n`: number of vectors to generate,
`mu`: mean vector,
`vmat`: covariance matrix,
`tol`: tolerance considered when testing symmetry of the covariance matrix.

Value

`ans`: n-row matrix containing the Gaussian vectors

Author(s)

Denis Allard and Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

4.14 sim.data

Description

This function is used when evaluating the local level alpha. It first generates a standard Gaussian random field at the data locations for a given covariance model. The chi-square field and its matrix of curvature $\mathbf{\Lambda}$ are then computed at each node of a grid superimposed on the domain.

Usage

```
sim.data(x, y, xygrid, vario.param)
```

Arguments

`x, y`: data locations,
`xygrid`: result of the function '`make.grid`',
`vario.param`: object of class "VarioParam".

Value

Al ist containing:

- `X2`: matrix of values of the chi-square field,
- `Ldet`: matrix of the determinant of $\mathbf{\Lambda}$ evaluated at each grid node,
- `x, y`: data locations,
- `z`: simulated Gaussian vector.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

See Also

`make.grid` and `as.vario.param`.

4.15 vario.zac

Description

This function computes the empirical variogram of a sample z evaluated at locations (x, y) by discarding all pairs $\{Z(x_i, y_i), Z(x_j, y_j)\}$ for which the segment $[(x_i, y_i), (x_j, y_j)]$ intersects a ZAC.

Usage

```
vario.zac(reszac, dist.seq=seq(10,200,by=10), only.sign=T)
```

Arguments

`reszac`: result of the function '`deteczac`',
`dist.seq`: distance vector at which the variogram is computed,
`only.sign`: logical; indicate whether the removed pairs of samples intersect only significant ZACs (if `TRUE`) or both significant and non-significant ZACs (if `FALSE`).

Details

This function provides an object that can be used by the function `variofit` (`geoR` package) to estimate covariance parameters by fitting a parametric model to a empirical variogram.

Value

An object of the class 'variogram' (even if slightly different) which is a list with the following components:

- `n`: number of points per bin,
- `u`: vector of distances,
- `v`: variogram values,
- `ncelim`: number of discarded pairs of samples.

Author(s)

Edith Gabriel, `e.gabriel@lancaster.ac.uk`.

See Also

`elim`.