# Introducción a la Geoestadística con R

*MBM: Estatística Espacial e Modelización*

*Marzo de 2017*

## El paquete geoR

El paquete `geoR` proporciona herramientas para el análisis de datos geoestadísticos utilizando el software `R` (otra alternativa puede ser emplear el paquete `gstat`, por ejemplo...). A continuación se ilustran algunas de las capacidades de este paquete.

### 1. Inicio de una sesión y de carga de datos

#### 1.1 Cargar el paquete

Después de iniciar la sesión R, cargar `geoR` con el comando `library` (o `require`). Si el paquete se carga correctamente aparece un mensaje.

```
library(geoR)
```

```
## ----------------------------------------------------------
##  Analysis of Geostatistical Data
##  For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
##  geoR version 1.7-5.2 (built on 2016-05-02) is now loaded
## ----------------------------------------------------------
```

#### 1.2 Archivos de datos

Normalmente, los datos se almacenan como un objeto (una lista) de la clase `geodata`. Un objeto de esta clase contiene obligatoriamente dos elementos:

- `$coords`: las coordenadas de las posiciones de los datos.
- `$data`: los valores observados de la variables.

Opcionalmente pueden tener otros elementos, como covariables y coordenadas de las fronteras de la zona de estudio.

Hay algunos conjuntos de datos incluidos en el paquete de distribución.

```
# data()                    # lista todos los conjuntos de datos disponibles
# data(package = "geoR")    # lista los conjuntos de datos en el paquete geoR

data(wolfcamp)              # carga el archivo de datos wolfcamp
summary(wolfcamp)
```

```
## Number of data points: 85
##
## Coordinates summary
##       Coord.X   Coord.Y
## min -233.7217 -145.7884
## max  181.5314  136.4061
##
```

```
## Distance summary
##         min         max
##   0.3669819 436.2067085
##
## Data summary
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##  312.1095  471.8218  547.7156  610.2845  774.1778 1088.4209
```

Se pueden importar directamente un archivo de datos en formato texto:

```
ncep <- read.geodata('ncep.txt', header = FALSE, coords.col = 1:2, data.col = 4)
# plot(ncep)
# summary(ncep)
```

También se puede convertir un `data.frame` a un objeto `geodata`:

```
ncep.df <- read.table('ncep.txt', header = FALSE)
names(ncep.df) <- c('x', 'y', 't', 'z')
# str(ncep.df)
# Nota: los datos son espacio-temporales, pero geoR sólo admite datos 2D

datgeo <- as.geodata(ncep.df, coords.col = 1:2, data.col = 4)
# plot(datgeo)
# summary(datgeo)
```

O objetos de datos espaciales (entre ellos los compatibles del paquete `sp`), por ejemplo el siguiente código crea un objeto `SpatialPointsDataFrame` y lo convierte a `geodata`:

```
library(sp)
load("caballa.galicia.RData")
coordinates(caballa.galicia) <- c("x","y")
proj4string(caballa.galicia) <- CRS("+proj=longlat +ellps=WGS84")

datgeo <- as.geodata(caballa.galicia["lcpue"])
# Problemas con coordenadas duplicadas  (ver ?duplicated)
# plot(datgeo)
# summary(datgeo)
```

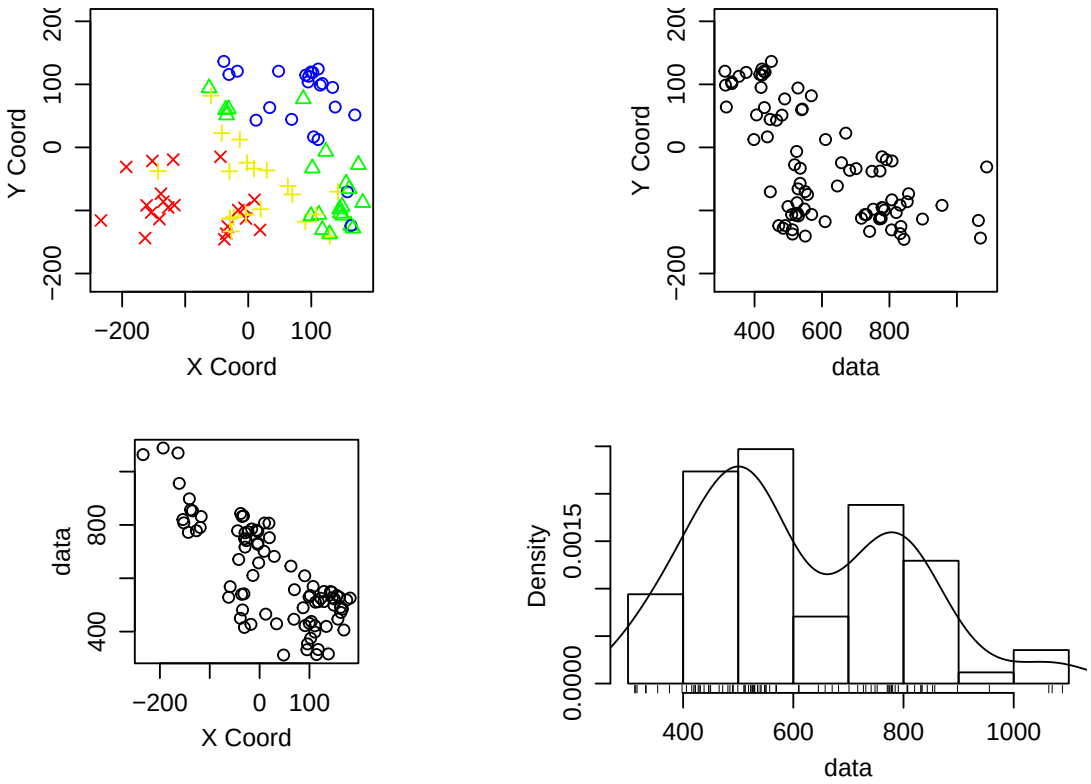
En la documentación de las funciones `as.geodata` y `read.geodata` hay más información sobre cómo importar/convertir datos.

## 2. Análisis descriptivo de datos geoestadísticos

Como se mostró anteriormente, el método `summary` proporciona un breve resumen descriptivo de los datos (ver `?summary.geodata`).

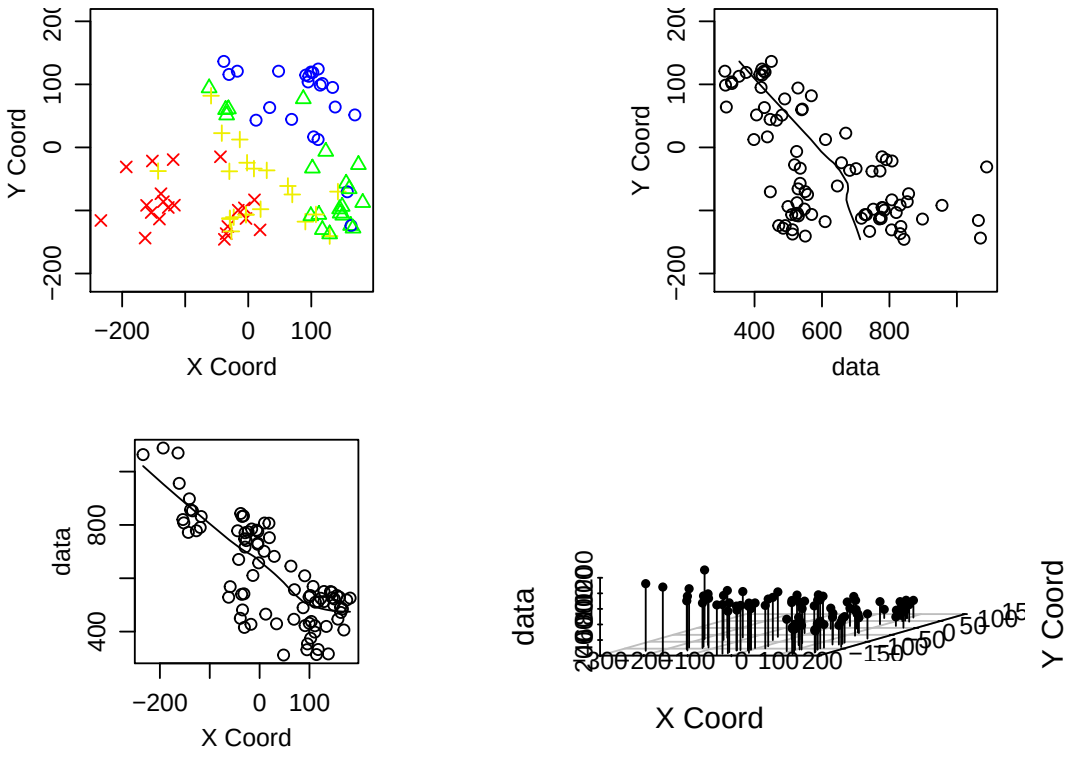
La función `plot()` genera por defecto gráficos de los valores en las posiciones espaciales (distinguiendo según cuartiles), los datos frente a las coordenadas y un histograma de los datos:

```
plot(wolfcamp)
```



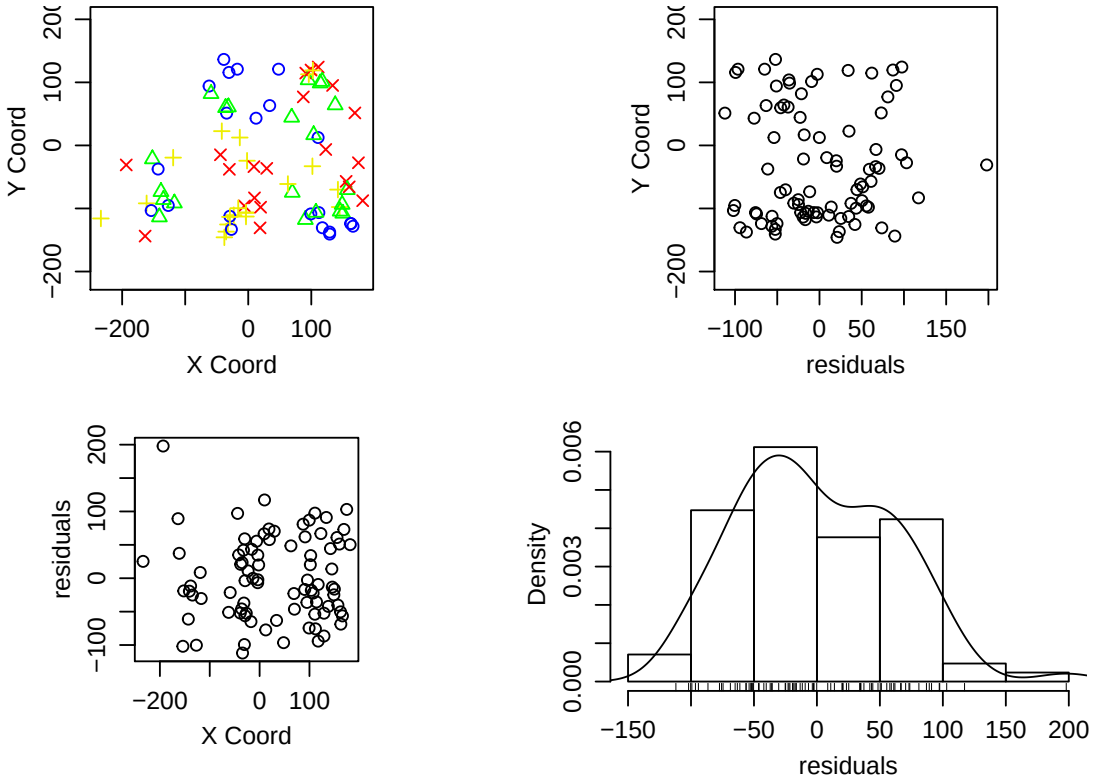

Los gráficos de dispersión de los datos frente a las coordenadas nos pueden ayudar a determinar si hay una tendencia. También, en lugar del histograma, nos puede interesar un gráfico de dispersión 3D

```
plot(wolfcamp, lowess = TRUE, scatter3d = TRUE)
```



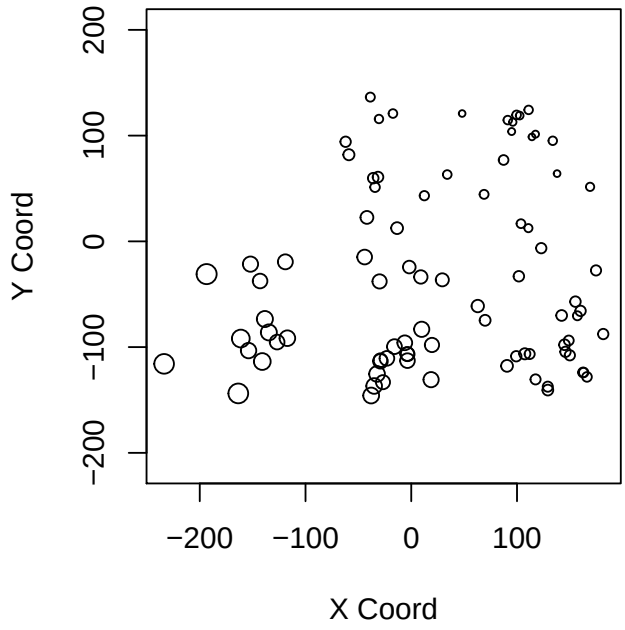

Si se asume que hay una tendencia puede interesar eliminarla:

```
plot(wolfcamp, trend=~coords)
```



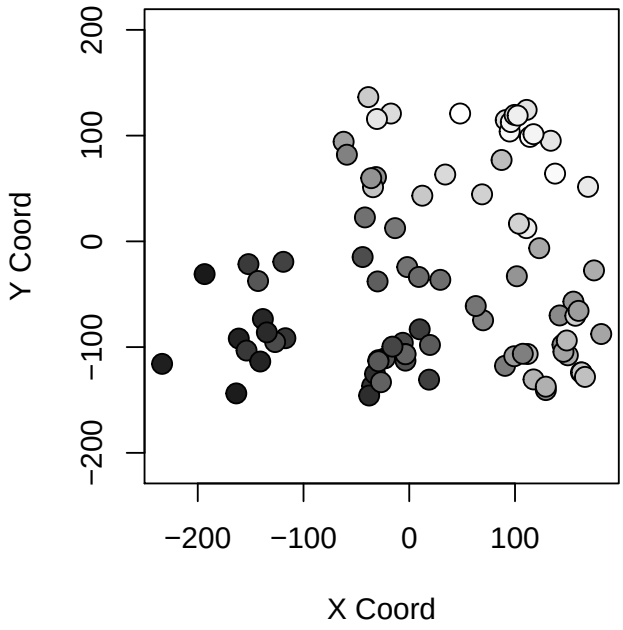

El comando `points(geodata)` (función `points.geodata`) genera un gráfico con las posiciones de los datos (y por defecto con el tamaño de los puntos proporcional al valor):

```
points(wolfcamp)
```




Se pueden establecer los tamaños de los puntos, simbolos y colores a partir de los valores de los datos. Por ejemplo, para los puntos, empleando el argumento: `pt.divide = c("data.proportional", "rank.proportional", "quintiles", "quartiles", "deciles", "equal")`.

```
points(wolfcamp, col = "gray", pt.divide = "equal")
```



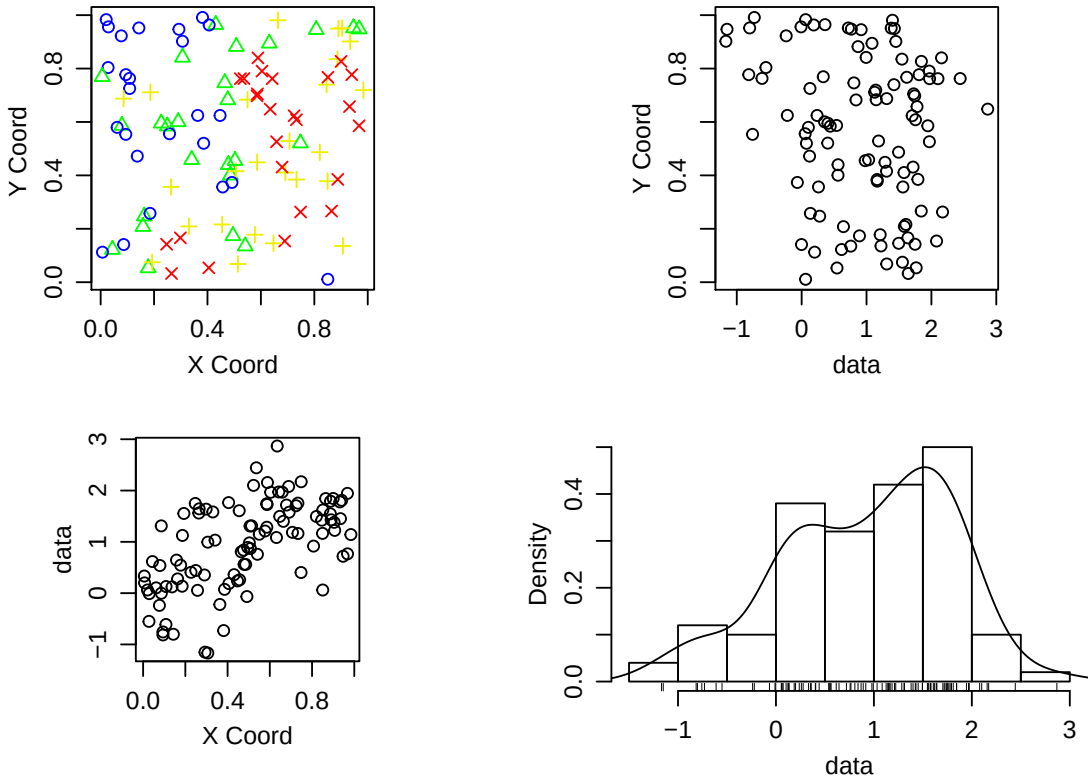

## 3. Modelado de la dependencia

En la primera parte de esta sección consideraremos un proceso espacial sin tendencia:

```
data(s100) # Cargar datos estacionarios
summary(s100)
```

```
## Number of data points: 100
##
## Coordinates summary
##         Coord.X    Coord.Y
## min 0.005638006 0.01091027
## max 0.983920544 0.99124979
##
## Distance summary
##         min         max
## 0.007640962 1.278175109
##
## Data summary
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -1.1676955  0.2729882  1.1045936  0.9307179  1.6101707  2.8678969
##
## Other elements in the geodata object
## [1] "cov.model" "nugget"    "cov.pars"  "kappa"     "lambda"
```

```
plot(s100)
```




En el último apartado se tratará el caso general.

### 3.1 Variogramas empíricos

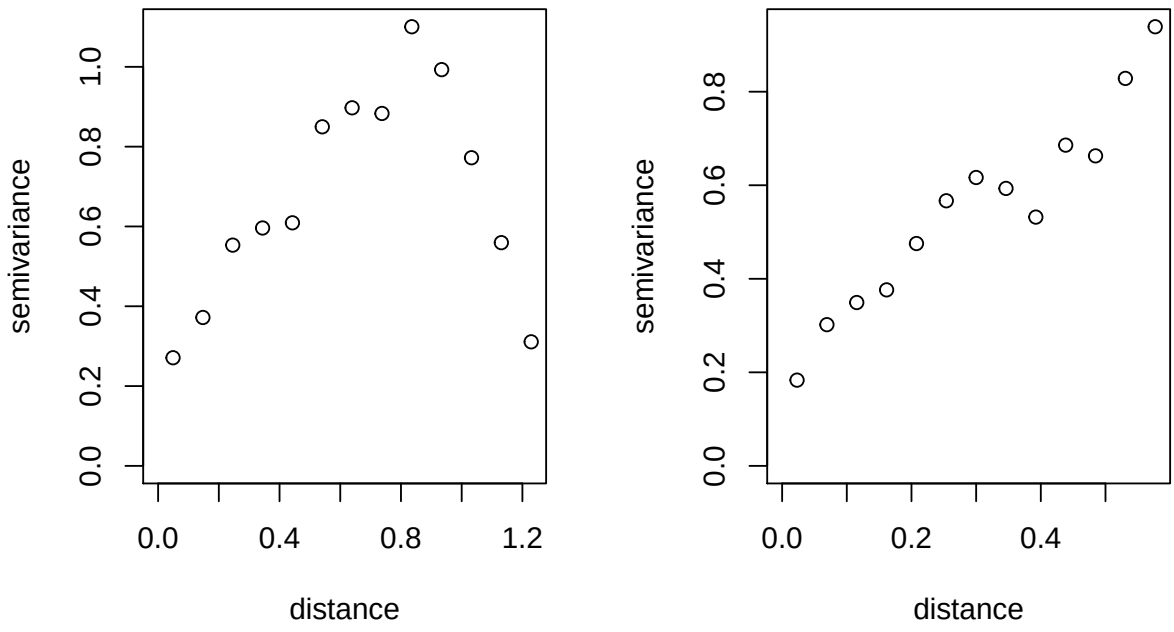
Los variogramas empíricos se calculan utilizando la función `variog`:

```
oldpar <- par(mfrow=c(1,2))
plot(variog(s100))
```

```
## variog: computing omnidirectional variogram
```

```
plot(variog(s100, max.dist = 0.6))
```

```
## variog: computing omnidirectional variogram
```




```
par(oldpar)
```

La recomendación es considerar solo saltos hasta la mitad de la máxima distancia (ver 'Distance summary' en resultados del sumario).

```
vario <- variog(s100, max.dist = 0.6)
```

```
## variog: computing omnidirectional variogram
```

```
names(vario)
```

```
##  [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"
```

```
# str(vario)
```

NOTA: La componente `u` contiene los saltos, `v` las estimaciones del semivariograma (semivarianzas) y `n` el número de aportaciones.

Los resultados pueden ser nubes de puntos (semivarianzas), valores discretizados (binned) o suavizados, dependiendo del parámetro: `option = c("bin", "cloud", "smooth")`

```
# Calculo de los variogramas empíricos
vario.b <- variog(s100, max.dist = 0.6) #discretizado
```

```
## variog: computing omnidirectional variogram
```

```
vario.c <- variog(s100, max.dist=0.6, op="cloud")  #nube
```

```
## variog: computing omnidirectional variogram
```

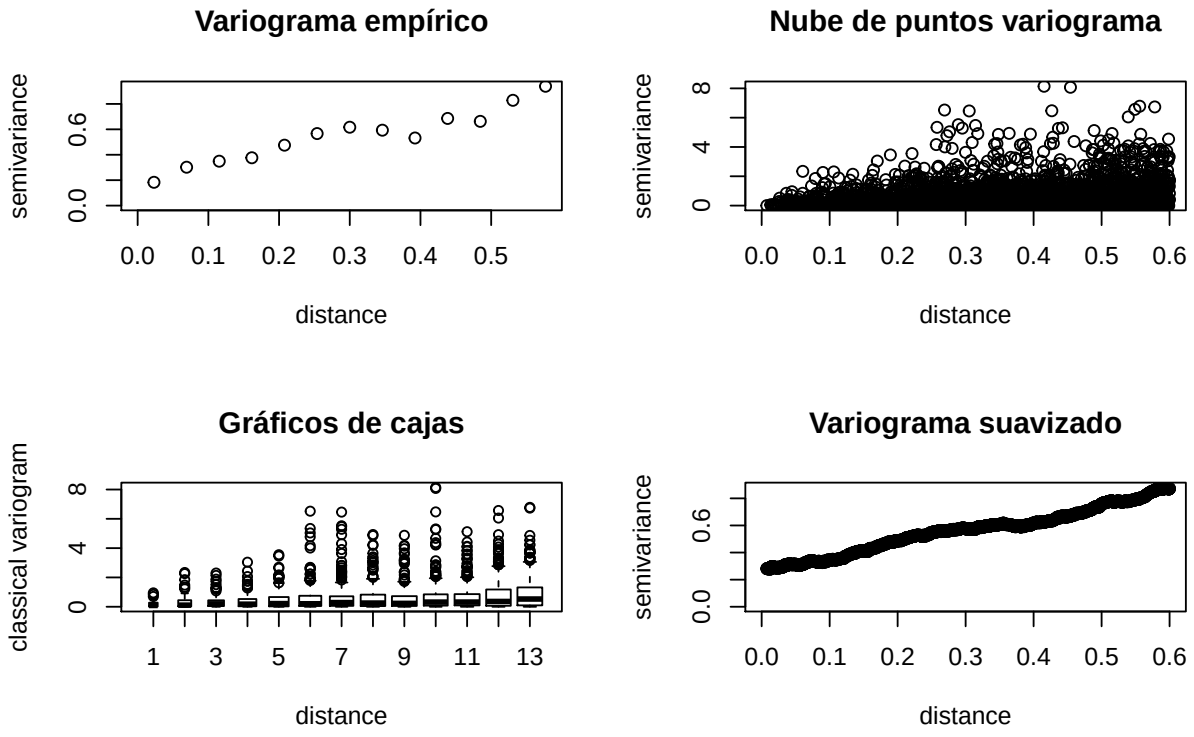
```
vario.bc <- variog(s100, max.dist=0.6, bin.cloud=TRUE)  #discretizado+nube
```

```
## variog: computing omnidirectional variogram
```

```
vario.s <- variog(s100, max.dist=0.6, op="sm", band=0.2)  #suavizado
```

```
## variog: computing omnidirectional variogram
```

```
# Representación gráfica
oldpar<-par(mfrow=c(2,2)) # Preparar para 4 gráficos por ventana
plot(vario.b, main="Variograma empírico")
plot(vario.c, main="Nube de puntos variograma")
plot(vario.bc, bin.cloud=TRUE, main="Graficos de cajas")
title("Gráficos de cajas") # Corregir fallo del comando anterior
plot(vario.s, main="Variograma suavizado")
```




```
par(oldpar) # Restaurar opciones de gráficos
```

Si hay valores atípicos (o la distribución de los datos es asimétrica) puede ser preferible utilizar el estimador robusto. Se puede calcular este estimador estableciendo `estimator.type = "modulus"`:

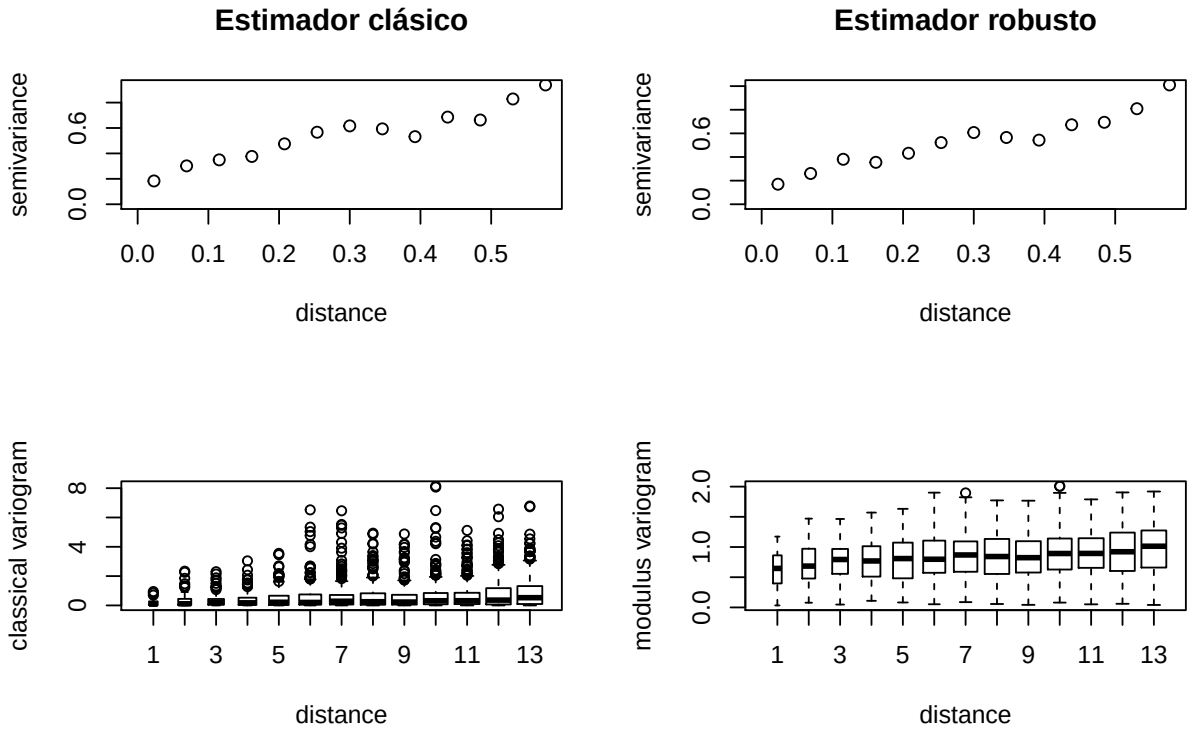
```
varior.b <- variog(s100, estimator.type = "modulus", max.dist=0.6)
```

```
## variog: computing omnidirectional variogram
```

```
varior.bc <- variog(s100, estimator.type = "modulus", max.dist=0.6, bin.cloud=TRUE)
```

```
## variog: computing omnidirectional variogram
```

```
oldpar<-par(mfrow=c(2,2)) #Preparar para 4 gráficos por ventana
plot(vario.b, main="Estimador clásico")
plot(varior.b, main="Estimador robusto")
plot(vario.bc, bin.cloud=TRUE)
plot(varior.bc, bin.cloud=TRUE)
```




```
par(oldpar) #Restaurar opciones de gráficos
```

En el caso de anisotropía, también se pueden obtener variogramas direccionales con la función `variog` mediante los argumentos `direction` y `tolerance`. Por ejemplo, para calcular un variograma en la dirección de 60 grados (con la tolerancia angular por defecto de 22.5 grados):

```
vario.60 <- variog(s100, max.dist = 0.6, direction = pi/3) #variograma en la dirección de 60 grados
```

```
## variog: computing variogram for direction = 60 degrees (1.047 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
```

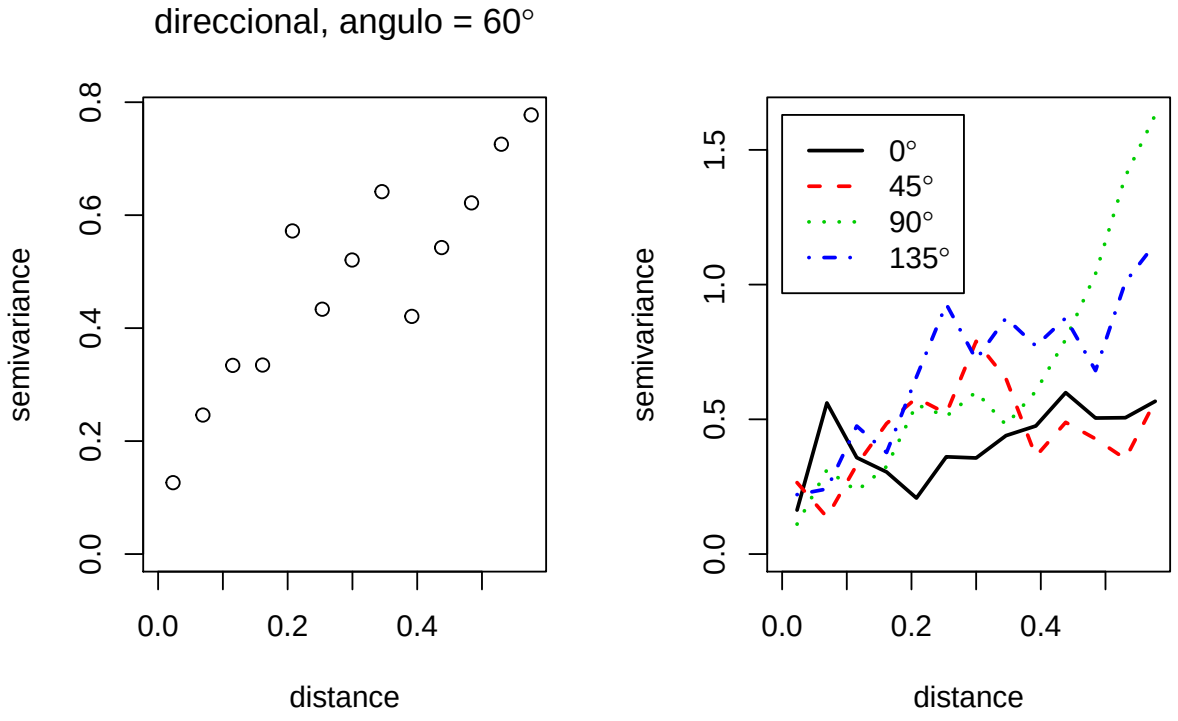
Para estudiar si hay anisotropía, se pueden cálcular de forma rápida variogramas direccionales con la función `variog4`. Por defecto calcula cuatro variogramas direccionales, correspondientes a los ángulos 0, 45, 90 y 135 grados:

```
vario.4 <- variog4(s100, max.dist = 0.6)
```

```
## variog: computing variogram for direction = 0 degrees (0 radians)
##         tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 45 degrees (0.785 radians)
```

```
##          tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 90 degrees (1.571 radians)
##          tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing variogram for direction = 135 degrees (2.356 radians)
##          tolerance angle = 22.5 degrees (0.393 radians)
## variog: computing omnidirectional variogram
```

```
oldpar <- par(mfrow=c(1,2))
plot(vario.60)
title(main = expression(paste("direccional, angulo = ", 60 * degree)))
plot(vario.4, lwd = 2)
```




```
par(oldpar)
```

### 3.2 Ajuste de un modelo de variograma

Los estimadores empíricos no pueden ser empleados en la práctica (no verifican necesariamente las propiedades de un variograma válido), por lo que se suele recurrir en la práctica al ajuste de un modelo válido. Con el paquete `geoR` podemos realizar el ajuste:

1. "A ojo": representando diferentes modelos sobre un variograma empírico (usando la función `lines.variomodel` o la función `eyefit`).
2. Por mínimos cuadrados: ajustando por mínimos cuadrados ordinarios (OSL) o ponderados (WLS) al variograma empírico (usando la función `variofit`),
3. Por máxima verosimilitud: estimando por máxima verosimilitud (ML) o máxima verosimilitud restringida (REML) los parámetros a partir de los datos (utilizando la función `likfit`),

4. Métodos bayesianos (utilizando la función `krige.bayes`).

Ejemplo de ajuste "a ojo":

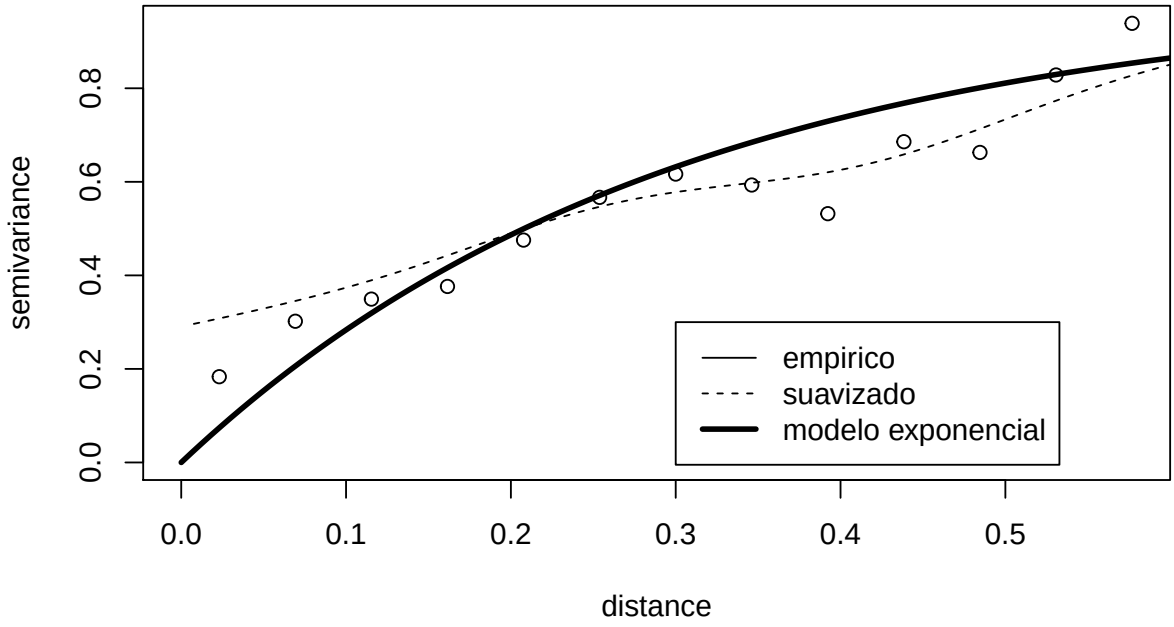
```
vario.b <- variog(s100, max.dist=0.6) #discretizado
```

```
## variog: computing omnidirectional variogram
```

```
vario.s <- variog(s100, max.dist=0.6,option = "smooth", kernel = "normal", band = 0.2)  #suavizado
```

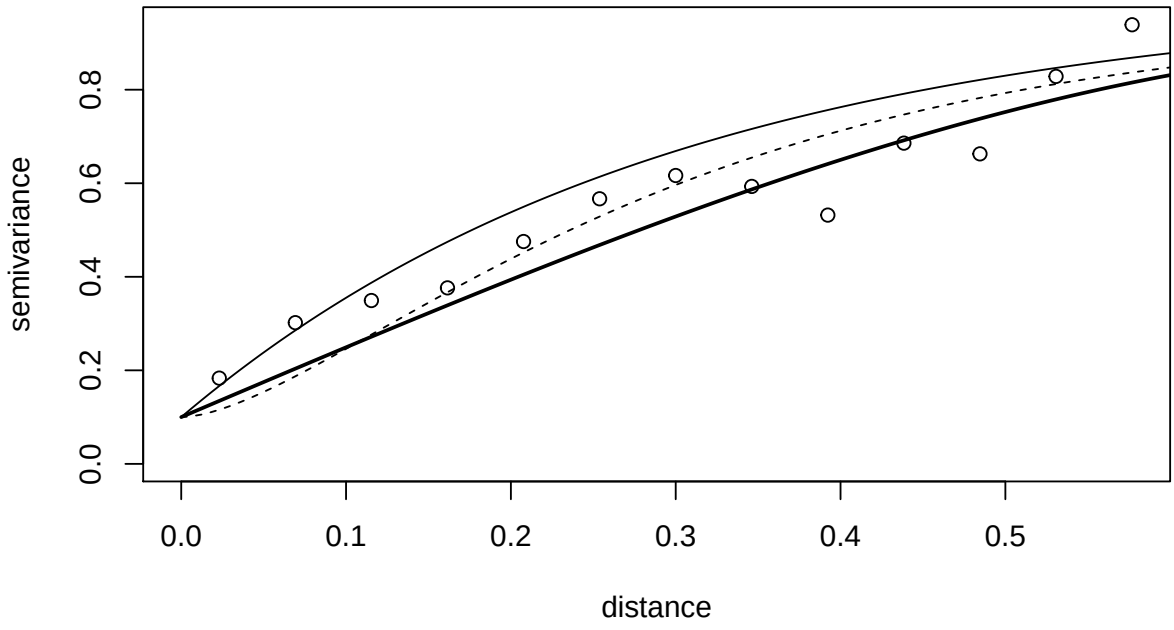
```
## variog: computing omnidirectional variogram
```

```
plot(vario.b)
lines(vario.s, type = "l", lty = 2)

lines.variomodel(cov.model = "exp", cov.pars = c(1,0.3), nugget = 0, max.dist = 0.6, lwd = 3)
legend(0.3, 0.3, c("empirico", "suavizado", "modelo exponencial"), lty = c(1, 2, 1), lwd = c(1, 1, 3))
```




Otros ajustes:

```
plot(vario.b)
lines.variomodel(cov.model = "exp", cov.pars = c(0.9,0.3), nug = 0.1, max.dist = 0.6)
lines.variomodel(cov.model = "mat", cov.pars = c(0.85,0.2), nug = 0.1, kappa = 1, max.dist = 0.6,lty = 1
lines.variomodel(cov.model = "sph", cov.pars = c(0.8,0.8), nug = 0.1, max.dist = 0.6, lwd = 2)
```




Nota: no hace falta escribir el nombre completo de los parámetros (basta con que no dé lugar a confusión).

En las versiones recientes de `geoR` está disponible una función para realizar el ajuste gráficamente de forma interactiva (cuadro de diálogo en tcl/tk):

```
eyefit(vario.b)
```

Cuando se utilizan las funciones `variofit` y `likfit` para la estimación de parámetros, el efecto pepita (nugget) puede ser estimado o establecido a un valor fijo. Lo mismo ocurre con los parámetros de suavidad, anisotropía y transformación de los datos. También se dispone de opciones para incluir una tendencia. Las tendencias pueden ser polinomios en función de las coordenadas y/o funciones lineales de otras covariables.

Ejemplos de estimación por mínimos cuadrados (llamadas a `variofit`):

```
#   Modelo exponencial con par ini umbral 1 y escala 0.5 (1/3 rango =1.5)

vario.ols <- variofit(vario.b, ini = c(1, 0.5), weights = "equal")  #ordinarios
```

```
## variofit: covariance model used is matern
## variofit: weights used: equal
## variofit: minimisation function used: optim
```

```
vario.wls <- variofit(vario.b, ini = c(1, 0.5), weights = "cressie")  #ponderados
```

```
## variofit: covariance model used is matern
## variofit: weights used: cressie
## variofit: minimisation function used: optim
```

```
vario.wls
```

```
## variofit: model parameters estimated by WLS (weighted least squares):
## covariance model is: matern with fixed kappa = 0.5 (exponential)
```

```
## parameter estimates:
##   tausq sigmasq     phi
##  0.1955  2.0110  1.4811
## Practical Range with cor=0.05 for asymptotic range: 4.437092
##
## variofit: minimised weighted sum of squares = 31.5115
```

```
summary(vario.wls)
```

```
## $pmethod
## [1] "WLS (weighted least squares)"
##
## $cov.model
## [1] "matern"
##
## $spatial.component
##  sigmasq      phi
## 2.010972 1.481138
##
## $spatial.component.extra
## kappa
##   0.5
##
## $nugget.component
##     tausq
## 0.1955322
##
## $fix.nugget
## [1] FALSE
##
## $fix.kappa
## [1] TRUE
##
## $practicalRange
## [1] 4.437092
##
## $sum.of.squares
##   value
## 31.5115
##
## $estimated.pars
##     tausq   sigmasq       phi
## 0.1955322 2.0109718 1.4811376
##
## $weights
## [1] "cressie"
##
## $call
## variofit(vario = vario.b, ini.cov.pars = c(1, 0.5), weights = "cressie")
##
## attr(,"class")
## [1] "summary.variomodel"
```

Ejemplo de estimación por máxima verosimilitud (llamada a `likfit`):

```r
vario.ml <- likfit(s100, ini = c(1, 0.5)) #Modelo exponencial con par ini umbral y escala (1/3 rango)
```

```
## ---------------------------------------------------------------
## 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.
## 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.
```

```r
vario.ml
```

```
## likfit: estimated model parameters:
##     beta    tausq  sigmasq      phi
## "0.7766" "0.0000" "0.7517" "0.1827"
## Practical Range with cor=0.05 for asymptotic range: 0.547383
##
## likfit: maximised log-likelihood = -83.57
```

```r
summary(vario.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)
##
## Practical Range with cor=0.05 for asymptotic range: 0.547383
##
## Maximised Likelihood:
##    log.L n.params      AIC      BIC
## "-83.57"      "4"  "175.1"  "185.6"
##
## non spatial model:
##    log.L n.params      AIC      BIC
## "-125.8"      "2"  "255.6"  "260.8"
```

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

Ejemplo de estimación por máxima verosimilitud restringida (opción de `likfit`):

```
vario.reml <- likfit(s100, ini = c(1, 0.5), method = "RML")
```

```
## Warning in likfit(s100, ini = c(1, 0.5), method = "RML"): argument "method"
## has changed and is now used as an argument to be passed to optim(). Use
## "lik.method" to define the likelihood method
```

```
## ---------------------------------------------------------------
## 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.
## 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.
```

```
summary(vario.reml)
```

```
## Summary of the parameter estimation
## ----------------------------------
## Estimation method: restricted maximum likelihood
##
## Parameters of the mean component (trend):
##   beta
## 0.7478
##
## Parameters of the spatial component:
##    correlation function: exponential
##       (estimated) variance parameter sigmasq (partial sill) =  0.8473
##       (estimated) cor. fct. parameter phi (range parameter)  =  0.2102
##    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)
##
## Practical Range with cor=0.05 for asymptotic range: 0.6296295
##
## Maximised Likelihood:
##    log.L n.params      AIC      BIC
## "-81.53"      "4"  "171.1"  "181.5"
##
## non spatial model:
##    log.L n.params      AIC      BIC
## "-125.1"      "2"  "254.1"  "259.3"
```

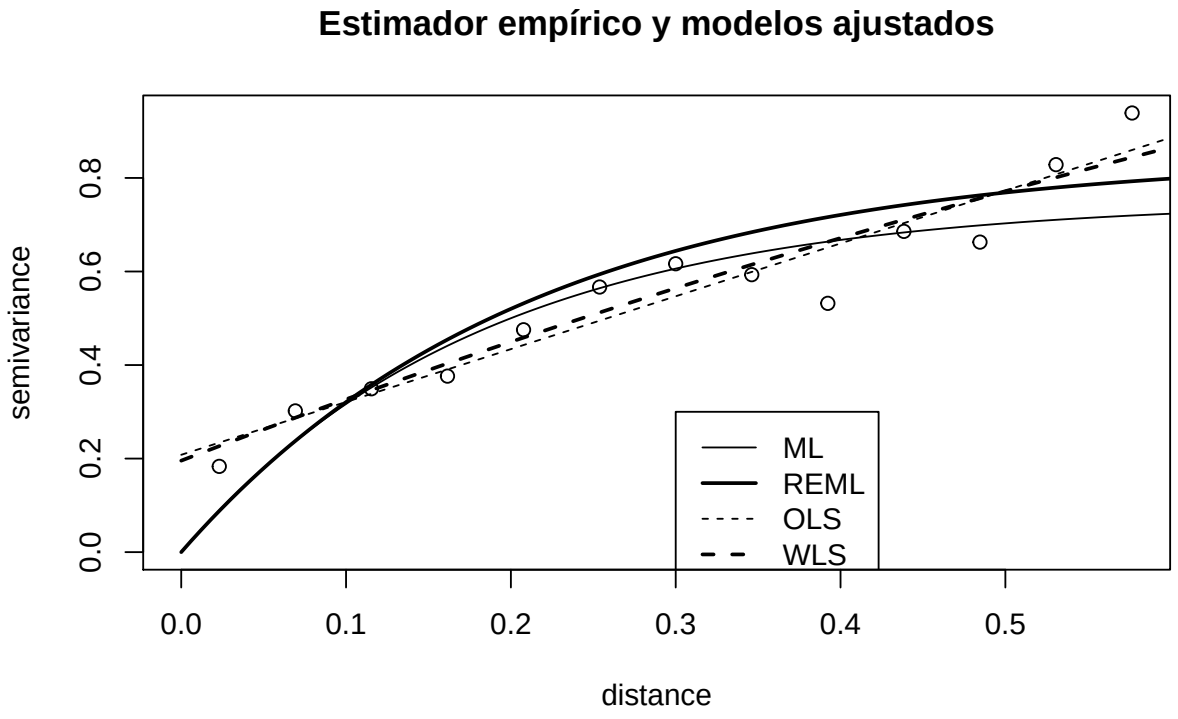
```
##
## Call:
## likfit(geodata = s100, ini.cov.pars = c(1, 0.5), method = "RML")
```

**NOTAS**:

- Para fijar el nugget a un valor p.e. 0.15 añadir las opciones: `fix.nugget = TRUE, nugget = 0.15`.
- Se puede tener en cuenta anisotropía geométrica en los modelos de variograma a partir de los parámetros `psiA` (ángulo, en radianes, de la dirección de mayor dependencia espacial i.e. con el máximo rango) y `psiR` (relación, mayor o igual que 1, entre los rangos máximo y mínimo). Se pueden fijar a distintos valores o estimarlos incluyendo las opciones `fix.psiA = FALSE` y `fix.psiR = FALSE` en las llamadas a las rutinas de ajuste.)

Representación gráfica junto al estimador empírico:

```
plot(vario.b, main = "Estimador empírico y modelos ajustados")
lines(vario.ml, max.dist = 0.6)
lines(vario.reml, lwd = 2, max.dist = 0.6)
lines(vario.ols, lty = 2, max.dist = 0.6)
lines(vario.wls, lty = 2, lwd = 2, max.dist = 0.6)
legend(0.3, 0.3, legend = c("ML", "REML", "OLS", "WLS"), lty = c(1, 1, 2, 2), lwd = c(1, 2,1, 2))
```




### 3.3 Inferencia sobre el variograma

Se pueden obtener dos tipos de envolventes (envelopes, i.e. valores máximos y mínimos aproximados) del variograma empírico mediante simulación:

- Bajo la hipótesis de que no hay correlación espacial (obtenidos por permutaciones aleatorias de los datos sobre las posiciones espaciales), para estudiar si hay una dependencia espacial "significativa".
- Bajo un modelo de variograma, para ilustrar la variabilidad del variograma empírico.

```
env.indep <- variog.mc.env(s100, obj.var = vario.b)
```

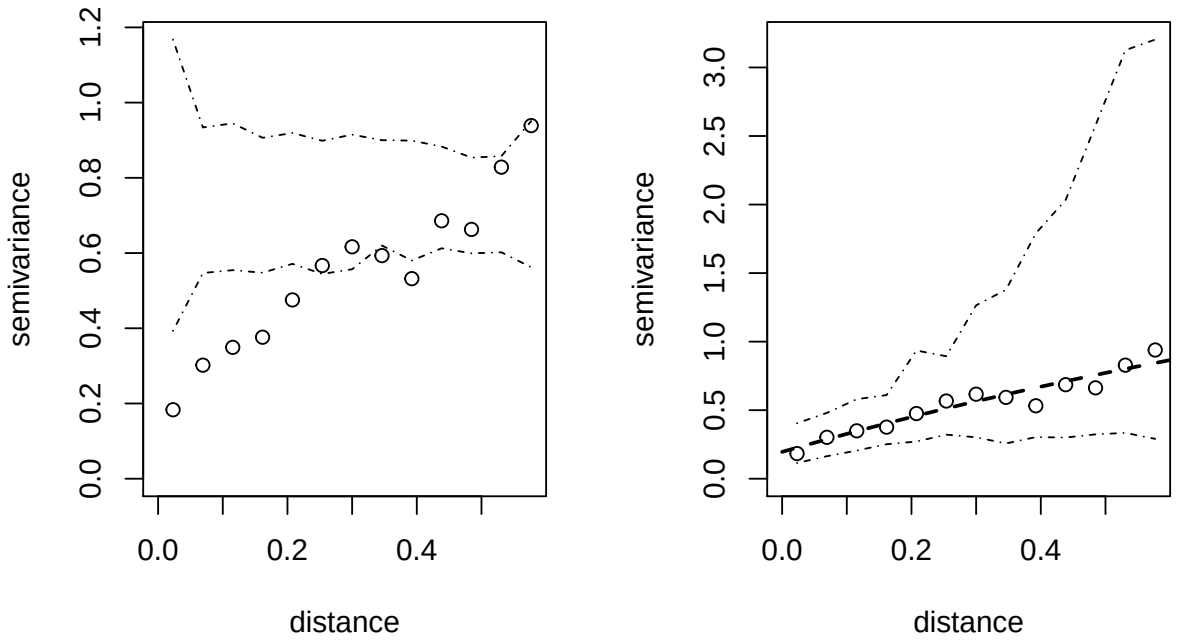
```
## variog.env: generating 99 simulations by permutating data values
## variog.env: computing the empirical variogram for the 99 simulations
## variog.env: computing the envelops
```

```
env.model <- variog.model.env(s100, obj.var = vario.b, model = vario.wls)
```

```
## Warning in if (class(model.pars) == "eyefit") {: la condición tiene
## longitud > 1 y sólo el primer elemento será usado
```

```
## variog.env: generating 99 simulations (with  100 points each) using the function grf
## variog.env: adding the mean or trend
## variog.env: computing the empirical variogram for the 99 simulations
## variog.env: computing the envelops
```

```
oldpar <- par(mfrow = c(1, 2))
plot(vario.b, envelope = env.indep)
plot(vario.b, envelope = env.model)
lines(vario.wls, lty = 2, lwd = 2, max.dist = 0.6)
```




```
par(oldpar)
```

Para estudiar si hay una dependencia espacial "significativa" se puede emplear también la rutina `sm.variogram` del paquete `sm`. Esta rutina devuelve un p-valor para contrastar la hipótesis nula de independencia (i.e. se acepta que hay una dependencia espacial si $p \leq \alpha = 0.05$) y un gráfico en el que se muestra el estimador

empírico robusto, un estimador suavizado y una región de confianza para el variograma suponiendo que el proceso es independiente (i.e. consideraríamos que hay dependencia espacial si el variograma suavizado no está contenido en esa región).

```
# library(sm)
# sm.variogram(s100$coords, s100$data)
```

### 3.4 Validación cruzada

Para verificar si un modelo de variograma describe adecuadamente la dependencia espacial de los datos (p.e. comparar modelos), se emplea normalmente la técnica de validación cruzada, función `xvalid` en `geoR`. Por defecto la validación se realiza sobre los datos eliminando cada observación (y utilizando las restantes para predecir), aunque se puede utilizar un conjunto diferente de posiciones (o de datos) mediante el argumento `location.xvalid` (y `data.xvalid`).

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

```
## xvalid: number of data locations       = 100
## xvalid: number of validation locations = 100
## xvalid: performing cross-validation at location ... 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
## xvalid: end of cross-validation
```

```
summary(xv.wls)
```

```
##                 Min.    1st Qu.     Median         Mean   3rd Qu.     Max.
## errors     -1.429944 -0.4017821 0.04881742 0.0008450629 0.3359677 1.319640
## std.errors -2.110654 -0.7048560 0.07804159 0.0011568059 0.5922810 2.228054
##                   sd
## errors     0.5299818
## std.errors 0.9190753
```

```
xv.reml <- xvalid(s100, model = vario.reml)
```

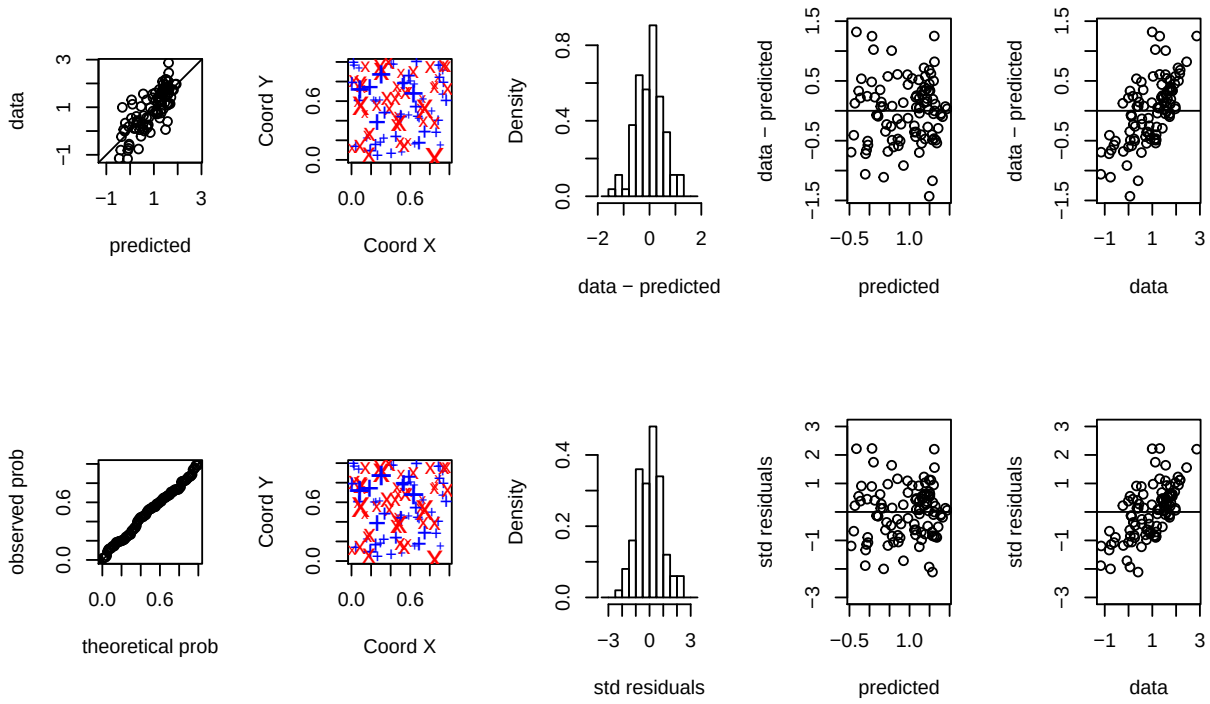
```
## xvalid: number of data locations       = 100
## xvalid: number of validation locations = 100
## xvalid: performing cross-validation at location ... 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
## xvalid: end of cross-validation
```

```
summary(xv.reml)
```

```
##                 Min.    1st Qu.     Median        Mean   3rd Qu.     Max.
## errors     -1.178020 -0.3109277 0.02326020 0.011894019 0.2631596 1.521489
## std.errors -2.419106 -0.7304294 0.07954355 0.009241635 0.5802049 2.690047
##                   sd
## errors     0.4813133
## std.errors 0.9906166
```

Por defecto la función `plot` (`plot.xvalid`) muestra 10 gráficos diferentes (para más información ejecutar `?plot.xvalid`), a grosso modo los cinco primeros se corresponden con residuos simples (valores observados menos predicciones) y los segundos con residuos estandarizados (dividiendo por la raíz cuadrada de la varianza de predicción).

```
oldpar <- par(mfrow = c(2, 5))
plot(xv.wls, ask = FALSE)
```




```
par(oldpar)

# plot(xv.reml)
```

**NOTA**: Para re-estimar los parámetros del modelo cada vez que se elimina una observación (i.e. validar el procedimiento de estimación) añadir la opción `reest = TRUE` (puede requerir mucho tiempo de computación).

### 3.5 Estimación del variograma en procesos no estacionarios

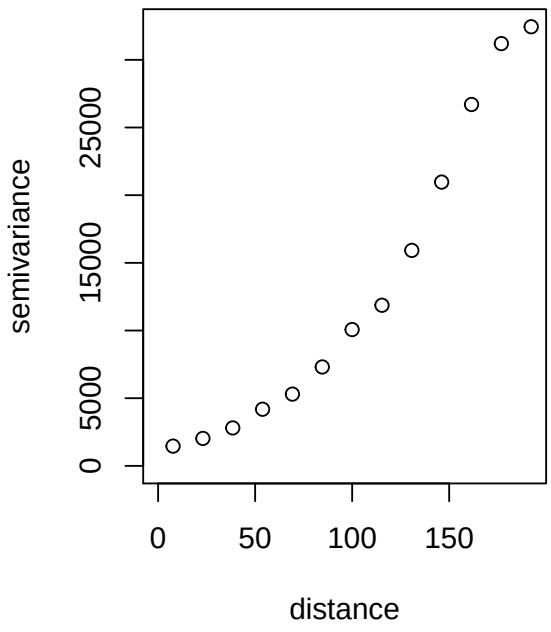
Cuando el proceso no es estacionario (no se puede emplear directamente los estimadores empíricos) hay que eliminar la tendencia para estimar el variograma:

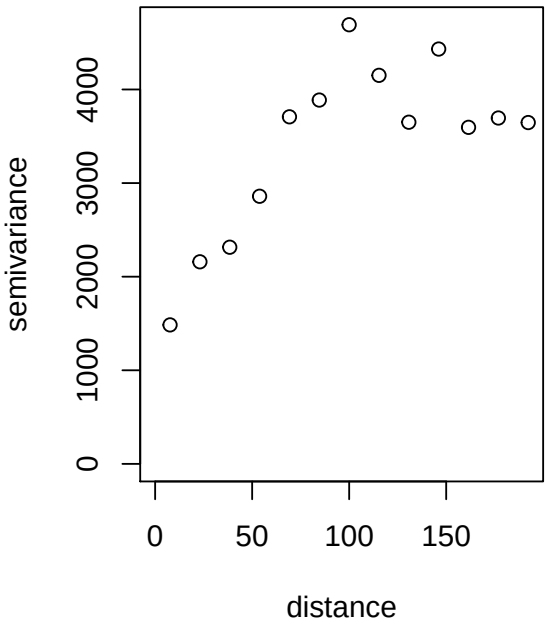
```
oldpar <- par(mfrow=c(1,2))
plot(variog(wolfcamp, max.dist = 200)) # Supone que el proceso es estacionario
```

```
## variog: computing omnidirectional variogram
```

```
plot(variog(wolfcamp, trend = ~coords, max.dist = 200)) # Asume una tendencia lineal en las coordenadas
```

```
## variog: computing omnidirectional variogram
```




```
par(oldpar)
```

## 4. Predicción espacial (kriging)

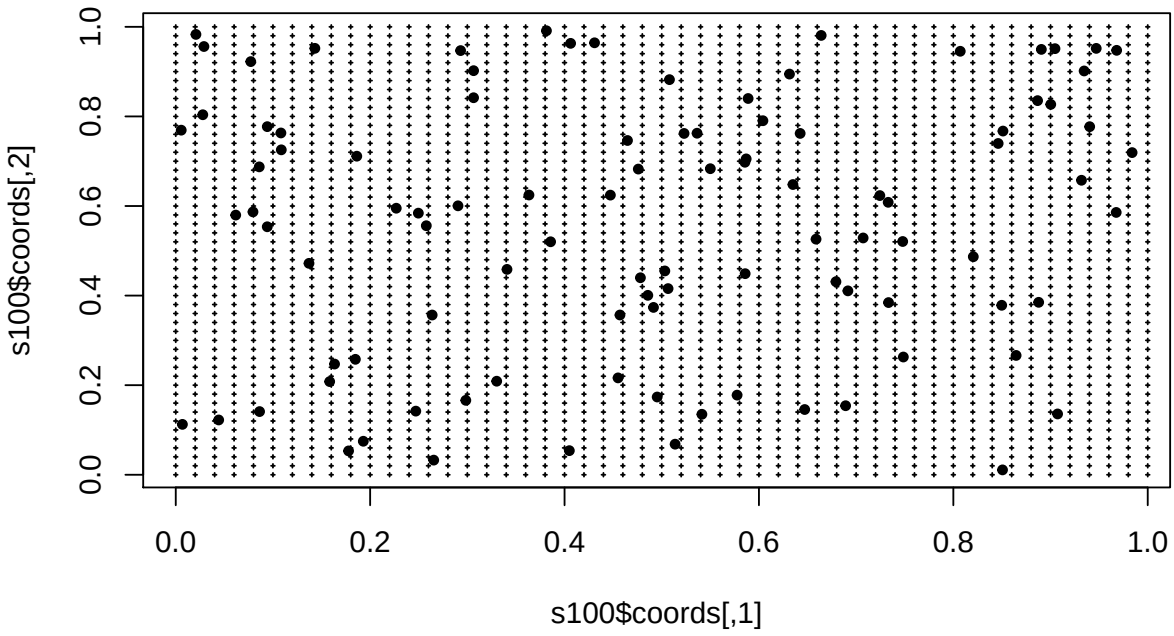
El paquete `geoR` dispone de opciones para los métodos kriging tradicionales, que dependiendo de las suposiciones acerca de la función de tendencia se clasifican en:

- *Kriging simple* (**KS**): media conocida
- *Kriging ordinario* (**KO**): se supone que la media es constante y desconocida.
- *Kriging universal* (**KU**): también denominado kriging con modelo de tendencia, se supone que la media es una combinación lineal (desconocida) de las coordenadas o de otras variables explicativas.

Existen también opciones adicionales para kriging trans-normal (con transformaciones Box-Cox para aproximarse a la normalidad y transformación de nuevo de resultados a la escala original manteniendo insesgadez). También admite modelos de variograma geométricamente anisotrópicos.

Para obtener una rejilla discreta de predicción puede ser de utilidad la función `expand.grid`:

```
pred.grid <- expand.grid(x = seq(0, 1, l = 51), y =  seq(0, 1, l = 51)) #rejilla regular 51x51 en cuadr
# pred.grid <- expand.grid(seq(0, 1, l = 51), seq(0, 1, l = 51)) #rejilla regular 51x51 en cuadrado uni
plot(s100$coords, pch = 20)
points(pred.grid, pch = 3, cex = 0.2)
```




El comando para realizar kriging ordinario con variograma `vario.wls` sería:

```
ko.wls <- krige.conv(s100, loc = pred.grid, krige = krige.control(obj.m = vario.wls))
```

```
## krige.conv: model with constant mean
## krige.conv: Kriging performed using global neighbourhood
```

El resultado es una lista incluyendo predicciones (`ko.wls$predict`) y varianzas kriging (`ko.wls$krige.var`):

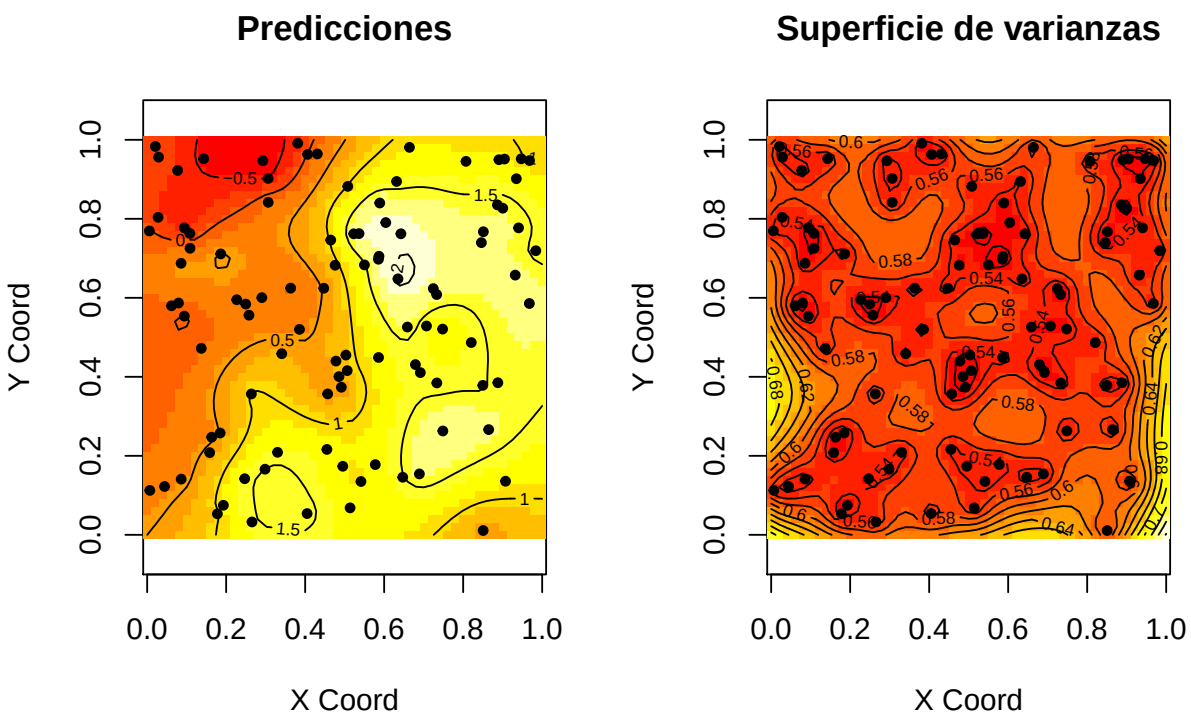
```
names(ko.wls)
```

```
## [1] "predict"      "krige.var"    "beta.est"     "distribution"
## [5] "message"      "call"
```

Para ver todas las opciones de kriging disponibles ejecutar `?krige.control`. Para kriging con vecindario local (archivos de datos grandes) se puede utilizar la función `ksline`.

Para representar las superficies se puede utilizar la función `image`:

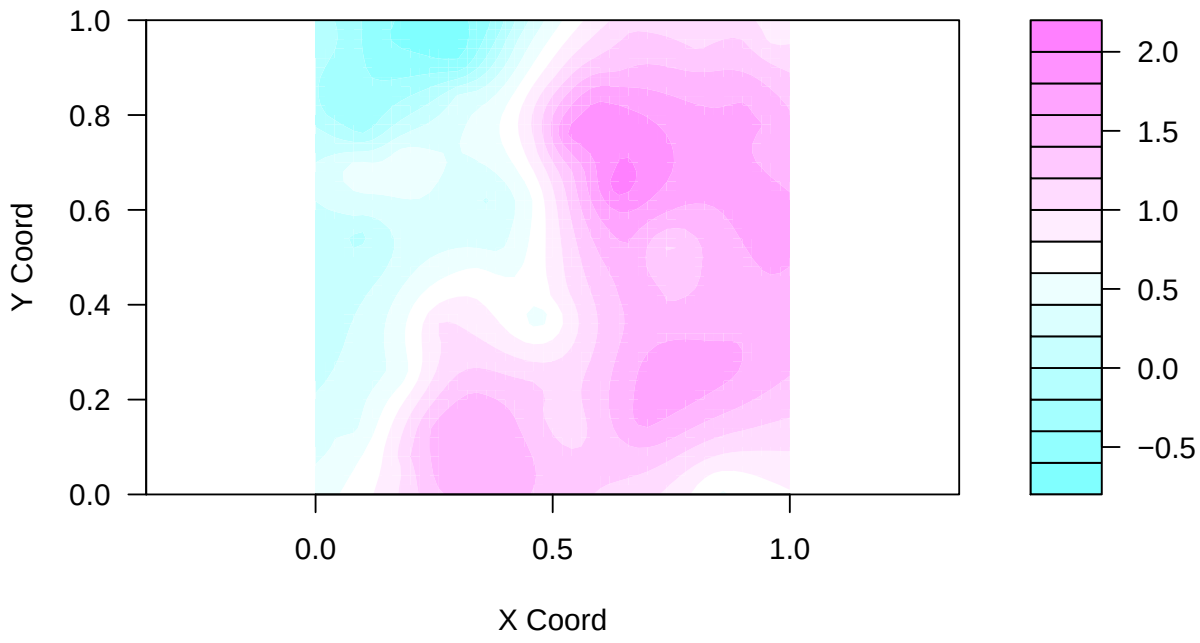
```
oldpar <- par(mfrow = c(1, 2))
image(ko.wls) #superficie de predicción
title("Predicciones")
points(s100$coords, pch=20) #añadir posiciones datos
contour(ko.wls,add=T) #añadir gráfico de contorno

image(ko.wls, val = ko.wls$krige.var) #superficie de varianzas
title("Superficie de varianzas")
points(s100$coords, pch=20)
contour(ko.wls,val=sqrt(ko.wls$krige.var),add=T)
```



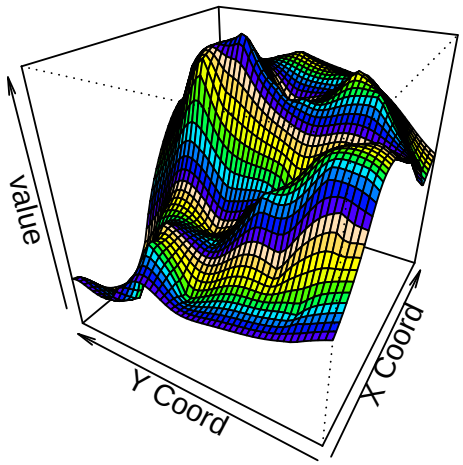

```
par(oldpar)
```

Otras opciones:

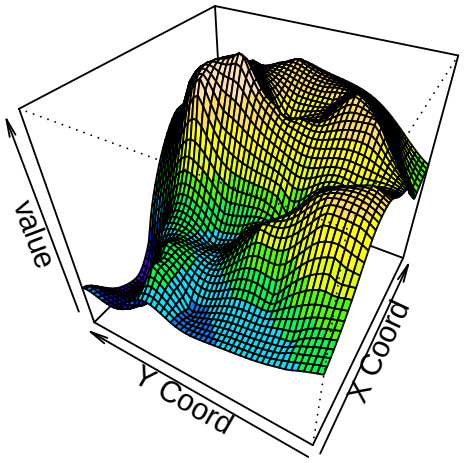
```
contour(ko.wls,filled = TRUE)
```




```
persp(ko.wls, theta=-60, phi=30, col=topo.colors(10))
```



```
fcol <- topo.colors(10)[cut(matrix(ko.wls$pred,nrow=51,ncol=51)[-1,-1],10,include.lowest=TRUE)]
persp(ko.wls, theta=-60, phi=40, col=fcol)
```




---