**TUTORIAL PARA ELABORACIÓN DE MAPAS DE PROPIEDADES EDÁFICAS**

Trabajo de fin de grado de Ignacio Fernández Ruiz

##### **Se sugiere leer el documento “README” antes de comenzar con el tutorial**

## **ÍNDICE**

#### [1.PREPARACIÓN DEL ENTORNO DE TRABAJO](#id1)

#### [2. CARGA DE LOS DATOS CRUDOS](#id2)

#### [3. PREPARACIÓN DEL ENTORNO DE TRABAJO](#id3)

##### [3.1 Generación del mapa base](#id31)

##### [3.2 Adaptación de los datos al mapa base](#id32)

#### [4. NORMALIZACIÓN DE LAS VARIABLES](#id4)

##### [4.1 Metodología de normalización de variables](#id41)

##### [4.2 Ejemplo de normalización de variables](#id42)

###### [4.2.a Normalización de la variable “fosfatasa”](#id42a)

###### [4.2.b Normalización de la variable “contenido en arena”](#id42b)

#### [5. GENERACIÓN DE LA CARTOGRAFÍA EDÁFICA](#id5)

##### [5.1 Metodología de cartografía edáfica](#id51)

##### [5.2 Ejemplo de generación mapas](#id52)

###### [5.2.a Cartografía de la variable “fosfatasa”](#id52a)

###### [5.2.b Cartografía de la variable “contenido en arena”](#id52b)

### |1| PREPARACIÓN DEL ENTORNO DE TRABAJO

En este primer paso, comprobamos si la versión de R está actualizada y cargamos los paquetes necesarios para realizar los análisis.

Con la siguiente función, el programa determina si nuestra versión de R es la 3.6.0 o superior, en caso de que la versión sea anterior se nos mostrará un mensaje informándonos de que debemos actualizarla.

if(getRversion() < "3.6.0") {stop("##########\nLa versión de R que posee está desactualizada\nPor favor, instale la última versión\n##########")}

Con la siguiente función comprobamos si la versión de RStudio es igual o superior a la 1.0.1. En caso contrario, el programa genera un mensaje informando de que tenemos una versión anterior de RStudio y que este programa debe actualizarse.

if(RStudio.Version()$version < "1.0.1"){stop("##########\nLa versión de RStudio que posee es antigua\nPor favor, instale la última versión\n##########")}

En el siguiente paso, comprobamos si los paquetes necesarios para la realización del tutorial están instalados y, en caso contrario, se instalan automáticamente.

PaquetesNecesarios <- c("lattice","sp","gstat","maptools","spatstat","raster","automap")  
installed\_packages <- .packages(all.available = TRUE)  
PaquetesNecesarios2 <- PaquetesNecesarios[!PaquetesNecesarios %in% installed\_packages]  
if(length(PaquetesNecesarios2) > 0){install.packages(PaquetesNecesarios2)}  
stopifnot(all(c(PaquetesNecesarios) %in% .packages(all.available = TRUE)))

Tras comprobar que todos los paquetes están instalados correctamente, utilizaramos la función rm(), que elimina los objetos no necesarios en el futuro.

rm(PaquetesNecesarios, PaquetesNecesarios2, installed\_packages)

Por último, cargamos las librerías de R que vamos a utilizar y que hemos descargado en el paso anterior.

library(lattice)  
library(sp)  
library(gstat)  
library(maptools)  
library(spatstat)  
library(raster)  
library(automap)

### |2| CARGA DE LOS DATOS CRUDOS

En este apartado, cargamos el conjunto de datos brutos con los que vamos a trabajar, y los tratamos para eliminar todos los datos que no nos resulten útiles para el tutorial. En nuestro caso, hemos generado un archivo en formato .txt con los datos, pero existen otros formatos útiles como .csvo .xls.

Para leer los datos, usamos la función read.delim(), indicamos que los datos están separados por tabulaciones, que el símbolo decimal es la , y que nuestro archivo tiene encabezados.

VariablesSuelo <- read.delim("data/Orusco\_suelos.txt", sep="\t", dec=",", header=T)  
load("data/AerialRoot.community.corregido.Rdata")

Para continuar trabajando, debemos homogenizar y limpiar los datos originales o crudos. En nuestro caso los datos ya están muy limpios, pero eliminamos aquellas variables con las que no se va a trabajar. ¡Importante! Nunca se modifican los datos originales y siempre se realiza una copia de seguridad.

En el objeto VariablesSuelo, renombramos la columna 11 y cambiamos COD por codigo\_muestra

colnames(VariablesSuelo)[11] <- "codigo\_muestra"

A continuación, eliminamos las columnas que no nos interesan para este tutorial en concreto, simplificando así el data frame:

VariablesSuelo$Fecha <- NULL  
VariablesSuelo$COND <- NULL  
VariablesSuelo$Altura.elipsoidal <- NULL  
VariablesSuelo$id\_suelo\_raiz <- NULL  
VariablesSuelo$Nº <- NULL  
VariablesSuelo$Marco <- NULL  
VariablesSuelo$ID\_GPS <- NULL

Posteriormente, vamos a transformar los datos al tipo SpatialPointDataFrame. Para ello, asignamos al objeto un nuevo sistema de coordenadas y limitamos el número de decimales que pueden tener los datos a 10.

Como estamos trabajando localmente, no necesitamos utilizar coordenadas globales. Dentro del objeto VariablesSuelo vamos a utilizar unas coordenadas que son relativas al sitio de estudio. Las columnas con los valores de las coordenadas locales se llaman Xlocal y Ylocal. Es posible que otros data sets contengan coordenadas globales, no es ningún problema, simplemente depende del trabajo que se esté realizando.

La función coordinates() del paquete sp nos permite asignar coordenadas a un data.frame transformándolo en SpatialPointDataFrame. Para ello tenemos que indicarle qué columnas son las que queremos que utilice como coordenadas, en este caso se llaman Xlocal y Ylocal.

coordinates(VariablesSuelo) <- ~ Xlocal + Ylocal

Comprobamos que efectivamente el objeto VariablesSuelo es ahora un objeto de clase SpatialPointsDataFrame con la función class()

class(VariablesSuelo)

Por último, limitamos el número de decimales de los datos a 10, para evitar cifras excesivamente largas utilizando la función options()

options(digits=10)

### |3| PREPARACIÓN DEL ENTORNO DE TRABAJO

En este apartado generamos un mapa al que denominamos “mapa base”, y sobre él vamos a dibujar cada variable posteriormente. Para ello, necesitamos indicar el tamaño del área de estudio o plot, las coordenadas donde se localiza y el tamaño de malla que utilizamos para el posterior análisis mediante Kriging.

En primer lugar, vamos a crear un rectángulo con las dimensiones del área de estudio. En nuestro caso, hacemos un rectángulo porque la parcela tiene esa forma, pero el proceso sería similar para áreas con otras geometrías. Para definir el rectángulo usamos las cuatro esquinas de la parcela.

Ya tenemos cargada una matriz denominada esquinas.parcela que contiene esos cuatro puntos. Los nombres de las columnas que contienen las coordenadas X e Y son Xlocal e Ylocal. Utilizamos la función Polygon() para crear el polígono que representa nuestra parcela.

p1 <- Polygon(esquinas.parcela[,1:2])

A continuación, hacemos dos transformaciones: con la función Polygons() convertimos el polígono en un objeto de tipo espacial, y con la función SpatialPolygons() lo convertimos en un objeto de tipo SpatialPolygons.

ps1 <- Polygons(list(p1),1)  
sps1 <- SpatialPolygons(list(ps1))

Podemos consultar la clase del objeto sps1 con la función class()

class(sps1)

Tras crear el polígono que representa nuestra parcela, creamos una rejilla de 0.05 x 0.05 m, utilizando la función spsample() y definiendo el tamaño de la celda o rejilla con el argumento cellsize. Además eliminamos todos los puntos de muestreo que se han quedado fuera de los límites del rectángulo.

grid = spsample(VariablesSuelo, type = "regular", cellsize = c(0.05, 0.05))  
pts1 <- as.data.frame(grid[!is.na(over(grid, sps1,))])

Ya tenemos la malla o rejilla lista. En el siguiente paso modificamos el nombre de las coordenadas X e Y para indicar que son coordenadas locales.

names(pts1) <- c("Xlocal", "Ylocal")

A continuación indicamos con la función coordinates() que esas dos columnas son columnas de coordenadas y transformamos el objeto pts1 en un objeto de clase SpatialPoints

coordinates(pts1) <- c("Xlocal", "Ylocal")  
pts1 <- SpatialPixelsDataFrame(as(pts1, "SpatialPoints"), data=as(pts1, "data.frame"), tolerance=0.077)

Podemos observar la apariencia gráfica del objeto pts1 con la función plot()

plot(pts1)

Por último, asignamos un sistema de coordenadas a la malla:

grid = spsample(VariablesSuelo, type = "regular", cellsize = c(0.05,0.05), proj4string = CRS("+proj=utm +ellps=WGS84 +datum=WGS84"))

### |4| NORMALIZACIÓN DE LAS VARIABLES

#### |4.1| Metodología de normalización de variables

Una vez tenemos preparado el polígono y la rejilla, tenemos que hacer una exploración de los datos de las variables para ver si poseen una distribución normal, puesto que el Kriging asume que los datos siguen dicha distribución. Para ello, realizamos diferentes análisis estadísticos (histograma, gráfico cuantil-cuantil y Test de Shapiro) para observar cómo se adaptan los datos de cada variable a una distribución normal. En aquellos casos en que los datos no sigan dicha distribución, los transformamos para ajustar nuestros datos a una distribución normal.

#### |4.2| Ejemplo de normalización de variables:

Para mostrar cómo es el proceso de normalización de las diferentes variables, vamos a poner como ejemplo las variables fosfatasa y pH. En estos ejemplos, estudiamos cómo es su histograma (mediante la función hist()), su gráfico cuantil-cuantil (mediante la función qqnorm()) y qué p-valor ofrece el test de Shapiro (mediante la función shapiro.test()). Si los valores no poseen una distribución normal, procedemos a realizar transformaciones para adaptarlos.

##### |4.2.a| Normalización de la variable “fosfatasa”

Comencemos con la variable fosfatasa:

hist(VariablesSuelo$FOSF)  
qqnorm(VariablesSuelo$FOSF)   
shapiro.test((VariablesSuelo$FOSF))

Esta variable no muestra un patrón normalizado y su p-valor es muy bajo. Por lo tanto, no es válida y debemos proceder a normalizarla. Probemos realizando el logaritmo de la variable:

hist(log(VariablesSuelo$FOSF))   
qqnorm(log(VariablesSuelo$FOSF))  
shapiro.test(log(VariablesSuelo$FOSF))

Ahora sí, la variable fosfatasa muestra un patrón normalizado y su p-valor es aceptable. Por lo tanto, será la que vamos a utilizar para realizar el Kriging.

##### |4.2.b| Normalización de la variable “contenido en arena”

Ahora, realizamos las mismas comprobaciones, pero con la variable relacionada con el contenido de arena en el suelo:

hist(VariablesSuelo$Arena)  
qqnorm(VariablesSuelo$Arena)   
shapiro.test((VariablesSuelo$Arena))

La variable arena, muestra un patrón normalizado y su p-valor es aceptable sin necesidad de realizar la normalización. Por lo tanto, utilizamos sus valores originales para realizar el Kriging.

### |5| GENERACIÓN DE CARTOGRAFÍA EDÁFICA

En esta última fase, vamos a realizar los mapas de cada una de las variables utilizando el método de estimación geoestadístico denominado Kriging. Esta técnica de interpolación, utiliza el modelo de variograma para poder estimar el resto de puntos intermedios donde no se tiene un dato real recogido directamente del campo.

#### |5.1| Metodología de cartografía edáfica

Existen dos formas para la realización de cartografía edáfica utilizando el método de Kriging en R : Por un lado, se puede utilizar la función autokriging(), donde el propio programa estadístico R realiza los cálculos, elige el sistema con una mejor relación con la realidad y genera un gráfico con el variograma y el mapa de la variable. Sus estimaciones, aunque bastante precisas, suelen incurrir en cierto error, aunque este puede ser asumible dependiendo del grado de precisión que desee el estudio.

La otra forma de realizar el Kriging, es ejecutarlo de forma manual. Este método exige que el usuario evalúe qué modelo matemático es el adecuado y se genera el Kriging partiendo del modelo elegido y posteriormente, se visualiza cómo quedaría gráficamente mediante la función plot().

Para conocer qué modelo matemático es el más adecuado para la variable que se quiere mapear, utilizamos la función autofitVariogram(). En ella, determinamos qué modelo matemático de los cinco disponibles vamos a utilizar (modelo exponencial Exp, modelo esférico Sph, modelo gaussiano Gau, modelo lineal Lin o la parametrización de Stein Ste) y si lo hacemos teniendo en cuenta la tendencia xlocal o no (representada en la función con un 1.

Ejemplo:

autofitVariogram(log(GLUC) ~ 1, VariablesSuelo, model = c("Exp"))$sserr

Esta línea de código informa cómo se adapta el variograma de datos de la glucosidasa GLUC al modelo exponencial Exp sin ninguna tendencia 1. La función, nos da como respuesta un valor de semivarianza. Cuanto más cercano a 0 sea este valor, indica un mejor ajuste al modelo seleccionado.

Para procesar todos los modelos a la vez y ahorrar tiempo, creamos una matriz vacía donde poner los resultados de semivarianza de los 5 modelos estudiados con y sin tendencia. De esta forma, podremos observar qué valor de semivarianza es menor (es decir, a qué modelo se ajustan mejor los datos) y utilizar ese modelo matemático para producir la cartografía mediante el Kriging.

#### |5.2| Ejemplos de generación de mapas

Para ilustrar cómo se generan los diferentes mapas edáficos, en este apartado se realizan dos mapas de las variables fosfatasa FOSF y contenido en arena Arena. A continuación se muestra cómo es el proceso de creación de los mapas usando tanto el autokriging como el Kriging manual.

##### |5.2.a| Cartografía de la variable “fosfatasa”

Autokriging de fosfatasa: Sin tendencia:

Autok.FOSF.ST <- autoKrige(log(FOSF) ~ 1, VariablesSuelo, pts1 )

Visualizamos como es la representación gráfica sin tendencia:

plot(Autok.FOSF.ST)

Con tendencia:

Autok.FOSF.CT <- autoKrige(log(FOSF) ~ Xlocal, VariablesSuelo, new\_data=pts1 )

Visualizamos como es la representación gráfica con tendencia:

plot(Autok.FOSF.CT)

###### Kriging manual de fosfatasa:

Primero, se debe generar una matriz donde exponer las semivarianzas de cada modelo. La siguiente línea de código produce una matriz de 2x5 y nombra las columnas y las filas con los nombres de los modelos y la tendencia respectivamente:

MatrizFOSF <- matrix(NA,2,5)  
colnames(MatrizFOSF) <- c("Exponencial","Esferico","Gausiano","Lineal","Ste")  
rownames(MatrizFOSF) <- c("Sin tendencia", "Con tendencia")

Posteriormente, se rellena la matriz con el valor producido del ajuste de la variable a cada modelo. Es imprescindible asegurarse de introducir los datos en el mismo orden que hemos facilitado a la matriz en el anterior paso:

Sin tendencia (se debe escribir un 1, para indicar que no hay tendencia):

MatrizFOSF[1,1] <- autofitVariogram(log(FOSF) ~ 1, VariablesSuelo, model = c("Exp"))$sserr  
MatrizFOSF[1,2] <- autofitVariogram(log(FOSF) ~ 1, VariablesSuelo, model = c("Sph"))$sserr  
MatrizFOSF[1,3] <- autofitVariogram(log(FOSF) ~ 1, VariablesSuelo, model = c("Gau"))$sserr  
MatrizFOSF[1,4] <- autofitVariogram(log(FOSF) ~ 1, VariablesSuelo, model = c("Lin"))$sserr  
MatrizFOSF[1,5] <- autofitVariogram(log(FOSF) ~ 1, VariablesSuelo, model = c("Ste"))$sserr

Con tendencia (debemos utilizarXlocal como tendencia):

MatrizFOSF[2,1] <- autofitVariogram(log(FOSF) ~ Xlocal, VariablesSuelo,model = c("Exp"))$sserr  
MatrizFOSF[2,2] <- autofitVariogram(log(FOSF) ~ Xlocal, VariablesSuelo,model = c("Sph"))$sserr  
MatrizFOSF[2,3] <- autofitVariogram(log(FOSF) ~ Xlocal, VariablesSuelo,model = c("Gau"))$sserr  
MatrizFOSF[2,4] <- autofitVariogram(log(FOSF) ~ Xlocal, VariablesSuelo,model = c("Lin"))$sserr  
MatrizFOSF[2,5] <- autofitVariogram(log(FOSF) ~ Xlocal, VariablesSuelo,model = c("Ste"))$sserr

El modelo que se ajuste mejor es el que tenga un valor de semivarianza menor. Con la siguiente función se averigua qué modelo nos aporta la semivarianza mínima:

which((MatrizFOSF) == min(MatrizFOSF), arr.ind=TRUE)

En este caso, el modelo gaussiano sin la utilización de la tendencia es el que más se aproxima a nuestra serie de datos, por lo tanto, se debe adaptar los valores de la variable a este modelo con la función autofitvariogram().

v.fitFOSFgauST = autofitVariogram(log(FOSF) ~ 1, VariablesSuelo, model = c("Gau"))$var\_model

Con nuestros datos ajustados al modelo matemático idóneo, procedemos a realizar el Kriging con la función Krige():

FOSF.mapa <- krige(log(FOSF) ~ 1, VariablesSuelo, pts1, v.fitFOSFgauST)

Por último, generamos el mapa de fosfatasa con la función plot():

plot(FOSF.mapa, main= "FOSFATASA")

##### |5.2.b| Cartografía de la variable “contenido en arena”

Autokriging de Arena:

Sin tendencia:

Autok.Arena.ST <- autoKrige((Arena) ~ 1, VariablesSuelo, pts1 )

Visualizamos la representación gráfica sin tendencia:

plot(Autok.Arena.ST)

Con tendencia:

Autok.Arena.CT <- autoKrige((Arena) ~ Xlocal, VariablesSuelo, new\_data=pts1 )

Visualizamos la representación gráfica con tendencia:

plot(Autok.Arena.CT)

###### Kriging manual de Arena:

Primero, se debe generar una matriz donde exponer las semivarianzas de cada modelo. La siguiente línea de código produce una matriz de 2x5 y nombra las columnas y las filas con los nombres de los modelos y la tendencia respectivamente:

MatrizArena <- matrix(NA,2,5)  
colnames(MatrizArena) <- c("Exponencial","Esferico","Gausiano","Lineal","Ste")  
rownames(MatrizArena) <- c("Sin tendencia", "Con tendencia")

Posteriormente, se rellena la matriz con el valor producido del ajuste de la variable a cada modelo. Es imprescindible asegurarse de introducir los datos en el mismo orden que hemos facilitado a la matriz en el paso anterior:

Sin tendencia (se debe escribir un 1, para indicar que no hay tendencia):

MatrizArena[1,1] <- autofitVariogram((Arena) ~ 1, VariablesSuelo, model = c("Exp"))$sserr  
MatrizArena[1,2] <- autofitVariogram((Arena) ~ 1, VariablesSuelo, model = c("Sph"))$sserr  
MatrizArena[1,3] <- autofitVariogram((Arena) ~ 1, VariablesSuelo, model = c("Gau"))$sserr  
MatrizArena[1,4] <- autofitVariogram((Arena) ~ 1, VariablesSuelo, model = c("Lin"))$sserr  
MatrizArena[1,5] <- autofitVariogram((Arena) ~ 1, VariablesSuelo, model = c("Ste"))$sserr

Con tendencia (Utilizamos Xlocal como tendencia):

MatrizArena[2,1] <- autofitVariogram((Arena) ~ Xlocal, VariablesSuelo,model = c("Exp"))$sserr  
MatrizArena[2,2] <- autofitVariogram((Arena) ~ Xlocal, VariablesSuelo,model = c("Sph"))$sserr  
MatrizArena[2,3] <- autofitVariogram((Arena) ~ Xlocal, VariablesSuelo,model = c("Gau"))$sserr  
MatrizArena[2,4] <- autofitVariogram((Arena) ~ Xlocal, VariablesSuelo,model = c("Lin"))$sserr  
MatrizArena[2,5] <- autofitVariogram((Arena) ~ Xlocal, VariablesSuelo,model = c("Ste"))$sserr

El modelo que se ajuste mejor es el que tenga un valor de semivarianza menor. Con la siguiente función se averigua qué modelo nos aporta la semivarianza mínima:

which((MatrizArena) == min(MatrizArena), arr.ind=TRUE)

En este caso, el modelo de parametrización de Stein sin la utilización de la tendencia es el que más se aproxima a nuestra serie de datos, por lo tanto, se debe adaptar los valores de la variable a este modelo con la función autofitvariogram().

v.fitArenasteST = autofitVariogram((Arena) ~ 1, VariablesSuelo, model = c("Ste"))$var\_model

Con nuestros datos ajustados al modelo matemático idóneo, procedemos a realizar el Kriging con la función Krige():

Arena.mapa <- krige((Arena) ~ 1, VariablesSuelo, pts1, v.fitArenasteST)

Por último, generamos el mapa de contenido en arena con la función plot():

plot(Arena.mapa, main= "CONTENIDO EN ARENAS")