

---

## 9. Elaboración de mapas

---

R es famoso por la producción de gráficas muy sofisticadas. Varios paquetes permiten elaborar mapas y figuras. Vamos a crear unos mapas muy simples para representar la densidad poblacional y el relieve (script9.R). Para ello, vamos primero a importar el mapa de los estados en el cual calculamos la densidad poblacional de cada estado (capítulo 5). Esta información se encuentra en la columna "dens" de la tabla de atributos.

```
library(maptools)

## Checking rgeos availability: TRUE

library(RColorBrewer)
library(classInt)
library(sf)
# Determina la ruta del espacio de trabajo
setwd("/home/jf/recursos-mx")
mx <- st_read("mx_lcc.shp")

## Reading layer `mx_lcc' from data source
`/home/jf/recursos-mx/mx_lcc.shp' using driver `ESRI Shapefile'
## Simple feature collection with 32 features and 9 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:            xmin: 907821.8 ymin: 319149.1 xmax: 4082993 ymax: 2349609
## epsg (SRID):    NA
## proj4string:    +proj=lcc +lat_1=17.5 +lat_2=29.5 +lat_0=12 +lon_0=-102
##                  +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs

# Nombres de las columnas de la tabla de atributos
names(mx)

## [1] "CveEdo"     "NOM_ENT"    "AreaEdo"    "Estado"      "Poptotal"   "Hombre"
## [7] "Mujer"       "dens"        "IM"          "geometry"
```

Los valores de densidad varían de 13 a casi 6000 habitantes por km<sup>2</sup> en la Ciudad de México. Vamos a representar la densidad a través de una escala de siete colores de amarillo a rojo (el valor de siete es arbitrario). Para ello, vamos a generar una paleta de siete colores usando la función **brewer.pal()** del paquete **RColorBrewer** y reclasificar los valores de densidad en siete rangos. Esta reclasificación se hace determinando los umbrales entre las siete categorías de forma automática usando la función **classIntervals()** del paquete **classInt**. En este caso usamos el método de cuantiles, es decir que se busca determinar los umbrales para tener el mismo número de observaciones (estados) en cada categoría. Finalmente, calculamos un vector que indica el código de color que corresponde a cada estado.

```

# Valores de densidad de los estados
print(mx$dens)

## [1] 213.179418 42.905160 8.610068 14.358966 18.241013
## [6] 113.042276 65.160616 13.792839 5954.384955 13.370325
## [11] 180.830811 53.311759 129.028259 94.280680 682.777950
## [16] 74.636623 365.727309 39.003692 73.214940 40.463692
## [21] 169.237737 157.727966 29.750704 42.735927 48.725636
## [26] 14.722608 90.651620 41.152106 294.398137 106.955498
## [31] 49.467001 20.014394

## Paleta de colores y categorización de la variable densidad
numclass <- 7 # número de categorías
# Generación de 7 colores en la escala amarillo a rojo
colores <- brewer.pal(numclass, "YlOrRd")
print(colores) # códigos de los colores

## [1] "#FFFFB2" "#FED976" "#FEB24C" "#FD8D3C" "#FC4E2A" "#E31A1C"
## [7] "#B10026"

# Determinación de los umbrales entre categorías (método cuantil)
var <- mx$dens
brks<-classIntervals(var, n=numclass, style="quantile")
brks <- brks$brks
print(brks)

## [1] 8.610068 16.230496 40.255120 48.937454 74.230428
## [6] 115.325988 199.315730 5954.384955

class(brks)

## [1] "numeric"

class(mx$dens)

## [1] "numeric"

# Determina qué color corresponde a cada valor de densidad
codigos_num <- findInterval(var, brks, all.inside=TRUE)
print(codigos_num)

## [1] 7 3 1 1 2 5 4 1 7 1 6 4 6 5 7 5 7 2 4 3 6 6 2 3 3 1 5 3 7 5 4 2

codigos_color <- colores[codigos_num]
print(codigos_color)

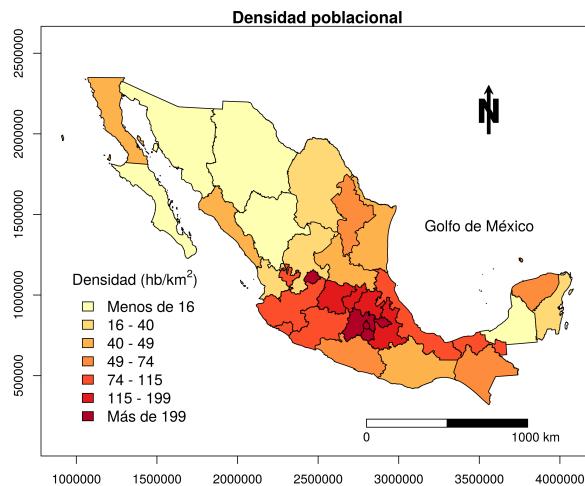
## [1] "#B10026" "#FEB24C" "#FFFFB2" "#FFFFB2" "#FED976" "#FC4E2A"
## [7] "#FD8D3C" "#FFFFB2" "#B10026" "#FFFFB2" "#E31A1C" "#FD8D3C"
## [13] "#E31A1C" "#FC4E2A" "#B10026" "#FC4E2A" "#B10026" "#FED976"
## [19] "#FD8D3C" "#FEB24C" "#E31A1C" "#E31A1C" "#FED976" "#FEB24C"
## [25] "#FEB24C" "#FFFFB2" "#FC4E2A" "#FEB24C" "#B10026" "#FC4E2A"
## [31] "#FD8D3C" "#FED976"

```

```

png(file="/home/jf/recursos-mx/mexico_densidad.png", height=16,
    width=20, units="cm", res=400) # Editar la ruta para salvar la figura
par(mar=c(10.1,4.1,4.1,10.1))
plot(mx["dens"], col=codigos_color, axes=T,main="")
## Pone un título
title(main="Densidad poblacional",cex.main=1.3)
# Pone texto en ciertas coordenadas
text(3500000,1500000,"Golfo de México",pos = 1,cex = 1.2)
# Pone el símbolo de Norte
SpatialPolygonsRescale(layout.north.arrow(1), offset= c(3500000,2000000),
                        scale = 300000,plot.grid=F)
# Leyenda: Genera texto de la leyenda (rangos de valores)
rangos <- leglabs(as.vector(round(brks, digit = 0)),under="Menos de",
                  over="Más de")
legend("bottomleft", inset=.05, title=expression("Densidad (hb/km"^(2*"))"),
       legend=rangos, fill=colores, horiz=FALSE,
       box.col = NA, cex = 1.2)
# Escala gráfica
SpatialPolygonsRescale(layout.scale.bar(), offset= c(2800000,180000),
                        scale= 1000000, fill=c("transparent", "black"), plot.grid= F)
text(2800000,120000,"0"); text(3850000,120000,"1000 km") # texto de la escala
par(mar=c(5.1,4.1,4.1,2.1))
dev.off()

```



En este segundo ejemplo, realizamos un mapa con base en un archivo raster utilizando una de las paletas de color disponible en R sin utilizar el paquete **RColorBrewer**.

```

# Importa y recorta el DEM
extension <- extent(2000000,3000000,500000,1500000)
dem <- crop(raster("DEM_Mx.tif"), extension)
# Crea una escala de color
colores <- terrain.colors(7, alpha = 1)
# Elabora el mapa

```

```

png(file="dem-mx.png", height=20, width=20, units="cm", res=400)
plot(dem, col=colores, breaks = c(0, 500, 1000, 1500, 2000, 2500, 3000, 5300),
     main = "Elevación")
SpatialPolygonsRescale(layout.scale.bar(), offset= c(2100000,600000),
                       scale= 300000, fill=c("transparent", "black"), plot.grid= F)
text(2100000,580000,"0"); text(2430000,580000,"300 km")
dev.off()

```

En este último ejemplo, realizamos una imagen de sombreado con base en el modelo digital de elevación, lo cual desplegamos en tonos de grises. En seguida, se añadió el raster de elevación con transparencia (opción alpha).

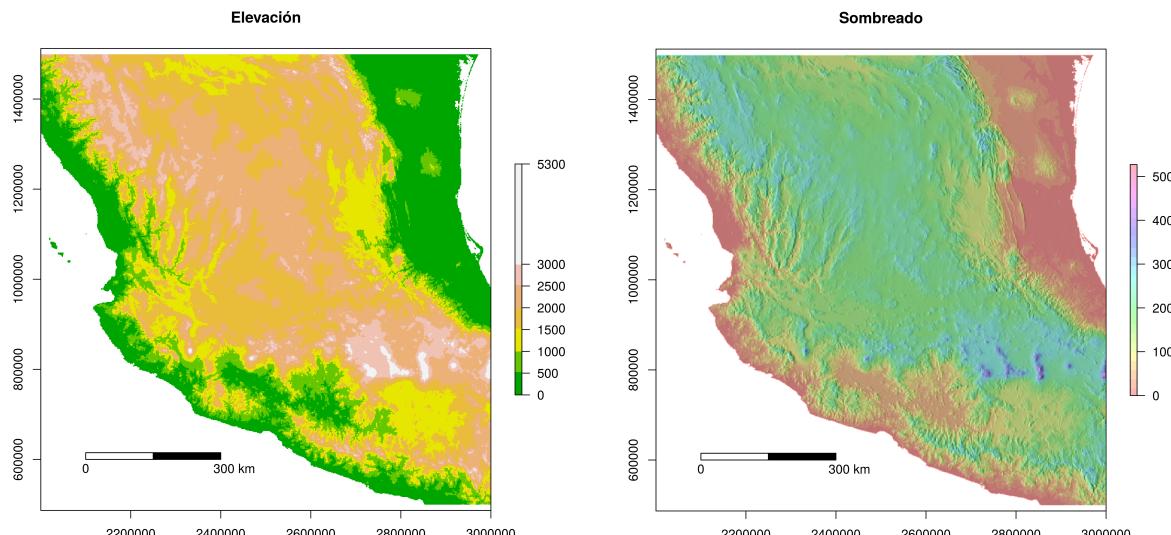
```

# Preparación de un sombreado
pend <- terrain(dem, opt = "slope")
orient <- terrain(dem, opt = "aspect")
sombra <- hillShade(pend, orient, 40, 270)

# Plotea el sombreado y la elevación con transparencia
png(file="sombreado-mx.png", height=20, width=20, units="cm", res=400)
plot(sombra, col = grey(0:100/100), legend = FALSE, main = "Sombreado")
plot(dem, col = rainbow(25, alpha = 0.3), add = TRUE)
SpatialPolygonsRescale(layout.scale.bar(), offset= c(2100000,600000),
                       scale= 300000, fill=c("transparent", "black"), plot.grid= F)
text(2100000,580000,"0"); text(2430000,580000,"300 km")
dev.off()

```

Existen muchas otras opciones para la elaboración de mapas en R, se pueden mencionar la función **spplot** de **sp** y los paquetes **Choroplethr** (Lamstein & Johnson, 2018), **PrettyMapr** (Dunnington, 2017) y **tmap** (Tennekes *et al.*, 2017). **RgoogleMaps** (Loecher, 2016) y **Rosm** (Dunnington & Giraud, 2017) permiten utilizar como fondo los mapas de GoogleMap, Open Street y Bing.



---

# 10. Poniendo R a interactuar con QGIS y Dinamica

---

Varios programas computacionales de manejo de información espacial permiten la interacción con R. En este capítulo vamos a revisar los mecanismos implementados en dos de ellos: el Sistema de Información Geográfica QGIS y la plataforma de modelación espacial Dinamica EGO.

## 10.1 Sistema de Información Geográfica QGIS

El programa Q-GIS es un sistema de información geográfica de código abierto que permite el manejo de datos en formato raster y vectorial a través de las librerías GDAL y OGR (<https://docs.qgis.org/>). Q-GIS puede ejecutar scripts de R utilizando Python, y los resultados del script pueden ser incorporados como capa de información en Q-GIS. Para ello, se debe configurar Q-GIS en Procesos>Opciones>Proveedores>Rscripts. Aparecerá una ventana, rellene los cuadros de Activar y Use64bitversion e indicar la ruta (*path*) del ejecutable de R. Pulse Aceptar, cerrar Q-GIS y volver a abrirlo<sup>1</sup>. Para escribir un nuevo script, seleccionar Procesos>cajadeherramientas>Rscripts>Herramientas>CreatenewRscript.

Al inicio del script tenemos de definir los argumentos de las funciones de R en unas líneas que inician con el doble símbolo ## y que son utilizadas para crear la interfaz gráfica. Por ejemplo, el script a continuación permite calcular el histograma de un raster abierto en el proyecto QGIS. Genera una interfaz que permite escoger el mapa raster y la ruta para salvar el histograma (Figura 6).

```
##Layer = raster
##showplots
hist(as.matrix(Layer),main="Histograma",xlab="Mapa")
```

Salvar el script (por *default* se salva en *User R Scripts*). El script permite calcular el histograma de un raster abierto en el proyecto de QGIS. Al correrlo con un doble clic sobre su nombre, genera una interfaz que permite escoger el mapa raster y la ruta para salvar el histograma (Figura 6).

En este segundo ejemplo, el script tiene como entradas un mapa de polígonos y un número cuyo valor, *por default*, es 10 y genera puntos aleatorios distribuidos en la cobertura de polígonos. La línea ##sp=group permite organizar los scripts en grupos. En este caso, el script se almacena en una carpeta llamada sp. La interfaz gráfica asociada al script permite capturar el nombre de la cobertura de polígonos, el número de puntos y el nombre y ruta de la cobertura de puntos aleatorios (Figura 7)

```
##polyg=vector
##numpoints=number 10
##output=output vector
##sp=group
```

---

<sup>1</sup>En Windows es posible que se genere un error "Wrong value for parameter "Msys folder". La solución a este problema se encuentra en <https://gis.stackexchange.com/questions/191590/qgis-2-14-1-lastools-install-error-wrong-value-for-parameter-msys-folder>

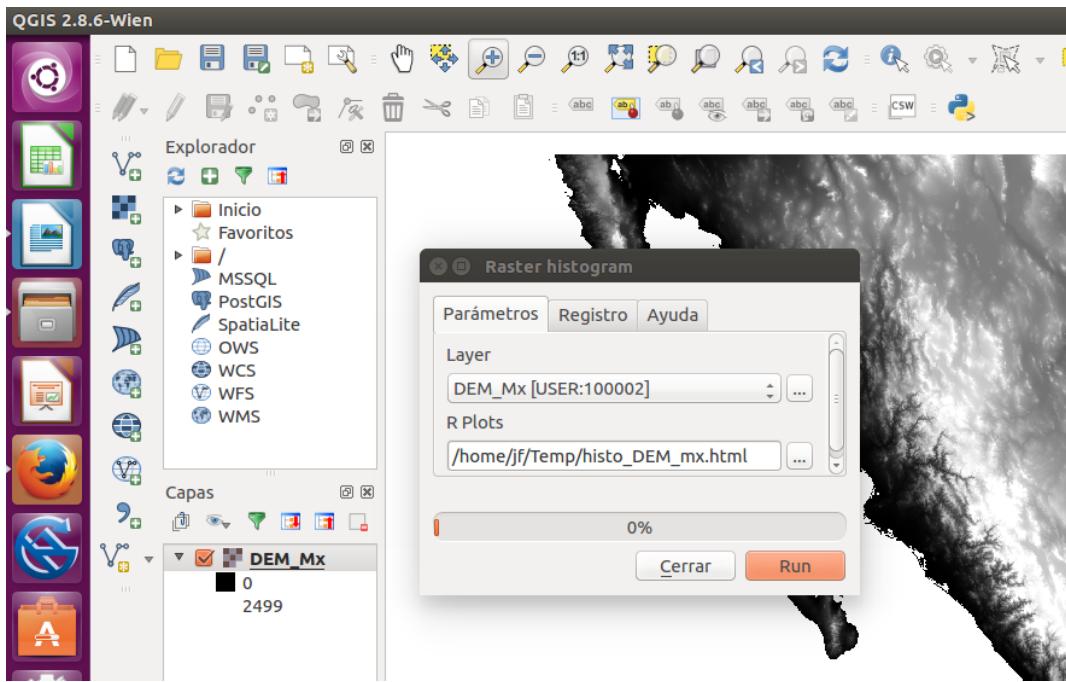


Figura 6. Interfaz gráfica

```
pts <- spsample(polyg, numpoints, type="random")
output=SpatialPointsDataFrame(pts, as.data.frame(pts))
```

Es por lo tanto, posible generar interfaces gráficas que permiten a usuarios no familiarizados con R ejecutar scripts de forma simple e interactiva. Por otro lado, el paquete RQGIS permite correr procesos de QGIS desde R.

## 10.2 Plataforma de modelación Dinamica EGO

Dinamica EGO es una plataforma de modelación espacio-temporal muy flexible y amigable para el usuario. Ha sido utilizada para desarrollar una gran variedad de modelos medioambientales como modelos de simulación de cambio de uso / cubierta del suelo, de crecimiento urbano, de incendios, de renta, entre otros (Mas *et al.*, 2014; Rodrigues & Soares-Filho, 2018). Se puede obtener gratuitamente este programa, disponible para Windows y Linux, en <http://csr.ufmg.br/dinamica/downloads/>.

A partir de su versión 4, Dinamica tiene módulos especialmente diseñados para integrar scripts de R dentro del proceso de modelación (*librería Integration*). Existen dos formas de hacer interactuar Dinamica y R:

- Realizar una sesión compartida entre R y Dinamica en la cual los dos programas corren de forma paralela y se intercambian información (valores, tablas).
- Ejecutar un script de R dentro de Dinamica gracias a la función ***Calculate R expression***.

En esta sección, vamos a explorar esta segunda opción que es más simple y generalmente más eficiente que la primera. Permite ejecutar uno o varios scripts de R dentro del flujo de operaciones de un modelo de Dinamica.

Se debe primero, instalar algunos paquetes de R que se encuentran en los repositorios (Rcpp, RcppProgress, rbenchmark, inline)

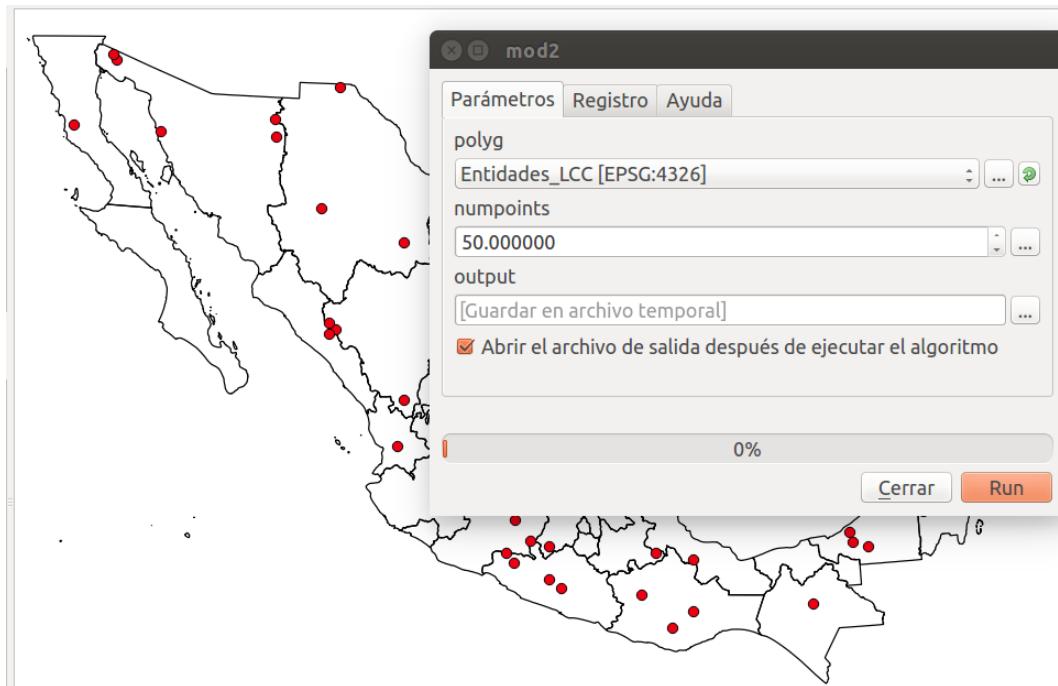


Figura 7. Interfaz gráfica del segundo modelo

```
install.packages(c("Rcpp", "RcppProgress", "rbenchmark", "inline"))
```

Se debe también instalar un paquete especialmente diseñado para la interacción entre R y Dinamica. Debe obtenerse de la página de DINAMICA<sup>2</sup> debido a que no se halla en los repositorios de R e instalarse localmente con RStudio o en comando de línea (modificar la ruta indicando la carpeta donde se encuentra el archivo). Los usuarios de Windows deben tener Rtools instalado en su máquina para poder realizar esta operación (ver sección 1.3).

```
install.packages("/home/jf/dinamica_1.0.2.tar.gz", repos=NULL, type="source")
```

Se debe también indicar a Dinamica donde encontrar el ejecutable de R llamado Rscript (Rscript.exe en Windows). Para ello, capturar la ruta de acceso a este archivo en una ventana que se obtiene en Tools > Options > Integration (ver figura 8). En windows la ruta sería parecida a C:\Program Files\R\R-3.3.2\bin\Rscript.exe.

En el primer ejemplo a continuación (Modelo *elabora\_hist\_R.egoml*), Dinamica va pasar a un script de R la tabla de los valores de una imagen para elaborar un histograma y salvar la figura (Figura 9). Esta tabla entra en el functor **Calculate R Expression** como en cualquier otro functor de Dinamica (Number table: Table #1, es decir que internamente la tabla se llama t1). Con edit Functor, se puede editar el script de R. En la primera línea, se asigna esta tabla al objeto tabla. Las líneas siguientes son exclusivamente de R. Permite asignar nombres a las columnas de la tabla y realizar una gráfica que se salva en formato png.

```
tabla <- t1
colnames(tabla) <- c("valores", "n")
png("histograma.png")
plot(tabla$n, tabla$valores, main="Histograma banda 2", xlab="", 
     ylab="Número de celdas")
dev.off()
```

<sup>2</sup>[http://www.csr.ufmg.br/dinamica/dokuwiki/lib/exe/fetch.php?media=dinamica\\_1.0.2.tar.gz](http://www.csr.ufmg.br/dinamica/dokuwiki/lib/exe/fetch.php?media=dinamica_1.0.2.tar.gz)

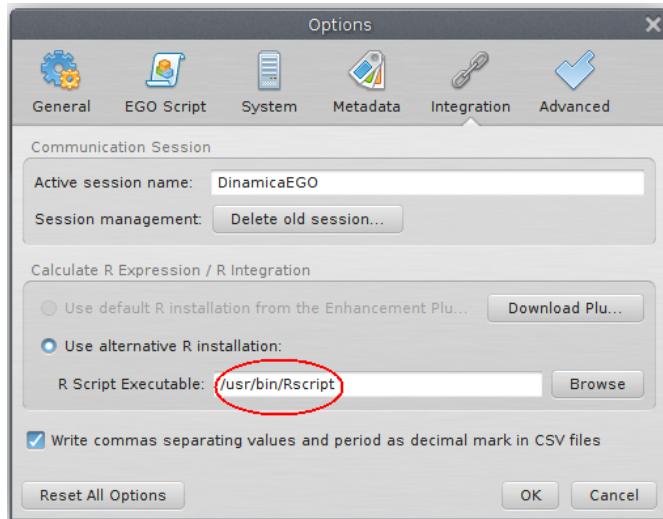


Figura 8. Captura de la ruta del ejecutable de R

En este segundo ejemplo (Modelo *regresion\_lineal.egoml*, disponible en la carpeta de recursos del libro), disponible en la carpeta Dinamica), vamos a ver cómo Dinamica puede recuperar datos arrojados por R. En este caso, el modelo lleva a cabo un análisis de la relación entre los valores de dos bandas de una imagen de satélite (Figura 10). Para ello, realiza un muestreo aleatorio de pixeles, ajusta un modelo de regresión para explicar los valores de una banda con base en la otra. Luego calcula, pixel a pixel, los residuales (diferencias entre los valores de una banda calculada por el modelo y los valores observados).

El modelo de Dinamica carga ambas imágenes y extrae el número de filas y columnas, estos valores entran en el primer script de R junto con el tamaño de muestreo deseado. En R la función *runif()* permite generar valores aleatorios dentro de un cierto rango. *Floor()* permite redondear los valores generados a números enteros. Estas coordenadas aleatorias se juntan en una tabla *dataframe* y se exportan a Dinamica gracias a la función *outputTable()* y al functor de Dinamica *Extract Struct Table*.

```
# Recibe datos de DINAMICA
numptos <- v1
numlin <- v2
numcol <- v3

# Genera coordenadas aleatorias
y <- floor(runif(numptos, min=1, max=numlin))
x <- floor(runif(numptos, min=1, max=numcol))
tab <- cbind(seq(1:numptos), x, y)

# Pasa la tabla a Dinamica
outputTable("tabxy", tab)
```

La tabla de filas/columnas permite a Dinamica extraer los valores espectrales de los pixeles correspondientes en ambas bandas para crear dos tablas que entran en el segundo script de R (estas tablas tienen una columna "key" con un número consecutivo y una segunda columna con el valor espectral). R crea una tabla *dataframe* con los valores espectrales de las dos bandas. Con base en estos datos elabora un diagrama de dispersión y ajusta un modelo lineal. Los comandos *outputDouble("b", coefficients[1])* y *outputDouble("a", coefficients[2])* permiten exportar los dos coeficientes en un mismo objeto de Dinamica (los cuales se distinguen por el nombre "a" y "b" que se les asignó). El

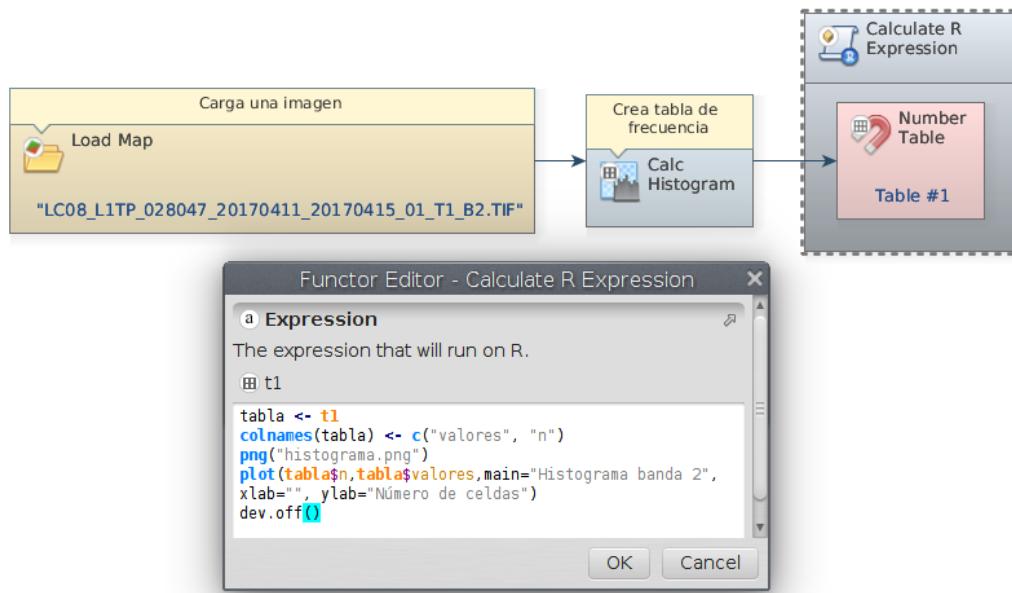


Figura 9. Modelo de Dinamica (primer ejemplo)

funcion de Dinamica ***Extract Struct Number*** permite recuperar estos dos valores utilizando los nombres. En un paso siguiente, Dinamica utiliza estos valores para calcular los residuales con operaciones de álgebra de mapas.

```

# Importa las dos tablas de Dinamica
tabla1 <- t1
tabla2 <- t2
colnames(tabla1) <- colnames(tabla2) <- c("key", "ND")
# Extrae la columna con los valores espectrales
x <- tabla1$ND
y <- tabla2$ND
# crea un data frame
tab <- data.frame(cbind(x,y))

# Elabora diagrama de dispersión
png("diag dispersion.png")
plot(x,y,main="Diagrama de dispersión banda 2 versus 1", xlab="banda 1",
ylab="banda 2")
dev.off()

# Ajusta modelo lineal y = a x + b
regression <- lm(formula=y~x, data=tab)
print(regression)
coefficients <- coef(regression)

# Exporta a Dinamica los coeficientes de la ecuación
outputDouble("b", coefficients[1])
outputDouble("a", coefficients[2])

```

Existen más paquetes para llevar a cabo operaciones de R en conjunto con paquetes de procesamiento de

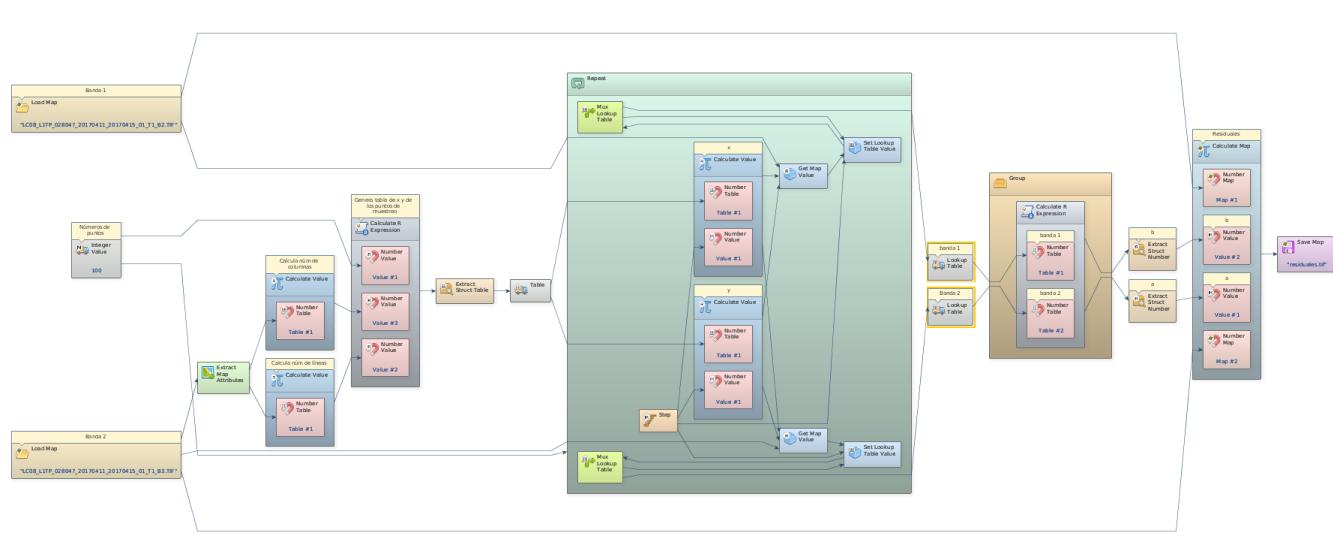


Figura 10. Modelo de Dinamica (Segundo ejemplo)

imágenes y SIG como spgrass6 (Bivand, 2016) y rgrass7 (Bivand *et al.*, 2017b, Bivand, 2007, programa grass), RSAGA (programa SAGA, Brenning, 2008) y RPyGeo (programa ArcGis, Brenning, 2012).

---

## Posfacio

---

La curva de aprendizaje de un lenguaje de programación como R, o de cualquier otro lenguaje, es lenta inicialmente, pero a mediano y largo plazo, el usuario empieza a entender la lógica y los patrones generales que gobiernan la sintaxis. Entonces, se abre la posibilidad de automatizar y combinar procesos y así lograr una versatilidad mucho mayor que la obtenida con el modelo estándar del software comercial. Espero que este libro haya permitido a los principiantes a superar esta difícil etapa inicial o a los usuarios de R provenientes de otras especilidades descubrir las aplicaciones de R en geografía y análisis espacial. En todos los casos, espero haberlos animados a utilizar este ambiente de computación!

Durante la última década, se incrementó de forma notoria la cantidad de datos espaciales disponibles al público. Muchas agencias espaciales están brindando datos de percepción remota (Landsat, MODIS, Sentinel, Aster...) de forma gratuita. Muchas instituciones, tanto internacionales como nacionales, están también abriendo el acceso al público de sus bases de datos. Por ejemplo, en México el Instituto Nacional de Estadísticas y Geografía (INEGI, <http://www.inegi.org.mx/>) y la Comisión Nacional para el Conocimiento y Uso de la Biodiversidad ([www.conabio.gob.mx/informacion/gis/](http://www.conabio.gob.mx/informacion/gis/)) o bien en España el Instituto Geográfico Nacional (IGN, <http://www.ign.es/web/ign/port>) e instituciones de las comunidades autónomas como el Instituto de Estadística y Cartografía, la Consejería de Medio Ambiente (Junta de Andalucía, <http://www.juntadeandalucia.es>) o en Cataluña el Institut Cartogràfic i Geològic de Catalunya (<http://www.icgc.cat/>). Además, recientemente, las nuevas tecnologías como los teléfonos celulares equipados de GPS están también produciendo enormes cantidades de datos geo-referenciados.

Son programas como R, gracias a la actualización y la creación constante de nuevos paquetes aportados por la comunidad global de usuarios y a su eficiencia para automatizar procesos complejos, que brindarán herramientas eficientes y transparentes para analizar esta enorme cantidad de datos.

Jean-François Mas  
Salvador, BA, Febrero de 2018



---

## Referencias

---

- Appelhans, T. & C. Reudenbach (2018). *link2GI: Linking Geographic Information Systems, Remote Sensing and Other Command Line Tools*. R package version 0.8-9.
- Bivand, R. (2007). “Using the R–GRASS Interface: Current Status”. En: *OSGeo Journal* 1: 36-38.
- (2016). *spgrass6: Interface between GRASS 6 and R*. R package version 0.8-9.
- Bivand, R. & C. Rundel (2017). *Rgeos: Interface to Geometry Engine - Open Source ('GEOS')*. <https://CRAN.R-project.org/package=rgeos>.
- Bivand, R., T. Keitt & B. Rowlingson (2017a). *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*. R package version 1.2-15.
- Bivand, R., R. Krug, M. Neteler & S. Jeworutzki (2017b). *rgrass7: Interface Between GRASS 7 Geographical Information System and R*. R package version 0.1-10.
- Bivand, R. S., E. J. Pebesma & V. Gómez-Rubio (2008). *Applied Spatial Data Analysis with R*. New York: Springer.
- Brenning, A. (2008). “Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models”. En: *SAGA – Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, vol. 19)*. J. Boehner, T. Blaschke, L. Montanarella. 23-32.
- (2012). *RPyGeo: ArcGIS Geoprocessing in R via Python*. R package version 0.9-3.
- Brunsdon, C. & L. Comber (2015). *An Introduction to R for Spatial Analysis and Mapping*. New York, USA: SAGE.
- Cabrero Ortega, M. & A. García Pérez (2014). *Análisis estadístico de datos espaciales con QGIS y R*. Madrid, España: UNED.
- Chávez, R. O., S. A. Estay & C. G. Riquelme (2017). *npphen: Vegetation Phenological Cycle and Anomaly Detection using Remote Sensing Data*. R package version 0.8-9.
- Collatón Chicana, R. (2014). *Introducción al uso de R y R Commander para el análisis estadístico de datos en ciencias sociales*. Disponible en <https://cran.r-project.org/doc/contrib/>.
- Dunnington, D. (2017). *prettymapr: Scale Bar, North Arrow, and Pretty Margins in R*. <https://CRAN.R-project.org/package=prettymapr>.
- Dunnington, D. & T. Giraud (2017). *rosm: Plot Raster Map Tiles from Open Street Map and Other Sources*. <https://CRAN.R-project.org/package=rosm>.
- Getis, A. & J. K. Ord (1992). “The Analysis of Spatial Association by Use of Distance Statistics”. En: *Geographical Analysis* 24: 189-206.
- Hijmans, R. J. (2017). *raster: Geographic Data Analysis and Modeling*. R package version 2.6-7.
- Klemmt, H.-J. (2015). *RTextureMetrics: Calculate Textures from Grey-Level Co-Occurrence Matrices (GLCMs)*. R package version 0.8-9.
- Lamstein, A. & B. P. Johnson (2018). *choroplethr: Simplify the Creation of Choropleth Maps in R*. <https://CRAN.R-project.org/package=choroplethr>.
- Lecigne, B., S. Delagrange & C. Messier (2015). *VoxR: Metrics extraction of trees from T-LiDAR data*. R package version 0.8-9.
- Leutner, B. & N. Horning (2017). *RStoolbox: Tools for Remote Sensing Data Analysis*. R package version 0.8-9.
- Loecher, M. (2016). *RgoogleMaps: Overlays on Static Maps*. <https://CRAN.R-project.org/package=RgoogleMaps>.
- Lovelace, R., J. Nowosad & J. Muenchow (2018). *Geocomputation with R*. <http://geocompr.robinlovelace.net/>. CRS Press.

- Mas, J.-F, R. Lemoine-Rodríguez, R. González, J. López-Sánchez, A. Piña-Garduño & E. Herrera-Flores (2017). “Evaluación de las tasas de deforestación en Michoacán a escala detallada mediante un método híbrido de clasificación de imágenes SPOT”. En: *Madera y Bosques* 23(2): 119-131.
- Mas, J., M. Kolb, M. Paegelow, M. C. Olmedo & T. Houet (2014). “Inductive pattern-based land use/cover change models: A comparison of four software packages”. En: *Environmental Modelling and Software* 51: 94 -111.
- Pebesma, E. (2017). *Map overlay and spatial aggregation in sp*. cran.r-project.org/web/packages/sp/vignettes/over.pdf. Institute for Geoinformatics, University of Muenster. Alemania.
- Pebesma, E. & R. Bivand (2018). *Sp: Classes and Methods for Spatial Data*. <https://CRAN.R-project.org/package=sp>.
- Rinaldi, M. F. (2015). *PROTOLIDAR: PRocess TOol LIidar DAta in R*. R package version 0.8-9.
- Rodrigues, H. & B. Soares-Filho (2018). “A Short Presentation of Dinamica EGO”. En: *Geomatic Approaches for Modeling Land Change Scenarios*. Ed. por M. T. Camacho Olmedo, M. Paegelow, J.-F. Mas & F. Escobar. Cham: Springer International Publishing. 493-498.
- Roussel, J.-R., D. Auty, F. D. Boissieu & A. S. Meador (2018a). *lidR: Airborne LiDAR Data Manipulation and Visualization for Forestry Applications*. R package version 0.8-9.
- Roussel, J.-R., M. Isenburg, D. Auty, P. Marie & F. D. Boissieu (2018b). *rlas: Read and Write 'las' and 'laz' Binary File Formats Used for Remote Sensing Data*. R package version 0.8-9.
- Santana, J. S. & E. M. Farfán (2014). *El arte de programar en R: un lenguaje para la estadística*. Disponible en <https://cran.r-project.org/doc/contrib/>. Instituto Mexicano de Tecnología del Agua.
- Sarparast, M. (2017). *LSRS: Land Surface Remote Sensing*. R package version 0.8-9.
- Silva, C. A., N. L. Crookston, A. T. Hudak, L. A. Vierling & C. Klauberg (2017). *rLIDAR: LiDAR Data Processing and Visualization*. R package version 0.8-9.
- Tennekes, M., J. Gombin, S. Jeworutzki, K. Russell & R. Zijdeman (2017). *tmap: Thematic Maps*. R package version 1.10.
- Xie, Y. (2013). *Dynamic Documents with R and Knitr*. Chapman & Hall/CRC.
- Zvoleff, A. (2016). *glcm: Calculate Textures from Grey-Level Co-Occurrence Matrices (GLCMs)*. R package version 0.8-9.

# ANEXOS



Initial letter R with garlands (mid-16th century).  
<https://metmuseum.org/art/collection/search/701236>  
Public Domain



---

# 1. Organización de los objetos espaciales en sp

---

Vamos a examinar los objetos espaciales de los paquetes **sp** desarrollado por Edzer Pebesma y **raster** desarrollado por Robert J. Hijmans. Para ello, necesitarán instalar ambos paquetes (ver sección ??). En el desarrollo de R, los objetos espaciales son de reciente creación y pertenecen por lo tanto a la última generación (clase S4). Presentan una definición formal que especifica el nombre y el tipo de componentes (llamados *slots*).

En **sp**, todos los objetos espaciales tienen por lo menos dos *slots*:

- El marco (*bounding box*) que es una matriz con las coordenadas extremas.
- El CRS que define el sistema de las coordenadas de referencia en el formato PROJ.4. Para conocer el marco de un objeto se puede usar la función **bbox()**, para conocer su sistema de proyección **proj4string()**.

Existen subclases de objetos espaciales dependiendo del tipo de información. En formato vector, las subclases **SpatialPoints**, **SpatialLines** y **SpatialPolygons** permiten manejar respectivamente geometrías de puntos, líneas y polígonos. **SpatialPointsDataFrame**, **SpatialLinesDataFrame** y **SpatialPolygonsDataFrame** permiten además de asociar a los rasgos de las coberturas una tabla de atributos. De la misma manera, en formato raster existen las subclases **SpatialPixels**, **SpatialGrid**, **SpatialPixelsDataFrame** y **SpatialGridDataFrame**.

La estructura de estos objetos es anidada y puede parecer un poco complicada al principio. En las siguientes secciones, vamos a construir objetos espaciales muy simples de estas diferentes clases para entender mejor la forma en la cual están estructurados. Vamos a representar estos objetos en un sistema de coordenadas arbitrario con valores, tanto en x como en y, entre cero y diez, eso con el fin de manejar datos sencillos. Empezamos por activar la librería **sp** con **library(sp)**.

## 1.1 Datos vectoriales

### 1.1.1 Cobertura de puntos

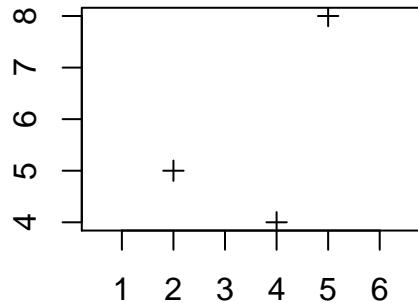
Vamos a crear una tabla con las coordenadas de tres puntos (puede ser una matriz o un dataframe). Esta tabla se transforma en una cobertura de puntos de la clase **SpatialPoints** con la función **SpatialPoints()**.

```
library(sp)
# SpatialPoints
# Crea un vector de coordenadas en x
Xs <- c(2,4,5)
# Crea un vector de coordenadas en y
Ys <- c(5,4,8)
# Pega Xs y Ys para crear una tabla de coordenadas
coords <- cbind(Xs,Ys)
print(coords)

##      Xs Ys
## [1,]  2  5
## [2,]  4  4
## [3,]  5  8
```

```
# Crea el objeto SpatialPoints (SP)
SP = SpatialPoints(coords)
```

```
plot(SP, axes = TRUE)
```



```
class(SP)

## [1] "SpatialPoints"
## attr(,"package")
## [1] "sp"
```

Diferentes funciones nos permiten conocer las características de este objeto. La función **summary()** y **print()** (o simplemente el nombre del objeto) dan las características generales del objeto. En particular, **proj4string()**, **bbox()** y **coordinates()** nos dan respectivamente el sistema de proyección (en este caso no tiene), la extensión y las coordenadas de los elementos del objeto. Note que existen dos sintaxis equivalentes, una con paréntesis y la otra usando el símbolo aroba (por ejemplo **bbox(SP)** y **SP@bbox**).

```
summary(SP)

## Object of class SpatialPoints
## Coordinates:
##   min max
## Xs   2   5
## Ys   4   8
## Is projected: NA
## proj4string : [NA]
## Number of points: 3

print(SP) # o simplemente SP

## class      : SpatialPoints
## features   : 3
## extent     : 2, 5, 4, 8  (xmin, xmax, ymin, ymax)
## coord. ref. : NA

proj4string(SP)

## [1] NA

SP@proj4string
```

```

## CRS arguments: NA

bbox(SP)

##      min max
## Xs    2    5
## Ys    4    8

SP@bbox

##      min max
## Xs    2    5
## Ys    4    8

coordinates(SP)

##      Xs Ys
## [1,] 2  5
## [2,] 4  4
## [3,] 5  8

SP@coords

##      Xs Ys
## [1,] 2  5
## [2,] 4  4
## [3,] 5  8

```

El objeto `SpatialPoints` que acabamos de crear tiene solo la información de las coordenadas de los tres puntos, no hay ninguna información adicional sobre los puntos. Vamos ahora a crear una tabla de atributos con información sobre cada uno de los puntos. La asociación de esta tabla con el objeto anterior nos permite crear un objeto más "sofisticado" de la clase `SpatialPointDataFrame`.

En un primer paso vamos a crear una tabla dataframe con dos columnas, la primera representa el número del punto y la segunda su descripción. El número del punto es el identificador de cada punto que podemos observar con `coordinates(SP)`, corresponde al orden original de las coordenadas cuando creamos el objetos `SpatialPoints`.

En un paso siguiente, relacionamos la tabla con el objeto espacial con `SpatialPointsDataFrame()`. En realidad existen varias opciones para realizar esta operación, con la opción `match.ID="num"` estamos utilizando el campo `num` de la tabla para relacionar cada fila con un punto.

Para crear el objeto SPDF2, se usan directamente las tablas de coordenadas y de atributos, sin pasar por el objeto `SpatialPoints`. En este caso, es muy importante que el orden de ambas tablas sea el mismo, es decir que una fila en la tabla de atributos describa el punto de las coordenadas ubicadas en la misma fila (posición) en la otra tabla.

```

# SpatialPoints data frame SPDF #####
# Tabla con ID (campo num) e información adicional (tabla de atributos)
num <- c(1, 2, 3)
nombre <- c("Pozo", "Gasolinera", "Pozo")

```

```

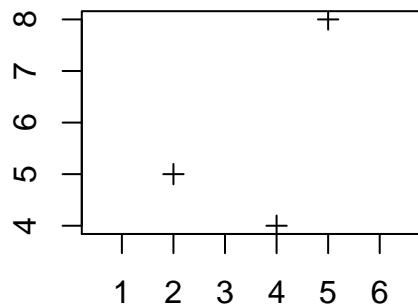
tabpuntos <- data.frame(cbind(num,nombre))
class(tabpuntos)

## [1] "data.frame"

# crea el SpatialPointsDataFrame con el SpatialPoints y la tabla
SPDF <- SpatialPointsDataFrame(SP, tabpuntos, match.ID="num")
# crea el SpatialPointsDataFrame con las coord y la tabla
SPDF2 <- SpatialPointsDataFrame(coords, tabpuntos)

plot(SPDF, axes=TRUE)

```



La estructura anidada de un objeto de la clase `SpatialPointsDataFrame` puede resumirse con la figura 1.

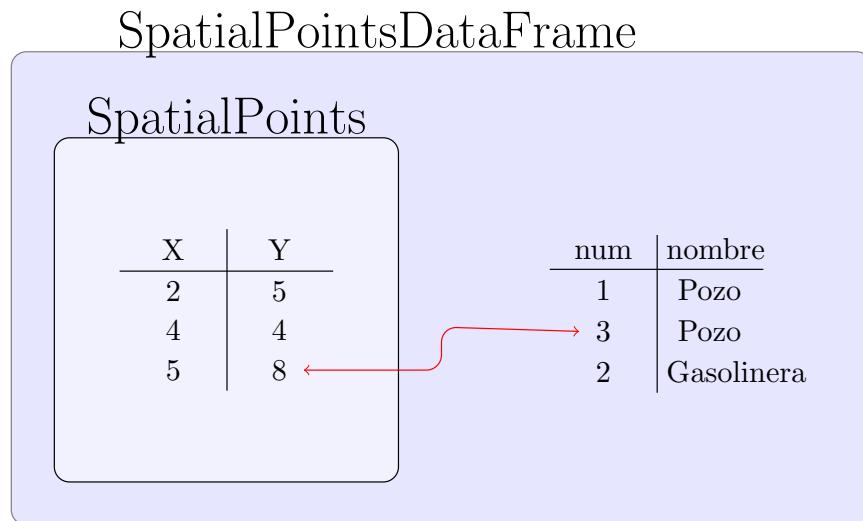


Figura 1. Estructura de un objeto `SpatialPointsDataFrame`

La tabla de atributos puede visualizarse fácilmente y permite seleccionar ciertos elementos de la cobertura. Por ejemplo, creamos un nuevo objeto `SpatialPointsDataFrame`, llamado Pozos, con los puntos cuyo nombre es "Pozo" en la tabla de atributo.

```

# Se puede extraer la tabla de atributos de un SPDF con
as.data.frame(SPDF)

##   num     nombre Xs Ys

```

```

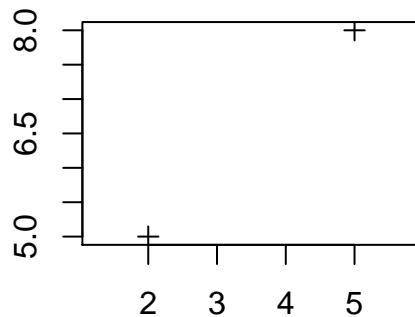
## 1   1      Pozo  2  5
## 2   2 Gasolinera  4  4
## 3   3      Pozo  5  8

SPDF@data

##     num      nombre
## 1   1      Pozo
## 2   2 Gasolinera
## 3   3      Pozo

# Selección de elementos dentro de la cobertura
Pozos <- SPDF [SPDF@data$nombre=="Pozo",]
plot(Pozos, axes=TRUE)

```



### 1.1.2 Cobertura de líneas

En formato vector, una línea está definida por las coordenadas de los vértices. Para el manejo de este tipo de datos, se manejan objetos de la clase `Line` que describen segmentos simples. Los objetos `Lines` agrupan segmentos `Line` con el mismo identificador (ID), y finalmente el objeto `SpatialLines` permite juntar diferentes objetos `Lines`. En el código a continuación, creamos tres objetos `Line` (`L1`, `L2` y `L3`) con base en una tabla de coordenadas usando la función `Line()`.

```

# Crea 3 objetos "Line": simple cadena de coordenadas (vértices)
# Linea 1
X1s <- c(0,3,5,8,10)
Y1s <- c(0,3,4,8,10)
Coord1 <- cbind(X1s,Y1s)
# Crea objeto de Clase Line
L1 <- Line(Coord1)

# Linea 2
X2s <- c(2,1,1)
Y2s <- c(2,4,5)
Coord2 <- cbind(X2s,Y2s)
# Crea objeto de Clase Line
L2 <- Line(Coord2)

# Linea 3

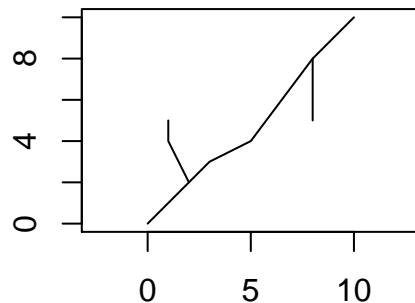
```

```
X3s <- c(8,8)
Y3s <- c(8,5)
Coord3 <- cbind(X3s,Y3s)
# Crea objeto de Clase Line
L3 <- Line(Coord3)
```

Luego, creamos un objeto `Lines` con base en los segmentos L1 y L2, asignándole el identificador "p". `Lines2` está compuesto únicamente del segmento L3, usando el ID "t". En un paso final, juntamos ambos objetos `Lines` en un objeto `SpatialLines`.

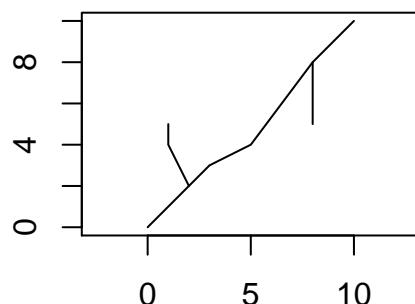
```
# Crea dos objetos Lines: conjunto de objetos Line con un mismo ID
Lines1 = Lines(list(L1,L2), ID="p")
Lines2 = Lines(L3, ID="t")

# Crea un objeto SpatialLines: conjunto de objetos Lines
SL <- SpatialLines(list(Lines1, Lines2))
plot(SL, axes=TRUE)
```



Para crear un objeto `SpatialLinesDataFrame`, se asocia una tabla de atributos a las líneas de una forma similar a lo que hicimos para las coberturas de puntos.

```
# Tabla con ID (campo num) e informacion adicional (tabla de atributo)
code <- c("t","p")
tipo <- c("Terraceria","Pavimentada")
tablineas <- data.frame(cbind(tipo,code))
SLDF = SpatialLinesDataFrame(SL,tablineas,match.ID="code")
plot(SLDF, axes=TRUE)
```



La estructura anidada de un objeto de la clase `SpatialLinesDataFrame` puede resumirse con la figura 2.

Se puede consultar los *slots* de estos objetos con las funciones `bbox()`, `coordinates()`, `proj4string()`. La tabla de atributos puede usarse para seleccionar líneas con ciertas características. Por ejemplo, en el ejemplo a continuación se

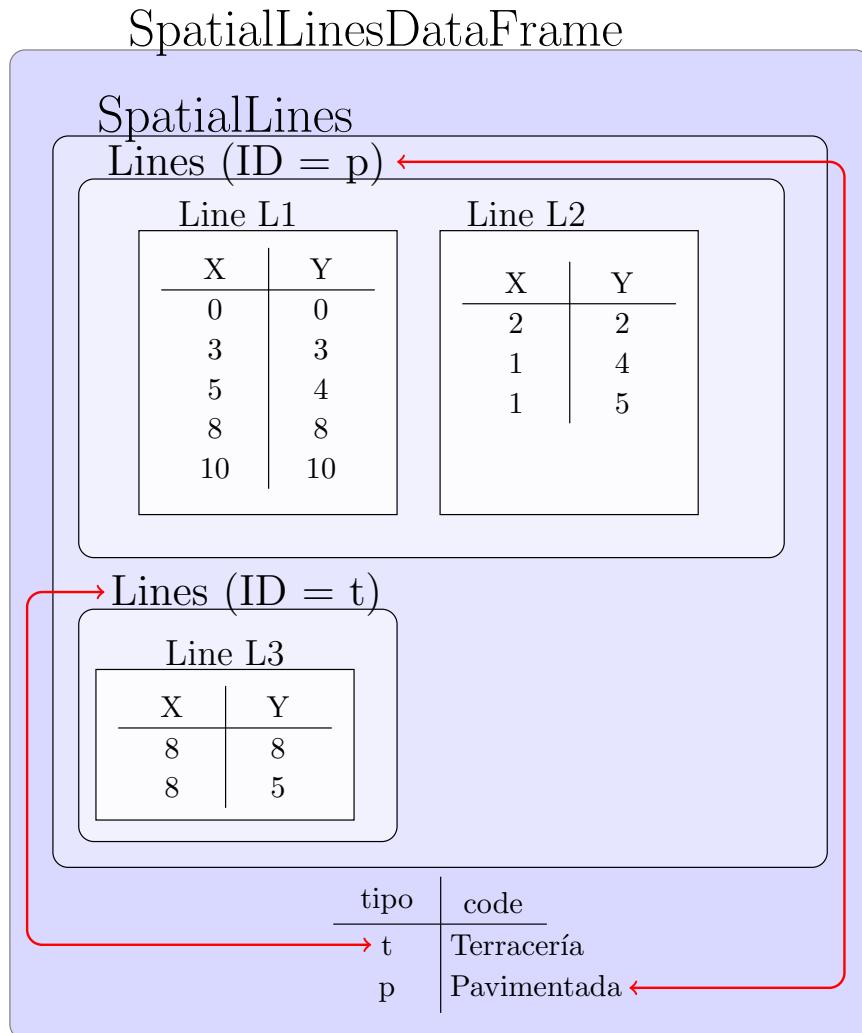


Figura 2. Estructura de un objeto `SpatialLinesDataFrame`

seleccionan y plotean primero las carreteras pavimentadas, y en un segundo tiempo, las de terracería usando el color rojo.

```

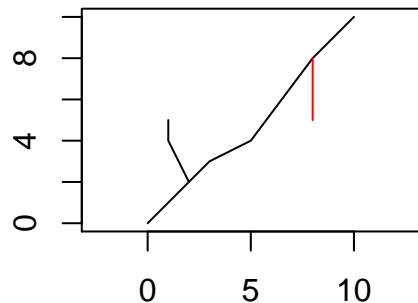
# Se puede extraer la tabla de atributos de un SPDF con
as.data.frame(SLDF)

##           tipo code
## p Pavimentada   p
## t Terraceria   t

SLDF@data

##           tipo code
## p Pavimentada   p
## t Terraceria   t
  
```

```
# Se puede seleccionar ciertos rasgos usando la tabla de atributos
plot(SLDF[SLDF@data$tipo=="Pavimentada",], axes=TRUE)
plot(SLDF[SLDF@data$tipo=="Terraceria",], add=TRUE, col="red")
```



En la figura 2, podemos observar la estructura anidada de los datos con SLDF@lines. El objeto SLDF está compuesto de dos objetos Lines, el primero constituido de dos objetos Line (ID "p") y el segundo solo de uno. Podemos por lo tanto extraer información específica de alguno de los elementos.

Por ejemplo SLDF@lines[[1]]@Lines[[2]]@coords nos permite obtener las coordenadas del segundo objeto Line del primer objeto Lines.

```
# Organización de los datos
SLDF@lines

## [[1]]
## An object of class "Lines"
## Slot "Lines":
## [[1]]
## An object of class "Line"
## Slot "coords":
##     X1s Y1s
## [1,] 0 0
## [2,] 3 3
## [3,] 5 4
## [4,] 8 8
## [5,] 10 10
##
## 
## [[2]]
## An object of class "Line"
## Slot "coords":
##     X2s Y2s
## [1,] 2 2
## [2,] 1 4
## [3,] 1 5
##
## 
## 
## Slot "ID":
## [1] "p"
##
```

```
## 
## [[2]]
## An object of class "Lines"
## Slot "Lines":
## [[1]]
## An object of class "Line"
## Slot "coords":
##      X3s Y3s
## [1,] 8   8
## [2,] 8   5
##
##
##
## Slot "ID":
## [1] "t"
```

### 1.1.3 Cobertura de polígonos

La organización de los polígonos es un poco más compleja. La estructura anidada de un objeto de la clase `SpatialPolygonsDataFrame` puede resumirse con la figura 3.

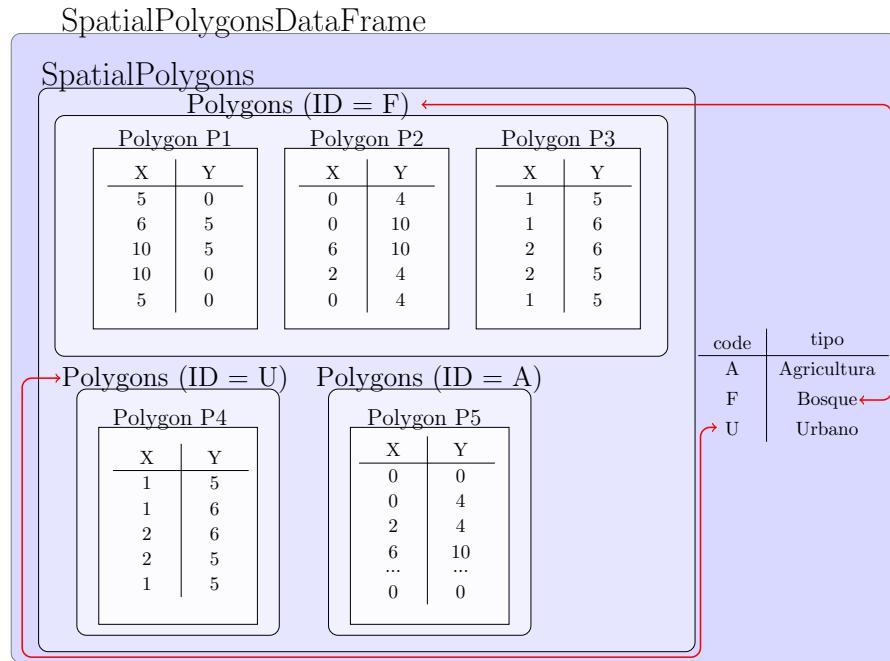


Figura 3. Estructura de un objeto de la clase `SpatialPolygonsDataFrame`

Para crear una cobertura de polígonos, se crean objetos `Polygon` a partir de las coordenadas de vértices que crean una forma cerrada utilizando la función `Polygon()`. La opción `hole = TRUE` permite crear huecos dentro de otro polígono. En un paso siguiente, se crean objetos `Polygons`, conjunto de objetos `Polygon` que comparten el mismo identificador (misma categoría). Por ejemplo `Polif = Polygons(list(P1,P2,P3), ID="F")` crea un objeto `Polygons` con base en los objetos `Polygon P1, P2 y P3` y les asigna el identificador (ID) "F". Finalmente un objeto `Spatial Polygons` es un conjunto de objeto `Polygons` creado utilizando la función `SpatialPolygons()`.

```
## Polígono forestal al SudEste
## Polygon
# Crea una cadena de coordenadas en X
X1 <- c(5,6,10,10,5)
# Crea una cadena de coordenadas en Y
# Djo tiene que cerrar (último par de coord = primero)
Y1 <- c(0,5,5,0,0)
# Pega X y Y para crear una tabla de coordenadas
c1 <- cbind(X1,Y1)
print(c1)

##          X1   Y1
## [1,]    5   0
## [2,]    6   5
## [3,]   10   5
## [4,]   10   0
## [5,]    5   0

class(c1)

## [1] "matrix"

# Crea el objeto Polygon. Un polygon es un forma simple cerrada
P1 = Polygon(c1)

# Polígono forestal al NorOeste
# Crea una cadena de coordenadas en X
X2 <- c(0,0,6,2,0)
# Crea una cadena de coordenadas en Y
Y2 <- c(4,10,10,4,4)
# Pega X y Y para crear una tabla de coordenadas
c2 <- cbind(X2,Y2)
P2 = Polygon(c2)

# Polígono hueco
# Crea una cadena de coordenadas en X
X3 <- c(1,1,2,2,1)
# Crea una cadena de coordenadas en Y
Y3 <- c(5,6,6,5,5)
# Pega X y Y para crear una tabla de coordenadas
c3 <- cbind(X3,Y3)
P3 = Polygon(c3, hole=TRUE)

# Polígono de agricultura
P4 = Polygon(c3) # esta vez no es hueco

# Polígono de área urbana
X5 <- c(0,0,2,6,10,10,6,5,0)
```

```

Y5 <- c(0,4,4,10,10, 5,5,0,0)
c5 <- cbind(X5,Y5)
P5 = Polygon(c5)

# Un Polygons es un conjunto de Polygon, incluyendo eventualmente huecos, con un ID

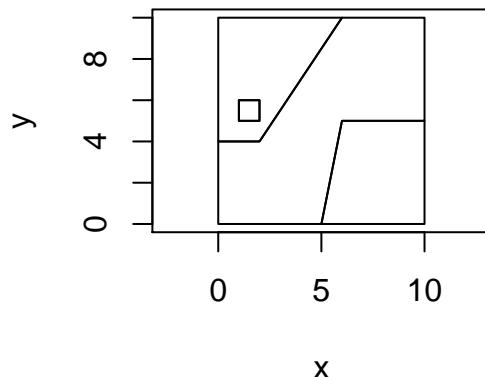
# Polígonos forestales: los dos polígonos con un hueco
Polif = Polygons(list(P1,P2,P3),ID="F")

# Polígono zona urbana
Poliu = Polygons(list(P4),ID="U")
# Polígono agricultura
Polia = Polygons(list(P5),ID="A")

# Spatial Polygons: conjunto de polygons
SPol = SpatialPolygons(list(PoliF,Polia,Poliu))

plot(SPol,axes=TRUE,xlab="x",ylab="y")

```



Se asocia esta cobertura de polígonos a una tabla de atributos para crear un objeto `SpatialPolygonsDataFrame` de forma similar a aquella utilizada para las coberturas de puntos y líneas. El comando `summary()` nos permite obtener una información básica sobre esta cobertura. Toda la información del objeto se obtiene con `print()` o simplemente con el nombre del objeto. Se puede acceder a información más específica en esta estructura anidada utilizando el símbolo `@`.

```

# Spatial PolygonsDataFrame

# Tabla con ID (campo num) e información adicional (tabla de atributo)
code <- c("A","F","U")
tipo <- c("Agricultura","Bosque","Urbano")
tabpol <- data.frame(cbind(code,tipo))

SPolDF = SpatialPolygonsDataFrame(SPol,tabpol,match.ID="code")

summary(SPolDF)

## Object of class SpatialPolygonsDataFrame
## Coordinates:

```

```
##   min max
## x   0   10
## y   0   10
## Is projected: NA
## proj4string : [NA]
## Data attributes:
##   code          tipo
##   A:1  Agricultura:1
##   F:1  Bosque      :1
##   U:1  Urbano     :1

SPoDF@polygons

## [[1]]
## An object of class "Polygons"
## Slot "Polygons":
## [[1]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 7.740741 2.407407
##
## Slot "area":
## [1] 22.5
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##   X1 Y1
## [1,] 5 0
## [2,] 6 5
## [3,] 10 5
## [4,] 10 0
## [5,] 5 0
##
##
##
## [[2]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 2.166667 7.500000
##
## Slot "area":
## [1] 24
##
## Slot "hole":
```

```
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##      X2 Y2
## [1,]  0  4
## [2,]  0 10
## [3,]  6 10
## [4,]  2  4
## [5,]  0  4
##
## 
## [[3]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 1.5 5.5
##
## Slot "area":
## [1] 1
##
## Slot "hole":
## [1] TRUE
##
## Slot "ringDir":
## [1] -1
##
## Slot "coords":
##      X3 Y3
## [1,]  1  5
## [2,]  2  5
## [3,]  2  6
## [4,]  1  6
## [5,]  1  5
##
## 
## 
## Slot "plotOrder":
## [1] 2 1 3
##
## Slot "labpt":
## [1] 2.166667 7.500000
##
## Slot "ID":
## [1] "F"
##
```

```
## Slot "area":  
## [1] 46.5  
##  
##  
## [[2]]  
## An object of class "Polygons"  
## Slot "Polygons":  
## [[1]]  
## An object of class "Polygon"  
## Slot "labpt":  
## [1] 5.118380 4.968847  
##  
## Slot "area":  
## [1] 53.5  
##  
## Slot "hole":  
## [1] FALSE  
##  
## Slot "ringDir":  
## [1] 1  
##  
## Slot "coords":  
##      X5 Y5  
## [1,]  0  0  
## [2,]  0  4  
## [3,]  2  4  
## [4,]  6 10  
## [5,] 10 10  
## [6,] 10  5  
## [7,]  6  5  
## [8,]  5  0  
## [9,]  0  0  
##  
##  
##  
## Slot "plotOrder":  
## [1] 1  
##  
## Slot "labpt":  
## [1] 5.118380 4.968847  
##  
## Slot "ID":  
## [1] "A"  
##  
## Slot "area":  
## [1] 53.5  
##
```

```
##  
## [[3]]  
## An object of class "Polygons"  
## Slot "Polygons":  
## [[1]]  
## An object of class "Polygon"  
## Slot "labpt":  
## [1] 1.5 5.5  
##  
## Slot "area":  
## [1] 1  
##  
## Slot "hole":  
## [1] FALSE  
##  
## Slot "ringDir":  
## [1] 1  
##  
## Slot "coords":  
##      X3 Y3  
## [1,]  1  5  
## [2,]  1  6  
## [3,]  2  6  
## [4,]  2  5  
## [5,]  1  5  
##  
##  
##  
## Slot "plotOrder":  
## [1] 1  
##  
## Slot "labpt":  
## [1] 1.5 5.5  
##  
## Slot "ID":  
## [1] "U"  
##  
## Slot "area":  
## [1] 1  
  
SPolDF@polygons[[1]]@Polygons[[1]]@hole  
## [1] FALSE  
  
SPolDF@polygons[[1]]@Polygons[[1]]@coords  
##      X1 Y1  
## [1,]  5  0
```

```
## [2,] 6 5
## [3,] 10 5
## [4,] 10 0
## [5,] 5 0

SPolDF@polygons[[1]]@Polygons[[1]]

## An object of class "Polygon"
## Slot "labpt":
## [1] 7.740741 2.407407
##
## Slot "area":
## [1] 22.5
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##      X1 Y1
## [1,] 5 0
## [2,] 6 5
## [3,] 10 5
## [4,] 10 0
## [5,] 5 0

slot(SPol,"polygons") # lista de objetos Polygons

## [[1]]
## An object of class "Polygons"
## Slot "Polygons":
## [[1]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 7.740741 2.407407
##
## Slot "area":
## [1] 22.5
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
```

```
##      X1 Y1
## [1,]  5  0
## [2,]  6  5
## [3,] 10  5
## [4,] 10  0
## [5,]  5  0
##
##
## [[2]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 2.166667 7.500000
##
## Slot "area":
## [1] 24
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##      X2 Y2
## [1,]  0  4
## [2,]  0 10
## [3,]  6 10
## [4,]  2  4
## [5,]  0  4
##
##
## [[3]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 1.5 5.5
##
## Slot "area":
## [1] 1
##
## Slot "hole":
## [1] TRUE
##
## Slot "ringDir":
## [1] -1
##
## Slot "coords":
##      X3 Y3
```

```
## [1,] 1 5
## [2,] 2 5
## [3,] 2 6
## [4,] 1 6
## [5,] 1 5
##
##
##
## Slot "plotOrder":
## [1] 2 1 3
##
## Slot "labpt":
## [1] 2.166667 7.500000
##
## Slot "ID":
## [1] "F"
##
## Slot "area":
## [1] 46.5
##
##
## [[2]]
## An object of class "Polygons"
## Slot "Polygons":
## [[1]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 5.118380 4.968847
##
## Slot "area":
## [1] 53.5
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##      X5 Y5
## [1,] 0 0
## [2,] 0 4
## [3,] 2 4
## [4,] 6 10
## [5,] 10 10
## [6,] 10 5
## [7,] 6 5
```

```
## [8,] 5 0
## [9,] 0 0
##
##
##
## Slot "plotOrder":
## [1] 1
##
## Slot "labpt":
## [1] 5.118380 4.968847
##
## Slot "ID":
## [1] "A"
##
## Slot "area":
## [1] 53.5
##
##
## [[3]]
## An object of class "Polygons"
## Slot "Polygons":
## [[1]]
## An object of class "Polygon"
## Slot "labpt":
## [1] 1.5 5.5
##
## Slot "area":
## [1] 1
##
## Slot "hole":
## [1] FALSE
##
## Slot "ringDir":
## [1] 1
##
## Slot "coords":
##      X3 Y3
## [1,] 1 5
## [2,] 1 6
## [3,] 2 6
## [4,] 2 5
## [5,] 1 5
##
##
##
## Slot "plotOrder":
## [1] 1
```

```
## 
## Slot "labpt":
## [1] 1.5 5.5
##
## Slot "ID":
## [1] "U"
##
## Slot "area":
## [1] 1
```

## 1.2 Datos raster

En el paquete **sp** existen dos formas de representar datos raster: **SpatialPixels** y **SpatialGrid**. **SpatialPixels** se parece a **SpatialPoints**, pero las coordenadas deben ser distribuidas regularmente en una malla. Las coordenadas son asociadas a los índices del grid. Los objetos **SpatialPixelsDataFrame** tienen atributos solo para las celdas (puntos) existentes, pero necesitan almacenar las coordenadas y los índices de estas celdas.

Los objetos **SpatialGridDataFrame** no necesitan almacenar las coordenadas de cada celda, porque contemplan la malla entera, pero necesitan almacenar los NA (información no disponible) cuando los atributos no existen.

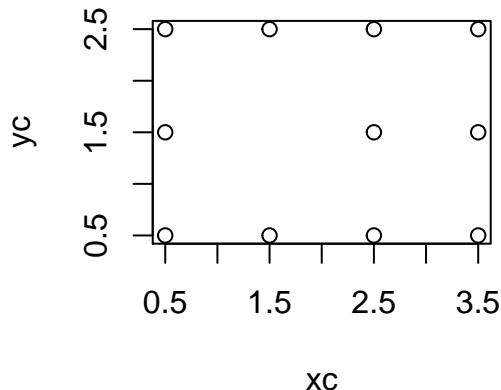
### 1.2.1 SpatialPixels y SpatialPixelsDataFrame

Para elaborar un objeto **SpatialPixels**, vamos a empezar por construir un objeto **SpatialPoints** con base en las coordenadas de las celdas con datos. La función **gridded()** nos permite transformar este objeto **SpatialPoints** en **SpatialPixels**.

```
## SpatialPixels
## Malla de 3 x 4 celdas de dimensión 1x1
## Coordenadas del centro de las celdas (no hay (1.5, 1.5) que es NA)
xc = c(0, 1, 2, 3, 0, 2, 3, 0, 1, 2, 3) + 0.5
yc = c(0, 0, 0, 0, 1, 1, 1, 2, 2, 2) + 0.5
df <- data.frame(xc, yc)
print(df)

##      xc    yc
## 1  0.5  0.5
## 2  1.5  0.5
## 3  2.5  0.5
## 4  3.5  0.5
## 5  0.5  1.5
## 6  2.5  1.5
## 7  3.5  1.5
## 8  0.5  2.5
## 9  1.5  2.5
## 10 2.5  2.5
## 11 3.5  2.5

plot(xc, yc, axes = T)
```



```

coordinates(df) <- ~xc+yc
class(df) # es un SpatialPoints!

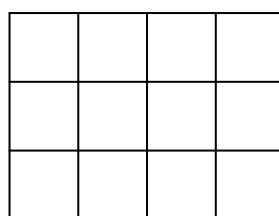
## [1] "SpatialPoints"
## attr(,"package")
## [1] "sp"

gridded(df) = TRUE # lo pasamos a raster SpatialPixels
class(df) # ya es SpatialPixels

## [1] "SpatialPixels"
## attr(,"package")
## [1] "sp"

plot(df)

```



Los objetos `SpatialPixels` tienen diferentes *slots*: `grid` indica las coordenadas de la celda de la esquina inferior izquierda, el tamaño de una celda y las dimensiones del raster en número de celdas en fila y columna, `grid.index` muestra el orden de los puntos del objeto `SpatialPoints`, `df@coords` indica las coodenadas de los centros de las celdas, `bbox` el marco (*bounding box*) y `proj4string` el sistema de coordenadas.

```

# Grid (topología del grid: offset, tamaño de celdas, num filas/columnas)
df@grid

##           xc  yc
## cellcentre.offset 0.5 0.5
## cellsize          1.0 1.0
## cells.dim         4.0 3.0

# Grid index: indexación de los puntos
df@grid.index

```

```

## [1] 9 10 11 12 5 7 8 1 2 3 4

# Coordenadas de los puntos (centro de las celdas)
df@coords

##      xc  yc
## 1  0.5 0.5
## 2  1.5 0.5
## 3  2.5 0.5
## 4  3.5 0.5
## 5  0.5 1.5
## 6  2.5 1.5
## 7  3.5 1.5
## 8  0.5 2.5
## 9  1.5 2.5
## 10 2.5 2.5
## 11 3.5 2.5

# Ventana
df@bbox

##      min  max
## xc    0    4
## yc    0    3

# Un sistema de proyección (inexistente en este caso)
df@proj4string

## CRS arguments: NA

```

Podemos asociar este objeto `SpatialPixels` a una tabla de atributos que contiene los valores de cada celda, obteniendo un objeto `SpatialPixelsDataFrame`.

```

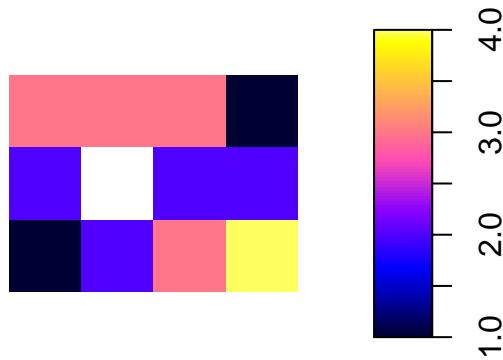
## SpatialPixelsDataFrame
# Data frame tabla de atributos (sin el dato NA)
tab_g <- data.frame(c(1,2,3,4,2,2,2,3,3,3,1))

# SpatialPixelsDataFrame
SPixDF <- SpatialPixelsDataFrame(df,tab_g)
class(SPixDF) # es un "SpatialPixelsDataFrame"

## [1] "SpatialPixelsDataFrame"
## attr(,"package")
## [1] "sp"

plot(SPixDF)

```



```
## Hay algunos slots supplementarios en comparación con el SpatialPixel
# Valores de los pixels
SPixDF@data

##   c.1..2..3..4..2..2..2..3..3..3..1.
## 1                      1
## 2                      2
## 3                      3
## 4                      4
## 5                      2
## 6                      2
## 7                      2
## 8                      3
## 9                      3
## 10                     3
## 11                     1

SPixDF@coords.nrs # Núm de la columna de la tabla de atributos

## numeric(0)

# que contiene las coordenadas (aqui no aplica)
```

## 1.2.2 SpatialGrid y SpatialGridDataFrame

Para construir un objeto `SpatialGrid`, empezamos por definir la estructura de la malla utilizando la función `GridTopology()` cuyos argumentos son las coordenadas mínimas (centro del pixel de la esquina inferior izquierda), el tamaño de la celdas y el número de celdas en fila y columna. Este objeto `GridTopology` se transforma en objeto `SpatialGrid` utilizando la función `SpatialGrid()`.

```
G = GridTopology(c(0.5,0.5), c(1,1), c(4,3))
class(G)

## [1] "GridTopology"
## attr(,"package")
## [1] "sp"

# Estructura de la malla con 3 slots
G@cellcentre.offset
```

```
## [1] 0.5 0.5

G@cellsize

## [1] 1 1

G@cells.dim

## [1] 4 3

summary(G)

## Grid topology:
##   cellcentre.offset cellsize cells.dim
## 1             0.5      1        4
## 2             0.5      1        3

coordinates(G)

##           s1   s2
## [1,] 0.5 2.5
## [2,] 1.5 2.5
## [3,] 2.5 2.5
## [4,] 3.5 2.5
## [5,] 0.5 1.5
## [6,] 1.5 1.5
## [7,] 2.5 1.5
## [8,] 3.5 1.5
## [9,] 0.5 0.5
## [10,] 1.5 0.5
## [11,] 2.5 0.5
## [12,] 3.5 0.5

coordinatevalues(G)

## $s1
## [1] 0.5 1.5 2.5 3.5
##
## $s2
## [1] 2.5 1.5 0.5

# SpatialGrid
# Casi lo mismo: tiene slots adicionales (proj, bbox)
SG = SpatialGrid(G)
class(SG)

## [1] "SpatialGrid"
## attr(,"package")
## [1] "sp"
```

```
summary(SG)

## Object of class SpatialGrid
## Coordinates:
##      min max
## [1,] 0   4
## [2,] 0   3
## Is projected: NA
## proj4string : [NA]
## Grid attributes:
##   cellcentre.offset cellsize cells.dim
## 1             0.5       1       4
## 2             0.5       1       3
```

Los objetos `SpatialGrid` representan los *slots* grid que indican las coordenadas de la celda de la esquina inferior izquierda, el tamaño de una celda y las dimensiones del raster en número de celda en fila y columna, bbox la extensión espacial (*bounding box*) y proj4string el sistema de coordenadas.

```
# slots adicionales
SG@grid

##           X1 X2
## cellcentre.offset 0.5 0.5
## cellsize          1.0 1.0
## cells.dim         4.0 3.0

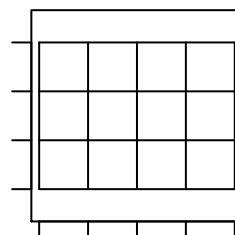
SG@bbox

##      min max
## [1,] 0   4
## [2,] 0   3

SG@proj4string

## CRS arguments: NA

plot(SG, axes=TRUE)
```



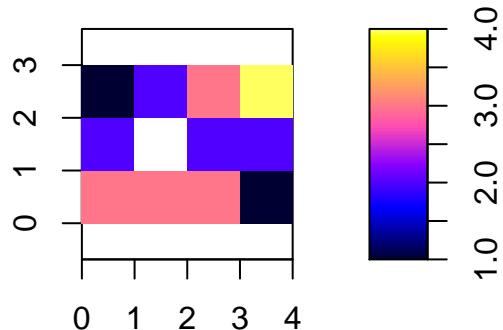
Se asocian las celdas a atributos contenidos en una tabla para elaborar un objeto `SpatialGridDataFrame`. Estos valores de atributo están en el slot data.

```
## SpatialGridDataFrame
# Un dataframe con los valores de las 12 celdas
tab_g <- data.frame(c(1,2,3,4,2,NA,2,2,3,3,3,1))

SGDF = SpatialGridDataFrame(SG, tab_g)
class(SGDF)

## [1] "SpatialGridDataFrame"
## attr(,"package")
## [1] "sp"

plot(SGDF, axes=T)
```



```
SGDF@data
```

```
##      c.1..2..3..4..2..NA..2..2..3..3..3..1.
## 1                  1
## 2                  2
## 3                  3
## 4                  4
## 5                  2
## 6                 NA
## 7                  2
## 8                  2
## 9                  3
## 10                 3
## 11                 3
## 12                 1
```

---

## 2. Importación/exportación de datos espaciales

---

### 2.1 Importación de archivos **shape** en **sp**

Presentamos aquí el procedimiento de importación con los paquetes **rgdal** y **maptools**. En **sp**, las geometrías de puntos, líneas y polígonos asociadas a una tabla de atributos son representadas en R por los objetos **SpatialPointsDataFrame**, **SpatialLinesDataFrame** y **SpatialPolygonsDataFrame** (ver script anexoB.R).

#### 2.1.1 Importación con **maptools**

La función **readShapePoly()** permite importar coberturas de polígonos. Los comandos **readShapePoints()** y **readShapeLines()** permiten importar coberturas de puntos y de líneas respectivamente. Note que en esta importación, se pierde la información sobre el sistema de coordenadas, mismo que se tiene que definir de nuevo con la función **proj4string()**. Sin embargo, **maptools** tiene la ventaja, en comparación con **rgdal**, de funcionar sin las librerías de gdal instaladas. Esto es útil cuando no se tiene permiso para instalar gdal en la máquina, por ejemplo cuando se está utilizando un servidor externo.

```
# Carga los paquetes
library(maptools)

# Determina la ruta del espacio de trabajo CAMBIAR A SU CARPETA!!
# En windows seria algo tipo setwd("C:/Users/jf/Dropbox/libro_SIG/datos-mx")

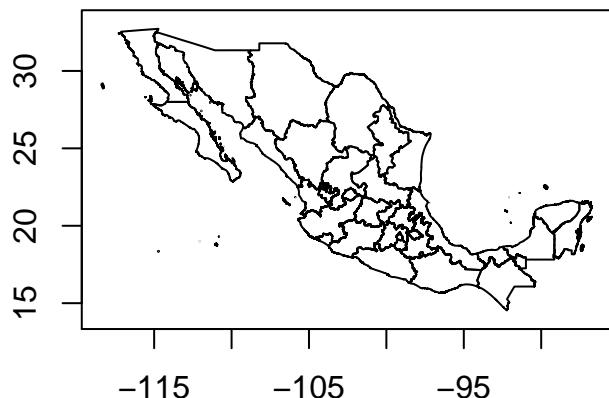
setwd("/home/jf/recursos-mx")

# Con maptools
mx <- readShapePoly("Entidades_latlong.shp")

# Pregunta la clase del objeto espacial
class(mx) # Es un "SpatialPolygonsDataFrame"

## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"

# plotea el mapa
plot(mx, axes=T)
```



```
# Pregunta por el sistema de proyección
proj4string(mx) # No tiene el sistema de proyección definido

## [1] NA

# Define el sistema de proyección
proj4string(mx) <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0"
```

## 2.1.2 Importación con rgdal

La función ***readOGR()*** permite importar coberturas de polígonos, líneas o puntos. Los argumentos de la función son la ruta de acceso al archivo shape el punto significa que el archivo se encuentra directamente en el espacio de trabajo) y el nombre del archivo sin la extensión ".shp" (***readOGR(".", "archivoshape")***).

```
library(rgdal)

## Loading required package: sp
## rgdal: version: 1.2-16, (SVN revision 701)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.11.3, released 2015/09/16
## Path to GDAL shared files: /usr/share/gdal/1.11
## GDAL binary built with GEOS: TRUE
## Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
## Path to PROJ.4 shared files: (autodetected)
## Linking to sp version: 1.2-7

setwd("/home/jf/recursos-mx")
# Importa el mismo shape, esta vez con el paquete rgdal
mx <- readOGR(".", "Entidades_latlong")

## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "Entidades_latlong"
## with 32 features
## It has 2 fields

# Plotea el mapa
```

```

# plot(mx, axes=T)

# Determina la clase
class(mx)

## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"

# Consulta la proyección
proj4string(mx) # Este vez el objeto tiene el sistema de proyección definido

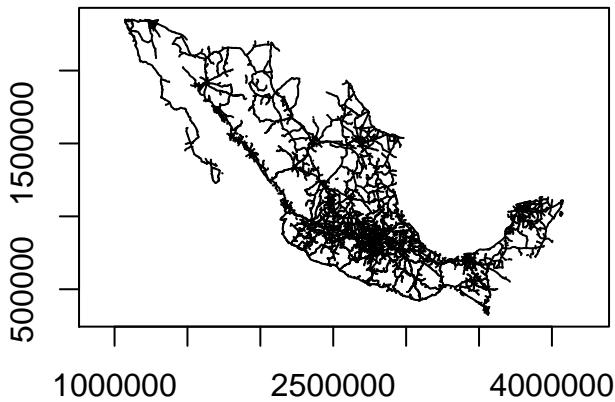
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

# Importación del shape de carreteras pavimentadas
carr <- readOGR(".", "carretera")

## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "carretera"
## with 12424 features
## It has 4 fields

# Plotea el mapa
plot(carr, axes = T)

```



```

# Consulta la proyección
proj4string(carr)

## [1] "+proj=lcc +lat_1=17.5 +lat_2=29.5 +lat_0=12 +lon_0=-102 +x_0=2500000 +y_0=0 +ellps=GRS80

# Son coordenadas en Cónica Conforme de Lambert (lcc)

```

### 2.1.3 Algunas operaciones más en **sp**

Vamos a reproyectar, desplegar en pantalla y exportar los datos. Para reproyectar el mapa de los Estados mexicanos en la proyección conforme de Lambert, se utilizó la función *spTransform()* que tiene como argumentos el nombre del objeto que se desea reproyectar y la proyección de destino. Note que en el código a continuación

anidamos la función `proj4string()` dentro de `spTransform()`.

Para desplegar en pantalla, podemos aquí también usar la función `plot()`, la opción `border` permite escoger el color de los límites de los polígonos. La función `lines()` permite añadir en el mismo despliegue la cobertura de líneas.

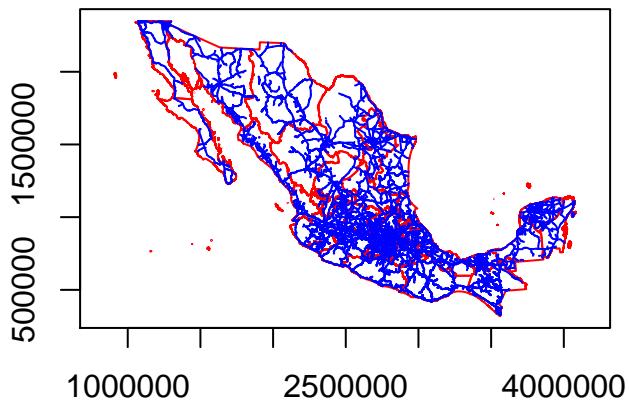
Finalmente, podemos salvar un objeto en shape con la función `writeSpatialShape()`.

```
## Reproyección
## Proyección del mapa de las entidades a la
# misma proyección que el mapa de carreteras (Cónica Conforme de Lambert)
mx_lcc <- spTransform(mx, proj4string(carr))
proj4string(mx_lcc) # Ahora son coord. en Conica Conforme de Lambert (lcc)

## [1] "+proj=lcc +lat_1=17.5 +lat_2=29.5 +lat_0=12 +lon_0=-102 +x_0=2500000 +y_0=0 +ellps=GRS80 +uni

# plotea el mapa (en LCC)
# plot(mx_lcc, axes = T)

# Plotea los estados juntos con las carreteras
plot(mx_lcc, border="red", axes = T)
lines(carr,col = "blue")
```



```
# salva el objeto en formato shape
writeSpatialShape(mx_lcc, "Entidades_LCC.shp")
```

## 2.2 Importación / exportación de datos raster

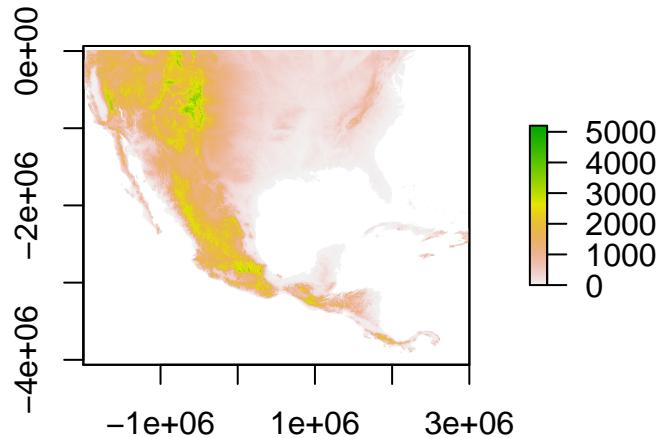
Vamos a importar un modelo digital de elevación, cambiar su proyección, recortarlo y salvar el resultado en formato TIFF utilizando comandos del paquete **raster**.

La función `raster()` permite importar imágenes de muchísimos formatos de imagen en objetos `RasterLayer`. Los comandos `projectRaster()` y `crop()` permiten reproyectar y recortar objetos raster. Note cómo la expresión `crop(projectRaster(raster("DEM_GTOPO1KM.tif"), crs = proj4string(carr)), extent(mx_lcc))` que anida varios comandos permite importar, reproyectar y recortar el DEM original en una sola línea de código. Finalmente, la función `writeRaster()` permite salvar los resultados en varios formatos de imagen incluyendo ENVI, ESRI, ERDAS, GeoTiff, IDRISI y SAGA (opción `format`). La opción `datatype` permite escoger la codificación de los datos. Por ejemplo `INT1U`, es la codificación en 8 bits con valores de 0 a 255 (Tabla 1).

Tabla 1. Tipo de codificación de imágenes

Datatype	Valor mínimo	Valor máximo
LOG1S	FALSE (0)	TRUE (1)
INT1S	-127	127
INT1U	0	255
INT2S	-32,767	32,767
INT2U	0	65,534
INT4S	-2,147,483,647	2,147,483,647
INT4U	0	4,294,967,296
FLT4S	-3.4e+38	3.4e+38
FLT8S	-1.7e+308	1.7e+308

```
library(raster)
setwd("/home/jf/recursos-mx")
# Importa imagen
tmp <- raster("DEM_GTOPO1KM.tif")
plot(tmp)
```



```
extent(tmp)

## class       : Extent
## xmin       : -1999500
## xmax       : 3000500
## ymin       : -3999500
## ymax       : 500

# checa la proyección
proj4string(tmp)

## [1] "+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs"

# checa la clase
class(tmp)
```

```

## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"

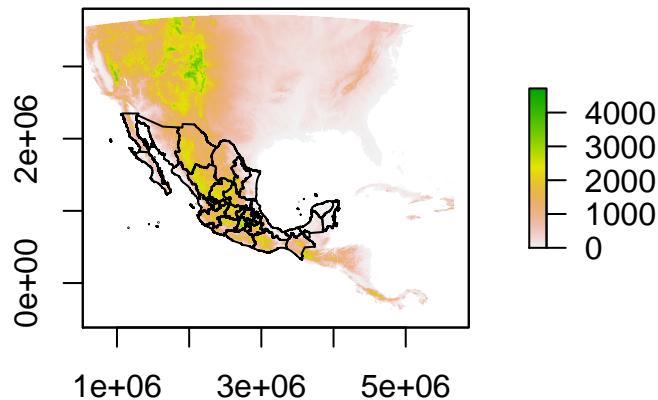
# Reproyecta en LCC

tmp_lcc <- projectRaster(tmp, crs = LCC)
# Plotea el dem con mapa de los estados
mx_lcc <- readOGR(".", "Entidades_LCC")

## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "Entidades_LCC"
## with 32 features
## It has 2 fields

plot(tmp_lcc); plot(mx_lcc, add=TRUE)

```



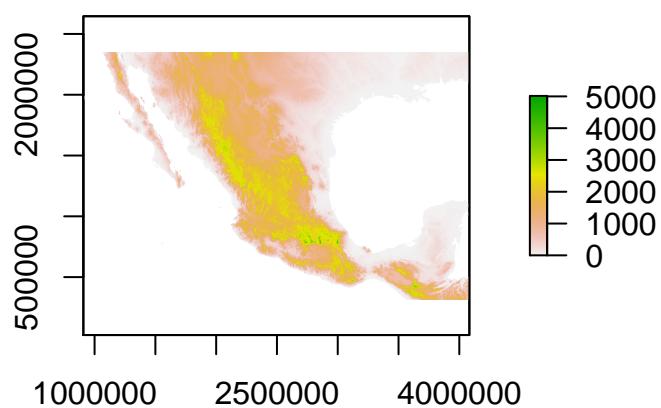
```

# Checa la extensión (coord extremas) del mapa de México
extent_mx <- extent(mx_lcc)
# Corta a la extensión
dem_mx <- crop(tmp_lcc, extent_mx)

# Lo mismo con comandos anidados
dem_mx <- crop(projectRaster(tmp, crs = LCC), extent(mx_lcc))

plot(dem_mx)

```



```
# Salva el raster en formato TIFF
```

European Scientific Institute



# European Scientific Institute

European Scientific Institute  
[www.euinstitute.net](http://www.euinstitute.net)

As a research scientist here at the James Hutton Institute, UK, and formerly at *Observatorio para Una Cultura de Territorio* in Madrid, I have worked with the R platform in the area of Geographical Information Science (GIS) for many years. As a researcher with a long term interest in land use change in Spain and Latin America, I have often felt that a Spanish language manual covering GIS functionality in R would be a great asset to the discipline. As Spanish is the second most spoken native language in the world (>500 million speakers), it is essential that cornerstone educational and scientific support material is available in this language. Aside from offering this essential basic function, Dr. Mas' book comes at an important time for the discipline, which is rapidly evolving away from stand-alone desktop systems and proprietary software suites costing in excess of \$10,000 per license, towards more flexible computing environments, of which R is the pre-eminent example. The book contains a number of essential updates, such as detailed application and use of the new (and still under development) sf package, which makes spatial data manipulation much easier than previously. Finally, I want to draw attention to the importance of making this material available for free. As researchers employed by public institutions, it is our responsibility ensure that the teaching and research materials we use are freely disseminated for all to access and use regardless of their individual economic situation. This is especially significant for the R platform, which has always been, and remains, free and open-source.

**Richard Hewitt**  
**(The James Hutton Institute, UK)**

Jean-François Mas, Ph.D. is a tenured Professor in the Center of Research in Environmental Geography (*Centro de Investigaciones en Geografía Ambiental*, CIGA) at the National University of Mexico (*Universidad Nacional Autónoma de México*, UNAM). He specializes in the fields of remote sensing, geographical information science, and spatial modelling. His research interests include land-use/land-cover change monitoring and modelling, accuracy assessment of spatial data, forest inventory, and vegetation cartography. He has published more than 70 peer-reviewed scientific publications, advised 8 Ph.D. dissertations, supervised 14 master's degree students, and participated in 34 research projects. Please visit his website at <http://www.ciga.unam.mx/index.php/mas> for more detailed information.



978-608-4642-66-4