Análisis espacial con R:

Usa R como un Sistema de Información Geográfica

Jean-François Mas

European Scientific Institute

ESI
European Scientific Institute

La obra de la portada fue realizada por José Cuerda.
Técnica: Acuarela.
Instagram: @josecuerda_
Facebook: https://www.facebook.com/J.CUERDA/

Jean-François Mas

ANÁLISIS ESPACIAL CON R

USA R COMO UN SISTEMA DE INFORMACIÓN GEOGRÁFICA

European Scientific Institute

Jean-François Mas

ANÁLISIS ESPACIAL CON R

USA R COMO UN SISTEMA DE INFORMACIÓN GEOGRÁFICA

Jean-François Mas

ANÁLISIS ESPACIAL CON R
USA R COMO UN SISTEMA DE INFORMACIÓN GEOGRÁFICA

EUROPEAN SCIENTIFIC INSTITUTE, Publishing

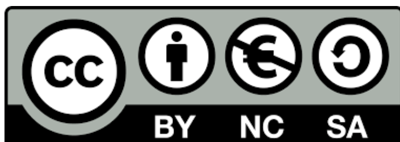
Impressum
Bibliographic information published by the National and University Library "St. Kliment Ohridskiïn Skopje; detailed bibliographic data are available in the internet at http://www.nubsk.edu.mk/;

CIP - 004.4/.6:528.8/.9
COBISS.MK-ID 107892490

Any brand names and product names mentioned in this book are subject to trademark, brand or patent protection and trademarks or registered trademarks of their respective holders.
The use of brand names, product names, common names, trade names, product descriptions etc. even without a particular marking in this works is in no way to be construed to mean that such names may be regarded as unrestricted in respect of trademark and brand protection legislation and could thus be used by anyone.

Publisher: European Scientific Institute
Street: "203", number "1", 2300 Kocani, Republic of Macedonia
Email: contact@eujournal.org

Printed in Republic of Macedonia
ISBN: 978-608-4642-66-4
Copyright © 2013 by the author, European Scientific Institute and licensors
This work is licensed under the Creative Commons *Attribution Non Commercial Share Alike 4.0 International* License. To view a copy of the license, visit https://creativecommons.org/licenses/by-nc-sa/4.0/.



ANÁLISIS ESPACIAL CON R

USA R COMO UN SISTEMA DE INFORMACIÓN GEOGRÁFICA

JEAN-FRANÇOIS MAS

July, 2018

AGRADECIMIENTOS

La presente publicación recibió apoyo financiero del Fondo Sectorial de Investigación para la Educación SEP-CONACYT (Proyecto número 178816 «*¿Puede la modelación espacial ayudarnos a entender los procesos de cambio de cobertura/uso del suelo y de degradación ambiental?*»).

Se terminó de elaborar durante una estancia sabática en la *Universidade Federal da Bahia* y la *Universidade Estadual de Feira de Santana*, BA, Brasil con el apoyo del Programa de Apoyos para la Superación del Personal Académico (PASPA) de la Dirección General de Asuntos del Personal Académico - Universidad Nacional Autónoma de México.

Agradezco a Gabriela Cuevas García y Richard Hewitt por la minuciosa revisión del manuscrito que realizaron.

Agradezco también a los desarrolladores de los programas de código abierto que se presentan en este libro y que se usaron para su elaboración (Knitr y LaTeX).

La obra de la portada fue realizada por José Cuerda, quién muy gentilmente me dió el permiso para usarla (https://www.facebook.com/J.CUERDA/).
Técnica: Acuarela.
Instagram: @josecuerda_

Índice general

Introducción 7

0.1 Qué es R? 7

0.2 Propósito de este libro 7

0.3 Organización del libro y convenciones 8

0.3.1 Organización del libro 8

0.3.2 Convenciones de escritura 8

1 Instalación y presentación de R y RStudio 9

1.1 Instalación de R y RStudio 9

1.1.1 Windows 9

1.1.2 Linux 9

1.1.3 Mac 10

1.2 Introdución a R y Rstudio 10

1.3 Instalación de paquetes de R 10

1.4 Una sesión de R 11

2 Operaciones básicas en R 13

2.1 Operaciones básicas 13

2.2 Importación de datos en R 16

2.3 Operaciones con tablas 18

2.4 Elaboración de gráficas 24

2.5 Relación entre dos variables 25

2.6 Operaciones marginales: apply 27

2.7 Operaciones por grupos 28

2.8 Creación de funciones 29

2.9 Repeticiones y condiciones 30

2.10 Operador pipe %$>$% 31

2.11 Más sobre R 32

3 Organización de los objetos espaciales en R 33

3.1 Datos vectoriales: modelo *simple feature* 33

3.1.1 Cobertura de puntos 34

3.1.2 Cobertura de líneas 38

3.1.3 Cobertura de polígonos . . . 40
3.2 Datos raster: Clase *RasterLayer* en el paquete raster 44

4 Importación/exportación de datos espaciales . . . 47
4.1 Importación de archivos *shape* 47
4.2 Importación de archivos vector de otros formatos 50
4.3 Exportación a *shape* o a otros formatos 51
4.4 Importación / exportación de datos raster 51

5 Operaciones básicas de SIG (*vector*) . . . 55
5.1 Algunas operaciones de análisis espacial 55
5.2 Análisis espacial en formato vector 59

6 Operaciones básicas de SIG (*raster*) . . . 67
6.1 Algunas operaciones de análisis espacial 67
6.2 Análisis espacial en formato *raster* 75

7 Análisis geoestadístico: Detección de *hot spots* . . . 81
7.1 Método de Getis Ord 81
7.2 Aplicación a la detección de áreas con altas tasas de deforestación 81

8 Análisis de imágenes de percepción remota . . . 85
8.1 Lectura de imágenes de satélite 85
8.2 Visualización y preprocesamintos 87
8.3 Clasificación 89

9 Elaboración de mapas . . . 91

10 Poniendo R a interactuar con QGIS y Dinamica . . . 95
10.1 Sistema de Información Geográfica QGIS 95
10.2 Plataforma de modelación Dinamica EGO 96

Posfacio . . . 101

Referencias . . . 103

Anexos . . . 105

Prefacio

El objetivo de un prefacio, antes que todo, es animar a los lectores a seguir adelante, poniendo en su contexto el objetivo principal del libro. Y aunque se puede acometer esta tarea simplemente resaltando la importancia y relevancia del libro tratada de manera imparcial, por este camino corremos el riesgo no solo de aburrirnos, autores y lectores por igual, sino también de caer en la trampa de pensar que R es simplemente un software, un paquete estadístico que podemos dominar en una tarde o dos, y de esa manera hacernos la vida más fácil. No tengo ganas de desanimar, a aprender a manejar R con suficiente fluidez para hacer la vida realmente más fácil (en vez de bastante más difícil, que es lo que suele ocurrir al principio sobre todo para quienes no estamos acostumbrados a trabajar con la línea de comandos), pero hay que ser consciente de que lleva su tiempo. Me acuerdo de la primera vez que intenté sacar unas simples gráficas de barra en R, y pasé varias tardes jalándome el pelo de la frustración, al principio ni siquiera capaz de dibujar la columna de datos que quería. Cuando conseguí por fin hacer esto, no podía ajustar la gráfica a la página, ni ponerle un título, ni aumentar la fuente tipográfica de los ejes, o dar a cada barra su propio color. Con una semana perdida, y ante la presión de tener algo concreto para mostrarle a mi jefe, volví, cabizbajo y humillado, a Microsoft Excel. Así aprendí la primera lección, que R, por muy bueno que sea (y es muy bueno) no es necesariamente la mejor opción para realizar muchas de las tareas sencillas. Si solo necesitas hacer una gráfica de barra o de pastel para visualizar unos datos, convertir una columna de datos en porcentaje, o, algo relacionado al análisis espacial, dibujar un par de mapas con escala y leyenda, hay sin duda maneras más fáciles de conseguirlo.

Pero claro, tarde o temprano, nos vamos a topar con algo más complicado, sobre todo en la época de "Big Data". ¿Qué pasa si tenemos que hacer, por ejemplo, no solo un par de gráficas de barra, sino doscientas? ¿O si queremos hacer un análisis espacial y luego presentar los resultados en forma de mapa, con las tablas o gráficas juntas, e incluso publicarlo en la web? ¿Y si no tenemos ArcGIS, o nos falta una de sus infinitas extensiones? ¿O trabajamos con Mac o con Linux? ¿O nos cansamos de instalar un programa diferente para cada paso de nuestro análisis? ¿O nos encontramos con un análisis o unos datos que no se ajustan a una de las funciones estándar del software SIG? En estos casos, y otros muchos, R sí te va a hacer la vida más fácil. Y el tiempo que inviertes en escribir un script lo vas a recuperar más adelante, y con creces, sobre todo cuando empiezas a memorizar algunas operaciones sencillas y a reciclar tu código antiguo para nuevos problemas.

Si eso te suena muy hipotético, consideremos un ejemplo concreto. Sabemos que la exportación de nutrientes como el nitrógeno desde los campos de cultivo perjudica a los ecosistemas acuáticos. También sabemos que algunos cultivos tienden a provocar más exportación de nutrientes que otros. Pero a la vez, los gobiernos tratan a menudo de incentivar los cultivos más rentables mediante subvenciones, y así surge la duda, ¿puede que se esté fomentando la exportación de nutrientes a los ríos a través de las subvenciones agrícolas? Para responder a esta pregunta, necesitamos ver si hay alguna relación entre la exportación de nutrientes desde los campos de cultivo y la subvención que recibe cada cultivo bajo el plan de desarrollo agrícola vigente. Por una parte, contamos con un mapa de cultivos, podemos estimar la exportación de nutrientes en cada punto del territorio, en función del cultivo en este lugar y otras variables como la pendiente del terreno, que podemos obtener a través de un modelo digital de terreno. Por otro lado, la información sobre los pagos que recibiría cada cultivo está en formato de tabla (.xlsx), y tendríamos que unirlo al mapa de cultivos (archivo "shape"). Luego, al mapa raster que hemos obtenido, mostrando exportación de nutrientes en cada celda, lo utilizamos para hacer la suma por cultivo (análisis de estadística de zonas) u otra unidad relevante

(p.e. parcelas, cuenca). Finalmente, para analizar la relación entre la cantidad de nutrientes exportados en un lugar determinado y el pago correspondiente, hacemos un análisis de regresión. Todo esto, claro está, se puede hacer con un software SIG, como ArcGIS, GRASS o QGIS. La regresión la podemos hacer exportando tablas a un software estadístico, como SPSS. ¡Pero espera un momento! Como los cultivos varían según la zona geográfica, deberíamos explorar, por ejemplo, 3 estudios de caso con regímenes agrícolas contrastantes. Y resulta que tenemos 3 mapas de cultivos, para 3 fechas distintas (2008, 2010, 2015). Hay 4 escenarios de subvención, que corresponden a diferentes planes de desarrollo agrícola. Queremos saber si el tamaño de la unidad geográfica influye, es decir queremos sumar los valores en el mapa raster por cultivos, por campos, y por cuenca hidrográfica y también por hectárea o por celda para eliminar la posibilidad de que los resultados de la correlación estén ligados al tamaño de la unidad geográfica, el conocido "Modificable Aerial Unit Problem" o MAUP. Y no estamos seguros de qué es más relevante: si el nutriente exportado depende del pago, o si el pago depende del nutriente exportado, es decir, deberíamos hacer el análisis de regresión en ambas direcciones. En fin, ahora necesitamos hacer 3 x 3 x 4 x 4 x 2 = 288 regresiones, cada una con su gráfica de tendencia en formato .jpg y exportar los resultados a una tabla que será incluida en nuestro informe. Los datos de entrada son de varios formatos, y es muy posible que quisiéramos introducir alguna modificación después: ¿en vez de nutrientes, quizás interesa ver sedimento exportado? ¿Y si clasificamos el mapa de cultivos de otra forma para arrojar luz sobre unos cultivos en particular?

Esto ya no es una situación hipotética, sino la descripción de una investigación actual mía, que relato aquí porque, sinceramente, no sé cómo la hubiera realizado sin haber contado con la amplísima gama de funciones estadísticas y geoespaciales que aporta el entorno R. Con R he podido hacer la mayor parte del análisis geográfico, también usé GRASS, aunque éste se puede vincular a R directamente y llamar sus potentes herramientas de procesamiento con la librería rgrass7. Como se explica en el Capítulo 10 del libro, lo mismo se puede hacer con QGIS con el paquete RQGIS. Con R he podido manejar los múltiples formatos en que se encuentran los datos originales, tanto espaciales (.tif, .adf, .shp, etc.) como no espaciales (.csv, .xlsx, .jpg, .png, .pdf) de manera simultánea en un solo script. Con R he podido importar todas las capas a la vez, pasar la tabla de atributos a un dataframe, limpiarla (eliminar filas con valor 0 o nulo), calcular el área de los polígonos y la exportación y pago por hectárea, realizar una regresión con un modelo aditivo generalizado (GAM) para cada capa, cada año y cada tipo de exportación (nutrientes o sedimentos) y generar cientos de graficas con una plantilla y formato idénticos. Aunque preparar los scripts me ha costado varios días de trabajo, hacer este análisis, que no es de ninguna manera complejo ni difícil, con un SIG ordinario y software de Office, me hubiera costado meses, y lo peor, me hubiera dejado al borde de un ataque de nervios.

Finalmente, para no extenderme más, quiero terminar con una reflexión menos técnica que la de los párrafos anteriores. En un mundo azotado por el neoliberalismo, ideología extremista que distribuye el poder y los recursos de la población hacia arriba – una especie de *Robin Hood* inverso – concentrándolos en manos de gigantescas empresas multinacionales bajo el falso paradigma de crecimiento sin límites, frente a eso, un sistema gratuito y compartido, propiedad de sus usuarios y contribuyentes, es una herramienta potente en la lucha por el control de información entre los ratones pequeños y los peces gordos. En este sentido, R tiene mucho en común con otros movimientos de conocimiento abierto que actualmente se están despertando en muchos rincones del mundo. Consideremos por ejemplo, Arduino, una plataforma electrónica open source basado en hardware y software fácil de usar (https://www.arduino.cc/). El co-fundador de esta, Massimo Banzi, ha comentado, a modo de analogía, de que si en el mundo solo podemos comprar comida preparada, tarde o temprano, va a llegar alguien que quiera cocinar con los ingredientes frescos. Arduino aporta un sistema para construir de modo abierto dispositivos electrónicos de bajo costo limitados solamente por la imaginación del desarrollador, todo eso, sin formación alguna en ingeniería o electrónica. R, a mi juicio, aporta algo parecido a la ciencia. Hoy en día R es ubicuo en la ciencia, su uso ya es habitual en disciplinas tan diversas como la medicina, la ecología, o la arqueología. Esta capacidad unificadora que tiene R es de especial relevancia para los SIG, enfoque del presente libro, ya que estos están dejando de ser sistemas, procesos o plataformas aparte y separados de otras herramientas, para volverse completamente integrados en entornos de diversa índole. El claro líder de este nuevo mundo integrado de los SIG, tanto por su capacidad de

adaptarse o enlazarse a otros lenguajes, disciplinas o plataformas, como por su espíritu abierto y libre, es R.

Por último, lo más importante que he aprendido a través de mi propia experiencia como usuario de R, es que una de sus grandes fortalezas es su comunidad de usuarios, que es muy grande y muy activa. Para solucionar la mayoría de los problemas que solemos encontrar en el día-a-día, basta con unos minutos de experimentación con base en sugerencias de otros usuarios. Es decir, he aprendido que saber encontrar la solución es más importante que saber la solución.

Dr Richard Hewitt

Introducción

0.1 Qué es R?

R, una plataforma de análisis estadístico con herramientas gráficas muy avanzadas, es un referente en el análisis estadístico desde hace muchos años. Se puede obtener y distribuir R gratuitamente debido a que se encuentra bajo la Licencia Pública General (GPL por sus siglas en inglés) del proyecto colaborativo de software libre GNU. Esta licencia tiene por propósito declarar que el software es libre, y protegerlo de intentos de apropiación que restrinjan esas libertades a nuevos usuarios cuando el software es distribuido o modificado. Por ser un programa de código abierto, R es por lo tanto gratis pero, sobre todo, es fruto del esfuerzo de miles de personas en todo el mundo que colaboran en su desarrollo. Esto permite solucionar problemas de errores de programación (*bugs*) muy rápidamente aí como el desarrollo de paquetes modulares, que son complementos especializados para temas específicos, desarrollados por especialistas en alguna parte del mundo. Estos paquetes se basan a menudo en métodos muy innovadores y permiten utilizar R para una amplia gama de problemas: existen paquetes para procesar datos tan diversos como censos, series de tiempo, secuencias genéticas o información económica así como implementar una gran variedad de métodos estadísticos[1]. Existen también, varios paquetes para la elaboración de gráficas y otros para hacer interactuar código de R y procesadores de texto como Latex para la elaboración de reportes. Este libro está elaborado utilizando estos paquetes, más específicamente el paquete knitr (Xie, 2013). Durante los últimos años, se crearon diferentes paquetes dirigidos al análisis espacial, volviendo R una potente herramienta para llevar a cabo el mapeo y el análisis de todo tipo de información georeferenciada.

Finalmente, es importante resaltar que la reproducibilidad es uno de los principios de la investigación científica. El uso de programas computacionales de código abierto como R garantiza que otro investigador pueda repetir el experimento, comprobar los resultados obtenidos y estar en condición de ampliar o refutar las interpretaciones del estudio realizado.

0.2 Propósito de este libro

A primera vista, R puede parecer poco amigable a usuarios acostrumbrados a manejar programas computacionales con menús y opciones selecionadas con el ratón debido a que se basa en líneas de comando. Sin embargo, después de haber (fácilmente) superado este obstáculo, estos usuarios verán que el uso de pequeños guiones ”*scripts*” que permiten ejecutar una secuencia de operaciones, es mucho más eficiente que una larga secuencia de ”clics”, sin olvidar la reducción del riesgo de tendinitis. Permite repetir facilmente el mismo procedimiento con datos diferentes o realizar modificaciones a una cadena de procesamiento ya implementada. Adicionalmente, reduce enormemente la posibilidad de cometer errores en una cadena de operaciones rutinarias y permite documentar el procesamiento realizado.

Desafortunadamente, existen muy pocos libros enfocados al análisis espacial con R[2] y, de nuestro conocimiento,

[1] Para convencerse ver la lista de los paquetes en https://cran.r-project.org/

[2] Consultar los libros de Bivand et al. (2008) y Brunsdon & Comber (2015)

solo uno en español (Cabrero Ortega & García Pérez, 2014). El presente libro se dirige a usuarios con conocimiento básico de Sistemas de Información Geográfica (SIG) que desean iniciarse en el manejo y análisis de datos espaciales en R. No requiere por lo tanto de ningún conocimiento previo de este programa pero si un conocimiento básico de los SIG. El libro pretende permitir al lector dar los primeros pasos en el manejo de R para el análisis espacial sin demasiados tropiezos. Para seguir con aplicaciones más avanzadas, existe un gran número de fuentes de información (ver anexos).

0.3 Organización del libro y convenciones

0.3.1 Organización del libro

El libro se organizó de la siguiente manera: en el primer capítulo se explica como instalar R y RStudio y se presentan los principales elementos de la interface RStudio. Se recomienda realizar la instalación de ambos programas para poder experimentar los códigos de los capítulos siguientes. En el segundo capítulo, se hace una iniciación al manejo básico de R. El lector con conocimiento previo de R puede pasar directamente al siguiente capítulo. En el tercer capítulo, se presenta como están estructurados los datos espaciales en R en los paquetes **sf** y **raster**, los dos paquetes que vamos utilizar a lo largo de este libro. La estructura de los datos en el paquete **sp** se encuentra en anexos. Este capítulo puede parecer un poco árido. De hecho se puede manejar información espacial sin entrar en los detalles de la organización de la información. Sin embargo, es importante, y ayuda mucho, conocer esta información. En el capítulo 4, se presentan algunas formas para intercambiar datos geográficos entre R y otros sistemas de manejo de información geográfica a través de procedimientos de importación / exportación entre R y datos en formato vectorial o de imagen, así como algunos métodos para convertir información entre vector y raster. En los capítulos 5 y 6, se presentan operaciones básicas de SIG, respectivemente con datos en formato vector y raster. En el capítulo 7, se muestran algunos de los numerosos análisis de tipo geoestadístico que se puede llevar a cabo con paquetes de R. En el octavo capítulo, se aborda el análisis de imágenes de satélite. En capítulo 9 muestra algunas formas de elaborar cartografía. Finalmente, el décimo capítulo introduce al lector las técnicas para hacer interactuar R con el programa SIG de código abierto Q-GIS y la plataforma de modelación espacial Dinamica EGO.

Los datos (tablas, scripts, mapas e imágenes) para llevar a cabo las operaciones presentadas en este libro se encuentran en http://lae.ciga.unam.mx/recursos/recursos-mx.zip o https://www.dropbox.com/s/avzsjb8vtraiyxi/recursos-mx.zip?dl=0.

0.3.2 Convenciones de escritura

Para facilitar la lectura de este documento, los nombres de paquetes, como **maptools** están resaltados en **negrita** y los nombres de comandos en ***"negrita/itálica"***. Las líneas de comando, tal como se escriben en un script o en la consola están en `verbatim` en párrafos especiales con un fondo gris. El tipo o clase de los objetos están en `typewriter`. Algunos anglicismos están en *itálicas* como en el ejemplo a continuación. La función ***stepAIC()***, del paquete **MASS** usa *por default* la opción "forward" como se puede observar en el código a continuación:

```
stepAIC(fit)
stepAIC(fit, direction="forward")
```

1. Instalación y presentación de R y RStudio

R está disponible para los sistemas operativos Linux, Windows y Mac. Existen varias formas de obtenerlo e instalarlo. RStudio es una interface gráfica muy útil para utilizar R y la usaremos en los ejercicios de este libro. Sin embargo, es posible utilizar R sin ninguna interface o bien con otras como, por ejemplo, R Commander, RKWard o Tinn-R.

1.1 Instalación de R y RStudio

1.1.1 Windows

- Para obtener R para Windows entre en la página del *Comprehensive R Archive Network* (CRAN) https://cran.r-project.org/mirrors.html.
- Escoja el espejo de su preferencia (*CRAN mirrors*).
- Clique en *Download R for Windows* e *Install R for the first time*.
- Clique en *Download R 3.4.0 for Windows* (o la versión más actualizada disponible), salve el archivo de R para Windows y ejecutarlo.

El ejecutable para instalar la versión gratis de RStudio para Windows puede bajarse de la página web de RStudio (https://www.rstudio.com/products/rstudio/download/).

1.1.2 Linux

R está incluido en los repositorios de la mayoría de las distribuciones de Linux. Por ejemplo, en Ubuntu, se instala fácilmente utilizando el centro de software. Estos repositorios no tienen siempre la última versión de R. Para instalar la última versión, se puede seguir los pasos a continuación:
Eliminar las versiones anteriores (en caso de existir instalaciones previas):

```
sudo apt-get remove --purge r-base*
```

Actualizar el sistema:

```
sudo apt-get update && apt-get -y upgrade
```

Importar la clave pública:

```
gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
gpg -a --export E084DAB9 | sudo apt-key add
```

Añadir el repositorio de R en el archivo /etc/apt/sources.list (la línea de comando abajo es para la versión Ubuntu Xenial (Ubuntu 16.04 VPS), adaptar a su propia versión de Linux):

```
sudo echo "deb http://cran.rstudio.com/bin/linux/ubuntu xenial/" |
sudo tee -a /etc/apt/sources.list
```

Instalar la última versión de R:

```
sudo apt-get update
sudo apt-get install r-base r-base-dev
```

En https://www.rstudio.com/products/rstudio/download/, se encuentran los archivos de instalación de RStudio para diferentes distribuciones de Linux. La instalación puede llevarse a cabo utilizando programas como Synaptic, el centro de software o en la terminal.

1.1.3 Mac

Los usuarios de Mac encontrarán los archivos de instalación de R en la página https://cran.r-project.org/bin/macosx/ y los de RStudio en http://www.rstudio.com/products/rstudio/download/.

1.2 Introdución a R y Rstudio

Como ya se mencionó, los usuarios acostumbrados con otros programas notarán la falta de "menús" (opciones para *clicar*). Para usar R es necesario escribir los comandos. Sin embargo, una vez acostumbrados, los usuarios notarán que el mecanismo de comandos es más flexible y permite la programación.

Es importante tomar en cuenta algunos detalles: R es *case-sensitive*, es decir sensible a la diferencia entre minúsculas y mayúsculas) y por lo tanto "Nombre" es diferente de "nombre". El separador de decimales es el punto ".", la coma se usa para separar los elementos de una lista[1]. Se recomienda evitar el uso de acentos en las rutas y los nombres de archivos. En los ejercicios a continuación usaremos R y Rstudio para analizar datos espaciales. RStudio es una interface gráfica de R que, a su vez es el lenguaje de programación. Demos un vistazo a esta interface (Figura 1):

- La ventana en la parte inferior izquierda es la consola, donde se digitan los comandos de línea. Ella interpreta cualquier entrada como un comando a ser ejecutado. Estos comandos y su sintaxis proporcionan una forma bastante natural e intuitiva de acceder a datos y realizar procesamientos y operaciones estadísticas. Sin embargo, es más fácil escribir su código como un guión (*script*): un texto con una secuencia de operaciones a ser ejecutadas.
- La ventana en la parte superior izquierda es el editor de texto de los *scripts*. Un *script* es un archivo de texto con una serie de instrucciones. Se puede ejecutar una sola línea del *script*, un conjunto (bloque) de líneas o bien el *script* entero.
- Los gráficos elaborados aparecen en la ventana inferior derecha. El área de trabajo y el histórico se encuentran en la esquina superior derecha.

1.3 Instalación de paquetes de R

Al instalar R, se instala solamente una configuración mínima para un funcionamento básico del *software*. Para realizar tareas más específicas, es a menudo necesario instalar paquetes adicionales (*packages*). Existen más de 10,000 paquetes disponibles en la página web de R (www.r-project.org). Esta estructura modular es una de los pilares de R, lo distingue de casi cualquier otro software y lo convierte en un entorno para computación muy completo.

[1] Para usuarios con archivos en el que el separador de decimales es la coma, R ofrece algunas opciones, como la función read.csv2 en vez de read.csv.

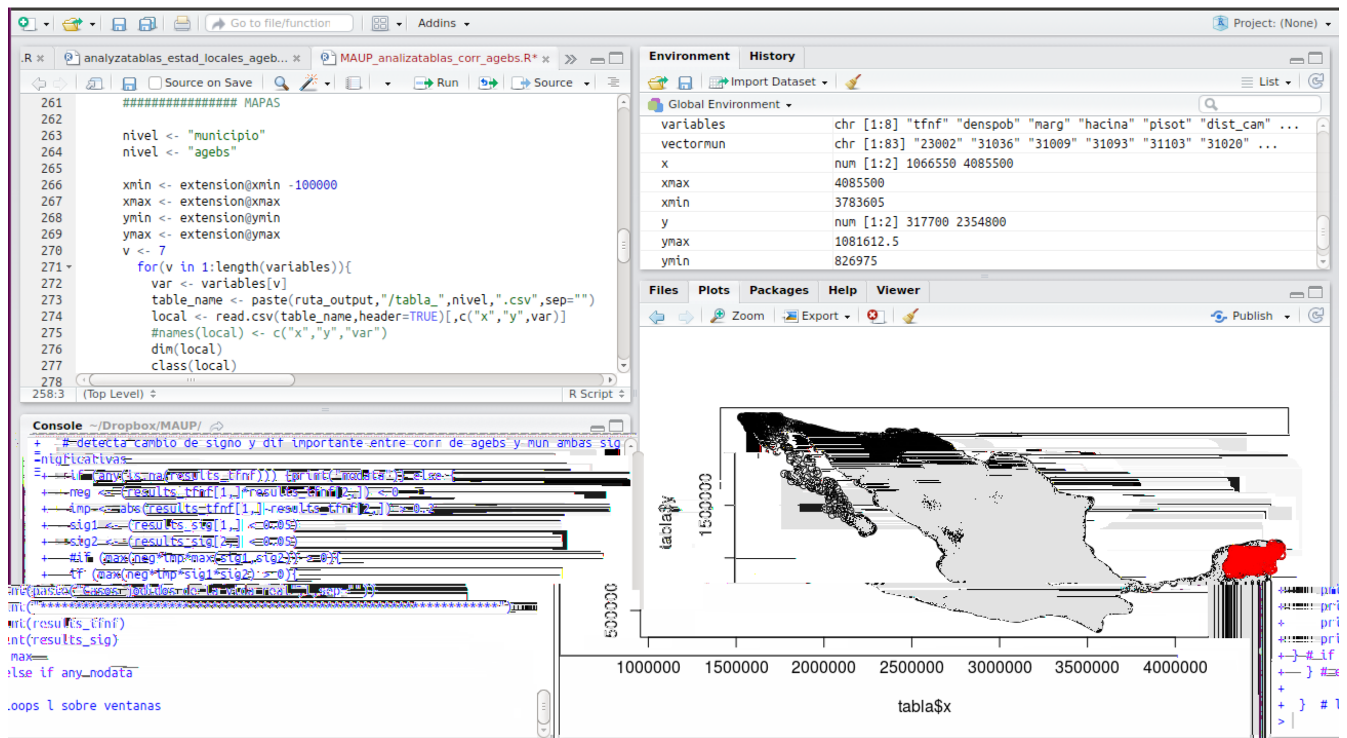


Figura 1. Ambiente de trabajo de RStudio

Para instalar un paquete, hay varias opciones. En la primera, se dispone de una conexión a internet y R se conecta directamente a un repositorio. De esta forma es posible instalar paquetes con la función ***install.packages("nombre-del-paquete")***. Por ejemplo, `install.packages("maptools")` instalará el paquete llamado **maptools**.

La segunda opción consiste en obtener los archivos de instalación (windows binary o bien Package source para Linux) de la página de R e instalar los paquetes a partir de estos archivos locales (sin conexión a internet) con la función ***install.packages()*** siguiendo la sintaxis:
Para Linux:

```
install.packages("maptools_0.9-2.tar.gz", repos = NULL, type=”source”)
```

Para Windows:

```
install.packages("maptools_0.9-2.zip", repos = NULL, type=”source”)
```

Los usuarios de Windows necesitan tener acceso al compilador de C (gcc) y al ejecutable *make.exe*. Para ello, tienen que instalar RTools. A veces R no detecta RTools y hay que añadir la ruta de RTools a la variable del sistema PATH y reiniciar la sesión. Desde R, la manera mas rápida de saber si tiene acceso a RTools es utilizando el comando `Sys.getenv('PATH')` que tiene que indicar Rtools en la lista arrojada:
`[1] "c:\\Rtools\\bin;c:\\Rtools\\gcc-4.6.3\\bin;`

1.4 Una sesión de R

Cuando se trabaja con R todos los datos están generalmente en la memoria viva de la computadora. A pesar de la capacidad de las computadoras actuales, eso puede ser un problema cuando se manejan bases de datos muy

grandes. La función ***rm()*** permite borrar un objeto para liberar memoria. Por exemplo, `rm(mapa)` eliminará el objeto llamado mapa. `rm(list=ls())` borrará todos los objetos del espacio de trabajo.

Al momento de salir de una sesión de R, se le preguntará *Save workspace image?*, lo cual permite salvar los datos y utilizarlos en una futura sesión.

Para llevar a cabo los análisis presentados en este libro y ejecutar el código de R, puede editar su propio script en RStudio (File > New File > R script) digitando el código (o utilizando las funciones de copiar y pegar en el archivo pdf del libro) o bien abrir el script correspondiente al capítulo disponible en la carpeta de recursos del libro. En windows, para que RStudio haga un despliegue correcto de los acentos (comentarios en español), es necesario volver a abrir el script con File > Reopen with encoding (UTF-8). En RStudio, seleccionar las líneas de comando que se desea ejecutar con el ratón, y dar un clic en el ícono de ”Run”. Para correr el script línea por línea, poner el cursor en la linea deseada, y dar clic en ”Run” (o alternavivamente, seleccionando las teclas control y R).

2. Operaciones básicas en R

En este capítulo, vamos a dar nuestros primeros pasos en R. Los lectores que ya conocen el manejo básico de este programa pueden pasar al capítulo siguiente. Invito a los lectores a probar los códigos a continuación en R, es la mejor forma para aprender este lenguaje.

2.1 Operaciones básicas

R es un lenguaje orientado a objetos: un objeto puede verse como un contenedor de información, se vierte contenido a este objeto con la flecha de asignación que se escribe con el símbolo ”mayor que” seguido de menos: <-. Los objetos son guardados en la memoria activa de la computadora, sin usar archivos temporales[1]. Por ejemplo, a continuación, asignamos respectivamente los valores 5 y 4 a dos objetos llamados primervalor y segundovalor. En seguida, creamos un nuevo objeto, llamado suma, que recibe el resultado de la suma de los dos primeros. La función ***print()*** permite visualizar el contenido de un objeto (el mismo resultado se obtiene simplemente con el nombre del objeto). En la consola, es posible repetir líneas de comando anteriores activando la flecha hacia arriba del teclado.

```
primervalor <- 5   # primervalor = 5 es aceptado pero no recomendable
segundovalor <- 4
suma <- primervalor + segundovalor
print(suma)

## [1] 9

suma

## [1] 9
```

R es un lenguaje interpretado y no compilado como C o Fortran por ejemplo, lo cual significa que los comandos son ejecutados directamente sin necesidad de construir ejecutables. Adicionalmente, la sintáxis de R es muy sencilla e intuitiva. Por ejemplo, la función ***sum()*** realiza la suma de todos los argumentos (elementos adentro de los paréntesis). En R, los comandos y el lenguaje de programación son lo mismo de tal manera que aprender a usar R y aprender a programar R son lo mismo, a diferencia de programas como ArcGIS por ejemplo, en el cual la linea de comandos usa una sintaxis diferente que sus lenguajes de scrips (VB, Python etc). En las líneas de código a continuación, el resultado de esta operación se vierte en el objeto suma, el cual ya creamos en la operación anterior y, por lo tanto, se sobreescribe. Esta práctica de sobreescribir (”reciclar”) variables permite evitar de llenar el espacio de trabajo con una gran cantidad de archivos temporales.

```
suma <- sum(primervalor,segundovalor,primervalor)
print(suma)

## [1] 14
```

[1] Algunos usuarios utilizan el símbolo = en vez de <-, lo cual no recomendamos.

Los objetos tienen un nombre y un contenido, pero también atributos que especifican el tipo básico de los datos representados por el objeto. Los tipos de datos en R pueden ser numéricos (números reales), enteros, carácteres, lógicos (TRUE/FALSE) y números complejos. Existen también tipos derivados como los factores, números enteros apuntando a niveles de carácteres (categorías). A menudo, los carácteres se convierten en factores cuando se importan datos en R. Existen muchas clases de objetos, en este capítulo vamos a ver algunas de ellas:

1. `vector`: secuencia de valores numéricos o de carácteres (letras, palabras).
2. `dataframe`: Tabla (arreglo bidimensional) formada por líneas (observaciones) y columnas (variables), acepta columnas de tipos diferentes (por ejemplo columnas numéricas y otras con carácteres).
3. `matriz`: Como el dataframe es un arreglo bidimensional. Es indizado por filas y columnas, es decir que una celda se identifica por su número de filas y columna. Todos los elementos deben ser del mismo tipo (todos numéricos por ejemplo).
4. `lista`: sirve para juntar objetos que pueden pertenecer a tipos distintos.

Abra RStudio en su computadora e inicie un nuevo Script en “File” > “New File” > “New Rscript”. Para elaborar su script, puede capturar los comandos manualmente en el editor de script o copiar y pegar de este documento. También, puede usar el script Cap2.R que se encuentra en la carpeta de los recursos del libro.

Vamos a crear un vector, llamado Prec, que contiene los valores de precipitación mensual observados en Ensenada, Baja California durante 2012. En el script, se pueden escribir comentarios, que comienzan con ”#”. Como ya lo vimos, la operación de asignación tiene la sintaxis objeto (Prec) recibe (<-) valor(es). Utilizamos la función ***c()*** que permite combinar elementos. De la misma forma, vamos a crear dos vectores más:

- Un vector llamado meses, que contiene el nombre de los cuatro primeros meses del año, note que estos nombres se escriben entre comillas, de lo contrario R interpretaría estos nombres como nombres de objetos y no como texto.
- Un vector llamado numeros, que va a recibir una secuencia de números consecutivos entre uno y cuatro escribiendo 1:4.

```
# Datos de precipitación mensual en Ensenada, Baja California
Prec <- c(15, 40, 37, 37, 0, 0, 0, 0, 0, 7, 3, 77)
meses <- c("Enero", "Febrero", "Marzo", "Abril")
numeros <- 1:4
```

La función ***ls()*** nos brinda una lista de los objetos contenidos en el espacio de trabajo. Para conocer el tipo de los elementos contenidos en estos objetos, se puede utilizar la función ***class()***, poniendo como argumento el nombre del objeto, por ejemplo `class(Prec)`. Cuidado! Recuerda que R hace la diferencia entre mayúsculas y minúsculas. Por lo tanto, prec no es lo mismo que Prec y en este caso el objeto prec no existe. De forma similar, podemos preguntar a R si un objeto es un vector con la función ***is.vector()***, la respuesta será TRUE o FALSE. La función ***length()*** nos indicará la longitud del objeto (número de elementos).

```
  # Lista de los objetos
  ls()

## [1] "meses"        "numeros"      "Prec"
## [4] "primervalor"  "segundovalor" "suma"

# Muestra el tipo de datos
class(Prec)

## [1] "numeric"
```

```
class(meses)

## [1] "character"

class(numeros)

## [1] "integer"

print(prec) # prec no existe, es Prec

## Error in print(prec): objeto 'prec' no encontrado

# Pregunta es un vector?
is.vector(Prec)

## [1] TRUE

is.vector(meses)

## [1] TRUE

is.vector(numeros)

## [1] TRUE

# Indica la longitud del vector
length(Prec)

## [1] 12

length(meses)

## [1] 4

length(numeros)

## [1] 4
```

Podemos desplegar en pantalla (”imprimir”) el contenido de un objeto con ***print()*** y realizar estadísticas básicas con ***max()***, ***min()***, ***mean()*** y ***sum()***.

```
# Muestra el contenido del objeto en pantalla
print(Prec)

##  [1] 15 40 37 37  0  0  0  0  0  7  3 77

print(meses)

## [1] "Enero"   "Febrero" "Marzo"   "Abril"

print(numeros)
```

```
## [1] 1 2 3 4

# Calcula estadísticas básicas (Máxima, mínima, promedio y suma)
max(Prec)  # máximo

## [1] 77

min(Prec)  # mínimo

## [1] 0

mean(Prec) # promedio

## [1] 18

sum(Prec)  # suma

## [1] 216
```

2.2 Importación de datos en R

Muchas veces, es laborioso capturar datos y es más conveniente utilizar archivos externos como hojas de cálculo o rchivos de texto guardados en alguna carpeta de la computadora. R utiliza el espacio de trabajo, es decir la ruta de una carpeta, para leer y escribir archivos *por default*. Para definir este espacio, se puede utilizar la función ***setwd()***. Para saber cual es el espacio de trabajo, se utiliza ***getwd()***. Las rutas de los archivos se definen con respecto a la ruta del espacio de trabajo. Por ejemplo, en la figura 2 si definimos `\home\User\Documentos\libroRGIS` como el espacio de trabajo, la ruta del archivo ensenada.csv que se encuentra en la carpeta datos_mx sería `datos_mx\ensenada.csv`. En Windows, se tiene que utilizar el símbolo de barra invertida / o doble barra \\ como por ejemplo `C:/User/Documentos/libroRGIS` o `C:\\User\\Documentos\\libroRGIS`.

En el código a continuación, definimos y verificamos la ruta del espacio de trabajo con las funciones ***setwd()*** y ***getwd()***. Cargamos el archivo de texto ensenada.csv gracias a la función ***read.csv()***. El producto de este proceso de importación se asigna a un objeto en formato tabular (Dataframe) que llamamos tab. La tabla muestra, para cada mes, la precipitación (P), el número de días de lluvia (dias), la temperatura mínima (Tmin), máxima (Tmax), la evaporación (Evapo) y el periodo de crecimiento de la uva (PCvid) en Ensenada, Baja California en 2012[2].

El comando `class(tab)` nos permite ver que este objeto es del tipo dataframe. Con `print(tab)` podemos visualizar esta tabla. Usando `head(tab)` se visualizan solamente las primeras filas de la tabla, lo cual resulta muy práctico cuando se manejan tablas muy grandes. De forma similar, la función ***tail()*** permite visualizar las últimas filas de la tabla.

```
# Determina la ruta del espacio de trabajo
setwd("/home/jf")  # Poner su propia ruta
# Muestra la ruta del espacio de trabajo
getwd()
```

[2] Datos climáticos diarios del CLICOM del SMN obtenidos a través de la plataforma web del CICESE (clicom-mex.cicese.mx)

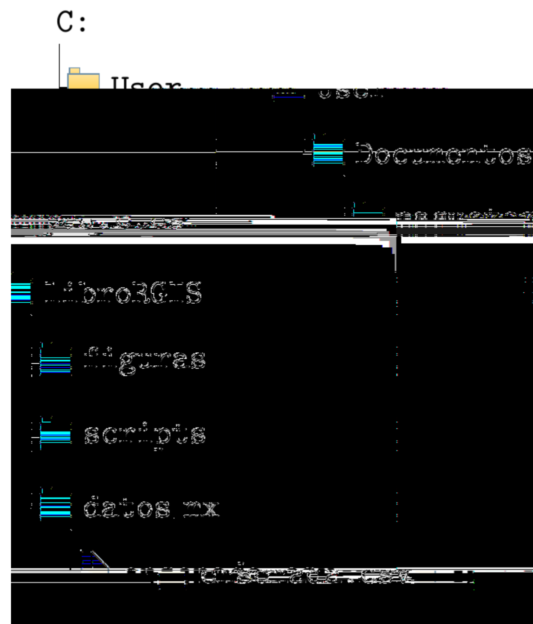


Figura 2. Organización jerárquica de carpetas y archivos

```
## [1] "/home/jf"

# Carga la tabla "ensenada.csv" en el objeto tab
tab <- read.csv("recursos-mx/ensenada.csv")

# Tipo del objeto tab
class(tab)

## [1] "data.frame"

# Muestra la tabla completa
print(tab)

##    mes    P dias  Tmin  Tmax Tprom  Evapo PCvid
## 1    1 14.9    4  7.03 21.21 14.12  86.97    no
## 2    2 40.2    7  7.32 19.57 13.45  84.00    no
## 3    3 36.9    4  7.84 20.00 13.92 107.92    no
## 4    4 36.9    9 10.02 20.45 15.23 131.97    si
## 5    5  0.0    0 12.31 21.89 17.10 170.66    si
## 6    6  0.0    0 14.33 22.70 18.52 182.39    si
## 7    7  0.0    0 16.24 24.76 20.50 182.23    si
## 8    8  0.1    1 18.61 26.85 22.73 175.90    si
## 9    9  0.4    1 18.22 28.18 23.20 136.50    si
## 10  10  7.4    2 13.79 26.34 20.06 128.42    si
## 11  11  3.1    2  9.60 24.02 16.81  87.55    no
## 12  12 77.1    8  7.84 19.60 13.72  73.68    no
```

```
# Muestra las primeras filas de la tabla
head(tab)

##   mes    P dias  Tmin  Tmax Tprom  Evapo PCvid
## 1   1 14.9    4  7.03 21.21 14.12  86.97    no
## 2   2 40.2    7  7.32 19.57 13.45  84.00    no
## 3   3 36.9    4  7.84 20.00 13.92 107.92    no
## 4   4 36.9    9 10.02 20.45 15.23 131.97    si
## 5   5  0.0    0 12.31 21.89 17.10 170.66    si
## 6   6  0.0    0 14.33 22.70 18.52 182.39    si

head(tab, 3)  # Muestra las 3 primeras filas de la tabla

##   mes    P dias Tmin  Tmax Tprom  Evapo PCvid
## 1   1 14.9    4 7.03 21.21 14.12  86.97    no
## 2   2 40.2    7 7.32 19.57 13.45  84.00    no
## 3   3 36.9    4 7.84 20.00 13.92 107.92    no

tail(tab, 2)  # Muestra las 2 últimas filas de la tabla

##    mes    P dias Tmin  Tmax Tprom Evapo PCvid
## 11  11  3.1    2 9.60 24.02 16.81 87.55    no
## 12  12 77.1    8 7.84 19.60 13.72 73.68    no
```

2.3 Operaciones con tablas

Los dataframes tienen encabezados de columnas, los cuales se pueden obtener o definir con el comando ***names()***. Se puede también manejar una sola columna de la tabla invocando el nombre del dataframe, seguido del operador $ y del nombre de la columna, por ejemplo tab$P. Para R esta columna es un simple vector como lo indica el resultado de la consulta `is.vector(tab$P)`.

```
# Muestra nombres (encabezados) de las columnas
names(tab)

## [1] "mes"   "P"     "dias"  "Tmin"  "Tmax"  "Tprom" "Evapo" "PCvid"

# Muestra una sola columna (P)
print(tab$P)

##  [1] 14.9 40.2 36.9 36.9  0.0  0.0  0.0  0.1  0.4  7.4  3.1 77.1

# Una columna de tabla es un vector
is.vector(tab$P)

## [1] TRUE
```

Es muy fácil realizar operaciones matemáticas entre elementos de diferentes columnas, un poco como en una hoja de cálculo. En el ejemplo que sigue, calculamos la amplitud térmica, restando la temperatura mímima de la

máxima para cada mes. Para ello, se crea una nueva columna en la tabla llamada rango. Finalmente, la función ***write.table()*** permite salvar la tabla dataframe.

```
  # Calculemos el rango de temperatura (T max - T min)
  # Nueva columna en la tabla: rango
  tab$rango <- tab$Tmax - tab$Tmin

# Muestra las primeras filas de la tabla
head(tab)

##   mes    P dias  Tmin  Tmax Tprom  Evapo PCvid rango
## 1   1 14.9    4  7.03 21.21 14.12  86.97    no 14.18
## 2   2 40.2    7  7.32 19.57 13.45  84.00    no 12.25
## 3   3 36.9    4  7.84 20.00 13.92 107.92    no 12.16
## 4   4 36.9    9 10.02 20.45 15.23 131.97    si 10.43
## 5   5  0.0    0 12.31 21.89 17.10 170.66    si  9.58
## 6   6  0.0    0 14.33 22.70 18.52 182.39    si  8.37
```

Observemos ahora los datos de la columna PCvid que indica si estamos dentro o fuera del periodo de crecimiento de la uva. Esta columna tiene solo dos tipos de respuesta ”si” o ”no” y R reconoció que se trataba de datos cualitativos con dos categorías e importó la información como nivel (factor) como lo indica class(tab$PCvid). Internamente, cada grupo o nivel está codificado de forma numérica.

```
  # factor
  tab$PCvid

##  [1] no no no si si si si si si si no no
## Levels: no si

class(tab$PCvid)

## [1] "factor"

as.numeric(tab$PCvid)

##  [1] 1 1 1 2 2 2 2 2 2 2 1 1
```

Existen numerosas operaciones que se pueden realizar con tablas. Una de ellas es seleccionar ciertos elementos de la tabla. Estas selecciones pueden realizarse utilizando la indexación de los datos con el operador `[]`. El nombre de la tabla, seguido del número de fila y de columna separado por una coma y entre corchetes permite acceder a una celda particular de la tabla. Por ejemplo `tab[1,2]` permite selecionar la celda situada en la primera fila y en la segunda columna, `tab[,c(1,2,6)]` permite selecionar las columnas 1, 2 y 6 de la tabla tab. El signo menos permite excluir ciertas filas o columnas de la selección como en la expresión `tab[-(10:12),-4]` que exluye las filas 10 a 12 y la cuarta columna. Para seleccionar columnas, se puede también usar los nombres en vez del número. Por ejemplo `tab[, c("mes","P","Tprom")]` es equivalente a la primera línea del código a continuación.

```
tab[1,2]

## [1] 14.9
```

```
tab[,c(1,2,6)] # selecciona las filas enteras

##    mes    P Tprom
## 1    1 14.9 14.12
## 2    2 40.2 13.45
## 3    3 36.9 13.92
## 4    4 36.9 15.23
## 5    5  0.0 17.10
## 6    6  0.0 18.52
## 7    7  0.0 20.50
## 8    8  0.1 22.73
## 9    9  0.4 23.20
## 10  10  7.4 20.06
## 11  11  3.1 16.81
## 12  12 77.1 13.72

tab[1:6, c(1:3,6)]

##   mes    P dias Tprom
## 1   1 14.9    4 14.12
## 2   2 40.2    7 13.45
## 3   3 36.9    4 13.92
## 4   4 36.9    9 15.23
## 5   5  0.0    0 17.10
## 6   6  0.0    0 18.52

tab[-(10:12),-4]

##   mes    P dias  Tmax Tprom  Evapo PCvid rango
## 1   1 14.9    4 21.21 14.12  86.97    no 14.18
## 2   2 40.2    7 19.57 13.45  84.00    no 12.25
## 3   3 36.9    4 20.00 13.92 107.92    no 12.16
## 4   4 36.9    9 20.45 15.23 131.97    si 10.43
## 5   5  0.0    0 21.89 17.10 170.66    si  9.58
## 6   6  0.0    0 22.70 18.52 182.39    si  8.37
## 7   7  0.0    0 24.76 20.50 182.23    si  8.52
## 8   8  0.1    1 26.85 22.73 175.90    si  8.24
## 9   9  0.4    1 28.18 23.20 136.50    si  9.96

tab[, c("mes","P","Tprom")] # equivalente a tab[,c(1,2,6)]

##    mes    P Tprom
## 1    1 14.9 14.12
## 2    2 40.2 13.45
## 3    3 36.9 13.92
## 4    4 36.9 15.23
## 5    5  0.0 17.10
## 6    6  0.0 18.52
## 7    7  0.0 20.50
```

```
## 8     8  0.1 22.73
## 9     9  0.4 23.20
## 10   10  7.4 20.06
## 11   11  3.1 16.81
## 12   12 77.1 13.72
```

La selección puede también basarse en condiciones, eventualmente usando el comando ***subset()*** como se muestra a continuación.

```
tab[tab$Tmax > 25,]

##    mes   P dias  Tmin  Tmax Tprom  Evapo PCvid rango
## 8    8 0.1    1 18.61 26.85 22.73 175.90    si  8.24
## 9    9 0.4    1 18.22 28.18 23.20 136.50    si  9.96
## 10  10 7.4    2 13.79 26.34 20.06 128.42    si 12.55

tab[tab$PCvid == "no",]

##    mes    P dias Tmin  Tmax Tprom  Evapo PCvid rango
## 1    1 14.9    4 7.03 21.21 14.12  86.97    no 14.18
## 2    2 40.2    7 7.32 19.57 13.45  84.00    no 12.25
## 3    3 36.9    4 7.84 20.00 13.92 107.92    no 12.16
## 11  11  3.1    2 9.60 24.02 16.81  87.55    no 14.42
## 12  12 77.1    8 7.84 19.60 13.72  73.68    no 11.76

subset(tab,Tmax > 25)

##    mes   P dias  Tmin  Tmax Tprom  Evapo PCvid rango
## 8    8 0.1    1 18.61 26.85 22.73 175.90    si  8.24
## 9    9 0.4    1 18.22 28.18 23.20 136.50    si  9.96
## 10  10 7.4    2 13.79 26.34 20.06 128.42    si 12.55

subset(tab,PCvid == "no")

##    mes    P dias Tmin  Tmax Tprom  Evapo PCvid rango
## 1    1 14.9    4 7.03 21.21 14.12  86.97    no 14.18
## 2    2 40.2    7 7.32 19.57 13.45  84.00    no 12.25
## 3    3 36.9    4 7.84 20.00 13.92 107.92    no 12.16
## 11  11  3.1    2 9.60 24.02 16.81  87.55    no 14.42
## 12  12 77.1    8 7.84 19.60 13.72  73.68    no 11.76

subset(tab,PCvid == "no",select= c("mes","P","Tprom"))

##    mes    P Tprom
## 1    1 14.9 14.12
## 2    2 40.2 13.45
## 3    3 36.9 13.92
## 11  11  3.1 16.81
## 12  12 77.1 13.72
```

Un paquete muy eficiente para el manejo de grandes tablas es `dplyr`. El comando ***filter()*** permite seleccionar filas (el primer argumento es el nombre de la tabla, el segundo la condición). El comando ***select()*** permite seleccionar columnas por su nombre (el primer argumento es el nombre de la tabla, el segundo los nombres de las columnas). Los dos comandos pueden anidarse para selecionar columnas y filas al mismo tiempo.

```
# install.packages("dplyr") # en caso que no esté instalado
library(dplyr)
filter(tab,PCvid == "no")

##   mes    P dias Tmin  Tmax Tprom  Evapo PCvid rango
## 1   1 14.9    4 7.03 21.21 14.12  86.97    no 14.18
## 2   2 40.2    7 7.32 19.57 13.45  84.00    no 12.25
## 3   3 36.9    4 7.84 20.00 13.92 107.92    no 12.16
## 4  11  3.1    2 9.60 24.02 16.81  87.55    no 14.42
## 5  12 77.1    8 7.84 19.60 13.72  73.68    no 11.76

select(tab,c(mes, Tmin:Tprom))

##    mes  Tmin  Tmax Tprom
## 1    1  7.03 21.21 14.12
## 2    2  7.32 19.57 13.45
## 3    3  7.84 20.00 13.92
## 4    4 10.02 20.45 15.23
## 5    5 12.31 21.89 17.10
## 6    6 14.33 22.70 18.52
## 7    7 16.24 24.76 20.50
## 8    8 18.61 26.85 22.73
## 9    9 18.22 28.18 23.20
## 10  10 13.79 26.34 20.06
## 11  11  9.60 24.02 16.81
## 12  12  7.84 19.60 13.72

select(filter(tab,PCvid == "no"),c(mes, Tmin:Tprom))

##   mes Tmin  Tmax Tprom
## 1   1 7.03 21.21 14.12
## 2   2 7.32 19.57 13.45
## 3   3 7.84 20.00 13.92
## 4  11 9.60 24.02 16.81
## 5  12 7.84 19.60 13.72
```

En cualquier momento, es posible guardar el dataframe en un archivo de texto usando, entre varias opciones, la función ***write.table()***.

```
# Guarda la tabla en un archivo de texto
write.table(tab, file="tabla.txt")
```

Finalmente, un aspecto importante es que es posible (no siempre!) convertir un objeto de un tipo a otro. Tal conversión, llamada ”coerción” en el argot de R, puede realizarse usando una función de la forma ”as.tipo” por

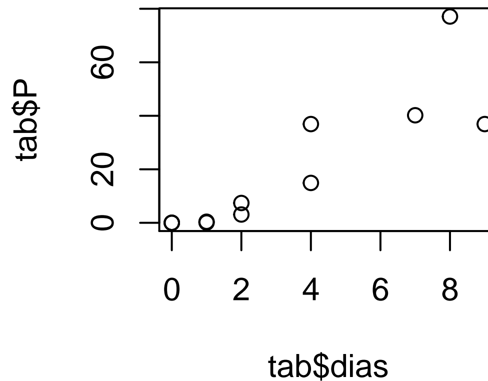
ejemplo ***as.matrix()*** permite convertir una tabla del tipo dataframe en matriz. Las matrices permiten también la indexación. En nuestro ejemplo `m[2,4]` permite acceder a la celda situada en la segunda fila y cuarta columna. La selección `m[1:3,4]` abarca los elementos de las filas 1 a 3 en la cuarta columna; `m[,4]` toda la cuarta columna; `m[1,]` la primera fila.

```
# Conversión de data.frame a matriz
m <- as.matrix(tab[,1:7]) # parte numérica de la tabla tab
class(m)

## [1] "matrix"

print(m)

##       mes    P dias  Tmin  Tmax Tprom  Evapo
##  [1,]   1 14.9    4  7.03 21.21 14.12  86.97
##  [2,]   2 40.2    7  7.32 19.57 13.45  84.00
##  [3,]   3 36.9    4  7.84 20.00 13.92 107.92
##  [4,]   4 36.9    9 10.02 20.45 15.23 131.97
##  [5,]   5  0.0    0 12.31 21.89 17.10 170.66
##  [6,]   6  0.0    0 14.33 22.70 18.52 182.39
##  [7,]   7  0.0    0 16.24 24.76 20.50 182.23
##  [8,]   8  0.1    1 18.61 26.85 22.73 175.90
##  [9,]   9  0.4    1 18.22 28.18 23.20 136.50
## [10,]  10  7.4    2 13.79 26.34 20.06 128.42
## [11,]  11  3.1    2  9.60 24.02 16.81  87.55
## [12,]  12 77.1    8  7.84 19.60 13.72  73.68

# indexación: 2a fila, 4a columna
print(tab[2,4])

## [1] 7.32

print(m[2,4])

## Tmin
## 7.32

print(m[1:3,4])

## [1] 7.03 7.32 7.84

print(m[,4])

##  [1]  7.03  7.32  7.84 10.02 12.31 14.33 16.24 18.61 18.22 13.79  9.60
## [12]  7.84

print(m[1,])

##   mes     P  dias  Tmin  Tmax Tprom Evapo
##  1.00 14.90  4.00  7.03 21.21 14.12 86.97
```

2.4 Elaboración de gráficas

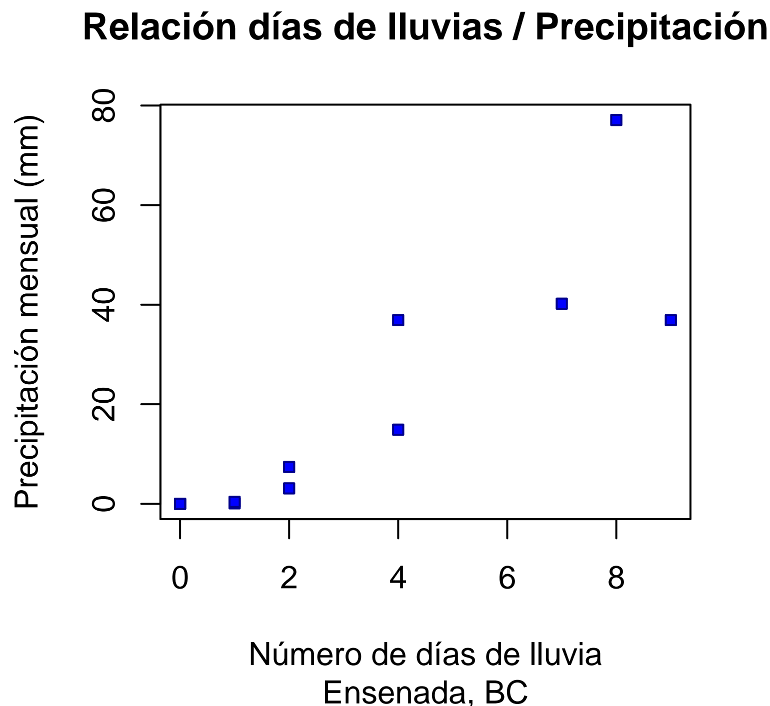
R es famoso por permitir la elaboración de gráficos muy elaborados. Sin embargo, nos conformaremos en elaborar dos diagramas de dispersión muy simples utilizando la función ***plot()***. Los dos argumentos de base son los nombres de dos vectores, el primero para el eje horizontal (x) y el segundo para el eje vertical (y), ambos vectores tienen que tener la misma longitud.

```
plot(tab$dias,tab$P)
```



Utilizando algunos argumentos opcionales adicionales, se puede muy fácilmente mejorar la gráfica: Con xlab=”texto eje x” y ylab=”texto eje y” se crean los títulos de ambos ejes; pch permite escoger el tipo de símbolo para los puntos, col su color; main y sub permiten de defimir un título principal y un subtítulo, cex permite controlar el tamaño de letra (cex = 0.8 reduce 20 % el tamaño de las letras en comparación con cex = 1, cex = 1.5 lo aumenta de 50 %).

```
plot(tab$dias,tab$P,  xlab="Número de días de lluvia", cex=0.8,
ylab="Precipitación mensual (mm)", pch=22, col="darkblue", bg="blue",
main="Relación días de lluvias / Precipitación", sub = "Ensenada, BC")
```



2.5 Relación entre dos variables

La gráfica nos sugiere que existe una relación positiva entre el número de días de lluvia y la cantidad de precipitación. Vamos a llevar a cabo algunos análisis estadísticos básicos, que nos permitirán entender mejor el manejo de los comandos, sus opciones y resultados. El comando ***cor()*** permite calcular el coeficiente de correlación entre dos variables. Sin embargo, sabemos que existen varias formas de calcular este coeficiente y no es obvio saber cual usa R *por default*. Se puede obtener ayuda sobre cualquier función con `help(nombre-del-comando)`. En la ayuda (Figura 3) podemos ver que ***cor()*** usa el método de Pearson si el usuario no especifica que método desea utilizar. Con el argumento method = ”nombre-método” se puede escoger entre tres métodos diferentes (Pearson, Kendall y Spearman). Adicionalmente, el comando `example("cor")` permite obtener algunos ejemplos de la utilización de la funcion ***cor()***.

cor {stats} R Documentation

Correlation, Variance and Covariance (Matrices)

Description

var, cov and cor compute the variance of x and the covariance or correlation of x and y if these are vectors. If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.

cov2cor scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

Figura 3. Inicio del documento de ayuda de la función ***cor()***

Se puede llevar a cabo una regresión lineal con la función ***lm()*** indicando la variable dependiente, y despúes del tilde la o las variables independientes o explicativas (fórmula). Un resumen de los resultados de los análisis de R puede obtenerse con ***summary()***. Por ejemplo `summary(reg)` nos proporciona información sobre el ajuste del modelo lineal llamado reg. Generalmente, los resultados se presentan de tal forma que es posible extraer un valor en particular a través de la indexación que vimos en el caso de las matrices. Por ejemplo resumen$coefficients es una matriz que contiene diferentes valores que resultan de la regresión (los coeficientes de regresión, el error, la significancia, etc.). Es posible extraer un elemento en particular en esta matriz. Por ejemplo, `resumen$coefficients[1,3]` nos da aqui el valor de t para la ordenada al origen (*intercept*).

```
# Correlación
cor(tab$dias,tab$P)

## [1] 0.8803083
```

```
# Para cualquier duda, pedir ayuda!
help(cor)
?cor()
cor(tab$dias,tab$P, method = "pearson")

## [1] 0.8803083

cor(tab$dias,tab$P, method = "spearman")

## [1] 0.9625061

# Una regresión lineal entre la prec y el número de días de lluvia
reg <- lm(tab$P ~ tab$dias)

# Los resultados del ajuste lineal
summary(reg)

##
## Call:
## lm(formula = tab$P ~ tab$dias)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -20.109  -4.444  -3.062   3.048  26.764
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3.048      5.043  -0.604 0.559099
## tab$dias       6.673      1.137   5.868 0.000158 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.23 on 10 degrees of freedom
## Multiple R-squared:  0.7749,Adjusted R-squared:  0.7524
## F-statistic: 34.43 on 1 and 10 DF,  p-value: 0.0001577

resumen <- summary(reg)

# Unas nuevas clases de objeto: lm (linear model) y summary.lm
class(reg)

## [1] "lm"

class(resumen)

## [1] "summary.lm"

# summary.lm guarda la información en una matriz llamada coeffcients
resumen$coefficients
```

```
##               Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) -3.047550   5.043041 -0.6043081 0.5590989828
## tab$dias     6.672911   1.137174  5.8679743 0.0001577273

# Recuperando un elemento particular de la matriz (t value del intercept)
resumen$coefficients[1,3]

## [1] -0.6043081

# Una lista (list) es una lista de objetos de diferentes tipos
lista <- list(Prec, reg, "lista rara")
lista

## [[1]]
##  [1] 15 40 37 37  0  0  0  0  0  7  3 77
##
## [[2]]
##
## Call:
## lm(formula = tab$P ~ tab$dias)
##
## Coefficients:
## (Intercept)     tab$dias
##      -3.048        6.673
##
##
## [[3]]
## [1] "lista rara"

# Muestra el primer y el tercer elemento de la lista
lista[[1]]

##  [1] 15 40 37 37  0  0  0  0  0  7  3 77

lista[[3]]

## [1] "lista rara"
```

2.6 Operaciones marginales: apply

Existen operaciones, llamadas operaciones marginales, que se efectúan para todas las columnas o todas las filas de una matriz. Por ejemplo, ***colSums()*** permite realizar la suma de las celdas de cada columna; ***rowSums()*** la suma de las celdas de cada fila mientras ***colMeans()*** realiza el cálculo del promedio de las celdas de cada columna.

```
# Crea una matriz de 3 x 3
# byrow=T: los números del vector entran por fila
m <- matrix(c(1,2,2,3,6,0,4,7,9),ncol=3,byrow=T)
print(m)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    2
## [2,]    3    6    0
## [3,]    4    7    9

colSums(m)

## [1]  8 15 11

rowSums(m)

## [1]  5  9 20

colMeans(m)

## [1] 2.666667 5.000000 3.666667
```

R provee la posibilidad de desarrollar el mismo tipo de operación marginal utilizando cualquier operador, mediante la función ***apply()***. Esta función tiene tres argumentos: el objeto sobre el cual se realiza el cálculo, un número que indica si la operación se realiza por fila (1) o por columna (2) y el operador. Por ejemplo, `apply(m,1,sum)` realiza la suma de las celdas de cada fila y `apply(m,2,sd)` calcula la desviación estándar de las celdas de cada columna.

```
apply(m,1,sum)

## [1]  5  9 20

apply(m,2,sum)

## [1]  8 15 11

apply(m,1,mean)

## [1] 1.666667 3.000000 6.666667

apply(m,1,max)

## [1] 2 6 9

apply(m,2,sd)

## [1] 1.527525 2.645751 4.725816
```

Existen varios comandos de la familia ”apply”. Por ejemplo ***lapply()*** permite aplicar operaciones marginales a listas.

2.7 Operaciones por grupos

Es a menudo necesario llevar a cabo el cálculo de algún índice estadístico por grupos de observaciones. Por ejemplo, en la tabla de datos climáticos de Ensenada, es interesante calcular la precipatación total y la temperatura

promedio durante los dos periodos de desarrollo de la vid. Este tipo de cálculo puede realizarse con la función ***aggregate()*** siguiendo diferentes sintaxis.

```
aggregate(P ~ PCvid, data = tab, FUN = "sum")

##   PCvid     P
## 1    no 172.2
## 2    si  44.8

aggregate(x = tab$P, by = list(tab$PCvid), FUN = "sum")

##   Group.1     x
## 1      no 172.2
## 2      si  44.8

aggregate(Tprom ~ PCvid, data = tab, FUN = "mean")

##   PCvid  Tprom
## 1    no 14.404
## 2    si 19.620
```

Otra opción es usar las funciones ***group_by()*** y ***summarise()*** del paquete `dplyr`, lo cual permite realizar la suma y el promedio juntos.

```
# con dplyr
por_estacion <- group_by(tab, PCvid)
summarise(por_estacion, Ptot = sum(P), Tprom_anual = mean(Tprom))

## # A tibble: 2 x 3
##   PCvid  Ptot Tprom_anual
##   <fct> <dbl>       <dbl>
## 1 no    172.         14.4
## 2 si     44.8        19.6
```

2.8 Creación de funciones

R permite también crear sus propias funciones empleando la directiva ”function”, que contiene la definición de los argumentos de la función, una o varias expresiones válidas del lenguaje y el resultado que arrojará esta función. Una vez definida una función, se puede llamarla tantas veces sea necesario durante una sesión de R.

```
# Definición de una función para sumar un valor numérico
# con el doble de un segundo valor
Func <- function (a, b) {
  resultado <- a + 2 * b
  resultado
}
# Ejecución de la función
Func(3,7)  # 3 + 2 * 7 = 3 + 14 = 17

## [1] 17
```

2.9 Repeticiones y condiciones

Finalmente, como cualquier lenguaje de programación, R permite realizar operaciones iterativas como bucles (*loops*). La función más sencilla para manejar bucles es ***for()*** que se usa con la sintaxis a continuación:
`for(i in lista-valores) {secuencia de comandos a ejecutar para cada valor de i}`
En el ejemplo número 1, i va tomar sucesivamente los valores 1, 2, 3 ... hasta 6 y la instrucción `print(i)` va imprimir cada uno de ellos.

```
for (i in 1:6){
  print(i)
}

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

En el segundo ejemplo, el valor del objeto fac se actualiza succesivmente multiplicando su valor en la iteración anterior por i que toma sucesivamente los valores 1, 2, 3 y 4. Por lo tanto el resultado final es $1 \times 2 \times 3 \times 4 = 24$.

```
fac <- 1
for (i in 1:4){
  fac <- fac * i
}
print(fac)

## [1] 24
```

En el ejemplo 3, el cálculo se hace sobre un vector e i sirve tanto para genera un valor como para escoger, por indexación, la posición del vector en la cual se va colocar el valor resultante. Por ejemplo, en la primera iteración (i = 1), la instrucción `vector[i] <- i*2` va colocar el valor 2 en el primero elemento del vector, etc.

```
  vector <- c(0,0,0,0)
  for (i in 1:4){
 vector[i] <- i*2
  }
  print(vector)

## [1] 2 4 6 8
```

En el cuarto ejemplo, i tomará valores de 1 a 4 ya que la longitud del vector es 4. Se anidó una condición dentro del bucle. La sintaxis de las condiciones es similar a la de las iteraciones:
`if(condición){secuencia de comandos cuando se cumple la condición} else {secuencia de comandos cuando no se cumple la condición}`

Por lo tanto, para cada iteración, se compara un elemento del vector con el valor cinco: si el valor es superior a cinco (if (vector[i] >5)), el valor de suma se actualiza recibiendo su valor en la iteración anterior más el valor del elemento del vector en la posición i (suma <- suma + vector[i]). En caso contrario, se le resta el valor uno (suma <- suma -1).

```
vector <- c(9,4,2,12,3,6)
suma <- 0
for (i in 1:length(vector)){
if (vector[i] > 5) {suma <- suma + vector[i]} else {suma <- suma -1}
print(suma)
}

## [1] 9
## [1] 8
## [1] 7
## [1] 19
## [1] 18
## [1] 24
```

En el último ejemplo, se crean, gracias a la función ***paste()***, nombres de archivos dinámicos. Eso permite cargar o salvar archivos que tienen nombres que siguen una estructura sistemática.

```
i <- 1
nombre <- paste("mapa",i,".txt",sep="")
print(nombre)

## [1] "mapa1.txt"

for (i in 1:4){
  nombre <- paste("mapa",i,".txt",sep="")
  print(nombre)
  }

## [1] "mapa1.txt"
## [1] "mapa2.txt"
## [1] "mapa3.txt"
## [1] "mapa4.txt"

# Nombres bandas imagen Sentinel 2
for (i in c("B02","B03","B8A","TCI")){
  nombre <- paste("T23KNT_20170701T131241_",i,".TIF",sep="")
  print(nombre)
  }

## [1] "T23KNT_20170701T131241_B02.TIF"
## [1] "T23KNT_20170701T131241_B03.TIF"
## [1] "T23KNT_20170701T131241_B8A.TIF"
## [1] "T23KNT_20170701T131241_TCI.TIF"
```

2.10 Operador pipe %>%

El operador pipe (%>%) permite encadenar funciones tomando la salida de una función y pasándola como entrada de la siguiente función. Ayuda enormemente a mejorar la legibilidad del código ya que anidar las funciones resulta rápidamente confuso:

```r
resultado <- funcion3(funcion2(funcion1(entrada)))
resultado <- entrada %>% funcion1() %>% funcion2() %>% funcion3()
```

Por ejemplo, si retomamos el ejemplo con funciones del paquete `dplyr` de la sección 2.7. Las dos líneas de código, que generan un objeto intermediario pueden resumirse en una sola línea:

```r
# Dos funciones que se aplican de forma succesiva:
# 2 líneas de código, un objeto intermediario (por_estacion)
por_estacion <- group_by(tab, PCvid)
summarise(por_estacion, Ptot = sum(P), Tprom_anual = mean(Tprom))

## # A tibble: 2 x 3
##   PCvid  Ptot Tprom_anual
##   <fct> <dbl>       <dbl>
## 1 no    172.         14.4
## 2 si     44.8        19.6

# Con pipe
group_by(tab, PCvid) %>% summarise(Ptot = sum(P), Tprom_anual = mean(Tprom))

## # A tibble: 2 x 3
##   PCvid  Ptot Tprom_anual
##   <fct> <dbl>       <dbl>
## 1 no    172.         14.4
## 2 si     44.8        19.6
```

2.11 Más sobre R

Existen varios manuales de introducción a R en Español disponibles en el repositorio CRAN (https://cran.r-project.org/other-docs.html). Consultar en particular Collatón Chicana (2014) y Santana y Farfán (Santana & Farfán, 2014). Existen también muchos libros que tratan de aspetos más especificos de R así como recursos en internet. Finalmente, existen varias listas de discusión en las cuales los usuarios se ayudan para resolver problemas encontrados al utilizar el programa. En la gran mayoría de los casos, los problemas que encontrarán ya han sido resueltos anteriormente: una simple búsqueda en Google permite encontrar la solución!

3. Organización de los objetos espaciales en R

En R, existen varias clases de objetos para representar información espacial. En este capítulo, presentamos, para datos vectoriales, el modelo geométrico de rasgo simple (*simple feature*), implementado en el paquete **sf** y, para raster, los obtejos del paquete **raster**. Se escogió el paquete **sf**, y no el paquete **sp** que es más difundido porque la estructura de los datos de **sf** es más simple y el paquete **sf** reemplazará paulatinamente a **sp**. Para datos raster, se presentan el formato *RasterLayer* del paquete **raster**. Sin embargo, debido a la importancia de **sp**, se encruentra en anexo una presentación de los modelos geométricos utilizados en este paquete.

3.1 Datos vectoriales: modelo *simple feature*

El modelo geométrico de rasgo simple (*simple feature*) es un estándar de código abierto desarrollado y respaldado por el *Open Geospatial Consortium* (OGC) para representar una amplia gama de información geográfica. Es un modelo de datos jerárquico que simplifica los datos geográficos al condensar un amplio rango de formas geográficas en una única clase de geometría. *Simple feature* es un modelo de datos ampliamente respaldado que subyace a las estructuras de datos en muchas aplicaciones de SIG, incluidas QGIS y PostGIS, pemitiendo la transferencia de datos a otras configuraciones.

El paquete **sf** (Pebesma, 2017) es totalmente compatible con los tipos de *simple feature* utilizados en la gran mayoría de las operaciones de análisis espacial: puntos, líneas, polígonos y sus respectivas versiones "múltiples". **sf** permite también crear colecciones de geometrías, que pueden contener diferentes tipos de geometría en un solo objeto. **sf** incorpora de forma integral la funcionalidad de los tres paquetes principales del paradigma **sp** (Pebesma & Bivand, 2018) para el sistema de clases de objetos espaciales, **rgdal** (Bivand *et al.*, 2017a) para leer y escribir datos y **rgeos** (Bivand & Rundel, 2017) para operaciones espaciales.

Lovelace et al. (2018) mencionan algunas razones para usar el paquete **sf** en vez de **sp**, aunque este último haya sido ampliamente probado:

- **sf** proporciona una interfaz casi directa para las funciones GDAL y GEOS C ++.
- La lectura y escritura de datos son rápidas.
- El rendimiento de despliegue fue mejorado.
- Los nombres de las funciones de **sf** son relativamente consistentes e intuitivos (todos comienzan con st_).
- Los objetos **sf** pueden procesarse de la misma forma que tablas *dataframe* en la mayoría de las operaciones.
- Las funciones de **sf** pueden combinarse usando el operador %>% y funcionan bien con la colección **tidyverse** de paquetes de R.

En R, los objetos *simple feature* se almacenan en una tabla *dataframe*, en la cual los datos geográficos ocupan una columna especial, que contiene una lista. Esta columna generalmente se llama "geom" o "geometry". Para instalar el paquete **sf**, véase sección 1.3). En Linux (Ubuntu), es necesario instalar previamente algunas librerías siguiendo los pasos a continuación:

```
sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable
sudo apt-get update
```

```
sudo apt-get install libgdal-dev libgeos-dev libproj-dev libudunits2-dev
sudo apt-get install liblwgeom-dev
```

En Windows, se recomienda instalar previamente **Rtools** siguiendo las instrucciones de la sección 1.3.

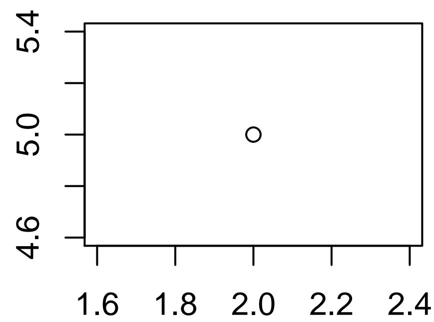
En **sf**, existen diferentes clases de objetos espaciales dependiendo del tipo de información. Las clases `Point` y `Multipoint`, `Linestring` y `Multilinestring` y las clases `Polygon` y `Multipolygon` permiten manejar respectivamente coberturas de puntos, líneas y polígonos. La clase `Geometrycollection` permite juntar, en un mismo objeto, diferentes geometrías.

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. En la práctica, los objetos espaciales que se manejan se obtienen a través de la importación de bases de datos, sin embargo, la construcción de objetos, desde la base, nos permitirá vislumbrar la jerarquía de los elementos que componen cada objeto. Vamos a crear algunos objetos espaciales 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. Se encuentra una descripción detallada de los objetos **sf** en la primera viñeta del paquete **sf** (https://cran.rstudio.com/web/packages/sf/vignettes/sf1.html). Empezemos por activar la librería **sf** con `library(sf)`.

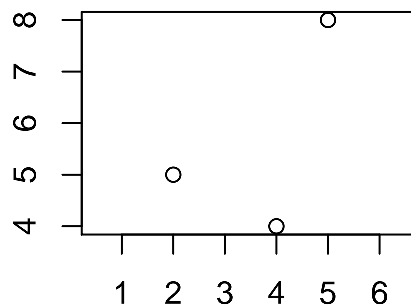
3.1.1 Cobertura de puntos

sf permite manejar datos en dos, tres o cuatro dimensiones. Típicamente, la tercera dimensión es la altura (z) y la cuarta (m) alguna medición (temperatura por ejemplo). Las coordenadas se presentan en forma de vector o de tabla y se transforman en una cobertura de puntos de la clase `sfg: simple feature geometry` con la función ***st_point()*** (un solo punto) o ***st_multipoint()*** (varios puntos).

```
library(sf) # install.packages("sf") si necesario instalar sf
# Geometrías Simple Features (sfg)
# Point: coordenadas de un punto en 2, 3 o 4 dimensiones
P <- st_point(c(2,5))  # 2 dimensiones (XY)
class(P)

## [1] "XY"    "POINT" "sfg"

P <- st_point(c(2,5,17,44),"XYZM")  # 4 dimensiones (XYZM)
class(P)

## [1] "XYZM"  "POINT" "sfg"

str(P)

## Classes 'XYZM', 'POINT', 'sfg'  num [1:4] 2 5 17 44

plot(P, axes = TRUE)
```




```
# Multipoint: coordenadas de varios puntos en 2, 3 o 4 dimensiones
# Crea un vector of coordenadas en x
Xs <- c(2,4,5)
# Crea un vector of 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 Multipoint (MP)
MP <- st_multipoint(coords)
plot(MP, axes = TRUE)
```




```
class(MP)

## [1] "XY"         "MULTIPOINT" "sfg"

print(MP)

## MULTIPOINT (2 5, 4 4, 5 8)

# Multipoint en 3 dimensiones
xyz <- cbind(coords,c(17, 22, 31))
print(xyz)
```

```
##       Xs Ys
## [1,]  2  5 17
## [2,]  4  4 22
## [3,]  5  8 31

MP <- st_multipoint(xyz)
print(MP)

## MULTIPOINT Z (2 5 17, 4 4 22, 5 8 31)
```

Se pueden juntar objetos en colecciones (sfc) con la función ***st_sfc()***. Por ejemplo, geometria1 es un objeto de la clase `simple feature collection (sfc)` que junta los puntos P1, P2 y P3. En este paso, es posible determinar el sistema de coordenadas de los datos con la opción `crs`. El sistema de coordenadas puede ser descrito por el código numérico EPSG o bien por el sistema de Proj4string. Por ejemplo, la proyección UTM comunmente utilizada en México para la zona 14 tiene el número EPSG 32614 y se describe de la forma siguiente en el sistema Proj4: +proj=utm +zone=14 +ellps=WGS84 +datum=WGS84 +units=m +no_defs. Se puede encontrar la equivalencia entre formatos en http://spatialreference.org/. La función ***make_EPSG()*** (paquete **rgdal**) permite crear una tabla con todos los códigos EPSG disponibles.

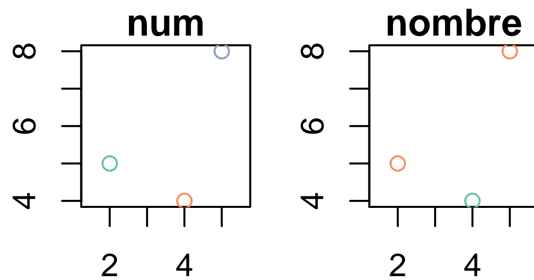
```
# Colecciones de Simple Features (sfc)
# Crea varios sfg
P1 <- st_point(c(2,5)); P2 <- st_point(c(4,4)); P3 <- st_point(c(5,8))

# Junta varios sfg en un sfc (colección de simple features)
geometria1 <- st_sfc(P1,P2,P3)
# st_sfc(P1,P2, crs = 4326) Proy geográfica LatLong datum WGS84
```

Los objetos `sfc` (geometrías) que acabamos de crear tienen solo la información de las coordenadas y eventualmente de los atributos z y m. No tiene aún tabla de atributos. 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 `simple feature`.

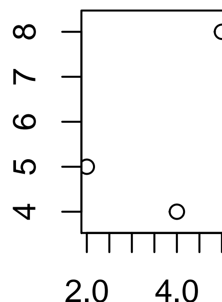
Diferentes funciones nos permiten conocer las características de este objeto. En particular, ***class()*** nos indica la naturaleza hídrida del objeto: `dataframe` y `sf`. ***print()*** nos permite observar la tabla de atributos con la ”columna-lista” en la cual se almacena la información espacial de cada punto. ***plot()*** despliega el mapa en pantalla: se crea un mapa para cada atributo de la tabla. Si se desea desplegar únicamente la geometría del objeto, en este caso la ubicación de los puntos, se usa la función ***st_geometry(SFP)***.

```
# Asociando una geometria sfc con una tabla de atributos (data frame)
# 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"

print(tabpuntos)
```

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

# sf object
SFP <- st_sf(tabpuntos, geometry = geometria1)
class(SFP) # doble clase: simple feature y dataframe

## [1] "sf"         "data.frame"

print(SFP) # ver columna lista "geometry"

## Simple feature collection with 3 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 2 ymin: 4 xmax: 5 ymax: 8
## epsg (SRID):    NA
## proj4string:    NA
##   num     nombre    geometry
## 1   1       Pozo POINT (2 5)
## 2   2 Gasolinera POINT (4 4)
## 3   3       Pozo POINT (5 8)

plot(SFP,axes=TRUE)
```




```
plot(st_geometry(SFP),axes=TRUE)
```



Se puede eventualmente extraer la tabla de atributos, perdiendo la información espacial, con la función ***as.data.frame()***. Es muy sencillo, extraer cierto rasgos de la cobertura usando la tabla de atributos. Por ejemplo, `Pozos <- SFP[nombre=="Pozo",]` crea un nuevo objeto `sf` con los puntos cuyo nombre es ”Pozo” en la tabla de atributos.

```
# Se puede extraer la tabla de atributos de un objeto SFC con
as.data.frame(SFP)

##   num     nombre geometry
## 1   1       Pozo     2, 5
## 2   2 Gasolinera     4, 4
## 3   3       Pozo     5, 8

# Selección de elementos dentro de la cobertura
Pozos <- SFP[nombre=="Pozo",]
```

3.1.2 Cobertura de líneas

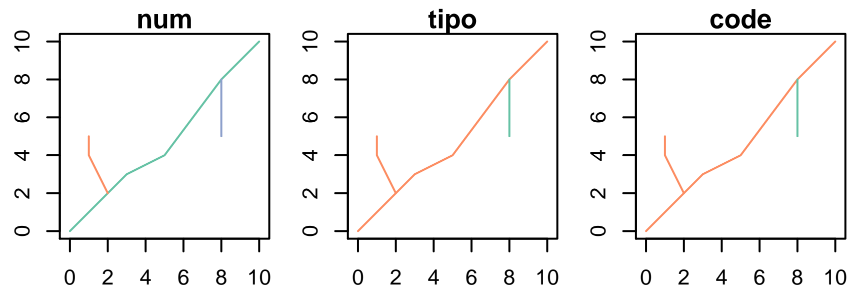
En formato vector, una línea simple está definida por las coordenadas de los vértices. Para el manejo de este tipo de datos, `sf` maneja objetos de la clase `Linestring` que describen segmentos simples (que no presentan intersecciones o bifurcaciones). Los objetos `Multilinestring` agrupan segmentos `Linestring`. En el código a continuación, creamos tres objetos `Linestring` (L1, L2 y L3) con base en una tabla de coordenadas usando la función ***st_linestring()***. De la misma forma que para los puntos, se pueden juntar objetos en colecciones (`sfc`) con la función ***st_sfc()***.

```
# Crea 3 objetos "Linestring": simple cadena de coordenadas (vértices)
# Línea L1
X1s <- c(0,3,5,8,10)
Y1s <- c(0,3,4,8,10)
Coord1 <- cbind(X1s,Y1s)
# Crea objeto de Clase Linestring
L1 <- st_linestring(Coord1)
class(L1)

## [1] "XY"         "LINESTRING" "sfg"

print(L1)

## LINESTRING (0 0, 3 3, 5 4, 8 8, 10 10)

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

# Línea 3
X3s <- c(8,8)
Y3s <- c(8,5)
Coord3 <- cbind(X3s,Y3s)
# Crea objeto de Clase Linestring
L3 <- st_linestring(Coord3)
```

```
# Crea un objeto Multilineas: conjunto de objetos Linestring
L1L2 <- st_multilinestring(list(L1,L2))

# Junta varios sfg en un sfc (colección de simple features)
geometria2 <- st_sfc(L1,L2,L3)
```

Para crear un objeto `sf`, se asocia una tabla de atributos a las líneas de una forma similar a la que seguimos para las coberturas de puntos.

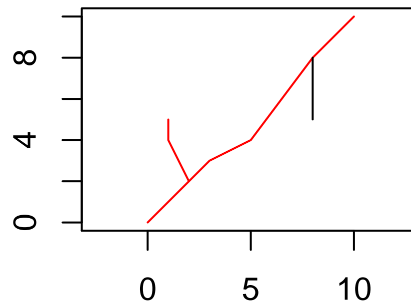
```
# Tabla de atributos
num <- c(1,2,3)
code <- c("t","t","p")
tipo <- c("Terraceria","Terraceria","Pavimentada")
tablineas <- data.frame(cbind(num,tipo,code))
print(tablineas)

##   num        tipo code
## 1   1  Terraceria    t
## 2   2  Terraceria    t
## 3   3 Pavimentada    p

# sf object
SFL <- st_sf(tablineas, geometry = geometria2)
plot(SFL,axes=TRUE)
```




```
class(SFL) # doble clase: simple feature y dataframe

## [1] "sf"         "data.frame"

print(SFL) # ver columna lista "geometry"

## Simple feature collection with 3 features and 3 fields
## geometry type:  LINESTRING
## dimension:      XY
## bbox:           xmin: 0 ymin: 0 xmax: 10 ymax: 10
## epsg (SRID):    NA
## proj4string:    NA
##   num        tipo code                       geometry
## 1   1  Terraceria    t LINESTRING (0 0, 3 3, 5 4, ...
## 2   2  Terraceria    t     LINESTRING (2 2, 1 4, 1 5)
## 3   3 Pavimentada    p          LINESTRING (8 8, 8 5)
```

Como para los puntos, la tabla de atributos puede usarse para selecionar líneas con ciertas características. Por ejemplo, en el ejemplo a continuación, se seleccionan y plotean las carreteras pavimentadas y de terracería usando diferentes colores.

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

##   num        tipo code                        geometry
## 1   1  Terraceria    t LINESTRING (0 0, 3 3, 5 4, ...
## 2   2  Terraceria    t     LINESTRING (2 2, 1 4, 1 5)
## 3   3 Pavimentada    p          LINESTRING (8 8, 8 5)

# Se puede selecionar ciertos rasgos usando la tabla de atributos
print(SFL[tipo=="Pavimentada",])

## Simple feature collection with 1 feature and 3 fields
## geometry type:  LINESTRING
## dimension:      XY
## bbox:           xmin: 8 ymin: 5 xmax: 8 ymax: 8
## epsg (SRID):    NA
## proj4string:    NA
##   num        tipo code              geometry
## 3   3 Pavimentada    p LINESTRING (8 8, 8 5)

plot(st_geometry(SFL[tipo=="Terraceria",]),col="red",axes=TRUE)
plot(st_geometry(SFL[tipo=="Pavimentada",]),add=TRUE)
```

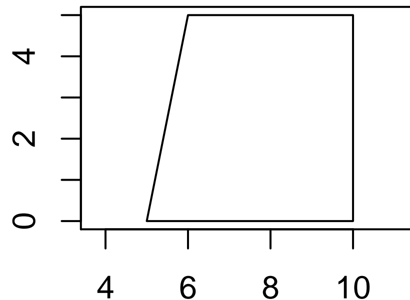


3.1.3 Cobertura de polígonos

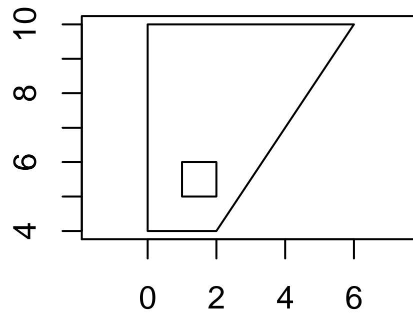
Para crear una cobertura de polígonos, se crean objetos `Polygon` a partir de las coordenadas de vértices que crean una forma cerrada utilizando la función ***st_polygon()***. La secuencia de los vértices es en sentido de las manecillas del reloj para delimitar "huecos" dentro de otro polígono y en sentido contrario para el contorno externo. Finalmente, se agrupan varios objetos en una colección utilizando la función ***st_sfc()***.

```
## P1 Polígono forestal al SudEste
## Polygon
# Crea una cadena de coordenadas en X
X1 <- c(5,10,10,6,5)
# Crea una cadena de coordenadas en Y
# Ojo tiene que cerrar (último par de coord = primero)
```

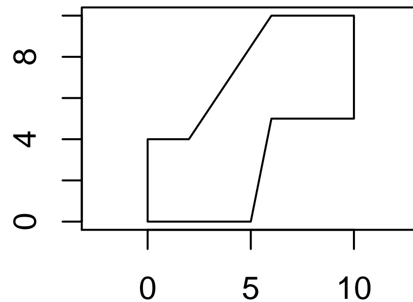
```
Y1 <-  c(0,0,5,5,0)
# Pega X y Y para crear una tabla de coordenadas
c1 <- cbind(X1,Y1)
print(c1)

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

class(c1)

## [1] "matrix"

# Crea el objeto Polygon. Un polygon es una forma simple cerrada
# eventualmente con hueco(s)
P1 <- st_polygon(list(c1))
plot(P1,axes=T)
```




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

# Polígono hueco  %%%%%%%%%%%%%% orden coordenadas !!!!!!!!!!!!!
# Crea una cadena de coordenadas en X
X3 <- c(1,1,2,2,1)
# Crea una cadena de coordenadas en Y
# La 1a y última coordenadas se repiten para "cerrar" el polígono
Y3 <- c(5,6,6,5,5)
# Pega X y Y para crear una tabla de coordenadas
c3 <- cbind(X3,Y3)
P2 <- st_polygon(list(c2,c3))
plot(P2,axes=T)
```



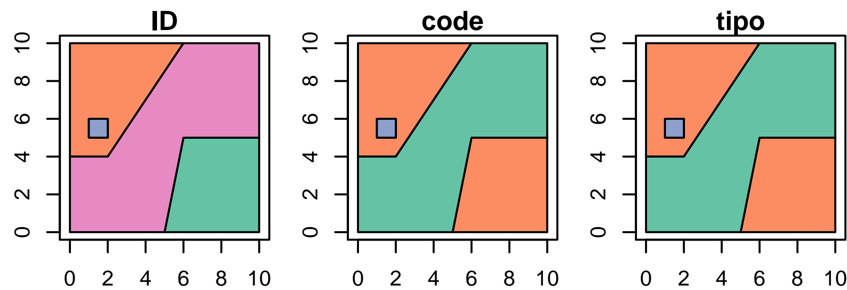

```
# P4 Polígono de agricultura
c3i <- c3[nrow(c3):1, ] # Invierte el orden de las coordenadas
P4 = st_polygon(list(c3i)) # Esta vez no es hueco

# P5 Polígono de área urbana
X5 <- c(0,5,6,10,10,6,2,0,0)
Y5 <- c(0,0,5,5,10,10,4,4,0)
c5 <- cbind(X5,Y5)
P5 <- st_polygon(list(c5))
plot(P5,axes=T)
```




```
# Junta varios sfg en un sfc (colección de simple features)
geometria3 <- st_sfc(P1,P2,P4,P5)
```

Se asocia esta cobertura de polígonos a una tabla de atributos para crear un objeto de la clase `sf` de forma similar a aquella utilizada para las coberturas de puntos y líneas.

```
# Tabla de atributos
ID <- c(1,2,3,4)
code <- c("F","F","U","A")
tipo <- c("Bosque","Bosque","Urbano","Agricultura")
tabpol <- data.frame(cbind(ID,code,tipo))

# sf object
SFPol <- st_sf(tabpol, geometry = geometria3)
plot(SFPol,axes=TRUE)
```

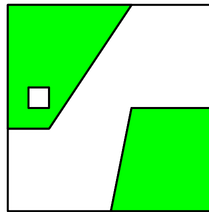



```
class(SFPol) # doble clase: simple feature y dataframe

## [1] "sf"         "data.frame"

print(SFPol) # ver columna lista "geometry"

## Simple feature collection with 4 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 0 ymin: 0 xmax: 10 ymax: 10
## epsg (SRID):    NA
## proj4string:    NA
##   ID code        tipo                       geometry
## 1  1    F      Bosque POLYGON ((5 0, 10 0, 10 5, ...
## 2  2    F      Bosque POLYGON ((0 4, 2 4, 6 10, 0...
## 3  3    U      Urbano POLYGON ((1 5, 2 5, 2 6, 1 ...
## 4  4    A Agricultura POLYGON ((0 0, 5 0, 6 5, 10...

summary(SFPol)

##   ID    code           tipo      geometry
##   1:1   A:1   Agricultura:1   POLYGON:4
##   2:1   F:2   Bosque     :2   epsg:NA:0
##   3:1   U:1   Urbano     :1
##   4:1
```

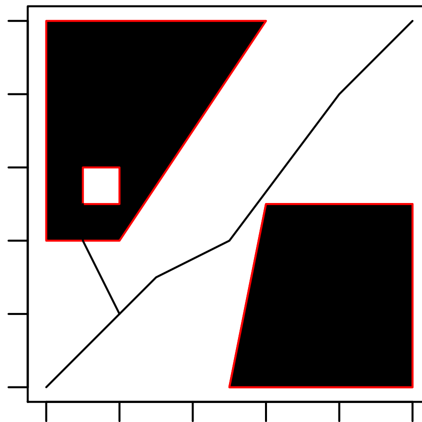
La tabla de atributos permite extraer ciertos rasgos. Las últimas líneas del código a continuación muestran que una colección puede eventualmente juntar diferentes tipos de geometría como puntos, lineas y polígonos.

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

##   ID code        tipo                       geometry
## 1  1    F      Bosque POLYGON ((5 0, 10 0, 10 5, ...
## 2  2    F      Bosque POLYGON ((0 4, 2 4, 6 10, 0...
## 3  3    U      Urbano POLYGON ((1 5, 2 5, 2 6, 1 ...
## 4  4    A Agricultura POLYGON ((0 0, 5 0, 6 5, 10...

# Se puede selecionar ciertos rasgos usando la tabla de atributos
print(SFPol[tipo=="Bosque",])
```

```
## Simple feature collection with 2 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 0 ymin: 0 xmax: 10 ymax: 10
## epsg (SRID):    NA
## proj4string:    NA
##   ID code   tipo                       geometry
## 1  1    F Bosque POLYGON ((5 0, 10 0, 10 5, ...
## 2  2    F Bosque POLYGON ((0 4, 2 4, 6 10, 0...

Bosque <- SFPol[tipo=="Bosque",]
plot(st_geometry(SFPol),axes=FALSE); plot(Bosque,add=TRUE, col = "green")
```



```
### Una colección puede eventualmente juntar diferentes tipos de geometría
Detodo <- st_sfc(P1,P2,L1,L2,P1)
plot(Detodo,border="red",axes=T)
```

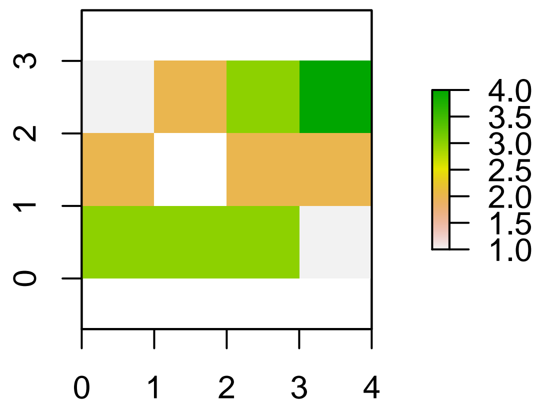


3.2 Datos raster: Clase *RasterLayer* en el paquete raster

El paquete **raster** permite manejar datos raster con el objeto de clase `RasterLayer`. El raster puede obtenerse a partir de una matriz que contiene los valores de cada celda. La extensión especial es definida por la función ***extent()*** introduciendo las coordenadas extremas del raster (x mínimo, x máximo, y mínimo, y máximo). Una descripción de los datos raster en **sp** se encuentra en los anexos digitales.

```
# install.packages("raster") # eliminar comentario para instalar paquete
library(raster)
# Creamos una matriz
m <- matrix(c(1,2,3,4,2,NA,2,2,3,3,3,1),ncol=4,nrow=3,byrow=TRUE)
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2   NA    2    2
## [3,]    3    3    3    1

r <- raster(m)
extent(r) <- extent(c(0,4,0,3))
class(r)

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

print(r)

## class       : RasterLayer
## dimensions  : 3, 4, 12  (nrow, ncol, ncell)
## resolution  : 1, 1  (x, y)
## extent      : 0, 4, 0, 3  (xmin, xmax, ymin, ymax)
## coord. ref. : NA
## data source : in memory
## names       : layer
## values      : 1, 4  (min, max)

plot(r)
```



4. Importación/exportación de datos espaciales

4.1 Importación de archivos *shape*

El formato *shapefile* es un formato de archivos para datos vectoriales desarrollado por la compañía ESRI y ampliamente utilizado. Está compuesto por lo menos por tres ficheros informáticos que tienen las extensiones siguientes:

- .shp: almacena las entidades geométricas de los objetos.
- .shx: almacena el índice de las entidades geométricas.
- .dbf: almacena la tabla de atributos de los objetos.

Es común encontrar un cuarto archivo con extensión .prj que contiene la información referida al sistema de coordenadas.

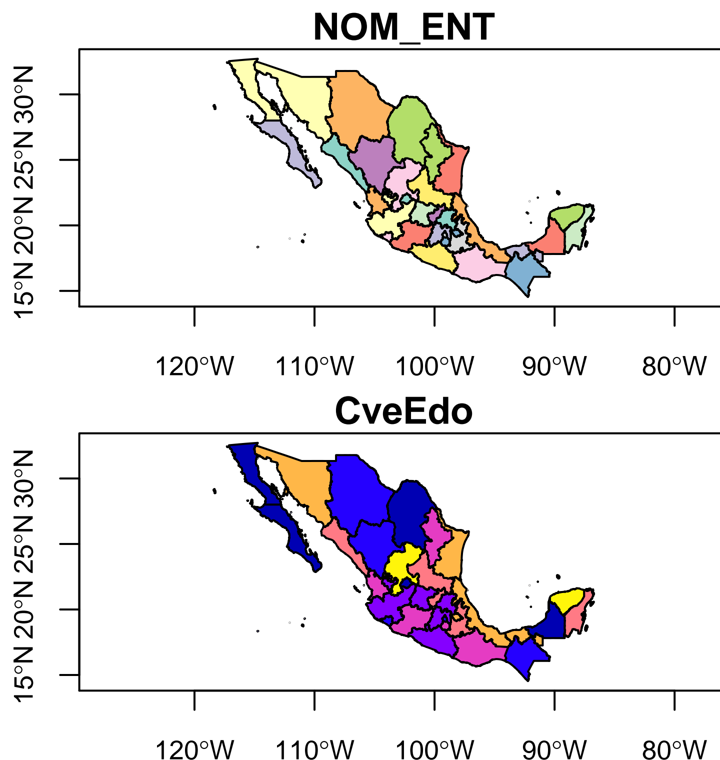
Existen varios paquetes para importar archivos *shape* en R. Presentamos aqui el procedimiento de importación con el paquete **sf**. Como se mencionó en el capítulo anterior, se escogió este paquete porque la estructura de los objetos espaciales es sencilla y está planeado como un reemplazo del paquete **sp**. El paquete **sf** permite manejar numerosas formas de representar información espacial, nos limitaremos a las formas más comunes presentadas en el capítulo anterior: geometrías de puntos, líneas y polígonos asociadas (o no) a una tabla de atributos (ver script cap3.R). En anexo se encuentra los métodos de importación y exportación a shape de los paquetes **rgdal** y **maptools** (objetos espaciales `SpatialPointsDataFrame`, `SpatialLinesDataFrame` y `SpatialPolygonsDataFrame` del paquete **sp**).

Para trabajar con el paquete **sf**, tenemos que cargar la librería con `library(sf)`. Para poder importar los archivos *shape* sin indicar la ruta de acceso, declaramos el espacio de trabajo como la carpeta en la cual se encuentran estos archivos. La función ***st_read()*** permite importar mapas en formato shape de cualquier geometría. ***class()*** nos permite comprobar que el objeto espacial creado es de la clase `sf`, ***summary()*** nos permite comprobar su estructura de *dataframe*.

Para los objetos de clase `sf`, ***plot()*** elabora un mapa para cada atributo. Si se desea mapear únicamente la geometría de los objetos (en este caso el límite de los polígonos), se usa la función ***st_geometry()***.

```
# Carga el paquete sf
library(sf)

## Linking to GEOS 3.5.1, GDAL 2.2.2, proj.4 4.9.2

# Determina la ruta del espacio de trabajo  CAMBIAR A SU CARPETA!!
# En windows sería algo tipo setwd("C:/Users/jf/Dropbox/libro_SIG/datos-mx")
setwd("/home/jf/recursos-mx")
mx <- st_read("Entidades_latlong.shp")

## Reading layer `Entidades_latlong' from data source
`/home/jf/recursos-mx/Entidades_latlong.shp' using driver `ESRI Shapefile'
## Simple feature collection with 32 features and 2 fields
```

```
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -118.407 ymin: 14.532 xmax: -86.7104 ymax: 32.7186
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs

# Pregunta la clase del objeto espacial
class(mx) # Es un simple feature "sf"

## [1] "sf"         "data.frame"

summary(mx) # un dataframe con una columna "lista" sobre la geometría

##                 NOM_ENT       CveEdo              geometry
##  Aguascalientes     : 1   Min.   : 1.00   MULTIPOLYGON :32
##  Baja California    : 1   1st Qu.: 8.75   epsg:4326    : 0
##  Baja California Sur: 1   Median :16.50   +proj=long...: 0
##  Campeche           : 1   Mean   :16.50
##  Chiapas            : 1   3rd Qu.:24.25
##  Chihuahua          : 1   Max.   :32.00
##  (Other)            :26

# plotea el mapa (um mapa para cada atributo)
plot(mx, axes = T,cex.axis=0.8) #cex.axis controla tamaño coordenadas
```



```
# Para plotear únicamente la geometría
plot(st_geometry(mx))
```

```
# Pregunta por el sistema de proyección
st_geometry(mx)  # equivalente a print(mx$geometry)

## Geometry set for 32 features
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -118.407 ymin: 14.532 xmax: -86.7104 ymax: 32.7186
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## First 5 geometries:

## MULTIPOLYGON (((-99.11124 19.5615, -99.11572 19...
## MULTIPOLYGON (((-99.9085 16.82495, -99.90839 16...
## MULTIPOLYGON (((-99.91237 20.28563, -99.91139 2...
## MULTIPOLYGON (((-99.06209 19.04872, -99.0598 19...
## MULTIPOLYGON (((-106.4129 23.17304, -106.4132 2...

st_crs(mx) # por código EPSG y proj4string

## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

Ciertos archivos no contienen la información del sistema de coordenadas, en este caso es importante determinarlo utilizando la función ***st_crs()***. La función ***st_is_valid()*** permite detectar errores de geometría como intersecciones.

```
carr <- st_read("carretera.shp")

## Reading layer `carretera' from data source
`/home/jf/recursos-mx/carretera.shp' using driver `ESRI Shapefile'
## Simple feature collection with 12424 features and 4 fields
## geometry type:  LINESTRING
## dimension:      XY
## bbox:           xmin: 1071375 ymin: 324561 xmax: 4076832 ymax: 2349296
## 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
```

```
st_crs(carr) # No hay información sobre el sistema de coordenadas

## Coordinate Reference System:
##   No EPSG code
##   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"

# se define:
st_crs(carr) <- "+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"

# Test de la validez de los mapas
st_is_valid(mx)

##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [14] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [27] TRUE TRUE TRUE TRUE TRUE TRUE

# st_is_valid(carr)
```

4.2 Importación de archivos vector de otros formatos

La función ***st_read()***, que utiliza GDAL, es capaz de importar una gran cantidad de formato vector incluyendo *ESRI File Geodatabase* (OpenFileGDB) entre otros. Para ver estos formatos, se puede utilizar ***sf_drivers*** que permite obtener una lista de los drivers soportados:

```
drivers_soportados <- st_drivers()
names(drivers_soportados)

## [1] "name"      "long_name" "write"     "copy"      "is_raster"
## [6] "is_vector"

# Lista de los 10 primeros drivers de la lista (hay más!)
head(drivers_soportados[,-2], n = 10)

##                           name write  copy is_raster is_vector
## PCIDSK                  PCIDSK  TRUE FALSE      TRUE      TRUE
## netCDF                  netCDF  TRUE  TRUE      TRUE      TRUE
## JP2OpenJPEG        JP2OpenJPEG FALSE  TRUE      TRUE      TRUE
## PDF                        PDF  TRUE  TRUE      TRUE      TRUE
## ESRI Shapefile ESRI Shapefile  TRUE FALSE     FALSE      TRUE
## MapInfo File      MapInfo File  TRUE FALSE     FALSE      TRUE
## UK .NTF                UK .NTF FALSE FALSE     FALSE      TRUE
## OGR_SDTS              OGR_SDTS FALSE FALSE     FALSE      TRUE
## S57                        S57  TRUE FALSE     FALSE      TRUE
## DGN                        DGN  TRUE FALSE     FALSE      TRUE
```

En general, ***st_read()*** determina cual driver debe utilizar con base en la extensión del archivo.

4.3 Exportación a *shape* o a otros formatos

Podemos salvar un objeto espacial de la clase `sf` en formato *shape* o en otro formato con la función ***st_write()***.

```
# Salva el objeto en formato shape
st_write(mx,"mexico.shp",delete_layer = TRUE)

## Deleting layer `mexico' using driver `ESRI Shapefile'
## Writing layer `mexico' to data source
 `/home/jf/recursos-mx/mexico.shp' using driver `ESRI Shapefile'
## features:       32
## fields:         2
## geometry type:  Multi Polygon

st_write(mx, dsn = "mx.gpkg", delete_layer = TRUE)

## Deleting layer `mx' using driver `GPKG'
## Updating layer `mx' to data source
 `/home/jf/recursos-mx/mx.gpkg' using driver `GPKG'
## features:       32
## fields:         2
## geometry type:  Multi Polygon
```

Dentro de R, es también posible transformar objetos espaciales entre formatos de varios paquetes. Esta opción puede ser particularmente útil para utilizar ciertos paquetes que solo manejan objetos de **sp**. En **sp** la función ***as()*** permite importar un objeto `sf` en clase `Spatial` de `sp`. Al inverso, ***st_as_sf()*** permite pasar de **sp** a **sf**.

```
library(sp)
mx_sp <- as(mx, Class = "Spatial")
# Regreso a sf con st_as_sf():
mx_sf <- st_as_sf(mx_sp, "sf")
```

4.4 Importación / exportación de datos raster

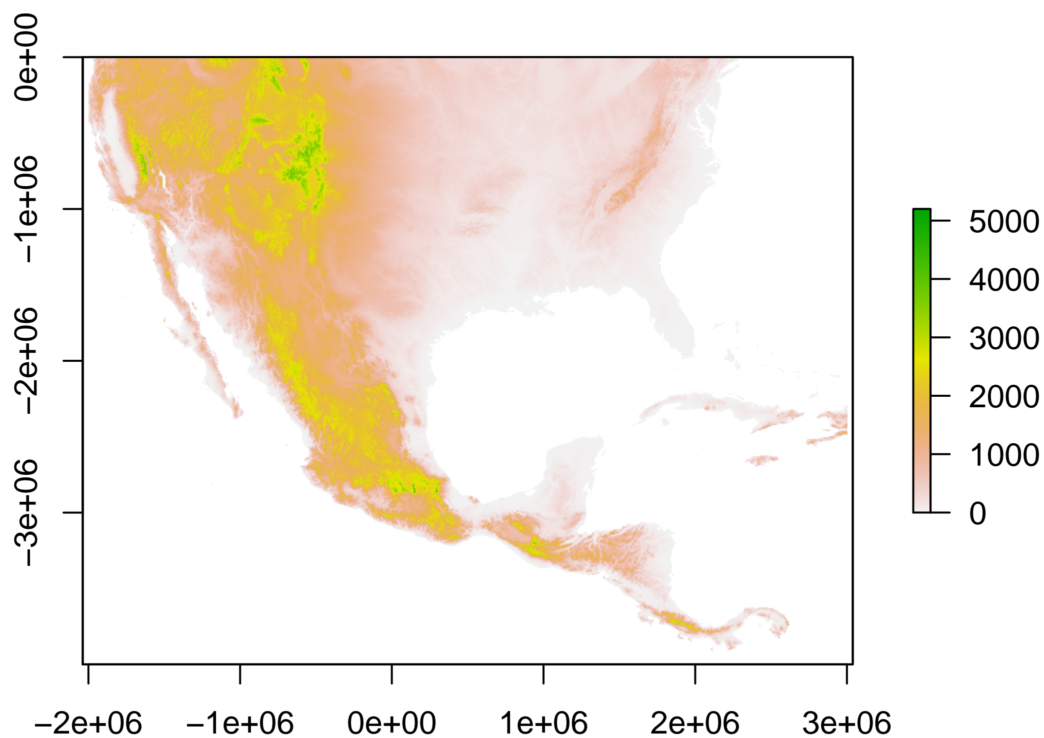
En el paquete **raster**, la función ***raster()*** permite importar imágenes de muchísimos formatos de imagen en objetos `RasterLayer`. Al contrario, la función ***writeRaster()*** permite salvar los objetos `RasterLayer` en varios formatos de imagen incluyendo ENVI, ESRI, ERDAS, GeoTiff, IDRISI y SAGA (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).

```
library(raster)

# Importa imagen
dem <- raster("DEM_GTOPO1KM.tif")
class(dem)

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

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

```
plot(dem)
```




```
extent(dem)

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

# Formatos disponibles para salvar
writeFormats()

##       name        long_name
```

```
##  [1,] "raster"    "R-raster"
##  [2,] "SAGA"      "SAGA GIS"
##  [3,] "IDRISI"    "IDRISI"
##  [4,] "IDRISIold" "IDRISI (img/doc)"
##  [5,] "BIL"       "Band by Line"
##  [6,] "BSQ"       "Band Sequential"
##  [7,] "BIP"       "Band by Pixel"
##  [8,] "ascii"     "Arc ASCII"
##  [9,] "CDF"       "NetCDF"
## [10,] "big"       "big.matrix"
## [11,] "ADRG"      "ARC Digitized Raster Graphics"
## [12,] "BMP"       "MS Windows Device Independent Bitmap"
## [13,] "BT"        "VTP .bt (Binary Terrain) 1.3 Format"
## [14,] "CTable2"   "CTable2 Datum Grid Shift"
## [15,] "EHdr"      "ESRI .hdr Labelled"
## [16,] "ELAS"      "ELAS"
## [17,] "ENVI"      "ENVI .hdr Labelled"
## [18,] "ERS"       "ERMapper .ers Labelled"
## [19,] "GS7BG"     "Golden Software 7 Binary Grid (.grd)"
## [20,] "GSBG"      "Golden Software Binary Grid (.grd)"
## [21,] "GTiff"     "GeoTIFF"
## [22,] "GTX"       "NOAA Vertical Datum .GTX"
## [23,] "HDF4Image" "HDF4 Dataset"
## [24,] "HFA"       "Erdas Imagine Images (.img)"
## [25,] "IDA"       "Image Data and Analysis"
## [26,] "ILWIS"     "ILWIS Raster Map"
## [27,] "INGR"      "Intergraph Raster"
## [28,] "ISIS2"     "USGS Astrogeology ISIS cube (Version 2)"
## [29,] "KRO"       "KOLOR Raw"
## [30,] "LAN"       "Erdas .LAN/.GIS"
## [31,] "Leveller"  "Leveller heightfield"
## [32,] "netCDF"    "Network Common Data Format"
## [33,] "NITF"      "National Imagery Transmission Format"
## [34,] "NTv2"      "NTv2 Datum Grid Shift"
## [35,] "PAux"      "PCI .aux Labelled"
## [36,] "PCIDSK"    "PCIDSK Database File"
## [37,] "PNM"       "Portable Pixmap Format (netpbm)"
## [38,] "RMF"       "Raster Matrix Format"
## [39,] "RST"       "Idrisi Raster A.1"
## [40,] "SAGA"      "SAGA GIS Binary Grid (.sdat)"
## [41,] "SGI"       "SGI Image File Format 1.0"
## [42,] "Terragen"  "Terragen heightfield"

# Salva el raster en formato LAN
writeRaster(dem, filename="DEM_Mx.lan", format="LAN", overwrite=TRUE,
            datatype="INT2S")
```

5. Operaciones básicas de SIG (*vector*)

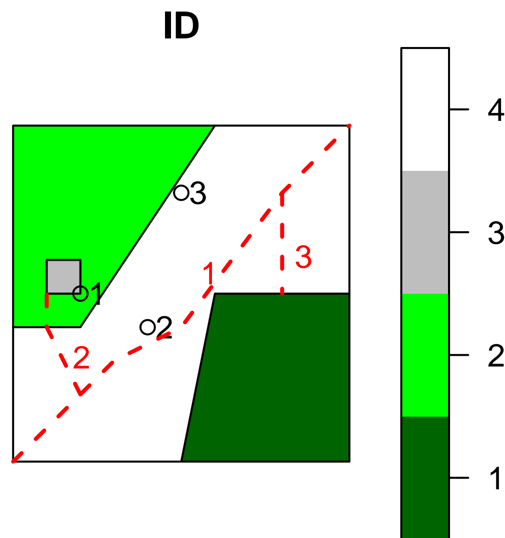
En este capítulo, vamos a revisar algunas funciones de análisis espacial de `sf` utilizando, en una primera sección, los mapas muy sencillos del capítulo 3 y en seguida datos más realistas.

5.1 Algunas operaciones de análisis espacial

Presentamos aquí algunas funciones de análisis espacial en `sf`, que son más fáciles de entender con datos minimalistas. Como se muestra a contiunación, algunas funciones de `sf` permiten verificar la geometría de objetos y calcular la longitud de líneas simples y el área de polígonos. La segunda parte del código, nos permite crear un mapa con los objetos espaciales que vamos a analizar. Los números indican el número de los puntos y las líneas.

```
# Determina la ruta del espacio de trabajo
setwd("/home/jf/recursos-mx")
library(sf)
# Importación de los mapitas con st_read (paquete sf)
# Mapa puntos
SFP <- st_read("SFP.shp")

## Reading layer `SFP' from data source
`/home/jf/recursos-mx/SFP.shp' using driver `ESRI Shapefile'
## Simple feature collection with 3 features and 2 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 2 ymin: 4 xmax: 5 ymax: 8
## epsg (SRID):    NA
## proj4string:    NA

# Mapa líneas
SFL <- st_read("SFL.shp")

## Reading layer `SFL' from data source
`/home/jf/recursos-mx/SFL.shp' using driver `ESRI Shapefile'
## Simple feature collection with 3 features and 3 fields
## geometry type:  LINESTRING
## dimension:      XY
## bbox:           xmin: 0 ymin: 0 xmax: 10 ymax: 10
## epsg (SRID):    NA
## proj4string:    NA

# Mapa polígonos
SFPol <- st_read("SFPol.shp")
```

```
## Reading layer `SFPol' from data source
`/home/jf/recursos-mx/SFPol.shp' using driver `ESRI Shapefile'
## Simple feature collection with 4 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 0 ymin: 0 xmax: 10 ymax: 10
## epsg (SRID):    NA
## proj4string:    NA

# Test de la validez de polígonos
st_is_valid(SFPol)

## [1] TRUE TRUE TRUE TRUE

# Test si líneas son simples
st_is_simple(SFL)

## [1] TRUE TRUE TRUE

# Calcula área de polígonos
st_area(SFPol)

## [1] 22.5 23.0  1.0 53.5

# Calcula longitud líneas (no funcion para multiline)
st_length(SFL)

## [1] 14.307136  3.236068  3.000000

# Mapita de los datos
plot(SFPol["ID"],col=c("Dark Green","green","grey","white"))
plot(st_geometry(SFP),add=TRUE)
plot(st_geometry(SFP) + c(0.5,0),add=TRUE,pch=c(49,50,51),cex=1)
plot(st_geometry(SFL),add=TRUE,lty=2,lwd = 2,col="red")
# agrega el número de las líneas
text(5,5.5,"1",pos=4,col="red",cex=1)
text(1.2,3,"2",pos=4,col="red",cex=1)
text(7.8,6,"3",pos=4,col="red",cex=1)
```



En un primer paso, vamos a utilizar la función ***st_intersects()***, que permite probar si los elementos de objetos espaciales se intersectan o no. La opción `sparse` permite manejar dos formas de presentación de los resultados: como una matriz (`sparse = FALSE`), en este case de 3 x 4 (3 líneas, 4 polígonos) o de una forma condensada (`sparse = TRUE`). ***st_distance()*** permite calcular la distancia euclidiana más corta entre elementos de dos objetos y presenta también los resultados en forma de matriz.

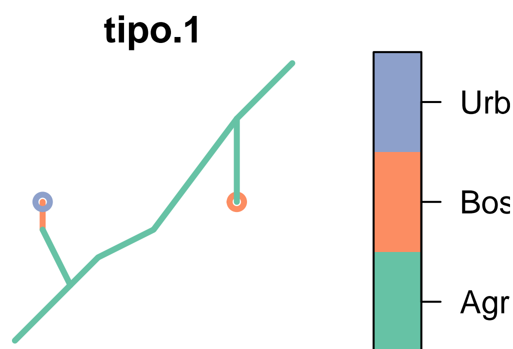
```
## Operadores lógicos binarios
# Intersección de puntos y polígonos
st_intersects(SFP, SFPol, sparse = FALSE)

##       [,1]  [,2]  [,3]  [,4]
## [1,] FALSE  TRUE  TRUE FALSE
## [2,] FALSE FALSE FALSE  TRUE
## [3,] FALSE FALSE FALSE  TRUE

st_intersects(SFP, SFPol, sparse = TRUE)

## Sparse geometry binary predicate list of length 3, where the predicate was
 `intersects'
##  1: 2, 3
##  2: 4
##  3: 4

# Intersección de líneas y polígonos
st_intersects(SFL, SFPol, sparse = FALSE)

##       [,1]  [,2]  [,3] [,4]
## [1,] FALSE FALSE FALSE TRUE
## [2,] FALSE  TRUE  TRUE TRUE
## [3,]  TRUE FALSE FALSE TRUE

st_intersects(SFL, SFPol, sparse = TRUE)
```

```
## Sparse geometry binary predicate list of length 3, where the predicate was
 `intersects'
##  1: 4
##  2: 2, 3, 4
##  3: 1, 4

# Distancia más corta entre elementos de dos objetos
st_distance(SFP, SFPol)

##          [,1]      [,2]     [,3]      [,4]
## [1,] 3.922323 0.0000000 0.000000 0.5547002
## [2,] 1.765045 1.6641006 2.236068 0.0000000
## [3,] 3.162278 0.2773501 3.605551 0.0000000
```

Finalmente, ***st_intersection()*** permite realizar una intersección geométrica entre dos objetos, obteniendo como resultado un objeto espacial cuyas geometría y tabla de atributos recoge la información de ambos objetos donde estos se sobrelapan. En el código a continuación, esta función permite calcular la longitud de los segmentos que coinciden espacialmente con las diferentes categorías de los polígonos.

```
# Operadores geométricos (intersección geométrica)
SFPxSFPol <- st_intersection(SFP,SFPol)
print(SFPxSFPol)  # punto 1 es duplicado por pertenecer a 2 polígonos

## Simple feature collection with 4 features and 5 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 2 ymin: 4 xmax: 5 ymax: 8
## epsg (SRID):    NA
## proj4string:    NA
##     num     nombre ID code        tipo    geometry
## 1     1       Pozo  2    F      Bosque POINT (2 5)
## 1.1   1       Pozo  3    U      Urbano POINT (2 5)
## 2     2 Gasolinera  4    A Agricultura POINT (4 4)
## 3     3       Pozo  4    A Agricultura POINT (5 8)

SFLxSFPol <- st_intersection(SFL,SFPol)
print(SFLxSFPol)  # hay puntos y  líneas

## Simple feature collection with 6 features and 6 fields
## geometry type:  GEOMETRY
## dimension:      XY
## bbox:           xmin: 0 ymin: 0 xmax: 10 ymax: 10
## epsg (SRID):    NA
## proj4string:    NA
##     num        tipo code ID code.1      tipo.1
## 3     3 Pavimentada    p  1      F      Bosque
## 2     2  Terraceria    t  2      F      Bosque
## 2.1   2  Terraceria    t  3      U      Urbano
## 1     1  Terraceria    t  4      A Agricultura
```

```
## 2.2   2  Terraceria    t  4      A Agricultura
## 3.1   3 Pavimentada    p  4      A Agricultura
##                            geometry
## 3                        POINT (8 5)
## 2              LINESTRING (1 4, 1 5)
## 2.1                      POINT (1 5)
## 1   LINESTRING (0 0, 3 3, 5 4, ...
## 2.2            LINESTRING (2 2, 1 4)
## 3.1            LINESTRING (8 8, 8 5)

plot(SFLxSFPol["tipo.1"],lwd=3)
```




```
# extraemos solo las líneas
SFLxSFPol_l <- st_collection_extract(SFLxSFPol,type="LINESTRING")
longitud <- st_length(SFLxSFPol_l) # Cálculo longitud
SFLxSFPol_l$long <- longitud # Resultados en nueva columna de la tabla
# Suma la longitud de los segmentos de cada tipo de cubierta
Suma_l <- aggregate(long ~ tipo.1, FUN = sum, data = SFLxSFPol_l)
print(Suma_l)

##        tipo.1    long
## 1 Agricultura 19.5432
## 2      Bosque  1.0000
```

Esta serie de operaciones puede condensarse utilizando el operador pipe (ver sección 2.10)

```
# Con pipe
longitud <-
  st_intersection(SFL,SFPol) %>% st_collection_extract(type="LINESTRING") %>% st_length()
print(longitud)

## [1]  1.000000 14.307136  2.236068  3.000000
```

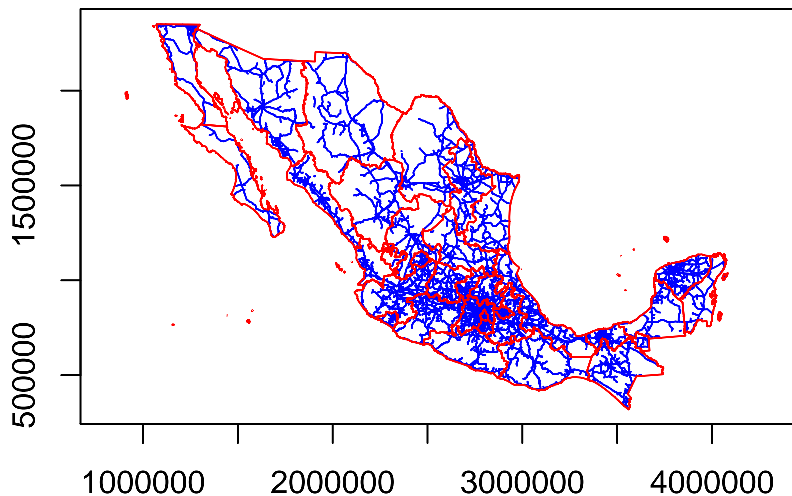
5.2 Análisis espacial en formato vector

En este sección, vamos a realizar un análisis basado en la intersección entre dos mapas. Vamos a importar dos mapas en formato shape: Un mapa de los Estados Mexicanos y otro de las principales carreteras. Vamos a visualizar

la tabla de atributos de estos mapas y reproyectar el mapa de los Estados que está en Lat/Long a Cónica Conforme de Lambert que es la proyección en la cual se encuentra el mapa de carreteras. Finalmente, calculamos el área de cada estado. Es importante notar que dividimos el área obtenida (em m^2) por un millón para obtener km^2 pero que el sistema indica m^2 como unidades.

```
# Carga los paquetes
library(sf)
# library(raster)

# Determina la ruta del espacio de trabajo
setwd("/home/jf/recursos-mx")

# Importación de los mapas con st_read (paquete sf)
# Mapa de los estados de México
mx <- st_read("Entidades_latlong.shp")

## Reading layer `Entidades_latlong' from data source
`/home/jf/recursos-mx/Entidades_latlong.shp' using driver `ESRI Shapefile'
## Simple feature collection with 32 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -118.407 ymin: 14.532 xmax: -86.7104 ymax: 32.7186
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs

st_crs(mx)

## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"

summary(mx)

##                  NOM_ENT       CveEdo               geometry
##  Aguascalientes     : 1   Min.   : 1.00   MULTIPOLYGON :32
##  Baja California    : 1   1st Qu.: 8.75   epsg:4326    : 0
##  Baja California Sur: 1   Median :16.50   +proj=long...: 0
##  Campeche           : 1   Mean   :16.50
##  Chiapas            : 1   3rd Qu.:24.25
##  Chihuahua          : 1   Max.   :32.00
##  (Other)            :26

# Mapa de carreteras
carr <- st_read("carretera.shp")

## Reading layer `carretera' from data source
`/home/jf/recursos-mx/carretera.shp' using driver `ESRI Shapefile'
## Simple feature collection with 12424 features and 4 fields
## geometry type:  LINESTRING
```

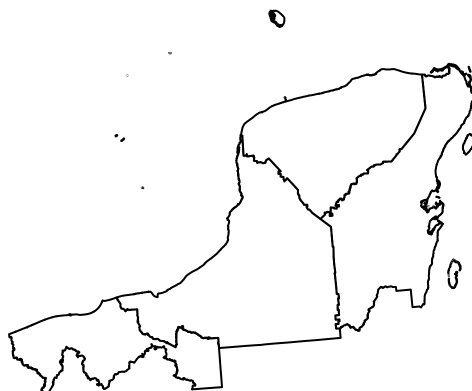
```
## dimension:      XY
## bbox:           xmin: 1071375 ymin: 324561 xmax: 4076832 ymax: 2349296
## epsg (SRID):    NA
## proj4string:    +proj=lcc +lat_1=17.5 +lat_2=29.5 +lat_0=12 +lon_0=-102
 +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs

st_crs(carr)

## Coordinate Reference System:
##   No EPSG code
##   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"

# Atributos del mapa de carretera
names(carr)

## [1] "ADMINISTRA" "ENTIDAD"    "NO_CARRILE" "SHAPE_len"  "geometry"

summary(carr)

##         ADMINISTRA        ENTIDAD                NO_CARRILE
##  Concesionada: 269   CARRETERA:12424   1 Carril        :  187
##  Desconocido :  52                     2 Carriles      :10767
##  Estatal     :6120                     4 Carriles      : 1326
##  Federal     :5453                     6 Carriles      :   58
##  N/A         :  81                     Mas de 6 carril:    5
##  Otro        : 449                     N/A             :   81
##    SHAPE_len                  geometry
##  Min.   :     2.23   LINESTRING   :12424
##  1st Qu.:  1346.58   epsg:NA      :    0
##  Median :  3780.05   +proj=lcc ...:    0
##  Mean   :  6843.31
##  3rd Qu.:  8942.17
##  Max.   :118873.89

# Reproyecta mx a LCC
mx_lcc <- st_transform(mx, st_crs(carr))
st_crs(mx_lcc)

## Coordinate Reference System:
##   No EPSG code
##   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"

# Calcula el área de los Estados
mx_lcc$AreaEdo <- st_area(mx_lcc) / 1000000  # km2
# Plotea los estados y carreteras juntos
plot(st_geometry(carr),col="blue",axes=TRUE)
plot(st_geometry(mx_lcc),border="red",add=TRUE)
```



A continuación, vamos a calcular la longitud de carreteras federales en cada Estado del sureste de México (Tabasco, Campeche, Quintana Roo y Yucatán). En **sf**, la selección de ciertos rasgos de un mapa se realiza con base en la tabla de atributos. Por ejemplo, `carr[ADMINISTRA="Federal",]` nos permite selecionar las líneas que corresponden a carreteras federales. Debido a que el mapa de carreteras "carr" es un objeto **sf**, el resutado de la selección es también un objeto espacial **sf** y no una simple tabla.

La función ***st_intersection()*** nos permite realizar la intersección entre los dos mapas. En este caso, el objeto obtenido son líneas (carreteras federales). La información del Estado en la cual se encuentran los segmentos de carretera está en la tabla de atributos. Se calcula la longitud de cada segmento, y se suman los valores de longitud para cada Estado.

```
# Qué longitud de carreteras federales en c/ estado del sureste?
# Selección carr federales
carrFed <- carr[ADMINISTRA="Federal",]
# Selección estados del sureste (Tabasco, Campeche, Quintana Roo y Yucatán)
sureste <- mx_lcc[mx_lcc$NOM_ENT %in% c("Tabasco", "Campeche",
                                        "Quintana Roo", "Yucatan"),]
class(sureste)

## [1] "sf"         "data.frame"

plot(st_geometry(sureste))
```



```r
# Intersección entre mapas de carr federales y Edos sureste
carrFedXedos <- st_intersection(carrFed,sureste)
names(carrFedXedos)

## [1] "ADMINISTRA" "ENTIDAD"    "NO_CARRILE" "SHAPE_len"  "NOM_ENT"
## [6] "CveEdo"     "AreaEdo"    "geometry"

summary(carrFedXedos)

##         ADMINISTRA       ENTIDAD               NO_CARRILE
##  Concesionada: 15   CARRETERA:1177   1 Carril       :166
##  Desconocido :  0                    2 Carriles     :933
##  Estatal     :749                    4 Carriles     : 75
##  Federal     :394                    6 Carriles     :  0
##  N/A         :  3                    Mas de 6 carril:  0
##  Otro        : 16                    N/A            :  3
##
##    SHAPE_len                   NOM_ENT        CveEdo
##  Min.   :     2.23   Yucatan        :467   Min.   : 4.00
##  1st Qu.:  1820.08   Tabasco        :384   1st Qu.:23.00
##  Median :  5673.34   Campeche       :192   Median :27.00
##  Mean   :  8697.05   Quintana Roo   :134   Mean   :24.38
##  3rd Qu.: 11550.34   Aguascalientes :  0   3rd Qu.:31.00
##  Max.   :118873.89   Baja California:  0   Max.   :31.00
##                      (Other)        :  0
##     AreaEdo                 geometry
##  Min.   :24695   LINESTRING     :1147
##  1st Qu.:24695   MULTILINESTRING:  30
##  Median :39533   epsg:NA        :   0
##  Mean   :38158   +proj=lcc ...  :   0
##  3rd Qu.:44556
##  Max.   :57277
##

# Convierte los objetos "Multilines" en "Lines" (para cálculo longitud)
carrFedXedos_l <- st_collection_extract(carrFedXedos,type="LINESTRING")
long <- st_length(carrFedXedos_l) / 1000  # km
head(long)

## Units: m
## [1]  1.219532  1.133280 19.162872 14.348592  1.500849  4.092854

# Agrega columna long a la tabla de atributos
carrFedXedos_l$long <- long
# Suma la longitud de los segmentos de cada tipo de cubierta
Suma_long <- aggregate(long ~ NOM_ENT, FUN = sum, data = carrFedXedos_l)
print(Suma_long)

##        NOM_ENT      long
```

```
## 1     Campeche 2060.536
## 2 Quintana Roo 1618.576
## 3      Tabasco 2382.986
## 4      Yucatan 3738.168
```

En seguida, se calcula la proporción del territorio de la región sureste que se encuentra a menos de 20 km de una carretera federal. Para ello, se elabora un área *buffer* de 20,000 m alrededor de las carreteras. Las etapas siguientes son similares al cálculo realizado para la longitud de carretera. Se puede observar que la mayoría de las operaciones son operaciones tabulares que se hacen de la misma forma que con cualquier tabla *dataframe*.

```
# Que proporción del territorio estatal está a menos de 20 km de una
# carretera federal?
# Creación de un buffer de 20 km alrededor carreteras fed
buf <- st_buffer(carrFed,dist=20000)
plot(st_geometry(buf),axes=F)
```

```
# Intersección entre buffer de carr federales y Edos sureste
carrbufXedos <- st_intersection(buf,sureste)
area <- st_area(carrbufXedos) / 1000000  # km2
head(area)  # Ojo: Pone m2 por default, en este caso son km2

## Units: m^2
## [1]   38.39414  274.58121  572.58037  680.35825 1197.92400 1329.92446

carrbufXedos$area <- area
# Suma la longitud de los segmentos de cada tipo de cubierta
Suma_area <- aggregate(area ~ NOM_ENT, FUN = sum, data = carrbufXedos)
print(Suma_area)

##        NOM_ENT     area
## 1     Campeche 276058.4
## 2 Quintana Roo 176800.9
## 3      Tabasco 506812.3
## 4      Yucatan 704789.8

# junta a sureste la tabla Suma_area (por clave en común)
sureste <- merge(sureste,Suma_area)
# calcula la proporción del área del buffer / área del estado
sureste$prop <- sureste$area / sureste$AreaEdo
```

En el código a continuación, se relaciona la tabla de abributo del mapa de los Estados con una tabla externa (población de los estados) y se calcula un índice basado en variables representadas por diferente columnas de la tabla. Finalmente, se exporta el objeto espacial obtenido a formato *shape*.

```
# Población Censo de Población y Vivienda 2010
tab_pop <- read.csv("Pop2010_INEGI.csv")
head(tab_pop)

##   NumEdo               Estado Poptotal  Hombre   Mujer
## 1      1       Aguascalientes  1184996  576638  608358
## 2      2      Baja California  3155070 1591610 1563460
## 3      3  Baja California Sur   637026  325433  311593
## 4      4             Campeche   822441  407721   41472
## 5      5 Coahuila de Zaragoza  2748391 1364197 1384194
## 6      6               Colima   650555   32279  327765

head(mx_lcc)

## Simple feature collection with 6 features and 3 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 907821.8 ymin: 485730.8 xmax: 2925387 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
##           NOM_ENT CveEdo       AreaEdo
## 1 Distrito Federal      9  1486.481 m^2
## 2         Guerrero     12 63565.113 m^2
## 3           Mexico     15 22226.643 m^2
## 4          Morelos     17  4859.432 m^2
## 5          Sinaloa     25 56802.974 m^2
## 6  Baja California      2 73535.910 m^2
##                       geometry
## 1 MULTIPOLYGON (((2802176 843...
## 2 MULTIPOLYGON (((2723198 539...
## 3 MULTIPOLYGON (((2717219 921...
## 4 MULTIPOLYGON (((2808476 786...
## 5 MULTIPOLYGON (((2050677 124...
## 6 MULTIPOLYGON (((1458026 185...

# Junta las dos tablas con la clave numérica
mx_lcc <- merge(mx_lcc,tab_pop,by.x="CveEdo",by.y="NumEdo")

head(mx_lcc)

## Simple feature collection with 6 features and 7 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 907821.8 ymin: 692158.7 xmax: 3859532 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
##   CveEdo              NOM_ENT        AreaEdo               Estado
## 1      1       Aguascalientes   5558.679 m^2       Aguascalientes
## 2      2      Baja California  73535.910 m^2      Baja California
## 3      3  Baja California Sur  73986.177 m^2  Baja California Sur
## 4      4             Campeche  57277.175 m^2             Campeche
## 5      5 Coahuila de Zaragoza 150670.960 m^2 Coahuila de Zaragoza
## 6      6               Colima   5754.971 m^2               Colima
##   Poptotal  Hombre   Mujer                       geometry
## 1  1184996  576638  608358 MULTIPOLYGON (((2470518 115...
## 2  3155070 1591610 1563460 MULTIPOLYGON (((1458026 185...
## 3   637026  325433  311593 MULTIPOLYGON (((1694646 122...
## 4   822441  407721   41472 MULTIPOLYGON (((3544897 946...
## 5  2748391 1364197 1384194 MULTIPOLYGON (((2469954 197...
## 6   650555   32279  327765 MULTIPOLYGON (((1158659 767...
```

```
names(mx_lcc)
```

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

```
# Densidad de población (núm hbs / km2)
mx_lcc$dens <- mx_lcc$Poptotal/mx_lcc$AreaEdo

# El índice de masculinidad, también llamado razón de sexo es un índice
# demográfico que expresa la razón de hombres por mujeres en un
# determinado territorio, expresado # por lo tanto en %. Se calcula
# usando la fórmula: 100 * H/M
mx_lcc$IM <- 100 * mx_lcc$Hombre/mx_lcc$Mujer

# salva el objeto en shape
st_write(mx_lcc,"mx_lcc.shp",delete_layer = TRUE)
```

```
## Deleting layer `mx_lcc' using driver `ESRI Shapefile'
## Writing layer `mx_lcc' to data source
 `/home/jf/libroRGIS/recursos-mx/mx_lcc.shp' using driver `ESRI Shapefile'
## features:       32
## fields:         9
## geometry type:  Multi Polygon
```

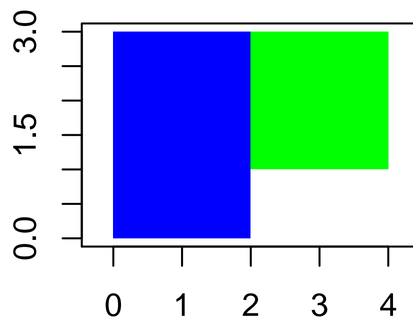
6. Operaciones básicas de SIG (*raster*)

6.1 Algunas operaciones de análisis espacial

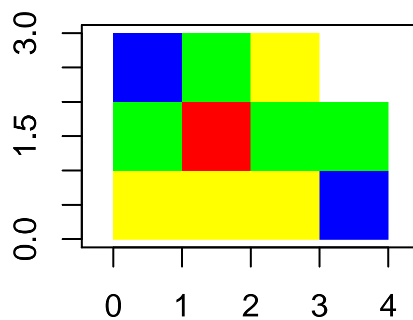
Presentamos aquí algunas funciones de análisis espacial del paquete `raster`, que son más fáciles de entender con datos ”minimalistas”. En la segunda sección del capítulo, llevamos a cabo algunas operaciones con mapas de uso y cubierta del suelo del Estado de Michoacán, México. Para una revisión más profunda del paquete `raster`, consultar Hijmans (2017).

Vamos a crear algunos datos *raster* de dimensión muy reducida (3 x 4 celdas) para mostrar fácilmente los resultados obtenidos, observando los mapas *raster* como matrices. Los datos *raster* puede eventualmente manejarse como imágenes multibanda con ***stack()*** o ***brick()***. Los comandos ***res()***, ***dim()*** y ***xmax()*** permiten obtener algunas de las características de los mapas como resolución (tamaño de las celdas), dimensión (número de filas, columnas y bandas) y coordenadas extremas. ***cellStats()*** permite calcular índices sobre el conjunto de la imagen (tomando en cuenta todas las celdas).

```
library(raster)
# Determina el espacio de trabajo
setwd("/home/jf/recursos-mx")
############## Datos muy sencillos
# Creamos unas matrices
m1 <- matrix(c(1,1,2,2,1,1,2,2,1,1,3,3),ncol=4,nrow=3,byrow=TRUE)
m2 <- matrix(c(1,2,3,4,2,NA,2,2,3,3,3,1),ncol=4,nrow=3,byrow=TRUE)
print(m1)

##      [,1] [,2] [,3] [,4]
## [1,]    1    1    2    2
## [2,]    1    1    2    2
## [3,]    1    1    3    3

print(m2)

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2   NA    2    2
## [3,]    3    3    3    1

r1 <- raster(m1); r2 <- raster(m2)
extent(r1) <- extent(r2) <- extent(c(0,4,0,3))
### stack
colortable(r1) <- c("red","blue","green")
colortable(r2)<- c("red","blue","green","yellow")
r12 <- stack(r1,r2)
plot(r1,axes=TRUE)
```




```
plot(r2,axes=TRUE)
```




```
# Características del raster
res(r1)

## [1] 1 1

dim(r12)

## [1] 3 4 2

xmax(r1) # existen también xmax, ymin y ymax

## [1] 4

cellStats(r1,"sum")  # suma de todos los pixeles

## [1] 20

cellStats(r1,"sd") # desv estándar de todos los pixeles

## [1] 0.7784989
```

El paquete `raster` permite realizar todas las operaciones de un SIG. Las operaciones de álgebra de mapa se aplican pixel por pixel. La función ***overlay()*** permite uilizar funciones definidas por el usuario.

```
# Algebra de mapa
sum1 <- r1 + 2; print(as.matrix(sum1))

##      [,1] [,2] [,3] [,4]
```

```
## [1,]    3    3    4    4
## [2,]    3    3    4    4
## [3,]    3    3    5    5

sum2 <- r1 + 2*r2; print(as.matrix(sum2))

##      [,1] [,2] [,3] [,4]
## [1,]    3    5    8   10
## [2,]    5   NA    6    6
## [3,]    7    7    9    5

sum3 <- overlay(r1, r2, sum1, fun=function(x, y, z){ x + 2*y -z} )
print(as.matrix(sum3))

##      [,1] [,2] [,3] [,4]
## [1,]    0    2    4    6
## [2,]    2   NA    2    2
## [3,]    4    4    4    0
```

Es posible ”pegar” entre ellos varios mapas para elaborar mosaicos con ***merge()***. Al contrario, ***crop()*** permite recortar una imagen con base en coordenadas extremas. Estas operaciones se basan en el *extent* de los mapas, que contiene las coordenadas extremas (esquinas inferior izquierda y superior derecha) en el orden xmin, xmax, ymin e ymax. La figura 4 ilustra las operaciones de elaboración y recorte de un mosaico realizadas en el código a continuación.

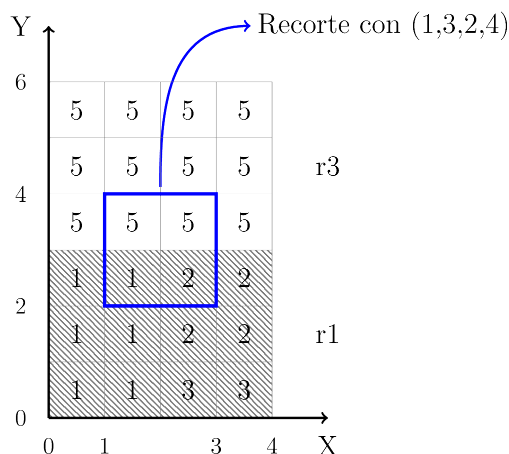


Figura 4. Mosaico y recorte de rasters

```
# Operaciones tipo SIG
# Mosaico (merge) y recorte (crop)
# Extent de r1 (xmin, xmin, ymin, ymax)
extent(r1)

## class       : Extent
## xmin        : 0
```

```
## xmax        : 4
## ymin        : 0
## ymax        : 3

m3 <- matrix(c(5,5,5,5,5,5,5,5,5,5,5,5),ncol=4,nrow=3,byrow=TRUE)
print(m3)

##      [,1] [,2] [,3] [,4]
## [1,]    5    5    5    5
## [2,]    5    5    5    5
## [3,]    5    5    5    5

r3 <- raster(m3)
# extent(xmin, xmax, ymin, ymax)
extent(r3) <- extent(c(0,4,3,6)) # Está al norte de r1
mosaico <- merge(r1,r3)
extent(mosaico)

## class       : Extent
## xmin        : 0
## xmax        : 4
## ymin        : 0
## ymax        : 6

print(as.matrix(mosaico))

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

extent_corte <- extent(c(1,3,2,4))
corte <- crop(mosaico,extent_corte)
print(as.matrix(corte))

##      [,1] [,2]
## [1,]    5    5
## [2,]    1    2
```

A continuación, presentamos algunas de las maneras de reclasificar los valores de un mapa disponibles en el paquete `raster`. Con `r2[r2 > 2] <- 6`, estamos asignando el valor 6 a las celdas de r2 que tienen un valor estrictamente superior a 2. Para reclasificaciones que involucran varios intervalos, es más fácil utilizar tablas de reclasificación. Estas tablas son matrices con tres columnas: las dos primeras indican los umbrales del intervalo de reclasificación y la tercera el valor de salida. El primer valor no está incluido en el intervalo, el segundo si. Por ejemplo, la primera fila de la tabla de reclasificación `0 2 0` indica que los valores estritamente superior a cero e inferior o igual a 2 se reclasifican en cero. La segunda fila `2 5 1` permite reclasificar los valores superiores a 2 hasta

5 (incluido) en el valor uno.

```
# Reclasificaciones
print(as.matrix(r2))

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2   NA    2    2
## [3,]    3    3    3    1

r2[r2 > 2] <- 6
print(as.matrix(r2))

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    6    6
## [2,]    2   NA    2    2
## [3,]    6    6    6    1

# Tabla de reclasificación (rangos)
m <- c(0, 2, 0, 2, 5, 1)
tabla_reclas <- matrix(m, ncol=3, byrow=TRUE)
print(tabla_reclas)

##      [,1] [,2] [,3]
## [1,]    0    2    0
## [2,]    2    5    1

reclas <- reclassify(mosaico, tabla_reclas)
print(as.matrix(reclas))

##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    1
## [2,]    1    1    1    1
## [3,]    1    1    1    1
## [4,]    0    0    0    0
## [5,]    0    0    0    0
## [6,]    0    0    1    1

# Substitución de valores
print(as.matrix(r2))

##      [,1] [,2] [,3] [,4]
## [1,]    1    2    6    6
## [2,]    2   NA    2    2
## [3,]    6    6    6    1

r2[r2 == 6] <- 9
print(as.matrix(r2))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    9    9
## [2,]    2   NA    2    2
## [3,]    9    9    9    1

# con tabla
tab_subs <- data.frame(id=c(1,2,3),v=c(1,56,84))
print(tab_subs)

##   id  v
## 1  1  1
## 2  2 56
## 3  3 84

sub <- subs(r1, tab_subs)
print(as.matrix(r1))

##      [,1] [,2] [,3] [,4]
## [1,]    1    1    2    2
## [2,]    1    1    2    2
## [3,]    1    1    3    3

print(as.matrix(sub))

##      [,1] [,2] [,3] [,4]
## [1,]    1    1   56   56
## [2,]    1    1   56   56
## [3,]    1    1   84   84
```

Se puede modificar el arreglo espacial de las celdas agrupando celdas. La función ***aggregate()*** permite reagrupar varios pixeles, obteniendo un mapa *raster* de menor resolución espacial. La función `fun` permite controlar el cálculo del valor del pixel de baja resolución que corresponde a varias celdas en los datos de entrada. En los dos ejemplos a continuación se reagrupan 2x2 pixeles. En la primera opción se suman los valores de las cuatro celdas, en la segunda se escoge el valor más común (mayoritario).

```
# Remuestreos
# Agregación espacial
agreg <- aggregate(mosaico, fact=2, fun="sum")
print(as.matrix(mosaico));print(as.matrix(agreg))

##      [,1] [,2] [,3] [,4]
## [1,]    5    5    5    5
## [2,]    5    5    5    5
## [3,]    5    5    5    5
## [4,]    1    1    2    2
## [5,]    1    1    2    2
## [6,]    1    1    3    3
##      [,1] [,2]
## [1,]   20   20
```

```
## [2,]   12   14
## [3,]    4   10

agreg2 <- aggregate(mosaico, fact=2, fun=modal,na.rm = TRUE)
print(as.matrix(agreg2))

##      [,1] [,2]
## [1,]    5    5
## [2,]    1    2
## [3,]    1    2
```

El paquete `raster` permite también realizar remuestreo con los métodos comúnmente utilizados en procesamiento de imágenes y en percepción remota.

```
# Remuestreo
# Imagen de mismo tamaño que mosaico con menos filas
m4 <- matrix(rep(1,20),ncol=4,nrow=5)
r4 <- raster(m4)
extent(r4) <- extent(mosaico)
resv <- resample(mosaico,r4,method="ngb") # vecino más cercano
resb <- resample(mosaico,r4,method="bilinear") # bilinear
print(as.matrix(resv)); print(as.matrix(resb))

##      [,1] [,2] [,3] [,4]
## [1,]    5    5    5    5
## [2,]    5    5    5    5
## [3,]    1    1    2    2
## [4,]    1    1    2    2
## [5,]    1    1    3    3
##      [,1] [,2] [,3] [,4]
## [1,]    5    5  5.0  5.0
## [2,]    5    5  5.0  5.0
## [3,]    3    3  3.5  3.5
## [4,]    1    1  2.0  2.0
## [5,]    1    1  2.9  2.9
```

El paquete `raster` permite realizar operaciones de filtrado espacial conocido como convolución (kernel). Las dos operaciones focales a continuación son equivalentes y calculan el promedio del valores de las celdas en una ventana móvil de 3 x 3 pixeles. El cálculo se realiza únicamente para las celdas que tienen ocho vecinos. Sin embargo, la opción "pad = TRUE" permite evitar este efecto en los bordes de la imagen (ver la ayuda).

```
# Operaciones focales
help(focal)
promedio <- focal(r1, w=matrix(1/9, ncol=3, nrow=3))
promedio <- focal(r1, w=matrix(1,3,3), fun="mean") # otra forma
print(as.matrix(r1))

##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    1    2    2
## [2,]    1    1    2    2
## [3,]    1    1    3    3

print(as.matrix(promedio))

##      [,1]     [,2]     [,3] [,4]
## [1,]   NA       NA       NA   NA
## [2,]   NA 1.444444 1.888889   NA
## [3,]   NA       NA       NA   NA
```

El paquete `raster` permite también llevar a cabo operaciones zonales que consiste en calcular algún indice sobre un mapa estratificando el cálculo con base en otro mapa. En el ejemplo a continuación, se suman el valor de las celdas del mapa r2 para cada categoría (valor) del mapa r1 (Figura 5). El resultado obtenido es una tabla.

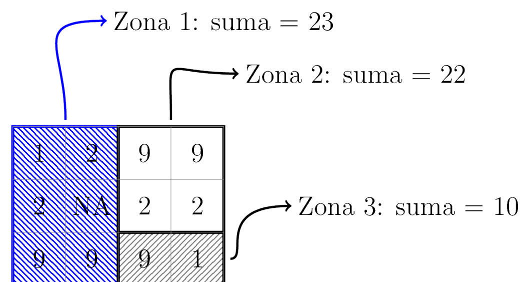


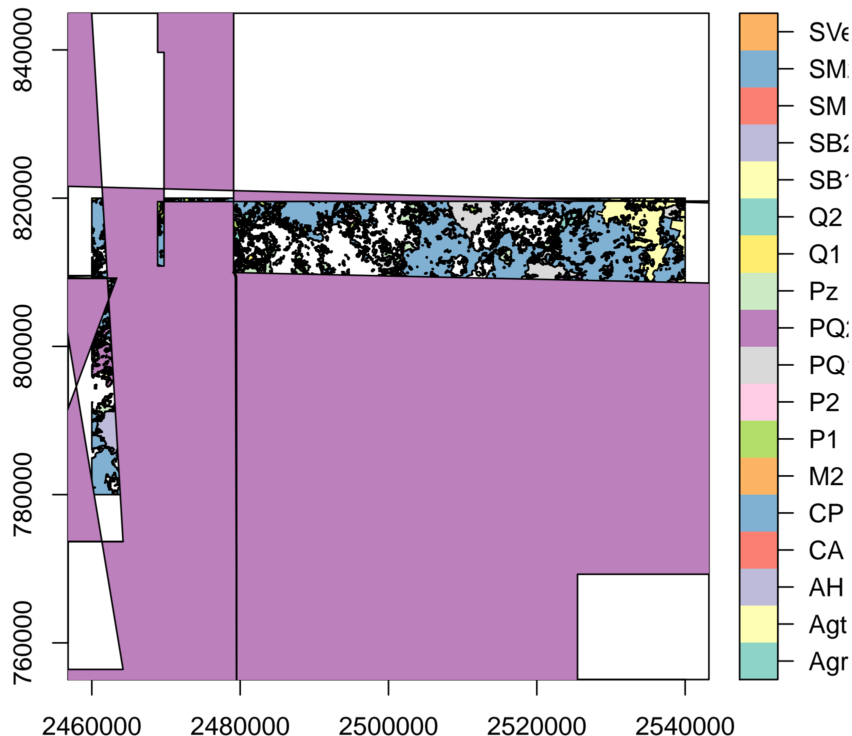
Figura 5. Suma zonal basada en el mapa r1 con 3 categorías

```
# Operaciones zonales
print(as.matrix(r1)); print(as.matrix(r2))

##      [,1] [,2] [,3] [,4]
## [1,]    1    1    2    2
## [2,]    1    1    2    2
## [3,]    1    1    3    3
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    9    9
## [2,]    2   NA    2    2
## [3,]    9    9    9    1

zonal_sum <- zonal(r2,r1,"sum") # map de valores, mapa de zonas
print(zonal_sum)

##      zone sum
## [1,]    1  23
## [2,]    2  22
## [3,]    3  10
```

6.2 Análisis espacial en formato *raster*

En esta sección, vamos a aplicar algunos de los métodos anteriores para elaborar un mapa de áreas deforestadas basados en dos mapas de cubierta/uso del suelo de una región de Michoacán para 2004 y 2014 (Mas *et al.*, 2017) [1]. Para ello vamos a importar y rasterizar los mapas que se encuentran en formato *shape* utilizando el campo GRIDCODE en la tabla de atributos del mapa en formato vector y una resolución espacial de 100 m. La importación se realizó con la función ***st_read()*** de `sf` y por lo tanto se tuvo que convertir el objeto espacial `sf` a `sp` porque ***rasterize()*** solo acepta objetos de `sp`. En este caso, dependiendo de su computador, la rasterización puede tardar unos minutos. El proceso puede optimizarse utilizando la función ***gdal_rasterize*** del paquete `gdalUtils`.

```
# Importación de un shape con sf
library(sf)
cs2004 <- st_read("cs2004.shp")

## Reading layer `cs2004' from data source
`/home/jf/recursos-mx/cs2004.shp' using driver `ESRI Shapefile'
## Simple feature collection with 4301 features and 5 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 2460000 ymin: 780000 xmax: 2540000 ymax: 820000
## 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 +datum=WGS84 +units=m +no_defs

plot(cs2004["Cl_abrev"], axes = T,cex.lab=0.8)
```

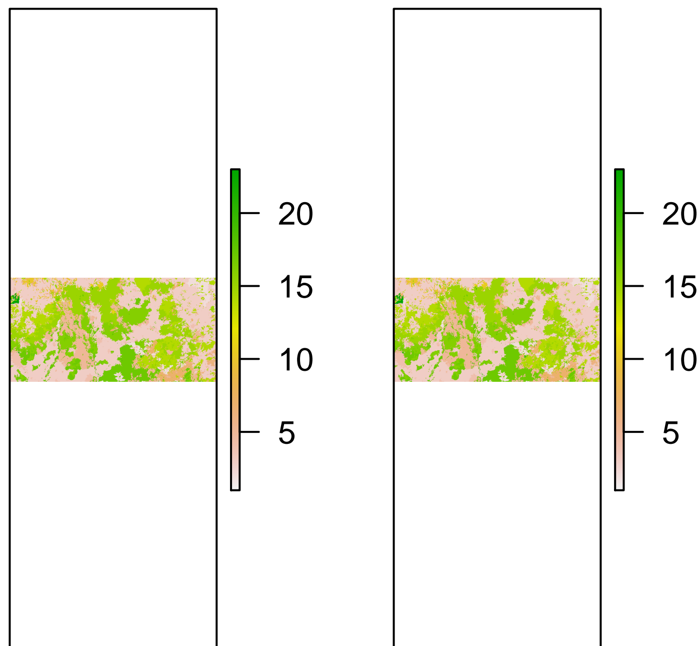


[1] Mapas elaborados en el ámbito del proyecto *Monitoreo de la cubierta del suelo y la deforestación en el Estado de Michoacán: un análisis de cambios mediante sensores remotos a escala regional* - Fondos FOMIX Clave MICH-2012-C03-192429

```
head(cs2004)

## Simple feature collection with 6 features and 5 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 2463605 ymin: 780000 xmax: 2525585 ymax: 816174.4
## 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 +datum=WGS84 +units=m +no_defs
##   OBJECTID GRIDCODE                  clase       area Cl_abrev
## 1      396        1    Agricultura de riego 167.408719      Agr
## 2      397        1    Agricultura de riego  27.766582      Agr
## 3      398        1    Agricultura de riego  83.592112      Agr
## 4      412        1    Agricultura de riego   2.912169      Agr
## 5      418        1    Agricultura de riego   4.006450      Agr
## 6     8596        2 Agricultura de temporal   1.966835      Agt
##                        geometry
## 1 MULTIPOLYGON (((2525547 780...
## 2 MULTIPOLYGON (((2497969 780...
## 3 MULTIPOLYGON (((2495969 780...
## 4 MULTIPOLYGON (((2463803 791...
## 5 MULTIPOLYGON (((2505625 793...
## 6 MULTIPOLYGON (((2523434 816...

st_bbox(cs2004)

##    xmin    ymin    xmax    ymax
## 2460000  780000 2540000  820000

st_crs(cs2004)

## Coordinate Reference System:
##   No EPSG code
##   proj4string: "+proj=lcc +lat_1=17.5 +lat_2=29.5 +lat_0=12 +lon_0=-102
 +x_0=2500000 +y_0=0 +datum=WGS84 +units=m +no_defs"
```

```
## Rasteriza con paquete raster
resol <- 100
# Para rasterizar usando un raster "master"
extension <- extent(c(2460000, 2540000,780000,  820000))
r <- raster(res=resol, ext=extension)
# Rasteriza con el campo GRIDCODE (as() para convertir a sp)
cs2004r <- rasterize(as(cs2004,Class = "Spatial"), r, "GRIDCODE")
plot(cs2004r,axes=F)

## Lo mismo con el mapa de 2014
cs2014 <- st_read("cs2014.shp")
```

```
## Reading layer `cs2014' from data source
`/home/jf/recursos-mx/cs2014.shp' using driver `ESRI Shapefile'
## Simple feature collection with 4347 features and 4 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 2460000 ymin: 780000 xmax: 2540000 ymax: 820000
## 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 +datum=WGS84 +units=m +no_defs

# plot(cs2014["GRIDCODE"], axes = T)
cs2014r <- rasterize(as(cs2014,Class = "Spatial"), r, "GRIDCODE")
plot(cs2014r,axes=F)
```



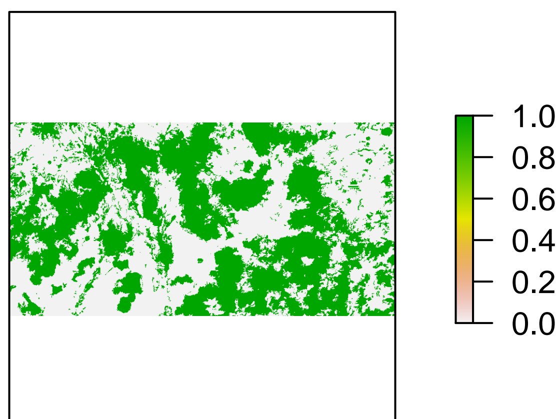
Elaboramos una tabla de correspondencia entre los valores de los campos GRIDCODE y clase realizando un resumen de la tabla de atributos con la función ***aggregate()*** que vimos en el capítulo 2 (sección 2.7). Note que este ***aggregate*** para el manejo de tablas pertenece al paquete `stats` y no es el mismo que el que realiza una agregación espacial de las celdas de un mapa *raster*. R sabe reconocer cual de los dos debe usar por el tipo de insumo en los argumentos de la función. Sin embargo, para evitar toda ambiguedad, se puede especificar el paquete utilizado como en `aggregate(GRIDCODE~clase,FUN=min,data=cs2004,package="stat")`.

```
# Obtiene GRIS CODE / clase
# aggregate(GRIDCODE~clase,FUN=min,data=cs2004,package="stat")
aggregate(GRIDCODE~clase,FUN=min,data=cs2004) # equivalente a anterior

##                                       clase GRIDCODE
## 1                      Agricultura de riego        1
## 2                   Agricultura de temporal        2
## 3                     Asentamientos humanos        4
## 4      Bosque de encino/veg primaria arbrea        6
```

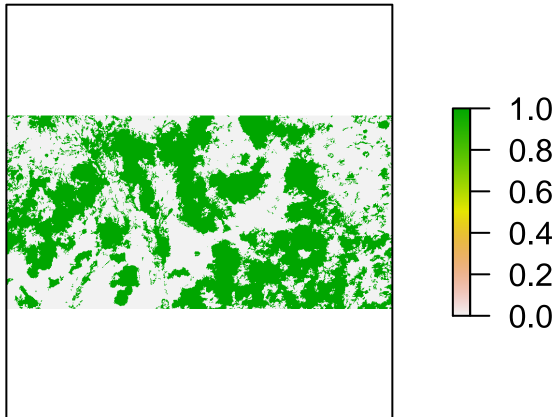
```
## 5   Bosque de encino/veg secundaria herbcea        7
## 6              Bosque de pino/veg primaria       10
## 7            Bosque de pino/veg secundaria       11
## 8               Bosque mesofilo secundario       13
## 9          Bosque Pino encino/veg primaria       14
## 10       Bosque Pino encino/veg secundaria       15
## 11                         Cuerpos de agua       20
## 12                         Cultivo perenne        3
## 13     Pastizal inducido pastizal cultivado        5
## 14      Selva baja caducifolia/veg primaria       16
## 15    Selva baja caducifolia/veg secundaria       17
## 16   Selva mediana caducifolia/veg primaria       18
## 17 Selva mediana caducifolia/veg secundaria       19
## 18                   Sin vegetacin aparente       23
```

En seguida, vamos a reclasificar los mapas en dos categorías (forestal y no forestal) y combinar los mapas de 2004 y 2014 para generar un mapa de las áreas deforestadas durante este periodo. La reclasificación se base en una tabla (matriz con tres columnas) que indica los límites de los intervalos de reclasificación y el valor que tomarán las celdas.

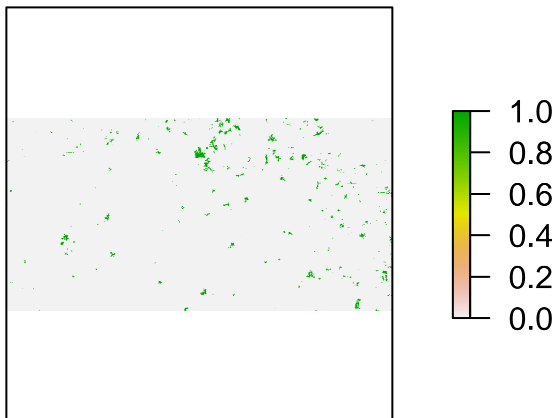
```
# Reclasificación para elaborar un mapa binario forestal / no forestal
# 6-15 y 16-19 son clases forestales
# Matriz tabla para reclasificar
# por default el valor inferior del rango no está incluido
m <- c(0, 5, 0, 5, 15, 1, 15, 19, 1,19,24,0)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
print(rclmat)

##      [,1] [,2] [,3]
## [1,]    0    5    0
## [2,]    5   15    1
## [3,]   15   19    1
## [4,]   19   24    0

F2004 <- reclassify(cs2004r, rclmat)
plot(F2004,axes=F)
```



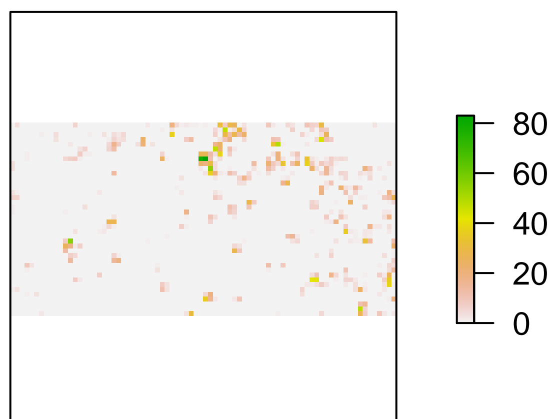
```
F2014 <- reclassify(cs2014r, rclmat)
plot(F2014,axes=F)
```




```
## Algebra de mapas: Deforestación
def <- overlay(F2004, F2014, fun = function(x,y)
               {ifelse( x == 1 & y == 0, 1, 0)})
plot(def,axes=F)
```




```
## Agrega pixels por paquetes de 10 x 10 pixels (suma)
def2 <- aggregate(def, fact=10, fun=sum)
plot(def2,axes=F)
```



```
# Salva el raster
writeRaster(def2, filename="defor.tif", format="GTiff", datatype="INT1U",
            overwrite=TRUE)
```

7. Análisis geoestadístico: Detección de *hot spots*

7.1 Método de Getis Ord

Algunos análisis geoestadísticos buscan analizar los patrones de la distribución espacial de variables cuantitativas. Por ejemplo, se consideran puntos calientes (*hot spots*) y fríos (*cold spots*) áreas que presentan una agregación espacial de valores altos o bajos de la variable considerada, agragación que no se puede expicar por una distribución aleatoria de los valores. En este capítulo, vamos identificar puntos calientes y fríos de deforestación, en los cuales ocurren respectivamente más o menos desmontes de lo que se esperaría si los cambios fueran distribuidas de manera aleatoria en el territorio.

Para dicho análisis, se utilizó el índice estadístico Getis-Ord Gi* (Getis & Ord, 1992). Este índice evalúa la agregación espacial de los datos a través de la comparación de los promedios locales (un sitio y su entorno) y globales (para toda el área de estudio). Esta comparación se basa en el cálculo del valor estandarizado z o z-score (ecuaciones 7.1 y 7.2). Los valores de z (desviaciones estándar) y de p indican si las características se agregan estadísticamente en una distancia dada. Un valor de z superior a 1.96 o inferior a -1.96 significa que la variable evaluada muestra un punto caliente o un punto frío estadísticamente significativos a un cierto nivel de significancia p (generalmente 0.05).

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j} x_j - \bar{X} \sum_{j=1}^{n} w_{i,j}}{S \sqrt{\frac{[n \sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2]}{n-1}}} \tag{7.1}$$

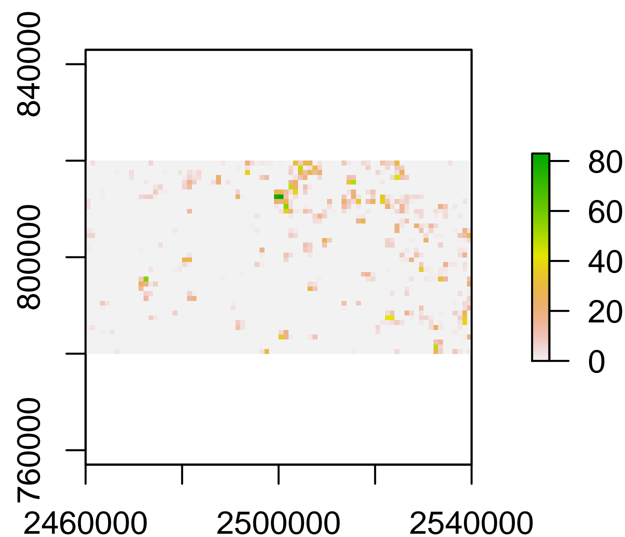
$$donde \qquad \bar{X} = \frac{\sum_{j=1}^{n} x_i}{n} \qquad y \qquad S = \sqrt{\frac{\sum_{j=1}^{n} x_i^2}{n} - (\bar{X})^2} \tag{7.2}$$

y x_j es el valor de la variable considerada (aqui la tasa de deforestación), w_{ij} es el peso de ponderación espacial entre el sitio i y j, n es el número total de sitios. $\bar{X}$ y S son respectivamente el promedio y la desviación estándar del valor de la variable para toda el área de estudio (estadísticas ”globales”). G_i^* se basa por lo tanto en la diferencia entre el promedio de los valores de la variable del punto y sus vecinos, $\sum_{j=1}^{n} w_{i,j} x_j$) y el valor que se podría esperar si la distribución fuera homogénea (valor basado en el promedio global, $\bar{X} \sum_{j=1}^{n} w_{i,j}$). Este diferencia se normaliza para evaluar si rebasa la variabilidad que se podría atribuir a efectos aleatorios (valores de z). Estos cálculos se realizan con el paquete `spdep` (Ver scrip7.R).

7.2 Aplicación a la detección de áreas con altas tasas de deforestación

Vamos a utilizar el mapa de deforestación elaborado en el capítulo 6 (sección 6.2). Este mapa tiene una resolución espacial de un kilómetro y representa el área deforestada (en ha) durante 2004-2014. En un primer paso, se importa el mapa de deforestación en formato *raster* y se convierte en un archivo de puntos (basados en el centro de cada celda) utilizando la función ***rasterToPoints()***. Se genera una tabla de las coordenadas de cada punto (tabla xy) y otra del valor de la tasa de deforestación (tabla tasa). Con base en la tabla de coordenadas y la distancia máxima (5000 m), se determinan los puntos vecinos de cada punto dentro de esta distancia utilizando la función ***dnearneigh()***.

```
library(raster)
library (rgdal)
library(maptools)
library(spdep)  # install.packages("spdep")

#  Ruta del espacio de trabajo
setwd("/home/jf/recursos-mx")

# def indica el núm de pixels deforestados de 100x100 m en cuadros de 1x1 km
def <- raster("defor.tif")
plot(def)
```




```
dim(def)

## [1] 40 80  1

res(def)

## [1] 1000 1000

# Transforma el raster en puntos
def_pts <- rasterToPoints(def)
# plot(def_pts)
head(def_pts,2)

##             x      y defor
## [1,] 2460500 819500     0
## [2,] 2461500 819500     4

summary(def_pts)

##        x                  y               defor
##  Min.   :2460500   Min.   :780500   Min.   : 0.000
```

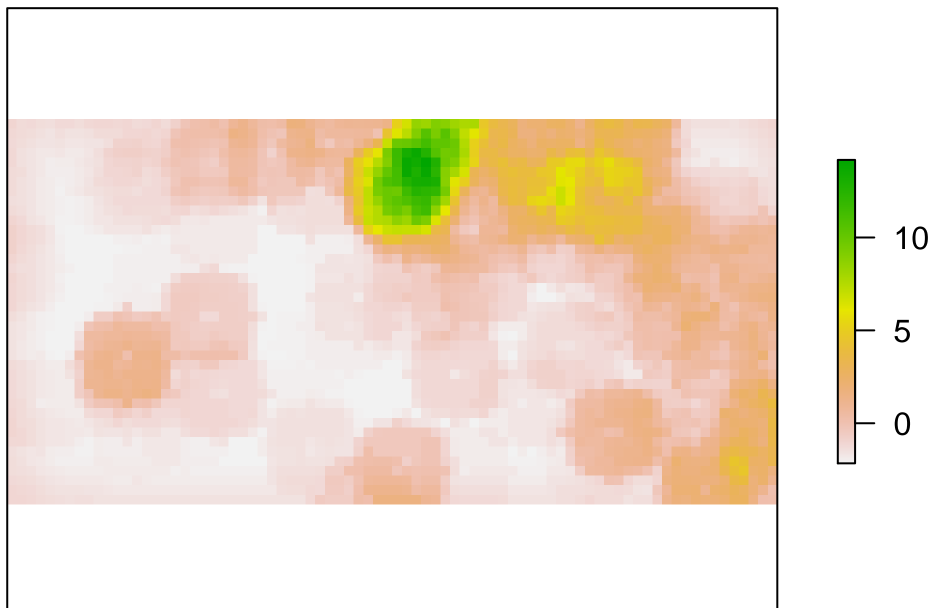
```
##  1st Qu.:2480250   1st Qu.:790250   1st Qu.: 0.000
##  Median :2500000   Median :800000   Median : 0.000
##  Mean   :2500000   Mean   :800000   Mean   : 1.249
##  3rd Qu.:2519750   3rd Qu.:809750   3rd Qu.: 0.000
##  Max.   :2539500   Max.   :819500   Max.   :83.000

class(def_pts)

## [1] "matrix"

# Distancia de búsqueda de los vecinos "locales"
dist_G <- 5000

# Genera matriz con 2 columnas para coordenadas (x e y)
xy <- as.matrix(def_pts[,1:2])
head(xy,3)

##            x      y
## [1,] 2460500 819500
## [2,] 2461500 819500
## [3,] 2462500 819500

tasa <- def_pts[,3]

# Identifica vecinos de cada punto en distancias de 0 a dist_G
vecino <-dnearneigh(xy, 0, dist_G)
print(vecino)

## Neighbour list object:
## Number of regions: 3200
## Number of nonzero links: 235456
## Percentage nonzero weights: 2.299375
## Average number of links: 73.58
```

Utilizando la función ***nb2listw()***, se calculan los pesos asignados a los vecinos de cada punto. En este caso, se utilizó la opción `style="B"`, el peso es por lo tanto binario. Finalmente, se calcula el valor de z para cada punto con la función ***localG()*** (ecuación 7.1).

```
library(spdep)
pesos <- nb2listw(vecino, style="B")
Getis<- localG(tasa, pesos)
class(Getis)

## [1] "localG"

head(Getis)

## [1] -1.042492 -1.309475 -1.254881 -1.364251 -1.446214 -1.265704

z <- as.numeric(Getis)
```

El resultado es un vector que contiene los valores de z de cada punto. Para poder visualizar la distribución espacial de z, tenemos que crear un objeto espacial. Vamos primero a crear una tabla *Dataframe* juntando las coordenadas y los valores de z. Luego, se transforma esta tabla en un objeto `SpatialPointsDataFrame` y `RasterLayer` en el paquete **raster**. El mapa final nos muestra, en verde, las zonas donde la agregación espacial de parches de deforestación rebasa la variabilidad que se podría atribuir a efectos aleatorios. Esta región corresponde al área de produccion aguacatera, que presenta una tasa de deforestación mucho más alta que el conjunto del área de estudio.

```
library(raster)
# Genera una tabla con x y z
spz <- as.data.frame(cbind(xy,z))
head(spz, 3)

##         x      y         z
## 1 2460500 819500 -1.042492
## 2 2461500 819500 -1.309475
## 3 2462500 819500 -1.254881

# Crea un objeto "spatial points data frame"
coordinates(spz) <- ~ x + y
# Convierte a SpatialPixelsDataFrame
gridded(spz) <- TRUE
# Convierte a raster
rasterz <- raster(spz)
plot(rasterz,axes=F)
```



8. Análisis de imágenes de percepción remota

Aunque las herramientas de procesamiento y análisis de imágenes de satélite son aún incipientes en R, existen algunos paquetes para llevar a cabo numerosas operaciones de preprocesamiento y classificación. En este capítulo, vamos a utilizar el paquete **RStoolbox** (2017) para analizar una imagen Landsat 8 de la región de Zirahuén, Michoacán, México (script8.R). El satélite Landsat 8 transporta dos sensores OLI y TIRS. OLI registra nueve bandas espectrales en el espectro visible e infrarrojo cercano, 8 de ellas con 30 m de resolución espacial y la banda pancromática con 15 m. TIRS registra dos bandas en el infrarrojo térmico con 100 m de resolución (ver tabla 2).

8.1 Lectura de imágenes de satélite

Como se muestra a continuación, **RStoolbox** tiene algunas funciones para leer los metadatos de las imágenes Landsat. Estos metadatos tiene una estructura jerárquica que permite extraer ciertos componentes. Por ejemplo, `print(M$PATH_ROW)` lee la franja vertical y la fila horizontal (*Path/Row*). **RStoolbox** permite también importar bandas especificas de la imagen como las bandas con 30 m de resolución, la banda pancromática o la de calidad (QA). Los objetos creados son `RasterStack` cuando la imagen es multibanda o `RasterLayer` cuando hay una sola banda (ver formatos del paquete **raster** en el capítulo 3).

```
#install.packages("RStoolbox") # Instala el paquete si necesario
# Librería RStoolbox y espacio de trabajo
library(RStoolbox)
setwd("/home/jf/recursos-mx")

# Lee los Metadatos
M <- readMeta("LC08_L1TP_028047_20170411_20170415_01_T1_MTL.txt")
head(M,10)  # Despliega las 10 primeras líneas del Metadata

## $METADATA_FILE
## [1] "LC08_L1TP_028047_20170411_20170415_01_T1_MTL.txt"
##
## $METADATA_FORMAT
## [1] "MTL"
##
## $SATELLITE
## [1] "LANDSAT8"
##
## $SENSOR
## [1] "OLI_TIRS"
##
## $SCENE_ID
## [1] "LC80280472017101LGN00"
##
```

Tabla 2. Bandas de Landsat 8 (Sensores OLI y TIRS)

Banda	Nombre	Longitud de onda (μm)	Resolución (m)
1	Costera - Aerosoles	0.435 - 0.451	30
2	Azul	0.452 - 0.512	30
3	Verde	0.533 - 0.590	30
4	Rojo	0.636 - 0.673	30
5	Infrarrojo cercano (NIR)	0.851 - 0.879	30
6	Infrarrojo cercano 1 (SWIR 1)	1.566 - 1.651	30
7	Infrarrojo cercano 2 (SWIR 2)	2.107 - 2.294	30
8	Pancromática	0.503 - 0.676	15
9	Cirrus	1.363 - 1.384	30
10	Infrarrojo térmico (TIR 1)	10.60 - 11.19	100
11	Infrarrojo térmico (TIR 2)	11.50 - 12.51	100

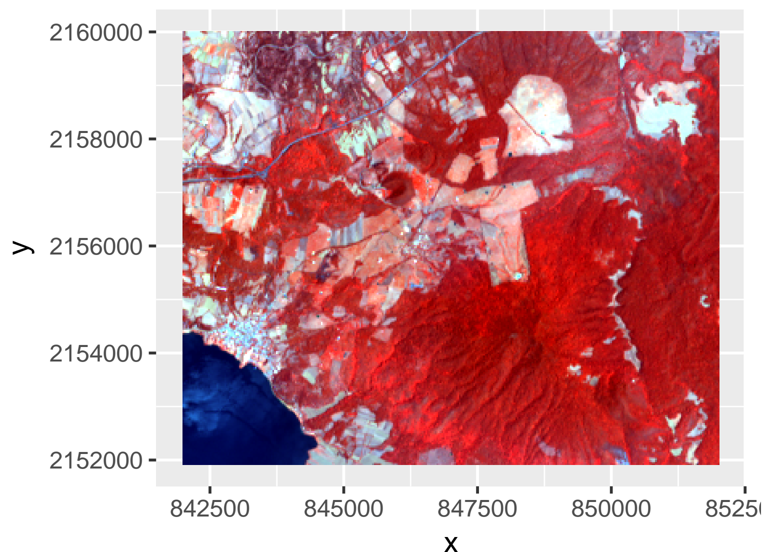
```
## $ACQUISITION_DATE
## [1] "2017-04-11 17:11:51 GMT"
##
## $PROCESSING_DATE
## [1] "2017-04-15 GMT"
##
## $PATH_ROW
## path  row
##   28   47
##
## $PROJECTION
## CRS arguments:
##  +proj=utm +zone=13 +units=m +datum=WGS84 +ellps=WGS84
## +towgs84=0,0,0
##
## $SOLAR_PARAMETERS
##   azimuth elevation  distance
## 109.96555  64.16008   1.00226

print(M$PATH_ROW)

## path  row
##   28   47

# Importa la imagen (por default las 10 bandas con 30 m res)
Metadata_file <- "LC08_L1TP_028047_20170411_20170415_01_T1_MTL.txt"
imagen <- stackMeta(Metadata_file,quantity = "dn")              # Bandas 30m
pan <- stackMeta(Metadata_file,quantity = "dn",category="pan")  # Pancro
qa <- stackMeta(Metadata_file,category="qa")                    # Calidad

print(imagen)

## class       : RasterStack
```

```
## dimensions  : 270, 334, 90180, 10  (nrow, ncol, ncell, nlayers)
## resolution  : 30, 30  (x, y)
## extent      : 841995, 852015, 2151915, 2160015  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=13 +datum=WGS84 +units=m +no_defs
 +ellps=WGS84 +towgs84=0,0,0
## names       : B1_dn, B2_dn, B3_dn, B4_dn, B5_dn, B6_dn, B7_dn, B9_dn, ...
## min values  :  9211,  8333,  7539,  6722,  5945,  5330,  5145,  5010,  ...
## max values  : 13465, 14025, 13971, 15300, 25127, 26331, 21688,  5120,  ...

class(imagen)  # Es un RasterStack

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

class(pan)  # Es un RasterLayer

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

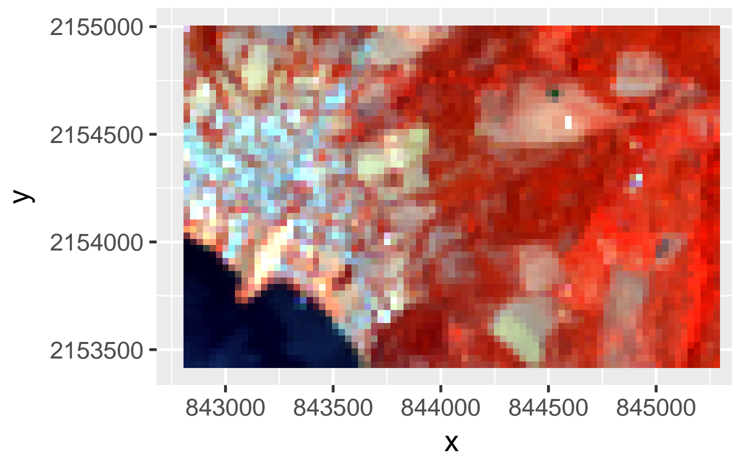
8.2 Visualización y preprocesamintos

Se puede fácilmente crear composiciones a color RGB, en el ejemplo a continuación, con base en las bandas 5, 3 y 2 y un realce linear. **RStoolbox** permite también realizar algunas operaciones como la corrección atmosférica de la imagen, la fusión entre la banda pancromática y la imagen multiespectral o bien la elaboración de índices como los índices de vegetación con base en diferentes métodos que se eligen gracias a los argumentos de las funciones.

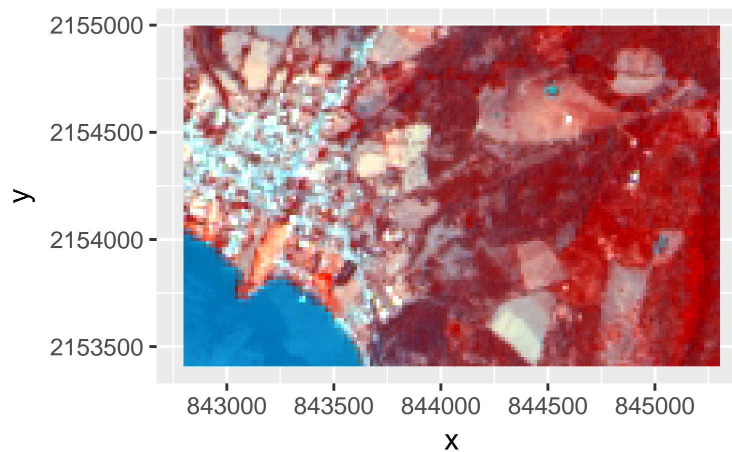
```
# Composición a color RGB
ggRGB(imagen,r=5,g=3,b=2,stretch = "lin")
```



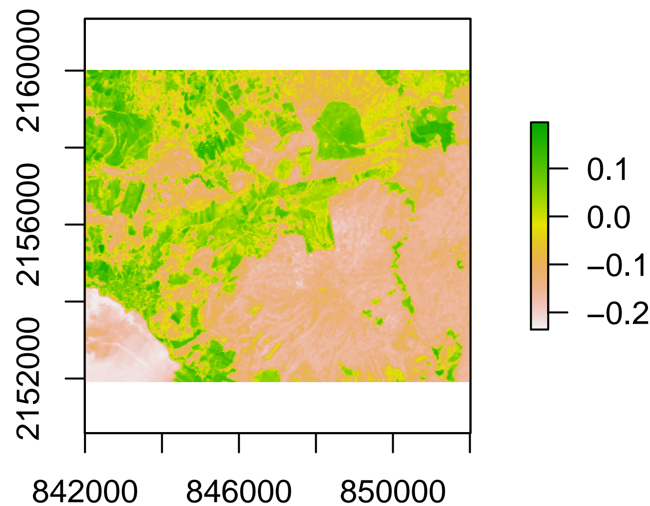
```
# Corrección atmosférica
# apref: Apparent reflectance (top-of-atmosphere reflectance)
imagen_c <- radCor(imagen, metaData = Metadata_file, method = "apref")
# ggRGB(imagen_c,r=5,g=3,b=2,stretch = "lin")

# Fusión bandas multiespectrales y banda pancromática
fusion <- panSharpen(imagen,pan,r=5,g=3,b=2,method="brovey")

# Zoom sobre pequeña ventana
ventana <- extent(842800,845300,2153400,2155000)
ggRGB(imagen,r=5,g=3,b=2,stretch="lin",ext=ventana) # Multiespectral 30 m
```




```
ggRGB(fusion,r=3,g=2,b=1,stretch="lin",ext=ventana) # Fusión 15 m
```



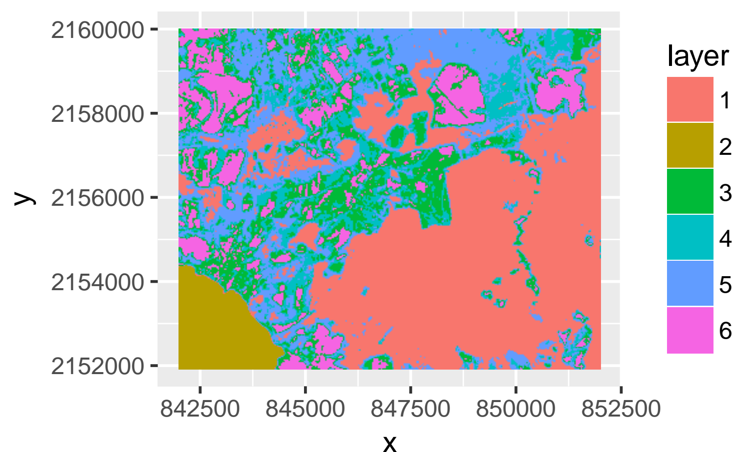

```
# Indices espectrales
Indices <- spectralIndices(imagen_c, red = "B3_tre", nir = "B4_tre",
                           indices = c("SAVI","NDVI"))
plot(Indices[[2]])  # plot NDVI (2a banda)
```



8.3 Clasificación

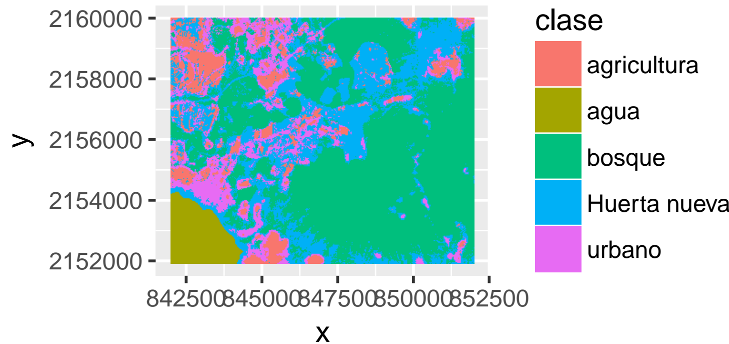
RStoolbox también permite realizar clasificaciones supervisadas o no supervisadas. La clasificación supervisada se basa en campos de entrenamiento que fueron digitalizados en QGIS con base en conocimiento de campo y una interpretación visual. En este caso, también existen muchas opciones, por ejemplo la clasificación supervisada puede llevarse a cabo con diferentes métodos incluyendo la máxima verosimilitud o *randomForest*.

```
# Clasificación no supervisada (método K-means)
clas_nosup <- unsuperClass(imagen, nSamples = 200, nClasses = 6, nStarts = 5)
class(clas_nosup)

## [1] "unsuperClass" "RStoolbox"

ggR(clas_nosup$map,geom_raster = TRUE,forceCat = TRUE)
```




```
# Clasificación supervisada
library(rgdal)
campos <- readOGR(".","campos_utmz13")  # importa shape campos entrenamiento

## OGR data source with driver: ESRI Shapefile
```

```
## Source: ".", layer: "campos_utmz13"
## with 7 features
## It has 2 fields

clas_sup <- superClass(imagen,trainData = campos, responseCol = "clase",
                       model="mlc")
ggR(clas_sup$map,geom_raster = TRUE,forceCat = TRUE)
```



Finalmente, es posible convertir la imagen multiespectral en una tabla en la cual cada pixel es una fila y cada columna corresponde a una banda espectral. Eso permite procesar los datos como paquetes que no fueron diseñados para imágenes.

```
# convierte raster en data.frame
library(ggplot2)
tabla <- fortify(imagen)
head(tabla)

##          x       y B1_dn B2_dn B3_dn B4_dn B5_dn B6_dn B7_dn B9_dn
## 1 842015.2 2159995 10283  9703  9367  9804 14594 17105 13153  5058
## 2 842055.6 2159995  9940  9222  8889  8735 14559 14252 11136  5075
## 3 842096.0 2159995 10319  9809 10301 11569 16350 17636 15516  5070
## 4 842136.4 2159995 10671 10314 11148 13377 17247 19729 18019  5073
## 5 842176.8 2159995 10635 10308 11083 13267 17155 19624 18048  5087
## 6 842217.2 2159995 10629 10296 11138 13307 17070 19296 17641  5072
##   B10_dn B11_dn
## 1  33551  29967
## 2  33944  30270
## 3  34516  30725
## 4  35462  31415
## 5  35831  31622
## 6  36035  31728
```

Existen otros paquetes para procesar imágenes de satélite. Los paquetes **glcm** (Zvoleff, 2016) y **RTextureMetrics** (Klemmt, 2015) permiten calcular índices de textura y **LSRS** (Sarparast, 2017), índices de vegetación. Algunos paquetes se enfocan en un cierto tipo de datos como **npphen** (series de tiempo, Chávez *et al.*, 2017) y los paquetes **lidR** (Roussel *et al.*, 2018a), **PROTOLIDAR** (Rinaldi, 2015), **rlas** (Roussel *et al.*, 2018b), **rLIDAR** (Silva *et al.*, 2017) y **VoxR** (Lecigne *et al.*, 2015) para datos Lidar. Finalmente, otros paquetes como **link2GI** (Appelhans & Reudenbach, 2018), **rgrass7** (Bivand *et al.*, 2017b) y **spgrass6** (Bivand, 2016) permiten utilizar dentro de R funciones de programas de procesamiento de imágenes como GRASS, OTB o SAGA.

9. Elaboración de mapas

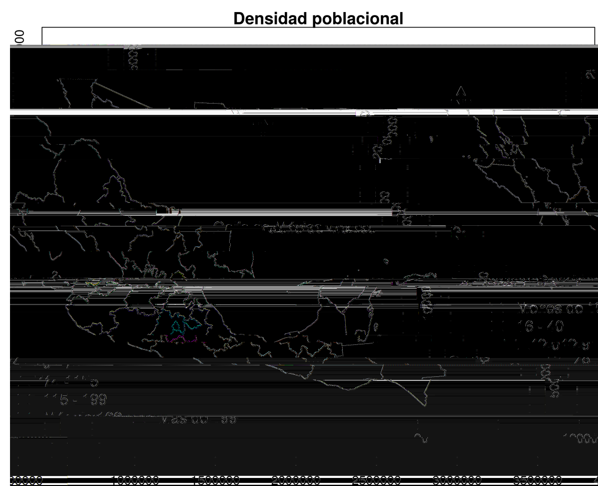
R es famoso por la producción de gráficas muy sofisticadas. Varios paquetes permitem elaborar mapas y figuras. Vamos a crear unos mapas muy simples para representar la densidad poblacional y el relieve (script9.R). Para ello, vamos primero a importar el mapa de los estados en el cual calculamos la densidad 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^2 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 autómática 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**.

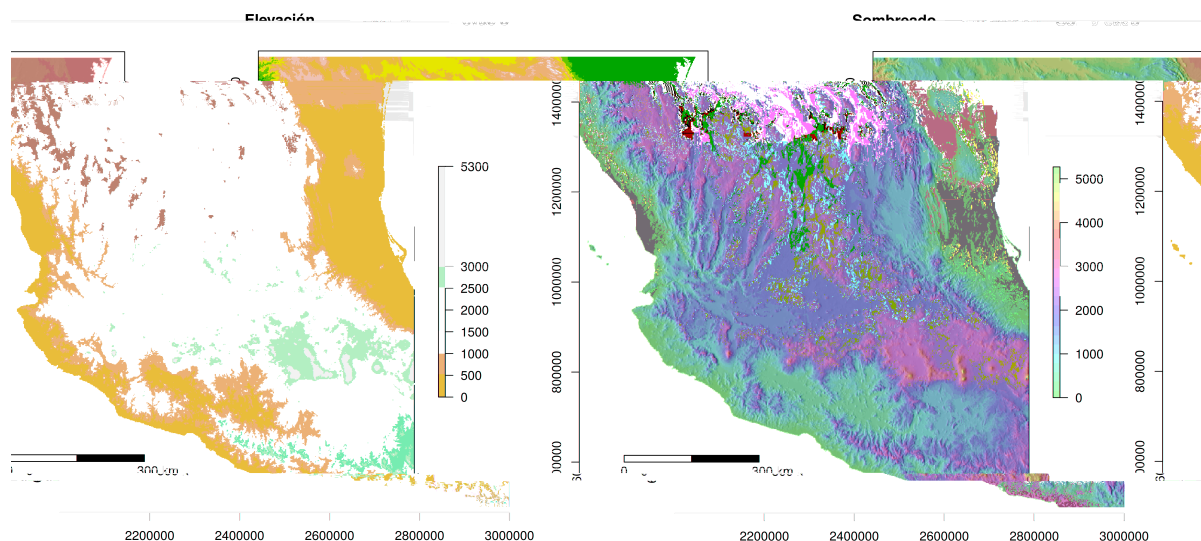
```
# Importa y recorta el DEM
extension <- extent(2000000,3000000,500000,1500000)
dem <- crop(raster("DEM_Mx.tif"), extension)
# Crea una escala de color
colores <- terrain.colors(7, alpha = 1)
# Elabora el mapa
```

```
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 ulilizar 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[1]. 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 proyeco 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
```

[1] 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

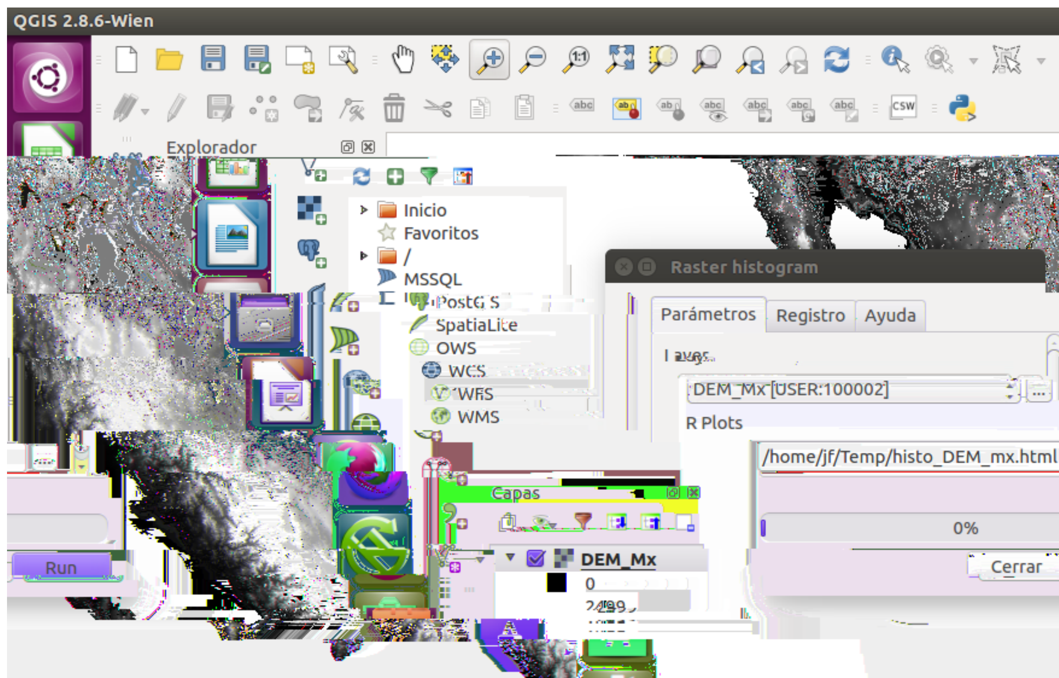


Figura 6. Interfaz gráfica

```
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)

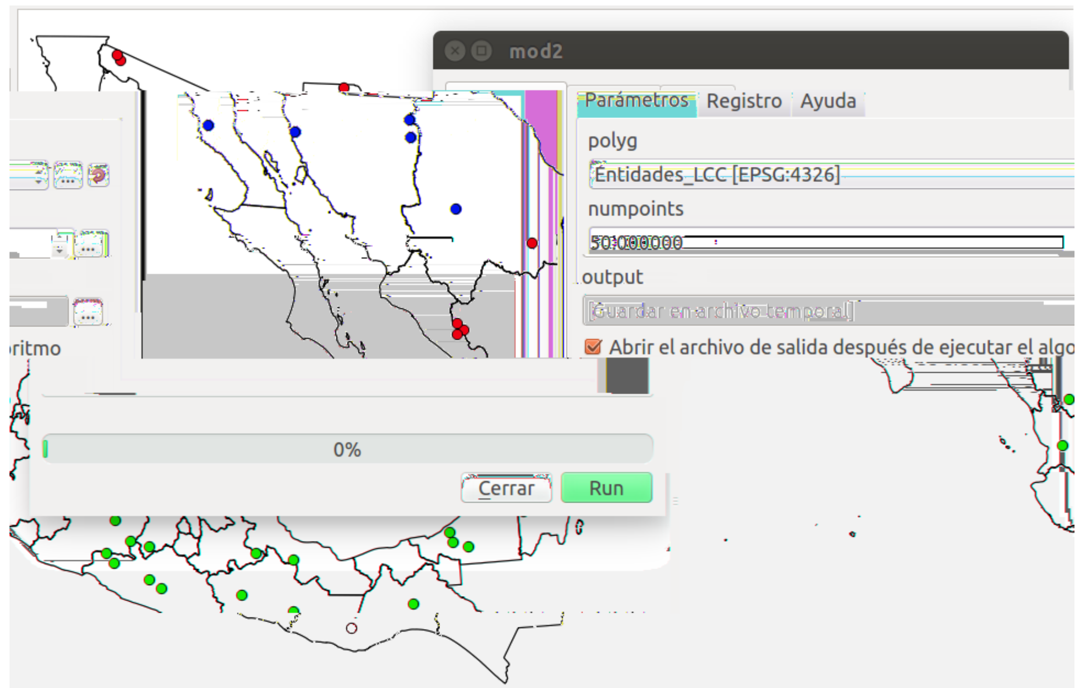


Figura 7. Interfaz gráfica del segundo modelo

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

Se debe también instalar un paquete especialmente diseñado para la interección entre R y Dinamica. Debe obtenerse de la página de DINAMICA [2] 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 esta 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()
```

[2]http://www.csr.ufmg.br/dinamica/dokuwiki/lib/exe/fetch.php?media=dinamica_1.0.2.tar.gz

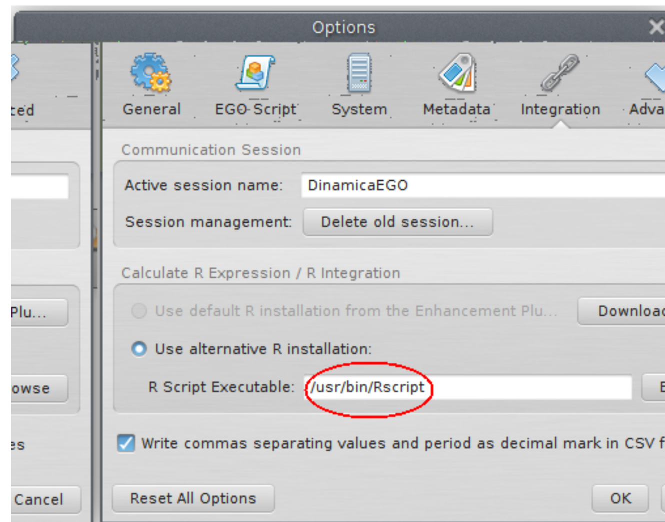


Figura 8. Captura de la ruta del ejecutable de R

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”, coefficents[1])*** y ***outputDouble(”a”, coefficents[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

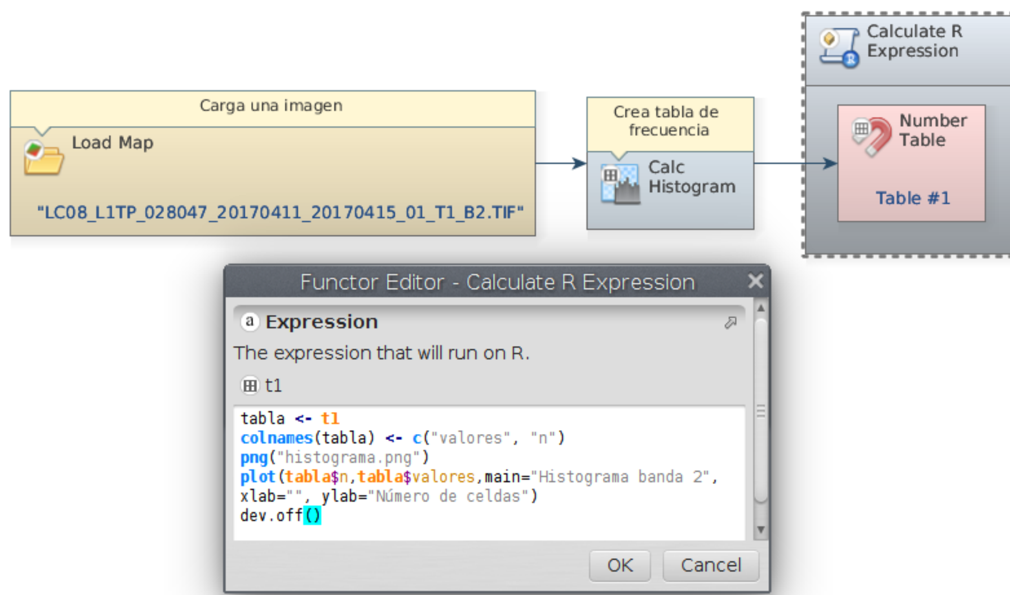


Figura 9. Modelo de Dinamica (primer ejemplo)

functor 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 algebra 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 dataframe
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)
coefficents <- coef(regression)

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

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

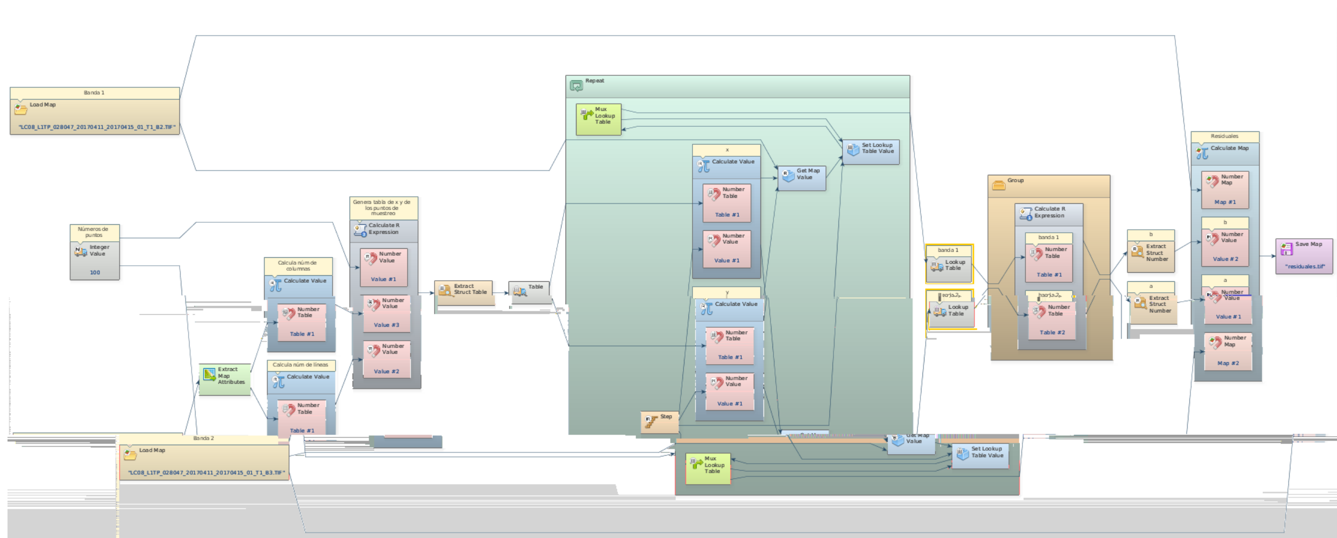


Figura 10. Modelo de Dinamica (Segundo ejemplo)

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 dificil 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/) 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 LIdar 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.

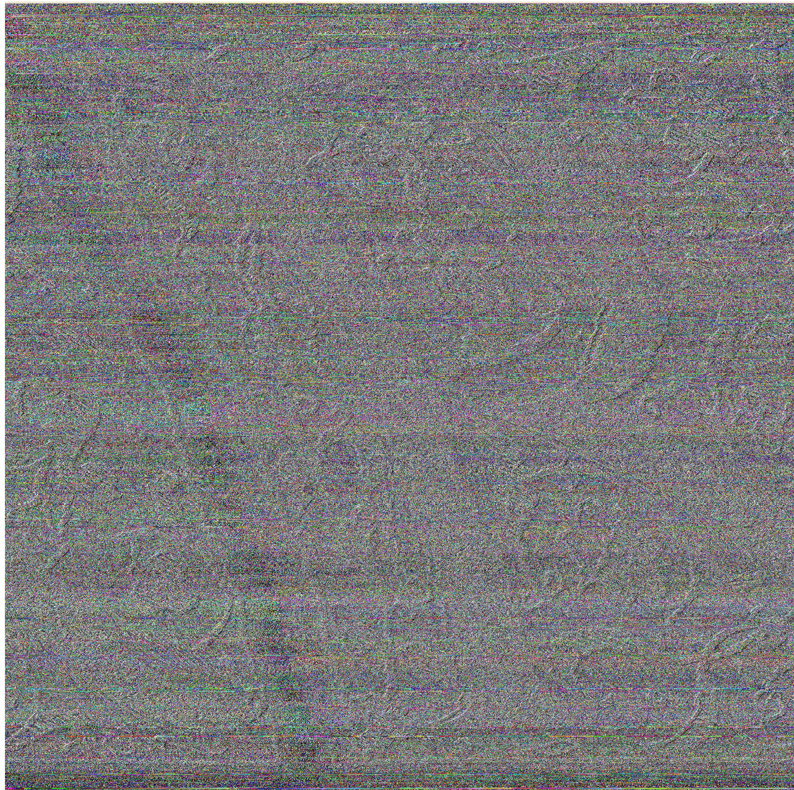
Silva, C. A., N. L. Crookston, A. T. Hudak, L. A. Vierling & C. Klauberg (2017). *rLIDAR: LiDAR Data Processing and Visualization*. R package version 0.8-9.

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** desarollado por Robert J. Hijmans. Para ello, necesitarán instalar ambos paquetes (ver sección **??**). En el desarrolllo 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 libreria **sp** con `library(sp)`.

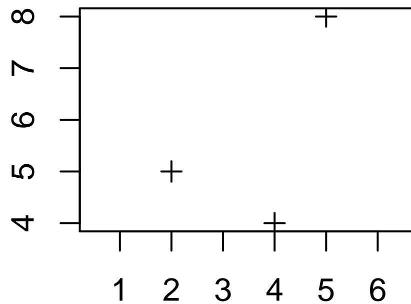
1.1 Datos vectoriales

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`.

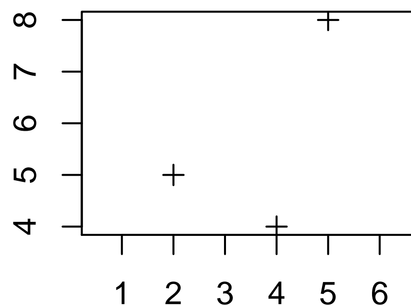
En un primer paso vamos a crear una tabla dataframe con dos columnas, la primera representa el número del punto y la segunda su descripción. El número del punto es el identificador de cada punto que podemos observar con `coordinates(SP)`, corresponde al orden original de las coordenadas cuando creamos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

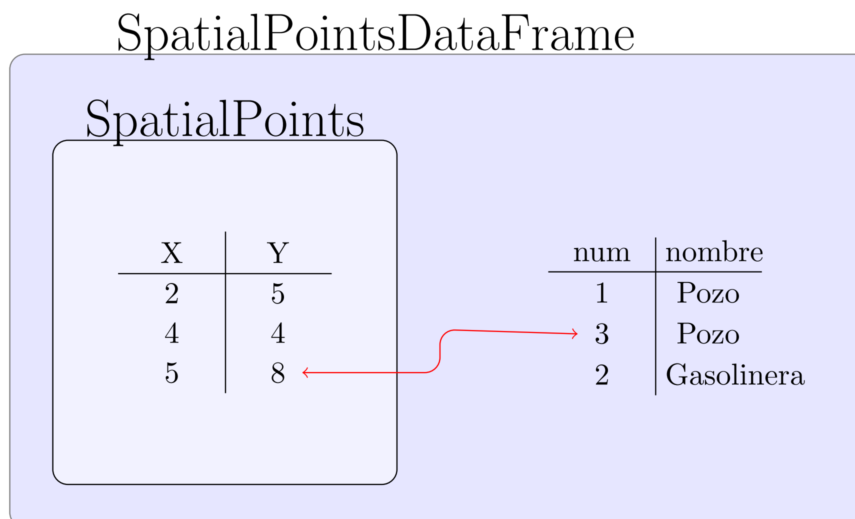
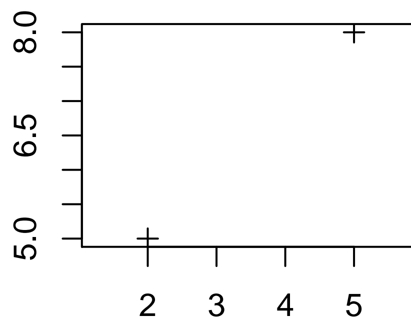


Figura 1. Estructura de un objeto `SpatialPointsDataFrame`

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

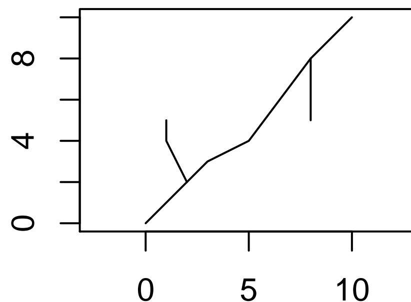
En formato vector, una línea está definida por las coordenadas de los vértices. Para el manejo de este tipo de datos, se manejan objetos de la clase `Line` que describen segmentos simples. Los objetos `Lines` agrupan segmentos `Line` con el mismo identificador (ID), y finalmente el objeto `SpatialLines` 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)
```

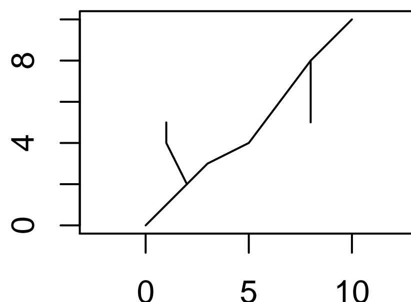
Luego, creamos un objeto `Lines` con base en los segmentos L1 y L2, asignándole el identificador "p". Lines2 está compuesto únicamente del segmento L3, usando el ID ”t”. En un paso final, juntamos ambos objetos `Lines` en un objeto `SpatialLines`.

```
# 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 selecionar líneas con ciertas características. Por ejemplo, en el ejemplo a continuación se

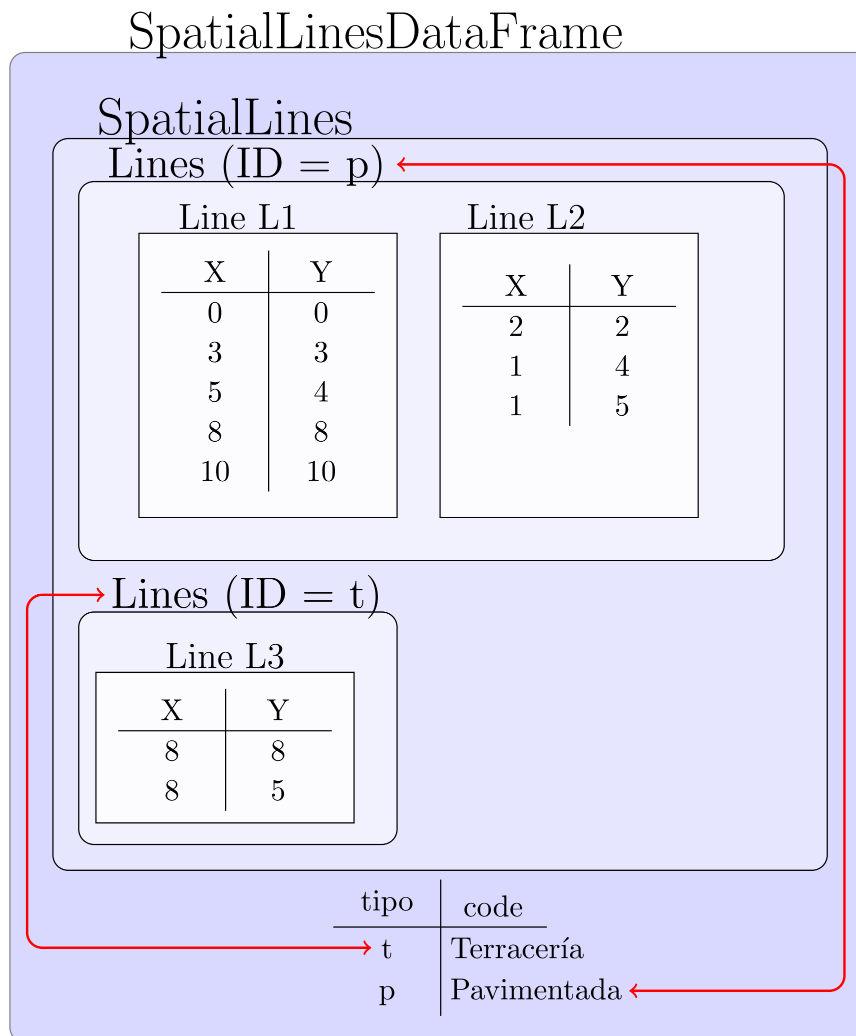
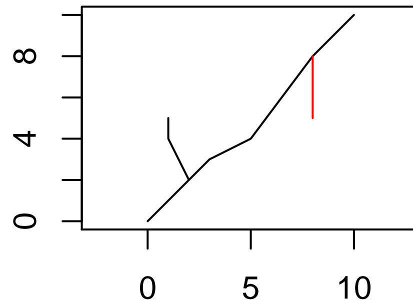


Figura 2. Estructura de un objeto `SpatialLinesDataFrame`

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 selecionar 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.

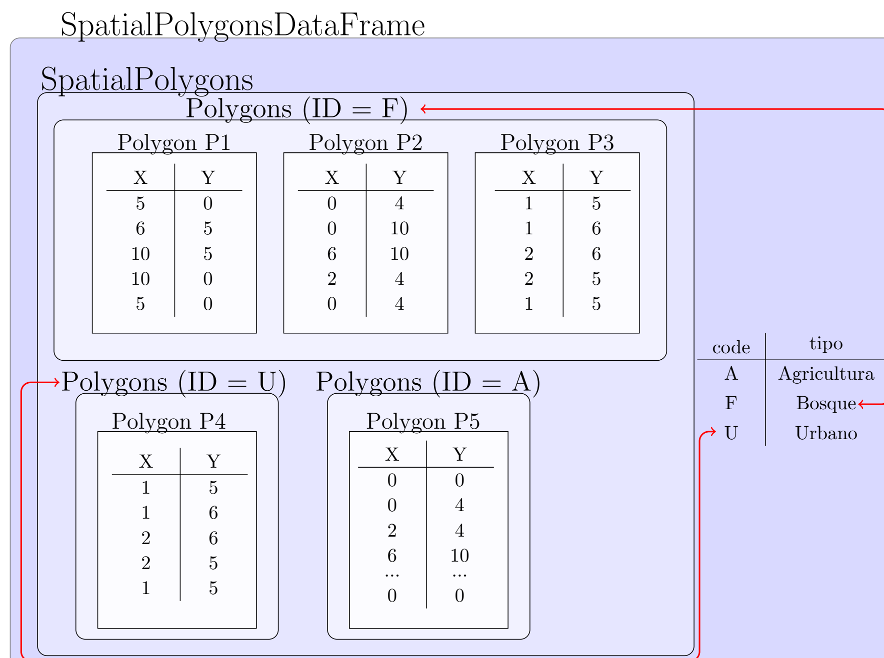
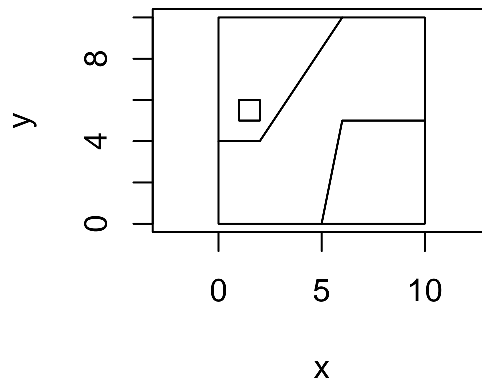


Figura 3. Estructura de un objeto de la clase `SpatialPolygonsDataFrame`

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 utilizndo 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
# Ojo 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 commando ***summary()*** nos permite obtener una informacón 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
```

```
SPolDF@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

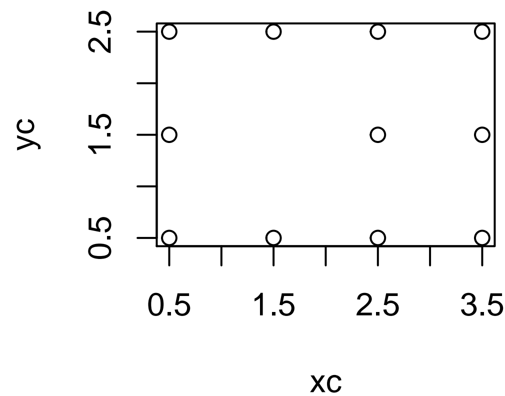
En el paquete **sp** existen dos formas de representar datos raster: `SpatialPixels` y `SpatialGrid`. `SpatialPixels` se parece a `SpatialPoints`, pero las coordenadas deben ser distribuidas regularmente en una malla. Las coordenadas son asociadas a los índices del grid. Los objetos `SpatialPixelsDataFrame` 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.

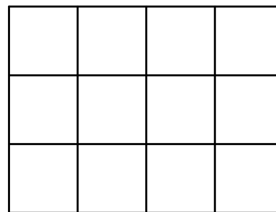
1.2.1 SpatialPixels y SpatialPixelsDataFrame

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, 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)
```



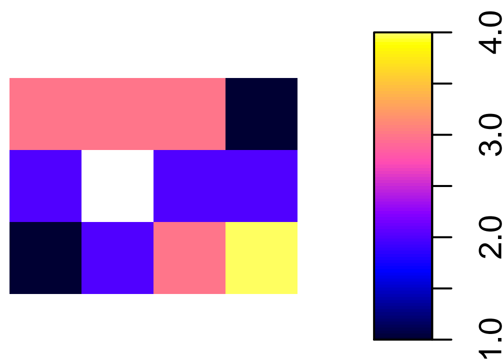
Los objetos `SpatialPixels` tienen diferentes *slots*: grid indica las coordenadas de la celda de la esquina inferior izquierda, el tamaño de una celda y las dimensiones del raster en número de celdas en fila y columna, grid.index muestra el orden de los puntos del objeto `SpatialPoints`, df@coords indica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

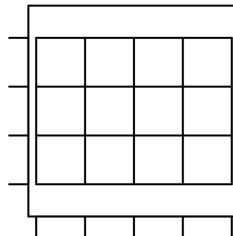
```
G = GridTopology(c(0.5,0.5), c(1,1), c(4,3))
class(G)

## [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
```

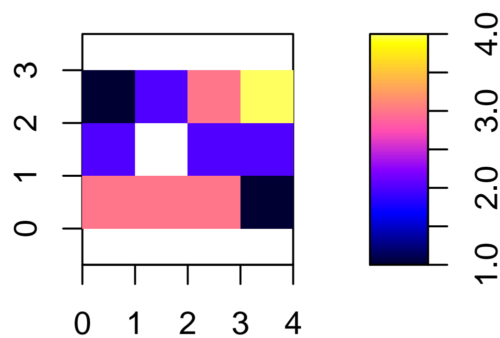
Los objetos `SpatialGrid` representan los *slots* grid que indican las coordenadas de la celda de la esquina inferior izquierda, el tamaño de una celda y las dimensiones del raster en número de celda en fila y columna, bbox la extensión espacial (*bounding box*) y proj4string el sistema de coordenadas.

```
# 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

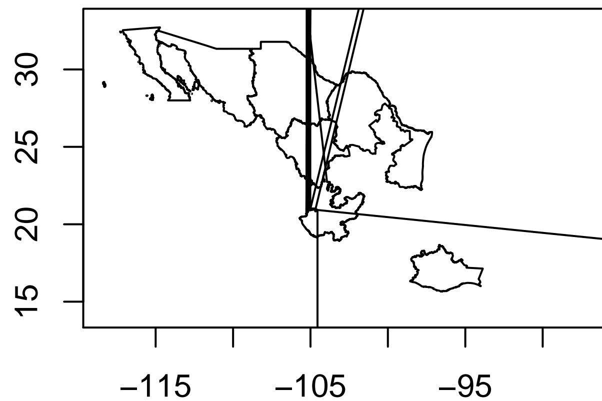
2.1 Importación de archivos *shape* en sp

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 util cuando no se tiene permiso para instalar gdal en la máquina, por exemplo 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)
```



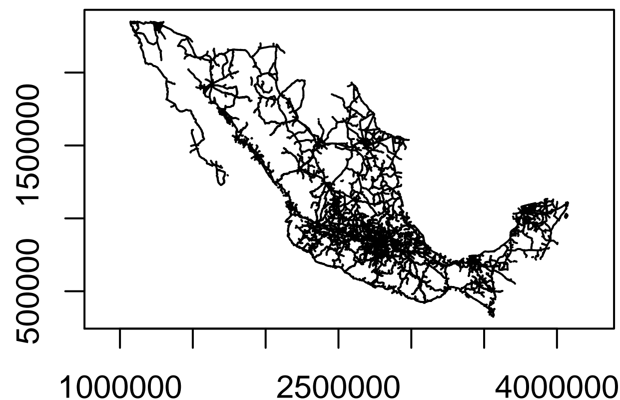

```
# Pregunta por el sistema de proyección
proj4string(mx) # No tiene el sistema de proyección definido

## [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 gdal

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

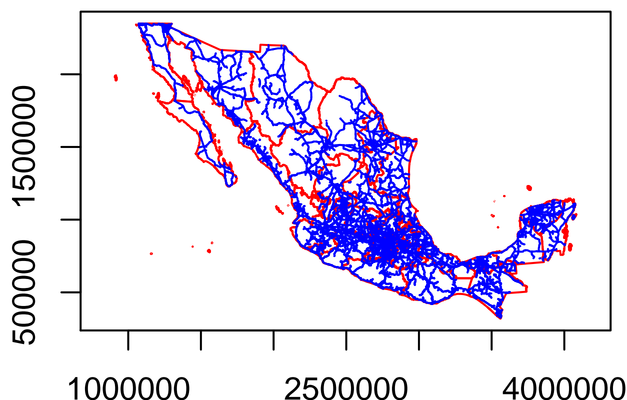
Vamos a reproyectar, desplegar en pantalla y exportar los datos. Para reproyectar el mapa de los Estados mexicanos en la proyección conforme de Lambert, se utilizó la función ***spTransform()*** que tiene como argumentos el nombre del objeto que se desea reproyectar y la proyección de destino. Note que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

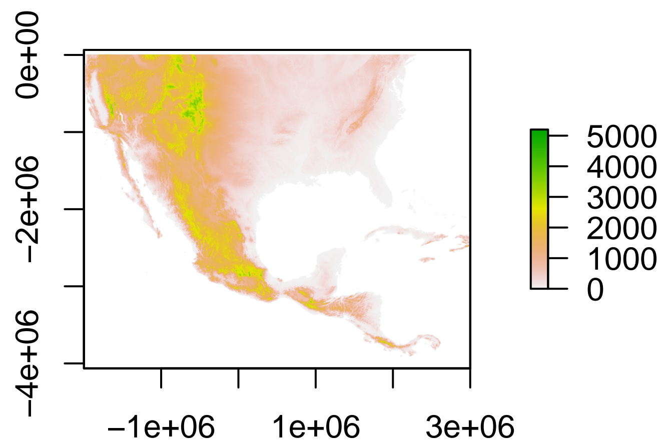
Vamos a importar un modelo digital de elevación, cambiar su proyección, recortarlo y salvar el resultado en formato TIFF utilizando comandos del paquete **raster**.

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

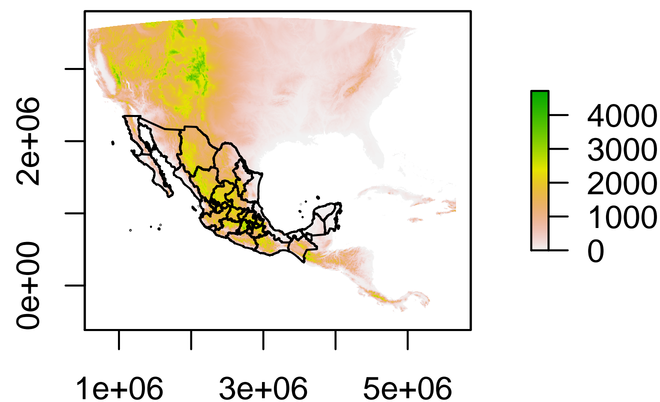
Datatype	Valor mínimo	Valor máximo
LOG1S	FALSE (0)	TRUE (1)
INT1S	-127	127
INT1U	0	255
INT2S	-32,767	32,767
INT2U	0	65,534
INT4S	-2,147,483,647	2,147,483,647
INT4U	0	4,294,967,296
FLT4S	-3.4e+38	3.4e+38
FLT8S	-1.7e+308	1.7e+308

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

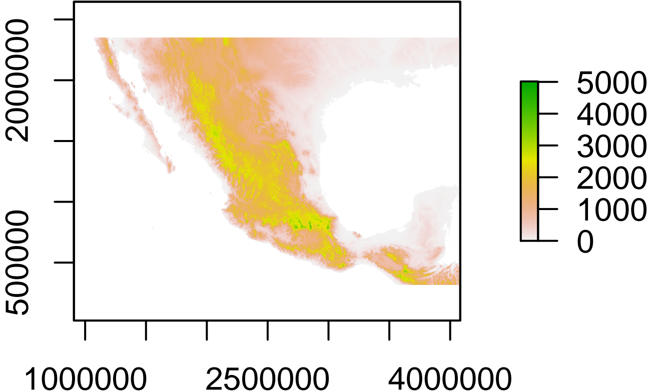



```
extent(tmp)

## 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

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 scientic 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