



Centro de
Investigación
en Matemáticas, A.C.



Introducción a la programación en R

Dr. José Benito Hernández Chaudary

CIMAT

Octubre/Noviembre 2019

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Nociones básicas de R

Nociones básicas de R

- Lo primero que nos aparece es una ventana, también llamada consola, donde podemos manejar R mediante la introducción de código.
- Sin embargo, esta no es la manera más eficiente de trabajar en R.
- A menos que estemos realizando un trabajo de mediana complejidad, será muy útil manejar todas las entradas que solicitemos a R en un entorno donde podamos corregirlas, retocarlas, repetirlas, guardarlas para continuar el trabajo en otro momento, etc.
- Esta es la función del *editor de R*.
- Para ello seleccionemos Nuevo **script** del menú Archivo y para mayor comodidad, elegimos la opción **Divida horizontalmente** del menú Ventanas de la consola.

Nociones básicas de R

Nociones básicas de R


- La utilidad de un script o guión de trabajo radica en que podemos modificar nuestras líneas de código con comodidad y guardarlas para el futuro.
- Para ello, utilizaremos la opción **Guardar** o **Guardar como** del menú Archivo de la consola.
- Después podremos recuperar el **script** previamente guardado mediante la opción **Abrir script** del mismo menú.
- Es posible incluir comentarios que R no leerá si utilizamos líneas que comiencen con el carácter **#**.
- Por el contrario, si escribimos cualquier orden no antecedida de **#** y queremos solicitar la respuesta a R, podemos hacerlo mediante la tecla **F5** situándonos en cualquier posición de esa línea (no necesariamente en el final) o la combinación de teclas **Control+R**.
- Asimismo, si seleccionamos con el ratón más de una línea, éstas pueden ser ejecutadas simultáneamente también con **F5** o **Control+R**.

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

R Studio

R Studio

- **RStudio**  es un entorno de desarrollo integrado (IDE) para R (lenguaje de programación) .
- Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.
- **RStudio** está disponible para Windows, Mac y Linux o para navegadores conectados a RStudio Server o RStudio Server Pro (Debian / Ubuntu, RedHat / CentOS, y SUSE Linux).
- **RStudio** tiene la misión de proporcionar el entorno informático estadístico R. Permite un análisis y desarrollo para que cualquiera pueda analizar los datos con R.
- **IDE construido exclusivo para R**
 - El resaltado de sintaxis, auto completado de código y sangría inteligente.
 - Ejecutar código R directamente desde el editor de código fuente.
 - Salto rápido a las funciones definidas.

R Studio

R Studio

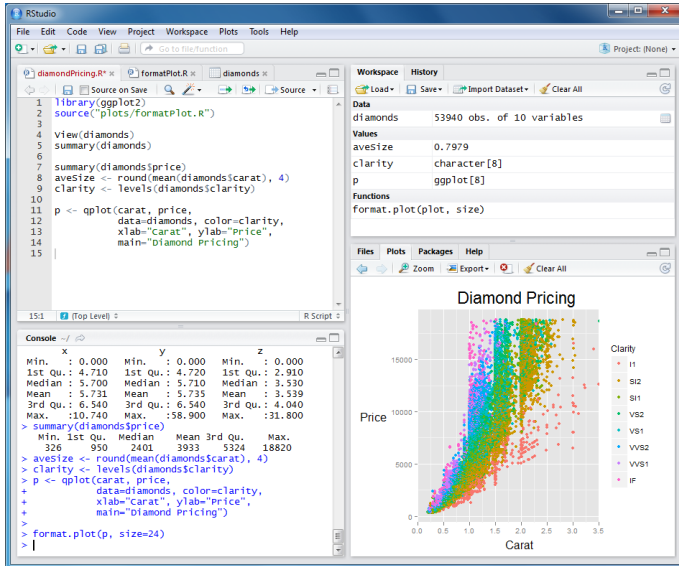
● Colaboración

- Documentación y soporte integrado.
- Administración sencilla de múltiples directorios de trabajo mediante proyectos.
- Navegación en espacios de trabajo y visor de datos.

● Potente autoría y depuración

- Depurador interactivo para diagnosticar y corregir los errores rápidamente.
- Herramientas de desarrollo extensas.
- Autoría con Sweave y R Markdown.

R Studio



Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

La consola de R

Sobre la consola podemos ejecutar diferentes acciones

- Ayuda
- Ver las librerías y objetos disponibles



Manera clásica de consultar la ayuda

```
help(sd)
?sd
help.search("solve") # Archivos relacionados con "solve"
help() # Ayuda para caracteres especiales o palabras reservadas
```



Librerías y objetos disponibles

```
library() # Muestra las librerías disponibles que pueden ser cargadas
search() # Muestra las librerías ya cargadas y disponibles
ls() # Objetos usados
```

La consola de R

Se puede usar R como una simple calculadora

```
5-1+10 # suma y resta
7*10/2 # multiplica y divide
pi # el número pi
sqrt(2) # Raíz cuadrada
```

Crear objetos mediante asignaciones

```
x <- 5 # El objeto x toma el valor 5
x # imprime x
6 = x # x ahora toma el valor 6
(x <- pi) # asigna el número pi a x e imprime
```

La consola de R

R es sensible a mayúsculas y minúsculas

```
# Dos objetos diferentes  
b <-2  
B <-4
```

Los objetos pueden ser de tipo:

- Numéricos, carácter, lógicos, factores
- Vectores, matrices, listas.
- Funciones, entre otros.

Comandos útiles

```
ls() # Muestra los objetos usados  
rm(b) # Borra objetos usados  
ls()
```

La consola de R

R tiene un gran número de funciones. Al crear objetos (como `x`) hay que evitar los nombres de funciones. Para descubrir si un nombre es de una función predeterminada de R, se pone simplemente el nombre que queremos usar para el objeto:

 **Saber si un nombre es una función de R**

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Tipos de objetos en R

Tipos de objetos en R

La información manipulada en R es en forma de objetos. Ejemplos de objetos son vectores de valores numéricos (reales) o valores complejos, vectores de valores lógicos y vectores de caracteres. **Las mismas funciones del R son objetos. R también opera con objetos llamados listas, que son del modo lista.** Estos son secuencias ordenadas de objetos que individualmente pueden ser de cualquier tipo. Concretamente, vamos a hablar de:

- Vectores.
- Matrices.
- Listas y Factores.
- Hojas de datos (Data frame).

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - **Vectores**
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Vectores

Vectores

- Los elementos de un vector pueden ser: enteros, decimales, caracteres, etc.
- R efectúa las operaciones aritméticas entre vectores componente a componente: si sumamos dos vectores de igual longitud el resultado es otro vector de la misma longitud, cuyas componentes son la suma de las componentes de los vectores que sumamos.
- De manera similar ocurre si realizamos cualquier otra operación aritmética, incluyendo la potenciación.
- Veamos algunos ejemplos.



Vectores

```
x<-0:19 # una secuencia de 20 números
x[5] # muestra el elemento en la 5ta. posición
x<-seq(1, 10, length = 5)
seq(1,20,by=2) # Secuencia de dos en dos
# Vector de ceros.
numeric(6)
integer(3)
rep(2,6) # Vector de constantes.
```

Vectores



Vector a través de la función "c"

```
# Vector de números reales y lo asignamos a x
x<-c(1.3,2.5,3.6,4.7,8.2)
y<-c(x,0,x) # Generamos un vector a partir de otro vector.
# Se pueden agregar nombres a los vectores.
z<-c(1,3,45)
names(z)
names(z)<-c("primero","segundo","tercero")
z
```



Vector lógico

```
x<-1:5
cond1<-x<4
cond1
cond2<-x>=3
cond1 & cond2 # Hacemos una 'y' lógica
logical(3) # Vector de tres elementos lógicos
```

Vectores

Vectores

Así como generamos los vectores lógicos también podemos generar caracteres



Generando caracteres

```
(x<-character(3)) # vector de 3 entradas vacías  
# Vector con las 4 primeras letras del alfabeto  
x2=c("A", "B", "C", "D")  
# Otra forma  
x3=LETTERS[1:4]
```

Vectores

Vectores

Para saber cuales son los atributos de una variable u objeto podemos utilizar las siguientes funciones:

`mode()`, `is.logical(x)` ; `is.numeric(x)` ; `is.integer(x)`, etc.

Algunos valores que pueden aparecer cuando trabajamos con vectores u otros objetos son **NA**, **NaN**, **Inf**. Estas tres cantidades tienden a confundirse. Vamos a tratar de aclarar su significado mediante un ejemplo.



Ejemplo

```
(x<-c(NA, 0 / 0, Inf - Inf, Inf, 5))  
NA: # representa un elemento ausente  
NaN: # Son las indeterminaciones  
Inf: # representa al infinito
```

Vectores

Vectores

- Podemos conocer la longitud o dimensión de un vector utilizando el comando `length()`
- Por ejemplo, podemos ver la longitud de uno de los vectores que ya hemos creado `length(y)`
- Podemos convertir los vectores de numéricos a lógicos, de lógicos a enteros y viceversa. Esto se hace con las funciones `as.numeric`, `as.logical`, `as.integer`, etc.



Ejemplo

```
h1=c(1.3,0.6,-3,8);h1
h2=as.integer(h1);h2
as.logical(h2)
```

Operaciones con vectores

Operaciones con vectores

Vamos a realizar ahora algunas operaciones con vectores tales como sumas, productos, potencias, etc.



Ejemplo

```
a<-5:2
```

```
b<-(1:4)*2
```

```
a
```

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

```
b
```

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

Operaciones con vectores

Ejemplo

```
a+b
```

```
[1] 7 8 9 10
```

```
a - b
```

```
[1] 3 0 -3 -6
```

```
a * b
```

```
[1] 10 16 18 16
```

```
a / b
```

```
[1] 2.50 1.00 0.50 0.25
```

```
a ^ b
```

```
[1] 25 256 729 256
```


Operaciones con vectores

Operaciones con vectores

- Es posible realizar operaciones aritméticas entre un vector y un escalar
- También es posible realizar operaciones con más de dos vectores simultáneamente



Ejemplo

```
# Producto de un escalar por un vector
2 * a
[1] 10 8 6 4
# Operación con más de dos vectores
d<-rep(2,4)
a + b + d
[1] 9 10 11 12
```

Operaciones con vectores

Ejemplo. Operaciones con vectores

Veamos ahora un ejemplo más interesante. Supongamos que queremos evaluar la función

$$f(x, y) = \log \left(\frac{x^2 + 2y}{(x + y)^2} \right)$$

para varios valores de x e y . Una posibilidad es tomar cada par de valores y calcular $f(x, y)$. Podemos, sin embargo, aprovechar la forma en la cual trabaja R con vectores para realizar todas las operaciones con una sola evaluación. Comenzamos por crear los vectores que contienen los valores de interés para x e y .

Operaciones con vectores



Ejemplo

```
# Definimos los vectores x e y
(x<-10:6)
[1] 10 9 8 7 6
(y<-seq(1,9,2))
[1] 1 3 5 7 9
# Definimos f(x,y) en términos de estos vectores.
# Guardamos los resultados en z
(z<-log(x^2 + 2*y) / (x + y)^2)
[1] -0.1708177 -0.5039052 -0.8258336
[4] -1.1349799 -1.4271164
```

Operaciones con vectores

Operaciones con vectores

Funciones de uso común para vectores:

| Nombre | Operación |
|--|--|
| <code>length()</code> | longitud |
| <code>sum()</code> | suma de las componentes del vector |
| <code>prod</code> | producto de las componentes del vector |
| <code>cumsum()</code> , <code>cumprod()</code> | suma y producto acumulados |
| <code>max()</code> , <code>min()</code> | máximo y mínimo del vector |
| <code>cummax()</code> , <code>cummin()</code> | máximo y mínimo acumulados |
| <code>sort()</code> | ordena el vector |
| <code>diff()</code> | calcula la diferencia entre las componentes |
| <code>which(x==a)</code> | vector de los índices de <code>x</code> para los cuales la comparación es cierta |

Table: Funciones vectoriales

Operaciones con vectores

Operaciones con vectores

| Nombre | Operación |
|---------------------------|--|
| <code>which.max(x)</code> | índice del mayor elemento |
| <code>which.min(x)</code> | índice del menor elemento |
| <code>range(x)</code> | valores del mínimo y el máximo de x |
| <code>mean(x)</code> | promedio de los elementos de x |
| <code>media(x)</code> | mediana de los elementos de x |
| <code>round(x, n)</code> | redondea los elementos de x a n decimales |
| <code>rank(x)</code> | rango de los elementos de x |
| <code>unique(x)</code> | vector con las componentes de x sin repeticiones |

Table: Funciones vectoriales

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - **Matrices y arreglos**
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Matrices y arreglos

Matrices

- Una matriz es un arreglo rectangular de celdas, cada una de las cuales contiene un valor.
- Para crear una matriz en R es posible usar la función `matrix`, cuya sintaxis es `matrix(datos, nrow, ncol, byrow=F, dimnames = NULL)` donde `nrow` y `ncol` representan, respectivamente, el número de filas y columnas de la matriz.
- Sólo el primer argumento es indispensable.
- Si no aparecen ni el segundo ni el tercero, los datos se colocan en una matriz unidimensional, es decir, en un vector.
- Si sólo uno de los valores se incluye, el otro se determina por división y se asigna.

Matrices y arreglos



Matrices

Por ejemplo, para definir la matriz

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Usaremos

```
matriz<-matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,ncol=3)
```


Matrices y arreglos

Matrices

- Las dimensiones de la matriz pueden obtenerse mediante la función `dim()`.
- Si queremos llegar a elementos concretos de una matriz lo haremos utilizando corchetes para indicar las filas y columnas. Por ejemplo,

```
matriz[2,3] # devolvería el valor 8,  
matriz[1:2,2:3]  
matriz[,c(1,3)] # devolverá ??
```
- Tanto para vectores como para matrices, funcionan las operaciones suma y diferencia, si queremos multiplicar dos matrices usaremos `matriz%%matriz` mientras que `matriz*matriz` devuelve la multiplicación elemento por elemento.

Matrices y arreglos



Matrices

```
matrix(1:6)
```

```
 [,1]
```

```
[1,] 1
```

```
[2,] 2
```

```
[3,] 3
```

```
[4,] 4
```

```
[5,] 5
```

```
[6,] 6
```

```
matrix(1:6, nrow=3)
```

```
 [,1] [,2]
```

```
[1,] 1 4
```

```
[2,] 2 5
```

```
[3,] 3 6
```

Matrices y arreglos

Arreglos

- Un arreglo (`array`) de datos es un objeto que puede ser concebido como una matriz multidimensional (hasta 8 dimensiones).
- Una ventaja de este tipo de objeto es que sigue las reglas que hemos descrito para las matrices.
- La sintaxis para definir un arreglo es

`array(data, dim).`

- Las componentes `data` y `dim` deben presentarse como una sola expresión.
- Por ejemplo `c(2, 4, 6, 8, 10)` o `c(2, 3, 2)`:
`x <- array(1:24, c(3, 4, 2))`
produce un arreglo tridimensional: la primera dimensión tiene tres niveles, la segunda tiene cuatro y la tercera tiene dos.
- Al imprimir el arreglo R comienza con la dimensión mayor y va bajando hacia la dimensión menor, imprimiendo matrices bidimensionales en cada etapa.

Matrices y arreglos



Arreglos (Vectores o matrices).

```
array(1:4,6) # Vector de tamaño 6
array(1:6,c(2,5)) # Matriz 2x5, se llena por columnas
# Matrices (Por defecto llena por columnas)
matrix(c(4,5,9,52,67,48),nrow=2,ncol=3)
# Para llenarla por filas, se le agrega
matrix(c(4,5,9,52,67,48),nrow=2,ncol=3, byrow=TRUE)
# Colocando vectores como columna
(y<-cbind(letters[1:4],LETTERS[1:4]))
```

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - **Listas y factores**
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Listas y factores

Listas

En **R** también podemos crear listas y factores. En **R**, una lista es un objeto consistente en una colección ordenada de objetos, conocidos como componentes. No es necesario que los componentes sean del mismo modo ni tampoco que tengan la misma estructura, así una lista puede estar compuesta de, por ejemplo, un vector numérico, un valor lógico, una matriz y una función.



Listas.

```
lista<-list(Marca="Chevrolet", Modelo="Aveo", n.puertas=5,
            Año=c(2006,2007))

lista
# Seleccionamos posiciones de la lista
lista[[1]] # Posición 1 de la lista
lista[[4]][2] # Posición 4 de la lista, subposición2
# También podemos referirnos a las posiciones por los nombres.
lista$Marca
lista$Modelo
```

Listas y factores

Factores

- Los factores son un tipo especial de vectores que permiten analizar un conjunto de datos clasificados según las características que definen el factor.
- Los factores, son vectores unidimensional que incluye los valores correspondientes a una variable categórica, pero también los diferentes niveles posibles de esta variable.
- Los factores permiten trabajar con modelos y gráficas de variables categóricas.
- Por eso son importantes.
- Hay dos formas de hacer factores, la primera es usar la función `gl()`, en donde especificamos el número de niveles, el número de repeticiones y el número total de datos respectivamente, y con el argumento `labels` se especifica el nombre de los niveles.
- La segunda forma es utilizar es usando la función `factor()`, en donde especificamos los datos tipo caracter, el número de niveles y el número total de datos.

Listas y factores

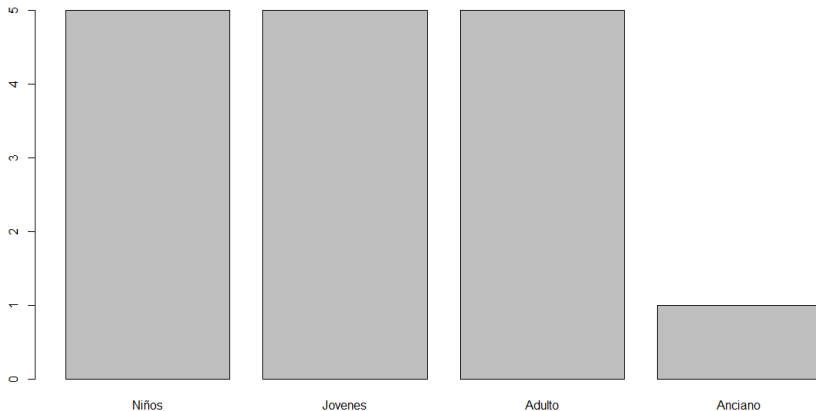


Factores

```
# Con la funcion gl()
edades<-gl(4,5,16,labels=c("Niños","Jovenes","Adulto","Anciano"))
edades
# Grafica del factor
plot(edades)
# Con la funcion factor()
sangre<-factor(rep(c("A","B","AB","O"),4,15))
sangre
# Grafica del factor
plot(sangre)
```

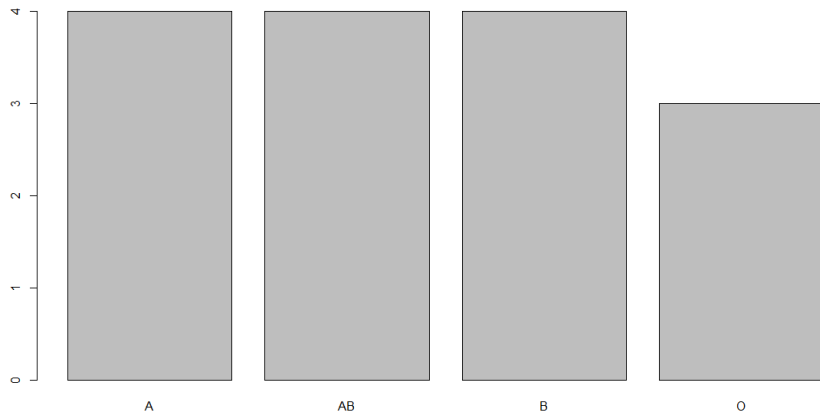

Listas y factores

Factores



Listas y factores

Factores



Listas y factores

Factores

En ocasiones es importante ordenar por nivel de importancia a los factores a estos se le llaman factores ordenados, para eso ocupamos la función `ordered()`: en donde colocamos primero el vector que queremos ordenar, seguido del argumentos `levels` en donde colocaremos primeros los niveles de mayor importancia.



Factores

```
escolaridad<-factor(rep(c("MedioSuperior","Primaria","Secundaria",  
                          "Superior","Prescolar"),5,15))  
  
escolaridad  
ordered(escolaridad,levels=c("Superior","MedioSuperior",  
                             "Secundaria","Primaria","Prescolar"))
```

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - **Data frame**
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Data frame

Data frame

Una **hoja de datos** (`Data frame`) es una lista que pertenece a la clase `"data.frame"`. Hay restricciones en las listas que pueden pertenecer a esta clase, en particular:

- Los componentes deben ser vectores (numéricos, cadenas de caracteres, o lógicos), factores, matrices numéricas, listas u otras hojas de datos.
- Las matrices, listas, y hojas de datos contribuyen a la nueva hoja de datos con tantas variables como columnas, elementos o variables posean, respectivamente.
- Los vectores numéricos y los factores se incluyen sin modificar, los vectores no numéricos se fuerzan a factores cuyos niveles son los únicos valores que aparecen en el vector.
- Los vectores que constituyen la hoja de datos deben tener todos la misma longitud, y las matrices deben tener el mismo tamaño de filas

Las hojas de datos pueden interpretarse, en muchos sentidos, como matrices cuyas columnas pueden tener diferentes modos y atributos. Pueden imprimirse en forma matricial y se pueden extraer sus filas o columnas mediante la indexación de matrices.

Data frame



Data frame

Creando un data.frame

```
dataf = data.frame(Nombre=c("Juan","María","José","Carla"),  
  Edad= c(27,34,40,39),  
  Población=c("Monterrey","Apodaca","Guadalupe","San Pedro"),  
  Sexo=c("M","F","M","F"),  
  Edo.Civil=c("C","S","S","C"))  
  
dataf
```

Por defecto, tanto las listas como los data frame están invisibles en R, para hacerlas visibles y dejar de usar \$ usamos la función **attach** y para hacerlas invisibles nuevamente usamos **detach**.



attach y detach

```
attach(lista)  
Marca  
attach(dataf)  
Edad
```

Data frame



Ejemplo

- Vamos a ver cómo se construye una hoja de datos con los datos de 3 personas, que incluye el color de sus ojos como factor, su peso y su altura.
- Empezaríamos definiendo el color de los ojos:
- Supongamos que los pesos y las alturas son, respectivamente: 68, 75, 88 y 1.65, 1.79, 1.85.
- Entonces, definiríamos la hoja de datos mediante:

```
ojos<-factor(c("Azules", "Marrones", "Marrones"),  
             levels=c("Azules", "Marrones", "Verdes", "Negros"))  
  
datos<-data.frame(Color.ojos=ojos,  
                  Peso=c(68, 75, 88),  
                  Altura=c(1.65, 1.79, 1.85))
```

Data frame

Data frame

- Podemos forzar a que una matriz se convierta en una hoja de datos mediante:
`datos2<-as.data.frame(matriz)`
- Si ponemos **names(datos2)** veremos los nombres que para las variables ha elegido por defecto R:
`[1] "V1" "V2" "V3"`
- Si queremos modificar esos nombres de las variables, podemos usar de nuevo la función `names()`:
`names(datos2)<-c("Variable 1","Variable 2","Variable 3")`
- Podemos usar el operador `$`, de la siguiente manera. Para obtener los datos de la variable `Color.ojos`, por ejemplo, escribiríamos
`datos$Color.ojos`

Tipos de objetos en R

Atributos

- Un **factor** es una variable categórica.
- Una **matriz** es una tabla con dos dimensiones.
- Un **arreglo** es una tabla con k dimensiones.
- Para una matriz o un arreglo los elementos deben ser todos del mismo tipo.
- Una **hoja de datos** (`data frame`) es una tabla formada por vectores y/o factores de la misma longitud pero posiblemente de modos distintos.
- Un **ts** es un conjunto de datos correspondiente a una serie temporal y tiene atributos adicionales como frecuencia y fechas.
- Finalmente, uno de los objetos más útiles es la **lista** (`list`), que puede ser usada para combinar cualquier colección de objetos de datos en un solo objeto (incluyendo otras listas).

Tipos de objetos en R

Tipos de objetos en R

| Objeto | Modos | Varios modos |
|---------------|--|--------------|
| Vector | Numérico, caracter, complejo o lógico | No |
| Factor | Numérico o caracter (categórico) | No |
| Matriz | Numérico, caracter, complejo o lógico | No |
| Arreglo | Numérico, caracter, complejo o lógico | No |
| Hoja de datos | Numérico, caracter, complejo o lógico | Si |
| ts | Numérico, caracter, complejo o lógico | No |
| Lista | Numérico, caracter, complejo o lógico, función,... | Si |

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos**
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Lectura/Escritura de datos

Lectura/Escritura de datos

- R utiliza el directorio de trabajo para leer y escribir archivos.
- Para saber cual es este directorio puede utilizar el comando `getwd()` (*get working directory*).
- Para cambiar el directorio de trabajo, se utiliza la función `setwd()`.
- Por ejemplo, `setwd("C:/data")` o `setwd("/Clases/Maestria INEGI")`.
- Es necesario proporcionar la direccion ('path') completa del archivo si este no se encuentra en el directorio de trabajo.

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos**
 - Leer datos de un archivo**
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Leer datos de un archivo

La función `read.table()`

Para poder leer una hoja de datos directamente, el archivo externo debe reunir las condiciones adecuadas. La forma más sencilla es:

- La primera línea del archivo debe contener el **nombre** de cada variable de la hoja de datos.
- En cada una de las siguientes líneas, el primer elemento es la **etiqueta de la fila**, y a continuación deben aparecer los valores de cada variable.

Si el archivo tiene un elemento menos en la primera línea que en las restantes, obligatoriamente será el diseño anterior el que se utilice. A continuación aparece un ejemplo de las primeras líneas de un archivo, **datos.casas**, con datos de viviendas, preparado para su lectura con esta función.

Leer datos de un archivo



Uso de la función read.table()

| | Precio | Superficie | Área | Habitaciones | Años | Calef |
|-----|--------|------------|------|--------------|------|-------|
| 01 | 52.00 | 111.0 | 830 | 5 | 6.2 | no |
| 02 | 54.75 | 128.0 | 710 | 5 | 7.5 | no |
| 03 | 57.50 | 101.0 | 1000 | 5 | 4.2 | no |
| 04 | 57.50 | 131.0 | 690 | 6 | 8.8 | no |
| 05 | 59.75 | 93.0 | 900 | 5 | 1.9 | no |
| ... | | | | | | |

```
PreciosCasas<-read.table("datos.casas.txt")
```

Leer datos de un archivo

Lectura de datos: Leer un archivo

Vamos a leer una tabla guardada en un archivo de texto usando la función `read.table()`. El archivo se llama `MEXpob.txt` que tiene una tabla que muestra la población (en millones) de México desde 1985.



La función `read.table()`

```
MEXpob<-read.table("MEXpob.txt", header=TRUE)
MEXpob
plot(Pob. ~ Año, data=MEXpob, pch=16)
```


Leer datos de un archivo

La función `scan()`

La función `scan` es más flexible que `read.table`. A diferencia de esta última es posible especificar el modo de las variables:



La función `scan()`

```
misdatos<-scan("entrada.txt", list(" ",0,0))  
misdatos
```

Leer datos de un archivo

La función `read.fwf()`

La función `read.fwf()` puede usarse para leer datos en archivos en *formato fijo ancho*:



La función `read.fwf()`

```
misdatos2 <- read.fwf("datos.txt", widths=c(1, 4, 3)) misdatos2
```

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos**
 - Leer datos de un archivo
 - Guardar datos**
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Guardar datos

Guardar datos

- Podemos salvar los objetos que hemos creado hasta ahora de la siguiente manera:
 - Archivo → Guardar área de trabajo
 - Colocamos el nombre `.Rdata`, ejemplo: `clase1.RData`.
- También podemos salvar todos los comandos ejecutados:
 - Archivo → Guardar histórico
 - Colocamos el nombre `.Rdata`, ejemplo: `claseh1`
- Para abrir el archivo guardado puedes dar doble click sobre el archivo y una vez abierto se realiza lo siguiente:
 - Archivo → Cargar área de trabajo. `Clase1.RData`.
- Para abrir el histórico:
 - Archivo → Histórico. `claseh1`

Guardar datos

Guardar datos

- La función `write.table` guarda el contenido de un objeto en un archivo.
- El objeto es típicamente una hoja de datos (`'data.frame'`), pero puede ser cualquier otro tipo de objeto (vector, matriz, . . .).
- Una manera sencilla de escribir los contenidos de un objeto en un archivo es utilizando el comando `write(x, file="data.txt")`, donde `x` es el nombre del objeto (que puede ser un vector, una matrix, o un arreglo).
- Esta función tiene dos opciones: `nc` (o `ncol`) que define el número de columnas en el archivo (por defecto `nc=1` si `x` es de tipo carácter, `nc=5` para otros tipos), y `append` (lógico) que agrega los datos al archivo sin borrar datos ya existentes (`TRUE`) o borra cualquier dato que existe en el archivo (`FALSE`, por defecto).
- Para guardar un grupo de objetos de cualquier tipo se puede usar el comando `save(x, y, z, file="xyz.RData")`.

Guardar datos

Guardar datos

- Para facilitar la transferencia de datos entre diferentes máquinas se pueden utilizar la opción `ascii = TRUE`.
- Los datos (denominados ahora como un *workspace* o "*espacio de trabajo*" en terminología de R) se pueden cargar en memoria más tarde con el comando `load("xyz.RData")`.
- La función `save.image()` es una manera corta del comando `save(list=ls(all=TRUE), file=".RData")` (guarda todos los objetos en memoria en el archivo `.RData`).

Guardar datos

Guardar datos

Vamos a guardar las tablas creadas anteriormente con la función `read.table()`.



La función `write.table()`

```
write.table(PreciosCasas, "./Datos/PreciosCasas2.txt")  
write.table(MEXpob, "./Datos/MEXpob2.txt")
```

Guardar datos

Guardar datos

Aparte de utilizar la función `write.table()`, podemos utilizar la función `write()`. Supongamos que queremos guardar en un fichero el contenido de un vector



La función `write()`

```
x <- c(1,2,3,4,5)
write(x, "./Datos/x.txt")
```

El contenido del fichero `x.txt` será el vector 1 2 3 4 5

Guardar datos

Guardar datos

Si lo que queremos es escribir una matriz con este procedimiento, hay que ser más cuidadosos. Creamos una matriz para probar y la escribimos en el fichero `xx.txt`:



La función write()

```
x <- matrix(1:9, ncol = 3, byrow = T)
x
write(t(x), "./Datos/xx.txt", ncol = ncol(x))
```

Lectura/Escritura de datos

Lectura/Escritura de datos

| Función | Descripción |
|----------------------------|---|
| <code>data.frame()</code> | Lee datos de un archivo de datos con formato específico |
| <code>scan()</code> | Lee datos de un archivo de datos |
| <code>write.table()</code> | Escribe datos en un fichero con formato de tabla de datos |
| <code>write()</code> | Escribe datos en un fichero .txt |
| <code>write.csv()</code> | Escribe datos en un fichero .csv (excel) |
| <code>save()</code> | Guarda un grupo de objetos en un archivo .RData |
| <code>save.image()</code> | Guarda todos los objetos en memoria del espacio de trabajo (.RData) |
| <code>load()</code> | Carga en memoria archivos .RData |

Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 Resumen

Como programar

Como programar

Al igual que otros lenguajes de programación, R incluye la ejecución de condicionales, bucles y esas construcciones. En esta parte se discute brevemente estas construcciones. Debido a su rica colección de funciones y paquetes y por su enfoque orientado a objetos, se evitará la programación en R tanto como sea posible. Hay, sin embargo, situaciones en las que tendremos que confiar en la programación.

- Para programar usaremos un Script. Entramos en:
Archivo → Nuevo Script
- Para guardar realizamos lo siguiente:
Archivo → Guardar como
clase1

Como programar



Condicional:

Un condicional se ejecuta de la siguiente manera:

```
if (condición) {  
  ejecutar algo  
} else {  
  ejecutar algo si la condición anterior  
  no se verifica  
}
```



Ejemplo

```
z=1:10  
# Un ejemplo de if  
z1=sample(z,1)  
if (z1>7) {w=1} else {w=0}  
# Un ejemplo con condición lógica  
x <- TRUE  
if(x) y <- 1 else y <- 0  
y
```

Como programar



Condicional:

Si se tiene más de una instrucción a ejecutar es necesario usar las llaves

```
# Dos asignaciones a ejecutar
x <- FALSE
if(!x){y <- 1 ; z <- 2}
# Dos condiciones a verificar
# Raíz n-ésima de un número real
n=5; x=-32
if(n%%2 == 1 || x >=0 ){
  sign(x)*abs(x)^(1/n)
} else{
  NaN
}
```

Como programar

Ciclos:

Para repetir una acción se pueden usar bucles (ciclos) usando: `for`, `repeat` y `while`. Se puede salir de un ciclo en cualquier instante mediante el uso de `break`

- Usando un **for**
`for(condición){acción a ejecutar}`
- El **while** se ejecuta de la siguiente manera
`while (condición) {Acción a ejecutar}`

Ejemplo

```
# Usando un 'for' y creando vectores lógicos
x1 <- as.logical(as.integer(runif(5, 0, 2)));
x1
y1 <- vector() ; y1
for(i in 1 : length(x1)){
  if(x1[i]){y1[i] <- 1}
  else {y1[i] <- 0}}
y1
```

Como programar



Ejemplo

```
# Un ejemplo de 'repeat' y 'break'
x2=1:10; x3=10:20; i=1
repeat{
  y=x2[i]+x3[i]
  i=i+1
  if(i==10) break
}
y
```


Como programar

Funciones:

- En R también se pueden crear funciones, estas pueden tener argumentos o no, un cuerpo y en general arroja valores.
- El uso de una función en R es similar al uso matemático, en matemáticas escribimos $y = f(x)$ y en R: `y<-f(x)`
- La forma general de una función es

```
nombre.funcion <- function(argumentos)
{
  acciones a ejecutar y valor a retornar
}
```

- El valor a retornar puede ser cualquier objeto: vector, valor numérico, un gráfico etc.
- Los valores pueden ser retornados colocando simplemente el nombre de la variable que contiene la información o usando la función `print`

Como programar



Ejemplo

Vamos a definir una función llamada cubo que toma un número y lo eleva a la potencia tres.

```
# Definimos la función cubo

cubo <- function(x)
{return(x^3)}

# Ejecutamos la función cubo con x=2
cubo(2)
[1] 8
```

Como programar

Funciones

- Una función puede tener muchos argumentos y en esos casos es cómodo no tener que incluir todos los parámetros.
- En esta situación es conveniente poder asignar valores 'por defecto' o automáticos a los parámetros. Esto es posible en R usando el signo =.
- Si llamamos una función sin especificar explícitamente el parámetro, se usará su valor 'por defecto'.



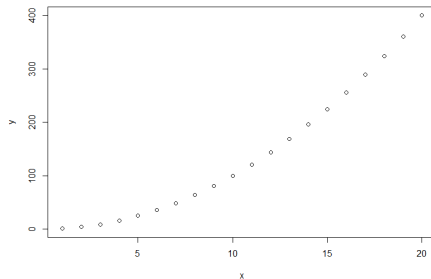
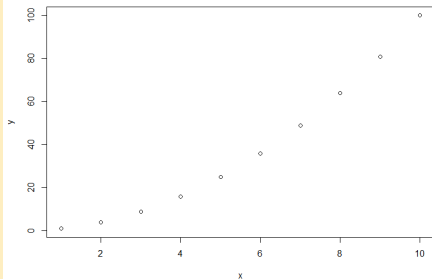
Ejemplo

```
# Definimos la función ff
ff <- function (x = 1:10, y = (1:10)^2, showgraph = T){
  if (showgraph) plot (x,y)
  else print (cbind (x,y))
  return (invisible ())
}

# Ejecutamos la función ff
ff (1:10, (1:10)^2, T)
ff (1:10, (1:10)^2)
ff (1:20, (1:20)^2)
```

Como programar

Funciones



Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R**
- 6 Resumen

Gráficos con R

Gráficos con R

- Una de las mayores ventajas de R es su capacidad gráfica.
- Para ver una demostración usamos los siguientes comandos:

```
demo(graphics)  
demo(persp)  
demo(image)
```

- Posiblemente es la función gráfica más usada en R es `plot`.
- Su efecto depende de la sintaxis y el argumento que usemos.
- A continuación veremos varios ejemplos de gráficos con R.

Gráficos con R



Ejemplo. Distribucion Weibull

```
tt <- seq(0,2,length=200)
par(mfrow=c(2,2),mar=c(3, 3, 2, 2), oma=c(0,0,2,0))
b1 <- .5; t1 <- 1/gamma(1+1/b1)
plot(tt, exp(-(tt/t1)^b1), xlim=c(0,2), cex.axis=.7, cex.lab=.7,
      mgp=c(1.5,.5,0), lwd=2, col="blue", type="l", xlab="t",
      ylab="",
      main= expression(paste(beta==.5, " ", " ", theta==.5)))
b2 <- 1.5; t2 <- 1/gamma(1+1/b2)
plot(tt, exp(-(tt/t2)^b2), xlim=c(0,2), cex.axis=.7, cex.lab=.7,
      mgp=c(1.5,.5,0), lwd=2, col="blue", type="l", xlab="t",
      ylab="",
      main= expression(paste(beta==1.5, " ", " ", theta==1.108)))
```

Gráficos con R

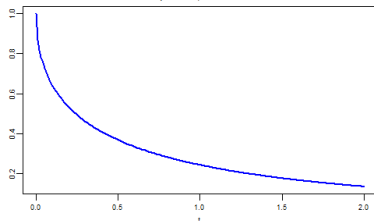
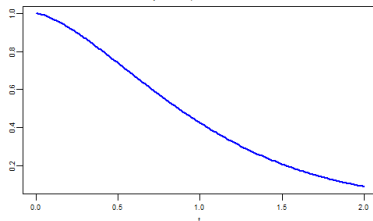
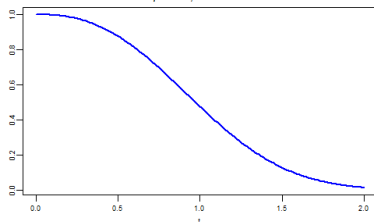
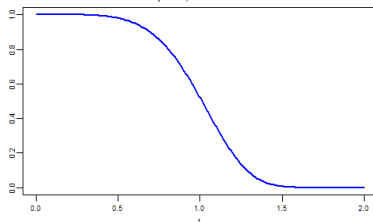


Ejemplo. Distribucion Weibull

```
b3 <- 2.5; t3 <- 1/gamma(1+1/b3)
plot(tt, exp(-(tt/t3)^b3), xlim=c(0,2), cex.axis=.7, cex.lab=.7,
     mgp=c(1.5,.5,0), lwd=2, col="blue", type="l", xlab="t",
     ylab="",
     main= expression(paste(beta==2.5, " ", " ", theta==1.127)))
b4 <- 5; t4 <- 1/gamma(1+1/b4)
plot(tt, exp(-(tt/t4)^b4), xlim=c(0,2), cex.axis=.7, cex.lab=.7,
     mgp=c(1.5,.5,0), lwd=2, col="blue", type="l", xlab="t",
     ylab="",
     main=
       substitute(paste(beta == 5, " ", " ", theta, " = ", t4),
         list(t4=round(t4,3))))
mtext("Funciones de Supervivencia", outer=TRUE, cex=1.2)
# (en esta grafica, notar en la ultima, el uso de substitute)
```


Gráficos con R

Funciones de Supervivencia

 $\beta = 0.5, \theta = 0.5$  $\beta = 1.5, \theta = 1.108$  $\beta = 2.5, \theta = 1.127$  $\beta = 5, \theta = 1.089$ 

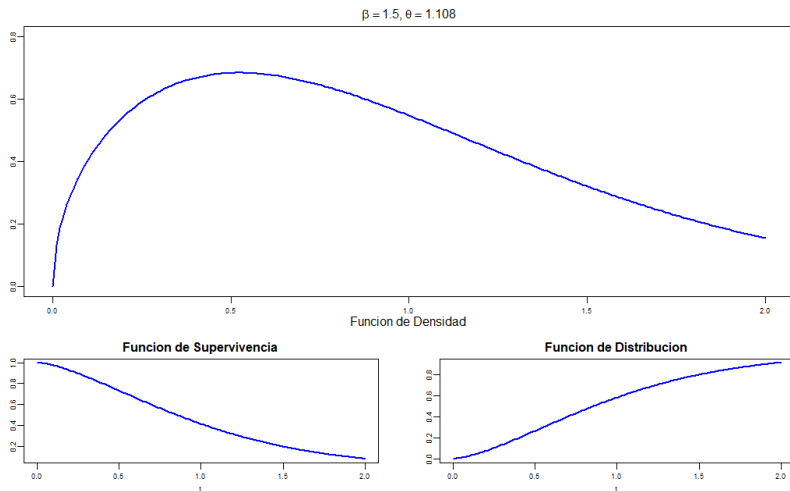
Gráficos con R



Ejemplo. Uso de la función layout()

```
layout( matrix(c(1,1,2,3),ncol=2,byrow=T), heights=c(2,1))
par(mar=c(3, 3, 2, 2))
b2 <- 1.5; t2 <- 1/gamma(1+1/b4)
plot(tt, dweibull(tt,shape=b2, scale=t2), xlim=c(0,2),
      cex.axis=.7, cex.lab=1.2, mgp=c(1.5,.5,0), lwd=2,
      col="blue", type="l", xlab="Funcion de Densidad",
      ylab="", ylim=c(0,.8),
      main= expression(paste(beta==1.5, " ", " ", theta==1.108)))
plot(tt, exp(-(tt/t2)^b2), xlim=c(0,2), cex.axis=.7, cex.lab=.7,
      mgp=c(1.5,.5,0), lwd=2, col="blue", type="l", xlab="t",
      ylab="", main= "Funcion de Supervivencia")
plot(tt, 1-exp(-(tt/t2)^b2), xlim=c(0,2), cex.axis=.7, cex.lab=.7,
      mgp=c(1.5,.5,0), lwd=2, col="blue", type="l", xlab="t",
      ylab="", main= "Funcion de Distribucion")
```

Gráficos con R



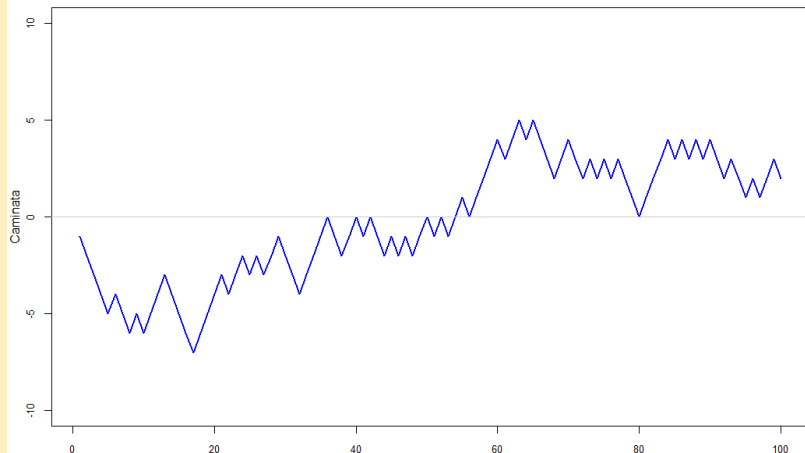
Gráficos con R



Ejemplo. Caminata aleatoria

```
par(mfrow=c(1,1))
n <- 100
xx <- 1:n
cam <- cumsum(sample(c(-1,1), size=n, replace=T))
camina <- function(k){
  plot(1:k, cam[1:k], xlim=c(1,n), ylim=c(-n/10,n/10), type="l",
       col="blue", lwd=2, mgp=c(2,1,0), ylab="Caminata",
       xlab="", cex.axis=.8)
  abline(h=0,col=gray(.8))
  Sys.sleep(0.1) }
# Sys.sleep() controla la rapidez de la animación
trash <- sapply(xx,camina)
```

Gráficos con R



Gráficos con R

Gráficos con R

Los siguientes gráficos fueron tomados del `demo(graphics)`

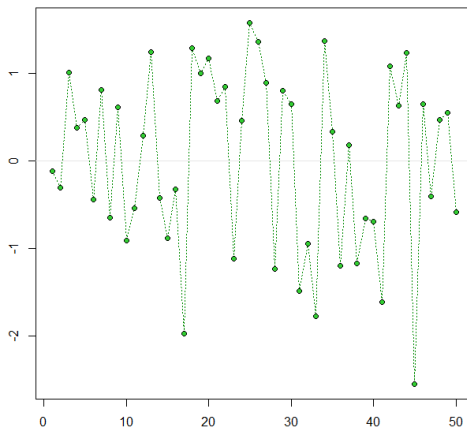


Ejemplo. Una grafica simple

```
# Uso de colores en sus distintos elementos.
opar <- par(bg = "white") # guardar parámetros
x <- rnorm(50)
plot(x, ann = FALSE, type = "n")
abline(h = 0, col = gray(.90))
lines(x, col = "green4", lty = "dotted")
points(x, bg = "limegreen", pch = 21)
title(main = "Ejemplo simple de uso de color en Plot",
      xlab = "Informacion con un color desvanecido",
      col.main = "blue", col.lab = gray(.7),
      cex.main = 1.2, cex.lab = 1.0, font.main = 4,
      font.lab = 3)
```

Gráficos con R

Ejemplo simple de uso de color en Plot



Información con un color desvanecido

Gráficos con R



Ejemplo. Diagrama de pastel.

```
par(bg = "gray")
pie(rep(1,24), col = rainbow(24), radius = 0.9)
title(main = "Una muestra del catalogo de colores",
      cex.main = 1.4, font.main = 3)
title(xlab = "(Use esto como una prueba de la linealidad del monitor)",
      cex.lab = 0.8, font.lab = 3)
```


Gráficos con R



Gráficos con R

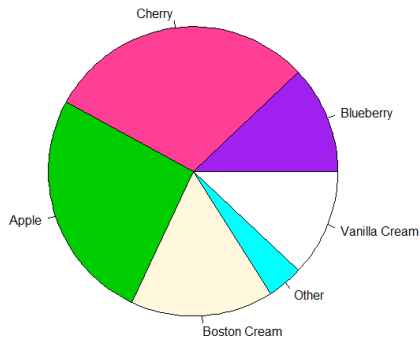
Ejemplo. Diagrama de pastel (de nuevo).

```
pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(pie.sales) <- c("Blueberry", "Cherry",
                      "Apple", "Boston Cream",
                      "Other", "Vanilla Cream")

pie(pie.sales,
    col=c("purple", "violetred1", "green3", "cornsilk", "cyan", "white"))
title(main = "Ventas de Pasteles en Enero", cex.main = 1.8,
      font.main = 1)
title(xlab = "Pasteleria Lety", cex.lab = 1.2, font.lab = 3)
```

Gráficos con R

Ventas de Pasteles en Enero



Pasteleria Lety

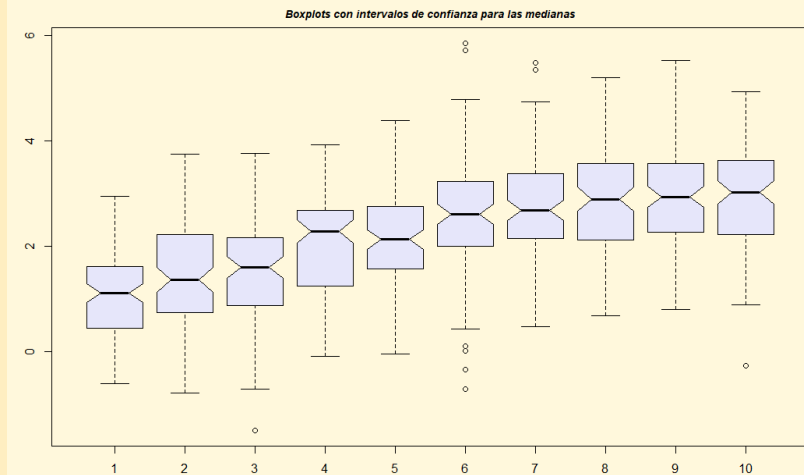
Gráficos con R



Ejemplo. Boxplots.

```
par(bg="cornsilk")
n <- 10
g <- gl(n, 100, n*100)
x <- rnorm(n*100) + sqrt(as.numeric(g))
boxplot(split(x,g), col="lavender", notch=TRUE)
title(main="Boxplots con intervalos de confianza para las medianas",
      xlab="Grupo", font.main=4, font.lab=1, cex.main=.9)
```

Gráficos con R

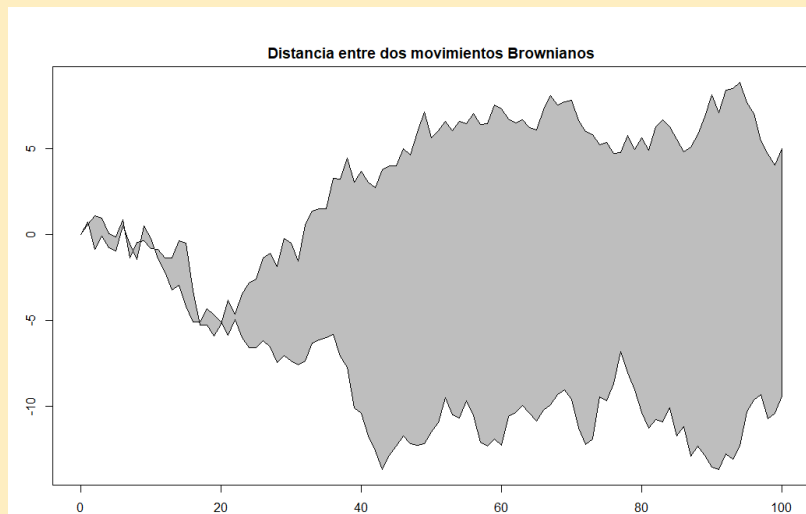


Gráficos con R

Ejemplo. Área sombreada entre dos gráficas.

```
par(bg="white")
n <- 100
x <- c(0,cumsum(rnorm(n)))
y <- c(0,cumsum(rnorm(n)))
xx <- c(0:n, n:0)
yy <- c(x, rev(y))
plot(xx, yy, type="n", xlab="Tiempo", ylab="Distancia")
polygon(xx, yy, col="gray")
title("Distancia entre dos movimientos Brownianos")
```

Gráficos con R

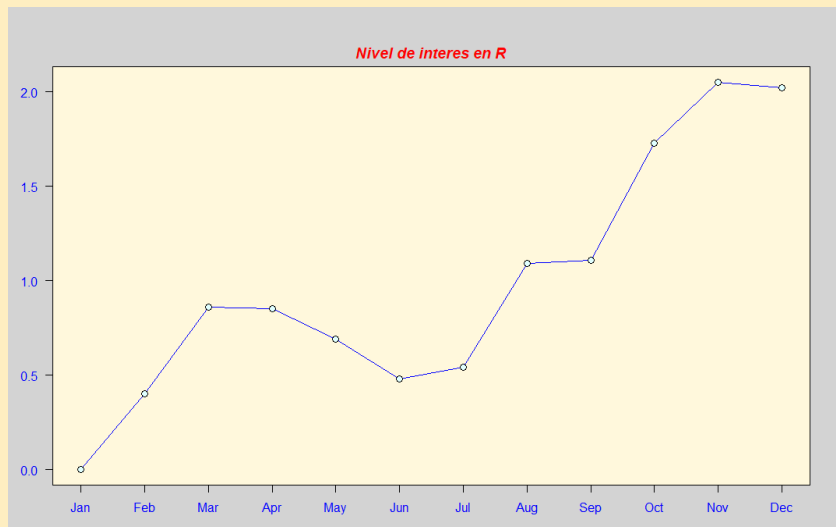


Gráficos con R

Ejemplo. Gráficas tipo Excel, o algo parecido.

```
x <- c(0.00, 0.40, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11,
      1.73, 2.05, 2.02)
par(bg="lightgray")
plot(x, type="n", axes=FALSE, ann=FALSE)
# c(x1,x2,y1,y2) coordenadas de region de graficación
usr <- par("usr")
rect(usr[1],usr[3],usr[2],usr[4], col="cornsilk", border="black")
lines(x, col="blue")
points(x, pch=21, bg="lightcyan", cex=1.25)
axis(2, col.axis="blue", las=1)
axis(1, at=1:12, lab=month.abb, col.axis="blue")
title(main= "Nivel de interes en R", font.main=4, col.main="red")
title(xlab= "1996", col.lab="red")
```


Gráficos con R



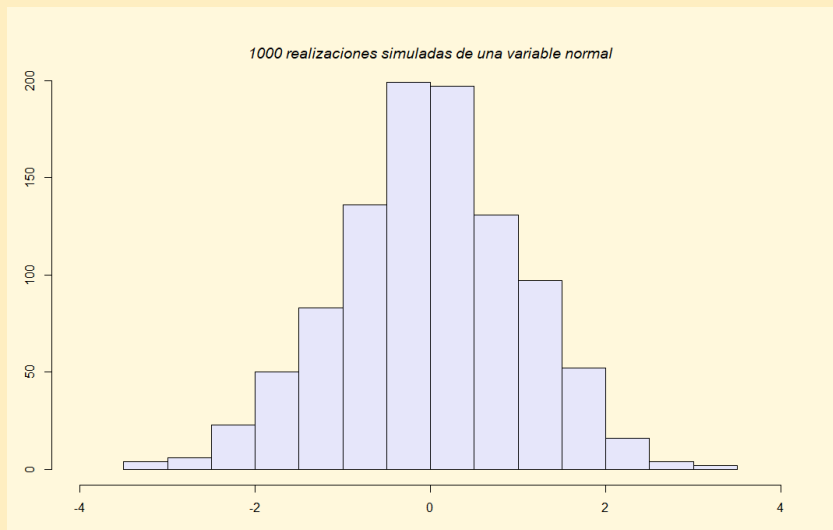
Gráficos con R



Ejemplo. Histograma.

```
par(bg="cornsilk")
x <- rnorm(1000)
hist(x, xlim=range(-4, 4, x), col="lavender", main="",
      ylab="Frecuencia")
title(main="1000 realizaciones simuladas de una variable normal",
      font.main=3)
```

Gráficos con R



Gráficos con R

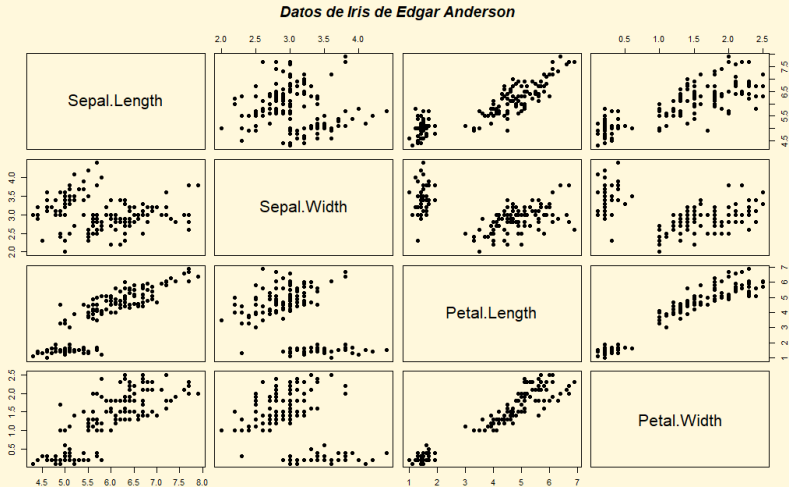


Ejemplo. Gráfica por parejas.

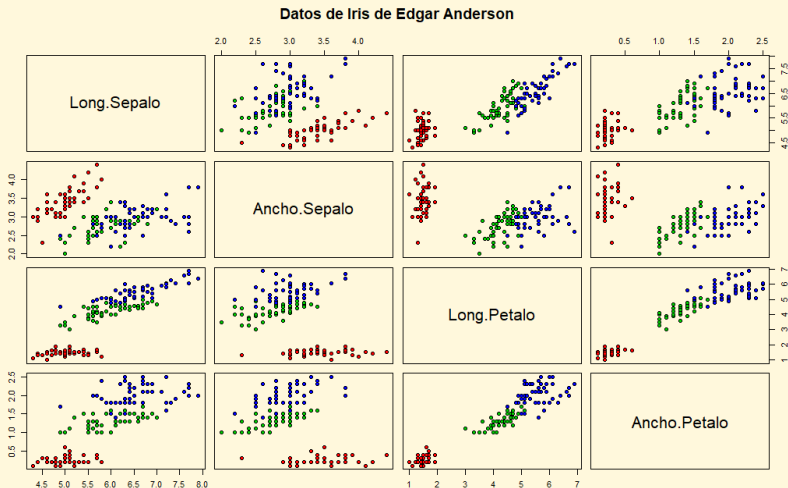
```
# Unicolor
pairs(iris[1:4], main="Datos de Iris de Edgar Anderson",
      font.main=4, pch=19)

#--
# Colores diferentes para cada especie.
aa <- iris
names(aa) <- c("Long.Sepalo", "Ancho.Sepalo",
               "Long.Petalo", "Ancho.Petalo",
               "Especie")
pairs(aa[1:4], main="Datos de Iris de Edgar Anderson", pch=21,
      bg = c("red", "green3", "blue")[unclass(iris$Species)])
```

Gráficos con R



Gráficos con R



Gráficos con R



Ejemplo. Gráfica de contornos

```
# volcano es la matriz 87 x 61 de elevaciones del volcán
# Maunga Whau en NZ
x <- 10*1:nrow(volcano)
y <- 10*1:ncol(volcano)
lev <- pretty(range(volcano), 10)
par(bg = "lightcyan")
pin <- par("pin")
xdelta <- diff(range(x))
ydelta <- diff(range(y))
xscale <- pin[1]/xdelta
yscale <- pin[2]/ydelta
scale <- min(xscale, yscale)
xadd <- 0.5*(pin[1]/scale - xdelta)
yadd <- 0.5*(pin[2]/scale - ydelta)
plot(numeric(0), numeric(0),
     xlim = range(x)+c(-1,1)*xadd, ylim = range(y)+c(-1,1)*yadd,
     type = "n", ann = FALSE)
```

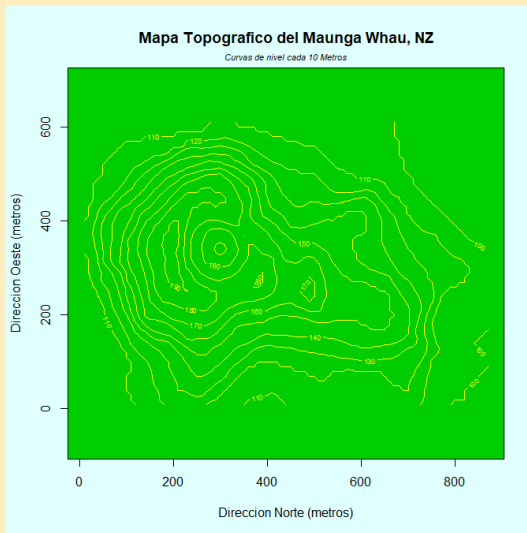
Gráficos con R



Ejemplo. Gráfica de contornos

```
usr <- par("usr")
rect(usr[1], usr[3], usr[2], usr[4], col="green3")
contour(x, y, volcano, levels = lev, col="yellow", lty="solid",
        add=TRUE)
title("Mapa Topografico del Maunga Whau, NZ", font= 4)
title(xlab = "Direccion Norte (metros)",
      ylab = "Direccion Oeste (metros)", font= 3)
mtext("Curvas de nivel cada 10 Metros", side=3, line=0.35,
      outer=FALSE, at = mean(par("usr")[1:2]), cex=0.7,
      font=3)
```


Gráficos con R



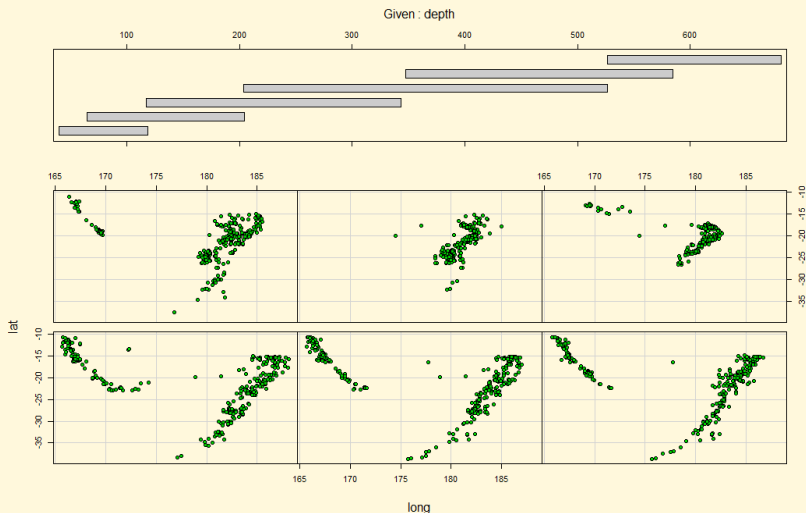
Gráficos con R



Ejemplo. Graficas condicionales.

```
# El conjunto de datos quakes es un data frame con 1000
# observaciones en 5 variables:
# lat = Latitud del evento
# long = Longitud
# depth = Profundidad (km)
# mag = Magnitud en escala de Richter
# stations = Numero de estaciones reportando el evento
par(bg="cornsilk")
coplot(lat ~ long|depth, data = quakes, pch = 21, bg = "green3")
```

Gráficos con R



Gráficos con R

Gráficos con R

A continuación presentamos algunas figuras interesantes, cuyo código en R es un poco más elaborado, aunque las funciones gráficas son los que ya hemos usado.

Gráficos con R



Ejemplo. Espiral de Ulam.

```
# prim = Un programa para calcular los primeros n primos
prim <- function(n){
  if(n==1){return(2)}
  primos <- 2
  notyet <- TRUE
  probar <- 2
  while(notyet){
    probar <- probar + 1
    pritst <- primos[ primos<=sqrt(probar) ]
    aa <- (probar %% pritst)
    if( any(aa==0) ){next}
    primos <- c(primos,probar)
    if( length(primos)==floor(n) ){ return(primos) }
  }
}
```

Gráficos con R



Ejemplo. Espiral de Ulam.

```
m <- 100
pp <- prim( (m+1)^2 )
ii <- seq(3,m+1,by=2)
jj <- length(ii)
par(mar=c(0,0,0,0)+1); xlim <- c(1,m+1)
plot(1,1,xlim=xlim, ylim=xylim, type="n", xaxt="n", yaxt="n",
     bty="n", xlab="", ylab="")
aa <- c(floor(m/2)+1,floor(m/2)+1)
for(k in 1:jj){
  r <- ii[k]
  co <- cbind(c(rep(r,r), (r-1):2, rep(1,r), 2:(r-1)),
             c(r:1, rep(1,r-2), 1:r, rep(r,r-2)))
  co <- co + (jj-k)
  n <- dim(co)[1]
  uu <- (r^2):((r-2)^2)
  rr <- is.element(uu[-(n+1)],pp)
  bb <- co[n,]
  segments(aa[1], aa[2], bb[1], bb[2], col="black", lwd=1)
  aa <- co[1,]
```

Gráficos con R

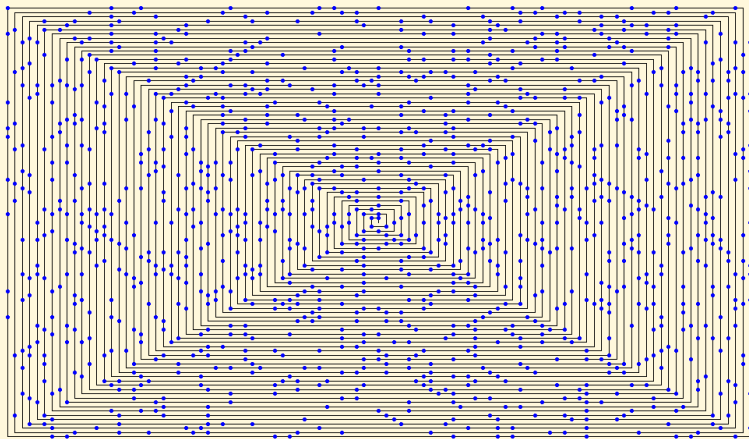


Ejemplo. Espiral de Ulam.

```
for(i in 1:(n-1)){  
  segments(co[i,1], co[i,2], co[i+1,1], co[i+1,2],  
           col="black", lwd=1)  
}  
points(co[rr,1], co[rr,2], col="blue", pch=20)  
}  
title("Espiral de Ulam", cex=.9, line=-.3)
```

Gráficos con R

Espiral de Ulam



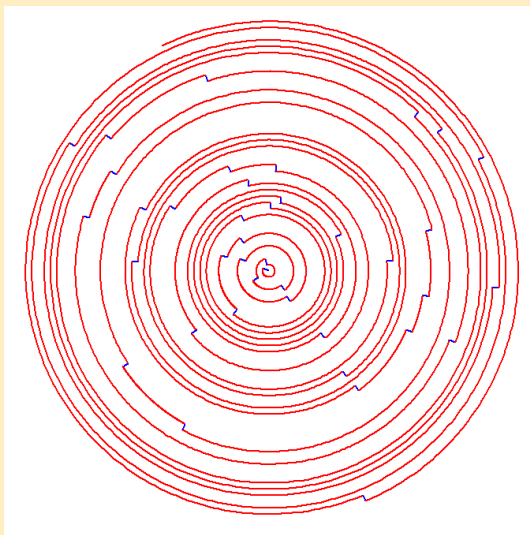
Gráficos con R



Ejemplo. Laberinto circular

```
M <- 40; m <- 120; n <- M; xylim <- .95*c(-M,M)
par(mar=c(0,0,0,0)+.6)
plot(0,0, type="n", xlim=xylim, ylim=xylim, xaxt="n", yaxt="n",
      xlab="", ylab="", bty="n")
pp <- c(0,0)
tet1 <- runif(1,min=0,max=2*pi)
for( r in 1:n ){
  qq <- r*c(cos(tet1),sin(tet1))
  segments(pp[1],pp[2],qq[1],qq[2], col="blue",lwd=2)
  tet2 <- tet1 + runif(1,min=0,max=2*pi)
  ts <- seq(tet1,tet2,length=200)
  nc <- r*cbind( cos(ts), sin(ts) )
  lines( nc[,1], nc[,2], col="red",lwd=2 )
  tet1 <- tet2
  pp <- nc[200,]
}
```

Gráficos con R



Gráficos con R



Ejemplo. El copo de nieve de Koch

```
# En este ejemplo tenemos funciones anidadas
KochSnowflakeExample <- function(){ # Función general
  iterate <- function(T,i){ # Primera función anidada
    A = T[,1]; B=T[,2]; C = T[,3];
    if (i == 1){
      d = (A + B)/2; h = (C-d); d = d-(1/3)*h;
      e = (2/3)*B + (1/3)*A; f = (1/3)*B + (2/3)*A;
    }
    if (i == 2){
      d = B; e = (2/3)*B + (1/3)*C; f = (2/3)*B + (1/3)*A;
    }
    if (i == 3){
      d = (B + C)/2; h = (A-d); d = d-(1/3)*h;
      e = (2/3)*C + (1/3)*B; f = (1/3)*C + (2/3)*B;
    }
    if (i == 4){
      d = C; e = (2/3)*C + (1/3)*A; f = (2/3)*C + (1/3)*B;
    }
  }
}
```

Gráficos con R



Ejemplo. El copo de nieve de Koch

```

if (i == 5){
  d = (A + C)/2; h = (B-d); d = d-(1/3)*h;
  e = (2/3)*A + (1/3)*C; f = (1/3)*A + (2/3)*C;
}
if (i == 6){
  d = A; e = (2/3)*A + (1/3)*C; f = (2/3)*A + (1/3)*B;
}
if (i == 0){
  d = A; e = B; f = C;
}

  Tnew = cbind(d,e,f)
  return(Tnew); # Devuelve un triángulo más pequeño.
}

# Segunda función anidada
draw <- function(T, col=rgb(0,0,0),border=rgb(0,0,0)){
  polygon(T[1,],T[2,], col=col, border=border)
}

```

Gráficos con R



Ejemplo. El copo de nieve de Koch

```
# Tercera función anidada
Iterate = function(T,v,col=rgb(0,0,0),border=rgb(0,0,0)){
  for (i in v) T = iterate(T,i);
  draw(T, col=col, border=border);
}

# Los vértices del triángulo inicial:
A = matrix(c(1,0),2,1);
B = matrix(c(cos(2*pi/3), sin(2*pi/3)),2,1);
C = matrix(c(cos(2*pi/3),-sin(2*pi/3)),2,1);
T0 = cbind(A,B,C);

plot(numeric(0), xlim=c(-1.1,1.1), ylim=c(-1.1,1.1),
     axes=FALSE, frame=FALSE, ann=FALSE);
par(mar=c(0,0,0,0),bg=rgb(1,1,1));
par(usr=c(-1.1,1.1,-1.1,1.1));
```

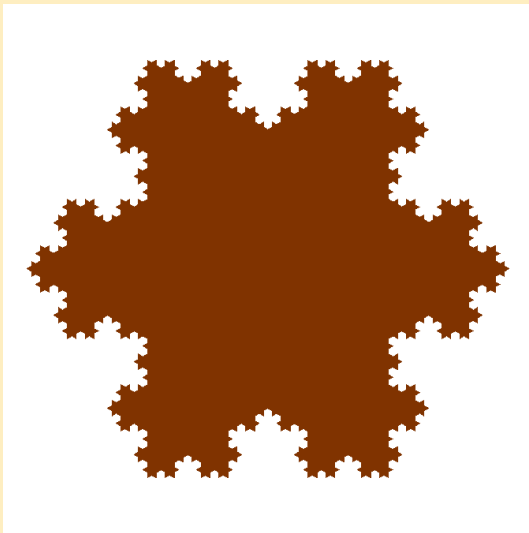
Gráficos con R



Ejemplo. El copo de nieve de Koch

```
# Dibujar copo de nieve:
for (i in 0:6) for (j in 0:6) for (k in 0:6) for (l in 0:6)
Iterate(T0,c(i,j,k,l));
}
# Ejecutamos la función
KochSnowflakeExample()
```

Gráficos con R



Índice

- 1 Nociones básicas de R
 - R Studio
 - La consola de R
- 2 Tipos de objetos en R
 - Vectores
 - Matrices y arreglos
 - Listas y factores
 - Data frame
- 3 Lectura/Escritura de datos
 - Leer datos de un archivo
 - Guardar datos
- 4 Como programar en R
- 5 Gráficos con R
- 6 **Resumen**

Resumen de comandos y operadores

Instrucciones en R:

| Operador | Símbolo | Operador/ Función | Símbolo/ Instrucción | Operador/ Función | Símbolo/ Instrucción |
|-----------------|---------|----------------------|-------------------------|----------------------|-------------------------|
| Igualdad | == | Suma | + | Tangente | tan |
| Diferente | != | Resta | - | Máximo | max |
| Menor | < | Multiplicación | * | Mínimo | min |
| Menor igual | <= | División | / | Rango | range |
| Mayor | > | Módulo | %% | Longitud | length |
| Mayor igual | >= | División entera | %/% | Sumatoria | sum |
| "y" lógico | & | Raíz cuadrada | sqrt | Producto | prod |
| "o" lógico | | Log. neperiano | log | Media | mean |
| Negación lógica | ! | Exponencial | exp | Dev. estándar | sd |
| | | Seno | sin | Varianza | var |
| | | Coseno | cos | Cuantiles | quantile |

Resumen de comandos y operadores

Algunas instrucciones:

| Operador | Símbolo |
|--------------------|-------------------|
| Núm. filas | <code>nrow</code> |
| Núm. columnas | <code>ncol</code> |
| Producto matricial | <code>%*%</code> |
| Transpuesta | <code>t</code> |
| Diagonal | <code>diag</code> |

Resumen de comandos y operadores

Más instrucciones

| Función | Operador | Función | Operador |
|-----------------|----------------------------|-----------------|----------------------------|
| Como caracter | <code>as.character</code> | ¿Es caracter? | <code>is.character</code> |
| Como entero | <code>as.integer</code> | ¿Es entero? | <code>is.integer</code> |
| Como complejo | <code>as.complex</code> | ¿Es complejo? | <code>is.complex</code> |
| Como numérico | <code>as.numeric</code> | ¿Es numérico? | <code>is.numeric</code> |
| Como lógico | <code>as.logical</code> | ¿Es lógico? | <code>is.logical</code> |
| Como factor | <code>as.factor</code> | ¿Es factor? | <code>is.factor</code> |
| Como data.frame | <code>as.data.frame</code> | ¿Es data.frame? | <code>is.data.frame</code> |
| Como matriz | <code>as.matrix</code> | ¿Es matriz? | <code>is.matrix</code> |
| Como tabla | <code>as.table</code> | ¿Es tabla? | <code>is.table</code> |
| Como vector | <code>as.vector</code> | ¿Es vector? | <code>is.vector</code> |
| Como lista | <code>as.list</code> | ¿Es lista? | <code>is.list</code> |