Estadística climatológica con R

Vectores

Los vectores son conjuntos de elementos del mismo tipo o modo. Es el tipo de objeto más utilizado en R, especialmente como argumento de funciones. Los vectores no tienen que ser numéricos necesariamente, pero ciertas funciones como ***sqrt()***, etc., sobre un vector no funcionaran entonces al no ser numéricos.

La manera más sencilla de formar un vector es:

```
> x <- c(1, 2, 3, 4, 5)  # definimos el vector x con numeros
> x <- c('rojo','azul','verde')  # definimos el vector x con cadenas de texto
> x <- 1:10   # secuencia desde el 1 al 10
> x <- seq(1,10)   # secuencia desde el 1 al 10
> x <- seq(10,1,-1)    # secuencia desde el 10 al 1 decreciente
> x <- seq(1,10,0.5)   # secuencia desde el 1 al 10, en incrementos de 0.5
> x <- seq(1,10,len=100)  # secuencia de 1 a 10, dividida en 100 elementos
> x <- rep(1,10)  # vector con 10 elementos iguales a 1.
> x<- c(seq(1,20),rep(1,12))  # secuencia del 1 al 20 y repetición de 1 12 veces.
```

Selección de elementos de un vector:

```
> x <- c(1, 2, 3, 4, 5)  # definimos el vector x
> x[5]   #  muestra el elemento 5
> x[3:5]    # muestra los elementos del 3 al 5
> x[c(3,5)]   # muestra los elementos de la posición 3 y 5
> x[x>3]  # mostrará solo los elementos mayores que 3
> x[x>2 | x <4]  #  muestra solo los elementos mayores que 2 o menores que 4
> x[x>2 &  x <4]  #  muestra solo los elementos mayores que 2 y menores que 4
> x[x!=3]  # mostrará los elementos distintos que 3
> sqrt(x)   # nos devuelve el vector de las raíces cuadradas de cada elemento
            # se aplica la función a cada uno de los elementos
```

Los vectores y otros objetos de R tiene un argumento ***length*** que da el numero de elementos del vector.

```
> length(x)
```

Ejercicios 1.1

- Hallar el seno, logaritmo decimal y neperiano, raíz cuadrada y el valor absoluto de un numero cualquiera.
- Ver el numero ***e*** por medio de ***exp()***
- Formar un vector ***x*** con elementos que vayan de 2 al 20, del 100 al 140, de 2 en 2 y del 45 al 35 en orden descendente.
- Mostrar en pantalla el elemento 5 y el 10 del vector ***x***.
- Ordenar este vector en orden decreciente y obtener su longitud.
- Obtener otro vector ***y*** con los elementos de ***x*** comprendidos entre 20 y 120, ambos incluidos.
- Calcular la suma y el producto de todos los elementos del vector ***x***
- Calcular el máximo, mínimo, media y cuasivarianza del vector ***x***.

Listas y Matrices (Ver anexo)

Dataframes

Una ***hoja*** o ***estructura de datos*** o ***marco de datos*** es una lista de un tipo particular llamado **data.frame**, con las siguientes particularidades:

- Los componentes tienes que ser vectores, aunque estos pueden ser numéricos, de caracteres, etc. y todos tienen que tener la misma longitud. También pueden estar compuestos por matrices, listas u otras hojas de datos.

Veremos la forma de construcción de marcos de datos. Si por ejemplo tenemos tres vectores:

```
> facturas<-c(3,4,5,7); recibos<-c(23,45,50,60) ; nombres <- c('Pepe', 'Juan', 'Maria', 'Rosa')
# (tienen que tener el mismo número de elementos)
> marco <- data.frame(fac=facturas, rec=recibos, nom=nombres) # creamos el marco de datos
> marco  # para visualizarlo
```

Se pueden listar como las listas, haciendo referencia siempre a las columnas:

```
> marco[3]  # columna 3 (nom)
> marco$nom  # Otra forma, por la cabecera de la columna

> marco$nom[2]  # Segundo elemento de la columna nom
> marco[[3]][2]  # Otra forma, segundo elemento de la columna 3 (nom)
```

Veremos una aplicación muy importante de los marcos de datos en la lectura y escritura de ficheros.

Leyendo datos desde archivos

R toma los datos, los ficheros y los scripts del directorio de trabajo. Cuando leamos o escribamos un fichero a no ser que le pongamos la ruta completa lo buscara en el directorio de trabajo.

Como ya sabemos el directorio de trabajo (***wd***) actual lo obtenemos con: ***getwd()***. El directorio de trabajo se puede cambiar desde la mismo GUI o de la forma

```
> setwd('C:/Curso_Estadistica')
```

Si tenemos puesto el directorio que vamos a usar como *wd* en el icono de *R* en Windows rellenando: (*iniciar en:* ***C:/Curso_Estadistica***) cuando empecemos ***R*** ya nos colocará directamente este directorio de trabajo. En Linux es un poco más complicado.

Para este curso vamos a necesitar unos ficheros. Antes de empezar es mejor descargarse la carpeta ***material.zip*** y copiarla y descomprimirla en el directorio de trabajo ***C:/Curso_Estadistica.***

R puede leer datos de archivos de texto (ASCII). También puede leer archivos en otros formatos como *Excel*, *csv* y otros muchos más, acceder a base de datos *SQL*, aunque a veces es necesario instalarse paquetes que no están incluidos en el base. Nos centraremos en este apartado en leer y escribir ficheros de texto.

R puede leer datos de archivos de texto (ASCII) con las funciones **read.table, scan** o **read.fwf**.

Cuando trabajemos con ficheros de texto que vengan estructurados en columnas de datos nos vendrá muy bien la función ***read.table***. Esta función lee los datos y los convierte a un ***data.frame***.

Al leer un fichero por ejemplo: ***datos.txt*** de la forma:

indicativo	lat	lon	temperatura	presion	humedad
08221	40.46	-3.56	18	1015.1	60
08481	36.73	-4.48	23	1016.2	72
08001	43.36	-8.41	15	1012.1	88

……

datos < -read.table('datos.txt' , header = TRUE)

si tecleamos ***datos*** R escribirá el fichero completo, pero además los datos de cada columna los asigna a un vector. Estos vectores pueden tener ya un nombre que toma del mismo fichero como indicativo, lat, etc., si hemos indicado: ***header=TRUE*** o sino por defecto toman los nombres *V1*, *V2*, etc. Así si escribimos ***datos$indicativo*** R listaria:

08221 08481 ….

Si no hubiésemos escrito la primera línea con los nombres (***header=FALSE***) habría que nombrarlos como ***datos$V1.*** Un elemento dentro de este vector se llamaría: ***datos$V[3]*** por ejemplo para el elemento 3.

Siempre nos podremos referir a los elementos como ***datos[1]*** al vector o columna 1, y como ***datos[[2]][1]*** al primer elemento de la variable o columna 2.

La función ***read.table*** tiene muchas opciones. La forma general con las opciones más importantes es:

read.table(file= 'nombre_fichero', header = FALSE, sep ="" , dec = ".",skip = 0, fill = FALSE, comment.char = "#")

> ***file*** es el nombre del archivo: entre comillas o una variable que contiene el nombre.
> ***Header=FALSE*** Si es TRUE toma la primera línea como nombres de variables.
> ***sep=*** es el separador entre datos, normalmente es “” para espacios en blanco o “\t” tabuladores.
> ***dec=”.”*** Indica a *R* el formato de los decimales en el fichero punto (defecto), pero se puede poner coma si por ejemplo vienen de una hoja excel.
> ***skip = 0*** Se salta las primeras *n* líneas del fichero, las ignora.
> ***fill*** si es TRUE rellena los espacios en blanco que encuentre.
> ***comment.char = "#"*** ignora las líneas que comiencen con #.

Hay unas variantes de ***read.table*** pensadas para ficheros ***csv*** que son prácticamente iguales pero con unas opciones definidas por defecto:
read.csv(file, header = TRUE, sep = ",", quote="\, dec=".",fill = TRUE, ...)
read.csv2(file, header = TRUE, sep = ";", quote="\, dec=",",fill = TRUE, ...)

La function ***read.csv*** considera por defecto el separador entre datos como ‘**,**’ mientras que la ***read.csv2*** toma como separador el ‘**;**’. De cualquier forma todo esto se puede configurar en la function ***read.table***.

La función ***scan*** es más flexible que ***read.table***. A diferencia de esta última es posible especificar el modo de las variables:
> datos <- scan("data.dat", what = list("", 0, 0))

En este ejemplo ***scan*** lee tres variables del archivo ***data.dat***; el primero es un carácter (***""***) y los siguientes dos son numéricos (***0***). Por defecto, es decir si se omite el argumento ***what***, ***scan()*** crea un vector numérico. Si ***scan*** no lee el tipo de datos esperados da un mensaje de error. Scan no tiene la opción head por lo que si tiene cabecera habría que saltarla al leer con ***skip=1***. Nos referiremos a los elementos como ***datos[[2]][5]*** para la segunda variable o columna y su elemento numero 5.

La forma con las opciones más importantes es:

scan(file= 'nombre_fichero', what = double(0), sep ="" , dec = ".",skip = 0, na.strings = "NA", fill = FALSE, blank, comment.char = "#")

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

Guardando datos a archivos

Se suelen usar *archivos de texto*, que son manejables desde cualquier editor.

Existen varias funciones para grabar los datos u objetos del workspace a ficheros.

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 o una matriz).

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*** que agrega los datos al archivo sin borrar datos ya existentes (***TRUE***) o borra cualquier dato que existe en el archivo (***FALSE***, por defecto).

Otra función, muy útil, para volcar datos a ficheros es ***write.table***. La función ***write.table*** guarda el contenido de un objeto en un archivo, pero lo normal es que guardamos objetos de tipo ***data.frame***.

Las opciones más importantes son:
write.table(x, file = 'nombre_fichero', append = FALSE, sep = " ", na = "NA", dec = ".", quote = TRUE, row.names = TRUE,col.names = TRUE)

> ***x*** es el nombre del objeto a exportar (*dataframe*)
> ***file*** es el fichero donde vamos a escribir
> ***append = FALSE*** significa que borramos el fichero si existiese previamente antes de escribir.
> ***sep*** = " " el separador entre los datos. Normalmente será blanco, coma, punto y coma o tabulador "\t"
> ***row.names/col.names*** pone el nombre o numero de las columnas
> ***quote=TRUE*** escribe los cadenas de caracteres con comillas.

Ejercicios 1.2

- Leer el fichero de ejemplo: '***ESTACIONES_radiacion.dat'*** con ***read.table*** asignándosela a una nueva variable ***datos***, sin la opción ***header=FALSE*** (por defecto)
- Listar el *dataframe* ***datos*** entero, despues la *segunda columna*, y luego el *tercer elemento* de esa columna.
- Haced lo mismo, leyendo primero el fichero con ***read.table***, pero añadiendo la opción ***header=TRUE***).
- Listar el *dataframe* ***datos***, después la *segunda columna*, y luego el *tercer elemento* de esa columna y observar las diferencias con el punto anterior.
- Escribir el *dataframe* ***datos*** leída anteriormente con ***write.table*** en un fichero llamado ***salida.txt***, desactivando las comillas en los caracteres y omitiendo nombres de columnas y filas ***(row.names = FALSE, col.names=FALSE, quote=FALSE).*** Ver el fichero creado con un editor de texto.
- Hacer lo mismo pero escribiendo solo la columna ***NOMBRE*** en el fichero.

R desde scripts

Para ciertas tareas sencillas con R basta con introducir órdenes desde la pantalla de comandos, pero cuando empezamos a hacer labores más complicadas, repetitivas o que necesitan introducir muchos comandos de R se hace imprescindible el trabajar a través de *programas* o *scripts*. Estos son simplemente ficheros de texto donde se escriben todas las órdenes que pensamos ejecutar, datos y objetos. Luego al final se ejecutan todas las órdenes del script secuencialmente.

Además de permitir trabajos repetitivos o extensos los scripts permiten trabajar con el *lenguaje de programación de R*, donde se disponen de *if* condicionales, bucles, definición de funciones y otras funcionalidades de cualquier lenguaje informático. Solo vamos a ver algunas cosas, pero como R está muy bien documentado se dispone de abundante información.

Los scripts tienen la extensión **.R** para que se interpreten correctamente tanto por R como por otros programas externos. Si necesitamos cargar algún librería se hace al principio con ***library('nombre_libreria')***.

Lógicamente los scripts tiene que estar en el directorio de trabajo o sino poner la ruta completa en el ***source().***

Para escribir un script nuevo desde el *GUI* en el menú: *Archivo/nuevo script* y luego *salvar como* donde se le pone el nombre que queramos. Una vez que esté abierto en el menú *Ventanas: dividir horizontalmente*. De forma que tendremos el script arriba y la ventana de comandos abajo.

Se puede *ejecutar* el script entero desde el menú *Editar/Ejecutar todo* o en la ventana de comandos ***source('nombre_script.R')***. Hay que salvarlo previamente con ***CTRL+S*** o *Archivo/guardar*)

Si solo se quiere ejecutar una línea del script donde tengamos el cursor ***CTRL+R*** o ***F5***.

Otra forma de trabajar es crear un fichero en la carpeta de trabajo con un editor de texto que reconozca el lenguaje datosde programación R como el ***notepad++,*** salvarlo y ejecutarlo desde líneas de comandos con ***source()***

Ejemplos de scripts en R

Abrir en el *GUI* un script nuevo y copiar primero el script del ejemplo 1 que viene a continuación y ejecutarlo con:

```
> source('prueba1.R')
```

o desde el *GUI*. Después hacer lo mismo con el script del Ejemplo 2.

Ejemplo1:

```
# Script de prueba: prueba1.R

# va a mostrar el directorio actual
a <- getwd()
print('El directorio actual es:')
print(a)

# va a mostrar los ficheros dentro del directorio
print(paste('Los directorios dentro de ',a,' son:'))
print(dir())

# paste() sirve para unir tanto cadenas de texto (character) y variables de cualquier tipo.
```

```
# van separadas por comas. Deja un espacio en blanco entre cadenas a no ser que se ponga al final  ,sep=' '
```

Ejemplo 2:

```
# Script de prueba: prueba2.R

# Define unas variables
n_inicial = 9
n_final = 8

if ( n_final < n_inicial){
  cat (' No se puede hacer el bucle, el num. final es menor que el inicial \n')
  stop(' El programa acabo mal')
}

# hace un bucle desde n_inicial a n_final
for (numero in n_inicial:n_final){
  # escribe el cuadrado del numero
  print(paste('El cuadrado de ',numero,' es ',numero*numero, sep=''))
}
```

Ejercicio 1.4

- Vamos a crear un script de prueba, ***prueba.R*** que cree un vector ***a*** de 7 números (reales o enteros) y primero que lo visualice en pantalla, después que calcule la media de los elementos del vector y los muestre en pantalla (con la explicación de lo que está mostrando). Por último que calcule el valor mínimo del vector y lo visualice.

Opcional: Mediante un bucle ***for*** calcular y mostrar la suma de los cuadrados de todos los elementos del vector.

ANEXO

Listas

Es una colección ordenada de objetos que no tienen porque ser del mismo tipo, así una Lista puede estar compuesta de un vector numérico, un valor lógico, una matriz y una función. Se definen con la función ***list***

```
Milista<-list(jefe='Juan',
encargado='Pedro',num.trabajadores=4,edad.trabajadores=c(21,34,48,50))
```

En este caso **Milista** tiene 4 componentes, referidos como **Milista[[1]]** etc. Como ademas el elemento 4 es un vector nos referiremos a cada componente como **Milista[[1]][n]** donde n es el numero de componente.

La función **length()** aplicada a una lista devuelve el número de componentes (del primer nivel) de la lista.

Los componentes de la lista pueden tener nombre y en ese caso nos podemos también referir a ellos por su nombre. En el ejemplo anterior la componente **Milista$jefe** es igual que **Milista[[1]]** y vale "Juan". Si no se incluye el nombre, los componentes estarán automáticamente numerados. No hace falta referirnos con el nombre completo, basta con solo decir las primeras letras de forma que no haya confusión con los otros componentes. Por ejemplo podemos decir **Milista$enc** para **Milista$encargado**

También es posible utilizar los nombres de los componentes entre dobles corchetes, por ejemplo, **Lst[["jefe"]]** coincide con **Lst$jefe**. Esta opción es muy útil en el caso en que el nombre de los componentes se almacena en otra variable, como en

x <- "jefe"; Lst[[x]]

Matrices

Una matriz (*matrix*) es una colección de datos del mismo tipo de dos dimensiones. (Los arrays son matrices, pero con más de dos dimensiones).

La matriz se crea con la función **matrix**:

matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)

Si no se ponen datos a rellenar la completa con ***NA*** (valores not avalaibles). La opción **byrow =FALSE** (por defecto) indica que los valores van llenando la matriz por columnas.

```
> matrix(data=6, nrow=3, ncol=2)
     [,1] [,2]
[1,]  6    6
[2,]  6    6
[3,]  6    6

> c <-  matrix(1:6, 2, 3)
     [,1] [,2] [,3]
[1,]  1   3    5
[2,]  2   4    6
```

La dimensión de la matriz se obtiene con ***dim(c)***

Si tenemos dos o más vectores del mismo tipo, numéricos o de caracteres, y tienen la misma longitud se pueden combinar para formar una matriz con ***cbind***

```
> x <- c(1,2,3,4); y <- c(5,6,7,8)
> c <- cbind(x,y)
```

cbind une los vectores por columnas. Existe la función ***rbind*** que los une por filas.

A los elementos de una matriz se accede de igual manera que los vectores pero con dos indices, el primero hace referencia a la fila y el segundo a la columna.

```
> c[1,2]
[1] 3
```

Si queremos acceder a una fila completa seria***:*** ***c[1,]*** y la columna ***c[,2]***

Operaciones con matrices:

Si creamos dos matrices, por ejemplo:

```
A <- matrix(data=c(2,0,1,3,0,0,5,1,1),nrow=3,ncol=3,byrow=T)
B <- matrix(data=c(1,0,1,1,2,1,1,1,0),nrow=3,ncol=3,byrow=T)
> A %*% B   # producto de matrices (dos matrices se dicen multiplicables si el número de
columnas de A coincide con el de filas de B, ver Nota abajo)
> A + B   # suma de matrices (cuando tienen las mismas dimensiones)
> t(A)   #  traspuesta de la matriz A
> det(A)   # determinante de la Matriz (solo matrices cuadradas)
> diag(A)  #  devuelve un vector con los elementos de la diagonal de la matriz A
```

Ejercicios Anexo.1

- Formar una matriz 5 por 6 con valores del 1 al 30 pero rellenándola por filas.
- Crear 2 matrices A y B y hallar su producto C. Mostrar después la dimensión, la transpuesta, su determinante y la matriz diagonal de C. Mostrar también el elemento de la fila 1,columna 2 de C y toda la fila 1 de C.

*<u>Nota:</u> se recuerda que dos matrices son multiplicables si el número de columnas de A coincide con el número de filas de B. (Anxm * Bpxn = Cpxm siendo n el numero de filas en A y m el num. de columnas en A y el elemento Cjxi se obtiene multiplicando la fila j de A por la columna i de B)*