CONTENTS

PREFACIO

El R puede ser considerado una implementación del lenguaje S, desarrollado en Bell Laboratories y que forma la base del producto comercial $S - PLUS$. Los primeros pasos de R se deben a Robert Gentleman y Ross Ihaka del Statistics Department de la University de Auckland, aunque las sucesivas versiones son controladas y desarrolladas por el R Development Core Team del que forman parte muchos colaboradores en todo el mundo. Realmente el objetivo actualmente es la formación de equipos de colaboradores dedicados a la utilización de R para distintas aplicaciones como el grupo de redes bayesianas gR, de visualización de datos multivariantes ggobi. y muchos más que aportan los diferentes paquetes que pueden cargarse gratuitamente desde la página origen de todo este proyecto de software libre www.r-project.org.

Hay una lista de correo para usuarios en español r-help-es, donde consultar y comentar incidencias en el uso de R.

En resumen se podría considerar que los paquetes R proporcionan una forma de manejar conjuntos de funciones o datos y sus documentación.

1. PRIMEROS PASOS

- R es un lenguaje y un entorno para efectuar cálculos estadísticos y análisis gráficos, de distribución libre que puede cargarse en www.r-project.org. Al igual que la versión comercial S-Plus, está basado en el lenguaje S. Es completamente programable, siendo posible automatizar los procedimientos repetitivos mediante *scripts* del usuario, también es fácil escribir las $funciones$ que se necesiten así como nuevos *paquetes*.

1.1 Zona de trabajo ($Workspace$)

- El $Workspace$ en R contiene los datos y procedimientos de un determinado trabajo o proyecto, así que después de arrancar con el fichero $RGui.exe$, es conveniente seleccionar un nuevo directorio de trabajo en la persiana $File/Change\ Directory$, ya que por defecto se posiciona en el directorio en el que se ha instalado R, solapando los sucesivos trabajos. Al cerrar R se preguntará si se quiere *"Save workspace image"*, diciendo *yes* se crearán dos ficheros en el directorio de trabajo: $.Rhistory$ con los comandos y sentencias generadas en la sesión y $.RData$ con los objetos almacenados pero si se le quiere dar un nombre es preferible guardarlo con ese nombre antes de cerrar en la persiana $File/Save\ Workspace$ y contestar *no* cuando aparece *"Save workspace image"*. La próxima vez con *doble-click* sobre ese fichero $.RData$ arrancará R en dicho espacio de trabajo cerrando con el procedimiento anterior. Se puede ver en que directorio se trabaja tecleando $getwd()$.

- Al arrancar R aparece la ventana de la $RConsole$ de trabajo donde se irán introduciendo los sucesivos comandos que se pueden guardar en un fichero $.Rhistory$. Si se obtienen gráficos aparecerán en otra ventana $RGraphics$ que pueden copiarse o guardarse en diferentes formatos así como imprimirse.

- R trabaja con "objetos", la mayoría de los cuales son funciones. Tecleando $objects()$ (o $ls()$)se obtiene la lista de los que se tienen almacenados. Estos objetos se guardan en el $Workspace$ que es un fichero $.RData$, y que por defecto se guarda en el *directorio de trabajo* en el que está instalado R.

- En R, como en la mayoría de los paquetes basados en UNIX, se consideran nombres diferentes los escritos con mayúsculas o minúsculas, así A y a representarán distintos símbolos. En general se pueden utilizar todos los símbolos alfanuméricos, aunque un nombre nunca podrá comenzar por un número.

- Los comandos en R pueden ser: *expresiones* como $plot(x, y)$ que serán evaluadas impresas y su valor perdido, *o asignaciones* utilizando $<-$ o $=$ como $x < -27$ que también serán evaluadas y su valor transmitido a una

variable, pero no automáticamente impreso. Los comandos estarán separados por ; o por una nueva línea, pudiendo agrupar comandos elementales entre { y }. Si un comando no está completo, el R devolverá el símbolo + hasta que se complete.

- Los comentarios en R se pueden poner casi en cualquier sitio, con tal de comenzar con # todo hasta el final de la línea es un comentario.

- Para cerrar la sesión de R se tiene que escribir $q()$ apareciendo la pregunta de si guardar o no "workspace". Si se contesta *yes* se guardará en $.RData$ y $R.History$ del directorio de trabajo.

1.2 Ayudas

R tiene una serie de documentos *html* (que arrancan con el navegador) para ayuda y manuales a los que se accede con $help.start()$. Si se conoce el nombre de la función, por ejemplo *nombre* se introduce $help\,(nombre)$ y en una ventana aparte aparecerá toda la información sobre su definición y características. Si no se conoce el nombre exacto de la función es preferible preguntar $help.search\,("nombre")$ y aparecerán los ficheros de ayuda con algún elemento que sea *nombre*. Se puede utilizar también $?help$ o $?help.search$ para obtener más información de estas ayudas. Para no tener que teclear lo mismo varias veces, con las teclas de movimiento, $\uparrow$ o $\downarrow$ se obtienen las sentencias introducidas con anterioridad.

Otra forma de obtener ayuda sobre nuestra función *nombre* es mediante ejemplos de ayuda tecleando $example(nombre)$. Como prueba escríbase $example\,(plot)$. En la mayoría de las funciones que se describirán, se deberá recurrir a $help(\ )$ para obtener información completa de las mismas.

1.3 Scripts

Son ficheros que contienen listas de comandos según se escriben en la consola, pudiendo incluir comentarios (líneas comenzando por #) que explican lo que se pretende hacer. Son muy útiles para repetir en sucesivas ocasiones, ciertos procedimientos que hemos tecleado en un determinado momento y para almacenar los comandos utlizados en las diferentes sesiones.

Ejemplo: *Para crear y hacer funcionar un script que genere dos muestras de tamaño 50 de una distribución* $N\,(2,3)$ *calculando su coeficiente de correlación.*

1. Arrancar R

2. En la persiana *File* seleccionar *New Script*, si se quiere utilizar un script almacenado seleccionar *Open Script*.

3. Escribir en el script las siguientes líneas

$x < -rnorm(50, 2, 3)$

$y < -rnorm(50, 2, 3)$

$plot(x, y)$

$print(cor.test\,(x, y))$

4. Seleccionar todas las líneas y presionar el botón derecho del ratón y aparecerá una ventana de diálogo en la que se selecciona "Run line or selection", así se ejecutarán todas las sentencias del *script*. Si no se selecciona nada, se ejecuta la sentencia al final de la cual está colocado el cursor. También se puede seleccionar un bloque de sentencias que son las que se ejecutarán.

5. Como salida aparecerá en una ventana *RGraphics*, una nube de puntos correspondiendo a las 50 observaciones pedidas y en en la ventana *RConsole* el valor del coeficiente de correlación muestral así como el correspondiente p-valor del contraste y los extremos del intervalo de confianza a nivel 0.95.

6 Para guardar y cerrar el script, cuando se está en la ventana del mismo abrir la persiana *File* y seleccionar *Save* guardándolo con la extensión *.R* y *Close script* para cerrarlo

1.4 Paquetes R

Los paquetes proporcionan las facilidades para manejar conjuntos de funciones o datos y la documentación correspondiente. Son destacables los siguientes aspectos:

- Se cargan y descargan ocupando memoria sólo cuando son utilizados.

- Se instalan y actualizan fácilmente. Un único comando, ejecutable dentro o fuera de R, coloca en su sitio las funciones datos y documentación.

- Se adapta a los usuarios o administradores pudiendo tener además una o más librerías privadas de paquetes.

- Se pueden validar ya que R posee comandos para verificar que la documentación existe, para eliminar errores comunes y para comprobar que los ejemplos funcionan realmente.

1.4.1 Estructura de los paquetes R

La estructura básica suele contener

- un fichero **descriptivo** del paquete, autor y condiciones de la licencia en formato texto
- un fichero **índice** con las lista de funciones y datos, que puede generarse automáticamente
- un subdirectorio man/ de ficheros de documentación
- un subdirectorio R/ de ficheros de códigos R
- un subdirectorio data/ de conjuntos de datos
- un subdirectorio src/ de fuentes *C, Fortran o C++*

de forma menos frecuente también contiene

- tests/ para contrastes de validación
- exec/ para otros ejecutables, por ejemplo en Java
- inst/ para otro material
- un "script" de configuración para comprobar otro software que se necesite o para manejar diferencias entre sistemas.

1.4.2 Formatos de bases de datos

El comando *data*() carga datos de los paquetes que se tienen cargados

- Ficheros de texto rectangulares con separador coma o espacio en blanco
- Código fuente S producido por la función *dump*() de R o $S - Plus$
- Ficheros R binarios que produce la función *save*()

El tipo de fichero se elige automáticamente según la extensión.

1.4.3 Carga opcional de paquetes

- En el *Help* menú se pueden encontrar los *Manuales* donde se describen las funciones que se cargan con los paquetes básicos que se instalan al arrancar el R. Hay otros paquetes suplementarios que no son más que grupos de funciones que se han escrito y hecho públicas por los creadores a través de la familia $CRAN$ de sitios en Internet (vía http://cran.r-project.org). En el menú *Packages* se pueden manejar estos otros paquetes, siendo algunos de los más utilizados el *mva* y $MASS$. También algunos se pueden cargar mediante *library*(*mva*) y *library*($MASS$) y con *library*() se pueden ver los paquetes disponibles para cargar directamente sin tener que acceder al sitio de R. Para saber cuáles se tienen cargados hay que teclear *search*().

Dentro del menú *Packages* se tienen las opciones de instalación o actualización de nuevos paquetes ya sea directamente del sitio $CRAN$ o desde un

fichero *.zip* que también se puede obtener en ese mismo sitio. Una vez instalado en nuestro directorio R podrá ser cargado con *Load Package* para trabajar con él en nuestra sesión.

Cabe señalar, aunque no sea de interés para la mayoría de los usuarios, que es posible acceder a los códigos fuente de los paquetes de R. No se distribuye de forma automática sino que hay que ir al sitio $CRAN$ y acceder a la opción R *Sources* para cargarla en nuestro sistema. A partir de ahí se puede visualizar el código de cualquier cálculo por ejemplo *cor.test* del paquete *statst* (para saber en qué paquete se encuentra hay que teclear ?*cor.test* y aparece en la primera línea de la ventana de $Help$), dentro del directorio R donde se guardó el código fuente, mediante el camino $src\backslash library\backslash stats\backslash R\backslash cor.test.R$.

1.5 Lenguaje R

Como se dijo al principio es un dialecto del lenguaje S, con una estructura común a muchos lenguajes tipo C, pero con unas características especiales que lo hacen especialmente versátil para el manejo de elementos estadísticos en concreto para operaciones con matrices y vectores. Fue diseñado en los 80 y desde entonces ha tenido un enorme desarrollo dentro de la comunidad estadística por sus posibilidades para la modelización estadística y la visualización mediante gráficos. Los comandos elementales en R consisten en **expresiones de cálculo y asignaciones** que pueden separarse por punto y coma ; o por línea nueva y agruparse en una expresión compuesta mediante llaves { }. Si un comando se escribe incompleto por error, R suele devolver + hasta que se completa.

Ejemplo: *Falta el paréntesis final*

$> (media.x < -mean(x)$

+

+

+)

[1]2.206341

Como se vio al utilizar los *scripts*, es posible utilizar comandos almacenados en un fichero externo mediante el procedimiento indicado en dicho apartado o mediante el comando *source*. Análogamente es posible almacenar todas las salidas de R en un fichero externo, por ejemplo $result1.lis$ mediante $sink($"*ubicación del fichero* $result1.lis$"$)$, que se podrá leer con cualquier procesador de textos. Para ver las salidas en la consola de nuevo se tecleará el comando $sink()$.

1.5.1 Cálculo

Si se plantea un cálculo en la línea de comandos, R efectuará el mismo, apareciendo el resultado en la consola.

Ejemplo: *Para calcular* $\exp(2) + 5$

$> exp(2) + 5$

obteniéndose

[1]12.38906

Los cálculos se efectuarán en la forma habitual, pudiendo utilizar los paréntesis para alterar el orden de los cálculos.

Ejemplo:

$> 7 - 5 * 4 + 3$

$[1]\ -10$

o bien

$> (7 - 5) * 4 + 3$

[1] 11

Los espacios se pueden utilizar libremente sin alterar el significado de las operaciones.

1.5.2 Asignación

Como se ha visto en el primer ejemplo, una forma de crear nuevos objetos es mediante el *operador de asignación* $<-$ es decir los símbolos "menor que" y "menos", tecleados uno a continuación del otro.

Ejemplo: *Para crear el objeto* x *como una muestra de tamaño 50 de una distribución* $N(2, 3)$

$> x <- rnorm(50, 2, 3)$

Si a continuación se teclea

$> x$

se obtendrá la descripción explícita de las 50 observaciones generadas en ese momento

[1] 1.9695542 2.0773641 3.9620694 -0.5074673 6.0494055 3.5239660
[7] 4.0615103 -0.2137977 -3.0360154 7.9673968 0.5715117 -0.4865489
[13] -0.7256490 -1.5236052 2.9982977 4.6363986 1.3246436 -0.1932609
[19] 0.4410943 10.4654333 1.2003788 3.0297676 -3.1974437 1.4925627
[25] 4.3648033 3.7473657 1.5032604 6.9660583 1.9510645 7.4039741
[31] 4.3971533 0.9633926 6.8757369 3.7359458 -1.3833869 1.2585773
[37] 1.7376304 -2.5316654 6.1148239 3.0794651 4.5315920 0.7024664
[43] 1.5024827 -1.5534974 -0.7111865 3.2873565 0.4772716 0.9519583
[49] 6.1283580 -1.0715138

y siempre que se utilice x en cualquier expresión, se estará considerando toda la muestra, así por ejemplo

$> media.x < -mean(x)$

permite calcular la media aritmética de las 50 observaciones y guardarla como objeto $media.x$

$> media.x$

[1] 2.206341

Para conseguir que se cree el objeto y se visualice se debe escribir entre paréntesis

$> (media.x < -mean(x))$

[1] 2.206341

Para **borrar objetos** de la zona de trabajo se utilizará el comando $rm(.)$

Ejemplo: *Para borrar* $media.x$

$> rm(media.x)$

Si se teclea

$> objects(\ )$

se observará la desaparición de $media.x$.

Dado que en la denominación de algunos objetos es frecuente la utilización de valores como $x, y, \ldots$, es recomendable abrir directorios de trabajo distintos al efectuar diferentes tareas en R porque las sucesivas creaciones de estos objetos anularán los anteriores.

Tipos de objetos: **vectores** (numéricos, de caracteres, de índices), **matrices** o más generalmente *arrays*, **factores**, **listas** (vectores cuyos elementos no tienen por qué ser del mismo tipo), **data frames** (generalización de las matrices ya que puede haber columnas de diferente tipo, aunque en cada columna del mismo tipo), **funciones**. De un objeto se puede conocer, por ejemplo, su tipo $mode(\ )$, su longitud $length(\ )$ y su estructura $str(\ )$.

Ejemplo:

$> mode(x)$

[1]"$numeric$"

$> length(x)$

[1]50

$> mode(mean)$

[1]"$function$"

Mediante la función $attributes(\ )$ se pueden obtener las características de un objeto.

Ejemplo: *Los datos "Nile" son una serie temporal del flujo del río Nilo.*

$> Nile$

$TimeSeries$:

$Start = 1871$

$End = 1970$

$Frequency = 1$

[1] 1120 1160 963 1210 1160 1160 813 1230 1370 1140 995 935 1110 994 1020

[16] 960 1180 799 958 1140 1100 1210 1150 1250 1260 1220 1030 1100 774 840

[31] 874 694 940 833 701 916 692 1020 1050 969 831 726 456 824 702

[46] 1120 1100 832 764 821 768 845 864 862 698 845 744 796 1040 759
[61] 781 865 845 944 984 897 822 1010 771 676 649 846 812 742 801
[76] 1040 860 874 848 890 744 749 838 1050 918 986 797 923 975 815
[91] 1020 906 901 1170 912 746 919 718 714 740
$> attributes(Nile)$
$\$tsp$
[1]1871 1970 1
$\$class$
[1]"ts"
Se puede comparar con la función $str(\ )$
$> str(Nile)$
$Time-Series[1:100] from 1871 to 1970 : 1120\ 1160\ 963\ 1210\ 1160\ 1160\ 813$
1230 1370 1140...
Con $attr(objeto, nombre)$ se puede seleccionar un atributo específico.
Ejemplo: *Para crear una matriz 2×2 de unos*
$> num <- c(1,1,1,1)$
$> num$
[1]1 1 1 1
$> attr(num, "dim") <- c(2,2)$
$> num$
$[,1] [,2]$
$[1,]$1 1
$[2,]$1 1

2. *VECTORES*

Uno de las facilidades de R es la manipulación de bases de datos. Dentro de estas estructuras el elemento más simple es la colección ordenada de números, el vector numérico, al que se dedicará este capítulo.

2.1 Vectores numéricos

Para crear vectores se suele utilizar la función $c(.)$, introduciendo los valores numéricos que definen el vector.

Ejemplo: *Para crear el vector* $v = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$

$> v < -c(1, 2, 3, 4, 5)$
$> v$
[1] 1 2 3 4 5

También se puede utilizar la función $assign(.)$.
Ejemplo:
$> assign("v", c(1, 2, 3, 4, 5))$
$> v$
[1] 1 2 3 4 5.

Una vez que el vector v existe como objeto, puede utilizarse para construir otros vectores y para efectuar operaciones sobre todos sus elementos.
Ejemplo:
$> exp(v)$
[1]2.718282 7.389056 20.085537 54.598150 148.413159
$> w < -c(2 * v, 0, 0, v)$
$> w$
[1] 2 4 6 8 1 0 0 0 1 2 3 4 5

2.2 Operaciones aritméticas

Como se ha visto anteriormente, las operaciones con vectores se efectúan elemento a elemento. Si alguno de los vectores es más corto, se alarga hasta

conseguir la longitud del vector más largo.

Ejemplo:

$> d < -c(1, 2, 3)$

$> d$

[1] 1 2 3

$> e < -c(d, 0, 0, 0, 2 * d)$

$> e$

[1] 1 2 3 0 0 0 2 4 6

$> w < -d + e$

$> w$

[1] 2 4 6 1 2 3 3 6 9

Si en lugar del vector anterior, e, introducimos el siguiente, de longitud menor,

$> e < -c(d, 0, 0, 2 * d)$

$> e$

[1]1 2 3 0 0 2 4 6

aparecerá el siguiente mensaje

$> w < -d + e$

$Warning\ message:$

$longer\ object\ length$

$is\ not\ a\ multiple\ of\ shorter\ object\ length\ in: d + e.$

Las operaciones básicas se realizarán con los símbolos habituales: $+, -, *, /$, y $\hat{}$ para elevar a una potencia. También se pueden utilizar las funciones: $log, exp, sin, cos, tan, sqrt, max, min$, así como $diff(range()), length, sum, prod$ que calculan la diferencia entre el valor max y min, el número de elementos, su suma y su producto, respectivamente.

Ejemplo:

$> w$

[1] 2 4 6 1 2 3 3 6 9

$> diff(range(w))$

[1]8

$> length(w)$

[1]9

$> sum(w)$

[1]36

$> prod(w)$

[1] 46656

$> w\hat{}2$

[1] 4 16 36 1 4 9 9 36 81

También se pueden efectuar las operaciones estadísticas más frecuentes: $mean$ y var, siendo

$$var\,(x) = sum((x - mean\,(x))\hat{}2)/\,(length\,(x) - 1)$$

es decir la varianza muestral si x es una muestra unidimensional. Para el caso en el que x es una matriz de datos $n \times p$, el objeto $var\,(x)$ pasará a ser la matriz de covarianzas muestral de dimensión $p \times p$.

Otra operación interesante es la ordenación de un vector mediante la función $sort(\ )$, si se quiere efectuar una permutación se utilizará $order(\ )$ o $sort.list(\ )$.

Hay ocasiones en las que se necesita construir vectores con una determinada sucesión de valores numéricos, por ejemplo de 1 a 10; ésto puede conseguirse mediante la operación $1 : 10$ asignando el resultado a un objeto, pero es más versátil utilizar la función $seq(\ )$ porque admite diferentes posibilidades. Por otra parte está la función $rep(\ )$ para repetir un determinado objeto de diferentes formas.

Ejemplo:

$> (f < -seq(1, 10))$
[1] 1 2 3 4 5 6 7 8 9 10
$> (f < -seq(2, length = 4))$
[1] 2 3 4 5
$> (f < -seq(2, length = 4, by = 0.5))$
[1] 2.0 2.5 3.0 3.5
$> rep(1 : 3, 2)$
[1] 1 2 3 1 2 3
$> rep(1 : 3, 2, each = 2)$
[1] 1 1 2 2 3 3 1 1 2 2 3 3

2.3 Vectores lógicos

Son vectores generados por condiciones que pueden darse o no, por lo que sus elementos posibles son $TRUE$(cierto), $FALSE$(falso) y NA(no disponible). Es preferible no simplificar estos valores a T, F porque si se utilizan como objetos pueden ser alterados.

Ejemplo:

$> f$
[1] 2.0 2.5 3.0 3.5
$> logi1 < -f >= 3$
$> (logi1 < -f >= 3)$
[1] $FALSE\ FALSE\ TRUE\ TRUE$

Los operadores utilizados son los habituales,$<, >, <=, >=, ==$(igualdad exacta), $!\ =$(desigualdad), &(intersección de expresiones), |(AltGr-1, para la unión) y !(negación).

2.4 Valores ausentes ("missing values")

Hay situaciones en las que un valor está no disponible porque es un resultado imposible o se ha perdido, son los valores ausentes de estadística y en estos casos aparece como valor NA. Cualquier operación con ellos seguirá siendo NA, no disponible, sin embargo con la función $is.na(\ )$ se puede construir un nuevo vector donde aparece $TRUE$ si y sólo si el elemento correspondiente es NA.

Si el resultado imposible surge de un cálculo, el símbolo que aparece es NaN(not a number), siendo otra categoría de valores ausentes que se cambia también con $is.na(\ )$, pero si sólo se quieren cambiar los de esta categoría se debe teclear $is.nan(\ )$.

Ejemplo:

$> l$

[1] NA 0.7071068 0.0000000 NaN 1.0000000 0.7071068 0.0000000

$> is.na(l)$

[1] $TRUE\ FALSE\ FALSE\ TRUE\ FALSE\ FALSE\ FALSE$

$> is.nan(l)$

[1] $FALSE\ FALSE\ FALSE\ TRUE\ FALSE\ FALSE\ FALSE$

3. MATRICES Y "ARRAYS"

Se puede definir un "array" como una colección de observaciones con subíndices, siendo las matrices casos particulares, en concreto "arrays" bidimensionales ya que sus elementos constan de dos subíndices. R es especialmente adecuado para crear y manejar este tipo de objetos.

Tienen como atributo la dimensión, que puede asignarse o describirse mediante $dim(\ )$. Así un vector numérico puede utilizarse como un "array" si se le asigna una dimensión.

Ejemplo:

$> w$
[1] 2 4 6 8 10 0 0 1 2 3 4 5
$> dim(w) < -c(2,6)$
$> w$

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	2	6	10	0	2	4
[2,]	4	8	0	1	3	5

$> dim(w) < -c(6,2)$
$> w$

	[,1]	[,2]
[1,]	2	0
[2,]	4	1
[3,]	6	2
[4,]	8	3
[5,]	10	4
[6,]	0	5

$> dim(w) < -c(2,3,2)$
$> w$
, , 1

	[,1]	[,2]	[,3]
[1,]	2	6	10
[2,]	4	8	0

, , 2

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

Como se puede observar en el ejemplo anterior, la forma de colocar los datos es rellenando por columnas los sucesivos bloques. A los elementos colocados en cada posición se les puede denominar con el nombre del "array" seguido de su posición entre [ ] separando los subíndices que describen dicha posición entre comas.

Ejemplo: *Si se considera la última descripción del objeto w en el ejemplo anterior, para obtener la segunda fila de la segunda columna del primer bloque*

$> w[2, 2, 1]$

[1] 8

También se pueden extraer columnas, filas y en general cualquier subconjunto describiendo sus posiciones entre [ ].

Ejemplo: *Con el w anterior se puede obtener la segunda columna del primer bloque*

$> w[, 2, 1]$

[1] 6 8

y las segundas columnas de los dos bloques

$> w[, 2,]$

```
     [,1] [,2]
[1,]    6    2
[2,]    8    3
```

Otra forma de construir "arrays" a partir de vectores, es mediante la función $array(\ )$, especificando el nombre del vector y la dimensión del nuevo objeto. Para construir matrices se utilizará la función $matrix(\ )$.

Ejemplo: *Teniendo en cuenta que x es* $x < -rnorm(50, 2, 3)$ *del primer ejercicio, se puede convertir en un "array" formado por dos bloques de cinco filas y cinco columnas*

$> (arx < -array(x, c(5, 5, 2)))$

```
, , 1
          [,1]      [,2]      [,3]       [,4]      [,5]
[1,] -3.2106300 0.3877666 7.5592096 -0.3403993 1.2039125
[2,] -0.7870847 8.7728136 1.0124324  2.7640037 -0.7761628
[3,]  0.4241971 0.4688422 0.2401416  3.4419892 -1.9599388
[4,]  0.2159416 1.0030262 -1.3389534 3.0205676 -4.2068276
[5,]  4.1927219 -6.1559740 4.4434380 1.6881193 -1.1155851
, , 2
          [,1]     [,2]      [,3]        [,4]     [,5]
[1,] -5.247075 2.850224 3.6138058 -0.04179251 1.423822
[2,]  1.189775 -2.359040 6.5736714 0.67692024 1.260775
[3,]  4.042855 -1.425400 -4.1827371 7.05517180 3.344217
[4,]  3.106876 6.196905 -1.2356566 -2.54117124 1.363164
[5,]  4.199660 -1.659834 0.2650083 0.97560422 4.381154
```

Si el vector no tiene elementos suficientes para rellenar los huecos del "array", con la función anterior se completa con los primeros valores del vector. También se pueden efectuar distintas operaciones con los "arrays" como permutar, el producto exterior...que se describirán a continuación en el caso particular y claramente más frecuente de las matrices.

3.1 Operaciones con matrices

Como se dijo anteriormente, una matriz es un "array" con dos subíndices. Entre las funciones que describen y manejan una matriz están $nrow(\ )$ y $ncol()$ para obtener el número de filas y columnas respectivamente, mientras que $t(\ )$ es la función que traspone la matriz. Otra función interesante es $diag(\ )$ que si se aplica a una matriz devuelve el vector formado con los elementos de la diagonal principal, si se aplica a un vector lo que se obtiene es una matriz diagonal con los valores del vector y por último sobre un escalar construye la matriz identidad de la dimensión dada por el escalar.

Ejemplo: *Sea la matriz mat*1 *la obtenida con el primer bloque del "array" arx obtenido en el ejemplo anterior.*

$> (mat1 < -arx[, , 1])$

```
        [,1]       [,2]      [,3]       [,4]       [,5]
[1,] -3.2106300 0.3877666 7.5592096 -0.3403993 1.2039125
[2,] -0.7870847 8.7728136 1.0124324 2.7640037 -0.7761628
[3,] 0.4241971 0.4688422 0.2401416 3.4419892 -1.9599388
[4,] 0.2159416 1.0030262 -1.3389534 3.0205676 -4.2068276
[5,] 4.1927219 -6.1559740 4.4434380 1.6881193 -1.1155851
```

$> ncol(mat1)$

```
[1] 5
```

$> nrow(mat1)$

```
[1] 5
```

$> (tmat1 < -t(mat1))$

```
        [,1]       [,2]      [,3]       [,4]       [,5]
[1,] -3.2106300 -0.7870847 0.4241971 0.2159416 4.192722
[2,] 0.3877666 8.7728136 0.4688422 1.0030262 -6.155974
[3,] 7.5592096 1.0124324 0.2401416 -1.3389534 4.443438
[4,] -0.3403993 2.7640037 3.4419892 3.0205676 1.688119
[5,] 1.2039125 -0.7761628 -1.9599388 -4.2068276 -1.115585
```

$> (dmat1 < -diag(mat1))$

```
[1] -3.2106300 8.7728136 0.2401416 3.0205676 -1.1155851
```

$> diag(dmat1)$

```
        [,1]       [,2]      [,3]       [,4]       [,5]
[1,] -3.21063 0.000000 0.0000000 0.000000 0.000000
[2,] 0.00000 8.772814 0.0000000 0.000000 0.000000
[3,] 0.00000 0.000000 0.2401416 0.000000 0.000000
[4,] 0.00000 0.000000 0.0000000 3.020568 0.000000
[5,] 0.00000 0.000000 0.0000000 0.000000 -1.115585
```

$> diag(5)$

```
   [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
```

```
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

3.1.1 Operaciones con matrices

El operador para aplicar esta operación entre matrices es $\%*\%$, utilizándose $*$ para el producto elemento por elemento de matrices del mismo tamaño. No obstante la función $crossprod(mat1, mat2)$ es equivalente a $mat1\%*\%mat2$ pero el proceso es más rápido.

Para obtener la inversa de una matriz cuadrada no singular se puede utilizar la solución de una ecuación lineal, por ejemplo de $q = Q\%*\%x$, ya que $x = Q^{-1}\%*\%q$, mediante la función $solve(Q, q)$, pudiendo calcular dicha matriz inversa mediante $solve(Q)$. Para obtener el determinante de Q se utiliza $det(Q)$.

Ejemplo: *Se calcula la inversa de la matriz mat*1 *definida en ejemplos anteriores con el vector x*1 *de unos*

```
> x1 <- rep(1,5)
> b1 <- mat1%*%x1
> solve(mat1,b1)
     [,1]
[1,]    1
[2,]    1
[3,]    1
[4,]    1
[5,]    1
> solve(mat1)
           [,1]       [,2]       [,3]        [,4]        [,5]
[1,] -0.12877872 0.15323841 -0.2164438 -0.01502533  0.19133400
[2,] -0.02089566 0.13530681 -0.1513859  0.03348884  0.02299115
[3,]  0.08597911 0.05810572 -0.1229635  0.04817971  0.08670692
[4,] -0.01601109 -0.04914441 0.5782148 -0.24706520 -0.06726225
[5,] -0.05045421 -0.01365353 0.4070992 -0.42322674 -0.06058936
```

3.1.2 Autovalores y autovectores

Aplicando la función $eigen(\ )$ a una matriz simétrica, se obtienen los correspondientes vector de autovalores y matriz de autovectores.

Ejemplo: *Se define la matriz* $mat2 = \begin{pmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{pmatrix}$ *y se determinan sus autovalores y autovectores*

```
> d1 <- c(10,0,2,0,10,4,2,4,2)
> (mat2 <- matrix(d1,ncol=3,nrow=3))
 [,1] [,2] [,3]
[1,] 10 0 2
[2,] 0 10 4
[3,] 2 4  2
> eigen(mat2)
$values
[1] 1.200000e+01 1.000000e+01 1.776357e-15
$vectors
      [,1]          [,2]          [,3]
[1,] -0.4082483 8.944272e-01 0.1825742
[2,] -0.8164966 -4.472136e-01 0.3651484
[3,] -0.4082483 5.551115e-17 -0.9128709
```

Si se utiliza *eigen( )$values* o bien *eigen( )$vectors* solamente aparecerán los autovalores o bien los autovectores que también pueden almacenarse como objetos.

Otra función aplicable a matrices muy útil es $svd(Q)$ ("singular value descomposition"), mediante la que se obtienen las matrices U, D y V tales que $Q = U \,\%*\%\, D \,\%*\%\, t(V)$, donde las columnas de U y de V son ortogonales y D es una matriz diagonal.

Ejemplo:

```
> d3 <- c(1,2,5,3,7,9,2,7,1)
> mat3 <- matrix(d3,c(3,3))
> mat3
  [,1] [,2] [,3]
[1,] 1 3 2
[2,] 2 7 7
[3,] 5 9 1
> svd(mat3)
$d
[1] 14.0785096 4.9778584 0.1284231
$u
       [,1]          [,2]          [,3]
[1,] -0.2628271 -0.1087879 -0.9586903
[2,] -0.6746563 -0.6896073 0.2632121
[3,] -0.6897541 0.7159658 0.1078531
$v
       [,1]          [,2]          [,3]
[1,] -0.3594777 0.4202262 0.8331781
[2,] -0.8323937 0.2591632 -0.4898522
```

[3,] -0.4217778 -0.8696231 0.2566303

3.1.3 Construcción de matrices particionadas

Las funciones *cbind*( ) y *rbind*( ) permiten obtener una matriz a base de agregar matrices por columnas y filas respectivamente.

Ejemplo.

$> cbind(mat3, mat3)$

```
   [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 1    3    2    1    3    2
[2,] 2    7    7    2    7    7
[3,] 5    9    1    5    9    1
```

$> rbind(mat3, mat3)$

```
  [,1] [,2] [,3]
[1,]  1    3  2
[2,]  2    7   7
[3,]  5    9   1
[4,]  1    3   2
[5,]  2  7    7
[6,]  5    9   1
```

Hay que señalar la diferencia con la función $c(\ )$ que encadena objetos eliminando su dimensión.

Ejemplo:

$> c(mat3, mat3)$

[1] 1 2 5 3 7 9 2 7 1 1 2 5 3 7 9 2 7 1

3.1.4 Tablas de contingencia

A partir de la función *table*( ) se pueden obtener tablas de frecuencia asociadas a factores de igual longitud. En el ejemplo siguiente se utiliza la función *sample*( ) que permite extraer muestras de un conjunto de observaciones y el cruce de dos factores para construir una tabla de doble entrada.

Ejemplo:

$> table(c(mat3, mat3))$

1 2 3 5 7 9

4 4 2 2 4 2

$> (v4 < -c(mat3, mat3))$

[1] 1 2 5 3 7 9 2 7 1 1 2 5 3 7 9 2 7 1

$> (mat4 < -sample(v4, 18))$

[1] 1 1 7 1 2 7 5 3 7 9 2 3 1 9 5 2 7 2

$> table(v4, mat4)$

	$mat4$					
$v4$	1	2	3	5	7	9
1	1	1	0	0	1	1
2	1	1	0	1	1	0
3	1	0	0	0	0	1
5	0	1	0	0	1	0
7	0	1	1	1	1	0
9	1	0	1	0	0	0

Para describir un factor mediante la agrupación en clases, se utilizan las funciones $factor(\ )$ y $cut(\ )$ como se indica en el siguiente ejemplo que trabaja con las columnas 4^a y 5^a de los datos $airquality$ y que representan $Temperatura$, 153 valores entre 56 y 97 y $Meses$, incluyendo solamente del 5 al 9. Si se hubiera descrito $Tempf$ solo con $cut(\ )$ y las 11 clases que se han pedido de forma aproximada en lugar de las 9 que efectivamente le corresponden, habrían salido dos filas de ceros en la tabla final.

$> Temp < -airquality[, 4]$

$> Tempf < -factor(cut(Temp, breaks = 55 + 5 * (0 : 11)))$

$> Month < -airquality[, 5]$

$> table(Tempf, Month)$

	$Month$				
$Tempf$	5	6	7	8	9
(55,60]	8	0	0	0	0
(60,65]	7	1	0	0	2
(65,70]	9	1	0	0	5
(70,75]	4	5	2	2	6
(75,80]	2	13	1	9	8
(80,85]	1	5	18	6	4
(85,90]	0	3	7	9	1
(90,95]	0	2	3	3	4
(95,100]	0	0	0	2	0

4. BASES DE DATOS

En este apartado se pueden incluir los objetos $list(\ )$, como colección ordenada, a su vez, de objetos y $data.frame(\ )$ quizá la estructura de datos más utilizada y que consiste en una colección de variables de la misma longitud que pueden ser de tipo diferente.

4.1 Listas

Con la función $length(\ )$ aplicada al nombre de la lista, se obtiene el número de objetos de que consta, utilizando [[ ]] para describir dichas componentes y seguidas del número de orden entre [ ] que ocupa cada elemento del objeto para describirlos.

Ejemplo:

$> (lmat2 < -list(mat1, mat2))$

```
[[1]]
       [,1]       [,2]      [,3]       [,4]       [,5]
[1,] -3.2106300 0.3877666 7.5592096 -0.3403993 1.2039125
[2,] -0.7870847 8.7728136 1.0124324 2.7640037 -0.7761628
[3,] 0.4241971 0.4688422 0.2401416 3.4419892 -1.9599388
[4,] 0.2159416 1.0030262 -1.3389534 3.0205676 -4.2068276
[5,] 4.1927219 -6.1559740 4.4434380 1.6881193 -1.1155851
[[2]]
[,1] [,2] [,3]
[1,] 10 0 2
[2,] 0 10 4
[3,] 2 4 2
```

$> lmat2[[2]][5]$

```
[1]10
```

$> lmat2[[1]][7]$

```
[1] 8.772814
```

Las listas o elementos de las mismas se pueden enlazar para conseguir nuevas listas.

Ejemplo:

$> lmat2[[3]] < -lmat2[[2]][5]$

$> lmat2$

```
[[1]]
       [,1]       [,2]      [,3]       [,4]       [,5]
[1,] -3.2106300 0.3877666 7.5592096 -0.3403993 1.2039125
```

[2,] -0.7870847 8.7728136 1.0124324 2.7640037 -0.7761628
[3,] 0.4241971 0.4688422 0.2401416 3.4419892 -1.9599388
[4,] 0.2159416 1.0030262 -1.3389534 3.0205676 -4.2068276
[5,] 4.1927219 -6.1559740 4.4434380 1.6881193 -1.1155851
[[2]]
[,1] [,2] [,3]
[1,] 10 0 2
[2,] 0 10 4
[3,] 2 4 2
[[3]]
[1] 10

4.2 Data frames

La forma de construir data frames, siempre que los elementos tengan la misma dimensión, es mediante $data.frame(\ )$ y $as.data.frame(\ )$.

Para conseguir que las variables del data frame sean objetos se utilizan las funciones $attach(\ )$ y $detach(\ )$

Ejemplo.

$> mat2$
[,1] [,2] [,3]
[1,] 10 0 2
[2,] 0 10 4
[3,] 2 4 2
$> (dfmat2 < -data.frame(mat2))$
X1 X2 X3
1 10 0 2
2 0 10 4
3 2 4 2
$> X1$
Error: Object "X1" not found
$> attach(dfmat2)$
$> X1$
[1] 10 0 2
$> detach(dfmat2)$
$> X1$
Error: Object "X1" not found

4.2.1 Importación de bases de datos

La mayoría de las veces, las observaciones se encontrarán almacenadas en ficheros externos que se tienen que incorporar mediante las funciones $read.table($), que pasa los datos a un $data.frame$ y $scan($) que los pasa a un vector.

Para poder leer datos con $read.table($) éstos deben de tener en la primera línea el nombre de cada variable y en las sucesivas filas una etiqueta y los valores de las variables. Con $read.table($) se pueden leer los ficheros directamente si se encuentran en el mismo directorio en el que se arranca R y dando los datos de ubicación del fichero en su caso.

Ejemplo: *Se copia la siguiente base de datos en un fichero de texto, cork.txt* (mediante *copy* y *paste* en Wordpad, por ejemplo) *y se coloca en el mismo directorio desde el que se arranca R*

A B C D
1 72 66 76 77
2 60 53 66 63
3 56 57 64 58
4 41 29 36 38
$> (dfcork <- read.table("cork.txt"))$
A B C D
1 72 66 76 77
2 60 53 66 63
3 56 57 64 58
4 41 29 36 38

Si se copia en un disquete habrá que introducir

$> (dfcork <- read.table("a:/cork.txt"))$

Para leer una base de datos que se encuentra en una dirección de internet

$> (examE <- read.table("http://www.stat.ucla.edu/data/cox-and-snell/exampleE.data"))$

V1 V2 V3 V4 V5 V6 V7
1 1 210 201 -9 130 125 -5
2 2 169 165 -4 122 121 -1
3 3 187 166 -21 124 121 -3
4 4 160 157 -3 104 106 2
5 5 167 147 -20 112 101 -11
6 6 176 145 -31 101 85 -16
7 7 185 168 -17 121 98 -23
8 8 206 180 -26 124 105 -19
9 9 173 147 -26 115 103 -12
10 10 146 136 -10 102 98 -4
11 11 174 151 -23 98 90 -8
12 12 201 168 -33 119 98 -21
13 13 198 179 -19 106 110 4
14 14 148 129 -19 107 103 -4

15 15 154 131 -23 100 82 -18

Si se utiliza la función scan( )

$> (sexamE < -scan("http : //www.stat.ucla.edu/data/cox - and - snell/exampleE.data"))$

Read 105 items

[1] 1 210 201 -9 130 125 -5 2 169 165 -4 122 121 -1 3 187 166 -21

[19] 124 121 -3 4 160 157 -3 104 106 2 5 167 147 -20 112 101 -11 6

[37] 176 145 -31 101 85 -16 7 185 168 -17 121 98 -23 8 206 180 -26 124

[55] 105 -19 9 173 147 -26 115 103 -12 10 146 136 -10 102 98 -4 11 174

[73] 151 -23 98 90 -8 12 201 168 -33 119 98 -21 13 198 179 -19 106 110

[91] 4 14 148 129 -19 107 103 -4 15 154 131 -23 100 82 -18

Para leer el fichero cork.txt con scan( ) hay que quitar el nombre de las variables. Por ejemplo guardamos en scork.txt solo los valores

1 72 66 76 77

2 60 53 66 63

3 56 57 64 58

4 41 29 36 38

$> (scork < -scan("scork.txt"))$

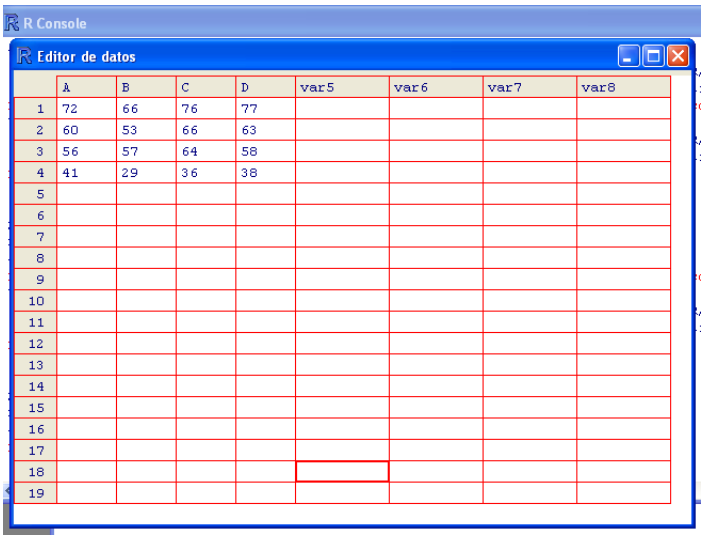
Read 20 items

[1] 1 72 66 76 77 2 60 53 66 63 3 56 57 64 58 4 41 29 36 38

Se puede utilizar a continuación la función *edit( )* sobre la base de datos que se ha leído, para manejarla con comodidad en una ventana auxiliar.

Ejemplo:

$> edfcork < -edit(dfcork)$



Importación de bases de datos de otros sistemas estadísticos

Es recomendable cargar el paquete *foreign* aunque en algunas ocasiones se podrán traer utilizando *read.table( )*, no obstante se podrá utilizar menos memoria con las siguientes funciones.

Para traer ficheros de MINITAB con extensión *.mtp* (Minitab Portable Worksheet) se tiene *read.mtp*( ) que lo trae en forma de lista. Para leer ficheros en formato SAS Transport (XPORT) y pasarlos a una lista de data frames, se tiene *read.xport*( ) y análogamente *read.spss*( ) para ficheros exportados por SPSS, *read.S*( ) para muchos objetos de S-PLUS y *read.dta*( ) para ficheros de STATA.

5. DISTRIBUCIONES DE PROBABILIDAD

Con R es posible determinar algunas características de las distribuciones de probabilidad más utilizadas. teniendo que dar en cada caso los parámetros correspondientes. En concreto según la letra que se coloque delante del nombre de la distribución de probabilidad se podrá obtener la densidad si se coloca d, la CDF(función de distribución) si p, la función cuantílica si q y para simular una muestra habrá que poner r.

Ejemplo: *La distribución Normal se denomina norm, teniendo que especificar los parámetros media y desviación típica así que para obtener la función de densidad, y de distribución en* $z = 1.96$ *para una* $N(0, 1)$

$> dnorm(1.96, 0, 1)$

[1]0.05844094

$> pnorm(1.96, 0, 1)$

[1]0.9750021

$> qnorm(0.975, 0, 1)$

[1]1.959964

$> qnorm(0.9750021, 0, 1)$

[1]1.96

Ejemplo: *Para obtener una muestra de tamaño* 100 *de una* $N(3, 2)$ *y otra independientemente de una* $t - Student$, *a la que se denomina* t, *con* 1 *grado de libertad, y representar ambos histogramas para comparar la diferencia en las colas. Para representar ambos histogramas se utilizará la función* $par(\ )$ *que establece parámetros en los gráficos y de la que se utiliza* $mfrow = c(2, 1)$ *para particionar la ventana del gráfico en dos filas y una columna.*

$> par(mfrow = c(2, 1))$

$> mt1 < -rt(100, 1)$

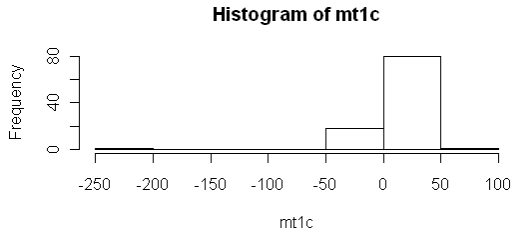
$> mt1c < -mt1 * 2 + 3$

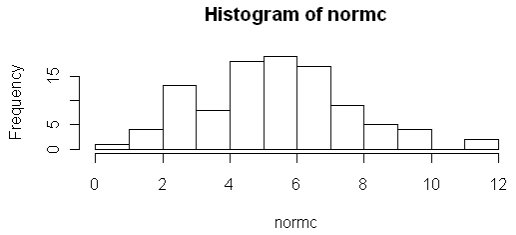
$> hist(mt1c)$

$> norm < -rnorm(100, 1)$

$> normc < -norm * 2 + 3$

$> hist(normc)$


Histogram of mt1c
Frequency
0
40
80
-250
-200
-150
-100
-50
0
50
100
mt1c


Histogram of normc
Frequency
0
5
15
0
2
4
6
8
10
12
normc

6. PROGRAMACIÓN

Los comandos en R, pueden ser agrupados entre "llaves" { } dando como valor el resultado final con la última expresión. A su vez puede integrarse dentro de una expresión más amplia y así sucesivamente. Dentro de estas expresiones puede haber ciertos comandos de control de la ejecución permitiendo condicionar la ejecución de un bloque o ejecutar repetidamente otro.

6.1 Estructuras de control

6.1.1 Ejecución condicionada

Responde al comando if / $else$ de la forma habitual. Hay una ($expresión\_1$) que debe evaluarse, si es $CIERTA$ entonces se evalúa la $expresión\_2$, de lo contrario se evalúa la $expresión\_3$, el valor final de la expresión global es el de la seleccionada finalmente. La sintaxis correspondiente al caso anterior sería

$> if\ (expresión\_1)$
$expresión\_2$
$else$
$expresión\_3$

También pueden enlazarse sucesivas condiciones

$> if\ (expresión\_1)$
$expresión\_2$
$else\ if\ (expresión\_3)$
$expresión\_4$
$else\ if\ (expresión\_5)$
$expresión\_6$
$else$
$expresión\_8$

evaluándose, en orden, las sucesivas expresiones impares hasta que una es $CIERTA$, valorándose la correspondiente expresión par de dicho caso. No hay límites para los $else\ if$ permitidos.

Ejemplo: *Donde se van alterando los valores de una muestra de números aleatorios según sea el valor de su suma.*

$> u < -runif(20)$
$> u$

[1] 0.8274505 0.7492379 0.9971830 0.6657915 0.2522081 0.2581410 0.6137507

[8] 0.1366209 0.8652314 0.6629930 0.6904928 0.8157469 0.7545211 0.9928052

[15] 0.8426746 0.8316724 0.3762088 0.3373698 0.1013896 0.8551755

$> s <- -sum(u)$
$> if\ (s < 5)\ u <- -1 + u\ else\ if\ (s < 10)\ u <- --u$
$> u$

[1] 0.8274505 0.7492379 0.9971830 0.6657915 0.2522081 0.2581410 0.6137507

[8] 0.1366209 0.8652314 0.6629930 0.6904928 0.8157469 0.7545211 0.9928052

[15] 0.8426746 0.8316724 0.3762088 0.3373698 0.1013896 0.8551755

$> s$

[1] 12.62666

$>if\ (s < 5)\ u <- -1 + u\ else\ if\ (s < 10)\ u <- --u\ else\ if\ (s < 15)\ u <- -2 * u$
$> u$

[1] 1.6549009 1.4984758 1.9943660 1.3315830 0.5044162 0.5162821 1.2275015

[8] 0.2732418 1.7304629 1.3259860 1.3809856 1.6314938 1.5090422 1.9856104

[15] 1.6853491 1.6633447 0.7524176 0.6747396 0.2027791 1.7103510

6.1.2 Ejecución repetida

Mediante las órdenes for, $while$ and $repeat$ se puede conseguir la valoración repetida de ciertas expresiones. Por ejemplo para for sería

$> for\ (nombre\ en\ expresión\_1)\ \{expresión\_2\}$

donde el $nombre$ es el contador de las repeticiones y en la $expresión\_2$ se incluyen las acciones a realizar, en función de la variable $nombre$.

Ejemplo: *A partir de una matriz* 20×10 *de observaciones aleatorias e independientes* $N(0,1)$ *se obtienen las medias muestrales de las columnas, representándolas con un gráfico de tallo y hojas*

$z <- -matrix(rnorm(200), 20, 10)$
$mean.samp <- -NULL$

$for(i\ in\ 1:10)$
{
$mean.samp[i] <- mean(z[,i])$
}
$stem(mean.samp)$

6.2 Funciones

En lenguaje R las funciones se pueden incorporar como objetos a los procedimientos o almacenar para su uso posterior. La forma general es

$nombre <- function(argumento\_1, argumento\_2, \ldots) expresión$

donde la *expresión* se escribe en lenguaje R y utiliza los argumentos introducidos. Para llamar a la función se utilizará el $nombre(valores\ de\ los\ argumentos)$ como se utilizan normalmente las funciones que ya están definidas en los paquetes que se cargan al comienzo.

Ejemplo: *Función para calcular y listar las medias aritméticas de distintas potencias de una colección de* 10 *observaciones aleatorias e independientes* $N(0,1)$.

$fourmom <- function(x)\{$
$m1 <- mean(x)$
$m2 <- mean(x\hat{}2)$
$m3 <- mean(x\hat{}3)$
$m4 <- mean(x\hat{}4)$
$list(m1 = m1, m2 = m2, m3 = m3, m4 = m4)$
}
$x <- rnorm(10)$
$fourmom(x)$