Programación con R

Investigación Operativa en la Empresa

Mención en Economía, Técnicas de Investigación y Comunicación en la Empresa

Curso 2024/2025

UCLM

¿POR QUÉ APRENDER R?

- Ampliamente utilizado en la comunidad científica
- Se distribuye gratuitamente
- Existen numerosas librerías para utilizar también de forma gratuita
- Se pueden resolver la mayoría de problemas de IO con esta herramienta
- Sirve de toma de contacto con la programación en el caso de que no se tenga experiencia

INSTALACIÓN

- Accedemos a https://www.r-project.org/ y buscamos las opciones de *download*. Posteriormente elegimos la versión según nuestro sistema operativo.
- Una vez descargado, lo lanzamos y seguimos las instrucciones.
- Esto nos dará una interfaz de R con una consola donde podremos escribir instrucciones, instalar librerías etc. Sin embargo esta es muy básica, y se recomienda utilizar otras herramientas de desarrollo que proporcionan más facilidad en su uso.
  - Existen herramientas como Rstudio o Rcommander que facilitan el trabajo. En nuestras prácticas usaremos Rstudio.

RSTUDIO

- Es un entorno de desarrollo gratuito que facilita el manejo del lenguaje R
- Lo podemos descargar de (RStudio Desktop):
  - https://www.rstudio.com/products/rstudio/download/
- En el caso de utilizar Windows XP podemos encontrar una versión más antigua que se puede utilizar en dicho sistema:
  - https://support.rstudio.com/hc/en-us/articles/206569407-Older-Versions-of-RStudio-Desktop

SECCIONES DE RSTUDIO

- Editor de texto para scripts (ver más adelante)
- Consola
  - En ella se pueden ejecutar diferentes instrucciones como operaciones o comandos de instalación de librerías etc. Se podría trabajar con ella directamente sin nada más, pero sería algo más complicado.
- Entorno de trabajo
  - Se nos muestran las variables y objetos que están actualmente en memoria. Podremos inspeccionarlos o borrarlos.
- Ventana de utilidades
  - En ellas se nos mostrará la ayuda, la visualización de gráficos y otras funciones útiles.

¿CÓMO TRABAJAR CON R?

- Podemos trabajar en R insertando una a una nuestras órdenes en la línea de comandos, pero esto puede resultar incómodo en ocasiones. Especialmente cuando queremos reutilizar el código.
- La mejor manera de trabajar con R es utilizar scripts:
  - Un **script** es un archivo de texto, que podemos crear con RStudio (por ejemplo), y del cual se ejecutarán todas las expresiones que seleccionemos del mismo una tras otra.
- Podemos ejecutar la sentencia de la línea actual o de las líneas que tenemos seleccionadas, pulsando Alt+Enter
- También podemos utilizar los iconos del RStudio

MANEJO DE VARIABLES

- Una variable guarda un valor en memoria que puede cambiar con el tiempo
- Podemos asignarlo con el operador <-
  - `n <- 1`
  - `n <- 2+8`
- Si hacemos una segunda asignación sobre la variable, tomará el nuevo valor
- Para ver su contenido, simplemente podemos escribir su nombre
  - `> n`
  - `10`
- Podemos escribir una expresión sin asignar a una variable, pero su resultado no será guardado en memoria
  - `> (10+2)*3`
  - `36`
- Las variables de texto se deben guardar entre comillas
  - `texto <- “variable de texto”`

MANEJO DE VARIABLES

- Para mostrar el contenido de una variable podemos usar entre otros
  - `print()`
  - `paste()`
- Ejemplo
  - `> s <- "si"`
  - `> a <- 10`
  - `> print(a)`
  - `10`
  - `> paste(s,s,sep="")`
  - `"sisi"`
  - `> paste(s,s,sep="-")`
  - `"si-si"`
  - `> paste(s,a,sep="-")`
  - `"si-10"`

MANEJO DE VARIABLES

- R distingue entre mayúsculas y minúsculas, por lo que dos variables que difieran en esto no serán la misma.
- Los nombres de las variables pueden contener letras, números y puntos.
  - Evitar caracteres “raros” tipo ñ
  - El punto aunque se puede usar no lo recomiendo mucho ya que en otros lenguajes se utiliza para acceder a propiedades de objetos
- El conjunto de objetos que tenemos en un determinado momento en memoria se denomina *workspace*
- Los objetos creados durante una sesión pueden guardarse en un archivo .Rdata para usarlo posteriormente
- Podemos eliminar objetos con el comando `rm()` o mediante RStudio directamente

TIPOS DE VARIABLES

- Numéricas (numeric)
  - `x <- 22`
  - `a <- 9.32`
- Complejos (complex) (Números de la forma a+bi)
- Cadenas de caracteres (character)
  - `nombre <- “Alfredo”`
- Lógicos (logical): Toman los valores lógicos TRUE o FALSE
  - `varon <- TRUE`

VECTORES

- Un vector se construye con la palabra c
  - `x <- c(3,4,8,5) # Crea un vector con los números pasados`
- Al escribir una almohadilla # se indica el inicio de un comentario
- Se puede crear un vector con secuencias crecientes (o decrecientes) de valores con:
  - `a <- 1:20`
- Se puede utilizar c para concatenar vectores
  - `> x<-c(1,2)`
  - `> y<-c(3,4)`
  - `> z<-c(x,y)`
  - `> x`
  - `[1] 1 2`
  - `> y`
  - `[1] 3 4`
  - `> z`
  - `[1] 1 2 3 4`
  - `> v<-c(z,5:10)`
  - `> v`
  - `[1] 1 2 3 4 5 6 7 8 9 10`

VECTORES

- `> x<-c(1,3,5,6,5,4,3,4,5,6,6,7,7,8,9,9,4,5,6)`
- `> x`
- `1 3 5 6 5 4 3 4 5 6 6 7 7 8 9 9 4 5 6`
- Para extraer elementos de un vector se utiliza [ ]
  - `> x[3]`
  - `5`
  - `> x[c(1,2)]`
  - `1 3`
  - `> x[3:6]`
  - `5 6 5 4`

VECTORES

- Las operaciones aritméticas afectan a todos los componentes de un vector. Y si son del mismo tamaño se operan componente a componente
  - `> v1 <- c(1,2)`
  - `> v2 <- c(3,4)`
  - `> v1*v2`
  - `3 8`
  - `> 10*v1`
  - `10 20`

OPERADORES LÓGICOS

- Los elementos de tipo lógico tienen dos posibilidades, TRUE o FALSE, que se pueden abreviar en T y F.
- Estos objetos lógicos son generalmente fruto de una comparación
  - Las comparaciones lógicas son <, <=, >, >=, ==, !=
- Si se utilizan elementos lógicos en operaciones aritméticas, se interpretarán como 1 (TRUE) y 0 (FALSE)
- Existe una aritmética para operadores lógicos (siguiente diapositiva)

OPERADORES LÓGICOS

- `x<-c(T,T,F,F)`
- `y<-c(T,F,T,F)`
- `x&y # produce T F F`
- `x|y # produce T T T`

Operador	Operación
$!x$	Negación de x. Los T los convierte en F y viceversa.
$x\&y$	Intersección, operador lógico y: T y T da T, otra comparación da F
$x\|y$	Unión, operador lógico o: F y F da F, otra comparación da T.
`xor(x,y)`	Exclusivo OR, `xor(T,F)==T`, otra comparación da F.
`all`	Para una secuencia de argumentos lógicos, `all` devuelve el valor lógico que indica si todos los elementos son TRUE.
`any`	Para una secuencia de argumentos lógicos, `any` devuelve el valor lógico que indica si algún elemento es TRUE.

FUNCIONES

- Cualquier función matemática que se nos ocurra seguramente ya estará en R:
  - `log(x),exp(x),log(x,n),log10(x),sqrt(x),factorial(x), choose(n,x),gamma(x),lgamma(x),floor(x),ceiling(x),trunc(x),round(x,digits=0),signif(x,digits=6), cos(x), sin(x), tan(x), acos(x),asin(x), atan(x), acosh(x), asinh(x), atanh(x), abs(x)`

- Hay funciones para infinidad de propósitos.
- Si queremos saber cómo funciona una función, podremos mostrar la ayuda mediante `help()` o mediante `?`
  - `help(round) # Muestra ayuda sobre la función round`
  - `? round # hace lo mismo que la anterior`

FUNCIONES CON VECTORES

- `names()`: Asigna nombres a los elementos de un vector
  - `> x<-c(3.141592, 2.718281)`
  - `> names(x)<-c("pi","e")`
  - `> x`
  - `pi e`
  - `3.141592 2.718281`
  - `> x["pi"]`
  - `pi`
  - `3.141592`
- `length()`: Devuelve la dimensión de un vector
- `sum()` y `cumsum()`: Proporcionan la suma y sumas acumuladas de las componentes de un vector
- `max()` y `min()`
- `mean()`, `median()`, `var()` y `sd()`: Dan la media, mediana, varianza y desviación típica
- `quantile()`: Calcula los cuartiles
- `sort()`: ordena en orden creciente las componentes de un vector
- `rev()`: Coloca los componentes en orden inverso

GENERACIÓN DE SECUENCIAS REGULARES

- Existen varias maneras de generar estas secuencias
  - Operador : , desde:hasta (de uno en uno)
  - `seq()`: Permite generar sucesiones más complejas. Tiene como argumentos `(from, to, by, length)` (Consultar la ayuda)
  - rep(): Repite un objeto de diferentes formas, la más sencilla es `rep(x,times=<nºveces>)`
  - `sequence(nvec)`: En el vector se especifican los limites superiores de subvectores que empiezan en 1 y se concatenan entre sí

```
pi:1
seq(0,1,length=10)
seq(1,5,by=0.5)
rep(1:4,2)
rep(1:4,c(2,3,1,2))
rep(1:4,c(2,2)) # error, tamaños no coinciden
sequence(c(2,3))
```

MATRICES

- Las matrices son un caso particular de variables indexadas (arrays) para dos dimensiones
- Todos los elemento deben ser del mismo tipo
- Una matriz se define con el comando `matrix()` especificando el número de filas y columnas o asignando la dim a un vector
- Se crean por columnas, aunque con la opción `byrow=TRUE` lo hace por filas
- Podemos asignar nombres a las filas y columnas con el atributo dimnames.

MATRICES

- Las creamos con `matrix()`
  - `e <- matrix(1:12,ncol=3,byrow=T)`
  - Ejemplos de selección:
  - `e[1,1]`
  - `e[,c(2,3)] # Selecciona las columnas 2 y 3`
  - `e[,c(-1,-3),drop=F] # selecciona todas las columnas menos la 1 y la 3, y lo devuelve en formato matriz`
  - `s <- matrix(rep(c(T,F),6),ncol=3) # genera una matriz de TRUE y FALSE (booleano)`
  - `e[s]`

ALGUNOS EJEMPLOS

- `x<-matrix(1:6,ncol=3)`
- `x[,2]`
- `x[1,1:2]`
- `y<-matrix(1:6,ncol=2)`
- `y[3,]`
- `ncol(x)`
- `nrow(y)`

LISTAS

- Objeto que consiste en una colección ordenada de objetos
- Los componentes no tienen por qué ser del mismo tipo ni longitud
- Se construyen con `list()` o concatenando otras listas
- Se pueden acceder por su posición entre corchetes o bien por su nombre (si tienen) precedidos del símbolo dólar

```
ejemplolista <- list(nombre="Pedro", casado=T, esposa="María",no.hijos=3, edad.hijos=c(4,7,9))
ejemplolista
ejemplolista[5]
ejemplolista[[5]]
ejemplolista$casado
ejemplolista$nombre
listamasgrande <- c(ejemplolista,list(edad=40))
```

HOJAS DE DATOS (DATA FRAMES)

- Sirven para describir información de diferentes individuos, que representarán cada uno una fila
- Los componentes deben ser vectores, matrices u otros data frames

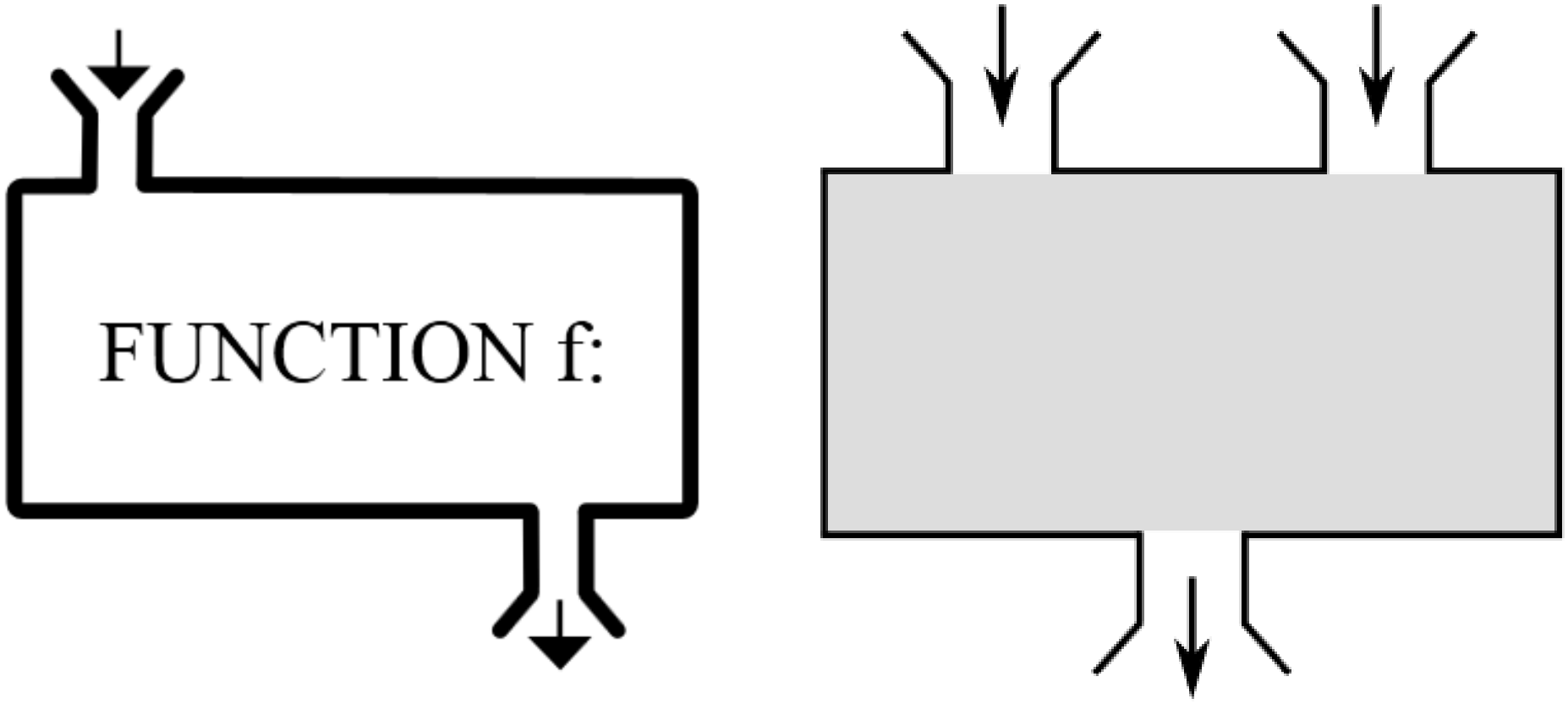
```
datosimp <-
data.frame(anyos=c(1.3,0.4,1.1,2.3,3.1,1.3),
tipo=c(2,3,3,1,3,1),edad=c(22,21,34,42,17,43),
sexo=c("H","M","H","H","M","H"))
```

- `datosimp$anyos`
- `maspeq <-subset(datosimp, anyos<1, select=c(edad,sexo)) # Podemos seleccionar un subconjunto con esta función`

SUBRUTINAS

- Una subrutina es un segmento de código que se escribe una sola vez pero puede ejecutarse muchas veces
- Hay dos tipos:
  - **Procedimientos** que no devuelven ningún valor
  - **Funciones** que devuelven un resultado
- La mejor manera de escribir este subrutinas es mediante los ficheros script.
- Muchas funciones de R están escritas en código interno, otras utilizan conexiones a C, Fortran etc. Pero muchas otras como *mean* o var están escritas en R y por tanto, no difieren tanto de las que nosotros podríamos escribir.
- Aprender a escribir funciones que nos puedan ser de futura utilidad es una de las mejores formas de conseguir que el uso de R nos sea cómodo y productivo

CONCEPTO DE FUNCIÓN



CREACIÓN DE UNA FUNCIÓN

- Crearemos un script nuevo, y lo guardaremos en este caso con el nombre funcion-sumar.R (puede ser cualquier nombre, cuidado con los caracteres "raros")

```
# Función para calcular la suma de dos números
sumaNumeros <- function(a,b){
 a + b
}
```

- Las variables que están entre paréntesis se llaman argumentos de entrada de la función.
- La última instrucción entre corchetes, será lo que devuelva la función como resultado, en nuestro caso la suma.
- También podemos devolver el resultado con `return()` y el resultado entre los paréntesis

CREACIÓN DE UNA FUNCIÓN

- Para poder llamar a la función que hemos creado, debemos cambiar el directorio de trabajo al del fichero. Para ello
  - Vamos a la ventana de utilidades, pestaña Files
  - Seleccionamos la carpeta donde hemos guardado nuestro archivo
  - Pulsamos sobre More y Set as Working Directory
- Ahora el directorio de trabajo será el mismo que el de la función, y la podremos cargar haciendo `source("funcion-sumar.R")`
- Hay otros modos de poder hacer esto, entre ellos escribir `source(RUTA_COMPLETA)` y entonces no tendremos que cambiar el directorio de trabajo.
- Podemos hacer una prueba
  - `a<-sumaNumeros(1,2) # nos guarda 3 en a (llamamos según el orden de los argumentos)`
  - `sumaNumeros(b=12,a=5) # nos muestra 17 (llamamos pasando los argumentos por nombre)`

EJECUCIÓN CONDICIONAL IF ELSE

- El lenguaje R tiene la posibilidad de ejecutar expresiones condicinalmente:
- `if(expre1){ expre2 } else { expre3 }`
  - Si la expresión expre1 es verdadera, ejecutará expre2 y si es falsa ejecutará expre3
  - `x <- 1`
  - `if(x>0){`
  - `  cat("El número es positivo")`
  - `}else{`
  - `  cat("El número es negativo o cero")`
  - `}`
- Hay una versión compacta `ifelse(condición,dev1,dev2)` que devuelve dev1 si la condición es verdadera, o dev2 si es falsa.

BUCLES Y CICLOS

- Hay órdenes que permiten ejecutar de forma repetitiva las instrucciones que contienen. Si es posible, es preferible utilizar otras funciones vectorizadas como `apply()`, `tapply()`, `sapply()`
- Si queremos utilizar bucles podemos emplear:
  - for
  - while

BUCLES Y CICLOS

- `for (name in values){ exp }`
  - Ejecuta el conjunto de expresiones entre corchetes, una vez por cada uno de los valores de values, en cada iteración (cada vez que se ejecuta), la variable name tomará cada uno de los valores en órden.
  - `for(i in 1:10) { cat("Iteración ",i,"\n") } # “\n” indica salto de línea`
- `while (condicion){ exp }`
  - Ejecutará las expresiones entre corchetes mientras la condición sea cierta
  - `i<-1`
  - `while(i<10){cat("Iteración ",i,"\n") ; i<-i+1}`
- Podemos utilizar break para salir de cualquier bucle (no recomendado)
- Podemos utilizar next para forzar el comienzo de una nueva iteración (a evitar si es posible)

RECURSOS

- Curso de introducción al entorno R:
  - http://www.uv.es/conesa/CursoR/cursoR.html (parte de los materiales del presente documento obtenidos de esta página)
- R para principiantes
  - https://cran.r-project.org/doc/contrib/rdebuts_es.pdf
- An introduction to R
  - https://cran.r-project.org/doc/manuals/R-intro.pdf
- Infinidad de materiales en Internet