

TECNOLOGÍA

DANIEL LIVIANO
MARIA PUJOL

ANÁLISIS CUANTITATIVO CON R

MATEMÁTICAS, ESTADÍSTICA Y ECONOMETRÍA



EDITORIAL UOC

 10 h

Análisis cuantitativo con R

Análisis cuantitativo con R

Matemáticas, estadística y
econometría

Daniel Liviano Solís
Maria Pujol Jover



EDITORIAL UOC

Director de la colección Manuales (Tecnología): Antoni Pérez

Diseño de la colección: Editorial UOC

Diseño de la cubierta: Natàlia Serrano

Primera edición en lengua castellana: mayo 2017

Primera edición digital: junio 2017

© Daniel Liviano Solís y Maria Pujol Jover, del texto

© Editorial UOC (Oberta UOC Publishing, SL), de esta edición, 2017

Rambla del Poblenou, 156,

08018 Barcelona

www.editorialuoc.com

Realización editorial: d ctilos

Impresi n: Prodigitalk

ISBN: 978-84-9116-804-1

La UOC queda facultada expresamente por el/la autor/a para digitalizar y publicar la Obra en un repositorio en l nea que ser  accesible al p blico bajo licencias Creative Commons, incluyendo la licencia ReconocimientoNoComercial-SinObraDerivada (BYNC-ND), v.4.0 Internacional (jurisdicci n internacional), que permitir  copiar, distribuir y transmitir p blicamente la Obra siempre citando la autor a y la fuente, sin hacer un uso comercial y sin hacer obra derivada. Si la Obra es transformada, la obra generada estar  sometida a una licencia Creative Commons similar o compatible con la licencia mencionada.

Autores

Daniel Liviano Solís

Licenciado en Administración y Dirección de Empresas por la Universidad Rovira i Virgili (URV) de Tarragona y doctorado por la misma universidad. Ha impartido clases en la Universitat Oberta de Catalunya (UOC), la URV y la Universidad de Barcelona (UB), en las siguientes materias: economía, finanzas, estadística, econometría y matemáticas financieras. Además, investiga en los campos de la economía regional y la demografía. Desde 2009 es profesor de los Estudios de Economía y Empresa de la UOC, y actualmente también forma parte del Área de Business Intelligence y Big Data de la UOC.

Maria Pujol Jover

Doctora en Estudios Empresariales por la Universidad de Barcelona. Desde su etapa de doctoranda se ha focalizado en el tratamiento, la gestión y el análisis de datos, con diversas contribuciones donde las técnicas estadísticas y el trabajo con grandes bases de datos son los protagonistas. Su experiencia va desde la investigación en marketing —incluyendo su propia tesis doctoral— hasta el eGovernment, pasando por la investigación en learning analytics. Además de dedicarse a la docencia de técnicas estadísticas y econométricas, forma parte de los Estudios de Economía y Empresa y del Área de Business Intelligence y Big Data de la UOC.

«Qué cosa más asombrosa es un libro. Es un objeto plano hecho de un árbol con partes flexibles en el que se imprimen un montón de oscuros y divertidos garabatos. Pero una mirada al libro y estás dentro de la mente de otra persona, tal vez alguien muerto hace miles de años. A través de los milenios, un autor está hablando con claridad y en silencio dentro de tu cabeza, directamente a ti. La escritura es quizás el mayor de los inventos humanos, que une a las personas que nunca se conocieron entre sí, ciudadanos de épocas distantes. Los libros rompen las cadenas del tiempo. Un libro es una prueba de que los seres humanos son capaces de hacer magia.»

Carl Sagan

Índice general

Introducción	13
I Estructura, lenguaje y sintaxis	19
1 Primeros pasos con R.	19
1.1 Instalación	19
1.2 Iniciar una sesión	20
1.3 Gestión de paquetes	22
1.4 ¡Ayuda!	24
2 Sintaxis y programación	25
2.1 Tipos de datos y objetos	25
2.2 Vectores	26
2.3 Matrices	33
2.4 Funciones.	42
2.5 Ciclos y condicionales	44
2.6 La familia apply	45
2.7 Bases de datos	48
2.8 Listas	51
2.9 Fórmulas	53
2.10 Factores	54
3 Importación y exportación de datos	57
4 Extensiones de R.	59
4.1 R-Commander	60
4.2 RStudio	65
Bibliografía	66

II	Álgebra lineal y cálculo	67
1	Motivación	67
2	Álgebra vectorial y matricial.	68
2.1	Operaciones algebraicas.	68
2.2	Ecuaciones algebraicas	79
2.3	Sistemas de ecuaciones lineales	80
2.4	Descomposición matricial	82
3	Funciones.	85
3.1	Clases de funciones	86
3.2	Representación gráfica	87
4	Cálculo diferencial e integral.	95
4.1	Aproximación a la derivada	96
4.2	Integración	103
4.3	Ecuaciones diferenciales	109
5	Optimización	114
6	Análisis de variable compleja	117
	Bibliografía y librerías adicionales	122
III	Análisis de datos y estadística descriptiva .	127
1	Motivación	127
2	Importación de datos.	127
3	Visualización y manejo de datos	129
4	Cálculo de estadísticos	135
5	Representación gráfica	140
	Bibliografía y librerías adicionales	146
IV	Distribuciones de probabilidad.	151
1	Motivación	151

2	Variables aleatorias y distribuciones asociadas	151
3	Distribuciones de probabilidad discretas	154
3.1	Permutaciones, combinaciones y el cociente binomial	154
3.2	Distribución binomial	156
3.3	Distribución geométrica.	159
3.4	Distribución hipergeométrica	161
3.5	Distribución de Poisson.	162
4	Distribuciones de probabilidad continuas	164
4.1	Distribución uniforme	164
4.2	Distribución exponencial	166
4.3	Distribución normal	168
4.4	Distribución t de Student	174
4.5	Teorema del límite central	176
	Bibliografía y librerías adicionales	178
V	Inferencia estadística	181
1	Inferencia sobre parámetros estadísticos	181
1.1	Motivación	181
1.2	Inferencia sobre la media poblacional	182
1.3	Inferencia sobre la varianza poblacional	188
1.4	Inferencia sobre la proporción	190
2	Modelos de regresión	194
2.1	Motivación	194
2.2	Especificación de modelos.	196
2.3	Estimación de modelos	200
2.4	Diagnóstico de modelos.	205

3	Análisis de la varianza (ANOVA).	209
	Bibliografía y librerías adicionales	212
VI	Programación y simulación	215
1	Motivación	215
2	Generación de valores aleatorios	216
3	Aplicación empírica: sesgo del estimador de la varianza muestral	218
4	Aplicación empírica: análisis del teorema central del límite	223
4.1	Definición	223
4.2	Aproximación de una distribución totalmente aleatoria	224
4.3	Aproximación de diferentes distribuciones a la normal.	230
	Bibliografía	235
	Bibliografía	237

Introducción

Tanto las matemáticas como la estadística son fundamentales en muchos planes de estudio: Física, Ingenierías, Ciencias Sociales, Turismo, etc. Por eso mismo, es tan importante conocer la teoría subyacente a los métodos y técnicas incluidas en estas disciplinas como saberla aplicar en la vida real. Es un hecho que hoy en día no se concibe el estudio de una asignatura cuantitativa —Matemáticas, Estadística, Física, etc.— sin un soporte computacional que facilite los cálculos que se deben realizar. De ahí, la importancia que normalmente adquieren los programas informáticos en el momento en el que los estudiantes optan por matricularse en este tipo de asignaturas. En este sentido, recientemente ha habido un gran desarrollo de programas informáticos destinados a realizar análisis numéricos de datos empíricos. Los más interesantes son los que se distribuyen bajo una licencia GNU General Public License, es decir, aquellos que no implican coste económico alguno y que son una clara alternativa a los programas informáticos privados más tradicionales, que suponen el pago de licencias para su uso. Uno de los programas que está ganando cada día mayor popularidad en la comunidad científica y docente es R. Esto se debe a su potencial y polivalencia.

Este manual pretende ser un complemento al material teórico de las asignaturas cuantitativas que se ofrecen en los distintos planes de estudio de muchos grados. Para ello, se ofrece una introducción al programa R. En su definición más amplia, R es un lenguaje de programación que permite al usuario

programar algoritmos y usar herramientas programadas por otros. Si nos limitamos al análisis estadístico y cuantitativo, podemos afirmar, sin miedo a exagerar, que R no tiene límites, es decir, es capaz de hacer cualquier cosa que el usuario pueda imaginar. Con R se pueden escribir funciones, hacer cálculos, aplicar técnicas estadísticas complejas e implementar técnicas *ad hoc* —la manera más sencilla de aplicar técnicas complicadas es aprovechar las que ya están disponibles en la red y, en caso necesario, optar por desarrollarlas nosotros mismos—, crear gráficos simples y complejos, e incluso escribir funciones propias. El hecho de que muchos centros de investigación y universidades lo estén usando supone que miles de contribuciones estén disponibles a todos los usuarios. Además, hay abundante material de alta calidad disponible gratuitamente —libros, manuales, ejemplos, etc. Una ventaja nada despreciable es que, a diferencia de muchos programas estadísticos, R no tiene coste alguno, ya que es un proyecto GNU, es decir, de software libre.

The R Project for Statistical Computing

En la página web oficial <http://www.r-project.org>, encontraremos en inglés toda la información que necesitamos sobre este programa: archivos de instalación, características, paquetes disponibles, manuales, etc. Para encontrar manuales en castellano, deberemos ir a la página <http://cran.es.r-project.org/> e ir a la sección «Contributed».

En este contexto, a modo de resumen, las principales ventajas de R se pueden resumir en la siguiente lista:

- Es un programa distribuido libremente bajo una licencia GNU.
- Existe abundante material de apoyo: manuales, tutoriales y ejemplos.

- Además, existen librerías que cubren casi todas las metodologías de la estadística y las matemáticas —optimización, series temporales, programación lineal, ecuaciones diferenciales, inferencia, etc.
- Permite sistematizar análisis largos y complejos en unas pocas instrucciones sintéticas. Así pues, así un programa muy eficiente.

No obstante, también podemos enumerar algunos inconvenientes:

- Existe una considerable barrera de entrada o curva de aprendizaje.
- No es el tipo de programa con ventanas y menús, como lo pueden ser Minitab, E-Views o SPSS. No obstante, existen extensiones de R, como R-Commander, que sí se basan en el uso de ventanas y menús y que, por tanto, minimizan este inconveniente.
- Aprender a usar R, al principio, requiere primero aprender a hacer programación.

Si bien es cierto que R fue inicialmente concebido para cubrir las necesidades de la estadística como disciplina, también permite el cálculo matricial y técnicas procedentes de otras disciplinas. Por tanto, en la actualidad R ofrece una versatilidad que permite su uso en cualquier disciplina que requiera el manejo de datos para obtener resultados. Además, es importante destacar que R está en constante evolución, ya que la globalización y el trabajo colaborativo de miles de usuarios, junto con el hecho de tratarse de un programa libre, permiten agilizar al máximo sus actualizaciones y minimizar el tiempo de lanzamiento de nuevas versiones. Esto garantiza que el usuario de R esté constantemente al día.

R permite el uso de un lenguaje de programación para realizar el estudio que el usuario desee: desde obtener gráficos y resultados de procedimientos estadísticos, hasta la creación de nuevas funciones destinadas a resolver situaciones concretas y la modelización mediante nuevos algoritmos diseñados *ad hoc* para un problema determinado. R no se caracteriza únicamente por tener licencia libre, y ser muy potente y polivalente. Además, se trata de un entorno de muy fácil instalación y es multiplataforma, es decir, que ofrece diversas interfaces que se pueden adaptar a las preferencias de cada usuario.

Este manual se compone de seis capítulos cuyo contenido específico se detalla a continuación:

- Capítulo 1. Contiene la introducción al programa R. Se empieza por la instalación y se repasan los principales aspectos de su lenguaje de programación.
- Capítulo 2. Cubre los aspectos de R relacionados con el álgebra lineal y el cálculo. Esto incluye operaciones algebraicas básicas, transformaciones matriciales, diferenciación e integración, ecuaciones diferenciales, optimización y análisis de variable compleja.
- Capítulo 3. Se hace una introducción al análisis estadístico descriptivo y gráfico mediante ejemplos ilustrativos.
- Capítulo 4. Se centra en el estudio de las distribuciones de probabilidad discretas y continuas.
- Capítulo 5. Está dedicado a la inferencia estadística: intervalos de confianza y contrastes de hipótesis, modelos de regresión y ANOVA.

- Capítulo 6. Este último capítulo se centra en aspectos de simulación avanzada y programación. Para ello, se analizan dos casos prácticos: la estimación del sesgo de la varianza muestral y el Teorema Central del Límite.

Capítulo I

Estructura, lenguaje y sintaxis

1. Primeros pasos con R

1.1. Instalación

Los vídeos son útiles

En internet podéis encontrar multitud de vídeos en inglés y en castellano sobre cómo instalar R. Muchos de estos están disponibles en www.youtube.com.

El primer paso obligado es la instalación del programa y para ello tendremos que visitar la página web del proyecto:

<http://www.r-project.org/>

Una vez allí, vemos que hay varias secciones, una de las cuales es el centro de descargas CRAN. Si accedemos a él, tendremos en pantalla una lista de *mirrors*, es decir, servidores desde donde descargar los archivos. Seleccionaremos el servidor que esté más próximo geográficamente a nuestra ubicación y nos aparecerá una página con diversas opciones de instalación. La manera más rápida es seleccionar una de las siguientes opciones, que dependerá de nuestro sistema operativo. Si este no es Windows y tenemos un Mac o una distribución de Linux, deberemos tener en cuenta la versión del sistema operativo que tenemos instalado.

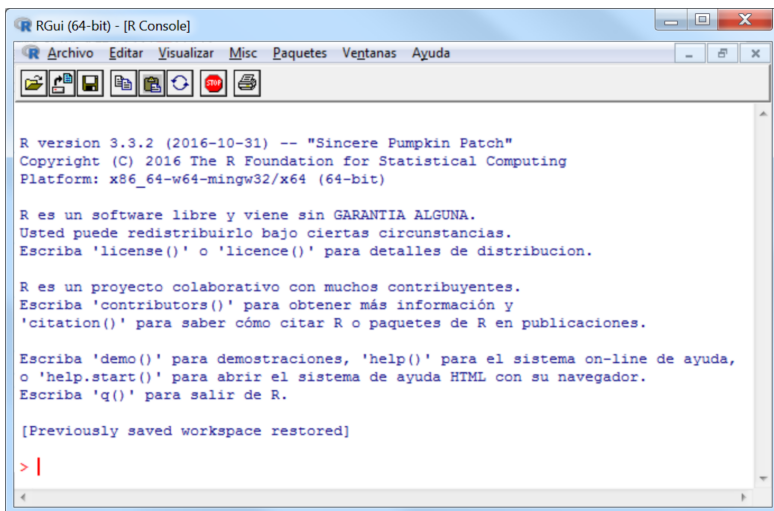
- Download R for Linux
- Download R for Mac OS X
- Download R for Windows

Suponiendo que tenemos un sistema operativo Windows, accedemos y descargamos la distribución base, a la cual se accede mediante la opción *install R for the first time*. Seguimos las instrucciones y descargamos el archivo ejecutable, tras lo que procedemos a la instalación sin más complicaciones.

1.2. Iniciar una sesión

Una vez instalado, ya podemos abrir el programa, ya sea mediante el acceso directo o mediante el archivo ejecutable incluido en la carpeta de instalación de R. De cualquier modo, al ejecutar R nos encontraremos con la ventana mostrada en la figura I.1.

Figura I.1: Consola de R



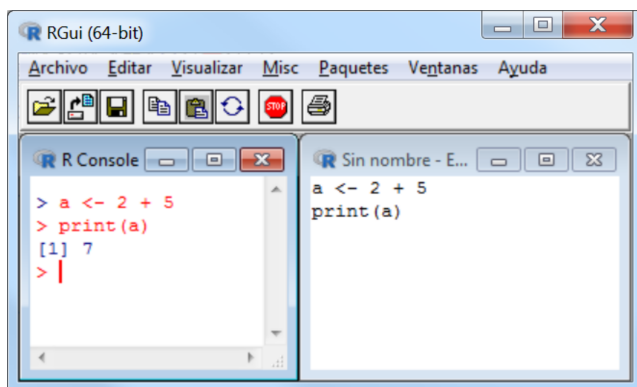
Como podemos ver, las instrucciones se pueden introducir directamente en la consola, en el cursor que aparece tras el símbolo `>`. Una vez introducidas las instrucciones, pulsando *Enter* se ejecutarán los cálculos, como si fuera una calculadora:

```
> 2 * 3
[1] 6
```

Sin embargo, a la hora de realizar un análisis largo, lo más conveniente es escribir el conjunto de instrucciones, con los cálculos que necesitamos, en un archivo de texto. Esto se puede hacer de dos maneras:

- En un documento de texto (*script*), lo cual tiene la ventaja de que puede abrirse y guardarse como un archivo con formato *.R* mediante la barra de menús del programa. Esta opción, que se muestra en la figura I.2, permite ejecutar cada línea de código o toda una selección de líneas mediante la instrucción *CONTROL + R* y aparecerá el resultado en la consola de color azul.

Figura I.2: Consola de R con editor de texto



El editor de textos gedit

gedit es un editor de textos compatible con Linux, Mac OS X y Windows. Está diseñado como un editor de textos de propósito general, y enfatiza la simplicidad y facilidad de uso. Además, incluye herramientas para la edición de código fuente y textos estructurados. Se puede descargar libremente desde la dirección projects.gnome.org/gedit.

- En un documento de texto aparte. Una vez se ha terminado de escribir, se copia y se pega en la consola. El resultado será el mismo que el de la ventana izquierda de la figura I.2, es decir, aparecerán las instrucciones y los resultados. Si optamos por esta opción, cualquier editor de textos servirá, aunque aquí recomendamos el editor *gedit*, ya que permite resaltar la sintaxis de las instrucciones.

1.3. Gestión de paquetes

El programa R es modular, es decir, se basa en distintos módulos o paquetes (*packages*) que los usuarios han elaborado. Estos paquetes son unidades específicas que proporcionan funciones y datos para poner en práctica análisis específicos. Actualmente, existen tantos paquetes que sería inviable, por motivos de memoria, cargarlos todos simultáneamente en una sesión. La lista completa de paquetes se encuentra en la página del CRAN, en el apartado «Packages». Allí podremos visualizar un total de más de 10.000 paquetes disponibles por orden alfabético y acompañados con una breve descripción. Es muy útil saber que tenemos a nuestra disposición un listado por categorías: matemáticas, estadística, econometría, etc. Accediendo a cada paquete obtendremos más información acerca

de este, así como distintos manuales de uso con las funcionalidades que incluyen. Es importante saber acudir a este tipo de herramientas cuando las necesitemos. Por eso, antes de intentar implementar nosotros mismos un complejo procedimiento numérico o estadístico, vale la pena comprobar si alguien ya ha hecho el trabajo. Por eso mismo, es muy recomendable visitar el repositorio de paquetes de R (<http://cran.r-project.org/>), en el que podemos buscar paquetes por orden alfabético o por categorías.

Cuando se abre una sesión de R se cargan automáticamente las funcionalidades básicas —denominadas *base*— y si el usuario está interesado en una metodología específica, deberá instalar el paquete correspondiente y cargarlo. Esto lo podemos hacer mediante el menú de la consola de R, mostrado en la figura I.1, en la opción *Paquetes*. Esta opción nos permite seleccionar un paquete de la lista que aparece, instalarlo y/o cargarlo. Otra opción es hacer esto manualmente mediante la consola. Si conocemos el nombre del paquete, para instalarlo introduciremos:

```
> install.packages("paquete")
```

Una vez instalado, bastará con cargarlo al principio de cada sesión en la que necesitemos este paquete mediante la siguiente instrucción:

```
> library(paquete)
```

Es importante recordar que, una vez instalado un paquete, este ya queda almacenado en nuestro ordenador, de manera que no hay que volverlo a instalar. Solo tendremos que cargarlo en las sesiones que lo necesitemos. Además, un análisis bien planificado incluirá al principio una lista de todos los pa-

quetes que hay que cargar para llevar a cabo los análisis que deseemos hacer.

1.4. ¡Ayuda!

Para un usuario novel, la interfaz de R, la sintaxis y, en general, el modo de uso pueden resultar complejos, especialmente en comparación con otros programas estadísticos. Sin embargo, R cuenta con muchos manuales y herramientas de apoyo y soporte al alcance de los usuarios. Para empezar, en la página del CRAN desde la que hemos descargado el archivo de instalación hay una sección llamada «Manuals». Si accedemos a ella, tendremos acceso a los manuales oficiales de R, el primero de los cuales es *An Introduction to R*, un completísimo manual muy bien estructurado que cubre todos los aspectos elementales del programa. Si deseamos manuales en otros idiomas, en el enlace *contributed documentation* disponemos de una lista ordenada por idiomas, entre ellos el castellano.

Además, existe la posibilidad de obtener ayuda específica para cada paquete y función. Simplemente hay que introducir en la consola la instrucción `help()`, con el nombre del paquete o la función sobre la que necesitamos ayuda entre los símbolos de paréntesis, y se nos abrirá una ventana con una completa página de ayuda con descripciones y ejemplos.

¡Cuidado con las mayúsculas!

Hay que tener en cuenta que el lenguaje de R distingue entre mayúsculas y minúsculas. De esta manera, la instrucción `HELP()` generaría un mensaje de error.

2. Sintaxis y programación

2.1. Tipos de datos y objetos

R es un lenguaje orientado a objetos o estructuras de datos diferenciadas con características específicas. Los tres tipos básicos de datos son:

- 1) Datos numéricos: números reales y complejos $(-1, 2, 3)$.
- 2) Caracteres: cadenas de texto representadas entre comillas ("x", "y", "z").
- 3) Datos lógicos: aquellos que únicamente toman los valores verdadero o falso (*TRUE*, *FALSE*).

Introducción de caracteres entre comillas

Es importante tener en cuenta que R interpreta de la misma manera las comillas simples 'x' que las dobles "x". Por tanto, podemos introducir las cadenas de texto de cualquiera de las dos maneras.

Esta orientación a objetos hace indispensable que conozcamos los diferentes tipos de objetos con los que podemos trabajar, ya que cada uno tiene características únicas. Sin embargo, como veremos en este capítulo, R nos permite transformar algunos de ellos en otros —por ejemplo, convertir vectores en matrices, y viceversa. Las principales clases de objetos están enumeradas y definidas a continuación:¹

- 1) *Data frame*: estructura de datos bidimensionales, donde se recogen diferentes variables por columnas.

¹Hemos optado por no traducir del inglés estos conceptos, ya que esto traería confusión.

- 2) *Vector*: colección ordenada de datos con una longitud determinada.
- 3) *Array*: conjunto de datos indexado por dos o más índices, caracterizado por tener dimensión.
- 4) *Matrix*: caso particular de *array* cuando hay dos índices, es decir, dos dimensiones —filas y columnas.
- 5) *List*: objeto que recoge varios tipos de elementos (componentes), que pueden ser de clases diferentes.
- 6) *Function*: pieza de código que puede ser un algoritmo, una función matemática o una estructura lógica.
- 7) *Factor*: vector que especifica una clasificación discreta de los elementos de otro vector de la misma longitud.
- 8) *Formula*: especificación con variables que recoge la relación funcional entre una variable explicada y una o varias variables explicativas.

A continuación, profundizaremos en cada una de estas estructuras.

2.2. Vectores

2.2.1. Almacenar datos en vectores

Hay dos maneras de introducir la información en el programa. Si introducimos información mediante expresiones sin asignarlas a ningún objeto, estamos utilizando el programa como una simple calculadora. En cambio, si la almacenamos y le damos un nombre, estamos guardando el resultado para utilizarlo posteriormente. Estas dos funcionalidades básicas quedan explicadas a continuación:

- 1) *Expresión*: la información se introduce en la consola, R la evalúa y la muestra en pantalla, pero no la almacena en la memoria.

```
> 4 + 3  
[1] 7
```

Cuestión de memoria

Cuando asignamos un nombre a un objeto, este quedará guardado en la memoria temporal. Esto es, podemos recuperar la información de dichos objetos siempre y cuando no cerremos la sesión de R.

- 2) *Asignación*: en este caso, la expresión se evalúa, pero no se muestra, y se almacena con el nombre que escribamos antes del símbolo de asignación `<-`. Para ver su valor en pantalla, deberemos introducir el nombre que hemos dado al vector, que es la estructura de datos más simple. La función `print` también muestra el resultado en pantalla. La manera más inmediata de introducir los valores en un vector es mediante la instrucción `c`, que significa *concatenar*.

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

Si introducimos una asignación entre paréntesis, además de quedar guardada en la memoria temporal, también se mostrará en pantalla.

```
> (b <- c(3,6,9))  
[1] 3 6 9
```

El espacio no existe

En el lenguaje de R los espacios no se interpretan, es decir, las expresiones `a <- 6` y `a<-6` son análogas. No hay ninguna diferencia entre ellas.

Una vez tenemos varios vectores creados, los podemos combinar de una manera muy flexible. Veamos un ejemplo:

```
> a <- c(1,2,3)
> b <- c(7,8,9)
> ab <- c(a,4,5,6,b)
> ab
[1] 1 2 3 4 5 6 7 8 9
```

2.2.2. Operaciones básicas

En la tabla I.1 se recogen las operaciones aritméticas básicas tomando como ejemplo los vectores **v1** y **v2**. Estas operaciones están vectorizadas, es decir, se aplican elemento a elemento.

Tabla I.1: Operaciones aritméticas básicas (I)

v1 <- c(1,2,3)		
v2 <- c(2,4,6)		
Descripción	Instrucciones	Resultado
Concatenación	c(v1,v2)	1,2,3,2,4,6
Suma	v1+v2	3,6,9
Resta	v1-v2	-1,-2,-3
Multiplicación	v1*v2	2,8,18
División	v1/v2	0.5,0.5,0.5
Potencia	v1^v2	1,16,729

Uso de los puntos y las comas

Cabe señalar que en R las comas quedan reservadas para la separación de los distintos valores o componentes de un mismo objeto o para los distintos parámetros de una función. Además, es fundamental tener en cuenta que, en R, los decimales se expresan mediante un punto y no con una coma, como se hace en España. Confusiones en este sentido pueden llevar a errores.

Cuando se intentan hacer operaciones matemáticas no definidas, como por ejemplo la raíz de un número negativo, aparecerá el valor `NaN`, que significa *Not a Number*. Además, los valores $\pm \text{Inf}$ aparecerán cuando el resultado tienda a $\pm\infty$. Veamos en la tabla I.2 más operaciones matemáticas elementales aplicadas elemento a elemento, tomando como ejemplo el vector `v3`.

Tabla I.2: Operaciones aritméticas básicas (II)

<code>v3 <- c(-1,0,1,2)</code>		
Descripción	Instrucciones	Resultado
Raíz cuadrada	<code>sqrt(v3)</code>	NaN 0.00 1.00 1.41
Valor absoluto	<code>abs(v3)</code>	1 0 1 2
Exponencial	<code>exp(v3)</code>	0.36 1.00 2.71 7.38
Log. base e	<code>log(v3)</code>	NaN -Inf 0.00 0.69
Log. base 10	<code>log10(v3)</code>	NaN -Inf 0.00 0.30
Factorial (!)	<code>factorial(v3)</code>	NaN 1 1 2

Con R podemos realizar las operaciones estadísticas básicas. En la tabla I.3 se recogen las más usuales, tomando como ejemplo los vectores `v4` y `v5`.

Tabla I.3: Operaciones estadísticas básicas

<code>v4 <- c(-2,3,5,0,0,3)</code> <code>v5 <- c(20,14,51,76,21,30)</code>		
Descripción	Instrucciones	Resultado
Longitud	<code>length(v4)</code>	6
Máximo	<code>max(v4)</code>	5
Mínimo	<code>min(v4)</code>	-2
Suma	<code>sum(v4)</code>	9
Producto	<code>prod(v4)</code>	0
Media	<code>mean(v4)</code>	1.5
Mediana	<code>median(v4)</code>	1.5
Desviación estándar	<code>sd(v4)</code>	2.58
Varianza	<code>var(v4)</code>	6.7
Covarianza	<code>cov(v4,v5)</code>	5.8
Correlación	<code>cor(v4,v5)</code>	0.09
Producto escalar	<code>sum(v4,v5)</code>	347

2.2.3. Secuencias

Una herramienta muy útil es la creación de secuencias, lo cual se puede hacer de diferentes maneras, entre las cuales se encuentran las funciones **seq** (*sequence*), **rev** (*reverse*) y **rep** (*repeat*). El dominio de estas instrucciones nos ahorrará muchas líneas de programación y nos permitirá programar complejos procedimientos numéricos de una forma simple, compacta y elegante. La tabla I.4 lo ilustra con algunos ejemplos, partiendo del vector **v6**.

Tabla I.4: Secuencias y repeticiones

v6 <- c(1,2,3)	
Instrucciones	Resultado
rep(v6,2)	1 2 3 1 2 3
rep(v6,each=2)	1 1 2 2 3 3
rep(v6,each=2,times=2)	1 1 2 2 3 3 1 1 2 2 3 3
rep(v6,c(2,3,4))	1 1 2 2 2 3 3 3 3
1:10	1 2 3 4 5 6 7 8 9 10
seq(1,10)	1 2 3 4 5 6 7 8 9 10
10:1	10 9 8 7 6 5 4 3 2 1
seq(10,1)	10 9 8 7 6 5 4 3 2 1
seq(4,length=8)	4 5 6 7 8 9 10 11
seq(0,40,by=5)	0 5 10 15 20 25 30 35 40
seq(0,1,length=4)	0.00 0.33 0.66 1.00
seq(from=2,to=8,by=3)	2 5 8
rev(1:5)	5 4 3 2 1
2*1:4	2 4 6 8

Además, existe una función que permite crear secuencias en varios vectores. Supongamos que deseamos crear dos vectores encadenados, el primero de los cuales haga referencia a un año y el segundo a un trimestre. Para este tipo de secuencias utilizaremos la función **expand.grid**. Supongamos que disponemos de datos con ocho observaciones, cuya primera observación corresponde al primer trimestre de 1996 y la última observación corresponde al cuarto trimestre de 1997. Las ocho observaciones temporales de calcularían de la siguiente manera.

```
> expand.grid(trim=1:4, año=1996:1997)
  trim año
1     1 1996
2     2 1996
3     3 1996
4     4 1996
5     1 1997
6     2 1997
7     3 1997
8     4 1997
```

2.2.4. Operadores lógicos

R también nos permite operar con vectores lógicos que pueden tomar los valores **TRUE** (verdadero) y **FALSE** (falso). Los principales operadores se muestran en la tabla I.5.

Tabla I.5: Operadores lógicos

Descripción	Operador
Menor	<
Mayor	>
Igual	==
No igual	!=
Menor o igual	<=
Mayor o igual	>=
Intersección	$a \& b$
Unión	$a b$
Negación	!a

Podemos interpretar el operador lógico intersección como a y b , mientras que la unión sería interpretable como a o b . Veamos algunos ejemplos del uso de estos operadores:

```
> a <- c(-2,-1,0,1,2)
> a >2
```

```
[1] FALSE FALSE FALSE FALSE FALSE
> a <= -1
[1] TRUE TRUE FALSE FALSE FALSE
> a >0 & abs(a) == 2
[1] FALSE FALSE FALSE FALSE TRUE
> a >0 | abs(a) == 2
[1] TRUE FALSE FALSE TRUE TRUE
```

Los símbolos = y ==

Cuando usamos el operador lógico igualdad, es importante tener en mente que se expresa mediante un doble símbolo de igualdad (==), diferente del símbolo de igualdad simple (=) utilizado en asignaciones.

2.2.5. Indexación

Para acceder a los elementos de un vector, existe la posibilidad de *indexar*, es decir, acceder a los valores de un vector especificándolo en una expresión entre corchetes []. Esto lo hacemos indicando la posición de los elementos o usando operadores lógicos. Un ejemplo, considerando el vector `v7`, se recoge en la tabla I.6.

Tabla I.6: Indexación de vectores

v7 <- c(-3,-2,-1,0,1,2,3) o de forma más compacta v7 <- c(-3:3)		
Expresión	Resultado	Descripción
v7[1]	-3	Primer elemento
v7[2:4]	-2 -1 0	Del segundo al cuarto elemento
v7[-(2:length(v7))]	-3	Todos menos aquellos desde el segundo hasta el último elemento
v7[c(2,4,6)]	-2 0 2	Elementos segundo, cuarto y sexto
v7[v7>0]	1 2 3	Elementos mayores que 0
v7[v7!=0]	-3 -2 -1 1 2 3	Elementos diferentes de 0
v7[abs(v7)==3]	-3 3	Elementos con valor absoluto 3
v7[v7>0 & v7!=2]	1 3	Elementos positivos y que no sean 2
v7[v7<0 v7==3]	-3 -2 -1 3	Elementos negativos o que sean 3

Además, la función `which` nos permite extraer la posición ordinal de los elementos descritos en la condición introducida. En el siguiente ejemplo, en el primer caso los elementos del vector `a` iguales a cero son el primero, tercero y sexto. En el segundo caso, el único elemento mayor que cero (`a>0`) y menor o igual que dos (`a<=2`) es el cuarto.

```
> a <- c(0,3,0,1,3,0)
> which(a==0)
[1] 1 3 6
> which(a>0 & a<=2)
[1] 4
```

De igual modo, las funciones `which.min` y `which.max` nos dan, respectivamente, la posición de los valores mínimos y máximos del vector.

2.3. Matrices

2.3.1. Los objetos `array` y `matrix`

A menudo tendremos la necesidad de almacenar los datos en estructuras más complejas que simples vectores. Esto es, podemos necesitar tener una estructura de filas y columnas, así como poder realizar operaciones matriciales. En R, hay dos estructuras —tipos de objetos— que satisfacen esta necesidad:

- 1) Variable indexada (`array`): estructura que organiza los datos en k dimensiones y es muy útil para tratar con datos con diversas clasificaciones simultáneas. Por ejemplo, la variable v_{ijt} puede representar las ventas de una empresa donde i es el sector, j la provincia y t el año, y se almacenaría en una estructura `array` con dimensión $k = 3$.

- 2) Matriz (`matrix`): se trata de un caso particular de *array* cuando hay dos dimensiones ($k = 2$), siendo la primera dimensión las filas y la segunda dimensión las columnas.

En muchos casos no hay diferencia entre los dos tipos de estructuras. Sin embargo, algunas operaciones, como la transposición y algunos tipos de operaciones algebraicas, están reservadas solo a las matrices. Por simplicidad, en lo que sigue nos centraremos en la creación y manipulación de estructuras bidimensionales (matrices). La manera más inmediata de crear una matriz es asignándole a un vector una dimensión, ya que, como hemos visto, los vectores tienen longitud y no dimensión:

El resultado NULL

Siempre que un objeto esté vacío aparecerá el resultado NULL. En este caso, al ser `a` un vector y no una matriz, su dimensión no existirá.

```
> a <- 1:8
> a
[1] 1 2 3 4 5 6 7 8
> length(a)
[1] 8
> dim(a)
NULL
```

2.3.2. Creación de matrices

Supongamos que deseamos convertir el vector `a` en una matriz de dimensión 4×2 , es decir, una estructura con 4 filas y 2 columnas. En este caso, podemos asignarle una dimensión:

```
> dim(a) <- c(4, 2)
> a
      [,1] [,2]
[1,]     1     5
[2,]     2     6
[3,]     3     7
[4,]     4     8
> dim(a)
[1] 4 2
```

Es importante destacar que, por defecto, R siempre empieza a colocar los valores de arriba abajo y de izquierda a derecha, como se puede observar en el ejemplo anterior.

Una manera más directa de crear matrices es mediante la función `matrix`. Esta función tiene tres argumentos: el conjunto de valores que compondrán la matriz, el número de filas y el número de columnas.

```
> matrix(1:8, 2, 4)
      [,1] [,2] [,3] [,4]
[1,]     1     3     5     7
[2,]     2     4     6     8
```

En el caso de necesitar una estructura de datos con más de dos dimensiones, necesitaremos la función `array`. La principal diferencia respecto a la función `matrix` es que el número de dimensiones se introduce mediante un vector. Crearemos, a modo de ejemplo, una estructura $2 \times 2 \times 2$:

El reciclaje es importante

Fijaos en que cuando se acaban de colocar todos los elementos del vector 1:4 en la estructura resultante y esta no está completa todavía, se vuelve a empezar de nuevo hasta que la estructura está completa. Esta funcionalidad se denomina *reciclaje*.

```
> array(1:4, c(2,2,2))

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

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

Una característica interesante de la construcción de matrices es el *reciclaje*, que consiste en que si el número total de elementos de la matriz (filas por columnas) es superior al número de valores iniciales, estos se repetirán hasta completar la matriz. En el ejemplo anterior, hemos querido transformar un vector con cuatro elementos en una estructura con $2 \times 2 \times 2 = 8$ elementos, esto es, el doble. Por eso, R ha tenido que repetir los elementos del vector inicial dos veces para completar la estructura final. Veamos otro ejemplo de reciclaje:

```
> matrix(1:3,3,3)
   [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
```

Además existe la opción de que la matriz se construya invirtiendo el orden, es decir, de izquierda a derecha y de arriba abajo. Se consigue añadiendo la función **byrow** al final, con la cual invertimos el orden de la construcción de matrices, empezando por filas en lugar de por columnas, que es la opción por defecto.

```
> matrix(1:8,2,4,byrow=TRUE)
      [,1] [,2] [,3] [,4]
[1,]     1     2     3     4
[2,]     5     6     7     8
```

También es posible especificar únicamente el número de filas y/o columnas de la matriz que queremos construir:

```
> matrix(1:6,nrow=2)
      [,1] [,2] [,3]
[1,]     1     3     5
[2,]     2     4     6
> matrix(1:4,ncol=2)
      [,1] [,2]
[1,]     1     3
[2,]     2     4
```

Es muy fácil combinar escalares, vectores y matrices. La función `rbind` une varias estructuras por filas, mientras que `cbind` hace lo propio por columnas. Veamos cómo se hace:

```
> a<-matrix(0,2,2)
> a
      [,1] [,2]
[1,]     0     0
[2,]     0     0
> b<-matrix(1,1,2)
> b
      [,1] [,2]
[1,]     1     1
> rbind(a,b)
      [,1] [,2]
[1,]     0     0
[2,]     0     0
[3,]     1     1
```

Como hemos visto, estas dos funciones nos ofrecen muchas posibilidades. Además, en el siguiente ejemplo vemos que unir un escalar y un vector provoca que el valor del escalar se repita tantas veces como la longitud del vector. Esto hace que la estructura resultante tenga dos columnas y tantas filas como elementos del vector:

```
> cbind(1,1:4)
      [,1] [,2]
[1,]    1    1
[2,]    1    2
[3,]    1    3
[4,]    1    4
```

La matriz identidad se crea mediante la función `diag`, especificando el número de filas y columnas. Recordemos que la matriz identidad siempre es cuadrada, esto es, tiene el mismo número de filas y de columnas:

```
> diag(2)
      [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Además, la función `diag` también se puede usar para acceder a los elementos de la diagonal principal de una matriz:

```
> a<-matrix(5*1:4,2,2)
> a
      [,1] [,2]
[1,]    5   15
[2,]   10   20
> diag(a)
[1]  5 20
```

2.3.3. Submatrices e indexación

Análogamente al caso de los vectores, también es posible acceder a subconjuntos de matrices mediante corchetes []. Estos se usan con tres propósitos:

- 1) Tener acceso a filas y columnas específicas.
- 2) Crear submatrices.
- 3) Modificar directamente partes de una matriz.

Esto nos ofrece una gran potencialidad para hacer todo tipo de operaciones con matrices y hacer que estas sean fácilmente replicables, modificables y combinables. A partir de la matriz **a**, la tabla I.7 muestra ejemplos de creación de submatrices.

```
> a <- matrix(1:9*3,3,3)
> a
      [,1] [,2] [,3]
[1,]    3   12   21
[2,]    6   15   24
[3,]    9   18   27
```

Tabla I.7: Indexación de matrices

Expresión	Resultado	Descripción
a[,3]	21 24 27	Todas las filas, 3. ^a columna
a[2,]	6 15 24	2. ^a fila, todas las columnas
a[-1,-3]	6 15 9 18	Todas las filas menos la 1. ^a y todas las columnas menos la 3. ^a
a[2:3,1]	6 9	De la 2. ^a a la 3. ^a fila, primera columna

Es interesante comprobar que las submatrices creadas, en caso de tener una sola fila o columna, se degradan a vectores,

con lo que pierden la dimensión. Para que estas submatrices sigan teniendo dimensión, debemos especificarlo desactivando la opción `drop`:

```
> a
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> b <- a[,1,drop=FALSE]
> b
      [,1]
[1,]    1
[2,]    2
> dim(b)
[1] 2 1
```

Veamos ahora cómo se haría para introducir elementos nuevos en una matriz. Supongamos, por ejemplo, que queremos sustituir la primera columna de valores por una columna de ceros. En este caso, utilizaremos `a[,1]<-0` para transformar la primera columna, inicialmente compuesta por los elementos 1 y 2, en una columna con dos ceros.

```
> a<-matrix(1:8,2,4)
> a[,1]<-0
> a
      [,1] [,2] [,3] [,4]
[1,]    0    3    5    7
[2,]    0    4    6    8
```

2.3.4. Operaciones matriciales básicas

R ofrece muchas posibilidades para aplicar operaciones y funciones a vectores y matrices. En la tabla I.8 se incluyen las funciones básicas aplicables a matrices.

Tabla I.8: Principales funciones con matrices

Descripción	Función
Dimensiones	<code>dim</code>
Transpuesta	<code>t</code>
Determinante	<code>det</code>
Inversa	<code>solve</code>
Valores y vectores propios	<code>eigen</code>
Valores singulares	<code>svd</code>
Factorización de Choleski	<code>chol</code>
Factorización QR	<code>qr</code>
Rango	<code>rank</code>
Número de filas	<code>nrow</code>
Número de columnas	<code>ncol</code>
Suma de las filas	<code>rowSums</code>
Suma de las columnas	<code>colSums</code>
Media de las filas	<code>rowMeans</code>
Media de las columnas	<code>colMeans</code>

Para acabar el apartado dedicado a matrices, vale la pena mencionar que hay algunas funciones que permiten, por una parte, identificar objetos y, por otra, transformarlos en objetos de otras clases. Este primer ejemplo es ilustrativo:

```
> a <- c(1,3,6)
> is.matrix(a)
[1] FALSE
> is.vector(a)
[1] TRUE
> class(a)
[1] 'numeric'
```

La familia de funciones `is.` responde a la pregunta de si se trata de un objeto determinado mediante un valor lógico, mientras que la función `class` devuelve la clase del objeto. Finalmente, la familia de funciones `as.` se utiliza para transformar un tipo de objeto en otro. El siguiente ejemplo muestra cómo hemos conseguido transformar la matriz **A** en el vector **a**:

```
> (A <- matrix(1,2,2))
      [,1] [,2]
[1,]    1    1
[2,]    1    1
> (a <- as.vector(A))
[1] 1 1 1 1
```

2.4. Funciones

Una herramienta muy útil y poderosa es el comando **function**, que se corresponde con la clase de objeto función (*function*). Podemos crear desde simples funciones matemáticas a complejos algoritmos con los que realizar multitud de cálculos de una manera estructurada y sintética. La forma más sencilla de función tiene la siguiente estructura:

```
nombre <- function (argumentos) expresión
```

Como ejemplo, supongamos que queremos estudiar la función matemática $y = x^3$ y evaluarla en el vector de valores iniciales $x_0 = (-4, -3, \dots, 3, 4)$. Lo haremos creando nuestra propia función **cubic** y evaluándola en el vector de valores iniciales **x0**:

```
> cubic <- function(x) x^3
> x0 <- (-4:4)
> cubic(x0)
[1] -64 -27 -8 -1 0 1 8 27 64
```

Un ejemplo de una función con más de un argumento es el de la fórmula financiera del interés simple $C_f = C_i (1 + rt)$. Según esta fórmula, el capital final (C_f) que hay que retornar en una operación financiera con un importe inicial C_i depende

del tiempo de duración de la operación en años (t) y del tipo de interés en tanto por uno (r). ¿Qué importe final resulta de una operación a nueve meses ($t = 0,75$) con un tipo nominal del 5% ($r = 0,05$) y capital inicial $C_i = 1.000$?

```
> op.fin <- function(Ci,r,t) Ci*(1+r*t)
> op.fin(1000,0.05,0.75)
[1] 1037.5
```

Para funciones más complejas a menudo es necesaria más de una línea para introducir las fórmulas. En este caso, suponiendo que necesitamos n líneas, la estructura será esta:

```
nombre <- function(argumentos){
expresión 1
...
expresión n
return(resultado)
}
```

Con el fin de ilustrar este tipo de estructura, crearemos una función para el cálculo de la covarianza muestral. Esto es un tanto innecesario, ya que podríamos usar la función `cov` que nos proporciona R. No obstante, lo hacemos para que sirva de ejemplo de funciones más elaboradas:

$$Cov(X, Y) = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})$$

Funciones multiusos

Crear funciones propias nos será muy útil para realizar un conjunto de cálculos que tengamos que usar en más de una ocasión, de manera estructurada y ordenada.

```
> mi.covar <- function(x,y){  
+   N <- length(x)  
+   demean.x <- x-mean(x)  
+   demean.y <- y-mean(y)  
+   sumat <- sum(demean.x*demean.y)  
+   return(sumat/(N-1))  
+ }
```

2.5. Ciclos y condicionales

R ofrece una serie de herramientas para hacer asignaciones múltiples y condicionales. Para usar eficientemente ciclos y condicionales es fundamental dominar los operadores lógicos, explicados anteriormente en este capítulo. `ifelse` permite realizar operaciones elemento por elemento a un vector sujeto a una condición. Supongamos que queremos aplicar a un vector la siguiente transformación:

$$f(x) = \begin{cases} \log(x) & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

Esto lo haríamos mediante el comando `ifelse`, indicado que, para aquellos valores $a > 0$, aplique su logaritmo y para el caso contrario —es decir, $a \leq 0$ —, muestre el valor cero.

```
> a <- c(-1,0,2,5)  
> ifelse(a>0,log(a),0)  
[1] 0.0000000 0.0000000 0.6931472 1.6094379
```

Uno de los componentes fundamentales de cualquier lenguaje de programación es el uso de estructuras iterativas, es decir, funciones que repiten una o más expresiones iterativamente en un ciclo. El principal comando que hace esta función

es `for`. La estructura general de esta función es, para un ciclo de n iteraciones:

```
for (i in 1:n){  
  expresión(es)  
  ...  
}
```

Un sencillo ejercicio para ilustrar un ciclo es la suma acumulativa. Partiendo de un vector $\vec{v} = (v_1, \dots, v_n)$, crearemos un vector $\vec{s} = (s_1, \dots, s_n)$ donde $s_j = \sum_{i=1}^j v_i$. Antes de implementar el ciclo crearemos un vector s vacío que iremos completando iterativamente:

```
> v <- 1:10  
> n <- length(v)  
> s <- rep(0,n)  
  
> for (i in 1:n){  
+   p <- v[1:i]  
+   s[i] <- sum(p)  
+ }  
  
> print(s)  
[1]  1  3  6 10 15 21 28 36 45 55
```

Obviamente, en un análisis real no usaremos este cálculo iterativo, ya que la función implementada `cumsum` ya nos ofrece esta funcionalidad de una manera más eficiente.

2.6. La familia `apply`

En la práctica, es importante no hacer un uso excesivo de los ciclos iterativos, ya que son ineficientes en cuanto al uso de la memoria y pueden ralentizar el tiempo requerido de análisis.

Por ello, la norma es que se crearán estructuras iterativas siempre que no haya alternativas. El grupo de funciones **apply** es una de ellas.

Para ilustrar este concepto, supongamos que tenemos una matriz A donde las filas corresponden a observaciones y las columnas son las variables X , Y y Z :

$$A = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{pmatrix}$$

La función **apply** nos permite aplicar cualquier operación a las filas y a las columnas, independientemente. Supongamos que deseamos aplicar la operación $\sqrt{x_i^2 + y_i^2 + z_i^2}$ a cada observación —es decir, a cada fila. El primer paso será definir esta operación —a la que llamaremos **hipot**— y el segundo, aplicarla a las filas de la matriz A :

```
> hipot <- function(x) sqrt(sum(x*x))
> apply(A,1,hipot)
```

La función **apply**

La instrucción **apply(A,1,hipot)** se lee como *aplica a la dimensión 1 (filas) de la matriz A la operación hipot*. El resultado será un vector con n elementos.

Si deseáramos un vector con la media aritmética de cada columna, es decir $(\bar{x}, \bar{y}, \bar{z})$, deberíamos introducir:

```
> apply(A,2,mean)
```

Para explicar la funcionalidad de la función de la misma familia `tapply`, es necesario introducir los factores, que son una clase de objeto que establece una clasificación discreta de una o más variables. Supongamos que estamos estudiando la estatura de los alumnos de una escuela, para lo que disponemos de dos vectores: uno con la estatura de cada estudiante y otro con el género —masculino m y femenino f . Así pues:

Introducción de caracteres

A la hora de introducir un vector de caracteres, como `genero`, los valores del vector han de ir entre comillas, para lo cual son válidos los símbolos " y ' indistintamente.

```
> altura <- c(131,125,126,140,152,119)
> genero <- c('m','m','f','m','f','f')
```

En este caso es conveniente convertir el vector `genero` en un factor con dos categorías: masculina y femenina. Para ello, empleamos la función `factor` para obtener un vector de factores con las categorías $f = 1$ y $m = 2$, al que llamaremos `f.genero`. Veamos que son objetos de diferente clase:

```
> f.genero <- factor(genero)
> class(genero)
[1] "character"
> class(f.genero)
[1] "factor"
```

La instrucción `levels` devuelve las categorías que el vector incluye:

```
> levels(f.genero)
[1] 'f' 'm'
```

```
> cbind(f.genero, altura)
      f.genero altura
[1,]         2    131
[2,]         2    125
[3,]         1    126
[4,]         2    140
[5,]         1    152
[6,]         1    119
```

Un análisis estadístico de la variable *altura* deberá diferenciar entre *f* y *m*, para lo cual se emplea la función `tapply`. Calcularemos la media aritmética y la desviación estándar de la altura diferenciando por género:

La función `tapply`

Utilizaremos la función `tapply` cuando queramos aplicar un cálculo a una variable diferenciando por grupos o segmentos, esto es, según las categorías de un factor.

```
> tapply(altura, f.genero, mean)
      f          m
132.3333 132.0000

> tapply(altura, f.genero, sd)
      f          m
17.387735  7.549834
```

2.7. Bases de datos

Una base de datos (*data frame*) es similar a una matriz, ya que es una estructura bidimensional donde las filas son las observaciones y las columnas son las variables, cada una con

un nombre específico. Es muy recomendable tener las variables almacenadas en este formato por varios motivos:

- Es muy fácil extraer subgrupos (*subsets*) de las variables usando operadores lógicos.
- Muchas funciones estadísticas, como los estimadores económicos, admiten datos solo en este formato.
- Es posible extraer variables de la base de datos mediante la función `attach`.

Hay varias maneras de crear una base de datos. La primera es importar datos externos de un archivo de texto plano. La segunda es agrupando vectores existentes, como mostramos en el siguiente ejemplo —continuación del anterior—:

Incorporar comentarios mediante el símbolo #

Muy a menudo nos interesará introducir aclaraciones y comentarios en nuestras líneas de código para entenderlo mejor. Para que R no los interprete y nos dé un mensaje de error, los introduciremos después del símbolo #.

```
> g <- c('m','m','f','m','f','f') # Género
> v1 <- c(131,125,126,140,152,119) # Altura
> v2 <- c(48,53,45,40,49,50)      # Peso

> datos <- data.frame(genero=g,altura=v1,peso=v2)
> datos
  genero altura peso
1      m    131   48
2      m    125   53
3      f    126   45
4      m    140   40
5      f    152   49
6      f    119   50
```

Como vemos, en la primera columna se identifica cada elemento de la muestra, mientras que cada columna está encabezada por el nombre de la variable. Es importante destacar que, alternativamente, la función `data.frame` también admite una matriz.

Supongamos que queremos hacer un análisis solo de aquellos chicos (*m*) por debajo de cincuenta kilos. Para ello, es muy práctico crear otra base de datos que sea un subgrupo de la base de datos original. Llamaremos a este subgrupo `datos2`, y que como vemos solo incluye dos observaciones, las que satisfacen las condiciones establecidas:

La función `subset`

La función `subset` nos permite crear una nueva base de datos que contenga una selección condicional de observaciones de la base de datos original.

```
> datos2<-subset(datos,genero=="m" & peso<50)
> datos2
  sexo altura peso
1    m    131   48
4    m    140   40
```

Si pretendemos operar con las variables incluidas en una base de datos, deberemos referenciarlas con el símbolo `$`, ya que las variables no están en el espacio de trabajo individualmente, solo como elementos de una base de datos. Por ejemplo, si necesitamos la correlación entre la altura y el peso, deberemos introducir la siguiente instrucción:

```
> cor(datos$altura,datos$peso)
[1] -0.3202351
```

Esto puede llegar a ser algo engorroso. Por ello, si la intención es hacer operaciones con las variables de la base de datos, es recomendable «volcarlas» previamente en el espacio de trabajo mediante la función **attach**, con lo cual ya podremos hacer referencia a las variables individualmente:

La función **attach**

Esta función será aplicable cuando queramos realizar operaciones y cálculos individuales con las variables incluidas en una base de datos.

```
> attach(datos)
> cor(altura,peso)
[1] -0.3202351
```

2.8. Listas

Las listas (*lists*) son comparables a un cajón de sastre, ya que permiten almacenar objetos de clases diferentes en una misma estructura. Esta clase de objeto nos permitirá agrupar en una sola estructura elementos que son, a su vez, objetos de clases diferentes. Esto es muy útil cuando un mismo análisis tiene como resultado diferentes objetos dispares, por ejemplo vectores o matrices de diferente longitud o dimensión. Supongamos que deseamos almacenar en una estructura la matriz A , su descripción, su inversa y el determinante:

```
> descr <- c('Matriz A')
> A <- matrix(c(3,5,6,1),2,2)
> inv <- solve(A)
> deter <- det(A)
```

La forma de almacenarlo en una única lista es mediante la función `list`. Para acceder a los diferentes elementos de una lista mediante su nombre, usaremos el símbolo `$` de la siguiente manera:

Los símbolos `>` y `+` en la consola

Como hemos visto, en la consola hemos de introducir cada línea de código tras el símbolo `>`. Sin embargo, algunas instrucciones de código son tan largas que ocupan más de una línea, como en el caso de la lista `MatA`. En este caso, R nos mostrará en la siguiente línea el símbolo `+` para recordarnos que la instrucción de la línea previa está incompleta.

```
> MatA <- list(descripcion=descr,
+             matriz=A,
+             inversa=inv,
+             deter=deter)

> MatA$descripcion
[1] 'Matriz A'

> MatA$matriz
      [,1] [,2]
[1,]     3     6
[2,]     5     1

> MatA$inversa
      [,1] [,2]
[1,] -0.03703704 0.2222222
[2,] 0.18518519 -0.1111111

> MatA$deter
[1] -27
```

También se puede acceder a los elementos de una lista no por su nombre, sino ordinalmente. Para ello usaremos el sím-

bolo `[[ ]]`. Veamos una comparación de ambas maneras de acceder a elementos de la lista:

```
> MatA$matriz
      [,1] [,2]
[1,]     3     6
[2,]     5     1

> MatA[[4]]
[1] -27
```

Fijémonos en que, en este ejemplo, las instrucciones `MatA[[2]]` y `MatA$matriz` darían el mismo resultado.

2.9. Fórmulas

Uno de los análisis estadísticos más comunes es la regresión, que consiste en la especificación, la estimación y la posterior inferencia de un modelo, en el cual la variable dependiente (y) queda explicada por un conjunto de variables explicativas —o regresores—:

$$y = f(x_1, x_2, \dots, x_k)$$

Esta relación funcional suele tomar varias formas, la más común de las cuales es la *estructura aditiva*:

$$y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki} + u_i$$

En esta estructura, β_j , $j = 0, \dots, k$ son los coeficientes, x_j , $j = 1, \dots, k$ son las variables explicativas y u_i es el término de perturbación. Suponiendo que queremos estimar un modelo de regresión con las variables `y`, `x1`, `x2` y `x3`; mediante la función `lm` (*linear model*) se obtiene el ajuste por mínimos cuadrados ordinarios (MCO).

```
> modelo.1 <- lm(y ~ x1 + x2 + x3)
```

Esta instrucción almacena en el objeto `modelo.1` el resultado de la estimación. Supongamos que deseamos utilizar la especificación `y ~ x1 + x2 + x3` a lo largo de varias instrucciones. En este caso, sería más eficiente almacenar esta especificación en un objeto del tipo *formula*, como se muestra a continuación.

```
> ecuacion.1 <- formula(y ~ x1 + x2 + x3)
```

Una vez creado el objeto `ecuacion.1`, se puede introducir en cualquier modelo de regresión, como se muestra a continuación.

```
> modelo.1 <- lm(ecuacion.1)
```

En este sentido, en el tema V.2 se desarrollan ampliamente los modelos de regresión, así como la sintaxis de R que se utiliza al respecto.

2.10. Factores

Muy a menudo, nos encontraremos con bases de datos que contienen tanto variables cuantitativas como variables cualitativas. Estas últimas son variables que aportan información a cada observación y son consideradas como factores por R. Sin embargo, a veces también nos encontraremos con variables numéricas, pero que incorporan información cualitativa. Supongamos, en el siguiente ejemplo, las variables numéricas x e y .

```
> x <- c(3,2,4,5,6,4)
> y <- c(0,0,1,1,0,1)
> cbind(x,y)
      x y
[1,] 3 0
[2,] 2 0
[3,] 4 1
[4,] 5 1
[5,] 6 0
[6,] 4 1
```

La variable *x* hace referencia a un atributo, mientras que la variable *y* indica si la observación hace referencia a un hombre ($y = 0$) o a una mujer ($y = 1$). Para realizar cualquier análisis estadístico a partir de estas variables, es importante declarar que la variable *y* es un factor, para lo cual crearemos la variable *genero* mediante la función **factor**.

```
> genero <- factor(y, labels=c("Hombre",
+ "Mujer"))
```

La variable *genero* incorporará las observaciones "*Hombre*" para los valores $y = 0$ y "*Mujer*" para los valores $y = 1$ —fijémonos en el orden creciente de la asignación. Veamos como la variable *y* es una variable numérica, mientras que *genero* es un factor.

```
> class(y)
[1] "numeric"
> class(genero)
[1] "factor"
```

El siguiente paso puede ser la creación de una base de datos (*data frame*) incluyendo la variable *x* y el factor creado a partir de la variable *y*.

```
> datos.1 <- data.frame(x,genero)
> print(datos.1)
  x genero
1 3 Hombre
2 2 Hombre
3 4  Mujer
4 5  Mujer
5 6 Hombre
6 4  Mujer
```

Existen varias funciones asociadas a los factores. Dos de ellas son las funciones **stack** y **unstack**. Esta última se usa para transformar categorías de un factor en nuevas variables. En nuestro ejemplo, crearemos la base de datos **datos.2**, la cual repartirá los valores de *x* entre las nuevas variables *Hombre* y *Mujer*.

```
> datos.2 <- unstack(datos.1)
> print(datos.2)
  Hombre Mujer
1      3     4
2      2     5
3      6     4
```

Este paso se puede deshacer mediante la función **stack**, la cual invierte el proceso apilando diversas variables en una sola y creando una variable asociada del tipo *factor*, donde cada categoría es el nombre original de la variable. Veamos cómo se aplica en nuestro ejemplo, creando la base de datos **datos.3**.

```
> datos.3 <- stack(datos.2)
> print(datos.3)
  values      ind
1      3 Hombre
2      2 Hombre
3      6 Hombre
```

4	4	Mujer
5	5	Mujer
6	4	Mujer

3. Importación y exportación de datos

Como veremos a lo largo de este manual, un procedimiento fundamental a la hora de efectuar un análisis con datos es, en primer lugar, importar datos a partir de diversos formatos y, al final, exportar los resultados en un formato determinado.

Antes de importar los datos, es un paso imprescindible definir el directorio de trabajo (*working directory*). Esto se hace mediante la función `setwd`. Todos los datos que se importen han de estar en ese directorio y todos los datos exportados se almacenarán igualmente en ese directorio. Veamos un ejemplo de esta función.

```
> setwd("C:/directorio")
```

Para importar datos, la principal función es `read.table`, la cual importa datos a partir de un archivo original y los convierte en una base de datos —esto es, un objeto del tipo `data.frame`. Esta función tiene varias opciones de lectura, las cuales tendremos que especificar para asegurarnos de que los datos se importan correctamente. Veamos un ejemplo sencillo del uso de esta función.

```
> base.datos <- read.table("Datos.txt",  
+   header = TRUE, sep = ",", dec = ".",  
+   row.names = , col.names)
```

En este ejemplo, hemos importado datos (por columnas) del archivo *"Datos.txt"*, depositando las variables incluidas en el objeto `base.datos`. Además, `header = TRUE` indica que la primera fila corresponde a los nombres de las variables, `sep = " "` indica que la separación entre valores en una fila es el espacio en blanco, `dec = "."` indica el carácter usado para marcar los decimales, y las opciones `row.names` y `col.names` se usan para indicar los nombres de las filas y las columnas, respectivamente.

Están disponibles, por otro lado, las funciones `read.csv`, `read.csv2`, `read.delim` y `read.delim2`. Estas son idénticas a la función `read.table`, siendo la única diferencia los valores por defecto. Así, `read.csv` y `read.csv2` se usan cuando los valores están separados por comas (*comma separated values*), y `read.delim` y `read.delim2` son indicadas si los valores están separados mediante tabulación. Una amplia explicación de todas estas opciones se puede obtener acudiendo a la ayuda de esta función (`help(read.table)`).

Además, también existen funciones específicas para importar datos de otros formatos —hojas de cálculo, bases de datos y formatos de otros programas estadísticos. Para una exhaustiva explicación de todas las opciones de importación, se recomienda la lectura del documento de ayuda *R Data Import/Export*, disponible en la web del CRAN —The Comprehensive R Archive Network, cuya dirección de internet es <http://cran.r-project.org>—, en la sección «Documentation/Manuals».

De manera análoga, para exportar datos existe la función `write.table`, la cual crea un archivo en el directorio de trabajo (*working directory*). Un ejemplo similar al caso anterior se muestra a continuación. Siendo `x` el resultado que queremos exportar —puede ser un vector, una matriz o una base de datos—, introduciremos la siguiente instrucción.

```
> write.table(x, file = "resultado.txt",  
+   sep = " ", na = "NA", dec = ".",  
+   row.names = TRUE, col.names = TRUE)
```

Así, se creará un documento de texto con el nombre "*resultado.txt*", donde los valores por fila están separados por espacios en blanco (`sep = "` "), los valores ausentes (*missing values*) se indicarán con el nombre NA (*not assigned*, en inglés), los decimales se indicarán con un punto (`dec = "."`) y se incluirán los nombres de las filas y columnas —`row.names = TRUE` y `col.names = TRUE`, respectivamente.

Si deseamos exportar los datos a un archivo con formato *xls*, esto es, a una hoja de cálculo, la opción recomendable es exportar a un archivo separado por comas (*comma separated values*, *csv*), el cual puede ser leído por programas como Microsoft Excel y OpenOffice. Para ello, usaremos la función `write.csv`, como muestra el siguiente ejemplo.

```
> write.csv(x, file = "resultado.csv")
```

Las opciones de la función `write.table` son igualmente aplicables a `write.csv`.

4. Extensiones de R

Uno de los principales problemas a los que se ha enfrentado R desde su nacimiento es la importante curva de aprendizaje que el usuario ha de afrontar para poder usarlo. De hecho, esto es un problema a la hora de implementar este programa en los primeros cursos de estadística de grado, ya que a la dificultad de aprender el contenido de la asignatura se suma la complicación del lenguaje de R. Por suerte, este programa

es tan flexible y maleable que se han creado varias aplicaciones basadas en R, pero que son más fáciles de utilizar. En esta sección presentamos tres de estas extensiones. La primera, R-Commander, es la única que utilizaremos a lo largo de todo el manual. RStudio está indicada para usuarios avanzados y Deducer es una aplicación basada en Java —y todavía en desarrollo—, similar a R-Commander, pero que potencia el apartado gráfico.

4.1. R-Commander

Una de las aplicaciones con más éxito ha sido R-Commander, que consiste en una interfaz gráfica de usuario —GUI, en sus siglas en inglés. Su creador, el profesor John Fox, lo desarrolló con el objetivo de crear un programa con *menús desplegables* y que fuera fácil de usar para estudiantes de estadística. Se puede considerar a R-Commander como una alternativa viable a los paquetes estadísticos comerciales (como Minitab o SPSS).

Otro aspecto destacable de R-Commander es que resulta muy útil para aquellos estudiantes que aprenden a utilizar R. Esto es así porque, al efectuar el análisis mediante el menú desplegable, el código en R subyacente también aparece en pantalla junto al resultado, de manera que, al final, el estudiante acabará introduciendo el código directamente.

Para instalar R-Commander hemos de tener instalada una versión de R. En general, como para todos los paquetes, hay dos opciones de instalación:

- 1) Ir al menú desplegable de R, seleccionar *Paquetes e Instalar paquetes*. Una vez allí, seleccionamos un servidor —por proximidad podríamos elegir Spain (Madrid), France (Toulouse) o Portugal— y pulsamos *Ok*. A continuación, nos aparecerá una lista de todas las librerías (paque-

tes o *packages*, en inglés) que podemos instalar y seleccionamos el paquete *Rcmdr*.

- 2) Como haríamos con cualquier otro paquete que deseáramos instalar, podríamos ir directamente a la consola de R e introducir `install.packages("Rcmdr")`.

Una vez acabada la descarga, la instalación se completa introduciendo en la consola la instrucción `library(Rcmdr)`. Entonces aparecerá una ventana de aviso indicándonos que hay que instalar algunos paquetes adicionales. Pulsamos en *Aceptar* y esperamos a que se termine el proceso. Si todo ha ido bien, al acabar nos aparecerá en pantalla la ventana de R-Commander, lista para poder usarla. En general, siempre que queramos usar R-Commander deberemos introducir la instrucción `library(Rcmdr)` en la consola.

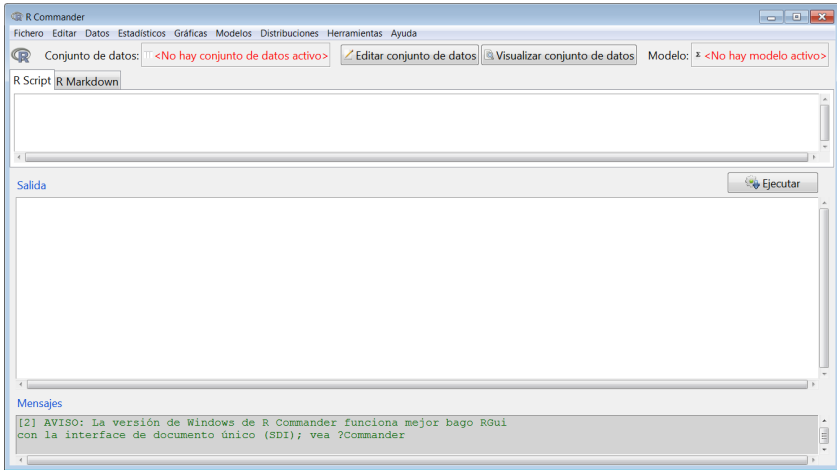
Ejecutar el programa más de una vez

Si cerramos el programa R-Commander, volveremos a la consola de R. Si entonces deseamos volver a cargar R-Commander sin tener que cerrar la sesión de R y abrirla de nuevo, deberemos introducir la instrucción `Commander()` en la consola de R.

La ventana principal de R-Commander se muestra en la figura I.3.

A lo largo de los capítulos de este libro iremos analizando cada uno de los elementos de este programa. Sin embargo, en esta sección veremos una visión general de todos los componentes:

- 1) *Barra de menús*. Esta barra se encuentra en la línea superior e incorpora las principales funcionalidades del programa. A continuación se describen sus elementos:

Figura I.3: La interfaz gráfica R-Commander

- a) *Fichero*. Instrucciones para crear, cargar y guardar documentos con código (*scripts*), resultados y espacios de trabajo (*workspaces*), así como para salir del programa.
- b) *Editar*. Opciones de cortar, copiar y pegar, y, en general, para editar el contenido de las ventanas de instrucciones y de resultados.
- c) *Datos*. Submenús que contienen elementos de menú para importar, leer y manipular datos y variables.
- d) *Estadísticos*. Submenús que contienen elementos de menú para una variedad de análisis estadísticos básicos.
- e) *Gráficas*. Elementos de menú para crear gráficos estadísticos simples.
- f) *Modelos*. Elementos para obtener todo tipo de información sobre la estimación de modelos estadísticos: resúmenes nu-

méricos, intervalos de confianza y contrastes de hipótesis, diagnósticos, análisis de residuos y gráficos derivados de un modelo estadístico.

- g) *Distribuciones*. Contiene las principales distribuciones de probabilidad discretas y continuas, incluyendo probabilidades, cuantiles, gráficos y generación de datos aleatorios.
- h) *Herramientas*. Permite cargar paquetes adicionales que no vienen en la distribución básica de R-Commander. Se pueden cargar a modo de extensiones si se necesita realizar un análisis específico. Además, en esta parte del menú también están las opciones de visualización —tamaño de letra, etc.
- i) *Ayuda*. Permite acceder a varios documentos de ayuda de R-Commander muy completos, de lectura recomendada a los no iniciados en el programa.

Un elemento fundamental que se debe tener en cuenta es que R-Commander solo incluye una pequeñísima parte de las potencialidades de R. Es decir, R-Commander ha sido diseñado para satisfacer las necesidades de estudiantes de cursos de estadística introductorios e intermedios. Aquellos usuarios que necesiten realizar análisis estadísticos y cuantitativos más avanzados necesitarán más funciones de las que se incluyen en la barra de menús.

2) *Barra de herramientas*. Esta barra incluye dos elementos básicos: el conjunto de datos y el modelo.

- a) *Conjunto de datos*. El conjunto incluye una serie de datos, distribuidos en variables por columnas. Hay dos botones que permiten *editar* y *visualizar* el conjunto de datos. Un

aspecto que se debe tener en cuenta es que se puede trabajar con varios conjuntos de datos simultáneamente, pero solo puede haber un conjunto de datos activo en cada momento. En el espacio de la figura I.3 donde se lee «No hay conjunto de datos activo», aparecerá el menú desplegable con los diferentes conjuntos de datos existentes, de entre los que elegiremos, en cada momento, el conjunto de datos activo que deseemos analizar.

- b) *Modelo*. El funcionamiento de este espacio es análogo al anterior. Esto es, podemos estimar diferentes modelos en cada sesión de trabajo, pero solo puede haber un modelo activo en cada momento, ya que la funcionalidad del menú desplegable se aplica siempre sobre el modelo activo.

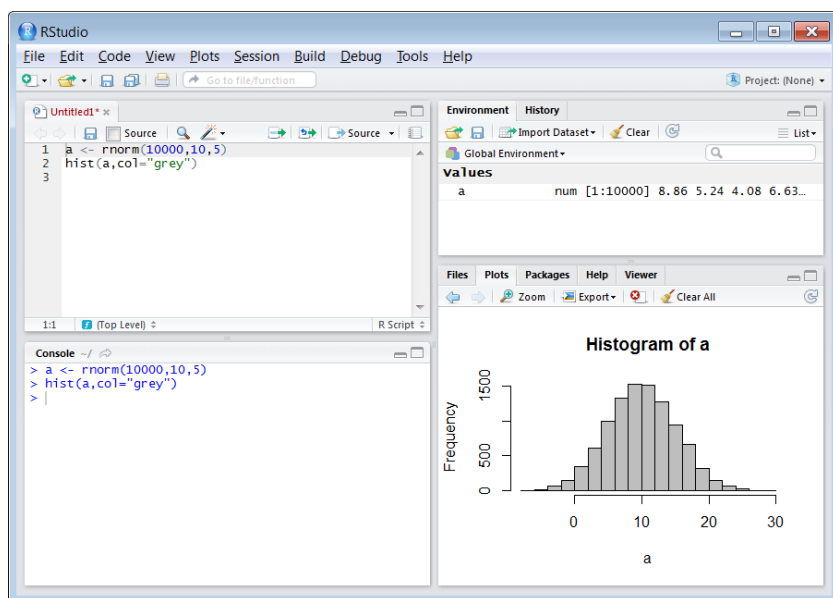
3) *Ventanas de instrucciones, resultados y mensajes*. La ventana de instrucciones tiene como objetivo desplegar las instrucciones en código de R. Es importante explicar su funcionamiento: cuando realizamos un análisis mediante las opciones disponibles en el menú desplegable, el código correspondiente aparecerá igualmente en la ventana de instrucciones y el resultado aparecerá en la ventana de resultados. Ahora bien, también es posible introducir las instrucciones «directamente» en la ventana de instrucciones sin necesidad de acudir al menú desplegable, seleccionar las líneas introducidas y pulsar en el botón *Ejecutar* situado a la derecha, con lo que el resultado aparecerá igualmente en la ventana de resultados. Esto lo haremos si conocemos el código y preferimos escribirlo en lugar de acudir al menú desplegable, pero sobre todo si queremos implementar un cálculo o técnica que no está disponible en el menú desplegable.

Por último, veremos una ventana inferior de mensajes, en la que aparecerán básicamente mensajes de error que nos informarán de qué hemos hecho de manera incorrecta.

4.2. RStudio

Para los usuarios de R avanzados, la extensión RStudio es una potente herramienta consistente en un entorno de desarrollo integrado (IDE), en el que se visualizan el documento con el código, la ventana de resultados, una lista con los objetos creados y un espacio para los gráficos. En este sentido, es similar al entorno de trabajo que ofrece el programa de análisis matemático Matlab. La figura I.4 ofrece una captura de pantalla de este programa.

Figura I.4: La aplicación RStudio



Como vemos, además de la barra del menú, esta aplicación divide la pantalla en cuatro cuadrantes. La ventana superior izquierda sirve para escribir el código, mientras que la ventana inferior izquierda es la consola donde se ejecuta el código

y se muestra el resultado. A la derecha, en la parte superior se muestra una lista con los objetos creados en la sesión —vectores, matrices, bases de datos, etc.— y sus características, mientras que la parte inferior sirve para mostrar gráficos y páginas de ayuda, entre otros. Un punto fuerte de RStudio es la exportación de gráficos, ya que se pueden almacenar en diversos formatos y tamaños.

Bibliografía

R es un entorno vasto y amplio, a la vez que cambiante, ya que permanentes actualizaciones lo mantienen al día y aumentan la colección de librerías. Así pues, es imposible cubrir todos sus aspectos en un manual introductorio como este. Por eso mismo, recomendamos al lector que desee zambullirse más en este entorno de programación que amplíe sus conocimientos con otros manuales y obras. En la página del CRAN hay disponibles muchos manuales introductorios en varios idiomas. En castellano, destacan el documento *R para principiantes*, de Emmanuel Paradis, y *Métodos Estadísticos con R y R Commander*, de A. J. S. Castillo. Además, A. F. Zuur y otros ofrecen una guía para principiantes denominada *A Beginner's Guide to R*, donde se cubren aspectos básicos de manejo de variables y bases de datos, herramientas gráficas y errores más comunes en R. De M. J. Crawley son los manuales *Statistics. An introduction using R* (2005) y el completo y exhaustivo *The R Book* (2012). También están recomendadas para no iniciados las obras *Using R for introductory statistics*, de J. Verzani, y *Introductory statistics with R*, de P. Dalgaard. Por último, es muy recomendable *Data manipulation with R*, de P. Spector, el cual es un breve y conciso manual sobre cómo manipular datos y variables con R.

Capítulo II

Álgebra lineal y cálculo

1. Motivación

El presente capítulo tiene como objetivo profundizar en aspectos matemáticos. De hecho, se podría decir que su objetivo es doble. Por una parte, cubre distintas partes del álgebra lineal necesarias para el estudio de la estadística, como son las operaciones matriciales y la descomposición de matrices. Por otra parte, además, pretende cubrir otros aspectos que van más allá de la estadística descriptiva, como pueden ser el cálculo o los números complejos. Por eso mismo, aunque R sea un programa inicialmente concebido para el estudio de la estadística, también puede ser usado por estudiantes de asignaturas de física, ingeniería y matemáticas en general, en las que se tratan profundamente elementos de álgebra lineal avanzada y análisis numérico. Así pues, se puede pensar que R es una alternativa viable a programas como MATLAB u Octave, ampliamente utilizados en diversas ramas de la ingeniería.

Tradicionalmente, el estudio de cuestiones de cálculo se ha realizado a nivel computacional con los llamados *sistemas de álgebra computacional* —CAS es el acrónimo en inglés—, cuyos principales programas son Mathematica, Maxima y SAGE. Estos programas son muy potentes y llevan a cabo operaciones de diferenciación, integración y resolución de expresiones algebraicas de una forma simbólica, es decir, introduciendo una función como $y = x^n$, el programa nos devuelve la expresión simbólica de la derivada, esto es, $y' = nx^{n-1}$. A diferencia

de estos programas, el cálculo simbólico no se puede considerar una ventaja de R, aunque cada vez más están apareciendo extensiones de R que permiten realizar cálculos de esta manera. Un ejemplo de estas extensiones de R que permiten hacer cálculo simbólico son *rSymPy*, *Ryacas* y *mosaic*. Sin embargo, el punto fuerte de R es el análisis numérico, que se centra en el estudio de los algoritmos que utilizan una *aproximación numérica* —en contraposición a la manipulación simbólica— para los problemas de análisis matemático.

2. Álgebra vectorial y matricial

2.1. Operaciones algebraicas

Retomando lo visto en el primer capítulo, R permite almacenar datos numéricos en vectores y matrices, lo cual hace posible efectuar operaciones algebraicas con vectores, matrices y escalares. Veamos las posibles operaciones que R permite efectuar.

2.1.1. Suma y resta de escalares, vectores y matrices

R permite realizar operaciones vectorizadas, lo cual hace posible sumar o restar un escalar y un vector. En este sentido, se suma o resta el valor del escalar a todos los elementos del vector, como muestra el siguiente ejemplo:

$$c \pm \mathbf{v} = c \pm (v_1, \dots, v_n) = (c \pm v_1, \dots, c \pm v_n)$$

```
> v <- 1:4  
> 5+v  
[1] 6 7 8 9
```

Eso sí, para sumar o restar vectores, estos han de tener la misma longitud. Veamos el siguiente ejemplo:

Secuencias numéricas

Fijaos en que el vector **a** incluye los valores (1, 2, 3, 4, 5), mientras que el vector **b** los incluye de forma inversa, esto es, (5, 4, 3, 2, 1).

```
> a <- 1:5
> b <- 5:1
> a+b
[1] 6 6 6 6 6
> a-b
[1] -4 -2 0 2 4
```

Análogamente al caso de los vectores, se puede efectuar la suma y resta de escalares y matrices:

$$c \pm A = c \pm \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} c \pm a_{11} & \dots & c \pm a_{1n} \\ \vdots & \ddots & \vdots \\ c \pm a_{n1} & \dots & c \pm a_{nn} \end{pmatrix}$$

Consejo para programar ordenadamente

Es recomendable, en un análisis en el que incluyamos vectores y matrices, nombrar los vectores con minúsculas y las matrices con mayúsculas, de manera que se eviten confusiones.

```
> A <- matrix(1:4,2,2)
> 3 + A
      [,1] [,2]
[1,]    4    6
[2,]    5    7
```

La suma y resta de matrices se realiza elemento por elemento, de manera que estas han de tener la misma dimensión; sino, obtendremos un mensaje de error.

```
> A <- matrix(1:4,2,2)
> B <- matrix(4,2,2)
> A + B
      [,1] [,2]
[1,]    5    7
[2,]    6    8
```

Para sumar y restar vectores y matrices no es necesario que estos concuerden en longitud y dimensión. Esta propiedad en R se denomina *reciclaje*, esto es, los valores del vector se suman a la matriz hasta completarla y en el caso de que el producto de filas y columnas sea superior a la longitud del vector, los valores de este se repetirán hasta completar la matriz. Veamos un ejemplo:

Primero columnas, luego filas

Recordad que, a la hora de hacer una operación entre un vector y una matriz, R empieza la operación por la primera columna, siempre de arriba abajo y de izquierda a derecha.

```
> (v <- 2*1:4)
[1] 2 4 6 8
> (A <- matrix(0,2,4))
      [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0

> v + A
      [,1] [,2] [,3] [,4]
[1,]    2    6    2    6
[2,]    4    8    4    8
```

En el caso de que la longitud de la matriz —filas por columnas— no sea múltiplo de la longitud del vector, la operación se realizará igualmente, pero obtendremos un mensaje de error para recordarnos esta circunstancia.

```
> (v <- 2*1:4)
[1] 2 4 6 8
> (A <- matrix(0,2,3))
      [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
> v + A
      [,1] [,2] [,3]
[1,]    2    6    2
[2,]    4    8    4
Mensajes de aviso perdidos
In v + A :
  longitud de objeto mayor no es múltiplo de la
  longitud de uno menor
```

2.1.2. Producto y división elemento por elemento

Análogamente al caso anterior, R también permite multiplicar o dividir un escalar y un vector. Para ello, utilizaremos los símbolos " $*$ " para la multiplicación y " $/$ " para la división.

$$c \cdot \mathbf{v} = c \cdot (v_1, \dots, v_n) = (c \cdot v_1, \dots, c \cdot v_n)$$

$$c/\mathbf{v} = c/(v_1, \dots, v_n) = (c/v_1, \dots, c/v_n)$$

```
> 2 * 1:4
[1] 2 4 6 8
> 2 / 1:4
[1] 2.0000000 1.0000000 0.6666667 0.5000000
```

La multiplicación elemento por elemento entre vectores requiere que estos tengan la misma longitud.

```
> 1:3 * 1:3  
[1] 1 4 9
```

De igual manera, la multiplicación elemento por elemento entre matrices solo es posible si estas tienen la misma dimensión.

¡Cuidado con las multiplicaciones de matrices!

Como veremos más adelante, hay más de un operador multiplicación definido para matrices, con lo cual hay que tener muy claro qué tipo de multiplicación queremos aplicar. Este ejemplo muestra una multiplicación elemento por elemento.

```
> (A <- matrix(1:4,2,2))  
      [,1] [,2]  
[1,]    1    3  
[2,]    2    4  
  
> A * t(A)  
      [,1] [,2]  
[1,]    1    6  
[2,]    6   16
```

Un caso especial es el de la multiplicación elemento por elemento de un vector y una matriz. Esta operación solo se podrá hacer si la longitud del objeto menor es múltiple de la del mayor, entendiendo la longitud de la matriz como el producto del número de filas y columnas. Hay que ir con cuidado al efectuar este tipo de operaciones, ya que es fácil equivocarnos y que el resultado no sea lo que deseamos obtener.

```
> 1:3 * matrix(1,2,3)
      [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3
```

Por último, es interesante destacar que otros operadores algebraicos también se pueden aplicar elemento a elemento. Veamos el caso del operador potencia:

```
> (A <- matrix(1:4,2,2))
      [,1] [,2]
[1,]    1    3
[2,]    2    4

> A^2
      [,1] [,2]
[1,]    1    9
[2,]    4   16

> A^A
      [,1] [,2]
[1,]    1   27
[2,]    4  256
```

2.1.3. Producto escalar entre dos vectores

Producto escalar

También conocido como producto interno, producto interior o producto punto (en inglés, *dot product*), se trata de una operación binaria definida sobre dos vectores de un espacio euclídeo cuyo resultado es un escalar.

Si tenemos dos vectores $\mathbf{a}$ y $\mathbf{b}$ de igual longitud (n elementos), el producto escalar entre ambos se define de la siguiente manera:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

En R, esta operación se realiza de la siguiente manera:

```
> a <- 1:3  
> b <- c(2,5,8)  
> sum(a*b)  
[1] 36
```

Ortogonalidad

Geométricamente, dos vectores ortogonales forman un ángulo recto entre ambos. La palabra proviene del griego *orthos* ('recto') y *gonía* ('ángulo').

Siempre que dos vectores sean ortogonales ($\mathbf{a} \perp \mathbf{b}$), su producto escalar será igual a cero.

```
> a <- c(1,1)  
> b <- c(-1,1)  
> sum(a*b)  
[1] 0
```

2.1.4. Norma de un vector

Espacio euclidiano

En geometría, el concepto de espacio euclidiano abarca el plano euclidiano —para un espacio de dos dimensiones— y el espacio euclidiano —para un espacio de tres o más dimensiones. Se denomina así en honor al matemático griego antiguo Euclides de Alejandría.

También denominada la *magnitud* de un vector, se trata de la distancia entre el origen y el vector. Esto es, en un espacio euclidiano $\mathbb{R}^n$, la norma del vector $\mathbf{v} = (v_1, \dots, v_n)$ viene definida por la siguiente fórmula:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + \dots + v_n^2}$$

No existe en la distribución básica de R ninguna función predefinida que haga esta operación con vectores. Sin embargo, es muy fácil crear una función que la realice. Esta es una de las principales ventajas de R, que podemos crear nuestras propias funciones personalizadas y usarlas cuando las necesitamos.

Pitágoras tenía razón

Geométricamente, en el plano euclidiano la norma se puede interpretar como la longitud de la hipotenusa, en este caso $\sqrt{1^2 + 1^2} = \sqrt{2} = 1.414214$.

```
> norm.vec <- function(x) sqrt(sum(x*x))
> a <- c(1,1)
> norm.vec(a)
[1] 1.414214
```

2.1.5. Producto matricial

¡No confundir los tipos de multiplicación!

En R, el símbolo `*` se emplea para la multiplicación elemento por elemento, mientras que el símbolo `%*%` se usa para el producto matricial.

El producto matricial entre $A_{n \times m}$ y $B_{m \times p}$ se obtiene aplicando la siguiente fórmula:

$$AB = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix} =$$

$$\begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Para poder aplicarlo, el número de columnas de A debe ser igual al número de filas de B , y la matriz resultante tendrá tantas filas como A y tantas columnas como B . En R, esta operación se realiza mediante el operador `%*%`:

```
> (A <- matrix(1:4,2,2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> (B <- matrix(2,2,1))
     [,1]
[1,]    2
[2,]    2

> A %*% B
     [,1]
[1,]    8
[2,]   12
```

También es posible utilizar el producto matricial con vectores. Esto es, reinterpretando los vectores $\mathbf{a}$ y $\mathbf{b}$ de longitud n como vectores columna $\mathbf{a}_{1 \times n}$ y $\mathbf{b}_{1 \times n}$. Los productos euclidiano interior y exterior son los casos más sencillos especiales del

producto de la matriz, mediante la transposición de los vectores columna en vectores fila.

El *producto interior* es el resultado de multiplicar un vector fila por un vector columna, lo cual equivale al producto escalar:

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = a_1 b_1 + \cdots + a_n b_n$$

$$= \sum_{i=1}^n a_i b_i$$

En R podemos aplicar esta operación de dos maneras: con el operador de producto matricial `%*%` y con la función del producto cruz `crossprod`, la cual efectúa la operación $\mathbf{A}^T \mathbf{B}$ tanto para vectores como matrices:

```
> a <- 1:3
> a %*% a
      [,1]
[1,]    14
> crossprod(a,a)
      [,1]
[1,]    14
```

El *producto exterior* es el resultado de multiplicar un vector columna por un vector fila:

$$\mathbf{a} \otimes \mathbf{b} = \mathbf{a} \mathbf{b}^T = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \begin{pmatrix} b_1 & \cdots & b_n \end{pmatrix} = \begin{pmatrix} a_1 b_1 & \cdots & a_1 b_n \\ \vdots & \ddots & \vdots \\ a_n b_1 & \cdots & a_n b_n \end{pmatrix}.$$

Aplicamos esta operación en R con el operador `%o%`, que calcula $\mathbf{A} \mathbf{B}^T$ tanto para vectores como matrices:

```
> a <- 1:3
> a %o% a
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
```

2.1.6. Producto de Kronecker

Leopold Kronecker

Leopold Kronecker (1823-1891) fue un matemático alemán que trabajó en la teoría de números y el álgebra.

El producto de Kronecker, denotado por el símbolo $\otimes$, es una operación entre dos matrices que resulta en una matriz en bloques. Es una generalización de vectores a matrices del producto exterior —denotado por el mismo símbolo. Esta operación consiste en que cada elemento de la primera matriz multiplica a todos los elementos de la segunda matriz. Partiendo de las matrices $A_{m \times n}$ y $B_{p \times q}$, el producto de Kronecker $A \otimes B$ resultará en una matriz en bloques de dimensión $mp \times nq$:

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

R dispone de la función `kronecker` para realizar esta operación.

```
> (A <- matrix(1,1,3))
      [,1] [,2] [,3]
[1,]    1    1    1
```

```

> (B <- matrix(1:4,2,2))
      [,1] [,2]
[1,]    1    3
[2,]    2    4

> kronecker(A,B)
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    1    3    1    3
[2,]    2    4    2    4    2    4

```

2.2. Ecuaciones algebraicas

Este tipo de ecuaciones involucra a polinomios. Recordemos que un polinomio de grado n tiene la siguiente forma:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

Las ecuaciones algebraicas, también denominadas *polinómicas*, son ecuaciones donde se establece la igualdad de dos polinomios $P = Q$, siendo el objetivo encontrar la solución, es decir, el valor de x que satisface la igualdad $P = Q$ o, lo que es lo mismo, $P - Q = 0$. Como veremos más adelante, R dispone de varias alternativas para solucionar ecuaciones. En el caso de polinomios, una manera sencilla consiste en crear vectores con los coeficientes del polinomio de forma creciente $(a_0, a_1, \dots, a_n)$, luego restarlos y buscar la raíz mediante la función `polyroot`, que es la solución. Veamos un sencillo ejemplo:

$$\begin{aligned}
 2x^2 + 5x + 5 &= x^2 + 3x + 4 \\
 x^2 + 2x + 1 &= 0
 \end{aligned}$$

Notación compleja

R devuelve las raíces de un polinomio en forma binomial $z = a + bi$, siendo a la parte real y b la parte imaginaria. En este caso, al ser $b = 0$, las soluciones son $(-1, -1)$.

Un número complejo se representa en forma binomial como:

```
> p1 <- c(5,5,2)
> p2 <- c(4,3,1)

> polyroot(p1-p2)
[1] -1-0i -1+0i
```

Es importante tener en cuenta que el número de raíces de un polinomio es igual a su grado, en este caso $n = 2$. Además, las raíces se pueden expresar en números complejos,¹ con lo cual en este caso la solución se interpreta como $x = -1 \pm 0 \cdot i = -1$.

2.3. Sistemas de ecuaciones lineales

En matemáticas, la teoría de sistemas lineales es una parte fundamental del álgebra lineal. La fórmula general de un sistema de m ecuaciones lineales con n incógnitas toma la siguiente forma:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \dots & & \dots & & \dots & & \dots & & \dots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m \end{array}$$

¹Recordemos que $i = \sqrt{-1}$. En la parte final de este capítulo veremos en detalle cómo R trabaja con números complejos.

En este sistema, x_j son las incógnitas, a_{ij} los coeficientes y b_i los términos independientes. Este tipo de sistemas puede ser:

- *Incompatible*: si no tiene solución.
- *Compatible*: si tiene solución. Será *determinado* si tiene una única solución e *indeterminado* si admite un conjunto infinito de soluciones.

Usando las propiedades del producto de matrices, es posible expresar el sistema de ecuaciones lineales anterior en forma matricial: $A \cdot x = b$.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Existen muchos métodos de resolución de este tipo de sistemas, que dependen de sus características. En R, la función básica que resuelve sistemas de ecuaciones es `solve`. Veamos su aplicación resolviendo el siguiente sistema:

$$\begin{aligned} 3x + y + 2z &= 10 \\ 4x + 3y + 4z &= 21 \\ 2x + y + 2z &= 9 \end{aligned}$$

Primero expresamos el sistema mediante matrices:

$$A = \begin{pmatrix} 3 & 1 & 2 \\ 4 & 3 & 4 \\ 2 & 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 10 \\ 21 \\ 9 \end{pmatrix}$$

```
> e1 <- c(3,1,2)
> e2 <- c(4,3,4)
> e3 <- c(2,1,2)

> A <- rbind(e1,e2,e3)

> b <- c(10,21,9)
```

Seguidamente, usamos la función `solve` para efectuar el cálculo $x = A^{-1}b$ y obtenemos la solución $x = 1$, $y = 3$ y $z = 2$.

```
> solve(A,b)
[1] 1 3 2
```

2.4. Descomposición matricial

2.4.1. Vectores y valores propios

En álgebra lineal, la descomposición espectral —también llamada eigendescomposición o descomposición en vectores y valores propios— es la factorización de una matriz en una forma canónica, en la que la matriz se representa en términos de sus valores y vectores propios. Solo las matrices diagonalizables se pueden factorizar de esta manera.

Analíticamente, un vector no nulo $\mathbf{v}$ de longitud N es un *vector propio* de una matriz cuadrada A de dimensión $N \times N$ si y solo si se satisface la siguiente ecuación lineal:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Siendo λ un escalar denominado el *valor propio* asociado a $\mathbf{v}$. La interpretación es la siguiente: la matriz A se interpreta como una transformación lineal aplicada al vector $\mathbf{v}$.

Entonces, este vector $\mathbf{v}$ será aquel al cual la transformación A solo alarga o encoge —es decir, cambia su longitud—, pero no cambia su dirección. Entonces, el valor propio λ será la magnitud en la que el vector propio se alarga o encoge.

Para cada valor propio λ_i disponemos de la siguiente ecuación:

$$(A - \lambda_i \mathbf{I}) \mathbf{v} = 0.$$

Siendo I la matriz identidad $N \times N$.

En R, los valores y vectores propios se calculan mediante la función `eigen`. Por ejemplo, supongamos que deseamos calcular los valores y vectores propios de la siguiente matriz:

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

La función `eigen`

Esta función devuelve una lista con dos elementos: los valores (*values*) y los vectores (*vectors*), a los cuales se accede mediante el operador `$`.

```
> A <- matrix(c(2,0,0,2),2,2)

> eigen(A)
$values
[1] 2 2

$vectors
      [,1] [,2]
[1,]    0  -1
[2,]    1    0
```

En este caso, hemos obtenido dos valores propios ($\lambda_1 = \lambda_2 = 2$) y dos vectores propios ($\mathbf{v}_1 = (0, 1)$ y $\mathbf{v}_2 = (-1, 0)$).

Es interesante apuntar que siempre obtendremos tantos valores y vectores propios como la dimensión de la matriz de transformación A .

2.4.2. Descomposición en valores singulares

Esta descomposición, abreviada como DVS —en inglés el acrónimo es SVD, *single value decomposition*)—, está relacionada con la descomposición espectral vista anteriormente. Esta descomposición tiene muchas aplicaciones en estadística y otras disciplinas, como el procesamiento de señales en ingeniería. Partiendo de una matriz A , igual que en el caso anterior, la DSV efectúa la siguiente operación:

$$A = UDV^T$$

Donde U y V son ortogonales, V^T es la matriz transpuesta de V y D es una matriz diagonal con los valores singulares D_{ii} . Equivalentemente, también se puede expresar como $D = U^T AV$. Veamos un ejemplo con la matriz anterior A . Como vemos, D contiene los valores singulares de A .

```
> A <- matrix(c(2,0,0,2),2,2)
> svd(A)
$d
[1] 2 2

$u
      [,1] [,2]
[1,]     1     0
[2,]     0     1

$v
      [,1] [,2]
[1,]     1     0
[2,]     0     1
```

Además, en R existen otras descomposiciones matriciales disponibles, cuyo estudio sobrepasa los objetivos de este libro. Las principales son la factorización QR (función `qr`) y la factorización de Cholesky (función `chol`).

3. Funciones

En este capítulo veremos las múltiples posibilidades que nos ofrece R para trabajar con funciones matemáticas del tipo $y = f(x)$. La función básica, explicada en el capítulo introductorio, es `function`. Sirva como ejemplo la función de una variable $y = x^2 - 3$:

```
> y <- function(x) x^2 - 3
```

Con la función y introducida, es inmediato evaluar la función para valores específicos de x , tanto escalares como vectores. En el siguiente ejemplo creamos un vector de valores iniciales x_0 con los valores $(-3, -2, \dots, 2, 3)$ y los hemos evaluado con la función creada $y(x)$:

```
> y(1)
[1] -2

> (x0 <- -3:3)
[1] -3 -2 -1  0  1  2  3

> y(x0)
[1]  6  1 -2 -3 -2  1  6
```

La función `uniroot` se emplea para buscar las raíces de una función, es decir, los valores de x para los que se cumple $f(x) = 0$. Para ello, hay que introducir un vector con la parte

del dominio en la que queremos buscar raíces. En este caso, buscaremos en el intervalo $x \in [-2, 0]$:

```
> uniroot(y,c(-2,0))
$root
[1] -1.73205

$f.root
[1] -3.278749e-06

$iter
[1] 5

$estim.prec
[1] 6.103516e-05
```

Como vemos, esta función nos ofrece mucha información. La parte relevante es **\$root**, que nos indica que $f(x_0) = 0$ si $x_0 = -1,73205$. El resto de la información hace referencia al número de iteraciones y a la precisión de la estimación. En este punto es importante destacar que el intervalo introducido ha de contener una y solo una raíz, es decir, la función $f(x)$ ha de cortar la recta horizontal una sola vez. Si no es así, obtendremos un mensaje de error:

```
> uniroot(y,c(-2,2))
Error in uniroot(y, c(-2, 2)) :
  f() values at end points not of opposite sign
```

3.1. Clases de funciones

Existen diferentes clasificaciones de funciones matemáticas. Una clasificación fundamental es según el número de argumentos o incógnitas que tiene:

- 1) Funciones de una variable: toman la forma $y = f(x)$, como la función antes descrita.
- 2) Funciones de varias variables: son una generalización del caso anterior a n argumentos, esto es, $y = f(x_1, \dots, x_n)$.

Un ejemplo de este segundo caso sería la función $y = x_1^2 + x_2^2$. La manera de introducirla en R es:

```
> y <- function(x1,x2) x1^2 + x2^2
```

Si nos restringimos a las funciones de una sola variable, estas se pueden resumir en la tabla II.1.

Tabla II.1: Clases básicas de funciones

Clase	Forma funcional
Lineal	$mx + b$
Potencia	x^n
Raíz	$\sqrt[n]{x}$
Racional	$P(x)/Q(x)$
Exponencial	a^x
Logarítmica	$\log_a(x)$
Trigonométrica	$\sin(x)$ $\cos(x)$ $\tan(x)$

Además, las funciones pueden ser algebraicas si se pueden construir con operaciones algebraicas —suma, resta, multiplicación, división y raíz— o trascendentales, cuando esto no sucede.

3.2. Representación gráfica

3.2.1. Gráficos de una función

Una de las principales ventajas de R sobre otros programas es la potencia y versatilidad de sus gráficos. En este apartado so-

lo cubrimos una minúscula parte de ese potencial. La función básica de representación gráfica en R es `plot`. Esta función admite varios tipos de argumentos, esto es, tipos de objetos que pueden ser representados. En el caso de funciones de una sola variable como y , se pueden representar directamente. Las principales opciones que nos ofrece `plot` están recogidas en la tabla II.2.

Parámetros gráficos

Si acudimos a la ayuda de R, introduciendo en la consola la instrucción `help(par)`, obtendremos una extensa lista de opciones gráficas con las que podremos personalizar y moldear todos los aspectos del gráfico.

Tabla II.2: Opciones gráficas elementales

Descripción	Instrucción	Opciones
Título	<code>main</code>	"texto"
Nombre del eje x	<code>xlab</code>	"texto"
Nombre del eje y	<code>ylab</code>	"texto"
Dominio	<code>xlim</code>	<code>c(a,b)</code>
Rango	<code>ylim</code>	<code>c(a,b)</code>
Tipo de gráfico	<code>type</code>	Líneas: "l" Puntos: "p" Ambos: "o"
Ancho de línea	<code>lwd</code>	1, 2, ...
Color de línea	<code>col</code>	"red", "blue"...

Consideremos un ejemplo de una función con una variable $y = x\sqrt{x+3}$:

```
> y <- function(x) x*sqrt(x+3)
```

La representación de la función $y = x\sqrt{x+3}$ con R se puede programar de la siguiente manera.

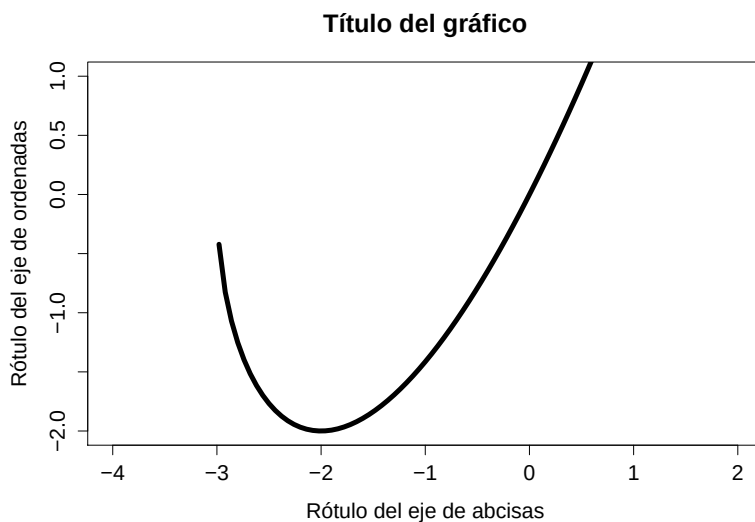
Consejo de programación

A la hora de programar gráficos, es aconsejable empezar una nueva línea después de cada coma, de manera que tengamos una instrucción por línea y visualmente sea todo más claro.

```
> plot(y,  
+      main="Título del gráfico",  
+      xlab="Rótulo del eje de abscisas",  
+      ylab="Rótulo del eje de ordenadas",  
+      xlim=c(-4,2),  
+      ylim=c(-2,1),  
+      type="l",  
+      lwd=5)  
Warning message:  
In sqrt(x + 3) : NaNs produced
```

El resultado se muestra en la figura II.1.

Figura II.1: Representación de la función $y = x\sqrt{x+3}$



El mensaje de aviso `In sqrt(x + 3) : NaNs produced` nos indica que hemos intentado representar el gráfico en el intervalo $x \in [-4, 3]$, pero el dominio de la función es $[-3, \infty]$, con lo cual los valores en el intervalo $[-4, -3]$ no se pueden mostrar, ya que no existen.

3.2.2. Gráficos de varias funciones

Otra manera de representar gráficamente funciones es mediante la creación de un vector de valores iniciales. Esto es, el argumento de la función `plot` no será la función propiamente, sino una serie de valores resultantes de esta. Esto resulta muy útil cuando representamos una función con más de una variable o más de una función en un gráfico, o cuando la función que queremos representar es por partes. Veremos un ejemplo con las funciones trigonométricas del seno, el coseno y la tangente. Estas son funciones implementadas en R y, por tanto, no hace falta crearlas. El primer paso será crear un vector de valores iniciales x_0 . Si queremos un gráfico de alta calidad, entre el máximo y el mínimo tendrá que haber muchos puntos intermedios. En nuestro caso, evaluaremos las tres funciones en el intervalo $x \in [-5, 5]$, conteniendo 1.000 subintervalos, de manera que el vector resultante tendrá una longitud de 1.001:

```
> x0 <- seq(-5, 5, 0.01)

> (lon <- length(x0))
[1] 1001
```

Después creamos tres vectores que sean la evaluación de las tres funciones en los puntos x_0 . Lógicamente, la longitud de los vectores resultantes será la misma:

```
> y1 <- sin(x0)
> y2 <- cos(x0)
> y3 <- tan(x0)
```

El siguiente paso es la representación gráfica. Como tenemos más de una función es recomendable crear primero el marco del gráfico, esto es, los valores máximo y mínimo en los ejes de coordenadas. Los rangos de los ejes serán $(1, 1.001)$ para las abscisas y $(-4, 4)$ para las ordenadas. Los dos primeros argumentos son los rangos, la instrucción `type="n"` especifica que no hay función, `xaxt="n"` elimina el eje de abscisas y tanto `xlab=""` como `ylab=""` dejan los nombres de los ejes en blanco:

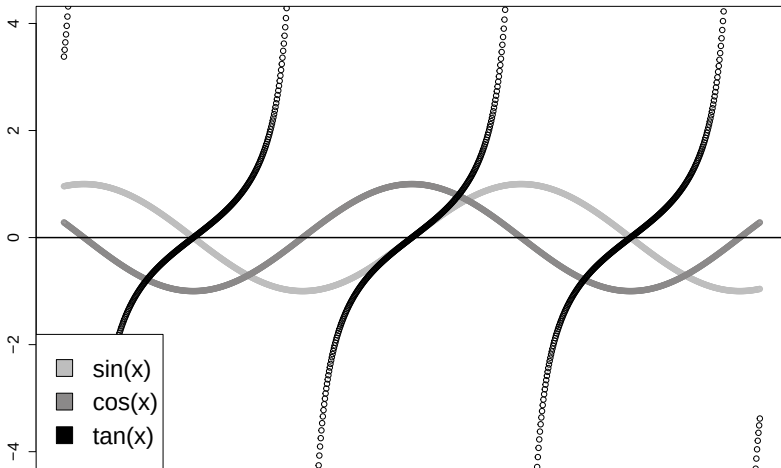
```
> (xrange <- c(1,1.001))
[1] 1 1.001
> (yrange <- c(-4,4))
[1] -4 4
> plot(xrange,yrange,type="n",xaxt="n",xlab="",
+      ylab="")
```

Ahora ya podemos ir añadiendo valores al cuadro mediante la función `lines`, donde especificamos el vector que vamos a representar, así como las características de la representación —tipo de línea, grosor, color, etc. Además, mediante la función `abline` dibujaremos una línea horizontal en el punto $y = 0$ mediante la opción `h=0`:

```
> lines(y1,type="p",lwd=1,col="grey")
> lines(y2,type="p",lwd=1,col="snow4")
> lines(y3,type="p",lwd=1,col="black")
> abline(h=0,lwd=2)
```

Es interesante notar que hemos elegido para las líneas tres colores de los muchos que nos ofrece la paleta de colores de R. El último paso es añadir la leyenda, que se efectúa con la función `legend`. Aquí especificamos dónde queremos la leyenda —abajo e izquierda—, el texto asociado a cada función, el color de cada función y, por último, `cex`, que indica la proporción de la leyenda con relación al gráfico —un valor mayor indicará un tamaño mayor. El resultado final se muestra en la figura II.2.

Figura II.2: Representación de las funciones $\sin(x)$, $\cos(x)$ y $\tan(x)$



Parámetros de la leyenda

Introduciendo la instrucción `help(legend)` obtendremos una detallada descripción de las opciones para personalizar el cuadro de la leyenda.

```
> legend("bottomleft",
+       c("sin(x)", "cos(x)", "tan(x)"),
```

```
+ fill=c(" grey","snow4","black"),
+ cex=1.5)
```

Un tipo de función muy común es el de la función por partes o función definida a trozos. Consideremos la siguiente función:

$$f(x) = \begin{cases} 2 & \text{si } x < -2 \\ x^2 - 4 & \text{si } x \in [-2, 3) \\ 8 - x & \text{si } x \geq 3 \end{cases}$$

Supongamos que queremos evaluar dicha función y representarla gráficamente en el intervalo $x \in (-5, 5)$. Para ello hemos de utilizar ciclos y condicionales, tal y como se muestra a continuación:

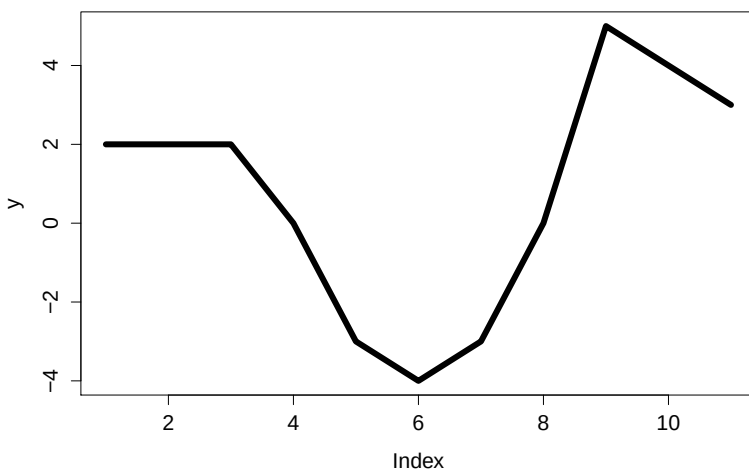
Ciclos y condicionales

Esta estructura puede parecer enrevesada y compleja. Vale la pena que la leáis más de una vez para intentar entender qué es lo que le estamos pidiendo a R.

```
> plot(y,type="l",lwd=8)
> x0 <- (-5:5)
> n <- length(x0)
> y <- rep(0,n)
> for (i in 1:n){
+     x <- x0[i]
+     if (x< -2){
+         y[i] <- 2
+     }else if (x >= -2 & x < 3){
+         y[i] <- x*x - 4
+     }else{
+         y[i] <- 8 - x
+     }
+ }
> plot(y,type="l",lwd=6)
```

Es interesante analizar esta pieza de código. Lo que hacemos es que mediante condicionales (`if` y `elseif`) asignamos valores al vector `y` según el valor que tome `x`. Además, recordemos que el símbolo `&` significa intersección, de manera que la expresión lógica `x >= -2 & x < 3` se lee como «que el valor de x sea mayor o igual que -2 y que sea menor que -3 ». El gráfico resultante se muestra en la figura II.3.

Figura II.3: Representación de una función definida a trozos



3.2.3. Gráficos en tres dimensiones

Este tipo de gráficos se emplean para representar funciones en $\mathbb{R}^3$, es decir, del tipo $y = f(x_1, x_2)$, con lo que su visualización ha de hacerse sobre un sistema de tres ejes de coordenadas —es decir, un espacio euclidiano tridimensional. Partamos de un ejemplo conocido: la función paraboloides hiperbólico —también conocida como punto de silla—:

$$z = f(x, y) = x^2 - y^2$$

Supongamos que deseamos evaluar esta función para los valores de x e y incluidos en el dominio $[-100, 100]$. El primer paso será crear los vectores de valores iniciales x_0 e y_0 y definir la función z :

```
> x0 <- -100:100
> y0 <- -100:100
> z <- function(x,y) x*x-y*y
```

Llegados a este punto, existe una función de R llamada **outer** que resulta de gran utilidad. Esta función utiliza como argumentos los vectores $\mathbf{x} = (x_1, \dots, x_n)$ e $\mathbf{y} = (y_1, \dots, y_n)$ y una función $f(x, y)$, y calcula la siguiente matriz $n \times n$:

$$\begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) & \cdots & f(x_1, y_n) \\ f(x_2, y_1) & f(x_2, y_2) & \cdots & f(x_2, y_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_n, y_1) & f(x_n, y_2) & \cdots & f(x_n, y_n) \end{pmatrix}$$

Una vez creada esta matriz, la podemos representar gráficamente mediante la función **persp**, que representa los valores recogidos en una matriz en un espacio tridimensional. El gráfico resultante se muestra en la figura II.4.

```
> f <- outer(x0,y0,z)
> persp(f)
```

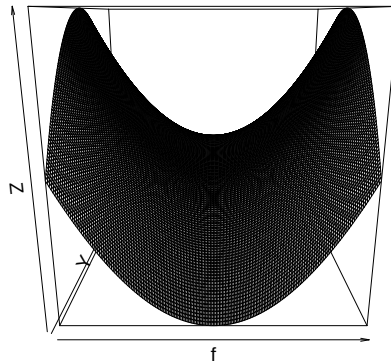
4. Cálculo diferencial e integral

Cálculo

La palabra *cálculo* proviene del latín *calculus*, que era una pequeña piedra usada para contar. El cálculo es el estudio matemático del cambio y tiene dos ramas principales: el cálculo

diferencial, que estudia las ratios de cambio y las pendientes de las curvas, y el cálculo integral, que se encarga de la acumulación de cantidades y de las áreas bajo las curvas. Ambas ramas están relacionadas por el teorema fundamental del cálculo.

Figura II.4: Representación de la función hiperbólica $z = x^2 - y^2$



4.1. Aproximación a la derivada

El objetivo de esta sección es estudiar las posibilidades que R ofrece en cuanto al cálculo diferencial. Un aspecto destacable es que R no ofrece la potencia de otros programas —como Mathematica— en lo que se refiere al cálculo simbólico de derivadas e integrales. Sin embargo, trabajando con un vector de datos iniciales, R nos permite el cálculo de rectas, curvas, secantes, tangentes y áreas mediante la aplicación de técnicas de análisis numérico.

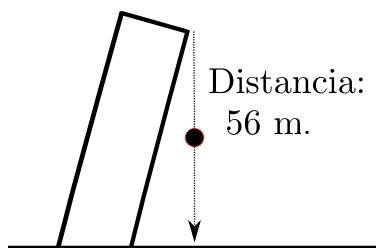
Para aproximar el concepto de derivada, consideraremos un ejemplo de física. El científico italiano Galileo estudió la caída de objetos desde las alturas para describir su comportamiento

y llegó a la conclusión de que la velocidad de caída no dependía de la masa del objeto. Él propuso la siguiente ecuación:

$$s(t) = 4,9t^2$$

Esta es una ecuación de movimiento, en la cual la distancia recorrida del objeto (s) es una función del tiempo de caída (t). La forma cuadrática de la ecuación sugiere que el objeto cae más rápido a medida que aumenta el tiempo de caída.² Galileo realizó una serie de experimentos dejando caer objetos desde lo alto de la Torre de Pisa —56 metros de alto—, como muestra la ilustración de la figura II.5.

Figura II.5: Experimentos de Galileo



Galileo Galilei

Galileo Galilei (1564-1642) fue un físico, matemático, astrónomo y filósofo italiano que ha tenido un papel fundamental en la historia de la ciencia. Se le considera el padre de la astronomía y de la física moderna y, en general, el padre de la ciencia moderna.

El primer paso para analizar este fenómeno en $\mathbb{R}$ es introducir la función de la caída, a la que llamaremos s :

²Experimentos posteriores han demostrado que llega un momento en el que el objeto en caída libre llega a una velocidad terminal, a partir del cual la velocidad es constante y la aceleración es nula.

```
> s <- function(t) 4.9*t^2
```

Un primer dato fundamental que hemos de saber es el tiempo total ($t = T$) que pasa desde que el objeto cae desde lo alto de la torre hasta que llega al suelo. Algebraicamente esto es sencillo, basta con realizar el siguiente cálculo:

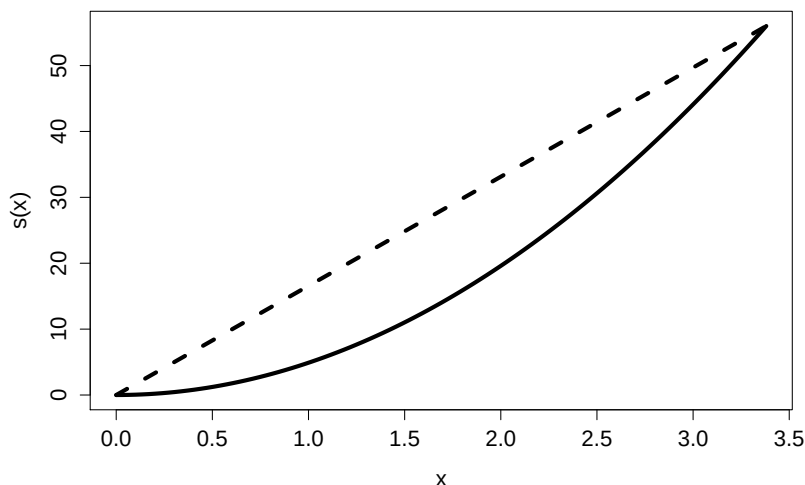
$$56 = 4,9T^2 \Rightarrow T = \sqrt{\frac{56}{4,9}} \approx 3,38 \text{ s}$$

Este cálculo en R es inmediato: bastará con buscar el valor de t^* que satisfaga la condición $s(t^*) - 56 = 0$, lo cual se hace buscando la raíz de esta expresión:

```
> (T <- uniroot(function(t) 4.9*t^2 - 56,
+   c(0,5))$root)
[1] 3.380618
```

Una vez calculado el valor de T , podemos visualizar la función en el dominio que nos interesa, esto es, $t \in (0, T)$. Además, representaremos también una línea recta (discontinua) que unirá los puntos $(0, 0)$ y $(T, 56)$. Se trata de una recta secante, ya que corta la parábola por dos puntos. La pendiente de la recta secante es constante y corresponde a la velocidad media del objeto desde su lanzamiento hasta el contacto con el suelo. Esto es, un objeto que recorre 56 metros en 3,38 segundos ha viajado a una velocidad media de $\bar{v} = 56/3,38 = 16,56 \text{ m/s}$. La figura II.6 ilustra el resultado.

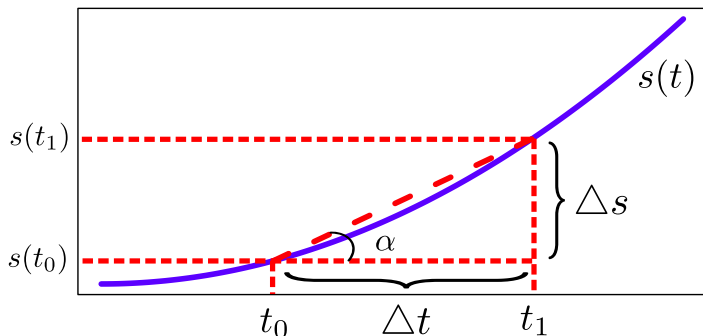
```
> plot(s, xlim=c(0,T), lwd=4, ylab="s(x)")
> lines(c(0,T), c(0,56), lty=2, lwd=4)
```

Figura II.6: Función del movimiento y recta secante

En la figura II.6 se aprecia cómo la línea continua muestra la distancia recorrida por el objeto en función del tiempo. La línea secante (discontinua) muestra cuál habría sido el recorrido del objeto si su velocidad hubiera sido constante durante toda la caída. De hecho, la pendiente de la línea discontinua coincide con la velocidad media del objeto durante la caída.

Sin embargo, la velocidad no es constante a lo largo de la función. Para obtener la velocidad específica en un tramo determinado de la caída, deberemos calcular la recta secante entre dos puntos. Esto es, la velocidad media entre los momentos t_0 y t_1 se calcula a partir de la recta que pasa por los puntos $(t_0, s(t_0))$ y $(t_1, s(t_1))$, tal y como muestra la figura II.7.

La figura II.7 ilustra el concepto de derivada, que sería la pendiente de la línea discontinua cuando $\Delta t = t_1 - t_0$ tiende a cero. En este caso, la derivada muestra cuál es la velocidad instantánea del objeto en un momento exacto del tiempo, no en un intervalo.

Figura II.7: Velocidad media entre dos puntos

Matemáticamente, la velocidad media entre t_0 y t_1 corresponde a la pendiente de la recta entre ambos puntos, es decir, la tangente del ángulo α :

$$\tan \alpha = \frac{s(t_1) - s(t_0)}{t_1 - t_0} = \frac{\Delta s}{\Delta t}$$

Llegados a este punto ¿cómo calcularíamos la velocidad instantánea en un punto? Si hacemos el cálculo anterior reduciendo cada vez más Δt , en el límite la recta secante se aproxima a una recta tangente en un punto determinado, con lo cual obtendríamos la velocidad para un valor de t en lugar de un intervalo. Matemáticamente la derivada de la función $s(t)$ en el punto t_0 viene dada por el siguiente límite:

$$s'(t_0) = \lim_{\Delta t \rightarrow 0} \frac{s(t_0 + \Delta t) - s(t_0)}{\Delta t}.$$

Donde Δt es también denominado diferencial de t y expresado como dt . La derivada se puede generalizar como una función $s'(t)$ derivada de la original $s(t)$, que indica, para cada valor de t , la pendiente de la función original $s(t)$. En R se puede calcular la función derivada de funciones sencillas de forma simbólica, mediante las funciones `deriv` y `D`:

```
> D(expression(4.9*t^2),"t")
4.9 * (2 * t)
```

Sin embargo, si seguimos una aproximación de análisis numérico, para calcular el valor de la derivada para un valor específico de t , lo que podemos hacer es definir un valor arbitrario de dt lo suficientemente pequeño y aplicar la fórmula antes descrita. Por ejemplo, la velocidad del objeto a los dos segundos de caída ($t = 2$) es:

```
> dt <- 0.0001
> t0 <- 2
> (s(t0+dt)-s(t0))/dt
[1] 19.60049
```

Esto es, aproximadamente 19,6 m/seg. Este cálculo se puede generalizar para calcular empíricamente la primera derivada en el dominio $t \in (0, T)$. El procedimiento es sencillo: atribuimos a dt un valor arbitrariamente pequeño y creamos un vector de valores iniciales $t_0 = (0, dt, 2dt, \dots, T)$. Seguidamente evaluamos la función con todos los valores del vector, esto es, $s(t_0) = (s(0), s(dt), s(2dt), \dots, s(T))$. Con esto ya tenemos los valores de la función $s(t)$ y para calcular los valores de la función derivada $s'(t)$ calcularemos un vector con los incrementos de $s(t)$:

$$[s(dt) - s(0), s(2dt) - s(dt), \dots, s(T) - s(T - dt)]$$

Este vector lo dividiremos entre dt , con lo cual obtendremos el vector con las pendientes en cada punto $ds.dt$. El resultado puede verse en la figura II.8.

```
> dt <- 0.01
> t.0 <- seq(0,3.38,by=dt)
> n <- length(t.0)
```

```

> f.s <- s(t.0)
> inc.s <- f.s[2:n] - f.s[1:(n-1)]
> ds.dt <- inc.s/dt

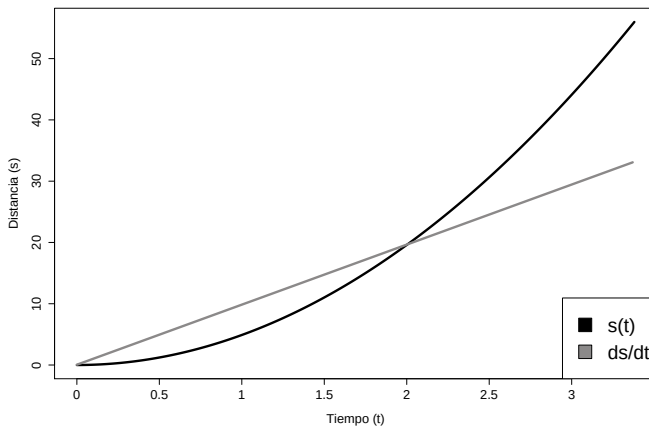
> xrange <- c(1,n)
> yrange <- c(0,max(f.s))

> plot(xrange,yrange,type='n',xaxt='n',
+       xlab='Tiempo (t)',ylab='Distancia (s)')
> lab=1+50*0:6
> axis(1,at=lab,labels=t.0[lab])
>
> lines(f.s,type='l',lwd=4,col='black')
> lines(ds.dt,type='l',lwd=4,col='snow4')

> legend('bottomright',c('s(t)', 'ds/dt'),
+       fill=c('black', 'snow4'),cex=1.5)

```

Figura II.8: Función original y primera derivada



En la figura II.8 se puede observar que el objeto aumenta la velocidad progresivamente hasta llegar al suelo a 33 m/s

aproximadamente. En otras palabras, el objeto tiene una aceleración positiva y constante, ya que la segunda derivada de $s(t)$ es igual a cero —es decir, la pendiente de la derivada no varía.

4.2. Integración

La integración se puede entender como el proceso inverso a la diferenciación. Siguiendo con el ejemplo anterior, hemos visto que la función de movimiento $s(t)$ tiene una derivada $s'(t) = ds/dt$ que representa la velocidad instantánea en cada momento t del objeto. Supongamos que disponemos de la función de la velocidad $s'(t)$ y deseamos obtener la función de la distancia total $s(t)$. La integración nos permite revertir el proceso de diferenciación: $s(t) = \int s'(t)dt$. Visto de otra manera, mientras la derivación se basa en calcular pendientes ($\Delta s/\Delta t$), la integración consiste en calcular áreas ($\Delta s \cdot \Delta t$).

Para ilustrar este concepto, seguiremos con el ejemplo de la caída de objetos en el vacío. La fórmula usada en el ejemplo anterior corresponde a una aproximación de Galileo, válida para distancias pequeñas. Sin embargo, hay muchos factores que afectan a la velocidad de un objeto en caída libre: la aceleración debida a la gravedad, el área del objeto, la densidad del aire, la resistencia aerodinámica, etc. Una aproximación más exacta propone que la velocidad de un objeto en caída libre aumentará hasta aproximarse asintóticamente a una velocidad terminal v_∞ , que depende de muchos factores. Así, la fórmula de la velocidad se define como:³

$$v(t) = v_\infty \tanh\left(\frac{gt}{v_\infty}\right)$$

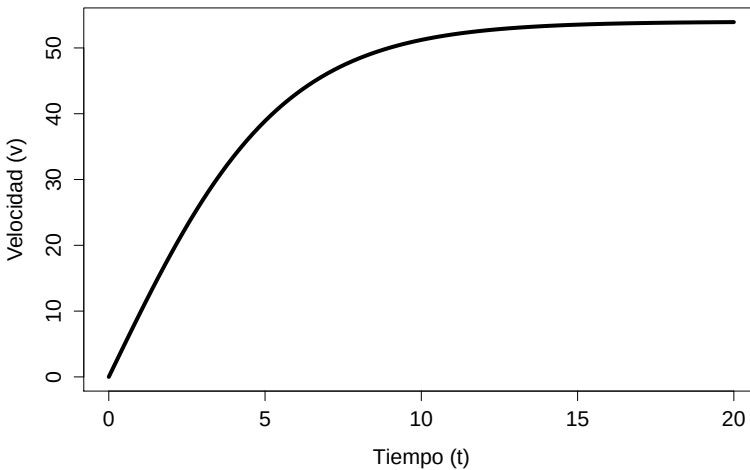
³Recordemos que la función tangente hiperbólica se define como $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

Donde $g = 9,81 \text{ m/s}^2$ es la aceleración gravitacional y v_∞ es la velocidad terminal, que toma el valor aproximado de 54 m/s en el caso de un paracaidista medio. El primer paso es introducir la función de la velocidad en R y representarla gráficamente en la figura II.9.

```
> f.v <- function(t) 54*tanh(9.81*t/54)

> plot(f.v,xlim=c(0,20),lwd=4,col='magenta',
+      xlab='Tiempo (t)',ylab='Velocidad (v)')
```

Figura II.9: Velocidad de un paracaidista en caída libre



Velocidad de un paracaidista

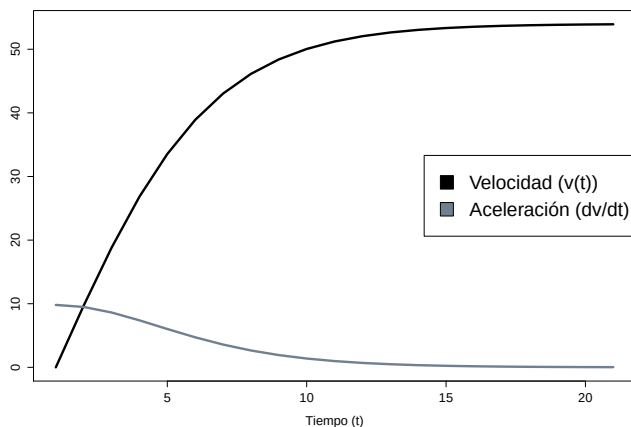
Este gráfico muestra la velocidad de un paracaidista en caída libre asumiendo una densidad del aire constante. El famoso salto estratosférico de Felix Baumgartner el 14 de octubre de 2012 logró una velocidad mucho mayor al saltar de una altura de 39 kilómetros, donde la densidad del aire ofrece mucha menos resistencia.

La figura II.9 se puede complementar representando la aceleración, es decir, el cambio de velocidad por unidad de tiempo. Matemáticamente se define como la derivada de la función de velocidad $v(t)$. Aprovechando la función `D` que nos ofrece `R`, obtenemos $v'(t)$ y le damos el nombre de `f.a`:

```
> D(expression(54*tanh(9.81*t/54)), "t")
> 54 * (9.81/54/cosh(9.81 * t/54)^2)
> f.a <- function(t) 54 * (9.81/54/cosh(9.81 * t/
  54)^2)
```

Ahora podemos evaluar ambas funciones en el dominio $t_0 \in (0, 20)$, cuyo resultado se muestra en la figura II.10. La aceleración al principio es aproximadamente $v'(0) = 10$ y a medida que el tiempo avanza se aproxima asintóticamente a 0.

Figura II.10: Velocidad y aceleración de un paracaidista en caída libre



```
> t.0 <- 0:20
> plot(f.v(t.0), type='l', lwd=4, col='black',
+       xlab='Tiempo (t)', ylab='')
```

```
> lines(f.a(t.0),lwd=4,col='slategray')

> legend('right',c('Velocidad (v(t))',
+ 'Aceleración (dv/dt)'),fill=c('black',
+ 'slategray'),cex=1.5)
```

Si estuviéramos interesados en la distancia que recorre el paracaidista deberíamos realizar la operación inversa, es decir, aproximar la función de movimiento a partir de la función de velocidad. Supongamos que estamos interesados en la distancia recorrida por el paracaidista durante los diez primeros segundos de caída. Una opción es registrar la velocidad en intervalos de dos segundos y multiplicarla por ese intervalo $\Delta t = 2$. El resultado se muestra en la tabla II.3.

Tabla II.3: Aproximación de la distancia a partir de la velocidad

j	t_j	$v(t_j)$	$\Delta t \cdot v(t_j)$
1	2	18,80	37,70
2	4	33,53	67,07
3	6	43,03	86,06
4	8	48,40	96,80
5	10	51,22	102,43
Distancia total:			389,98

En R, reproducir los cálculos de la tabla II.3 es inmediato:

```
> dt <- 2

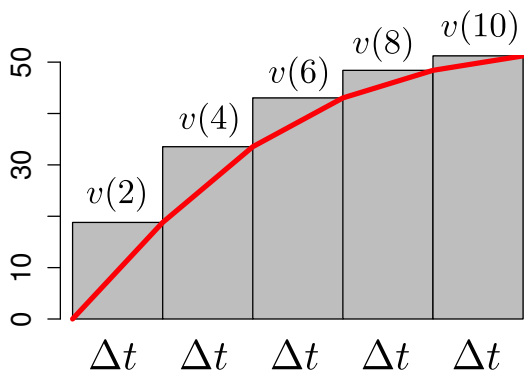
> t.0 <- dt*1:5

> (v.0 <- f.v(t.0))
[1] 18.79992 33.53518 43.03142 48.40290 51.21927

> sum(2*v.0)
[1] 389.9774
```

La figura II.11 ofrece una interpretación geométrica de los cálculos de distancia mostrados en la tabla II.3. En esta figura se puede comprobar que la suma de las áreas de las cinco barras verticales se corresponde con los cinco valores obtenidos en la tabla II.3, cuya suma es aproximadamente 390 metros. Vemos también que esta cantidad aproximada es superior a la real, ya que la suma de estas áreas es superior al área que hay debajo de la curva.

Figura II.11: Cálculo aproximado de la distancia



El cálculo del área que hemos hecho se denomina *suma de Riemann* y corresponde a un método para aproximar el área bajo una curva:

$$S = \sum_{j=1}^n v(t_j) \cdot \Delta t$$

En la figura II.11 hemos visto una aproximación con $\Delta t = 2$ y $n = 5$, con lo cual no es una aproximación muy exacta. Se puede comprobar que el área gris es superior al área que queda por debajo de la función $v(t)$, con lo que la distancia de 389,98 metros es superior a la real. Obtendremos un cálculo más exacto si hacemos que $\Delta t \rightarrow 0$ y $n \rightarrow \infty$. Esto nos da paso a la definición de la *integral de Riemann*: para calcular

el área bajo una curva entre los puntos a y b , calculamos una partición del intervalo $[a, b]$ tal que $a = t_0 < t_1, \dots, t_n < b$, siendo Δt la distancia entre los valores del intervalo.

$$S = \lim_{\Delta \rightarrow 0} \sum_{j=1}^n v(t_j) \cdot \Delta t = \int_a^b v(t) dt$$

La integral de Riemann es, pues, el límite de una suma. Para su cálculo en R crearemos una función a nuestra medida:

```
> int.riemann <- function(fun,a,b,dt){  
+   t.0 <- seq(a,b,by=dt)  
+   f.0 <- fun(t.0)  
+   suma <- sum(f.0*dt)  
+   return(suma)  
+ }
```

En el caso anterior habíamos calculado el área en el intervalo $[0, 10]$ con $\Delta t = dt = 2$:

```
> int.riemann(f.v,0,10,2)  
[1] 389.9774
```

Este resultado será más exacto a medida que $dt \rightarrow 0$:

```
> int.riemann(f.v,0,10,1)  
[1] 366.5903  
  
> int.riemann(f.v,0,10,0.1)  
[1] 344.2708  
  
> int.riemann(f.v,0,10,0.01)  
[1] 341.9732  
  
> int.riemann(f.v,0,10,0.001)  
[1] 341.7428
```

Esta manera de aproximar la integral la hemos realizado con el objetivo de ilustrar el concepto de integral. En general, hay varios métodos para aproximar el área bajo una curva y R incorpora una función denominada `integrate`, que aproxima el área bajo una curva mediante el uso de una cuadratura adaptativa con una tolerancia de error de aproximadamente $1,22 \times 10^{-4}$.

```
> integrate(f.v,0,10)
341.7172 with absolute error < 3.8e-12
```

Calculando la integral de esta manera, se comprueba que la distancia real se aproxima a 341,71 metros. Por tanto, la función de R `integrate` es la que mejor aproxima esta cantidad.

4.3. Ecuaciones diferenciales

Las ecuaciones diferenciales

Estas ecuaciones desempeñan un papel destacado en la ingeniería, la física, la economía y otras disciplinas. Surgen cuando existe una relación determinística entre variables continuas (funciones) y sus tasas de cambio en el espacio y/o tiempo —sus derivadas.

Las ecuaciones diferenciales son ecuaciones que contienen una función desconocida —de una o varias variables—, además de una o varias derivadas de esta función. El orden de una ecuación diferencial será el orden de la derivada más alta en la ecuación y la solución será una función f si la ecuación se satisface cuando $y = f(x, \dots)$.

Las ecuaciones diferenciales más sencillas se pueden resolver usando las reglas de diferenciación e integración. Veamos cómo se resolvería la ecuación diferencial $y' = cx$:

$$\begin{aligned}y' = cx &\quad \Rightarrow \quad \frac{dy}{dx} = cx \\dy = cx \, dx &\quad \Rightarrow \quad \int dy = \int cx \, dx \\y = \frac{cx^2}{2} + C\end{aligned}$$

Sin embargo, resolver ecuaciones diferenciales más complejas no es una tarea nada fácil. En este sentido, R no es un programa capaz de resolver ecuaciones diferenciales encontrando la solución general de manera simbólica. Aun así, R es capaz de encontrar una solución particular que satisfaga una condición del tipo $y(x_0) = y_0$. Esta se denomina *condición inicial* y el problema de encontrar una solución para la ecuación diferencial que satisfaga esta condición inicial se denomina *problema de valor inicial*.

El paquete de R que resuelve este tipo de problemas se denomina *deSolve*, que hay que instalar previamente.⁴ Es un paquete muy completo y en la página web del CRAN se puede encontrar un tutorial muy detallado de cómo aplicar sus diferentes usos. Además, si accedemos a «cran.r-project.org / Packages / CRAN Task Views», veremos una categoría llamada «Differential Equations», bajo la cual hay disponibles otros paquetes que desarrollan otras funcionalidades. El artículo *Solving differential equations in R: Package deSolve* (Soetaert, 2009) proporciona una introducción muy didáctica a esta librería con elementos teóricos y aplicaciones prácticas.

Consideremos un ejemplo del campo de la psicología: el estudio de las curvas de aprendizaje. Para un estudiante que está

⁴Esto es tan fácil como introducir `install.packages("deSolve")` en la consola.

aprendiendo R, su curva de aprendizaje es una función $Y(t)$, donde Y es su rendimiento, que depende del tiempo (t). Un modelo recurrente de aprendizaje viene dado por la siguiente ecuación diferencial:

$$\frac{dY}{dt} = k(M - Y)$$

Donde dY/dt es la velocidad de aprendizaje —la ratio a la que mejora el rendimiento—, k es una constante y M es el nivel máximo al cual puede llegar un estudiante, de manera que siempre se cumple que $Y \leq M$. La condición inicial que fijamos para este problema es que $Y(t_0) = 0$, es decir, que el estudiante al principio no sabe nada de R.

Para aproximar este problema, hemos de introducir la condición inicial y los parámetros, además del periodo temporal en el que queremos estudiar la ecuación. Asumiremos que la escala de aprendizaje va de 0 a 10, de manera que $M = 10$ y que $k = 0,02$. Además, estudiaremos la progresión del aprendizaje en un año, de modo que $t_0 = 0, \dots, 365$ días. En la siguiente pieza de código se puede comprobar que, introduciendo los valores en vectores de esta manera, estamos asignando un nombre a cada valor, lo cual es importante para los cálculos posteriores:

```
> library(deSolve)
> parametros<-c(k=0.02,M=10)
> cond.ini <- c(Y=0)
> tiempo <- 0:365
```

El siguiente paso es introducir la ecuación diferencial como una función. Es fundamental introducirla de una manera específica, tal y como se muestra a continuación:

```
> f.apren <- function(t, cond.ini,
+   parametros){
+   with(as.list(c(cond.ini, parametros)),{
+     dY <- k*(M-Y)
+     return(list(dY))
+   })}
```

Esto es, los argumentos de la función son el tiempo, las condiciones iniciales y los parámetros. Además, mediante la opción `with` estamos incluyendo los valores de los vectores `cond.ini` y los `parámetros` creados anteriormente. Por último, es fundamental que el *output* de esta función sea una lista, tal como muestra la instrucción `return(list(dY))`. Una vez hecho esto, ya podemos aproximar la solución a la ecuación diferencial mediante la función `ode`:

```
> sol <- ode(cond.ini, tiempo, f.apren, parametros)
```

El resultado será una matriz con dos columnas y tantas filas como valores de t . Veamos las partes superior e inferior de este resultado:

```
> head(sol)
      time      Y
[1,]    0 0.0000000
[2,]    1 0.1980142
[3,]    2 0.3921054
[4,]    3 0.5823544
[5,]    4 0.7688362
[6,]    5 0.9516254

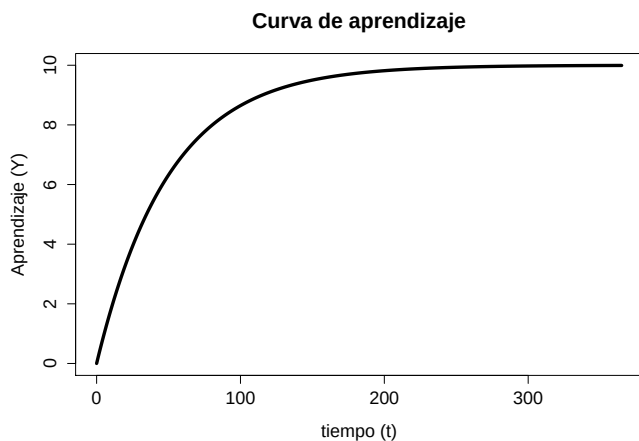
> tail(sol)
      time      Y
[361,]  360 9.992534
[362,]  361 9.992682
[363,]  362 9.992827
```

```
[364,] 363 9.992969  
[365,] 364 9.993108  
[366,] 365 9.993245
```

La interpretación de los resultados de una ecuación diferencial se realiza gráficamente. En nuestro caso, una representación en un plano bidimensional de las variables Y y t , que se muestra en la figura II.12.

```
> plot(sol,xlab="tiempo (t)",  
+ ylab="Aprendizaje (Y)",  
+ lwd=4,col="red",  
+ main="Curva de aprendizaje")
```

Figura II.12: Solución de la ecuación diferencial



En esta curva se puede ver que en los primeros días el valor de la derivada de la curva (dY/dt) es muy grande, esto es, se aprende muy rápidamente. A medida que se van consolidando los conocimientos, esta derivada se reduce, ya que el aprendizaje tiende a su límite superior $Y = 10$.

Hay que destacar que este es un ejemplo muy sencillo. R y el paquete **deSolve**, además de otros paquetes destinados a ecuaciones diferenciales, ofrecen muchas posibilidades para resolver sistemas de ecuaciones diferenciales y otros problemas más complejos.

5. Optimización

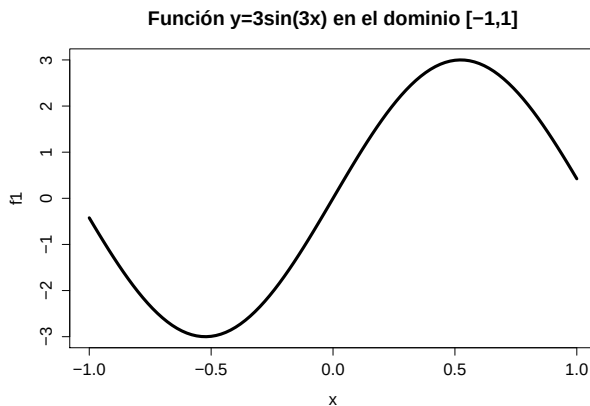
En general, la función de R **optimize** busca, para una función, el mínimo y el máximo en el intervalo del dominio $[a, b]$ con respecto a su primer argumento. Consideremos la sencilla función $y = 3\sin(3x)$: la vamos a introducir y la vamos a representar gráficamente en el dominio $[-1, 1]$ en la figura II.13.

```
> f1 <- function(x) 3*sin(3*x)

> plot(f1,from=-1,to=1,lwd=4)

> title("Función  $y=3\sin(3x)$  en el dominio  $[-1,1]$  ")
)
```

Figura II.13: Representación de la función $y = 3 \cdot \sin(3x)$



Para encontrar el valor mínimo de esta función en el intervalo $[-1, 1]$ bastará con introducir la siguiente instrucción:

```
> optimize(f1,c(-1,1))
$minimum
[1] -0.5235812

$objective
[1] -3
```

Vemos que el resultado final es una lista con dos elementos: el valor de x donde está el mínimo $(-0,523)$ y el valor de la función $f(x)$ en ese punto, es decir, -3 . Si deseáramos buscar el máximo de la función en lugar del mínimo, bastará con introducir la opción `maximum=TRUE`, como se muestra en el siguiente ejemplo:

```
> optimize(f1,c(-1,1),maximum=TRUE)
$maximum
[1] 0.5235812

$objective
[1] 3
```

En general, para funciones complejas y de más de un argumento, existe la función `optim`. Consideremos como ejemplo la función con dos argumentos $y = f(x_1, x_2) = 2x_1^2 + 2x_2^2 - 4$. Para buscar el mínimo de esta función tendremos que introducirla en R de una manera ligeramente diferente. Esto es, en esta función, los dos argumentos de la función (x_1 y x_2) están introducidos conjuntamente en un único vector x , que contiene las dos variables.

```
> f.y <- function(x){
+   x1 <- x[1]
```

```
+   x2 <- x[2]
+   2*x1^2 + 2*x2^2 - 4
+ }
```

Para buscar el mínimo de esta función, deberemos introducir unos valores iniciales de x_1^* y x_2^* , ya que la función `optim` aplicará un proceso iterativo a partir de estos valores para buscar la convergencia en los valores de x_1 y x_2 que minimicen la función y . En este ejemplo, probaremos aleatoriamente con los valores $x_1^* = -1$ y $x_2^* = 1,5$:

La función `optim`

Técnicamente, esta función se basa en los algoritmos de gradiente conjugado, Nelder-Mead y cuasi-Newton. Para procesos de optimización más complejos, se recomienda ver la ayuda de esta función escribiendo `help(optim)` y considerar las diferentes opciones que esta función ofrece.

```
> optim(c(-1,1.5),f.y)
$par
[1] 9.038020e-05 -4.059674e-05
$value
[1] -4
$counts
function gradient
      69      NA
$convergence
[1] 0
$message
NULL
```

El resultado que nos interesa de esta lista es `par`, el cual muestra un vector con dos valores en notación científica, muy próximos a cero. Así que podemos asumir que los valores $x_1 = 0$ y $x_2 = 0$ constituyen un mínimo, siendo el valor de la función $y(0,0) = -4$.

Las opciones de optimización con R no se acaban aquí. Se pueden resolver problemas de optimización con restricciones, optimización cuadrática, etc. En la página web del CRAN hay disponible una lista de categorías de paquetes por disciplinas:

[cran.r-project.org / Packages / CRAN Task Views](http://cran.r-project.org/Packages/)

Ahí encontraremos un elemento llamado *Optimization* y si accedemos a él veremos los diferentes paquetes disponibles y sus capacidades. Además, el manual *Optimización matemática con R: Volumen I*, de Enrique Gabriel Baquela y Andrés Redchuk, disponible en la página web del CRAN, ofrece una completa revisión de los procedimientos de optimización aplicados con R.

6. Análisis de variable compleja

En matemáticas, muy a menudo nos encontramos con la necesidad de resolver ecuaciones del tipo $x^2 = -1$. Sabemos que no existe ningún número real cuyo cuadrado sea un número negativo, pero eso no significa que no exista ningún número que satisfaga esta ecuación. La respuesta está en los números complejos. Estos se basan en el número $i = \sqrt{-1}$, que satisface que $i^2 = (\sqrt{-1})^2 = -1$. El número i recibe el nombre de número imaginario.

Hay dos formas de expresar números complejos. La primera es la forma binómica, esto es $a+bi$, siendo a y b números reales. Se dice entonces que a es la parte real del número complejo y bi es la parte imaginaria. Como hemos visto anteriormente, a la hora de buscar las raíces de una ecuación, R nos da números en notación compleja. Sirva como ejemplo la ecuación $x^2 - 10x + 40 = 0$, cuyas raíces son $x^* = 5 \pm \sqrt{-15} = 5 \pm \sqrt{15}i$:

```
> polyroot(c(40,-10,1))  
[1] 5+3.872983i 5-3.872983i
```

Evidentemente, también es posible introducir números complejos y hacer operaciones con ellos. Veamos dos formas de introducir el número $z = 10 + 5i$:

```
> (z <- 10-5i)  
[1] 10-5i  
  
> (z <- complex(real=10,imaginary=-5))  
[1] 10-5i
```

Veamos cómo se pueden extraer y aislar de los números complejos la parte real (**Re**) e imaginaria (**Im**).

```
> Re(z)  
[1] 10  
  
> Im(z)  
[1] -5
```

En cuanto a las operaciones algebraicas permitidas, la suma (resta) de números complejos se efectúa sumando (restando) las partes reales y las partes imaginarias por separado:

```
> z1<-5+1i  
> z2<-10+4i  
  
> z1+z2  
[1] 15+5i  
  
> z1-z2  
[1] -5-3i
```

En cuanto a la multiplicación de un número complejo por un número real, multiplicamos las partes real e imaginaria del número complejo por el número real. Para multiplicar dos números complejos, hemos de aplicar la propiedad distributiva de los polinomios, teniendo en cuenta que $i^2 = -1$. Y en cuanto a su división, multiplicaremos y dividiremos por el conjugado del número complejo del denominador:

```
> 5*z1
[1] 25+5i

> z1*z2
[1] 46+30i

> z1/z2
[1] 0.4655172-0.0862069i
```

Una transformación de interés es el conjugado de un número complejo $z = a + bi$, que es otro número complejo llamado $\bar{z}$ que tiene la parte imaginaria cambiada de signo. Una de sus propiedades es que multiplicar un número complejo por su conjugado resulta en un número real. En R, el conjugado se extrae mediante el operador `Conj`:

```
> (z3<-4+2i)
[1] 4+2i

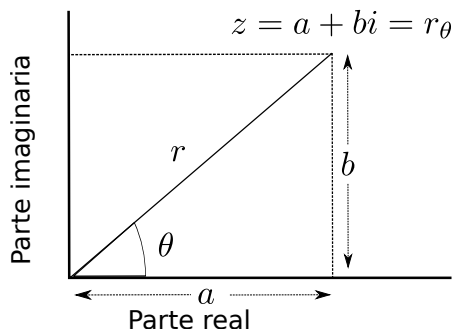
> (z4<-Conj(z3))
[1] 4-2i

> z3*z4
[1] 20+0i
```

El número complejo $z = a + bi$ se puede expresar alternativamente en forma polar. Para ello se consideran dos valores: r es el módulo, que es la distancia del punto al origen a las

coordenadas (a, b) ; y θ es el argumento, que es el ángulo que forman estas coordenadas con el eje de abscisas. De esta manera, en forma polar tenemos que $z = r_\theta$. La figura II.14 lo muestra gráficamente.

Figura II.14: Número complejo en forma polar



A partir de un número complejo en forma binómica $z = a + bi$, su expresión en forma polar se obtiene aplicando las siguientes fórmulas:

$$r = \sqrt{a^2 + b^2}$$

$$\theta = \arctan \frac{a}{b} \quad \text{si } a > 0$$

$$\theta = \arctan \frac{a}{b} + \pi \quad \text{si } a < 0, b > 0$$

$$\theta = \arctan \frac{a}{b} - \pi \quad \text{si } a < 0, b < 0$$

En todos los casos, obtendremos un ángulo θ dentro del intervalo $[-\pi, \pi]$. En R, introduciremos números complejos en forma polar de la siguiente manera:

```
> (z5<-complex(modulus=sqrt(2),argument=pi/4))
[1] 1+1i
```

En este caso, vemos que $a = b = 1$, ya que $\sqrt{1^2 + 1^2} = \sqrt{2}$ y $\tan(\pi/4) = 1$.

Por último, veremos cómo calcular la raíz n -ésima de cualquier número. Considerando la ecuación de Euler, $z^n = re^{\theta i}$ tendrá n soluciones si consideramos también números complejos:

$$z = \sqrt[n]{r} e^{\left(\frac{\theta}{n} + \frac{2\pi k}{n}\right)i}, \quad k = 0, 1, \dots, n-1.$$

Sabiendo que $e^{\pi i} = \cos(\pi) + i \sin(\pi)$, podemos programar una función que calcule las n raíces complejas de cualquier número real. A la función la llamaremos **raiz.comp** y tendrá dos argumentos: el número del que queremos la raíz (r) y el orden de la raíz (n). La solución será un vector de números complejos con las n raíces: $(z_1, \dots, z_n)$:

```
> raiz.comp <- function(r,n){
+   sol <- array(data=NA, dim=n)
+   m1 <- (abs(r)^(1/n))
+   for (j in 0:n-1){
+     if (r<0){
+       m2 <- (pi/n)+2*j*(pi/n)
+     }else{
+       m2 <- 2*j*(pi/n)
+     }
+     sol[j+1] <- m1*complex(real=cos(m2),
+                           imaginary=sin(m2))
+   }
+   return(sol)
+ }
```

Veamos un par de ejemplos. Primero comprobaremos que $\sqrt{9}$ tiene las raíces -3 y 3 :

```
> raiz.comp(9,2)
[1] 3+0i -3+0i
```

Veamos ahora las soluciones de $\sqrt[3]{90}$:

```
> raiz.comp(90,3)
[1] 4.481405+0.00000i -2.240702+3.88101i
    -2.240702-3.88101i
```

Como hemos visto en este caso, R es un programa muy flexible que nos permite implementar cualquier tipo de función o algoritmo.

Bibliografía y librerías adicionales

Aunque la mayoría de manuales de R se centran en aspectos relacionados con la estadística —modelos, simulación, distribuciones e inferencia, etc.—, también se pueden encontrar referencias que tratan aspectos más cuantitativos y analíticos. El completo manual *Using R for Numerical Analysis in Science and Engineering*, de V. A. Bloomfield, está dedicado a ofrecer una alternativa al análisis numérico que se ha venido haciendo con Matlab en los últimos años: funciones, ecuaciones algebraicas, optimización y ecuaciones diferenciales, entre otros aspectos. Otro manual muy recomendable es *Introduction to scientific programming and simulation using R*, de O. Jones, R. Maillerdet y A. Robinson, en el cual se cubren ampliamente aspectos de programación en R y técnicas numéricas, como la integración numérica y la optimización. H. D. Vinod nos ofrece *Hands-on Matrix Algebra Using R: Active and Motivated Learning with Applications*, un manual sobre álgebra matricial avanzada en R, que combina elementos de teoría con aplicaciones empíricas. Por último, el manual *Solving Differential Equations in R*, de K. Soetaert, J. Cash

y F. Mazzia, constituye una completa revisión del análisis de ecuaciones diferenciales de varios tipos en R.

Aunque las librerías disponibles sobre aspectos relacionados con las matemáticas son numerosísimas, he aquí algunas de las recomendadas:

- *matrix*: proporciona clases y métodos de matrices y operaciones con estas, como por ejemplo la descomposición de Cholesky y de Schur.
- *MASS*: proporciona la inversa generalizada y espacios nulos de matrices, entre otras funcionalidades.
- *expm*: calcula el exponencial, logaritmo y raíz cuadrada de matrices cuadradas, además de potencias de matrices o la derivada de Frechet.
- *svd*: proporciona implementaciones de la descomposición en valores singulares (SVD) y cálculos de valores y vectores propios.
- *polynom*: permite la manipulación de polinomios univariantes —evaluación, diferenciación, integración, resolución de polinomios mediante la búsqueda de raíces, etc.
- *numDeriv*: establece el estándar para la diferenciación numérica en R, proporcionando gradientes numéricos, jacobianos y hessianos.
- *rSymPy*: permite acceder al sistema simbólico del álgebra *SymPy* —escrito en Python. Es compatible con los cálculos de precisión arbitraria, álgebra lineal y cálculo, resolución de ecuaciones y matemáticas discretas, entre otras aplicaciones.

- *Ryacas*: esta librería interconecta el sistema de álgebra computacional *yacas* con R. Es compatible con los cálculos de precisión simbólicos y arbitrarios en el cálculo y álgebra lineal.
- *R.matlab*: proporciona herramientas para leer y escribir archivos de Matlab.
- *optimx*: ayuda a especificar adecuadamente el problema de optimización no lineal, incluyendo la función objetivo, la función gradiente y el escalado.
- *ROI*: proporciona un marco para el manejo de problemas de optimización en R. Utiliza un enfoque orientado a objetos para definir y resolver diversas tareas de optimización —lineal, cuadrática y no lineal.
- *deSolve*: este es un paquete para resolver ecuaciones diferenciales ordinarias (EDO), ecuaciones diferenciales algebraicas (EDA) y ecuaciones diferenciales parciales (EDP).

Además, R dispone de varias librerías específicas relacionadas con la física. He aquí algunas de las principales:

- *fmri*: contiene imágenes funciones para analizar datos de imágenes de resonancia magnética funcional (fMRI) usando técnicas de *smoothing*.
- *diffractionmetry*: esta librería contiene herramientas para el análisis de difractogramas de rayos X.
- *moonSun*: proporciona funciones para cálculos astronómicos básicos.
- *cosmoFns*: contiene expresiones estándar para distancias, tiempos, luminosidades y otras cantidades útiles en la cosmología observacional.

- *solaR*: proporciona funciones para simular y modelar sistemas implicados en la captación y utilización de energía solar, incluyendo la energía fotovoltaica. Además, proporciona funciones para determinar la posición del Sol respecto a la Tierra y la radiación solar incidente.

Capítulo III

Análisis de datos y estadística descriptiva

1. Motivación

Esta sección tiene como objetivo proporcionar algunos ejemplos sobre cómo analizar datos en R. Debido a que la extensión de este manual es limitada, solo podemos cubrir algunas pinceladas de las numerosísimas posibilidades que este programa ofrece al respecto. Comenzando por la importación de datos desde diferentes formatos, se cubren aspectos como la creación y manipulación de variables, matrices y bases de datos, el cálculo de diferentes estadísticos, la visualización de los datos y, por último, su representación gráfica de diferentes formas.

2. Importación de datos

A la hora de realizar un análisis estadístico, es básico saber importar y manejar los datos correctamente. Por eso, antes de empezar, es fundamental disponer de la información y de los archivos bien organizados en una carpeta. Esto nos facilitará el trabajo a la hora de importar y exportar datos provenientes del análisis estadístico. Así pues, empezaremos indicando cuál es nuestro directorio de trabajo, que es donde tenemos los datos y donde se irá guardando todo lo que produzcamos. La

función que utilizaremos será `setwd` (*set working directory*). Si nuestro directorio de trabajo se encuentra en `C:/directorío`, simplemente hemos de introducir la siguiente instrucción, sin olvidarnos de introducir el directorio entre comillas:

```
> setwd("C:/Directorio")
```

En esta sección trabajaremos con datos del Instituto de Estadística de Catalunya (Idescat). Disponemos, para el año 2009 y para 941 municipios catalanes, de las siguientes variables:

- *mun*: código de cada municipio.
- *pob*: población total del municipio.
- *tra*: población activa del municipio.
- *mig*: población inmigrante del municipio.
- *edad*: media de edad de la población en el municipio.
- *costa*: variable cualitativa dicotómica (*sí/no*), que indica si el municipio se encuentra en la costa o no.

Se pueden importar los datos desde diferentes formatos y uno de los más comunes es el de una hoja de cálculo con extensión `.xls` o `.xlsx`. Para importar datos de documentos con estas extensiones, necesitaremos cargar primero la librería `xlsx` y después usar la función `read.xlsx`, como se muestra a continuación.

```
> library(xlsx)
> base.datos <- read.xlsx("Datos.xlsx", 1)
```

Donde `base.datos` es el objeto de la clase *data frame* donde almacenamos los datos y el número 1 se refiere al número de hoja en el documento *Datos.xlsx* donde se encuentran los datos que se importan.

Si los datos hubieran estado en un archivo de texto con formato *.txt*, hubiéramos utilizado la función `read.table`, `read.delim` o `read.delim2` para importarlos, y si fuera un documento separado por comas *.csv* (del inglés *comma separated values*), hubiésemos usado la función `read.csv`. Todas estas funciones nos permiten especificar si la primera fila de las columnas con las variables contiene o no el nombre (`header=TRUE` o `header=FALSE`). Además, si en los valores numéricos los decimales fueran puntos en lugar de comas, deberíamos especificarlo mediante el comando `dec="."`.¹

3. Visualización y manejo de datos

Una manera de comprobar que los datos se han importado correctamente es mediante la función `head`, la cual muestra en consola las seis primeras observaciones en filas, con las variables en columnas.

```
> head(base.datos)
mun      pob   tra  mig  edad  costa
1  M1 11521 5712 1152 36.77    No
2  M2   257   17   12 43.54    No
3  M3  9397 1456  874 39.91    No
4  M4   311   27   23 41.23    No
5  M5  7949 1843  589 36.99    No
6  M6 14627 2487 1719 41.84    Si
```

¹Por último, la librería *Hmisc* incorpora la función `spss.get` para importar datos de SPSS y `sasxport.get` para importar datos de SAS. Por su parte, la librería *foreign* nos permite importar datos de Stata mediante la función `read.dta`.

Visualizar bases de datos

Las funciones `head` y `tail` (cabeza y cola) nos permiten visualizar el principio y el final de un conjunto de datos, respectivamente, y se suelen utilizar para comprobar que los datos han sido importados o introducidos correctamente.

La función `dim` se aplica tanto a matrices como a bases de datos y nos indica la dimensión del objeto. En este caso, comprobamos que tenemos seis variables, cada una con 941 observaciones.

```
> dim(base.datos)
[1] 941    6
```

La función `dim`

Esta función, aplicada a una base de datos, nos muestra, en primer lugar, el número de filas (observaciones) y, en segundo lugar, el número de columnas (variable).

Siempre es recomendable empezar un análisis con una visión general de las variables. Para ello, existen diferentes funciones entre las cuales se encuentra `summary`. Recordemos que esta función muestra, para cada variable, el valor mínimo, máximo, la media y los tres cuartiles.

```
> summary(base.datos)
mun          pob
M1           : 1   Min.      :    29
M10          : 1   1st Qu.:   334
M100         : 1   Median :   971
M101         : 1   Mean   :  7944
M102         : 1   3rd Qu.:  3726
M103         : 1   Max.    :1621537
```

```
(Other):935
```

```
tra          mig
Min.      :    0   Min.      :    0
1st Qu.:   39   1st Qu.:   23
Median :  161   Median :   94
Mean     : 2607   Mean     : 1264
3rd Qu.:  848   3rd Qu.:  384
Max.     :880584   Max.     :284385
```

```
edad          costa
Min.      :33.63   No:871
1st Qu.:39.37   Si: 70
Median   :42.22
Mean     :42.79
3rd Qu.:45.87
Max.     :60.33
```

La función `subset`

Esta es una función fundamental en el análisis de datos, ya que permite crear bases de datos menores a partir de una base de datos original mediante la introducción de condiciones.

En muchas ocasiones, para agilizar la explotación de los datos que queremos analizar, nos encontramos con la necesidad de crear bases de datos más reducidas a partir de la base de datos original. En R esto se puede realizar mediante la función `subset`. Empecemos con un primer ejemplo muy sencillo: extraeremos de la base de datos inicial solo tres variables —el código del municipio, la población y la edad— y solo aquellas observaciones que cumplan la condición `edad>58`. Así pues, la base de datos resultante contiene solo dos variables y dos observaciones que cumplen esta condición.

```
> subset(base.datos, edad>58,
+        select=c(mun, pob, edad))
```

	mun	pob	edad
819	M819	43	60.32558
916	M916	119	58.02521

Seguiremos con otro ejemplo un poco más complejo. En este caso solo nos interesan las observaciones que cumplan dos condiciones: media de edad menor de cuarenta años y población menor de doscientos habitantes. Vemos que la base de datos resultante solo contiene cuatro municipios.

```
> subset(base.datos, edad<40 & pob<200,
+       select=c(mun,pob,edad))
      mun pob   edad
253 M253 163 39.71779
311 M311 152 38.71053
451 M451 141 38.64539
598 M598 152 37.86184
```

La función order

Esta función sirve para ordenar vectores de forma ascendente. Si se le coloca al vector el signo negativo delante, el resultado es el orden descendente.

El siguiente paso es ver cómo podemos reordenar las observaciones de una base de datos según el orden creciente o decreciente de una variable. En este caso, el vector `order(edad)` será un vector que contiene los valores 1 : 941 según el orden creciente de la variable edad. Si ordenamos la base de datos según este vector, y observamos sus últimos seis valores (`tail`), vemos los municipios más envejecidos de la muestra:

```
> tail(base.datos[order(edad),])
      mun pob tra mig      edad costa
585 M585 109  32   3 54.94495     No
858 M858 159  22  12 55.11950     No
621 M621 138   3   2 56.65217     No
571 M571  63   7   3 56.87302     No
916 M916 119  15  20 58.02521     No
819 M819  43   5   0 60.32558     No
```

Para hacer referencia a una variable de una base de datos, la sintaxis adecuada será `base.datos$variable`. Así pues, si deseamos aplicar la función `mean` (media aritmética) a la variable población `pob`, deberemos proceder de la siguiente manera.

```
> mean(base.datos$pob)
[1] 7944.123
```

El símbolo \$

Recordemos que este símbolo se utiliza para acceder a los distintos componentes de un objeto. En este caso, se usa para acceder a las diferentes variables que componen una base de datos.

No obstante, puede resultar engorroso tener que escribir `base.datos$` delante de cada variable a la hora de programar un análisis. Por ello, es recomendable usar la función `attach`. De esta manera, se podrá acceder a las variables simplemente dando sus nombres, sin hacer referencia a la base de datos.

```
> attach(base.datos)
```

La función `attach`

Esta función, aplicada a una base de datos, incrusta todas las variables de esta en la memoria en la forma de objetos (vectores) independientes, de manera que para referirnos a estas variables no haga falta usar repetidamente el símbolo `$`.

Así, el cálculo de la media aritmética de la población se simplifica:

```
> mean(pob)
[1] 7944.123
```

Seguidamente vamos a ocuparnos de la creación de nuevas variables a partir de la base de datos importada. Concretamente, vamos a calcular la tasa de actividad y la tasa de inmigración para cada municipio, según las siguientes fórmulas:²

$$\text{Tasa de actividad} = 100 \cdot \frac{\text{Población activa}}{\text{Población total}}$$

$$\text{Tasa de inmigración} = 100 \cdot \frac{\text{Población inmigrante}}{\text{Población total}}$$

En R, la creación de estas variables es inmediata. Fijémonos que, al estar las operaciones vectorizadas, las siguientes multiplicaciones y divisiones se aplican a cada observación de las variables.³

²Técnicamente, la tasa de actividad se calcula sobre la población mayor de dieciséis años, pero vamos a obviar este hecho debido a la naturaleza ilustrativa de este ejemplo.

³Si quisiéramos estos indicadores en tanto por uno, bastaría con no multiplicar por 100.

```
> t.act<-100*tra/pob  
> t.mig<-100*mig/pob
```

Supongamos que deseamos calcular estas tasas a nivel agregado de toda Cataluña, no a nivel municipal. Para ello, deberíamos sumar primero todas las variables y luego calcular las fórmulas previamente descritas.

```
> 100*sum(tra)/sum(pob)  
[1] 32.81848  
> 100*sum(mig)/sum(pob)  
[1] 15.90919
```

La función `sum`

Recordemos que la función `sum` aplica la operación suma a todos los componentes de un objeto, en este caso suma todas las observaciones de cada variable.

4. Cálculo de estadísticos

R ofrece muchas posibilidades para hacer un análisis descriptivo de datos. Además de disponer de muchas librerías con estadísticos y procedimientos, su flexibilidad permite que podamos programar a medida nuestro propio análisis descriptivo. Para ilustrarlo, vamos a crear una matriz de indicadores (`indic`) con tres variables de interés: la tasa de actividad, la tasa de inmigración y la media de la edad. Además, vamos a nombrar cada una de las observaciones (filas) según el código del municipio `mun`.

```
> indic<-cbind(t.act,t.mig,edad)
> row.names(indic)<-mun
```

Las funciones `cbind` y `rbind`

Estas funciones nos permiten unir en matrices objetos por columnas y por filas, respectivamente.

Al igual que antes, aplicando la función `summary` obtenemos la media, el mínimo, el máximo y los tres cuartiles de las columnas de esta matriz.

```
> summary(indic)
t.act      t.mig      edad
Min.   : 0.000   Min.   : 0.000   Min.   :33.63
1st Qu.: 8.878   1st Qu.: 5.007   1st Qu.:39.37
Median :15.494   Median : 9.127   Median :42.22
Mean   :21.213   Mean   :10.442   Mean   :42.79
3rd Qu.:25.506   3rd Qu.:14.238   3rd Qu.:45.87
Max.   :266.138   Max.   :50.896   Max.   :60.33
```

Una función muy útil a la hora de realizar un cálculo para un conjunto de variables es `apply`. En este caso, aplicaremos a la matriz de variables `indic`, y solo para la dimensión 2 (columnas), la función `mean` (media aritmética). El resultado será un vector con las medias aritméticas de las tres variables de interés.

```
> apply(indic,2,mean)
t.act      t.mig      edad
21.21259 10.44170 42.79377
```

Un cálculo un poco más complejo, pero potencialmente útil, es el de crear una función propia para obtener un conjunto de

estadísticos de interés para un grupo de variables. En nuestro caso, llamaremos a esta función `estad.basic`, y consistirá en la media, la varianza, la desviación estándar, el mínimo, los tres cuartiles y el máximo —estos últimos calculados con la función `quantile`). Mediante la función `colnames` asignaremos un nombre a cada estadístico y el resultado final lo redondearemos a dos decimales mediante la función `round`.

```
> estad.basic <- function(x){
+   est <- cbind(mean(x), var(x), sd(x),
+   t(quantile(x)))
+   colnames(est) <- c("media", "var",
+   "desv.est", "min", "Q1", "Q2", "Q3", "max")
+   return(round(est, 2))
+ }
```

Esta pieza de código puede parecer engorrosa, pero tiene una ventaja fundamental: solo hay que escribirla una vez. De hecho, la podemos guardar en un archivo de texto con el mismo nombre y almacenarla en una librería de funciones propias. Así, cuando nos haga falta, solo tendremos que cargar esta función mediante la siguiente instrucción:⁴

```
> source("estad.basic.txt")
```

Si aplicamos esta función a la variable *tasa de actividad* (`t.act`), obtenemos el siguiente resultado:

```
> estad.basic(t.act)
      media      var desv.est  min    Q1     Q2     Q3     max
[1,]  21.21  484.02      22     0  8.88  15.49  25.51  266.14
```

⁴En el caso de que este archivo no esté en el directorio de trabajo, tendremos que especificar toda la ruta entre comillas, de igual manera que hicimos con la función `setwd`.

Si combinamos las funciones `apply` y `estad.basic`, podremos obtener con un solo cálculo los estadísticos seleccionados aplicados a todas las variables simultáneamente y quedarán los resultados registrados en una matriz de nombre `est.total`.

```
> est.total <- apply(indic,2,estad.basic)
> rownames(est.total)<- c("media","var",
+ "desv.est","min","Q1","Q2","Q3","max")
> print(est.total)
```

	t.act	t.mig	edad
media	21.21	10.44	42.79
var	484.02	54.49	18.65
desv.est	22.00	7.38	4.32
min	0.00	0.00	33.63
Q1	8.88	5.01	39.37
Q2	15.49	9.13	42.22
Q3	25.51	14.24	45.87
max	266.14	50.90	60.33

Variables discretas, dicotómicas y binarias

Una variable discreta puede tomar un número limitado de valores enteros y una variable dicotómica o binaria solo toma dos valores. Es decir, una variable dicotómica es una variable discreta, pero lo contrario no tiene por qué ser cierto.

Uno de los puntos fuertes de R es el tratamiento de los factores, es decir, variables cualitativas que categorizan las observaciones. En nuestro ejemplo, la variable `costa` es una variable no numérica que toma los valores *sí* o *no* dependiendo de si el municipio está o no en la costa. La función `class` nos indica la clase de objeto que es la variable `costa`, mientras

que la función `levels` nos devuelve los valores que esta puede tomar.

```
> class(costa)
[1] "factor"
> levels(costa)
[1] "No" "Si"
```

Los tipos de análisis que se pueden hacer con factores son múltiples. Uno de los más inmediatos es por medio de la función `tapply`. En el siguiente ejemplo, aplicamos la función media aritmética (`mean`) a la variable `t.mig` según las categorías del factor `costa`. Como vemos, la tasa de inmigración es mucho más alta en la costa que en el interior.

```
> tapply(t.mig, costa, mean)
No      Si
9.691  19.770
```

A nivel multivariante, la función `cov` devuelve la matriz de covarianzas, mientras que la función `cor` corresponde a la matriz de correlaciones. En el siguiente ejemplo, combinamos la función `cor` con la función `print`, la cual nos permite limitar el nombre de dígitos significativos.

```
> print(cor(indic), digits=2)
t.act   t.mig   edad
t.act   1.0000   0.0072 -0.25
t.mig   0.0072   1.0000 -0.28
edad   -0.2518  -0.2752   1.00
```

La función `digits`

Esta opción nos permite ajustar el número de decimales del resultado obtenido. Específicamente, nos permite establecer el número mínimo de dígitos significativos para todos los valores mostrados.

5. Representación gráfica

Las posibilidades gráficas que ofrece R son enormes y en esta sección veremos solo una pequeña parte de todo ese potencial. Una descripción de los tipos de gráficos básicos que ofrece R se encuentra en la tabla III.1.

Tabla III.1: Gráficos básicos con R

Descripción	Instrucción
Gráfico de dispersión con puntos	<code>plot(x,type="p")</code>
Gráfico con línea	<code>plot(x,type="l")</code>
Gráfico con líneas y puntos superpuestos	<code>plot(x,type="o")</code>
Gráfico con líneas y puntos discontinuos	<code>plot(x,type="b")</code>
Gráfico de escalera	<code>plot(x,type="s")</code>
Histograma	<code>hist(x)</code>
Densidad estimada	<code>plot(density(x))</code>
Gráfico de puntos	<code>dotchart(x)</code>
Gráfico de barras	<code>barplot(x)</code>
Gráfico circular (de pastel)	<code>pie(x)</code>
Diagrama de caja	<code>boxplot(x)</code>
Diagrama de dispersión	<code>plot(x)</code>

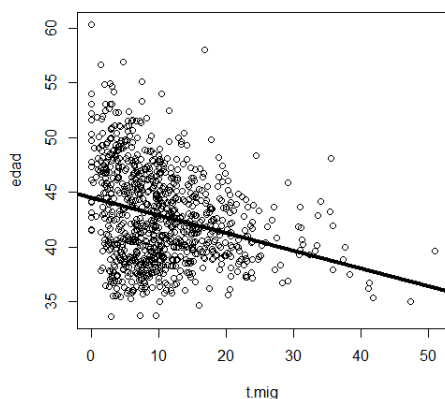
Diagrama de dispersión

Este tipo de gráfico es muy útil a la hora de realizar una exploración previa de los datos, es decir, un análisis descriptivo, ya que proporciona información visual sobre cómo se comportan las variables y cómo están relacionadas entre ellas.

Empezaremos con un diagrama de dispersión, que es un gráfico bidimensional que utiliza las coordenadas cartesianas para mostrar los valores de dos variables, una en cada eje. Si deseamos este diagrama para las variables *tasa de inmigración* y *media de edad*, bastará con introducirlas como argumentos de la función `plot`. Además, opcionalmente podemos superponer a este gráfico más elementos, por ejemplo rectas mediante la función `abline`. En nuestro ejemplo, añadiremos una recta ajustada a los puntos, para lo cual usaremos la función `lm` (*linear model*). Dibujaremos esta recta con un grosor `lwd=4`. El resultado lo podemos ver en la figura III.1.

```
> plot(t.mig,edad)
> abline(lm(edad~t.mig),lwd=4)
```

Figura III.1: Diagrama de dispersión (I)



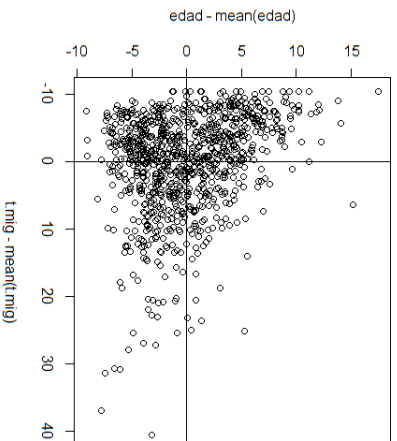
La opción `abline`

Esta opción nos permite añadir todo tipo de líneas a gráficos ya creados, a modo de capas superpuestas.

Otro ejemplo de diagrama de dispersión es el de las variables centradas en el valor cero, para lo cual hay que aplicar la transformación $x_i - \bar{x}$, es decir, restar la media aritmética. En este caso, restamos la media aritmética de la variable, aunque también podríamos haber utilizado la función `scale`, usada para escalar variables. Adicionalmente, se puede añadir una línea horizontal $h=0$ y vertical $v=0$ en el valor cero. En este caso, veremos el gráfico de dispersión entre la tasa de inmigración y la media de edad, en la figura III.2.

```
> plot(t.mig~mean(t.mig), edad~mean(edad))  
> abline(h=0)  
> abline(v=0)
```

Figura III.2: Diagrama de dispersión (II)



La función `scale`

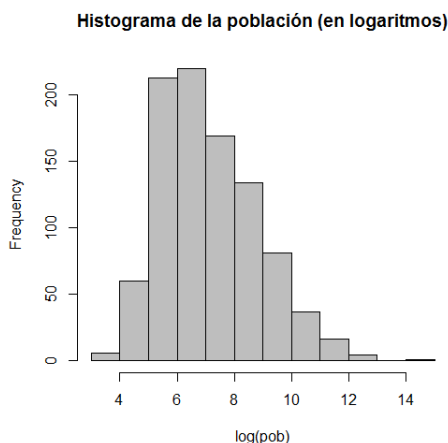
Esta función se usa para estandarizar —también llamado tipificar— variables, centrando y escalando las columnas de

una matriz numérica. Esto es, para cada valor x , se aplica la transformación $(x - \bar{x})/\hat{\sigma}_x$, siendo los valores $\bar{x}$ y $\hat{\sigma}_x$ la media aritmética y la desviación típica de cada columna, respectivamente. Opcionalmente, se pueden introducir valores predeterminados de estos estadísticos mediante las opciones **center** y **scale**. Además, si una de estas opciones es declarada **FALSE**, no se aplicará.

El siguiente ejemplo que veremos es el histograma, el cual se emplea para visualizar la distribución de una variable. El histograma se representa mediante la función **hist** de la siguiente manera, tal y como vemos en la figura III.3.

```
> hist(log(pob),main="Histograma de la población  
+ (en logaritmos)",col="grey")
```

Figura III.3: Histograma



La opción main

Esta opción permite introducir un título general a un gráfico.

Histograma y diagrama de barras

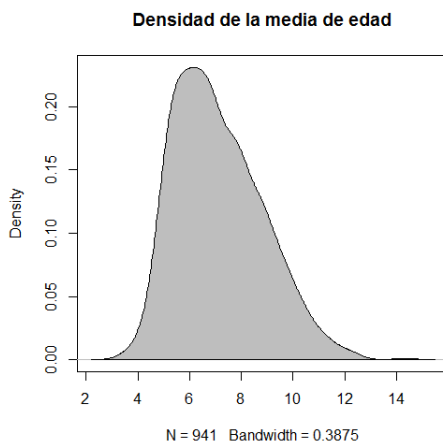
Aunque se parecen, estos dos tipos de gráfico no son idénticos. El histograma se emplea para representar datos cuantitativos continuos, mientras que el diagrama de barras se usa para representar gráficamente datos cuantitativos discretos o datos cualitativos.

La función de densidad, calculada con técnicas no paramétricas, es la generalización del histograma asumiendo que el grosor de las barras tiende a cero. Visto de otra manera, el histograma es discreto y la función de densidad continua. El primer paso será estimar la función de densidad mediante la función `density`, cuyo resultado será un conjunto de valores. Para representarlos gráficamente, usaremos la función `plot`. Opcionalmente, además, podremos colorear esta función —tanto el borde como el interior— mediante la función `polygon`. Es importante ver cómo R elabora gráficos complejos añadiendo capas a partir de una instrucción inicial. El resultado lo vemos en la figura III.4.

```
> den.edad<-density(edad)
> plot(den.pob,main="Densidad de la media de edad
")
> polygon(den.pob,col="grey",border="black")
```

Las opciones col y lwd

Estas opciones son muy útiles a la hora de personalizar los gráficos, ya que permiten definir el color y el grosor de los puntos y líneas representadas.

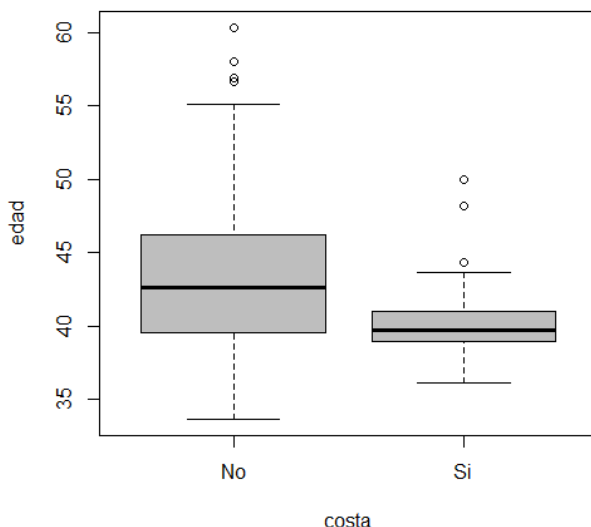
Figura III.4: Función de densidad

Por último, veremos un tipo de gráfico llamado diagrama de caja (*boxplot*, en inglés). El diagrama de caja consiste en un gráfico basado en cuartiles, mediante el cual se puede visualizar la simetría de la distribución de los datos. La función de R que produce este tipo de gráfico es `boxplot`. Mediante esta función, vamos a ver si existen diferencias en la distribución de la variable media de edad según el factor *costa*. El resultado se muestra en la figura III.5, donde podemos ver que las poblaciones en la costa son más jóvenes que las de interior.

```
> boxplot(edad~costa,col="grey")
```

Diagrama de caja

El diagrama de caja proporciona información sobre los valores mínimo y máximo, los cuartiles y posibles valores atípicos, además de la simetría de la distribución de los datos.

Figura III.5: Diagrama de caja

Guardar y exportar gráficos

En la consola original de R, una vez seleccionada la ventana del gráfico, si vamos al menú *Archivo*, se desplegará un menú con opciones para guardar el gráfico en un documento. Veremos que podemos elegir entre varios formatos —EPS, JPG, PDF, etc.

Bibliografía y librerías adicionales

Existen muchos libros y manuales de estadística donde se cubren muchos temas de estadística descriptiva y análisis gráfico. Al incluir estos manuales otros aspectos que van más allá la estadística descriptiva, se introducirán y comentarán en capítulos posteriores. En todo caso, podemos comentar

dos manuales muy generales. Por un lado, *Statistics and Data with R: An applied approach through examples*, de Y. Cohen y J. Y. Cohen, es un amplio y extenso manual que cubre múltiples aspectos de la estadística con R, que incluye temas como análisis exploratorio, manejo de datos y variables y distribuciones de probabilidad. Por otro lado, *Introductory statistics with R*, P. Dalgaard, es un libro dedicado tanto a estudiantes como a investigadores que proporciona una cobertura amplia de métodos estadísticos. Otra obra muy introductoria es *R by Example*, de J. Albert y M. Rizzo, la cual cubre el análisis de datos cuantitativos y categóricos, el análisis exploratorio de datos, la presentación de gráficos, los modelos de inferencia y de regresión, además de algunos temas avanzados de estadística —simulación y estadística bayesiana.

En cuanto al análisis gráfico, existen algunos manuales dedicados íntegramente a estos métodos. Un manual de contenido avanzado es *Graphical models with R*, de S. Højsgaard, D. Edwards y S. Lauritzen, el cual se dedica al análisis gráfico de modelos log-lineales, redes bayesianas y modelos estocásticos. Además, *Interactive and dynamic graphics for data analysis: with R and GGobi*, de D. Cook y D. F. Swayne, trata la generación de gráficos para el análisis clúster y la gestión de los valores ausentes (*missing values*) en la generación de gráficos.

Por otro lado, las principales librerías relacionadas con la estadística descriptiva son las siguientes:

- *Hmisc*: contiene muchas funciones útiles para el análisis de datos, gráficos de alto nivel, funciones para calcular el tamaño y la potencia de la muestra, importar y anotar conjuntos de datos, imputar *missing values*, etc.
- *moments*: proporciona funciones para el cálculo de los momentos de una distribución, así como su asimetría y su curtosis.

- *prettyR*: funciones para dar formato a estadísticos descriptivos, bases de datos y convertir resultados a HTML.
- *compareGroups*: paquete diseñado para leer, interpretar, sintetizar, visualizar y analizar los datos epidemiológicos.
- *descr*: contiene funciones para la descripción de datos categóricos ponderados y funciones para la codificación de objetos en caracteres.
- *DescTools*: la intención del paquete es ofrecer una caja de herramientas, lo que facilita las tareas descriptivas en el análisis de datos. Contiene funciones básicas —estadística descriptiva, resúmenes gráficos e informes de los resultados— y además facilita la descripción de datos de manera eficiente, haciendo informes mediante MS Word, Excel y PowerPoint.
- *ds*: proporciona funciones de análisis de estadísticos descriptivos y correlaciones.

En cuanto a las librerías relacionadas con el análisis gráfico, se destacan las siguientes:

- *lattice*: es un potente y elegante sistema de visualización de datos de alto nivel, y que hace énfasis en datos multivariantes. Un manual dedicado íntegramente a esta librería es *Lattice: multivariate data visualization with R*, de D. Sarkar.
- *ggplot2*: es una potente y popular librería que proporciona una sintaxis alternativa para elaborar gráficos en R. Combina las ventajas del sistema gráfico base y la elegancia gráfica: acondicionamiento y ejes compartidos se gestionan automáticamente, y las múltiples fuentes

de datos se pueden incorporar paso a paso. Además, también implementa un sistema de acondicionamiento multidimensional sofisticado y una interfaz consistente para asignar datos a los atributos estéticos. Un completo manual con múltiples ejemplos es *Ggplot2. Elegant Graphics for Data Analysis*, de H. Wickham.

- *scatterplot3d*: proporciona herramientas para representar gráficamente en tres dimensiones.
- *colorspace*: librería que incluye una amplia paleta de colores.
- *vcd*: incluye técnicas de visualización y procedimientos de inferencia dirigido especialmente a datos categóricos.
- *RColorBrewer*: proporciona una paleta de colores para el sombreado de mapas según el valor de una variable.
- *rgl*: proporciona funciones de alto nivel para gráficos de tres dimensiones interactivos.

Capítulo IV

Distribuciones de probabilidad

1. Motivación

En este capítulo se aborda todo lo relativo a las variables aleatorias y sus distribuciones de probabilidad. En un primer apartado se aprenderá a trabajar con variables discretas, sus distribuciones asociadas y el cálculo de sus probabilidades. Posteriormente, se hará lo mismo con variables continuas. De esta manera, se estudiarán las principales distribuciones de probabilidad tanto discretas como continuas que se utilizan en la mayoría de las asignaturas de estadística.

2. Variables aleatorias y distribuciones asociadas

Una parte fundamental de la estadística es el análisis de variables aleatorias, las cuales se pueden distribuir según diferentes funciones de probabilidad. R ofrece la posibilidad de analizar estas distribuciones de probabilidad de varios modos. De manera específica, para cada distribución de probabilidad encontramos cuatro opciones de cálculo:

- 1) *Cuantiles*: la función cuantil es la inversa de la función de distribución. Es decir, un cuantil será el valor x tal que

$P(X \leq x) = p$, siendo p la probabilidad, que como tal se encuentra en el intervalo $[0, 1]$.

- 2) *Probabilidades*: la probabilidad será aquel valor p tal que $P(X \leq x) = p$, es decir, se trata de la función inversa a la función cuantil.
- 3) *Gráficas*: cada distribución de probabilidad ofrece la posibilidad de calcular los gráficos de su función de densidad o de cuantía asociados, así como su función de distribución acumulada.
- 4) *Muestras aleatorias*: para cada distribución se pueden generar muestras de n valores aleatorios, dados unos parámetros iniciales determinados.

Variables aleatorias discretas y continuas

Conviene recordar, por una parte, que para variables continuas no tiene sentido calcular $P(X = x) = p$. Por lo tanto, para este tipo de variables es más correcto calcular $P(X < x) = p$ para obtener la probabilidad de la cola de la izquierda y $P(X > x) = p$, para la de la derecha. Además, la función de cuantía solo existe para las variables aleatorias discretas y la de densidad, para las continuas. Adicionalmente, el área bajo la curva de todas las funciones de distribución siempre será la unidad, ya que incluye toda la probabilidad acumulada de una variable aleatoria.

En lo que respecta al cálculo de cuantiles y probabilidades, hay que especificar que R ofrece la posibilidad de hacer cálculos para las dos colas —izquierda y derecha—, es decir, tanto $P(X \leq x) = p$ como $P(X > x) = p$. Es fundamental subrayar el símbolo $\leq$ para la cola izquierda y el símbolo $>$ para la cola

derecha, para evitar confusiones en el cálculo de probabilidades. Además, también es posible calcular mediante funciones en código R el valor de la función de densidad —para variables continuas— y de la función de cuantía —para variables discretas.

A continuación, ofrecemos una relación de las funciones en código R de las distribuciones de probabilidad incorporadas en la distribución básica de R. Por una parte, en la tabla IV.1, se incluyen las distribuciones de probabilidad discretas.

Tabla IV.1: Distribuciones de probabilidad discretas

Distribución	Función de cuantía	Probabilidades	Cuantiles	Muestra aleatoria
Binomial	dbinom	pbinom	qbinom	rbinom
Poisson	dpois	ppois	qpois	rpois
Geométrica	dgeom	pgeom	qgeom	rgeom
Hipergeométrica	dhyper	phyper	qhyper	rhyper
Binomial negativa	dnbinom	pnbinom	qnbinom	rnbinom

De igual manera, en la tabla IV.1 se ofrecen las distribuciones de probabilidad continuas y las funciones de R asociadas.

Tabla IV.2: Distribuciones de probabilidad continuas

Distribución	Función de densidad	Probabilidades	Cuantiles	Muestra aleatoria
Normal	dnorm	pnorm	qnorm	rnorm
t-Student	dt	pt	qt	rt
Chi-cuadrado	dchisq	pchisq	qchisq	rchisq
F-Snedecor	df	pf	qf	rf
Exponencial	dexp	pexp	qexp	rexp
Uniforme	dunif	punif	qunif	runif
Beta	dbeta	pbeta	qbeta	rbeta
Cauchy	dcauchy	pcauchy	qcauchy	rcauchy
Logística	dlogis	plogis	qlogis	rlogis
Log-normal	dlnorm	plnorm	qlnorm	rlnorm
Gamma	dgamma	pgamma	qgamma	rgamma
Weibull	dweibull	pweibull	qweibull	rweibull

A continuación, ofrecemos varios ejemplos prácticos de análisis de variables aleatorias con diferentes distribuciones.

3. Distribuciones de probabilidad discretas

3.1. Permutaciones, combinaciones y el cociente binomial

Antes de revisar las distribuciones discretas, es importante saber realizar permutaciones y combinaciones, ya que son la base, entre otras, de la distribución binomial. Además, será muy útil entender qué es y qué utilidades tiene el cociente binomial $\binom{n}{k}$, el cual da como resultado las combinaciones de n elementos tomados de k en k .

Empezamos con un ejemplo de permutaciones. Supongamos que disponemos de dos sillas (1 y 2) y tres personas (a , b y c). La pregunta es ¿cuántas maneras diferentes tienen estas personas de sentarse? Siendo tan pocos números, se puede hacer manualmente, esto es, $\{ab, ac, ba, bc, ca, cb\}$. El resultado es $3 \cdot 2 = 6$ permutaciones posibles, en las que la posición de cada individuo importa —esto es, no es lo mismo ab que ba . En general, si tenemos n elementos que hay que colocar en k posiciones, el número total de permutaciones será $(n - k + 1)!$. De una forma más elegante, la fórmula general de las permutaciones es la siguiente:

$$P(n, k) = \frac{n!}{(n - k!)}$$

En el ejemplo anterior, $P(3, 2) = 3!/1! = 6$. Supongamos ahora que tenemos $n = 7$ personas y $k = 3$ sillas. ¿Cuántas maneras diferentes hay de sentarse? Con R, podemos crear una función que se llame, supongamos, `perm` y evaluarla con los valores que deseemos. En este caso vemos que obtenemos 210 posibles permutaciones.

```
> perm<-function(n,k) factorial(n)/factorial(n-k)
> perm(7,3)
[1] 210
```

El concepto de combinaciones es similar al de permutaciones, pero con la excepción de que el orden no importa. Esto es, siguiendo con el ejemplo anterior, *ab* sería equivalente a *ba*. Hagamos el mismo cálculo y veamos cuántas combinaciones salen de $n = 3$ personas y $k = 2$ sillas. Para ello, usaremos la función de R `combn`:

```
> (personas <- letters[1:3])
[1] "a" "b" "c"
> n <- length(personas)
> k <- 2
> combn(personas,k)
      [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "b"  "c"  "c"
```

En este ejemplo, hemos usado la función `letters` para introducir las tres primeras letras del abecedario. Vemos que la función `combn` resulta en una matriz, donde las columnas son diferentes combinaciones de los elementos con una longitud de k . Si lo que nos interesa es el número de combinaciones total —es decir, el número de columnas de la matriz anterior—, la fórmula coincide con el cociente binomial:

$$C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Esto es, el cociente binomial define el número de subconjuntos de k elementos elegidos de un conjunto con n elementos. La función de R que calcula el cociente binomial es `choose`.

Veamos que el número de combinaciones en el ejemplo anterior es tres:

```
> choose(3,2)
[1] 3
```

3.2. Distribución binomial

La distribución de Bernoulli

Las variables aleatorias dicotómicas que toman únicamente el valor 1 cuando un ensayo o suceso se ha llevado a cabo con éxito y un valor 0 en caso contrario se distribuyen bajo una distribución de Bernoulli. Por tanto, la distribución binomial será una generalización de una distribución de Bernoulli.

La distribución binomial es una de las principales distribuciones de probabilidad discretas. De manera específica, esta mide el número de éxitos en una secuencia de n ensayos de Bernoulli independientes entre sí y con una probabilidad p de éxito entre los ensayos. Es decir, definimos formalmente una variable aleatoria X que se distribuye como una binomial con parámetros n y p :

$$X \sim B(n, p)$$

Además, los parámetros de tendencia central y de dispersión para variables binomiales son $E(X) = np$ y $V(X) = np(1 - p)$. La función de probabilidad tiene la forma siguiente:

$$P(X = x) = \binom{n}{x} p^x (1 - p)^{n-x}$$

Para ver cómo trabajamos con variables aleatorias distribuidas como una binomial, estudiaremos un ejemplo ficticio. Sabemos que la probabilidad que tiene un jugador de baloncesto determinado de encestar un triple es del 30 %, es decir, $p = 0,3$, y queremos calcular las probabilidades de que enceste 0, 1, 2, ... hasta n triples, siendo n el número de lanzamientos —ensayos. Suponiendo que este jugador lanza cinco veces ($n = 5$), usaremos la función `dbinom`, en la que introduciremos un vector de posibles resultados —desde cero aciertos hasta cinco aciertos—, la probabilidad de éxito (`prob=0.3`) y el número de ensayos (`size=5`). Guardaremos el resultado en el vector `ensayo`. Después, asignaremos un nombre a cada observación y, por último, visualizaremos este vector, que nos devuelve las probabilidades de que enceste 0, 1, 2, ... hasta cinco triples.

```
> ensayo <- dbinom(0:5, size=5, prob=0.3)
> rownames(ensayo) <- 0:5
> names(ensayo) <- 0:5
> print(ensayo)
0          1          2          3          4          5
0.16807 0.36015 0.30870 0.13230 0.02835 0.00243
```

¿Cómo se interpreta este resultado? Si definimos X como el número de éxitos, tenemos que $P(X = 0) = 0,168$. Esto no es más que la probabilidad de que en cinco ensayos nuestro jugador no acierte ningún triple. En el otro extremo, la probabilidad de que acierte cinco triples en cinco intentos es de $P(X = 5) = 0,002$. Resulta importante observar que la suma de estas cinco probabilidades es igual a uno.

Si quisiéramos realizar el cálculo de las probabilidades acumuladas —también denominadas de la cola—, tendríamos que usar la función `pnbinom`, con los mismos valores que en la función anterior, pero además especificando la cola de la distribu-

ción que nos interesa. Si queremos $P(X \leq x)$, especificaremos `lower.tail=TRUE` —la opción por defecto—:

```
> pbinom(0:5,size=5,prob=0.3,lower.tail=TRUE)
[1] 0.16807 0.52822 0.83692 0.96922 0.99757
    1.00000
```

En cambio, si queremos calcular $P(X > x)$, especificaremos `lower.tail=FALSE`:

```
> pbinom(0:5,size=5,prob=0.3,lower.tail=FALSE)
[1] 0.83193 0.47178 0.16308 0.03078 0.00243
    0.00000
```

Se puede observar que, si sumamos las probabilidades obtenidas por columnas, la suma de cada par de valores siempre es uno. Esto sucede porque, por definición, siempre se cumple la igualdad siguiente:

$$P(X \leq x) + P(X > x) = 1.$$

En el ejemplo anterior, ¿cuál sería la probabilidad de encestar más de tres triples? La respuesta es $P(X > 3) = 0,03078$. ¿Y la probabilidad de encestar menos de dos triples? La respuesta es $P(X \leq 1) = 0,52822$.

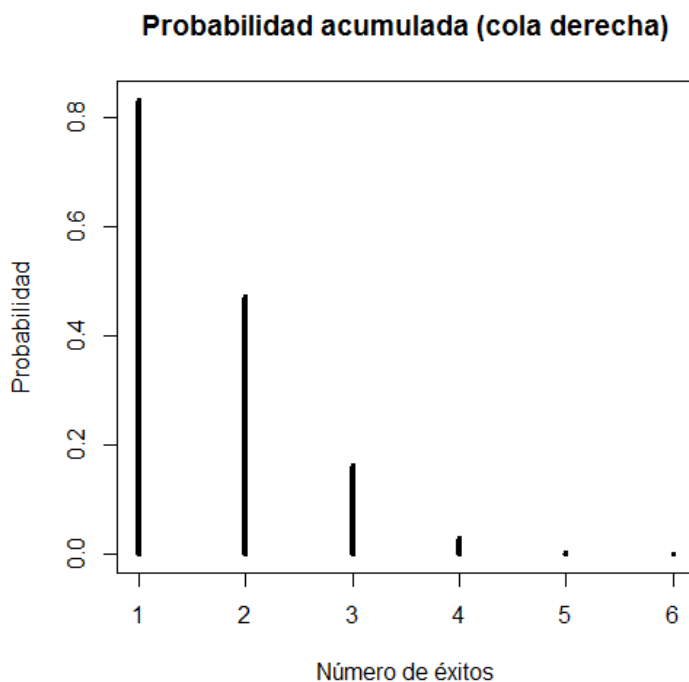
Mediante la función `plot()` podemos representar gráficamente cualquiera de estos resultados y personalizar el gráfico a nuestro gusto. Supongamos que deseamos representar visualmente las probabilidades acumuladas ($P(X \leq x)$) en un gráfico de barras. Una opción sería la siguiente:

```
> plot(pbinom(0:5,size=5,prob=0.3,
+ lower.tail=FALSE),type="h",
+ lwd=4,xlab="Número de éxitos",
```

```
+ ylab="Probabilidad",  
+ main="Probabilidad acumulada (cola derecha)")
```

El resultado se puede ver en la figura IV.1.

Figura IV.1: Probabilidad binomial acumulada



3.3. Distribución geométrica

La distribución geométrica está asociada a la distribución binomial que acabamos de ver. De manera específica, esta distribución describe la probabilidad del número x de ensayos de Bernoulli necesarios para obtener un éxito. Si la probabilidad de éxito en cada ensayo es p , entonces la probabilidad de que

x ensayos sean necesarios para obtener un éxito viene definida por la expresión siguiente:

$$P(X = x) = (1 - p)^{x-1}p.$$

Para $x = 1, 2, 3, \dots$ La secuencia de probabilidades obtenida se denomina una progresión geométrica. Así pues, podemos definir la función de distribución de la manera siguiente.

$$P(X \leq x) = F(x) = 1 - (1 - 0,3)^x.$$

Si retomamos el ejemplo anterior del jugador de baloncesto que intenta encestar triples, en este caso podemos definir una nueva variable aleatoria X , que será el número del intento en el que el jugador encesta el primer triple, es decir, el número de ensayos necesarios hasta encestar. Formalmente, decimos que X sigue una distribución geométrica con parámetro $p = 0,3$, que se corresponde con la probabilidad de encestar un triple, como vimos en el ejemplo anterior. Si queremos saber la probabilidad de que enceste el primer triple dentro de los seis primeros lanzamientos a canasta, haremos el cálculo siguiente.

$$P(Y \leq 6) = F(6) = 1 - (1 - 0,3)^6 = 0,8823.$$

Para hacer este cálculo en R usaremos la función `pgeom`. El primer argumento es el número de fallos antes del primer acierto, es decir, $x - 1$. En nuestro ejemplo, esto es $6 - 1 = 5$. El segundo argumento será la probabilidad (`prob=0.3`) y, por último, ya que estamos calculando $P(Y \leq x)$, especificaremos la opción de la cola izquierda (`lower.tail=TRUE`). Así, obtendremos el siguiente resultado:

```
> pgeom(5, prob=0.3, lower.tail=TRUE)
[1] 0.882351
```

3.4. Distribución hipergeométrica

Hasta ahora hemos considerado que las pruebas u observaciones hechas eran independientes, pero con frecuencia no se da esta condición y el uso del modelo binomial puede resultar equivocado cuando las poblaciones estudiadas son pequeñas. De manera específica, supongamos que tenemos una muestra de N bolas, de las cuales N_1 son verdes y N_2 son rojas, de modo que $N_1 + N_2 = N$. Ante una extracción de n bolas de este conjunto —sin retorno—, la variable X será el número de bolas verdes obtenidas. Decimos entonces que X sigue una distribución hipergeométrica tal que:

$$P(X = x) = \frac{\binom{N_1}{x} \binom{N_2}{n-x}}{\binom{N}{n}}.$$

Supongamos un ejemplo empírico. Un estudiante de oposiciones ha preparado doce de las dieciocho lecciones de las que consta un programa. El examen consiste en tres lecciones elegidas de manera aleatoria. ¿Qué probabilidad tiene el estudiante de conocer los tres temas? Sabemos que $N_1 = 12$, $N_2 = 6$ y $N = 18$. Además, la extracción es $n = 3$. En R, usaremos la función `dhyper`, especificando los posibles valores de la variable x (0, 1, 2, 3), el número de bolas blancas N_1 (`m=12`), el de bolas negras $N_2 = 6$ (`n=6`) y el número de extracciones (`k=3`).

```
> ensayo <- dhyper(0:3,m=12,n=6,k=3)
> names(ensayo) <- 0:3
> print(ensayo)
0          1          2          3
0.0245 0.2205 0.4852 0.2696
```

Como vemos, la probabilidad de que los tres temas que ha estudiado entren en el examen es del 26,9%. La proba-

bilidad más alta es $P(X = 2) = 0,485$, es decir, que haya estudiado dos de los tres temas. Si nos preguntamos, por ejemplo, cuál es la probabilidad de que sepa como mínimo dos temas ($P(X \geq 2)$), necesitamos la probabilidad acumulada. Concretamente, en este caso necesitamos la cola derecha (`lower.tail=FALSE`). Con esta opción, R calcula $P(X > x)$, de manera que tendremos que introducir en el cuadro de diálogo el valor $x = 1$ si queremos calcular $P(X > 1) = P(X \geq 2)$. Al trabajar con distribuciones discretas, es fundamental tener cuidado con estos detalles.

```
> phyper(1, m=12, n=6, k=3, lower.tail=FALSE)
[1] 0.754902
```

Es decir, hay una probabilidad del 75 % de que conozca dos o tres temas en el examen.

3.5. Distribución de Poisson

Otra de las distribuciones discretas importantes es la de Poisson, también derivada de la distribución binomial. Vamos a tomar el ejemplo de una variable aleatoria X , definida como el tiempo de espera del autobús en minutos, y que tiene como único parámetro λ , que representa tanto la media como la varianza de la variable aleatoria. Esto es, $E(X) = V(X) = \lambda$.

Supongamos que el tiempo medio de espera es de 3 minutos. Entonces $\lambda = 3$ minutos y, por tanto, $X \sim \text{Pois}(3)$ o, de manera alternativa, $X \sim P(3)$. Para saber la probabilidad de que el autobús tarde cinco minutos, el cálculo que hay que hacer es el siguiente:

$$P(X = 5) = \frac{\lambda^k}{k!} e^{-\lambda} = \frac{3^5}{5!} e^{-3} = 0,1008.$$

R nos facilita un resultado inmediato a este cálculo mediante la función `dpois()`, donde introduciremos los valores de $x = 5$ y $\lambda = 3$:

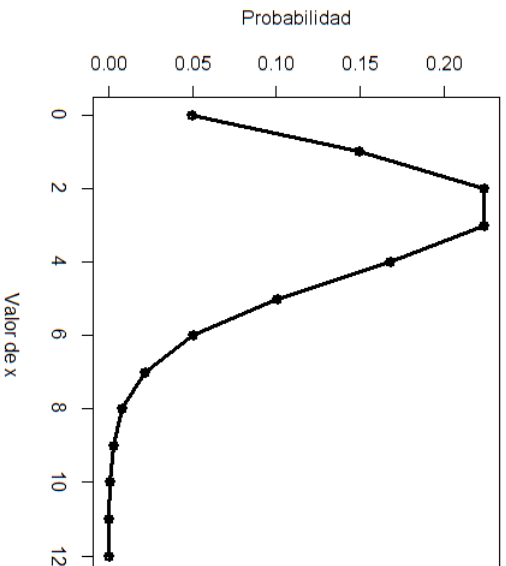
```
> dpois(5,3)
[1] 0.1008188
```

De manera similar a las anteriores distribuciones descritas en este manual, con la distribución de Poisson también se pueden calcular probabilidades acumuladas, gráficos y simulaciones muestrales. Para acabar esta sección, vamos a visualizar gráficamente la función de densidad de la distribución de Poisson con un parámetro $\lambda = 3$, y para los valores de $x \in [0 \ 12]$. Primero, introduciremos el vector x y, después, calcularemos los valores de la función de densidad evaluada con los valores de x en el vector **ensayo**:

```
> x <- 0:12
> ensayo <- dpois(0:12,3)
```

Ahora, mediante la función `plot()` representamos estos valores gráficamente, como vemos en la figura IV.2. En este gráfico bidimensional el primer argumento (**x**) es el vector de valores del eje horizontal y el segundo argumento (**ensayo**) es el vector de valores del eje vertical —las densidades. El resto de argumentos hacen referencia a parámetros gráficos.

```
> plot(x,ensayo,type="o",
+       xlab="Valor de x",ylab="Probabilidad",
+       lwd=3)
```

Figura IV.2: Función de densidad de la distribución de Poisson

4. Distribuciones de probabilidad continuas

4.1. Distribución uniforme

La distribución uniforme modela variables aleatorias continuas, de tal modo que todos los intervalos en el rango de la distribución tienen la misma longitud y son igualmente probables. El dominio de esta distribución está definido por los valores máximo y mínimo a y b , de manera respectiva. Una variable aleatoria X distribuida según una uniforme se denota como $X \sim U(a, b)$. Su función de densidad viene dada por la expresión siguiente:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{para } a \leq x \leq b, \\ 0 & \text{para } x < a \text{ o } x > b. \end{cases}$$

Mediante un ejemplo, veremos cómo trabajamos con variables aleatorias distribuidas según una uniforme. Supongamos que una mujer está dando a luz a un bebé y que la hora exacta del alumbramiento (X) sucederá en cualquier momento entre la hora 0 —ahora— y la hora 24 —la misma hora del día siguiente—, de modo que se sigue una distribución uniforme, $X \sim U(0, 24)$.

La probabilidad de que la madre dé a luz dentro de las primeras cinco horas se calculará de la manera siguiente: en la función `punif()`, introduciremos el valor de la variable de interés ($x = 5$), el mínimo (0) y el máximo (24). Además, hay que tener presente que buscamos la probabilidad que queda en la cola de la izquierda, $P(X < 5)$, y esto deberemos indicarlo también mediante la opción `lower.tail=TRUE`.

```
> punif(5, min=0, max=24, lower.tail=TRUE)
[1] 0.2083333
```

¿Cómo se interpreta este resultado? Hemos definido X como la hora del parto y hemos obtenido que $P(X < 5) = 0,208$. Es decir, la probabilidad de que la madre dé a luz dentro de las primeras cinco horas es aproximadamente del 21%. Es importante volver a insistir en que, cuando trabajamos con distribuciones continuas, $P(X < x)$ equivale a $P(X \leq x)$, ya que se calculan las áreas de la distribución y no puntos exactos. De esta manera, como ya hemos mencionado, en una distribución continua el cálculo de una probabilidad del tipo $P(X = x)$ será siempre nula, es decir, $P(X = x) = 0$.

Hay que recordar que podemos calcular probabilidades en las dos colas de la distribución. En el ejemplo anterior hemos calculado la cola izquierda ($P(X \leq x)$). Si ahora queremos calcular la probabilidad de que la madre dé a luz en las últimas dos horas —es decir, a partir de la hora 22—,

estaremos calculando la cola derecha $P(X > 22)$ y entonces deberemos especificar `lower.tail=FALSE`.

```
> punif(22, min=0, max=24, lower.tail=FALSE)
[1] 0.08333333
```

Es decir, una probabilidad un poco superior al 8 %.

4.2. Distribución exponencial

La distribución exponencial está relacionada con la distribución de Poisson. Esta distribución modeliza el intervalo de tiempo que transcurre entre dos sucesos. Formalmente, tiene un único parámetro $\theta = \lambda > 0$ y está definida para valores no negativos de la variable aleatoria. Sus parámetros son $E(X) = 1/\theta$ y $V(X) = 1/\theta^2$, y las funciones de densidad y de distribución toman la forma siguiente:

$$f(x) = \theta e^{-\theta x}$$
$$F(x) = P(X \leq x) = 1 - e^{-\theta x}$$

Veamos un ejemplo: en un gran hospital, el tiempo que transcurre entre dos partos —medido en horas— sigue una distribución exponencial con un parámetro $\theta = 4$. Esto significa que, ya que en esta distribución la esperanza viene definida como $E(X) = 1/\theta = 0,25$, de media transcurre un cuarto de hora ($1/4 = 0,25$) entre un parto y otro. Vamos a calcular la probabilidad de que transcurra una hora o más entre dos partos, es decir, queremos calcular $P(X > 1)$. En la función `pexp()`, introducimos el valor de la variable ($x = 1$), el del parámetro ($\theta = 4$) y especificamos que deseamos la cola derecha.

```
> pexp(1,4,lower.tail=FALSE)
[1] 0.01831564
```

Es decir, $P(X > 1) = 0,018$. Esto indica que hay aproximadamente un 1,8% de probabilidad de que transcurra una hora o más entre dos partos. De manera alternativa, podemos calcular la probabilidad complementaria, es decir, la probabilidad de que transcurra como máximo una hora entre dos partos ($P(X < 1)$). Esto lo conseguimos cambiando la opción de `lower.tail`.

```
> pexp(1,4,lower.tail=TRUE)
[1] 0.9816844
```

Este resultado no es de extrañar, ya que si sumamos las probabilidades obtenidas en las dos colas, la suma siempre es uno. Esto es así porque siempre se cumple que:

$$P(X \leq x) + P(X > x) = 1.$$

En este caso, $P(X \leq 1) = 1 - P(X > 1) = 1 - 0,018 = 0,982$. Un último aspecto que veremos de la distribución exponencial es el gráfico de su función de densidad. Para ello, necesitamos una serie de valores de x en los que evaluar la función de densidad. Crearemos una serie que vaya de 0 a 2, con intervalos igual a 0,1, de manera que habrá 21 valores.

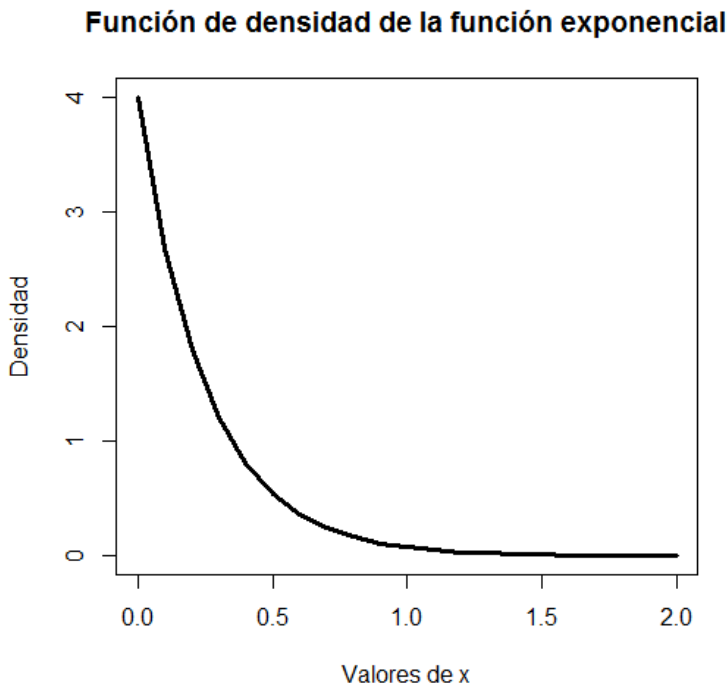
```
> x <- seq(0,2,by=0.1)
```

Ahora, evaluaremos todos los valores de x con la función `dexp()` y representaremos los valores resultantes gráficamente mediante la función `plot()`. Igual que en el gráfico anterior, sustituiremos los valores del eje vertical por defecto y colocaremos en su lugar los valores de la variable x .

```
> plot(x,dexp(x,4),type="l",lwd=3,  
+ xlab="Valores de x",ylab="Densidad",  
+ main="Función de densidad de la función  
exponencial")
```

En la figura IV.3 vemos el resultado.

Figura IV.3: Función de densidad de la distribución exponencial



4.3. Distribución normal

La distribución normal o gaussiana es la principal distribución de probabilidad en estadística y se utiliza en infinitud de ámbitos. Esta distribución se caracteriza por tener dos parámetros: la media μ y la desviación típica o estándar σ . De este

modo, se dice que una variable aleatoria X obedece a una ley normal si $X \sim N(\mu, \sigma)$. Su función de densidad viene definida por la expresión siguiente:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

La notación de la distribución normal

Con frecuencia, se suele denotar como $X \sim N(\mu, \sigma)$ aquella variable que se distribuye según una normal. No obstante, no hay consenso sobre esta notación, ya que a veces se expresa la varianza (σ^2) en lugar de la desviación típica —dependiendo del libro o manual. Aquí utilizaremos siempre σ por defecto.

Veamos un primer ejemplo. En el hospital del ejemplo anterior, el peso de los bebés al nacer sigue una distribución normal con media $\mu = 3,5$ y desviación típica $\sigma = 0,25$. El primer paso va a ser calcular el gráfico de la función de densidad entre los valores 2 y 5. Para ello, seguiremos el mismo procedimiento que en las figuras IV.2 y IV.3. Primero crearemos el vector de valores de x . Para calcular los valores de la función de densidad usaremos la función `dnorm`, donde el primer argumento es el vector x , el segundo argumento es la media $\mu = 3,5$ y el tercer argumento es la desviación estándar $\sigma = 0,25$.

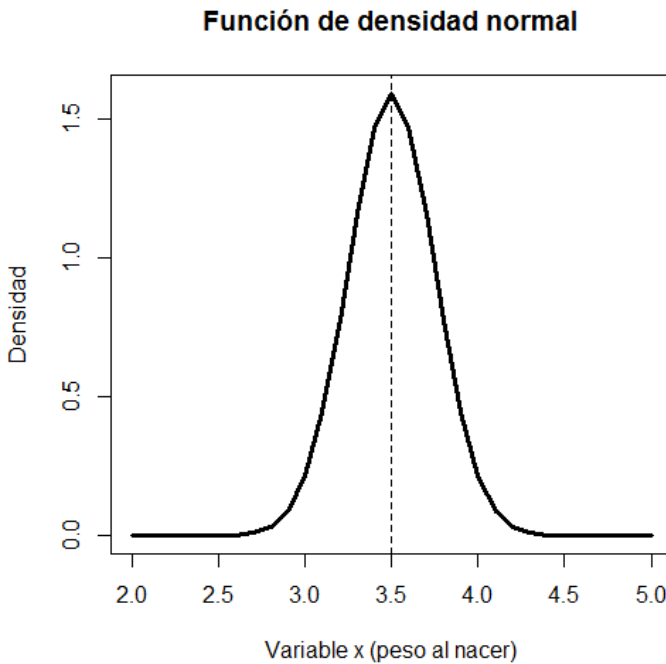
```
> x <- seq(2,5,by=0.1)
> densidad <- dnorm(x,3.5,0.25)
```

Ahora solo queda invocar la función `plot()` para obtener el gráfico. Fijaos que hemos colocado una línea discontinua vertical sobre el valor $\mu = 3,5$, para resaltar la simetría de esta distribución.

```
> plot(x,densidad,type="l",lwd=3,xlab="Variable x  
  (peso al nacer)",ylab="Densidad",main="Función de densidad normal")  
> abline(v=3.5,lty=2)
```

La figura IV.4 muestra el resultado.

Figura IV.4: Función de densidad normal



Imaginemos que deseamos calcular la probabilidad de que un niño pese entre tres y cuatro kilos al nacer, es decir, $P(3 \leq X \leq 4)$. El hecho de que sea una distribución simétrica nos ofrece diferentes opciones de calcular probabilidades. Si tomamos el gráfico de arriba, vemos que si trazamos una línea

vertical en la media aritmética (3,5), en las dos partes de la distribución queda el 50 % de la masa probabilística. Esto mismo también se puede ver si nos fijamos en que $\mu = 3,5$ se sitúa en medio de los valores 3 y 4; entonces, el área que quedará a la izquierda de 3 y el área que quedará a la derecha de 4 serán iguales debido a la simetría de la distribución. Por tanto, si queremos calcular el área que queda entre 3 y 4, es decir, $P(3 \leq X \leq 4)$, una opción será calcular uno menos las dos colas de los extremos, $P(X \leq 3)$ y $P(X \geq 4)$:

$$P(3 \leq X \leq 4) = 1 - P(X \leq 3) - P(X \geq 4)$$

Como sabemos que estos dos extremos han de tener el mismo área o valor, es decir, $P(X \leq 3) = P(X \geq 4)$, la fórmula anterior se puede simplificar, por ejemplo, de la manera siguiente.

$$P(3 \leq X \leq 4) = 1 - 2 P(X \leq 3)$$

Otras maneras alternativas de obtener esta probabilidad son las siguientes:

$$P(3 \leq X \leq 4) = 1 - 2 P(X \geq 4)$$

$$P(3 \leq X \leq 4) = P(X \leq 4) - P(X \leq 3)$$

Si seguimos la primera opción, la probabilidad $P(X \leq 3)$ con R se calcula así:

```
> pnorm(3, mean=3.5, sd=0.25, lower.tail=TRUE)
[1] 0.02275013
```

De esta manera, obtenemos que $P(3 \leq X \leq 4) = 1 - 2 \cdot 0,023 = 0,954$. Es decir, más del 95 % de los bebés nacerán dentro del intervalo de pesos [3,4].

Veamos ahora un ejemplo de cálculo de cuantiles. Supongamos que en este hospital, el jefe de pediatría quiere separar a los bebés en tres grupos: *a*) muy poco peso, *b*) poco peso y *c*) peso normal, con el objetivo de hacer un uso eficiente de las incubadoras y del personal médico para la vigilancia de los bebés. El doctor decide que habrá un 5 % de los bebés en el primer grupo, un 15 % en el segundo y el resto —un 80 %— en el tercero. La pregunta es: ¿cuáles serán los puntos de corte entre los tres grupos? Para responder a esta pregunta, tenemos que calcular los percentiles 0,05 y 0,2, los cuales dividen el área total en las tres partes que nos interesan —es decir, 5 %, 15 % y 80 %, en este orden. Para calcularlo, usaremos la función `qnorm()`, donde el primer argumento será el vector de valores 0,05 y 0,2, y el resto de argumentos son los parámetros de la distribución.

```
> qnorm(c(0.05,0.2), mean=3.5, sd=0.25,  
+ lower.tail=TRUE)  
[1] 3.088787 3.289595
```

Estos percentiles indican que el primer grupo —bebés con muy poco peso— estará formado por los bebés que pesan menos de 3,08 kilos; el segundo —bebés con poco peso—, por los que pesen entre 3,08 y 3,29 kilos; y el tercero —bebés con peso normal—, por los que pesen más de 3,29 kilos.

Un caso relevante en el estudio de la distribución normal es el de la distribución de la media muestral de una variable aleatoria. Para ilustrarlo, partamos del ejemplo anterior del hospital. Los doctores quieren hacer estadísticas de control del peso de los bebés que nacen. Interpretando los n bebés que nacen cada día como una muestra aleatoria independiente, se analiza para cada una de estas muestras la media muestral. Ya que cada día —es decir, cada muestra— se obtendrá una media muestral distinta, esta se puede analizar como una

variable aleatoria denominada $\bar{X}$, que tiene su propia distribución. A la hora de analizar y calcular probabilidades sobre $\bar{X}$, es imprescindible distinguir si conocemos o no la dispersión de la población (σ) de la que se obtiene $\bar{X}$.

Si esta dispersión es conocida —como pasa en el ejemplo anterior—, $\bar{X}$ se distribuye como una normal, tal que:

$$\bar{X} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

Si n es el tamaño de la muestra, vamos a suponer que disponemos de una muestra de 150 bebés ($n = 150$) y queremos calcular $P(\bar{X} \geq 3,75)$, sabiendo que $\mu = 3,5$ y asumiendo que σ es conocida e igual a 0,25.

Antes de nada, debemos calcular cuál es la desviación típica de la variable media muestral $\bar{X}$:

$$\frac{\sigma}{\sqrt{n}} = \frac{0,25}{\sqrt{150}} = 0,02.$$

De este modo, si sabemos que $\bar{X} \sim N(3,5; 0,02)$, calcularemos esta probabilidad con la función `pnorm`.

```
> pnorm(3.75, mean=3.5, sd=0.02, lower.tail=FALSE)
[1] 3.732564e-36
```

Observamos que el resultado, en notación científica, es prácticamente cero. Este resultado indica que la probabilidad de que, en un día, la media muestral de los $n = 150$ bebés nacidos sea de 3,75 kilos ($P(\bar{X} \geq 3,75)$) es prácticamente nula.

4.4. Distribución t de Student

Muchas funciones de distribución se derivan de la normal estándar

A partir de la distribución normal tipificada, obtenemos la función chi-cuadrado $\chi_n^2 = \sum_{i=1}^n Z_i^2$; la t de Student $t_n = \frac{Z}{\sqrt{\frac{\chi_n^2}{n}}}$ y

la F de Snedecor $F_{n,d} = \frac{\frac{\chi_n^2}{n}}{\frac{\chi_d^2}{d}}$.

La distribución t de Student está asociada a la distribución normal que acabamos de ver. Se utiliza, entre otros casos, cuando desconocemos la dispersión de la variable que hay que analizar. Retomemos el ejemplo anterior del peso medio de los bebés que nacen en el hospital citado ($\bar{X}$). Supongamos que, como antes, tomamos una muestra de $n = 150$ bebés, pero en este caso solo sabemos que $\mu = 3,5$. Así pues, nos faltaría un parámetro para modelizar $\bar{X}$ según una distribución normal.

En este caso, deberemos hacer una estimación de la desviación típica calculando la desviación típica muestral:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2},$$

de manera que en los cálculos del ejercicio anterior reemplazaremos σ por s . Entonces, la distribución muestral de la media ya no es una distribución normal, como pasaba cuando conocíamos el verdadero valor de la desviación (σ).

Si σ es desconocida y n es el tamaño de la muestra, calcularemos el error estándar mediante este cociente:

$$\text{Error estándar} = \frac{s}{\sqrt{n}}$$

Así pues, si la variable que estudiamos sigue una distribución normal con media μ y desviación típica desconocida, entonces

$$\frac{\bar{X} - \mu}{\frac{s}{\sqrt{n}}}$$

sigue una distribución t de Student con $n - 1$ grados de libertad. En nuestro caso podemos definir una nueva variable aleatoria $\bar{Y}$, que representará el peso medio de estos bebés que nacen en el hospital citado. De manera formal, establecemos así esta distribución:

$$\bar{Y} \sim t_{n-1}$$

Supongamos que queremos saber la probabilidad de que el peso medio de los bebés sea inferior a tres kilos, es decir, $P(\bar{X} \leq 3)$, sabiendo únicamente que $\mu = 3,5$, $n = 150$ y $s = 0,18$. El primer paso será transformar la variable $\bar{X}$ en la variable $\bar{Y}$ con la que podamos operar. En este caso, el objetivo será calcular $P(\bar{Y} \leq y)$. El valor de y lo obtenemos a partir de la expresión:

$$y = \frac{x - \mu}{\frac{s}{\sqrt{n}}} = \frac{3 - 3,5}{\frac{0,18}{\sqrt{150}}} = -34,02.$$

Una vez ya tenemos el valor de la nueva variable distribuida como una t de Student, para efectuar el cálculo de $P(Y \leq -34,02)$ con R usaremos la función `pt`, donde el primer argumento es $-34,02$ y el segundo, los grados de libertad (`df`), que son igual a $n - 1 = 150 - 1 = 149$. Por último, especificamos que deseamos la cola izquierda.

```
> pt(-34.02, df=149, lower.tail=TRUE)
[1] 1.973326e-72
```

Así pues, hay una probabilidad nula de que el peso medio de los 150 bebés nacidos ese día sea inferior a los tres kilos.

4.5. Teorema del límite central

El teorema de Moivre-Laplace

Según este teorema, una distribución binomial se puede aproximar a una distribución normal siempre que se cumpla que $np(1 - p) > 5$.

El teorema del límite central (TLC) establece que si una muestra es lo bastante grande ($n > 30$), sea cual sea la distribución de la variable de interés, la distribución de la media muestral seguirá aproximadamente una distribución normal. Además, la media será la misma que la de la variable de interés y la desviación típica de la media muestral será aproximadamente el error estándar. El TLC tiene algunas implicaciones, como que una variable aleatoria X distribuida según una binomial o una Bernoulli se puede aproximar a una normal. En el caso de la distribución binomial, cuando $n > 30$ y np y $n(1 - p)$ son mayores que 5, la variable X se aproxima a una normal con $E(X) = np$ y $Var(X) = np(1 - p)$.

Retomemos el ejemplo visto en el caso de la distribución binomial, el del jugador de baloncesto que encestabá triples. Recordemos que $p = 0,3$, pero supongamos ahora que el número de intentos (ensayos) se eleva hasta $n = 50$. Queremos calcular la probabilidad de que el jugador enceste más de veinte triples. Por el teorema de Moivre-Laplace y el TLC sabemos que:

$$E(X) = np = 15$$

$$Var(X) = np(1 - p) = 10,5$$

Calculemos entonces $P(X > 20)$ con la función `pnorm`.¹

¹Fijémonos en el hecho de que hemos introducido la raíz cuadrada de la varianza en el espacio de la desviación típica y no la varianza tal cual.

```
> pnorm(c(20), mean=15, sd=sqrt(10.5),  
+ lower.tail=FALSE)  
[1] 0.06141132
```

Es decir, el jugador tiene un poco más del 6 % de probabilidades de encestar veinte triples en cincuenta intentos.

El segundo caso planteado hace referencia a las variables discretas dicotómicas que solo toman los valores 0 y 1, y que siguen una distribución de Bernoulli. Lógicamente estas variables están estrechamente vinculadas con las proporciones, que de hecho son la media de n variables aleatorias de Bernoulli de parámetro p , donde n es el tamaño de la muestra y p , la probabilidad de éxito de cada evento individual. Suponiendo la variable aleatoria dicotómica Y , cuando el tamaño de la muestra n sea grande, la distribución de la proporción Y será aproximadamente una distribución normal cuyos parámetros son $E(Y) = p$ y $Var(Y) = p(1 - p)/n$. Así pues, al calcular la desviación típica, siempre que hablemos de variables de Bernoulli esta será igual al error estándar de dicha variable.

En nuestro ejemplo, teníamos que $n = 50$ y $p = 0,3$, y queríamos averiguar la probabilidad de que el jugador de baloncesto encestará más de veinte triples. Si suponemos que estamos ahora aproximando una distribución de Bernoulli a una normal, lo que estaremos especificando es la proporción de triples que efectúa el jugador en cincuenta intentos. Entonces, si queremos calcular la probabilidad de que esta proporción sea superior al 40 %, los parámetros de la distribución normal que deberemos utilizar serán los siguientes:

$$E(Y) = p = 0,3$$
$$Var(Y) = \frac{p(1 - p)}{n} = 0,0042$$

Calcularemos $P(Y > 0,4)$ de forma análoga al ejemplo anterior:

```
> pnorm(0.4, mean=0.3, sd=sqrt(0.0042),  
+ lower.tail=FALSE)  
[1] 0.06141132
```

Fijémonos en que el resultado de esta probabilidad es exactamente el mismo que el de la calculada en el ejemplo anterior. Esto resulta del todo lógico, ya que una distribución de Bernoulli no es más que un caso particular de la distribución binomial. En nuestro caso, el 40 % de triples equivale a encestar veinte triples en cincuenta intentos. Es decir, el jugador tiene un poco más del 6 % de probabilidades de encestar el 40 % de los triples.

Bibliografía y librerías adicionales

La modelización y el estudio en profundidad de diferentes distribuciones de probabilidad es uno de los puntos fuertes de R, incluyendo la estimación de sus parámetros, la inferencia, los modelos de regresión y la simulación, entre otros aspectos. Por este motivo, existen numerosísimas librerías que amplían las capacidades de análisis sobre distribuciones que incorpora la distribución base de R. He aquí algunas de las más relevantes:

- *gamlss.dist*: contiene todas las distribuciones que se utilizarán para los modelos GAMLSS (Generalized Additive Models for Location, Scale and Shape). Estos modelos fueron introducidos para R por Stasinopoulos y Rigby (2007). Estos modelos permiten, en un contexto de regresión, ampliar la colección de distribuciones para la variable explicada, acomodando aquellas distribuciones altamente asimétricas y/o con una alta curtosis. En este paquete, cada distribución tiene su función de probabilidad *d*, su función de probabilidad conmutativa *p*,

la inversa de la función de probabilidad conmutativa q y su función de generación de valores aleatorios r .

- *distr*: proporciona un tratamiento conceptual de las distribuciones usando clases del tipo S4. Introduce la clase de objeto **Distribution**, incorporando también las funciones d , p , q y r .
- *VGAM*: estas son las siglas de *vector generalized linear and additive models*. Este paquete permite especificar y estimar muchos modelos y distribuciones mediante máxima verosimilitud (MV) y MV penalizada.
- *ActuDistns*: calcula la función de densidad de probabilidad, la función del *hazard rate* y la función cuantil de 44 modelos de supervivencia (*survival models*).
- *SuppDists*: incluye varias distribuciones suplementarias: Gauss Inversa, Kruskal-Wallis, Tau de Kendall, chi-cuadrado de Friedman, Ro de Spearman y distribuciones de Johnson, entre otros. Además incluye dos generadores de valores aleatorios.
- *lmomco*: este paquete es una implementación de la teoría de estimación por los momentos L. Estos momentos son análogos a los momentos clásicos, pero incorporan ventajas como la ausencia de sesgo, robustez y consistencia. El paquete incluye métodos de estimación para una gran variedad de distribuciones.

Capítulo V

Inferencia estadística

1. Inferencia sobre parámetros estadísticos

1.1. Motivación

Empezaremos esta sección definiendo qué entendemos por inferencia estadística.

Inferencia estadística

Este concepto hace referencia al proceso de analizar o deducir las propiedades de una distribución subyacente mediante el análisis de datos (muestra) provenientes de esta distribución. La inferencia consiste en calcular contrastes de hipótesis, intervalos de confianza y en estimar modelos.

Esta sección se dedica a la inferencia estadística sobre los principales parámetros de las distribuciones: la media poblacional (μ), la varianza (σ^2) y la proporción (π). Para ello se usan las principales herramientas de la inferencia estadística: los intervalos de confianza (IC) y los contrastes de hipótesis (CH). Normalmente, disponemos de una serie de datos muestrales de los que ignoramos los verdaderos parámetros poblacionales que han generado esta muestra. Para esto, los tendremos que estimar. En concreto, aprenderemos a trabajar con IC y CH, y veremos cómo se calcula un IC y se resuelve

un CH para la media aritmética, para la proporción y para la varianza. Además, trabajaremos la construcción de IC y la resolución de CH para la diferencia de medias tanto para muestras apareadas como independientes, la diferencia de proporciones y el cociente de varianzas.

Para realizar todos los contrastes de esta sección vamos a seguir utilizando las variables sobre demografía en Cataluña que introdujimos en el capítulo anterior. En particular, usaremos estas dos variables:

- *edad*: media de edad de la población en el municipio.
- *costa*: variable cualitativa dicotómica (*sí/no*), que indica si el municipio se encuentra en la costa o no.

1.2. Inferencia sobre la media poblacional

1.2.1. Varianza poblacional desconocida

A la hora de plantear el cálculo de intervalos de confianza para el parámetro de la media poblacional μ , si desconocemos la varianza poblacional (σ^2), tendremos que utilizar la varianza muestral s^2 y la desviación estándar muestral (s). Como consecuencia, al utilizar un parámetro estimado trabajaremos con la distribución t de Student. De este modo, con una muestra n y un nivel de significación α , el IC para el parámetro μ vendrá dado por la siguiente expresión que se obtiene a partir de la media muestral $\bar{X}$:

$$\bar{x} \pm t_{\alpha/2, n-1} \frac{s}{\sqrt{n}}.$$

Como veremos a continuación, IC y CH están íntimamente relacionados. Para el CH, partiremos de la hipótesis nula (H_0), que establece un valor determinado para la media ($\mu = \mu_0$). La hipótesis alternativa (H_1) puede estar planteada a dos colas (bilateral), si se quiere averiguar si $\mu \neq \mu_0$, o a una cola

(unilateral). En este último caso, puede ser por la izquierda ($H_1 : \mu < \mu_0$) o por la derecha ($H_1 : \mu > \mu_0$). Por último, hemos de asumir un nivel de significación (α) o, equivalentemente, un nivel de confianza ($1 - \alpha$).

Las hipótesis nunca se aceptan

Es muy importante aclarar que, aunque parezca lo mismo, estadísticamente no tiene ningún sentido aceptar una hipótesis. Lo que hacemos es no rechazarla con la información de la que disponemos y al nivel de confianza dado. Por tanto, nuestros resultados pueden cambiar si nos modifican el nivel de confianza o la información con la que estamos trabajando.

En R, utilizaremos la función `t.test` tanto para el IC como para el CH. Esta función tiene las siguientes opciones:

- **alternative**: estableceremos la opción `'greater'` si es un CH unilateral por la derecha, `'less'` si es unilateral por la izquierda y `'two.sided'` si es un contraste bilateral —opción por defecto.
- **mu**: se corresponde con el valor de μ_0 .
- **conf.level**: nivel de confianza ($1 - \alpha$). La opción por defecto es $1 - \alpha = 0,95$.

Veamos un ejemplo práctico de inferencia sobre el parámetro μ de la variable media de edad (`edad`) de cincuenta años. Una primera aproximación consiste en hacer un contraste de hipótesis bilateral, mediante el cual plantearemos las hipótesis siguientes:

$$\begin{array}{l} \overline{H_0 : \mu = 50} \\ \overline{H_1 : \mu \neq 50} \end{array}$$

Vamos a realizar este contraste en R:

```
> t.test(edad,mu=50)

One Sample t-test

data:  edad
t = -51.183, df = 940, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to
      50
95 percent confidence interval:
42.51746 43.07007
sample estimates:
mean of x
42.79377
```

Este resultado nos da diversa información. Proporciona el valor del estadístico (**t**), los grados de libertad (**df**) y el p-valor (**p-value**).

Interpretación del p-valor

El p-valor obtenido ya considera si el contraste es unilateral o bilateral. Por tanto, este p-valor siempre se ha de comparar con α y no con $\alpha/2$. Además, el resultado **p-value < 2.2e-16** se interpreta como que el p-valor no es estadísticamente diferente de cero.

Con el valor que obtenemos del p-valor, la decisión será la siguiente:

$$\begin{array}{lcl} \hline p\text{-valor} \leq \alpha & \Rightarrow & \text{Rechazo de } H_0 \\ p\text{-valor} > \alpha & \Rightarrow & \text{No rechazo de } H_0 \\ \hline \end{array}$$

Así, ya que el p-valor no es estadísticamente diferente de cero y es menor que $\alpha = 0,05$, resolvemos que rechazamos la hipótesis nula de que $\mu = 50$. Además, la función **t.test** también nos proporciona el IC con el nivel de confianza dado. En este caso, el parámetro μ se encontrará en el inter-

valo $[42,51 ; 43,07]$ con una confianza del 95 %. Por último, esta función también nos proporciona la media muestral ($\bar{x} = 42,79$).

1.2.2. Varianza poblacional conocida

A diferencia del caso anterior, cuando conozcamos el parámetro de la varianza poblacional —con desviación estándar σ —, utilizaremos la distribución normal para el cálculo del intervalo para el parámetro μ . De manera específica, tendremos la siguiente expresión para el intervalo de confianza también basado en la media muestral de la que disponemos ($\bar{x}$):

$$\bar{x} \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}}.$$

En el ejemplo anterior hemos asumido que no conocemos la desviación típica poblacional y para hacer inferencia (IC y CH) hemos tomado la desviación típica muestral de la variable `edad`. Supongamos que conocemos la desviación típica poblacional y que esta es $\sigma = 4,5$. En este caso, para el cálculo de IC y de CH tendremos que utilizar la distribución normal y no la *t* de Student.

Para hacer este cálculo en R necesitaremos instalar un paquete estadístico denominado PASWR —del inglés *Probability And Statistics With R*. Una vez instalado, lo tendremos que cargar antes de realizar el test.

```
> install.packages("PASWR")  
> library(PASWR)
```

Vamos a realizar el siguiente contraste de hipótesis con una confianza del 90 %:

$$\begin{array}{l} \overline{H_0 : \mu = 42} \\ \overline{H_1 : \mu > 42} \end{array}$$

Esto lo haremos con la función `z.test`, similar a la función `t.test`, pero donde tendremos que introducir el valor de la desviación estándar poblacional en la opción `sigma.x`. El comando es el siguiente:

```
> z.test(edad, alternative='greater', mu=42,
+  sigma.x=20, conf.level=0.9)

One-sample z-Test

data:  edad
z = 1.2175, p-value = 0.1117
alternative hypothesis: true mean is greater than
      42
90 percent confidence interval:
41.95822      Inf
sample estimates:
mean of x
42.79377
```

En este caso, al ser el p-valor = 0,1117, por lo tanto mayor que $\alpha = 0,1$, no podemos rechazar la hipótesis nula de que $\mu = 42$.

1.2.3. Diferencia de medias

El objetivo de este análisis es comprobar si hay diferencias estadísticamente significativas entre la media de una misma variable —en este caso, la edad— extraída en dos muestras independientes diferenciadas por una variable dicotómica, en este caso el factor costa. Así, la primera muestra está compuesta por municipios de costa (*c*) y la segunda, por municipios de interior (*i*). El objetivo del análisis va a ser, pues, tratar

de averiguar si el parámetro μ de la media de edad es igual para municipios costeros que para los de interior.

$$\begin{array}{l} \overline{H_0 : \mu_i = \mu_c} \\ \overline{H_1 : \mu_i \neq \mu_c} \end{array}$$

De manera alternativa, este contraste también se puede expresar así:

$$\begin{array}{l} \overline{H_0 : \mu_i - \mu_c = 0} \\ \overline{H_1 : \mu_i - \mu_c \neq 0} \end{array}$$

Contraste de diferencia de medias

Igual que en los casos anteriores, realizaremos este test con la `t` de Student (`t.test`) si no conocemos las varianzas poblacionales y utilizaremos la distribución Normal (`z.test`) si las conocemos.

En este caso, vamos a suponer que no conocemos las varianzas poblacionales. Lo que haremos es usar la función `t.test` especificando `edad costa` —es decir, la variable `edad` separada en dos submuestras según el valor de `costa`. Además, especificaremos `mu=0` para indicar que la H_0 es $\mu_c - \mu_i = 0$.

```
> t.test(edad~costa, mu=0)

Welch Two Sample t-test

data:  edad by costa
t = 8.9344, df = 112.83, p-value = 8.97e-15
alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
2.194662 3.445345
sample estimates:
```

```
mean in group No mean in group Si
43.00354          40.18354
```

La interpretación de este resultado es análoga a los casos anteriores. El hecho de tener un p-valor $< \alpha$ hace que rechacemos la H_0 de que ambas medias son iguales, de manera que concluimos que la media de edad es estadísticamente diferente en los municipios costeros y en los de interior. De hecho, esta función nos proporciona un CI de la diferencia de medias de $[2, 19 ; 3, 44]$ con un 5 % de significación.

Respecto al contraste de diferencia de medias, también existe el contraste sobre la diferencia de medias con muestras apareadas, que consiste en que se toman dos o más observaciones de los mismos individuos. En este caso, se realizaría en test con la función `t.test`, especificando la opción `paired = TRUE`.

1.3. Inferencia sobre la varianza poblacional

Esta sección incluye la inferencia basada en la distribución F de Snedecor que se lleva a cabo para evaluar si las varianzas de dos muestras son iguales o no. Cabe mencionar que se supone que las dos muestras proceden de dos variables normales que, además, son independientes e idénticamente distribuidas. El objetivo es comprobar si las varianzas de estas variables son iguales o no.

La varianza muestral

Recordemos que, para n observaciones de una variable X , la varianza muestral se calcula como $S_X^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$.

El test F se basa en la ratio de varianzas muestrales, de manera que el estadístico toma la forma siguiente.

$$F = \frac{S_1^2}{S_2^2} \sim F_{n-1, m-1}.$$

Donde n y m son las dimensiones de las dos muestras, de manera respectiva. Igual que en el apartado anterior, vamos a realizar este test para la variable media de edad (**edad**). En concreto, queremos comprobar si su varianza es igual para los municipios de costa que para los de interior. Siendo σ_i^2 la varianza de la variable en los municipios de interior y σ_c^2 la varianza en los municipios de costa, el contraste (bilateral) que queremos plantear es el siguiente, ya que cuando hablamos de igualdad de varianzas nos referimos a que el cociente (ratio) de las mismas sea la unidad:

$$\begin{array}{l} \overline{H_0 : \frac{\sigma_i^2}{\sigma_c^2} = 1} \\ \overline{H_1 : \frac{\sigma_i^2}{\sigma_c^2} \neq 1} \end{array}$$

La función que se ha de utilizar es **var.test** y sus opciones son análogas a las funciones **t.test** y **z.test**. En este caso, el contraste bilateral al 90 % de confianza, y donde fijamos que la H_0 es que el ratio de varianzas es igual a 1, es el siguiente:

```
> var.test(edad~costa, ratio=1, conf.level=0.9)

F test to compare two variances

data:  edad by costa
F = 3.519, num df = 870, denom df = 69, p-value =
  2.689e-09
alternative hypothesis: true ratio of variances
  is not equal to 1
90 percent confidence interval:
```

```
2.571425 4.617085
sample estimates:
ratio of variances
3.519
```

La interpretación de este resultado es, también, análoga a los casos anteriores. El hecho de tener un p-valor $< \alpha$ hace que rechacemos la H_0 de que ambas varianzas son iguales. Esta función nos proporciona un CI del cociente de varianzas de $[2,57 ; 4,61]$ con un 10 % de significación. De esta manera, concluimos que la varianza de la variable es mayor para los municipios del interior que para los de la costa.

1.4. Inferencia sobre la proporción

El parámetro de la proporción

Algunos manuales y R-Commander denominan el parámetro de proporción poblacional p en lugar de π . Los dos son correctos: lo que ocurre es que el alfabeto griego se suele reservar para los parámetros referidos a la población y, por este motivo, nosotros hemos decidido utilizar π .

Al hacer inferencia sobre la proporción, estamos interesados en hacer inferencia sobre el parámetro π , es decir, la proporción poblacional. Si $\hat{p}$ es la proporción muestral y utilizamos la distribución normal, el intervalo de confianza vendrá dado por esta expresión:

$$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

Intervalos de máxima holgura

En ocasiones, no disponemos de suficiente información como para utilizar la proporción muestral $\hat{p}$. En estos casos se utiliza $\hat{p} = 0,5$, ya que da lugar al IC más ancho posible, denominado de máxima holgura.

Para poner un ejemplo de proporciones, vamos a suponer que hacemos el siguiente experimento: cojo una moneda, efectúo una serie de lanzamientos y cuento las caras. El objetivo es efectuar un contraste bilateral donde la H_0 sea que la proporción es del 50 %, con una confianza del 95 %.

$$\begin{array}{l} H_0 : \pi = 0,5 \\ H_1 : \pi \neq 0,5 \end{array}$$

En R, la función que hace este contraste es `prop.test`. Tenemos dos opciones para introducir los datos del experimento como argumentos de la función:

- Dos vectores **x** y **n**, conteniendo el número de resultados favorables y de intentos totales, respectivamente. La longitud del vector se corresponde con el número de muestras que se han de contrastar.
- Una sola tabla, o una matriz con dos columnas, conteniendo, en la primera columna, el número de eventos favorables y, en la segunda columna, el número de eventos desfavorables. Igual que antes, el número de filas se corresponde con el número de muestras que se han de contrastar.

Además, introduciremos un valor **p** con la probabilidad de éxito que especifica el contraste. En el caso de que haya más de una muestra, tendremos que introducir un vector con las

probabilidades en cada uno de las muestras. Las opciones `alternative` y `conf.level` son las mismas que en las funciones anteriores, con los mismos valores por defecto.

Vamos a efectuar el primer contraste. Supongamos que cojo la moneda, efectúo 1.000 lanzamientos y cuento en total 525 caras. En este caso, al haber una sola muestra, introduzco los valores directamente en la función `prop.test`.

```
> prop.test(525, 1000, p=0.5)

1-sample proportions test with continuity
correction

data: 525 out of 1000, null probability 0.5
X-squared = 2.401, df = 1, p-value = 0.1213
alternative hypothesis: true p is not equal to
0.5
95 percent confidence interval:
0.4935129 0.5562927
sample estimates:
p
0.525
```

Como vemos, el valor del p-valor no me permite rechazar la hipótesis nula de que el parámetro poblacional es de 0,5. El siguiente paso será ampliar el experimento a cuatro muestras: efectúo cuatro series de 1.000 lanzamientos cada una y apunto al final de cada serie el número de caras que obtengo. Una vez he finalizado el experimento, introduzco en R un vector, al que denominaremos `caras` con los catorce resultados, uno por cada serie de lanzamientos:

```
> caras<-c(510,497,521,485)
```

Ahora introduzco otro vector, esta vez con el número de lanzamientos de cada serie, que se llamará simplemente `n`. Ya

que se trata de un vector con el número 1.000 repetido cuatro veces, voy a utilizar la función `rep` para generar este vector.

```
> n<-rep(1000,4)
```

Igualmente, necesito un vector con las probabilidades de cada experimento.

```
> prob<-rep(0.5,4)
```

Ahora ya puedo efectuar el contraste. Al igual que antes, deseamos contrastar si la proporción es estadísticamente diferente de $\pi = 0,5$. Al 95 % de confianza y planteado de modo bilateral, tenemos el siguiente resultado:

```
> prop.test(caras,n,p=prob)

4-sample test for given proportions without
continuity correction

data:  caras out of n, null probabilities prob
X-squared = 3.1, df = 4, p-value = 0.5412
alternative hypothesis: two.sided
null values:
prop 1 prop 2 prop 3 prop 4
0.5    0.5    0.5    0.5
sample estimates:
prop 1 prop 2 prop 3 prop 4
0.510  0.497  0.521  0.485
```

Al igual que antes, no rechazamos la H_0 de que la proporción es del 50 %, cosa que es de esperar si la moneda no está trucada.

2. Modelos de regresión

2.1. Motivación

Este capítulo tiene como principal objetivo introducir el análisis de regresión utilizando el entorno estadístico R. Los modelos de regresión son imprescindibles en diversos ámbitos como la economía, la empresa y el marketing. De hecho, la econometría podría definirse como la estadística aplicada a la economía y se basa en el uso de la modelización.

El análisis de regresión

En inferencia estadística, el análisis de regresión es un proceso para estimar las relaciones entre variables. Incluye muchas técnicas para modelar y analizar varias variables, en especial la relación entre una variable dependiente y una o más variables independientes —o predictores.

Específicamente, definimos un análisis econométrico como un estudio de las relaciones que se establecen entre un conjunto de variables. El análisis parte de una serie de observaciones empíricas de las variables que se desea estudiar:

$$\{(y_1, x_1), (y_2, x_2), \dots, (y_i, x_i), \dots, (y_n, x_n)\} = \{(y_i, x_i) : i = 1, \dots, n\}$$

Cada par $\{y_i, x_i\} \in \mathbb{R} \times \mathbb{R}^k$ corresponde a una observación de una unidad —individuo, empresa, familia... El conjunto de observaciones consideradas configuran la muestra, la cual está compuesta de n elementos. Cada par (y_i, x_i) está compuesto por dos elementos: la variable dependiente y_i —también denominada endógena o explicada— y el vector de regresores o variables independientes, explicativas o exógenas x_i .

El objetivo de una regresión es el de encontrar la tendencia central de la distribución condicional de y_i , dado x_i . Una medida de tendencia central por excelencia de una variable es su media. Para el caso condicional, la medida análoga es la media condicional o esperanza condicionada $m(x) = E(y_i|x_i = x)$ que puede tomar cualquier forma: lineal, cuadrática, logarítmica, etc. Sin embargo, como veremos a continuación, la forma más usual es la lineal.

Un elemento importante de la regresión es el error o término de perturbación u_i y viene definido como la diferencia entre y_i y su media condicional, esto es:

$$u_i = y_i - m(x)$$

Matemáticamente, podemos reorganizar la ecuación anterior, obteniendo la fórmula:

$$y_i = m(x) + u_i$$

Los únicos supuestos en los que nos basamos a la hora de construir esta nueva ecuación son que las variables (y_i, x_i) tienen una distribución conjunta y que $E|y_i| < \infty$.

Es fundamental establecer las propiedades de u_i , es decir, del término de error o perturbación de la regresión:

- $E(u_i|x_i) = 0$
- $E(u_i) = 0$
- $E(h(x_i)u_i) = 0$, para cualquier función $h(\cdot)$
- $E(x_i u_i) = 0$

La ecuación $y_i = m(x) + u_i$ y la propiedad $E(u_i|x_i) = 0$ son considerados con frecuencia el marco de una regresión pero no un modelo. En este marco no se está suponiendo ninguna

restricción —es decir, no se supone ninguna característica de la distribución conjunta de las variables, como podría ser la linealidad, por ejemplo. Así, ambas ecuaciones son ciertas por definición.

Modelo de regresión lineal

En el momento en el que asumimos ciertas restricciones en la distribución conjunta de las variables, obtenemos un modelo. La restricción más común en econometría, por simplicidad y por facilidad de estimación e inferencia es la de linealidad, es decir, se asume a priori que $m(x)$ es una función lineal de (x) .

Por su carácter introductorio, esta sección se ocupa de los modelos de regresión lineales, aunque R ofrece un potencial tremendo para estudiar multitud de modelos de regresión. Un modelo de regresión lineal tiene la siguiente formulación:

$$y_i = x_i' \beta + u_i$$
$$E(u_i | x_i) = 0.$$

La restricción de linealidad no tiene por qué cumplirse en cualquier aplicación específica. La validez de esta restricción —y de cualquier otra— dependerá, en todo caso, de las características de la distribución conjunta de las variables consideradas.

Como veremos a continuación, existen tres etapas principales en el análisis de regresión: la especificación, la estimación y el diagnóstico. Una cuarta etapa, que consiste en la predicción, no se incluye en este capítulo.

2.2. Especificación de modelos

La especificación es la primera fase en el análisis de regresión. Consiste en la construcción del modelo que se ha de estimar.

La primera cuestión es qué variables se incluyen en el modelo y la segunda qué forma funcional adoptan, esto es: lineal, aditiva, multiplicativa, etc.

En R, las diferentes especificaciones se almacenan en fórmulas, introducidas mediante la función homónima `formula`. La especificación aditiva es la más simple y tiene la siguiente estructura:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

En R, esta especificación se introduce de la siguiente manera:

$$y \sim x_1 + \dots + x_k$$

Sin embargo, hay muchas especificaciones alternativas, donde las variables tienen una relación funcional diferente. La tabla V.1 muestra una lista de las principales maneras en que se pueden introducir variables en un modelo.

Tabla V.1: Especificaciones alternativas de modelos

Modelo	Fórmula	Instrucción
Aditivo	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$	<code>y ~ x1 + x2</code>
Aditivo sin constante	$y = \beta_1 x_1 + \beta_2 x_2$	<code>y ~ 0 + x1 + x2</code>
Con regresores sumados	$y = \beta_0 + \beta_1 (x_1 + x_2)$	<code>y ~ I(x1 + x2)</code>
Con regresor x_2 al cuadrado	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2^2$	<code>y ~ x1 + I(x2^2)</code>
Con regresores multiplicados	$y = \beta_0 + \beta_1 (x_1 \cdot x_2)$	<code>y ~ I(x1 * x2)</code> <code>y ~ x1 : x2</code>
Con término de interacción	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 (x_1 \cdot x_2)$	<code>y ~ x1 * x2</code> <code>y ~ x1 + x2 + x1 : x2</code> <code>y ~ (x1 + x2)^2</code>

Vamos a especificar algunos modelos con los datos de demografía en Cataluña que introdujimos en el capítulo anterior. En particular, vamos a considerar las siguientes variables:

- *edad*: media de edad de la población en el municipio.
- *t.act*: tasa de actividad en el municipio.
- *t.mig*: tasa de inmigración en el municipio.
- *costa*: variable cualitativa dicotómica —*sí/no*—, que indica si el municipio se encuentra en la costa o no.

La variable **costa** merece un comentario, ya que no es una variable numérica, sino cualitativa, con valores *sí* y *no*.

```
> class(costa)
[1] "factor"
> levels(costa)
[1] "No" "Si"
```

Esta variable, como vemos, es considerada por R como un factor, de manera que es posible incluirla en un modelo de regresión. Otra aproximación sería introducirla como una variable numérica con valores 0 y 1, cosa que vamos a hacer. La idea es crear una nueva variable numérica, a la que llamaremos **costa2**, y que será un vector con tantos ceros como el tamaño de la muestra:

```
> costa2 <- rep(0, length(costa))
```

Ahora, utilizaremos la función **which()** para encontrar qué observaciones se corresponden con el *sí* de la variable **costa**. Al vector con la posición de los *síes* le llamaremos **si**.

```
> si <- which(costa=="Si")
```

Por último, mediante los corchetes [] accederemos a las posiciones que nos da el vector `si` y les asignaremos el valor 1. Con ello, ya tenemos la variable dicotómica, donde 1 se corresponde con la categoría *municipio en la costa*, y el 0 es el caso contrario.

```
> costa2[si] <- 1
```

Vamos ahora con las especificaciones. Supongamos que queremos estudiar si las tasas de actividad e inmigración, además de la localización en la costa, determinan la media de edad en los municipios. Vamos a introducir tres especificaciones, con nombres `e1`, `e2` y `e3`, respectivamente:

- Primer modelo:

$$\text{edad}_i = \beta_0 + \beta_1 \cdot \text{t.act}_i + \beta_2 \cdot \text{t.mig}_i + \beta_3 \cdot \text{costa2}_i + u_i$$

```
> e1<-formula(edad ~ t.act + t.mig + costa2)
```

- Segundo modelo:

$$\text{edad}_i = \beta_0 + \beta_1 \cdot \text{t.act}_i + \beta_2 \cdot \text{t.mig}_i^2 + u_i$$

```
> e2<-formula(edad ~ t.act + I(t.mig^2))
```

- Tercer modelo:

$$\begin{aligned} \text{edad}_i &= \beta_0 + \beta_1 \cdot \text{t.mig}_i + \beta_2 \cdot \text{costa}_i \\ &+ \beta_3 \cdot (\text{t.mig}_i \cdot \text{costa2}_i) + u_i \end{aligned}$$

```
> e3<-formula(edad ~ t.mig*costa2)
```

2.3. Estimación de modelos

Una vez tenemos las especificaciones, el proceso de estimación hace referencia a la técnica utilizada para obtener estimaciones de los parámetros del modelo. El conocimiento previo del fenómeno analizado y la disponibilidad de contrastes ayudarán a hacer una elección correcta del estimador. Para los modelos lineales, el modelo más simple, y también el más utilizado, es el método de los mínimos cuadrados ordinarios (MCO). Este método se utiliza para estimar los parámetros desconocidos en un modelo de regresión lineal, mediante la minimización de la suma de los cuadrados de los errores o residuos del modelo. Recordemos que los errores o residuos son la diferencia entre los valores muestrales de la variable dependiente (y_i) y sus valores estimados por el modelo ($\hat{y}_i$), es decir:

$$e_i = y_i - \hat{y}_i$$

Errores y residuos

Es importante no confundir el término de perturbación (u) con los residuos (e). El primer concepto es teórico y no observable, mientras que el segundo depende de la muestra y del método de estimación elegido, con lo cual es medible y analizable.

La función de R que estima modelos de regresión lineales por MCO es `lm` (del inglés, *linear model*). Esta función tiene estas opciones principales:

- **formula:** este es el primer argumento, y el más importante. Se trata de una fórmula, tal y como vimos en la sección previa.

- **data**: este argumento es opcional. Si las variables incluidas en la fórmula están en el espacio de trabajo,¹ no hay que especificar ninguna base de datos. En cambio, si las variables no están en el espacio de trabajo y pertenecen a una base de datos específica, hay que usar este argumento.
- **subset**: este argumento es opcional y se usa cuando queremos hacer una estimación con una selección de los datos. Podemos, o bien introducir las observaciones que deseamos incluir en la estimación, o bien uno o más argumentos lógicos.

Vamos a estimar los tres modelos con las tres especificaciones que tenemos, a los cuales llamaremos **m1**, **m2** y **m3**, respectivamente:

```
> m1<-lm(e1)
> m2<-lm(e2)
> m3<-lm(e3)
```

De esta manera, hemos creado tres objetos, cada uno de los cuales contiene toda la información relativa al modelo estimado. El siguiente paso es, lógicamente, la visualización de los resultados, la cual se hace aplicando el comando **summary()** al modelo que deseamos visualizar. Vamos a ver la información relativa a la estimación del primer modelo.²

¹Recordemos que, para volcar las variables de una base de datos en el espacio de trabajo, hay que usar la función **attach**.

²Hemos reducido el número de decimales para poder mostrar la información correctamente.

```

> summary(m1)

Call:
lm(formula = e1)

Residuals:
Min       1Q   Median       3Q      Max
-10.836  -2.834  -0.074   2.661  15.585

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  45.42      0.259  175.201 < 2e-16 ***
t.act        -0.04      0.005   -8.367 < 2e-16 ***
t.mig        -0.14      0.018   -7.423 2.57e-13 ***
costa2       -1.52      0.532   -2.867 0.00424 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.998 on 937 degrees of
freedom
Multiple R-squared:  0.1457
Adjusted R-squared:  0.1429
F-statistic: 53.25 on 3 and 937 DF
p-value: < 2.2e-16

```

Este resultado es un amplio sumario de la regresión. Veamos sus principales componentes:

- 1) *Residuals*: mínimo, máximo y cuartiles de los residuos de la regresión, los cuales proporcionan información sobre su distribución.
- 2) *Coefficients*: cuadro en el que aparece información de la estimación de los parámetros —o coeficientes— estimados.
- 3) *Estimate*: estimación de cada parámetro —*intercept* significa constante.

- 4) *Std. Error*: desviación —o error— estándar de cada parámetro estimado.
- 5) *t value*: estadístico t de cada parámetro estimado, obtenido dividiendo la estimación del parámetro entre su desviación estándar. Este estadístico es el que utilizamos para hacer el contraste de significación individual de los parámetros estimados. Recordemos que se trata de un contraste cuyas hipótesis son $H_0 : \text{parámetro} = 0$ y $H_1 : \text{parámetro} \neq 0$.
- 6) $Pr(> |t|)$: *p-valor* del contraste de significación individual de cada parámetro estimado, el cual indica su significación estadística.
- 7) *Signif. codes*: muestra, con asteriscos y puntos, para qué niveles de significación los coeficientes estimados son o no significativos. En este caso, vemos que todos los coeficientes estimados son significativos con un nivel de significación del 1 %.
- 8) *Residual standard error*: desviación —o error— estándar de los residuos.
- 9) *Multiple R-squared*: coeficiente de determinación.
- 10) *Adjusted R-squared*: coeficiente de determinación ajustado.
- 11) *F-statistic*: estadístico F para el contraste de la significación global o conjunta de los parámetros estimados del modelo. Recordemos que se trata de un contraste cuyas hipótesis son las siguientes:

H_0 : todos los parámetros son iguales a 0

H_1 : como mínimo, un parámetro es diferente de 0

12) *DF*: grados de libertad del estadístico *F*.

13) *p-value*: p-valor asociado al contraste anterior. En este caso, vemos que el conjunto de parámetros estimados es significativo con un nivel de significación del 0,1 % —ya que $p\text{-valor} < 0,001$.

Una manera alternativa de estudiar la significación individual de los parámetros estimados es el cálculo de intervalos de confianza. Tomando un nivel de confianza del $(1 - \alpha)$ —es decir, una significación de α —, existe una probabilidad del $(1 - \alpha)$ de que el parámetro β esté incluido en el siguiente intervalo:

$$\beta \in \left[\hat{\beta} \pm t_{\alpha/2; n-k} s_{\hat{\beta}} \right].$$

En R, la función que devuelve los intervalos es `confint`, donde el primer argumento es el modelo estimado y el segundo argumento se corresponde con el nivel de confianza, como vemos a continuación:

```
> confint(m1, level=0.9)
              5 %      95 %
(Intercept) 45.0002 45.8540
t.act       -0.0593 -0.0398
t.mig       -0.1716 -0.1093
costaSi     -2.4029 -0.6497
```

Como vemos, al 90 % de confianza, ningún coeficiente estimado contiene el valor 0, con lo que todos los coeficientes son estadísticamente significativos, dado este nivel de confianza.

Hay un último aspecto que querríamos cubrir en esta sección. Hay casos en los que deseamos hacer una estimación con una parte de los datos, una selección de acuerdo con algún criterio. Para ello usaremos la opción `subset` de la función `lm`. Supongamos que deseamos estimar el segundo modelo, pero

solo para aquellos municipios que sean costeros y que además tengan una tasa de inmigración superior al 15%. Para ello tendremos que utilizar el operador lógico intersección (&):

```
> summary(lm(e2,subset=(costa2==1 & t.mig >15)))

Call:
lm(formula = e2, subset = (costa2 == 1 & t.mig >
  15))

Residuals:
Min      1Q  Median      3Q      Max
-3.7548 -1.0159 -0.2594  1.0331  7.0819

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  41.6038    0.9166   45.386  <2e-16 ***
t.act        -0.0428    0.0339   -1.263    0.213
I(t.mig^2)   -0.0008    0.0006   -1.200    0.237
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
                 '.' 0.1 ' ' 1

Residual standard error: 2.207 on 43 degrees of
  freedom
Multiple R-squared:  0.05629,    Adjusted R-
  squared:  0.0124
F-statistic: 1.282 on 2 and 43 DF,  p-value:
  0.2877
```

Como vemos, en esta estimación solo el coeficiente de la constante es significativo al 90% de confianza.

2.4. Diagnóstico de modelos

Una vez estimados el modelo —o los modelos— que nos interesan, el siguiente paso es analizarlos y compararlos, para ver cuál se adapta mejor al modelo subyacente, es decir,

cuál describe mejor las relaciones asumidas entre variables. Lo primero que hemos de saber es que los modelos estimados —objetos creados con la función `lm`— almacenan mucha más información que la que se muestra en el sumario con la función `summary()`. En la tabla V.2 se muestran las principales funciones genéricas que se pueden aplicar a los modelos estimado.

Tabla V.2: Funciones genéricas para modelos estimados

Función	Descripción
<code>print</code>	Muestra un resumen simple
<code>summary</code>	Muestra un resumen completo
<code>coef</code>	Extrae los coeficientes
<code>coefficients</code>	Extrae los coeficientes
<code>residuals</code>	Extrae los residuos
<code>fitted.values</code>	Extrae los valores ajustados
<code>predict</code>	Realiza una predicción para datos nuevos
<code>plot</code>	Produce el gráfico de diagnóstico
<code>confint</code>	Extrae los intervalos de confianza
<code>deviance</code>	Extrae la suma de cuadrados de los residuos
<code>vcov</code>	Extrae la matriz de varianzas y covarianzas estimada
<code>logLik</code>	Estadístico de log-verosimilitud (<i>log-likelihood</i>)
<code>AIC</code>	Extrae el criterio de información AIC (<i>Akaike Information Criterion</i>)

A la hora de extraer los coeficientes, hay que tener en cuenta que se pueden extraer los coeficientes con y sin los errores estándar, los *valores t* y los *valores p*. A partir del primer modelo estimado, para extraer solo los coeficientes estimados procederemos de la siguiente manera.

```
> coefficients(m1)
(Intercept)      t.act      t.mig      costa2
45.42711338 -0.04963013 -0.14049617 -1.52637227
```

Sin embargo, si queremos extraer toda la información relativa a los coeficientes estimados, los extraeremos de la función `summary`, tal y como se muestra a continuación.

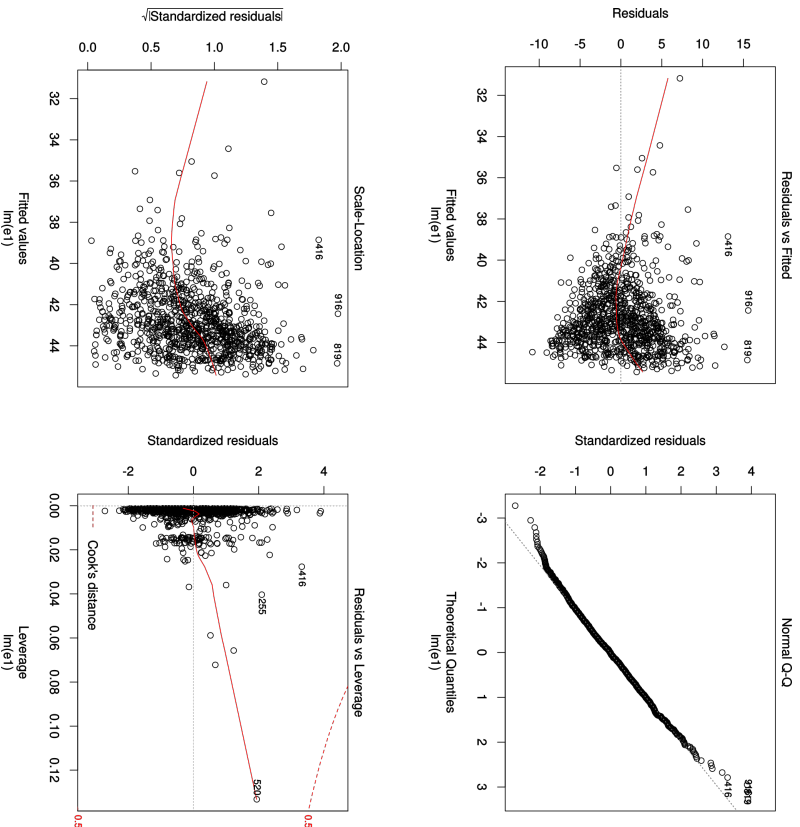
```
> coefficients(summary(m1))
              Estimate Std. Error  t value Pr(>|t|)
(Intercept)  45.4271     0.2592  175.2008  0.00e+00
t.act        -0.0496     0.0059   -8.3673  2.12e-16
t.mig        -0.1404     0.0189   -7.4228  2.57e-13
costa2       -1.5263     0.5324   -2.8669  4.23e-03
```

Una herramienta muy potente para el diagnóstico de un modelo es el análisis gráfico. En este sentido, como hemos visto en la tabla V.2, podemos aplicar la función `plot()` a un modelo estimado, y obtener los gráficos que se muestran en la figura V.1.

Los cuatro indicadores mostrados en este gráfico compuesto son los siguientes:

- 1) *Residuals vs Fitted*. Este diagrama representa los valores ajustados a la recta ($\hat{y}_i$) en el eje de abscisas y los residuos (e_i) en el eje de ordenadas. Es el principal diagnóstico de los residuos, ya que muestra si hay alguna tendencia o patrón de comportamiento en estos que pueda hacer sospechar de una posible correlación y/o heteroscedasticidad. Recordemos que, bajo las hipótesis de la estimación MCO, los residuos han de ser independientes e idénticamente distribuidos (*iid*).
- 2) *Scale-Location*. Este diagrama revela esencialmente la misma información que el gráfico anterior, con la diferencia de que en el eje de ordenadas aparece la raíz cuadrada de los residuos estandarizados.
- 3) *Normal Q-Q*. Este diagrama representa, en el eje de abscisas, los cuantiles teóricos de una distribución normal y, en

Figura V.1: Gráficas del diagnóstico de modelos



el eje de ordenadas, los cuantiles de la distribución de los residuos calculados. De esta manera, este diagrama muestra cómo se asemeja, o se aleja, la distribución de los residuos de una distribución normal. Mientras más coincidan los puntos de este gráfico con la bisectriz, más se asemejará la distribución de los residuos a la normal.

- 4) *Residuals vs Leverage*. Este diagrama sirve para detectar puntos que puedan ser variables influyentes, esto es, que retirarlos del modelo pueda causar un gran efecto en la

estimación. Esto sucede, por ejemplo, cuando obtenemos una observación de la variable explicada (y_i) que se aleja considerablemente de su valor estimado ($\hat{y}_i$). En el eje de abscisas tenemos el *leverage* —que indica cómo está de lejos cada punto de su valor medio explicado— y en el eje de ordenadas los residuos estandarizados. De esta manera, los puntos del diagrama identificados con una cifra —el número de observación i — merecen ser estudiados. En el próximo capítulo se abordará este tema en profundidad.

Comparación de modelos estimados

Cuando queramos comparar dos modelos que tengan la misma variable endógena, pero con distinto número de variables explicativas escogeremos aquel que tenga un mayor valor de la R^2 ajustada.

3. Análisis de la varianza (ANOVA)

El término ANOVA es muy utilizado en estadística. Pero ¿qué significa?

Análisis de la varianza (ANOVA)

Bajo ese concepto se incluyen diferentes modelos estadísticos multivariantes utilizados para analizar las diferencias entre las medias de grupo, en aquellos casos en los que las observaciones puedan ser categorizadas en uno o más grupos —o factores.

El análisis ANOVA divide la variación observada, o dispersión de una determinada variable, en componentes atribuibles a distintas fuentes de variación. En su forma más simple, un

ANOVA proporciona una prueba estadística para contrastar si las medias de varios grupos son iguales o no. Partiendo de los datos de la sección anterior, suponemos que la variable media de edad municipal (**edad**) se divide en dos grupos: las poblaciones que están en la costa y las que no. Esta clasificación nos viene dada por el factor **costa**. En este caso, mediante el ANOVA vamos a llevar a cabo el siguiente contraste de hipótesis:

$$\begin{array}{l} \overline{H_0 : \mu_c = \mu_i} \\ \overline{H_1 : H_0 \text{ no es cierta}} \end{array}$$

Es decir, estadísticamente ¿la media de la variable edad para la costa (μ_c) es igual a la media de esta variable para el interior (μ_i)? Este test se basa en descomponer la suma total de cuadrados (SCT) en dos partes: la dispersión explicada mediante los grupos (SCE) y la dispersión explicada por medio de otros factores distintos de los grupos en los que hemos dividido la población (SCD). Por lo tanto, siempre se cumplirá que:

$$SCT = SCE + SCD$$

Además, cuanto más se aproxime SCE a SCT , querrá decir que hay factores dentro de cada grupo que hacen que sean diferentes entre sí —medias diferenciadas. En cambio, un SCD proporcionalmente elevado significa que la variación de los datos se debe a factores externos no relacionados con los grupos.

El estadístico que hay que calcular y la distribución del mismo son:

$$F^* = \frac{\frac{SCE}{k-1}}{\frac{SCD}{n-k}} \sim F_{\alpha, k-1, N-k}$$

Donde n es el tamaño de la muestra y k es el número de grupos. La decisión final se tomará siguiendo este criterio:

$F^* > F_{\alpha, k-1, N-k}$	$\Rightarrow$	Rechazo de H_0
$F^* \leq F_{\alpha, k-1, N-k}$	$\Rightarrow$	No rechazo de H_0

Para aplicar este contraste en R, utilizaremos la función `aov` —acrónimo del inglés *análisis of variance*. Esta función es muy similar al modelo lineal (`lm()`) que vimos en la sección anterior. Su principal argumento será una especificación en forma aditiva entre la variable que se ha de descomponer y sus factores —tal y como aparece en la tabla V.1. En este caso, descomponemos la variable `edad` según el factor `costa` y guardamos el resultado en un modelo estimado al que llamamos `a1`. Al igual que en el caso de la regresión, mediante el comando `summary()` visualizamos el resultado.

```
> a1 <- aov(edad~costa)
> summary(a1)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
costa	1	515	515.3	28.43	1.22e-07 ***
Residuals	939	17019	18.1		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
                '.' 0.1 ' ' 1
```

Como vemos, esta función nos devuelve los grados de libertad (Df), la suma de cuadrados, el valor del estadístico F y el p-valor, igual que en un modelo de regresión. En este caso, el p-valor —estadísticamente no diferente de cero— nos indica que rechazamos la hipótesis nula, de manera que concluimos que la media de edad en municipios costeros (μ_c) y de interior (μ_i) sí que es diferente.

Bibliografía y librerías adicionales

Existen varios manuales que exponen las diferentes técnicas y librerías disponibles de análisis multivariante en R. Uno de los más completos es el manual *An Introduction to Applied Multivariate Analysis with R*, de B. S. Everitt y T. Hothorn, ya que en este se cubren aspectos generales del análisis multivariante, entre los cuales se encuentran: manejo y visualización de datos multivariados, el análisis de componentes principales, escalado multidimensional, análisis exploratorio y confirmatorio de factores y análisis clúster.

En cuanto a los modelos de regresión, vale la pena citar los libros *Applied Econometrics with R*, de C. Kleiber y A. Zeileis, el cual cubre aspectos básicos del lenguaje de programación en R, así como las funciones y procedimientos para estimar modelos de regresión lineales y para llevar a cabo contrastes sobre estos modelos; y *Linear Models with R*, de J. J. Faraway, el cual constituye una muy sólida introducción a la especificación y estimación de modelos lineales con R.

Un área fundamental de la econometría, que no cubrimos en este manual, es el de las series temporales. Para profundizar en este tema, el manual *Introductory Time Series with R*, de P. S. Cowpertwait y A. V. Metcalfe, proporciona una breve y concisa introducción a sus métodos y librerías: predicción, modelos estocásticos, regresión, modelos estacionarios y análisis espectral, entre otros temas. La obra *Dynamic linear models with R*, de G. Petris, P. Campagnoli y S. Petrone, presenta técnicas bayesianas en la estimación de modelos de regresión dinámicos, incluyendo métodos secuenciales de Monte Carlo. Por último, la integración de series temporales se expone en *Analysis of integrated and cointegrated time series with R*, de B. Pfaff. Este manual incluye el análisis de series estacionarias univariantes y multivariantes, las series no estacionarias,

el contraste de la orden de integración y la cointegración de varias series temporales.

En cuanto a las librerías adicionales, cabe destacar las siguientes:

- *car*: el nombre de este paquete es el acrónimo de *Companion to Applied Regression* e incluye muchas funciones que se aplican a un modelo de regresión ajustado, realiza cálculos adicionales en el modelo, calcula posibles modelos alternativos, y genera valores y gráficos sobre los modelos estimados.
- *cluster*: completo paquete con múltiples funciones destinadas al análisis clúster.
- *mda*: contiene herramientas para el análisis discriminante.
- *Hmisc*: proporciona muchas funciones útiles para el análisis con múltiples variables: gráficos de alto nivel, funciones para el cálculo del tamaño muestral, herramientas para la importación de datos, elaboración de tablas con resultados del análisis, exportación de objetos a L^AT_EX y recodificación de variables, entre otras funciones.
- *FactoMineR*: ofrece un largo número de funciones adicionales en el análisis factorial exploratorio, como el uso de variables cuantitativas y cualitativas, y la inclusión de variables y observaciones suplementarias.
- *fGarch*: incluye la modelización de modelos autoregresivos condicionados por heteroscedasticidad (*Autoregressive conditional heteroskedasticity models*).
- *forecast*: funciones para la predicción de series temporales y modelos lineales.

- *fracdiff*: cubre la estimación de modelos ARIMA diferenciados fraccionalmente.
- *lmtests*: es una colección de bases de datos, contrastes y ejemplos para el diagnóstico de modelos de regresión lineales.
- *psych*: paquete dedicado al análisis psicológico experimental y a la psicometría. Incluye una serie de rutinas para la construcción de escalas mediante análisis factorial, análisis clúster, además de ofrecer funciones que proporcionan estadísticos descriptivos básicos.
- *sandwich*: paquete dedicado a la estimación robusta de la matriz de covarianzas en un modelo de regresión. Específicamente, se proporcionan estimadores consistentes por heteroscedasticidad (HC) y estimadores consistentes por heteroscedasticidad y autocorrelación (HAC).
- *sem*: paquete dedicado a la especificación y estimación de modelos de ecuaciones estructurales mediante varias técnicas de estimación, como los mínimos cuadrados en dos etapas.
- *strucchange*: diseñado para determinar y contrastar cambios estructurales en modelos de regresión lineales. Incluye el test F (test de Chow) de cambio estructural, entre otros contrastes y métodos de visualización.
- *tsDyn*: librería dedicada a la estimación de modelos de series temporales no lineales.
- *tseries*: librería para el análisis de series temporales y las finanzas computacionales.
- *vars*: librería que cubre la modelización de modelos VAR (*Vector autoregression*).

Capítulo VI

Programación y simulación

1. Motivación

En estadística, a menudo, no se dispone de datos empíricos para validar teorías u obtener información estadística. En ese caso, la simulación es una alternativa muy eficaz. Esta consiste en el proceso de generar valores aleatorios y tratarlos como si fueran datos reales extraídos de una población o generados por un experimento real. Los datos generados de esta manera se denominan simulados o sintéticos.

La simulación tiene diversas aplicaciones, como la estimación de probabilidades y parámetros estadísticos —como la media y la varianza—, la verificación del supuesto de normalidad, la estimación de sesgos y la estimación de distribuciones de probabilidad para la realización de inferencia.

Esta sección se estructura así: la siguiente sección cubre la generación de datos aleatorios con **R** y las tres secciones subsiguientes son aplicaciones empíricas con ejemplos de cómo realizar simulación con **R**. La primera aplicación trata de los juegos de azar, la segunda del estudio del sesgo del estimador de la varianza muestral y el tercero es un análisis mediante simulación del teorema central del límite.

2. Generación de valores aleatorios

A la hora de programar una simulación generando valores aleatorios, a veces es interesante fijar los valores aleatorios obtenidos, de manera que el resultado sea replicable. En otras palabras, si generamos n valores aleatorios de una distribución normal, cada vez que ejecutemos la misma instrucción obtendremos valores diferentes. Para obtener, con cada instrucción, el mismo vector de variables aleatorias, usamos la función `set.seed`.

La función `set.seed(seed)`

Esta función se coloca antes de una instrucción en la que se generan valores aleatorios. Fijando un valor arbitrario de `seed`, nos aseguramos de obtener siempre los mismos valores en la instrucción.

Una de las principales funciones usadas en simulación es `sample`, la cual genera números aleatorios a partir de unas condiciones iniciales. Esta función tiene la siguiente estructura y componentes.

La función `sample(x,size,replace,prob)`

Esta función extrae aleatoriamente valores del vector `x` hasta completar un vector de longitud `size`. La opción `replace` puede ser `TRUE` o `FALSE`, que indica si las extracciones son con reemplazamiento o sin él. Por último, el vector `prob` es un vector opcional de probabilidades de extracción de los elementos de `x`. Si no es específica, todos los elementos de `x` son equiprobables.

Veamos un ejemplo de aplicación de esta función. Supongamos que queremos obtener diez extracciones de un experimento consistente en lanzar una moneda al aire. En este caso,

el espacio muestral es $\{C, X\}$, esto es, cara o cruz. Veamos el resultado.

```
> sample(c("C","X"),10,replace=TRUE)
[1] "X" "C" "X" "C" "X" "X" "C" "X" "C" "X"
```

Supongamos ahora que tenemos un vector con los números del 1 al 20 y deseamos extraer tres al azar *sin reemplazo*. Este experimento se haría de la siguiente manera. Si deseamos que al ejecutar esta instrucción siempre obtengamos los mismos valores, deberemos fijar una semilla (*seed*).

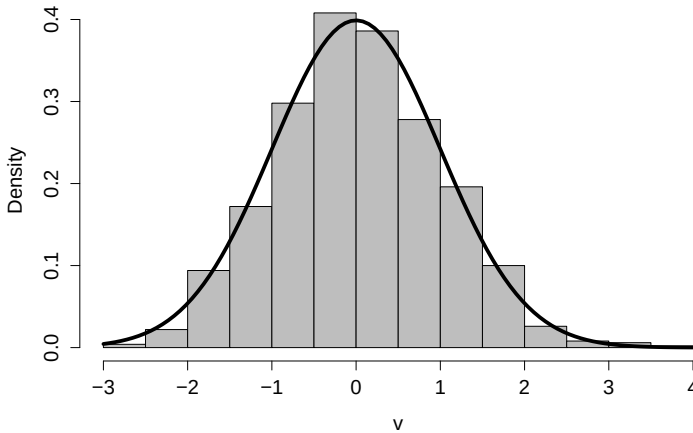
```
> set.seed(5)
> sample(1:20,3,replace=FALSE)
[1] 5 14 17
```

De igual manera, R dispone de generadores de datos provenientes de diversas distribuciones de probabilidad, tanto discretas como continuas.

Como se vio ampliamente en el capítulo dedicado a las distribuciones de probabilidad, la distribución base de R incorpora funciones específicas para generar valores aleatorios de diversas funciones de probabilidad discretas y continuas —la lista de estas funciones se encuentra en las tablas IV.1 y IV.2. Supongamos que deseamos obtener $n = 1.000$ valores aleatorios de una distribución normal con parámetros $\mu = 0$ y $\sigma = 1$, y después queremos obtener el histograma. Mediante la función `curve`, añadimos la función de densidad teórica al histograma de la figura VI.1.

```
> v <- rnorm(1000,0,1)
> hist(v,probability=TRUE,col="grey",main="")
> curve(dnorm(x,0,1),add=TRUE,lwd=4)
```

Figura VI.1: Histograma de los valores aleatorios de una distribución normal con $\mu = 0$ y $\sigma = 1$



De igual manera, el histograma y la función de densidad teórica para una distribución exponencial con parámetro $\lambda = 1$ se calculan de la siguiente manera.

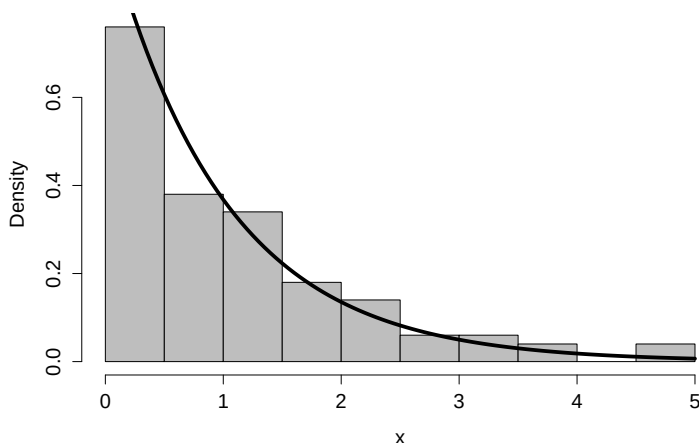
```
> v <- rexp(100,1)
> hist(v,probability=TRUE,col="grey",main="")
> curve(dexp(x,1),add=T,lwd=4)
```

El resultado se puede ver en la figura VI.2

3. Aplicación empírica: sesgo del estimador de la varianza muestral

En esta sección veremos un ejemplo práctico, en el cual demostraremos gráficamente el porqué de la fórmula de la varianza

Figura VI.2: Histograma de los valores aleatorios de una distribución exponencial con $\lambda = 1$



poblacional. Para una población determinada con N individuos y una media poblacional determinada μ , la fórmula de la varianza poblacional toma la siguiente forma.

$$\sigma^2 = \frac{\sum_{i=1}^N (x_i - \mu)^2}{N}$$

Sin embargo, en el caso de que dispongamos de una muestra de esta población, con media muestral $\bar{x}$ y tamaño muestral de n observaciones, la varianza muestral sesgada tiene la siguiente fórmula.

$$\hat{\sigma}_*^2 = \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{n}$$

Este estimador es sesgado porque su valor esperado no es el valor de la varianza poblacional, sino que es $E[\hat{\sigma}_*^2] = \frac{n-1}{n} \sigma^2$.

Para corregir ese sesgo hay que hacer una corrección, como muestra la fórmula de la varianza muestral insesgada.

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{n - 1}$$

La corrección del denominador $(n - 1)$ hace que su valor esperado sea el parámetro poblacional, esto es, $E[\hat{\sigma}^2] = \sigma^2$.

En lugar de hacer un desarrollo teórico de esta demostración, optaremos por hacer una simulación. La estrategia será la siguiente:

- 1) Generaremos un número elevado de muestras aleatorias S_n de una distribución normal estándar $N(\mu = 0, \sigma = 1)$ para diferentes tamaños muestrales ($n = 2, \dots, N$, siendo $N = 10$). Fijando para el número de muestras el valor arbitrariamente alto $K = 100.000$, generaremos K muestras de dos observaciones (S_2), K muestras de tres observaciones (S_3)... hasta K muestras con $N = 10$ observaciones S_N . Este es el total de muestras estimadas:

$$\begin{array}{cccc} S_{1,1} & S_{2,1} & \cdots & S_{N,1} \\ S_{1,2} & S_{2,2} & \cdots & S_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ S_{1,K} & S_{2,K} & \cdots & S_{N,K} \end{array}$$

- 2) Para cada una de estas muestras calcularemos *la varianza muestral sesgada* y las almacenaremos en la matriz V :

$$V = \left\{ \begin{array}{cccc} \hat{\sigma}_{*1,1}^2 & \hat{\sigma}_{*2,1}^2 & \cdots & \hat{\sigma}_{*N,1}^2 \\ \hat{\sigma}_{*1,2}^2 & \hat{\sigma}_{*2,2}^2 & \cdots & \hat{\sigma}_{*N,2}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\sigma}_{*1,K}^2 & \hat{\sigma}_{*2,K}^2 & \cdots & \hat{\sigma}_{*N,K}^2 \end{array} \right\}_{K \times 9}$$

- 3) El siguiente paso será calcular la media aritmética de cada columna, esto es:

$$\bar{\hat{\sigma}}_n^2 = \frac{\sum_{i=1}^K \hat{\sigma}_{*n,i}^2}{K}, \quad n = 2, \dots, 10.$$

- 4) El objetivo de esta simulación es comprobar si los cocientes observados $\frac{\bar{\hat{\sigma}}_n^2}{\sigma^2}$ se corresponden con los valores teóricos $\frac{n-1}{n}\sigma^2/\sigma^2 = \frac{n-1}{n}$. Teniendo en cuenta que la varianza poblacional toma el valor $\sigma^2 = 1$, los cocientes observados $\frac{\bar{\hat{\sigma}}_n^2}{\sigma^2}$ se convierten simplemente en $\bar{\hat{\sigma}}_n^2$.
- 5) El último paso es representar gráficamente los valores $\bar{\hat{\sigma}}_n^2$, cuyos valores estarán entre 0 y 1, en un diagrama de barras. Además, se superpone en el gráfico la línea $(n-1)/n$, la cual muestra la predicción teórica del valor de los cocientes representados.

Para realizar estos cinco pasos en R, empezaremos por definir las magnitudes iniciales K y N .

```
> K <- 100000
> N <- 10
```

El siguiente paso será crear la matriz V vacía, esto es, con el valor NA (*not assigned*).

```
> V <- matrix(NA, K, N-1)
```

El siguiente paso será ir completando la matriz V iterativamente. Esto es, mediante el ciclo iterativo `for(k in 1:K)` completaremos las filas y el ciclo iterativo `for(i in 2:N)` completará las columnas. Así, a cada elemento de la matriz

V le corresponderá un valor diferente de la varianza muestral sesgada con la instrucción `sum((x - mean(x)) ^ 2)/i`. La instrucción previa `set.seed(k)` asegura que los valores obtenidos sean replicables siempre que ejecutemos esta pieza de código.

```
> for(k in 1:K){
+   set.seed(k)
+   for(i in 2:N){
+     x <- rnorm(i)
+     V[k,(i-1)] <- sum((x-mean(x))^2)/i
+   }}
```

El penúltimo paso será calcular los valores finales calculando la media aritmética de las columnas.

```
> V.pond <- colMeans(V)
```

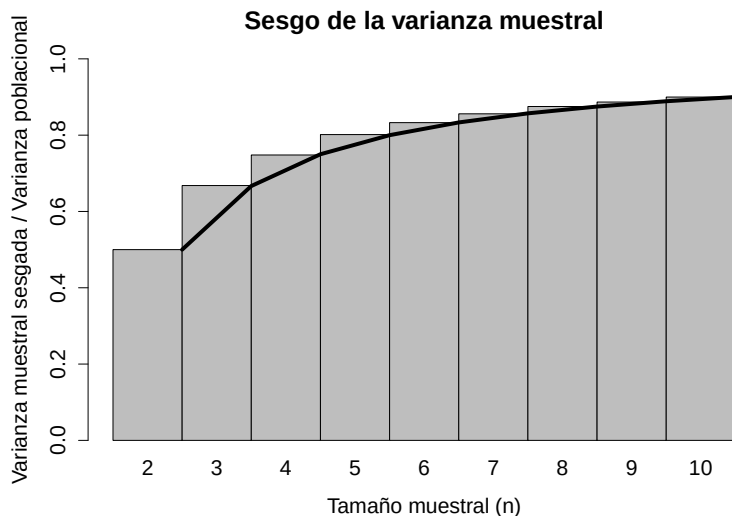
El último paso es la representación gráfica de estos valores. Ya que cada uno de los $N - 1 = 9$ valores se interpreta como una proporción entre 0 y 1, utilizaremos la función `barplot` para que sea un diagrama de barras. Además, superpuesto en el diagrama, mediante la instrucción `lines`, representaremos la línea $(n - 1)/n$.

```
> barplot(V.pond,xlab="Tamaño muestral (n)",
+ ylim=c(0,1), ylab="Varianza muestral sesgada /
+   Varianza poblacional",
+ names.arg=2:10,space=0)
> lines(((2:10)-1)/2:10,lwd=4)
> title("Sesgo de la varianza muestral")
```

El resultado se muestra en la figura VI.3.

Vemos que el gráfico corrobora exactamente el resultado teórico de que la varianza muestral sesgada converge según la

Figura VI.3: Comparación entre el sesgo observado del estimador sesgado de la varianza (barras) y su valor teórico $(n-1)/n$



fórmula $E[\hat{\sigma}_*^2] = \frac{n-1}{n}\sigma^2$. Así, vemos que la primera columna tiene un valor de 0,5, que se corresponde con $E[\hat{\sigma}_*^2] = \frac{2-1}{2} \cdot 1 = 0,5$, y así sucesivamente, hasta que a medida que $n \rightarrow \infty$ el sesgo tiende asintóticamente a 0, ya que $\lim_{n \rightarrow \infty} \frac{n-1}{n} = 1$.

4. Aplicación empírica: análisis del teorema central del límite

4.1. Definición

El teorema central del límite es fundamental para la estadística. De hecho, gran parte de los métodos estadísticos y de inferencia se basan en este teorema para asegurar su validez. Aunque hay muchas versiones de este teorema, en esencia establece que si obtenemos una muestra aleatoria de una población,

entonces la distribución de la media muestral se distribuirá como una normal, independientemente de la distribución de la población original. Así, este teorema nos permite calcular probabilidades para el parámetro media muestral usando una distribución normal estándar $N(0, 1)$.

Formalmente, supongamos que extraemos una muestra aleatoria simple de una distribución cualquiera con media μ y desviación estándar σ definidos. Cada una de estas muestras tendrá n observaciones, de manera que $S_n = X_1 + \dots + X_n$. Si definimos el parámetro de la media muestral ($\bar{X}$) como la media aritmética de las muestras S_n , obtenemos que:

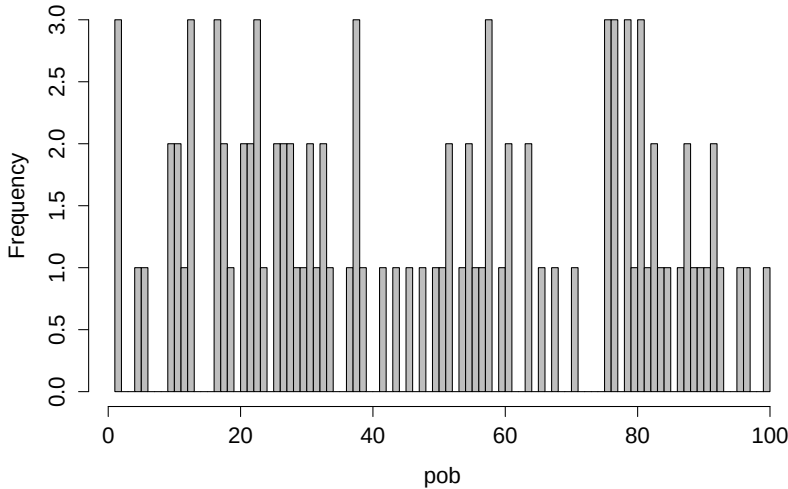
$$\bar{X} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right).$$

Además, el teorema establece que, a medida que el tamaño de las muestras tiende a infinito ($n \rightarrow \infty$), la distribución de $\bar{X}$ convergerá a una normal también en asimetría y curtosis.

4.2. Aproximación de una distribución totalmente aleatoria

Empezaremos este apartado creando unos datos aleatorios cuyo histograma no recuerde a ninguna distribución conocida. Para esto, mediante la función `sample` extraeremos 100 valores aleatorios de la secuencia numérica $1, \dots, 100$ y esta será la población (`pob`). Para comprobar la distribución totalmente aleatoria de esta población de datos, representaremos el histograma en la figura VI.4. Como podemos comprobar, la forma de esta distribución dista mucho de parecerse a una normal.

```
> set.seed(3)
> pob <- sample(1:100, 100, replace=TRUE)
> hist(pob, breaks=80, col="grey", main="")
```

Figura VI.4: Histograma de los datos generados aleatoriamente

Para comprobar el teorema central del límite, realizaremos el siguiente experimento. Extraeremos $K = 10.000$ muestras aleatorias de la población mediante la función `sample`, cada una de ellas con tamaño de muestra de $n = 1, \dots, 4$. De cada una de estas K muestras calculamos la media aritmética ($\bar{x}$) y la almacenamos en un vector al que llamaremos `x.bar`. Esto lo hacemos usando una función iterativa —en inglés conocida como un *for loop*. La pieza de código con estas instrucciones se muestra a continuación.

```
> n <- 4
> K <- 10000

> x.bar <- rep(NA, K)

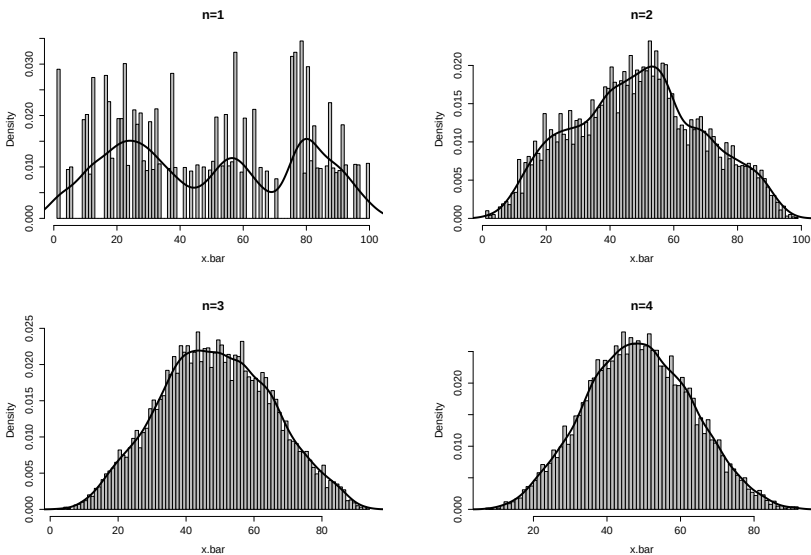
> for (i in 1:K){
+   set.seed(i)
+   v <- sample(pob, n, replace=TRUE)
+   x.bar[i] <- mean(v)
+ }
```

El próximo paso es representar el histograma de cada una de estas muestras de medias muestrales. El siguiente código muestra cómo representar el histograma y la estimación de la función de densidad superpuesta para el caso $n = 4$.

```
> hist(x.bar,breaks=80,col="grey",freq=FALSE,
+      main="n=4")
> lines(density(x.bar),lwd=4)
```

En la figura VI.5 se muestran cuatro histogramas, para $n = 1, \dots, 4$. Esto permite comprobar cómo, a partir de $n = 2$, la distribución de la muestra de medias empieza a aproximarse a una normal, tal y como establece el teorema central de límite.

Figura VI.5: Histogramas del parámetro media muestral para varios valores de tamaño muestral n



Sin embargo, una inspección visual del histograma no basta para determinar que una distribución se aproxima a una normal. Para complementar esta información podemos calcular el coeficiente de asimetría, que para la normal es igual a 0, y el coeficiente de curtosis, que para la normal es igual a 3. Las funciones en R que calculan estos estadísticos están incluidos en la librería **moments** y se denominan **skewness** (asimetría) y **kurtosis** (curtosis). Así, el primer paso será cargar esta librería.

```
> library(moments)
```

Seguidamente llevaremos a cabo una ampliación del experimento anterior. Partiendo de la base que la distribución de la población original tiene una media (μ) y una desviación estándar (σ) definidas, la distribución de las medias muestrales ($\bar{X}$) obtenidas de las K muestras S_n tendrá las siguientes características a medida que $n \rightarrow \infty$:

- $Media(\bar{X}) \rightarrow \mu$
- $Desv.est(\bar{X}) \rightarrow \frac{\sigma}{\sqrt{n}}$
- $Asimetria(\bar{X}) \rightarrow 0$
- $Curtosis(\bar{X}) \rightarrow 3$

Siguiendo con el ejemplo anterior, lo ampliaremos para calcular iterativamente $N = 100$ muestras ($n = 1, \dots, 100$). Para eso, definiremos N y crearemos la matriz **m.x.bar** con dimensión K filas (número de muestras) y N columnas (tamaño muestral). El objetivo será, mediante un doble *for loop*, completar iterativamente esta matriz.

```
> N <- 100
> m.x.bar <- matrix(NA,K,N)
```

A continuación se muestra la pieza de código donde, mediante un doble ciclo, se completa la matriz de medias muestrales.

```
> for (i in 1:K){
+   set.seed(i)
+   for (j in 1:N){
+     v <- sample(pob,j,replace=TRUE)
+     m.x.bar[i,j] <- mean(v)
+   }
+ }
```

Una vez realizado este cálculo, solo hemos de calcular los estadísticos de interés para cada una de las N columnas —esto es, agregando las K filas que componen cada muestra. Esta operación la realizamos usando la función `apply` —recordemos que esta función está explicada en la sección 1.3.6. Los estadísticos de interés son la media (`mean`), la desviación estándar (`sd`), el coeficiente de asimetría (`skewness`) y el coeficiente de curtosis (`kurt`).

```
> mu <- apply(m.x.bar,2,mean)
> sigma <- apply(m.x.bar,2,sd)
> asim <- apply(m.x.bar,2,skewness)
> curt <- apply(m.x.bar,2,kurtosis)
```

El siguiente paso es representar visualmente estos estadísticos. Ya que disponemos de $N = 100$ observaciones para cada uno de ellos, una opción es elaborar cuatro gráficos con un dominio $1 : 100$. R permite calcular gráficos compuestos mediante la función `par`. Así, en el siguiente ejemplo tendremos cuatro gráficos en dos filas y dos columnas mediante

la instrucción `mfrow=c(2,2)`. Los gráficos que introduzcamos a continuación se irán colocando por filas. Una vez obtenido el gráfico, para restaurar los gráficos a sus valores iniciales, concluiremos con la instrucción `dev.off()`, con lo que se borrará el gráfico compuesto.

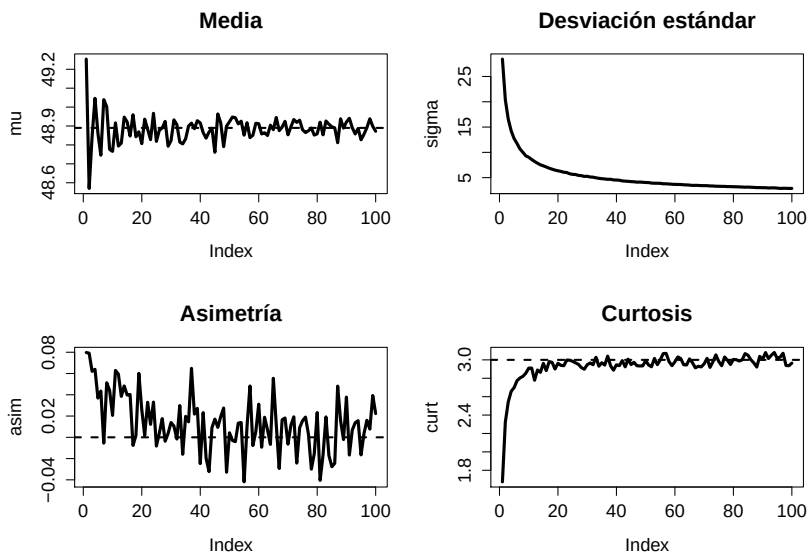
```
> par(mfrow=c(2,2))
> plot(mu,type="l",lwd=3,main="Media")
> abline(h=mean(pob),lty=2,lwd=2)
> plot(sigma,type="l",lwd=3,
+   main="Desviación estándar")
> plot(asim,type="l",lwd=3,main="Asimetría")
> abline(h=0,lty=2,lwd=2)
> plot(curt,type="l",lwd=3,main="Curtosis")
> abline(h=3,lty=2,lwd=2)
> dev.off()
```

La instrucción `dev.off()`

Esta instrucción borra el gráfico elaborado y restablece los valores gráficos por defecto. Si queremos guardar el gráfico, deberemos hacerlo antes de introducir esta última instrucción.

Así, estas instrucciones producen el gráfico compuesto mostrado en la figura VI.6. Como vemos, la tendencia de estos cuatro estadísticos confirman el resultado que el teorema central del límite establece, esto es, $\bar{X} \sim N(\mu, \frac{\sigma}{\sqrt{n}})$.

Al obtener estos gráficos es importante introducir en la consola la instrucción `dev.off()`, para así restablecer los parámetros gráficos a sus valores por defecto. Si no lo hacemos, todos los gráficos que realicemos a partir de entonces tendrán el formato introducido por la función `par`.

Figura VI.6: Parámetros de la distribución de $\bar{X}$ a medida que $n \rightarrow \infty$ 

4.3. Aproximación de diferentes distribuciones a la normal

Para acabar esta sección proponemos una generalización de esta simulación. Esto es, podemos «empaquetar» los cálculos realizados en la simulación previa en una función de creación propia, a la que podemos llamar `tcl`. Esta función solo tendrá el argumento `x`, que será un vector con valores numéricos. El objetivo de la función `tcl` será realizar la simulación hecha anteriormente: extraer de esta población inicial `x` $K = 10.000$ muestras con tamaño $n = 1$, otras K con tamaño $n = 2$, y así hasta $n = N = 100$. De esta colección muestral calcularemos los cuatro estadísticos de interés y los representaremos en un gráfico compuesto, como acabamos de hacer. La propuesta de función `tlc` se presenta a continuación.

```

> tcl <- function(x){
+   library(moments)
+   K <- 10000
+   N <- 100
+   m.x.bar <- matrix(NA,K,N)
+   for (i in 1:K){
+     set.seed(i)
+     for (j in 1:N){
+       v <- sample(x,j,replace=TRUE)
+       m.x.bar[i,j] <- mean(v)
+     }}
+   mu <- apply(m.x.bar,2,mean)
+   sigma <- apply(m.x.bar,2,sd)
+   asim <- apply(m.x.bar,2,skewness)
+   curt <- apply(m.x.bar,2,kurtosis)
+   par(mfrow=c(2,2))
+   plot(mu,type="l",lwd=3,main="Media")
+   abline(h=mean(x),lty=2,lwd=2)
+   plot(sigma,type="l",lwd=3,
+   main="Desviación estándar")
+   plot(asim,type="l",lwd=3,main="Asimetría")
+   abline(h=0,lty=2,lwd=2)
+   plot(curt,type="l",lwd=3,main="Curtosis")
+   abline(h=3,lty=2,lwd=2)
+ }

```

Veamos una aplicación de nuestra función `tcl`. Mediante la función `rpois` generaremos aleatoriamente 10.000 valores de una distribución de Poisson con parámetro $\lambda = 1$ y le daremos a este vector el nombre `pob.2`, como se muestra a continuación.

```

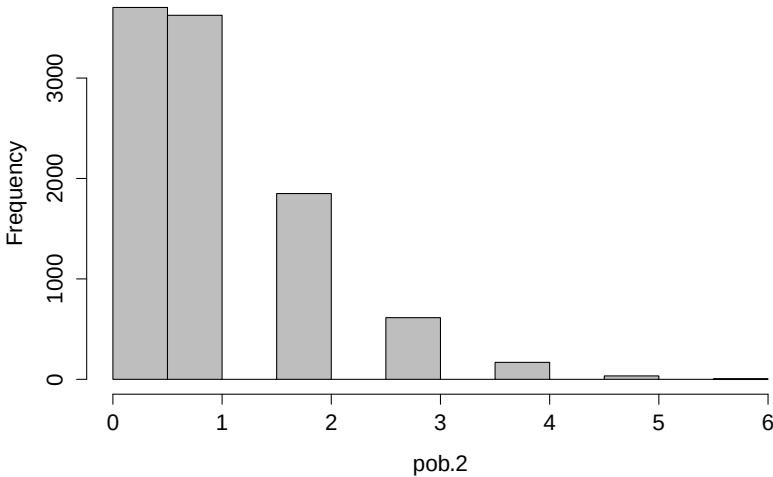
> set.seed(5)
> pob.2 <- rpois(10000,1)

```

El histograma de `pob.2` se muestra en la figura VI.7, donde se puede apreciar la asimetría positiva característica de la distribución de Poisson para valores reducidos de λ .

```
> hist(pob.2,col="grey",main="")
```

Figura VI.7: Histograma de la variable aleatoria $X \sim \text{Poisson}(1)$



La no normalidad de estos datos se puede comprobar calculando los coeficientes de asimetría y curtosis, que confirman que estamos ante unos datos con una distribución asimétrica positiva y leptocúrtica —curtosis mayor que 3.

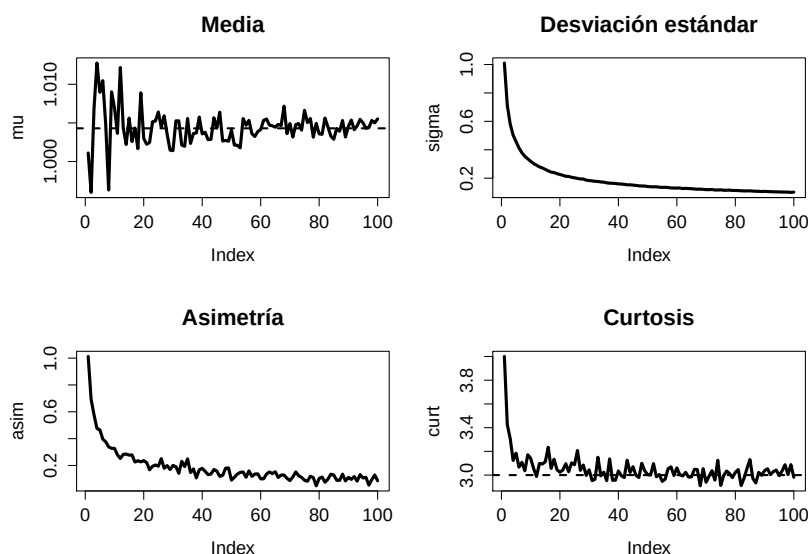
```
> cbind(skewness(pob.2),kurtosis(pob.2))
      [,1]      [,2]
[1,] 0.9991975 3.914486
```

La aplicación de la función `tlc` a estos datos se muestra en la figura VI.8. Podemos apreciar cómo, a medida que $n \rightarrow \infty$, la media fluctúa menos y tiende a su valor poblacional, y la desviación estándar sigue la evolución marcada

por la fórmula $\frac{\sigma}{\sqrt{n}}$. En cuanto a los coeficientes de asimetría y curtosis, estos parten de los valores iniciales de la población de origen —0,999 y 3,914, respectivamente—, para ir aproximándose a los valores de la normal 0 y 3, respectivamente.

```
> tcl(pob.2)
```

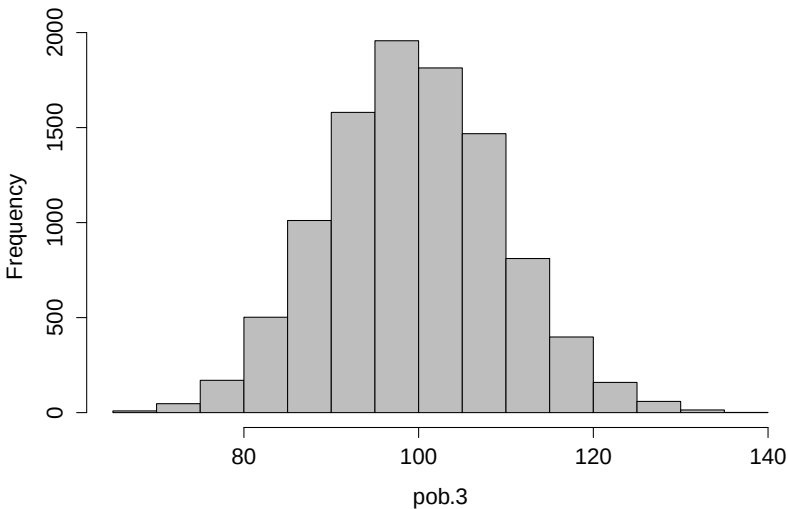
Figura VI.8: Parámetros de la distribución de $\bar{X}$ a medida que $n \rightarrow \infty$. Distribución original $X \sim \text{Poisson}(1)$



Veamos otro ejemplo de generación de datos aleatorios. Un resultado del teorema central del límite establece que, si $X \sim \text{Poisson}(\lambda)$, donde $\lambda > 10$, entonces aproximadamente se cumple que $X \sim N(\lambda, \lambda)$. Generaremos 10.000 valores aleatorios de una distribución de Poisson con parámetro $\lambda = 100$ y representaremos su histograma en la figura VI.9.

```
> set.seed(5)
> pob.3 <- rpois(10000,100)
> hist(pob.3,col="grey")
```

Figura VI.9: Histograma de la variable aleatoria $X \sim \text{Poisson}(100)$



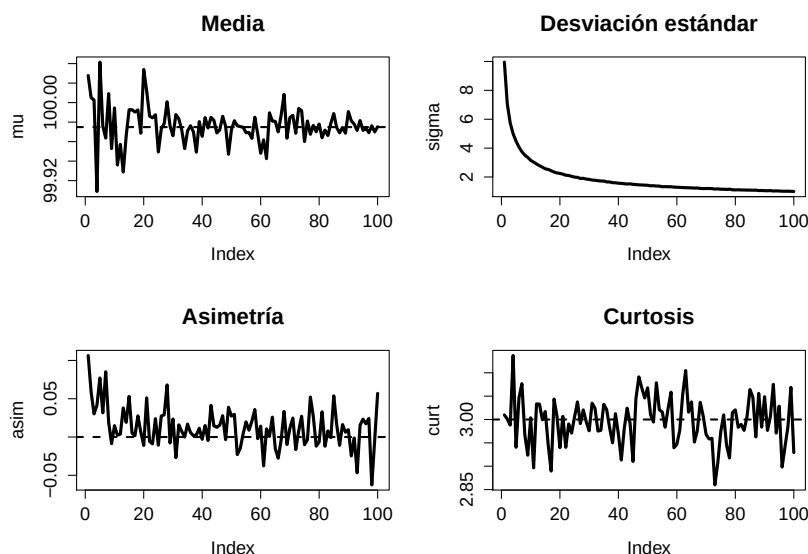
Como se puede observar, el histograma está centrado en el valor $\lambda = 100$, tal y como establece el resultado teórico $X \sim N(\lambda, \lambda)$. Además, comprobamos que los valores de los coeficientes estimados de asimetría y curtosis se aproximan ciertamente a los valores normales.

```
> cbind(skewness(pob.3),kurtosis(pob.3))
      [,1]      [,2]
[1,] 0.09443229 2.969321
```

La aplicación de nuestra función `tcl` corrobora que los coeficientes de asimetría y curtosis ya son aproximadamente normales cuando n es reducido y se mantienen así a medida que $n \rightarrow \infty$. Este resultado se aprecia en la figura VI.10.

```
> tcl(pob.3)
```

Figura VI.10: Parámetros de la distribución de $\bar{X}$ a medida que $n \rightarrow \infty$. Distribución original $X \sim \text{Poisson}(100)$



Bibliografía

Existen algunos manuales dedicados a la simulación estadística, los cuales incluyen metodología avanzada. Por citar algunos de estos, el libro *Introduction to Probability Simulation*

and Gibbs Sampling with R, de E. A. Suess y B. E. Trumbo, cubre la generación de valores aleatorios, la integración por Monte Carlo, el muestro desde modelos de probabilidad aplicada, las cadenas de Markow y elementos de estimación bayesiana. Otro manual destacable es *Introducing Monte Carlo Methods with R*, de C. Robert y G. Casella, en el cual se tratan principalmente la generación de variables aleatorias, la integración y optimización por Monte Carlo, algoritmos de Metropolis-Hastings y la metodología MCMC (*Markow-Chain Monte Carlo*). Por último, una obra muy recomendable para la introducción a los métodos bayesianos con R es *Bayesian computation with R*, que introduce los modelos con uno y varios parámetros, los métodos MCMC (*Markow-Chain Monte Carlo*), la modelización jerárquica y el método de muestreo *Gibbs sampling*.

Bibliografía

Albert, J. (2009). *Bayesian computation with R*. Springer, New York.

Albert, J.; Rizzo, M. (2012). *R by Example*. Springer, New York.

Bloomfield, V. A. (2014). *Using R for Numerical Analysis in Science and Engineering*. Chapman and Hall/CRC.

Castillo, A. J. S. (2010). *Métodos Estadísticos con R y R-Commander*.

Cohen, Yosef; Cohen, Jeremiah Y. (2008) *Statistics and Data with R: An applied approach through examples*. Wiley.

Cook, D.; Swayne, D. F. (2007). *Interactive and dynamic graphics for data analysis: with R and GGobi*. Nueva York: Springer.

Cowpertwait, P. S.; Metcalfe, A. V. (2009). *Introductory time series with R*. Nueva York: Springer.

Crawley, M. J. (2005). *Statistics: an introduction using R*. Nueva York: Wiley.

Crawley, M. J. (2012). *The R book*. Chichester: John Wiley & Sons Ltd.

Croissant, Y.; Millo, G. (2008). «Panel data econometrics in R: The plm package». *Journal of Statistical Software* (vol. 27(2), págs. 1-43).

Dalgaard, P. (2008). *Introductory statistics with R*. Nueva York: Springer.

Everitt, B. S.; Hothorn, T. (2011). *An introduction to applied multivariate analysis with R*. Nueva York: Springer.

Faraway, J. J. (2004). *Linear models with R*. CRC Press.

Faraway, J. J. (2005). *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. CRC press.

Højsgaard, S.; Edwards, D.; Lauritzen, S. (2012). *Graphical models with R*. Nueva York: Springer.

Jackson, C. H. (2011). «Multi-state models for panel data: the msm package for R». *Journal of Statistical Software* (vol. 38(8), págs. 1-29).

Jones, O.; Maillardet, R.; Robinson, A. (2009). *Introduction to scientific programming and simulation using R*. CRC Press.

Kleiber, C.; Zeileis, A. (2008). *Applied Econometrics with R*. Nueva York: Springer.

Petris, G.; Campagnoli, P.; Petrone, S. (2009). *Dynamic linear models with R*. Nueva York: Springer.

Pfaff, B. (2008). *Analysis of integrated and cointegrated time series with R*. Nueva York: Springer.

Robert, C.; Casella, G. (2010). *Introducing Monte Carlo Methods with R*. Nueva York: Springer..

Sarkar, D. (2008). *Lattice: multivariate data visualization with R*. Nueva York: Springer.

Soetaert, K.; Petzoldt, T.; Setzer, R. W. (2010). «Solving differential equations in R: Package deSolve». *Journal of Statistical Software* (vol. 33(9), págs. 1-25).

Soetaert, K.; Cash, J.; Mazzia, F. (2012). *Solving Differential Equations in R*. SNueva York: Springer.

Spector, P. (2008). *Data manipulation with R*. Nueva York: Springer.

Stasinopoulos, D. M.; Rigby, R. A. (2007). «Generalized additive models for location scale and shape (GAMLSS) in R». *Journal of Statistical Software* (vol. 23(7), págs. 1-46).

Suess, E. A.; Trumbo, B. E. (2010). *Introduction to Probability Simulation and Gibbs Sampling with R*. Nueva York: Springer.

Verzani, J. (2004). *Using R for introductory statistics*. CRC Press.

Vinod, H. D. (2011). *Hands-on Matrix Algebra Using R: Active and Motivated Learning with Applications*. World Scientific.

Wickham, H. (2009). *Ggplot2: elegant graphics for data analysis*. Nueva York: Springer.

White, H. (1980). «A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity». *Econometrica: Journal of the Econometric Society* (págs. 817-838).

Zuur, A. F.; Ieno, E. N.; Meesters, E. H. (2009). *A Beginner's Guide to R*. Nueva York: Springer.

Son numerosos los programas informáticos de estadística que se caracterizan por funcionar mediante menús desplegables. Esto ayuda al estudiante o al investigador, pero supone, a su vez, una limitación. El análisis numérico y estadístico mediante código ha estado tradicionalmente limitado a informáticos y ha supuesto una barrera de entrada a los demás.

Con el paquete estadístico R, esto no tiene por qué ser así. Este manual se ofrece como una completa introducción al entorno R, acompañando al lector mediante sencillos ejemplos de cómo programar un análisis numérico con este potente paquete estadístico, y cubre temas de estadística, matemáticas y econometría.

Con este libro aprenderás sobre:

- ✓ R; ✓ matemáticas; ✓ estadística; ✓ econometría; ✓ programación;
- ✓ análisis numérico; ✓ análisis matemático; ✓ análisis cuantitativo;
- ✓ análisis descriptivo; ✓ optimización; ✓ cálculo; ✓ modelos de regresión;
- ✓ representación gráfica de datos; ✓ inferencia estadística; ✓ simulación

Daniel Liviano Solís

*Doctor en Organización Industrial por la URV.
Profesor de los Estudios de Economía y Empresa
de la UOC. Forma parte del Área de Business
Intelligence y Big Data de la UOC.*

Maria Pujol Jover

*Doctora en Estudios Empresariales por la UB.
Profesora de los Estudios de Economía y Empresa
y del Área de Business Intelligence y Big Data de la UOC.*

