como aumentar la probabilidad de que recomendaciones basadas en él sean aceptadas. Una buena documentación facilita las modificaciones y asegura que el modelo puede ser usado o algunas de sus partes reutilizadas, aun si no están presentes sus desarrolladores.

1.6. ANÁLISIS DE DATOS: INTRODUCCIÓN A R

El análisis de los datos es una parte esencial en un proyecto de simulación. Por una parte es preciso analizar los datos del sistema bajo estudio, con el fin de modelar su comportamiento. Por otra parte es preciso analizar los datos obtenidos como resultado de la simulación del modelo, con el fin de extraer conclusiones.

Como ya se ha indicado anteriormente, algunos entornos de simulación incorporan herramientas de ayuda para el análisis de los datos. Estas herramientas típicamente permiten representar los datos gráficamente (por ejemplo, mediante gráficas X-Y, boxplots, histogramas y gráficas Q-Q) y realizar ajustes (por ejemplo, ajuste de distribuciones de probabilidad y superficies de respuesta). Sin embargo, un buen número de entornos de simulación no proporcionan estas ayudas, limitándose a facilitar la descripción y simulación del modelo. En este segundo grupo se encuentran la mayoría de los entornos y herramientas de simulación gratuitas, las cuales deben por tanto usarse en combinación con alguna herramienta para el análisis y modelado de los datos.

Existe una amplia variedad de herramientas especializadas en el análisis y modelado de los datos. En esta sección se presenta una de ellas: el lenguaje R y la plataforma software que lo soporta. Se trata de software gratuito y de código abierto, con versiones para los sistemas operativos Windows, Mac OS X y Linux. Puede descargarse del Comprehensive R Archive Network (CRAN), que está ubicado en `http://cran.r-project.org/`.

R es un lenguaje interpretado, en el cual se distingue entre letras mayúsculas y minúsculas. Es posible ir introduciendo los comandos uno a uno en la línea de comandos de la consola, tras el símbolo del sistema (>), o bien ejecutar un conjunto de comandos escritos en un fichero. En la Figura 1.13 se muestra un ejemplo de la consola de R en Windows.

El lenguaje R tiene una gran variedad de tipos de datos, incluyendo vectores, matrices, data frames y listas. La mayor parte de la funcionalidad se consigue mediante el uso de funciones, tanto las proporcionadas por el lenguaje como las definidas por el propio usuario. Algunas funciones básicas están disponibles por

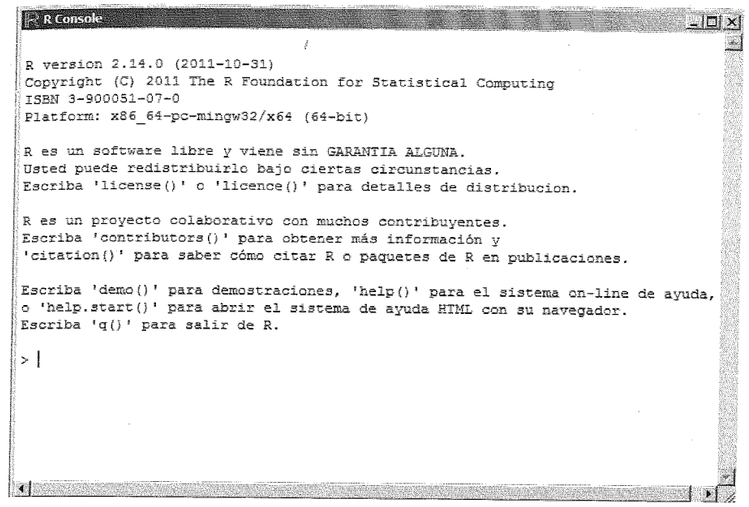

Figura 1.13: Ejemplo de la consola de R en Windows.

defecto. Otras se encuentran en paquetes, que deben ser abiertos (esta acción se denomina *attach*) para poder usar las funciones que contienen.

Las sentencias consisten en funciones y asignaciones. R usa el símbolo `<-` para las asignaciones, en lugar del típico símbolo `=`. Por ejemplo,

```
> x <- rnorm(10)
```

crea un objeto de tipo vector llamado `x`, que contiene 10 observaciones de la distribución normal estándar.

Los comentarios son precedidos por el símbolo `#`. El intérprete de R ignora todo el texto que aparezca tras el símbolo `#`. Para finalizar una sesión debe ejecutarse la función `q()`.

La función `c()` convierte sus argumentos en un vector o una lista. Por ejemplo, supongamos que se realizan 5 réplicas independientes de la simulación del modelo de la oficina, obteniéndose los siguientes 5 valores del tiempo medio de espera en cola: 1.2, 2.4, 1.4, 2.2 y 3.2 minutos. La primera de las siguientes dos sentencias crea un objeto del tipo vector llamado `tCola`, cuyos componentes son esos cinco valores. La segunda sentencia dibuja el valor del componente del vector frente al índice. En la Figura 1.14 se muestra el gráfico.

```
> tCola <- c(1.2, 2.4, 1.4, 2.2, 3.2)
> plot(tCola)
```

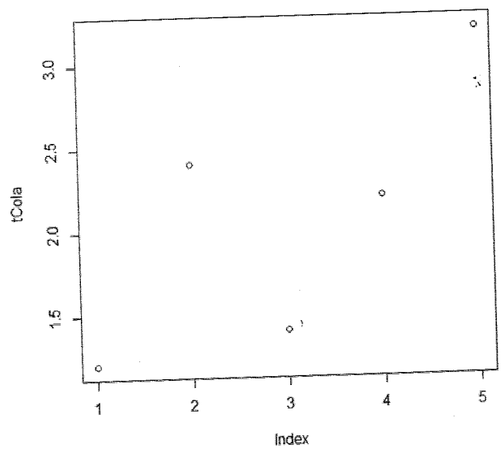

Figura 1.14: Ejemplo de uso de la función *plot*.

La función `help()` permite acceder a la documentación de las funciones. Por ejemplo,

```
> help(plot)
```

abre una ventana en la cual se muestra la documentación para la función `plot`.

R incluye demos que ilustran algunas de sus capacidades para la representación gráfica. Por ejemplo,

```
> demo(graphics)
```

arranca una de ellas. Otras dos demos relacionadas con las capacidades gráficas se arrancan mediante: `demo(persp)` y `demo(image)`. Puede obtenerse la lista completa de demos ejecutando:

```
> demo()
```

1.6.1. El espacio de trabajo

El **espacio de trabajo** es el espacio de memoria en el cual se guardan todos los objetos creados durante la sesión. Es posible salvar el espacio de trabajo al finalizar la sesión (R pregunta antes de cerrarse si debe hacerlo), de modo que automáticamente sea vuelto a cargar la siguiente vez que se arranque R. Puede obtenerse una lista de todos los objetos del espacio de trabajo mediante:


```
> ls()
```

La función `rm()` elimina del espacio de trabajo los objetos que se le pasan como argumento. Por ejemplo,

```
> rm(x)
```

elimina del espacio de trabajo el objeto llamado `x`.

El **directorio de trabajo** es donde R salva por defecto los resultados y también de donde intenta por defecto leer los ficheros. Para saber cuál es el directorio de trabajo hay que ejecutar

```
> getwd()
```

Es recomendable separar en diferentes directorios los diferentes proyectos. La función `setwd()` permite modificar el directorio de trabajo, pasando como argumento a la función el nombre del nuevo directorio escrito entre comillas. Por ejemplo:

```
> setwd("F:/Simulacion")
```

La función `save.image` salva el espacio de trabajo completo a un fichero. Para salvar únicamente algunos objetos puede emplearse `save()`. La función `load()` carga en la sesión actual el espacio de trabajo almacenado en el fichero que se le pasa como argumento. Si se desea acceder a un fichero que no se encuentra en el directorio de trabajo, es necesario especificar el path completo en la llamada.

R guarda memoria de los comandos ejecutados durante la sesión. Con las flechas del teclado es posible moverse por la historia de comandos. Las funciones `savehistory()` y `loadhistory()` permiten salvar a fichero los comandos de la sesión y cargar dicha historia desde un fichero.

Es posible escribir los comandos en un fichero de texto y cargar este **fichero de comandos** desde R, de manera que se ejecuten en secuencia todos los comandos escritos en él. Por convenio, el nombre del fichero con los comandos (también llamado *fichero script*) se escribe con extensión .R. La función `source()` permite cargar y ejecutar el fichero. Por ejemplo,

```
> source("F:/Simulacion/script1.R")
```

carga y ejecuta el fichero de comandos llamado *script1.R*. En la práctica suele resultar más cómodo trabajar empleando ficheros de comandos que hacerlo escribiendo directamente las sentencias en la consola de R.

1.6.2. Estructuras de datos

El primer paso en el análisis de los datos es crear una estructura de datos que contenga los datos a estudiar. Los datos pueden ser cargados en la estructura de datos bien manualmente o bien pueden ser importados desde una fuente externa. R tiene básicamente cinco tipos diferentes de estructura de datos: vector, matriz, array, data frame y lista. Los vectores, matrices y arrays contienen números y tienen una, dos o más dimensiones.

Un data frame es una tabla bidimensional, pero más general que una matriz, dado que en el data frame el tipo de los datos almacenados en una columna puede diferir del tipo de los datos almacenados en otra. Los datos almacenados en una misma columna deben ser del mismo tipo: cadena de caracteres, numérico o lógico (TRUE, FALSE).

Las sentencias siguientes crean un data frame llamado `exp1` con tres columnas (las columnas del data frame son denominadas sus *variables*):

```
> numOper <- c(1,1,2,2)
> horario <- c("std", "nuevo", "std", "nuevo")
> tCola   <- c(12.3, 6.1, 5.2, 2.5)
> exp1    <- data.frame(numOper, horario, tCola)
```

Escribiendo el nombre de un objeto, R muestra su valor. En este caso, escribiendo `exp1` se obtiene:

```
> exp1
  numOper horario tCola
1       1     std  12.3
2       1   nuevo   6.1
3       2     std   5.2
4       2   nuevo   2.5
```

Otra forma de introducir o modificar los datos del data frame es mediante la función `edit()`. Para ello, en primer lugar debe crearse el objeto vacío, especificando el nombre y tipo de las variables. A continuación, se invoca el editor de texto mediante la función `edit()` y se introducen los datos empleando dicho editor. Por ejemplo:

```
> exp2 <- data.frame( numOper = numeric(0),
                      horario = character(0),
                      tCola   = numeric(0) )
> exp2 <- edit(exp2)
```


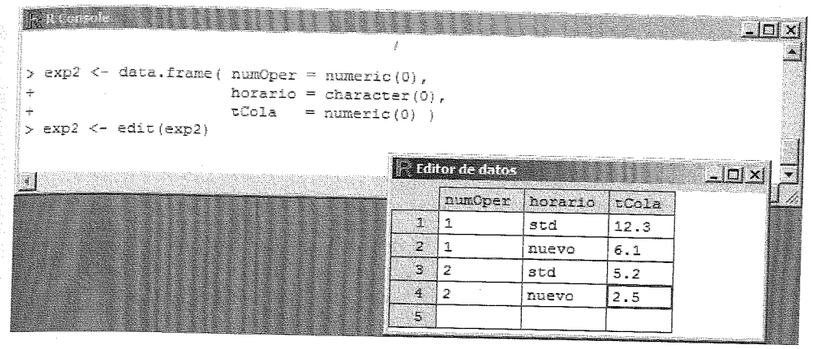

Figura 1.15: Ejemplo de uso de la función *edit* para editar el contenido de un data frame.

En la Figura 1.15 se muestra el aspecto de la ventana de edición una vez se han introducido en ella los datos.

Otra forma de introducir datos en el data frame es importarlos desde un fichero de texto. Para ello puede emplearse la función `read.table()`. Supongamos que el contenido del fichero *data.txt* es:

```
numOper horario tCola
1       std     12.3
1       nuevo   6.1
2       std     5.2
2       nuevo   2.5
```

Obsérvese que en la primera fila se han escrito los nombres de las variables. En las siguientes filas están los valores separados por un delimitador, que en este caso es el espacio en blanco (puede usarse cualquier otro símbolo como delimitador).

Mediante las dos siguientes sentencias se crea un data frame llamado `exp3`, se cargan en él valores contenidos en el fichero *data.txt* y a continuación se muestra el contenido de `exp3`. Si el fichero con los datos no se encuentra en el directorio de trabajo, debe especificarse el path completo. Mediante `header=TRUE` se indica que la primera fila del fichero contiene los nombres de las variables.

```
> exp3 <- read.table("data.txt", header=TRUE)
> exp3
  numOper horario tCola
1       1     std  12.3
2       1   nuevo   6.1
3       2     std   5.2
4       2   nuevo   2.5
```

Puede accederse a una variable del data frame indicando su número de orden. La primera variable es la columna 1, la segunda la 2 y así sucesivamente. Otra posibilidad es escribir el nombre del data frame, seguido del símbolo `$` y del nombre de la variable. Por ejemplo, las dos siguientes sentencias son equivalentes:

```
> exp1[,3]
[1] 12.3  6.1  5.2  2.5
> exp1$tCola
[1] 12.3  6.1  5.2  2.5
```

De forma similar puede accederse a los elementos de cada variable. Por ejemplo, mediante estas dos sentencias se accede al cuarto elemento de la variable `tCola` (los índices en R comienzan en el valor 1):

```
> exp1$tCola[4]
[1] 2.5
> exp1[4,3]
[1] 2.5
```

Las siguientes funciones facilitan la manipulación de los datos. La función `pretty()` se emplea a menudo para dibujar gráficos.

`length(x)`	Devuelve la longitud del objeto `x`.
`seq(desde,hasta,paso)`	Devuelve una secuencia.
`rep(x,n)`	Repite `x` `n` veces.
`pretty(x,n)`	Divide `x` en aproximadamente `n` intervalos iguales, de modo que los puntos de división tengan valores redondeados. Devuelve los puntos de división.

Los siguientes son algunos ejemplos de uso de estas funciones:

```
> x <- seq(1,10,2)
> x
[1] 1 3 5 7 9
> length(x)
[1] 5
> y <- rep(x,2)
> y
[1] 1 3 5 7 9 1 3 5 7 9
> pretty(c(1,5),10)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```


1.6.3. Gráficos

La representación gráfica de los datos es una herramienta fundamental para su análisis. En ocasiones representar gráficamente los datos permite detectar patrones o anomalías que difícilmente pueden ser detectadas mediante análisis estadísticos numéricos. Por su potencia y sencillez, la mayoría de las técnicas de análisis descritas en este texto son técnicas gráficas.

R proporciona excelentes recursos para la construcción de gráficos. La construcción del gráfico se realiza ejecutando varias sentencias, cada una de las cuales va añadiendo nuevas características al gráfico y definiendo su aspecto. Se muestra un ejemplo a continuación. Supongamos un data frame llamado `datos` cuyo contenido es:

```
> datos
   x    y
1 1.2  2.4
2 2.3  6.4
3 1.4  2.8
4 3.7 10.1
5 1.2  2.0
6 0.6  1.0
```

La primera de las siguientes tres sentencias dibuja la variable `x` frente a `y`. En la sentencia se especifican las etiquetas de ambos ejes (`xlab`, `ylab`) y el rango de valores de cada eje (`xlim`, `ylim`). La segunda sentencia añade la línea correspondiente al ajuste de los datos y la tercera añade el título.

```
> plot(datos$x, datos$y, xlab="x", ylab="y", xlim=c(0,4), ylim=c(0,12))
> abline( lm(datos$y ~ datos$x) )
> title("Representación y vs x")
```

La gráfica obtenida se muestra en la Figura 1.16. Es posible configurar otras características del gráfico, como son su tamaño, los símbolos usados para representar los valores y su color, el tipo y color de las líneas, el tamaño, color y fuente del texto de los ejes, del título y del subtítulo, las marcas de los ejes, etc. Asimismo, es posible añadir etiquetas al gráfico y superponer varios gráficos.

Otros tipos de gráficos útiles para el análisis de los datos son el histograma y el boxplot. Las funciones `hist()` y `boxplot()` dibujan el histograma y el boxplot de los datos pasados como argumento.

La función `dev.new()` abre una nueva ventana gráfica sobre la cual es posible construir un nuevo gráfico.

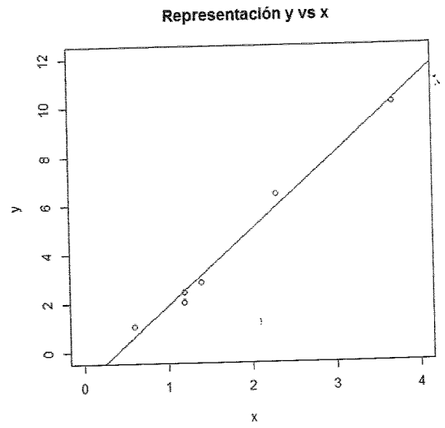

Figura 1.16: Ejemplo de construcción de un gráfico X-Y.

La función `par()` permite asignar valor a los parámetros del gráfico. Puede usarse para asignar valor al parámetro `mfrow`, que define el número de filas y columnas de gráficos que van a representarse en la ventana de dibujo. Por ejemplo,

```
> par( mfrow(2,3) )
```

prepara la ventana de dibujo para que aloje 6 gráficos, dispuestos formando una matriz de 2 filas y 3 columnas.

A continuación se muestra un ejemplo. Se crea un objeto de tipo vector llamado `x`, en el cual se guardan 500 observaciones independientes de una distribución exponencial de media 1, y se grafica de tres maneras diferentes: grafico X-Y, histograma y boxplot. Las tres gráficas obtenidas se muestran en la Figura 1.17.

```
> x <- rexp(500)
> par(mfrow=c(1,3))
> plot(x)
> hist(x)
> boxplot(x)
```

El **boxplot** es una herramienta de análisis gráfico muy útil cuando se desea comparar la distribución de diferentes grupos de datos. En la Figura 1.18 se muestra nuevamente el boxplot que aparece en la Figura 1.17, indicando en esta ocasión cómo ha sido construido a partir de los datos.

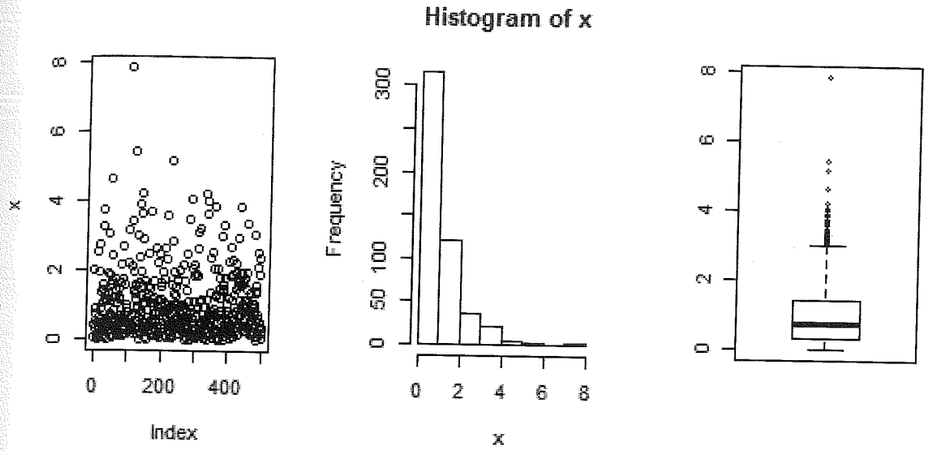

Figura 1.17: Ejemplo de construcción de un gráfico X-Y, un histograma y un boxplot.

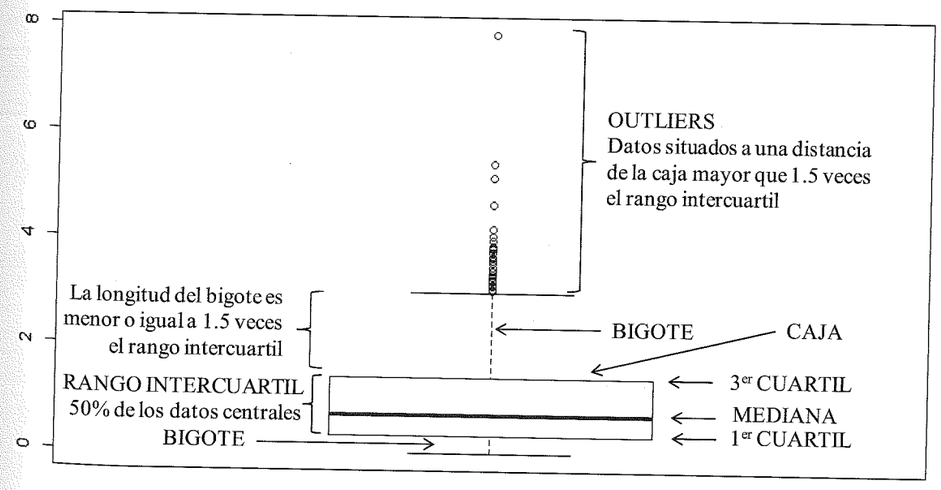

Figura 1.18: Construcción del boxplot de un grupo de datos.

1.6.4. Manejo básico de los datos

Sentencias del tipo

```
> variable <- expresion
```

permiten crear nuevas variables y transformar las existentes. La expresión puede contener operadores y funciones. También es posible añadir nuevas variables a un data frame. Por ejemplo, si el data frame `data` tiene dos variables `x` e `y`

```
> data <- data.frame(x=c(1,2),y=c(10,20))
> data
  x  y
1 1 10
2 2 20
```

mediante la sentencia

```
> data$media <- (data$x+data$y)/2
```

se crea una nueva variable llamada `media` en el data frame. Cada elemento de la nueva variable es la media de los correspondientes elementos de las otras dos variables.

```
> data
  x  y media
1 1 10   5.5
2 2 20  11.0
```

Una sentencia del tipo

```
> variable[condicion] <- expresion
```

sólo realiza la asignación si la condición vale TRUE. Por ejemplo, puede incluirse una nueva variable en el data frame que indique si la media es mayor o menor que 10.

```
> data$criterio10 [data$media < 10]  <- "menor"
> data$criterio10 [data$media == 10] <- "igual"
> data$criterio10 [data$media > 10]  <- "mayor"
> data
  x  y media criterio10
1 1 10   5.5      menor
2 2 20  11.0      mayor
```

Igualmente,

```
> variable1 <- variable2[condicion]
```


crea el objeto `variable1` y copia en él los elementos de `variable2` que satisfacen la condición. Por ejemplo:

```
> x1 <- data$x[data$criterio10 == "mayor"]
> x1
[1] 2
```

Las expresiones lógicas en R pueden tomar dos valores: TRUE y FALSE. Los operadores de comparación son `<`, `<=`, `>`, `>=`, `==` y `!=`. Las operaciones lógicas se representan: `!x` (not x), `x | y` (x or y), `x & y` (x and y).

La función `order()` permite ordenar los elementos de una estructura de datos. Puede por ejemplo aplicarse a un data frame de la forma siguiente (se ordenan las filas de `exp3` tomando como criterio el valor de la variable `tCola`):

```
> exp3
  numOper horario tCola
1       1     std  12.3
2       1   nuevo   6.1
3       2     std   5.2
4       2   nuevo   2.5
> exp3[order(tCola),]
  numOper horario tCola
4       2   nuevo   2.5
3       2     std   5.2
2       1   nuevo   6.1
1       1     std  12.3
```

Las funciones `merge()`, `cbind()` y `rbind()` permiten realizar la unión de los datos de dos data frames.

1.6.5. Valor NA (Not Available)

El valor `NA` significa Not Available (No Disponible). Se emplea para indicar que no se dispone de ese dato. Supongamos por ejemplo un vector de 5 elementos definido de la forma:

```
> x <- c(1:5)
> x
[1] 1 2 3 4 5
```

Si se asigna valor a un elemento del vector que no existe, R aumenta automáticamente el tamaño del vector rellenando con valores NA. Por ejemplo:

```
> x[8] <- 8
> x
[1]  1  2  3  4  5 NA NA  8
```

La función `is.na()` acepta un objeto como argumento y devuelve un objeto del mismo tamaño con las entradas reemplazadas por TRUE si el elemento es NA y por FALSE si no lo es. Por ejemplo:

```
> x
[1]  1  2  3  4  5 NA NA  8
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
> x[ !is.na(x) ]
[1] 1 2 3 4 5 8
```

Si alguno de los operandos es NA, el resultado de la operación aritmética es también NA. Lo mismo sucede cuando se pasa el valor NA como argumento a una función. Esto debe ser tenido en cuenta al operar sobre conjuntos de datos: los valores NA deben ser excluidos del análisis. Con este fin, la mayor parte de las funciones tienen un parámetro llamado `na.rm`, tal que si se le asigna el valor TRUE los valores NA son eliminados de los datos pasados como argumento a la función. Por ejemplo, la función `sum` realiza la suma de los datos contenidos en el objeto que es pasado como argumento (puede consultarse su documentación ejecutando `help(sum)`):

```
> x
[1]  1  2  3  4  5 NA NA  8
> sum(x)
[1] NA
> sum(x, na.rm=TRUE)
[1] 23
```

La función `na.omit()` devuelve un objeto en el cual se han eliminado todos los valores NA. Cuando se pasa como argumento un data frame, el objeto devuelto es un data frame en el cual se han eliminado todas las filas que contienen algún valor NA.

1.6.6. Conversión del tipo de datos

R proporciona funciones para identificar el tipo de dato de un objeto y convertirlo a un tipo diferente. Las funciones cuyo nombre tiene la forma `is.tipodato()` devuelven TRUE o FALSE dependiendo de que el objeto pasado como argumento sea o no del tipo de dato. Por el contrario, las funciones cuyo nombre es de la forma

`as.tipodato()` realizan la conversión del argumento al tipo de dato. En la Tabla 1.1 se muestran las funciones disponibles.

Tabla 1.1: Funciones para la comprobación y conversión de tipos de datos.

Comprobación	Conversión
`is.numeric()`	`as.numeric()`
`is.character()`	`as.character()`
`is.vector()`	`as.vector()`
`is.matrix()`	`as.matrix()`
`is.data.frame()`	`as.data.frame()`
`is.factor()`	`as.factor()`
`is.logical()`	`as.logical()`

1.6.7. Control del flujo

R tiene sentencias para controlar el flujo de ejecución. La sentencias *for* y *while* permiten definir bucles, mientras que las sentencias *if-else* y *switch* permiten especificar que una sentencia se ejecute sólo cuando se satisface determinada condición.

La sentencia *for* tiene la siguiente sintaxis:

```
> for (var in secuencia) sentencia
```

Si el cuerpo del bucle consta de varias sentencias, debe escribirse entre llaves (`{ }`). La función `length()` devuelve el número de elementos del objeto que se le pasa como argumento. El siguiente ejemplo ilustra el uso de dicha función y del bucle *for*.

```
> x <- c(1,2,3,5,7,11)
> y <- numeric(0)
> k <- numeric(0)
> for (i in 1:length(x)) {
      y[i] <- i*x[i]
      k[i] <- y[i]/2
  }
> y
[1]  1  4  9 20 35 66
> k
[1]  0.5  2.0  4.5 10.0 17.5 33.0
```

La sentencia *if-else* permite definir que una o varias sentencias sean ejecutadas sólo si se satisface determinada condición. La sintaxis es:

```
> if (condición) sentencia
> if (condición) sentencia1 else sentencia2
```

1.6.8. Definición de funciones

R proporciona las funciones matemáticas más habituales, tales como `abs(x)`, `sqrt(x)`, `ceiling(x)` (entero más pequeño no menor que `x`), `floor(x)` (mayor entero no mayor que `x`), `trunc(x)` (entero formado truncando a cero los decimales de `x`), `round(x,digits=n)` (redondea `x` al número de dígitos decimales especificado), `signif(x,digits=n)` (redondea `x` al número de dígitos especificado, incluyendo dígitos enteros más decimales), `sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`, `acos(x)`, `atan(x)`, `sinh(x)`, `cosh(x)`, `tanh(x)`, `asinh()`, `acosh()`, `atanh()`, `log(x,base=n)`, `log(x)` (logaritmo en base e), `log10(x)` (logaritmo en base 10) y `exp(x)`.

El usuario puede definir sus propias funciones. La sintaxis para ello es la siguiente:

```
> nombreFunción <- function(arg1, arg2, ...) {
      sentencias
      return(objeto)
  }
```

Los objetos creados dentro del cuerpo de la función son locales a la función. El objeto devuelto por la función puede ser cualquier tipo de dato. La función se invoca de la misma forma que las demás funciones definidas en R: especificando su nombre y escribiendo a continuación los argumentos entre paréntesis, separados por coma.

Aquí concluye esta breve introducción al lenguaje R. En los siguientes temas irán mostrándose otras capacidades del lenguaje.

1.7. LECTURAS RECOMENDADAS

Las definiciones de sistema y modelo dadas en la Sección 1.2.1 han sido extraídas de (Cellier 1991). En este excelente libro pueden encontrarse otras definiciones de estos términos, así como una discusión más elaborada acerca de en qué situaciones resulta imposible o inconveniente experimentar con el sistema real, y las ventajas que tiene en estos casos el empleo de modelos.

La clasificación de los modelos en cuatro tipos (mental, verbal, físico y matemático) está explicada con más detalle en (Ljung & Torkel 1994). En la Sección 1.2.4

se presentaron algunas clasificaciones de los modelos matemáticos. En (Cellier 1991, Ljung & Torkel 1994) puede encontrarse una discusión más extensa acerca de las distintas clasificaciones de los modelos matemáticos.

En la Sección 1.2.2 se describe una clasificación en cuatro niveles del conocimiento que puede poseerse acerca de un sistema. Puede encontrarse una descripción más detallada en (Klir 1985, Zeigler et al. 2000). En el texto (Zeigler et al. 2000) se propone una clasificación similar, que está enfocada al modelado y la simulación. La clasificación de Zeigler consta de 5 niveles de conocimiento: (0) marco de observación; (1) comportamiento de entrada/salida; (2) función de entrada/salida; (3) transición de estado; y (4) componentes acoplados.

El marco formal para el modelado y la simulación descrito en la Sección 1.2.3 está extraído de (Zeigler et al. 2000), donde puede encontrarse una explicación más detallada.

En la Sección 1.3.3 se han introducido algunos conceptos básicos acerca de los autómatas celulares. En (Wolfram 1986) se investigan sistemáticamente todas las posibles funciones de transición en autómatas celulares unidimensionales. Puede encontrarse una descripción más detallada del Juego de la Vida en (Gardner 1970).

En (Tyszer 1999) se explica en uso de C++ en la programación de simulaciones de eventos discretos. Este texto complementa las explicaciones dadas sobre la programación de modelos basados en planificación de eventos.

Los pasos que típicamente se siguen en un estudio de simulación están descritos con mayor detalle en (Pedgen et al. 1995, Law & Kelton 2000). Se recomienda acudir a esas referencias para ampliar las explicaciones de la Sección 1.5. Asimismo, en (Bratley et al. 1987) y (Hoover & Perry 1989) puede encontrarse abundante información acerca de lenguajes de simulación como pueden ser Simscript, GPSS, SIMAN y Simula.

Finalmente, existe gran cantidad de documentación en Internet y excelentes referencias sobre el lenguaje R y también sobre el lenguaje S, en el cual se basa R. Excelentes textos sobre análisis estadístico en S y S-PLUS son (Venables & Ripley 1997) y (Becker et al. 1988). Los dos siguientes textos sobre R son excepcionales por la claridad de sus explicaciones: (Kabacoff 2011) y (Maindonald & Braun 2010).

1.8. EJERCICIOS DE AUTOCOMPROBACIÓN

Ejercicio 1.1

Describa una forma de estudiar cada uno de los sistemas siguientes, en términos de las posibilidades mostradas en la Figura 1.1.

1. Un ecosistema compuesto por varias especies (animales y vegetales) y por recursos (agua, luz, etc.).
2. Una glorieta en la que convergen varias calles y que frecuentemente presenta atascos.
3. Una presa para el suministro de agua y electricidad que se planea construir en un río.
4. El servicio de urgencias de un hospital que se encuentra en funcionamiento.
5. Un circuito eléctrico.

Ejercicio 1.2

Describa qué características tiene cada uno de los tipos de modelo siguientes: mental, verbal, físico y matemático. Ponga un ejemplo de modelo de cada tipo, indicando cuál es su finalidad.

Ejercicio 1.3

Para cada uno de los sistemas mencionados en el Ejercicio 1.1, suponga que se ha decidido realizar el estudio mediante simulación por ordenador. Discuta de qué tipo podría ser, en su opinión, el modelo matemático: estático o dinámico, determinista o estocástico, de tiempo continuo, de tiempo discreto, eventos discretos o híbrido.

Ejercicio 1.4

Plantee un ejemplo de conocimiento de un sistema en cada uno de los cuatro niveles de la clasificación de Klir.

Ejercicio 1.5

Para un sistema de su elección, plantee un ejemplo de análisis del sistema, inferencia sobre el sistema y diseño del sistema. Indique el nivel en el conocimiento del sistema que se precisa, según la clasificación de Klir, para realizar cada una de estas tres actividades.

Ejercicio 1.6

Ejecute manualmente el algoritmo que realiza la simulación del modelo representado por la tabla de transición/salidas siguiente:

Estado actual	Entrada actual	Estado siguiente	Salida actual
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

para la trayectoria de entrada siguiente:

x(0)=0 x(1)=1 x(2)=0 x(3)=1 x(4)=1 x(5)=0 x(6)=0 x(7)=1 x(8)=0 x(9)=1

Realice dos réplicas de la simulación. En la primera considere que el estado inicial es q(0)=0 y en la segunda considere que es q(0)=1. A la vista de los resultados, explique por qué este sistema se denomina "contador binario". Finalmente, escriba el código en lenguaje R que permite simular el comportamiento del modelo para esa trayectoria de entrada y la condición inicial q(0)=0.

Ejercicio 1.7

Escriba en lenguaje R un simulador para el autómata celular unidimensional con Regla 90 mostrado en la Figura 1.5. Realice el programa de manera que pueda configurarse de manera sencilla el número de células, el estado inicial y la función de transición de estados. Ejecútelo para el mismo número de células (61) y el mismo estado inicial mostrados en la Figura 1.5.

Ejercicio 1.8

El método de Monte Carlo permite estimar el valor de la integral definida de una función real monovaluada $g(x)$ entre dos valores reales finitos a y b:

$$I = \int_a^b g(x) \cdot dx \tag{1.12}$$

Para ello, se define una variable aleatoria Y de la forma siguiente:

$$Y = (b-a) \cdot g(X) \tag{1.13}$$

donde la variable aleatoria X está distribuida uniformemente en el intervalo $[a, b]$.

Puede demostrarse que el valor esperado de la variable aleatoria Y, que representaremos μ_Y, es igual al valor de la integral definida I que se desea calcular.

La media $\bar{Y}(n)$ de n observaciones independientes de Y es una estimación de μ_Y y por tanto de I. $\bar{Y}(n)$ puede calcularse de la Ecuación (1.14), donde $\{x_1, \ldots, x_n\}$ son n observaciones independientes de la variable aleatoria X.

$$\bar{Y}(n) = \frac{b-a}{n} \cdot \sum_{i=1}^{n} g(x_i) \tag{1.14}$$

Realice un programa en lenguaje R que aplique la técnica anteriormente descrita para estimar el valor de la integral definida siguiente:

$$I = \int_0^{\pi} \sin(x) \cdot dx \tag{1.15}$$

Esta integral posee primitiva analítica: $-\cos(x)$. Por ello es posible comparar el resultado obtenido aplicando la técnica de Monte Carlo con el resultado analítico exacto, que es $I = 2$.

Ejecute 5 veces el programa para cada uno de los tres tamaños de muestra siguientes: $n = 50$, 500 y 5000. Con el fin de comparar la distribución de las estimaciones obtenidas para cada tamaño de muestra, dibuje un gráfico con un boxplot para cada tamaño de muestra, en el cual se representen las correspondientes cinco estimaciones.

1.9. SOLUCIONES DE LOS EJERCICIOS

Solución al Ejercicio 1.1

La forma más adecuada de estudiar cada uno de los sistemas depende de cuál sea en cada caso el objetivo del estudio. Puesto que en el enunciado no se indican los objetivos, al contestar a la pregunta debe plantearse el objetivo del estudio de cada sistema y, una vez fijado el objetivo, debería explicarse qué forma o formas de estudio del sistema podrían ser válidas para alcanzar ese objetivo.

En algunos de los casos citados en el enunciado sería posible experimentar con el sistema real, mientras que en otros puede ser costoso y complejo (como en el caso del ecosistema), o imposible (como en el caso del diseño de una presa, puesto que el sistema aun no existe).

La experimentación con el sistema real y el modelado matemático son frecuentemente actividades complementarias, dado que los datos experimentales sirven de base para la construcción del modelo y para realizar su validación. En aquellos casos en que no se disponga de datos del sistema real, la construcción del modelo puede basarse en el conocimiento teórico disponible acerca del comportamiento de los diferentes componentes del sistema. Por ejemplo, el diseño de la presa puede realizarse empleando modelos matemáticos basados en una combinación de leyes físicas, relaciones empíricas y conjuntos de datos que describen el comportamiento mecánico de los materiales de construcción, del agua, etc.

A continuación, se plantea un posible objetivo en el estudio de cada sistema y una forma de estudiarlo que podría permitir alcanzar el objetivo. Otros objetivos y formas de estudio podrían ser igualmente válidas.

1. *Un ecosistema compuesto por varias especies (animales y vegetales) y por recursos (agua, luz, etc.).*

   El objetivo del estudio podría ser analizar la evolución del número de individuos de determinadas especies animales y vegetales, a partir de unas determinadas condiciones iniciales y para una cierta evolución en el tiempo de los factores ambientales. Para ello, podría plantearse un modelo matemático del sistema y experimentar con él mediante simulación.

   Puede considerarse que la variación en el tiempo de la cantidad de individuos de una especie vegetal depende de determinados factores ambientales (agua, luz, etc.) y de su tasa de reproducción, así como de la densidad de depredadores

(herbívoros) e individuos de otras especies vegetales que compitan con ella por los recursos (por ejemplo, por la luz).

Análogamente, podría suponerse que la variación en el número de individuos de una especie animal depende de su frecuencia reproductora, así como de la densidad de depredadores y presas, y de la abundancia de recursos tales como el agua.

2. *Una glorieta en la que convergen varias calles y que frecuentemente presenta atascos.*

   El objetivo del estudio podría ser estudiar la conveniencia de colocar semáforos en las calles de acceso a la glorieta y determinar qué programación de los semáforos resulta más adecuada en función de la hora del día. Podría emplearse para ello la simulación de un modelo matemático.

   Si el objetivo del estudio fuera señalizar los accesos a la glorieta, de modo que se diera prioridad a unos frente a otros, quizá podría emplearse un modelo mental.

3. *Una presa para el suministro de agua y electricidad que se planea construir en un río.*

   Si la finalidad del estudio es determinar la forma, altura y espesor de la presa, podría emplearse simulación de modelos matemáticos. Una vez determinada la forma, dimensiones y ubicación de la presa, sería útil realizar un modelo físico (una maqueta a escala) con el fin de dar a conocer las conclusiones del estudio.

4. *El servicio de urgencias de un hospital que se encuentra en funcionamiento.*

   Si el objetivo es estimar el tiempo de espera de los pacientes en función de su frecuencia de llegada, del tiempo del proceso de admisión, del número de boxes, enfermeros y médicos, entonces podría emplearse la simulación de un modelo matemático.

5. *Un circuito eléctrico.*

   Si el objetivo es calcular la tensión en los nodos del circuito y la corriente que circula a través de los componentes eléctricos, entonces podría emplearse un modelo matemático. En algunos casos sencillos el modelo matemático podría ser resuelto de manera analítica. En la práctica lo más habitual es analizar el modelo matemático mediante su simulación por ordenador.

   Otra posibilidad sería construir el circuito y realizar directamente las medidas.

En algunos casos sencillos podría emplearse un modelo mental, para decidir si es necesario aumentar o disminuir el valor de ciertos componentes, con el fin de conseguir modificar la tensión o la corriente de determinada forma.

Solución al Ejercicio 1.2

Un *modelo mental* es un conjunto de ideas que sirve de ayuda para predecir el comportamiento de un sistema. El modelo mental puede haber sido desarrollado a través de la experiencia, la observación o el aprendizaje. Por ejemplo, cuando vamos a cruzar una calle y vemos un coche que se aproxima, empleamos un modelo mental para decidir si nos dará tiempo a cruzar o si, por el contrario, debemos esperar.

Asimismo, los *modelos mentales* constituyen el estadio inicial en el desarrollo de cualquiera de los otros tipos de modelos, ya que no será posible desarrollar un modelo verbal, físico o matemático sin previamente haber desarrollado un modelo mental del sistema.

Un *modelo verbal* es la descripción del comportamiento del sistema mediante palabras. Se ajusta al esquema: "si se cumple esta condición, entonces debería ocurrir aquello". Un ejemplo de modelo verbal sería la descripción verbal del funcionamiento de un electrodoméstico (por ejemplo, una lavadora) donde se van describiendo las posibles acciones a realizar y sus resultados.

Un *modelo físico* es una maqueta que reproduce total o parcialmente un sistema. Un ejemplo de modelo físico es una maqueta de un vehículo cuyas características aerodinámicas quieren estudiarse en el túnel de viento. Otra finalidad de los modelos físicos es analizar los diseños desde el punto de vista estético.

Un *modelo matemático* consiste en la representación de las magnitudes de interés del sistema mediante variables y de la relación entre estas magnitudes mediante ecuaciones.

Solución al Ejercicio 1.3

Existen múltiples formas igualmente válidas de contestar a esta pregunta. A continuación se explica una de ellas.

El modelo matemático del *ecosistema* podría ser dinámico, determinista y de tiempo continuo. Dinámico si se quiere estudiar la evolución en el tiempo de las variables. Determinista si se conocen de manera precisa las condiciones iniciales y

además se conoce en cada instante el valor de las variables de entrada al modelo (los factores ambientales). De tiempo continuo si las variables analizadas varían de forma continua en el tiempo.

El modelo de la *glorieta* podría ser dinámico, estocástico y de eventos discretos. Dinámico si se pretende analizar la evolución temporal del sistema. Estocástico si el proceso de llegada de los coches por cada una de las calles que confluye en la glorieta se describe mediante un proceso estocástico. Es decir, si se supone que el tiempo que transcurre entre las sucesivas llegadas de coches está distribuido de acuerdo a una cierta distribución de probabilidad. De eventos discretos si el modelo se construye de modo que el valor de sus variables sólo cambia en los instantes de tiempo en que se producen eventos: cuando se produce la llegada de un nuevo vehículo, cuando cambia el estado de un semáforo (rojo, verde, amarillo), etc.

Otra alternativa sería describir la *glorieta* mediante un modelo de tiempo continuo. En lugar de representar cada coche como una entidad independiente, puede ser más eficiente computacionalmente considerar que el tráfico de coches es un flujo continuo, de valor cero cuando los coches están detenidos y de valor $f(t)$ vehículos/minuto cuando los coches están en movimiento. Se escribe $f(t)$ para indicar que el flujo f es una función del tiempo t.

El modelo de la *presa* podría ser estático y determinista. Estático si se trata de calcular qué distribución de estrés en los materiales de la pared corresponde a una cierta presión ejercida por el agua. Determinista si el valor de las variables de entrada al modelo (por ejemplo, la presión ejercida por el agua) es conocida.

Como sucede habitualmente con los modelos de los sistemas logísticos, el modelo del *servicio de urgencias* podría ser un modelo dinámico, estocástico y de eventos discretos. Estocástico si los procesos de llegada son estocásticos y los tiempos de proceso son variables aleatorias. De eventos discretos si las variables del sistema sólo cambian en los instantes en que se producen eventos. Eventos serían la llegada de cada paciente, y el inicio y la finalización de la atención a cada paciente en cada proceso (por ejemplo recepción, primera diagnosis y curas). Las variables del modelo permanecen constantes en los intervalos de tiempo entre eventos.

Finalmente, el modelo del *circuito eléctrico* podría ser dinámico, determinista y de tiempo continuo. Dinámico si se desea conocer la evolución temporal de tensiones y corrientes. Determinista si se conoce de manera precisa el estado inicial del modelo y en cada instante el valor de las variables de entrada. De tiempo continuo si las variables del modelo varían continuamente en el tiempo.

Solución al Ejercicio 1.4

Veamos dos ejemplos. En primer lugar, consideremos los niveles en el conocimiento de un *colector solar*. Se trata de un dispositivo que forma parte de los calentadores solares y que aprovecha la energía de la radiación solar para calentar un fluido (por ejemplo, una mezcla de agua y glicol) que circula por unos conductos situados dentro del panel. El fluido caliente circula desde el panel hasta su punto de uso (por ejemplo, los radiadores del sistema de calefacción de una vivienda), retornando luego al panel.

- En el Nivel 0 del conocimiento se identifica la porción del mundo que vamos a modelar y las maneras mediante las cuáles vamos a observarlo. Esta porción del sistema real, que en nuestro caso será el panel solar térmico, constituye el sistema fuente. La forma en que vamos a observar el colector será midiendo la radiación solar incidente, la temperatura de entrada del fluido y su temperatura de salida.

- En el Nivel 1 disponemos de una base de datos de medidas del sistema. En este caso, de las temperaturas de entrada y salida del fluido que se han medido para un cierto valor medido de la radiación incidente.

- En el Nivel 2 hemos sido capaces de correlacionar el incremento en la temperatura del fluido con la intensidad de la radiación incidente. Esta correlación la hemos obtenido ajustando los parámetros de una fórmula a los datos experimentales.

- Finalmente, en el Nivel 3 conocemos la estructura del sistema y somos capaces de entender su comportamiento a partir del comportamiento y la interacción de sus componentes. Sabemos que el panel solar térmico está compuesto por una caja rectangular, dentro del cual está el sistema captador de calor. Una de las caras de esta caja está cubierta de un vidrio muy fino y las otras cinco caras son opacas y están aisladas térmicamente. Dentro de la caja, expuesta al sol, se sitúa una placa metálica, que está tratada para que aumente su absorción del calor, y a la cual están soldados los conductos por los que circula el fluido transportador del calor.

En segundo lugar, consideremos los niveles en el conocimiento de una gasolinera, compuesta por varios surtidores, una tienda y varias cajas para realizar el pago.

- En el Nivel 0 conocemos qué porción del mundo vamos a estudiar y las maneras mediante las cuales vamos a observarlo. Una forma de observar el funcionamiento de la gasolinera sería medir el instante de tiempo en que se produce

la llegada de un cliente y el instante en que se produce su marcha. Con ello podría estimarse la distribución de probabilidad del tiempo que se tarda en atender a un cliente. También podría conocerse el número de clientes que están siendo atendidos en la gasolinera en cada instante. Entre ambos estadísticos existirá una correlación.

- En el Nivel 1 dispondríamos de una base de datos de medidas: número de clientes en la gasolinera y tiempos de atención.

- En el Nivel 2 dispondríamos de un modelo que permite predecir el tiempo medio de atención al cliente en función del número de clientes que se encuentran en la gasolinera.

- En el Nivel 3 conocemos de manera detallada la estructura de la gasolinera: número de surtidores, número de cajas de pago, tipo de cola que se forma frente a las cajas de pago (una cola común a todas o una cola frente a cada caja), etc. Para un determinado proceso de llegada de clientes y conociendo la distribución de los tiempos de proceso de llenado en los surtidores, compra en la tienda y cobro en las cajas, podemos simular el funcionamiento de la gasolinera y calcular estadísticos representativos del mismo. Es decir, podemos conocer el funcionamiento del sistema completo a partir del funcionamiento e interacción de sus partes.

Solución al Ejercicio 1.5

Consideremos el *colector solar* descrito al contestar al Ejercicio 1.4. El *análisis* del sistema consistiría en intentar comprender su funcionamiento basándose para ello en el conocimiento que se tiene de su estructura. A partir del análisis de su estructura, podemos comprender que el colector solar funciona aprovechando el efecto invernadero. El vidrio deja pasar la luz visible del sol, que incide en la placa metálica calentándola. Sin embargo, el vidrio no deja salir la radiación infrarroja de baja energía que emite la placa metálica caliente. A consecuencia de ello, y a pesar de las pérdidas por transmisión (el vidrio es mal aislante térmico), el volumen interior de la caja se calienta por encima de la temperatura exterior.

En la *inferencia*, disponemos de una base de datos del comportamiento del sistema fuente y tratamos de encontrar una representación del conocimiento al Nivel 2 (generación) o al Nivel 3 (estructura), que nos permita recrear los datos de que disponemos. Por ejemplo, hemos observado experimentalmente que el rendimiento de los colectores mejora cuanto menor sea la temperatura de trabajo. El modelo

que describe la estructura del sistema debería reproducir este comportamiento: a mayor temperatura dentro de la caja (en relación con la exterior), mayores serán las pérdidas por transmisión en el vidrio. También, a mayor temperatura de la placa captadora, más energética será su radiación, y más transparencia tendrá el vidrio a ella, disminuyendo por tanto la eficiencia del colector.

En el *diseño* de un sistema se investigan diferentes estructuras alternativas para un sistema completamente nuevo o para el rediseño de uno ya existente. Como parte del diseño del colector solar podrían investigarse el efecto sobre la absorción de calor de diferentes longitudes de los conductos, diferente volumen interno de la caja, diferentes pinturas aplicadas sobre la capa metálica, diferentes condiciones de operación (por ejemplo, caudal de fluido), etc.

Solución al Ejercicio 1.6

La trayectoria de entrada es la siguiente:

x(0)=0 x(1)=1 x(2)=0 x(3)=1 x(4)=1 x(5)=0 x(6)=0 x(7)=1 x(8)=0 x(9)=1

Para esta trayectoria, con un estado inicial q(0)=0, se obtiene las trayectorias de los estados y de salida siguientes:

q(0)=0	q(1)=0	q(2)=1	q(3)=1	q(4)=0	q(5)=1	q(6)=1	q(7)=1	q(8)=0	q(9)=0
x(0)=0	x(1)=1	x(2)=0	x(3)=1	x(4)=1	x(5)=0	x(6)=0	x(7)=1	x(8)=0	x(9)=1
q(1)=0	q(2)=1	q(3)=1	q(4)=0	q(5)=1	q(6)=1	q(7)=1	q(8)=0	q(9)=0	q(10)=1
y(0)=0	y(1)=0	y(2)=0	y(3)=1	y(4)=0	y(5)=0	y(6)=0	y(7)=1	y(8)=0	y(9)=0

Para la misma trayectoria de entrada, pero con estado inicial q(0)=1, se obtiene:

q(0)=1	q(1)=1	q(2)=0	q(3)=0	q(4)=1	q(5)=0	q(6)=0	q(7)=0	q(8)=1	q(9)=1
x(0)=0	x(1)=1	x(2)=0	x(3)=1	x(4)=1	x(5)=0	x(6)=0	x(7)=1	x(8)=0	x(9)=1
q(1)=1	q(2)=0	q(3)=0	q(4)=1	q(5)=0	q(6)=0	q(7)=0	q(8)=1	q(9)=1	q(10)=0
y(0)=0	y(1)=1	y(2)=0	y(3)=0	y(4)=1	y(5)=0	y(6)=0	y(7)=0	y(8)=0	y(9)=1

El estado del sistema almacena el bit menos significativo de la suma binaria de los valores de entrada. La salida es el acarreo.

Existen varias maneras de escribir un programa en lenguaje R que simule el comportamiento del sistema para la trayectoria de entrada dada. Se muestra a continuación un posible programa. Lo más cómodo es escribir el programa en un fichero de texto y cargarlo desde la consola de R empleando la función `source()`.

```
x <- c(0,1,0,1,1,0,0,1,0,1) # Trayectoria de entrada
t <- c(0:(length(x)-1))     # Tiempo
q <- 0                      # Estado inicial
y <- numeric(0)             # Salida
# Bucle de la simulación
for (i in 1:length(x)) {
   q[i+1] <-      if ( q[i]==0 & x[i]==0 ) 0
             else if ( q[i]==0 & x[i]==1 ) 1
             else if ( q[i]==1 & x[i]==0 ) 1
             else                          0
   y[i]   <-      if ( q[i]==0 & x[i]==0 ) 0
             else if ( q[i]==0 & x[i]==1 ) 0
             else if ( q[i]==1 & x[i]==0 ) 0
             else                          1
}
# Salida de resultados
q <- q[1:length(x)]
resultados <- data.frame(t,x,q)
```

Una vez ejecutado el código anterior, el resultado está almacenado en el data frame `resultados`. Su contenido es el siguiente:

```
> resultados
   t x q
1  0 0 0
2  1 1 0
3  2 0 1
4  3 1 1
5  4 1 0
6  5 0 1
7  6 0 1
8  7 1 1
9  8 0 0
10 9 1 0
```

Solución al Ejercicio 1.7

Se muestra a continuación el programa. Se crea una matriz llamada `evolAutom`, en cuyas filas se va almacenando el estado del autómata en los sucesivos instantes de tiempo. En la primera fila se almacena el estado inicial, en la segunda fila el estado en el siguiente instante de tiempo y así sucesivamente. Una vez realizada la simulación, se emplea la función `matrixplot()` para visualizar el contenido de `evolAutom`. Para poder usar esa función es necesario tener cargado el paquete `VIM` (es posible descargarlo de Internet, instalarlo y cargarlo desde la propia plataforma R). En la Figura 1.19 se muestra el resultado de la ejecución del programa.


```
 # Estado inicial
 estadoInicial <- c( rep(0,30), 1, rep(0,30) );
 # ------------------------------
 # Tabla de transición de estados
 # ------------------------------
 tabla <- numeric(0)
 #  Estado          Entrada   Estado   Entrada
 # siguiente         izqda    actual    drcha
 tabla[1] <- 0  #     0         0         0
 tabla[2] <- 1  #     0         0         1
 tabla[3] <- 0  #     0         1         0
 tabla[4] <- 1  #     0         1         1
 tabla[5] <- 1  #     1         0         0
 tabla[6] <- 0  #     1         0         1
 tabla[7] <- 1  #     1         1         0
 tabla[8] <- 0  #     1         1         1
 # ------------------------------
 # Número de réplicas
 numReplicas <- 30
 # ------------------------------
 # Simulación
 evolAutom <- matrix( 0, nrow=numReplicas , ncol=length(estadoInicial) )
 evolAutom[1,] <- estadoInicial
 for (rep in c(2:numReplicas)) {
    for (cel in c(2:(length(estadoInicial)-1))) {
       evolAutom [rep,cel] <- tabla[ 4*evolAutom[rep-1,cel-1]+
                                     2*evolAutom[rep-1,cel]  +
                                       evolAutom[rep-1,cel+1]+ 1 ]
    }
 }
 # La función matrixplot está definida en el paquete VIM
 matrixplot(evolAutom, labels=FALSE)
```

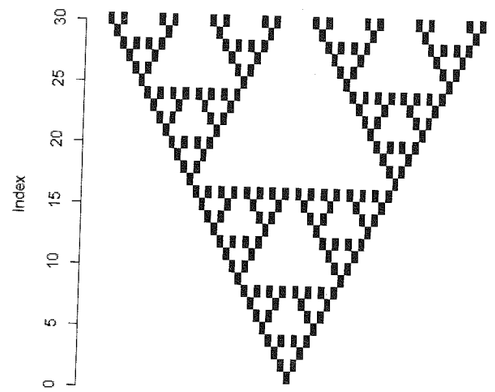

Figura 1.19: Evolución del celular unidimensional con Regla 90.

Solución al Ejercicio 1.8

A continuación se muestra una forma de realizar el programa.

```
# Función para la estimación de la integral
estimaI <- function(n,a,b) {
   x <- runif(n,a,b)
   I <- (b-a)*sum( sin(x) )/n
   return (I)
}

# Extremos del intervalo de integración
a <- 0
b <- 3.14159265358979323
# Número de réplicas
nReplicas <- 5
# ------------------------
# Tamaño de la muestra 50
I_50 <- numeric(0)
for (rep in c(1:nReplicas))
   I_50[rep] <- estimaI(50,a,b)
# ------------------------
# Tamaño de la muestra 500
I_500 <- numeric(0)
for (rep in c(1:nReplicas))
   I_500[rep] <- estimaI(500,a,b)
# ------------------------
# Tamaño de la muestra 5000
I_5000 <- numeric(0)
for (rep in c(1:nReplicas))
   I_5000[rep] <- estimaI(5000,a,b)
# ------------------------
# Representación resultados
I <- data.frame(I_50, I_500, I_5000)
boxplot(I)
abline(h=2, lty=3)  # Dibuja linea horizontal
```

El resultado obtenido de la ejecución del programa se encuentra almacenado en el data frame `I`. En la Figura 1.20 se muestran los boxplots. Puede observarse que al aumentar el tamaño de muestra los resultados tienden hacia el valor verdadero y su variabilidad disminuye.

Obsérvese también que en general se obtendrán diferentes resultados si se repite la ejecución del programa, ya que R usa en cada caso secuencias diferentes de números pseudoaleatorios.

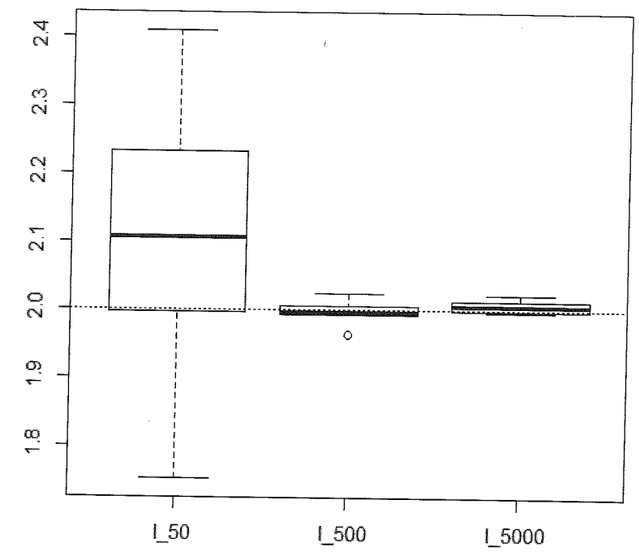

Figura 1.20: Boxplots de las estimaciones de la integral para cada tamaño de muestra.

```
> I
      I_50     I_500    I_5000
1 2.231706 1.964716 2.013596
2 2.106927 1.992722 1.997176
3 2.410476 2.006765 2.006508
4 1.751520 2.024015 2.022142
5 1.996247 1.996353 1.998435
```