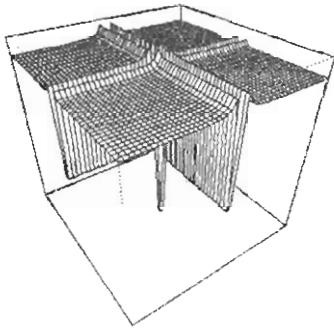


# *R: Un Language Estadístico*



Juan Carlos Correa Morales

y

Juan Carlos Salazar Uribe

Universidad Nacional de Colombia-Sede Medellín  
Facultad de Ciencias  
Departamento de Matemáticas  
Posgrado en Estadística

2000  
UNAL-Medellin



6 4000 00128442 4



UNIVERSIDAD NACIONAL DE COLOMBIA  
MEDÉLLIN  
DEPTO. DE BIBLIOTECAS  
BIBLIOTECA "EFE" GOMEZ

I  
519.7  
C67

# TABLA DE CONTENIDO

<b>1. <i>Introducción</i></b>	<b>3</b>
1.1 Instalación de R	4
1.2 Iniciando una sesión en R	4
1.3 Terminando una sesión en R	5
1.4 Cómo ejecutar comandos desde un archivo externo	6
1.5 Ejecución del "demo" de gráficos	6
<b>2. <i>Manejo de Datos en R</i></b>	<b>7</b>
2.1 Ventana para Gráficos	7
2.2 Lectura de Datos	7
2.2.1 Entrando Datos Desde el Teclado	8
2.2.2 Entrando Listas	10
2.2.3 Datos Faltantes	11
2.2.4 Generando Secuencias de Números	11
2.2.5 Replicando una Estructura	12
2.2.6 Entrando Datos Desde un Archivo en ASCII	12
2.2.7 Leyendo Bases de Datos	13
2.2.8 Utilizando el Editor de R	14
2.2.9 Escribiendo una Matriz a un Archivo Externo	14
2.2.10 Enviando Todos los Resultados a un Archivo de Texto	15
2.3 Manipulando Datos	15
2.3.1 Operadores	15
2.3.2 Operadores de comparación	16
2.3.3 Operadores Lógicos	16

1284424

2.3.4 Operadores de Control	17
2.3.5 Operaciones Básicas	17
2.3.6 Subíndices	17
2.3.6.1 Enteros positivos	18
2.3.6.2 Valores lógicos	18
2.3.6.3 Enteros negativos	18
2.3.7 Funciones que producen escalares	18
2.3.8 Operaciones con matrices	19
2.3.9 Trabajando con Distribuciones	21
2.4 Escribiendo Datos a un Archivo ASCII	22
2.5 Ejecuciones Condicionales y Loops	23
2.6 Objetos en R	26
2.7 Para Remover Objetos	27

### ***3 Gráficos en R*** **28**

3.1 ¿Cómo Imprimir Gráficos?	28
3.2 Para Graficar una variable	28
3.3 ¿Cómo graficar más de una variable?	33
3.4 Manejo de Parámetros Gráficos	38
3.5 Adicionando Elementos a un Gráfico	40
3.6 Gráficas de Funciones	41
3.7 Elementos de simulación	42
3.7.1. Simulación del Teorema de límite central.	42
3.7.2. Simulación de una Chi-cuadrado con diferentes grados de libertad	45
3.7.3 Simulación del lanzamiento de dos dados	46
3.7.8 Simulación del concepto de nivel de significación	46

## **4. Creación de Nuevas Funciones en R 47**

4.1. Ejemplo: Función para calcular estadísticas básicas	50
4.2. Ejemplo: Prueba para varianzas de Levene	52
4.3. Ejemplo: Gráficos simultáneos	53
4.4. Ejemplo: Gráfico de contornos	54
4.5. Ejemplo: Codificación de variables	56
4.6. Ejemplo: Gráfico de una silla de montar	58
4.7 Algunas reglas para escribir funciones en R	60

## **5. Modelos Estadísticos en R 61**

5.1 Estadística Básica	61
5.2 Fórmulas en R	63
5.2.1 Interacción y Encajamiento	63
5.3 Regresión	64
5.4 Análisis Multivariable	70
5.5 Análisis de Datos Categóricos	71
5.6 Modelo Lineal Generalizado	74
5.7 Análisis de Varianza	76
5.7.1. Ejemplo: Diseño de Bloques Completamente Aleatorizado	78
5.7.2. Ejemplo: Experimentos factoriales Encajados	79
5.7.3. Ejemplo: Diseños Split-Plot	80
5.8 ¿Cómo utilizar paquetes adicionales?	83
5.9 Problemas de Memoria en R	84

<b><i>Apéndice A: Datos</i></b>	<b>85</b>
A.1 Datos de Velocidades de Autos	85
A.2 Datos de Precios de Carros Reanult 4 y 12	85
A.3 Datos de las pruebas nacionales del ICFES	85
<b><i>Apéndice B. Listado de Funciones</i></b>	<b>90</b>
B.1 Funciones Instaladas en RHOME	90
B.1.1. Operadores, Variables globales	90
B.1.2. Diccionario	93
<b><i>Referencias</i></b>	<b>133</b>

## Capítulo 1. Introducción

Estas notas son para uso local en la Universidad Nacional de Colombia-Sede Medellín y para cualquier persona que desee explorar un nuevo y poderoso lenguaje estadístico. Inicialmente se realizaron para motivar el uso del programa R entre los estudiantes y profesores de pregrado y del posgrado de estadística y de otras carreras ya que su contenido es bastante amplio.

El programa R fue escrito por Ross Ihaka y Robert Gentleman en la Universidad de Auckland, Nueva Zelanda a comienzos de los 90 (Ihaka, 1998). La liberación del programa como "programa de dominio público" le dió un impulso grande y se formó el "R Core Team", equipo conformado con estadísticos programadores de gran experiencia como Doug Bates, Peter Dalgaard, Robert Gentleman, Kurt Hornik, Ross Ihaka, Friedriech Leisch, Thomas Lumley, Martin Maechler, Guido Masarotto, Paul Murrel, Brian Ripley, Heiner Schwarte, y Luke Tierney. Muchas de las anteriores personas fueron pilares fundamentales en el desarrollo del S-Plus.

Más que un programa de estadística R puede ser considerado un lenguaje de alto nivel. Es completamente estructurado. La programación es dinámica ya que el uso de la memoria y el dimensionamiento de matrices es ejecutado automáticamente. Permite definir funciones que pasan a ser parte del sistema automáticamente y pueden ser llamadas en posteriores sesiones sin tener que definir las nuevamente. R puede pensarse, aunque es mucho más, como un lenguaje matricial. Las siguientes son unas ventajas:

- 1) Opera con matrices,
- 2) Posee una amplia base de operadores,
- 3) Usa operadores que se aplican a matrices completas, por ejemplo, si A y B son matrices las siguientes operaciones son posibles:  $A+B$ ,  $A-B$ , etc.
- 4) Es interactivo,

5) Produce gran variedad de gráficos de excelente calidad.

## 1.1 Instalación de R

Para la instalación de R en su computador usted debe poseer Windows 95, 98 o Windows NT. Se procede como sigue:

- 1) Cree un directorio llamado R.
- 2) Copie todos los archivos `rwinst.exe`, `rw0642b.zip`, `rw0642h.zip`, `rw0642l.zip`, `rw0642sp.zip` y `rw0642w.zip` en este directorio
- 3) Ejecute el programa `rwinst.exe`. Este programa instalará automáticamente todos los archivos que necesita para trabajar.
- 4) Cree el ícono de R, para ello presione la tecla derecha de su ratón (mouse). Seleccione la opción *Nuevo* y luego la opción *Acceso Directo*. Busque el subdirectorio `rw0642` y seleccione el archivo `rgui.exe`.

## 1.2 Iniciando una sesión en R

Trabajando con la versión de Windows simplemente con el ratón escogemos el ícono apropiado y hacemos click dos veces sobre él. El programa al iniciar le mostrará una nota sobre la versión y los derechos de autor. También aparecerá un prompt, "`>`", y el cursor se ubicará en esa línea. El programa espera que usted le dé los comandos respectivos.

El siguiente gráfico le muestra el pantallazo inicial cuando ud. comienza R. Se observa el comentario sobre su licencia y su versión.

```

RGui
File Edit Misc Windows Help
R Console
R : Copyright 1999, The R Development Core Team
Version 0.64.2 (July 3, 1999)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type "?license" or "?licence" for distribution details.

R is a collaborative project with many contributors.
Type "?contributors" for a list.

Type "demo()" for some demos, "help()" for on-line help, or
"help.start()" for a HTML browser interface to help.
Type "q()" to quit R.

> █
R 0.64.2 - A Language and Environment

```

## 1.3 Terminando una sesión en R

Para finalizar una sesión en R escriba el siguiente comando:

```
> q()
```

El símbolo > que aparece al lado izquierdo es el prompt. Ud. no lo tiene que escribir y lo seguiremos escribiendo para conservar uniformidad con los otros textos sobre R.

**OJO:** Es importante notar que para el programa las mayúsculas son diferentes a las minúsculas, por lo tanto para terminar usamos `q()` y no `Q()`. El programa reconoce las letras mayúsculas y las diferencia de las minúsculas y nombres como los siguientes hacen referencia a diferentes objetos: `datos`, `Datos`, `dAtos`, `DATOS`, etc.

## 2.2 Lectura de Datos

La lectura de datos puede hacerse desde teclado o desde un archivo ASCII. Una vez los datos han sido leídos estos se almacenan en un objeto.



## 1.4 Cómo ejecutar comandos desde un archivo externo

Cuando el análisis que vamos a realizar requiere muchos comandos, es conveniente editarlos y guardarlos primero en un archivo ASCII, digamos comandos.txt, la extensión .txt es solo con el fin de enfatizar que el archivo es en ASCII, ellos pueden ser ejecutados en cualquier momento en una sesión de R simplemente escribiendo el comando

```
> source('comandos.txt')
```

Esto tiene además la ventaja de que en un caso infortunado, por ejemplo se vaya la luz, etc., no tengamos que repetir todo el análisis nuevamente.

Otra forma alternativa es seleccionando el código fuente copiarlo y luego pegarlo en la consola de R.

## 1.5 Ejecución del "demo" de gráficos.

Ubicándose en el prompt escriba el comando

```
> demo(graphics)
```

A continuación presione la tecla "enter". Debe esperar algunos segundos mientras comienza la demostración.

## Capítulo 2. Manejo de Datos en R

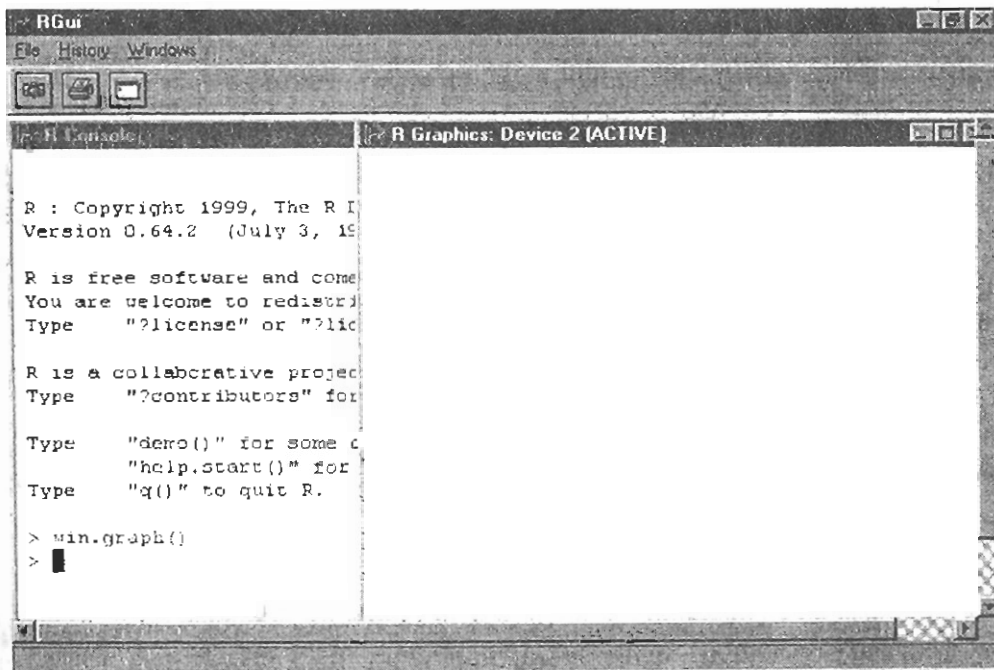
El manejo de datos en R puede parecer complejo al principio, pero no es sino cuestión de costumbre. El programa cuenta con funciones poderosas tanto para la lectura como la escritura de datos.

### 2.1 Ventana para Gráficos

Los gráficos en R aparecen en ventanas separadas a la pantalla donde se teclean los comandos. El comando que se da para abrir una nueva ventana para gráficos es

```
>win.graph()
```

lo cual produce una nueva ventana como aparece a continuación



### 2.2 Lectura de Datos

La lectura de datos puede hacerse desde teclado o a través de un archivo en ASCII. Una vez los datos han sido leídos estos quedan en forma permanente

en el disco duro en formato R. Estos datos en formato especial pueden usarse repetidamente en diferentes sesiones.

## 2.2.1 Entrando Datos Desde el Teclado

R opera con lo que se conoce como estructura de datos. La más simple de tales estructuras es el vector, que es una sola entidad consistente de una colección ordenada de números o caracteres. Para crear un vector llamado *x*, que tenga seis elementos, digamos 3.6, 2.5, 1.2, 0.6, 1.3, y 2.1, utilizamos el comando

```
> x<-c(3.6,2.5,1.2,0.6,1.3,2.1)
```

Este es un comando de asignación que utiliza la función `c()` con la cual creamos vectores y en este contexto puede tomar un número arbitrario de argumentos. El resultado final es un vector que concatena los argumentos de la función `c()`.

El símbolo "`<-`" es el de asignación (se logra tecleando primero "`<`", el símbolo "menor que" y luego "`-`", el símbolo "menos"). Lo que está sobre la derecha es asignado al objeto de la izquierda. Podemos tener asignaciones simples o múltiples. Por ejemplo,

```
> x <-0
```

```
> y<-x<-0
```

Si tenemos el siguiente conjunto de frutas: pera, manzana, banano, pera, curuba, lo podemos asignar a un vector así:

```
> frutas<-c('pera','manzana','banano','pera','curuba')
```

Esto nos crea un vector llamado *frutas* de caracteres.

Datos pueden también entrarse con la función `scan()` de una forma muy simple

```
> x<-c(scan())
```

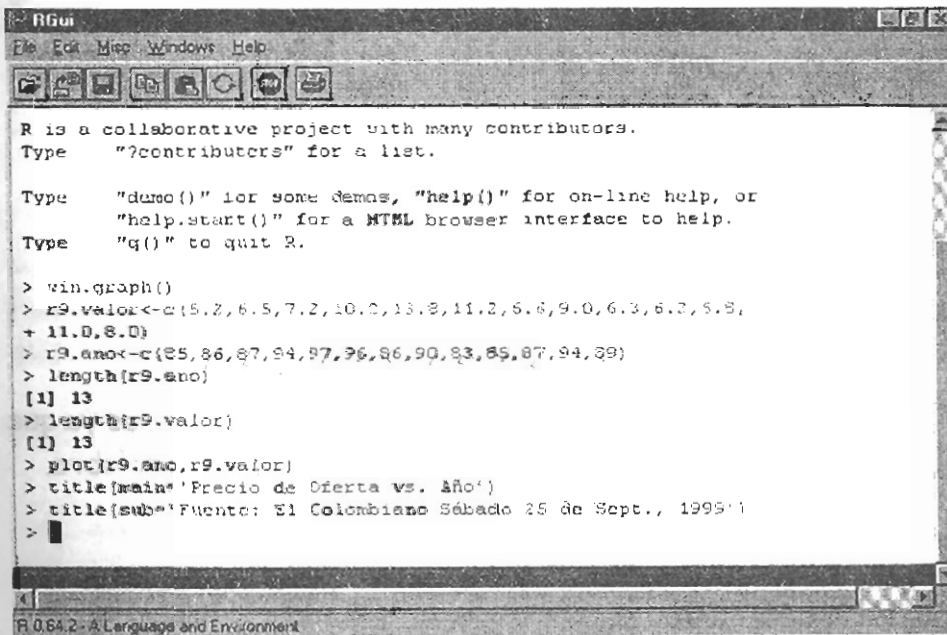
```
1: 3.6 2.5
2: 1.2 0.6 1.3 2.1
7:
```

```
>
```

Esto nos permite entrar tantos datos por línea como queramos, dejando espacio entre datos consecutivos. Los números que aparecen antes de los dos puntos aparecen automáticamente y nos indica el número de la siguiente observación a ser entrada desde el teclado. Si queremos ver en pantalla el vector de datos  $x$  la instrucción será:

```
>x
```

El siguiente ejemplo presenta un caso simple en el cual se analiza un conjunto de datos referentes al precio de oferta de vehículos Renault-9 aparecidos en el periódico 'El Colombiano' en los avisos clasificados el sábado 25 de septiembre de 1999. Así se ve la ventana de R una vez hemos tecleado los datos correspondientes al *precio de oferta* y el *año del vehículo*:



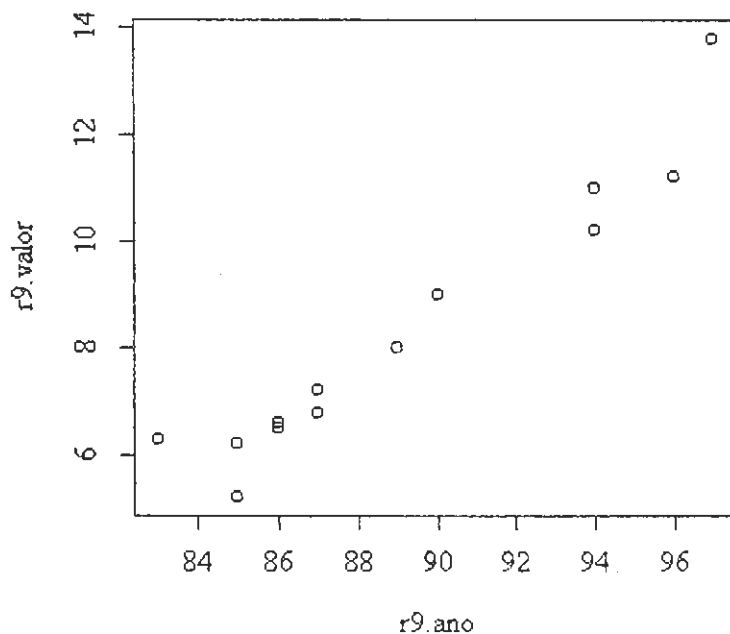
```
RGui
File Edit Misc Windows Help
R is a collaborative project with many contributors.
Type "?contributors" for a list.

Type "demo()" for some demos, "help()" for on-line help, or
"help.start()" for a HTML browser interface to help.
Type "q()" to quit R.

> win.graph()
> r9.valor<-c(5.2,6.5,7.2,10.0,13.8,11.2,8.6,9.0,6.3,6.7,5.8,
+ 11.0,8.0)
> r9.ano<-c(85,86,87,84,87,86,86,90,83,85,87,84,89)
> length(r9.ano)
[1] 13
> length(r9.valor)
[1] 13
> plot(r9.ano,r9.valor)
> title(main="Precio de Oferta vs. Año")
> title(sub="Fuente: El Colombiano Sábado 25 de Sept., 1999")
>
```

La gráfica siguiente presenta el resultado de los tres últimos comandos tecleados en la ventana anterior.

### Precio de Oferta vs. Año



Fuente: El Colombiano Sábado 25 de Sept., 1999

Si deseamos entrar vectores alfanuméricos entonces la función `scan()` llevará opciones adicionales, por ejemplo,

```
>ciudades<-scan(what=character(0))
```

```
1: Medellín "Santa Fe de Bogota" Cali Pereira
```

```
5: Cartagena Barranquilla
```

```
7:
```

```
>
```

Observe como el nombre "Santa Fe de Bogota" debe ir entre comillas ya que si no se hace así nos crea cuatro nombres diferentes, uno por cada palabra.

## 2.2.2 Entrando Listas

Si se desea entrar un conjunto de datos que contienen tanto variables numéricas como alfanuméricas, por ejemplo nombre, peso y edad, se hará

así:

```
> datos<-scan(what=list("", "", 0,0))
```

```
1: Juan 70 25
4: Pedro 65 28
7: Maria 50 22
10: Norma 55 39
13:
```

```
>
```

## 2.2.3 Datos Faltantes

Cuando tenemos valores de las variables faltantes debemos reemplazarlo en R con la palabra NA.

## 2.2.4 Generando Secuencias de Números

R permite crear secuencias de una forma ágil. Por ejemplo 1:5 es equivalente a `c(1,2,3,4,5)`.

```
> s1<-2:100
```

El anterior comando crea un vector llamado `s1` con los elementos 2, 3, 4, ..., 100. Otra forma, aún más poderosa de crear secuencias es usar la función `seq()`, ya que esta permite definir un incremento. El símbolo `:` tiene la más alta prioridad dentro de una expresión. Un ejemplo de esto es,

```
> s2<-seq(1,2,by=.2)
```

Este comando crea un vector `s2` con los elementos 1.0, 1.2, 1.4, 1.6, 1.8, 2.0.

## 2.2.5 Replicando una Estructura

La función `rep()` puede usarse para replicar una estructura particular cierto número de veces. La forma más simple es

```
> s3<-rep(x,times=4)
```

lo cual nos crea un objeto que es cuatro veces el objeto `x`.

## 2.2.6 Entrando Datos Desde un Archivo en ASCII

La lectura de datos que están guardados en un archivo en ASCII se hace con la función `scan()`, es de notar la similaridad con el lenguaje C.

```
> pesos.monedas<-c(scan('a:pesos.txt'))
Read 100 items
> anos.monedas<-c(scan('a:anos.txt'))
Read 100 items
>
```

Si tenemos una matriz de datos con  $d$  columnas y  $n$  filas, por el momento suponemos que solo contiene elementos numéricos, en un archivo llamado "misdatos", los cuales se encuentran grabados en un diskette que se coloca en el drive a. El comando es

```
> datos<-matrix(scan('a:misdatos'),ncol=d,byrow=T)
```

Aquí tenemos una función nueva: `matrix()`. Esta nos permite crear un objeto que es una matriz. Si los datos estuvieran separados por comas se coloca una opción adicional de la siguiente forma:

```
> datos<-matrix(scan('a:misdatos',sep=','),ncol=d,byrow=T)
```

La función `scan` tiene la opción `sep` que por defecto considera un espacio como la separación normal entre dos datos, pero puede indicarse otra cosa. Por ejemplo si estamos leyendo dato por línea entonces el comando es





## 2.2.8 Utilizando el Editor de R

A partir de la versión 3.3 se puede usar la función `data.ed` para editar datos de los siguientes objetos:

**vectores**

**matrices**

**marcos de datos**

Esta función abre una ventana que presenta los datos de forma similar a una hoja electrónica. Solo pueden editarse objetos que existan previamente, no pueden editarse objetos que no existan. Tampoco es posible cambiar las dimensiones de un objeto con esta función. Para editar un objeto y salvar los cambios se debe asignar un objeto que recibe los cambios, por ejemplo:

```
> mi.vector.modificado<-data.ed(mi.vector)
```

## 2.2.9 Escribiendo una Matriz a un Archivo Externo

Para escribir una matriz a un archivo ASCII utilizamos la función `write()`, que se utiliza de la siguiente forma

```
> write(matriz, 'archivo_de_salida', nro_columnas, append=FALSE)
```

Debido a la forma en que el R guarda las matrices, las matrices son escritas al revés. Para evitar esto escribimos, por ejemplo,

```
> write(t(X), file='matriz.txt', ncol=ncol(X))
```

Si queremos crear un archivo en ASCII que contenga un objeto usamos la función `dump()`.

```
> dump(objeto, 'archivo')
```

## 2.2.10 Enviando Todos los Resultados a un Archivo de Texto

A veces se requiere tener un archivo ASCII con los resultados que se obtengan en una sesión. Para ello usamos la función `sink()`. Esta nos redirecciona la salida de los comandos de la pantalla a un archivo. En la terminal no observamos nada. Es muy útil cuando se ejecutan trabajos tales como simulaciones.

```
> sink('resulta.txt')
```

Cuando se desea restablecer el modo normal de operación, esto es, volver a ver los resultados en la terminal, se da el comando

```
> sink()
```

## 2.3 Manipulando Datos

La manipulación de datos se hace de diferentes formas. Usualmente se toma ventaja de la vectorización del lenguaje. Esto permite trabajar sobre un conjunto de elementos en lugar de trabajar elemento a elemento como en lenguajes normales.

### 2.3.1 Operadores

+ : Suma

- : Resta

\* : Multiplicación

/ : División

^ : Exponenciación

%/% : División entera

%% : Operador módulo

## 2.3.2 Operadores de comparación

< : menor

> : mayor

<= : menor o igual

>= : mayor o igual

== : igual

!= : diferente

## 2.3.3 Operadores Lógicos

& : y

| : ó

! : no

all(...): Dada una secuencia de argumentos lógicos, esta función retorna un valor lógico que indica cuales de todos los elementos de x son "TRUE".

any(...): Dada una secuencia de argumentos lógicos, esta función retorna un valor lógico que indica cuales de todos los elementos de x son "TRUE".

## 2.3.4 Operadores de Control

`&&` : Si el primer operando es cierto se evalúa el segundo operando

`||` : Si el primer operando es falso se evalúa el segundo operando.

## 2.3.5 Operaciones Básicas

Siendo el lenguaje R un lenguaje vectorizado, los vectores pueden usarse en expresiones aritméticas, en cuyo caso las operaciones son ejecutadas elemento a elemento. Si  $x$  y  $z$  son vectores, no necesariamente de la misma dimensión, entonces podemos ejecutar los siguientes comandos

```
> y<-x+z
```

```
> y2<-x-z
```

```
> y3<-2*x+z-3
```

La dimensión de  $y$ ,  $y_2$  y  $y_3$  será igual a la dimensión mayor de los vectores  $x$  y  $z$ .

```
> y4<-1/x
```

El anterior comando produce un vector cuyos elementos corresponden a los inversos de  $x$ .

## 2.3.6 Subíndices

Frecuentemente se requiere extraer subconjuntos de datos. Estos datos pueden ser extraídos de un objeto de la siguiente forma:

```
x[ subíndice ]
```

La forma del subíndice puede expresarse de varias formas dependiendo de lo que se desee.

### 2.3.6.1 Enteros positivos

`x[1]` Solo nos quedamos con el primer elemento de `x`

`x[1:3]` Solo nos quedamos con los tres primeros elementos de `x`

```
> icfes.dat<-read.table('c:/datos/icfes.dat')
> icfes.dat[1:2,]
  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12
1  F 89 51 41 40 52 55 49 54  57  47 CON
2  F 89 46 29 40 40 39 49 50  48  51 CON
>
```

`x[2:5]` Solo nos quedamos con elementos 2, 3, 4, y 5 de `x`

### 2.3.6.2 Valores lógicos

`x[x>0]` Sólo tomamos los valores de `x` que sean positivos.

```
nombre[estatura>175 & edad<30]
```

Nos quedamos con aquellos nombres cuyas correspondientes estaturas sean mayores a 175 y la edad sea menor de 30.

### 2.3.6.3 Enteros negativos

`x[-1]` Nos quedamos con todo el vector `x` pero eliminamos el primer elemento.

## 2.3.7 Funciones que producen escalares

Existe una gran cantidad de funciones que al ser aplicadas a un vector

producen como resultado un escalar. Entre ellas tenemos:

**max()**: retorna el máximo del argumento

**min()**: retorna el mínimo del argumento

**sum()**: retorna la suma de todos los elementos del argumento

**prod()**: retorna el producto de todos los elementos del argumento

**ncol()**: número de columnas

**nrow()**: número de filas

**var()**: retorna la varianza de un conjunto de datos

**mean()**: retorna la media aritmética de un conjunto de datos

### 2.3.8 Operaciones con matrices

Si  $A$  y  $B$  son matrices entonces podemos usar los operadores  $*$ ,  $\%*\%$ . El primero es la operación producto elemento a elemento, obviamente las matrices deben tener la misma dimensión. El segundo operador corresponde a la multiplicación de matrices.

```
> c<-A*B
```

```
> c<-A%*%B
```

R proporciona una gran variedad de funciones para trabajar con matrices. Aquí mencionamos unas cuantas:

**sqrt(A)**: calcula la raíz cuadrada de cada elemento

**log(A)**: regresa una matriz con los logaritmos de las componentes

**exp(A):** genera una matriz donde cada entrada corresponde a la función exponencial de cada elemento de la matriz dada en el argumento

**t():** esta función retorna la matriz transpuesta del argumento

**crossprod():** esta función retorna la matriz producto cruzado del argumento

**svd():** produce la descomposición en valores singulares de una matriz

**qr():** Produce la descomposición QR de una matriz

**cbind():** junta dos matrices por columnas

**rbind():** junta dos matrices por fila

**chol():** produce la descomposición de Cholesky de una matriz simétrica

**eigen():** produce los valores propios de una matriz cuadrada

**diag():** crea una matriz diagonal si el argumento es escalar o retorna la matriz diagonal de una matriz

**kroncker():** producto kronecker entre matrices

**Nota:** *La matriz inversa no se encuentra directamente sino a través de solve()*

**solve():** produce la inversa si solo se entra un argumento, o resuelve un sistema de ecuaciones lineales si se entran dos argumentos.

**sort():** ordena en forma ascendente el vector dado como argumento.

**apply():** aplica una función a partes de una matriz. Esta función simplifica los cálculos que operan de forma iterada sobre columnas o filas de una matriz. Si tenemos arreglos tridimensionales opera sobre las matrices en caso requerido. Por ejemplo, si deseamos calcular las medias de las columnas de una matriz X damos el comando

```
>medias.col<-apply(X,2,mean)
```

## 2.3.9 Trabajando con Distribuciones

El R proporciona varias funciones para generar densidades, funciones acumuladas de probabilidad, cuantiles y valores aleatorios.

<i>Distribución</i>	<i>Densidad.</i>	<i>Función de Prob Acumulada</i>
Uniforme	dunif(q, min,max)	punif(q, min,max)
Normal	dnorm(q,mu, sigma)	pnorm(q,mu ,sigma )
Binomial	dbinom(q,n,pi )	pbinom(q,n,pi)
Lognormal	dlnorm(q, mu ,sigma )	plnorm(q, \mu , \sigma )
Beta	dbeta(q, f1,f2)	pbeta(q, f1,f2)
Geométrica	dgeom(q, prob)	pgeom(q, prob)
Gamma	dgamma(q, f)	pgamma(q, f)
Chi-cuadrado	dchisq(q, df)	pchisq(q, df)
Exponencial	dexp(q,tasa)	pexp(q,tasa)
F	df(q,df1,df2)	pf(q,df1,df2)
Hipergeométrica	dhyper(q,m,n,k)	phyper(q,m,n,k)
t	dt(q, df)	pt(q, df)
Poisson	dpois(q,lambda)	ppois(q,lambda)



***Cuantiles***

qunif(p, min,max)  
 qnorm(p,mu ,sigma )  
 qbinom(p,n,pi )  
 qlnorm(p, \mu , \sigma )  
 qbeta(p, f1,f2)  
 qgeom(p, prob)  
 qgamma(p, f)  
 qchisq(p, df)  
 qexp(p,tasa)  
 qf(p,df1,df2)  
 qhyper(p,m,n,k)  
 qt(p, df)  
 qpois(p,lambda)

***Números******Aleatorios***

runif(N, min,max)  
 rnorm(N, mu, sigma )  
 rbinom(N,n, pi )  
 rlnorm(N, mu, sigma )  
 rbeta(N, f1,f2)  
 rgeom(N, prob)  
 rgamma(N, f)  
 rchisq(N, df)  
 rexp(N,tasa)  
 rf(N,df1,df2)  
 rhyper(N,m,n,k)  
 rt(N, df)  
 rpois(N,lambda)

## 2.4 Escribiendo Datos a un Archivo ASCII

La función `write()` le permite escribir los datos en formato R a un archivo en formato ASCII, los cuales pueden ser editados posteriormente. Las funciones `print()`, `format()`, `cat()` y `paste()` pueden usarse para dar formato a los resultados que van a escribirse en estos archivos.

La función `sink()` permite escribir los resultados obtenidos en una sesión R a un archivo. Por ejemplo, el siguiente es el resultado de la redirección de resultados a un diskette, creando un archivo de texto llamado *result.txt*:

```

> sink('a:/result.txt')
> options(echo=T)
> summary(lm(r9.valor~r9.ano))
> anova(lm(r9.valor~r9.ano))
> sink()
>

```

Los resultados de esos comandos los hallamos abriendo en cualquier editor de texto el archivo 'result.txt':

Call:

```
lm(formula = r9.valor ~ r9.ano)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.90764	-0.36693	-0.03728	0.12558	1.33666

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-38.91191	3.91815	-9.931	7.92e-07 ***
r9.ano	0.52964	0.04389	12.067	1.10e-07 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7046 on 11 degrees of freedom

Multiple R-Squared: 0.9298, Adjusted R-squared: 0.9234

F-statistic: 145.6 on 1 and 11 degrees of freedom, p-value: 1.099e-007

Analysis of Variance Table

Response: r9.valor

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
r9.ano	1	72.288	72.288	145.6	1.099e-07 ***
Residuals	11	5.461	0.496		

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

## 2.5 Ejecuciones Condicionales y Loops

El lenguaje permite la construcción de condiciones para el control del flujo como cualquier lenguaje de programación.

**if....else**

El uso de este condicional es el siguiente:

```
> if (expresión 1) expresión 2 else
expresión 3
```

donde si la expresión 1 es cierta se evalúa la expresión 2. En caso contrario se evalúa la expresión 3. Si la expresión 2 ó expresión 3 son complejas, esto es, tienen más de un comando entonces deben encerrarse entre llaves:

```
> if (expresión 1) {
+ ...comandos R...
+ }
+ else {
+ ...comandos R...
+ }
```



## Cuidado:

En lenguajes vectorizados se pueden cometer errores que son difíciles de detectar. Por ejemplo, supongamos que tenemos la variable `sexo` codificada como (1, 2) y supongamos que deseamos recodificarla a (1, 0). La siguiente instrucción, que en su estructura es similar a la que utilizan muchos lenguajes, produce un error:

```
> if(sexo==2) sexo<-0 else sexo<-1
```

Una solución es la siguiente:

```
> sexo1<-rep(NA, length(sexo))
> sexo1[sexo==2]<-0
```

```
> sexo1[sex!=2]<-1
```

## ifelse

Otra solución al problema anterior es utilizar la función `ifelse()`

```
> sexo<-ifelse(sexo==2,0,1)
```

## for

Para loops el R proporciona varias formas. La más usada es

```
> for ( nombre in expresión1) expresión 2
```

`nombre` es una variable de control del loop. `expresión1` es un vector, por ejemplo del tipo `1:20`; y `expresión2` usualmente es un conjunto agrupado de instrucciones a ser repetidas a medida que cambie la variable que controla el loop.

En lenguajes vectorizados, como el R, el uso de loops (ciclos) no es recomendado. Uno de los problemas que se presenta es que el programa no libera memoria sino cuando se regresa al prompt. De esta forma cualquier operación iterativa puede requerir mucha memoria (no necesariamente por un número grande de iteraciones, con números tan pequeños como 10 puede ocurrir). Esto causa que el computador se quede sin memoria o que se presente un uso intensivo del disco duro. Otro síntoma, cuando se trabaja en estaciones con posibilidad de realizar múltiples tareas simultáneamente o con varios usuarios, es que el sistema se vuelva en extremo lento. Algunos remedios pueden ser los siguientes:

- 1) Evite las iteraciones tanto como sea posible utilizando los arreglos y las operaciones sobre listas (ejemplos: `apply`, `lapply`; además de las operaciones matriciales corrientes).
- 2) Use `For` (con `F` mayúscula) para iteraciones.

3) Codifique la iteración básica en Fortran o C y llámela desde el programa. Esto requiere un esfuerzo extra de programación y puede no funcionar en algunas plataformas.

Otras funciones para controlar loops son

> repeat expresión

y el comando

> while ( condición) expresión

## 2.6 Objetos en R

La información es manipulada en R es en forma de objetos. Ejemplos de objetos son vectores de valores numéricos (reales) o valores complejos, vectores de valores lógicos y vectores de caracteres. Estos son conocidos como estructuras "atómicas" ya que sus componentes son todos del mismo tipo o modo. Las mismas funciones del R son objetos.

R también opera con objetos llamados listas, que son del modo list. Estos son secuencias ordenadas de objetos que individualmente pueden ser de cualquier tipo.

Un tipo de objeto importante en R es el arreglo de doble entrada, u objeto matriz. Para crear una matriz usamos la función `matrix()`. Toma como argumentos un vector y dos números que especifican las filas y las columnas. Por ejemplo:

```
> matrix(1:16,nrow=4,ncol=4)
```

crea una matrix de 4x4 con los enteros desde el 1 hasta el 16. Si queremos llenar la matriz "por filas", usamos la opción `byrow=T` en `matrix`. En el ejemplo anterior pudimos haber escrito

```
> matrix(1:16,ncol=4,byrow=T)
```

Otro objeto con el que trabaja R es el objeto lista, es el más general y flexible para guardar datos. Una lista es una colección ordenada de componentes. Estos componentes pueden ser de diversas clases. Podemos tener una lista con dos componentes: un vector de caracteres y una matriz de números. Se puede crear una lista con la función `list()`. Por ejemplo,

```
> x<-list(a=1:10, b=c('manzana', 'pera'))
```

nos crea una lista, llamada `x`, que tiene componentes `a` y `b`. `a` es un vector de números y `b` es un vector de caracteres. Observe que los componentes tienen un número diferente de componentes. Si queremos hacer referencia al componente `a` del objeto `x` escribimos `x$a`.

```
> x$a
```

nos mostrará los elementos de `a`.

Muchas funciones en R cuando son ejecutadas entregan un objeto. Este objeto debe ser manipulado en la forma descrita para obtener los resultados deseados.

## 2.7 Para Remover Objetos

Si objetos viejos no son removidos; eventualmente, el sistema podría coparse el disco duro con estos objetos y colapsar. Por esto es necesario hacer una labor de limpieza regularmente. Dentro de una sesión R se deben eliminar los objetos que no se utilizarán nuevamente. Para hacer esto se usa la función `rm()`. El argumentos de esta función es la lista de objetos, separados por comas, que se quieren eliminar.

Si se quiere saber que objetos hay en el sistema se usa la función `objects()`, cuyo resultado es una lista de los objetos presentes.