Álgebra lineal y cálculo con R

Daniel Liviano Solís
Maria Pujol Jover

PID_00208272

Universitat Oberta de Catalunya

Índice

Introducción 5

Objetivos 7

1. Álgebra vectorial y matricial 9
1.1. Operaciones algebraicas 9
1.1.1. Suma y resta de escalares, vectores y matrices 9
1.1.2. Producto y división elemento por elemento 11
1.1.3. Producto escalar entre dos vectores 12
1.1.4. Norma de un vector 13
1.1.5. Producto matricial 13
1.1.6. Producto de Kronecker 15
1.2. Ecuaciones algebraicas 16
1.3. Sistemas de ecuaciones lineales 17
1.4. Descomposición matricial 18
1.4.1. Vectores y valores propios 18
1.4.2. Descomposición en valores singulares 19

2. Funciones 21
2.1. Introducción 21
2.2. Clases de funciones 22
2.3. Representación gráfica 23
2.3.1. Gráficos de una función 23
2.3.2. Gráficos de varias funciones 24
2.3.3. Gráficos en tres dimensiones 26
2.4. Ejemplos de funciones en diversos campos 28
2.4.1. Física: teoría de la relatividad 28
2.4.2. Ingeniería: la función catenaria 29
2.4.3. Finanzas: análisis de inversiones 31
2.4.4. Economía: funciones de oferta y demanda 32

3. Cálculo diferencial e integral 34
3.1. Aproximación a la derivada 34
3.2. Integración 38
3.3. Ecuaciones diferenciales 42

4. Optimización 45

5. Análisis de variable compleja 48

Bibliografía 52

Introducción

El presente módulo tiene como objetivo profundizar más en aspectos matemáticos que en estadísticos. De hecho, se podría decir que el objetivo de este módulo 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, 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 han 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 llevando 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 en R que permiten realizar cálculos de esta manera[1]. 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. Los temas que se incluyen en el análisis numérico cubren casi todos los ámbitos de las matemáticas y la estadística. He aquí los más destacados:

1) Cálculo de los valores la función.

2) Interpolación, extrapolación y regresión.

3) Resolución de ecuaciones y sistemas de ecuaciones.

4) Estudio de problemas de descomposición matricial: vectores y valores propios, y valores singulares.

5) Optimización lineal y cuadrática.

[1] Por ejemplo, algunas librerías que permiten hacer cálculo simbólico con R son *rSymPy*, *ryacas* y *mosaic*.

6) Evaluación de integrales mediante integración numérica.

7) Estudio de ecuaciones diferenciales.

Por todo esto, desde aquí animamos a todos aquellos estudiantes que estudien en sus asignaturas algunos de estos temas a que le den una oportunidad a R, ya que les permitirá consolidar los conocimientos teóricos adquiridos, además de ponerlos en práctica con datos reales o simulados.

Objetivos

1. Aprender a manejar vectores y matrices y a hacer operaciones entre ellos.

2. Dominar las principales técnicas de descomposición matricial.

3. Encontrar la raíz de ecuaciones algebraicas y resolver sistemas de ecuaciones.

4. Estudiar y representar gráficamente funciones de diferentes clases.

5. Aproximar numéricamente el cálculo de derivadas e integrales.

6. Saber resolver problemas de valor inicial con ecuaciones diferenciales ordinarias y parciales.

7. Efectuar algoritmos de optimización lineal y cuadrática.

8. Dominar la notación y las operaciones básicas con números y variables complejas.

1. Álgebra vectorial y matricial

1.1. Operaciones algebraicas

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

1.1.1. Suma y resta de escalares, vectores y matrices

R permite realizar operaciones vectorizadas, lo cual permite 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:

$$c \pm \boldsymbol{v} = c \pm (v_1, \ldots, v_n) = (c \pm v_1, \ldots, c \pm v_n)$$

```
> 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:

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

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, $(5, 4, 3, 2, 1)$.

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

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

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

Consejo para programar ordenadamente

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

La suma y resta de matrices se realiza elemento por elemento, de manera que estas han de tener la misma dimensión; si no, 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:

```
> (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
```

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 a abajo y de izquierda a derecha.

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
```

1.1.2. 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.

$$c\,\boldsymbol{v} = c\,(v_1, \ldots, v_n) = (c\,v_1, \ldots, c\,v_n)$$

$$c/\boldsymbol{v} = c/(v_1, \ldots, v_n) = (c/v_1, \ldots, c/v_n)$$

```
> 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.

```
> (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
```

¡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**.

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
```

1.1.3. Producto escalar entre dos vectores

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

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.

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

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

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

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

Ortogonalidad

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

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

1.1.4. Norma de un vector

También denominada la *magnitud* del 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 $\boldsymbol{v} = (v_1, \ldots, v_n)$ viene definida por la siguiente fórmula:

Espacio euclidiano

En geometría, el concepto de espacio euclidiano abarca el plano euclidiano para un espacio de 2 dimensiones y el espacio euclidiano para un espacio de 3 o más dimensiones. Se denomina así en honor al matemático griego antiguo Euclides de Alejandría.

$$\|\boldsymbol{v}\| = \sqrt{v_1^2 + \cdots + v_n^2}$$

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 necesitemos.

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

Pitágoras tenía razón

Geométricamente, en el plano euclidiano la norma se puede interpretar como la longitud de la hipotenusa, en este caso $\sqrt{1^2 + 1^2} = \sqrt{2} = 1{,}414214$.

1.1.5. Producto matricial

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

¡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.

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

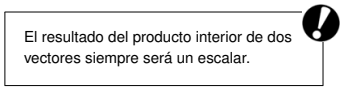
$$\begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

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 $\boldsymbol{a}$ y $\boldsymbol{b}$ de longitud n como vectores columna $\boldsymbol{a}_{1\times n}$ y $\boldsymbol{b}_{1\times n}$, el producto 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:



$$\boldsymbol{a}\cdot\boldsymbol{b}=\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}=\begin{pmatrix}a_1 & a_2 & \cdots & a_n\end{pmatrix}\begin{pmatrix}b_1\\ b_2\\ \vdots\\ b_n\end{pmatrix}=a_1b_1+a_2b_2+\cdots+a_nb_n=\sum_{i=1}^{n}a_ib_i$$

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 A^TB 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:

El resultado del producto exterior de dos vectores siempre será una matriz.

$$\boldsymbol{a} \otimes \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^{\mathrm{T}} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n \end{pmatrix}.$$

Aplicamos esta operación en R con el operador `%o%`, que calcula AB^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
```

1.1.6. 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 ⊗, 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
```

1.2. Ecuaciones algebraicas

Este tipo de ecuaciones involucra a polinomios. Recordemos que un polinomio de grado n toma 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 $(a_0, a_1, \ldots, a_n)$, luego restarlos y buscar la raíz mediante la función `polyroot`, que es la solución. Veamos un sencillo ejemplo:

$$2x^2 + 5x + 5 = x^2 + 3x + 4$$

$$x^2 + 2x + 1 = 0$$

Un número complejo se representa en forma binomial como:

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 $(-1, -1)$.

```
> 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 $x = -1 \pm 0 \cdot i = -1$.

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

1.3. 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:

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

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:

$$3x + y + 2z = 10$$
$$4x + 3y + 4z = 21$$
$$2x + y + 2z = 9$$

Primero expresamos el sistema mediante matrices:

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

```
> 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
```

1.4. Descomposición matricial

1.4.1. 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 $\boldsymbol{v}$ 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:

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

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

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

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:

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

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

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

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

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 $.

En este caso, hemos obtenido dos valores propios ($\lambda_1 = \lambda_2 = 2$) y dos vectores propios ($\boldsymbol{v}_1 = (0, 1)$ y $\boldsymbol{v}_2 = (-1, 0)$). Es interesante apuntar que siempre obtendremos tantos valores y vectores propios como la dimensión de la matriz de transformación A.

1.4.2. 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. Partiendo de una matriz A, igual que en el caso anterior, la DSV efectúa la siguiente operación:

DVS

Esta descomposición tiene muchas aplicaciones en estadística y otras disciplinas, como el procesamiento de señales en ingeniería.

$$A = UDV^T$$

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 estos apuntes. Las principales son la factorización QR (función `qr`) y la factorización de Cholesky (función `chol`).

2. Funciones

2.1. Introducción

En este capítulo veremos las múltiples posibilidades para trabajar con funciones matemáticas del tipo $y = f(x)$ que nos ofrece R. La función básica, explicada en el módulo 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:

```
> 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
```

Evaluación de funciones

En este ejemplo, hemos creado un vector de valores iniciales x_0 con los valores $(-3, -2, \ldots, 2, 3)$ y los hemos evaluado con la función creada $y(x)$.

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 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
```

2.2. 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, $y = f(x_1, \ldots, x_n)$.

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 1:

Tabla 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.

2.3. Representación gráfica

2.3.1. Gráficos de una función

Una de las principales ventajas de R sobre otros programas es la potencia y versatilidad de sus gráficos. En este apartados solo 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 el siguiente cuadro:

Tabla 2. Opciones gráficas elementales

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

Parámetros gráficos

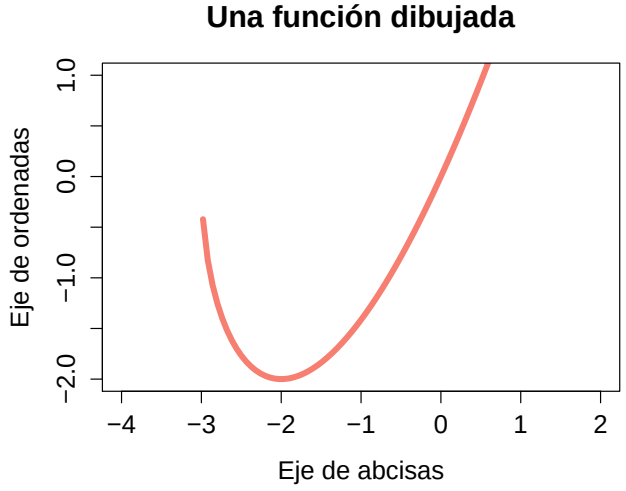
Si acudimos a la ayuda de R, introduciendo en la consola la instrucción `help(par)`, obtendremos una extensa lista de opciones gráficas, con las que podremos personalizar y moldear todos los aspectos del gráfico.

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

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

Una representación de la función $y = x\sqrt{x+3}$ podría ser la siguiente. El resultado se muestra en la figura 1:

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



```
> plot(y,
+      main="Una función dibujada",
+      xlab="Eje de abcisas",
+      ylab="Eje de ordenadas",
+      xlim=c(-4,2),
+      ylim=c(-2,1),
+      type="l",
+      lwd=5,
+      col="salmon")
Warning message:
In sqrt(x + 3) : NaNs produced
```

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.

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.

2.3.2. 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 1001:

```
> 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 $(1, 1001)$ para las abscisas y $(-4, 4)$ para las ordenadas:

```
> (xrange <- c(1,lon))
[1]    1 1001

> (yrange <- c(-4,4))
[1] -4  4

> plot(xrange,yrange,type="n",xaxt="n",xlab="",ylab="")
```

Elaboración de un gráfico

Este paso nos ha creado un cuadro vacío. 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.

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="red")
> lines(y2,type="p",lwd=1,col="blue")
> lines(y3,type="p",lwd=1,col="orange")
> abline(h=0,lwd=2)
```

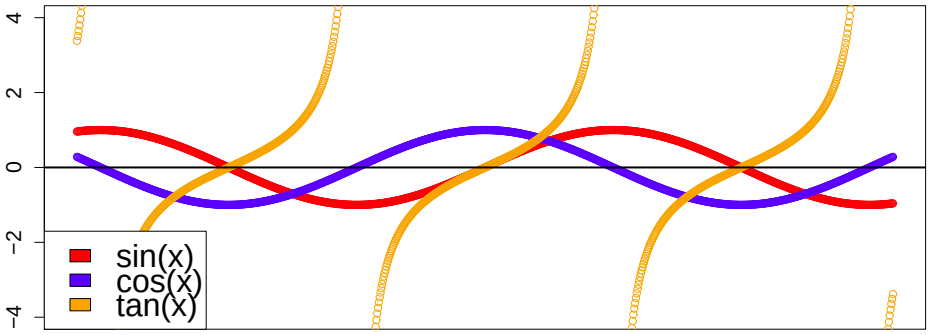
El último paso es añadir la leyenda, que se efectúa con la función `legend`. Aquí especificamos dónde queremos la leyenda (abajo e izquierda), el texto asociado a cada función, el color de cada función y por último `cex`, que indica la proporción de la leyenda con relación al gráfico (un valor mayor indicará un tamaño mayor). El resultado final se muestra en la figura 2:

```
> legend("bottomleft",
+        c("sin(x)","cos(x)","tan(x)"),
+        fill=c(" red","blue","orange"),
+        cex=1.5)
```

Parámetros de la leyenda

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

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



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

$$f(x) = \begin{cases} 4x^2 + \frac{x!}{x^x} & \text{si } x > 0 \\ 5 & \text{si } x = 0 \\ 6\log(|x|) & \text{si } x < 0 \end{cases}$$

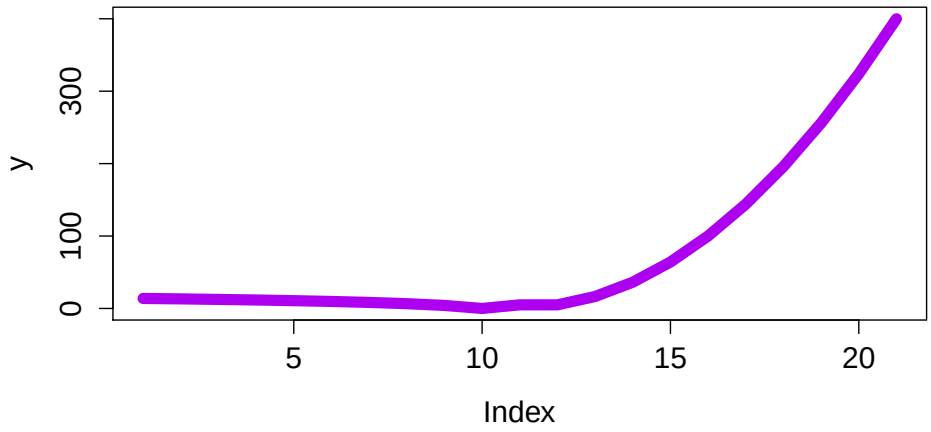
Supongamos que queremos evaluar dicha función y representarla gráficamente en el intervalo $x \in (-10, 10)$. Para ello hemos de utilizar ciclos y condicionales, tal y como se muestra a continuación:

```
> x0 <- (-10:10)
> n <- length(x0)
> y <- rep(0,n)
> for (i in 1:n){
+     x <- x0[i]
+     if (x>0){
+         y[i] <- 4*x*x + (factorial(x)/x^x)
+     }else if (x==0){
+         y[i] <- 5
+     }else{
+         y[i] <- 6*log(abs(x))}
+ }

> plot(y,type="l",lwd=8,col="purple")
```

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.

Figura 3. Representación de una función definida a trozos



2.3.3. 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 paraboloide 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
```

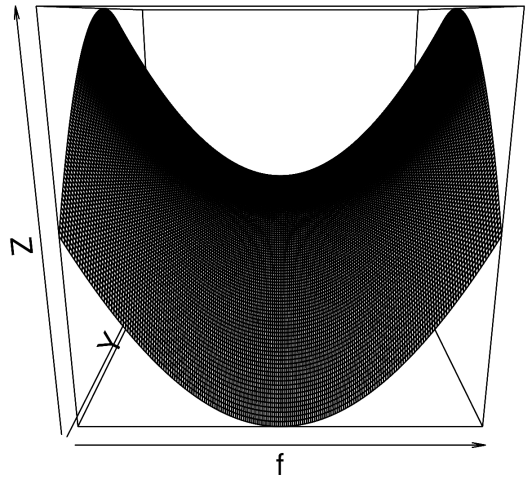
Llegados a este punto, existe una función de R llamada `outer` que resulta de gran utilidad. Esta función utiliza como argumentos los vectores $\boldsymbol{x} = (x_1, \ldots, x_n)$ e $\boldsymbol{y} = (y_1, \ldots, y_n)$ y una función $f(x, y)$, y calcula la siguiente matriz $n \times n$:

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

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 4.

```
> f <- outer(x0,y0,z)
> persp(f,col="gold")
```

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



2.4. Ejemplos de funciones en diversos campos

2.4.1. Física: teoría de la relatividad

La teoría de la relatividad establece que la masa de un cuerpo se incrementa con su velocidad. Así, la fórmula de la *masa relativística* establece la siguiente relación:

$$M = \frac{m}{\sqrt{1 - \frac{v^2}{c^2}}}.$$

Donde la masa M depende de la masa del cuerpo en reposo m, su velocidad v y la velocidad de la luz en el vacío $c = 3,0 \times 10^5$ km/s. El primer paso es introducir la función `M`:

```
> M <- function(m,v) m/sqrt(1-(v^2)/(300000^2))
```

La masa en la teoría de la relatividad

La masa relativista es la masa que se le asigna a un cuerpo en movimiento. Antes de la teoría de la relatividad, la mecánica clásica establecía que el momento lineal de un objeto venía dado por la fórmula $p = mv$, y su energía $E = E_0 + \frac{1}{2}mv^2$, siendo E_0 la energía en reposo (un concepto indeterminado) y $\frac{1}{2}mv^2$ la energía cinética. La teoría de la relatividad especial completó estas definiciones definiendo la masa en función de la velocidad del objeto en relación con la velocidad de la luz c, esto es, $M = \gamma m$, siendo $\gamma = 1/\sqrt{1 - v^2/c^2}$.

1. Si una persona de 100 kg viajara por el espacio a 2.000.000 km/h, ¿cuál sería su masa?

Primero hemos de convertir la velocidad de km/h a km/s, y luego evaluar la función con estos datos.

```
> v0 <- 2000000/3600

> M(100,v0)
[1] 100.0002
```

Es decir, habría aumentado su masa en 2 decigramos, es decir, 0,2 gramos.

2. ¿A qué velocidad ha de viajar esta persona para aumentar su masa en un 10 *%?*

Para resolver esta cuestión hemos de utilizar la función `uniroot`, que busca las raíces de una función de manera que se cumpla $f(x, ...) = 0$. En nuestro caso, estamos buscando el valor de la velocidad v_0 que cumpla la siguiente condición:

$$\frac{100}{\sqrt{1 - \frac{v_0^2}{c^2}}} - 110 = 0.$$

Si creamos una segunda función (`M2`) con la velocidad como argumento, podemos encontrar el valor que buscamos:

```
> M2 <- function(v) 100/sqrt(1-(v^2)/(300000^2))-110

> uniroot(M2,c(0,200000))$root
[1] 124979.3
```

Esto es, 124,979, 3 km/s.

3. ¿Cómo se representa gráficamente la relación entre la masa de esta persona y su velocidad?

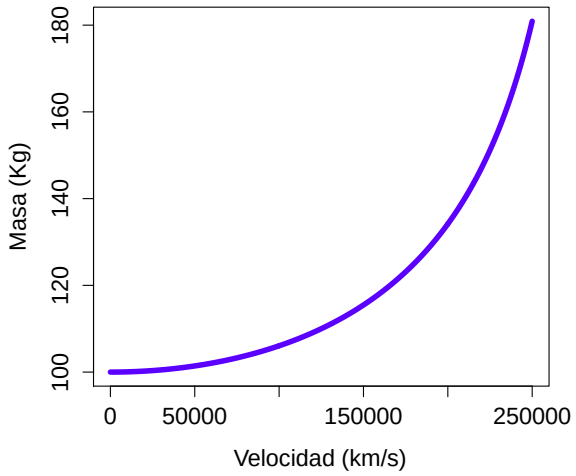
Primero creamos el vector v_0 de valores iniciales, evaluamos la función `M` con los valores de v_0 y con el valor fijado $m = 100$. Una vez tenemos el vector `y`, procedemos a elaborar el gráfico (figura 5). Es interesante destacar que mediante la función `axis` hemos personalizado el eje de abscisas, haciendo que se marquen solo los puntos del vector `lab` con sus correspondientes valores de v_0.

Personalizando el eje de abscisas

Os recomendamos que repliquéis este ejemplo en vuestro ordenador, prestando atención especialmente a los valores de `lab` y `v0[lab]`.

```
> v0 <- seq(0,250000,by=100)

> y <- M(v0,m=100)

> plot(y,type="l",lwd=5,col="blue",xaxt="n",
+ xlab="Velocidad (km/s)",ylab="Masa (Kg)")

> lab <- c(1+500*0:5)
> axis(1,at=lab,labels=v0[lab])
```

Figura 5. Representación de la función de la masa relativística



2.4.2. Ingeniería: la función catenaria

La función catenaria es la curva que describe una cadena suspendida por sus extremos, sometida a un campo gravitatorio uniforme. Matemáticamente, la función que hay que estudiar es la siguiente[1]:

$$y = a\,\cosh\left(\frac{x}{a}\right) = \frac{a}{2}\left(e^{x/a} + e^{-x/a}\right).$$

La función catenaria

Los primeros matemáticos que trataron este problema supusieron, erróneamente, que esta curva era una parábola. Finalmente, la ecuación fue obtenida por Gottfried Leibniz, Christiaan Huygens y Johann Bernoulli en 1691.

[1] Recordemos que la función coseno hiperbólico se define como $\cosh(x) = \frac{1}{2}\,(e^x + e^{-x})$.

El parámetro a es el cociente entre la tensión horizontal y el peso por unidad de longitud. El primer paso es introducir la función en lenguaje de R:

```
> f.cat <- function(x,a){
+   f1 <- exp(x/a)
+   f2 <- exp(-x/a)
+   return(0.5*a*(f1+f2))
+ }
```

1. Cread una representación gráfica de tres curvas con $a = 0,5$, $a = 1$ *y* $a = 1,5$.

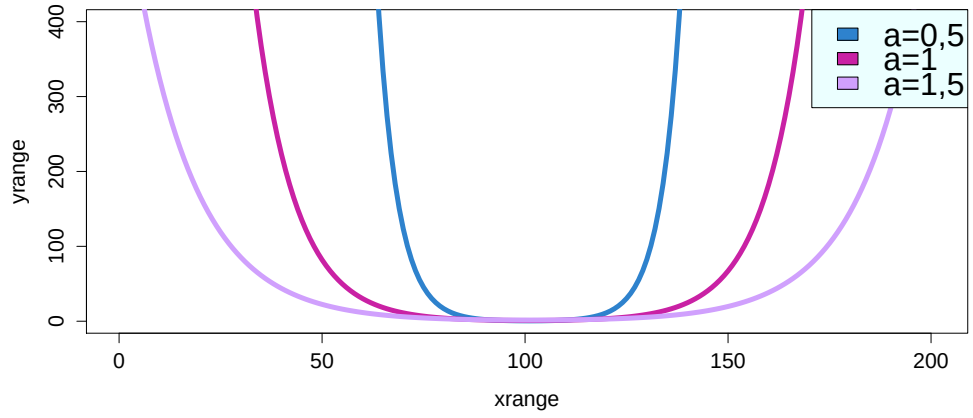
Igual que hemos hecho anteriormente, evaluaremos la función para estos tres valores de a en el rango $x \in (-10, 10)$, con lo que obtenemos `y1`, `y2` e `y3`. Finalmente creamos el gráfico con la leyenda correspondiente (figura 6).

Posición de la leyenda

Fijaos en que en este caso hemos situado la leyenda en la parte superior derecha del gráfico (`topright`).

```
> x0 <- seq(-10,10,by=0.1)

> y1 <- f.cat(x0,a=0.5)
> y2 <- f.cat(x0,a=1)
> y3 <- f.cat(x0,a=1.5)

> xrange <- c(0,length(x0))
> yrange <- c(0,400)

> colores <- c("steelblue","violetred","plum")

> plot(xrange,yrange,type="n")
> lines(y1,type="l",lwd=5,col=colores[1])
> lines(y2,type="l",lwd=5,col=colores[2])
> lines(y3,type="l",lwd=5,col=colores[3])
> legend("topright",c(" a=0,5"," a=1"," a=1,5"),
         fill=colores,cex=1.5)
```

Figura 6. Representación de curvas catenarias

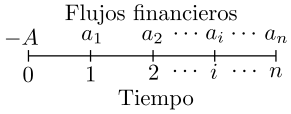


2.4.3. Finanzas: análisis de inversiones

Valor actual neto (VAN)

El VAN compara el valor de un euro hoy con el valor de ese mismo euro en el futuro, teniendo en cuenta la inflación y los flujos de caja que el proyecto genere en el futuro. Si el VAN de un proyecto es positivo, este puede ser aceptado. Sin embargo, si el VAN es negativo, el proyecto probablemente debería ser rechazado porque los flujos de caja actualizados con la tasa de retorno no superarían la inversión inicial.

Un proyecto de inversión se compone de una inversión inicial $-A$ y una serie de flujos futuros esperados a_i durante los n periodos de la inversión. La estructura temporal es la siguiente:



Los dos principales indicadores para evaluar un proyecto de inversión son el valor actual neto (VAN) y la tasa interna de retorno (TIR). El VAN consiste en evaluar en el periodo $t = 0$ la inversión inicial y los flujos a_i aplicando una tasa de descuento r:

$$VAN = -A + \frac{a_1}{(1+r)} + \frac{a_2}{(1+r)(1+r)} + \ldots + \frac{a_n}{(1+r)(1+r)\cdots(1+r)}.$$

$$VAN = -A + \sum_{i=1}^{n} \frac{a_i}{(1+r)^i}.$$

Tasa interna de retorno (TIR)

La TIR es la tasa de descuento que provoca que el valor presente neto de los flujos de efectivo de un proyecto sea igual a cero. En términos generales, cuanto mayor sea la TIR de un proyecto, será más recomendable llevarlo a cabo. Por este motivo, la TIR se puede utilizar para clasificar varios proyectos potenciales considerados por una empresa.

La lógica de este indicador es que los flujos futuros valen menos que los flujos presentes, ya que el valor del dinero cambia con el tiempo. Así, el valor en $t = 0$ de a_i se verá reducido por un factor $(1 + r)^i$. Si evaluamos la inversión tomando un valor elevado de r, el VAN será menor. Por su parte, la TIR se define como el valor de r que hace que el VAN sea cero, es decir, que se cumpla $VAN = 0$. Una TIR positiva implica que $A < \sum_{i=1}^{n} a_i$, y por tanto se recupera la inversión inicial.

1. Se considera un proyecto que requiere una inversión inicial de $10,000$ *euros y una vida útil de 20 años. Suponiendo un flujo constante de efectivo de* 700 *euros y una tasa de descuento del* $3\,\%$*, ¿cuál es el valor actual neto de este proyecto?*

El primer paso será crear la función `VAN` que tenga como argumentos el vector $V = (-A, a_1, \ldots, a_n)$ y la tasa de descuento r:

```
> VAN <- function(V,r) sum(V/(1+r)^(0:(length(V)-1)))
```

Una vez introducida esta función, solo hay que introducir los valores iniciales y evaluar la función `VAN`:

```
> A <- 10000
> a <- 700
> n <- 20
> v0 <- c(-A,rep(a,n))

> VAN(v0,0.03)
[1] 414.2324
```

Este resultado indica que la inversión tiene un valor neto positivo con $r = 3\,\%$.

2. ¿Cuál es la TIR de esta inversión?

Para calcular la TIR hay que buscar el valor de r que iguala la función del `VAN` a cero, lo que equivale a buscar la raíz de la función. Para contemplar valores tanto positivos como negativos de r fijamos un rango de búsqueda `c(-1,1)`. Además, especificamos cuál es el vector de valores iniciales (`v0`).

```
> uniroot(VAN,c(-1,1),V=v0)$root
[1] 0.03443455
```

Según este criterio, la rentabilidad de esta inversión es del $3,44\,\%$.

2.4.4. Economía: funciones de oferta y demanda

Supongamos un mercado determinado donde el precio y la cantidad de equilibrio (p^* y q^* respectivamente) se determinan a partir del punto de intersección de las funciones de oferta (S) y demanda (D), que dependen de la cantidad q:

$$S(q) = 10 + \frac{3}{q}$$

$$D(q) = 100 \cdot q^2$$

La ley de la oferta y la demanda

En economía, esta teoría explica la interacción entre la oferta de un recurso (*supply* en inglés) y la demanda de ese recurso. La ley de la oferta y la demanda define el efecto que la disponibilidad de un producto y la demanda de este tienen sobre la cantidad producida y su precio de mercado. En general, si hay una oferta baja y una alta demanda, el precio será más alto. Por el contrario, cuanto mayor es la oferta y más baja la demanda, menor será el precio.

1. Representad gráficamente las funciones de oferta y de demanda.

El primer paso es introducir las funciones analíticamente:

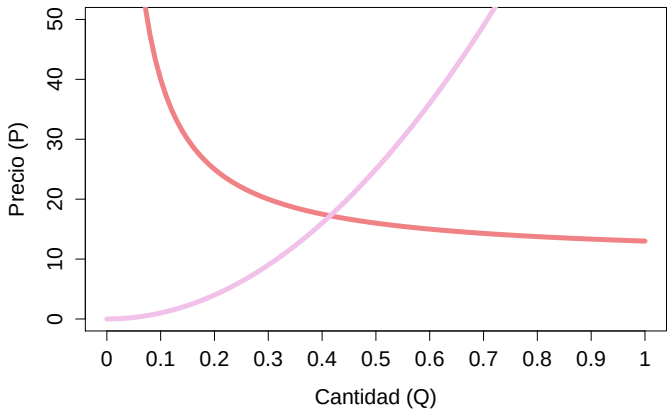
```
> S <- function(q) 10 + 3/q
> D <- function(q) 100*q^2
```

Similarmente a los casos estudiados anteriormente, la representación gráfica se inicia con la creación del vector de cantidades iniciales q_0. La figura 7 muestra el resultado.

```
> q0 <- seq(0,1,by=0.01)

> s0 <- S(q0)
> d0 <- D(q0)

> xrange <- c(1,length(q0))
> yrange <- c(0,50)

> plot(xrange,yrange,type="n",xaxt="n",
+      xlab="Cantidad (Q)",ylab="Precio (P)")
> lab <- 1+10*0:10
```

```
> axis(1,at=lab,labels=q0[lab])
> lines(s0,type="l",lwd=5,col="salmon")
> lines(d0,type="l",lwd=5,col="pink")
```

Figura 7. Funciones de oferta y demanda



2. ¿Cuáles son la cantidad y el precio de equilibrio?

R no ofrece (todavía) la posibilidad de sumar y restar funciones simbólicamente, con lo cual deberemos obtener estas magnitudes a partir de los vectores `s0` y `d0`. Si calculamos la diferencia `s0-d0`, obtendremos un vector con la diferencia de las funciones. El valor más cercano a cero en este vector (en valor absoluto) corresponderá al punto de equilibrio:

```
> dif <- abs(s0-d0)

> which.min(dif)
[1] 43
```

Este resultado indica que el valor número 43 del vector `q0` corresponde a la cantidad de equilibrio. El precio de equilibrio lo podemos obtener de las funciones de oferta y demanda, indistintamente. Como vemos, la cantidad y el precio de equilibrio serán aproximadamente $q^* \approx 0,42$ y $p^* \approx 17,15$.

```
> dif <- abs(s0-d0)

> v.min <- which.min(dif)

> (q.equil <- q0[v.min])
[1] 0.42

> (p.equil <- s0[v.min])
[1] 17.14286
```

3. Cálculo diferencial e integral

3.1. Aproximación a la derivada

El objetivo de esta sección es estudiar las posibilidades que R ofrece en cuanto a 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.

Cálculo

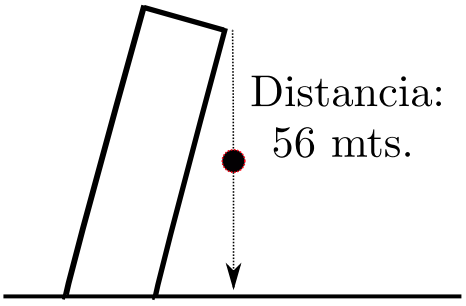
La palabra *cálculo* proviene del latín *calculus*, que era una pequeña piedra usada para contar. El cálculo es el estudio matemático del cambio, y tiene dos ramas principales: el *cálculo diferencial*, que estudia las ratios de cambio y las pendientes de las curvas, y el *cálculo integral*, que se encarga de la acumulación de cantidades y de las áreas bajo las curvas. Ambas ramas están relacionadas por el *teorema fundamental del cálculo*.

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[1]. 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 fFigura 8.

Figura 8. 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.

En primer paso para analizar este fenómeno en R es introducir la función de la caída, a la que llamaremos `s`:

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

[1] Experimentos 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.

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

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

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 (roja y 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 $\bar{v} = 56/3,38 = 16,56$ m/seg.

```
> plot(s,xlim=c(0,T),lwd=4,col='blue')
> lines(c(0,T),c(0,56),lty=2,lwd=4,col='red')
```

Figura 9. Función del movimiento y recta secante

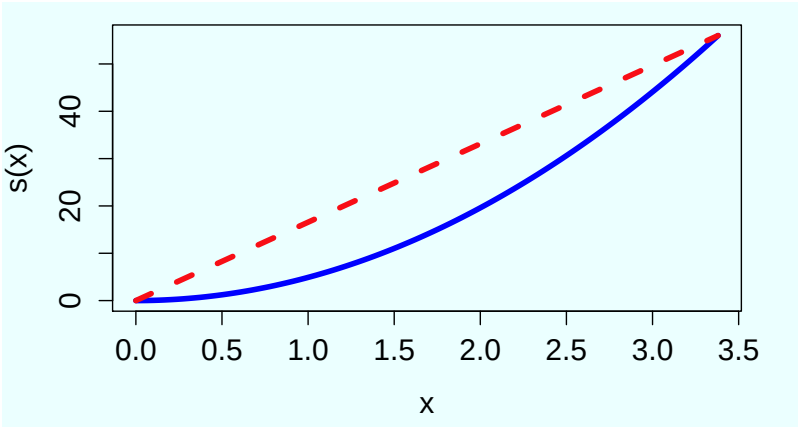


Figura 9

Fijaos 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 10:

Figura 10. Velocidad media entre dos puntos

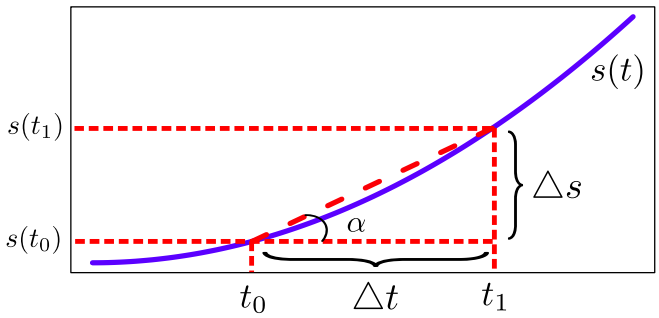


Figura 10

Este gráfico ilustra el concepto de derivada, que sería la pendiente de la línea roja 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.

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:

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

Donde $\triangle t$ 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 sen-

cillo: atribuimos a dt un valor arbitrariamente pequeño y creamos un vector de valores iniciales $t_0 = (0, dt, 2dt, \ldots, T)$. Seguidamente evaluamos la función con todos los valores del vector, esto es, $s(t_0) = (s(0), s(dt), s(2dt), \ldots, s(T))$. 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 $s(t)$:

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

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 11.

```
> 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='blue')
> lines(ds.dt,type='l',lwd=4,col='red')

> legend('bottomright',c('s(t)','ds/dt'),
+        fill=c('blue','red'),cex=1.5)
```

Figura 11. Función original y primera derivada

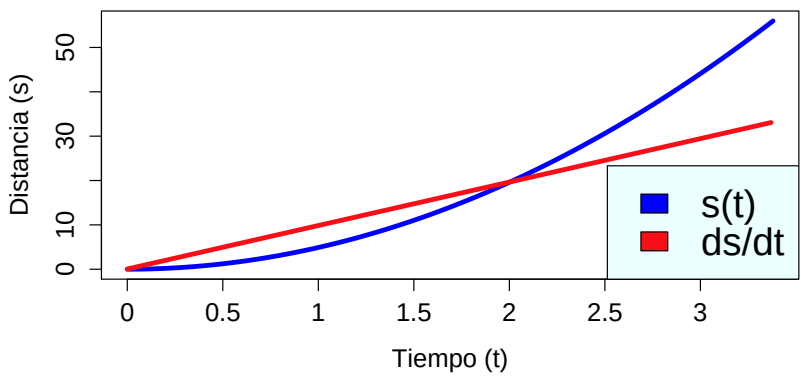


Figura 11

Aquí podemos observar cómo 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).

3.2. 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 ($\Delta s\ \Delta t$).

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 determinantes 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[2]:

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

Donde $g = 9,81$ m/s^2 es la aceleración gravitacional y v_∞ es la velocidad terminal, la cual 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 12:

```
> 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 12. Velocidad de un paracaidista en caída libre

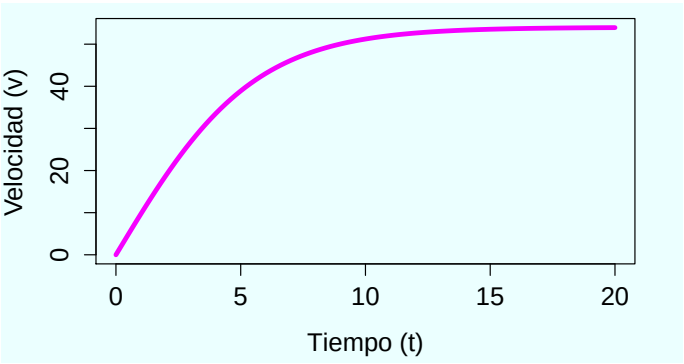


Figura 12

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 del 2012 logró una velocidad mucho mayor al saltar de una altura de 39 kilómetros, donde el aire ofrece mucha menos resistencia.

[2] Recordemos que la función tangente hiperbólica se define como $\tanh x = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$.

Este gráfico 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 13. La aceleración al principio es aproximadamente $v'(0) = 10$ y a medida que el tiempo avanza se aproxima asintóticamente a 0.

```
> t.0 <- 0:20

> plot(f.v(t.0),type='l',lwd=4,col='magenta',
+      xlab='Tiempo (t)',ylab='')

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

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

Figura 13. Velocidad y aceleración de un paracaidista en caída libre

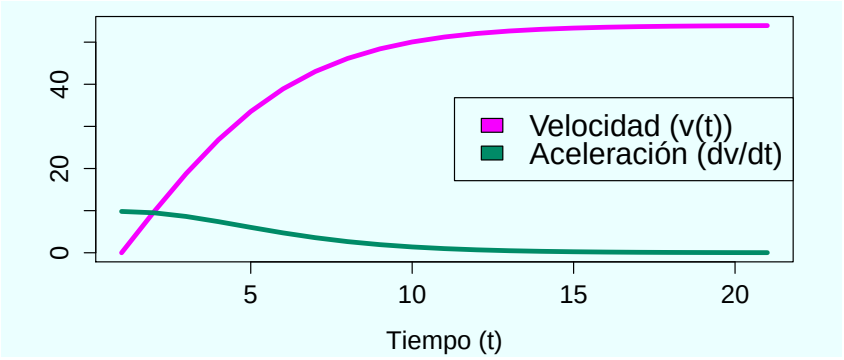


Figura 13

Recordemos que, en física, la aceleración indica el cambio de velocidad por unidad de tiempo.

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 10 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 3.

Tabla 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 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 14 ofrece una interpretación geométrica de los cálculos de distancia mostrados en la tabla 3:

Figura 14. Cálculo aproximado de la distancia

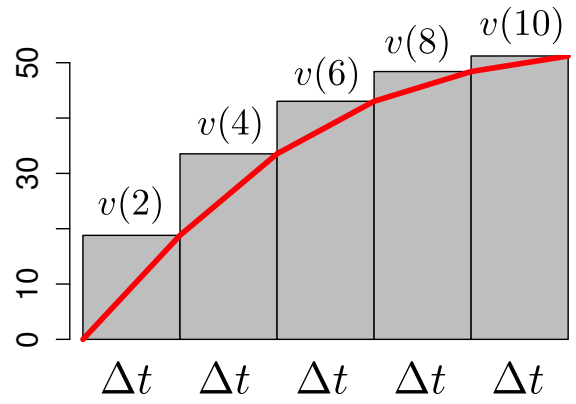


Figura 14

Es importante comprobar cómo la suma de las áreas de las cinco barras verticales se corresponde con los cinco valores obtenidos en la tabla 3, cuya suma es aproximadamente 390 metros. Vemos también cómo esta cantidad aproximada es superior a la real, ya que la suma de estas áreas es superior al área debajo de la curva.

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 14 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 $\Delta t \to 0$ y

$n \to \infty$. 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 $a = t_0 < t_1, \ldots, t_n < b$, siendo Δt la distancia entre los valores del intervalo.

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

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 $dt \to 0$:

```
> 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.

3.3. 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 matemáticas 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 $y = f(x, \ldots)$. Las ecuaciones diferenciales más sencillas se pueden resolver usando las reglas de diferenciación e integración. Veamos cómo se resolvería la ecuación diferencial $y' = cx$:

$$y' = cx \qquad \Rightarrow \qquad \frac{dy}{dx} = cx$$

$$dy = cx\,dx \qquad \Rightarrow \qquad \int dy = \int cx\,dx$$

$$y = \frac{cx^2}{2} + C$$

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 una **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[3]. Es un paquete muy completo, y en la página web del CRAN se puede encontrar un tutorial muy completo de cómo aplicar las diferentes aplicaciones de éste. 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.

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

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

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

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 $t_0 = 0, \ldots, 365$ días:

```
> library(deSolve)

> parametros<-c(k=0.02,M=10)

> cond.ini <- c(Y=0)

> tiempo <- 0:365
```

Nombres de las variables

Fijaos en que introduciendo los valores en vectores de esta manera estamos asignando un nombre a cada valor, lo cual es importante para los cálculos posteriores.

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 `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`:

¡Ojo! Si no hemos instalado y/o cargado el paquete `deSolve` correctamente, R no reconocerá la función `ode` y obtendremos un mensaje de error.

```
> 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 15.

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

Figura 15. Solución de una ecuación diferencial

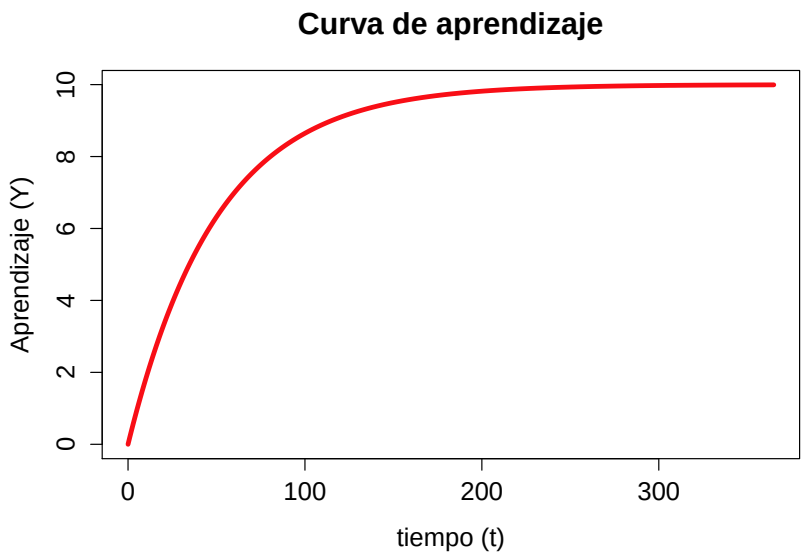


Figura 15

En esta curva se puede ver cómo 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 $Y = 10$.

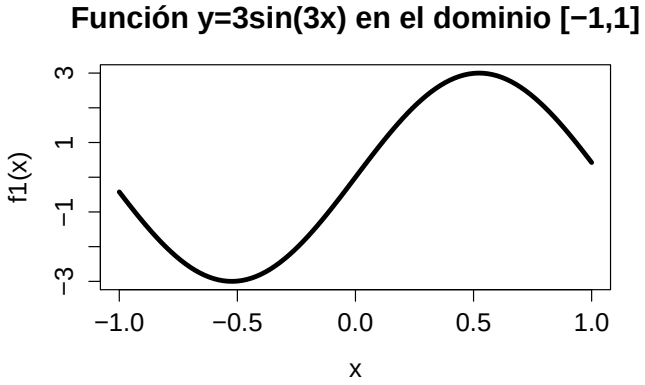
Hay que destacar que este es un ejemplo muy sencillo. R y el paquete `deSolve`, además de otros paquetes destinados a ecuaciones diferenciales, ofrecen muchas posibilidades para resolver sistemas de ecuaciones diferenciales y otros problemas más complejos.

4. 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 $y = 3sin(3x)$: la vamos a introducir y la vamos a representar gráficamente en el dominio $[-1, 1]$ en la figura 16:

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

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

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

Figura 16. Representación de la función $y = 3sin(3x)$



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 cómo 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 más 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, de la siguiente manera:

Argumentos de la función

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

Ahí encontraremos un elemento llamado *Optimization*, y si accedemos a él veremos los diferentes paquetes disponibles y sus capacidades.

5. 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 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
```

Una manera alternativa de expresar los números complejos es mediante **forma polar**, tal y como se muestra en la figura 17:

Figura 17. Número complejo en forma polar

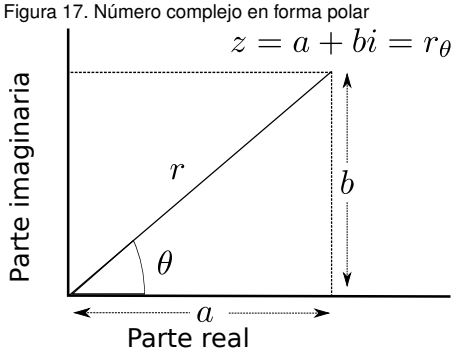


Figura 17

El número complejo $z = a + bi$ se puede expresar también 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 $z = r_\theta$.

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

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

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

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

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

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:

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

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: $(z_1, \ldots, z_n)$:

```
> 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 cómo $\sqrt{9}$ 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

Gibernans Bàguena, J.; Gil Estallo, À. J.; Rovira Escofet, C. (2009). *Estadística*. Barcelona: Material didáctico UOC.