Breve manual de Mathematica

Marzo 2022

Índice

Introducción a *Mathematica*

¿Para qué se puede utilizar *Mathematica*?

Para tener una visión global de lo que es el programa *Mathematica*, explicaremos primero cuales son sus posibilidades.

Mathematica se puede usar como:

- **Una calculadora:** podemos hacer las mismas operaciones (y muchas más) que cualquier calculadora de bolsillo.
- **Un paquete con rutinas de cálculo numérico:** con *Mathematica* se pueden hacer muchas operaciones que requieran de funciones o procedimientos, como integración numérica o programación lineal, porque ya están implementadas.
- **Una calculadora simbólica:** *Mathematica* está diseñado para hacer operaciones de matemática infinita, es decir, como si las hicieramos nosotros, utilizando parámetros, sustituyendo una variable por otra, etc.
- **Una potente herramienta de cálculo simbólico:** *Mathematica* permite derivar, integrar, calcular límites, resolver ecuaciones diferenciales, hacer operaciones con funciones, etc.
- **Un paquete gráfico:** con *Mathematica* se puede dibujar gráficos y objetos de 2 y 3 dimensiones. Se puede cambiar el punto de vista, cambiar de coordenadas. También hace animaciones.
- **Un lenguaje de programación de alto nivel:** *Mathematica* incluye un lenguaje de programación que reduce significativamente el tiempo de implementación respecto a otros lenguajes, como C o Pascal.
- **Un sistema para crear documentos interactivos:** este documento que lees ahora, está escrito mediante *Mathematica*. Igual que éste, se pueden crear documentos en los cuales se incluya textos, gráficos, ejecuciones de funciones y algoritmos, animaciones, sonidos, etc.
- **Un sistema que puede integrarse con otros programas:** *Mathematica* permite que otros programas puedan comunicarse con él y enviarle instrucciones para que las ejecute, y devuelva la respuesta al programa que se la ha solicitado.

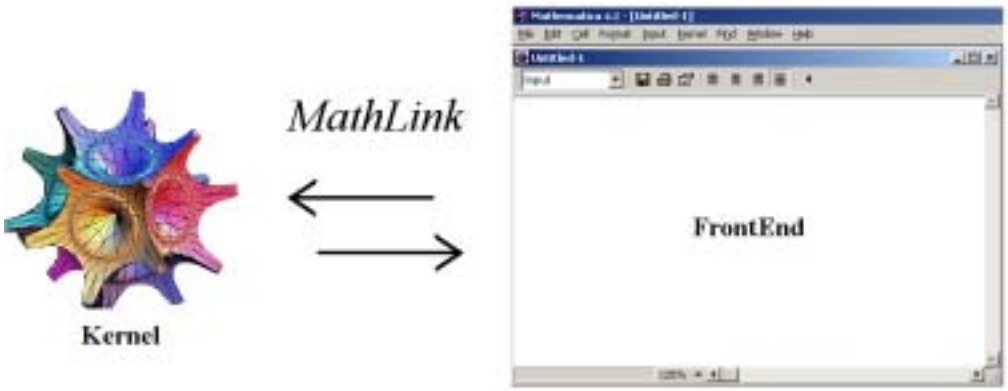
Introducirse en *Mathematica*

Mathematica es un paquete de cálculo simbólico. Ésto significa que puede trabajar con números y con símbolos. *Mathematica* trabaja con operaciones exactas, siempre que no forcemos a que trabaje numéricamente o que el volumen de cálculos obligue a pasar a aritmética de punto flotante para que no se produzca un desbordamiento *(overflow)*.

■ El FrontEnd y el Kernel

Mathematica consta de dos partes, el *Kernel* (o núcleo) y el *FrontEnd*. El *FrontEnd* es la interface que vemos, en la cual escribimos nuestras instrucciones. El *FrontEnd* interpreta las instrucciones que escribimos y las envía al *Kernel*. Éste las ejecuta y retorna el resultado al *FrontEnd* para que nos lo traduzca y lo presente en un formato que nosotros entendamos.

El lenguaje que permite la comunicación entre el *FrontEnd* y el *Kernel* se llama *MathLink*.



■ Notebooks y packages

Los *notebooks* son los ficheros en los cuales trabajamos con *Mathematica*. Estos ficheros tienen extensión **.nb**.

Los *packages* son ficheros de funciones específicas relacionadas de alguna manera. Por ejemplo, el directorio de *Mathematica Linear Algebra* contiene *packages* formados por funciones de álgebra lineal como *GaussianElimination*, *MatrixManipulation* o *Tridiagonal*. Cada uno de estos *packages* tiene funciones por el tratamiento de un aspecto del álgebra lineal. Estos ficheros tienen extensión **.m**. Para hacer uso, por ejemplo, de las funciones del *package Tridiagonal* que están en el directorio *Linear Algebra* se cargan con la instrucción siguiente:

```
<< LinearAlgebra`Tridiagonal`
```

o la siguiente:

```
Needs["LinearAlgebra`Tridiagonal`"]
```

Esta última instrucción tiene la ventaja de que si el *package* ya está cargado, no vuelve a cargarlo. Las comillas ` son el acento grave del teclado.

El símbolo **<<** se puede utilizar para cargar cualquier fichero con instrucciones de *Mathematica.* Por ejemplo, si queremos cargar un fichero de funciones definidas por nosotros que se llama *FuncMias.nb* y que está en el directorio *c:\math*, tendremos que escribir la instrucción:

```
<< c:\math\FuncMias.nb
```

■ Celdas

Las celdas son agrupaciones mínimas de información dentro del *notebook*. Las celdas son diferentes, dependiendo de su contenido.

Si son de línea sencilla con un triángulo en la parte superior, son ejecutables.

Si son de líneas dobles, es que está ejecutándose.

Si en la parte superior hay una pequeña raya doble transversal, es una celda que no podrá ejecutarse porque contiene texto o una imagen.

Si en el canto superior hay un triángulo y una raya, es una celda de respuesta a una ejecución.

■ Las paletas

Las paletas son *notebooks* que facilitan la escritura de instrucciones con símbolos matemáticos. Las paletas predefinidas de *Mathematica* se pueden obtener desde el menú *File-Palettes*. El usuario también puede definirse sus propias paletas e integrarlas en *Mathematica*. En la siguiente figura aparece la paleta *Basic Input.*

■ El botón derecho del ratón

El botón derecho del ratón activa un menú emergente con las acciones más comunes que se pueden realizar sobre la celda, como por ejemplo, copiar, pegar, evaluar, cambiar la celda de formato, etc.

■ Los menús

Los menús de *Mathematica*, como los de cualquier programa Windows, aparecen en la parte superior. Los menús de que consta *Mathematica* son:

- **File:** Con este menú podremos hacer las acciones clásicas de cualquier programa Windows en el mismo menú, es decir, crear ficheros nuevos (*New*), abrir ficheros (*Open...*), cerrar ficheros (*Close*), guardarlos (*Save* y *Save As...*), enviarlos por correo electrónico (*Sent To*), imprimirlos (*Print...*), salir del programa (*Exit*). Además, podremos gestionar las paletas de *Mathematica*.
- **Edit:** Con este menú podremos deshacer (*Undo*), cortar (*Cut*), copiar (*Copy*), pegar (*Paste*), borrar (*Clear*), y otras acciones de edición y traducción de entre los diversos formatos que gestiona *Mathematica*.
- **Cell:** Debido a que *Mathematica* se divide en celdas, se necesita un menú que nos permita gestionar las celdas, asignarles propiedades, unirlas, agruparlas, dividirlas, etc. Asimismo, este menú puede hacer acciones sobre imágenes, gráficos y sonidos.
- **Format:** Aquí podemos cambiar el formato de las celdas. Fuentes, tipos de letras, cabeceras, estilo del *notebook*, etc. También podemos cambiar la apariencia del área de trabajo, y ponerle una barra de herramientas en la cabecera del *notebook*, hacer que la letra aparezca más grande, o que muestre los saltos de página para saber como quedará el escrito cuando se imprima.
- **Input:** Este menú es un cajón de sastre, porque tiene lo que no cabe poner en otros menús. Aquí se gestionan los objectos 3D y sonidos, se crean botones y paletas, etc.
- **Kernel:** Aquí gestionaremos todo lo que tiene que ver con la conexión entre el *FrontEnd* y el *Kernel*. Conectarlo, desconectarlo, configurar la conexión, ejecutar instrucciones, interrumpirlas, abortarlas. Asimismo, se puede quitar/poner las etiquetas *In[]* y *Out[]* de las instrucciones ejecutadas y borrar todas las salidas (*Out[]*) del *notebook*.
- **Find:** Con este menú tenemos todas las opciones posibles para buscar, encontrar y reemplazar texto.
- **Window:** Menú típico de un programa Windows para organizar las ventanas, y seleccionar la deseada.
- **Help:** Aquí podemos acceder a todas las opciones de la ayuda. La ayuda ha mejorado mucho desde la versión 3.0, y permite editar y cambiar los ejemplos y ejecutarlos dentro de la misma ayuda. Es una de las mejores ayudas que hemos visto en un programa Windows y recomendamos que se use por su funcionalidad y las facilitades que tiene para buscar, por lo bien estucturada que está y la gran cantidad de ejemplos modificables que contiene.

■ Cómo comenzamos a escribir

Para comenzar a escribir, conviene situarse con el ratón en una zona del *notebook* donde el puntero del ratón se ponga horizontal. Estos sitios pueden ser entre dos celdas o después de la última celda.

Una vez el puntero del ratón esté horizontal, pinchamos y aparecerá una recta horizontal y ya podremos escribir.

■ Cómo ejecutamos una instrucción

Mathematica es un programa que para ejecutar una celda se necesita estar situado dentro de ella o tenerla seleccionada, y pulsar la combinación de teclas **<Shift + Enter>** o **<Mayúsculas + Intro>** (en la figura, combinación de teclas más oscuras) o simplemente la tecla **<Intro>** del teclado numérico.

■ Mayúsculas y minúsculas

Es importante resaltar que *Mathematica* distingue entre mayúsculas y minúsculas. Es decir, no es la misma función o variable "CASA", que "CaSa", que "casa". En cualquiera de los casos, cuando se evalúa una celda, automáticamente *Mathematica* detecta si hay variables o nombres de funciones parecidas y muestra un mensaje de advertencia (General::spelling).

■ El punto y coma (;)

En *Mathematica* el punto y coma tiene diversos usos:

- **Separar instrucciones dentro de un programa o procedimento.** Por ejemplo, la función siguiente calcula para cada x, el seno de x y tanto x como el seno de x se muestra por pantalla.

```
f[x_] := Module[{aux},
                aux = Sin[x];
                Print["Para el valor ", x, ", el seno vale ", aux, "."]]
f[π]
```

```
Para el valor π, el seno vale 0.
```

- **Evitar respuestas (*Out[]*) de *Mathematica* que no son interesantes o que no son relevantes.** Ésto puede pasar cuando definimos una variable o hacemos cálculos muy grandes que después usaremos pero que no tenemos interés de visualizarlos. Hemos de

tener en cuenta que la visualización de resultados largos es una de las cosas que más memoria del sistema consume. Veamos unos ejemplos. Primero cuando no usamos el punto y coma,

```
variable = {1, 2, 3}
```

```
{1, 2, 3}
```

mientras que si usamos el punto y coma

```
variable = {1, 2, 3};
```

Aquí la variable la hemos definido nosotros, por tanto no necesitamos que *Mathematica* nos responda con el valor de la variable.

■ Comentarios

En *Mathematica* los comentarios se escriben entre **(* comentario *)**. Por ejemplo,

```
g[x_] := Sin[x] (* Definimos la función g[x] como el seno de x *)
```

Cuando se ejecute la instrucción, lo que está dentro de **(* *)** no se evalúa.

■ ? ¡ ??

Pese a que ya hemos hablado de la ayuda (*Help*) de *Mathematica*, hay otras maneras de obtener información de una función, como son las instrucciones **?** y **??** Las dos tienen la ventaja de que podemos usar caracteres comodines. A continuación veremos diversos ejemplos para aclarar su funcionamiento.

? puede ayudarnos a localizar funciones con un patrón. Por ejemplo, la instrucción siguiente nos dará todas las funciones que tiene *Mathematica* y que comienzan por la palabra *Graphics*.

```
?Graphics*
```

```
Graphics          GraphicsArray     GraphicsGrouping
Graphics3D        GraphicsData      GraphicsSpacing
```

Con **?** también podemos obtener información sobre la definición de una función. Por ejemplo, de la funció *Graphics3D*

```
?Graphics3D
```

```
Graphics3D[primitives, options]
  represents a three-dimensional graphical image.
```

?? hace lo mismo que **?** pero obtenemos mucha más información sobre la función. Por ejemplo,

```
?? Graphics3D
```

```
Graphics3D[primitives, options]
  represents a three-dimensional graphical image.
```

```
Attributes[Graphics3D] = {Protected, ReadProtected}
```

```
Options[Graphics3D] =
 {AmbientLight → GrayLevel[0], AspectRatio → Automatic, Axes → False,
  AxesEdge → Automatic, AxesLabel → None, AxesStyle → Automatic,
  Background → Automatic, Boxed → True, BoxRatios → Automatic,
  BoxStyle → Automatic, ColorOutput → Automatic, DefaultColor → Automatic,
  Epilog → {}, FaceGrids → None, ImageSize → Automatic,
  Lighting → True, LightSources → {{{1., 0., 1.}, RGBColor[1, 0, 0]},
    {{1., 1., 1.}, RGBColor[0, 1, 0]}, {{0., 1., 1.}, RGBColor[0, 0, 1]}},
  Plot3Matrix → Automatic, PlotLabel → None, PlotRange → Automatic,
  PlotRegion → Automatic, PolygonIntersections → True,
  Prolog → {}, RenderAll → True, Shading → True,
  SphericalRegion → False, Ticks → Automatic, ViewCenter → Automatic,
  ViewPoint → {1.3, -2.4, 2.}, ViewVertical → {0., 0., 1.},
  DefaultFont :→ $DefaultFont, DisplayFunction :→ $DisplayFunction,
  FormatType :→ $FormatType, TextStyle :→ $TextStyle}
```

■ Operaciones básicas: suma, resta, multiplicación y división

En *Mathematica*, cuando estamos trabajando con números, la suma se hace con el signo **+**, la resta con el signo **–**, la multiplicación con ***** (¡y no con ., que está reservado para multiplicar matrices!) o un espacio en blanco, y la división con **/**. Veamos unos ejemplos.

```
2 + 3
```

```
5
```

```
2 - 3
```

```
-1
```

```
2 * 3
```

```
6
```

```
2 3
```

```
6
```

```
2 / 3
```

$$\frac{2}{3}$$

Como es costumbre, en caso de combinar estas funciones conviene tener en cuenta las procedencias de las operaciones y utilizar los paréntesis cuando haga falta.

■ Print

La función **Print[]** es la que nos permite sacar resultados con formato. Esta función no presenta más que la salida de datos solicitados. Veamos algunos ejemplos.

```
Print["Esto es una prueba."];

x = 3;
Print["El doble de ", x, " es ", 2 x,
      ", y su coseno es ", Cos[x], "."]
```

```
Esto es una prueba.
El doble de 3 es 6, y su coseno es Cos[3].
```

■ Definición normal y definición diferida

Mathematica tiene dos formas de asignar y/o definir variables y funciones, que llamaremos normal y otra denominada diferida. En la normal, por ejemplo,

```
x = 3 * 4 + 2 - 4 * y;
```

Mathematica calcula la expresión del miembro de la derecha y la asigna a la variable *x*. En la diferida, por ejemplo,

```
x := 3 * 4 + 2 - 4 * y;
```

el cálculo del miembro de la derecha no se hace hasta que no necesitemos hacer operaciones con *x*. Veamos un ejemplo donde se muestra la diferencia entre las dos definiciones. Asignemos el valor 3 a la variable *b* y calculemos qué vale *xNormal* y *xDiferida*.

```
b = 4;
xNormal = 1 + b;
xDiferida := 1 + b;

Print["xNormal = ", xNormal];
Print["xDiferida = ", xDiferida];
```

```
xNormal = 5
xDiferida = 5
```

Ahora cambiemos el valor de *b* y volvamos a evaluar *xNormal* y *xDiferida*.

```
b = 6;
Print["xNormal = ", xNormal];
Print["xDiferida = ", xDiferida];
```

```
xNormal = 5
xDiferida = 7
```

Como se puede apreciar, los resultados son diferentes debido a que la definición diferida se recalcula cada vez que se llama a la variable y de esta manera incluye cualquier cambio en las variables que componen la definición, en este caso, de *xDiferida*.

■ La función N

Anteriormente, indicamos que *Mathematica* es un paquete de cálculo simbólico, y por tanto trabaja de forma simbólica. Eso quiere decir que no trabaja con decimales si no es necesario (cálculos muy grandes) o si nosotros no lo forzamos. Una manera de forzarlo es utilizando la función **N[]**. La función **N[]** da una aproximación decimal de un valor exacto. Por ejemplo, para obtener una aproximación del número π, debemos escribir

```
N[π]
3.14159
```

Por defecto, *Mathematica* suele mostrar 5 ó 6 cifras decimales. Para mostrar más, tan sólo hemos de decírselo a la función **N[]**. Para obtener 30 decimales de π:

```
N[π, 30]
3.14159265358979323846264338328
```

■ Notación normal y postfija

En *Mathematica* hay dos maneras de escribir una instrucción para ejecutarla después. De forma normal, por ejemplo

```
N[√2]
1.41421
```

o de forma postfija, por ejemplo

```
√2 // N
1.41421
```

Ésto tan sólo es válido cuando apliquemos la función a un único argumento.

■ Funciones Clear y Remove

Mathematica guarda durante la sesión todos aquellos que se ha ido efectuando. Por tanto, es fácil utilizar como argumento de una función una variable que hemos definido previamente y eso conlleva a cálculos erróneos. Para poder corregir estos errores, disponemos de dos funciones, **Clear[]** y **Remove[]**.

Clear[] limpia los valores y las definiciones de los símbolos que aparecen dentro de **[]**.

Remove[] elimina las variables y funciones de los símbolos que aparecen dentro de **[]** hasta el punto de que *Mathematica* ya no los reconoce más.

Las dos funciones admiten caracteres comodín. Veamos algunos ejemplos:

```
a = 34
Clear[a];
?a
34
Global`a
```

Para **Clear[]**, *Mathematica* entiende que tiene una variable en el sistema que se llama *a*, pero que no contiene ningún valor, aunque al principio lo hemos definido como 34.

```
a = 34
Remove[a];
?a
34
Information::notfound : Symbol a not found.
```

Para **Remove[]**, *Mathematica* no sabe a qué nos referimos cuando le preguntamos sobre la variable *a*.

En el siguiente ejemplo, *Mathematica* limpiará (**Clear[]**) todas las variables de la sesión que comienzan por la letra a.

```
Clear["a*"];
```

y en el siguiente, *Mathematica* limpiará todas las variables de la sesión que contengan la letra b,

```
Clear["*b*"];
```

y finalmente, en el siguiente, *Mathematica* limpiará todas las variables.

```
Clear["*"];
```

■ Matemáticas en *Mathematica*

En las últimas versiones de *Mathematica* se ha cuidado que las instrucciones que se escriban sean lo más parecido a las expresiones matemáticas reales, y de esta manera, se consigue que enseñar *Mathematica* sea cada vez más fácil.

Para ayudarnos en esta empresa, se ha incluido en *Mathematica* las paletas. Por ejemplo, con la paleta *Basic Input* que hemos mostrado anteriormente podemos escribir la mayoría de los símbolos que necesitamos. Pero hay otros con símbolos más específicos, o con diversos tipos de letras, o con funciones de simplificación de polinomios, etc.

Para usar un símbolo o función de una paleta, siempre hemos de hacer lo mismo, pinchar en la paleta y rellenar el cuadrado que nos aparece. Por ejemplo, para escribir (y después calcular) $\int \frac{x^2}{2}\, dx$ tendremos que pinchar en el icono

y aparecerá en el *notebook*

$$\int \square \, d\square$$

Rellenando los cuadrados

$$\int x \, dx$$

seleccionamos la primera x, pinchamos sobre el icono

y aparecerá

$$\int x^{\square} \, dx$$

Rellenamos con un 2 el cuadrado del exponente de x, seleccionamos x^2 y pinchamos sobre el icono

y aparecerá

$$\int \frac{x^2}{\square} \, dx$$

Rellenando el cuadro del denominador con un 2, tendremos:

$$\int \frac{x^2}{2} \, dx$$

$$\frac{x^3}{6}$$

Aritmética básica

■ Clases de números

Mathematica nos permite trabajar con toda clase de números conocidos, enteros,

$$2^{400}$$

$$258224987808690858965591917200301187432970579282922351283065935654064762\ldots$$
$$20168411946296453532801378314359031719727474933 76$$

números racionales,

```
1/2 (2/3 - 3/5) 5
```

$$\frac{1}{6}$$

irracionales (que se trata simbólicamente, si no decimos lo contario),

```
Sqrt[2]
```

$$\sqrt{2}$$

y complejos

```
((2 - 3 I) (1 - I))/(-1 + 2 I)
```

$$-\frac{9}{5}+\frac{7\,i}{5}$$

donde i es la unidad imaginaria.

■ Diferentes precisiones en el cálculo

Cuando hemos hablado de la función **N[]** hemos dicho que *Mathematica* trabaja simbólicamente (o en precisión infinita, como también se dice) si no es que tenga que hacer cálculos muy grandes o lo forcemos a trabajar no simbólicamente, sino numéricamente. Una manera de forzarlo es usar la función **N[]**. Ahora veremos otras formas.

En este primer ejemplo, calcularemos de manera exacta (simbólica) una suma de fracciones

```
3/4 + 2/3 + 1/2
```

$$\frac{23}{12}$$

Y el resultado es exacto y simplificado. El siguiente ejemplo es igual a la anterior, pero hemos cambiado $\frac{1}{2}$ por 0.5, que es una forma de forzarlo para que nos devuelva un resultado numérico

```
3/4 + 2/3 + 0.5
```

```
1.91667
```

Recordemos que con la función **N[]** podemos obtener valores aproximados de números y resultados de operaciones con los decimales que queramos, por ejemplo

```
N[Sqrt[2]]
N[2 Sqrt[2], 50]
```

```
1.41421
```

```
2.8284271247461900976033774484193961571393437507539
```

Una instrucción de interés es la función **Chop[]** que reemplaza valores reales cerca del cero por el cero con una tolerancia de 10^{-10}. Esta instrucción és muy útil cuando se trabaja con números, porque siempre hay valores pequeños que si se eliminan facilitan los cálculos.

Por ejemplo, el siguiente cálculo da una parte imagiaria muy pequeña

```
valor = Exp[N[2 π i]]
```

$1. - 2.44921 \times 10^{-16}\, i$

Podemos hacer desaparecer la parte imaginaria con **Chop[]**

```
Chop[valor]
```

```
1.
```

■ Definición de funciones

Las funciones en *Mathematica* tienen una estructura como la siguiente

```
NombredelaFunción[argumento1_, argumento2_, ..., argumentoN_] :=
 Operaciones con los argumentos.
```

La _ denota que la palabra a la cual acompaña es un argumento. Por ejemplo, definiremos la función $h(x) = x^2 + 1$,

```
h[x_] := x^2 + 1
```

De esta manera podremos evaluar la función *h* en el punto que queramos,

```
h[1]
h[y]
h[Sin[x^3 - 8]]
```

```
2
```

$1 + y^2$

$1 + \text{Sin}[8 - x^3]^2$

Aunque se puedan definir funciones de manera normal y de manera diferida, nosotros siempre lo haremos de manera diferida porque es la manera más natural.

Las funciones se pueden unir y combinar en la misma instrucción, por ejemplo

```
e^Sin[h[5]] // N
```

```
2.14375
```

■ Algunas funciones

Vamos a realizar un pequeño listado de funciones conocidas y su sintáxis con *Mathematica.* Por ejemplo:

■ polinomios

```
p[x_] := x^3 + 2 x^2 - 3 x + 7
```

■ funciones racionales (cociente de polinomios)

$$\mathbf{q[x\_] := \frac{x^3 + 2\,x^2 - 3\,x + 7}{x^4 - 5\,x - 1}}$$

■ funciones irracionales

```
i[x_] := (x^3 + 2 x^2 - 3 x + 7)/(x^4 - 5 x - 1)^(1/3)
```

■ funciones trigonométricas: muchas de éstas ya vienen definidas en *Mathematica*, como el seno, coseno y la tangente

```
Sin[x]
Cos[x]
Tan[x]
```

■ funciones exponenciales: la de base e

```
e^x
```

Y de otras bases, 2, 10 etc.

```
2^x
10^x
```

■ funciones logarítmicas: neperiano o natural

```
Log[x]
```

Y de otras bases, 2, 10, etc., aunque se reducen fácilmente a un logaritmo neperiano, como se puede ver

```
Log[2, x]
```

$$\frac{\text{Log}[x]}{\text{Log}[2]}$$

```
Log[10, x]
```

$$\frac{\text{Log}[x]}{\text{Log}[10]}$$

■ Listas en *Mathematica*

Una lista es una relación de expresiones separadas por comas y entre llaves. Por ejemplo,

```
lista = {2, x, π, 8 + 2 I};
```

Las listas son muy importantes en *Mathematica*. Internamente, casi todas las funciones trabajan con listas que después se transforman para el usuario en expressiones que podemos leer. Además, conviene tener en cuenta que los vectores son listas y las matrices, una lista de listas.

En *Mathematica* hay muchas funciones que tienen el atributo de ser listables, es decir, que se pueden aplicar a una lista de manera que se obtiene una lista donde cada elemento es el valor de la función aplicada a cada elemento de la lista original. Por ejemplo, si queremos aplicar la función seno a la lista llamada *lista* definida anteriormente.

```
Sin[lista]
```

```
{Sin[2], Sin[x], 0, Sin[8 + 2 i]}
```

La función **Length[]** aplicada a una lista nos devuelve el número de elementos que tiene la lista, por ejemplo

```
Length[lista]
```
```
4
```

También se pueden hacer operaciones básicas con dos listas. Definimos dos listas

```
lista1 = {a, b, c, d, e, f};
lista2 = {b, c, g, h};
```

La función **Complement[]** devuelve los elementos de la *lista1* que no están en la *lista2*.

```
Complement[lista1, lista2]
```
```
{a, d, e, f}
```

Ahora calculamos la intersección de las dos listas, es decir, aquellos elementos que están en las dos listas al mismo tiempo.

```
Intersection[lista1, lista2]
```
```
{b, c}
```

o también

```
lista1 ⋂ lista2
```
```
{6, c}
```

Ahora calcularemos la unión de las dos listas, los elementos que están en una o en otra lista, sin repeticiones.

```
Union[lista1, lista2]
```
```
{a, b, c, d, e, f, g, h}
```

o también

```
lista1 ⋃ lista2
```
```
{a, b, c, d, e, f, g, h}
```

La función **Join[]** junta las dos listas. Es como la función **Union[]** (⋃), pero aparecen las repeticiones.

```
Join[lista1, lista2]
```
```
{a, 6, c, d, e, f, 6, c, g, h}
```

Álgebra lineal

■ Construcción de vectores y matrices

Como hemos dicho antes, en *Mathematica* un vector es una lista y una matriz es una lista dónde cada elemento es una lista. Veremos dos ejemplos. En el primero definimos la variable *v* com una lista de 3 elementos

```
v = {1, x - 1, x^2 + 1};
```

En el segundo, tenemos una lista de 3 elementos que cada uno es una lista de 3 elementos. Por tanto, es una matriz 3×3

```
m = {{1, 2, 0}, {0, -1, 1}, {3, 1, 1}};
```

Estamos acostumbrados a ver las matrices arregladas como cajas de números, no como listas de listas. Por esta razón *Mathematica* incorpora la función **MatrixForm[]** que permite ver la matriz en forma de caja. Conviene decir que esta función no devuelve nada, sólo es una función de formato, por tanto es inútil hacer una asignación del tipo **m=MatrixForm[mat]** porque *m* no guardará nada.

```
MatrixForm[m]
```

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & -1 & 1 \\ 3 & 1 & 1 \end{pmatrix}$$

La función **Dimensions[]** devuelve una lista con la dimensión del vector o matriz que haya en el argumento. Dos ejemplos.

```
Dimensions[v]
```

```
{3}
```

v es un vector de 3 componentes.

```
Dimensions[m]
```

```
{3, 3}
```

m es una matriz 3×3. Obsérvese que **Dimensions[]** es diferente de **Length[]**

```
Length[v]
Length[m]
```

```
3
```

```
3
```

Para **Length[]**, *m* también tiene 3 elementos (una lista con 3 elementos). La siguiente instrucción nos devolverá la primera fila de la matriz *m*

```
m[[1]]
{1, 2, 0}
```

Con la siguiente instrucción obtendremos el elemento {1,1} de la matriz *m*

```
m[[1, 1]]
1
```

La función **Table[]** está diseñada para generar de manera fácil vectores y matrices en las cuales los elementos siguen un patrón. Veamos un ejemplo de vectores en que obtendremos una lista donde los elementos son de la forma 2^k para $k = 0, 1, 2, 3, 4, 5$

```
Table[2^k, {k, 0, 5}]
{1, 2, 4, 8, 16, 32}
```

En el siguiente ejemplo obtendremos una matriz en la que sus elementos son de la forma $x^n y^m$ para $n = 0, 1, 2, 3$ y para $m = 0, 1, 2$

```
Table[x^n y^m, {n, 0, 3}, {m, 0, 2}]
{{1, y, y^2}, {x, x y, x y^2}, {x^2, x^2 y, x^2 y^2}, {x^3, x^3 y, x^3 y^2}}
```

Podremos generar matrices diagonales con la función **DiagonalMatrix[];** por ejemplo,

```
DiagonalMatrix[{π, π/2, π/4}]
{{π, 0, 0}, {0, π/2, 0}, {0, 0, π/4}}
```

y matrices identidades con la función **IdentityMatrix[];** por ejemplo la de orden 2,

```
IdentityMatrix[2]
{{1, 0}, {0, 1}}
```

■ Combinaciones lineales de vectores y productos de matrices

Para multiplicar un escalar por un vector usaremos el símbolo *, mientras que para multiplicar matrices haremos uso del . (punto). Veamos primero un ejemplo de combinación lineal. Multiplicamos el escalar *c* por el vector {*x*, *y*} y lo sumamos al producto del escalar *d* por el vector {*s*, *t*}

```
c * {x, y} + d * {s, t}
{d s + c x, d t + c y}
```

Ahora multiplicamos dos vectores (producto escalar)

```
{x, y}.{s, t}
s x + t y
```

A continuación, haremos el producto de una matriz por un vector. Primero definiremos la matriz y el vector, y después los multiplicaremos

```
Clear[a, b, c];
M = {{1, 2, 0}, {2, 0, -1}};
V = {a, b, c};
M.V
```

$$\{a + 2\,b,\ 2\,a - c\}$$

Con *Mathematica* podemos hacer muchas otras operaciones con matrices, como calcular la traspuesta,

```
Transpose[{{x, y, z}, {a, b, c}}] // MatrixForm
```

$$\begin{pmatrix} x & a \\ y & b \\ z & c \end{pmatrix}$$

calcular el determinante de una matriz cuadrada,

```
n = {{1, 3, 4}, {-1, 0, 1}, {1, 3, -1}};
Det[n]
```

$$-15$$

calcular su inversa

```
m = Inverse[n];
MatrixForm[m]
```

$$\begin{pmatrix} \frac{1}{5} & -1 & -\frac{1}{5} \\ 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{5} & 0 & -\frac{1}{5} \end{pmatrix}$$

y comprobar que, efectivamente, es la inversa multiplicándolas

```
n.m // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

■ Sistemas de ecuaciones

Aparte de calcular la inversa de una matriz para resolver un sistema de ecuaciones lineales, hay otras formas y *Mathematica* nos las ofrece, como es la función **`LinearSolve[]`**. Si queremos resolver el sistema $m\,x = b$ donde

```
m = {{1, 2, -1}, {2, 3, 1}, {0, -2, 1}};
b = {1, 1, -2};
```

la siguiente función nos da la solución usando la eliminación gaussiana

```
LinearSolve[m, b]
```

$$\{-1,\ 1,\ 0\}$$

Comprobamos que por el método del cálculo de la inversa obtenemos el mismo resultado

```
Inverse[m].b
```

```
{-1, 1, 0}
```

■ Valores y vectores propios

Los valores propios de una matriz cuadrada son las raíces del polinomio característico de esa matriz. El polinomio característico de una matriz A se calcula como $\det(A-\lambda I)$, donde λ es un parámetro e I es la matriz identidad del mismo tamaño que A. Con *Mathematica*

```
A = {{3, 1, 0}, {2, 2, 0}, {1, 0, 1}};
polCar = Det[A - λ * IdentityMatrix[3]]
```

$$4 - 9\,\lambda + 6\,\lambda^2 - \lambda^3$$

Como las raíces del polinomio característico son los valores propios, las calculamos mediante la instrucción **Solve[]** (esta instrucción se explicará con detalle más adelante)

```
Solve[polCar == 0, λ]
```

```
{{λ → 1}, {λ → 1}, {λ → 4}}
```

La función **Eigenvalues[]** calcula los valores propios de una matriz cuadrada. Por ejemplo:

```
n = {{3, 1, 0}, {2, 2, 0}, {1, 0, 1}};
Eigenvalues[n]
```

```
{1, 1, 4}
```

La función **Eigenvectors[]** calcula los vectores propios asociados a los valores propios de la matriz. Por ejemplo:

```
Eigenvectors[n]
```

```
{{0, 0, 1}, {0, 0, 0}, {3, 3, 1}}
```

Hay ocasiones en las que queremos calcular al mismo tiempo los valores y los vectores propios, conocer qué vectores van asociados a qué valores propios, cuál es la matriz diagonal y cuál la matriz de paso. Es decir, diagonalizar la matriz.

Todo esto podemos hacerlo con la función **Eigensystem[],** que devuelve una lista *{valores,vectores}* de valores propios y vectores propios asociados y ordenados. Por ejemplo,

```
Eigensystem[n]
```

```
{{1, 1, 4}, {{0, 0, 1}, {0, 0, 0}, {3, 3, 1}}}
```

Que quiere decir que los valores propios son 1 (doble) y 4. Asociado al 1 tenemos el vector propio $(0, 0, 1)$ (el (0,0,0) no es libre y por tanto no cuenta). Y asociado al 4 está el vector $(3, 3, 1)$.

Como hemos dicho antes, con **Eigensystem[]** podemos diagonalizar una matriz. Veamos un ejemplo

```
m4 = N[{{1, 2, 3, 2}, {5, 3, 1, 8}, {6, 2, 8, 9}, {4, 2, 1, 7}}];

{valorProp, vectorProp} = Eigensystem[m4];
```

Los valores propios dan la forma diagonal

```
matDiag = Chop[DiagonalMatrix[valorProp]];
MatrixForm[matDiag]
```

$$\begin{pmatrix} 14.9901 & 0 & 0 & 0 \\ 0 & 4.72699 & 0 & 0 \\ 0 & 0 & -1.27955 & 0 \\ 0 & 0 & 0 & 0.562502 \end{pmatrix}$$

Los vectores propios por columnas son la matriz de paso

```
matP = Transpose[vectorProp];
MatrixForm[matP]
```

$$\begin{pmatrix} -0.278736 & -0.217944 & -0.847457 & -0.713683 \\ -0.411127 & 0.454619 & 0.407251 & -0.406348 \\ -0.797578 & -0.796445 & 0.179532 & -0.00537884 \\ -0.342272 & 0.333911 & 0.289363 & 0.570534 \end{pmatrix}$$

Comprobamos la diagonalización

```
Chop[Inverse[matP].m4.matP] // MatrixForm
```

$$\begin{pmatrix} 14.9901 & 0 & 0 & 0 \\ 0 & 4.72699 & 0 & 0 \\ 0 & 0 & -1.27955 & 0 \\ 0 & 0 & 0 & 0.562502 \end{pmatrix}$$

■ Potencias y exponencial de una matriz

Para calcular potencias de una matriz, se utiliza **MatrixPower[].** A esta instrucción hay que ponerle dos parámetros, la matriz y la potencia a la que queremos elevarla. Veamos un ejemplo

```
A = {{1, 2, 3, 2}, {5, 3, 1, 8}, {6, 2, 8, 9}, {4, 2, 1, 7}};

Print["A^3 = ", MatrixPower[A, 3] // MatrixForm]
```

$$A^3 = \begin{pmatrix} 549 & 308 & 436 & 910 \\ 843 & 493 & 582 & 1411 \\ 1570 & 868 & 1259 & 2594 \\ 700 & 408 & 488 & 1171 \end{pmatrix}$$

Para calcular la exponencial de una matriz, la función que se utiliza es **MatrixExp[].** Veamos un ejemplo

```
B = {{1, 0, 0}, {0, 2, 1}, {2, 0, 0}};
Print["e^B = ", MatrixExp[B] // MatrixForm]
```

$$e^B = \begin{pmatrix} e & 0 & 0 \\ 1 - 2e + e^2 & e^2 & -\frac{1}{2} + \frac{e^2}{2} \\ -2 + 2e & 0 & 1 \end{pmatrix}$$

Resolución de ecuaciones

Mathematica tiene las funciones **Solve[]** y **NSolve[]** para resolver ecuaciones. **Solve[]** sirve para obtener soluciones exactas y **NSolve[]** para obtener soluciones numéricas. La sintaxis es

Solve[eqns, vars] resuelve la ecuación *eqns* para las variables *vars*.

Para **NSolve[]** es la misma sintaxis. Veamos algunos ejemplos. Obsérvese que la igualdad de la ecuación se hace con **==** (dos iguales)

```
Solve[x^3 - 3 x^2 - x == 0, x]
```

$$\left\{\{x \to 0\}, \left\{x \to \frac{1}{2}\left(3 - \sqrt{13}\right)\right\}, \left\{x \to \frac{1}{2}\left(3 + \sqrt{13}\right)\right\}\right\}$$

```
NSolve[x^3 - 3 x^2 - x == 0, x]
```

```
{{x → -0.302776}, {x → 0.}, {x → 3.30278}}
```

```
Solve[Sin[x] Cos[x] == 0, x]
```

```
Solve::ifun : Inverse functions are being
   used by Solve, so some solutions may not be found.
```

$$\left\{\{x \to 0\}, \left\{x \to -\frac{\pi}{2}\right\}, \left\{x \to \frac{\pi}{2}\right\}\right\}$$

En este último ejemplo, *Mathematica* nos avisa que no puede calcular inversas de las funciones que aparecen en la ecuación y que puede ser, por eso, que no aparezcan todas las soluciones, o que las obtenidas no sean correctas.

También podemos usar estas funciones para resolver sistemas de ecuaciones. Obsérvese que el sistema de ecuaciones está entre llaves

```
Solve[{x + Sqrt[y] == 1, (x + y)^2 == y}, {x, y}]
```

```
{{y → -1, x → 1 - i}, {y → 1, x → 0}}
```

```
NSolve[{x^2 + y^2 == 1, x^2 == y}, {x, y}]
```

```
{{y → -1.61803, x → 1.27202 i}, {y → -1.61803, x → -1.27202 i},
 {y → 0.618034, x → -0.786151}, {y → 0.618034, x → 0.786151}}
```

```
Solve[{(x + y)^3 == 1, (x + y)^2 == 1}, {x, y}]
```

```
Solve::svars :
 Equations may not give solutions for all "solve" variables.
```

```
{{x → 1 - y}}
```

En este último ejemplo, una ecuación es función de la otra. Por eso *Mathematica* nos contesta que la solución viene dada en función de la otra variable.

Límites, derivadas e integración

■ Límites de funciones

Con *Mathematica* podemos calcular límites de funciones. Veamos unos ejemplos.

```
Limit[(x^2 + x - 2)/(x - 1), x → 1]
```

```
3
```

```
Limit[Tan[x]/x, x → 0]
```

```
1
```

```
Limit[e^x, x → ∞]
```

```
∞
```

La flecha → se consigue con **->**. *Mathematica* hace después la traducción.

■ Derivadas

Mathematica tiene dos formas (sintaxis) de calcular derivadas. Para verlo, definimos la función *f* como el seno de *x*,

```
f[x_] := Sin[x]
```

La primera manera de calcular la derivada es con **f'[]**

```
f'[x]
```

```
Cos[x]
```

y la segunda manera es con la función **D[]** incluyendo la variable respecto a la cual derivamos,

```
D[f[x], x]
```

```
Cos[x]
```

Asimismo, de forma directa podemos calcular la derivada de una función en un punto. En el siguiente ejemplo, calcularemos (sin tener que hacer el cálculo de la derivada previamente), $f'(\frac{\pi}{2})$.

```
f'[π/2]
```

```
0
```

■ Integrales

Con *Mathematica* podemos calcular primitivas de funciones

```
∫1/((x-1)^2+4) dx
```

```
1/2 ArcTan[1/2 (-1 + x)]
```

integrales definidas

```
∫_0^π Sin[x] dx
```

```
2
```

e integrales impropias

```
∫_0^+∞ e^-x^2 dx
```

```
√π/2
```

Ecuaciones diferenciales

Para resolver ecuaciones diferenciales *Mathematica* tiene la función **`DSolve[]`**. La sintaxis es

`DSolve[eqn, y, x]` reuelve la ecuación diferencial para la función *y* con la variable independiente *x*.

Veamos dos ejemplos:

```
DSolve[y'[x]-y[x]==0, y[x], x]
```

```
{{y[x] → e^x C[1]}}
```

```
DSolve[x''[t]+x[t]==0, x[t], t]
```

```
{{x[t] → C[2] Cos[t] - C[1] Sin[t]}}
```

En los ejemplos anteriores, la solución depende de una constante debido a que no hemos incluido ninguna condición inicial. En los siguientes ejemplos la incluiremos. Obsérvese que las condiciones iniciales son una parte de la ecuación y que está entre llaves.

```
DSolve[{y'[x]-y[x]==0,
        y[0]==1}, y[x], x ]
```

$$\{\{y[x] \to e^{x}\}\}$$

La ecuación anterior tiene una condición inicial porque es de orden 1. La que veeremos ahora tiene dos condiciones iniciales porque es de orden 2.

```
DSolve[{x''[t]-x'[t]-2x[t]==0,
       x[0]==0,
       x'[0]==1}, x[t], t]
```

$$\left\{\left\{x[t] \to \frac{1}{3} e^{-t} \left(-1 + e^{3 t}\right)\right\}\right\}$$

También se pueden resolver sistemas de ecuaciones diferenciales. Veamos un ejemplo

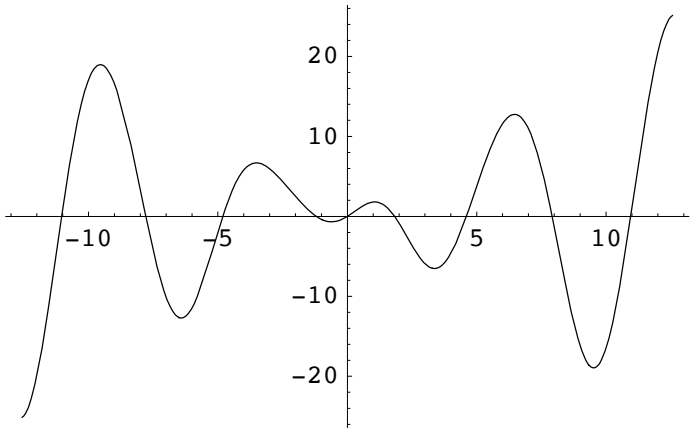
```
DSolve[ {x'[t] == y[t],
         y'[t] == x[t]}, {x[t], y[t]}, t ]
```

$$\left\{\left\{x[t] \to \frac{1}{2} e^{-t} \left(C[1] + e^{2 t} C[1] - C[2] + e^{2 t} C[2]\right),\right.\right.$$
$$\left.\left. y[t] \to \frac{1}{2} e^{-t} \left(-C[1] + e^{2 t} C[1] + C[2] + e^{2 t} C[2]\right)\right\}\right\}$$

Gráficas en 2D

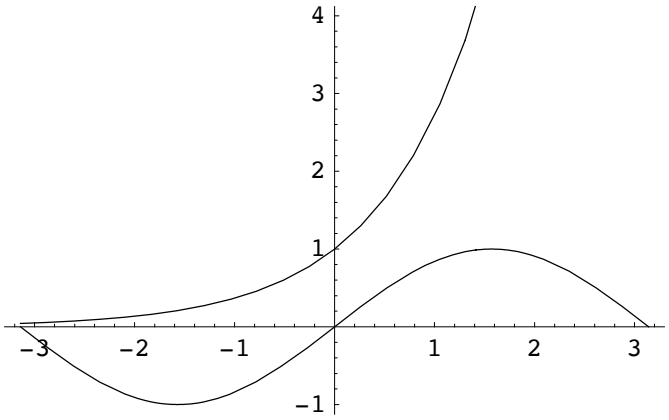
La función que nos permite con *Mathematica* dibujar gráficas en dos dimensiones es **Plot[]**. Esta función tiene muchas opciones, alguna de las cuales estudiaremos. Comenzamos con un ejemplo sencillo. Dibujaremos la función $\text{Sin}[x]^2 + 2x\,\text{Cos}[x]$ en el intervalo $[-4\pi, 4\pi]$

```
Plot[Sin[x]^2 + 2 x Cos[x], {x, -4 π, 4 π}];
```



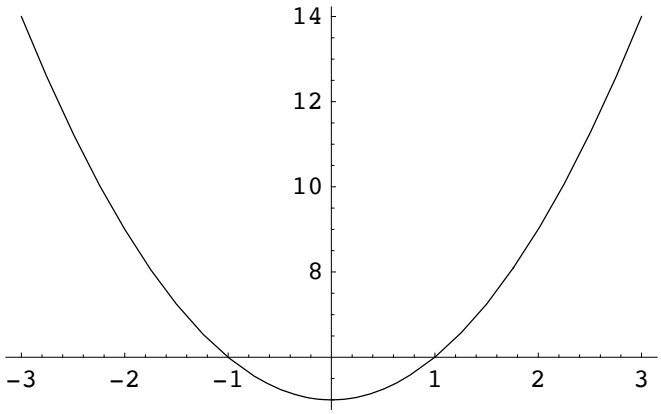
Para dibujar dos funciones (o más) juntas, las ponemos entre llaves y separadas por comas, como en el siguiente ejemplo:

```
Plot[{Sin[x], e^x}, {x, -π, π}];
```



Mathematica dibuja la función en el sitio que cree que es más apropiado, pero eso no quiere decir que sea el más relevante para nosotros. Hemos de tener mucha precaución con esto y para mostrarlo, veamos un ejemplo. Sabemos que la función x^2+5 no tiene raíces reales, pero si nos fijamos en la gráfica que nos da *Mathematica*

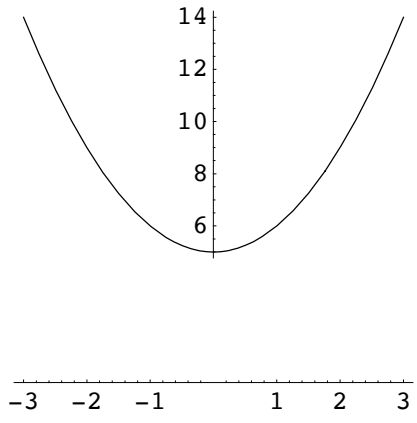
```
Plot[x^2 + 5, {x, -3, 3}];
```



podríamos pensar (erróneamente) que sí. Si nos fijamos, el origen de coordenadas no está al $(0, 0)$. *Mathematica* ha estimado (y la verdad sea dicha, tiene razón como a continuación veremos) que la función no tiene nada relevante en el $(0, 0)$ y la ha dibujado en otro lugar.

Ahora, para fijar el origen de coordenadas en un punto, en nuestro caso el $(0, 0)$, usaremos la opción **AxesOrigin->{0,0}**. Veamos

```
Plot[x^2 + 5, {x, -3, 3}, AxesOrigin → {0, 0}];
```



Ahora sí que se ve que no hay raíces reales. Y también se ve que la información de la función cerca del (0, 0) no es relevante, porque no hay nada.

■ Un ejemplo completo

Veamos ahora un ejemplo donde incluiremos diversas opciones para hacer una gráfica casi "profesional". Primero definimos las funciones a dibujar

```
f1[x_] := 2 x;
f2[x_] := Sin[x];
f3[x_] := Sin[x] + 2;
f4[x_] := Sin[x] + 1;
```

Ahora definiremos una variable a la cual llamaremos *ls* y le asignaremos el límite superior del intervalo donde queremos dibujar las gráficas. Añadiremos las opciones

AxesOrigin→{0,0}, que pondrán el origen de coordenadas en el (0, 0),

AxesLabel→{"Bien x ","Bien y"}, que etiquetará los ejes X con el título *Bien x* y al eje Y con el título *Bien y*,

PlotRange→{0,4}, que hará que sólo se dibujen los valores del eje Y que estén en el intervalo [0, 4],

PlotStyle→{...}, que nos dejará modificar el dibujo estándar de las funciones como ahora explicamos:

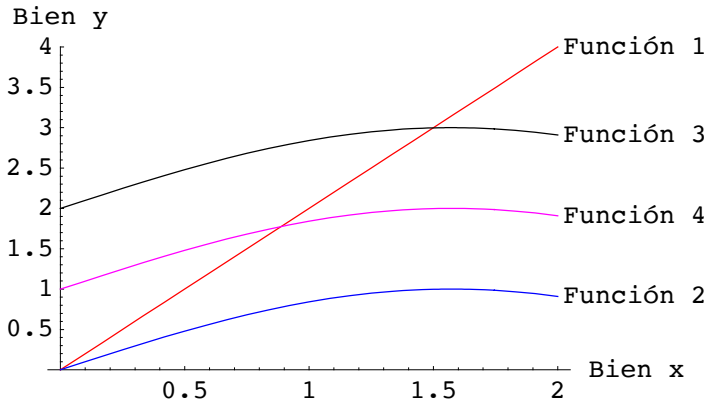
Text["Función 1",{ls,f1[ls]},{-1,0}] hará que en el punto {ls, f1[ls]} (en este caso, el extremo derecho de la función) se escriba el título "Función 1". Los parámetros {-1,0} indican que se escriba comenzando desde el punto hacia la derecha.

RGBColor[1,0,0] le da color a la gráfica como combinación de los colores rojo (R), verde (G) y azul (B). Los valores se toman desde 0 hasta 1. En este caso, {1,0,0} es el color rojo, el {0,0,1} es el azul y el {1,0,1} el fucsia.

GrayLevel[0] da nivel de gris a la gráfica. Va de 0 a 1, donde el 0 es el negro y el 1 es el blanco.

¡Podeis hacer pruebas cambiando los parámetros!

```
ls = 2; (* límite superior del intervalo donde vamos a dibujar *)
Plot[{f1[x], f2[x], f3[x], f4[x]}, {x, 0, ls},
      AxesOrigin → {0, 0},
      AxesLabel → {"Bien x      ", "Bien y"},
      PlotRange → {0, 4},
      PlotStyle →
    {{Text["Función 1", {ls, f1[ls]}, {-1, 0}],
          RGBColor[1, 0, 0]},
      {Text["Función 2", {ls, f2[ls]}, {-1, 0}],
          RGBColor[0, 0, 1]},
      {Text["Función 3", {ls, f3[ls]}, {-1, 0}], GrayLevel[0]},
      {Text["Función 4", {ls, f4[ls]}, {-1, 0}],
          RGBColor[1, 0, 1]}}];
```



Un poco de programación

Creemos que es importante hacer una pequeña introducción a la programación con *Mathematica*, porque aunque lo que hemos visto pueda parecernos importante y permite aliviar los cálculos tediosos, la potencia real de este software es la programación, que puede ayudarnos a implementar con facilidad procedimientos que *a priori* nos resultan dificultosos.

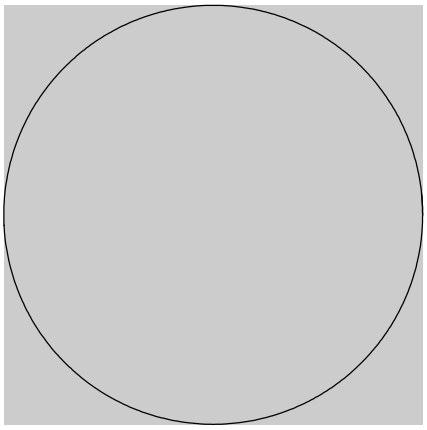
La mejor manera para introducirnos en la programación es hacer un ejemplo. En el ejemplo plantearemos el problema y propondremos la solución. Después, presentaremos las funciones de *Mathematica* que necesitaremos para implementar el procedimiento, presentaremos el código, lo explicaremos y lo ejecutaremos.

■ Presentación del problema

Queremos hacer una estimación del valor del número π para un método probabilístico llamado método MonteCarlo.

Suponemos que tenemos un cuadrado de madera cuyo lado mide 1 y en el que hemos dibujado un círculo de radio $\frac{1}{2}$ como se ve en la siguiente figura:

```
cercle = Graphics[{GrayLevel[0], Circle[{1/2, 1/2}, 1/2]}];
quadrat = Graphics[{GrayLevel[0.8], Rectangle[{0, 0}, {1, 1}]}];
Show[quadrat, cercle, AspectRatio → Automatic];
```



Ahora procederemos a jugar a los dardos. Suponemos que cada dardo que tiremos cae dentro del cuadrado. Contaremos un punto si cae dentro del círculo y no contaremos ninguno si cae fuera del círculo (y dentro del cuadrado).

Según la teoría, la probabilidad de que lancemos un dardo y caiga dentro del círculo es el área del círculo ($\pi\left(\frac{1}{2}\right)^2 = \frac{\pi}{4}$) dividido por el área del cuadrado $(1 \times 1 = 1)$. Por tanto

Probabilidad que un lanzamiento de dardo caiga dentro del círculo $= \frac{\pi}{4}$.

■ Solución del problema y propuesta de algoritmo

Así, para aproximar el valor de π haremos muchos lanzamientos de dardos, dividiremos los que caigan dentro del círculo por los que hayamos tirado (aproximación a $\frac{\pi}{4}$) y el resultado lo multiplicaremos por 4. Así obtendremos una aproximación de π.

Recordamos que una circunferencia de centro (a, b) y de radio r tiene como ecuación $(x-a)^2 + (y-b)^2 = r^2$, y los puntos (x, y) que están dentro del círculo son los que satisfacen $(x-a)^2 + (y-b)^2 \le r^2$. Para nuestro círculo eso se traduce en $\left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2 \le \frac{1}{4}$.

Para hacer un diseño de algoritmo, éste podría ser:

PASO 1. Simulación de un lanzamiento de dardo.

Guardar en la variable x un valor aleatorio real entre 0 y 1.

Guardar en la variable y un valor aleatorio real entre 0 y 1.

PASO 2. Comprobar si la tirada ha caído dentro del círculo o fuera.

(x, y) ha caído dentro del círculo si $\left(x-\frac{1}{2}\right)^2+\left(y-\frac{1}{2}\right)^2 \le \frac{1}{4}$. En este caso, le sumaremos 1 a la variable contadora DENTRO.

Sumar 1 a la variable contadora TIRADAS.

REPETIREMOS los pasos 1 y 2 tantas veces como tiros queramos hacer.

PASO 3. Una vez hechas todas las tiradas, aproximamos el valor de π por $4 * \frac{\text{DENTRO}}{\text{TIRADAS}}$.

■ Funciones de *Mathematica* necesarias

Para escribir el código necesario necesitaremos las funciones:

- **`Module[{x=x0,y,...},expr]`** hace que las variables inicializadas o no *x*, *y*, ..., que aparezcan en *exp*, se traten como variables locales.
- **`For[comienzo,test,incr,cos]`** ejecuta *comienzo*, repetidamente evalúa *cos* y *incr* hasta que *test* devuelva verdadero.
- **`Random[]`** devuelve un valor real aleatorio entre 0 y 1.
- **`If[condicion,v,f]`** devuelve *v* si *condición* es verdadera y *f* si es falsa.
- **`Return[expr]`** devuelve el valor *expr* dentro de una función.

■ Implementación y explicación del código

```
ValordePi[TIRADAS_] := Module[{DENTRO = 0, x, y, i},
                        For[i = 1, i ≤ TIRADAS, i++,
                              x = Random[];
                              y = Random[];
                              If[ (x - 1/2)^2 + (y - 1/2)^2 ≤ 1/4, DENTRO += 1]
                        ];
                        Return[ DENTRO/TIRADAS * 4]
                        ]
```

En la primera línea hacemos que las variables que definimos sean locales y desaparezcan del sistema cuando acabe el procedimiento. Dentro del bucle **`For[]`** generamos tiradas aleatoriamente y dentro de **`If[]`** comprobamos si caen dentro del círculo y, en caso afirmativo, sumamos 1 a la variable DENTRO. Una vez acabado el **`For[]`**, calculamos la aproximación del valor de π.

■ Ejecución y pruebas

Evaluaremos ahora la función anteriormente implementada para 100.000 tiradas de dardos.

```
TIRADAS = 100000;
ap = ValordePi[TIRADAS] // Timing;
Print["El valor aproximado de π obtenido con ",
  TIRADAS, " tiradas es ", ap[[2]], " ≈ ", N[ap[[2]], 10],
  " en un tiempo de ", ap[[1]] / Second, " segundos."];
Print["El valor exacto de π con cinco cifras decimales exactas es ",
  N[π, 6], "."];
```

```
El valor aproximado de π obtenido con 100000
  tiradas es 78577/25000 ≈ 3.14308 en un tiempo de 10.54 segundos.
El valor exacto de π con cinco cifras decimales exactas es 3.14159.
```

Como vemos, con 100.000 tiradas obtenemos dos cifras decimales exactas. Además hemos incluido la función **Timing[]** que nos devuleve el tiempo que ha usado *Mathematica* para hacer los cálculos. Diferentes ejecuciones del procedimiento, darán diferentes resultados.

Observación final

Recordar que las ejecuciones de *Mathematica* siempre devuelven algún mensaje o resultado, pero no tiene porque ser lo que buscamos o lo que queramos. Para que *Mathematica* sea una herramienta útil, conviene saber qué hacemos en cada paso, analizar los resultados y ¡utilizar el sentido común!

¿Cómo se hace ... ?

Resolución de ecuaciones

■ Resolución de sistemas de ecuaciones lineales

Vamos a resolver el sistema de ecuaciones lineales

$$\begin{cases} 3x - 4y = 13 \\ 2x + 3y = 3 \end{cases}$$

Podemos utilizar la instrucción **Solve[]**, bien definiendo cada una de las ecuaciones

```
Solve[{3 x - 4 y == 13, 2 x + 3 y == 3}, {x, y}]
```

```
{{x → 3, y → -1}}
```

o bien con la forma matricial

```
A = {{3, -4}, {2, 3}};
b = {13, 3};

Solve[A.{x, y} == b, {x, y}]
```

```
{{x → 3, y → -1}}
```

Como hemos obtenido una única solución se trata de un sistema compatible determinado.

Mathematica dispone de una instrucción específica para resolver sistemas de ecuaciones lineales, se trata de **LinearSolve[]**. Para resolver el sistema anterior con esta instrucción escribiremos

```
LinearSolve[A, b]
```

```
{3, -1}
```

Hay que tener en cuenta que con la instrucción **LinearSolve[]** *Mathematica* sólo nos devuelve una solución. Para saber si es la única solución tendremos que comprobar si el sistema es compatible determinado. En este caso, es suficiente que el determinante de la matriz *A* sea distinto de cero

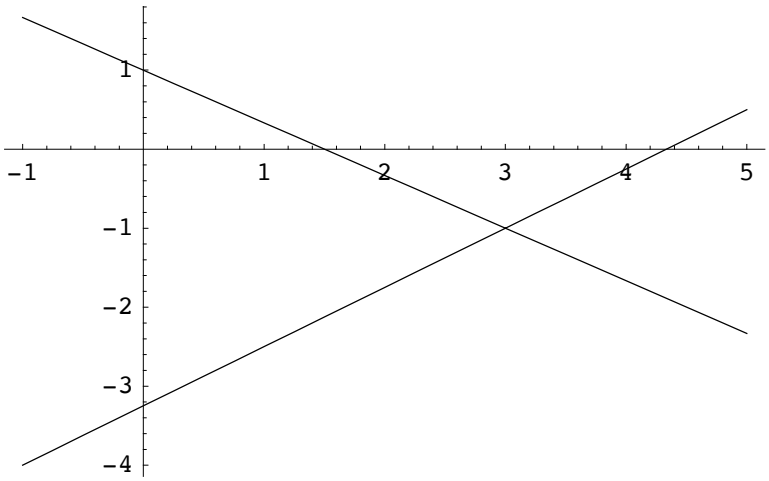
```
Det[A]
```

```
17
```

Como el sistema que estamos resolviendo es de dos ecuaciones con dos incógnitas, podemos obtener su solución gráficamente.

Si despejamos en cada una de las ecuaciones la variable y, obtenemos las ecuaciones de las rectas $y = (3x - 13)/4$, $y = (3 - 2x)/3$. Obtener la solución gráficamente es calcular el punto de corte de estas rectas

```
Plot[{(3 x - 13) / 4, (3 - 2 x) / 3}, {x, -1, 5}]
```



Como el sistema es compatible determinado, obtenemos un único punto de corte, que corresponde a la solución del sistema $x = 3$, $y = -1$.

Vamos a resolver el sistema de ecuaciones lineales

$$\begin{cases} 3x - 4y = 13 \\ -\frac{3}{2}x + 2y = -\frac{13}{2} \end{cases}$$

Utilizando la instrucción **Solve[]**, con ecuaciones y matricialmente, obtenemos, respectivamente

```
Solve[{3 x - 4 y == 13, -3/2 x + 2 y == -13/2}, {x, y}]
```

```
Solve::svars :
 Equations may not give solutions for all "solve" variables.
```

$$\left\{\left\{x \to \frac{13}{3} + \frac{4y}{3}\right\}\right\}$$

```
A = {{3, -4}, {-3/2, 2}};
b = {13, -13/2};
Solve[A.{x, y} == b, {x, y}]
```

```
Solve::svars :
 Equations may not give solutions for all "solve" variables.
```

$$\left\{\left\{x \to \frac{13}{3} + \frac{4y}{3}\right\}\right\}$$

Mathematica nos avisa que se trata de un sistema compatible indeterminado, y su solución es $x = \frac{13}{3} + \frac{4y}{3}$, $y \in \mathbb{R}$.

Si utilizamos la instrucción

```
LinearSolve[A, b]
```

$$\left\{\frac{13}{3}, 0\right\}$$

sólo obtenemos una solución. Eso no quiere decir que el sistema sea compatible determinado, como podemos comprobar si calculamos el determinante

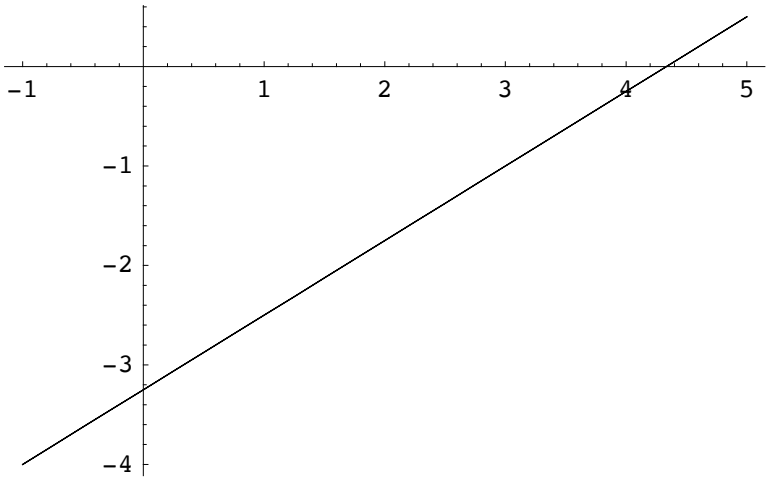
```
Det[A]
```

```
0
```

En este caso, la solución obtenida es la correspondiente a $y = 0$.

Gráficamente

```
Plot[{(3 x - 13) / 4, (-13 + 3 x) / 4}, {x, -1, 5}]
```



se trata de dos rectas coincidentes, por tanto la solución es la recta $y = (3x - 13)/4$.

Cuando el sistema es incompatible, *Mathematica* no muestra ninguna solución

```
Solve[{3 x - 4 y == 13, -3/2 x + 2 y == 10}, {x, y}]
```

```
{}
```

```
A = {{3, -4}, {-3/2, 2}};
b = {13, 10};

Solve[A.{x, y} == b, {x, y}]
```

```
{}
```

o nos avisa que no hay solución

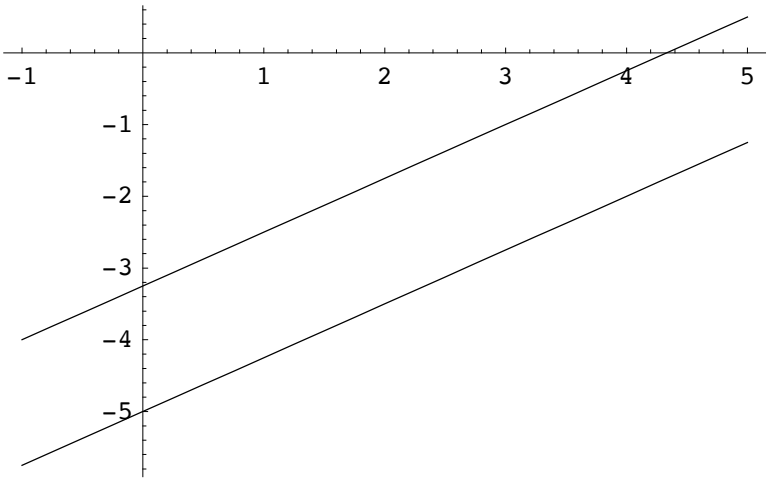
```
LinearSolve[A, b]
```

```
LinearSolve::nosol : Linear equation encountered which has no solution.
```

```
LinearSolve[{{3, -4}, {-3/2, 2}}, {13, 10}]
```

Gráficamente

```
Plot[{(3 x - 13) / 4, (-10 + 3 x / 2) / 2}, {x, -1, 5}]
```



se trata de dos rectas paralelas, por tanto no se cortan en ningún punto.

■ Resolución de ecuaciones no lineales

Para resolver ecuaciones o sistemas de ecuaciones no lineales utilizaremos la instrucción **Solve[]** para obtener la solución de forma exacta y **NSolve[]** para obtenerla de forma aproximada. Por ejemplo

```
Solve[1/Sqrt[x] + 1/x == 3, x]
```

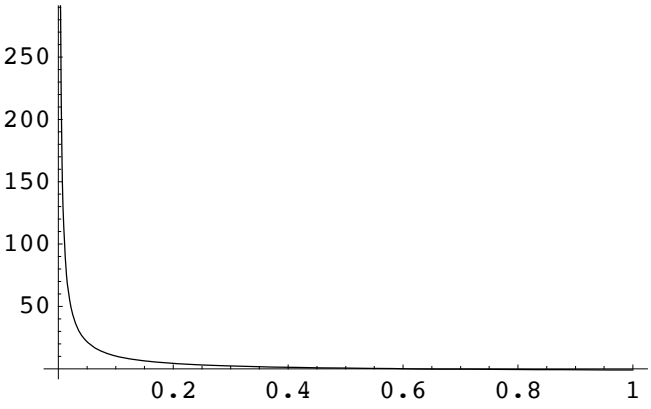
$$\left\{\left\{x \to \frac{1}{18}\left(7+\sqrt{13}\right)\right\}\right\}$$

```
NSolve[1/Sqrt[x] + 1/x == 3, x]
```

$$\{\{x \to 0.589197\}\}$$

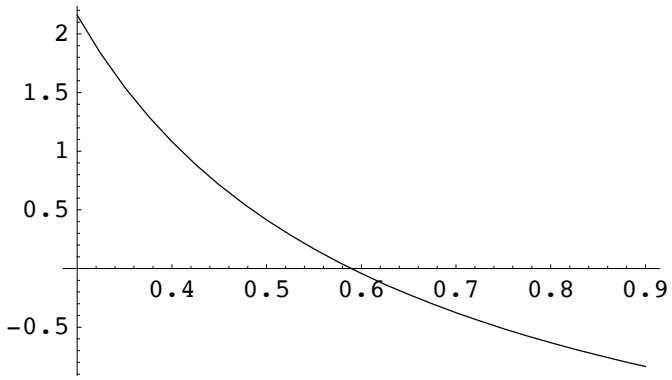
Para resolverlo gráficamente, buscaremos la intersección de la función $\frac{1}{\sqrt{x}}+\frac{1}{x}-3$ con el eje *X*

```
Plot[1/Sqrt[x] + 1/x - 3, {x, 0, 1}];
```



Como no podemos apreciar cuál es el punto de corte con el eje X, pero vemos que se encuentra entre 0.3 y 0.9, volveremos a representar la situación, pero reduciendo el intervalo a [0.3, 0.9]

```
Plot[1/Sqrt[x] + 1/x - 3, {x, 0.3, 0.9}]
```



De donde deducimos que la solución de la ecuación $\frac{1}{\sqrt{x}} + \frac{1}{x} = 3$ es aproximadamente $x = 0.59$.

■ Resolución de ecuaciones en diferencias

Para resolver ecuaciones en diferencias, *Mathematica* tiene el package *RSolve* que carga mediante el mandato.

```
<<DiscreteMath`RSolve`
```

La instrucción de este package que resuelve ecuaciones en diferencias es **`RSolve[]`** y funciona de forma similar a **`DSolve[]`**.

La sintáxis es *RSolve[{eqn1, eqn2, ... }, {a1, a2, ... }, n]* donde las *eqn* son las ecuaciones, las *a* son las funciones que queremos calcular y *n* la variable discreta. Veamos ejemplos.

La solución de la ecuación $a(n+1) = 2\,a(n)$, $n = 1, 2, \ldots$ es

```
RSolve[a[n+1] == 2 a[n], a[n], n]
```

```
{{a[n] → 2^n C[1]}}
```

La solución de la ecuación $a(n+1) = 2\,a(n)$, $a(0) = 5$, $n = 1, 2, \ldots$ es

```
RSolve[{a[n+1] == 2 a[n],
        a[0] == 5}, a[n], n]
```

```
{{a[n] → 5 2^n}}
```

La solución de la ecuación de Fibonacci $a(n) = a(n-1) + a(n-2)$, $a(0) = 1$, $a(1) = 1$, $n = 1, 2, \ldots$ es

```
RSolve[{a[n] == a[n-1] + a[n-2],
        a[0] == 1,
        a[1] == 1}, a[n], n]
```

$$\left\{\left\{a[n] \to \frac{2^{-1-n}\left(-\left(1-\sqrt{5}\right)^{1+n}+\left(1+\sqrt{5}\right)^{1+n}\right)}{\sqrt{5}}\right\}\right\}$$

La solución del sistema de ecuaciones en diferencias

$a(n+1)-3\,b(n)-4\,a(n)=1,$
$a(n+1)+b(n+1)+b(n)=n\,,$
$a(0)=0,$
$b(0)=0.$

es

```
RSolve[{a[n+1] - 3 b[n] - 4 a[n] == 1,
        a[n+1] + b[n+1] + b[n] == n,
        a[0] == 0,
        b[0] == 0}, {a[n], b[n]}, n]
```

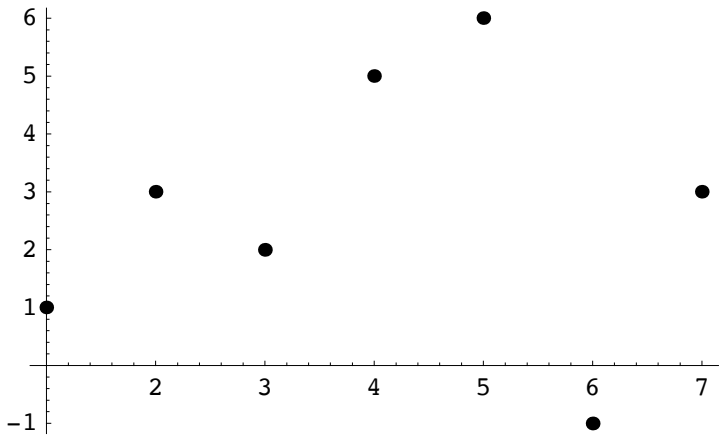
$$\left\{\left\{a[n] \to \frac{1}{6}\,(-8-(-2)^n+9\,2^n-6\,n),\ b[n] \to \frac{1}{3}\,(2+(-2)^n-3\,2^n+3\,n)\right\}\right\}$$

Modelos discretos

■ Dibujar puntos con *Mathematica*: ListPlot y MultipleListPlot

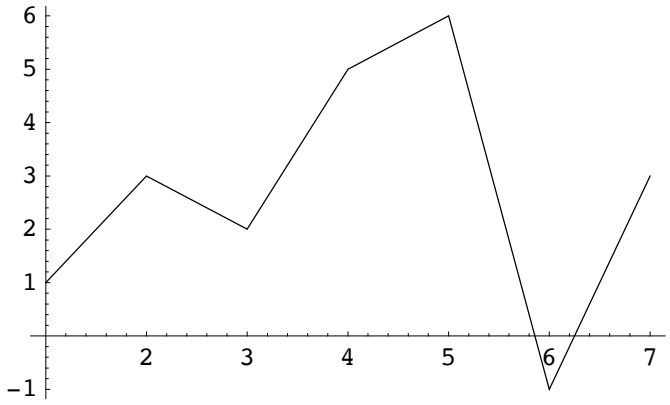
Con la instrucción **ListPlot[]**, *Mathematica* permite dibujar una lista de puntos en un gráfico. Si consideramos la lista siguiente, y la dibujamos

```
lista = {1, 3, 2, 5, 6, -1, 3};
ListPlot[lista, PlotStyle -> PointSize[0.02]];
```



Podemos unir los puntos con una línea usando la opción **PlotJoined->True**.

```
ListPlot[lista, PlotJoined → True];
```

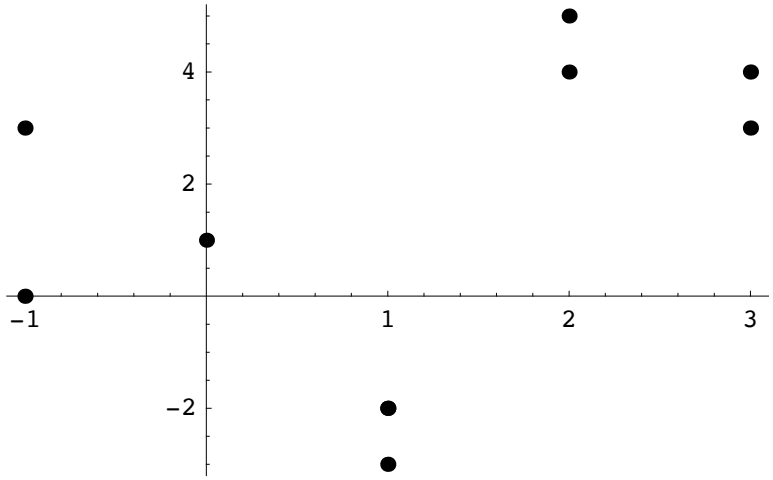


En el ejemplo que hemos visto, *Mathematica* traduce **lista** a la lista siguiente

```
{{1, 1}, {2, 3}, {3, 2}, {4, 5}, {5, 6}, {6, -1}, {7, 3}}
```

es decir, le añadimos un índice a la primera componente, antes de dibujarla. Esto quiere decir que podemos dibujar listas de parejas de elementos. Veámoslo.

```
lista2 = {{1, -3}, {-1, 3}, {3, 4}, {3, 3},
   {1, -2}, {-1, 0}, {2, 4}, {2, 5}, {1, -2}, {0, 1}};
ListPlot[lista2, PlotStyle -> PointSize[0.02]];
```



Si lo que queremos es dibujar dos listas en la misma gráfica, tendremos que hacer uso del package

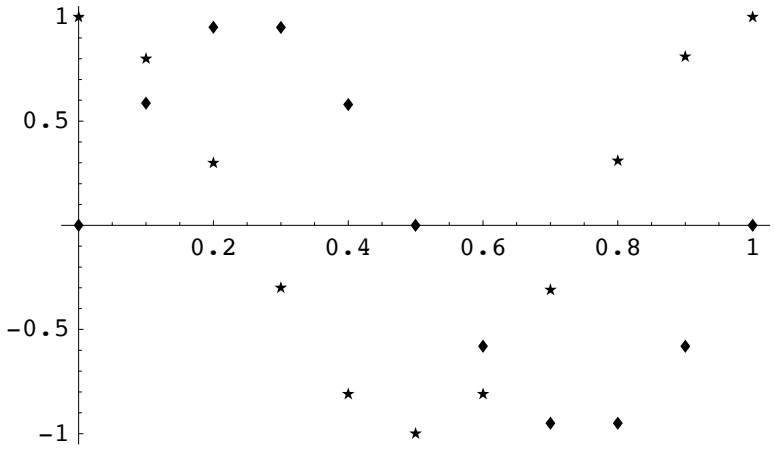
```
<<Graphics`MultipleListPlot`
```

Consideramos dos listas

```
lis1 = {{0, 0}, {0.1, 0.587}, {0.2, 0.951}, {0.3, 0.95}, {0.4, 0.58}, {0.5, 0},
   {0.6, -0.58}, {0.7, -0.95}, {0.8, -0.95}, {0.9, -0.58}, {1, 0}};
lis2 = {{0, 1}, {0.1, 0.8}, {0.2, 0.3}, {0.3, -0.3}, {0.4, -0.81},
   {0.5, -1}, {0.6, -0.81}, {0.7, -0.31}, {0.8, 0.31}, {0.9, 0.81}, {1, 1}};
```

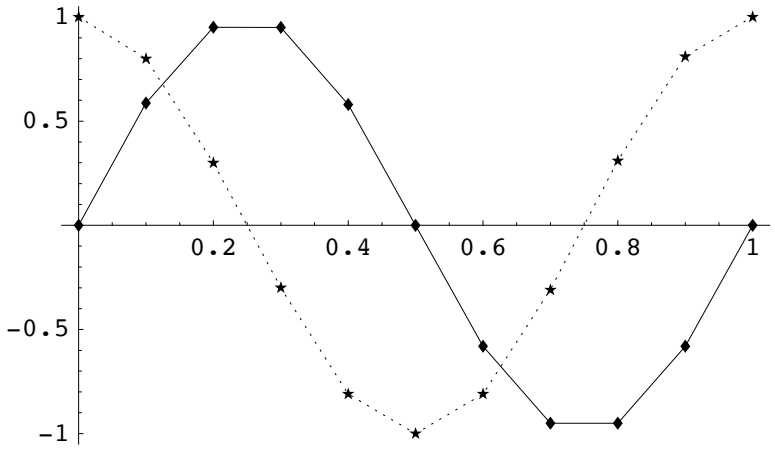
y las dibujamos juntas.

```
MultipleListPlot[lis1, lis2];
```



La verdad, es que a pesar de que cada gráfica es dibujada con tipos diferentes de puntos, queda un poco enredada. Para poder desenredarla, usaremos la opción **PlotJoined→True**. Veamos

```
MultipleListPlot[lis1, lis2, PlotJoined→True];
```



■ Interés y acumulación de interés

Ejemplo 1:

Si recibimos un préstamo de 10000€ que hemos de devolver a una tasa de interés anual del 5%, cuando transcurra un año tendremos que pagar

```
Print[10000 * (1 + 0.05), " €"];
```

```
10500. €
```

Ejemplo 2:

Si recibimos un préstamo de 10000€ que hemos de devolver a una tasa de interés del 5% anual compuesta cuatrimestralmente, cuando transcurra un año tendremos que pagar

```
Print[10000 * (1 + 0.05/3)^3, " €"];
```

```
10508.4 €
```

Ejemplo 3:

Si recibimos un préstamo de 10000€ que hemos de devolver a una tasa de interés del 5% anual compuesta continua, cuando transcurra un año tendremos que pagar

```
Print[10000 * ℯ^0.05, " €"];
```

```
10512.7 €
```

Ejemplo 4:

Si recibimos un préstamo de 10000€ que hemos de devolver a una tasa de interés del 5% anual compuesta continua, cuando transcurran *t* años tendremos que pagar

```
Print[10000 * ℯ^(0.05*t), " €"];
```

```
10000 ℯ^(0.05 t) €
```

Ejemplo 5:

Si invertimos 400€ al principio de cada uno de los siguientes 30 meses. ¿Qué capital obtendremos al final de los 30 meses si la tasa de interés nominal anual es del 5% compuesta mensualmente?

Al final de los 30 meses obtendremos

$$400 \sum_{i=1}^{30} \left(1 + \frac{0.05}{12}\right)^i \text{€}.$$

Fijémonos que lo que tenemos es la suma de los 30 primeros términos de una sucesión geométrica.

Recordemos que la suma de los *n* primeros términos de la sucesión geométrica

$$r, r^2, r^3, \ldots, r^n, \ldots$$

es

$$\sum_{i=1}^{n} r^i = \frac{r^{n+1} - r}{r - 1}.$$

Por tanto

$$400 \sum_{i=1}^{30} \left(1 + \frac{0.05}{12}\right)^i = 400 \, \frac{\left(1 + \frac{0.05}{12}\right)^{31} - \left(1 + \frac{0.05}{12}\right)}{1 + \frac{0.05}{12} - 1}$$

Si hacemos los cálculos con *Mathematica*

```
Print["400*((1 + 0.05/12)^31 - (1 + 0.05/12))/(0.05/12) = ", 400 * ((1 + 0.05/12)^31 - (1 + 0.05/12))/(0.05/12)];
```

$$400 * \frac{\left(1 + \frac{0.05}{12}\right)^{31} - \left(1 + \frac{0.05}{12}\right)}{\frac{0.05}{12}} = 12807.1$$

De tal manera que al cabo de 30 meses obtendremos 12807.1 €.

También podríamos haber hecho los cálculos directamente con *Mathematica* de la forma siguiente

```
400 * Sum[(1 + 0.05/12)^i, {i, 1, 30}]
12807.1
```

que efectivamente, da el mismo resultado.

Ejemplo 6:

Supongamos que, a partir del mes que viene, vamos a recibir 400€ al principio de cada mes de forma perpetua (cuando muramos algún heredero nuestro seguirá recibiendo dicha renta, y cuando el muera, un heredero suyo, y así sucesivamente). Si la tasa de interés nominal anual es del 5% compuesta mensualmente, ¿cuál es el valor actual de este flujo perpetuo?

Obsérvese que tenemos una cantidad infinita de ingresos. Lo que tendremos que hacer es calcular el valor actual de cada uno de estos ingresos y después sumarlos para obtener el valor actual de todo el flujo.

El valor actual del flujo perpetuo es

$$400\sum_{i=1}^{\infty}\left(1+\frac{0.05}{12}\right)^{-i}\text{€}.$$

Nótese que lo que tenemos es la suma de los infinitos términos de una sucesión geométrica.

Recordemos que la suma de los infinitos terminos de la sucesión geométrica

$$r, r^2, r^3, \ldots, r^n, \ldots$$

es, utilizando lo que vale la suma de los n primeros términos de una sucesión geométrica vista en el ejemplo anterior

$$\sum_{i=1}^{\infty} r^i = \lim_{n\to\infty}\sum_{i=1}^{n} r^i = \lim_{n\to\infty}\frac{r^{n+1}-r}{r-1} \underset{\text{Si } |r|<1}{=} \frac{r}{1-r}. \qquad (1)$$

Insistimos en que la suma de los infinitos términos de una sucesión geométrica sólo se puede calcular en el caso en que el valor absoluto de la razón sea menor que 1. En otro caso, la suma divergerá (será infinita).

En el caso en que la sucesió geométrica sea de la forma

$$r^{-1}, r^{-2}, r^{-3}, \ldots, r^{-n}, \ldots$$

podemos utilizar lo razonado anteriormente haciendo

$$\sum_{i=1}^{\infty} r^{-i} = \sum_{i=1}^{\infty}\left(\frac{1}{r}\right)^i$$

y ahora, siempre que $\left|\frac{1}{r}\right| < 1$, podemos calcular la suma utilizando la fórmula (1) y teniendo en cuenta que la razón es $\frac{1}{r}$,

$$\sum_{i=1}^{\infty} r^{-i} = \sum_{i=1}^{\infty} \left(\frac{1}{r}\right)^{i} \underset{(1)}{=} \frac{\frac{1}{r}}{1-\frac{1}{r}} = \frac{r^{-1}}{1-r^{-1}}$$

Volviendo a nuestro problema y utilizando esta expresión, el valor actual de este flujo perpetuo será

```
Print["400 ∑_{i=1}^{∞} (1 + 0.05/12)^-i = 400 (1 + 0.05/12)^-1 / (1 - (1 + 0.05/12)^-1) = ", 400 (1 + 0.05/12)^-1 / (1 - (1 + 0.05/12)^-1), "€"]
```

$$400 \sum_{i=1}^{\infty} \left(1 + \frac{0.05}{12}\right)^{-i} = 400 \frac{(1 + \frac{0.05}{12})^{-1}}{1 - (1 + \frac{0.05}{12})^{-1}} = 96000.€$$

También podríamos calcular la suma de la serie directamente con *Mathematica* mediante la instrucción

```
Limit[Sum[400 (1 + 0.05/12)^-i, {i, 1, n}], n → ∞]
```

```
96000
```

obteniendo el mismo resultado.

■ Mínimos cuadrados

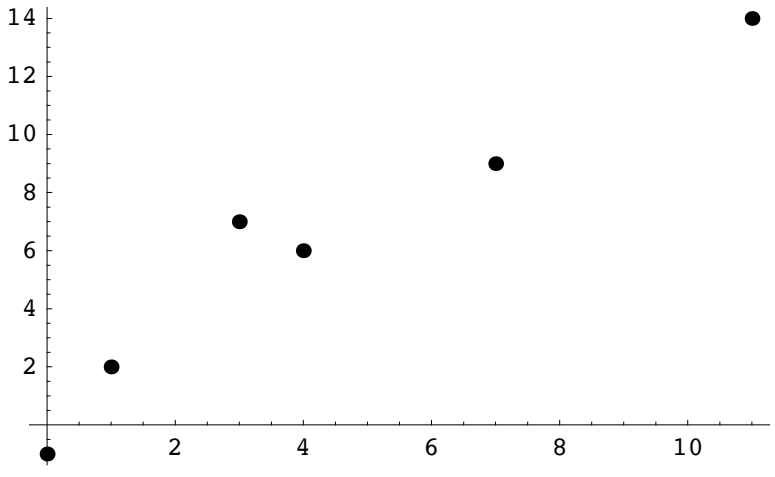
Mathematica permite calcular funciones que se ajustan a unas medidas dadas por el sistema de mínimos cuadrados. La función que lo hace posible es **`Fit[]`**. La sintáxis de esta función es *Fit[datos, funciones,var]* donde los *datos* son parejas de puntos del plano que se han de ajustar, *funciones* las funciones que harán el ajuste y *var* la variable de las funciones de ajuste. Veamos algunos ejemplos.

Suponemos que tenemos los puntos siguientes que queremos ajustar

```
puntos = {{0, -1}, {1, 2}, {3, 7}, {4, 6}, {7, 9}, {11, 14}};
```

Estos puntos dibujados en el plano son

```
plotpoints = ListPlot[puntos, PlotStyle -> PointSize[0.02]];
```



Los ajustes pueden hacerse con las funciones que queramos, aunque algunas se ajustarán mejor que otras. Veámoslo. En este primer caso, vamos a ajustar un polinomio de grado 1.

```
grado1 = Fit[puntos, {1, x}, x]
```

$0.776 + 1.244\,x$

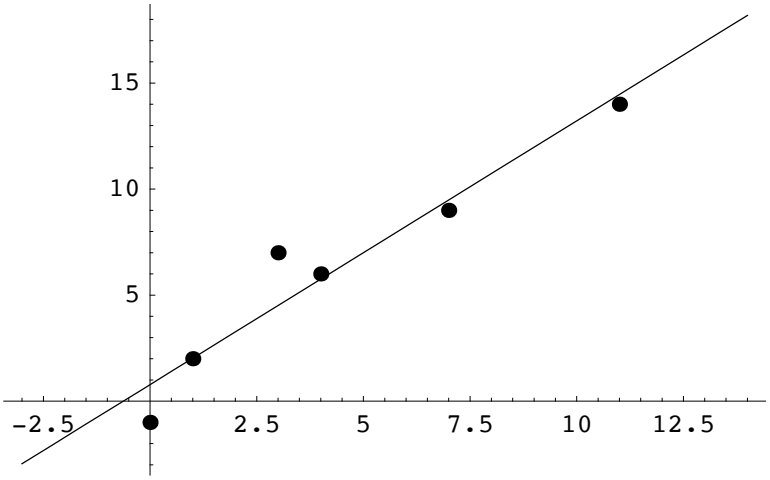
Este polinomio de grado 1, $0.776 + 1.244\,x$, es la recta que pasa más cercana (globalmente) de los puntos.

```
plotline = Plot[grado1, {x, -3, 14}, DisplayFunction → Identity];
```

DisplayFunction→Identity hace que no se dibuje la función porque lo haremos luego.

A continuación presentamos el gráfico con los puntos y la recta. El ajuste es bueno.

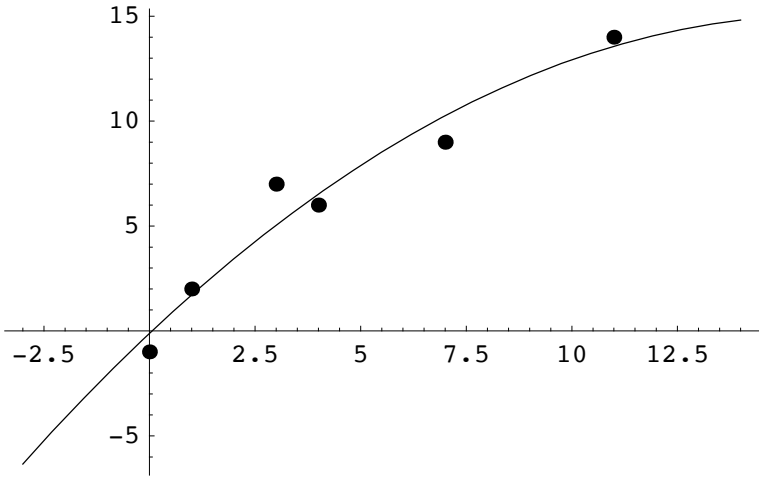
```
Show[plotpoints, plotline];
```



Podríamos hacerlo con un polinomio de grado dos

```
grado2 = Fit[puntos, {1, x, x^2}, x]
plotline = Plot[grado2, {x, -3, 14}, DisplayFunction → Identity];
Show[plotpoints, plotline];
```

$-0.113262 + 1.89613\,x - 0.0592841\,x^2$



que también parece ajustarse bien. Aunque en los ejemplos las funciones son polinomios, se puede poner cualquier tipo de función.

Álgebra lineal

■ Generación de matrices

En ocasiones nos hace falta generar matrices mediante una cierta regla numérica. Por ejemplo, si queremos generar una matriz cuadrada de orden tres que contenga en la posición (i, j) el valor $(-1)^{i+j} \cdot i \cdot j$, se utiliza la orden **Table[]**

```
Table[(-1)^(i+j) * i * j, {i, 1, 3}, {j, 1, 3}] // MatrixForm
```

$$\begin{pmatrix} 1 & -2 & 3 \\ -2 & 4 & -6 \\ 3 & -6 & 9 \end{pmatrix}$$

■ Cálculo de determinantes

Para calcular el determinante de una matriz (cuadrada) se utiliza la orden **Det[]**

```
A = {{1, 0, 2, 8, 0}, {2, 1, 4, 4, 3},
   {5, 6, 10, 0, 1/2}, {0, 0, 5, -3, 1}, {0, -1, 0, -2, 0}};
Det[
 A]
```

```
745
```

```
B = {{1, x, x^2}, {0, 1, -x}, {1/2, 0, x/2}};
Det[B]
```

$$\frac{x}{2} - x^2$$

Con esta orden podemos obtener las fórmulas para calcular el determinante de matrices cuadradas de orden dos

```
Det[ {{a, b}, {c, d}} ]
```

```
-b c + a d
```

o tres (regla de Sarrus)

```
m = Array[a, {3, 3}];
Det[ m ]
```

```
-a[1, 3] a[2, 2] a[3, 1] + a[1, 2] a[2, 3] a[3, 1] + a[1, 3] a[2, 1] a[3, 2] -
 a[1, 1] a[2, 3] a[3, 2] - a[1, 2] a[2, 1] a[3, 3] + a[1, 1] a[2, 2] a[3, 3]
```

■ Cálculo del rango de una matriz

Recordemos que el rango de una matriz de orden $n \times m$ es el número de filas o columnas linealmente independientes. Además, se puede probar que este cálculo se puede hacer por filas o por columnas, porque coincide. Luego, si $n \leq m$, tendremos que $\text{Rang}(A) \leq n$.

Para hacer el cálculo práctico del rango de una matriz lo más sencillo es calcularlo mediante sus menores, porque el rango es el tamaño del mayor menor no nulo que tenga la matriz.

Por ejemplo, como la matriz siguiente A es de orden 4×5, su rango será como máximo de 4

```
A = {{1, 2, 0, 1, 3}, {0, 10, 1, 0, 3},
   {1, 1, 1, 0, -1}, {0, 0, 2, 1, 0}} // MatrixForm
```

$$\begin{pmatrix} 1 & 2 & 0 & 1 & 3 \\ 0 & 10 & 1 & 0 & 3 \\ 1 & 1 & 1 & 0 & -1 \\ 0 & 0 & 2 & 1 & 0 \end{pmatrix}$$

Con el orden siguiente calcularemos los menores de orden cuatro mediante la instrucción **Minors[]**

```
A = {{1, 2, 0, 1, 3}, {0, 10, 1, 0, 3}, {1, 1, 1, 0, -1}, {0, 0, 2, 1, 0}};
Minors[A, 4]
```

```
{{31, 74, 37, 13, 7}}
```

y como que hay al menos un menor de orden cuatro no nulo (en realidad todos los menores de orden cuatro son no nulos), se tiene que $\text{Rang}(A) = 4$.

Consideramos ahora la matriz B de tamaño 3×4

```
B = {{2, 1, 1, 3}, {3, 4, -1, 2}, {1, 1, 0, 1}};
Minors[B, 3]
```

```
{{0, 0, 0, 0}}
```

como todos los menores de orden tres son nulos, su rango es como máximo 2. Y de hecho, $\text{Rang}(B) = 2$ ya que, tiene al menos un menor de orden dos no nulo

```
B = {{2, 1, 1, 3}, {3, 4, -1, 2}, {1, 1, 0, 1}};
Minors[B, 2]
```

```
{{5, -5, -5, -5, -10, 5}, {1, -1, -1, -1, -2, 1}, {-1, 1, 1, 1, 2, -1}}
```

Consideremos finalmente un ejemplo donde el rango se ha de calcular en función de un parámetro:

```
M = {{1, 2, 1, 1}, {-1, 1, 0, 1}, {0, -3, -k, -3}, {1, 2, k, k}};
Minors[M, 4]
```

```
{{-6 + 9 k - 3 k^2}}
```

Luego, como

```
Solve[-6 + 9 k - 3 k^2 == 0, k]
```

```
{{k → 1}, {k → 2}}
```

si $k \in \mathbb{R} - \{1, 2\}$, la matriz M tiene un menor (el único) de orden cuatro no nulo y por tanto $\text{Rang}(M) = 4$. Estudiamos los casos $k = 1$ y $k = 2$

```
M1 = {{1, 2, 1, 1}, {-1, 1, 0, 1}, {0, -3, -1, -3}, {1, 2, 1, 1}};
M2 = {{1, 2, 1, 1}, {-1, 1, 0, 1}, {0, -3, -2, -3}, {1, 2, 2, 2}};
Minors[M1, 3]
Minors[M2, 3]
```

```
{{0, -3, -1, 1}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, -3, -1, 1}}
```

```
{{-3, -3, 1, 2}, {3, 3, -1, -2}, {-3, -3, 1, 2}, {0, 0, 0, 0}}
```

en consecuencia, en ambos casos $\text{Rang}(M) = 3$.

■ Matriz inversa

Para calcular la inversa de una matriz (cuadrada) se utiliza la orden **Inverse[]**

```
A = {{1, 2, 3, 0}, {1, 0, 2, 3}, {3, 0, 4, 0}, {0, 0, 2, -4}};
Inverse[A] // MatrixForm
```

$$\begin{pmatrix} 0 & -\frac{8}{13} & \frac{7}{13} & -\frac{6}{13} \\ \frac{1}{2} & -\frac{5}{13} & -\frac{1}{26} & -\frac{15}{52} \\ 0 & \frac{6}{13} & -\frac{2}{13} & \frac{9}{26} \\ 0 & \frac{3}{13} & -\frac{1}{13} & -\frac{1}{13} \end{pmatrix}$$

Si la matriz es singular, es decir, no tiene inversa (porque su determinante es nulo), *Mathematica* nos lo dice

```
B = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
Inverse[B] // MatrixForm
```

```
Inverse::sing : Matrix {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}} is singular.
```

■ Cálculo del polinomio característico de una matriz

El cálculo del polinomio característico de una matriz (cuadrada) se hace mediante la orden **CharacteristicPolynomial[]**

```
A = {{4, 1, 0, 1}, {2, 3, 0, 1}, {-2, 1, 2, 3}, {2, -1, 0, 5}};
CharacteristicPolynomial[A, λ]
```

$$96 - 136\,\lambda + 68\,\lambda^2 - 14\,\lambda^3 + \lambda^4$$

■ Cálculo de los valores y vectores propios de una matriz

El cálculo del espectro de una matriz (cuadrada), es decir, de sus valores propios se hace mediante la orden **Eigenvalues[]**

```
A = {{4, 1, 0, 1}, {2, 3, 0, 1}, {-2, 1, 2, 3}, {2, -1, 0, 5}};
Eigenvalues[A]
```

```
{2, 2, 4, 6}
```

Observamos que eso es equivalente a

```
A = {{4, 1, 0, 1}, {2, 3, 0, 1}, {-2, 1, 2, 3}, {2, -1, 0, 5}};
Solve[CharacteristicPolynomial[A, λ] == 0, λ]
```

```
{{λ → 2}, {λ → 2}, {λ → 4}, {λ → 6}}
```

Mientras que para determinar los vectores propios se utiliza la orden **Eigenvectors[]**

```
A = {{4, 1, 0, 1}, {2, 3, 0, 1}, {-2, 1, 2, 3}, {2, -1, 0, 5}};
Eigenvectors[A]
```

```
{{0, 0, 1, 0}, {0, 0, 0, 0}, {-1, -1, 2, 1}, {2, 2, 1, 2}}
```

Pero estas dos tareas se pueden hacer a la vez mediante una única orden **Eigensystem[]**

```
A = {{4, 1, 0, 1}, {2, 3, 0, 1}, {-2, 1, 2, 3}, {2, -1, 0, 5}};
Eigensystem[A]
```

```
{{2, 2, 4, 6}, {{0, 0, 1, 0}, {0, 0, 0, 0}, {-1, -1, 2, 1}, {2, 2, 1, 2}}}
```

donde conviene señalar que primero nos da los valores propios y después los vectores propios ordenados de forma que están asociados a cada uno de los valores propios, respectivamente.

■ Diagonalizar una matriz

Recordemos que diagonalizar una matriz cuadrada A consiste en factorizarla como $A = P\,D\,P^{-1}$, siendo D y P dos matrices del mismo orden que A, de manera que D es una matriz diagonal y está formada por los valores propios de A y P es una matriz invertible que tiene como columnas los vectores propios asociados a los valores propios (siguiendo el orden en que estén colocados los valores propios en la diagonal de la matriz D).

Por ejemplo, diagonalizamos la matriz $A = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 3 & 1 \\ -4 & -3 & -1 \end{pmatrix}$, y comprobamos el resultado. A continuación, las dos primeras instrucciones calcularán los valores y los vectores propios pero no devolverán ningún resultado. La tercera instrucción, nos devuelve la matriz de paso P. Y la cuarta, la matriz diagonal D.

```
{λ, μ, η} = Eigenvalues[A = {{1, 0, 0}, {4, 3, 1}, {-4, -3, -1}}];
{u, v, w} = Eigenvectors[A];
Transpose[{u, v, w}] // MatrixForm
DiagonalMatrix[{λ, μ, η}] // MatrixForm
Inverse[Transpose[{u, v, w}]] // MatrixForm
```

$$\begin{pmatrix} 0 & 1 & 0 \\ -1 & -4 & -1 \\ 3 & 4 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \\ -4 & -\frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

Ahora comprobemos que la diagonalización es correcta.

```
Transpose[{u, v, w}].DiagonalMatrix[{λ, μ, η}].
  Inverse[Transpose[{u, v, w}]] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 3 & 1 \\ -4 & -3 & -1 \end{pmatrix}$$

Conviene observar que si la comprobación la tuvieramos que hacer a mano, evitaríamos el cálculo de la matriz inversa que es muy pesado, y estudiaríamos $P\,D = A\,P$, que es mas corto.

Recordemos que no toda matriz cuadrada es diagonalizable. Por ejemplo, la matriz A del apartado anterior sobre valores y vectores propios no es diagonalizable, ya que sus vectores propios no son linealmente independientes (observemos que *Mathematica* nos da el vector nulo entre éstos).

■ Potencia de una matriz

Si una matriz cuadrada A es diagonalizable ($A = P\,D\,P^{-1}$) entonces sabemos que podemos calcular su potencia como $A^n = P\,D^n\,P^{-1}$ ya que

$$A^n = (P\,D\,P^{-1})\ (P\,D\,P^{-1})\cdots(P\,D\,P^{-1})\ (P\,D\,P^{-1}) = P\,D(P^{-1}\,P)\,D(P^{-1}\,P)\cdots\mathrm{D}\,(P^{-1}\,P)\,D\,P^{-1} = P\,D^n\,P^{-1}$$

pero si la matriz no es diagonalizable esta técnica no se puede aprovechar. Sin embargo, *Mathematica* tiene una orden para calcular la potencia de una matriz cuadrada cualquiera, **MatrixPower[].** Calculamos como ejemplo la potencia décima de una matriz particular

```
A = {{1, 0, 2}, {-1, 0, 0}, {1, -1, 0}};
Print["A^10 = ", MatrixPower[A, 10] // MatrixForm];
```

$$\mathrm{A}^{10} = \begin{pmatrix} 2095 & -814 & 1846 \\ -923 & 358 & -814 \\ 1330 & -516 & 1172 \end{pmatrix}$$

■ La exponencial de una matriz

Para calcular la exponencial de una matriz cuadrada A, es decir, e^A se utiliza la orden **MatrixExp[]**

```
A = {{2, 1}, {0, 1}};
Print["e^A = ", MatrixExp[A, 10] // MatrixForm];
```

$$\mathrm{e}^{\mathrm{A}} = \begin{pmatrix} \mathrm{e}^{20} & -\mathrm{e}^{10} + \mathrm{e}^{20} \\ 0 & \mathrm{e}^{10} \end{pmatrix}$$

Funciones

■ Definición de funciones a trozos: la instrucción Which

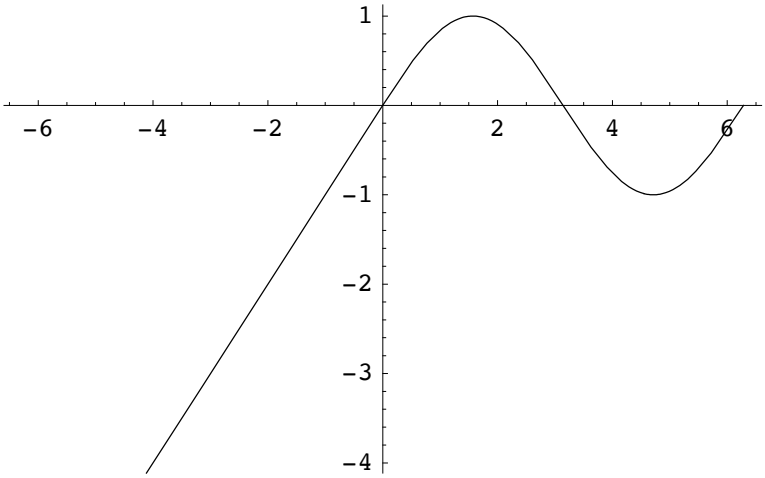
Muchas veces conviene definir funciones a trozos, es decir, funciones que según el intervalo en que se muevan tienen comportamientos diferentes. La instrucción de *Mathematica* **Which[]** es la que nos permite definir funciones con estas características. Esta instrucción tiene como sintaxi *Which[test1, valor1, test2, valor2,...]* donde comprueba si el *test1* satiface, si es así devolverá el *valor1*, si no pasa a comprobar si satisface el *test2*, y así sucesivamente.

Veamos un ejemplo. La función siguiente es la identidad para valores negativos y *sen(x)* para valores positivos.

```
f[x_] := Which[x < 0, x,
               x ≥ 0, Sin[x]]
```

La dibujaremos entre -2π i 2π para ver el efecto del cambio de una función otra en $x = 0$

```
Plot[f[x], {x, -2 π, 2 π}];
```



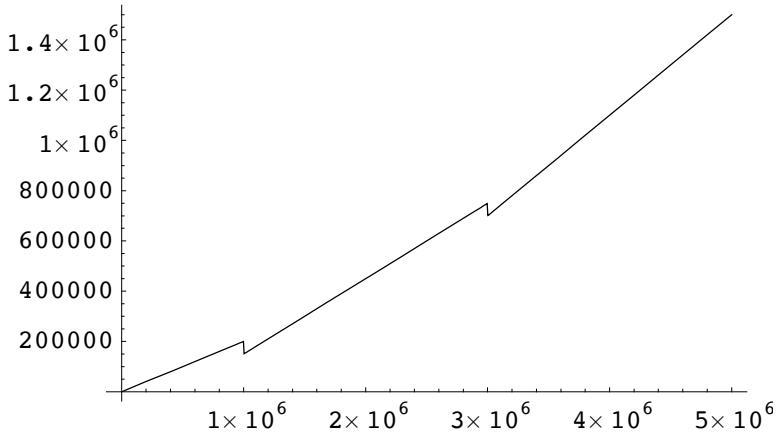
Otro ejemplo. La cantidad de impuestos a pagar *I* para una persona es: el 20% de su renta *R* si no supera el millón de pesetas; 150.000 pesetas más el 30% de lo que pase del millón hasta 3 millones; 700.000 pesetas más el 40% del que pase de 3 millones.

Vamos a escribir la función de impuestos en función de la renta, que será la función definida a trozos siguiente

```
Imp[R_] := Which[R ≤ 1000000,              R * 0.2,
                 1000000 < R ≤ 3000000,    150000 + 0.3 (R - 1000000),
                 R > 3000000,              700000 + 0.4 (R - 3000000)]
```

La dibujaremos para ver los efectos de cambio entre funciones

```
Plot[Imp[R], {R, 0, 5000000}, AxesOrigin → {0, 0}];
```



En los dos picos que aparecen, debería de haber una discontinuidad (un salto) que *Mathematica* dibuja continuamente.

■ Cálculo de límites

Para el cálculo de límites, *Mathematica* dispone de la instrucción **`Limit[]`** con la sintáxis *Limit[expr,x->a]*, que calcula el valor del límite de *expr* cuando *x* tiende a *a*. El valor de *a* puede ser finito o infinito, como podemos observar en los ejemplos siguientes,

```
Limit[Sin[x], x → π/2]
```

```
1
```

```
Limit[e^-x, x → ∞]
```

```
0
```

```
Limit[e^-x * x, x → -∞]
```

```
-∞
```

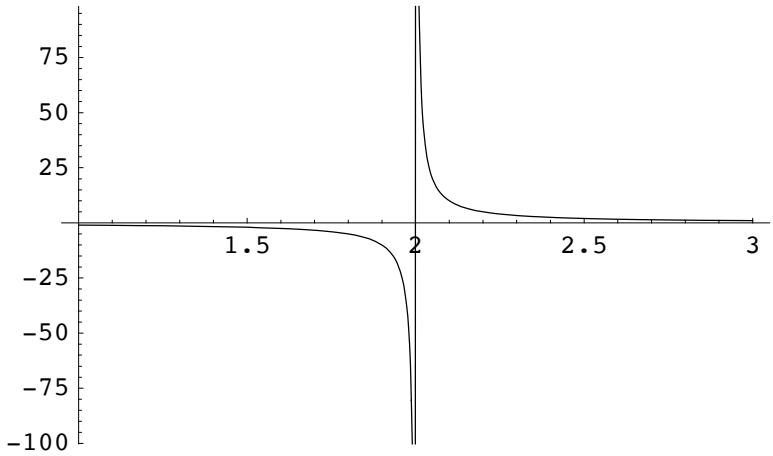
Mathematica también nos permite calcular límites laterales con la opción *Direction*. Si queremos calcular el límite por la izquierda escribimos *Direction->1*, y si lo queremos calcular por la derecha, *Direction->-1.* Por ejemplo

```
Limit[1/(x - 2), x → 2]
```

```
∞
```

este límite no existe.

```
Plot[1/(x - 2), {x, 1, 3}];
```



Eso significa o que los límites laterales son distintos o que alguno de estos no existen, comprobémoslo

```
Limit[1/(x - 2), x → 2, Direction → 1]
```

$-\infty$

```
Limit[1/(x - 2), x → 2, Direction → -1]
```

∞

De lo que deducimos que $\frac{1}{x-2}$ presenta una asíntota vertical en $x = 2$.

Si existe el límite es porque los límites laterales existen y son iguales,

```
Limit[Sin[x] / x, x → 0, Direction → 1]
```

1

```
Limit[Sin[x] / x, x → 0, Direction → -1]
```

1

por ello

```
Limit[Sin[x] / x, x → 0]
```

1

Hasta ahora hemos calculado límites de modelos continuos, es decir, cuando la variable independiente puede tomar cualquier valor real (tenemos funciones $f(x)$, con x un número real). Con la misma instrucción podemos calcular límites de modelos discretos (sucesiones), es decir, cuando la variable independiente sólo toma valores naturales (1,2,...), que representamos con $\{a(n); n = 1, 2, 3, \ldots\}$

Por ejemplo, vamos a considerar la sucesión $\{a(n); n = 1, 2, 3, \ldots\}$, siendo $a(n) = \left(1 + \frac{1}{n}\right)^n$. Para calcular el valor al cual tiende $a(n)$, cuando n tiende a infinito, escribimos

```
Limit[(1 + 1/n)^n, n → ∞]
```

```
e
```

■ Cálculo de derivadas y derivadas sucesivas

Una forma de calcular derivadas es con la instrucción **D[]** con la sintáxis *D[f[x],x]*. Por ejemplo

```
f[x_] := (x^3 - 8 x^2 + 7)^2 / (x^2 - 1)
```

```
D[f[x], x]
```

$$\frac{2\,(-16\,x+3\,x^2)\,(7-8\,x^2+x^3)}{-1+x^2}-\frac{2\,x\,(7-8\,x^2+x^3)^2}{(-1+x^2)^2}$$

También podemos deducir las reglas de derivación, por ejemplo la del producto

```
D[f1[x] * f2[x], x]
```

```
f2[x] f1'[x] + f1[x] f2'[x]
```

Podemos calcular las derivadas sucesivas de una función utilizando la sintáxis *D[f[x],{x,n}]*, que calcula la derivada *n*-ésima de $f(x)$.

```
f[x_] := k Sin[x]
```

```
D[f[x], {x, 1}]
```

```
k Cos[x]
```

```
D[f[x], {x, 2}]
```

```
-k Sin[x]
```

```
D[f[x], {x, 20}]
```

```
k Sin[x]
```

Otra manera de calcular la derivada de la función anterior es

```
f'[x]
```

```
k Cos[x]
```

```
f''[x]
```

```
-k Sin[x]
```

Como cabía esperar, hemos obtenido el mismo resultado.

También podemos evaluar directamente derivadas en un punto

```
f'[0]
```

```
k
```

■ Desarrollo de Taylor

Dada una función $f(x)$ derivable en $x = a$ $n+1$ veces, se puede expresar de la forma

$$f(x) = f(a) + f'(a)(x-a) + f''(a)\frac{(x-a)^2}{2!} + \cdots + f^{(n)}(a)\frac{(x-a)^n}{n!} + R_{n+1}(c)$$

para un determinado c cercano a a. $R_{n+1}(c)$ es el residuo de Lagrange y viene dado por

$$R_{n+1}(c) = f^{(n+1)}(c)\frac{(x-a)^{n+1}}{(n+1)!}$$

y

$$P_n f(x, a) = f(a) + f'(a)(x-a) + f''(a)\frac{(x-a)^2}{2!} + \cdots + f^{(n)}(a)\frac{(x-a)^n}{n!}$$

es el polinomip de Taylor de grado n.

Las instrucciones **Normal[Series[]]**con la sintáxis Normal[Series[f[x],{x,a,n}]] nos proporcionan el polinomio de Taylor de a lo sumo grado n de la función $f(x)$ alrededor de a. Por ejemplo,

```
Normal[Series[Exp[x], {x, 0, 5}]]
```

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}$$

es el polinomio de Taylor de e^x de grado 5 alrededor de 0.

Si omitimos la instrucción **Normal[]** obtenemos el polinomio de Taylor y el orden del residuo de Lagrange

```
Series[Exp[x], {x, 0, 5}]
```

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O[x]^6$$

El polinomio de Taylor de una función nos da una aproximación de la función cerca del punto a. Cuanto mayor sea el grado, n, mejor será la aproximación.

■ Cálculo y clasificación de los máximos y mínimos de una función

Vamos a desarrollar dos ejemplos.

Ejemplo 1:

Obtener los extremos (máximos y mínimos) de la función $y = \frac{x^3}{3} - 6x^2 + 32x$ definida en el intervalo [0,12].

Como se trata de una función definida en un intervalo cerrado, tendremos que estudiar los puntos críticos y los extremos del intervalo.

Comenzamos calculando los puntos críticos, que son aquellos que anulan la primera derivada

```
f[x_] := x^3/3 - 6 x^2 + 32 x
```

```
Solve[f'[x] == 0, x]
```

$\{\{x \to 4\}, \{x \to 8\}\}$

De modo que los puntos críticos son $x = 4$ y $x = 8$. Vamos a ver si en estos puntos la función tiene un máximo o mínimo relativo. Por ello evaluamos la segunda derivada en estos puntos

```
f''[4]
```

-4

```
f''[8]
```

4

Como $f''(4) < 0$, en $x = 4$ $f(x)$ tiene un máximo y como $f''(8) > 0$, en $x = 8$ la función tiene un mínimo.

Vamos a ver qué ocurre en los extremos. Evaluaremos lo que vale la función en los extremos y en los puntos críticos

```
{f[0], f[4], f[8], f[12]}
```

$\left\{0, \frac{160}{3}, \frac{128}{3}, 96\right\}$

Así que el máximo y mínimo absoluto se encuentra en los extremos. Resumiendo, hemos obtenido:

en $x = 0$, $f(x)$ tiene el mínimo absoluto, 0

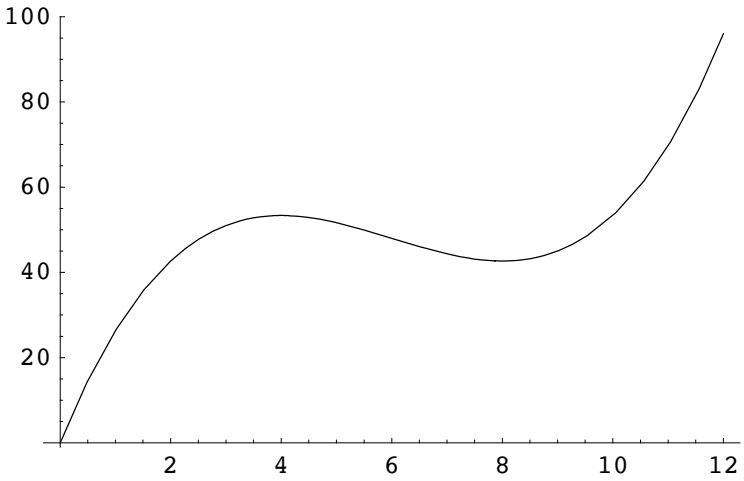
en $x = 4$, $f(x)$ tiene un máximo relativo, $\frac{160}{3}$

en $x = 8$, $f(x)$ tiene un mínimo relativo, $\frac{128}{3}$

en $x = 12$, $f(x)$ tiene el máximo absoluto, 96.

Lo podemos comprobar representando la función

```
Plot[x^3/3 - 6 x^2 + 32 x, {x, 0, 12}, PlotRange -> {-1, 100}]
```



Ejemplo 2:

Obtener los extremos (máximos y mínimos) de la función $y = x^4 - 8x^3 + 24x^2 - 32x + 16$.

Para obtener los candidatos a extremos, calcularemos los puntos críticos

```
g[x_] := x^4 - 8 x^3 + 24 x^2 - 32 x + 16

Solve[g'[x] == 0, x]
{{x → 2}, {x → 2}, {x → 2}}
```

El único punto crítico es $x = 2$. Analizaremos lo que pasa en este punto con el criterio de la segunda derivada

```
g''[2]
0
```

Como la segunda derivada en el punto crítico nos da cero, no obtenemos ninguna información. Deberemos de utilizar el criterio de la derivada *n*-ésima. Calcularemos las derivadas sucesivas $g^{(n)}(x)$ hasta obtener la primera que no se anule

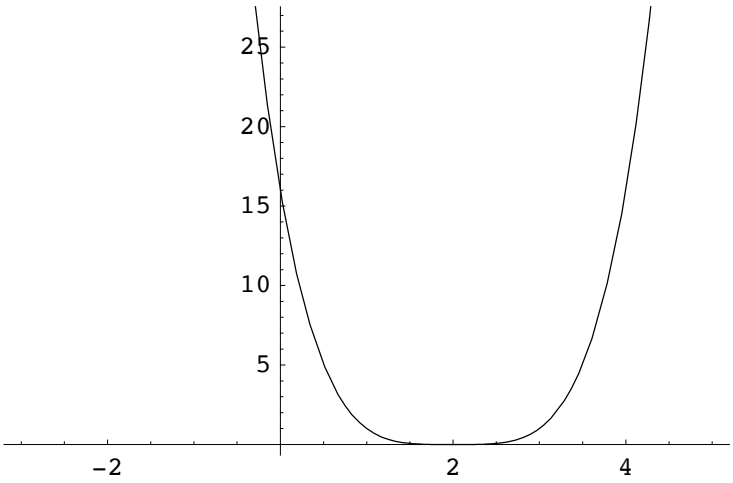
```
g'''[2]
0

g''''[2]
24
```

Como la primera derivada que no se anula es la cuarta, cuatro es un número par, y además $g^{(4)}(2) > 0$, en $x = 2$ la función tiene un mínimo relativo.

Si dibujamos la función

```
Plot[g[x], {x, -3, 5}]
```



observamos que el mínimo es absoluto.

Resumiendo, en $x = 2$, la función $g(x)$ tiene un mínimo absoluto y no tiene ningún máximo.

■ Cálculo de integrales

Mathematica nos permite calcular integrales definidas e indefinidas con la instrucción **`Integrate[]`**. Para el cálculo de integrales indefinidas utilizaremos la sintáxis *Integrate[f[x],x]*, o $\int_{\square}^{\square} \square \, d\square$ de la paleta, por ejemplo

```
Integrate[1/(E^x (8 + E^-x)^2), x]
```

$$-\frac{e^{-2x}\,(1+8\,e^{x})}{8\,(8+e^{-x})^{2}}$$

Fijémonos que *Mathematica* sólo nos devuelve una primitiva. Para obtenerlas todas tendremos que sumar una constante de integración, por tanto

$$\int \frac{1}{e^{x}\,(8+e^{-x})^{2}}\,dx = -\frac{e^{-2x}\,(1+8\,e^{x})}{8\,(8+e^{-x})^{2}} + C$$

Cuando la integral sea definida utilizaremos la sintáxis *Integrate[f[x],{x,a,b}]*, donde *a* y *b* son los límites de integración inferior y superior, respectivamente.

Para calcular $\int_0^p \frac{2x^3+x}{x^4+x^2+1}\,dx$ escribimos

```
Integrate[(2 x^3 + x)/(x^4 + x^2 + 1), {x, 0, p}]
```

$$\frac{1}{2}\,\text{Log}[1+p^2+p^4]$$

La sintáxis no cambia si tenemos una integral impropia. Por ejemplo

```
Integrate[1/(x^2 + 1), {x, -∞, ∞}]
```

$$\pi$$

Miscelánea

■ Comprobación de que dos resultados son iguales

Muchas veces nos salen expresiones que no parecen iguales, pero que se hace necesario comprobarlo. Por eso se puede usar **`TrueQ[]`** y **`FullSimplify[]`**.

`TrueQ[]` devuelve verdadero (*True*) si la igualdad es cierta o falso (*False*) si no lo es, aunque devolverá *False* si la respuesta no está clara. En estos casos convendrá usar **`FullSimplify[]`** para facilitar a **`TrueQ[]`** la tarea. Veamos algunos ejemplos.

```
TrueQ[x/2 == 1/2 x]
```

```
True
```

```
TrueQ[x/2 == 1/2 x^2]
```

```
False
```

Hay ocasiones, en que si las expresiones son un poco complicadas devuelve *False* cuando es verdadero. Hacemos una descomposición en fracciones simples

```
Apart[1/((3 + x) (1 + x)^3)]
```

$$\frac{1}{2\,(1+x)^3} - \frac{1}{4\,(1+x)^2} + \frac{1}{8\,(1+x)} - \frac{1}{8\,(3+x)}$$

Por tanto, la expresión siguiente debería de ser verdadera

```
TrueQ[1/((3 + x) (1 + x)^3) == 1/(2 (1 + x)^3) - 1/(4 (1 + x)^2) + 1/(8 (1 + x)) - 1/(8 (3 + x))]
```

```
False
```

pero resulta Falsa. Para salvar este tipo de contingencias, hemos de usar **FullSimplify[]**

```
TrueQ[1/((3 + x) (1 + x)^3) == FullSimplify[1/(2 (1 + x)^3) - 1/(4 (1 + x)^2) + 1/(8 (1 + x)) - 1/(8 (3 + x))]]
```

```
True
```

Para matrices también es válido

```
m = {{1, 6}, {3, 2}};
TrueQ[m.Inverse[m] == IdentityMatrix[2]]
```

```
True
```

■ Descomposición en fracciones simples

La descomposición en fracciones simples se puede hacer en *Mathematica* con la función **Apart[]**. Por ejemplo

```
Apart[1/((3 + x) (1 + x)^3)]
```

$$\frac{1}{2\,(1+x)^3} - \frac{1}{4\,(1+x)^2} + \frac{1}{8\,(1+x)} - \frac{1}{8\,(3+x)}$$

Eso quiere decir que

```
Print[1/((3 + x) (1 + x)^3), " = ", Apart[1/((3 + x) (1 + x)^3)]]
```

$$\frac{1}{(3+x)\,(1+x)^3} = \frac{1}{2\,(1+x)^3} - \frac{1}{4\,(1+x)^2} + \frac{1}{8\,(1+x)} - \frac{1}{8\,(3+x)}$$

La operación contraria a **Apart[]** es **Together[]**. Veámoslo haciendo lo contrario que en el ejemplo anterior

```
Together[1/(2 (1 + x)^3) - 1/(4 (1 + x)^2) + 1/(8 (1 + x)) - 1/(8 (3 + x))]
```

$$\frac{1}{(1+x)^3\,(3+x)}$$

■ Binomio de Newton y fórmulas derivadas de este

El binomio de Newton es el correspondiente a la fórmula

$$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}\,b + \binom{n}{2}a^{n-2}\,b^2 + \cdots + \binom{n}{n-2}a^2\,b^{n-2} + \binom{n}{n-1}a\,b^{n-1} + \binom{n}{n}b^n$$

donde

$\binom{n}{m} = \frac{n!}{m!\,(n-m)!}$.

A continuación, vamos a poner una serie de igualdades derivadas del binomio de Newton y que son muy utilizadas en operaciones matemáticas

```
Print["(a+b)^2 = ", Expand[(a + b)^2]];
Print["(a-b)^2 = ", Expand[(a - b)^2]];
Print[""];
Print["(a+b)^3 = ", Expand[(a + b)^3]];
Print["(a-b)^3 = ", Expand[(a - b)^3]];
Print[""];
Print["(a+b)(a-b) = ", Expand[(a + b) (a - b)]];
Print[""];
Print["(a+b+c)^2 = ", Expand[(a + b + c)^2]];
```

```
(a+b)^2 = a^2 + 2 a b + b^2
(a-b)^2 = a^2 - 2 a b + b^2

(a+b)^3 = a^3 + 3 a^2 b + 3 a b^2 + b^3
(a-b)^3 = a^3 - 3 a^2 b + 3 a b^2 - b^3

(a+b) (a-b) = a^2 - b^2

(a+b+c)^2 = a^2 + 2 a b + b^2 + 2 a c + 2 b c + c^2
```

Variando estas fórmulas, podemos obtener el desarrollo de la expresión que deseemos.

■ Mensajes de error y de advertencia de *Mathematica*

No hay mucha costumbre de leer los mensajes en rojo o azul que da *Mathematica*, y que una lectura apropiada permitiría corregir errores y ahorrar desesperaciones del tipo "no sé, no va" o "es que no me sale".

Por otra parte, la sintáxis de *Mathematica* es en ocasiones difícil y es conveniente comprobar cada término que escribimos para no cometer errores.

Para ilustrar los errores y mensajes más comunes, veremos unos ejemplos e interpretaremos sobre lo que quieren decir.

Errores *General spell*

Cuando aparece un aviso *General spell* quiere decir que *Mathematica* ha detectado que alguna de las palabras que hemos escrito es muy parecida a alguna que ya tiene definida *Mathematica* o una que hemos definido nosotros anteriormente. De esta manera *Mathematica* nos avisa que este parecido puede ser porque nos hemos equivocado en escribirla.

Veamos un ejemplo en que definiremos la variable *Numero* y después querremos llamarla como *numero*

```
Numero = 30;

numero
General::spell1 : Possible spelling error: new symbol
   name "numero" is similar to existing symbol "Numero".
numero
```

Como se puede leer en el mensaje 'el nuevo símbolo "numero" es parecido al símbolo existente "Numero"'.

Eso también puede ocurrir con las funciones de *Mathematica*

```
sin[π/2]
General::spell1 : Possible spelling error: new
   symbol name "sin" is similar to existing symbol "Sin".
sin[π/2]
```

Si lo hubiésemos escrito correctamente sería

```
Sin[π/2]
1
```

Errores de reutilización de variables

Este error ocurre con mucha frecuencia y **siempre se ha de estar muy alerta**. Ya que durante una sesión *Mathematica* se guarda TODAS las variables y operaciones que hacemos, resulta fácil, sin querer, usar una variable que ya hace tiempo que hemos usado y que tiene un valor asignado, para otras tareas, y claro, el resultado que sale no tiene nada que ver con el esperado.

Supongamos que al principio de la sesión definimos la variable *a* como

```
a = {{1, 2}, {3, 4}};
```

y que al cabo de una hora, que ya no nos acordamos que habíamos definido la variable *a* después de haber hecho operaciones, queremos calcular

```
{{a, b}, {c, d}}.{{x}, {y}} // MatrixForm
```

```
Dot::rect : Non-rectangular tensor encountered.
```

```
{{{{1, 2}, {3, 4}}, b}, {c, d}}.{{x}, {y}}
```

Evidentemente nos da un error, indicando que la matriz $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ no es una matriz rectangular y por tanto no se puede hacer ninguna operación. Y además, no sabemos qué hace aquí *{{1,2},{3,4}}* porque ya no nos acordamos de cuándo definimos la variable *a y* que éste era su valor asignado.

Para evitar esto, es conveniente hacer uso de la instrucción **Clear[]**, que borra el contenido de las variables y nos evita errores, o lo que es peor, resultados aparentemente correctos que no tienen nada que ver con el bueno.

Para que la anterior operación no nos produzca errores, tendremos que hacer

```
Clear[a];
{{a, b}, {c, d}}.{{x}, {y}} // MatrixForm
```

$$\begin{pmatrix} a\,x + b\,y \\ c\,x + d\,y \end{pmatrix}$$

y el resultado ahora es correcto y no tiene ningún error.

Errores al multiplicar matrices

```
{{2}, {3}}.{{a, b}, {c, d}}
```

```
Dot::dotsh :
 Tensors {{2}, {3}} and {{a, b}, {c, d}} have incompatible shapes.
```

```
{{2}, {3}}.{{a, b}, {c, d}}
```

Este mensaje de error nos indica que el producto de matrices que queremos hacer NO se puede porque el tamaño de las matrices no es compatible.

Errores al resolver ecuaciones

```
Solve[Cos[x] == Sin[x], x]
```

```
Solve::ifun : Inverse functions are being
   used by Solve, so some solutions may not be found.
```

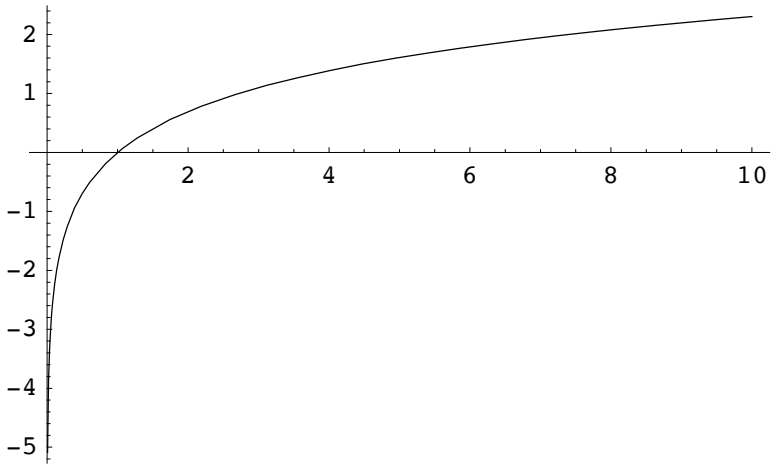
$$\left\{\left\{x \to -\frac{3\,\pi}{4}\right\}, \left\{x \to \frac{\pi}{4}\right\}\right\}$$

La instrucción *Solve[]* intenta resolver de forma exacta la ecuación calculando inversas de las funciones seno y coseno, pero eso significa que las soluciones sólo se pueden encontar en unos ciertos intervalos. En pocas palabras, *Mathematica* nos da unas soluciones, aunque no todas, porque le es posible, y nos informa de ello.

Errores al dibujar funciones

También pueden aparecer errores cuando se dibuja una función. Por ejemplo

```
Plot[Log[x], {x, -1, 10}];
Plot::plnr : Log[x] is not a machine-size real number at x = -1..
Plot::plnr : Log[x] is not a machine-size real number at x = -0.553763.
Plot::plnr : Log[x] is not a machine-size real number at x = -0.0671032.
General::stop :
 Further output of Plot::plnr will be suppressed during this calculation.
```



Mathematica nos indica que la función logaritmo no existe para algunos valores del intervalo en que queremos dibujarla, en este caso, los valores no positivos. Así y todo, la gráfica la dibuja donde puede hacerlo, aunque hay ocasiones que no dibuja nada.