Práctica 1:
Introducción a *Mathematica*

Grado en Administración y Dirección de Empresas

José Manuel Cascón Barbero (casbar@usal.es)
María Dolores García Sanz (dgarcia@usal.es)
Bernardo Ramón García-Bernalt Alonso (bgarcia@usal.es)
María Aurora Manrique García (amg@usal.es)
Gustavo Santos García (santos@usal.es)

1. Introducción

Mathematica es un lenguaje de cálculo simbólico de gran potencia desarrollado por Wolfram. En estas páginas mostraremos algunas características básicas y nos centraremos en las funcionalidades relacionadas con el material desarrollado en las asignaturas de Álgebra y Análisis Matemático del Grado en Administración y Dirección de Empresas de la Facultad de Economía y Empresa de la Universidad de Salamanca. Pretendemos implementar en seminarios la aplicación de este material, además de utilizarlo en las clases magistrales, en las que creemos será de gran ayuda para mantener la atención de nuestros estudiantes y conseguir que sean más interactivas.

Para comenzar es importante recordar:

1. *Mathematica* es un lenguaje de cálculo simbólico, lo que quiere decir que es capaz de trabajar con funciones y variables tal y como lo hacemos de forma habitual en matemáticas. No obstante, también se puede utilizar como un editor de texto científico (por ejemplo, este documento está escrito con *Mathematica*).
2. En *Mathematica* la información se agrupa en celdas. Existen diferentes tipos y formatos de celdas (ver menú *Format->Style*). Esta en concreto es una celda de texto. Por defecto las celdas son de tipo input, mostramos una a continuación:

```
In[1]:= 2 + 2
Out[1]= 4
```

3. Para evaluar una celda de tipo input se deben pulsar al mismo tiempo Shift+Return (o menú *Evaluation->Evaluate Cell*). Tras cada evaluación, *Mathematica* devuelve una celda de tipo output, donde aparece el resultado.

Mathematica también permite otros tipos de celda: Title, Section, Subsection, Text...
A veces es conveniente crear una celda de inicialización. Esto quiere decir que cada vez que se inicia una sesión y se evalúa una celda, *Mathematica* también evaluará esta celda.

4. *Mathematica* distingue entre mayúsculas y minúsculas.

5. Todas las funciones en *Mathematica* empiezan con letra mayúscula.
6. *Mathematica* emplea los operadores aritméticos clásicos. (+ Suma, - Resta, * Producto, / Cociente, ^Potencia). Un espacio en blanco entre dos variables es interpretado por *Mathematica* como un producto.
7. Se debe distinguir entre **=** y **==**. Un solo = representa asignación (cargar de contenido una variable), mientras que **==** es el operador de comparación (aparecerá en ecuaciones). Por ejemplo, la expresión

```
In[2]:= a = 3 + 4
Out[2]= 7
```

calcula la suma, y le asigna ese valor a la variable **a**. En cambio, el operador **==**, actúa como comparación. La expresión:

```
In[3]:= a + 4 == 11
Out[3]= True
```

Devuelve el valor *True* puesto que la ecuación es una igualdad (recuerda que el valor de **a** es 3).

8. En general, *Mathematica* incorpora cualquier función que podamos conocer. Habitualmente es el nombre inglés de la misma con la primera letra en mayúscula. Así, por ejemplo, el comando para **resolver** una ecuación algebraica es **Solve**.
9. Para solicitar ayuda sobre una función basta escribir en una celda input el signo ? seguido del nombre de la función. También se puede acceder al menú *Help* y navegar en *Documentation Center*. Por ejemplo, para tener información sobre la orden **Solve**:

```
In[4]:= ? Solve
```

Out[4]=

Symbol

Solve[*expr*, *vars*] attempts to solve the system *expr* of equations or inequalities for the variables *vars*.
Solve[*expr*, *vars*, *dom*] solves over the domain *dom*. Common choices of *dom* are Reals, Integers, and Complexes.

```
In[5]:= Solve[x^2 + 3 x - 8 == 0, x]
        resuelve
Out[5]= {{x → 1/2 (-3 - √41)}, {x → 1/2 (-3 + √41)}}
```

10. En matemáticas utilizamos indistintamente (), [], {} para indicar preferencia en el cálculo. En *Mathematica* cada delimitador tiene su significado y se debe respetar escrupulosamente. Así, **los paréntesis ( ) se utilizan para indicar preferencia en el cálculo, los corchetes [ ] se utilizan para indicar los argumentos de las funciones y, por último, las llaves { } se utilizan para delimitar listas**.

11. El tipo básico de archivo en *Mathematica* se denomina *Notebook* y tiene extensión 'nb'. *Mathematica* permite exportar el contenido generado a otro tipo de archivos: pdf, html, TEX, etc.

12. Las últimas versiones de *Mathematica* incorporan lo que se denomina modo lingüístico, que permite realizar cálculos sin conocer la sintaxis exacta o solicitar cualquier tipo de información al servidor *Wolfram|Alpha* https : // www.wolframalpha.com /. Para esto se requiere disponer de conexión a la red.

Por ejemplo, si queremos resolver la ecuación $x^2 + 3x - 8 = 0$, pero no conocemos la sintaxis en *Mathematica,* simplemente escribimos en una celda (en inglés) lo que precisamos, anteponiendo un signo igual (**=**) que *Mathematica* interpretará y coloreará en naranja.

In[6]:=
```
=Solve x^2 +3 x - 8 = 0
 resuelve

Solve[x^2 + 3*x - 8 == 0, x]
```

Out[6]=
$$\left\{\left\{x \to \frac{1}{2}\left(-3-\sqrt{41}\right)\right\}, \left\{x \to \frac{1}{2}\left(-3+\sqrt{41}\right)\right\}\right\}$$

Mathematica interpreta, resuelve y muestra una sintaxis correcta .

También podemos solicitar información más general, para lo cual comenzamos la celda con "dos símbolos igual": **==**

In[7]:= **Temperature Salamanca 2010-2020**

Assuming Salamanca (Spain) | Use Salamanca (Mexico) or Salamanca (Chile) or more instead

Input interpretation:

temperature	Salamanca, Spain
	2010 to 2020

Result:

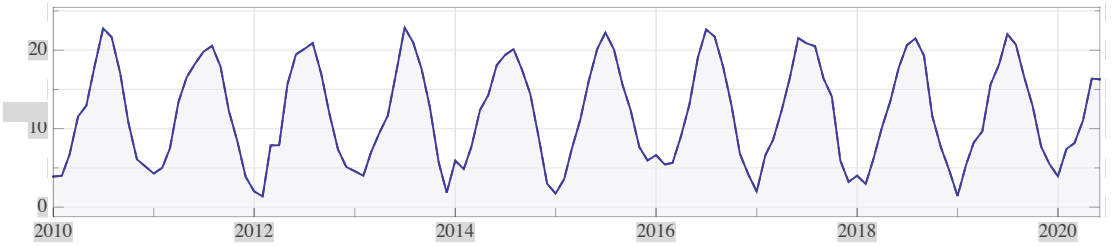
Show non-metric

(−11 to 41) °C (average low: 5 °C | average high: 20 °C)
(2010 to 2020)

History:

Show non-metric More

(monthly means)



low: −10 °C Jan 2017	average high: 27 °C average low: −1 °C	high: 41 °C Aug 2012

Weather station information:

Show non-metric More

name	LESA (Salamanca Airport)
relative position	14 km E (from center of Salamanca)
relative elevation	(comparable to center of Salamanca)

✚ Units

Satellite image »

Weather station comparisons:

Show non-metric More

	position	elevation	min	average	max
LESA	14 km E	795 m	−10 °C	12 °C	41 °C
LEVD	107 km NE	846 m	−10 °C	13 °C	38 °C
LPBG	131 km NW	692 m	−9 °C	13 °C	39 °C

(sorted by distance and inferred reliability)

✚ Units

WolframAlpha

In[8]:= **Microsoft Market capitalization 2020**

Input interpretation:

Microsoft	market capitalization	Wednesday, January 1, 2020 to Thursday, December 31, 2020

Result:

mean	$1.309 trillion (US dollars)
highest	$1.503 trillion (US dollars) (Wednesday, June 10, 2020)
lowest	$1.034 trillion (US dollars) (Monday, March 16, 2020)
volatility	56%
change	21%

History: Log scale

(from Jan 2, 2020 to Jun 19, 2020)

WolframAlpha

13. Además, el servidor *Wolfram|Alpha* https : // www.wolframalpha.com/ permite realizar cálculos on-line de forma inmediata sin necesidad de utilizar *Mathematica.*

Si las operaciones que se solicitan son demasiado complejas, los resultados presentados no son siempre correctos. El Póster A118- *El valor del método matemático en la docencia como garantía del uso del software* (Pestano-Gabino, C.; González-Concepción, C.; García-de la Rosa, Z; Gil.-Fariña, M.C,; Cruz-Báez, D.I:; Carrillo-Fernández,M; Sosa-Martín, D.) presentado en las **XXV Jornadas Asepuma | XXII Encuentro Internacional** que se celebró en A Coruña 2017 recogía varios de estos errores.

14. La mejor forma de aprender *Mathematica* es practicar y, sobre todo, equivocarse.

2. Variables

2.1 Definición

Mathematica reconoce como variable toda expresión que cumpla tres reglas básicas: no puede ser ni empezar por un número, no puede ser una variable reservada (o comando de *Mathematica*), no puede contener caracteres especiales (¿?, !¡, _, ., :,%,$)

In[9]:=
```
a = 2;
```

In[10]:=
```
a + 7
```

Out[10]=
```
9
```

En este caso reconoce **a** como variable porque no es un número, una palabra reservada o un comando.
Las variables permiten guardar en ellas cualquier valor o expresión y facilitan la labor del usuario. Se puede solicitar el contenido de una variable escribiéndola en una celda y evaluando la misma o utilizando un signo **?**

In[11]:=
```
a
```

Out[11]=
```
2
```

In[12]:=
```
? a
```

Out[12]=

Symbol

Notebook$$15$802415`a

Assignment a = 2

Full Name Notebook$$15$802415`a

En el primero de los casos *Mathematica* tan solo devuelve el valor, con el operador ? además informa del tipo de variable (Global).

Como ya avanzamos antes *Mathematica* es un lenguaje de cálculo simbólico. Entre otras cosas, permite trabajar con aritmética exacta:

In[13]:=
```
a = Sqrt[2]
    raíz cuadra
```

Out[13]=
$$\sqrt{2}$$

In[14]:=
```
a
```

Out[14]=
$$\sqrt{2}$$

Podemos observar que *Mathematica,* tal y como hacemos en matemáticas, entiende el concepto de raíz cuadrada. En ocasiones, podemos precisar de una aproximación numérica, para ello podemos usar la función **N[ ]**:

In[15]:=
```
N[a, 20]
  valor numérico
```

Out[15]=
```
1.4142135623730950488
```

Aquí el 20 denota el número de cifras decimales que se mostrarán.

2.2 Borrar el contenido de una variable

Para borrar el contenido de una variable se utiliza el comando **Clear**[ ]:

In[16]:=
```
a
```
Out[16]= $\sqrt{2}$

In[17]:=
```
Clear[a]
```
borra

In[18]:=
```
a
```
Out[18]= a

Después de borrar la variable, el sistema reconoce la existencia de una variable **a**, a la que no se ha asignado ningún valor. Otra opción es redefinir la variable:

In[19]:=
```
a = 5
```
Out[19]= 5

2.3 Variable %

La variable **%** contiene el valor del último output evaluado. En ocasiones puede resultar muy útil:

In[20]:=
```
%
```
Out[20]= 5

Es importante tener en cuenta que el orden de In (input) y Out (output) depende del orden de evaluación y que solo prevalece durante esa sesión. Si queremos referirnos a una celda de output específico, se añade el valor del número del output a la variable

In[21]:=
```
%17
```

La evaluación anterior hace referencia al output de la novena evaluación desde que se inicia la sesión.

2.4 Asignación estática y dinámica

Mathematica permite dos tipos de asignaciones. Hasta ahora solo hemos hecho uso de la asignación estática (**=**), la variable se inicia en el momento de la asignación, y prevalece mientras no haya una nueva definición. Por otro lado, la asignación retardada (**:=**) indica a *Mathematica* que debe reiniciar el contenido de la variable (volver a la definición) cada vez que aparezca en una celda de tipo In. El siguiente fragmento de código permite observar las diferencias entre ambos tipos de asignación:

In[22]:=
```
a = 2;
b = a + 2;
c := a + 2;
```

El valor de **a**,**b**,**c**, es obviamente:

In[25]:=
```
a
```
Out[25]=
```
2
```

In[26]:=
```
b
```
Out[26]=
```
4
```

In[27]:=
```
c
```
Out[27]=
```
4
```

Ahora modificamos el valor de **a**:

In[28]:=
```
a = 3;
```

Reevaluando,

In[29]:=
```
b
```
Out[29]=
```
4
```

In[30]:=
```
c
```
Out[30]=
```
5
```

Notad que **b** no se ha visto afectado por el cambio de valor de **a**, en cambio **c** sí. La asignación dinámica se suele usar de forma habitual con funciones o sucesiones.

3. Funciones

3.1 Definición y representación gráfica

Mathematica, como cualquier lenguaje de programación, permite definir funciones. Como se avanzó en la sección anterior las funciones se suelen definir utilizando asignación dinámica. La forma (prototipo) de hacerlo es:

Nombre[Argumento1_,Argumento2_,Argumento3_,...] := Definición

El **Nombre** y/o **Argumentos** deben cumplir las mismas reglas comentadas para el caso de las variables. Los argumentos deben estar entre corchetes [ ]. Después de cada argumento debe figurar el guión bajo _. El guión es crucial e indica a *Mathematica* que es la variable (o variables) de la ecuación que será sustituida en la definición de la derecha. Observad que antes de la definición

aparece el operador de asignación dinámica **:=**. Este operador indica a *Mathematica* que cada vez que se encuentre el nombre de la función debe recurrir a la definición.

Ejemplos:

- Funciones de una variable:

In[31]:=
```
f[x_] := x^2 + 3 x - 2
```

Podemos evaluar la función en un punto

In[32]:=
```
f[3]
```

Out[32]=
```
16
```

In[33]:=
```
f[Sqrt[2]]
```
raíz cuadrad

Out[33]=
$$3 \sqrt{2}$$

In[34]:=
```
f[α]
```

Out[34]=
$$-2 + 3\,\alpha + \alpha^2$$

Es interesante apuntar que la función **Plot** permite dibujar la función:

In[35]:=
```
? Plot
```

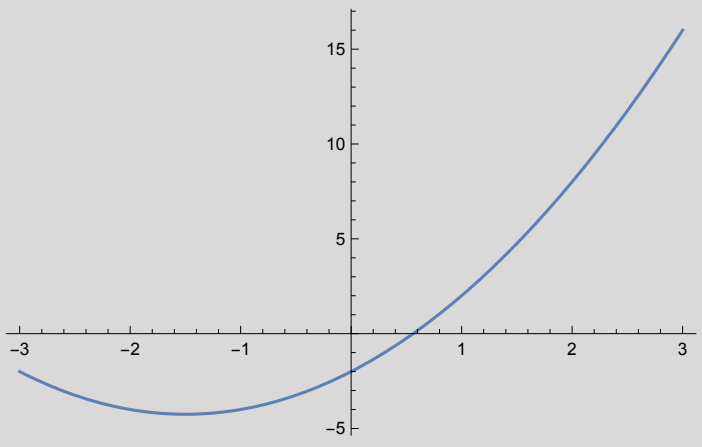
Out[35]=

Symbol

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}.
Plot[{f_1, f_2, …}, {x, x_{min}, x_{max}}] plots several functions f_i.
Plot[{…, w[f_i], …}, …] plots f_i with features defined by the symbolic wrapper w.
Plot[…, {x} ∈ reg] takes the variable x to be in the geometric region reg.

In[36]:=
```
Plot[f[x], {x, -3, 3}]
```
representación gráfica

Out[36]=



- Funciones de varias variables:

In[37]:=
```
g[x_, y_] := (x y) / (1 + x^2 + y^2)
```

In[38]:=
```
g[1, 2]
```

Out[38]=
$$\frac{1}{3}$$

In[39]:=
```
g[a, b]
```

Out[39]=
$$\frac{6}{13}$$

En este caso, la función **Plot3D**, permite representar la superficie:

In[40]:=
```
? Plot3D
```

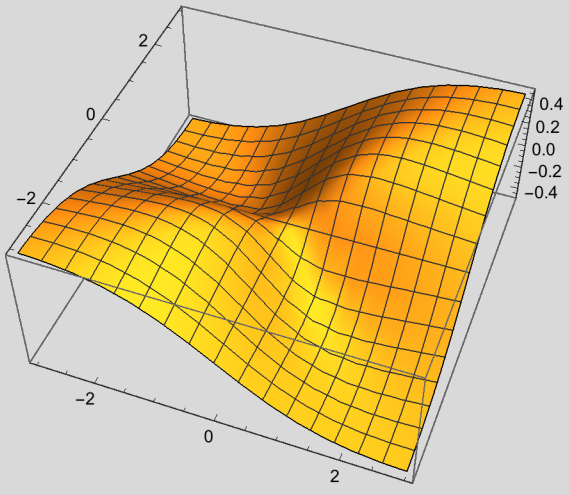
Out[40]=

Symbol

Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three–dimensional plot of f as a function of x and y.
Plot3D[{f_1, f_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions.
Plot3D[{…, w[f_i], …}, …] plots f_i with features defined by the symbolic wrapper w.
Plot3D[…, {x, y} ∈ reg] takes variables {x, y} to be in the geometric region reg.

In[41]:=
```
Plot3D[g[x, y], {x, -3, 3}, {y, -3, 3}]
```
representación gráfica 3D

Out[41]=



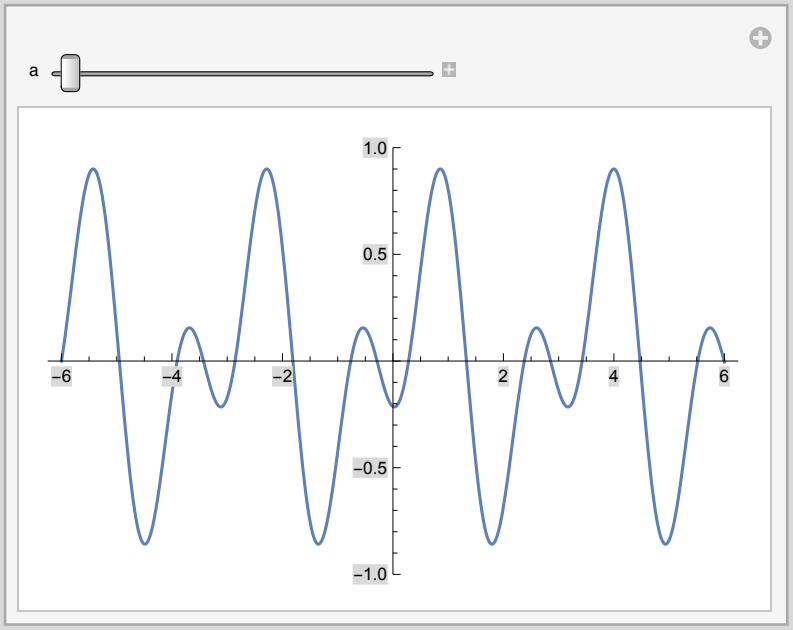
4. Animaciones en *Mathematica*

En *Mathematica* la función **Manipulate** genera animaciones en función de un parámetro. Por ejemplo, podemos observar el efecto de las traslaciones horizontales y verticales en la función

In[42]:=
```
f[x_] := Sin[3 x] Cos[x - 1]
         seno     coseno
```

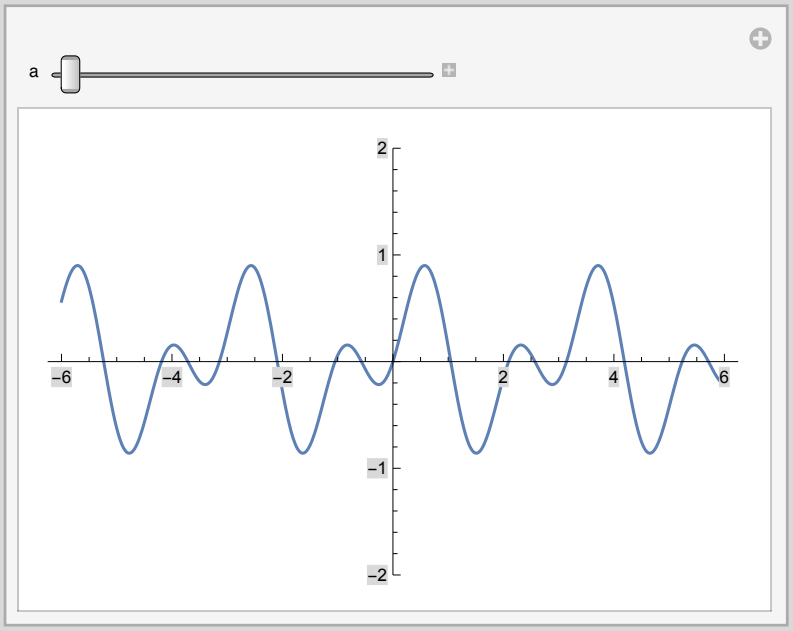
In[43]:=
```
Manipulate[Plot[f[x - a], {x, -6, 6}, PlotRange → {-1, 1}], {a, -6, 6}]
manipula   representación gráfica      rango de representación
```

Out[43]=



In[44]:=
```
Manipulate[
manipula
  Plot[a + f[x], {x, -6, 6}, AxesOrigin → {0, 0}, PlotRange → {-2, 2}], {a, 0, 4}]
  representación gráfica     origen de ejes          rango de representación
```

Out[44]=

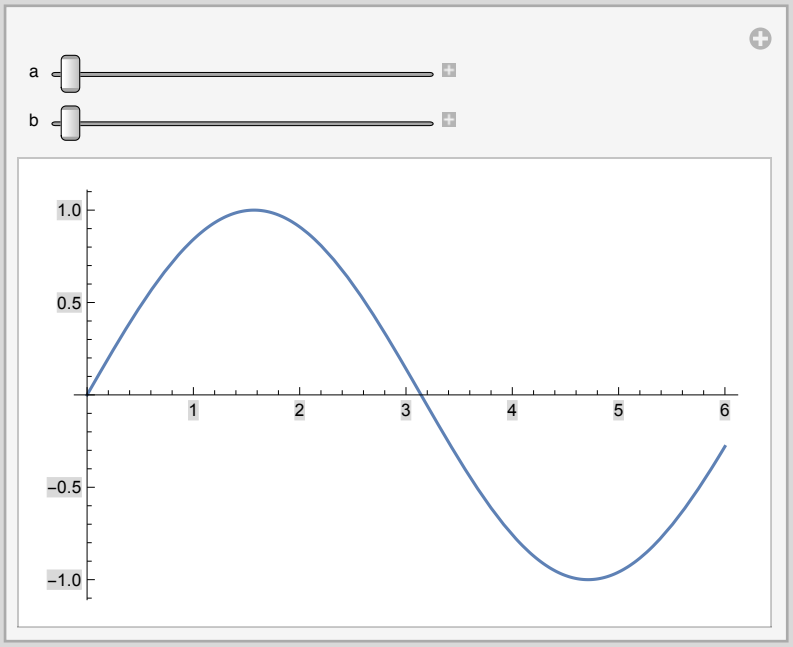


Es posible definir varios parámetros. Al evaluar la celda, *Mathematica* crea barras deslizantes para cada uno de ellos.

Al pulsar en aparece *Autoejecución*.

In[45]:=
```
Manipulate[Plot[Sin[a x + b], {x, 0, 6}], {a, 1, 4}, {b, 0, 10}]
```
manipula repr··· seno

Out[45]=



En los siguientes capítulos presentamos prácticas elaboradas para las asignaturas de Álgebra, siguiente capítulo, y Análisis Matemático, para esta última en tres apartados diferentes: Sucesiones y Series de números reales, Funciones de una variable real y Funciones de varias variables. Terminamos esta introducción con un glosario de los comandos utilizados en estas prácticas.

5. Glosario de comandos en Mathematica

Axix (ejes en una representación gráfica**)**

In[46]:=
```
? Axis
```

Out[46]=

Symbol

Axis is a symbol that represents the axis for purposes of alignment and positioning.

AxesOrigin (origen de ejes**)**

In[47]:=
```
? AxesOrigin
```

Out[47]=

Symbol

AxesOrigin is an option for graphics functions that specifies where any axes drawn should cross.

Clear (borrar**)**

In[48]:=
```
? Clear
```

Out[48]=

Symbol

Clear[$symbol_1$, $symbol_2$, …] clears values and definitions for the $symbol_i$.
Clear["$form_1$", "$form_2$", …] clears values and definitions
for all symbols whose names match any of the string patterns $form_i$.

CharacteristicPolynomial (polinomio característico de una matriz**)**

In[49]:=
```
? CharacteristicPolynomial
```

Out[49]=

Symbol

CharacteristicPolynomial[m, x] gives the characteristic polynomial for the matrix m.
CharacteristicPolynomial[{m, a}, x] gives the generalized characteristic polynomial with respect to a.

ContourPlot (gráfico de funciones en forma implícita**)**

In[50]:=
```
? ContourPlot
```

Out[50]=

Symbol

ContourPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a contour plot of f as a function of x and y.
ContourPlot[f == g, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots contour lines for which f = g.
ContourPlot[{f_1 == g_1, f_2 == g_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several contour lines.
ContourPlot[…, {x, y} ∈ reg] takes the variables {x, y} to be in the geometric region reg.

ContourPlot3D (representación 3D de superficies dadas en forma explícita**)**

In[51]:=
```
? ContourPlot3D
```

Out[51]=

Symbol

ContourPlot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}, {z, z_{min}, z_{max}}]
produces a three–dimensional contour plot of f as a function of x, y, and z.
ContourPlot3D[f == g, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}, {z, z_{min}, z_{max}}]
plots the contour surface for which f = g.

Cos (función coseno**)**

In[52]:=
```
? Cos
```

Out[52]=

Symbol

Cos[z] gives the cosine of z.

D (derivadas**)**

In[53]:=
```
? D
```

Out[53]=

Symbol

D[f, x] gives the partial derivative $\partial f/\partial x$.
D[f, {x, n}] gives the multiple derivative $\partial^n f/\partial x^n$.
D[f, x, y, …] gives the partial derivative $\cdots(\partial/\partial y)(\partial/\partial x)f$.
D[f, {x, n}, {y, m}, …] gives the multiple partial derivative $\cdots(\partial^m/\partial y^m)(\partial^n/\partial x^n)f$.
D[f, {{x_1, x_2, …}}] for a scalar f gives the vector derivative $(\partial f/\partial x_1, \partial f/\partial x_2, \ldots)$.
D[f, {$array$}] gives an array derivative.

Det (determinante de una matriz**)**

In[54]:=
```
? Det
```

Out[54]=

Symbol

Det[m] gives the determinant of the square matrix m.

Eigensystem (valores propios y vectores propios de una matriz**)**

In[55]:=
```
? Eigensystem
```

Out[55]=

Symbol

Eigensystem[m] gives a list {$values$, $vectors$}
of the eigenvalues and eigenvectors of the square matrix m.
Eigensystem[{m, a}] gives the generalized eigenvalues and eigenvectors of m with respect to a.
Eigensystem[m, k] gives the eigenvalues and eigenvectors for the first k eigenvalues of m.
Eigensystem[{m, a}, k] gives the first k generalized eigenvalues and eigenvectors.

Eigenvalues (valores propios de una matriz**)**

In[56]:=
```
? Eigenvalues
```

Out[56]=

Symbol

Eigenvalues[m] gives a list of the eigenvalues of the square matrix m.
Eigenvalues[{m, a}] gives the generalized eigenvalues of m with respect to a.
Eigenvalues[m, k] gives the first k eigenvalues of m.
Eigenvalues[{m, a}, k] gives the first k generalized eigenvalues.

Eigenvectors (vectores propios de una matriz**)**

In[57]:=
```
? Eigenvectors
```

Out[57]=

Symbol

Eigenvectors[m] gives a list of the eigenvectors of the square matrix m.
Eigenvectors[{m, a}] gives the generalized eigenvectors of m with respect to a.
Eigenvectors[m, k] gives the first k eigenvectors of m.
Eigenvectors[{m, a}, k] gives the first k generalized eigenvectors.

Erf (función error**)**

In[58]:=
```
? Erf
```

Out[58]=

Symbol

Erf[z] gives the error function erf(z).
Erf[z_0, z_1] gives the generalized error function erf(z_1) – erf(z_0).

Exp (función exponencial**)**

In[59]:=
```
? Exp
```

Out[59]=

Symbol

Exp[z] gives the exponential of z.

Expand (expande factores)

In[60]:=
```
? Expand
```

Out[60]=

Symbol

Expand[$expr$] expands out products and positive integer powers in $expr$.
Expand[$expr$, $patt$] leaves unexpanded any parts of $expr$ that are free of the pattern $patt$.

Factor (factoriza)

In[61]:=
```
? Factor
```

Out[61]=

Symbol

Factor[$poly$] factors a polynomial over the integers.
Factor[$poly$, Modulus → p] factors a polynomial modulo a prime p.
Factor[$poly$, Extension → {a_1, a_2, …}] factors a polynomial
allowing coefficients that are rational combinations of the algebraic numbers a_i.

Filling (relleno**)**

In[62]:=
```
? Filling
```

Out[62]=

Symbol

Filling is an option for ListPlot, Plot, Plot3D, and related
functions that specifies what filling to add under points, curves, and surfaces.

Gamma (función Gamma de Euler**)**

In[63]:=
```
? Gamma
```

Out[63]=

Symbol

Gamma[z] is the Euler gamma function Γ(z).
Gamma[a, z] is the incomplete gamma function Γ(a, z).
Gamma[a, z_0, z_1] is the generalized incomplete gamma function Γ(a, z_0) – Γ(a, z_1).

Grad (gradiente de una función**)**

In[64]:=
```
? Grad
```

Out[64]=

Symbol

Grad[f, {x_1, …, x_n}] gives the gradient $(\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$.
Grad[f, {x_1, …, x_n}, $chart$] gives the gradient in the coordinates $chart$.

Inverse (matriz inversa)

In[65]:=
```
? Inverse
```

Out[65]=

Symbol

Inverse[m] gives the inverse of a square matrix m.

Integrate (integral**)**

In[66]:=
```
? Integrate
```

Out[66]=

Symbol

Integrate[f, x] gives the indefinite integral $\int f\, dx$.
Integrate[f, {x, x_{min}, x_{max}}] gives the definite integral $\int_{x_{min}}^{x_{max}} f\, dx$.
Integrate[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}, …] gives the multiple integral $\int_{x_{min}}^{x_{max}} dx \int_{y_{min}}^{y_{max}} dy \ldots f$.
Integrate[f, {x, y, …} ∈ reg] integrates over the geometric region reg.

Labeled (etiquetas**)**

In[67]:=
```
? Labeled
```

Out[67]=

Symbol

Labeled[*expr*, *lbl*] displays *expr* labeled with *lbl*.
Labeled[*expr*, *lbl*, *pos*] places *lbl* at a position specified by *pos*.
Labeled[*expr*, {lbl_1, lbl_2, …}, {pos_1, …}] places the lbl_i at positions pos_i.
Labeled[*expr*, {lbl_1, lbl_2, lbl_3, lbl_4}, All] places the lbl_i at the bottom, left, top, and right, respectively.

Limit (límite**)**

In[68]:=
```
? Limit
```

Out[68]=

Symbol

Limit[$f[x], x \to x^*$] gives the limit $\lim_{x \to x^*} f(x)$.
Limit[$f[x_1, \ldots, x_n], \{x_1 \to x_1^*, \ldots, x_n \to x_n^*\}$] gives the nested limit $\lim_{x_1 \to x_1^*} \cdots \lim_{x_n \to x_n^*} f(x_1, \ldots, x_n)$.
Limit[$f[x_1, \ldots, x_n], \{x_1, \ldots, x_n\} \to \{x_1^*, \ldots, x_n^*\}$] gives the multivariate limit $\lim_{\{x_1, \ldots, x_n\} \to \{x_1^*, \ldots, x_n^*\}} f(x_1, \ldots, x_n)$.

LinearSolve (resuelve sistemas de ecuaciones lineales**)**

In[69]:=
```
? LinearSolve
```

Out[69]=

Symbol

LinearSolve[*m*, *b*] finds an *x* that solves the matrix equation *m*.*x* == *b*.
LinearSolve[*m*] generates a LinearSolveFunction[…] that can be applied repeatedly to different *b*.

ListPlot (representa una lista)

In[70]:=
```
? ListPlot
```

Out[70]=

Symbol

ListPlot[{y_1, y_2, …}] plots points {1, y_1}, {2, y_2}, ….
ListPlot[{{x_1, y_1}, {x_2, y_2}, …}] plots a list of points with specified *x* and *y* coordinates.
ListPlot[{$data_1$, $data_2$, …}] plots data from all the $data_i$.
ListPlot[{…, *w*[$data_i$, …], …}] plots $data_i$ with features defined by the symbolic wrapper *w*.

Log (función logarítmica**)**

In[71]:=
```
? Log
```

Out[71]=

Symbol

Log[z] gives the natural logarithm of z (logarithm to base e).
Log[b, z] gives the logarithm to base b.

Manipulate (animaciones**)**

In[72]:=
```
? Manipulate
```

Out[72]=

Symbol

Manipulate[$expr$, {u, u_{min}, u_{max}}] generates a version of $expr$
with controls added to allow interactive manipulation of the value of u.
Manipulate[$expr$, {u, u_{min}, u_{max}, du}] allows the value of u to vary between u_{min} and u_{max} in steps du.
Manipulate[$expr$, {{u, u_{init}}, u_{min}, u_{max}, …}] takes the initial value of u to be u_{init}.
Manipulate[$expr$, {{u, u_{init}, u_{lbl}}, …}] labels the controls for u with u_{lbl}.
Manipulate[$expr$, {u, {u_1, u_2, …}}] allows u to take on discrete values u_1, u_2, ….
Manipulate[$expr$, {u, …}, {v, …}, …] provides controls to manipulate each of the u, v, ….
Manipulate[$expr$, $c_u \to \{u, \ldots\}$, $c_v \to \{v, \ldots\}$, …]
links the controls to the specified controllers on an external device.

MatrixForm (visualiza una matriz**)**

In[73]:=
```
? MatrixForm
```

Out[73]=

Symbol

MatrixForm[$list$] prints with the elements of $list$ arranged in a regular array.

MatrixRank (rango de una matriz**)**

In[74]:=
```
? MatrixRank
```

Out[74]=

Symbol

MatrixRank[m] gives the rank of the matrix m.

Maximize (maximiza**)**

In[75]:=
```
? Maximize
```

Out[75]=

Symbol

Maximize[f, x] maximizes f with respect to x.
Maximize[f, {x, y, …}] maximizes f with respect to x, y, ….
Maximize[{f, $cons$}, {x, y, …}] maximizes f subject to the constraints $cons$.
Maximize[…, $x \in reg$] constrains x to be in the region reg.
Maximize[…, …, dom] constrains variables to the domain dom, typically Reals or Integers.

Minimize (minimiza**)**

In[76]:=
```
? Minimize
```

Out[76]=

Symbol

Minimize[f, x] minimizes f with respect to x.
Minimize[f, {x, y, …}] minimizes f with respect to x, y, ….
Minimize[{f, $cons$}, {x, y, …}] minimizes f subject to the constraints $cons$.
Minimize[…, $x \in reg$] constrains x to be in the region reg.
Minimize[…, …, dom] constrains variables to the domain dom, typically Reals or Integers.

N (evalúa numéricamente**)**

In[77]:=
```
? N
```

Out[77]=

Symbol

N[$expr$] gives the numerical value of $expr$.
N[$expr$, n] attempts to give a result with n-digit precision.

NIntegrate (integra numéricamente**)**

In[78]:=
```
? NIntegrate
```

Out[78]=

Symbol

NIntegrate[f, {x, x_{min}, x_{max}}] gives a numerical approximation to the integral $\int_{x_{min}}^{x_{max}} f\, dx$.
NIntegrate[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}, …] gives
a numerical approximation to the multiple integral $\int_{x_{min}}^{x_{max}} dx \int_{y_{min}}^{y_{max}} dy \ldots f$.
NIntegrate[f, {x, y, …} $\in reg$] integrates over the geometric region reg.

NMaximize (maximiza numéricamente**)**

In[79]:=
```
? NMaximize
```

Out[79]=

Symbol

NMaximize[f, x] maximizes f numerically with respect to x.
NMaximize[f, {x, y, …}] maximizes f numerically with respect to x, y, ….
NMaximize[{f, $cons$}, {x, y, …}] maximizes f numerically subject to the constraints $cons$.
NMaximize[…, $x \in reg$] constrains x to be in the region reg.

NMinimize (minimiza numéricamente**)**

In[80]:=
```
? NMinimize
```

Out[80]=

Symbol

NMinimize[f, x] minimizes f numerically with respect to x.
NMinimize[f, {x, y, …}] minimizes f numerically with respect to x, y, ….
NMinimize[{f, $cons$}, {x, y, …}] minimizes f numerically subject to the constraints $cons$.
NMinimize[…, $x \in reg$] constrains x to be in the region reg.

Normal (expresión normal**)**

In[81]:=
```
? Normal
```

Out[81]=

Symbol

Normal[$expr$] converts $expr$ to a normal expression from a variety of special forms.
Normal[$expr$, h] converts objects with head h in $expr$ to normal expressions.
Normal[$expr$, {h_1, h_2, …}] converts objects with head h_i to normal expressions.

NSolve (resuelve numéricamente**)**

In[82]:=
```
? NSolve
```

Out[82]=

Symbol

NSolve[$expr$, $vars$] attempts to find numerical approximations to the
solutions of the system $expr$ of equations or inequalities for the variables $vars$.
NSolve[$expr$, $vars$, Reals] finds solutions over the domain of real numbers.

NSum (aproximación numérica de suma)

In[83]:=
```
? NSum
```

Out[83]=

Symbol

NSum[f, {i, i_{min}, i_{max}}] gives a numerical approximation to the sum $\sum_{i=i_{min}}^{i_{max}}$ f.
NSum[f, {i, i_{min}, i_{max}, di}] uses a step di in the sum.

NullSpace (calcula una base del sistema de ecuaciones lineales homogéneo**)**

In[84]:=
```
? NullSpace
```

Out[84]=

Symbol

NullSpace[m] gives a list of vectors that forms a basis for the null space of the matrix m.

ParametricPlot (gráfico de funciones en forma paramétrica)

In[85]:=
```
? ParametricPlot
```

Out[85]=

Symbol

ParametricPlot[{f_x, f_y}, {u, u_{min}, u_{max}}] generates a parametric
plot of a curve with x and y coordinates f_x and f_y as a function of u.
ParametricPlot[{{f_x, f_y}, {g_x, g_y}, …}, {u, u_{min}, u_{max}}] plots several parametric curves.
ParametricPlot[{f_x, f_y}, {u, u_{min}, u_{max}}, {v, v_{min}, v_{max}}] plots a parametric region.
ParametricPlot[{{f_x, f_y}, {g_x, g_y}, …}, {u, u_{min}, u_{max}}, {v, v_{min}, v_{max}}] plots several parametric regions.
ParametricPlot[{…, w[{f_x, f_y}], …}, …] plots
the curve {f_x, f_y} with features defined by the symbolic wrapper w.
ParametricPlot[…, {u, v} ∈ reg] takes parameters {u, v} to be in the geometric region reg.

Plot (representación gráfica**)**

In[86]:=
```
? Plot
```

Out[86]=

Symbol

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}.
Plot[{f_1, f_2, …}, {x, x_{min}, x_{max}}] plots several functions f_i.
Plot[{…, w[f_i], …}, …] plots f_i with features defined by the symbolic wrapper w.
Plot[…, {x} ∈ reg] takes the variable x to be in the geometric region reg.

PlotLegends (leyendas en una representación gráfica**)**

In[87]:=
```
? PlotLegends
```

Out[87]=

Symbol

PlotLegends is an option for plot functions that specifies what legends to use.

PlotRange (rango de representación**)**

In[88]:=
```
? PlotRange
```

Out[88]=

Symbol

PlotRange is an option for graphics functions
that specifies what range of coordinates to include in a plot.

Plot3D (representación gráfica 3D**)**

In[89]:=
```
? Plot3D
```

Out[89]=

Symbol

Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three–dimensional plot of f as a function of x and y.
Plot3D[{f_1, f_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions.
Plot3D[{…, w[f_i], …}, …] plots f_i with features defined by the symbolic wrapper w.
Plot3D[…, {x, y} ∈ reg] takes variables {x, y} to be in the geometric region reg.

Product (producto**)**

In[90]:=
```
? Product
```

Out[90]=

Symbol

Product[f, {i, i_{max}}] evaluates the product $\prod_{i=1}^{i_{max}} f$.
Product[f, {i, i_{min}, i_{max}}] starts with i = i_{min}.
Product[f, {i, i_{min}, i_{max}, di}] uses steps di.
Product[f, {i, {i_1, i_2, …}}] uses successive values i_1, i_2, ….
Product[f, {i, i_{min}, i_{max}}, {j, j_{min}, j_{max}}, …] evaluates the multiple product $\prod_{i=i_{min}}^{i_{max}} \prod_{j=j_{min}}^{j_{max}} \ldots f$.
Product[f, i] gives the indefinite product $\prod_i f$.

Reals (conjunto de números reales**)**

In[91]:=
```
? Reals
```

Out[91]=

Symbol

Reals represents the domain of real numbers, as in $x \in$ Reals.

Reduce (reduce ecuaciones)

In[92]:=
```
? Reduce
```

Out[92]=

Symbol

Reduce[*expr*, *vars*] reduces the statement *expr* by
solving equations or inequalities for *vars* and eliminating quantifiers.
Reduce[*expr*, *vars*, *dom*] does the reduction over the domain *dom*.
Common choices of *dom* are Reals, Integers, and Complexes.

Series (serie)

In[93]:=
```
? Series
```

Out[93]=

Symbol

Series[f, {x, x_0, n}] generates a power series expansion for f about the point $x = x_0$ to order $(x - x_0)^n$.
Series[f, {x, x_0, n_x}, {y, y_0, n_y}, …] successively finds series expansions with respect to x, then y, etc.

Simplify (simplifica)

In[94]:=
```
? Simplify
```

Out[94]=

Symbol

Simplify[*expr*] performs a sequence of algebraic and
other transformations on *expr* and returns the simplest form it finds.
Simplify[*expr*, *assum*] does simplification using assumptions.

Sin (función seno)

In[95]:=
```
? Sin
```

Out[95]=

Symbol

Sin[z] gives the sine of z.

Solve (resuelve)

In[96]:=
```
? Solve
```

Out[96]=

Symbol

Solve[$expr$, $vars$] attempts to solve the system $expr$ of equations or inequalities for the variables $vars$.

Solve[$expr$, $vars$, dom] solves over the domain dom. Common choices of dom are Reals, Integers, and Complexes.

Sqrt (raíz cuadrada)

In[97]:=
```
? Sqrt
```

Out[97]=

Symbol

Sqrt[z] or $\sqrt{z}$ gives the square root of z.

Sum (suma)

In[98]:=
```
? Sum
```

Out[98]=

Symbol

Sum[f, {i, i_{max}}] evaluates the sum $\sum_{i=1}^{i_{max}} f$.

Sum[f, {i, i_{min}, i_{max}}] starts with $i = i_{min}$.

Sum[f, {i, i_{min}, i_{max}, di}] uses steps di.

Sum[f, {i, {i_1, i_2, …}}] uses successive values i_1, i_2, ….

Sum[f, {i, i_{min}, i_{max}}, {j, j_{min}, j_{max}}, …] evaluates the multiple sum $\sum_{i=i_{min}}^{i_{max}} \sum_{j=j_{min}}^{j_{max}} \ldots f$.

Sum[f, i] gives the indefinite sum $\sum_i f$.

Table (genera una lista)

In[99]:=
```
? Table
```

Out[99]=

Symbol

Table[*expr*, *n*] generates a list of *n* copies of *expr*.
Table[*expr*, {*i*, i_{max}}] generates a list of the values of *expr* when *i* runs from 1 to i_{max}.
Table[*expr*, {*i*, i_{min}, i_{max}}] starts with $i = i_{min}$.
Table[*expr*, {*i*, i_{min}, i_{max}, *di*}] uses steps *di*.
Table[*expr*, {*i*, {i_1, i_2, …}}] uses the successive values i_1, i_2, ….
Table[*expr*, {*i*, i_{min}, i_{max}}, {*j*, j_{min}, j_{max}}, …] gives a nested list. The list associated with *i* is outermost.

Tr (Traza)

In[100]:=
```
? Tr
```

Out[100]=

Symbol

Tr[*list*] finds the trace of the matrix or tensor *list*.
Tr[*list*, *f*] finds a generalized trace, combining terms with *f* instead of Plus.
Tr[*list*, *f*, *n*] goes down to level *n* in *list*.

Transpose (traspuesta de una matriz**)**

In[101]:=
```
? Transpose
```

Out[101]=

Symbol

Transpose[*list*] transposes the first two levels in *list*.
Transpose[*list*, {n_1, n_2, …}] transposes *list* so that the k^{th} level in *list* is the $n_k{}^{th}$ level in the result.
Transpose[*list*, $m \leftrightarrow n$] transposes levels *m* and *n* in *list*, leaving all other levels unchanged.

Zeta (función Zeta)

In[102]:=
```
? Zeta
```

Out[102]=

Symbol

Zeta[*s*] gives the Riemann zeta function ζ(s).
Zeta[*s*, *a*] gives the generalized Riemann zeta function ζ(s, a).