

**Prácticas de Análisis Matemático**  
**con *Mathematica* 3.0** ©

**E.T.S. Ingeniería Informática**

---

**Departamento de Análisis Matemático**

---

## Índice general

---

<b>1. Primeros pasos</b>	<b>4</b>
1.1. Cálculo simbólico con <i>Mathematica</i>	10
1.1.1. Polinomios	12
1.1.2. Expresiones trigonométricas	15
<b>2. Cómo “dibujar” con <i>Mathematica</i></b>	<b>17</b>
2.1. La orden Plot	17
2.1.1. Opciones de Plot	19
2.2. Gráficos en coordenadas paramétricas	20
2.2.1. Algunas curvas planas	21
2.3. El comando Show	22
2.4. Ejercicios	22
<b>3. Vectores y matrices</b>	<b>24</b>
<b>4. Resolución de ecuaciones</b>	<b>30</b>
4.1. Ecuaciones “sencillas”	30
4.2. Sistemas de ecuaciones lineales	32
4.3. Resolución de sistemas de ecuaciones	34
4.4. Otros métodos	35
4.4.1. Breves conceptos de programación	36
4.4.2. Método de Bisección	38
4.4.3. Método de Newton-Raphson	40
4.5. El comando FindRoot	42
<b>5. Extremos de funciones de una variable</b>	<b>44</b>
5.1. Continuidad y límites	44
5.2. Máximos y mínimos relativos	45
<b>6. Fórmula de Taylor</b>	<b>47</b>

<b>7. Integración</b>	<b>52</b>
7.1. Integrales definidas e indefinidas	52
7.2. Longitudes, áreas y volúmenes	54
7.3. Integrales impropias.	55
<b>8. Gráficos en 3D</b>	<b>57</b>
8.1. El comando Plot3D	57
8.1.1. Opciones del comando Plot3D	58
8.2. Gráficos de contorno. Curvas de nivel.	59
8.3. Gráficos paramétricos. Curvas y superficies.	60
8.3.1. Superficies de revolución	61
<b>9. Extremos relativos y condicionados</b>	<b>65</b>
9.1. Derivadas parciales	65
9.2. Representación gráfica	66
9.2.1. “Campos” de vectores	69
9.3. Extremos relativos.	71
9.4. Extremos condicionados.	75
<b>10.Integrales múltiples</b>	<b>81</b>
<b>11.Números complejos</b>	<b>87</b>
11.1.Operaciones básicas	87
11.2.Funciones de variable compleja	89
11.2.1.Gráficos en “cuatro” dimensiones	90
<b>A. Avisos y mensajes de error</b>	<b>91</b>
<b>B. Glosario</b>	<b>94</b>
B.1. Algunas funciones	94
B.2. Comandos usuales	94
B.3. Números complejos	94
B.4. Matrices	95
B.5. Resolución de ecuaciones	95
B.6. Gráficos	95
B.7. Cálculo	95
B.8. Otros	95

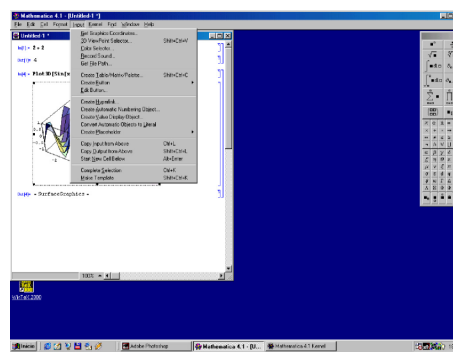
# CAPÍTULO 1




## Primeros pasos

Comenzaremos a familiarizarnos con el *Mathematica* haciendo, al principio, cosas muy simples que nos ayudarán a conocer los principales comandos y cómo se deben usar.

Para efectuar cálculos numéricos, el programa *Mathematica* funciona como una simple calculadora, con la salvedad de que con el *Mathematica* puedes obtener una precisión que no tienen las calculadoras normales.

Una vez que hemos empezado el programa nos encontramos delante de una pantalla más o menos como la de la figura. Arriba tienes la barra de Menú, debajo tienes una ventana vacía que es donde trabajaremos, y a la derecha tenemos una “paleta” con operaciones que ya iremos comentando con más detalle.





Una vez iniciada la ejecución del programa, pulsa con el botón izquierdo del ratón en la ventana principal y escribe:  $3+2$ , y luego pulsa   o  en el teclado numérico. Observarás que en la pantalla aparece lo siguiente:

```
In[1]:=
  3+2
Out[1]=
  5
```

No intentes escribir los símbolos “In[1]:=” y “Out[1]=”, ya que éstos los escribe el programa para llevar un control sobre las operaciones que va efectuando. No importa si dicho dígito de control no coincide con el que aparezca en este texto.

A continuación, con el ratón puedes pulsar sobre los números que habías escrito, y cambiarlos, o añadir nuevos sumandos, etc... y recuerda que siempre que quieras obtener el resultado, deberás

pulsar  . También podrás escribir los siguientes comandos debajo del resultado anterior, con lo que quedará constancia escrita de las operaciones que vas haciendo. Observarás que en las sucesivas operaciones el ordenador responde más rápido que en la primera, ya que al efectuar la primera operación, el programa debió leer del disco las “reglas de cálculo” (esto es, el *kernel* del programa), y dichas reglas permanecen en su memoria a partir de ese momento.

Para multiplicar números no es necesario escribir el símbolo de la multiplicación (opcionalmente “\*”), y basta con poner un espacio entre los factores.



```
In[2]:=
  3 2
Out[2]=
  6
```

Para efectuar potencias, puedes escribir

```
In[3]:=
  3^85
Out[3]=
  35917545547686059365808220080151141317043
```

Ya que lo sabemos hacer directamente, comentemos que la paleta se puede usar, entre otras cosas, para escribir potencias. Si pulsas en el primer botón de la paleta te aparecerá en la ventana de comandos lo siguiente:



Puedes teclear directamente la base y, cuando termines, el tabulador te lleva al exponente. Una vez escrito pulsa como siempre   y obtendrás el resultado, algo así:

```
In[4]:=
  2^3
Out[4]=
  8
```



Puedes hacer operaciones con fracciones, y obtener la fracción resultante...

```
In[5]:=
  2+3/13
Out[5]=
  29
  13
```

(¿Por qué el resultado no ha sido  $\frac{5}{13}$ ?) Si lo prefieres, puedes aproximar el resultado mediante unos cuantos dígitos de su expresión decimal:

```
In[6]:=
  N[2+3/13]
Out[6]=
  2.23077
```

... y si quieres que el *Mathematica* calcule 40 dígitos (incluyendo la parte entera)...

```
In[7]:=
  N[2+3/13,40]
Out[7]=
  2.2307692307692307692307692307692308
```

¿por qué el último dígito del comando anterior fue un 7 y no un 6? y ¿por qué ahora el último dígito ha sido un 8 y no un 7?

Si quieres, puedes poner dos mil decimales. Haz la prueba.

Sigamos probando... para obtener la raíz de un número se usa el comando `Sqrt` o utilizar la paleta (observa que la "S" es mayúscula, y el número debe ir entre corchetes):

```
In[8]:=
  Sqrt[5]
Out[8]=
   $\sqrt{5}$ 
```

... pues vale... y encima es hasta verdad... pero si quieres la expresión decimal con quince dígitos,

```
In[9]:=
  N[Sqrt[5],15]
Out[9]=
  2.23606797749979
```

También puedes hacer la raíz cuadrada de un número, elevando dicho número al exponente  $\frac{1}{2}$

```
In[10]:=
  5^(1/2)
Out[10]=
   $\sqrt{5}$ 
```

¿Podrías ahora obtener una aproximación decimal de  $\sqrt[3]{86}$  con doscientas cifras decimales?.  
Inténtalo.

Además de saber calcular las raíces, *Mathematica* también conoce las reglas de cálculo para operar con ellas:

```
In[11]:=
  Sqrt[2] Sqrt[3]
Out[11]=
   $\sqrt{6}$ 
```

...o bien...

```
In[12]:=
  Sqrt[5^(1/3)]
Out[12]=
   $5^{1/6}$ 
```

Como ya habrás notado, en la paleta tienes botones que permiten escribir fracciones, raíces de cualquier orden y algunas de las constantes más usuales como el número  $e$  o  $\pi$ .




Se supone que conocemos aproximadamente el valor del número  $\pi$ , y también *Mathematica* lo conoce...

```
In[13]:=
  Pi
Out[13]=
   $\pi$ 
```

...ya... parece más interesante así:

```
In[14]:=
  N[Pi]
Out[14]=
  3.14159
```

Es probable que alguno de nosotros conozca más dígitos, pero *Mathematica* es capaz de decirnos los primeros cinco mil dígitos de  $\pi$ . Prueba.

Ahora vamos a trabajar con variables. Prueba el siguiente comando (ten en cuenta que para cambiar de línea dentro de un comando es necesario pulsar sólo , y finalmente   para ejecutar todo el comando)

```
In[15]:=
  a=20;
  b=4;
  a/b
Out[15]=
  5
```

Observa que las líneas de comando que terminan en “punto y coma” no producen ninguna respuesta del *Mathematica*. Prueba a quitar (por ejemplo) el punto y coma de la primera línea en el ejemplo anterior.

Razona la respuesta que da el *Mathematica* al siguiente comando:

```
In[16]:=
  a=5;
  b=12;
  N[Sqrt[a+b], b]
Out[16]=
  4.12310562562
```

Con *Mathematica* podemos usar el resultado de una operación anterior sin necesidad de teclearlo. Esto se consigue con la orden `%`. Si queremos el resultado de la salida  $n$  (Out[n]), podemos obtenerlo con `%n`. Por ejemplo, si queremos el resultado de la operación número 15,

```
In[17]:=
  %15
Out[17]=
  5
```

además podemos usar esa información como cualquier otro dato

```
In[18]:=
  %15^2
Out[18]=
  25
```

Para obtener el resultado inmediatamente anterior usamos `%` sin ningún número,

```
In[19]:=
  %
Out[19]=
  25
```



%% para el penúltimo, etc...

Además de las operaciones elementales que hemos visto, *Mathematica* tiene definidas la mayor parte de las funciones elementales. Los nombres de estas funciones suelen ser su abreviatura en inglés, que algunas veces difiere bastante de su nombre castellano. En general, cualquier comando de *Mathematica* se escribe con la primera letra en mayúscula. Por ejemplo

```
In[20]:=
  Sqrt[4]
Out[20]=
  2
```

Algunos ejemplos de funciones con *Mathematica* son:

- *Función exponencial*: Exp[x]

```
In[21]:=
  Exp[2]
Out[21]=
  E2
```

(para *Mathematica* el número *e* se escribe E) y si queremos su expresión decimal

```
In[22]:=
  N[Exp[2]]
Out[22]=
  7.38906
```

Otra forma de calcular la función exponencial aplicada a  $x$  es elevando E a  $x$ . ¿Podrías calcular así  $e^5$ ? ¿y su primera cifra decimal? Haz lo mismo usando la función Exp y comprueba que da el mismo resultado.

- *Función logaritmo neperiano*: Log[x]

```
In[23]:=
  Log[20]
Out[23]=
  Log[20]
```

...ya empezamos... Bueno, si lo que nos interesa es su expresión decimal

```
In[24]:=
  N[Log[20]]
Out[24]=
  2.99573
```

En *Mathematica*, si no decimos lo contrario, los logaritmos serán neperianos. Si queremos calcular el logaritmo en base  $b$  de  $x$  usamos  $\text{Log}[b, x]$ . Por ejemplo, el logaritmo decimal de 100

```
In[25]:=
  Log[10,100]
Out[25]=
  2
```

¿Cuánto valdrá el logaritmo en base 2 de 64? Compruébalo. ¿Cuánto debe valer  $\text{Log}[\text{Exp}[7]]$ ? ¿Y  $\text{Exp}[\text{Log}[7]]$ ? Pide a *Mathematica* que los calcule.

- *Funciones trigonométricas:*  $\text{Sin}[x]$ ,  $\text{Cos}[x]$ ,  $\text{Tan}[x]$  (seno, coseno, tangente)

```
In[26]:=
  Sin[Pi/4]
Out[26]=
   $\frac{1}{\sqrt{2}}$ 
```

También podemos usar las funciones trigonométricas inversas, esto es, el arcoseno, arccoseno y arcotangente, que se escriben respectivamente  $\text{ArcSin}[x]$ ,  $\text{ArcCos}[x]$  y  $\text{ArcTan}[x]$ . Observa que la cuarta letra de cada comando está en mayúscula; escríbelo así, si no, *Mathematica* no lo entenderá.

```
In[27]:=
  ArcTan[1]
Out[27]=
   $\frac{\pi}{4}$ 
```

Prueba a componer dos o más de todas estas funciones y a hacer cálculos con ellas.

## 1.1. Cálculo simbólico con *Mathematica*

Hasta ahora sólo hemos usado el *Mathematica* como una calculadora muy potente, pero prácticamente todo lo que hemos aprendido puede hacerse sin dificultad con una calculadora convencional. Entonces, ¿qué puede hacer *Mathematica* que sea imposible con una calculadora? Bueno, entre otras muchas cosas que veremos posteriormente, la principal utilidad de *Mathematica* es el cálculo simbólico, es decir, el trabajar con expresiones algebraicas (expresiones donde intervienen variables, constantes... y no tienen porqué tener un valor numérico concreto) en vez de con números. Por ejemplo, el programa sabe que la función  $\text{Log}$  es inversa de  $\text{Exp}$ , con lo que si ponemos

```
In[1]:=
  Exp[Log[x]]
Out[1]=
  x
```

es decir, sin saber el valor de la variable  $x$  el programa es capaz de trabajar simbólicamente con ella. Más ejemplos

```
In[2]:=
  Exp[x]Exp[y]
Out[2]=
  Ex+y

In[3]:=
  a+2a+5b
Out[3]=
  3 a + 5 b
```

Para *Mathematica* cualquier letra o combinación de letras puede ser una variable o una constante. Hay excepciones como la letra E, que siempre indica el número e y no puede usarse como variable ni constante. Sin embargo, las letras  $x, y, z, \dots a, b, c, d, \dots$  pueden usarse sin problemas como nombres de constantes y variables. ¿Cómo podemos hacer que *Mathematica* diferencie entre constantes (valor fijo) y variables (valor indeterminado)? Para el programa, si no se le ha asignado un valor a una letra, ésta será una variable. Si le hemos asignado un valor a una letra, ésta será una constante, es decir, tendrá siempre un valor concreto. ¿Podemos quitar el valor a una constante? Sí, con la orden `Clear[nombre variable]`. Un ejemplo:

```
In[4]:=
  a=7
Out[4]=
  7


In[5]:=
  a2
Out[5]=
  49
```

es decir, le damos a la letra  $a$  el valor 7 y *Mathematica* la sustituye siempre por ese valor; en este caso,  $a$  es una constante. Si usamos `Clear`

```
In[6]:=
Clear[a]
```

(observa que esta entrada no produce ninguna salida)

```
In[7]:=
a2
Out[7]=
a2
```

ahora,  $a$  es una variable, esto es, no tiene un valor concreto y *Mathematica* debe tratarla simbólicamente. Si queremos que todas las constantes que hayamos definido pierdan su valor concreto (que pasen a ser variables) usaremos `Clear["Global`*"]` (el acento que hay que usar es el que está a la derecha de la ).

Vamos a practicar con comandos de *Mathematica* para manejar expresiones algebraicas: polinomios, funciones racionales, expresiones trigonométricas, ecuaciones...

### 1.1.1. Polinomios

Si introducimos el siguiente polinomio

```
In[8]:=
x2 + 2 x + 1
Out[8]=
1 + 2 x + x2
```

*Mathematica* no intenta simplificarlo. Si escribimos

```
In[9]:=
(x+1)2
Out[9]=
(1+x)2
```

*Mathematica* no desarrolla el cuadrado. Probemos ahora a restar las dos expresiones:

```
In[10]:=
%% - %
Out[10]=
1 + 2 x + x2 - (1 + x)2
```

*Mathematica* no se da cuenta de que la expresión vale cero. Esto es porque no factoriza ni desarrolla automáticamente, sino que debemos decirle que lo haga. ¿Cómo lo hacemos? Con las órdenes `Expand[expresión]` (desarrollar), `Simplify[expresión]` (simplificar) y `Factor[expresión]` (factorizar):

```
In[11]:=
  Expand[(x-2)(x-1)x(x+1)(x+2)]
Out[11]=
  4 x - 5 x3 + x5
```

también con varias variables

```
In[12]:=
  Expand[(2y-x)(-3x+y)(x+y)]
Out[12]=
  3 x3 - 4 x2 y - 5 x y2 + 2 y3

In[13]:=
  Factor[%]
Out[13]=
  (x - 2 y) (3 x - y) (x + y)
```

```
In[14]:=
  Expand[1+2x+x2 - (1+x)2]
Out[14]=
  0

In[15]:=
  Factor[1+2x+x2 - (1+x)2]
Out[15]=
  0
```

La orden `Simplify` sirve para simplificar una expresión, factorizando o no, según convenga:

```

In[16]:=
  Simplify[1+x3]
Out[16]=
  1 + x3

In[17]:=
  Simplify[x2+2x+1+(1+x)2]
Out[17]=
  2(1 + x)2

```

Hay veces que la expresión es “demasiado” complicada y `Simplify` no da ningún resultado práctico. En este caso se puede intentar usar la orden `FullSimplify`, pero hay que tener en cuenta que el tiempo para realizar los cálculos puede aumentar mucho.

```

In[18]:=
  Simplify[3√5√13 - 18]
Out[18]=
  (-18 + 5 √13)(1/3)

In[19]:=
  FullSimplify[3√5√13 - 18]
Out[19]=
   $\frac{1}{2}(-3 + \sqrt{13})$ 

```

El comando `Short` sirve para ver mejor polinomios muy grandes, mostrando sólo algunos de los términos de mayor y menor grado

```

In[20]:=
  Expand[(x+1)5]
Out[20]=
  1 + 5 x + 10 x2 + 10 x3 + 5 x4 + x5

In[21]:=
  Short[%]
Out[21]=
  1 + 5 x + «3» + x5

```

### 1.1.2. Expresiones trigonométricas

*Mathematica* conoce las identidades trigonométricas y puede usarlas para simplificar expresiones en las que aparezcan dichas funciones. En lugar de `Expand` y `Factor`, utilizaremos los órdenes `TrigExpand` y `TrigFactor`. Por ejemplo,

```
In[22]:=
  TrigExpand[Cos[α + β]]
Out[22]=
  Cos[α] Cos[β]-Sin[α] Sin[β]

In[23]:=
  TrigExpand[Sin[2ArcTan[t]]]
Out[23]=
   $\frac{2t}{1+t^2}$ 

In[24]:=
  TrigExpand[Sin[x+3*y]+Cos[2* z]
  Sin[x-y]]
Out[24]=
  Cos[y]3 Sin[x] +
  Cos[y] Cos[z]2 Sin[x] +
  3 Cos[x] Cos[y]2 Sin[y] -
  Cos[x] Cos[z]2 Sin[y] -
  3 Cos[y] Sin[x] Sin[y]2 -
  Cos[x] Sin[y]3 -
  Cos[y] Sin[x] Sin[z]2 +
  Cos[x] Sin[y] Sin[z]2

In[25]:=
  TrigExpand[8 Sin[2*x]2 Cos[x]3]
Out[25]=
  4Cos[x]3 - 4 Cos[x]7 +
  24 Cos[x]5 Sin[x]2 -
  4Cos[x]3 Sin[x]4
```

TrigExpand también trabaja con funciones hiperbólicas:

```
In[26]:= TrigExpand[Sinh[2 x]^3]
Out[26]= - $\frac{3}{2}$  Cosh[x] Sinh[x] +
 $\frac{3}{2}$  Cosh[x]^5 Sinh[x] +
5 Cosh[x]^3 Sinh[x]^3 +
 $\frac{3}{2}$  Cosh[x] Sinh[x]^5
```



---

## Cómo “dibujar” con *Mathematica*

---

### 2.1. La orden Plot

Una de las grandes virtudes de *Mathematica* es lo fácil y completo que es su tratamiento de los gráficos para funciones de una y dos variables. Es posible dibujar a la vez varias funciones y personalizar el resultado en cuanto a escalas, color, etc. También se pueden representar funciones en coordenadas paramétricas e incluso se pueden realizar animaciones.

Un primer paso antes de empezar a representar gráficamente una función es tener una manera cómoda de definirla. Siguiendo la costumbre usaremos las letras  $f$ ,  $g$ , ... para nombrarlas. Para definir la función  $\text{sen}(x)$  haremos lo siguiente

```
In[1]:=
  f[x_]=Sin[x]
Out[1]=
  Sin[x]
```

Observa que se usan los corchetes para todo y que después de la variable  $x$  aparece  $_$ . No hay que olvidar este guión después de cada variable *cuando estemos definiendo una función*.

Una vez definida la función podemos evaluarla en un punto o representarla gráficamente.

Por ejemplo,

```
In[2]:=
  f[1]
Out[2]=
  Sin[1]
```

o si queremos su valor numérico

```
In[3]:=
  N[f[1]]
Out[3]=
  0.841471
```

También podemos asignar un valor a la variable  $x$  y luego evaluar:

```
In[4]:=
  x=0;
  f[x]
Out[4]=
  0
```

Observa que a la hora de evaluar una función que hayamos definido no se usa el “\_”.

Pueden surgir problemas al definir una función si la variable que usamos tiene asignado un valor concreto, con lo que la función sería constante; en este caso, resolvemos el problema con la sentencia `Clear` que vimos en la practica pasada. Por ejemplo, si queremos definir  $f(x) = \ln(x)$ , podemos poner

```
In[5]:=
  Clear[x, f];
  f[x_]=Log[x]
```

y la función queda bien definida aunque antes hubiésemos asignado un valor a  $x$ .

Por último, también se pueden definir funciones “a trozos”. Por ejemplo:

```
f[t_]:= t /; t>0 && t<2
f[t_]:= -t+4 /; t<=0 || t>=2
```

(no produce ninguna salida) define la función

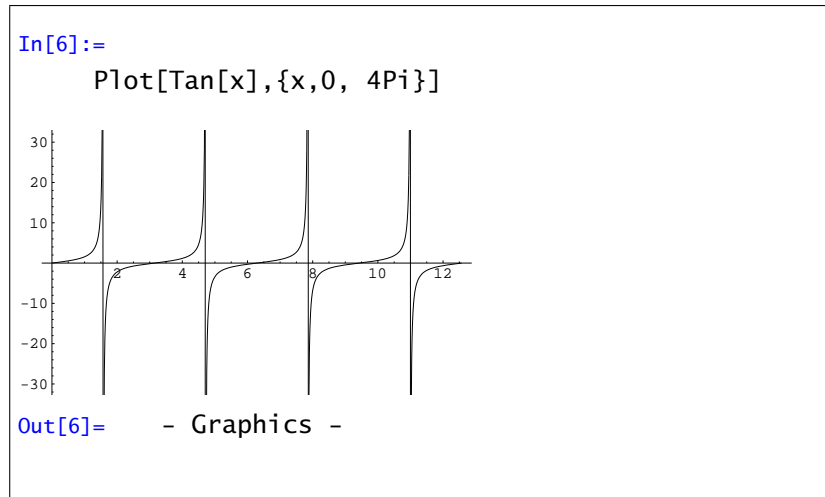
$$f(t) = \begin{cases} t & \text{si } 0 < t < 2 \\ -t + 4 & \text{si } t \leq 0 \text{ ó } t \geq 2 \end{cases}$$

Como has visto, hace falta escribir `:=` para definir una función a trozos. El símbolo `/;` hace las veces de **si** (condicional); los símbolos `||` y `&&` son respectivamente **o** e **y** (suma y producto lógicos).

Para representar gráficamente una función de una variable, el comando que se utiliza es

```
Plot[función, {x, xmin, xmax}, opciones]
```

*Mathematica* decide automáticamente cual es la escala más apropiada para que la gráfica se vea lo mejor posible. Si la función que pretendemos representar tiene alguna discontinuidad o no está definida en algún punto, *Mathematica* intentará dibujarla poniendo de manifiesto este hecho. Por ejemplo, intenta dibujar la función tangente entre 0 y  $4\pi$ :



Un comentario sobre lo que acabamos de hacer: como se puede ver, podemos escribir directamente la función a representar “dentro” del comando Plot. El inconveniente que tiene hacerlo así es que cada vez que queramos trabajar con esa función tenemos que volver a escribirla, siendo más conveniente ponerle nombre (p.e.  $f[x_]=\text{Tan}[x]$ , y usar después el nombre ( $\text{Plot}[f[x], \{x, 0, 4\text{Pi}\}]$ ).

### 2.1.1. Opciones de Plot

Cuando presentamos el comando Plot dijimos que podíamos poner “opciones”; veamos cuáles son algunas de ellas:

- AspectRatio→*número*

Determina la proporción entre los ejes de abscisas y ordenadas. Si especificamos el valor 1 los dos ejes tendrán el mismo tamaño.

```
In[7]:=
Plot[Tan[x],{x,0,4Pi},AspectRatio->2]
```

- PlotRange→{*número1*, *número2*}

Sólo se presentan los valores de  $f(x)$  comprendidos entre *número1* y *número2*. Si no se especifica esta opción aparecen todos los valores posibles de  $f(x)$ , en función del rango de variación de  $x$ .

```
In[8]:=
Plot[Tan[x],{x,0,4Pi},
PlotRange->{2,20}]
```

- PlotStyle

Permite escoger, entre otras cosas, el color de la siguiente forma:

```
RGBColor[número1, número2, número3]
```

Permite escoger un color en función de la cantidad de rojo (*número1*), verde (*número2*) y azul (*número3*), donde los números pueden tomar cualquier valor entre 0 y 1.

```
In[9]:=
Plot[Tan[x],{x,0,4Pi},
PlotStyle->{RGBColor[1,0,0]}]
```

Veamos algunos ejemplos de cómo utilizar estas opciones. Prueba las siguientes órdenes, compara los resultados e intenta variar algunos de los parámetros “a ver que pasa”.

```
Clear[x,f];
f[x_]=Exp[x] ArcTan[x];
Plot[f[x],{x,0,5},
PlotRange->{-1,30}];
```

También se pueden dibujar varias gráficas de funciones a la vez. Una de las formas de distinguirlas es dibujando cada una de un color diferente:

```
Plot[{Sin[x],Cos[x]},{x,-2 Pi,2 Pi},
PlotStyle->{RGBColor[1,0,0],
RGBColor[0,0,1]}];
```

**Nota:** si quieres volver a usar los mismos nombres para funciones distintas recuerda “limpiar” los nombres de las variables y las funciones con el comando `Clear`; si quieres borrar los valores de *todas* las variables, usa `Clear["Global`*"]`. Por ejemplo, si quieres volver a usar las letras  $f$  y  $g$  para otras dos funciones haz lo siguiente:

```
Clear[f,g,x];
f[x_]=Sin[2 x];
g[x_]=Cos[4 x];
Plot[{f[x],g[x]},{x,0,10 Pi},
PlotStyle->{RGBColor[1,0,1],RGBColor[0,1,0.5]}];
```

- Prueba lo mismo que hemos hecho con las funciones  $\sin(x)$  y  $\sin(-x)$  o con las funciones  $\cos(x)$  y  $\cos(-x)$ . ¿Qué función es par y cuál es impar?.
- Intenta lo mismo con las siguientes 3 funciones:  $f(x) = 2^{-x} \sin(6x)$ ,  $g(x) = 2^{-x}$  y  $h(x) = -2^{-x}$ .

## 2.2. Gráficos en coordenadas paramétricas

Como ya recordarás, al menos para rectas, podemos representar una curva del plano en coordenadas paramétricas, o sea, a partir de las coordenadas  $(x(t), y(t))$  variando  $t$  a lo largo de un intervalo.

Para esto tenemos la siguiente orden:

```
ParametricPlot[{x(t),y(t)},{t,min,max}]
```

junto con las opciones ya conocidas de la orden `Plot`. Prueba con alguno de los siguientes ejemplos:

```

ParametricPlot[{t,t^2},{t,-2,2}]

ParametricPlot[{Cos[t],Sin[t]},{t,0,2 Pi}]

ParametricPlot[{{4 Cos[t],3 Sin[t]},{8 Cos[t],
7 Sin[t]}},{t,0,2 Pi}]

ParametricPlot[{Cos[2 t],Sin[2 t]},{t,0,2 Pi}]

```

¿Hay alguna diferencia entre el segundo y el último de los ejemplos?.

### 2.2.1. Algunas curvas planas

**Astroide** el camino recorrido por un punto de un círculo de radio  $r$  rodando dentro de otro círculo de radio  $4r$  parametrizado en coordenadas rectangulares es el siguiente:

```

ParametricPlot[{3*Cos[t]/4+ Cos[3*t]/4, 3*Sin[t]/4 -
Sin[3*t]/4},{t, 0, 2 Pi}]

```

**Cardioide**

```

ParametricPlot[{2 Cos[t] + Cos[2 t], 2 Sin[t]
+ Sin[2 t]},{t,0,2 Pi}]

```

**Catenaria** es la forma de un cable ideal con densidad uniforme colgando de dos puntos

```

ParametricPlot[{t,Cosh[t]},{t,-2,2}]

```

**Lemniscata de Bernoulli** Es una parametrización de  $(x^2 + y^2)^2 = (x^2 - y^2)$ , también llamada hipérbola lemniscata.

```

ParametricPlot[{Cos[t]/(1+Sin[t]^2), Sin[t]
Cos[t]/(1+Sin[t]^2)},{t,0,Pi}]

```

**Espiral equiangular** es aquella espiral en la que el radio corta a la curva en un ángulo constante  $\alpha$

```

ParametricPlot[{ E^(t*Cot[α])*Cos[t],
E^(t*Cot[α])*Sin[t]},{t,0,10 Pi}]

```

**Serpentina** Es una parametrización de  $x^2y + a^2y - b^2x = 0$

```

ParametricPlot[{t,(b^2 t)/(a^2 + t^2)},{t,-10,10}]

```

**Hipérbola** la representación en coordenadas cartesianas de una hipérbola con vértice  $(1, 0)$  y foco  $(a, 0)$  (excentricidad  $a$ ) es  $x^2 - y^2/(a^2 - 1) = 1$ . Una hipérbola rectangular tiene excentricidad  $\sqrt{2}$  y su ecuación es  $xy = 1$ .

```
ParametricPlot[{-Sec[t], Sqrt[a^2-1] Tan[t]}, {t, 0, 2 Pi}]
```

**Eplicicloide** es la curva que describe un punto fijo de un círculo que rueda sin deslizamiento por el exterior de otro círculo ( $n$  es el número de vueltas).

```
ParametricPlot[{(1+1/n) Cos[t]+ Cos[(1+1/n)t]/n,
(1+1/n) Sin[t]+ Sin[(1+1/n)t]/n}, {t, 0, 2 Pi}]
```

### 2.3. El comando Show

Como recordarás, el símbolo % era utilizado para referirse al resultado anterior. Cuando el resultado es un gráfico que queremos volver a visualizar usaremos el comando Show en la forma siguiente

```
Show[%1]
```

Si queremos volver a representar los gráficos 1 y 2:

```
Show[%1,%2]
```

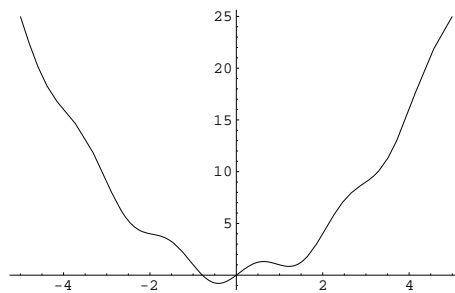
### 2.4. Ejercicios

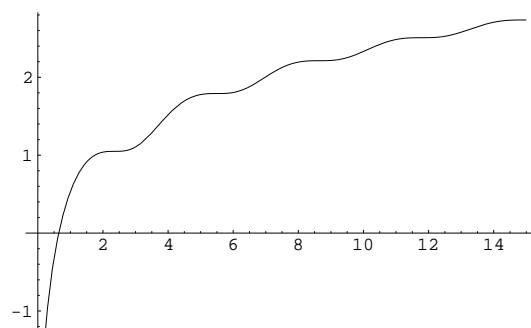
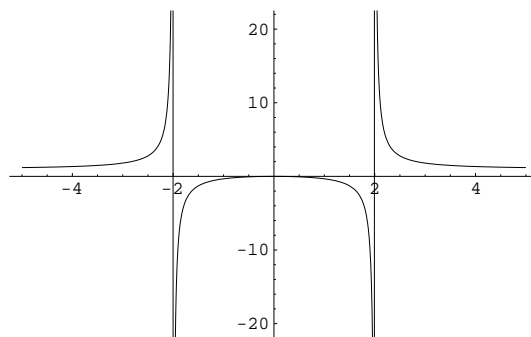
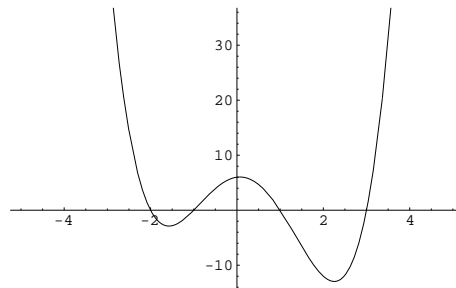
**Ejercicio 1.** Dibuja la función logaritmo neperiano, exponencial y  $f(x) = x^2$  con colores diferentes. Compara el crecimiento de estas funciones cerca de cero y lejos de cero. ¿Qué pasa si la base de la exponencial y el logaritmo son menores que uno?

**Ejercicio 2.** Igual que en el ejercicio anterior compara las gráficas de las funciones trigonométricas con las respectivas funciones hiperbólicas.

**Ejercicio 3.** ¿Cómo cambia la gráfica de una función  $f(x)$  cuando la cambiamos por  $f(a * x)$ ,  $a * f(x)$ ,  $f(x + a)$ , o  $f(x) + a$ ? Prueba con alguna de las funciones anteriores.

**Ejercicio 4.** A continuación puedes encontrar la representación de varias funciones. Encuentra de qué funciones provienen.





---

## Vectores y matrices

---

Para comenzar vamos a ver como trabajar en *Mathematica* con vectores y matrices. Una matriz siempre está delimitada por un par de llaves y, separadas por comas, se escriben las filas agrupadas también con llaves. Por ejemplo,

```
In[1]:=
  {{a11, a12, a13}, {a21, a22, a23}, {a31, a32, a33}}
Out[1]=
  {{a11, a12, a13}, {a21, a22, a23}, {a31, a32, a33}}
```

*Mathematica* no muestra el resultado en la forma de matriz que estamos acostumbrados a ver. Conseguiremos esto mediante el uso de la orden `MatrixForm [ matriz ]`.

```
In[2]:=
  MatrixForm[%]
Out[2]=
```

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Si la matriz es diagonal podemos ahorrarnos algo de trabajo con la orden

```
DiagonalMatrix[{a11, a22, ...}]
```



Por ejemplo,

```
In[3]:=
  DiagonalMatrix[{a,b,c,d}]
Out[3]=
  {{a,0,0,0},{0,b,0,0},{0,0,c,0},{0,0,0,d}}
```

En el caso que la matriz diagonal sea la identidad, podemos usar `IdentityMatrix[n]`. Por ejemplo:

```
In[4]:=
  IdentityMatrix[3]
Out[4]=
  {{1,0,0},{0,1,0},{0,0,1}}

In[5]:=
  MatrixForm[IdentityMatrix[3]]
Out[5]=

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

```

Para sumar matrices, si los órdenes lo permiten, solo tenemos que emplear el signo + y para multiplicarlas por un número podemos usar la simple yuxtaposición o el signo \*. Sin embargo, el producto de matrices, cuando los órdenes lo permiten, tiene una definición distinta. Para multiplicar dos matrices (filas por columnas) o calcular el producto escalar de dos vectores mediante *Mathematica*, debemos usar el punto "." en lugar del signo de la multiplicación "\*". A modo de ejemplo de estas definiciones puedes comprobar los siguientes cálculos

```
In[6]:=
  A={{1,2,3},{4,5,6},{1,3,2}}
  B={{4,3,2},{7,3,1},{1,0,0}}
  a={1,5,8}
  b={1,3,4}
Out[6]=
  {{1,2,3},{4,5,6},{1,3,2}}
  {{4,3,2},{7,3,1},{1,0,0}}
  {1,5,8}
  {1,3,4}
```

Suma de A y B

```
In[7]:=
  A+B
Out[7]=
  {{5,5,5},{11,8,7},{2,3,2}}
```

Producto de una matriz por un número

```
In[8]:=
  5 A
Out[8]=
  {{5,10,15},{20,25,30},{5,15,10}}
```

Producto escalar de a y b

```
In[9]:=
  a.b
Out[9]=
  48
```

Producto de una matriz por un vector A.a (ojo que a se considera columna)

```
In[10]:=
  A.a
Out[10]=
  {35,77,32}
```

Con *Mathematica* podemos definir matrices de forma automática mediante la orden `Table`. La sintaxis es la siguiente

```
Table[Función de i,j, {i, límite de i},{j, límite de j}]
```

Puedes comprenderla mejor con el siguiente ejemplo:

```
In[11]:=
Table[ i*x+ j*y, {i, 3},{j,3}]
Out[11]=
{{x+y,x+2y,x+3y},{2x+y,2x+2y,2x+3y},
{3x+y,3x+2y,3x+3y}}

In[12]:=
MatrixForm[%]
Out[12]=
```

$$\begin{pmatrix} x+y & x+2y & x+3y \\ 2x+y & 2x+2y & 2x+3y \\ 3x+y & 3x+2y & 3x+3y \end{pmatrix}$$

Los siguientes comandos son los que nos permiten extraer elementos, filas, columnas y submatrices de una matriz. Para extraer el elemento (i,j)-ésimo de una matriz A, escribimos

$$A[[i,j]]$$

Para extraer la fila i-ésima,

$$A[[i]]$$

Para obtener una submatriz de A el comando es:

$$A[[\{\text{índices de filas}\},\{\text{índices de columnas}\}]]$$

Obtener la columna j-ésima de A equivale a extraer la fila j-ésima de la traspuesta de A. *Mathematica* usa el comando `Transpose` para transponer matrices:

$$\text{Transpose}[\text{matriz}]$$

Veamos algún ejemplo. Primero buscamos el elemento (2,3) de la matriz A definida en los ejemplos anteriores

```
In[13]:=
A[[2,3]]
Out[13]=
6
```

Seguidamente la fila segunda de A

```
In[14]:=
A[[2]]
Out[14]=
{4, 5, 6}
```

La traspuesta de A

```
In[15]:=
  Transpose[A]
Out[15]=
  {{1, 4, 1}, {2, 5, 3}, {3, 6, 2}}
```

Y la tercera columna de A

```
In[16]:=
  Transpose[A][[3]]
Out[16]=
  {3, 6, 2}
```

Por último, la submatriz de A que tiene las filas (1,2) y las columnas (2,3)

```
In[17]:=
  A[{{1, 2}, {2, 3}}]
Out[17]=
  {{2, 3}, {5, 6}}
```

Seguidamente vamos a ver los comandos que nos permiten calcular inversas y determinantes de matrices con *Mathematica*. Para calcular el determinante el comando es `Det[matriz]` y para calcular la inversa usamos `Inverse[matriz]`. Como ejemplo puedes calcular los determinantes de las matrices A y B del primer ejemplo y, si son no nulos, sus inversas.

Si una matriz no es cuadrada, o si siendo cuadrada su determinante vale 0, ¿cómo podemos calcular su rango?. *Mathematica* no tiene una orden específica para calcular rangos de matrices, pero podemos calcularlos de tres formas:

- Mediante la orden `Minors[matriz,orden]`, que calcula los menores de cierto orden de una matriz dada, dando el resultado en forma de matriz. El rango de la matriz es el mayor orden para el que hay algún menor no nulo.
- Mediante la orden `RowReduce[matriz]`, que convierte la matriz en otra equivalente, pero triangular superior. Expresada de esta forma es fácil ver el rango de una matriz: basta ver cuantos elementos no nulos hay en la diagonal principal (aunque no sea cuadrada).
- Si la matriz es cuadrada, podemos calcularle los valores propios, y su rango será el número de valores propios no nulos. Para calcular los valores propios de una matriz cuadrada, usamos la orden `Eigenvalues[matriz]`.

Por último, la orden `Eigensystem[matriz]` da como resultado una lista con los valores propios y los vectores propios de la matriz. Si sólo nos interesan los vectores propios podemos usar `Eigenvectors[matriz]`. Por ejemplo,

```
In[18]:=
  Eigensystem[{{2,1},{3,-1}}]
Out[18]=
  {{1/2(1-√21), 1/2(1+√21)},
   {1/6(3-√21), 1}, {1/6(3+√21), 1}}
```

**Ejercicio 1.** Consideremos las matrices

$$A = \begin{pmatrix} 1 & -2 & 0 \\ 2 & 5 & 3 \\ -3 & 1 & -4 \end{pmatrix} \quad B = \begin{pmatrix} 0 & -2 & 6 \\ 12 & 2 & 0 \\ -1 & -1 & 3 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 2 & 0 & -5 \\ -4 & -2 & 1 & 0 \\ 3 & 2 & -1 & 3 \\ 5 & 4 & -1 & -5 \end{pmatrix} \quad D = \begin{pmatrix} -1 & 2 & 3 & 0 \\ 12 & -5 & 0 & 3 \\ -6 & 0 & 0 & 1 \end{pmatrix}$$

- Calcular  $A.B$ ,  $A + B$ ,  $D.C$ .
- Extraer la segunda fila de  $A$ , la tercera columna de  $C$  y el elemento  $(3, 3)$  de  $D$ .
- Calcular  $\text{Det}(A)$ ,  $\text{Det}(B)$  y  $\text{Det}(C)$ . Para las matrices cuyo determinante sea no nulo, calcular su inversa. Calcular sus valores propios.
- Calcular el rango de las matrices  $A$ ,  $B$ ,  $C$ ,  $D$ ,  $D.C$  y  $A + B$ .
- Usar la orden `Table` para generar una matriz del orden  $3 \times 3$ , de forma que el elemento  $(i, j)$  sea  $i * j$ . Calcula el determinante, su inversa si la tiene, y su rango. ¿Cuáles son sus valores propios?.

### 4.1. Ecuaciones “sencillas”

*Mathematica* puede resolver los tipos más comunes de ecuaciones y sistemas de ecuaciones algebraicas, tanto de forma exacta (si se puede) como aproximada. Resolvamos el sistema de ecuaciones

$$\begin{cases} x^2 + y^2 = 1 \\ (x - 2)^2 + (y - 1)^2 = 4 \end{cases}$$

Para ello usamos la sentencia `Solve`

```
In[1]:=
Solve[{x^2 + y^2==1, (x-2)^2
+ (y-1)^2==4},{x,y}]
Out[1]=
{{x->0,y->1},{x->4/5,y->-3/5}}
```

Observa que en el comando `Solve` tenemos que decirle a *Mathematica* las variables respecto de las cuales queremos resolver el sistema. La sintaxis es

```
Solve[{ecuación1, ecuación2,...},{variable1, variable2,...}]
```

Para *Mathematica* una ecuación está formada por dos expresiones igualadas con el signo “==”.

A veces, *Mathematica* no puede resolver de forma exacta una ecuación

```
In[2]:=
Solve[x6+x+1==0,x]
Out[2]=
{{x → Root[1 + #1 + #16&, 1]},
 {x → Root[1 + #1 + #16&, 2]},
 {x → Root[1 + #1 + #16&, 3]},
 {x → Root[1 + #1 + #16&, 4]},
 {x → Root[1 + #1 + #16&, 5]},
 {x → Root[1 + #1 + #16&, 6]}}
```

Este error aparece porque estamos usando un polinomio de grado alto y es posible que la ecuación no pueda resolverse con operaciones elementales. En este caso usamos la orden `NSolve` (tiene la misma sintaxis que `Solve`) para una resolución numérica

```
In[3]:=
NSolve[x6+x+1==0,x]
```

y *Mathematica* da 6 soluciones aproximadas (en este caso complejas) de la ecuación anterior. Después de la variable podemos añadir el número de cifras significativas que queremos que tengan las soluciones

```
In[4]:=
NSolve[x6+x+1==0,x,50]
```

Además de ecuaciones polinómicas, `Solve` puede resolver ecuaciones en las que aparecen otro tipo de funciones. Por ejemplo,

```
In[5]:=
Solve[ArcCos[x]-ArcTan[x]==0,x]
Out[5]=
{{x →  $\sqrt{\frac{-1}{2} + \frac{\sqrt{5}}{2}}$ }}
```

En estos casos lo más frecuente es que no se pueda resolver el sistema o que no se puedan calcular todas las soluciones. Si ocurre esto *Mathematica* da un mensaje de advertencia avisando de que puede haber más soluciones:

```
In[6]:=
Solve[Sin[x] Cos[x]==0,x]
Solve::ifun: Inverse functions are being used by
Solve, so some solutions may not be found.
Out[6]=
{{x → 0}, {x →  $-\frac{\pi}{2}$ }, {x →  $\frac{\pi}{2}$ }}
```

## 4.2. Sistemas de ecuaciones lineales

Vamos a aplicar lo que hemos visto a la resolución de sistemas lineales de la forma

$$A.x = b,$$

donde  $A$  es una matriz de orden  $m \times n$  y  $x$  y  $b$  son vectores de órdenes  $n$  y  $m$  respectivamente.

El caso más fácil es que la matriz  $A$  sea cuadrada e invertible, entonces la solución del sistema es tan sencilla como multiplicar (matricialmente) la inversa de  $A$  por  $b$ .

Cuando resolvemos sistema de ecuaciones con *Mathematica*, la soluciones nos aparecen como listas de valores, es decir como vectores. El problema que nos surge es que no podemos trabajar directamente con estos resultados. La solución para este problema resulta sencilla, podemos definir un vector cuyas componentes sean las soluciones de nuestra ecuación y podemos utilizar sus componentes como hemos visto anteriormente. Para esto llamamos "S" al vector solución y lo definimos cuando ponemos  $S$  igual a la inversa de  $A$  por  $b$  (cuando es posible). Veamos un ejemplo. Como en este caso la matriz es cuadrada y tiene determinante distinto de cero, sólo tenemos que calcular su inversa:

```
In[7]:=
A={{1,3,5},{9,7,5},{2,4,5}}
b={4,7,3}
Det[A]
Out[7]=
{{1,3,5},{9,7,5},{2,4,5}}
{4,7,3}
20

In[8]:=
S=Inverse[A].b
Out[8]=
{7/4,-11/4,21/10}
```

De este modo tenemos definido el vector de soluciones del sistema como  $S$ . Si queremos conocer la solución de la segunda variable solo tenemos que buscar la segunda componente de  $S$ :

```
In[9]:=
S[[2]]
Out[9]=
-11/4
```

Otra posibilidad para resolver el sistema es usar la orden

`LinearSolve[matriz A,vector b].`

¿Qué pasa si la solución no es única?. Puedes comprobar con ejemplos sencillos que la orden `LinearSolve` da como resultado un vector aunque el sistema tenga infinitas soluciones:



```
In[10]:=
  LinearSolve[{{1,1}},{1}]
Out[10]=
  {1,0}
```

en este ejemplo,  $(1, 0)$  es solución del sistema  $x + y = 1$ , pero hay más soluciones, como por ejemplo  $(0, 1)$ . La forma de obtener las demás soluciones es calcular el núcleo de la matriz de coeficientes mediante la orden `NullSpace`, que da una base de dicho núcleo. De esta forma, si sabemos una solución particular del sistema  $A.x = b$ ,  $x_0$  (obtenida mediante `LinearSolve`) y una base del núcleo de  $A$ ,  $x_1, x_2, \dots$ , todas las soluciones del sistema serán de la forma  $x_0 + \lambda_1 x_1 + \lambda_2 x_2 + \dots$ , donde  $\lambda_1, \lambda_2, \dots$  son parámetros. En nuestro ejemplo anterior ( $x + y = 1$ ) habíamos obtenido

```
In[11]:=
  LinearSolve[{{1,1}},{1}]
Out[11]=
  {1,0};
```

el núcleo de la matriz  $\begin{pmatrix} 1 & 1 \end{pmatrix}$  tiene base

```
In[12]:=
  NullSpace[{{1,1}}]
Out[12]=
  {{-1,1}};
```

con lo que todas las soluciones del sistema serán de la forma  $(1, 0) + \lambda(-1, 1)$ , con  $\lambda$  un parámetro real.

Otra forma de resolver sistemas es usar la orden `Solve` que ya conocemos. Un ejemplo nos ayudará a entender la mecánica:

```
In[13]:=
  A={{1,1,1},{2,2,2},{1,0,1}};
  v={x,y,z};
  b={0,0,1};
  Solve[A.v==b,{x,y,z}]
Out[13]=
  {{y->-1, x->1-z}}
```

En este caso, como la matriz  $A$  tiene rango dos, las soluciones nos vienen dadas de forma implícita mediante dos ecuaciones:  $\{y = -1, x = 1 - z\}$ ; por tanto, el espacio afín de soluciones tendrá dimensión 1.

**Ejercicio.** Usando las matrices

$$A = \begin{pmatrix} 1 & -2 & 0 \\ 2 & 5 & 3 \\ -3 & 1 & -4 \end{pmatrix} \quad B = \begin{pmatrix} 0 & -2 & 6 \\ 12 & 2 & 0 \\ -1 & -1 & 3 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 2 & 0 & -5 \\ -4 & -2 & 1 & 0 \\ 3 & 2 & -1 & 3 \\ 5 & 4 & -1 & -5 \end{pmatrix} \quad D = \begin{pmatrix} -1 & 2 & 3 & 0 \\ 12 & -5 & 0 & 3 \\ -6 & 0 & 0 & 1 \end{pmatrix}$$

resolver los siguientes sistemas de ecuaciones lineales:  $A.x = (1, 1, 0)^t$ ,  $B.x = (2, 0, 0)^t$ ,  $C.x = (-1, 2, 0, 0)^t$  y  $D.x = (1, -1, 0, 0)^t$ .

### 4.3. Resolución de sistemas de ecuaciones

Veamos todos los comandos con algunos ejemplos:

- Comandos `Solve` y `NSolve`: Sirven para resolver sistemas de ecuaciones polinómicas. El primero sólo resuelve sistemas de grado bajo pero con soluciones exactas, expresadas mediante radicales. El segundo da soluciones aproximadas pero sirve para cualquier grado.

- Comando `Reduce`: simplifica sistemas de ecuaciones dependientes de parámetros, haciendo una discusión de casos si es necesario. Su sintaxis es

```
Reduce[{ecuación1, ecuación2, ...}, {var1, var2, ...}]
```

Al usar este comando hay que tener en cuenta que todo lo que no se le introduce como variable es considerado parámetro. Además, el resultado obtenido al usar `Reduce` difiere bastante del obtenido al usar `Solve` o `NSolve`, pues este comando no devuelve valores numéricos, sino ecuaciones más simples que habrá que resolver; dichas ecuaciones vendrán expresadas usando “operadores lógicos”: `&&` es la conjunción “y”, `||` es la disyunción “o”. Un ejemplo:

```
In[14]:=
Reduce[{x + a y == a,
b y + b x == 0}, {x, y}]
Out[14]=
x == a(1 - y) && b == 0 ||
-1 + a != 0 && x == -a/(-1 + a)
&& y == a/(-1 + a)
```

*Mathematica* da dos soluciones: la primera  $x = a(1 - y)$  con  $b = 0$ ; la segunda:  $x = \frac{-a}{a - 1}$ ,  $y = \frac{a}{a - 1}$  que sólo vale si  $a \neq 1$  y para cualquier  $b$ .

**Ejercicio 1.** Discutir en función de los parámetros  $a$  y  $b$ , los siguientes sistemas de ecuaciones:

$$\begin{cases} x + az = 0 \\ x + 2y = b \\ x - y + az = 0 \end{cases} \quad \begin{cases} x + az = 1 \\ x + 2y = -1 \\ x - y + az = 3 \end{cases} \quad \begin{cases} ax^2 + by = 5 \\ x - by = 1 \end{cases}$$

$$\begin{cases} x - z = 1 \\ x + 2y = 1 \\ x + az = b \end{cases} \quad \begin{cases} x + az = 0 \\ x + 2y = b \\ x - y + az = 0 \end{cases} \quad \begin{cases} x + 2z = 0 \\ 7x - 2y = 0 \\ 6x - 2y + az = 0 \end{cases}$$

## 4.4. Otros métodos

Hemos visto como resolver ecuaciones y sistemas de ecuaciones con *Mathematica* mediante la orden `Solve`. El problema de usar la orden `Solve`, es que solo nos permite resolver ecuaciones y sistemas de ecuaciones para los que es posible aplicar un método algebraico sencillo. En otras palabras, la orden `Solve` está limitada a ecuaciones y sistemas de ecuaciones que se puedan resolver mediante operaciones elementales.

Recordemos el siguiente ejemplo que ya nos habíamos encontrado:

```
In[15]:=
  Solve[x^6+x+1==0,x]
Out[15]=
  {{x -> Root[1 + #1 + #1^6&, 1]},
   { x -> Root[1 + #1 + #1^6&, 2]},
   { x -> Root[1 + #1 + #1^6&, 3]},
   { x -> Root[1 + #1 + #1^6&, 4]},
   { x -> Root[1 + #1 + #1^6&, 5]},
   { x -> Root[1 + #1 + #1^6&, 6]}}
```

Ahora es un buen momento para plantearle a *Mathematica* algunas ecuaciones, por ejemplo polinomiales de grado alto, para ver las limitaciones de la orden `Solve`.

```
In[16]:=
  Solve[x^6+x^3 + 1 == 0, x ]
In[17]:=
  Solve[x^6+ x^5 + 1 == 0, x]
```

Prueba con otras ecuaciones de grado alto la orden `Solve`.

En estas condiciones, nos damos cuenta de la necesidad de encontrar o aproximar soluciones para ecuaciones del tipo  $f(x) = 0$ , donde, en principio, podemos considerar como  $f$  cualquier función real de una variable. El objetivo de esta práctica es aprender a programar algoritmos con *Mathematica* para aproximar la solución de estas ecuaciones.

Lo primero que tenemos que tener en cuenta es que no existe ningún método general para resolver todo este tipo de ecuaciones en un número finito de pasos.

Lo que sí tendremos es condiciones para poder asegurar, bajo ciertas hipótesis sobre la función  $f$ , que un determinado valor es una aproximación de la solución de la ecuación con un error prefijado.

El principal resultado para asegurar la existencia de solución para la ecuación  $f(x) = 0$  en un intervalo  $[a, b]$ , es el Teorema de Bolzano. Dicho teorema asegura que si  $f$  es continua en  $[a, b]$  y cambia de signo en el intervalo, entonces existe al menos una solución de la ecuación en el intervalo  $[a, b]$ . Los dos métodos que vamos a usar se basan en este resultado.

Ambos métodos nos proporcionan un algoritmo para calcular una sucesión de aproximaciones, y condiciones sobre la función  $f$  para poder asegurar que la sucesión que obtenemos converge a la solución del problema. Una vez asegurada esta convergencia, bastará tomar alguno de los términos de la sucesión que se aproxime a la sucesión con la exactitud que deseemos.

#### 4.4.1. Breves conceptos de programación

Antes de introducirnos en el método teórico de resolución, vamos a presentar algunas sentencias sencillas de programación que necesitaremos más adelante.

La primera de las órdenes que vamos a ver es el comando For, usada para realizar bucles. Un “bucle” es un proceso repetitivo que se realiza un cierto número de veces. Un ejemplo de bucle puede ser el siguiente: supongamos que queremos obtener los múltiplos de siete comprendidos entre 7 y 70; para ello, multiplicamos 7 por cada uno de los números naturales comprendidos entre 1 y 10, es decir, repetimos 10 veces la misma operación: multiplicar por 7.

La sintaxis de la orden es la siguiente,

```
For[expresión inicial, condición, incremento, expresión]
```

- *expresión inicial* nos sitúa en las condiciones de comienzo del bucle.
- *condición* dirá a *Mathematica* el momento de detener el proceso.
- *incremento* expresará la forma de aumentar la condición inicial.
- *expresión* dirá a *Mathematica* lo que tiene que realizar en cada paso; la *expresión* puede estar compuesta de varias sentencias separadas mediante punto y coma.

Para terminar de comprender bien el funcionamiento de esta orden puedes teclear y evaluar los siguientes ejemplos:

En primer lugar, intentemos escribir los múltiplos de 7 comprendidos entre 7 y 70:

```
In[18]:=
For[i=1, i<=10, i=i+1, Print[7*i]]
7
14
21
28
35
42
49
56
63
70
```

El segundo ejemplo consiste en hacer a *Mathematica* sumar de cinco en cinco, comenzando en 5 y terminando en 25.

```
In[19]:=
  For[i=5, i<=25, i=i+5, Print[i]]
5
10
15
20
25
```

El tercero genera el seno de 1, el seno de 11 y el seno de 21.

```
In[20]:=
  For[i=1, i<=30, i=i+10, Print[Sin[i]]]
Sin[1]
Sin[11]
Sin[21]
```

**Ejercicio** Usa el comando *For* en los siguientes ejemplos:

- Sumar los números naturales entre 400 y 450.
- Sumar los cuadrados de los primeros 10 naturales.

La segunda sentencia es la orden condicional *If*. Esta sentencia comprueba si se verifica una condición, después, si la condición es verdadera *Mathematica* ejecutará una *expresión1*, y si es falsa ejecutará otra *expresión2*. Su sintaxis es la siguiente,

```
If[condición, expresión1, expresión2]
```

Las expresiones 1 y 2 pueden estar formadas por varias órdenes separadas por punto y coma. Para familiarizarnos con esta orden vamos a teclear y ejecutar el siguiente ejemplo,

```
In[21]:=
a=5;
If[a<=0, Print["No existe el logaritmo
neperiano de ", a] , Print["El
logaritmo neperiano de ", a,"vale ",
N[Log[a]]]]
El logaritmo neperiano de 5 vale 1.60944
```

Ahora puedes probar a cambiar el valor de *a* para ver como la sentencia *If* va cambiando sus resultados.

La última sentencia de programación que vamos a ver es la orden *Break* [ ] cuya única finalidad es la de interrumpir un bucle en el momento que se ejecuta y no terminar todos los pasos que faltan hasta la condición final del bucle. En el siguiente ejemplo se puede comprender rápidamente el uso de esta orden.

```

In[22]:=
For[i=1,i<=30,i=i+1,
If[N[Log[i]]<=2,Print[Log[i]],
Print["El logaritmo de ", i, "es mayor
que 2"]; Break[] ]]
0
Log[2]
Log[3]
Log[4]
Log[5]
Log[6]
Log[7]
El logaritmo de 8 es mayor que 2

```

#### 4.4.2. Método de Bisección

En este método sólo es necesario que la función  $f$  sea continua en el intervalo  $[a, b]$  y verifique  $f(a)f(b) < 0$ . En estas condiciones, el Teorema de Bolzano nos asegura la existencia de una solución de la ecuación en  $[a, b]$ . El siguiente paso consiste en tomar como nueva aproximación,  $c = \frac{a+b}{2}$  (el punto medio del segmento  $[a, b]$ ). Si  $f(c) = 0$ , hemos encontrado una solución de la ecuación y por tanto hemos terminado. Si  $f(c) \neq 0$ , consideramos como nuevo intervalo, o bien  $[a, c]$  (si  $f(a)f(c) < 0$ , o bien  $[c, b]$  si es que  $f(c)f(b) < 0$  y repetimos la estrategia en el nuevo intervalo.

En este método podemos conocer el error que cometemos en cada una de las aproximaciones. Si nos damos cuenta, partimos de un intervalo de longitud  $(b-a)$  y lo partimos por la mitad quedándonos con el subintervalo en el que podemos asegurar la existencia de una raíz si ésta no era el punto medio de  $[a, b]$ . El nuevo subintervalo tiene longitud  $\frac{b-a}{2}$  y contiene a la solución y a la primera aproximación. En consecuencia, el error cometido es menor o igual que  $\frac{b-a}{2}$ . De la misma forma, el error cometido en el paso  $n$ -ésimo es menor o igual que  $\frac{b-a}{2^n}$ .

A partir de aquí, podemos deducir el número de iteraciones necesarias para obtener una aproximación con un error o exactitud prefijados. Si notamos por “ $Ex$ ” a la exactitud prefijada, entonces para conseguir dicha precisión, el número “ $n$ ” de iteraciones necesarias deberá satisfacer

$$\frac{b-a}{2^n} < Ex$$

así,

$$n = E \left[ \text{Log}_2 \left( \frac{b-a}{Ex} \right) \right] + 1,$$

donde  $E[\cdot]$  denota la “parte entera” de un número (esto es, el mayor de los enteros que son menores que el número. Para obtener la “parte entera” de un número que está expresado en notación decimal, *Mathematica* tiene el comando `Floor`.

Para definir un algoritmo de cálculo de la sucesión de aproximaciones de este método mediante *Mathematica*, vamos a resolver como ejemplo la ecuación  $x^6 + x - 5 = 0$  en el intervalo  $[0, 2]$ .

Definiremos en primer lugar la función, el intervalo y la exactitud. Seguidamente calcularemos el número “ $P$ ” de iteraciones (o pasos) necesarias, para a continuación llevar a cabo el cálculo de dichas aproximaciones, que se irán visualizando a medida que se van haciendo los sucesivos cálculos.

```

In[23]:=
Clear["Global`*"]
f[x_]=x6+x-5;
a=0;
b=2;
Ex=10^(-6);

P=Floor[N[Log[2, (b-a)/Ex]]]+1;
For[i=1,i<=P,i=i+1,
  c=(a+b)/2;
  If[f[c]==0,
    Print["Sol. exacta: ",N[c,10],
      "hallada en ",i,"pasos."
    ];
    Break[]
  ];
  If[f[c]f[a]<0,b=c,a=c];
  Print[i," - aprox: ",N[c,10],
    "(error <",N[b-a],")"
  ]
]

```

Este comando da lugar a la siguiente respuesta por parte del *Mathematica* :

```

1 - aprox: 1. (error <1.)
2 - aprox: 1.5 (error <0.5)
3 - aprox: 1.25 (error <0.25)
4 - aprox: 1.125 (error <0.125)
5 - aprox: 1.1875 (error <0.0625)
6 - aprox: 1.21875 (error <0.03125)
7 - aprox: 1.234375 (error <0.015625)
8 - aprox: 1.2421875 (error <0.0078125)
9 - aprox: 1.24609375 (error <0.00390625)
10 - aprox: 1.248046875 (error <0.00195312)
11 - aprox: 1.247070313 (error <0.000976562)
12 - aprox: 1.246582031 (error <0.000488281)
13 - aprox: 1.246826172 (error <0.000244141)
14 - aprox: 1.246704102 (error <0.00012207)
15 - aprox: 1.246643066 (error <0.0000610352)
16 - aprox: 1.246612549 (error <0.0000305176)
17 - aprox: 1.246627808 (error <0.0000152588)
18 - aprox: 1.246635437 (error <7.62939 10-6)
19 - aprox: 1.246631622 (error <3.8147 10-6)
20 - aprox: 1.246629715 (error <1.90735 10-6)
21 - aprox: 1.246628761 (error <9.53674 10-7)

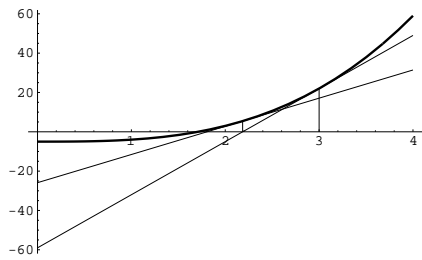
```

A continuación, prueba a cambiar la función, los extremos del intervalo (en los cuales dicha función cambia de signo), así como la exactitud exigida. Intenta también buscar un caso simple en el que se encuentre la solución exacta en unos pocos pasos. Por último, intenta usar el algoritmo anterior para calcular  $\sqrt[3]{5}$  con una exactitud de  $10^{-10}$ .

Graba ese algoritmo en disco... más tarde compararemos su efectividad con la del siguiente método.

### 4.4.3. Método de Newton-Raphson

El método para la construcción del algoritmo que vamos a estudiar ahora es conocido con el nombre de “método de Newton-Raphson” debido a sus autores. Este método nos proporciona un algoritmo para obtener una sucesión de aproximaciones. Para asegurar la convergencia de la sucesión (hacia la solución de la ecuación), bajo ciertas condiciones, usaremos el Teorema de Newton-Raphson, cuyo enunciado daremos más adelante.



La forma de construir los distintos términos de la sucesión de aproximaciones es bastante sencilla y responde a una idea muy intuitiva. Primero suponemos que  $f$  es derivable al menos dos veces, con primera derivada no nula en el intervalo donde trabajamos. Notaremos por  $f'$  y  $f''$  a la derivadas primera y segunda de  $f$  respectivamente. Una vez fijado un valor inicial  $x_1$ , el término  $x_2$  se obtiene como el punto de corte de la recta tangente a  $f$  en  $x_1$  con el eje  $OX$ . De la misma forma, obtenemos  $x_{n+1}$  como el punto de corte de la recta tangente a  $f$  en el punto  $x_n$  con el eje  $OX$ .

De lo dicho hasta aquí se deduce:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Para comprender el algoritmo observa el gráfico donde se ve cómo se generan los valores de las aproximaciones.

El mencionado Teorema de Newton-Raphson, es el siguiente:

**Teorema** (de Newton-Raphson). Tomemos una función  $f$  que admite derivada segunda en el intervalo  $[a, b]$  y verifica:

1.  $f(a)f(b) < 0$ ,
2.  $f'(x) \neq 0$ , para todo  $x \in [a, b]$ ,
3.  $f''(x)$  no cambia de signo en  $[a, b]$ .

Entonces, tomando como primera aproximación el extremo del intervalo  $[a, b]$  donde  $f$  y  $f''$  tienen el mismo signo, la sucesión de valores  $x_n$  del método de Newton-Raphson es convergente hacia la única solución de la ecuación  $f(x) = 0$  en  $[a, b]$ .

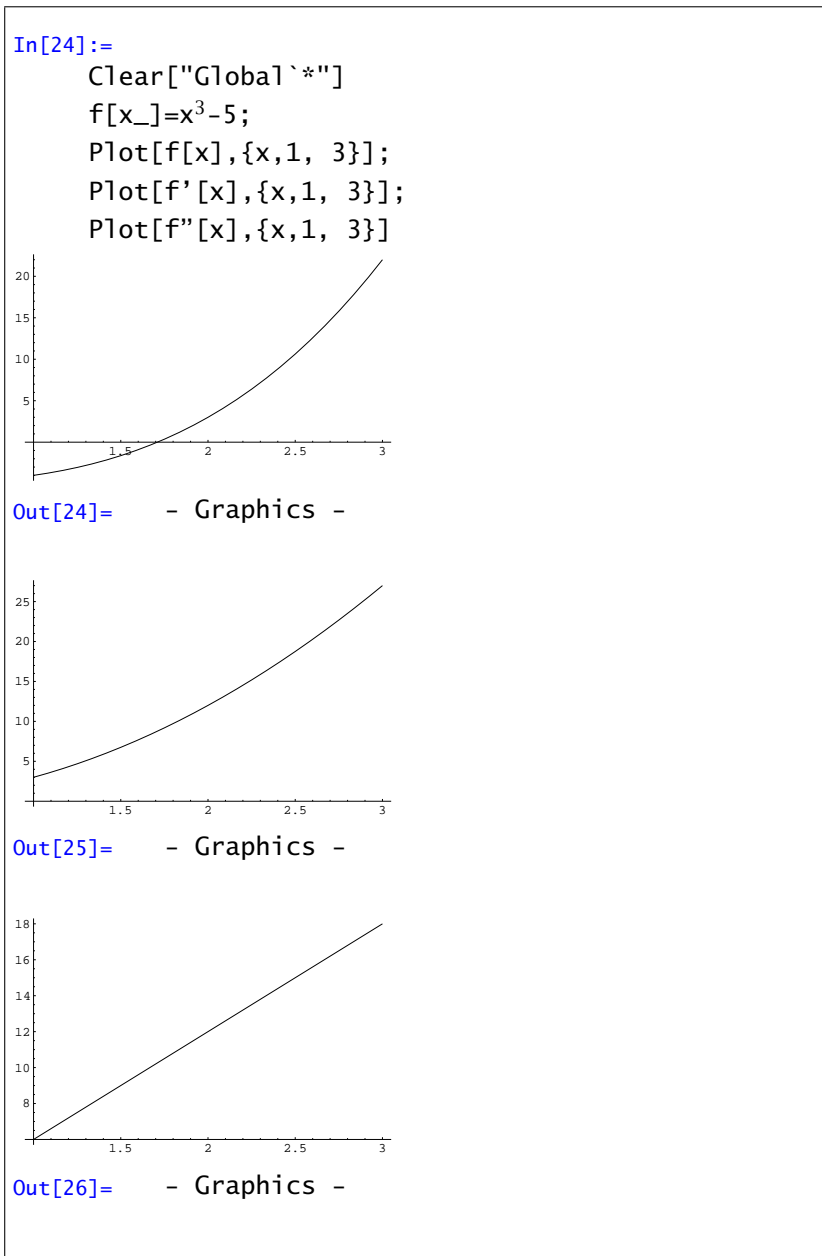
Una vez que tenemos asegurada la convergencia de la sucesión hacia la solución de la ecuación, deberíamos decidir la precisión. Sin embargo, veremos que el método es tan rápido en su convergencia que por defecto haremos siempre 10 iteraciones. Otra posibilidad sería detener el cálculo de cuando el valor absoluto de la diferencia entre  $x_n$  y  $x_{n+1}$  sea menor que la precisión buscada (lo cual no implica necesariamente que el error cometido sea menor que la precisión).

Utilizaremos ahora *Mathematica* para generar la sucesión de aproximaciones. Resolvamos de nuevo el ejemplo de  $x^3 - 5 = 0$  en el intervalo  $[1, 3]$ .

Podemos comprobar, dibujando las gráficas de  $f(x) = x^3 - 5$ ,  $f'(x)$  y  $f''(x)$  en el intervalo  $[1, 3]$ , que estamos en las condiciones bajo las cuales el Teorema de Newton-Raphson nos asegura



convergencia. Las gráficas se pueden obtener de la forma que aparece en la página siguiente (ojo, para escribir la “prima” en  $f'[x]$ , debemos usar la tecla que está al lado del 0 en el teclado).



A continuación, generaremos los términos de la sucesión de aproximaciones mediante el siguiente algoritmo. Comenzaremos por definir la función  $f$ , y el valor de la primera aproximación. Inmediatamente después definimos el algoritmo del método de Newton-Raphson, e iremos visualizando las sucesivas aproximaciones. Como dijimos, pondremos un límite de 10 iteraciones, aunque usando mayor precisión decimal puedes probar con un número mayor de iteraciones.

```
In[27]:=
Clear["Global`*"]
f[x_]=x3-5;
y=3;

For[i=1,i<=10,i=i+1,
  y1=N[y-f[y]/f'[y],20];
  Print[i, aprox: ",y1];
  y=y1
]
```

Observarás al ejecutar este grupo de comandos que ya en la sexta iteración se han “estabilizado” veinte cifras decimales de la solución. Prueba a continuación a cambiar a un mayor número de decimales en las aproximaciones. Si es necesario, aumenta también el número de iteraciones. Observa qué rápidamente se “estabilizan” los decimales de las iteraciones. Sí, como puedes ver, el método de Newton-Raphson es *muy rápido*.

## 4.5. El comando FindRoot

El método que hemos visto en la sección anterior se puede usar directamente con la orden FindRoot.

```
FindRoot[función de x,{x,aproximación inicial}]
```

La aproximación inicial debería ser un punto que este “más o menos” cerca del cero de la función que queremos buscar. En cualquier caso hay que tener en cuenta que aún escogiendo puntos iniciales cercanos el resultado puede ser completamente distinto.

Por ejemplo,

```
In[28]:=
FindRoot[x3-5,{x,3}]
Out[28]=
{x->1.70998}
```

El comando FindRoot también se puede usar con sistemas de ecuaciones. Sólo hay que escribir las ecuaciones agrupadas entre llaves y dar un punto inicial para cada variable:

```
FindRoot[{eq1,eq2,...},{x,x0},{y,y0,...}]
```

**Ejercicios 1.**

1. Considérese la ecuación

$$e^{(x^2+x+1)} - e^{x^3} - 2 = 0.$$

Calcular programando los métodos de bisección y de Newton-Raphson, la solución de dicha ecuación en el intervalo  $[-0,3, 1]$  con exactitud  $10^{-10}$ . Encontrar la solución mediante la orden FindRoot y comparar los resultados.

2. Dada la ecuación

$$\tan(x) = \frac{1}{x}$$

buscar la solución que posee en el intervalo  $[0, \frac{\pi}{2}]$ , usando los métodos estudiados.

**Ejercicio 2.** Encontrar una solución del siguiente sistema de ecuaciones “cerca” de (0,0):

$$\begin{cases} \operatorname{sen} x \cos y = \frac{1}{4} \\ xy = 1 \end{cases}$$

---

## Extremos de funciones de una variable

---

### 5.1. Continuidad y límites

Básicamente todas las funciones que han aparecido hasta ahora eran continuas. En el caso de que no lo fueran *Mathematica* tampoco parece haber tenido demasiados problemas. Cuando hemos dibujado la gráfica de la función tangente, *Mathematica* ha dibujado una asíntota vertical en los puntos donde el coseno valía cero.

En el caso de que queramos estudiar la continuidad de una función en un punto  $a$ , tenemos que calcular  $\lim_{x \rightarrow a} f(x)$ . *Mathematica* puede estudiar límites (también laterales). La orden es la siguiente:

```
Limit[expresión,variable->a]
Limit[expresión,variable->a,Direction->1]
Limit[expresión,variable->a,Direction->-1]
```

La primera calcula el límite de la expresión cuando la variable tiende a  $a$ . Las siguientes calculan el límite por la izquierda y la derecha respectivamente. Por ejemplo,

```
In[1]:=
  Limit[ $\frac{n}{n+1}$ ,n->+∞]
Out[1]=
  1

In[2]:=
  Limit[Tan[x],x->Pi/2,Direction->1]
Out[2]=
  ∞
```

Cuando la función oscila entre dos valores *Mathematica* intenta dar el intervalo aunque no exista el límite.

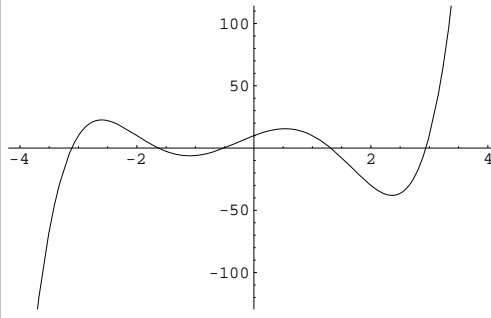
```
In[3]:=
  Limit[Sin[1/x],x->0]
Out[3]=
  Interval[{-1,1}]
```

## 5.2. Máximos y mínimos relativos

Esencialmente, lo que haremos para localizar los extremos relativos de una función  $f$  será encontrar soluciones de la ecuación  $f'(x) = 0$ . Para solucionar dicha ecuación, podemos usar (cuando sea posible por la simplicidad de la ecuación) comandos directos como `Solve` ó `NSolve`, o bien usaremos el comando `FindRoot` cuando las soluciones no se puedan obtener directamente.

Comencemos buscando los extremos relativos de la función polinómica  $f(x) = x^5 + x^4 - 11x^3 - 9x^2 + 18x + 10$  en el intervalo  $[-4, 4]$ . Para ello, definiremos convenientemente la función  $f$ , y dibujaremos su gráfica entre  $-4$  y  $4$ .

```
In[4]:=
  Clear["Global`*"]
  f[x_]= x^5+x^4-11x^3-9x^2+18x+10;
  Plot[f[x],{x,-4,4}]
Out[4]= - Graphics -
```



A simple vista observaremos que hay:

- un máximo relativo entre  $-3$  y  $-2$ ,
- un mínimo relativo entre  $-2$  y  $-1$ ,
- un máximo relativo entre  $0$  y  $1$ ,
- un mínimo relativo entre  $2$  y  $3$ .

En cualquier caso, dada la simplicidad de esta función (polinómica de grado bajo), es posible calcular directamente los ceros de la derivada con el comando

```
In[5]:=
NSolve[f'[x]==0,x]
```

Como vemos, aparecen cuatro soluciones que se corresponden con los extremos que habíamos localizado a simple vista.

Sin embargo, no todas las funciones permiten calcular de un modo simple las soluciones de la ecuación  $f'(x) = 0$ . Por ejemplo, consideremos ahora la función  $f(x) = \sin(x) + e^x$  en el intervalo  $[-7, 0]$ . Pintémosla igual que antes, sustituyendo la antigua función por la nueva. De nuevo, a simple vista se observa que hay:

- un máximo relativo cerca de  $-5$ ,
- un mínimo relativo cerca de  $-2$

Sin embargo, ahora no funciona satisfactoriamente el comando `NSolve` (inténtalo). Optaremos por aplicar el método de Newton-Raphson (mediante el comando `FindRoot`) para encontrar las raíces de la derivada de  $f$ . Recordemos que en el comando `FindRoot` es necesario introducir una “primera aproximación” de la solución, que servirá para iniciar el proceso iterativo. En este caso concreto, dada la situación aproximada de los extremos, probaremos a iniciar el método desde  $-5$  y también desde  $-2$ . Concretamente, escribiremos el comando

```
In[6]:=
FindRoot[f'[x],{x,-5}]
FindRoot[f'[x],{x,-2}]
```

El programa nos da entonces una buena aproximación de los puntos donde la función alcanza los extremos observados.

A continuación veremos un caso en el que la gráfica diseñada por el programa nos puede llevar a engaño. Descubriremos el error gracias al “test de la segunda derivada”. Sea  $f(x) = x^2 - 10x - 40 + \frac{1}{10x^2 - 100x + 251}$ , y pintémosla en el intervalo  $[-15, 15]$ .

Aparentemente, hay un mínimo cerca de 5. Si buscamos raíces de la derivada ejecutando el comando `FindRoot` con dato inicial 5, obtenemos que precisamente para  $x = 5$  la derivada se anula (se podía haber comprobado antes pidiendo al *Mathematica* el valor de  $f'(5)$ ). A la vista de la gráfica, en  $x = 5$  parece haber un mínimo de la función  $f$ . Sin embargo, pidámosle al *Mathematica* que calcule el valor de la segunda derivada de  $f$  en 5. Obtendremos que  $f''(5) = -18 < 0$ . La segunda derivada es negativa y por tanto en  $x = 5$  tiene que haber un máximo.

Sería conveniente ver “más de cerca” lo que ocurre cerca de 5. Prueba a pintar la función en el intervalo  $[4, 6]$ . Ahora parece haber un mínimo cerca de 4,5 y otro cerca de 5,5, y efectivamente, un máximo en 5... ¿cómo podemos asegurar que cerca de los extremos que ahora vemos no vuelven a ocurrir nuevas “oscilaciones ocultas” como la que antes no veíamos?...

---

## Fórmula de Taylor

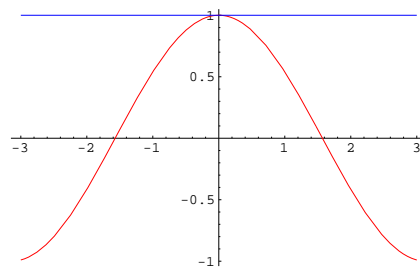
---

Seguro que alguna vez te has preguntado cómo pueden una calculadora o un ordenador usar las funciones seno, coseno, exponencial, logaritmo... que no tienen una definición fácil, y que se basan en números “esencialmente” irracionales como  $\pi$  o  $e$ .

En esta práctica trataremos de entender cómo se pueden construir funciones conociendo únicamente las funciones suma (resta), producto (división) y potenciación. Necesitaremos además comprender el concepto de **límite** y la derivación de funciones. Se trata de “sustituir” funciones complicadas por otras más sencillas de calcular (que en este caso serán polinomios), controlando la “pérdida” que se produce.

El primer ejemplo puede ser “sustituir” una función por su recta tangente (si recuerdas, esto nos sirvió para encontrar ceros de funciones mediante el método de Newton-Raphson): dada una función  $f$  derivable en un punto  $a$ , la recta  $y = f(a) + f'(a)(x - a)$  pasa por el punto  $(a, f(a))$  y tiene la misma pendiente que la función  $f$ . En cierto sentido, lo que hemos hecho es encontrar una función más sencilla (una recta, o sea, un polinomio de grado uno) que se parece a la función de partida (coinciden el valor de la función y el de la derivada). ¿Hasta que punto puedo “cambiar” una función por su recta tangente? Si la función oscila demasiado, la aproximación no será buena, como por ejemplo en la función coseno

En el gráfico podemos ver que en cuanto nos alejamos un poco del punto de tangencia (en este caso el 0), la función coseno y su tangente no se parecen nada. La forma de mejorar la aproximación será aumentar el grado del polinomio que usamos, y el problema es, fijado un grado, qué polinomio de grado menor o igual al fijado es el que más se parece a la función. El criterio con el que elegiremos el polinomio será hacer coincidir las sucesivas derivadas. Por ejemplo, si buscamos un polinomio  $P$  de grado dos que aproxime a una función dos veces derivable,  $f$ , vamos a exigir que  $P(a) = f(a)$ ,  $P'(a) = f'(a)$  y  $P''(a) = f''(a)$ . Con estas tres condiciones estamos en disposición de calcular los coeficientes de  $A(x - a)^2 + B(x - a) + C$ . Si lo hacemos,



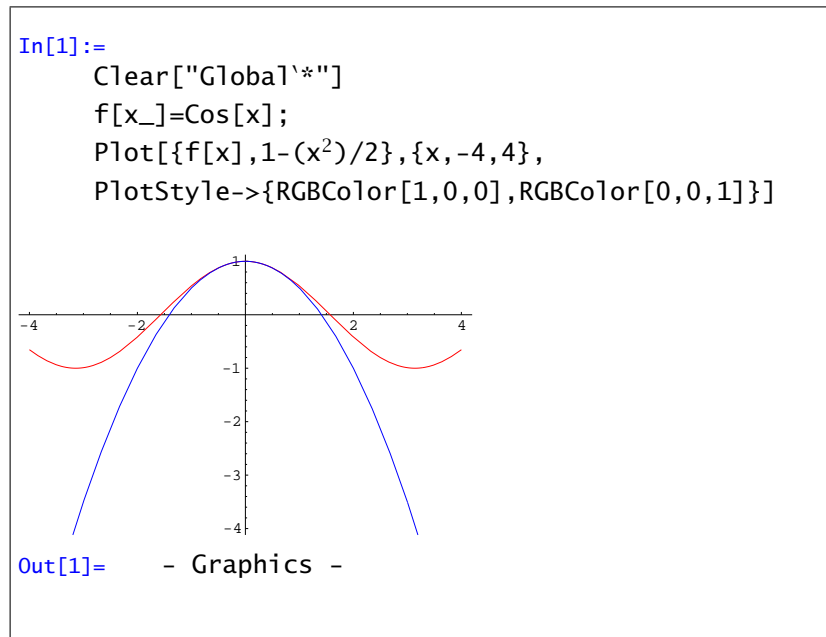
veremos que dicho polinomio “aproximante” de grado 2 es el siguiente (deriva el polinomio  $P$  y verás como sus derivadas hasta orden 2 coinciden con las de la función  $f$ )

$$P(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!} (x - a)^2$$

Veamos qué pasa con una función concreta. Consideremos la función  $f(x) = \cos(x)$  y vamos a calcular el polinomio “aproximante” en 0. En este caso  $f(0) = 1$ ,  $f'(0) = 0$ ,  $f''(0) = -1$ . Por tanto el polinomio “de aproximación” de grado 2 es el siguiente:

$$\begin{aligned} P_2(x) &= f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 \\ &= 1 - \frac{1}{2}x^2 \end{aligned}$$

Si dibujamos ahora las funciones  $f$ ,  $P_1$  y  $P_2$  podremos comprobar qué es lo que ocurre:



El polinomio de orden 2 se parece un poco más a la función coseno que la recta tangente, pero de todas formas, si aumentamos el dominio (prueba por ejemplo  $x \in [-10, 10]$ ) se puede ver que el “parecido” se pierde al alejarnos del origen. El problema es que estamos calculando polinomios de grado bajo.

Todo lo que hemos hecho hasta ahora se basa en un resultado teórico ya conocido: si  $I$  es un intervalo,  $a, x \in I$  y  $f : I \rightarrow \mathbb{R}$  es una función  $n$ -veces derivable, se define el *polinomio de Taylor* de orden  $n$  en el punto  $a$  como

$$\begin{aligned} P_n(x) &= f(a) + f'(a)(x - a) + \frac{f''(a)}{2!} (x - a)^2 + \dots \\ &\quad \dots + \frac{f^{(n)}(a)}{n!} (x - a)^n \end{aligned}$$

**Teorema (Fórmula de Taylor).** Si  $f : I \rightarrow \mathbb{R}$  es una función  $(n + 1)$ -veces derivable, entonces existe  $c$  entre  $a$  y  $x$  tal que

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



El teorema anterior nos permite incluso acotar el error que estamos cometiendo: basta acotar el

llamado *resto de Taylor*,  $\frac{f^{(n+1)}(c)}{(n+1)!}$ , y para ello, acotaremos la derivada de orden  $n+1$  en el intervalo considerado. Volvamos al ejemplo anterior: Si trabajamos en el intervalo  $[-\pi, \pi]$  y queremos usar  $P_2$  en lugar de la función coseno, sabemos que el punto  $c$  que nos da el error también está en ese intervalo. Vamos a calcular el valor máximo que puede alcanzar:

$$\begin{aligned} |\cos(x) - P_2(x)| &= \left| \frac{f'''(c)}{3!} x^3 \right| \\ &= \left| \frac{\text{sen}(c)}{3!} x^3 \right| \\ &\leq \frac{1}{6} \pi^3 \end{aligned}$$

La última acotación se basa en que el valor máximo que alcanza la tercera derivada,  $\text{sen}(x)$ , es 1, y que  $x$  en valor absoluto vale a lo sumo  $\pi$ .

*Mathematica* tiene una orden que permite calcular directamente el polinomio de Taylor centrado en un punto  $a$ . Se trata del comando `Series`, cuya sintaxis es

`Series[función, {x, a, grado}]`

Volviendo al ejemplo anterior, si pruebas

```
In[2]:=
Series[Cos[x], {x, 0, 2}]
Out[2]=
1 -  $\frac{x^2}{2}$  + O[x]3
```

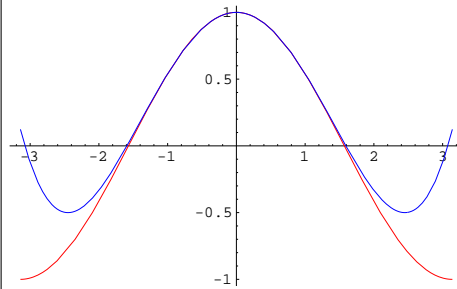
verás como, además del desarrollo del polinomio, aparece al final el orden de error ( $O[x]^3$ ); representa que los términos que faltan son todos de grado mayor o igual que tres. Si quieres usar directamente el resultado (sin que aparezca el orden del error) puedes utilizar el comando

`Normal[expresión]`

Este comando simplemente elimina el orden de error de la expresión que obtenemos con el comando `Series`, quedando sólo el polinomio de Taylor.

En nuestro ejemplo, para comparar la función coseno y el correspondiente polinomio de Taylor de grado 4 en cero haremos lo siguiente:

```
In[3]:=
Clear["Global`*"];
f[x_]=Cos[x];
g[x_]=Normal[Series[f[x],{x,0,4}]];
Plot[{f[x],g[x]},{x,-Pi,Pi},
PlotStyle->{RGBColor[1,0,0],RGBColor[0,0,1]}]
```



```
Out[3]= - Graphics -
```

Ya habíamos visto la orden `Table`. Con ella podemos calcular de una vez el desarrollo de Taylor de varios órdenes y comprobar como se va pareciendo cada vez más a la función. Si escribes

```
In[4]:=
Table[Series[Cos[x],{x,0,n}],{n,1,10}]
```

el resultado son todos los desarrollos de la función coseno centrados en 0 y de orden 1,2,...,10. Si queremos dibujarlos tenemos que quitar los errores y usar la orden `Plot` de la lista que acabamos de hacer. En lo que sigue  $a$  será el punto donde centramos el desarrollo de Taylor,  $[b, c]$  el intervalo donde queremos representarlo, la variable orden será el grado hasta el que queremos llegar y  $f$  la función que vamos a desarrollar:

```
In[5]:=
Clear["Global`*"];
f[x_]=Cos[x];
a=0;
b=-2Pi;
c=2Pi;
orden=10;
Plot[Evaluate[
  Table[Normal[Series[f[x],{x,a,n}]],
    {n,0,orden}]],{x,b,c}
Plot[f[x],{x,b,c},PlotStyle->RGBColor[1,0,0]]
Show[%,%%]
```

La orden `Evaluate[Table...]` que hemos escrito calcula los desarrollos antes de que hagamos el dibujo. Prueba a cambiar la función, el punto donde desarrollamos, el intervalo donde se representa...

Hasta aquí hemos visto como comparar la gráfica de una función con la de su polinomio de Taylor. La orden `Animate` permite dibujar la gráfica de una función en función de un parámetro para después poder reproducirlo como una secuencia de fotogramas (una “película”). Un ejemplo sencillo de como funciona sería el siguiente:

```
In[6]:=
  Animate[Plot[Sin[x*t],{x,0,2
  Pi}],{t,1,10}]
```

```
In[7]:=
  Clear["Global`*"];
  f[x_]=Cos[x]+Sin[x];
  a=0;
  b=-2Pi;
  c=2Pi;
  orden=30;
  Animate[Plot[Evaluate[
    {f[x],Normal[Series[f[x],{x,a,n}]]},
    {x,b,c},
  PlotStyle->{RGBColor[1,0,0],
    RGBColor[0,0,1]},PlotRange->{-3,3}],
  {n,1,orden,1}]
```

Cuando se ejecuta la orden anterior, *Mathematica* dibuja la función y los polinomios de Taylor de orden 1 hasta 30. Prueba a pinchar dos veces en la última gráfica que ha aparecido.

**Ejercicio 1.** Aproxima la función exponencial ( $f(x) = e^x$ ) mediante su polinomio de Taylor de grado 4 centrado en 0 en el intervalo  $[-20, 20]$ . Prueba a aumentar el grado y a usar intervalos cada vez más grandes.

**Ejercicio 2.** Acota el error que estás cometiendo al aproximar  $\sqrt{e} = e^{\frac{1}{2}}$  por el valor obtenido usando el polinomio de Taylor de orden 4 centrado en cero.

**Ejercicio 3.** Lo mismo que los dos ejercicios anteriores para la función logaritmo neperiano (centrando en  $a = 1$ ) y para  $\ln(2)$ .

## 7.1. Integrales definidas e indefinidas

En esta práctica aprenderemos a calcular integrales con el programa *Mathematica*, tanto integrales indefinidas (es decir, primitivas) como integrales definidas. El objetivo es poder calcular áreas entre curvas, áreas y volúmenes de sólidos de revolución, longitudes de curvas... sin que el cálculo de primitivas sea el principal escollo.

Básicamente hay dos comandos para integrar funciones en *Mathematica*: `Integrate` y `NIntegrate`. Como siempre también puedes usar la paleta para escribir las integrales. La primera de las órdenes mencionadas calcula primitivas o integrales definidas en un intervalo  $[a, b]$ :



```
Integrate[función, variable]  
Integrate[función, {variable, a, b}]
```

Por ejemplo,

```
In[1]:=  
Integrate[Cos[x], x]  
Out[1]=  
Sin[x]  
  
In[2]:=  
 $\int$  Cos[x] dx  
Out[2]=  
Sin[x]
```

*Mathematica* puede obtener prácticamente cualquier primitiva calculable por métodos elementales (integración por partes, integración de funciones racionales, trigonométricas, cambio de variable...). Por otro lado, sabemos que toda función continua admite primitiva, pero esta primitiva no siempre puede calcularse (en el sentido de tener una expresión de la primitiva en términos de funciones “elementales”). Por eso, al integrar algunas funciones en el resultado que obtenemos aparecen funciones que no conocemos:

```
In[3]:=

$$\int \frac{\text{Sin}[x]}{x} dx$$

Out[3]=
SinIntegral[x]
```

En este caso, aunque no conozcamos la función que nos aparece, podemos obtener su valor en un punto:

```
In[4]:=
N[SinIntegral[2], 7]
Out[4]=
1.605413
```

Para obtener un valor numérico del resultado se puede usar el comando N:

```
N[Integrate[función, {x, a, b}]]
```

Puede pasar que *Mathematica* no sepa calcular la primitiva de una función:

```
In[5]:=

$$\int \text{Exp}[x^3 + x] dx$$

Out[5]=

$$\int E^{x^3+x} dx$$

```

En este caso siempre podemos intentar aproximar el valor de la integral. Esto es precisamente lo que hace el comando NIntegrate

```
NIntegrate[función, {variable, a, b}]
```

En el ejemplo anterior,

```
In[6]:=
NIntegrate[Exp[x^3+x], {x, 0, 2}]
Out[6]=
1846.36
```

Observa en este ejemplo que no es lo mismo usar el comando `NIntegrate[...]` que `N[Integrate[...]]`. En el segundo caso *Mathematica* tiene que saber calcular una primitiva para después calcular cuanto vale, mientras que el primer comando intenta aproximar el valor de la integral sin calcular una primitiva.

Curiosamente, con funciones a las que podemos calcular fácilmente una primitiva, el comando `NIntegrate` no siempre funciona bien; tecleando

```
In[7]:=
NIntegrate[Cos[x], {x, 0, Pi}]
```

te darás cuenta que *Mathematica* “protesta”. El programa intenta calcular una aproximación con métodos numéricos; estos métodos pueden tener problemas cuando la función oscila mucho. En estos casos pueden aparecer respuestas erróneas.

**Ejercicios.** Calcular:

1. área limitada por las curvas  $y = x^2$  e  $y^2 = 8x$
2. área limitada por  $y = xe^{-x^2}$ , el eje  $OX$ , la ordenada en el punto  $x = 0$  y la ordenada en el máximo.
3. área de la figura limitada por la curva  $y = x^3 - x^2$  y el eje  $OX$ .
4. área comprendida entre la curva  $y = \operatorname{tg}(x)$ , el eje  $OX$  y la recta  $x = \pi/3$ .
5. área del recinto limitado por las rectas  $x = 0$ ,  $x = 1$ ,  $y = 0$  y la gráfica de la función  $f: \mathbb{R} \rightarrow \mathbb{R}$  definida por

$$f(x) = \frac{1}{(1+x^2)^2}.$$

## 7.2. Longitudes, áreas y volúmenes

Algunas de las aplicaciones de la integral que se han visto en las clases teóricas son calcular longitudes de curvas, y volúmenes y superficies de cuerpos de revolución.

### • Volúmenes de sólidos de revolución:

Sea  $f: [a, b] \rightarrow \mathbb{R}$  una función continua; el volumen del sólido generado al girar el área bajo la curva  $y = f(x)$  respecto del eje  $OX$  es

$$V = \pi \int_a^b f(x)^2 dx$$

y el volumen del sólido generado al girar dicha área respecto al eje  $OY$  es

$$V = 2\pi \int_a^b x f(x) dx.$$

### • Longitudes de curvas:

Sea  $f$  una función de clase 1 en el intervalo  $[a, b]$ . La longitud del arco de la curva  $y = f(x)$  entre  $x = a$  y  $x = b$  es

$$l = \int_a^b \sqrt{1 + [f'(x)]^2} dx.$$

• **áreas de sólidos de revolución:**

Sea  $f : [a, b] \rightarrow \mathbb{R}$  una función de clase 1; el área de la superficie generada haciendo girar alrededor del eje  $OX$  el arco de curva  $y = f(x)$  en  $[a, b]$  es igual a

$$S = 2\pi \int_a^b f(x) \sqrt{1 + [f'(x)]^2} dx.$$

**Ejercicio 1.** Calcular el volumen del sólido generado al rotar respecto al eje  $OX$  las siguiente curvas:

1.  $y = \sec(x)$ ,  $x \in [0, \frac{\pi}{3}]$
2.  $y = \sqrt{\cos(x)}$ ,  $x \in [0, \frac{\pi}{2}]$
3.  $y = 9 - x^2$
4.  $y = e^x$ ,  $x \in [0, \ln 3]$

**Ejercicio 2.** Calcular el volumen del sólido generado al rotar respecto al eje  $OY$  las siguiente curvas:

1.  $y = 1/x$ ,  $x \in [1, 3]$
2.  $y = \frac{1}{1 + x^2}$ ,  $x \in [0, 1]$
3.  $y = e^{x^2}$ ,  $x \in [1, \sqrt{3}]$

**Ejercicio 3.** Calcular la longitud de las siguientes curvas:

1.  $y = \frac{1}{3}(x^2 + 2)^{3/2}$ ,  $x \in [0, 1]$
2.  $y = \frac{1}{2}(e^x + e^{-x})$ ,  $x \in [0, 3]$

**Ejercicio 4.**

1. Calcular la superficie de una esfera de radio  $R$ .
2. Calcular la superficie de la figura que se obtiene al girar la función  $y = \operatorname{tg}(x)$ ,  $x \in [0, \pi/4]$  alrededor del eje  $OX$ .

## 7.3. Integrales impropias.

Como recordarás de las clases teóricas, la integral que en principio se define para funciones continuas en intervalos cerrados y acotados, puede extenderse a funciones continuas definidas en intervalos de longitud infinita, y a funciones que no están acotadas en un intervalo de longitud finita. Es lo que se conoce como *Integración impropia*. Con el programa *Mathematica*, el trabajar con integrales impropias no supone ningún problema, ya que las trata exactamente igual que las integrales de funciones continuas en intervalos cerrados y acotados.

Con la misma orden *Integrate* se pueden calcular integrales impropias. Simplemente prueba con una función como  $f(x) = \frac{1}{\sqrt{1-x^2}}$  en el intervalo  $[-1, 1]$  o con la función  $g(x) = e^{-x^2}$  en  $[0, +\infty[$  (en *Mathematica* para escribir infinito usamos `+Infinity` o `-Infinity`, o la paleta). En este segundo caso, escribiríamos

```
In[8]:=
      \int_0^{+\infty} \text{Exp}[-x^2] dx
Out[8]=
      \frac{\sqrt{\pi}}{2}
```

Intenta calcular las integrales anteriores en el intervalo dado y después intenta calcular una primitiva de esas funciones. Como verás, la primitiva de la primera función ya la conocías, en cambio en la primitiva de la segunda aparece una función que, supongo, no conocías.

También podemos usar las fórmulas para calcular longitudes, áreas y volúmenes con integrales impropias.

**Ejercicio.** Calcular:

1. La integral de  $f(x) = \frac{1}{x^2}$  con  $x \in [1, +\infty]$ .
2. El volumen y la superficie lateral del sólido obtenido al girar la gráfica de la anterior función respecto del eje  $OX$ .
3. Ídem a los dos anteriores con  $g(x) = \frac{1}{x}$  con  $x \in [1, +\infty]$ .



## 8.1. El comando Plot3D

En el segundo capítulo pudimos comprobar que *Mathematica* es una herramienta muy potente para “dibujar” gráficas de funciones de una variable. Ahora aprenderemos a representar gráficamente funciones reales de dos variables. La principal aplicación de la representación gráfica de una función de dos variables será dar una idea aproximada de la variación de dicha función, lo que será especialmente útil para buscar extremos.

Al igual que para funciones de una variable, suele ser cómodo asignarle un nombre (normalmente  $f, g, h\dots$ ). Para definir la función  $\sin(xy)$  haremos lo siguiente:

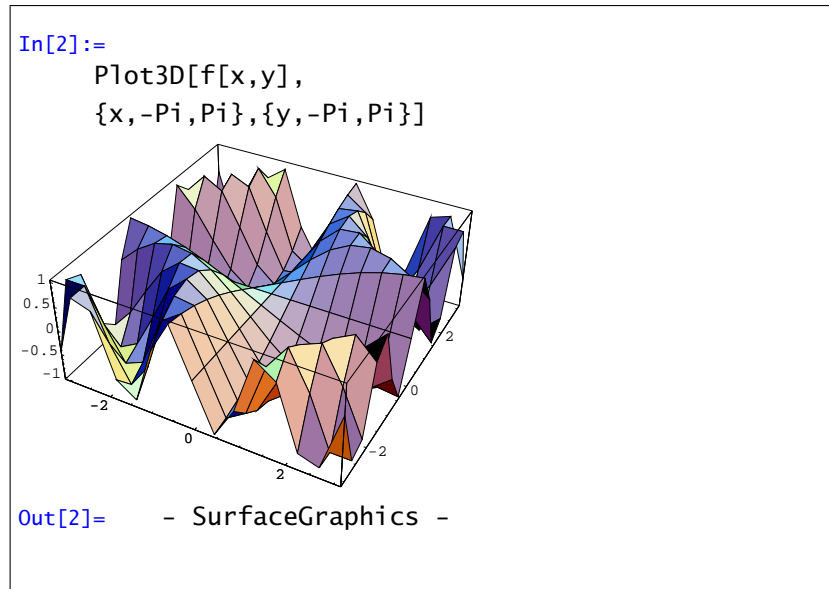
```
In[1]:=
  f[x_,y_]=Sin[x*y]
Out[1]=
  Sin[x y]
```

y usaremos  $f$  de igual forma a como lo hacíamos con funciones de una variable.

El comando que sirve para representar gráficas de funciones de dos variables es

```
Plot3D[función de  $x, y$ , { $x, x_{\min}, x_{\max}$ }, { $y, y_{\min}, y_{\max}$ }]
```

Veamos un ejemplo:



### 8.1.1. Opciones del comando Plot3D

- Mesh->True/False

Se dibuja (True) o no (False) la retícula o malla sobre la que se construye la gráfica. En el ejemplo anterior:

```
In[3]:=
Plot3D[f[x,y],{x,-Pi,Pi},
{y,-Pi,Pi},Mesh->False]
```

- Shading->True/False

Se colorea (True) o no (False) la malla anterior. Si unimos las opciones Mesh->False y Shading->False, no aparecerá gráfico alguno. Un ejemplo:

```
In[4]:=
Plot3D[f[x,y],{x,-Pi,Pi},
{y,-Pi,Pi},Shading->False]
```

- PlotPoints->número de puntos

Representa el número de puntos que usará *Mathematica* para dibujar la gráfica. Un número muy alto producirá un gráfico más “suave”, pero aumentará considerablemente el tiempo empleado por *Mathematica* para realizarlo.

- ViewPoint

Establece el punto de vista desde el que se dibujará la gráfica. La forma de usar esta opción será “pegarlo” desde una ventana que automatiza el proceso. A dicha ventana se accede a través del menú Input, submenú 3D ViewPoint Selector.

**Ejercicio.** Prueba las opciones anteriores con las siguientes funciones de dos variables:

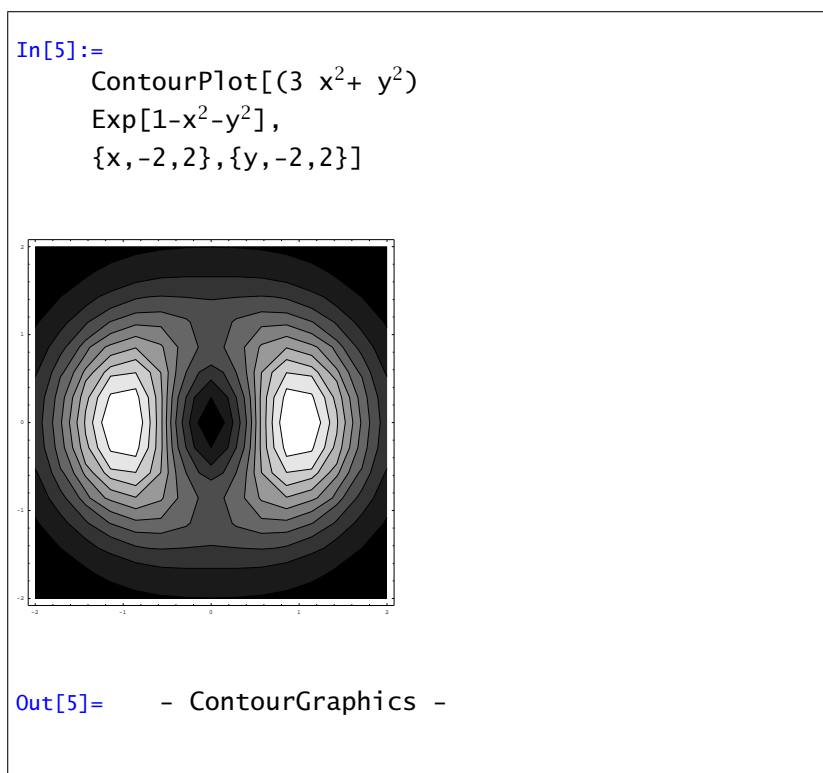
1.  $f(x, y) = (x^2 + y^4)e^{1-x^2-y^2}$  con  $x, y \in [-2, 2]$ .
2.  $g(x, y) = \text{sen}(x - 2xy)$  con  $x \in [0, \pi], y \in [-\pi, \pi]$ .
3.  $h(x, y) = \frac{xy}{x^2 + y^2}$  con  $x, y \in [-1, 1]$ .

## 8.2. Gráficos de contorno. Curvas de nivel.

Una forma cómoda de ver la gráfica de una función es mediante curvas de nivel. Se trata de representar en el plano una figura tridimensional como es la gráfica de una función de dos variables, permitiendo ver las zonas de crecimiento y decrecimiento de dicha función. El comando que representa funciones de dos variables mediante curvas de nivel es

```
ContourPlot[función de x,y, {x, xmin, xmax}, {y, ymin, ymax}]
```

Por ejemplo,



Entre otras, las opciones del comando ContourPlot son:

- ContourShading->False

Dibuja sólo las curvas de nivel y no utiliza la escala de grises. Ejemplo:

```
In[6]:=
ContourPlot[(3 x^2 + y^2)Exp[1-x^2-y^2],
{x, -2, 2}, {y, -2, 2}, ContourShading->False]
```

■ PlotPoints->número de puntos

Representa el número de puntos que usará *Mathematica* para dibujar la gráfica. Al igual que en Plot3D, un número muy alto producirá un gráfico más “suave”, pero aumentará considerablemente el tiempo empleado por *Mathematica* para realizarlo.

**Ejercicio 1.** Representa los gráficos de contorno de las funciones del ejercicio anterior, comparando la información que obtienes de una y otra forma.

**Ejercicio 2.** Consideremos la función  $f : [-2, 2] \times [-2, 2] \rightarrow \mathbb{R}$  dada por  $f(x, y) = (x^2 + y^4)e^{1-x^2-y^2}$ . Utiliza los comandos anteriores para obtener información sobre la ubicación de sus posibles extremos relativos.

### 8.3. Gráficos paramétricos. Curvas y superficies.

Ya aprendimos a representar gráficamente curvas planas expresadas en forma paramétrica. Ahora veremos como representar curvas alabeadas (esto es, curvas en el espacio) y superficies paramétricas. El comando de *Mathematica* para ello es ParametricPlot3D.

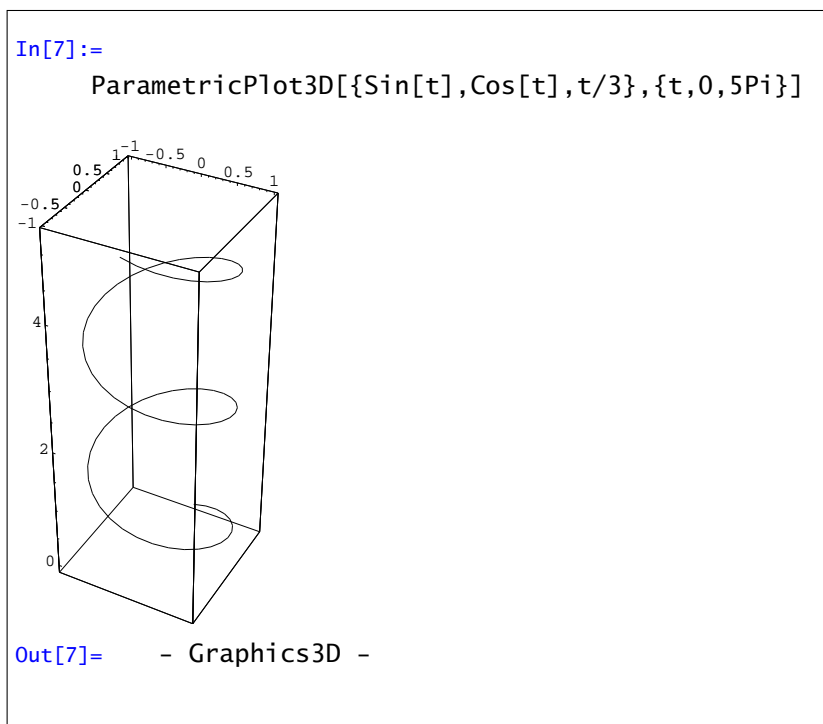
Cuando tenemos las ecuaciones paramétricas de una curva en el espacio

$$x = x(t), y = y(t), z = z(t) \text{ con } t \in [a, b]$$

usamos la orden

```
ParametricPlot3D[{x(t), y(t), z(t)}, {t, a, b}]
```

Por ejemplo, para dibujar una hélice:



Si lo que tenemos son las ecuaciones paramétricas de una superficie

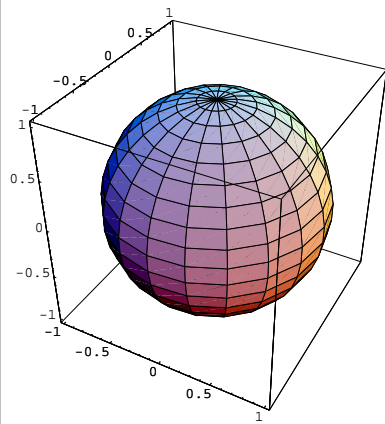
$$x = x(u, v), y = y(u, v), z = z(u, v) \quad \text{con } u \in [a, b] \text{ y } v \in [c, d]$$

se usa

```
ParametricPlot3D[{x(u,v),y(u,v),z(u,v)},{u,a,b},{v,c,d}]
```

Dibujemos una esfera:

```
In[8]:=
ParametricPlot3D[{Cos[u]Cos[v],
Sin[u]Cos[v],Sin[v]},
{u,0,2 Pi},{v,-Pi/2,Pi/2}]
```



```
Out[8]= - Graphics3D -
```

### 8.3.1. Superficies de revolución

En el caso particular en que la superficie que queremos representar se obtiene girando una curva respecto a un eje se puede usar el paquete `SurfaceOfRevolution`. Para cargarlo se escribe

```
In[9]:=
<<Graphics`SurfaceOfRevolution`
```

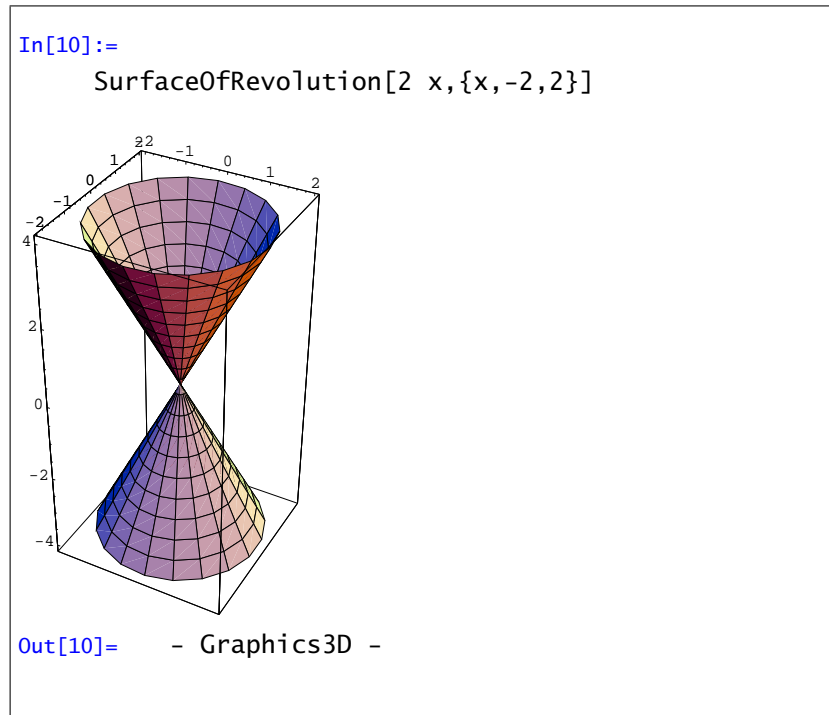
y desde este momento está disponible la orden

```
SurfaceOfRevolution[f(x),{x,a,b}]
SurfaceOfRevolution[{f(t),g(t)},{t,a,b}]
SurfaceOfRevolution[{f(t),g(t),h(t)},{t,a,b}]
```

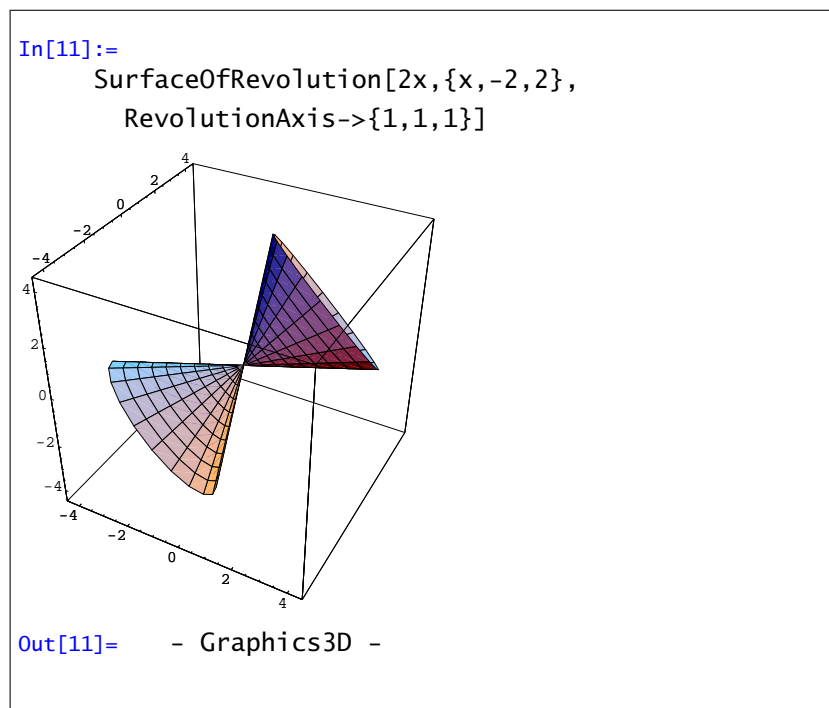
La primera orden dibuja la superficie de revolución obtenida al girar la curva  $f$  en el plano  $XZ$  entre  $a$  y  $b$ . La segunda y la tercera funcionan análogamente pero estamos dando una curva en el plano  $XZ$  (segundo caso) o en el espacio (último caso). En cualquier caso el giro se hace respecto al eje  $Z$ . Si se quiere girar respecto a la recta que une el origen con el punto  $\{x_1, x_2, x_3\}$  se puede utilizar la siguiente opción:

```
SurfaceOfRevolution[f(t),{t,a,b},
  RevolutionAxis->{x1,x2,x3}]
```

Veamos algunos ejemplos. Un cono se consigue fácilmente girando una recta:

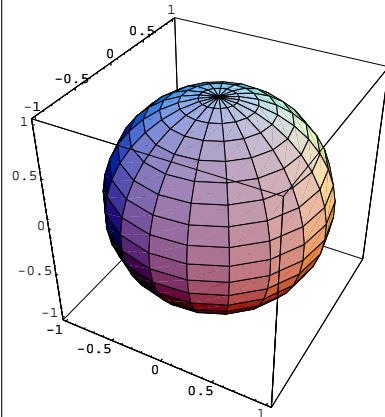


Si quisiéramos girar respecto a la bisectriz del primer octante, o sea, alrededor de la recta que pasa por el vector  $(1,1,1)$ :



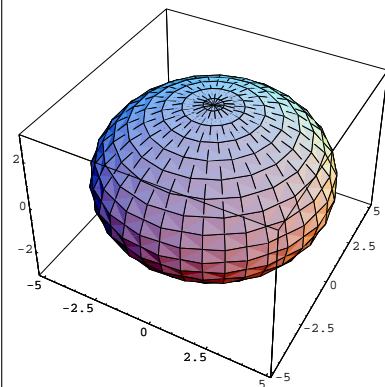
Dibujar una esfera o un elipsoide es ahora simplemente cuestión de girar media circunferencia o media elipse:

```
In[12]:=
SurfaceOfRevolution[{Cos[x],Sin[x]},{x,-Pi,Pi},
AspectRatio->1]
```



```
Out[12]= - Graphics3D -
```

```
In[13]:=
SurfaceOfRevolution[{5Cos[x],3Sin[x]},{x,-Pi,Pi},
AspectRatio->1]
```



```
Out[13]= - Graphics3D -
```

Por último comentar que, como habrás visto en el último ejemplo, la mayoría de las opciones del comando `Plot3D` siguen siendo válidas.

**Ejercicio 1.** Representar las siguientes curvas y superficies:

1.

$$\begin{cases} x(t) = \text{sen}(t) \\ y(t) = \text{sen}(2t) \\ z(t) = t/5 \end{cases} \quad \text{con } t \in [0, 5\pi].$$

2.

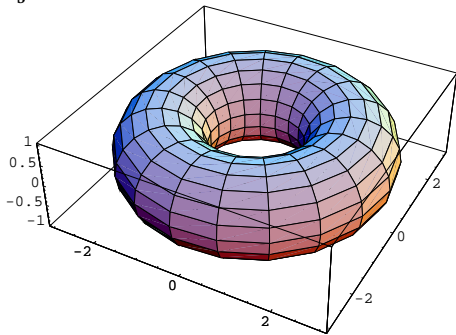
$$\begin{cases} x(u, v) = \cos(u)(3 + \cos(v)) \\ y(u, v) = \operatorname{sen}(u)(3 + \cos(v)) \\ z(u, v) = \operatorname{sen}(v) \end{cases} \quad \text{con } u, v \in [0, 2\pi].$$

3.

$$\begin{cases} x(u, v) = \operatorname{sen} u \\ y(u, v) = \operatorname{sen} v \\ z(u, v) = v \end{cases} \quad \text{con } u \in [-\pi, \pi] \text{ y } v \in [0, 5].$$

**Ejercicio 2.** ¿Cómo se puede dibujar un cilindro? ¿Y un cono truncado?.

**Ejercicio 3.** A la figura siguiente, obtenida al girar una circunferencia, se la llama toro. Intenta dibujarla.





---

## Extremos relativos y condicionados

---

En esta práctica veremos cómo calcular extremos relativos y condicionados de funciones de varias variables. Se trata de usar el programa *Mathematica* como apoyo para realizar los cálculos necesarios en los métodos de optimización vistos en las clases teóricas.

### 9.1. Derivadas parciales

El primer objetivo será aprender a calcular derivadas parciales, gradientes, hessianos... de funciones de varias variables, y aplicar todo ello al cálculo de extremos relativos y condicionados.

Ya conocemos de prácticas anteriores la forma de calcular derivadas de funciones de una variable: mediante el comando `D[f[x], x, n]` obtenemos la derivada  $n$ -ésima de la función  $f$ . Para funciones escalares de varias variables (esto es, funciones que “salen” de  $\mathbb{R}^n$  y “llegan” a  $\mathbb{R}$ ), el comando

`D[f[x, y, ...], {x, n1}, {y, n2}, ...]`

nos devuelve la derivada parcial de  $f$  respecto de  $x$   $n1$ -veces, respecto de  $y$   $n2$ -veces...pudiendo omitirse el número de veces si éste es 1. También puede usarse la paleta. En la figura tienes los símbolos que nos permiten escribir derivadas de cualquier orden y derivadas de segundo orden.

Calculemos como ejemplo  $\frac{\partial^3 f}{\partial x^2 \partial y}$  y  $\frac{\partial^2 f}{\partial x \partial y}$ , donde  $f(x, y) = \sin x \cos y^2$ :



```
In[1]:=
  f[x_,y_]=Sin[x] Cos[y^2];
  ∂x,x,yf[x,y]
  ∂x,yf[x,y]
Out[1]=
  2 y Sin[x]Sin[y^2]
  -2 y Cos[x] Sin[y^2]
```

Evidentemente esta no es la forma más cómoda de calcular una derivada parcial de orden 15. Para eso es mejor utilizar  $D[f[x, y], \{x, 8\}, \{y, 7\}]$ .

Por ejemplo para calcular el gradiente de una función de varias variables basta construir el vector formado por las derivadas parciales primeras de la función respecto de todas las variables:

```
In[2]:=
  J[x_,y_]={∂xf[x,y],∂yf[x,y]}
Out[2]=
  {Cos[x]Cos[y^2], -2 y Sin[x] Sin[y^2]}
```

Usando la paleta, el hessiano se calcula de forma análoga:

```
In[3]:=
  ( ∂x,xf[x,y] ∂x,yf[x,y] )
  ( ∂y,xf[x,y] ∂y,yf[x,y] )
```

Recuerda que las derivadas cruzadas de segundo orden coinciden para funciones “suficientemente buenas”.

## 9.2. Representación gráfica

Ya que sabemos cómo se calculan las derivadas parciales de una función, vamos a intentar representar gráficamente el plano tangente y cómo se obtiene a partir de las derivadas parciales.

Comencemos, por ejemplo, con la función

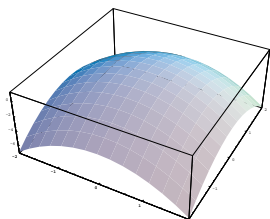
```
In[4]:=
  f[x_,y_]=1-(x^2+y^2)
```

y vamos a calcular en el punto (1,1) sus derivadas parciales. La definición de derivada parcial respecto a la primera variable era

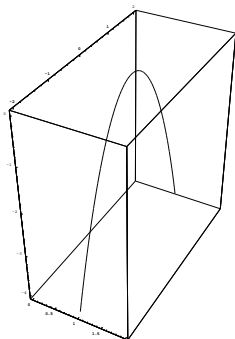
$$\frac{\partial f}{\partial x}(1,1) = \lim_{h \rightarrow 0} \frac{f(1+h,1) - f(1)}{h}$$

Lo que hacemos es trabajar únicamente con la función definida sobre la recta que pasa por (1,1) y es esa función (de una variable) la que derivamos. La segunda derivada parcial tiene una definición análoga. Dibujemos la gráfica de la función y la imagen de cada una de esas dos rectas:

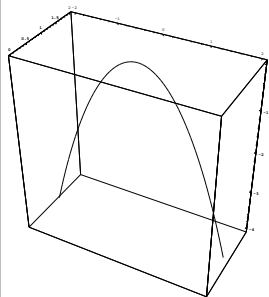
```
In[5]:=
Plot3D[f[x,y],{x,-2,2},{y,-2,2},Mesh->False]
ParametricPlot3D[{1,t,f[1,t]},{t,-2,2}]
ParametricPlot3D[{t,1,f[t,1]},{t,-2,2}]
Show[%%,%%,%]
```



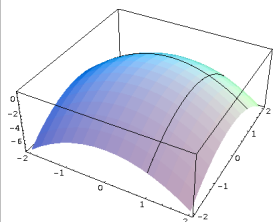
```
Out[5]= - Graphics3D -
```



```
Out[6]= - Graphics3D -
```



```
Out[7]= - Graphics3D -
```



```
Out[8]= - Graphics3D -
```

Los vectores tangentes a las dos curvas que hemos dibujado son los dos vectores que generan el plano tangente. Por tanto su ecuación será

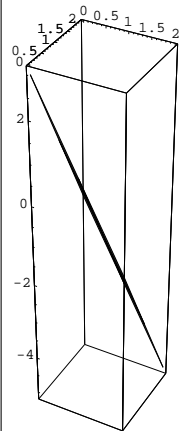
$$z = f(1,1) + \frac{\partial f}{\partial x}(1,1)(x-1) + \frac{\partial f}{\partial y}(1,1)(y-1)$$

Para acabar nos falta dibujar el plano tangente y unirlo con la gráfica anterior:

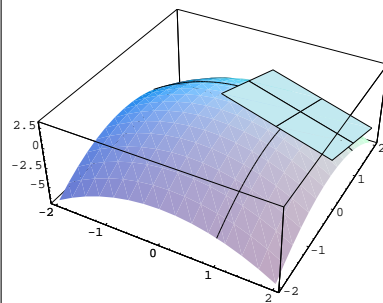
```
In[9]:=
parcial1[x_,y_]=D[f[x,y],x]
parcial2[x_,y_]=D[f[x,y],y]
ParametricPlot3D[{x,y,f[1,1]+
parcial1[1,1](x-1)+
parcial2[1,1](y-1)},
{x,0,2},{y,0,2},PlotPoints->3]
Show[%%%,%]
```

```
Out[9]= -2x
```

```
Out[10]= -2y
```



```
Out[11]= - Graphics3D -
```



```
Out[12]= - Graphics3D -
```

**Ejercicio.** ¿Cómo se podría hacer esto con derivadas direccionales en lugar de derivadas parciales?

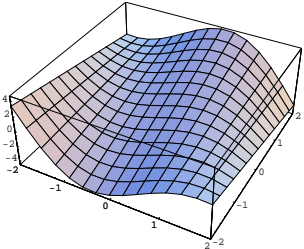
### 9.2.1. “Campos” de vectores

Sabíamos que el gradiente de una función en cada punto era el vector formado por las derivadas parciales e indicaba la dirección de máxima pendiente. Recordemos que el comando `ContourPlot` dibujaba las curvas de nivel de la gráfica de una función, por tanto el gradiente debería ser perpendicular a estas. Para verlo necesitamos algunos comandos que podemos usar cargando el paquete `PlotField`:

```
In[13]:=
  <<Graphics`PlotField`

In[14]:=
  f[x_, y_]:=2(x+y)Cos[x]
Out[14]=
  2(x+y)Cos[x]

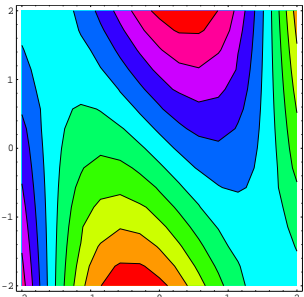
In[15]:=
  Plot3D[f[x,y], {x, -2, 2}, {y, -2, 2}]
```



```
Out[15]= - Graphics -
```

Dibujemos ahora las curvas de nivel. La opción `ColorFunction->Hue` utiliza colores para diferenciar las distintas alturas en lugar de la escala de grises que habíamos visto.

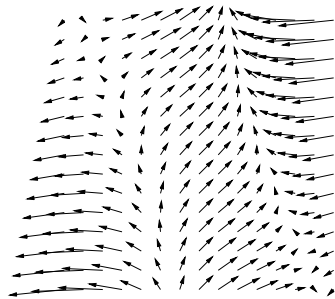
```
In[16]:=
  ContourPlot[f[x,y], {x, -2, 2}, {y, -2, 2},
  ColorFunction->Hue]
```



```
Out[16]= - Graphics -
```

La orden `PlotGradientField` dibuja los vectores  $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ . Las dos opciones que aparecen (`ScaleFunction` y `ScaleFactor`) no son necesarias.

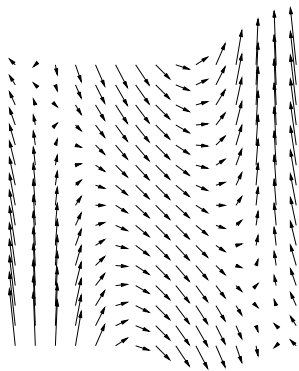
```
In[17]:=
PlotGradientField[f[x,y],{x,-2,2},{y,-2,2},
ScaleFunction->(.#&),
ScaleFactor->None]
```



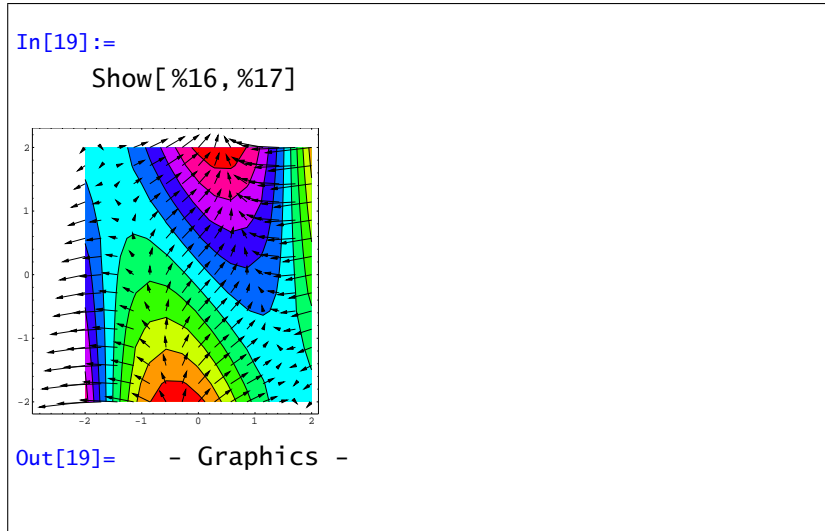
```
Out[17]= - Graphics -
```

También existe la orden `PlotHamiltonianField` que dibuja los vectores  $(-\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x})$ . Estos son perpendiculares al gradiente y por tanto nos indican las curvas de nivel.

```
In[18]:=
PlotHamiltonianField[f[x,y],{x,-2,2},{y,-2,2},
ScaleFunction->(.#&),
ScaleFactor->None]
```



```
Out[18]= - Graphics -
```



**Ejercicio 1.** Calcular las derivadas parciales de:

1.  $f(x, y, z) = x^{y+z}, \forall x \in \mathbb{R}^+, y, z \in \mathbb{R}$
2.  $f(x, y, z) = (x + y)^z, \forall x, y \in \mathbb{R}^+, z \in \mathbb{R}$
3.  $f(x, y) = \text{sen}(x\text{sen}y), \forall x, y \in \mathbb{R}$

**Ejercicio 2.** Sea  $f : \mathbb{R}^2 \setminus \{(0, 0)\} \rightarrow \mathbb{R}$  dada por  $f(x, y) = \log(x^2 + y^2)$  para todo  $(x, y) \neq (0, 0)$ . Se pide:

- (I) Calcúlese el gradiente de  $f$  en todo punto así como la matriz hessiana.
- (II) Compruébese que

$$\frac{\partial^2 f}{\partial x^2}(x, y) + \frac{\partial^2 f}{\partial y^2}(x, y) = 0 \quad \forall (x, y) \in \mathbb{R}^2 - \{(0, 0)\}.$$

**Ejercicio 3.** Sea  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  una función diferenciable con continuidad. Si  $v = f(x, y)$  donde  $x = \rho \cos \theta, y = \rho \text{sen} \theta$ , comprobar que

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 v}{\partial \rho^2} + \frac{1}{\rho^2} \frac{\partial^2 v}{\partial \theta^2} + \frac{1}{\rho} \frac{\partial v}{\partial \rho}$$

### 9.3. Extremos relativos.

El método para encontrar extremos relativos de funciones de varias variables consiste en buscar primero los puntos críticos, es decir, puntos donde se anula el gradiente, y después estudiar la matriz hessiana en esos puntos. Los resultados que conocemos nos aseguran que todos los puntos extremos de una función están entre los puntos críticos, con lo que una vez calculados éstos nos dedicaremos a estudiar la matriz hessiana en ellos, viendo si es definida, indefinida, semidefinida... Para ello podemos usar el criterio de los valores propios: si todos son del mismo signo, la matriz es definida y hay extremo; si aparecen valores propios de distinto signo es indefinida y hay punto de silla; en otro caso, la matriz es semidefinida y el criterio no decide.

**Ejemplo 1:** Calculemos los extremos relativos de la función

$$f(x, y) = x^3 + 3xy^2 - 15x - 12y.$$

Primero definimos la función, su gradiente y su hessiano:

```
In[20]:=
f[x_,y_]=x3 + 3 x y2 -15 x -12 y
z=f[x,y]

In[21]:=
J[x_,y_]={∂xz, ∂yz}
H[x_,y_]= $\begin{pmatrix} \partial_{x,x}z & \partial_{x,y}z \\ \partial_{y,x}z & \partial_{y,y}z \end{pmatrix}$ 
```

Para calcular los puntos críticos podemos usar el comando `Solve`, ya que el gradiente está formado por polinomios de grado bajo; guardaremos el resultado que nos de `Solve` en una variable, `pcrit`, que mas tarde usaremos:

```
In[22]:=
pcrit=Solve[J[x,y]==0,{x,y]}

Out[22]=
{{x ->-2, y ->-1},{x ->-1, y ->-2},
{x ->1, y ->2},{x ->2, y ->1}}
```

Hemos obtenido así 4 puntos críticos entre los que estarán los extremos. Calculemos ahora los valores propios de la matriz hessiana en los cuatro puntos críticos obtenidos (observa la utilización del comando `/.` que ahora comentaremos)

```
In[23]:=
Eigenvalues[H[x,y]] /. pcrit

Out[23]=
{{-6, -18}, {6,-18}, {-6, 18}, {6, 18}}
```

El comando `/.` sirve para asignar valores; en el caso anterior, asignamos la lista de valores que hay en `pcrit` (que son los puntos críticos de  $f$ ) al comando `Eigenvalues[H[x,y]]`, que calcula los valores propios de la matriz hessiana en un punto  $(x, y)$ . De esta forma, lo que hemos obtenido es una lista formada por cuatro parejas de valores propios de  $H[x, y]$  correspondientes a los cuatro puntos críticos. Así, la matriz hessiana en el primer punto crítico,  $(-2, -1)$ , tiene sus dos valores propios negativos, con lo que es definida negativa y por tanto hay *máximo*; los puntos críticos 2.º y 3.º,  $(-1, -2)$  y  $(1, 2)$ , son *puntos de silla*, ya que sobre ellos la matriz hessiana es indefinida; por último, el 4.º es un punto de *mínimo*. Si queremos saber lo que vale la función  $f$  sobre sus puntos críticos, basta teclear



```
In[24]:=
  f[x,y]/.pcrit
Out[24]=
  {28, 26, -26, -28}
```

que nos da una lista con los valores de  $f$  en los puntos críticos, o bien escribir directamente  $f[-2, -1]$ ,  $f[-1, -2]$ , etc...

Podemos resumir diciendo que la función  $f$  tiene máximo valor relativo 28 en el punto  $(-2, -1)$ , mínimo valor relativo -28 en  $(2, 1)$ , y dos puntos de silla en  $(-1, -2)$  y  $(1, 2)$ .

**Ejemplo 2:** Extremos relativos de

$$f(x, y) = (x^2 + 3y^2) e^{1-x^2-y^2}.$$

Comenzamos definiendo  $f$ , y calculando el gradiente y la matriz hessiana:

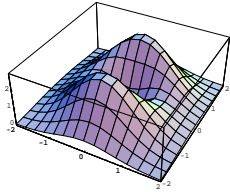
```
In[25]:=
  Clear["Global`*"]
  f[x_,y_]=(x^2+ 3 y^2) Exp[1-x^2-y^2];
  z=f[x,y]

In[26]:=
  J[x_,y_]={∂xz, ∂yz}
  H[x_,y_]= $\begin{pmatrix} \partial_{x,x}z & \partial_{x,y}z \\ \partial_{y,x}z & \partial_{y,y}z \end{pmatrix}$ 
```

Si intentamos calcular los puntos críticos resolviendo el sistema  $J[x, y] == 0$  con la orden `NSolve`, *Mathematica* informa que no puede resolverlo, pues aparecen exponenciales, y `NSolve` (igual que `Solve` o `Reduce`) sólo sirven para polinomios y funciones sencillas. En este punto, tenemos dos posibilidades. La primera, trazar la gráfica de  $f$  (como superficie o con curvas de nivel) para tener una idea de dónde están los extremos, y usar el comando `FindRoot` con las aproximaciones iniciales que nos da la gráfica.

In[27]:=

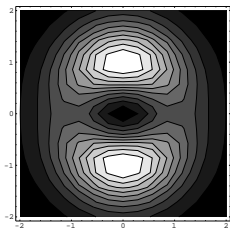
```
Plot3D[f[x,y],{x,-2,2},{y,-2,2}]
```



Out[27]= - SurfaceGraphics -

In[28]:=

```
ContourPlot[f[x,y],{x,-2,2},{y,-2,2}]
```



Out[28]= - ContourGraphics -

A la vista de las gráficas podemos calcular aproximaciones a los extremos y calcularlos con `FindRoot`, pero ¿qué nos asegura que no hay más extremos que no entran en el rango de la función, o que no vemos por la imprecisión de los gráficos?.

La segunda posibilidad es realizar operaciones algebraicas en el sistema de ecuaciones  $J[x, y] = 0$ , para convertirlo en un sistema polinómico. Esto no siempre puede hacerse, pero en este caso sí: basta multiplicar el sistema por  $e^{-(1-x^2-y^2)}$  (que no se anula), y resolver el nuevo sistema que es polinómico:

In[29]:=

```
G[x_,y_]=Simplify[J[x,y]*Exp[x^2+y^2-1]]
```

Out[29]=

```
{2x(1-x^2-3y^2), 2y(3-x^2-3y^2)}
```

Las soluciones de  $G[x, y] = 0$  serán los puntos críticos de  $f$ , que almacenaremos en la variable `pcrit`:

In[30]:=

```
pcrit=Solve[G[x,y]==0,{x,y}]
```

Out[30]=

```
{{x ->-1, y ->0}, {x ->0, y ->0}, {x  
->1, y ->0}, {y ->-1, x ->0}, {y ->1, x  
->0}}
```

Estudiamos el hessiano en los cinco puntos:

```
In[31]:=
  Eigenvalues[H[x,y]]/.pcrit
Out[31]=
  {{-4, 4}, {2 E, 6 E},{-4, 4}, {-12,
  -4}, {-12, -4}}
```

Tenemos dos puntos de máximo:  $(0, -1)$  y  $(0, 1)$ ; un punto de mínimo:  $(0, 0)$ ; y dos puntos de silla:  $(-1, 0)$  y  $(1, 0)$ . La función  $f$  vale sobre sus puntos críticos

```
In[32]:=
  f[x,y]/.pcrit
Out[32]=
  {1, 0, 1, 3, 3}
```

con lo que, vale 3 en los dos máximos y 0 en el mínimo.

**Ejercicio.** Calcular los extremos relativos de las siguientes funciones.

1.  $f(x, y) = x^2 + y^2 - 2x + 4y + 20$ .
2.  $f(x, y) = x^3 + x^2y + y^2 + 2y + 5$ .
3.  $f(x, y) = (x^2 + y^2) e^{x^2 - y^2}$ .
4.  $f(x, y, z) = x^2 + y^2 + z^2 - 2x$ .
5.  $f(x, y) = x^3y^3 - y^4 - x^4 + xy$ .

## 9.4. Extremos condicionados.

Usaremos el método de los multiplicadores de Lagrange para calcular extremos condicionados. Se trata de optimizar una función de varias variables en el conjunto de puntos que verifiquen una cierta ecuación o ecuaciones.

Dada una función suficientemente diferenciable, y una curva o superficie en forma implícita, el método consiste en encontrar los puntos de dicha curva o superficie que verifican un sistema auxiliar, llamado “sistema de Lagrange”, que dará el equivalente a los puntos críticos; después habrá una condición sobre el hessiano equivalente a la condición para extremos relativos. Hay que notar que no todas las condiciones son válidas, sino que se ha de imponer una hipótesis técnica, pero en la práctica supondremos hecha la comprobación.

**Ejemplo 1:** Calculemos los extremos de la función

$$f(x, y) = x^2 - y^2$$

condicionados a la ecuación

$$g(x, y) = x^2 + y^2 - 1 = 0.$$

Comenzamos definiendo la función  $f$ , la condición, y la función auxiliar de Lagrange

```
In[33]:=
f[x_,y_]=x^2-y^2;
g[x_,y_]=x^2+y^2-1;
F[x_,y_,lambda_]=f[x,y]-lambda*g[x,y]
z=F[x,y,lambda]
```

Definimos gradiente y hessiano de la función de Lagrange

```
In[34]:=
J[x_,y_,lambda_]={∂xz, ∂yz};
H[x_,y_]= $\begin{pmatrix} \partial_{x,x}z & \partial_{x,y}z \\ \partial_{y,x}z & \partial_{y,y}z \end{pmatrix}$ ;
```

y resolvemos el sistema de Lagrange, esto es,  $J[x, y, \lambda] == 0$  y  $g[x, y] == 0$ . Como estamos con ecuaciones polinómicas, usaremos `Solve`:

```
In[35]:=
pcrit=Solve[{J[x,y,lambda]==0,
g[x,y]==0},{x,y,lambda}]
Out[35]=
{{lambda ->-1, y ->-1, x ->0}, {lambda
->-1, y ->1, x ->0}, {lambda ->1, x
->-1, y ->0}, {lambda ->1, x ->1, y
->0}}
```

Obtenemos 4 puntos críticos, y para ver si son o no extremos, estudiamos la matriz hessiana auxiliar  $H[x, y, \lambda]$  restringida al espacio tangente a la curva en los puntos críticos. Desafortunadamente, no podemos aplicar de una vez la asignación de valores, sino que deberemos ir punto a punto

```
In[36]:=
G[x_,y_]={{∂xg[x,y], ∂yg[x,y]}};
In[37]:=
nucleo=NullSpace[G[x,y]/. pcrit[[1]]]
Out[37]=
{{1,0}}
```

(observa que `pcrit[[1]]` nos da el primer punto crítico)

```
In[38]:=
nucleo.(H[x,y,lambd]/.pcrit[[1]]).
Transpose[nucleo]
Out[38]=
{{4}}
```

luego el primer punto es un mínimo condicionado;

```
In[39]:=
nucleo=NullSpace[G[x,y]/.pcrit[[2]]]
Out[39]=
{{1,0}}

In[40]:=
nucleo.(H[x,y,lambd]/.pcrit[[2]]).
Transpose[nucleo]
Out[40]=
{{4}}
```

el segundo punto también es mínimo;

```
In[41]:=
nucleo=NullSpace[G[x,y]/.pcrit[[3]]]

In[42]:=
nucleo.(H[x,y,lambd]/.pcrit[[3]]).
Transpose[nucleo]
Out[42]=
{{-4}}
```

el tercer punto es un máximo;

```

In[43]:=
nucleo=NullSpace[G[x,y]/. pcrit[[4]]]
Out[43]=
{{0,1}}

In[44]:=
nucleo.(H[x,y,lambda]/.pcrit[[4]]).
Transpose[nucleo]
Out[44]=
{{-4}}

```

otro máximo.

El valor de la función en los extremos:

```

In[45]:=
f[x,y]/.pcrit
Out[45]=
{-1, -1, 1, 1}

```

esto es, vale 1 en los máximos y -1 en los mínimos.

**Ejemplo 2:** Calcular los extremos relativos de

$$f(x, y, z) = x^2 + y^2 + z^2$$

condicionados a la superficie

$$g(x, y, z) = 2x + 2y - z + 3 = 0.$$

Comenzamos con las definiciones:

```

In[46]:=
f[x_,y_,z_]=x^2+ y^2+z^2;
g[x_,y_,z_]=2x+2y-z+3;
F[x_,y_,z_,lambda_]=f[x,y,z]-lambda*g[x,y,z];
k1=F[x,y,z,lambda];
J[x_,y_,z_,lambda_]={∂xk1,∂yk1,∂zk1};
H[x_,y_,z_,lambda_]={∂x,xk1,∂x,yk1,∂x,zk1},
{∂y,xk1,∂y,yk1,∂y,zk1},{∂z,xk1,∂z,yk1,∂z,zk1}};

```

Resolvemos el sistema de Lagrange

```
In[47]:=
pcrit=Solve[{J[x,y,z,lambda]==0,
g[x,y,z]==0},{x,y,z,lambda]}
Out[47]=
{{x ->-(2/3), y ->-(2/3), z ->(1/3),
lambda ->-(2/3)}}
```

y estudiemos el hessiano de la función auxiliar en el espacio tangente a la superficie. Primero calculamos una base de dicho tangente con el comando `NullSpace`:

```
In[48]:=
k2=g[x,y,z];
G[x_,y_,z_]={{∂xk2,∂yk2,∂zk2}};
nucleo=NullSpace[G[x,y,z]/.
pcrit[[1]]]
Out[48]=
{{1, 0,2}, {-1, 1, 0}}
```

Ahora calculamos la matriz hessiana restringida al tangente

```
In[49]:=
M=nucleo.
(H[x,y,z,lambda]/.pcrit[[1]]).
Transpose[nucleo]
Out[49]=
{{10, -2}, {-2, 4}}
```

y estudiamos si es definida mediante sus valores propios

```
In[50]:=
N[Eigenvalues[M]]
Out[50]=
{3.39445, 10.6056}
```

Como ambos son positivos, el único punto crítico calculado es un mínimo condicionado, con valor

```
In[51]:=
f[x,y,z]/.pcrit
Out[51]=
{1}
```

**Ejercicio 1:** Calcular los extremos condicionados en los siguientes casos:

1.  $f(x, y) = x^2 - xy + y^2$  condicionados a  $x^2 + y^2 - 4 = 0$ .
2.  $f(x, y) = x^3 + xy^2$  condicionados a  $xy = 1$ .
3.  $f(x, y, z) = xyz$  condicionados a  $x^2 + y^2 + z^2 = 1$ .

**Ejercicio 2:**

1. Hallar la mínima distancia de  $(0, 0)$  a  $x^2 - y^2 = 1$ .
2. Entre todos los ortoedros de volumen 1, determinar el que tiene superficie lateral mínima.
3. Calcular las dimensiones de un ortoedro de superficie lateral 2, para que su volumen sea máximo
4. Se quiere construir un canal cuya sección sea un trapecio isósceles de área dada  $S$ . Calcular la profundidad del canal y el ángulo que deben formar las paredes con la horizontal para que la superficie mojada sea mínima. (SOL: se trata de la mitad de un hexágono regular)



---

Integrales múltiples

---

Ya conocemos el comando `Integrate` para realizar integrales de funciones de una variable y, como podrás imaginar por lo visto hasta ahora, pasar a varias variables sólo será cuestión de poner algunas variables más y alguna coma. En general, la principal diferencia entre una y varias variables no es tanto la función a integrar como el recinto de integración (por ejemplo, no hay una manera directa de decirle a *Mathematica* que queremos integrar en un círculo). Veremos en esta práctica como se pueden escribir los recintos de integración en *Mathematica*, aprovechando para ello las capacidades gráficas del programa.

Expondremos sólo el caso de dos variables; para integrales triples los comandos serán totalmente análogos.

Básicamente sólo sabemos calcular la integral de una función  $f(x, y)$  en recintos de uno de los siguientes tipos:

$$A = \{(x, y) \in \mathbb{R}^2 : a \leq x \leq b, g_1(x) \leq y \leq g_2(x)\} \quad (10.1)$$

o

$$B = \{(x, y) \in \mathbb{R}^2 : a \leq y \leq b, h_1(y) \leq x \leq h_2(y)\} \quad (10.2)$$

Las correspondientes integrales serían, vía el teorema de Fubini,

$$\int_A f(x, y) \, d(x, y) = \int_a^b \left( \int_{g_1(x)}^{g_2(x)} f(x, y) \, dy \right) dx$$

y

$$\int_B f(x, y) \, d(x, y) = \int_a^b \left( \int_{h_1(y)}^{h_2(y)} f(x, y) \, dx \right) dy$$

La orden para calcular el valor de una integral doble es `Integrate` o, en caso de que sólo nos interese una aproximación numérica, `NIntegrate`. Dado que el comando `Integrate` intenta calcular una primitiva y después evaluar en los extremos de integración, es más lento que el comando `NIntegrate`. La orden completa es

```
Integrate[función, {x, a, b}, {y, c, d}]
```

y lo mismo para NIntegrate.

El caso más fácil se presenta cuando integramos en rectángulos. En este caso las funciones  $g_i$  o  $h_i$  son constantes. Por ejemplo, si queremos calcular la integral de  $f(x, y) = x^2 + y^2$  en  $[0, 2] \times [1, 5]$  tendríamos que hacer lo siguiente:

```
In[1]:=
  Integrate[x^2 + y^2, {x, 0, 2}, {y, 1, 5}]
Out[1]=
  280
  3
```

Pasemos a integrales un poco más complicadas. Supongamos que queremos integrar la función  $f(x, y) = x$  en el conjunto

$$A = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1, x \geq 0, y \geq 0\}.$$

El primer paso es intentar averiguar cómo es el conjunto, para lo que nos haría falta dibujar  $x^2 + y^2 = 1$ . Para ello, necesitamos un paquete de *Mathematica* para dibujar curvas que vienen dadas por una ecuación (llamada “ecuación implícita”). La orden para cargar este paquete extra es:

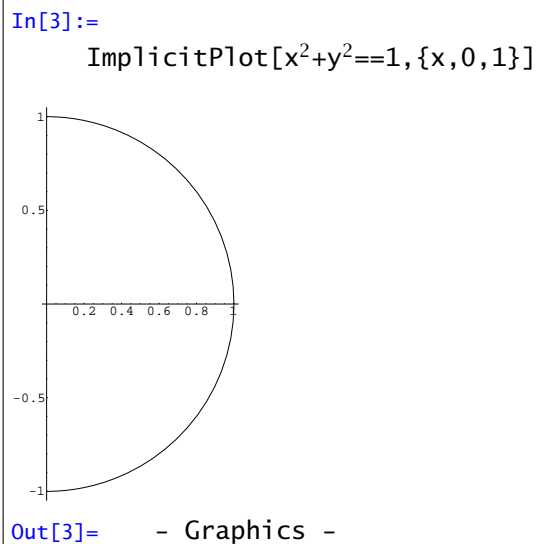
```
In[2]:=
  <<Graphics`ImplicitPlot`
```

(observa que no produce ninguna salida). Ahora ya podemos pedir que nos pinte  $x^2 + y^2 = 1$  para  $x \in [0, 1]$ . La orden es

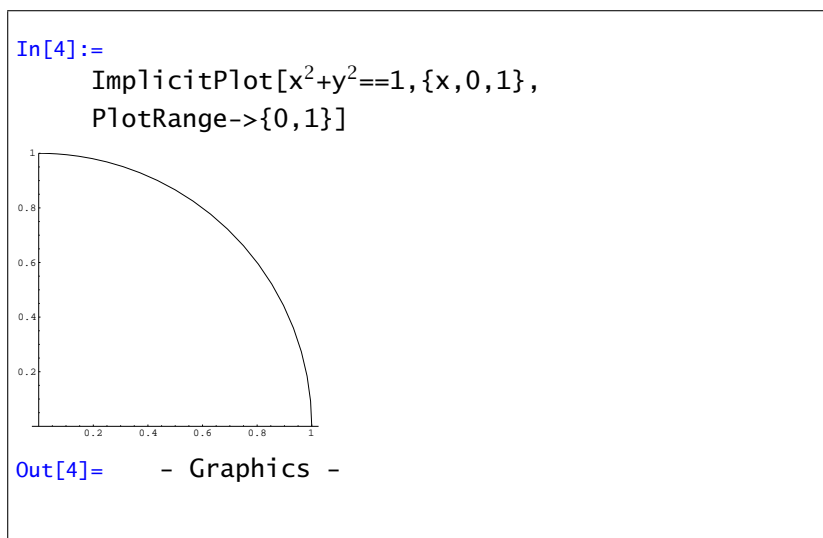
```
ImplicitPlot[ecuación, {x, a, b}]
```

donde  $a$  y  $b$  representan el rango de variación de la variable  $x$ , rango que normalmente tendremos que calcular “a ojo”. En nuestro caso

```
In[3]:=
  ImplicitPlot[x^2+y^2==1, {x, 0, 1}]
Out[3]=
  - Graphics -
```



Cómo se puede ver, *Mathematica* dibuja la circunferencia para “ $y$ ” positivo y negativo, en cambio sólo nos interesa la parte del primer cuadrante. Para representar esta parte podemos restringir el rango de la imagen con la opción `PlotRange` (la orden `ImplicitPlot` tiene básicamente las mismas opciones que la orden `Plot`).



Después de ver el conjunto  $A$  donde estamos integrando, es claro cómo podemos poner una variable en función de la otra. Por ejemplo, si ponemos  $y$  en función de  $x$ , para un valor concreto de  $x$  entre 0 y 1,  $y$  varía entre 0 y  $\sqrt{1-x^2}$ . Podemos calcular ahora el valor de la integral:

```
In[5]:=
Integrate[x,{x,0,1},{y,0,Sqrt[1-x^2]}]
```

```
Out[5]=
1/3
```

Como se puede observar el orden para evaluar la integral es de derecha a izquierda, así que si  $y$  depende de  $x$ , el rango de  $x$  va primero.

Observarás que no hemos cambiado a coordenadas polares. Para integrales en varias variables, interesa cambiar de variable cuando el dominio o la función son complicados. En el primer caso, si no podemos escribir el dominio de una las formas (1) o (2) tendremos que cambiar de variable nosotros (*Mathematica* no lo hace automáticamente). El caso de funciones complicadas no nos debe preocupar, pues *Mathematica* se encarga de resolver la integral en la inmensa mayoría de los casos, y si no se puede resolver directamente, siempre queda la posibilidad de aproximar el valor de integral.

Veamos un par de ejemplos:

**Ejemplo 1.** Calcular la integral de la función  $f(x, y) = e^{x/y}$ , en el conjunto  $A = \{(x, y) \in \mathbb{R}^2 : 0 \leq y^3 \leq x \leq y^2\}$ .

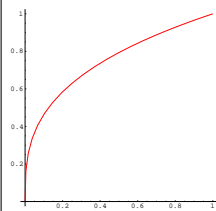
Estamos en el caso (2). Tenemos la variable  $x$  en función de  $y$ . Vamos primero a ver los puntos de corte de las ecuaciones  $y^3 = x$ ,  $y^2 = x$  y luego las dibujaremos para hacernos una idea de cuál es el dominio de integración:

```
In[6]:= Needs["Graphics`ImplicitPlot`"]
```

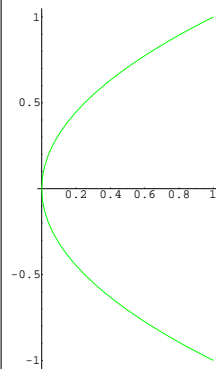
```
In[7]:= Solve[y3 == y2, y]
```

```
Out[7]= {{y->0}, {y->0}, {y->1}}
```

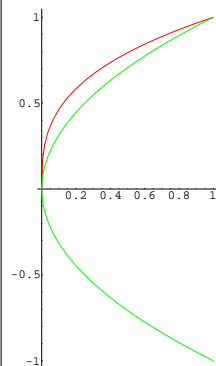
```
In[8]:= graf1=ImplicitPlot[y3==x, {x,0,1},  
PlotStyle->RGBColor[1,0,0]]
```



```
In[9]:= graf2=ImplicitPlot[y2==x, {x,0,1},  
PlotStyle->RGBColor[0,1,0]]
```



```
In[10]:= Show[graf1, graf2]
```



De lo hecho se deduce que la integral que queremos calcular es

```
In[11]:=
Integrate[Exp[x/y], {y, 0, 1}, {x, y^3, y^2}]
Out[11]=
 $\frac{3}{2} - \frac{E}{2}$ 
```

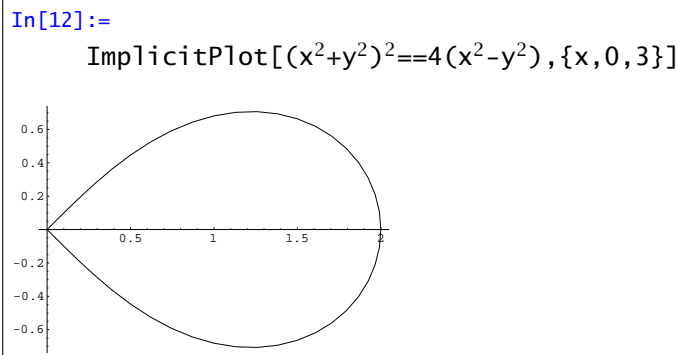
**Ejemplo 2.** Calcular la integral de la función

$$f(x, y) = x^2 + y^2$$

en el conjunto

$$A = \{(x, y) \in \mathbb{R}^2 : (x^2 + y^2)^2 \leq 4(x^2 - y^2), x \geq 0\}.$$

En este caso la función es relativamente sencilla pero el dominio no lo es tanto. No hay una manera fácil de hacer la integral directamente, así que es mejor cambiar a coordenadas polares y luego plantear la integral. Dibujemos el recinto:



Las condiciones del dominio escritas en coordenadas polares son:

$$\rho^4 \leq 4\rho^2(\cos^2(\theta) - \sin^2(\theta)), \rho \cos \theta \geq 0$$

Después de operar, nos quedaría

$$\rho \leq 2\sqrt{\cos^2(\theta) - \sin^2(\theta)} \text{ y } \theta \in [-\pi/4, \pi/4]$$

Ahora ya estamos en una de los casos conocidos y podemos plantear la integral, sin olvidar el cambio a coordenadas polares de la función y la “derivada” del cambio (en este caso  $\rho$ ):

```
In[13]:=
Integrate[\rho * \rho^2, {t, -Pi/4, Pi/4},
{ \rho, 0, 2 Sqrt[Cos[t]^2 - Sin[t]^2] }]
Out[13]=
 $\pi$ 
```

**Ejercicio 1.** Calcular la integral de las siguientes funciones en los recintos determinados por las siguientes ecuaciones:

1.  $f(x, y) = \sqrt{4x^2 - y^2}$ ,  $x = 1$ ,  $y = 0$ ,  $y = x$
2.  $f(x, y) = xe^{-x^2/y}$ ,  $x = 0$ ,  $y = 1$ ,  $y = x^2$
3.  $f(x, y) = y$ ,  $y > 0$ ,  $x^2 + y^2 = a^2$ ,  $y^2 = 2ax$ ,  $x = 0$

**Ejercicio 2.** Calcular el volumen del conjunto  $A$  en cada uno de los siguientes casos:

1.  $A = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 \leq z \leq \sqrt{x^2 + y^2}\}$
2.  $A = \{(x, y, z) \in \mathbb{R}^3 : \frac{x^2}{a^2} + \frac{y^2}{b^2} \leq 1, 0 \leq z \leq \sqrt{\frac{x^2}{a^2} + \frac{y^2}{b^2}}\}$
3.  $A = \{(x, y, z) \in \mathbb{R}^3 : 0 \leq z \leq x^2 + y^2, x + y \leq 1, x, y \geq 0\}$
4.  $A = \{(x, y, z) \in \mathbb{R}^3 : 0 \leq z \leq \sqrt{x^2 + y^2}, x^2 + y^2 \leq 2y\}$
5.  $A = \{(x, y, z) \in \mathbb{R}^3 : 0 \leq z \leq 4 - y^2, 0 \leq x \leq 6\}$
6.  $A = \{(x, y, z) \in \mathbb{R}^3 : \sqrt{x} \leq y \leq 2\sqrt{x}, 0 \leq z \leq 9 - x\}$
7.  $A = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 \leq z^2, x^2 + y^2 + z^2 \leq 2z\}$

**Ejercicio 3.** Calcular el volumen limitado superiormente por el cono  $4x^2 + 4y^2 - z^2 = 0$ , inferiormente por el plano  $z = 0$  y lateralmente por el cilindro  $x^2 + (y - 2)^2 = 4$ .

**Ejercicio 4.** Calcular el volumen de los sólidos siguientes:

1. Intersección de los cilindros  $x^2 + y^2 = r^2$ ,  $y^2 + z^2 = r^2$ .
2. El limitado por el plano  $z = 0$ , el cilindro  $x^2 + y^2 = 2ax$  ( $a > 0$ ) y el cono  $x^2 + y^2 = z^2$ .

### 11.1. Operaciones básicas

En esta práctica vamos a recordar algunas de las propiedades básicas de los números complejos y de la funciones de variable compleja.

En primer lugar hay que decir que *Mathematica* trabaja con números complejos siempre. Recordemos, por ejemplo que cuando intentábamos calcular las raíces de un polinomio nos daba todas, ya fueran reales o complejas.

Escribiremos los números complejos usando la letra I o usando la paleta. Obsérvese que la I es mayúscula. Veamos algunas de las operaciones elementales:

```
In[1]:=
  z=3+4I
  Re[z]
  Im[z]
Out[1]=
  3+4I

Out[2]=  3

Out[3]=  4
```

Ya puedes imaginar que Re e Im indican la parte real y la parte imaginaria del número complejo dado. Hay que tener cuidado con lo que hemos mencionado antes: *Mathematica* trabaja con números complejos por defecto. Si probamos con variables esto puede dar algunos “problemas”:

```
In[4]:=
  Re[a+b I]
Out[4]=
  -Im[b]+Re[a]
```

Lo que ocurre es que *Mathematica* piensa que  $a$  y  $b$  son números complejos. Podemos imponer que las variables sean reales con la orden `ComplexExpand`:

```
In[5]:=
  ComplexExpand[Re[a+b I]]
Out[5]=
  a
```

Además podemos calcular módulos, argumentos, ...

```
In[6]:=
  Abs[-3]
Out[6]=
  3

In[7]:=
  Abs[1+I]
Out[7]=
   $\sqrt{2}$ 

In[8]:=
  Arg[1-I]
Out[8]=
   $-\frac{\pi}{4}$ 

In[9]:=
  Conjugate[2+I]
Out[9]=
  2-I
```



**Ejercicio 1.** Efectuar las operaciones indicadas:

1.  $\frac{2-3i}{4-i}$
2.  $\frac{(2+i)(3-2i)(1+2i)}{(1-i)^2}$
3.  $(2i-1)^2 \left[ \frac{4}{1-i} + \frac{2-i}{1+i} \right]$
4.  $\frac{i^4 + i^9 + i^{16}}{2 - i^5 + i^{10} - i^{15}}$

**Ejercicio 2.** Si  $z_1 = 1 - i$ ,  $z_2 = -2 + 4i$ , y  $z_3 = \sqrt{3} - 2i$ , calcular las siguientes expresiones:

1.  $z_1^2 + 2z_1 - 3$
2.  $|2z_2 - 3z_1|^2$
3.  $(z_3 - \bar{z}_3)^5$
4.  $|z_1 \bar{z}_2 + z_2 \bar{z}_1|$
5.  $\left| \frac{z_1 + z_2 + 1}{z_1 - z_2 + i} \right|$

$$6. |z_1^2 + \bar{z}_2^2|^2 + |\bar{z}_3^2 - z_2^2|^2$$

**Ejercicio 3.** Calcular el módulo y el argumento de los números complejos:

1. 1
2.  $i$
3.  $-1$
4.  $-i$
5.  $\frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}$

**Ejercicio 4.** Expresar en forma polar los números complejos:

1.  $3+3i$
2.  $-1 + \sqrt{3}i$
3.  $-1$
4.  $-2 - \sqrt{3}i$

## 11.2. Funciones de variable compleja

Recordemos que podemos ver los números complejos como pares de números reales y por tanto cualquier función  $f : \mathbb{C} \rightarrow \mathbb{C}$  como una función de  $\mathbb{R}^2$  en  $\mathbb{R}^2$ . Por tanto podemos definir funciones complejas de la siguiente forma:

```
In[10]:=
      f[x_,y_]=(x+I y)^2
Out[10]=
      (x+Iy)^2

In[11]:=
      D[f[x,y],x]
Out[11]=
      2(x+Iy)
```

Teniendo esto en cuenta es fácil comprobar si una función verifica por ejemplo las ecuaciones de Cauchy-Riemann.

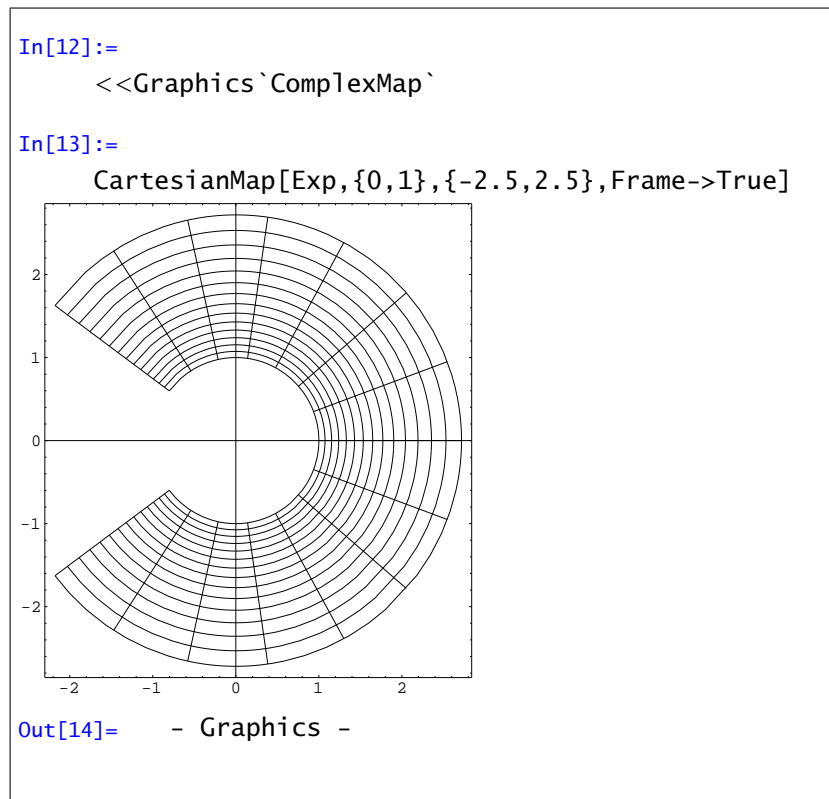
### 11.2.1. Gráficos en “cuatro” dimensiones

La función exponencial compleja estaba definida de la siguiente forma:

$$e^z = e^x(\cos(y) + i \operatorname{sen}(y)), \text{ donde } z = x + iy$$

De la definición se deduce que es una función periódica con periodo  $2\pi i$ , no toma nunca el valor cero y toma todos los demás infinitas veces. Además si fijamos la parte imaginaria y variamos la parte real obtenemos una recta que pasa por el origen y la parte imaginaria nos indica la pendiente. Al revés, si fijamos la parte real y movemos la parte imaginaria obtenemos una circunferencia centrada en el origen cuyo radio es  $e^x$ .

Si quisiéramos dibujar una función compleja de variable compleja como esta necesitaríamos 4 dimensiones. Lo que se puede hacer es representar lo que acabamos de comentar: en que se transforman rectas paralelas a alguno de los dos ejes. Para esto necesitamos un paquete extra



Comentemos un momento que es lo que pasa aquí. La orden `CartesianMap` representa en que se transforman rectas paralelas a los ejes, en este caso la primera coordenada varía entre 0 y 1 y la segunda entre -2.5 y 2.5, mediante la función dada (en este caso la exponencial). También podemos trabajar en coordenadas polares con la orden `PolarMap`. En ese caso, además de la función, tenemos que decir dónde varía el módulo y el argumento.

---

## Avisos y mensajes de error

---

```
In[1]:=
C=3
Set::wrsym: Symbol C is Protected
```

La letra C no se puede usar como nombre de variable. *Mathematica* ya lo usa para algo.

```
In[2]:=
Plot[Log[x],{x,-2,2}]
Plot::"plnr":Log[x] is not a machine-size real
number at x = -2...
```

La función (en este caso logaritmo neperiano) no está definida en ese punto. Probablemente estemos intentando dibujar una función fuera de su dominio o, si la función la hemos definido nosotros, hay algún error en la definición de la función.

```
In[3]:=
Plot[x^2 +1,{x,o,10}]
Plot::"plln": Limiting value o in x, o, 2 Pi is
not a machine-size real number.
```

Has escrito la letra “o” en lugar de 0.

```
In[4]:=
Plot[sin[x],{x,0,Pi}]
General::"spell": "Possible spelling error: new
symbol name sin is similar to existing symbol
Sin.
```

La función `sin` no está definida. La función seno es `Sin`.

```
In[5]:=
Plot[Sin[x],{x,-Pi,Pi},
PlotStyle->RGBColor[1,0,0]

Syntax::"bktmcp": "Expression "Plot[ « 1» "has no
closing "]".
```

Falta cerrar un corchete.

```
In[6]:=
Solve[Cos[x]==0,x

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

Out[6]=
{x- > - $\frac{\pi}{2}$ }, {x- >  $\frac{\pi}{2}$ }
```

Para resolver la ecuación  $\cos(x) = 0$  tiene que usar la función `arccos` (su inversa) y *Mathematica* avisa de que es posible que falten soluciones

```
In[7]:=
NIntegrate[Cos[x],{x,0,Pi}]

NIntegrate::"ploss": Numerical integration
stopping due to loss of precision. Achieved
neither the requested PrecisionGoal nor
AccuracyGoal; suspect highly oscillatory
integrand, or the true value of the integral
is 0. If your integrand is oscillatory try using
the option Method->Oscillatory in NIntegrate.
```

*Mathematica* no puede asegurar que el valor de la integral que da sea exacto, probablemente debido a que la función oscila.

```
In[8]:=
FindRoot[{x*y==1,x^2+y^2=1},{x,0},{y,0}]

FindRoot::jsing: Encountered a singular Jacobian
at the point {x, y} = {0.,0.}. Try perturbing
the initial point(s).
```

No se puede tomar el punto (0,0) como condición inicial. Intenta cambiar el punto por otro.

```
In[9]:=
FindRoot[5 x^2+3,{x,1}]

FindRoot::cvnwt: Newton's method failed to
converge to the prescribed accuracy after 15
iterations

Out[9]=
{x->-0.6}
```

Aunque en este caso la función no se anula nunca, *Mathematica* está avisando que el comando `FindRoot` no puede afirmar que el resultado que presenta sea un cero de la función.

```
In[10]:=
A={{1,2,3},{4,5,6},{1,3,2}};
B={{4,3},{7,3},{1,0}};
A.B

Dot::dotsh: Tensors {{1, 2, 3}, {4, 5, 6}, {1,
3, 2}} and {{4, 3}, {7, 3}} have incompatible
shapes.
```

No se puede multiplicar una matriz 3x3 y una matriz 2x2.

```
In[11]:=
Integrate[1/x^3,{x,-1,1}]

Integrate::idiv: Integral of 1/x^3 does not
converge on{-1,1}.
```

La función  $\frac{1}{x^3}$  no es impropia integrable en el intervalo  $[-1, 1]$ .

### B.1. Algunas funciones

- `Log[b, x]` logaritmo de  $x$  en base  $b$
- `Log[x]` logaritmo neperiano
- `Exp[x]` exponencial de base  $e$
- `Sqrt[x]` raíz cuadrada
- `Sin[x]` seno
- `Cos[x]` coseno
- `Tan[x]` tangente
- `Csc[x]` cosecante
- `Sec[x]` secante
- `Cot[x]` cotangente
- `ArcSin[x]` arcoseno
- `ArcCos[x]` arcocoseno
- `ArcTan[x]` arcotangente
- `ArcCsc[x]` arcocosecante
- `ArcCot[x]` arcocotangente
- `Sinh[x]` seno hiperbólico
- `Cosh[x]` coseno hiperbólico
- `Tanh[x]` tangente hiperbólica
- `Sech[x]` secante hiperbólica
- `Coth[x]` cotangente hiperbólica
- $n!$  factorial de  $n$
- `Binomial[n, m]`  $\binom{n}{m}$
- `Abs[x]` valor absoluto de  $x$
- `Floor[x]` parte entera de  $x$

### B.2. Comandos usuales

- `Expand[expresión]` desarrolla una expresión algebraica
- `Factor[expresión]` factoriza una expresión algebraica
- `FullSimplify[expresión]` simplifica la expresión
- `TrigExpand[expresión]` desarrolla una expresión con términos trigonométricos
- `TrigFactor[expresión]` factoriza una expresión con términos trigonométricos
- `Simplify[expresión]` simplifica la expresión
- `N[expresión]` da un valor numérico de la expresión

### B.3. Números complejos

- `Abs[z]` módulo
- `Arg[z]` argumento principal
- `ComplexExpand`
- `Conjugate[z]` conjugado
- `Im[z]` parte imaginaria
- `Re[z]` parte real

## B.4. Matrices

- `Det[matriz]` determinante
- `DiagonalMatrix[{a,b,c,...}]` matriz diagonal
- `Eigenvalues[matriz]` valores propios
- `Eigenvectors[matriz]` vectores propios
- `Eigensystem[matriz]` valores y vectores propios
- `IdentityMatrix[n]` matriz identidad de orden  $n$
- `Inverse[matriz]` inversa de la matriz dada
- `Minors[matriz]`
- `RowReduce[matriz]`
- `Table[expresión, {variable, índice}]` genera una lista usando la expresión dada
- `Transpose[matriz]` matriz transpuesta

## B.5. Resolución de ecuaciones

- `FindRoot[función, {x, x0}]` encuentra un cero de la función tomando como punto inicial  $x = x_0$
- `LinearSolve[matriz A, vector b]` resuelve el sistema lineal  $Ax = b$ .
- `NSolve[{ecuación1, ecuación2,...}, {variable1, variable2,...}]`
- `NullSpace[matriz]` calcula una base del núcleo de la matriz
- `Reduce[{ecuación1, ecuación2,...}, {variable1, variable2,...}]` simplifica el sistema ecuaciones
- `Solve[{ecuación1, ecuación2,...}, {variable1, variable2,...}]` calcula las soluciones genéricas del sistema de ecuaciones

## B.6. Gráficos

- `Plot[f[x], {x, a, b}]` gráfica de  $f$  en  $[a, b]$
- `ParametricPlot`
- `Plot3D[f[x,y], {x, a, b}, {y, c, d}]` gráfica de  $f$  en  $[a, b] \times [c, d]$
- `ParametricPlot3D`
- `SurfaceOfRevolution`
- `PlotGradientField`
- `PlotHamiltonianField`

## B.7. Cálculo

- `D[función, {variable, n}]` derivada de orden  $n$  de la función
- `Integrate[función, {variable, a, b}]` integral entre  $a$  y  $b$  de la función
- `Limit[función, {x->x0}]` calcular el límite cuando  $x$  tiende a  $x_0$  de la función
- `NIntegrate[función, {variable, a, b}]` integral entre  $a$  y  $b$  de la función
- `Series[función, {x, a, n}]` desarrollo de Taylor de orden  $n$  en el punto  $a$

## B.8. Otros

- `Clear[variable1, variable2, ...]` olvida el valor de las variables
- `Clear["Global`*"]` olvida el valor de *todas* las variables
- `Infinity` infinito