Introducción a Maple

Maple es una poderosa herramienta para cálculo simbólico. Esto significa que además de operar como una calculadora con números, también es capaz de hacer multitud de tareas matemáticas que incluyan símbolos. Como calculadora, Maple se diferencia de las convencionales, aparte de su potencia de cálculo, en que puede realizar las operaciones de forma exacta. Para poner algunos ejemplos con los que ilustrar estas ventajas, necesitaremos familiarizarnos con la forma de usar Maple.

Primeras operaciones

Maple es un software interactivo que espera que le demos una orden ("input") y nos devuelve un resultado ("output"). En pantalla aparece el símbolo ">" como una indicación de que Maple está esperando nuestras instrucciones. Terminadas éstas (escritas con la sintaxis adecuada), debemos poner **;** seguido de INTRO (retorno de carro) y entonces Maple responderá. Empecemos con las habituales operaciones aritméticas.

```
> 2+3;
```

$$5$$

Como se observa, de forma predeterminada, la orden aparece de color rojo alineada a la izquierda y el resultado de color azul y centrado.

```
> 4*6; 12/6; 2^5;
```

$$24$$

$$2$$

$$32$$

Hemos introducido tres órdenes, pero solo hemos pulsado INTRO al final de la tercera.
Con las sencillas operaciones siguientes ya podemos ilustrar el hecho de que Maple trabaja con cantidades exactas:

```
> 12/8; 12/8 * 2/3;
```

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

$$1$$

Con según qué calculadoras, es muy posible que esta última operación se hubiera realizado en la siguiente forma aproximada: 12/8=1.5, 2/3=0.6666667, y por tanto el producto es 1.0000005. Por supuesto que Maple permite utilizar aproximaciones de un número. Esto se hace así:

```
> evalf(11/43);
```

$$0.2558139535$$

Ya vamos viendo la sintaxis de Maple. Generalmente las órdenes son expresiones bastante significativas (¡en inglés!), encerrando entre paréntesis los argumentos. Pero no nos preocupemos por esto, pues con unas pocas horas de práctica y la ayuda de Maple y las barras de herramientas, la sintaxis no debería crearnos mayores problemas. Para una mejor aproximación, hacemos

```
> evalf[50](11/43);
```

$$0.25581395348837209302325581395348837209302325581395$$

Hemos indicado entre corchetes **[ ]** que queremos 50 cifras decimales. Volviendo a lo anterior, parece más interesante que Maple trabaje con números exactos. De hecho, opera y simplifica:

```
> 3/7 * 5/9; (2^(1/3))^6;
```

$$\frac{5}{21}$$

$$4$$

En la última operación, raíz cúbica de 2 elevada a la sexta, una calculadora convencional podría haber dado el resultado aproximado. Este ejemplo nos da pie para hablar de otro problema. ¿Qué ocurre cuando se reiteran operaciones? Como es regla general en matemáticas, Maple realiza primero las operaciones entre paréntesis, después las potencias, seguido de las multiplicaciones y divisiones y, por último, las sumas y restas. Veamos qué ocurre si en la última orden prescindimos de los paréntesis:

```
> 2^1/3^6;
```

$$\frac{2}{729}$$

Es claro que no queríamos hacer esta operación. Aunque el orden es el anteriormente expuesto, recomendamos no tener pereza en poner paréntesis, aunque muchas veces puedan sobrar.
La raíz cuadrada tiene orden propia:

```
> sqrt(12);
```

$$2\sqrt{3}$$

Insistimos en la forma exacta de computar que tiene Maple:

```
> sqrt(12)*((7/5)+(2/9));
```

$$\frac{146}{45}\sqrt{3}$$

```
> ((17/24)-(1-(5/4))^2)^7;
```

$$\frac{27512614111}{587068342272}$$

El símbolo **%** hace referencia al último resultado y se usa a veces porque nos permite no tener que volver a escribir éste. Por ejemplo, obtengamos una aproximación con 60 decimales de la fracción anterior.

```
> evalf[60](%);
```

$$0.046864414464121928866944140803612254229538180155464106081390 0$$

¿Qué ocurre si volvemos a poner **%**? Veámoslo:

```
> % + 15;
```

$$15.04686441$$

Hay que tener cuidado al usar **%** porque se refiere al último resultado que ha obtenido Maple y podría no coincidir con el resultado que vemos en el párrafo anterior.
Cuando Maple opera con decimales da el resultado con decimales, aunque el número sea entero. Por eso, para obtener una aproximación decimal de 11/43 también podemos escribir 11.0/43 ó sin más 11./43 u 11/43.. Siempre que a algún número de los que intervienen en una operación le acompañemos de un punto obtendremos el resultado aproximado:

```
> 0.5*2;
  11/43.;
  sqrt(3.);
```

$$1.0$$

$$0.2558139535$$

$$1.732050808$$

Cálculo simbólico

Aquí empezaremos a atisbar la importancia del uso de Maple. Esta clase de programas pueden operar con símbolos y hacer simplificaciones como uno las haría en un papel. Vamos a ver un primer ejemplo. Todos sabemos que (x^2–1)/(x+1) = x–1.

```
> (x^2 -1)/(x+1);
```

$$\frac{x^2-1}{x+1}$$

¿Maple no lo sabe? No nos rendimos:

```
> simplify(%);
```

$$x-1$$

Si manejamos expresiones más complicadas, puede que Maple no simplifique como nosotros. De hecho, el concepto de simplificación no es nada claro; por ejemplo, ¿qué expresión está más simplificada: x^2–1 ó (x–1)/(x+1)? Dejamos a la experimentación de cada cual saber dónde está el límite de Maple. No hay ninguna dificultad en usar dos o más variables:

```
> (a^2 + a*b + b^2)*(a-b);
```

$$\left(a^2+a\,b+b^2\right)\,(a-b)$$

```
> simplify(%);
```

$$\left(a^2+a\,b+b^2\right)\,(a-b)$$

No ha hecho caso, para Maple dicha expresión ya está simplificada. Pero el lector notará que vamos buscando a^3-b^3. Hay órdenes similares que nos pueden ayudar.

```
> collect(%,a);
```

$$a^3-b^3$$

Esta última orden resalta el valor de **a** en la expresión, poniendo ésta como un polinomio en **a**. Otro ejemplo,

```
> (a-b)*(a^2 -2)+(b-3)*(5-a);
```

$$(a-b)\,\left(a^2-2\right)+(b-3)\,(5-a)$$

```
> simplify(%);
```

$$a^3+a-b\,a^2+7\,b-a\,b-15$$

```
> collect((a-b)*(a^2 -2)+(b-3)*(5-a),a);
```

$$a^3-b\,a^2+(1-b)\,a+7\,b-15$$

```
> collect((a-b)*(a^2 -2)+(b-3)*(5-a),b);
```

$$\left(-a^2+7-a\right)\,b+a\,\left(a^2-2\right)-15+3\,a$$

Sería muy costoso explicar aquí cómo opera exactamente Maple. El lector interesado debería acudir a la ayuda relativa a las órdenes anteriores. La forma más rápida de obtener ayuda sobre una orden es la siguiente: colocamos el cursor en una cualquiera de las letras de la orden (por ejemplo, en **collect**), en el menú superior vamos a *Help/Help on "collect"* y nos aparecerá una pantalla con explicación detallada sobre la orden. Dicha información es bastante exhaustiva y organizada de la siguiente manera: *Calling sequence* (explica la sintaxis correcta de la orden), *Parameters* (explica la clase de parámetros que admite la orden), *Description* (explica el funcionamiento de la orden), *Examples* y *See also* (órdenes relacionadas). Para los primeros escarceos con Maple recomendamos acudir principalmente a *Examples* e investigar las órdenes relacionadas.

Asignación de valores a variables

Si a alguna variable le asignamos un valor determinado, en todo lo siguiente Maple entenderá que esa letra tiene ese valor. La forma de hacerlo es con la sintaxis **variable:=valor**.

```
> a:=1; (a-b)*(a^2-2)+(b-3)*(5-a);
```

$$a := 1$$

$$-13+5\,b$$

Si queremos que después de alguna orden Maple no ponga en pantalla el resultado, aunque sí lo guarde, escribiremos **:** en vez de **;** . Por ejemplo:

```
> a:=1:(a-b)*(a^2-2)+(b-3)*(5-a);
```

$$-13+5\,b$$

Si ahora queremos utilizar la expresión a^3, no podemos porque esto para Maple es 1. En efecto,

```
> a^3;
```

$$1$$

O usamos otra letra o decimos a Maple que borre la asignación de **1** a la letra **a**. Esto se hace así:

```
> unassign('a');expand((a+5)^2);
```

$$a^2+10\,a+25$$

¿Qué ocurrirá si ponemos **a:=b**? En todo lo que siga **a** será igual a **b**.

```
> a:=b: a^3; a+b+x; b:=2: a+b+x;
```

$$b^3$$

$$2\,b+x$$

$$4+x$$

Dejemos que las cosas vuelvan a la normalidad y que **a** y **b** vuelvan a funcionar como variables.

```
> unassign('a,b');a+b+x;
```

$$a+b+x$$

Se puede poner **a:='a';** como alternativa a **unassign('a');**. La orden **restart** borra el contenido de la memoria de Maple. Es conveniente usarla cuando queremos programar algo nuevo. Con determinadas órdenes se puede conseguir que el valor que damos a una variable no quede asignado para siempre, sino sólo en lo que afecta a dicha orden. Es el caso de **simplify**. Por ejemplo, si queremos saber cuánto vale el polinomio 3x^2+2x+1 en x=(a–1)^2, podríamos hacer:

```
> 3*x^2+2*x+1;x:=(a-1)^2:simplify(3*x^2+2*x+1);
```

$$3\,x^2+2\,x+1$$

$$6+20\,a^2-16\,a+3\,a^4-12\,a^3$$

El problema es que así x se ha quedado con el valor (a–1)^2 . En efecto,

```
> 5*x+6;
```

$$5\,(a-1)^2+6$$

Parece ser mejor hacer lo siguiente:

```
> restart;simplify(3*x^2+2*x+1,{x=(a-1)^2});5*x+6;
```

$$6+20\,a^2-16\,a+3\,a^4-12\,a^3$$

$$5\,x+6$$

Otra orden que actúa de manera similar es **eval**.

```
> eval((y+2)/(y-2),y=z+5);y;
```

$$\frac{z+7}{z+3}$$

$$y$$

Sucesiones y funciones

```
> restart;
```

Empezaremos hablando de funciones. En realidad, una sucesión es una función donde la variable es un número natural y, en muchos aspectos, Maple las trata de igual forma. Maple tiene predefinidas las funciones elementales; para usarlas solo hay que conocer la sintaxis apropiada. Ejemplos de funciones:

```
> log(x);ln(x);#  las dos expresiones son el logaritmo neperiano
```

$$\ln(x)$$

$$\ln(x)$$

```
> arctan(z);sin(y);exp(a^2+1);
```

$$\arctan(z)$$

$$\sin(y)$$

$$e^{a^2+1}$$

Por supuesto, podemos calcular valores particulares y operar con estas funciones.

```
> exp(3);arctan(1);
```

$$e^3$$

$$\frac{1}{4}\pi$$

```
> sin(Pi/4);log(exp(5));
```

$$\frac{1}{2}\sqrt{2}$$

$$5$$

Analicemos unos cuantos ejemplos de simplificación:

```
> log(exp(x));
```

$$\ln(e^x)$$

```
> simplify(%);
```

$$\ln(e^x)$$

¿Por qué el resultado no es **x**? Las funciones elementales se extienden a variable compleja, y sus propiedades en el cuerpo de los números complejos son diferentes de las que tienen consideradas en el de los reales. Maple trabaja con estas funciones más generales. Por ejemplo, con números complejos la función exponencial no es inyectiva y su rango es mayor que los reales positivos. Esto hace que existan muchos logaritmos (inversas de la exponencial) y que exista una definición de logaritmo de números no necesariamente positivos. Así,

```
> log(-1);
```

$$I\pi$$

(es el número complejo $i\pi$). Pero digamos al programa que la variable es real:

```
> simplify(log(exp(x)),assume=real);
```

$$x$$

También puede hacerse así:

```
> assume(x,real);simplify(log(exp(x)));
```

$$x\sim$$

El resultado es el mismo. El significado de la tilde que acompaña a la x es que Maple nos avisa de que sobre la variable x hay hecha una suposición que se mantendrá hasta que "desasignemos" dicha variable. En la forma anterior no ha aparecido la tilde, puesto que Maple entiende que no hay suposición permanente sobre la variable; se puede decir que la suposición de ser real actúa solamente dentro de la orden **simplify**. Es como el caso de asignación de valores a una variable que comentábamos en el apartado anterior. Analicemos otros ejemplos de posible simplificación que nos muestran que Maple no tiene incorporadas todas las reglas.

```
> sin(arcsin(y));
```

$$y$$

```
> assume(y<pi/2,y>-pi/2);
  arcsin(sin(y));
```

$$\arcsin(\sin(y\sim))$$

En este caso no ha funcionado la simplificación.

```
> unassign('x'):sqrt(x^2);
```

$$\sqrt{x^2}$$

```
> assume(x,real);
  simplify(sqrt(x^2));
```

$$|x\sim|$$

```
> unassign('x'):simplify(sqrt(x^2),assume=real);
```

$$|x|$$

```
> assume(x,positive);
  simplify(sqrt(x^2));
```

$$x\sim$$

```
> simplify(sqrt(u^2),assume=positive);
```

$$u$$

Analícese en estas órdenes por qué aparecen o no las tildes.

Maple conoce algunas relaciones entre funciones trigonométricas. Para simplificar, existen dos funciones específicas: **simplify(**expresión**,trig)** y **combine(**expresión**,trig)**.

```
> unassign('a,b'):
  c:=sin(a)*cos(b)+cos(a)*sin(b);
```

$$c := \sin(a)\cos(b) + \cos(a)\sin(b)$$

```
> simplify(c,trig);
  combine(c,trig);
```

$$\sin(a)\cos(b) + \cos(a)\sin(b)$$

$$\sin(a+b)$$

```
> d:=sin(u)^6;simplify(d,trig);
  combine(d,trig);
```

$$d := \sin(u)^6$$

$$\sin(u)^6$$

$$\frac{5}{16} - \frac{1}{32}\cos(6u) + \frac{3}{16}\cos(4u) - \frac{15}{32}\cos(2u)$$

Para expresiones en que intervienen potencias, logaritmos, exponenciales y más, Maple también dispone de órdenes específicas para la simplificación. Remitimos al lector a la ayuda sobre **simplify**.

Definición de funciones

```
> restart;
```

Pasemos al proceso de definir una función. La forma más sencilla es la siguiente

```
> f:=x->x^3+1;
```

$$f := x \to x^3 + 1$$

La orden anterior es clara, como nos indica el correspondiente resultado: hemos definido una

función con el nombre **f** que, a cualquier cosa, le asigna su cubo más uno.

```
> f(2);f(a);f(b^(1/3));f(gato);
```

$$9$$
$$a^3+1$$
$$b+1$$
$$gato^3+1$$

Podemos usar cualquier letra o conjunto de letras y números para designar a una función. Si empleáramos alguna denominación que interfiere con algo ya predefinido, Maple nos advertirá.

```
> sin:=x->x+2;
Error, attempting to assign to `sin` which is protected
```

También podemos definir funciones de más variables. Por ejemplo,

```
> suma:=(x,y)->x+y;suma(4,11);suma(suma,suma);
```

$$suma := (x, y) \rightarrow x + y$$
$$15$$
$$2\ suma$$

Para definir funciones por trozos utilizamos la orden **piecewise**.

```
> g:=x->piecewise(x<0,sin(x),x<1,log(x+1),7);
```

$$g := x \rightarrow piecewise(x < 0, \sin(x), x < 1, \log(x+1), 7)$$

Estamos definiendo la función de la siguiente manera:
1) Si $x<0$, vale sen(x).
2) en otro caso (es decir, si $x \geq 0$), si $x<1$, vale log(x+1) (luego vale log(x+1) para $0 \leq x<1$).
3) en otro caso, es decir, si $x \geq 1$ vale 7.
Pidamos el valor en algunos puntos:

```
> g(-1);g(1/2);g(3);g(a);
```

$$-\sin(1)$$
$$\ln\left(\frac{3}{2}\right)$$
$$7$$
$$\begin{cases} \sin(a) & a<0 \\ \ln(a+1) & a<1 \\ 7 & otherwise \end{cases}$$

Otra forma de definir una función es mediante un *procedimiento* (*procedure* en inglés o subrutina en lenguaje de programación). Por ejemplo, vamos a definir de otra manera la función suma.

```
> restart;
> f := proc(x,y)
  x+y;
  end proc;
```

$$f := \mathbf{proc}(x, y)\ x+y\ \mathbf{end\ proc}$$

Primero indicamos que queremos la letra **f** para indicar un procedimiento que va a afectar a dos variables. Después indicamos el procedimiento, en este caso la suma de las dos variables. Con **end proc** terminamos el procedimiento.

```
> f(2,5);f(silla,mesa);
```

$$7$$
$$silla+mesa$$

Sucesiones

Las sucesiones se tratan como funciones. Estamos demasiado acostumbrados a usar la variable n para sucesiones y la variable x para funciones. Para Maple las dos variables van a ser reales (o incluso complejas); por eso, en determinadas situaciones habrá que indicarlo. Por ejemplo,

```
> restart;
> f:=n->n^2 +1;
```

$$f := n \rightarrow n^2+1$$

```
> frac(f(n));#frac indica la parte fraccionaria de un número
```

$$\mathrm{frac}(n^2+1)$$

```
> assume(n,integer);frac(f(n));
```

$$0$$

Los procedimientos son imprescindibles para definir sucesiones que no vengan dadas por una expresión funcional.
Como ejemplo, hagamos un procedimiento para crear la sucesión **g(n):=n!**.

```
> g:= proc( n::posint ) #  Estamos indicando que la variable sea número
  natural
  g(1):=1;
  g(n):=n*g(n-1);
  end proc;
```

$$g := \mathbf{proc}(n::posint)\ g(1) := 1;\ g(n) := n * g(n-1)\ \mathbf{end\ proc}$$

```
> g(5);g(50);factorial(50);
```

$$120$$
$$30414093201713378043612608166064768844377641568960512000000000000$$
$$30414093201713378043612608166064768844377641568960512000000000000$$

Ahora, crearemos la sucesión de Fibonacci, en la que sus dos primeros términos son 1 y cada uno es la suma de los dos anteriores:

```
> fibo:=proc(n::posint)
  fibo(1):=1; fibo(2):=1;
  fibo(n):=fibo(n-2)+fibo(n-1);
  end proc;
```

$$fibo := \mathbf{proc}(n::posint)$$
$$fibo(1) := 1;\ fibo(2) := 1;\ fibo(n) := fibo(n-2) + fibo(n-1)$$
$$\mathbf{end\ proc}$$

```
> fibo(7); fibo(1000);
```

$$13$$
$$43466557686937456435688527675040625802564660517371780402481729089536555417 9\backslash$$
$$49051890403879840079255169295922593080322634775209689623239873322471161\backslash$$
$$64299644090653318793829896964992851600370447613779516684922887 5$$

Para obtener un número determinado de primeros términos de esta sucesión, usamos la siguiente orden:

```
> seq(fibo(i),i=1..30);
```

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040

A veces, para operar con los términos de una sucesión, necesitaremos incorporarlos a una lista. Las listas (y algo menos los conjuntos) son una de las estructuras más importantes de un programa como Maple.

Conjuntos y listas

Un conjunto es una colección de objetos (números, variables, funciones o lo que sea). Para definir un conjunto deben indicarse sus elementos entre llaves y separados por comas. El nombre de un conjunto es el que queramos ponerle.

```
> restart;
  conjunto1:={3,0.5,x,sin(y)};
```

$$conjunto1 := \{3, 0.5, x, \sin(y)\}$$

```
> conjunto2:={0.5,asdf,exp(3),12};
```

$$conjunto2 := \{12, 0.5, asdf, e^3\}$$

```
> conjunto3={x,x,y,y,y,x};
```

$$conjunto3 = \{x, y\}$$

Con los conjuntos se pueden hacer operaciones:

```
> conjunto1 union conjunto2; conjunto1 intersect conjunto2;
```

$$\{3, 12, 0.5, asdf, x, e^3, \sin(y)\}$$

$$\{0.5\}$$

Obsérvese que los elementos del conjunto no guardan ninguna ordenación. En **conjunto2** nuestro cuarto elemento era 12 y en el resultado ha sido el primero. Por esto, se manejan las listas, que son conjuntos ordenados.

```
> lista1:=[3,0.5,x,sin(y)];
```

$$lista1 := [3, 0.5, x, \sin(y)]$$

Maple sabe que es una lista porque va entre corchetes: **[.,.,.,.]**. Si el **conjunto3** fuera una lista, ¿qué pasaría?

```
> lista3:=[x,x,y,y,y,x];
```

$$lista3 := [x, x, y, y, y, x]$$

Para extraer los elementos de una lista se procede así:

```
> lista1[4];lista3[2..4];
```

$$\sin(y)$$

$$[x, y, y]$$

A su vez, los elementos de una lista también pueden ser listas

```
> L:=[1,3,[5,2],[3,4,[2,7]]];
```

$$L := [1, 3, [5, 2], [3, 4, [2, 7]]]$$

Extraigamos alguno de sus elementos,

```
> L[3];
```

$$[5, 2]$$

```
> L[3,1];# o también L[3][1]
```

$$5$$

```
> L[4,3,2];
```

$$7$$

Gran parte de la importancia de las listas radica en que se puede operar con sus elementos de uno en uno. Por ejemplo,

```
> lista3/lista3;lista3+2*lista3;
```

$$1$$

$$[3\,x, 3\,x, 3\,y, 3\,y, 3\,y, 3\,x]$$

```
> zip((x,y)->x^y,[1,2,3,4,5],[2,3,2,3,4]);
```

$$[1, 8, 9, 64, 625]$$

Programación

```
> restart;
```

Maple dispone de los clásicos bucles **for**, **while**, **do**, así como de **if**, **else**, etc.. Estas órdenes no serán apenas usadas en este curso, pero para el lector interesado, desarrollaremos un par de ejemplos.

Vamos a definir una función, que llamaremos **numdiv**, que nos dé el número de divisores de un número natural

```
> numdiv:=proc(n::nonnegint)
  local d,k;
  d:=0;
  for k from 1 to n do
  if floor(n/k)=n/k then
  d:=d+1;
  end if;
  end do;
  print(d);
  end proc:
```

Y esta otra, **div**, nos devuelve el conjunto de divisores:

```
> div:=proc(n::nonnegint)
  local d,k;
  d:={};
  for k from 1 to n do
  if floor(n/k)=n/k then
  d:=d union {k};
  end if;
  end do;
  print(d);
  end proc:
> numdiv(20);div(20);
```

$$6$$

$$\{1, 2, 4, 5, 10, 20\}$$