Primeros pasos

1

1.1 Introducción

Vamos a comenzar familiarizándonos con *Maxima* y con el entorno de trabajo *wxMaxima*. Cuando iniciamos el programa se nos presenta una ventana como la de la Figura 1.1. En la parte superior tienes el menú con las opciones usuales (abrir, cerrar, guardar) y otras relacionadas con las posibilidades más "matemáticas" de *Maxima*. En segundo lugar aparecen algunos iconos que sirven de atajo a algunas operaciones y la ventana de trabajo. En ésta última, podemos leer un recordatorio de las versiones que estamos utilizando de los programas *Maxima* y *wxMaxima* así como el entorno Lisp sobre el que está funcionando y la licencia (GNU Public License):[1]

```
wxMaxima 0.8.3a http://wxmaxima.sourceforge.net
Maxima 5.19.2 http://maxima.sourceforge.net
Using Lisp SBCL 1.0.30
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
```

Ya iremos comentando con mayor profundidad los distintos menús y opciones que se nos presentan pero antes de ir más lejos, ¿podemos escribir algo? Sí, situa el cursor dentro de la ventana, pulsa y escribe 2+3. Luego pulsa las teclas Shift+Return. Obtendrás algo similar a esto:

```
(%i1)   2+3;
(%o1)   5
```

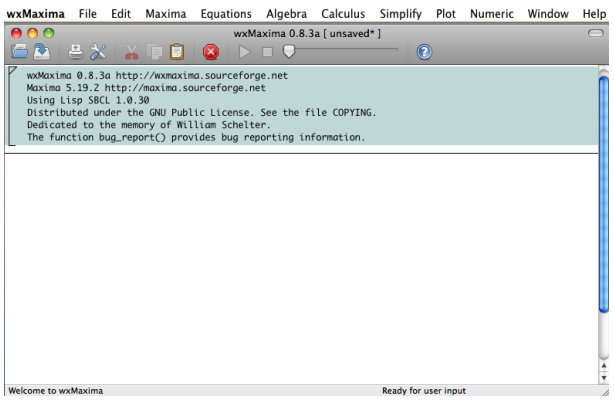


Figura 1.1 Ventana inicial de *wxMaxima*

Como puedes ver *Maxima* da la respuesta correcta: 5. Bueno, no parece mucho. Seguro que tienes una calculadora que hace eso. De acuerdo. Es sólo el principio.

Observación 1.1. Conviene hacer algunos comentarios sobre lo que acabamos de hacer:

a) No intentes escribir los símbolos "(%i1)" y "(%o1)", ya que éstos los escribe el programa para llevar un control sobre las operaciones que va efectuando. "(%i1)" se refiere a la primera entrada (input) y "(%o1)" a la primera respuesta (output).

[1] Por defecto, la ventana de *wxMaxima* aparece en blanco. En las preferencias del programa, se puede elegir que aparezca la versión instalada al inicio del mismo como se ve en la figura.

b) La entradas terminan en punto y coma. *wxMaxima* lo añade si tú te has olvidado de escribirlo. Justamente lo que nos había pasado.

Operaciones básicas

+	suma
*	producto
/	división
^ o **	potencia
sqrt( )	raíz cuadrada

El producto se indica con "*":

```
(%i2)  3*5;
(%o2)  15
```

Para multiplicar números es necesario escribir el símbolo de la multiplicación. Si sólo dejamos un espacio entre los factores el resultado es un error:

```
(%i3)  5 4;
       Incorrect syntax: 4 is not an infix operator
(%o3)  5Space4;
         ^
```

También podemos dividir

```
(%i4)  5+(2*4+6)/7;
(%o4)  7
(%i5)  5+2*4+6/7;
(%o5)  97/7
```

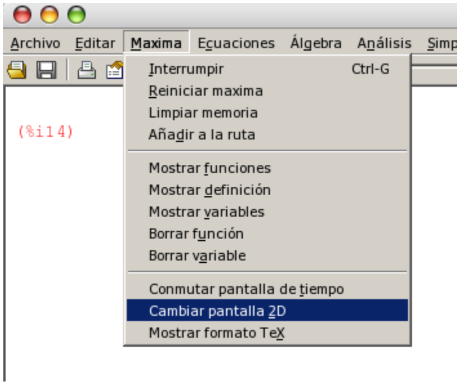
eso sí, teniendo cuidado con la precedencia de las operaciones. En estos casos el uso de paréntesis es obligado.

Podemos escribir potencias

```
(%i6)  3^57;
(%o6)  157004289908208161164053456 3
```

Fíjate en el número de dígitos que tiene el resultado. Es un primer ejemplo de que la potencia de cálculo de *Maxima* es mayor que la de una calculadora que no suele tener más allá de 10 o 12. Ya sé lo que estarás pensando en este momento: en lugar de elevar a 57, elevemos a un número más grande. De acuerdo.

```
(%i7)   3^1000;
        13220708194808066368904552597
(%o7)   5[418 digits]6143661321731027
        68902855220001
```



Como puedes ver, *Maxima* realiza la operación pero no muestra el resultado completo. Nos dice que, en este caso, hay 418 dígitos que no está mostrando. ¿Se puede saber cuáles son? Sí. Nos vamos al menú **Maxima→Cambiar pantalla 2D** y escogemos **ascii**. Por último, repetimos la operación.

```
(%i8)   set_display('ascii)$
(%i9)   3^1000;
        1322070819480806636890455259752144365965422032752148167664
        9203682268285973467048995407783138506080619639097776968725
        8235595095458210061891186534272525795367402762022519832080
        3878014774228964841274390400117588618041128947815623094438
(%o9)   0615661730540866744905061781254803444055470543970388958174
        6536825491613622083026856377858229022841639830788789691855
        6404084898937609373242171846359938695516765018940588109060
        4260896714388641028143503856487471658320106143661321731027
        68902855220001
```

La salida en formato ascii es la que tiene por defecto *Maxima*. La salida con formato xml es una mejora de *wxMaxima*. Siempre puedes cambiar entre una y otra vía el menú o volviendo a escribir

```
(%i10)  set_display('xml)$
(%i11)  3^1000;
(%o11)  13220708194808066368904552597 5[418 digits]61436613217310
        2768902855220001
```

Observación 1.2. Antes de seguir, ¿por qué sale $ y no punto y coma al final de la salida anterior? El punto y coma sirve para terminar un comando o separar varios de ellos. El dólar, $, también termina un comando o separa varios de ellos pero, a diferencia del punto y coma, *no* muestra el resultado en pantalla.

Si trabajamos con fracciones, *Maxima* dará por defecto el resultado en forma de fracción

```
(%i12)  2+5/11;
        27
(%o12)  --
        11
```

simplificando cuando sea posible

```
(%i13)  128/234;
         64
(%o13)  ---
        117
```

Cálculo simbólico

Cuando hablamos de que *Maxima* es un programa de cálculo simbólico, nos referimos a que no necesitamos trabajar con valores concretos. Fíjate en el siguiente ejemplo:

```
(%i14)  a/2+3*a/5
(%o14)  11a/10
```

Raíces

Bueno, hasta ahora sabemos sumar, restar, multiplicar, dividir y poco más. *Maxima* tiene predefinidas la mayoría de las funciones usuales. Por ejemplo, para obtener la raíz de un número se usa el comando `sqrt`

```
(%i15)  sqrt(5);
(%o15)  sqrt(5)
```

lo cuál no parece muy buena respuesta. En realidad es la mejor posible: *Maxima* es un programa de cálculo simbólico y siempre intentará dar el resultado en la forma más exacta.

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

```
(%i16)  5^(1/2);
(%o16)  sqrt(5)
```

float

Si queremos obtener la expresión decimal, utilizamos la orden `float`.

```
(%i17)  float(sqrt(5));
(%o17)  2.23606797749979
```

Constantes

Además de las funciones usuales (ya iremos viendo más), *Maxima* también conoce el valor de algunas de las constantes típicas.

`%pi`	el número π
`%e`	el número e
`%i`	la unidad imaginaria
`%phi`	la razón aúrea, $\frac{1+\sqrt{5}}{2}$

Podemos operar con ellas como con cualquier otro número.

```
(%i18)  (2+3*%i)*(5+3*%i);
(%o18)  (3*%i+2)*(3*%i+5)
```

Evidentemente necesitamos alguna manera de indicar a *Maxima* que debe desarrollar los productos, pero eso lo dejaremos para más tarde.

¿Cuál era el resultado anterior?

%	último resultado
%i*número*	entrada *número*
%o*número*	resultado *número*

Con *Maxima* podemos usar el resultado de una operación anterior sin necesidad de teclearlo. Esto se consigue con la orden %. No sólo podemos referirnos a la última respuesta sino a cualquier entrada o salida anterior. Para ello

```
(%i19)  %o15
(%o19)  √5
```

además podemos usar esa información como cualquier otro dato.

```
(%i20)  %o4+%o5;
                146
(%o20)  ---
                 7
```

1.2 Resultados exactos y aproximación decimal

Hay una diferencia básica entre el concepto abstracto de número real y cómo trabajamos con ellos mediante un ordenador: la memoria y la capacidad de proceso de un ordenador son finitos. La precisión de un ordenador es el número de dígitos con los que hace los cálculos. En un hipotético ordenador que únicamente tuviera capacidad para almacenar el primer decimal, el número π sería representado como 3.1. Esto puede dar lugar a errores si, por ejemplo, restamos números similares. *Maxima* realiza los cálculos de forma simbólica o numérica. En principio, la primera forma es mejor, pero hay ocasiones en las que no es posible.

Maxima tiene dos tipos de "números": exactos y aproximados. La diferencia entre ambos es la esperable. $\frac{1}{3}$ es un número exacto y 0.333 es una aproximación del anterior. En una calculadora normal todos los números son aproximados y la precisión (el número de dígitos con el que trabaja la calculadora) es limitada, usualmente 10 o 12 dígitos. *Maxima* puede manejar los números de forma exacta, por ejemplo

```
(%i21)  1/2+1/3;
                5
(%o21)  ---
                6
```

Mientras estemos utilizando únicamente números exactos, *Maxima* intenta dar la respuesta de la misma forma. Ahora bien, en cuanto algún término sea aproximado el resultado final será siempre aproximado. Por ejemplo

```
(%i22)  1.0/2+1/3;
(%o22)  0.83333333333333
```

Este comportamiento de *Maxima* viene determinado por la variable `numer` que tiene el valor `false` por defecto. En caso de que cambiemos su valor a `true`, la respuesta de *Maxima* será aproximada. **numer**

```
(%i23)  numer;
```

```
(%o23)  false
(%i24)  numer:true$
(%i25)  1/2+1/3
(%o25)  0.83333333333333
(%i26)  numer:false$
```

Recuerda cambiar el valor de la variable `numer` a `false` para volver al comportamiento original de *Maxima*. En *wxMaxima*, podemos utilizar el menú **Numérico→conmutar salida numérica** para cambiar el valor de la variable `numer`.

`float(`*número*`)`	expresión decimal de *número*
número`,numer`	expresión decimal de *número*
`bfloat(`*número*`)`	expresión decimal larga de *número*

Si sólo queremos conocer una aproximación decimal de un resultado exacto, tenemos a nuestra disposición las órdenes `float` y `bfloat`.

```
(%i27)  float(sqrt(2));
(%o27)  1.414213562373095
```

En la ayuda de *Maxima* podemos leer

> `Valor por defecto: 16.`
> `La variable ‘fpprec’ guarda el número de dígitos significativos en la aritmética con números decimales de punto flotante grandes ("bigfloats"). La variable ‘fpprec’ no afecta a los cálculos con números decimales de punto flotante ordinarios.`

Maxima puede trabajar con cualquier precisión. Dicha precisión la podemos fijar asignando el valor que queramos a la variable `fpprec`. Por ejemplo, podemos calcular cuánto valen los 100 primeros decimales de π:

```
(%i28)  fpprec:100;
(%o28)  100
(%i29)  float(%pi);
(%o29)  3.141592653589793
```

No parece que tengamos 100 dígitos...de acuerdo, justo eso nos decía la ayuda de máxima: “La variable `fpprec` no afecta a los cálculos con números decimales de punto flotante ordinarios”. Necesitamos la orden `bfloat` para que *Maxima* nos muestre todos los decimales pedidos (y cambiar la pantalla a `ascii`):

```
(%i30)  bfloat(%pi);
(%o30)  3.14159265358979323846264338327[43 digits]628620899862803482
        53421170679b0
(%i31)  set_display('ascii)$
```

```
(%i32)  bfloat(%pi);
(%o32)  3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 8b0
```

Si te has fijado, en la salida anterior la expresión decimal del número π termina con "b0". Los números en coma flotante grandes siempre terminan con "b" seguido de un número n para indicar que debemos multiplicar por 10^n. En el caso anterior, la expresión decimal de π deberíamos multiplicarla por $10^0 = 1$.

Por último, observa que, como se puede ver en la Figura 1.2, también se puede utilizar el menú **Numérico→A real** o **Numérico→A real grande(bigfloat)** para obtener la expresión decimal buscada.

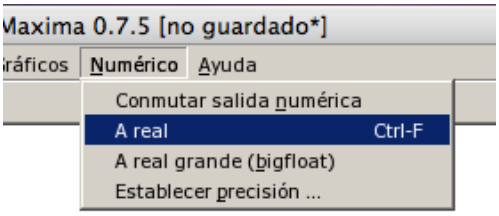


Figura 1.2 En el menú, la pestaña **Numérico** permite obtener la expresión decimal con la precisión que se desee

1.3 Funciones usuales

Además de las operaciones elementales que hemos visto, *Maxima* 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 en castellano. Por ejemplo, ya hemos visto raíces cuadradas

```
(%i33)  sqrt(4);
(%o33)  2
```

`sqrt(x)`	raíz cuadrada de x
`exp(x)`	exponencial de x
`log(x)`	logaritmo neperiano de x
`sin(x), cos(x), tan(x)`	seno, coseno y tangente *en radianes*
`csc(x), sec(x), cot(x)`	cosecante, secante y cotangente *en radianes*
`asin(x), acos(x), atan(x)`	arcoseno, arcocoseno y arcotangente
`sinh(x), cosh(x), tanh(x)`	seno, coseno y tangente hiperbólicos
`asinh(x), acosh(x), atanh(x)`	arcoseno, arcocoseno y arcotangente hiperbólicos

Potencias, raíces y exponenciales

Hemos visto que podemos escribir potencias utlizando (^) o (**). No importa que el exponente sea racional. En otras palabras: podemos calcular raíces de la misma forma

```
(%i34)  625^(1/4);
(%o34)  5
(%i35)  625^(1/3)*2^(1/3):
(%o35)  2^(1/3) 5^(4/3)
```

En el caso particular de que la base sea el número e, podemos escribir

```
(%i36)  %e^2;
```

```
(%o36)  %e^2
```

o, lo que es más cómodo especialmente si el exponente es una expresión más larga, utilizar la función exponencial `exp`

```
(%i37)  exp(2);
(%o37)  %e^2
(%i38)  exp(2),numer;
(%o38)  7.38905609893065
```

Logaritmos

Maxima sólo tiene la definición del logaritmo neperiano o natural que se consigue con la orden `log`:

```
(%i39)  log(20);
(%o39)  log(20)
```

y si lo que nos interesa es su expresión decimal

```
(%i40)  log(20),numer;
(%o40)  2.995732273553991
```

Observación 1.3. Mucho cuidado con utilizar `ln` para calcular logaritmos neperianos:

```
(%i41)  ln(20);
(%o41)  ln(20)
```

puede parecer que funciona igual que antes pero en realidad *Maxima* no tiene la más remota idea de lo que vale, sólo está repitiendo lo que le habéis escrito. Si no te lo crees, pídele que te diga el valor:

```
(%i42)  ln(20),numer;
(%o42)  ln(20)
```

¿Cómo podemos calcular $\log_2(64)$? Para calcular logaritmos en cualquier base podemos utilizar que

$$\log_b(x) = \frac{\log(x)}{\log(b)}.$$

Se puede definir una función que calcule los logaritmos en base 2 de la siguiente manera

```
(%i43)  log2(x):=log(x)/log(2)$
(%i44)  log2(64);
(%o44)  log(64)/log(2)
```

Te habrás dado cuenta de que *Maxima* no desarrolla ni simplifica la mayoría de las expresiones. En segundo lugar, la posibilidad de definir funciones a partir de funciones conocidas nos abre una amplia gama de posibilidades. En el segundo capítulo veremos con más detalle cómo trabajar con funciones.

Funciones trigonométricas e hiperbólicas

Maxima tiene predefinidas las funciones trigonométricas usuales seno, `sin`, coseno, `cos`, y tangente, `tan`, que devuelven, si es posible, el resultado exacto.

```
(%i45)  sin(%pi/4)
(%o45)  1/sqrt(2)
```

Por defecto, las funciones trigonométricas están expresadas en radianes.

También están predefinidas sus inversas, esto es, arcoseno, arcocoseno y arcotangente, que se escriben respectivamente `asin(x)`, `acos(x)` y `atan(x)`, así como las funciones recíprocas secante, `sec(x)`, cosecante, `csc(x)`, y cotangente, `cot(x)`[2].

```
(%i46)  atan(1);
(%o46)  %pi/4
(%i47)  sec(0);
(%o47)  1
```

De forma análoga, puedes utilizar las correspondientes funciones hiperbólicas.

Otras funciones

Además de las anteriores, hay muchas más funciones de las que *Maxima* conoce la definición. Podemos, por ejemplo, calcular factoriales

```
(%i48)  32!
(%o48)  263130836933693530167218012160000000
```

o números binómicos

```
(%i49)  binomial(10,4);
(%o49)  210
```

¿Recuerdas cuál es la definición de $\binom{m}{n}$?

$$\binom{m}{n} = \frac{m(m-1)(m-2)\cdots(m-(n-1))}{n!}$$

En el desarrollo de Taylor de una función veremos que estos números nos simplifican bastante la notación.

[2] El número π no aparece como tal por defecto en *wxMaxima*. Para que aparezca así, puedes marcar **Usar fuente griega** dentro de **Preferencias→Estilo**.

`n!`	factorial de n
`entier(x)`	parte entera de x
`abs(x)`	valor absoluto o módulo de x
`random(x)`	devuelve un número aleatorio
`signum(x)`	signo de x
`max(`x_1`,`x_2`,...)`	máximo de x_1,x_2,...
`min(`x_1`,`x_2`,...)`	mínimo de x_1,x_2,...

random Una de las funciones que usaremos más adelante es `random`. Conviene comentar que su comportamiento es distinto dependiendo de si se aplica a un número entero o a un número decimal, siempre positivo, eso sí. Si el número x es natural, `random(x)` devuelve un natural menor o igual que $x-1$.

```
(%i50)  random(100);
(%o50)  7
```

Obviamente no creo que tú también obtengas un 7, aunque hay un caso en que sí puedes saber cuál es el número "aleatorio" que vas a obtener:

```
(%i51)  random(1);
(%o51)  0
```

efectivamente, el único entero no negativo menor o igual que $1-1$ es el cero. En el caso de que utilicemos números decimales `random(x)` nos devuelve un número decimal menor que x. Por ejemplo,

```
(%i52)  random(1.0);
(%o52)  0.9138095996129
```

nos da un número (decimal) entre 0 y 1.

La lista de funciones es mucho mayor de lo que aquí hemos comentado y es fácil que cualquier función que necesites esté predefinida en *Maxima*. En la ayuda del programa puedes encontrar la lista completa.

1.4 Operadores lógicos y relacionales

Maxima puede comprobar si se da una igualdad (o desigualdad). Sólo tenemos que escribirla y nos dirá qué le parece:

```
(%i53)  is(3<5);
(%o53)  true
```

`is(`*expresión*`)`	decide si la expresión es cierta o falsa
`assume(`*expresión*`)`	supone que la expresión es cierta
`forget(`*expresión*`)`	olvida la expresión
`and`	y
`or`	o

No se pueden encadenar varias condiciones. No se admiten expresiones del tipo $3 < 4 < 5$. Las desigualdades sólo se aplican a parejas de expresiones. Lo que sí podemos hacer es combinar varias cuestiones como, por ejemplo,

```
(%i54)  is(3<2 or 3<4);
(%o54)  true
```

En cualquier caso tampoco esperes de *Maxima* la respuesta al sentido de la vida:

```
(%i55)  is((x+1)^2=x^2+2*x+1);
(%o55)  false
```

=	igual
notequal	distinto
x>y	mayor
x<y	menor
x>=y	mayor o igual
x<=y	menor o igual

Pues no parecía tan difícil de responder. Lo cierto es que *Maxima* no ha desarrollado la expresión. Vamos con otra pregunta fácil:

```
(%i56)  is((x+1)^2>0);
(%o56)  unknown
```

Pero, ¿no era positivo un número al cuadrado? Hay que tener en cuenta que x podría valer -1 . ¿Te parece tan mala la respuesta ahora? Si nosotros disponemos de información adicional, siempre podemos "ayudar". Por ejemplo, si sabemos que x es distinto de -1 la situación cambia:

```
(%i57)  assume(notequal(x,-1));
(%o57)  [notequal(x,-1)]
(%i58)  is((x+1)^2>0);
(%o58)  true
```

Eso sí, en este caso *Maxima* presupone que x es distino de -1 en lo que resta de sesión. Esto puede dar lugar a errores si volvemos a utilizar la variable x en un ambiente distinto más adelante. El comando `forget` nos permite "hacer olvidar" a *Maxima*. **forget**

```
(%i59)  forget(notequal(x,-1));
(%o59)  [notequal(x,-1)]
(%i60)  is(notequal(x,-1));
(%o60)  unknown
```

1.5 Variables

El uso de variables es muy fácil y cómodo en *Maxima*. Uno de los motivos de esto es que no hay que declarar tipos previamente. Para asignar un valor a una variable utilizamos los dos puntos

```
(%i61)  a:2
(%o61)  2
(%i62)  a^2
(%o62)  4
```

`var:expr`	la variable *var* vale *expr*
`kill(a1,a2,...)`	elimina los valores
`remvalue(var1,var2,...)`	borra los valores de las variables
`values`	muestra las variables con valor asignado

Cuando una variable tenga asignado un valor concreto, a veces diremos que es una constante, para distinguir del caso en que no tiene ningún valor asignado.

Observación 1.4. El nombre de una variable puede ser cualquier cosa que no empiece por un número. Puede ser una palabra, una letra o una mezcla de ambas cosas.

```
(%i63)  largo:10;
(%o63)  10
(%i64)  ancho:7;
(%o64)  7
(%i65)  largo*ancho;
(%o65)  70
```

Podemos asociar una variable con prácticamente cualquier cosa que se nos ocurra: un valor numérico, una cadena de texto, las soluciones de una ecuación, etc.

```
(%i66)  solucion:solve(x^2-1=0,x);
(%o66)  [x=-1,x=1]
```

para luego poder usarlas.

Los valores que asignamos a una variable no se borran por sí solos. Siguen en activo mientras no los cambiemos o comencemos una nueva sesión de *Maxima*. Quizá por costumbre, todos tendemos a usar como nombre de variables x, y, z, t, igual que los primeros nombres que se nos vienen a la cabeza de funciones son f o g. Después de trabajar un rato con *Maxima* es fácil que usemos una variable que ya hemos definido antes. Es posible que dar un valor a una variable haga que una operación posterior nos de un resultado inesperado o un error. Por ejemplo, damos un valor a x

```
(%i67)  x:3;
(%o67)  3
```

y después intentamos derivar una función de x, olvidando que le hemos asignado un valor. ¿Cuál es el resultado?

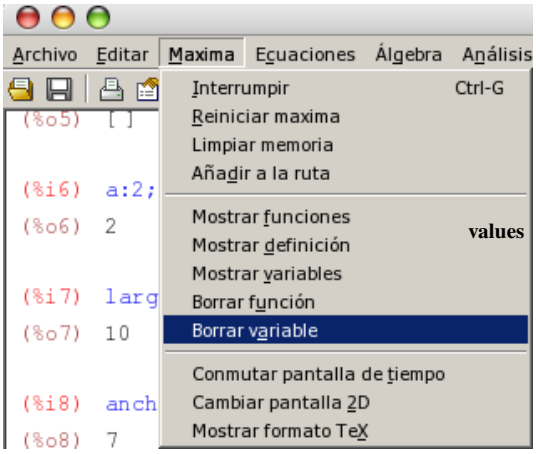
```
(%i68)   diff(sin(x),x);
         Non-variable 2nd argument to diff:
         3
         -- an error. To debug this try debugmode(true);
```

Efectivamente, un error. Hay dos maneras de evitar esto. La primera es utilizar el operador comilla, ', que evita que se evalúe la variable:

```
(%i69)   diff(sin('x),'x);
(%o69)   cos(x)
```

La segunda es borrar el valor de x. Esto lo podemos hacer con la orden `kill` o con la orden `remvalue`. También puedes ir al menú **Maxima→borrar variable** y escribir las variables que quieres borrar. Por defecto se borrarán todas.

Si te fijas, dentro del menú **Maxima** también hay varios ítems interesantes: se pueden borrar funciones y se pueden mostrar aquellas variables (y funciones) que tengamos definidas. Esto se consigue con la orden `values`.



values

```
(%i70)   values;
(%o70)   [a,largo,ancho,x,solucion]
```

Una vez que sabemos cuáles son, podemos borrar algunas de ellas

remvalue

```
(%i71)   remvalue(a,x);
(%o71)   [a,x]
```

o todas.

```
(%i72)   remvalue(all);
(%o72)   [largo,ancho,solucion]
```

La orden `remvalue` sólo permite borrar valores de variables. Existen versiones similares para borrar funciones, reglas, etc. En cambio, la orden `kill` es la versión genérica de borrar valores de cualquier cosa.

kill

```
(%i73)   ancho:10$
(%i74)   kill(ancho);
(%o74)   done
(%i75)   remvalue(ancho);
```

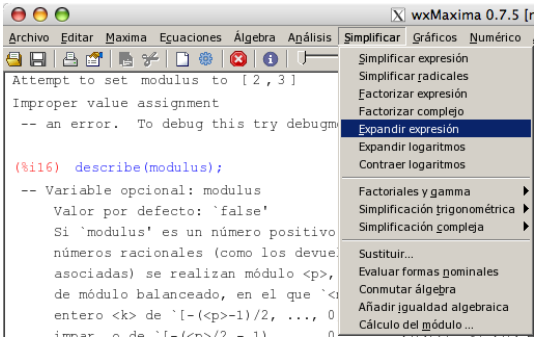
```
(%o75)   [false]
```

Una de las pequeñas diferencias entre `kill` y `remvalue` es que la primera no comprueba si la variable, o lo que sea, estaba previamente definida y siempre responde *done*. Existe también la posibilidad de borrar *todo*:

```
(%i76)   kill(all);
(%o0)    done
```

y, si te fijas, *Maxima* se reinicia: es como si empezáramos de nuevo. Hemos borrado cualquier valor que tuviésemos previamente definido.

1.6 Expresiones simbólicas



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

```
(%i1)   exp(log(x));
(%o1)   x
```

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

```
(%i2)   exp(x)*exp(y);
(%o2)   %e^(y+x)
```

Aunque parece que no siempre obtenemos el resultado esperado

```
(%i3)   log(x*y);
(%o3)   log(x y)
(%i4)   log(x)+log(y);
(%o4)   log(y)+log(x)
```

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

Casi todas las órdenes de esta sección, ya sea expandir o simplificar expresiones, se encuentran en el menú **Simplificar** y, opcionalmente, en los paneles de *wxMaxima*.

1.6.1 Desarrollo de expresiones simbólicas

La capacidad de *Maxima* para trabajar con expresiones es notable. Comencemos con funciones sencillas. Consideremos el polinomio

```
(%i5)  p:(x+2)*(x-1);
(%o5)  (x-1)(x+2)
```

lo único qué hace *Maxima* es reescribirlo. ¿Y las potencias?

```
(%i6)  q:(x-3)^2
(%o6)  (x-3)^2
```

Vale, tampoco desarrolla el cuadrado. Probemos ahora a restar las dos expresiones:

```
(%i7)  p-q;
(%o7)  (x-1)(x+2)-(x-3)^2
```

Si no había desarrollado las expresiones anteriores, no era lógico esperar que desarrollara ahora la diferencia. *Maxima* no factoriza ni desarrolla automáticamente: debemos decirle que lo haga. ¿Cómo lo hacemos?

`expand(`*expr*`)`	realiza productos y potencias
`partfrac(`*frac*`,`*var*`)`	descompone en fracciones simples
`num(`*frac*`)`	numerador
`denom(`*frac*`)`	denominador

La orden `expand` desarrollo productos, potencias, **expand**

```
(%i8)  expand(p);
(%o8)  x^2+x-2
```

y cocientes.

```
(%i9)  expand(p/q);
(%o9)  x^2/(x^2-6*x+9) + x/(x^2-6*x+9) - 2/(x^2-6*x+9)
```

Como puedes ver, `expand` sólo divide la fracción teniendo en cuenta el numerador. Si queremos dividir en fracciones simples tenemos que usar `partfrac`. **partfrac**

```
(%i10)  partfrac(p/q,x);
```

```
(%o10)   7/(x-3) + 10/(x-3^2)+1
```

num **denom**

Por cierto, también podemos recuperar el numerador y el denominador de una fracción con las órdenes `num` y `denom`:

```
(%i11)  denom(p/q);
(%o11)  (x-3)^2
(%i12)  num(p/q);
(%o12)  (x-1)(x+2)
```

Comportamiento de expand

El comportamiento de la orden `expand` viene determinado por el valor de algunas variables. No vamos a comentar todas, ni mucho menos, pero mencionar algunas de ellas nos puede dar una idea del grado de control al que tenemos acceso.

`expand(`*expr*`,n,m)`	desarrolla potencias con grado entre $-m$ y n
`logexpand`	variable que controla el desarrollo de logaritmos
`radexpand`	variable que controla el desarrollo de radicales

Si quisiéramos desarrollar la función

$$(x+1)^{100}+(x-3)^{32}+(x+2)^2+x-1-\frac{1}{x}+\frac{2}{(x-1)^2}+\frac{1}{(x-7)^{15}}$$

posiblemente no estemos interesados en que *Maxima* escriba los desarrollos completos de los dos primeros sumandos o del último. Quedaría demasiado largo en pantalla. La orden `expand` permite acotar qué potencias desarrollamos. Por ejemplo, `expand(`*expr*`,3,5)` sólo desarrolla aquellas potencias que estén entre 3 y -5.

```
(%i13)  expand((x+1)^100+(x-3)^32+(x+2)^2+x-1-1/x+2/((x-1)^2)
          +1/((x-7)^15),3,4);
(%o13)  2/(x^2-2x+1)+( x+1)^100+x^2+5x-1/x+(x-3)^32+1/(x-7)^15+3
```

Las variables `logexpand` y `radexpand` controlan si se simplifican logaritmos de productos o radicales con productos. Por defecto su valor es `true` y esto se traduce en que `expand` no desarrolla estos productos:

```
(%i14)  log(a*b);
(%o14)  log(a b)
(%i15)  sqrt(x*y)
(%o15)  sqrt(x y)
```

Cuando cambiamos su valor a `all`,

```
(%i16)  radexpand:all$ logexpand:all$
```

```
(%i17)  log(a*b);
(%o17)  log(a)+log(b)
(%i18)  sqrt(x*y)
(%o18)  sqrt(x)*sqrt(y)
```

Dependiendo del valor de `logexpand`, la respuesta de *Maxima* varía cuando calculamos $\log(a^b)$ o $\log(a/b)$.

Compara tú cuál es el resultado de $\sqrt{x^2}$ cuando `radexpand` toma los valores `true` y `all`.

Factorización

`factor(`*expr*`)`	escribe la expresión como producto de factores más sencillos

La orden `factor` realiza la operación inversa a `expand`. La podemos utilizar tanto en números **factor**

```
(%i19)  factor(100);
(%o19)  2^2 5^2
```

como con expresiones polinómicas como las anteriores

```
(%i20)  factor(x^2-1);
(%o20)  (x-1)(x+1)
```

El número de variables que aparecen tampoco es un problema:

```
(%i21)  (x-y)*(x*y-3*x^2);
(%o21)  (x-y)(xy-3x^2)
(%i22)  expand(%);
(%o22)  -xy^2+4x^2y-3x^3
(%i23)  factor(%);
(%o23)  -x(y-3x)(y-x)
```

Evaluación de valores en expresiones

`ev(`*expr*`,`*arg1*`,`*arg2*`,...)`	evalua la expresión según los argumentos

Ahora que hemos estado trabajando con expresiones polinómicas, para evaluar en un punto podemos utilizar la orden `ev`. En su versión más simple, esta orden nos permite dar un valor en una expresión: **ev**

```
(%i24)  ev(p,x=7);
(%o24)  54
```

que puede escribirse también de la forma

```
(%i25)  p,x=7;
(%o25)  54
```

También se puede aplicar `ev` a una parte de la expresión:

```
(%i26)  x^2+ev(2*x,x=3);
(%o26)  x^2+6
```

Este tipo de sustituciones se pueden hacer de forma un poco más general y sustituir expresiones enteras

```
(%i27)  ev(x+(x+y)^2-3*(x+y)^3,x+y=t);
(%o27)  x-3*t^3+t^2
```

En la ayuda de *Maxima* puedes ver con más detalle todos los argumentos que admite la orden `ev`, que son muchos.

1.6.2 Simplificación de expresiones

Es discutible qué queremos decir cuando afirmamos que una expresión es más simple o más sencilla que otra. Por ejemplo, ¿cuál de las dos siguientes expresiones te parece más sencilla?

```
(%i28)  radcan(p/q);
(%o28)  (x^2+x-2)/(x^2-6*x+9)
(%i29)  partfrac(p/q,x);
(%o29)  7/(x-3)+10/(x-3^2)+1
```

`radcan(`*expr*`)`	simplifica expresiones con radicales
`ratsimp(`*expr*`)`	simplifica expresiones racionales
`fullratsimp(`*expr*`)`	simplifica expresiones racionales

Maxima tiene algunas órdenes que permiten simplificar expresiones pero muchas veces no hay nada como un poco de ayuda y hay que indicarle si queremos desarrollar radicales o no, logaritmos, etc como hemos visto antes.

Para simplificar expresiones racionales, `ratsimp` funciona bastante bien aunque hay veces que es necesario aplicarlo más de una vez. La orden `fullratsimp` simplifica algo mejor a costa de algo más de tiempo y proceso.

```
(%i30)  fullratsimp((x+a)*(x-b)^2*(x^2-a^2)/(x-a));
(%o30)  x^4+(2a-2b)x^3+(b^2-4ab+a^2)x^2+(2ab^2-2a^2b)x+a^2b^2
```

Para simplificar expresiones que contienen radicales, exponenciales o logaritmos es más útil la orden `radcan`

```
(%i31)  radcan((%e^(2*x)-1)/(%e^x+1));
(%o31)  %e^x-1
```

1.6.3 Expresiones trigonométricas

Maxima conoce las identidades trigonométricas y puede usarlas para simplificar expresiones en las que aparezcan dichas funciones. En lugar de `expand` y `factor`, utilizaremos las órdenes `trigexpand`, `trigsimp` y `trigreduce`.

`trigexpand(`*expresion*`)`	desarrolla funciones trigonométricas e hipérbolicas
`trigsimp(`*expresion*`)`	simplifica funciones trigonométricas e hiperbólicas
`trigreduce(`*expresion*`)`	simplifica funciones trigonométricas e hiperbólicas

Por ejemplo,

```
(%i32)  trigexpand(cos(a+b));
(%o32)  cos(a)cos(b)-sin(a)sin(b);
(%i33)  trigexpand(sin(2*atan(x)));
(%o33)  2x/(x^2+1)
(%i34)  trigexpand(sin(x+3*y)+cos(2*z)*sin(x-y));
(%o34)  -(cos(x)sin(y)-sin(x)cos(y))(cos(z)^2-sin(z)^2)+cos(x)sin(3y)+
        sin(x)cos(3y)
(%i35)  trigexpand(8*sin(2*x)^2*cos(x)^3);
(%o35)  32 cos(x)^5 sin(x)^2
```

Compara el resultado del comando `trigexpand` con el comando `trigreduce` en la última expresión:

```
(%i36)  trigreduce(8*sin(2*x)^2*cos(x)^3);
(%o36)  8 (-(cos(7 x)/2 + cos(x)/2)/8 - 3 (cos(5 x)/2 + cos(3 x)/2)/8 + cos(3 x)/8 + 3 cos(x)/8)
```

Quizás es complicado ver qué está ocurriendo con estas expresiones tan largas. Vamos a ver cómo se comportan en una un poco más sencilla:

```
(%i37)  eq:cos(2*x)+cos(x)^2$
(%i38)  trigexpand(eq);
(%o38)  2cos(x)^2-sin(x)^2
(%i39)  trigreduce(eq);
```

```
(%o39)  (cos(2x)+1)/2+cos(2x)
(%i40)  trigsimp(eq);
(%o40)  cos(2x)+cos(x)^2
```

Como puedes ver, `trigsimp` intenta escribir la expresión de manera simple, `trigexpand` y `trigreduce` desarrollan y agrupan en términos similares pero mientras una prefiere usar potencias, la otra utiliza múltiplos de la variable. Estos es muy a grosso modo.

Cualquiera de estas órdenes opera de manera similar con funciones hiperbólicas:

```
(%i41)  trigexpand(sinh(2*x)^3);
(%o41)  8 cosh(x)^3 sinh(x)^3
(%i42)  trigreduce(cosh(x+y)+sinh(x)^2);
(%o42)  cosh(y+x)+(cosh(2x)-1)/2
```

Observación 1.5. Al igual que con `expand` o `ratsimp`, se puede ajustar el comportamiento de estas órdenes mediante el valor de algunas variables como `trigexpand`, `trigexpandplus` o `trigexpandtimes`. Consulta la ayuda de *Maxima* si estás interesado.

1.7 La ayuda de *Maxima*

El entorno *wxMaxima* permite acceder a la amplia ayuda incluida con *Maxima* de una manera gráfica. En el mismo menú tenemos algunos comandos que nos pueden ser útiles.

`describe(`*expr*`)`	ayuda sobre *expr*
`example(`*expr*`)`	ejemplo de *expr*
`apropos("`*expr*`")`	comandos relacionados con *expr*
`??`*expr*	comandos que contienen *expr*

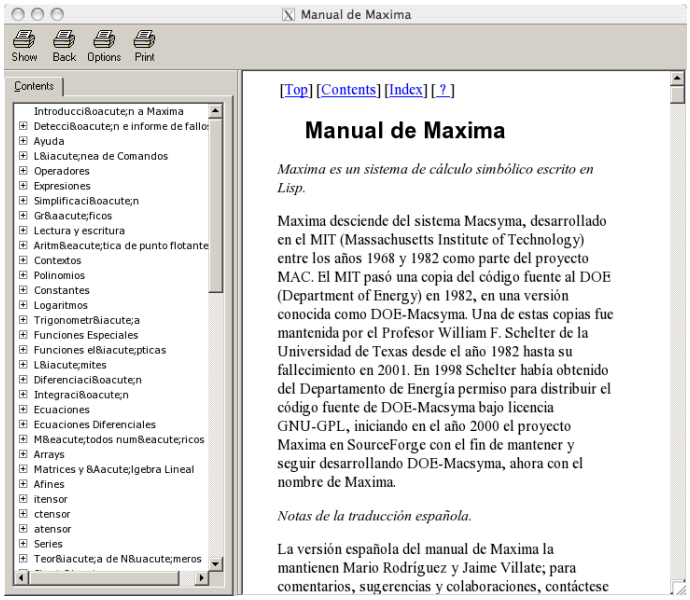


Figura 1.3 Ayuda de *wxMaxima*

En el caso de que conozcamos el nombre del comando sobre el que estamos buscando ayuda, la orden `describe`[3] nos da una breve, a veces no tan breve, explicación sobre la variable, comando o lo que sea que hayamos preguntado. **describe**

```
(%i43)  describe(dependencies);  -- Variable del sistema: dependencies
        Valor por defecto: '[]'
        La variable 'dependencies' es la lista de átomos que tienen algún
        tipo de dependencia funcional, asignada por 'depends' o 'gradef'.
        La lista 'dependencies' es acumulativa: cada llamada a 'depends' o
        'gradef' añade elementos adicionales.
        Véanse 'depends' y 'gradef'.
(%o43)  true
```

Claro que a veces nos equivocamos y no nos acordamos exactamente del nombre del comando

```
(%i44)  describe(plot);  No exact match found for topic 'plot'.
        Try '?? plot' (inexact match) instead.
(%o44)  false
```

La solución la tenemos escrita justo en la salida anterior: `??` busca en la ayuda comandos, variables, etc. que contengan la cadena "plot".

```
(%i45)  ??plot  0: Funciones y variables para plotdf
        1: Introducción a plotdf
        2: barsplot (Funciones y variables para gráficos estadísticos)
        3: boxplot (Funciones y variables para gráficos estadísticos)
        4: contour_plot (Funciones y variables para gráficos)
        5: gnuplot_close (Funciones y variables para gráficos)
        6: gnuplot_replot (Funciones y variables para gráficos)
        7: gnuplot_reset (Funciones y variables para gráficos)
        8: gnuplot_restart (Funciones y variables para gráficos)
        9: gnuplot_start (Funciones y variables para gráficos)
        10: plot2d (Funciones y variables para gráficos)
        11: plot3d (Funciones y variables para gráficos)
        12: plotdf (Funciones y variables para plotdf)
        13: plot_options (Funciones y variables para gráficos)
        14: scatterplot (Funciones y variables para gráficos estadísticos)
        15: set_plot_option (Funciones y variables para gráficos)
        Enter space-separated numbers, 'all' or 'none':none;
(%o45)  true
```

Si, como en este caso, hay varias posibles elecciones, *Maxima* se queda esperando hasta que escribimos el número que corresponde al ítem en que estamos interesados, o `all` o `none` si estamos interesados en todos o en ninguno respectivamente. Mientras no respondamos a esto no podemos realizar ninguna otra operación.

[3] El comando `describe(`*`expr`*`)` produce el mismo resultado que `?`*`expr`*. Es obligatorio el espacio en blanco entre la interrogación y la expresión.

apropos Si has mirado en el menú de *wxMaxima*, seguramente habrás visto **Ayuda→A propósito**. Su propósito es similar a las dos interrogaciones, `??`, que acabamos de ver pero el resultado es levemente distinto:

```
(%i46)  apropos("plot");
(%o46)  [plot,plot2d,plot3d,plotheight,plotmode,plotting,plot_format,
         plot_options,plot_realpart]
```

nos da la lista de comandos en los que aparece la cadena `plot` sin incluir nada más. Si ya tenemos una idea de lo que estamos buscando, muchas veces será suficiente con esto.

example Muchas veces es mejor un ejemplo sobre cómo se utiliza una orden que una explicación "teórica". Esto lo podemos conseguir con la orden `example`.

```
(%i47)  example(limit);
(%i48)  limit(x*log(x),x,0,plus)
(%o48)  0
(%i49)  limit((x+1)^(1/x),x,0)
(%o49)  %e
(%i50)  limit(%e^x/x,x,inf)
(%o50)  ∞
(%i51)  limit(sin(1/x),x,0)
(%o51)  ind
(%o51)  done
```

Por último, la ayuda completa de *Maxima* está disponible en la página web de *Maxima*

http://maxima.sourceforge.net/es/

en formato PDF y como página web. Son más de 800 páginas que explican prácticamente cualquier detalle que se te pueda ocurrir.

1.8 Ejercicios

Ejercicio 1.1. Calcula
a) Los 100 primeros decimales del número e,
b) el logaritmo en base 3 de 164232032682606581462314678007092552 89.
c) el arcocoseno hiperbólico de 1,
d) el seno y el coseno de i, y
e) el logaritmo de -2.

Ejercicio 1.2.
a) ¿Qué número es mayor 1000^{999} o 999^{1000}?
b) Ordena de mayor a menor los números π, $\frac{73231844868435875}{37631844868435563}$ y $\cosh(3)/3$.

Ejercicio 1.3. Descompón la fracción $\frac{x^2-4}{x^5+x^4-2x^3-2x^2+x+1}$ en fracciones simples.

Ejercicio 1.4. Escribe $\text{sen}(5x)\cos(3x)$ en función de $\text{sen}(x)$ y $\cos(x)$.

Ejercicio 1.5. Comprueba si las funciones hiperbólicas y las correspondientes "arco"-versiones son inversas.