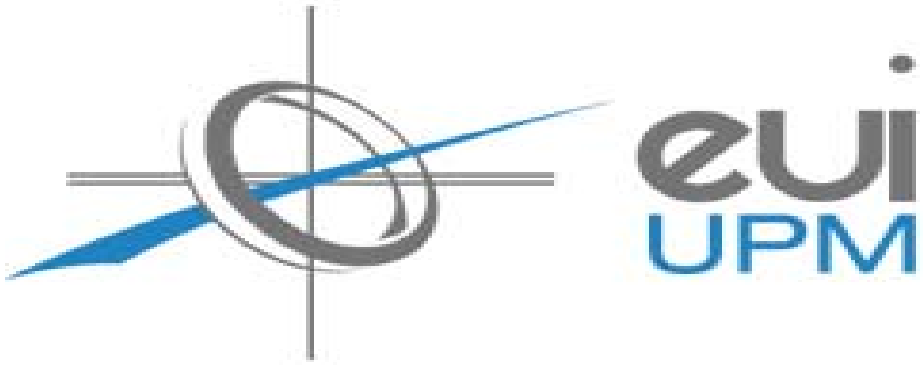
2011

Departamento de
Matemática Aplicada

EU Informática

1



# [MINI-MANUAL WXMAXIMA]

**Elaborado por: Rubén Haro Sanz**

**Supervisado por: Alfonsa García, Francisco García, Rafael Miñano y Blanca Ruiz**


**Trabajo parcialmente financiado por la UPM, proyecto IE10620102, del grupo GIEMATIC**

Mini-manual Maxima por Rubén Haro Sanz se encuentra bajo una Licencia Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported.
Permisos que vayan más allá de lo cubierto por esta licencia pueden encontrarse en http://www.eui.upm.es/escuela/dptos/ma

# ÍNDICE

8.4 Hallar el término general de una sucesión recursiva ( resolver ecuaciones en diferencias)

8.5 Series

9. Matrices

9.1 Definir una matriz

9.2 Recuperar elementos y submatrices

9.3 Operaciones con matrices

10. Programación en Maxima

# 0. ¿Cómo se usa esta guía?

Esta guía está diseñada bajo la premisa de que lo importante es aprender y hacer matemáticas, y que el programa ha de ser una herramienta. Por eso, no os aconsejamos leerla "de una vez", sino usarla según se va necesitando en el trabajo diario. Más en concreto, recomendamos que, cuando se quiera saber cómo hace con wxMaxima aquello que se necesita, se busque primero en el Índice, donde aparece todo lo que se explica en esta guía.

# 1. Introducción. Uso general del programa

## 1.1 Instalación

En primer lugar, vamos con la instalación del programa. El programa es de libre distribución y se puede encontrar en la siguiente dirección:

http://wxmaxima.sourceforge.net/wiki/index.php/Main_Page

Para instalar la versión wxMaxima 0.8.7, pulsamos en la parte inferior de la página que pone proyect page, y buscamos dicha versión del software.

Descargamos el paquete adecuado para nuestro sistema operativo, y lo instalamos. Ya tendríamos nuestro lugar de trabajo.

Además, el programa también está accesible en el servidor del departamento:

ftp://orio.eui.upm.es

(El usuario y la contraseña han de pedirse al profesorado de la asignatura.)

## 1.2 Configuración del programa

Antes de trabajar con wxMaxima, os aconsejaría cambiar unos aspectos de su configuración:

- Para que se ejecuten las instrucciones al pulsar la tecla INTRO, vamos al menú **Editar-Preferencias-Opciones** y activamos la casilla correspondiente a **Tecla de retorno evalúa celdas**, damos aceptar y listo. En caso contrario, para que se ejecute lo que escribimos tendremos que pulsar CTRL+INTRO al mismo tiempo.

- Para añadir a la plantilla un menú cómodo y accesible para realizar ciertas tareas , vamos a **Maxima-Paneles** y activamos: **Matemáticas generales**, **Barra de Herramientas** e **Insertar Celdas**.

Nuestro lugar de trabajo está listo para ser usado.

## 1.3 Pedir ayuda

Para utilizar la ayuda de wxMaxima, basta pulsar F1 y accederíamos a toda la ayuda disponible. Aparece un menú donde buscar la información necesaria.

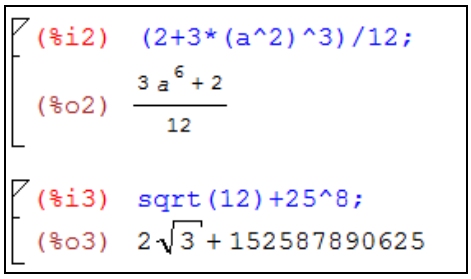
También hay una opción de menú y un icono ? en la barra de herramientas.

# 2. Operaciones básicas

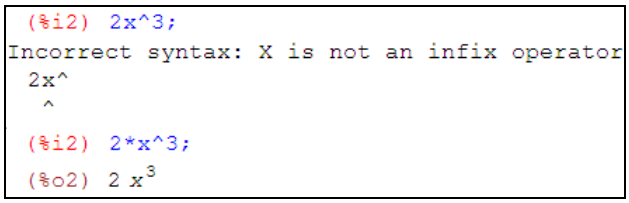
## 2.1 Operaciones aritméticas en modo exacto

El sistema wxMaxima utiliza la notación estándar para escribir las operaciones matemáticas básicas: suma [+], resta [-], producto [*], cociente [/] y potencia [^]. Tras introducir una expresión, basta pulsar INTRO para que Maxima lo simplifique.

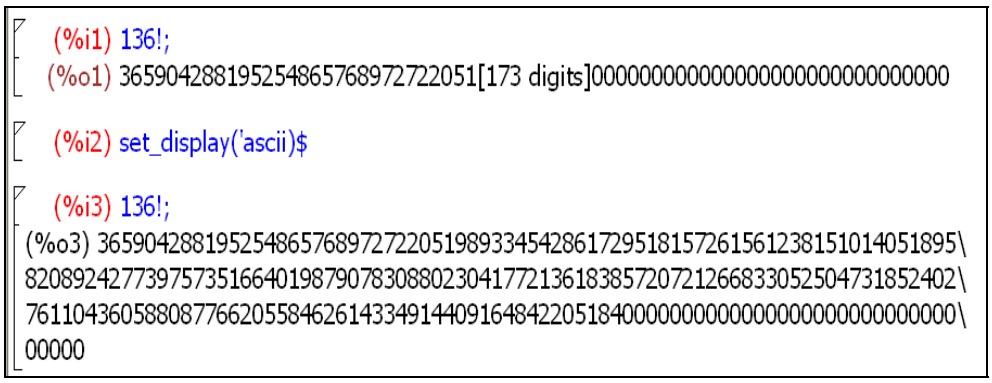
Ejemplo:




Es importante recordar que en Maxima es necesario escribir siempre el operador de la multiplicación (*). De lo contrario dará un error de sintaxis.




Maxima trabaja en aritmética exacta. Si el resultado de un cálculo es un número con muchos dígitos, en principio puede no mostrarlos todos. Pero podemos conseguir que los muestre con el menú **Maxima-Cambiar pantalla 2D**, sin más que elegir **ascii**.




## 2.2 Obtener el resultado aproximado de una operación

Para obtener el resultado aproximado, a la hora de evaluar la expresión en un valor determinado, añadiremos la directiva ***numer.***

Ejemplo:

```
(%i9)  1/5+sqrt(7);
(%o9)  sqrt(7)+1/5

(%i10)  1/5+sqrt(7),numer;
(%o10)  2.845751311064591
```

## *2.3 Introducir algunas funciones matemáticas elementales*

Señalamos la forma de escribir con wxMaxima algunas funciones matemáticas elementales:

- ✓ Para introducir $e^x$ se escribe exp(x) o %e^x
- ✓ Para introducir raíz de x se escribe sqrt(x).
- ✓ Para introducir |x| se escribe abs(x).
- ✓ Para introducir ln(x) se escribe log(x).
- ✓ Para introducir otro tipo de logaritmo, decimal por ejemplo, se escribe log10(x):=log(x)/log(10) o bien directamente log(x)/log(10)

## *2.4 Operar con números complejos*

Para trabajar con números complejos hay que tener en cuenta que:

- La unidad imaginaria se escribe **%i**
- Para la forma binómica (a+bi) hay que introducir a+b*%i
- Para la forma exponencial ( $re^{i\alpha}$ ) hay que introducir r*%e^(%i*$\alpha$).
- Para hallar el módulo de un número complejo z: cabs(z)
- Para hallar el argumento de un número complejo z: carg(z)
- Para hallar la forma binómica de z; rectform(z)
- Para hallar la forma exponencial de z: polarform(z)

Ejemplo:

```
(%i5)  2-2*%i;
(%o5)  2-2 %i
(%i6)  cabs(2-2*%i);
(%o6)  2^3/2
(%i7)  carg(2-2*%i);
(%o7)  -%pi/4
(%i8)  polarform(2-2*%i);
                    -%i %pi/4
(%o8)  2^3/2 %e
(%i9)  rectform(2^(3/2)*%e^(-(%i*%pi)/4));
(%o9)  2-2 %i
```

## *2.5 Asignar nombres a datos o expresiones*

Podemos asignar nombres utilizando ":", tal y como se muestra a continuación:

```
(%i8)  z:2-2*%i;
(%o8)  2-2 %i
(%i9)  cabs(z);
(%o9)  2^3/2
(%i10)  carg(z);
(%o10)  -%pi/4
(%i14)  k:%e^%i;
(%o14)  %e^%i
(%i15)  rectform(k);
(%o15)  %i sin(1)+cos(1)
```

## *2.6 Introducir comentarios*

En Maxima, un comentario es cualquier texto encerrado entre las marcas ***/* y */.***

Ejemplo:

```
(%i16) /*Apartado (a)*/ f(x):=x^2;
(%o16) f(x):=x^2
```

Otra forma de añadir comentarios es insertar una región para escribir texto, con la opción de menú **Cell / Insert Text Cell** o **F6**.

```
Apartado (a)

(%i17) f(x):=x^2;
(%o17) f(x):=x^2
```

# 3. Funciones básicas de Maxima

## 3.1 Introducir y manejar funciones

❖ *DEFINIR UNA FUNCIÓN*

Se puede definir la función con una o varias variables.

Ejemplo:

```
(%i9)  f(x):=2*x;
(%o9)  f(x):=2 x

(%i10)  g(x,y):=2*x^2+5*y^4;
(%o10)  g(x,y):=2 x^2+5 y^4
```

Es fundamental identificar las variables y usar **:=** para definirla.

Para definir una función a trozos se utiliza: ***If*** *condición* ***then*** *sentencia1* ***else*** *sentencia2*

Ejemplo:

La función $h(x)=\begin{cases} x & si\ x\le 0 \\ sen(x) & si\ x>0 \end{cases}$, se define

```
(%i26) h(x):=if x<=0 then x else sin(x);
(%o26)          h(x) := if x <= 0 then x else sin(x)
```

❖ *EVALUAR UNA FUNCIÓN*

Una vez definida la función, para evaluarla en x=a bastará con ejecutar f(a). A continuación, vamos a evaluar las funciones definidas anteriormente.

```
(%i12)  f(2);
(%o12)  4

(%i13)  g(2,1);
(%o13)  13

(%i15)  h(5),numer;
(%o15)  -0.95892427466314
```

## ❖ *MODIFICAR UNA FUNCIÓN*

Para modificar una función solo debemos situarnos sobre su definición, modificarla y compilarla de nuevo (INTRO). Una vez compilada, si volvemos a ejecutar cualquier expresión que la contenga, nos devolverá el valor actual de esa expresión. No importa que la expresión esté al principio, o entremedias.

Otra opción es escribirla de nuevo, ya que la última que compilemos, con un determinado nombre, es la función actual con la que se está trabajando y la que recuerda el sistema al pedir ese nombre.

## ❖ *BORRAR UNA FUNCIÓN Y DESASIGNAR*

ATENCIÓN, borrar una función de la pantalla, no significa quitarla de la memoria del sistema. Para hacer esto, vamos al menú a **Maxima-Borrar función**. Podemos elegir borrar todas las funciones, o bien dar el nombre de las que deseamos borrar.

Otra forma es utilizar el comando ***kill.***

- ***kill(f),*** elimina todas las asignaciones de la etiqueta ***f***.
- ***kill(all),*** elimina todas las asignaciones de todas las variables y funciones, pero no reinicia las variables globales a sus valores por defecto.

## ❖ *TABLA DE VALORES DE UNA FUNCIÓN*

Para obtener con wXMaxima varios valores de una expresión se utiliza la función ***makelist***, cuya sintaxis es ***makelist (expresión, variable, inicio, fin)***

Al ejecutar la instrucción anterior, se evalúa la expresión para los distintos valores de la variable, desde inicio hasta fin, con paso de longitud 1.

Ejemplo:

A continuación definimos una función f(x) y obtenemos sus valores para x=0, 1,...,10. Además, utilizamos ***numer***, para aproximar los resultados obtenidos:

```
(%i27) f(x):=5*x+sin(x);
(%o27) f(x):=5 x+sin(x)

(%i29) makelist(f(x),x,0,10),numer;
(%o29) [0.0,5.841470984807897,10.90929742682568,15.14112000805987,19.24319750469207,24.04107572533686,
29.72058450180107,35.65698659871879,40.98935824662338,45.41211848524176,49.45597888911063]
```

Si no queremos evaluar la variable en todos los valores de un rango sino solo en unos pocos, podemos usar una lista de valores. La sintaxis es:
***makelist(expresión,variable,[valor1,valor2,…valorn])***

Ejemplos:

```
(%i3)  makelist(f(x),x,[-%pi,1,3·%pi/2]);
(%o3)  [-5 %pi, sin(1)+5, 15 %pi/2 - 1]
```

Por otra parte, el primer argumento de ***makelist*** puede ser una lista o una condición. Un ejemplo de cada tipo se muestra a continuación:

```
(%i28)  makelist([x,f(x)],x,[-%pi,1,%pi/2]);
(%o28)  [[-%pi,-5 %pi],[1,sin(1)+5],[%pi/2, 5 %pi/2+1]]

(%i31)  makelist([x,is(f(x)>0)],x,-2,2);
(%o31)  [[-2,false],[-1,false],[0,false],[1,true],[2,true]]
```

## *3.2 Resolver ecuaciones*

El comando básico para resolver ecuaciones de todo tipo es ***solve***.

Ejemplo:

```
(%i31)  solve(4*x^2+x=5,x);
(%o31)  [x = -5/4, x = 1]
```

En algunos casos el sistema no sabe muy bien cómo resolver la ecuación y la devuelve sin cambios:

```
(%i6)  solve(sqrt(x-4)+x=6,x);
(%o6)  [x = 6 - sqrt(x-4)]
```

En estos casos, se puede "ayudar" al programa tal y como lo haríamos a mano. En este caso, dejando en un miembro la raíz ( $\sqrt{x-4} = 6 - x$ ) y elevando al cuadrado.

```
(%i32)  solve(x-4=(6-x)^2,x);
(%o32)  [x = 5, x = 8]
```

Las ecuaciones se pueden etiquetar en Maxima. Esto facilita mucho la labor de saber si el resultado que obtenemos con ***solve*** es efectivamente una solución o no, cosa que podemos hacer con ***subst***.

Ejemplo:

En primer lugar, definimos una ecuación, y posteriormente la resolvemos. Por último, aplicamos el comando mencionado para saber si la solución obtenida es la correcta o no.

```
(%i33)  eq1:x^3-x^2-x+1=0;
(%o33)  x^3 - x^2 - x + 1 = 0

(%i34)  solve(eq1,x);
(%o34)  [x = -1, x = 1]

(%i35)  subst(x=-1,eq1);
(%o35)  0 = 0
```

El comando ***solve*** puede resolver sistemas de ecuaciones (incluso no lineales). Para ello, hay que pasarle una lista con las ecuaciones a resolver. Las listas siempre se dan entre corchetes [ecuación1, ecuación2,...] y de nuevo hay que especificar la(s) variable(s) respecto de la(s) que se quiere resolver el sistema. Las soluciones, si existen, también se dan en forma de lista.

Ejemplo:

```
(%i39)  eq1:2*x+y=5;
(%o39)  y + 2 x = 5

(%i40)  eq2:2*x+5*y=10;
(%o40)  5 y + 2 x = 10

(%i41)  solve([eq1,eq2],[x,y]);
(%o41)  [[x = 15/8, y = 5/4]]
```

Otra forma de resolver ecuaciones o sistemas es utilizar la opción de menú:

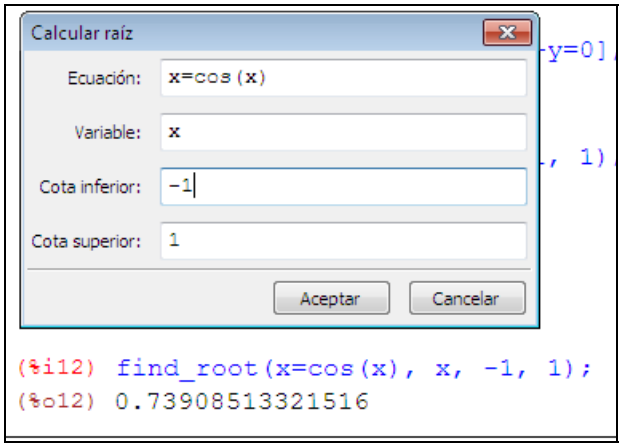
**Ecuaciones / Resolver** o **Ecuaciones / Resolver sistema**

- ***RESOLUCIÓN APROXIMADA DE ECUACIONES***

Cuando Maxima no puede calcular la solución exacta, devuelve la propia ecuación.

```
(%i13)  solve(x=cos(x),x);
(%o13)  [x = cos(x)]
```

Podemos buscar una solución aproximada en un determinado intervalo utilizando la opción de menú **Ecuaciones / Calcular raíz**.

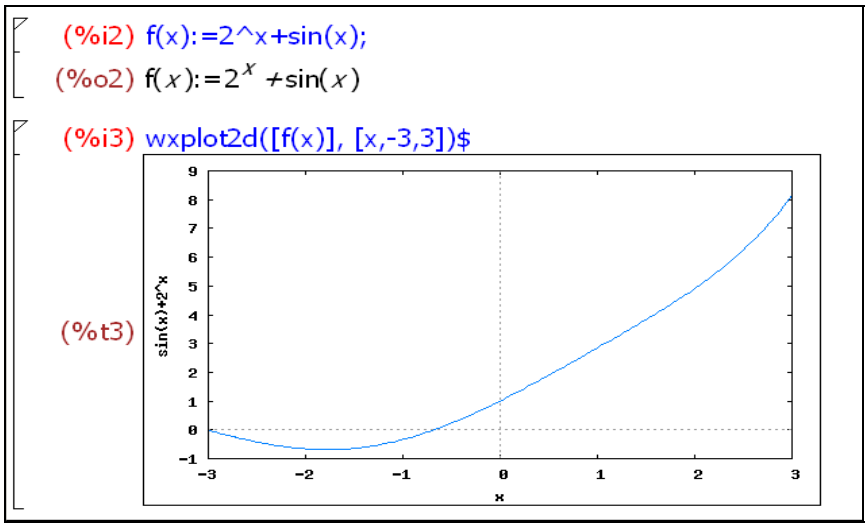



# 4. Representaciones gráficas

## 4.1 Funciones de una variable. Gráficas en 2D

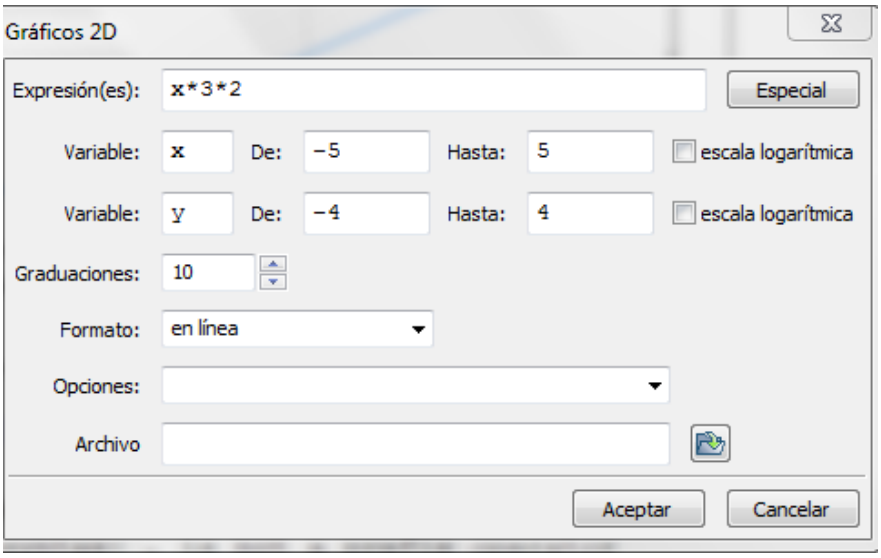
Se puede representar gráficamente una función, previamente definida, usando el botón correspondiente del menú de comandos, para ello se marca con el ratón la expresión a representar (no la asignación), es decir si queremos representar una función que tenemos definida en la forma $f(x) := 2^x + sin(x)$, marcaríamos solo $2^x + sin(x)$, o solo $f(x)$. Posteriormente, pinchamos en el menú de **Matemáticas generales** el botón **Gráfico 2D**. Aparece una pantalla para introducir el rango y el formato deseado y al pulsar **Aceptar** aparece el gráfico.

En la hoja de trabajo se mostrará lo siguiente:




Si queremos que el gráfico se abra en otra pestaña, tiene la ventaja de que te indica las coordenadas del punto marcado por el cursor en cada momento, deberemos elegir el formato: **openmath** o bien **gnuplot**. En este caso, es recomendable cerrar la ventana gráfica al terminar, para seguir trabajando.

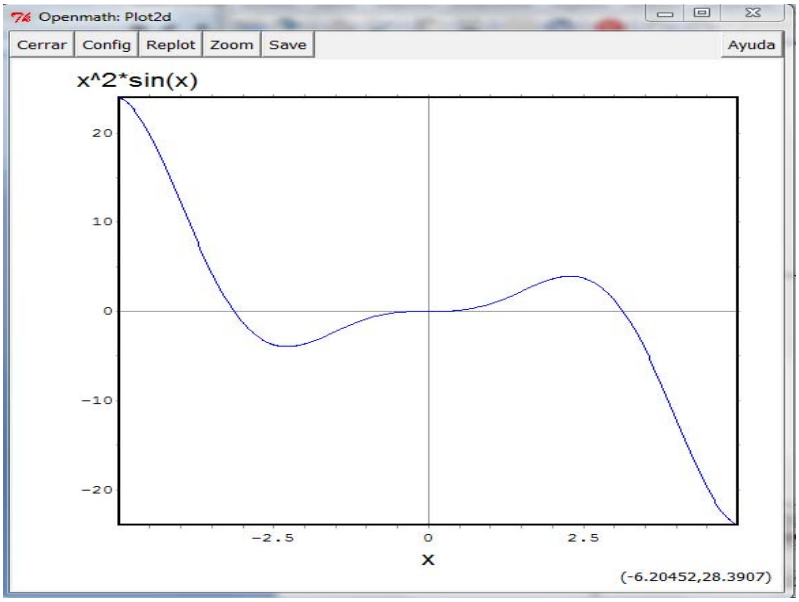
Otra forma de crear un gráfico 2D es introduciendo la expresión o el nombre de la función manualmente en la interfaz de los gráficos. Para ello, elegimos la opción de menú **Gráficos / Gráficos 2D**, se abre la ventana de diálogo en la que podemos introducir todos los datos, incluida la expresión a representar:




❖ *<u>TRABAJAR EN MODO GRÁFICO</u>*

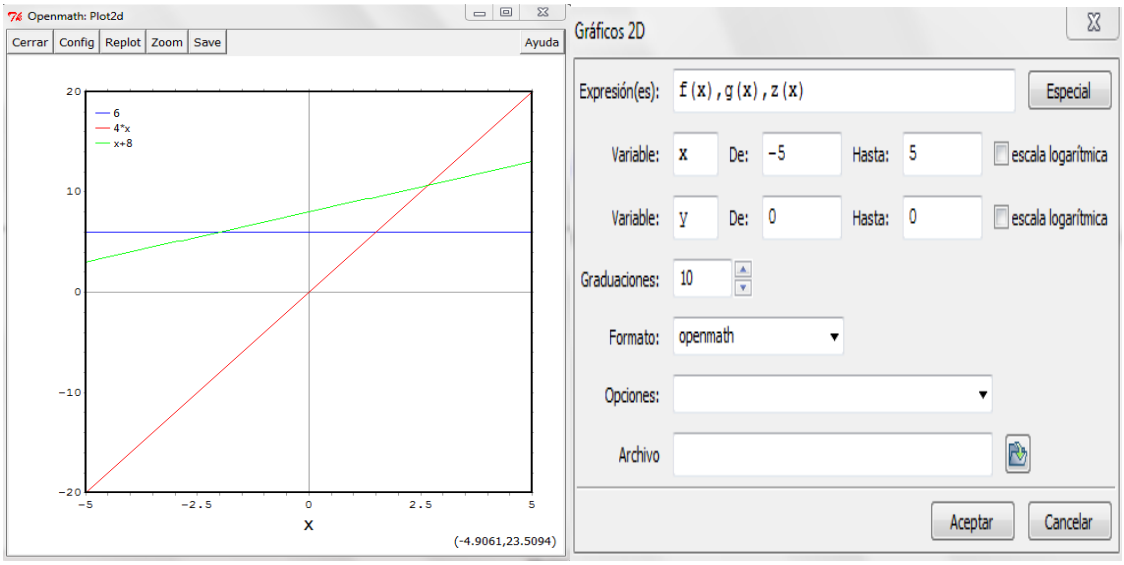
Para interactuar con un gráfico se debe elegir el formato: **openmath**.

En la ventana gráfica, además del menú de comandos propios de este tipo de ventanas, tenemos en la parte inferior las coordenadas del punto en que se encuentra el cursor. En la zona superior, aparece la función representada.




❖ *<u>REPRESENTAR VARIAS FUNCIONES SIMULTÁNEAMENTE</u>*

Una vez definidas las funciones, por ejemplo, f(x), g(x), y z(x), abrimos el menú de **Gráficos 2D** y las introducimos separadas por comas. Al pulsar **Aceptar** aparece la pantalla con todas las funciones pintadas.

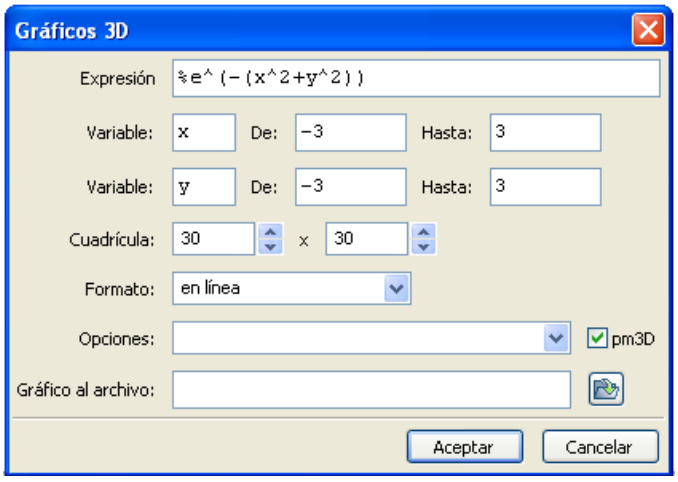



En la parte superior de la primera imagen se muestra la correspondencia entre funciones y colores de la gráfica, y abajo las coordenadas en donde esté situado el cursor.

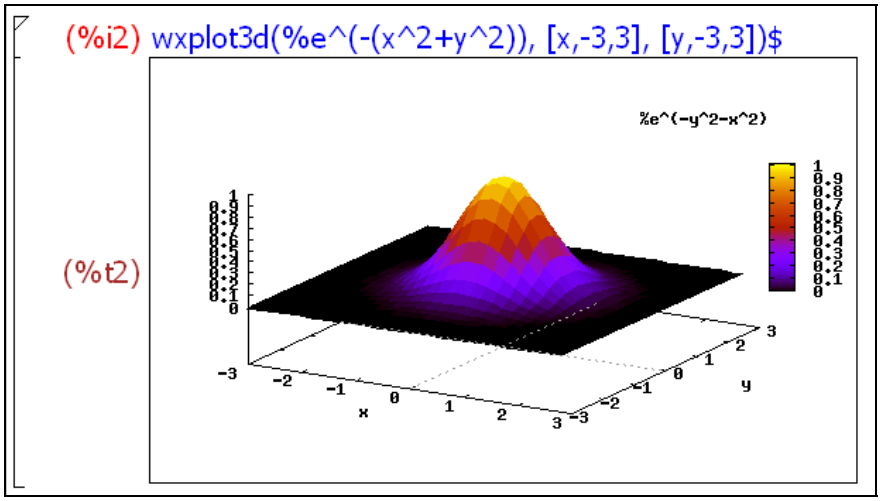
## *4.2 Funciones de dos variables. Gráficas en 3D*

La gráfica de una función de dos variables es una superficie, que se puede representar con el botón **Gráficos 3D**, introduciendo los datos en la ventana de diálogo, de modo análago a como se hace con una función de una variable

Ejemplo:

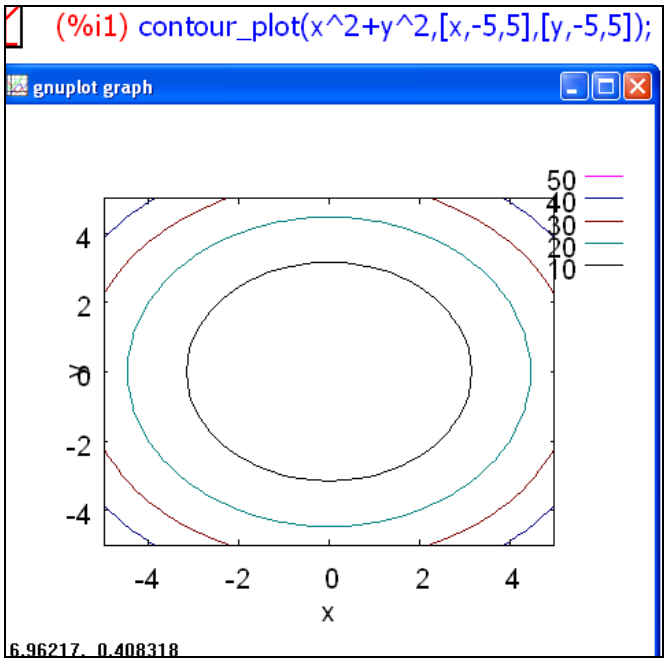



Al pulsar **Aceptar**, aparecerá en la hoja de trabajo la instrucción y el dibujo de la superficie:




También se pueden representar las curvas de nivel de una superficie utilizando la función **contour_plot**:

Ejemplo:




# 5. *Límites y Derivadas*

## *5.1 Límites*

El comando para hallar límites, ***limit***, es uno de los más sencillos de usar. Para calcular $\lim_{x \to a} f(x)$ se usa ***limit (f(x),x,a)***

Ejemplo:

```
(%i20)  f(x):=(x+a)*(x^2+b*x+c);
(%o20)  f(x):=(x+a)(x^2+b x+c)

(%i21)  limit(f(x),x,1);
(%o21)  (a+1)c+(a+1)b+a+1

(%i22)  limit(f(x),x,k);
(%o22)  k^3+(b+a)k^2+(c+a b)k+a c
```

Maxima identifica límites finitos e infinitos. También puede calcular límites laterales, con las directivas ***minus*** (límites por la izquierda) y ***plus*** (límites por la derecha).

Ejemplo:

```
(%i23)  limit(1/x,x,0,minus);
(%o23)  -∞

(%i24)  limit(1/x,x,0,plus);
(%o24)  ∞
```

Otra forma, de calcular un límite sería marcar la expresión deseada, y seleccionar el botón **Límite** de la barra de herramientas. Indicamos la variable (x), el punto (x0) y la dirección (Izquierda, Derecha o ambos), y pulsamos aceptar.

Ejemplo:


Aparece en pantalla el siguiente resultado:

```
(%i1)  limit(2^n/n!, n, inf);
(%o1)  0
```

## *5.2 Derivadas*

La instrucción para derivar respecto a una variable es ***diff (función, variable)***.

Ejemplo:

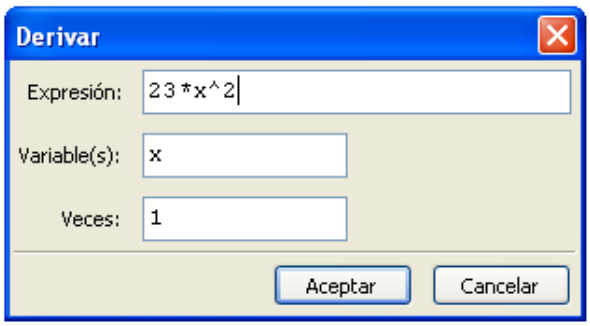
```
(%i29)  g(x,y):=sin(x*y);
(%o29)  g(x, y):= sin(x y)

(%i30)  diff(g(x,y),x);
(%o30)  y cos(x y)

(%i31)  diff(g(x,y),y);
(%o31)  x cos(x y)
```

Se pueden calcular derivadas segundas, terceras, etc., sin más que indicar el orden de derivación a continuación de la variable.

Ejemplo:

```
(%i32)  diff(g(x,y),x,2);
(%o32)  - y^2 sin(x y)

(%i33)  diff(g(x,y),x,4);
(%o33)  y^4 sin(x y)

(%i34)  diff(diff(g(x,y),y,2),x,2);
(%o34)  x^2 y^2 sin(x y)- 2 sin(x y)- 4 x y cos(x y)
```

Otra manera, sería marcar la expresión que queremos derivar, seleccionar el botón **Derivar** de la barra de herramientas, y en la ventana de diálogo, indicamos la variable (x) y el orden de la derivada que queremos calcular.



En la hoja de trabajo se muestra:

```
(%i1) diff(23·x^2,x,1);
(%o1) 46 x
```

## 6. *Polinomio de Taylor*

Para calcular el Polinomio de Taylor de una función f(x), de orden n en torno al punto x0, se utiliza la instrucción ***taylor (funcion, variable, punto,orden-del-polinomio).***

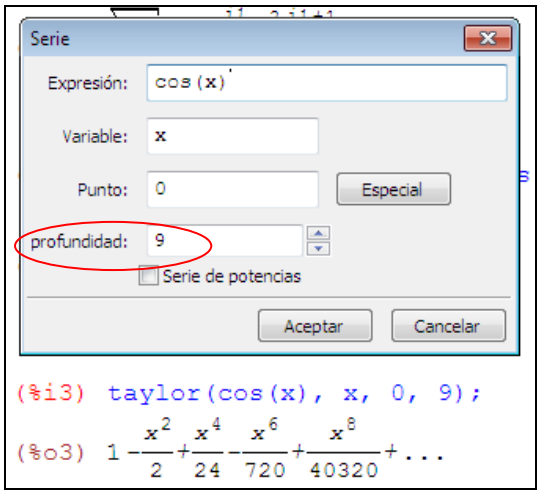
Ejemplo:

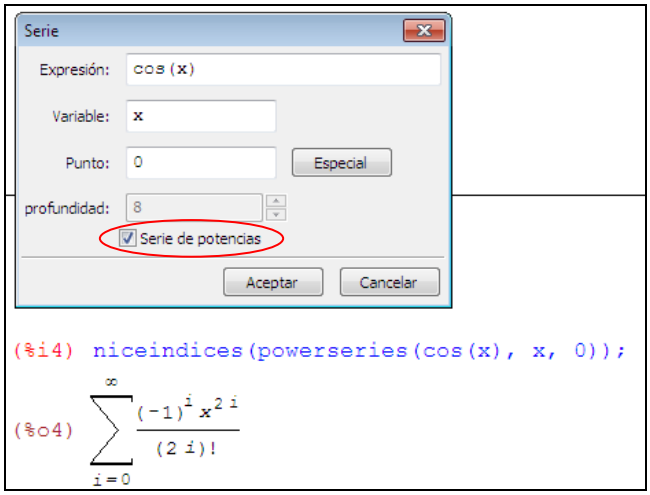
z

Por otra parte, como se ve en el ejemplo, al usar el comando **taylor** Maxima devuelve el polinomio de Taylor seguido de puntos suspensivos. Para que éstos no aparezcan bastará con utilizar la sentencia ***taytorat.*** El inconveniente de usar taytorat es que reduce a común denominador las fracciones. Si quiere recuperarse, por ejemplo para obtener una regla general, se puede usar ***expand (expresión)***.

```
(%i17)  f(x):=sin(x);
(%o17)  f(x):=sin(x)

(%i21)  taylor(f(x),x,0,8);
                                 3    5    7
                                x    x    x
(%o21)                   x - -- + --- - ---- + ...
                                6   120  5040

(%i22)  taytorat(taylor(f(x),x,0,8));
                                 7    5    3
                                x  - 42 x  + 840 x  - 5040 x
(%o22)                   - ------------------------------
                                        5040

(%i23)  expand(taytorat(taylor(f(x),x,0,8)));
                                 7    5    3
                                x    x    x
(%o23)                   - ---- + --- - -- + x
                                5040 120  6
```

También se puede obtener el polinomio de Taylor, y la serie de Taylor, mediante la opción de menú **Análisis / Calcular serie:**




## 7. *Ecuaciones diferenciales*

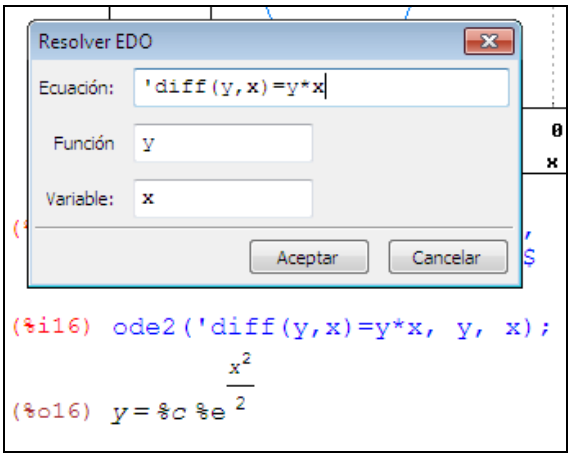
Para obtener la solución general de una Ecuación diferencial (de primer o segundo orden), utilizaremos la opción de menú **Ecuaciones / Resolver EDO**.

Un detalle importante, que hay que tener en cuenta al introducir una ecuación diferencial es que para escribir la derivada hay que poner un **acento grave en la izquierda de diff**. Así, por ejemplo:

- para escribir y' pondremos 'diff (y, x),
- para la derivada segunda y'' pondremos 'diff(y, x, 2).

Ejemplo:

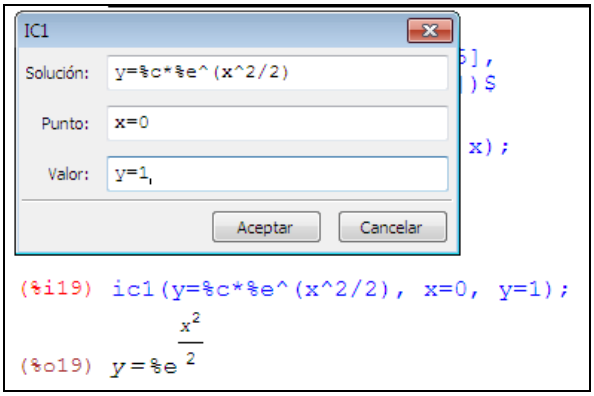
Para hallar la solución general de $y' = y \cdot x$, utilizamos **Ecuaciones / Resolver EDO**:




Si existen condiciones iniciales, una vez obtenida la solución general se podrá obtener la solución particular utilizando las opciones **Ecuaciones / Problema de valor inicial (1)** o **Ecuaciones / Problema de valor inicial (2)** según sea de primer o segundo orden respectivamente.

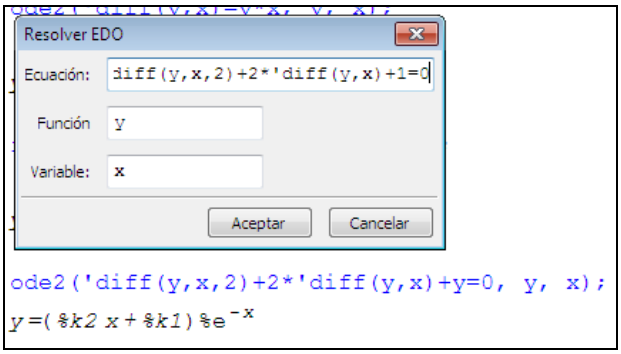
Ejemplo:

Para resolver el problema de valor inicial $y' = y \cdot x;\ y(0) = 1$, hallamos la solución general (como hemos visto antes) y utilizamos el menú **Ecuaciones** / **Problema de valor inicial (1)** :

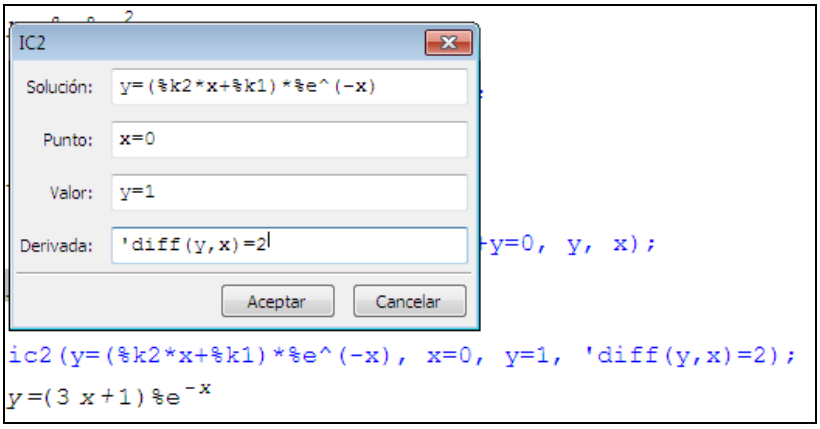



Si lo que tenemos es un problema de valor inicial de orden 2, el proceso sería análogo.

Por ejemplo, para resolver $y'' + 2y' + y = 0;\ y(0) = 1,\ y'(0) = 2$, utilizamos primero **Ecuaciones / Resolver EDO**:




Y luego **Ecuaciones** / **Problema de valor inicial (2)**:




También se pueden utilizar directamente las instrucciones:

***ode2 (ecu, v_d, v_i)*** (resolver ecuaciones de primer y segundo orden). El primer parámetro es la ecuación diferencial, el segundo el nombre de la variable dependiente y el tercero el de la variable independiente.

***ic1 ( So1_general, x=x0,y=y0)*** (problema de valor inicial de primer orden).

***ic2(So1_general,x=x0,y=y0,diff(y,x)=y'0)*** (problema de valor inicial de segundo orden).

# *8. Sucesiones y series de números reales*

## *8.1 Definir una sucesión en modo explícito*

Podemos definir una sucesión, conociendo su término general, de igual forma que lo haríamos con una función. Por ejemplo, si $a_n = \frac{1}{3^n}$, introducimos en la línea de comandos la instrucción: a(n):=1/3^n.

## 8.2 *Generar términos de una sucesión*

Para generar términos de la sucesión *a*(n), previamente definida, podemos utilizar la instrucción ***makelist(a(n), n,n_inicio,n_fin).***

Si queremos el resultado en modo aproximado, basta añadir al final la sentencia ***numer.***

Ejemplo:

```
(%i6)  a(n):=1/3^n;
(%o6)  a(n):=1/3^n

(%i8)  makelist(a(n),n,1,10);
(%o8)  [1/3,1/9,1/27,1/81,1/243,1/729,1/2187,1/6561,1/19683,1/59049]

(%i9)  makelist(a(n),n,1,10),numer;
(%o9)  [0.33333333333333,0.11111111111111,0.037037037037037,0.012345679012346,
0.0041152263374486,0.0013717421124829,4.5724737082761773 10^-4,1.5241579027587258 10^-4
,5.0805263425290857 10^-5,1.6935087808430286 10^-5]
```

## 8.3 *Definir sucesiones de forma recursiva*

Las sucesiones definidas recursivamente se pueden implementar en wxMaxima utilizando el operador **:** para asignar valores a los primeros elementos.

Ejemplos:

1. Introducimos la definición recursiva de la sucesión $a_n$=n! y calculamos $a_3$:

```
(%i10)  a[0]:1;
(%o10)  1

(%i11)  a[n]:=n*a[n-1];
(%o11)  a_n := n a_(n-1)

(%i12)  a[3];
(%o12)  6
```

La misma sucesión, también se puede definir como función usando ***if***:

```
(%i26) a(n):=if n=0 then 1 else n*a(n-1);
(%o26) a(n):=if n=0 then 1 else n a(n-1)
```

2. Definición de la sucesión de Fibonacci y obtención de los diez primeros términos:

Sucesión de Fibonacci

```
(%i11) f[0]:0;
(%o11) 0

(%i12) f[1]:1;
(%o12) 1

(%i13) f[n]:=f[n-1]+f[n-2];
(%o13) f_n := f_(n-1) + f_(n-2)

(%i14) makelist(f[n],n,1,10);
(%o14) [1,1,2,3,5,8,13,21,34,55]
```

## *8.4 Hallar el término general de una sucesión recursiva (Resolver ecuaciones en diferencias)*

Para hallar el término general de una sucesión definida de forma recursiva (resolver ecuaciones en diferencias) usando wxMaxima es preciso cargar previamente la librería ***solve_rec*.** Para ello hay que ejecutar ***load (solve_rec).***

Una vez cargada, para resolver una ecuación recurrente se usa la instrucción:

***solve_rec( ecuación,variable,valor inicial 1, valor inicial2, …., valor inicial n)***

Ejemplos:

1. Resolución de la ecuación $\begin{cases} x_1 = 2 \\ x_n - 2x_{n-1} = n \end{cases}$

```
(%i1)  load(solve_rec);
(%o1)  C:/ARCHIV~1/MAXIMA~1.1/share/maxima/5.18.1/share/contrib/solve_rec/solve_rec.mac

(%i2)  solve_rec(x[n]-2·x[n-1]=n,x[n],x[1]=2);
(%o2)  x_n = 2^n + 3 2^(n-1) - n - 2
```

2. Obtener el término general de la sucesión de Fibonacci:

$$x_n = \begin{cases} x_0 = 1; x_1 = 1 \\ x_n = x_{n-1} + x_{n-2} \end{cases}$$

```
(%i3)  solve_rec(x[n]=x[n-1]+x[n-2],x[n],x[0]=1,x[1]=1);
```

$$(\%o3)\quad x_n=\frac{(\sqrt{5}+1)^n(\sqrt{5}+5)}{10\,2^n}-\frac{(\sqrt{5}-5)(\sqrt{5}-1)^n(-1)^n}{10\,2^n}$$

## 8.5 Series

Con Maxima es posible obtener la suma de n sumandos o el valor numérico de la suma de algunas series, por ejemplo series geométricas. Para ello, disponemos de los comandos:

- ***sum(expr,n,m,p)***: suma ***expr*** usando ***n*** como variable, desde el valor ***m*** al ***p*** ( que puede ser infinito). Si no puede sumarla, la expresa como un sumatorio.
- ***nusum(expr,n,m,p)***: como ***sum*** pero emplea otro algoritmo más eficaz en expresiones racionales.
- ***load(simplify_sum):*** carga el paquete ***symply_sum***, el más potente de Maxima para sumar series.
- ***simplify_sum(serie):*** calcula la suma exacta de la serie o indica si la serie es divergente.

También podemos acceder a los comandos ***sum*** y ***nusum*** desde el menú **Análisis / Calcular Suma**.

Ejemplos:

Dada la serie $\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{n}$, podemos sumar los 20 primeros términos, y luego hallar su suma exacta:

```
(%i18) sum((-1)^(n+1)/n,n,1,20);
```

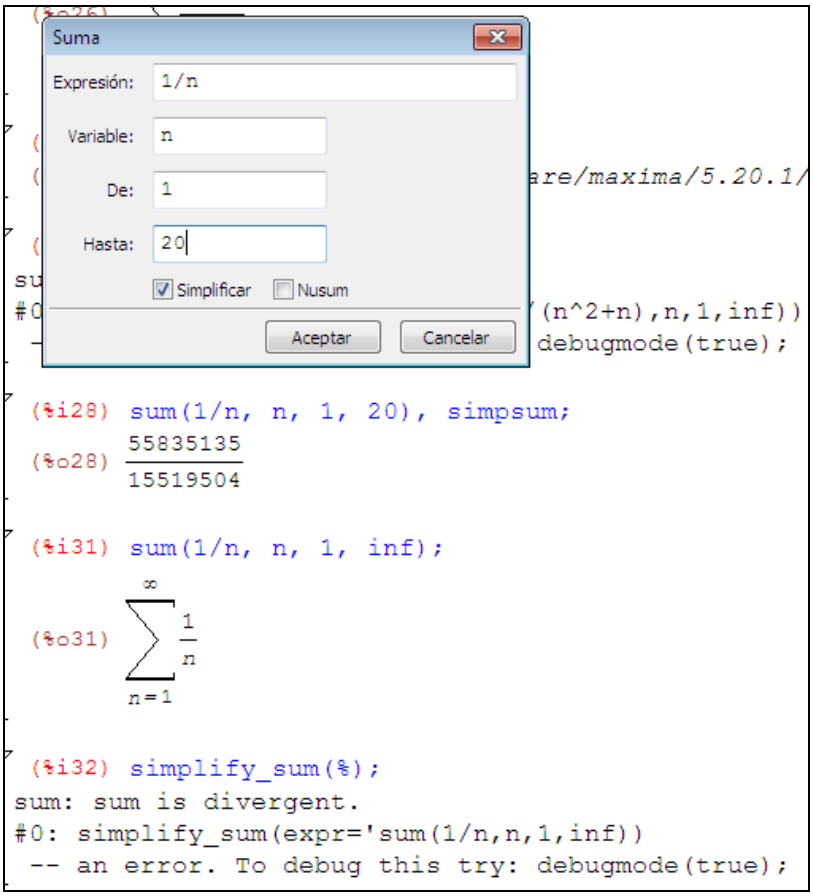
$$(\%o18)\quad \frac{155685007}{232792560}$$

```
(%i19) sum((-1)^(n+1)/n,n,1,20), numer;
(%o19) 0.66877140317543
(%i20) sum((-1)^(n+1)/n,n,1,inf);
```

$$(\%o20)\quad \sum_{n=1}^{\inf}\frac{(-1)^{n+1}}{n}$$

```
(%i23) load(simplify_sum);
(%o23)
C:/PROGRA~1/MAXIMA~2.1/share/maxima/5.20.1/share/contrib/solve_rec/simplify_sum.ma
(%i22) simplify_sum(%);
(%o22) log(2)
```

Con la serie divergente $\sum_{n=1}^{\infty}\frac{1}{n}$, hacemos lo mismo:




# 9. Matrices

## 9.1 Definir una matriz

Se pueden definir matrices de diferentes formas:

a) Utilizando las distintas opciones del **menú Álgebra:** introducir matriz, generar matriz a partir de expresión,…

b) Declarando sus elementos mediante listas, una para cada fila, con la instrucción ***matrix ([a11, a12, a13,...],[a21,a22,a23,...],...[an1,an2,...])***

c) Introduciendo interactivamente bajo demanda sus elementos con la instrucción ***entermatrix (NúmeroFilas,NúmeroColumnas)***

d) Mediante una fórmula que define el elemento genérico de la matriz: ***a[i,j]:=Fórmula de i y j$ genmatrix (a,NúmeroFilas,NúmeroColumnas)***

Ejemplo:

```
(%i2)  a:matrix([3,3,3],[1,3,5],[4,2,7]);
                [ 3  3  3 ]
                [         ]
(%o2)           [ 1  3  5 ]
                [         ]
                [ 4  2  7 ]

(%i3)  genmatrix(b,2,2);
                [ b1,1  b1,2 ]
                [            ]
(%o3)           [ b2,1  b2,2 ]

(%i4)  b[i,j]:=i^2+j^2;
                      2    2
(%o4)           bi,j := i  + j

(%i5)  genmatrix(b,2,2);
                [ 2  5 ]
                [      ]
(%o5)           [ 5  8 ]
```

Por otra parte, las matrices especiales, como las diagonales, simétricas, nulas o la identidad, pueden construirse utilizando el menú Álgebra **/ Introducir matriz**, o bien comandos específicos:

- ***diagmatrix(Número,Valor),*** que genera una matriz diagonal de orden *Número* con elementos no nulos en la diagonal, todos ellos con el mismo *Valor*
- ***ematrix(m,n,Z,i,j)***, que genera una matríz *mxn casi* nula en la que todos los elementos nulos salvo el (*i,j*) cuyo valor es *Z*
- ***zeromatrix(n,m)***, que genera la matriz nula de *n* filas y *m* columnas
- ***ident(n),*** que genera la matriz identidad *nxn.*

Ejemplo:

```
(%i19)  diagmatrix(4,5);
                [ 5  0  0  0 ]
                [             ]
                [ 0  5  0  0 ]
(%o19)          [             ]
                [ 0  0  5  0 ]
                [             ]
                [ 0  0  0  5 ]

(%i20)  ematrix(3,3,5,3,2);
                [ 0  0  0 ]
                [         ]
(%o20)          [ 0  0  0 ]
                [         ]
                [ 0  5  0 ]

(%i21)  zeromatrix(2,2);
                [ 0  0 ]
(%o21)          [      ]
                [ 0  0 ]

(%i22)  ident(5);
                [ 1  0  0  0  0 ]
                [               ]
                [ 0  1  0  0  0 ]
                [               ]
(%o22)          [ 0  0  1  0  0 ]
                [               ]
                [ 0  0  0  1  0 ]
                [               ]
                [ 0  0  0  0  1 ]
```

## *9.2 Recuperar elementos y submatrices*

Es posible asignar una matriz a una variable en Maxima, y luego extraer de forma independiente filas(row), columnas(column) u otro tipo de submatrices haciendo uso de los comandos siguientes:

- ***col(Matriz,NúmColumna)***, que recupera la columna cuyo número se indica.
- ***row(Matriz,NúmFila)***,que recupera la fila cuyo número se indica.
- ***Matriz[i,j]***, que recupera el elemento de la fila *i*, columna *j*.
- ***submatrix(i_1,i_2,...i_p, Matriz,j_1,j_2,...j_q)***, que elimina de la *Matriz* las filas cuyos números son *i_1...i_p* y las columnas cuyos números son *j_1...j_q*. Pueden eliminarse únicamente filas o columnas.
- ***addrow (Matriz, lista_1, ..., lista_p)***, que añade en la base de *Matriz* las filas dadas por las listas (o matrices) *lista_1, ..., lista_p*. Las longitudes deben ser concordantes.
- ***addcol (Matriz, lista_1, ..., lista_p)***, que añade a la derecha de *Matriz* las filas dadas por las listas (o matrices) *lista_1, ..., lista_p*. Las longitudes deben ser concordantes.

Ejemplos:

```
(%i1)  c[i,j]:=i^2+j^2;
(%o1)  c_{i,j} := i^2 + j^2

(%i2)  miMatriz:genmatrix(c,3,3);
(%o2)  matrix([2,5,10],[5,8,13],[10,13,18])

(%i3)  miMatriz[2,1]/*recupera el elemento [2,1]*/;
(%o3)  5

(%i4)  col(miMatriz,2)/*recupera la columna 2*/;
(%o4)  matrix([5],[8],[13])

(%i5)  row(miMatriz,2)/*recupera la fila 2*/;
(%o5)  matrix([5,8,13])
```

```
(%i7)  submatrix(2,miMatriz,1)/*elimina la fila 2 y columna 1*/;
(%o7)  matrix([5,10],[13,18])

(%i9)  addrow(miMatriz,[1,3,3])/*añade una nueva fila*/;
(%o9)  matrix([2,5,10],[5,8,13],[10,13,18],[1,3,3])

(%i12)  addcol(miMatriz,[2,3,3])/*añade una nueva columna*/;
(%o12)  matrix([2,5,10,2],[5,8,13,3],[10,13,18,3])
```

## *9.3 Operaciones con matrices*

Pueden realizarse diferentes operaciones con matrices usando los siguientes operadores:

- + suma de dos matrices
- - diferencia de dos matrices
- . producto ordinario de dos matrices
- * multiplicación de dos matrices, elemento a elemento, y también multiplicar por un número fijo todos los elementos
- / división de dos matrices, elemento a elemento
- ^^ elevar una matriz a una potencia
- ^ elevar cada uno de los elementos de una matriz a una potencia

Ejemplos:

```
(%i1) a:matrix([1,2,3],[3,2,1]); b:matrix([2,1,2],[2,3,1]);
(%o1) [1 2 3]
      [3 2 1]
(%o2) [2 1 2]
      [2 3 1]

(%i4) print("la suma es")$ a + b;
la suma es
(%o5) [3 3 5]
      [5 5 2]

(%i12) print("el producto elemento a elemento")$ a*b;
el producto elemento a elemento
(%o13) [2 2 6]
       [6 6 1]
```

Además, el **menú Álgebra** ofrece opciones para calcular la **transpuesta**, la **inversa**, el **determinante**, **polinomio característico, vectores y valores propios** de una matriz.

Todas esas operaciones, también pueden realizarse mediante comandos:

- ***transpose(NombreMatríz)***, que calcula la transpuesta

```
(%i3) a:matrix([1,2,3],[3,2,1]); transpose(a);
(%o3) [1 2 3]
      [3 2 1]
(%o4) [1 3]
      [2 2]
      [3 1]
```

- ***adjoint(NombreMatríz),*** que calcula la adjunta

```
(%i6)  b:matrix([2,1,2],[3,1,1],[2,1,2]); adjoint(b);
                                [2  1  2]
                                [       ]
(%o6)                           [3  1  1]
                                [       ]
                                [2  1  2]
                                [ 1   0  - 1 ]
                                [            ]
(%o7)                           [ - 4  0    4 ]
                                [            ]
                                [ 1   0  - 1 ]
```

- ***invert(NombreMatríz)***, que calcula la inversa utilizando el método de los adjuntos

```
(%i10)  c:matrix([2,1,2],[1,2,4],[3,1,1]); invert(c);
                                [2  1  2]
                                [       ]
(%o10)                          [1  2  4]
                                [       ]
                                [3  1  1]
                                [ 2    1      ]
                                [ -    -   0  ]
                                [ 3    3      ]
                                [              ]
                                [ 11   4      ]
(%o11)                          [ --   -   2  ]
                                [ 3    3      ]
                                [              ]
                                [ 5    1      ]
                                [ -    -   - 1 ]
                                [ 3    3      ]
```

- ***invert(NombreMatríz),detout***, que calcula la inversa con el determinante fuera

```
(%i12)  c:matrix([2,1,2],[1,2,4],[3,1,1]); invert(c),detout;
                                [2  1  2]
                                [       ]
(%o12)                          [1  2  4]
                                [       ]
                                [3  1  1]
                                [ - 2   1    0 ]
                                [             ]
                                [ 11   - 4  - 6 ]
                                [             ]
                                [ - 5   1    3 ]
(%o13)                          ---------------
                                       3
```

- ***determinant(NombreMatríz)***, que calcula el determinante de una matríz
- ***rank(NombreMatríz)***, que calcula el rango
- ***charpoly(NombreMatriz,x),*** que calcula el polinomio característico
- ***eigenvalues(NombreMatriz),*** que calcula los valores propios
- ***eigenvalues(NombreMatriz),*** que calcula los vectores propios

```
(%i35)  d:matrix([1,2,3],[1,3,1]); print("rango")$ rank(d);
                  [ 1  2  3 ]
(%o35)            [         ]
                  [ 1  3  1 ]
rango
(%o37)  2
```

## 10. *Programación en Maxima*

Maxima dispone de un lenguaje de programación propio que permite definir nuevas funcionalidades.

Ejemplos:

- Definir el área de un triangulo:

```
(%i38)  area_tri(base,altura):=base*altura/2;
                                          base altura
(%o38)  area_tri(base, altura):=  ----------------
                                          2
```

- Definir el área de un cuadrado

```
(%i39)  area_cua(lado):=lado^2;
                                  2
(%o39)  area_cua(lado):= lado
```

En el lenguaje de Maxima existen distintos tipos de expresiones y el proceso de programación consiste en la creación y manipulación de éstas para obtener nuestros objetivos.

❖ *BUCLES*

Para definir un bucle, lo más usual es utilizar la estructura "**for",** cuya sintaxis es:

***for*** *NombreVariable:valor inicial* ***thru*** *valor final de la variable* ***do*** *acción a realizar*

Ejemplo:

```
(%i13)  for b:0 thru 2 do print(b,"al cuadrado es igual a ",b^2);
0 al cuadrado es igual a  0
1 al cuadrado es igual a  1
2 al cuadrado es igual a  4
(%o13)  done
```

Además, la cantidad a incrementar la variable en cada etapa se puede determinar mediante la palaba clave "**step".**

Ejemplo:

```
(%i9)  for a:0 thru -5 step -2 do display(a);
a = 0
a = -2
a = -4
(%o9)  done
```

También se puede introducir en un bucle una condición de parada utilizando las palabras claves **while** (mientras que) y **unless** (salvo que).

Ejemplos:

```
(%i16)  for i:5 while i>0 step -1 do print("i =", i);
i = 5
i = 4
i = 3
i = 2
i = 1
(%o16)  done
```

```
(%i17)  for i:1 unless i^3>100 do display(i^3);
1^3 = 1
2^3 = 8
3^3 = 27
4^3 = 64
(%o17)  done
```

- ***CONDICIONAL***

La sintaxis del condicional: ***if*** *condición* ***then*** *sentencia1* ***else*** *sentencia2*

Ejemplo:

```
(%i2) for b:0 thru 5 do if b^2>5 then print([b, "valido"]) else print([b, "no es válido"]);
[0,no es válido]
[1,no es válido]
[2,no es válido]
[3,valido]
[4,valido]
[5,valido]
(%o2) done
```

❖ *PROGRAMAR UNA SECUENCIA DE INSTRUCCIONES*

La instrucción ***block (expr_1, ..., expr_n)*** evalúa *expr_1*, ..., *expr_n* secuencialmente y devuelve el valor de la última expresión evaluada.

Esta instrucción nos permite programar funciones, que lleven a cabo unas cuantas instrucciones consecutivas. Además pueden llamar a funciones previamente definidas.

Ejemplo:

A continuación se define una función f(x) y después otra función, denominada mifunc, que recibe como parámetros de entrada los entremos de un intervalo [a,b] y un entero n.

Se define una variable local h=(b-a)/n y se evalua la función f en los puntos de la forma a+i*h, cuando el valor obtenido es positivo se muestra por pantalla y en caso contrario se imprime un mensaje diciendo que no lo es.

```
(%i4) f(x):=x^2-5·x+6;
(%o4) f(x):=x^2-5 x+6

(%i10) mifunc(a,b,n):=block(h:(b-a)/n, for i:0 thru n do
        if f(a+i·h)>0 then display(f(a+i·h))  else print( "no es positivo"));
(%o10) mifunc(a,b,n):=block(h:(b-a)/n,for i from 0 thru n do if f(a+i h)>0 then display(f(a+i h)) else print(no es positivo))

(%i13) mifunc(1,3,8);
f(1)=2
f(5/4)=21/16
f(3/2)=3/4
f(7/4)=5/16
no es positivo
no es positivo
no es positivo
no es positivo
no es positivo
(%o13) done
```