PRÁCTICA 1:
Introducción general a Maxima

1.1 ¿Qué es Maxima?

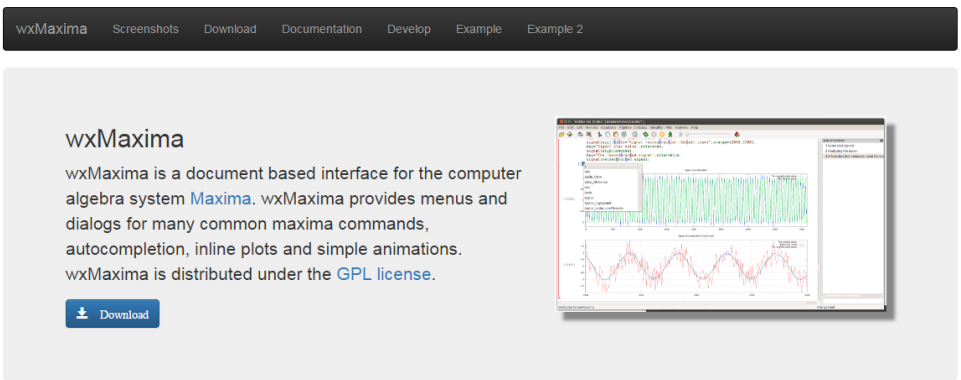
Maxima es un sistema para la manipulación de expresiones simbólicas y numéricas, incluyendo diferenciación, integración, desarrollos en series de Taylor, transformadas de Laplace, ecuaciones diferenciales ordinarias, sistemas de ecuaciones lineales, y vectores, matrices y tensores. Maxima produce resultados con alta precisión usando fracciones exactas y representaciones con aritmética de coma flotante arbitraria. También realiza representaciones gráficas de funciones y datos en dos y tres dimensiones.

Es un sistema multiplataforma y su código fuente puede ser compilado sobre varios sistemas incluyendo Windows, Linux y MacOS X. El código fuente para todos los sistemas y los binarios precompilados para Windows y Linux están disponibles en el Administrador de archivos de SourceForge.

Maxima es un descendiente de Macsyma, el legendario sistema de álgebra computacional desarrollado a finales de 1960 en el instituto tecnológico de Massachusetts (MIT). Este es el único sistema basado en el esfuerzo voluntario y con una comunidad de usuarios activa, gracias a la naturaleza del open source. Macsyma fue revolucionario es sus días y muchos sistemas posteriores, tales como Maple y Mathematica, estuvieron inspirados en él.

La rama Maxima de Macsyma fue mantenida por William Schelter desde 1982 hasta su muerte en 2001. En 1998 él obtuvo permiso para liberar el código fuente bajo la licencia pública general (GPL) de GNU. Desde su paso a un grupo de usuarios y desarrolladores, Maxima ha adquirido una gran audiencia. Maxima está en constante actualización, corrigiendo bugs y mejorando el código y la documentación. Debido, en buena parte, a las sugerencias y contribuciones de su comunidad de usuarios. Información acerca de cómo conseguirlo, instalarlo y demás puede encontrarse en **http://maxima.sourceforge.net/es/**

Se puede trabajar con Maxima en línea de comandos desde la consola, pero hay dos opciones gráficas, una de ellas llamada Maxima Algebra System (xmaxima) es bastante antiestética, mientras que wxMaxima (wxmaxima) es bastante más agradable a la vista y será la opción que utilizaremos en este curso. Centrémonos pues en esta última opción gráfica wxMaxima, que se puede descargar de su página web, **http://wxmaxima.sourceforge.net/**



1.2 Primeros pasos y ayuda

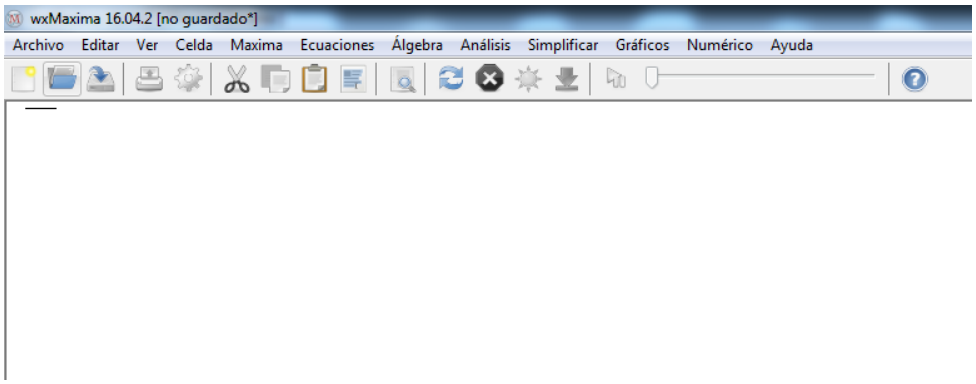
En ella, al igual que en otros CAS como Mathematica, se trabaja a partir de celdas, que pueden contener más de una línea de ejecución. La entrada (en inglés, input) queda asignada a una expresión (%i) junto con un número, mientras que la salida (output en inglés) aparece como (%o) y el mismo número.

Nótese que en wxMaxima la tecla Intro por sí sola cambia de línea, para ejecutar una celda es necesario utilizar la combinación Mayúsculas+Intro (shift+enter) o el Intro del teclado numérico. Este comportamiento se puede intercambiar editando las preferencias de wxMaxima.

Cada sentencia o comando de Maxima puede finalizar con punto y coma (**;**) (wxMaxima escribe automáticamente el último ;) o bien con dólar (**$**). Se pueden escribir varias sentencias una a continuación de otra, cada una de las cuales finaliza con su correspondiente punto y coma (;) o dólar ($), y pulsar shift+enter al final de las mismas para ejecutarlas.

En el caso del punto y coma, aunque aparezcan escritas en la misma línea cada una de tales sentencias aparecerá con una etiqueta diferente. En cambio si se usa el dólar como finalización de sentencia, las operaciones que correspondan serán realizadas por Maxima, pero el resultado de las mismas no será mostrado, salvo la última.

Cuando arrancamos Maxima aparece una página en blanco en la que podemos escribir las operaciones que queremos que realice.



Una vez escritas hemos de pulsar shift+enter o bien el Intro del teclado numérico para ejecutar, entonces aparecerá lo que sigue:

```
(%i1)   2+4;
(%o1)   6
(%i2)   3*5;
(%o2)   15
(%i3)   7/2;
(%o3)   7/2
(%i4)   7.0/2;
(%o4)   3.5
```

```
(%i5)    7/3;
(%o5)    7/3
(%i6)    7/3.0;
(%o6)    2.333333333333333
(%i7)    5!;
(%o7)    120
(%i8)    3^4;
(%o8)    81
(%i9)    3**4;
(%o9)    81
```

Los **(%i1)**, **(%o1)**, etc., los escribe el programa para indicar el número de entrada o salida correspondiente. Con el símbolo de **%** indicaremos la salida anterior
Las operaciones aritméticas que realiza son las siguientes:

- suma: x + y
- resta: x - y
- producto: x*y
- división: x/y
- factorial: x!
- potencia: x^y o bien x**y

Desde la línea de comandos de Maxima podemos obtener ayuda sobre un comando concreto conociendo su nombre. Esto es cierto para la mayor parte de los comandos, y la sintaxis es:

? comando, cuando se conoce el nombre exacto.
?? comando, cuando no se conoce el nombre exacto.

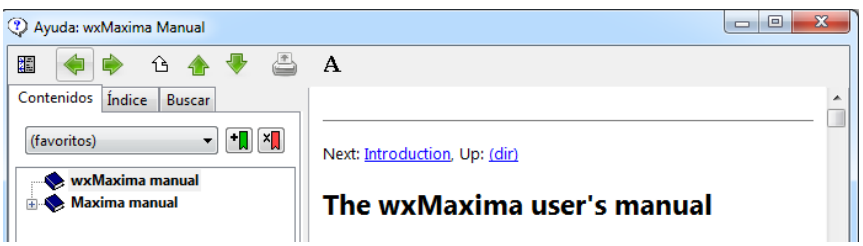
El espacio de separación después de **?** es muy importante. Puede el lector comprobar, por ejemplo el resultado de la entrada **? integrate** y de **?? integrat**.

Para algunos comandos puede además obtenerse ejemplos de uso mediante:

example(comando)

Compruebe el lector el resultado obtenido al pedir: **example(sum)**.

Además de manuales y todo tipo de información accesible a través de la red, Maxima incorpora también una ayuda en el propio programa:

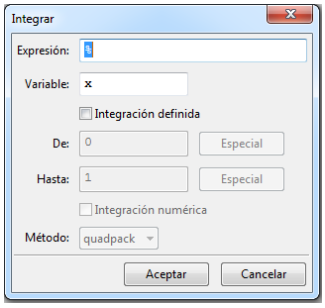


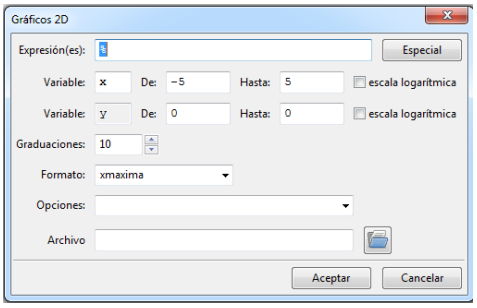
Una de las ventajas de wxMaxima es que, como veremos, permite trabajar de dos formas:

- Mediante la introducción de los comandos directamente sobre las celdas, en la ventana de trabajo.
- Mediante el uso de paletas. Estas paletas, simplemente rellenando unos sencillos cuadros de diálogo que se abren en ventanas emergentes, vuelcan a una nueva celda en la ventana de trabajo la sintaxis correspondiente al comando deseado, sin necesidad de que el usuario lo memorice.

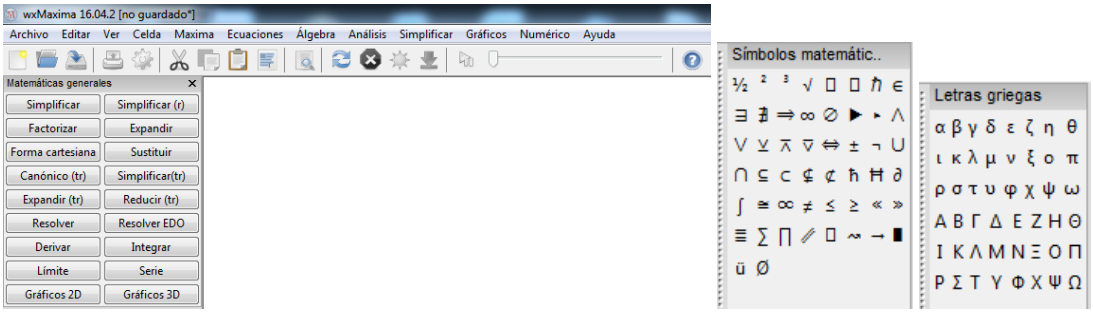
A medida que veamos los comandos en estos guiones, comentaremos también cómo localizarlos en el menú de wxMaxima. El alumno puede trabajar con cualquiera de los dos métodos, a su elección. De todas formas debemos aclarar que no todos los comandos tienen su correspondiente paleta.

A través de la barra superior, y organizados por temas, están las paletas correspondientes a Álgebra, Cálculo (Análisis), Ecuaciones, etc. Por ejemplo, estas son las paletas para **Integrar**, y realizar **Gráficos 2D**:





Además podemos, desde el menú **Ver**, dejar fijas en la pantalla, las paletas más usadas, mediante la opción: **Matemáticas Generales**, o también las paletas que dan acceso a los símbolos matemáticos, las letras griegas, etc.



En Maxima los comentarios se escriben entre /* y */. Todo lo que esté entre ellos es ignorado por Maxima. Es muy importante que los programas que se realicen estén debidamente documentados para que sean fáciles de entender y, en su caso, modificar. Por ejemplo, si haces esto te devuelve tan sólo 3:

```
(%i10)   /* esto es ignorado por Maxima */1+2;
(%o10)   3
```

Además en wxMaxima puedes usar ↑↓ del teclado para recuperar instrucciones anteriores. De esta forma puedes cortar y pegar entradas anteriores y editarlas.

1.3 Constantes, funciones, asignaciones

1.3.1 Constantes matemáticas

Las constantes matemáticas más utilizadas en Maxima se escriben anteponiendo el símbolo de % en la forma:

- **%pi** = número pi = 3.14159...
- **%e** = es el número e = 2.71828...
- **%inf** = es el infinito
- **%i** = es la unidad imaginaria = sqrt(-1)
- **%phi** = es el número áureo = (sqrt(5)+1)/2

1.3.2 Funciones matemáticas elementales

Las funciones matemáticas elementales se introducen como sigue:

- **sqrt(x)** = raíz cuadrada de x
- **exp(x)** = exponencial de x
- **log(x)** = logaritmo neperiano de x
- **log(x)/log(b)** = logaritmo en base b de x
- **sin(x)** = seno de x
- **cos(x)** = coseno de x
- **tan(x)** = tangente de x
- **sec(x)** = secante de x
- **csc(x)** = cosecante de x
- **asin(x)** = arco seno de x
- **acos(x)** = arco coseno de x
- **atan(x)** = arco tangente de x
- **sinh(x)** = seno hiperbólico de x
- **cosh(x)** = coseno hiperbólico de x
- **tanh(x)** = tangente hiperbólica de x
- **asinh(x)** = arco seno hiperbólico de x
- **acosh(x)** = arco coseno hiperbólico de x
- **atanh(x)** = arco tangente hiperbólica de x
- **abs(x)** = valor absoluto de x
- **entier(x)** = parte entera de x
- **round(x)** = redondeo de x
- **max(x1, x2, ...), min(x1, x2, ...)** = máximo, mínimo (x1, x2, ...)

Veamos algunos ejemplos.

```
(%i11)   sin(%pi/2);
(%o11)   1
(%i12)   exp(log(1));
(%o12)   1
(%i13)   abs(cos(%pi));
(%o13)   1
```

1.3.3 Funciones relativas al cálculo con números naturales

Para un número natural n, podemos señalar las siguientes:

- **n!** = factorial de n
- **primep(n)** = nos dice si n es o no primo
- **oddp(n)** = nos dice si n es o no impar
- **evenp(n)** = nos dice si n es o no par
- **factor(n)** = da la descomposición en factores primos de n
- **divisors(n)** = da los divisores de n
- **binomial(m,n)** = es el número combinatorio m sobre n

1.3.4 Funciones relativas al cálculo con números complejos

Las operaciones aritméticas entre números complejos se realizan como antes, pero para un número complejo z, podemos utilizar también las siguientes funciones:

- **rectform(z)** = que nos da la forma binómica de z
- **realpart(z)** = da la parte real de z
- **imagpart(z)** = da la parte imaginaria de z
- **polarform(z)** = da la forma polar de z
- **conjugate(z)** = da el conjugado de z
- **abs(z)** = da el módulo de z

1.3.5 Definición de nuevas funciones

Utilizando las funciones y operaciones de Maxima es posible definir nuevas funciones de una o más variables mediante el uso del comando **:=**, en la forma:

NombreFunción(x1,x2,...,xn):= expresión

Veamos algunos ejemplos de funciones escalares y vectoriales:

```
(%i14)   F1(x):=x^3+x^2+x+1;
(%o14)   F1(x):=x^3+x^2+x+1
```

Si ahora queremos evaluarla en un punto basta con escribir:

```
(%i15)   F1(3);
(%o15)   40
```

Para una función escalar de dos variables escribimos:

```
(%i17)   F2(x,y):=3*x^2-y^5;
         F2(2,1);
(%o16)   F2(x,y):=3 x^2-y^5
 (%o17)   11
```

En tanto que para una función vectorial de tres variables se pone:

```
(%i19)   F3(x,y,z):=[x-y,x+2*y^2+z,3*y-z^3];
         F3(1,2,-1);
(%o18)   F3(x,y,z):=[x-y,x+2 y^2+z,3 y-z^3]
 (%o19)   [-1,8,7]
```

Antes de definir una nueva función conviene comenzar limpiando la función, por si hubiese sido definida otra con el mismo nombre con anterioridad, lo que se hace con el comando **kill( )**.

El comando **kill(all)** borra todas las asignaciones y es equivalente a elegir en el menú: **Maxima-Limpiar memoria**. Las entradas y salidas vuelven a contar desde 1. Por ejemplo:

```
(%i20)   kill(all)$
(%i1)    f(x,y):=x^2-y^2;
(%o1)    f(x,y):=x^2-y^2
(%i2)    f(5,4);
(%o2)    9
```

Cuando se define una función mediante el comando **:=** no se desarrolla la expresión que sirve para definirla. Si se desea que la expresión se evalúe hay que emplear el comando (que utilizamos normalmente en la programación):

define(NombreFun(x1,x2,...xn), expresión)

La diferencia entre ambas formas de definir una función se ilustra en el ejemplo siguiente:

```
(%i5)    expr : cos(y) - sin(x)$
         f(x,y):=expr;
         define(g(x,y),expr);
(%o4)    f(x,y):=expr
 (%o5)    g(x,y):=cos(y)-sin(x)
(%i7)    f(0,0);
         g(0,0);
(%o6)    cos(y)-sin(x)
 (%o7)    1
```

1.3.6 Asignaciones y Evaluaciones

Para asignar valores a una constante o variable se utiliza el comando **:**

Supongamos que queremos calcular a^2+b^2 para $a = 2$ y $b = 3$. Le asignamos valores como sigue:

```
(%i8)    kill(all)$
(%i3)    a:2$ b:3$
         a^2+b^2;
(%o3)    13
```

La asignación de valores, como hemos señalado, utiliza **:** y no utiliza **=**. El símbolo de igualdad se emplea fundamentalmente con las ecuaciones, si bien también aparece en las operaciones de sustitución y evaluación de expresiones.

Debemos recalcar que la asignación de valores tiene un ámbito de aplicación más amplio que establecer el valor de una constante, tal y como ha sido utilizado antes. Mediante : se puede definir una función sin declarar la variable. Por ejemplo con la instrucción:

```
(%i4)    F1:x^2+1;
(F1)     x^2+1
```

En Maxima se puede sustituir cualquier expresión, **e1**, por otra, **e2**, añadiendo una coma y a continuación una indicación del tipo **e1=e2**. También se puede utilizar el comando **subst**. Esta función está disponible en wxMaxima, a través del menú **Simplificar-Sustituir**.

```
(%i5)    F1,x=3;
(%o5)    10
(%i6)    subst(x=3,F1);
(%o6)    10
```

Otro comando diferente para realizar la evaluación puede ser **ev( )** :

```
(%i7)    ev(F1, x=3);
(%o7)    10
```

Obsérvese la distinta sintaxis de los comandos, en cuanto al orden de escritura.

1.4 Álgebra

1.4.1 Listas

Las listas son objetos básicos a la hora de representar estructuras de datos; de hecho, toda expresión de Maxima se representa internamente como una lista, lo que no es de extrañar habida cuenta de que Maxima está programado en Lisp (List Processing). De ahí que Maxima herede la potencia y versatilidad en el manejo de listas.

En Maxima las listas se pueden definir como series de expresiones separadas por comas y encerradas entre corchetes. Los elementos de una lista pueden ser también listas, expresiones matemáticas o cadenas de caracteres entre comillas.

En esta sección nos ocuparemos brevemente de la generación y manipulación de listas. Veamos un ejemplo:

```
(%i8)    kill(all)$
(%i1)    lista1:[1, 2+1/2,[a,2,c],1+3*x+y^2];
(lista1) [1, 5/2, [a, 2, c], y^2+3 x+1]
```

Se puede acceder a los 10 primeros elementos de una lista mediante las funciones **first**, **second**,..., **tenth**, y para el último se accede con **last**. Por ejemplo:

```
(%i3)    second(lista1);
         last(lista1);
(%o2)    5/2
  (%o3)    y^2+3 x+1
```

También se puede indexar mediante corchetes el elemento enésimo, por ejemplo el tercer elemento de lista se puede pedir en la forma:

```
(%i4)    lista1[3];
(%o4)    [a,2,c]
```

Con las listas pueden hacerse las operaciones aritméticas básicas de sumar, restar, multiplicar y dividir, término a término, cuando tienen igual longitud. Se puede hacer el producto escalar de vectores (listas), así como potencias y radicales.

```
(%i6)    a:[1,2,3,4,5];
         b:[6,7,3,9,0];
(a)      [1,2,3,4,5]
  (b)      [6,7,3,9,0]
(%i9)    a^2;
         b/a;
         7*a;
(%o7)    [1,4,9,16,25]
  (%o8)    [6,7/2,1,9/4,0]
  (%o9)    [7,14,21,28,35
```

El producto escalar de los vectores *a* y *b* se muestra a continuación:

```
(%i10)   a.b;
(%o10)   65
```

Diferente de este otro:

```
(%i11)   a*b;
(%o11)   [6,14,9,36,0]
```

Las listas se pueden manipular de muchas maneras, por ejemplo con la función **delete** se puede suprimir un elemento de una lista, en tanto que **cons** devuelve la lista que resulta de añadir un elemento al principio de una lista, y se pueden unir listas mediante el comando **append**.

```
(%i12)   ar:delete(3,a);
(ar)     [1,2,4,5]
```

```
(%i13)   aa:cons(%e,a);
(aa)     [%e,1,2,3,4,5]
(%i14)   append(ar,aa);
(%o14)   [1,2,4,5,%e,1,2,3,4,5]
```

Podemos construir listas mediante el uso del comando **makelist**, formadas por un número finito de expresiones que se ajusten a un término general. Esta función está disponible en wxMaxima a través del menú **Algebra-Construir lista**. Como se aprecia, sus parámetros son, respectivamente, el término general, la variable utilizada como índice y dos números enteros que expresan el rango en que varía este índice de forma consecutiva.

```
(%i15)   lista:makelist(3*i+1,i,0,10);
(lista)  [1,4,7,10,13,16,19,22,25,28,31]
```

Para aplicar ciertas instrucciones a las listas podemos utilizar el comando **apply**.

```
(%i16)   apply(max,lista);
(%o16)   31
(%i17)   apply("+",lista);
(%o17)   176
```

Pero para construir una nueva lista formada por el valor de una función sobre cada uno de los elementos de la lista original, no sirve **apply**, que da error. Debemos usar el comando **map** (en wxMaxima a través del menú **Algebra-Distribuir sobre lista**.

```
(%i18)   f1(x):=x^2;
(%o18)   f1(x):=x^2
(%i19)   map(f1,lista);
(%o19)   [1,16,49,100,169,256,361,484,625,784,961]
```

Otras instrucciones relativas a listas son las siguientes:

- **part** = busca un elemento dando su posición en la lista
- **reverse** = invertir lista
- **sort** = ordenar lista
- **flatten** = unifica las sublistas en una lista
- **length** = longitud de la lista
- **unique** = devuelve lista sin elementos repetidos
- **abs** = devuelve la lista de los valores absolutos de los elementos de la lista dada
- **lmax y lmin** = devuelven el máximo y el mínimo de una lista

```
(%i20)   part(lista,5);
(%o20)   13
(%i21)   reverse(lista);
(%o21)   [31,28,25,22,19,16,13,10,7,4,1]
(%i22)   length(lista);
(%o22)   11
```

```
(%i23)   flatten([[4,5],4,[1,2,3,z]]);
(%o23)   [4,5,4,1,2,3,z]
(%i24)   sort(%);
(%o24)   [1,2,3,4,4,5,z]
(%i25)   unique([1,1,2,3,3,4]);
(%o25)   [1,2,3,4]
(%i26)   lmax(lista);
(%o26)   31
```

1.4.2 Matrices

Las matrices se introducen en wxMaxima mediante la función **matrix**, a la que pasamos cada una de sus filas, constituidas por el mismo número de elementos numéricos o simbólicos, escribiendo estos entre corchetes. También cabe la posibilidad de hacerlo a través del menú **Álgebra-Introducir matriz**, ofreciéndonos distintas posibilidades.

```
(%i1)    kill(all)$
(%i1)    A: matrix([1,2,3,4],[4,5,6,7],[7,8,9,0]);
```

$$(A)\quad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 7 & 8 & 9 & 0 \end{bmatrix}$$

Se puede acceder al elemento de la fila 3, columna 1, sin más que poner:

```
(%i2)    A[3,1];
(%o2)    7
```

Se pueden obtener submatrices de una dada mediante el comando: **submatrix(l1,l2,...,B,c1,c2,..)**, en el se indica primero las filas a eliminar, segundo el nombre de la matriz y finalmente las columnas a eliminar.

```
(%i3)    submatrix(1,A,1,2);
```

$$(\%o3)\quad \begin{bmatrix} 6 & 7 \\ 9 & 0 \end{bmatrix}$$

Se puede pedir una fila o columna de una matriz, así como añadir filas o columnas mediante los comandos:

- **row(A,i)**
- **col(A,j)**
- **addrow(A,[a,...,z])**
- **addcol(A,[x,...,z])**

```
(%i4)    row(A,2);
(%o4)    [4  5  6  7]
```

```
(%i5)    col(A,4);
         [ 4 ]
         [   ]
(%o5)    [ 7 ]
         [   ]
         [ 0 ]
```

Se pueden realizar operaciones algebraicas sobre matrices, así por ejemplo:

- El operador (*****) se interpreta como producto elemento a elemento
- El operador (**.**) representa el producto matricial usual
- El operador (^) calcula las potencias de los elementos de la matriz dada
- El operador (^^) calcula las potencias de la matriz dada

Veamos algunos ejemplos.

```
(%i7)    m1:matrix([1,2,3],[4,5,6],[7,8,9])$
         m2:matrix([3,2,1],[6,5,4],[9,8,7])$

(%i8)    m1*m2;
         [ 3   4   3 ]
         [           ]
(%o8)    [ 24  25  24 ]
         [           ]
         [ 63  64  63 ]
(%i9)    m1.m2;
         [ 42   36   30  ]
         [               ]
(%o9)    [ 96   81   66  ]
         [               ]
         [ 150  126  102 ]
(%i10)   m1^2;
         [ 1   4   9  ]
         [            ]
(%o10)   [ 16  25  36 ]
         [            ]
         [ 49  64  81 ]
(%i11)   m1^^2;
         [ 30   36   42  ]
         [               ]
(%o11)   [ 66   81   96  ]
         [               ]
         [ 102  126  150 ]
```

Que coincide con la matriz producto *m*1.*m*1

```
(%i12)   m1.m1;
         [ 30   36   42  ]
         [               ]
(%o12)   [ 66   81   96  ]
         [               ]
         [ 102  126  150 ]
```

Otras funciones aplicables sobre matrices:

- **transpose** = da la matriz traspuesta
- **invert** = da la matriz inversa
- **determinant** = calcula el determinante de una matriz cuadrada
- **rank** = calcula el rango
- **mat_trace** = devuelve la traza
- **triangularize** = devuelve una matriz triangular superior resultante de aplicar el método de Gauss a una matriz dada
- **eigenvalues** = devuelve dos listas, la primera formada por los autovalores de una matriz y la segunda por sus multiplicidades
- **matrix_size** = devuelve una lista con el número de filas y columnas, respectivamente
- **rowswap(M, i, j)** = si M es una matriz, intercambia las filas *i* y *j*
- **columnswap(M, i, j)** = si M es una matriz, intercambia las columnas *i* y *j*

Algunos ejemplos:

```
(%i1)   kill(all)$
(%i1)   A:matrix([0,-1,2],[2,-3,3],[1,1,1]);
(A)     [ 0  -1  2 ]
        [ 2  -3  3 ]
        [ 1   1  1 ]
(%i2)   transpose(A);
(%o2)   [  0   2  1 ]
        [ -1  -3  1 ]
        [  2   3  1 ]
(%i3)   invert(A);
(%o3)   [ -2/3   1/3   1/3 ]
        [  1/9  -2/9   4/9 ]
        [  5/9  -1/9   2/9 ]
(%i4)   determinant(A);
(%o4)   9
```

Recordemos que el determinante de una matriz coincide con el producto de los autovalores.

```
(%i5)   eigenvalues(A);
(%o5)   [[-(sqrt(13)-1)/2, (sqrt(13)+1)/2, -3], [1, 1, 1]]
(%i6)   -(sqrt(13)-1)/2*(sqrt(13)+1)/2*(-3),numer;
(%o6)   9.0
```

Recordemos que la traza es la suma todos los elementos de la diagonal principal y que la traza de una matriz coincide con la suma de todos los autovalores de la matriz (considerando los autovalores repetidos).

```
(%i7)    mat_trace(A);
0 errors, 0 warnings
  (%o7)    -2
(%i8)    -(sqrt(13)-1)/2+(sqrt(13)+1)/2+(-3),numer;
(%o8)    -2.0
```

El número de autovalores nulos de una matriz cuadrada *A* de rango *r* es mayor o igual que *n-r*.

```
(%i9)    rank(A);
(%o9)    3
```

La orden **triangularize** la usaremos después para comprobar la eficacia del método de Gauss.

```
(%i10)   B:triangularize(A);
```

$$(B)\quad \begin{bmatrix} 2 & -3 & 3 \\ 0 & -2 & 4 \\ 0 & 0 & -9 \end{bmatrix}$$

Y también sabemos que los autovalores de una matriz triangular son los elementos de la diagonal principal.

```
(%i11)   determinant(B);
(%o11)   36
```

Y finalmente también nos serán de utilidad en la programación los siguientes comandos:

```
(%i12)   matrix_size(B);
(%o12)   [3,3]
```

```
(%i13)   columnswap(B,1,2);
```

$$(\%o13)\quad \begin{bmatrix} -3 & 2 & 3 \\ -2 & 0 & 4 \\ 0 & 0 & -9 \end{bmatrix}$$

```
(%i14)   rowswap(B,1,2);
```

$$(\%o14)\quad \begin{bmatrix} 0 & -2 & 4 \\ 2 & -3 & 3 \\ 0 & 0 & -9 \end{bmatrix}$$

1.5 Cálculo

1.5.1 Cálculo Diferencial

El cálculo de derivadas se realiza con los comandos:

> **diff(expr,variable)** derivada de expr respecto de variable
> **diff(expr,variable,n)** derivada n-ésima de expr respecto de variable
> **diff(expr,variable1,n1,variable2,n2)** derivada de expr respecto de la variable1, n1 veces, y respecto de la variable2, n2 veces.

O también a través del menú **Análisis-Derivar.**

```
(%i19)    kill(all)$

(%i2)     f(x):=x^3$
          diff(f(x),x);
(%o2)     3 x^2
(%i4)     f2(x,y):=x^y+y*sin(1/x)$
          diff(f2(x,y),y,2);
(%o4)     x^y log(x)^2
```

Si se quiere reutilizar la derivada de una función como una nueva función debe hacerse con el comando **define (g(x), diff(f(x),x))**, que fuerza a evaluar la derivada.

Obsérvese que el comando **:=** no evalúa la derivada y por tanto no es capaz de definir de forma correcta la función, como comprobamos al pedir su valor en 1. Este hecho ya lo comentamos con anterioridad y es muy importante. Le sucede al operador **:=** cada vez que la expresión de la derecha no es directamente el valor de la función a definir.

```
(%i5)     define(g(x), diff(f(x),x));
(%o5)     g(x):=3 x^2
(%i7)     der(x):=diff(f(x),x);
          der(1);
(%o6)     der(x):=d/dx f(x)
diff: second argument must be a variable; found
#0: der(x=1)
 -- an error. To debug this try: debugmode(true)
```

El operador comilla hace que no se evalúe la expresión que le sigue. Veamos un ejemplo.

```
(%i8)     'diff(f(x),x);
(%o8)     d/dx x^3
```

1.5.2 Cálculo integral

En wxMaxima es posible calcular integrales de funciones, tanto definidas como indefinidas, mediante el comando **integrate** o bien en el menú **Análisis-Integrar** e introduciendo en la ventana de diálogo la correspondiente función, en la forma:

integrate(f(x),x) que da la primitiva de la función f(x)
integrate(f(x),x,a,b) que da la integral definida de la función f(x) en el intervalo [a, b]

```
(%i1)  kill(all)$
(%i1)  integrate(atan(x),x);
(%o1)  x atan(x) - log(x^2+1)/2
(%i2)  integrate(atan(x),x,0,1);
(%o2)  - (2 log(2) - %pi)/4
```

La integración numérica de una función *f*(*x*) en un intervalo [*a*, *b*] se realiza con alguno de los siguientes comandos:

quad_qags(f(x),x,a,b) para a y b finitos
quad_qagi(f(x),x,a,b) para intervalos ilimitados
romberg(f(x),x,a,b) para a y b finitos

Si se introduce en el menú **Análisis-Integrar** (y marcamos integración definida e integración numérica) podemos elegir entre el método **quadpack** o **romberg**. **quadpack** ofrece como salida 4 valores: el primero es el resultado aproximado, el segundo la cota del error absoluto del método numérico, el tercero el número de iteraciones precisadas y el cuarto un código que alerta sobre posibles errores. Por el contrario **romberg** sólo proporciona el valor aproximado.

```
(%i3)  quad_qags(1/(x^2+1), x, 0, 1);
(%o3)  [0.7853981633974484,8.719671245021583 10^-15,21,0]
(%i4)  romberg(1/(x^2+1),x,0,1);
(%o4)  0.785398159599199
```

Si comparamos con el comando **integrate**, y la aproximación a aritmética flotante:

```
(%i5)  integrate(1/(x^2+1), x, 0, 1);
(%o5)  %pi/4
(%i6)  %,numer;
(%o6)  0.7853981633974483
```

Estos comandos los veremos con más detalle en la Práctica 9, al estudiar la integración numérica de funciones y los utilizaremos para comparar los resultados obtenidos con nuestros programas

1.6 Gráficas de funciones

1.6.1 Funciones en 2-D

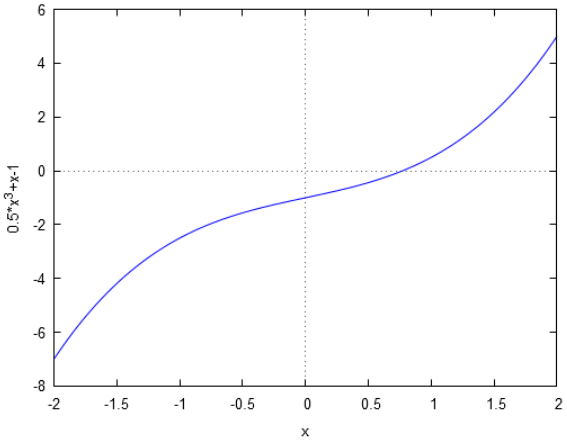
La gráfica de una función de una variable real se hace con el comando **plot2d** que actúa, como mínimo, con dos parámetros: la función (o lista de funciones a representar), y el intervalo de valores para la variable x.

plot2d(f(x),[x,a,b]) da la gráfica de f(x) en [a, b]
plot2d([f1(x),f2(x),...],[x,a,b]) da las gráficas de las funciones f1(x),f2(x),... en [a, b]

Al comando **plot2d** se puede acceder también a través del menú **Gráficos - Gráficos 2D**.

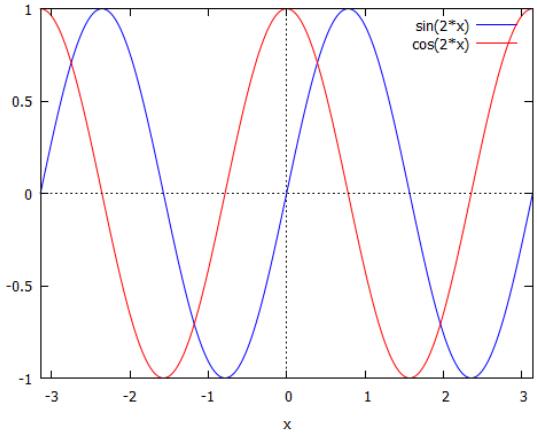
Si se le añade el prefijo **wx** delante, presenta los gráficos en la misma pantalla del wxMaxima y no en una ventana emergente, como hace **plot2d**. Por cierto, debemos ser cuidadosos ya que si no cerramos las ventanas emergentes, el programa no responderá a nuestras instrucciones.

```
(%i1)    kill(all)$
(%i1)    plot2d(0.5*x^3+x-1,[x,-2,2])$
```



Cuando pulsamos el botón **Gráficos 2D** en el menú de Gráficos de wxMaxima, aparece una ventana de diálogo con varios campos que podemos completar o modificar. Veamos unos ejemplos. Comenzamos dibujando varias gráficas simultáneamente.

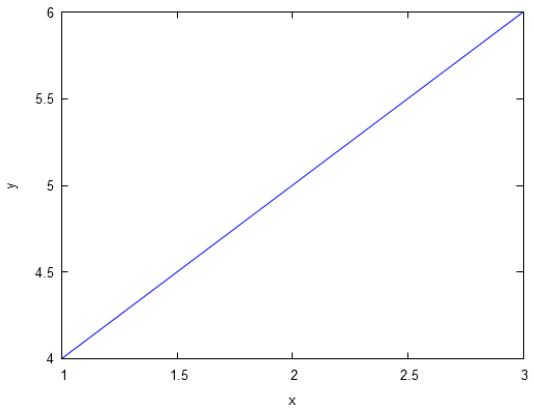
```
(%i2)    plot2d([sin(2*x),cos(2*x)],[x,-%pi,%pi])$
```



En numerosas ocasiones necesitaremos representar gráficos de puntos, Para ello usaremos la opción **discrete**, que permite trabajar de dos formas para introducir los puntos (los cuales por defecto los une con una línea):

Por listas: **[x0,x1,…], [y0,y1,…]**
Por pares: **[[x0,y0], [x1,y1], …]**

```
(%i3)    plot2d([discrete, [1,2,3], [4,5,6]])$
(%i4)    plot2d([discrete, [[1,4], [2,5],[3,6]]])$
```

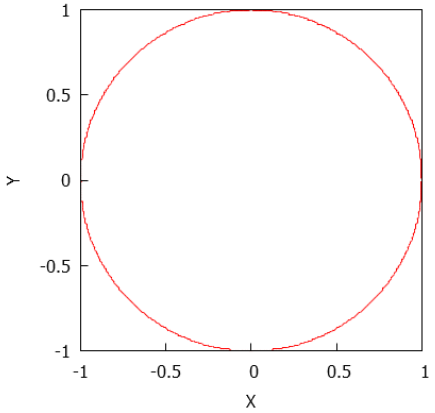


Por otro lado, el módulo **draw** también permite dibujar gráficos con relativa comodidad, En versiones antiguas de Maxima era necesario cargarlo previamente mediante la orden **load(draw)**, pero ahora ya no es necesario**.** Una vez cargado el módulo **draw**, podemos utilizar la orden **draw2d** para dibujar gráficos en 2 dimensiones:

draw2d(opciones, objeto gráfico,...)

Un gráfico está compuesto por varias opciones y el objeto gráfico que queremos dibujar. Las opciones son numerosas y permiten controlar prácticamente cualquier aspecto imaginable. Aquí comentaremos algunas de ellas pero conviene recurrir a la ayuda del programa para ampliar conocimientos. En segundo lugar aparece el objeto gráfico que puede ser de varios tipos aunque el que más usaremos es **implicit**, pues este caso no está contemplado en el comando **plot2d.**

```
(%i5)    draw2d(proportional_axes=xy,color=red,
             implicit(x^2+y^2=1,x,-1,1,y,-1,1))$
```



Para finalizar mencionar que son innumerables las opciones que presenta Maxima para realizar gráficos, incluyendo gráficos en 3D, pero hemos preferido limitar esta presentación a los comandos que luego vamos a usar en esta asignatura.

1.7 Ejercicios propuestos

Completar con Maxima los ejercicios siguientes:

Ejercicio 1. Calcula en la misma celda y en distintas celdas: ln(π/4), ln(e^2), sen(π/2), arctan(1), logaritmo en base 2 de 10. Cuando estén en la misma celda, evita que salga el resultado en pantalla del último y del primer valor.

Ejercicio 2. Sean los vectores: *a* = (2,-1, 4, 0, 3); *b* = (-1, 0, 2, 3, 1); *c* = (0, 1,-3, 4,-7). Se pide calcular: 2*a* - 3*b* + c, *a.b*, *a*b*.

Ejercicio 3. Define *lista1* mediante la orden **makelist**(i,i,2,21), y *lista2* mediante la orden **makelist**(i,i,22,31). Multiplica el segundo elemento de *lista1* por todos los elementos de *lista2*. El resultado será una lista a la que denominaréis *productos*. Divide *productos* entre *lista2.*

Ejercicio 4. Escribe una matriz cuadrada, calcula su determinante y, si es posible, su inversa. Comprueba multiplicando que la matriz por su inversa es la matriz unidad. Calcula el polinomio característico a partir del determinante de $A-\lambda I$.

Ejercicio 5. Haz que la variable *z* tome el valor (*x*+1)*(*x*-1), elévala al cuadrado, súmale 2*x*+2 y simplifica el resultado. Desasigna *z*.

Ejercicio 6. Define la función $f(x)=\text{sen}(x^2)$ y la función $g(x)=\text{sen}^2(x)$. Hazlo con el comando **define** y llámalas *f*(*x*) y *g*(*x*), con el comando **:** y llámalas *f*1y *g*1 y con el comando **:=** y llámalas *f2*(*x*) y *g2*(*x*). Después calcula su valor para *x*=1 mediante distintos métodos (**subst**, **ev**,…).

Ejercicio 7. Calcula la primera y la segunda derivada de *f*(*x*). Define una función, que llamarás *der*(*x*), que sea la primera derivada de *f*(*x*). Calcula la primitiva de *der*(*x*). Calcula el valor del área limitada por *f*(*x*) entre *x*=0 y *x*= π. Hazlo mediante integración exacta y numérica.

Ejercicio 8. Dibuja las gráficas de *f*(*x*) y *g*(*x*) mediante el comando plot2d y mediante draw2d, en el intervalo -1<x<1. Cambia dicho intervalo a -2<x<2. Cambia el color en el comando draw2d y utiliza la opción de ejes proporcionales.

Ejercicio 9. Dibuja *cos*(*x*)-*x* y estima donde se hace cero. Dibuja simultáneamente e^x; x^e y estima donde se cortan las curvas.

Ejercicio 10. Dibuja |*x*|/*x* desde -5 a 5; si no lo ves bien repite el dibujo con un rango vertical de [-2,2] y deduce que pasaba antes.