RESUMEN DE WXMAXIMA

Podéis descargaros el programa del siguiente ENLACE http://andrejv.github.i

SEGUID LAS INSTRUCCIONES CUIDADOSAMENTE PARA SU INSTALACIÓN

CUESTIONES GENERALES

Esto es una celda de texto abriendo la pestaña Celda y eligiendo Nueva celd

```
(%i1) /* También podemos escribir un comentario escribiendo entre
      los símbolo barra asterisco poner comentario y cerrar con asterisco b


      Mira los límites (barra y asterisco) */
      /* Limpiemos la memoria */
      kill(all);
(%o0) done
```

```
(%i1) /* Las líneas de comando acaban con ';' y se ejecutan
      con mayúscula-enter */
      /* Asignar variables no se hace con igual sino con ":" */
      a:3;
      b:4;
      c:53;
(%o1) 3
(%o2) 4
(%o3) 53
```

```
(%i4) /*Para borrar una variable o todas las variables */

      remvalue(a);
      remvalue(all);
(%o4) [a]
(%o5) [b,c]
```

```
(%i6) /* El contador de instrucciones sigue corriendo.
      Si has estado trabajando y quieres reiniciar todo,
      variables e instrucciones y empezar en la instrucción 1,
      teclea kill(all) y desaparecen las variables,
      reiniciando el contador de instrucciones que has ejecutado */
      kill(all);
(%o0) done
```

```
(%i1) /* Si no quieres que te aparezca mensaje de salida porque
      la instrucción es sencilla,acaba el comando con $ */
      a:3$
      b:52$
```

```
(%i3) /* Veamos si, aunque no lo veamos, hemos definido a y b */
      a;
      b;
(%o3) 3
(%o4) 52
```

```
(%i5) /* Operaciones básicas */
      a+b;
      a*b;
      a/b;
      a^2;
      sqrt(a);
(%o5) 55
(%o6) 156
(%o7) 3/52
(%o8) 9
(%o9) sqrt(3)
```

```
(%i10) /* Podemos operar con la última instrucción */
       %*3;
(%o10) 3^(3/2)
```

```
(%i11) /* U operar con una salida en concreto */
       %o171*2;
(%o11) 2 %o171
```

```
(%i12) /* Por defecto, wxmaxima usa la mejor opción y no evalúa
       cocientes aproximados*/
       a/b;
(%o12) 3/52
```

```
(%i13) /* Podemos decirle que nos dé el valor numérico
       sólo en esta ocasión */
       a/b,numer;
(%o13) 0.0576923076923077
```

```
(%i14) /* Pero si volvemos a pedir a/b, vuelve al resultado sin aproximar */
       a/b;
(%o14) 3/52
```

```
(%i15) /* Esto sucede porque wxmaxima tiene una variable interna
       llamada numer que está asiganda al valor false */
       numer;
(%o15) false
```

```
(%i16) /* Si queremos que nos dé todos los resultados evaluados,
          es mejor que cambiemos esta variable a true */
       numer:true;
(%o16) true
```

```
(%i17) /* Ahora veamos si evalúa o deja indicado el cociente */
       a/b;
(%o17) 0.0576923076923077
```

```
(%i18) /* Podemos controlar el número de cifras significativas en
       los decimales del resultado con la variable fpprintprec.
       Ojo que ls sicfras significativas no es el número de decimales,
       sino el número de cifras a partir del primer
       cero a la derecha de la coma decimal */
       fpprintprec:5$
       a/b;
       a/b^2;
       fpprintprec:4$
       a/b;
       a/b^2;
       /* Si la cifra no tiene decimales, no
       tiene efecto */
       3034589*a;
       a*b^8;
(%o19) 0.057692
(%o20) 0.0011095
(%o22) 0.05769
(%o23) 0.001109
(%o24) 9103767
(%o25) 160379185594368
```

```
(%i26) /* Volvamos a la expresión más exacta fijando la variable
       numer a false */
       numer:false$
       a/b;
(%o27) 3/52
```

```
(%i28) /* Algunas constantes útiles se asignan con el
       símbolo % delante  (pi,e,sqrt(-1),número de oro */
       %pi;
       %e;
       %i;
       %phi;
(%o28) π
(%o29) %e
(%o30) %i
(%o31) φ
```

```
(%i32) /* Podemos combinar varias sentencias en una línea */
       %pi,numer;%e,numer;%phi,numer;
(%o32) 3.142
(%o33) 2.718
(%o34) 1.618
```

```
(%i35) /* Algunas funciones de uso común */
       sin(%pi/2);
       cos(0.);
       tan(%pi/4);
       asin(1);
       acos(0.);
       atan(inf);
       b:10^4;
       a:exp(2);
       /* Si queremos ver estos resultados */ a,numer;
(%o35) 1
(%o36) 1
(%o37) 1
(%o38) π/2
(%o39) π/2
(%o40) π/2
(%o41) 10000
(%o42) %e^2
(%o43) 7.389
```

```
(%i44) /* Logaritmo neperiano o natural */
       c:log(a);
(%o44) 2
```

```
(%i45) /* Si queremos hacer logaritmo decimal */
       d:log(b)/log(10),numer;
(%o45) 4.0
```

Uso de Vectores (o listas) y Matrices

```
(%i46) /* ********************************************************** */
       /* Declaración de vectores (o listas) y operaciones con ellos */
       /* ********************************************************** */

       kill(all)$

       /* Carguemos un paquete para poder evaluar
       productos vectoriales y fijemos la precisión a 6
       cifras significativas */
       load(vect)$
       fpprintprec:6$
```

```
(%i3) /* definición de vectores. Maxima los llama listas */
      a:[1,0.,3.];
      b:[2.,-1.,4.];
      c:[3,-2,1];
(%o3) [1,0,3]
(%o4) [2,-1,4]
(%o5) [3,-2,1]

(%i6) /* Operaciones básicas */
      /* Componentes del vector a */
      a[1];a[2];a[3];
(%o6) 1
(%o7) 0
(%o8) 3

(%i9) /* Suma */ a+b;
(%o9) [3,-1,7]

(%i10) /* Producto por un escalar */ 3*a;
(%o10) [3,0,9]

(%i11) /* Producto escalar, vectorial y mixto */
       a.b;
       a~b;
       c.(a~b);
(%o11) 14
(%o12) [1,0,3]~[2,-1,4]
(%o13) [3,-2,1] . [1,0,3]~[2,-1,4]

(%i14) /* Por defecto, una operación con productos vectoriales
       se deja indicada.Para que exprese los productos vectoriales,
       usemos express() */
       express(a~b);
       express(c.(a~b));
(%o14) [3,2,-1]
(%o15) 4

(%i16) /* Módulo de un vector y vector unitario */
       amod:sqrt(a.a);
       au:a/amod;
(%o16) sqrt(10)
(%o17) [1/sqrt(10),0,3/sqrt(10)]
```

```
(%i18) /* ******** */
       /* Matrices */
       /* ******** */
       kill(all)$
       d:matrix([1,2,3],[3,2,-4],[0,1,5]);
       e:matrix([0,-2,1],[-2,0,3],[2,-5,4]);
```

$$(\%o1)\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & -4 \\ 0 & 1 & 5 \end{bmatrix}$$

$$(\%o2)\begin{bmatrix} 0 & -2 & 1 \\ -2 & 0 & 3 \\ 2 & -5 & 4 \end{bmatrix}$$

```
(%i3) /* Las matrices y las listas son muy parecidas */
      /* Puede distinguirse a simple vista si tenemos un vector
       o una matriz,observando que en las componentes de
      los vectores se separan con comas, las matrices
      no incluyen la coma */
      v:[1,2,3];
      m:matrix([1,2,3]);
(%o3) [1,2,3]
```

$$(\%o4)\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
(%i5) /* Aunque tenemos dos instrucciones que determinan si
      tenemos una lista (vector) o una matriz */
      listp(v);
      listp(m);
      matrixp(v);
      matrixp(m);
(%o5) true
(%o6) false
(%o7) false
(%o8) true
```

```
(%i9) /* ***************************** */
      /* Operaciones con matrices */
      /* ***************************** */

      /* Se pueden definir elemento a elemento */
      a:matrix([1,2,3],[3,2,4],[0,1,5]);
      b:matrix([0,-2,1],[-2,0,3],[2,-5,4]);
```

$$(\%o9)\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 4 \\ 0 & 1 & 5 \end{bmatrix}$$

$$(\%o10)\begin{bmatrix} 0 & -2 & 1 \\ -2 & 0 & 3 \\ 2 & -5 & 4 \end{bmatrix}$$

```
(%i11) /* ¿Cuál es el elemento a(2,3)? Tenemos dos maneras equivalentes */

       a[2,3];
       a[2][3];
(%o11) 4
(%o12) 4
```

```
(%i13) /* Suma y multiplicaciones por un escalar */
       a+b;
       3*a;
       a/3;
```

$$(\%o13)\begin{bmatrix}1 & 0 & 4\\ 1 & 2 & 7\\ 2 & -4 & 9\end{bmatrix}$$

$$(\%o14)\begin{bmatrix}3 & 6 & 9\\ 9 & 6 & 12\\ 0 & 3 & 15\end{bmatrix}$$

$$(\%o15)\begin{bmatrix}\frac{1}{3} & \frac{2}{3} & 1\\ 1 & \frac{2}{3} & \frac{4}{3}\\ 0 & \frac{1}{3} & \frac{5}{3}\end{bmatrix}$$

```
(%i16) /* Producto de matrices */
       a.b;
```

$$(\%o16)\begin{bmatrix}2 & -17 & 19\\ 4 & -26 & 25\\ 8 & -25 & 23\end{bmatrix}$$

```
(%i17) /* ten cuidado que a*b multiplica cada elemento de
       la primera matriz por cada elemetno de la segunda */
       a*b;
```

$$(\%o17)\begin{bmatrix}0 & -4 & 3\\ -6 & 0 & 12\\ 0 & -5 & 20\end{bmatrix}$$

```
(%i18) /* Una operación sobre una matriz se aplica a cada
          uno de sus elementos */
       a;
       a+2;
       a^2;
       sqrt(a);
```

$$(\%o18)\begin{bmatrix}1 & 2 & 3\\ 3 & 2 & 4\\ 0 & 1 & 5\end{bmatrix}$$

$$(\%o19)\begin{bmatrix}3 & 4 & 5\\ 5 & 4 & 6\\ 2 & 3 & 7\end{bmatrix}$$

$$(\%o20)\begin{bmatrix}1 & 4 & 9\\ 9 & 4 & 16\\ 0 & 1 & 25\end{bmatrix}$$

$$(\%o21)\begin{bmatrix}1 & \sqrt{2} & \sqrt{3}\\ \sqrt{3} & \sqrt{2} & 2\\ 0 & 1 & \sqrt{5}\end{bmatrix}$$

```
(%i22) /* Inversa, determinante y transpuesta de una matriz */
       invert(a);
       determinant(a);
       transpose(a);
```

$$(\%o22)\begin{bmatrix}-\frac{2}{5} & \frac{7}{15} & -\frac{2}{15}\\ 1 & -\frac{1}{3} & -\frac{1}{3}\\ -\frac{1}{5} & \frac{1}{15} & \frac{4}{15}\end{bmatrix}$$

$$(\%o23)\ -15$$

$$(\%o24)\begin{bmatrix}1 & 3 & 0\\ 2 & 2 & 1\\ 3 & 4 & 5\end{bmatrix}$$

Creación de tablas, listas y matrices,
Orden makelist y orden map

```
(%i25) kill(all)$
```

```
(%i1) /* Se pueden crear tablas automáticamente con makelist */
      makelist(k^2,k,-5,5);
(%o1) [25,16,9,4,1,0,1,4,9,16,25]
```

```
(%i2) /* Podemos modificar la lista con una operación que se
      aplicará a cada elemento */
      makelist(2*k,k,1,8)-1;
(%o2) [1,3,5,7,9,11,13,15]
```

```
(%i3) /* Podemos guardarlo en una lista o vector */
      v:makelist(3*i,i,0,4)-1.5;
(%o3) [-1.5,1.5,4.5,7.5,10.5]
```

```
(%i4) /* Podemos sumar o multiplicar todos los elementos de una lista */
      apply("+",v);
      apply("*",v);
(%o4) 22.5
(%o5) -797.344
```

```
(%i6) /* A veces queremos evaluar una lista a través de la sustitución
      de unos valores en una función previamente definida.
      Esto se hace con map(función,[valor1,valor2,etc,..]) */

      /* Luego daremos detalles sobre definición de funciones,
      de momento definamos la función mediante la orden */
      f(x):=x^2;
(%o6) f(x) := x^2
```

```
(%i7) /* Con map aplicada a una lista, sustituimos cada elemento
      de la lista por el valor de f aplicada a ese elemento,
      obteniendo una nueva lista */
      map(f,[1,2,y,4]);
(%o7) [1,4,y^2,16]
```

```
(%i8) /* Ya que estamos con la función map,
      nótese la siguiente diferencia. Si se aplica a una suma
      de términos, map evalúa la suma de f
      aplicada a cada sumando */
      map(f,a+b+y^2+c);
(%o8) y^4+c^2+b^2+a^2
```

```
(%i9) /* Aunque hay que tener cuidado, porque, si se puede, antes de
      evaluar map, wxmaxima intenta hacer la operación que se indica
      en el argumento de map.
      En el siguiente ejemplo, primero se suma 1+2+4 y se evalua f(7) */
      map(f,1+2+y^2+4);
(%o9) y^4+49
```

```
(%i10) /* Nótense los resultados no deseados que generan las siguientes
       instrucciones */
       map(f(x),[1,2,y,4]);
       map(x^2,[1,2,y,4]);
(%o10) [(x^2)(1),(x^2)(2),(x^2)(y),(x^2)(4)]
(%o11) [(x^2)(1),(x^2)(2),(x^2)(y),(x^2)(4)]
```

```
(%i12) /* Esta función map nos será muy útil para manejar ecuaciones
       vectoriales que resulten de igualar dos vectores a y b */
       a:[x+y,x-y,3*z+4*y-x];
       b:[5,2,7];
```

$$(\%o12)\quad [y+x,x-y,3z+4y-x]$$

$$(\%o13)\quad [5,2,7]$$

```
(%i14) /* Si queremos resolver a=b, podríamos pensar en definir una
       variable, eq, que contenga esta ecuación en la forma */
       eq:a=b;
```

$$(\%o14)\quad [y+x,x-y,3z+4y-x]=[5,2,7]$$

```
(%i15) /* Aunque a nosotros nos parece una ecuación, wxmaxima ve una
       lista igualada a otra lista y no será capaz de resolverla,
       pues no se ajusta a su sintaxis.

       Cada ecuación escalar debe ser completamente declarada en cada
       elemento de la lista.

       En otras palabras, para resolver ecuaciones vectoriales,
       necesitaremos escribirlas en la forma [y+x=5,x-y=2,3*z+4*y-x=7].
       De este modo, el primer elemento es una ecuación escalar,
       el segundo elemento de esta lista es otra ecuación escalar,
        y lo mimso ocurre con el tercero.

       Para conseguir esta sintaxis, usemos la función map con la
       función "=" que actúe sobre los dos términos y genere una
       lista donde el  primer elemento sea una ecuación y el segundo
        sea otra ecuación */

       c:map("=",a,b);
```

$$(\%o15)\quad [y+x=5,x-y=2,3z+4y-x=7]$$

Resolución de ecuaciones

```
(%i16) /* ****************************************** */
       /* Resolución de ecuaciones con una variable */
       /* ****************************************** */
       kill(all)$
```

```
(%i1) eq:x+5=x^2-4;
      solve(eq,x);
```

$$(\%o1)\quad x+5=x^2-4$$

$$(\%o2)\quad \left[x=-\frac{\sqrt{37}-1}{2},x=\frac{\sqrt{37}+1}{2}\right]$$

```
(%i3) /* A veces la ecuación tiene dos partes que se calculan por
      separado y después se igualan */
      a:x+5;
      b:x^2-4;
```

$$(\%o3)\quad x+5$$

$$(\%o4)\quad x^2-4$$

```
(%i5) /* Definamos una ecuación igualando ambas partes */
      eq:a=b;
```

(%o5) $x+5=x^2-4$

```
(%i6) solve(eq,x);
```

(%o6) $[x=-\frac{\sqrt{37}-1}{2},x=\frac{\sqrt{37}+1}{2}]$

```
(%i7) /* Si queremos guardar las soluciones en una variable,
      debemos almacenar la solución en una variable averiguar qué
      estructura tiene (es una lista) y extraer la información
      de la misma */
      sol:solve(eq,x);
```

(%o7) $[x=-\frac{\sqrt{37}-1}{2},x=\frac{\sqrt{37}+1}{2}]$

```
(%i8) listp(sol);
```

(%o8) *true*

```
(%i9) /* Es una lista (o un vector). Ya se adivinaba pues teníamos
      dos variables separadas con coma entre corchetes.
      ¿Cuánto mide? */
      length(sol);
```

(%o9) 2

```
(%i10) /* Efectivamente, tiene dos elementos porque la ecuación
       tiene dos soluciones ¿Cuáles son esos elementos? */
       sol[1];sol[2];
```

(%o10) $x=-\frac{\sqrt{37}-1}{2}$

(%o11) $x=\frac{\sqrt{37}+1}{2}$

```
(%i12) /* Cada elemento de sol tiene dos partes, la izquierda y la derecha
       (lefth hand side y right hand side) */
       lhs(sol[1]);
       rhs(sol[1]);
```

(%o12) x

(%o13) $-\frac{\sqrt{37}-1}{2}$

```
(%i14) /* Está claro que la parte izquierda es un texto que indica qué
          variable estabamos resolviendo y que el valor numérico que buscamnos
          está en la parte derecha de cada elemento de la lista.
          Si queremos guardar las dos soluciones en variables que luego
          podamos utilizar, almacenemos esta parte derecha en variables,
          cuyo nombre podemos elegir libremente */

          x1:rhs(sol[1]);
          x2:rhs(sol[2]);
```

$$(\%o14)\quad -\frac{\sqrt{37}-1}{2}$$

$$(\%o15)\quad \frac{\sqrt{37}+1}{2}$$

```
(%i16) /* Vemos que ahora tenemos dos variables con el valor numérico
           obtenido al resolver */
          x1;x2;
```

$$(\%o16)\quad -\frac{\sqrt{37}-1}{2}$$

$$(\%o17)\quad \frac{\sqrt{37}+1}{2}$$

Resolución de sistemas de ecuaciones

```
(%i18) solve([x+y=5,x-y=1],[x,y]);
(%o18) [[x=3,y=2]]

(%i19) /* Podemos guardar primero las ecuaciones en una variable
           y luego resolver */
          eq:[x+y=5,x-y=1];
          solve(eq,[x,y]);
(%o19) [y+x=5,x-y=1]
(%o20) [[x=3,y=2]]

(%i21) /* Nótese que cada ecuación se corresponde con un elemento
           de la lista eq */

          /* A veces  calculamos la parte izquierda de la
          ecuación y la guardamos en una variable vectorial o lista, a,
          mientras que calculamos la parte de la derecha y la guardamos
          en una lista o vector, b.
          Probemos a resolver como hicimos antes */
          a:[x+y,x-y];
          b:[5,1];
          eq:a=b;
(%o21) [y+x,x-y]
(%o22) [5,1]
(%o23) [y+x,x-y]=[5,1]
```

```
(%i24) /* Vemos que la lista eq no es exactamente como antes.
       Aparece una lista, un igual y otra lista.
       La ecuación 1 no está en el elemento 1 de una lista, sino que
       una parte se define en un lado de la igualdad y la otra parte
       en el otro.
       Si intentamos su resolución, no obtenemos el resultado buscado */

       solve(eq,[x,y]);
(%o24) []
```

```
(%i25) /* Como intuíamos, wxmaxima no reconoce la ecuación como hemos
       hecho nosotros. Necesita que se la pongamos en una lista
       cuyo primer elemento sea la primera ecuación completa, y+x=5,
       y cuyo segundo elemento sea la segunda ecuación completa, x-y=1.

       Usemos la función map que describimos en el apartado de
       creación de tablas, listas y marices */

       eq:map("=",a,b);
(%o25) [y+x=5,x-y=1]
```

```
(%i26) /* Ahora sí tenemos la sintaxis adecuada y wxmaxima resuelve */
       solve(eq,[x,y]);
(%o26) [[x=3,y=2]]
```

```
(%i27) /* También lo podíamos haber hecho generando una lista con makelist.
       Elegid la forma que mejor entendáis */

       a:[x+y,x-y];
       b:[5,1];
       eq:makelist(a[i]=b[i],i,1,length(a));
(%o27) [y+x,x-y]
(%o28) [5,1]
(%o29) [y+x=5,x-y=1]
```

```
(%i30) /* La asignación del resultado en variables para poder operar
       después es algo más compleja que antes,
       pues la lista resultante tiene más índices
       (uno para cada ecuación y otro para cada solución, si hay más de una)
       Si queremos asignar el resultado a variables, guardemos el resultado
       en una variable y veamos la estructura de
       la variable tipo lista resultante */

       sol:solve(eq,[x,y]);
       listp(sol);
       length(sol);
(%o30) [[x=3,y=2]]
(%o31) true
(%o32) 1
```

```
(%i33) /* Vemos que es una lista de longitud uno, porque la solución
       es única.
       Pero observemos bien la variable sol */
       sol;
(%o33) [[x=3,y=2]]
```

```
(%i34) /* Vemos que aunque tiene longitud 1, aparecen dos corchetes
       anidados, en lugar del único corchete que obtuvimos al resolver
        una sola ecuación.
       Esto es debido a que le hemos pedido dos variables, x e y.
       Esto significa que tenemos una lista de una fila
       (una solución y un índice principal)
       y dos índices secundarios (variables x e y).
       Veamos su único elemento */
       sol[1];
(%o34) [x=3,y=2]
```

```
(%i35) /* Este elemento es una lista con subíndices (para x e y) */
       sol[1][1];
       sol[1][2];
(%o35) x=3
(%o36) y=2
```

```
(%i37) /* Ya podemos identificar que el valor numérico está en
       la parte derecha y asignarlo a la variable que queramos */
       a:rhs(sol[1][1]);
       b:rhs(sol[1][2]);
(%o37) 3
(%o38) 2
```

```
(%i39) /* A veces el sistema de ecuacines tiene más de una
       solución. El vector de salida tiene más componentes.
       Se deja al lector el análisis del contenido
       de cada comopnente mostrada */
       sol:solve([x^2+y-5=0,x-y=4],[x,y]);
       sol[1];
       sol[2];
       sol[1][1];
       sol[1][2];
       sol[2][1];
       sol[2][2];
```

$$(\%o39)\quad \left[\left[x=-\frac{\sqrt{37}+1}{2},y=-\frac{\sqrt{37}+9}{2}\right],\left[x=\frac{\sqrt{37}-1}{2},y=\frac{\sqrt{37}-9}{2}\right]\right]$$

$$(\%o40)\quad \left[x=-\frac{\sqrt{37}+1}{2},y=-\frac{\sqrt{37}+9}{2}\right]$$

$$(\%o41)\quad \left[x=\frac{\sqrt{37}-1}{2},y=\frac{\sqrt{37}-9}{2}\right]$$

$$(\%o42)\quad x=-\frac{\sqrt{37}+1}{2}$$

$$(\%o43)\quad y=-\frac{\sqrt{37}+9}{2}$$

$$(\%o44)\quad x=\frac{\sqrt{37}-1}{2}$$

$$(\%o45)\quad y=\frac{\sqrt{37}-9}{2}$$

```
(%i46) /* Podemos resolver ecuaciones eligiendo las variables a resolver */
       solve([x+y+z-5=0,x-y+3*z=4],[x,y]);
```

$$(\%o46)\quad \left[\left[x=-\frac{4\,z-9}{2},y=\frac{2\,z+1}{2}\right]\right]$$

```
(%i47) /* A veces la solución exacta de maxima no nos gusta y queremos
       que evalúe el valor numérico */
       eq:[x*2-1/3=0];
       solve(eq);
```

$$(\%o47)\quad \left[2\,x-\frac{1}{3}=0\right]$$

$$(\%o48)\quad \left[x=\frac{1}{6}\right]$$

```
(%i49) /* A veces la solución exacta de maxima no nos gusta y cuando
          decimos que evalúe el valor numérico, tampoco */
       eq:[x^2-1/2=0];
       solve(eq),numer;
```

(%o49) $[x^2-\frac{1}{2}=0]$

```
rat: replaced -0.5 by -1/2 = -0.5
rat: replaced -0.5 by -1/2 = -0.5
rat: replaced 0.5 by 1/2 = 0.5
rat: replaced 0.5 by 1/2 = 0.5
rat: replaced -0.707107 by -15994428/22619537 = -0.707107
rat: replaced 2.0 by 2/1 = 2.0
rat: replaced -0.707107 by -15994428/22619537 = -0.707107
rat: replaced 4.42096e-8 by 1/22619537 = 4.42096e-8
rat: replaced 2.0 by 2/1 = 2.0
rat: replaced -0.707107 by -15994428/22619537 = -0.707107
rat: replaced -0.707107 by -15994428/22619537 = -0.707107
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced -0.707107 by -15994428/22619537 = -0.707107
rat: replaced 4.42096e-8 by 1/22619537 = 4.42096e-8
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced -0.707107 by -15994428/22619537 = -0.707107
```

(%o50) $[x=0.707107\ \%e^{1.0\ \%i\ \pi},x=0.707107\ \%e^{2.0\ \%i\ \pi}]$

```
(%i51) /* ¡Tranquilos! No hemos hecho nada mal, el programa nos avisa de
          que ha hecho algunas aproximaciones con las fracciones (rationals)
          Podemos decirle que no nos informe de estas aproximaciones */

       ratprint:false$
       sol:solve(eq),numer;
```

(%o52) $[x=0.707107\ \%e^{1.0\ \%i\ \pi},x=0.707107\ \%e^{2.0\ \%i\ \pi}]$

```
(%i53) /* Aunque es correcta, el término exponencial complejo puede no
          gustarnos. Podemos decirle que desarolle o expanda la solución */

       expand(sol);
```

(%o53) $[x=-0.707107,x=0.707107]$

```
(%i54) /* La función expand desarrolla productos, polinomios,etc.
          Si tuviéramos expresiones trigonométricas,
          utilizaríamos trigexpand() */

       sin(x+y);
```

(%o54) $\sin(y+x)$

```
(%i55) trigexpand(sin(x+y));
```

(%o55) $\cos(x)\sin(y)+\sin(x)\cos(y)$

```
(%i56) /* Para finalizar, en caso de sistemas de ecuaciones lineales,
       es más eficiente utilizar linsolve */
       eq:[x+y=0,x-y=5];
       linsolve(eq,[x,y]);
(%o56) [y+x=0,x-y=5]
(%o57) [x=5/2,y=-5/2]
```

POLINOMIOS

```
(%i58) /* ********************************************************** */
       /* Definición de polinomios y expresiones.
       Simplificaciones, desarrollo de expresiones, factorización,... */
       /* ********************************************************** */
       kill(all)$
       p:(x-4)*(x+1);
(%o1) (x-4) (x+1)

(%i2) p;
(%o2) (x-4) (x+1)

(%i3) /* Para verlo desarrollado */
      expand(p);
(%o3) x^2-3 x-4

(%i4) /* Pare evaluarlo, tenemos distintas opciones */
      ev(p,x=2);
      p,x=y;
      expand(p),x=y;
(%o4) -6
(%o5) (y-4) (y+1)
(%o6) y^2-3 y-4

(%i7) /* Hay otras funciones que ayudan a simplifiar, factorizar
      operaciones con polinomios */
      p:(x-4)*(x+1);
      q:expand(p);
      factor(q);
(%o7) (x-4) (x+1)
(%o8) x^2-3 x-4
(%o9) (x-4) (x+1)

(%i10) /* Si tenemos  logaritmos, exponenciales y radicales,
       la simplificación se hace con radcan() */
       (%e^x-1)/(1+%e^(x/2));
       radcan((%e^x-1)/(1+%e^(x/2)));
(%o10) (%e^x-1)/(%e^(x/2)+1)
(%o11) %e^(x/2)-1
```

```
(%i12) /* Hay otras funciones útiles cuyo significado puedes
       buscar en la ayuda */
       /* radcan radsimp, trigexpand,trigsimp,trigreduce,...
        son algunos ejemplos */

       /* Por ejemplo, si tenemos expresiones racionales.
       Podemos simplificarlas con ratsimp() */

       p:(x+1)*(x-1)/(x^2+3*x-4);
       ratsimp(p);
```

$$(\%o12)\ \frac{(x-1)\ (x+1)}{x^2+3\,x-4}$$

$$(\%o13)\ \frac{x+1}{x+4}$$

FUNCIONES Y REPRESENTACIONES GRÁFICAS

```
(%i14) /* ************************ */
       /* Definición de funciones   */
       /* Representaciones gráficas */
       /* ************************ */

       f(x):=3*x^2-4;
       g(x):=exp(x)-2;
       /* Aunque resulta lo mismo si escribimos */

       h(x):=%e^x-2;
```

$$(\%o14)\ f(x):=3\,x^2-4$$

$$(\%o15)\ g(x):=\exp(x)-2$$

$$(\%o16)\ h(x):=\%e^x-2$$

```
(%i17) /* También podemos definir una función así */

       define(j(x),x^3);
```

$$(\%o17)\ j(x):=x^3$$

```
(%i18) /* Para evaluarlas */
       f(2);
       g(3);
       g(y);
```

$$(\%o18)\ 8$$

$$(\%o19)\ \%e^3-2$$

$$(\%o20)\ \%e^y-2$$

```
(%i21) /* Podemos definir funciones en tramos usando condiciones if */
       f(x):=if x<0 then x^2 else sin(x);
       g(x):=if x<=0 then x^2 else if x<3 then sqrt(x) else 3*x;
```

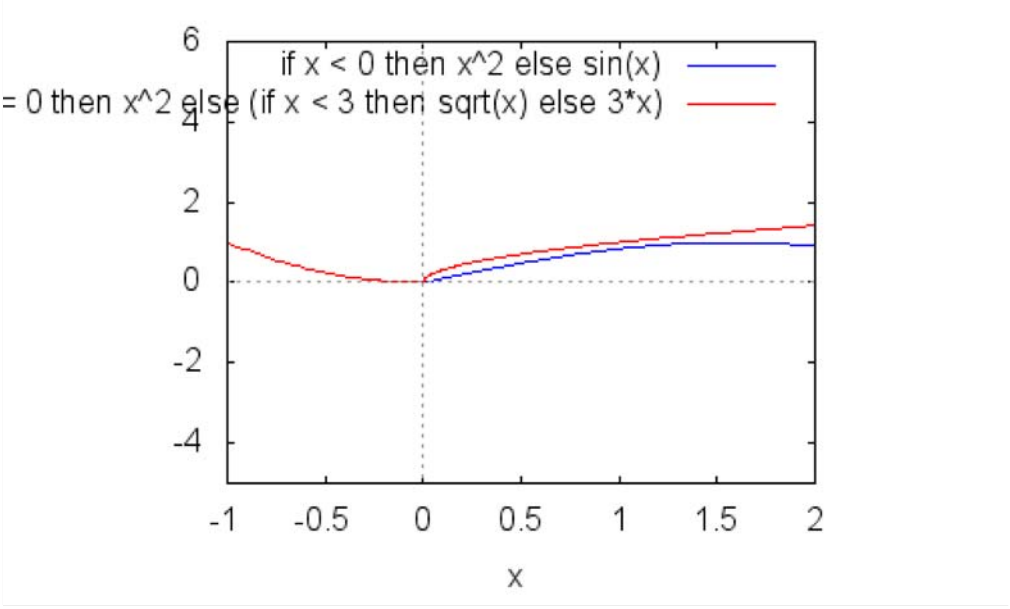
$$(\%o21)\ f(x):=\text{if}\ \ x<0\ \ \text{then}\ \ x^2\ \ \text{else}\ \ \sin(x)$$

$$(\%o22)\ g(x):=\text{if}\ \ x<=0\ \ \text{then}\ \ x^2\ \ \text{else}\ \ \text{if}\ \ x<3\ \ \text{then}\ \ \sqrt{x}\ \ \text{else}\ \ 3\,x$$

```
(%i23) /* Si queremos representarlas */
       plot2d(f(x),[x,-1,2],[gnuplot_preamble, "set grid;"])$
```

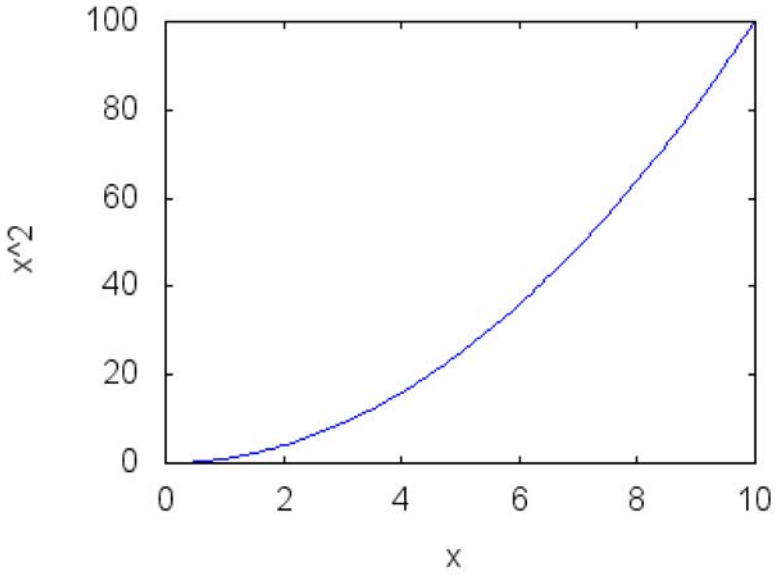
```
(%i24) /* Esto hace que se abra una ventana de gnuplot. Podemos
       decirle que represente la función sin salir de esta pantalla */
       wxplot2d([f(x),g(x)],[x,-1,2],[y,-5,6])$
```

(%t24)




```
(%i25) /* Podemos controlar las marcas y decirle que dibuje
       marcas en el eje x, xtic, empezando desde 0 y
       aumentando de 2 en 2 hasta 10 */
       wxplot2d(x^2,[x,0,10],[gnuplot_preamble,
        "set xtics 0,2,10; set grid;"]);
```

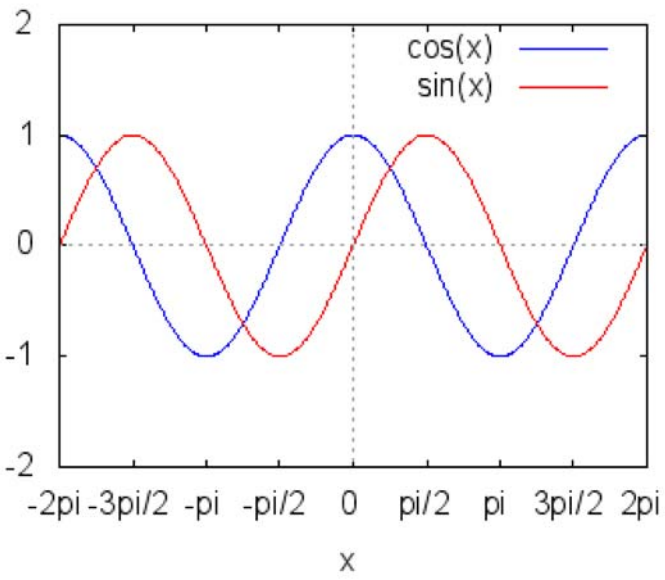
(%t25)



(%o25)

```
(%i26) /* Podemos controlar las marcas y asignar una leyenda a cada marca */
       wxplot2d([cos(x),sin(x)],[x,-2*%pi,2*%pi],[y,-2,2],
       [gnuplot_preamble,"set xzeroaxis;
       set xtics ('-2pi' -6.283, '-3pi/2' -4.712,
       '-pi' -3.1415, '-pi/2' -1.5708,  '0' 0,'pi/2' 1.5708,
        'pi' 3.1415,'3pi/2' 4.712,
       '2pi' 6.283);set ytics ('-2' -2, '-1' -1,
       '0' 0, '1' 1,  '2' 2); set grid;"]);
```

(%t26)

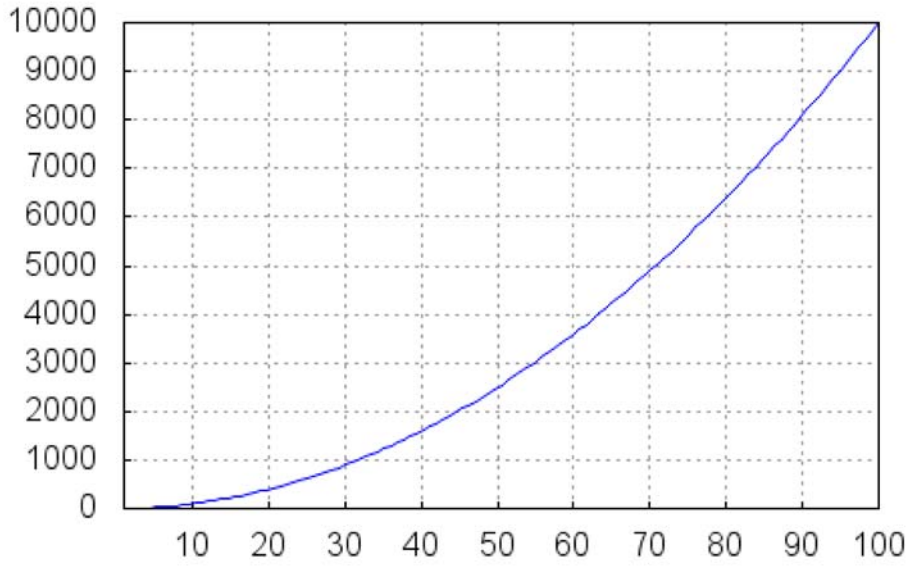


(%o26)

```
(%i27) /* Son más prácticos los comandos draw y wxdraw.
       Hay que cargar el paquete de dibujo */
       load(draw)$
;; loading #P"C:/Users/usuario/maxima/binary/5_35_1_2/sbcl/1_2_7/share/draw/
;; loading #P"C:/Users/usuario/maxima/binary/5_35_1_2/sbcl/1_2_7/share/draw/
;; loading #P"C:/Users/usuario/maxima/binary/5_35_1_2/sbcl/1_2_7/share/draw/
;; loading #P"C:/Users/usuario/maxima/binary/5_35_1_2/sbcl/1_2_7/share/draw/
```

(%i28)
```
/* Podemos dibujar funciones definidas de forma explícita */
wxdraw2d(explicit(x^2,x,1,100),grid=true);
```

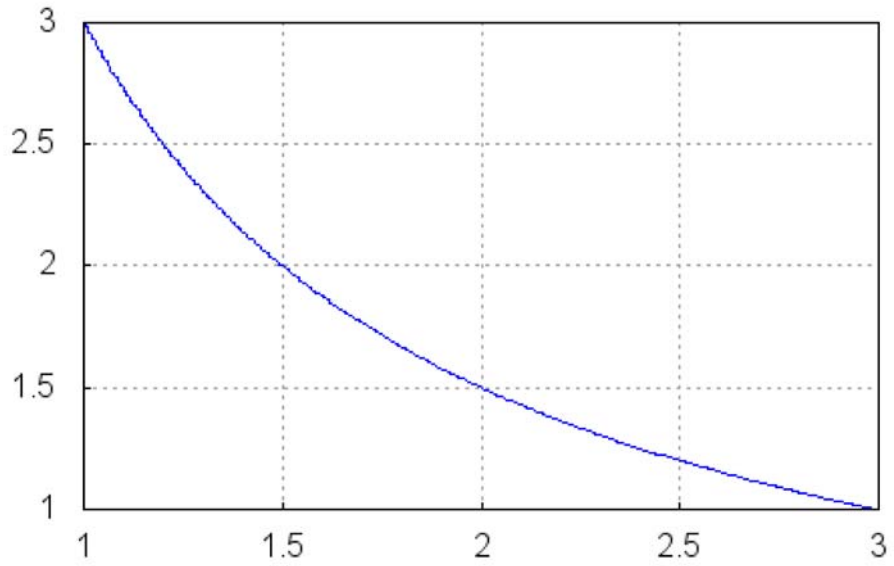
(%t28)



(%o28)

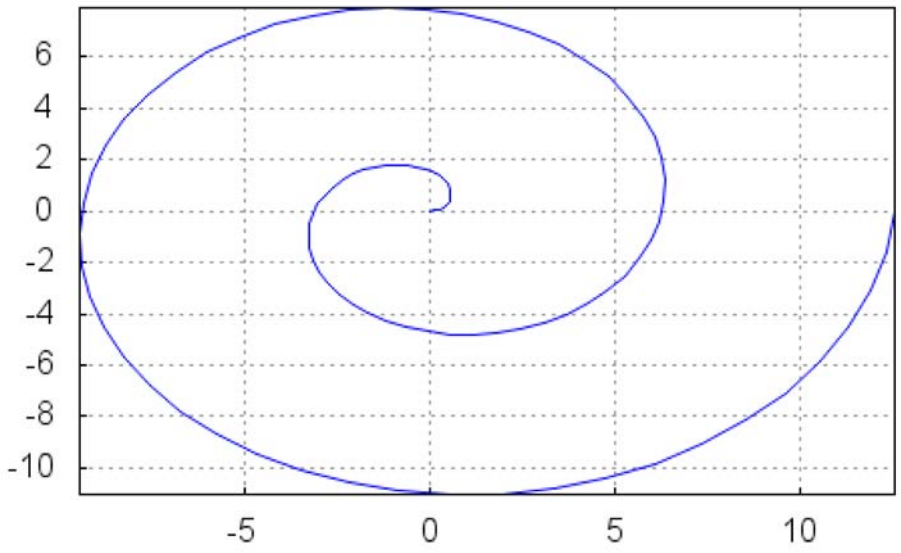
(%i29)
```
/* Podemos dibujar funciones definidas de forma implícita
 mediante una ecuación */
wxdraw2d(color=blue,nticks=100,
implicit(y*x=3,x,1,3,y,1,3),grid=true)$
```

(%t29)



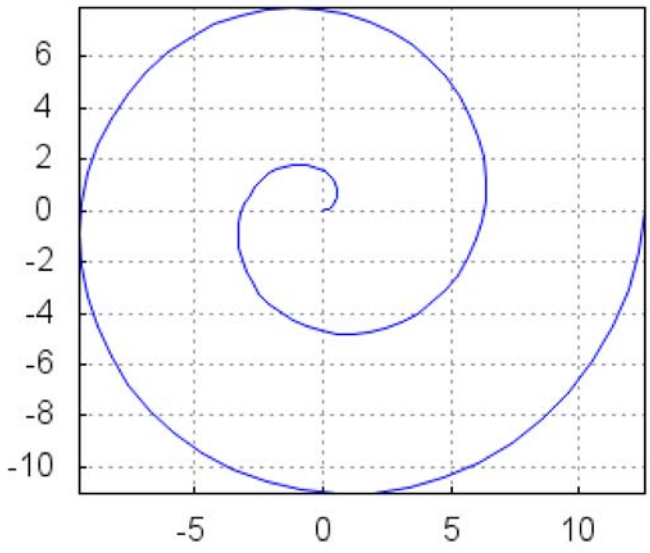
```
(%i30) /* O podemos dibujar funciones definidas en forma paramétrica */
       wxdraw2d(color=blue,nticks=100,
       parametric(t*cos(t),t*sin(t),t,0,4*%pi),grid=true)$
```

(%t30)



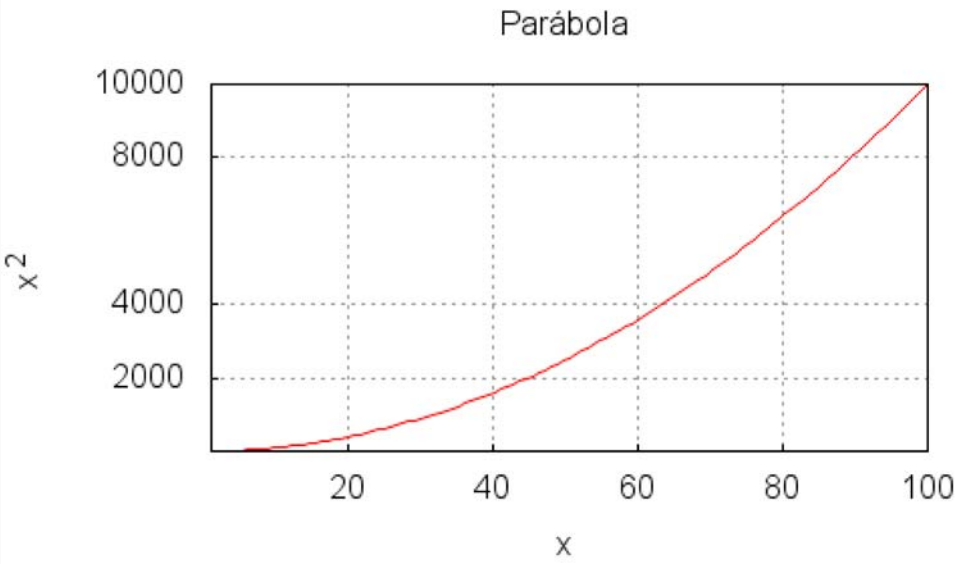

```
(%i31) /* El eje x es más largo que el y, lo que deforma la imagen,
       fijemos longitud proporcional para ambos ejes */
       wxdraw2d(color=blue,nticks=100,
       parametric(t*cos(t),t*sin(t),t,0,4*%pi),
       grid=true,proportional_axes=xy)$
```

(%t31)



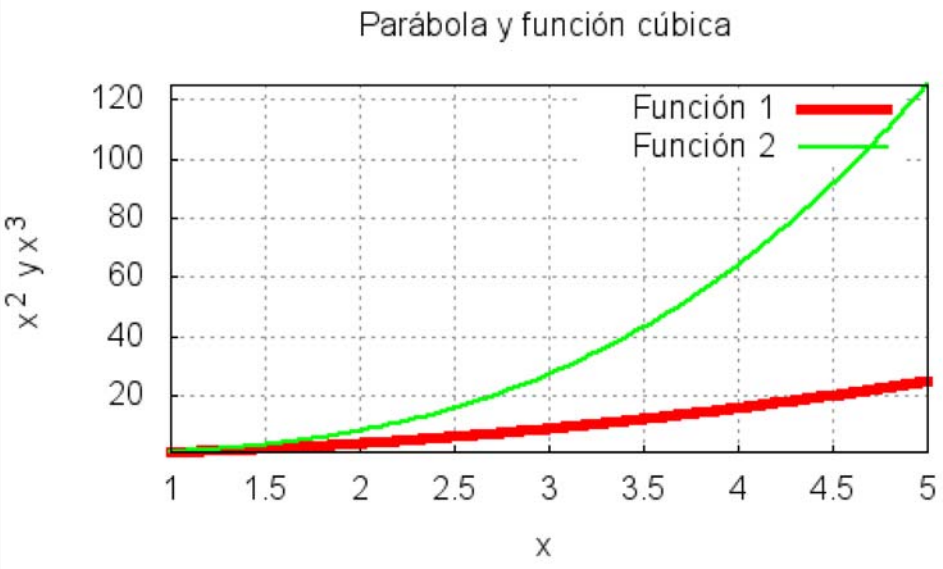
(%i32)
```
/* Podemos controlar las marcas y el grid, poner título a los
ejes y la gráfica, cambiar el color de la curva */
wxdraw2d(
  color=red,explicit(x^2,x,1,100),
  grid=true,
  xtics={0,20,40,60,80,100},
ytics={2000,4000,8000,10000},
title="Parábola",
xlabel="x",ylabel="x^2")$
```

(%t32)



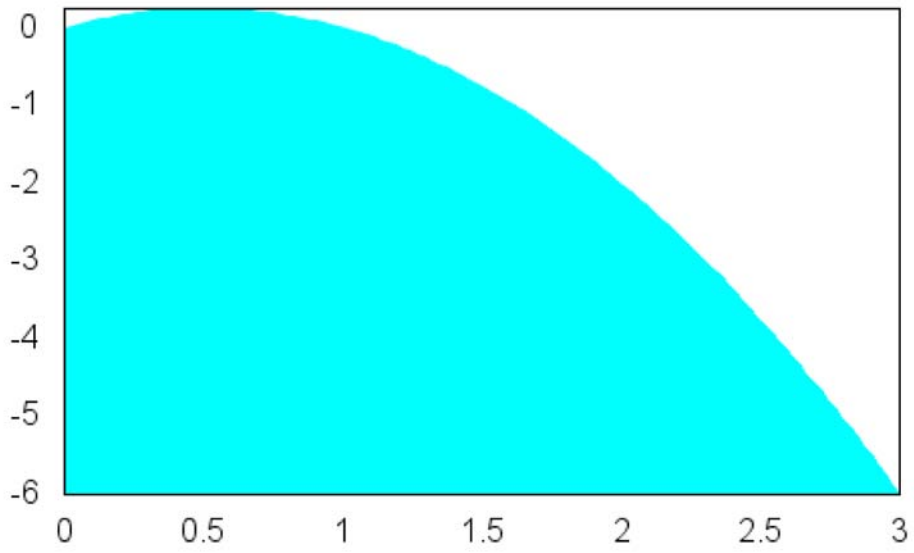
(%i33)
```
/* Podemos dibujar dos funciones, controlando color,
ancho de línea */
wxdraw2d(
  color=red,line_width=5,key="Función 1",explicit(x^2,x,1,5),
  color=green,line_width=2,key="Función 2",explicit(x^3,x,1,5),
grid=true,
title="Parábola y función cúbica",
xlabel="x",ylabel="x^2 y x^3")$
```

(%t33)



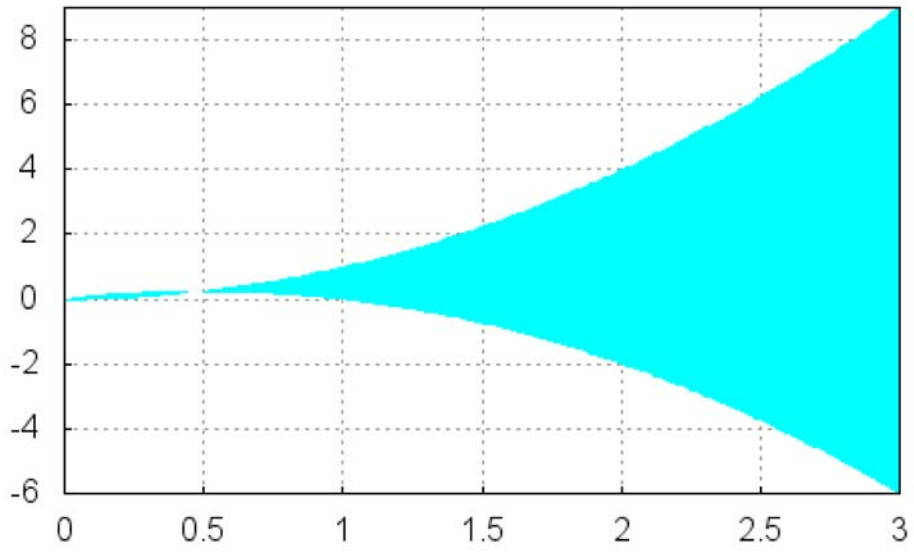
(%i34)
```
/* Podemos decirle que rellene con un color desde la
función hasta el eje x */
wxdraw2d(
  filled_func=true,fill_color=cyan,explicit(-x^2+x,x,0,3))$
```

(%t34)



(%i35)
```
/* Si filled_func es una función, rellenamos desde esta
función hasta la función que estemos representando */
wxdraw2d(
filled_func=x^2,fill_color=cyan,explicit(-x^2+x,x,0,3),grid=true)$
```

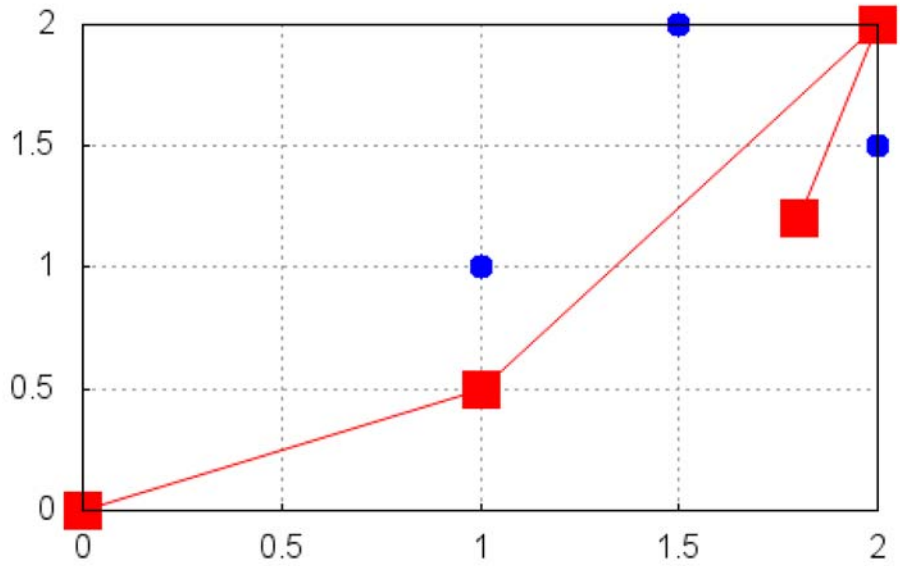
(%t35)



```
(%i36) /* Podemos dibujar una lista de puntos, con o son línea de unión
       Podemos darlos en la forma [[x1,y1],[x2,y2],...] */
       puntos1:[[1,1],[2,1.5],[1.5,2]];
       puntos2:[[0,0],[1,0.5],[2,2],[1.8,1.2]];


       wxdraw2d(color=blue,
       point_size=2,
       point_type=7,
       points_joined=false,
       points(puntos1),
       color=red,
       point_size=3,
       point_type=5,
       points_joined=true,
       points(puntos2)
       ,grid=true)$
(%o36) [[1,1],[2,1.5],[1.5,2]]
(%o37) [[0,0],[1,0.5],[2,2],[1.8,1.2]]
```

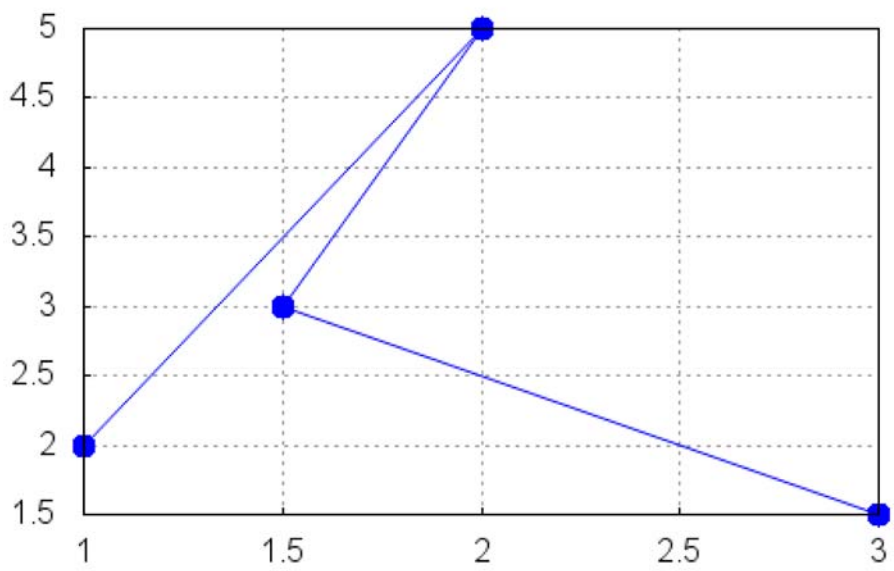
(%t38)




```
(%i39) /*  bien mediante points ([x1,x2,...], [y1,y2,...]) */
       puntosx:[1.,2.,1.5,3.];
       puntosy:[2.,5.,3.,1.5];
(%o39) [1,2,1.5,3]
(%o40) [2,5,3,1.5]
```

```
(%i41) wxdraw2d(color=blue,
       point_size=2,
       point_type=7,
       points_joined=true,
       points(puntosx,puntosy),
       grid=true)$
```

(%t41)




```
(%i42) /* Hay muchas posibilidades.
       Podemos dibujar puntos y figuras más complejas.
       Podéis buscar ejemplos como éste y más complejos */

       /* Preparamos una lista con
       los puntos [j,sin(j)]
       variando j de 0 a 8 de 0.5 en 0.5 */

       ejexseno:makelist(j,j,0,16)/2;
       pseno(j):=[j,sin(j)];
       puntosseno:map(pseno,ejexseno);
```

(%o42) $[0,\frac{1}{2},1,\frac{3}{2},2,\frac{5}{2},3,\frac{7}{2},4,\frac{9}{2},5,\frac{11}{2},6,\frac{13}{2},7,\frac{15}{2},8]$

(%o43) $\text{pseno}(j):=[j,\sin(j)]$

(%o44) $[[0,0],[\frac{1}{2},\sin\left(\frac{1}{2}\right)],[1,\sin(1)],[\frac{3}{2},\sin\left(\frac{3}{2}\right)],[2,\sin(2)],[\frac{5}{2},\sin\left(\frac{5}{2}\right)],[3,\sin(3)],[\frac{7}{2},\sin\left(\frac{7}{2}\right)],[4,\sin(4)],[\frac{9}{2},\sin\left(\frac{9}{2}\right)],[5,\sin(5)],[\frac{11}{2},\sin\left(\frac{11}{2}\right)],[6,\sin(6)],[\frac{13}{2},\sin\left(\frac{13}{2}\right)],[7,\sin(7)],[\frac{15}{2},\sin\left(\frac{15}{2}\right)],[8,\sin(8)]]$

```
(%i45) /* Preparamos una lista con
       los puntos [j,-1+j/4]
       variando j de 0 a 8 de 1 en 1 */

       ejexrecta:makelist(j,j,0,8);
       precta(j):=[j,-1+j/4];
       puntosrecta:map(precta,ejexrecta);
```

(%o45) $[0,1,2,3,4,5,6,7,8]$

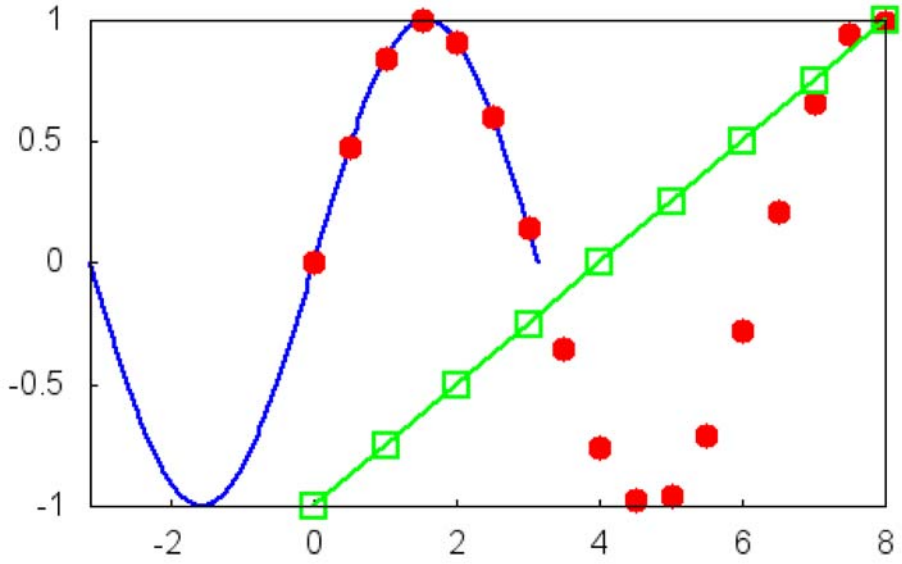
(%o46) $\text{precta}(j) := [j, -1+\frac{j}{4}]$

(%o47) $[[0,-1],[1,-\frac{3}{4}],[2,-\frac{1}{2}],[3,-\frac{1}{4}],[4,0],[5,\frac{1}{4}],[6,\frac{1}{2}],[7,\frac{3}{4}],[8,1]]$

```
(%i48) /* Ya podemos combinar el dibujo de una curva continua
       correspondiente a sin(x) desde -pi a pi,
       la gráfica de puntos puntosseno y
       la gráica e puntos puntosrecta */


       wxdraw2d(
       color=blue,line_width=2,explicit(sin(x),x,-%pi,%pi),
       color=red,point_size=2,point_type=7,points(puntosseno),
       points_joined=true,point_type=4,color=green,points(puntosrecta)
       )$
```

(%t48)



Derivación de funciones

```
(%i49) kill(all)$

       f(x):=x^3-4*x^2+5;
```

(%o1) $\text{f}(x) := x^3 - 4\,x^2 + 5$

```
(%i2) g(x):=diff(f(x),x);
```

(%o2) $\text{g}(x) := \text{diff}(\text{f}(x), x)$

```
(%i3) g(x);
```
$$3\,x^2-8\,x$$

```
(%i4) g(1);
diff: second argument must be a variable; found 1
#0: g(x=1)
 -- an error. To debug this try: debugmode(true);
```

```
(%i5) /* No sabe evaluar g(1) ¿Qué ocurre?
      La función no se ha evaluado, g(x) es una cadena de texto.
      El comando dobles comilas obliga a wxmaxima a evaluar
      lo que sigue a dichas comillas
      Veamos que ahora g(x) ya sí es una función */

      g(x):=''(diff(f(x),x));
      g(1);
```
$$g\left(x\right):=3\,x^2-8\,x$$
$$-5$$

```
(%i7) /* Podemos evaluar la derivada en un punto */
      ''(diff(f(x),x)),x=1;
```
$$-5$$

```
(%i8) /* Las derivadas de orden superior se evalúan
         especificando el orden */
      ''(diff(f(x),x,2));
      ''(diff(f(x),x,3));
```
$$6\,x-8$$
$$6$$

```
(%i10) /* Si la función depende de varias variables */
       f(x,y):=3*x^2*y-x^3*y^2;
```
$$f\left(x,y\right):=3\,x^2\,y-x^3\,y^2$$

```
(%i11) ''(diff(f(x,y),x));
       ''(diff(f(x,y),y));
```
$$6\,x\,y-3\,x^2\,y^2$$
$$3\,x^2-2\,x^3\,y$$

```
(%i13) /* Para derivadas cruzadas  respecto de x y de y, anidemos
       dos instrucciones diff */
       /* Esto es d/dy(df/dx)) */
        ''(diff(diff(f(x,y),x),y));
       /* Y esto es d/dx(df/dy)) */
        ''(diff(diff(f(x,y),y),x));
```
$$6\,x-6\,x^2\,y$$
$$6\,x-6\,x^2\,y$$

Integración de funciones

```
(%i15) kill(all)$

       f(x):=x^2;
(%o1) f(x) := x^2
```

```
(%i2) /* Integrales indefinidas */
      integrate(f(x),x);
      integrate(x^4,x);
(%o2) x^3/3
(%o3) x^5/5
```

```
(%i4) /* Si la integral tiene algún condicionante,
      wxmaxima lo pregunta y actúa en consecuencia */
      integrate(x^n,x);
Is n equal to -1?n;
(%o4) x^(n+1)/(n+1)
```

```
(%i5) integrate(x^n,x);
Is n equal to -1?y;
(%o5) log(x)
```

```
(%i6) /* Integrales definidas */
      integrate(x^2,x,0,1);
      integrate(x^2,x,a,b);
(%o6) 1/3
(%o7) b^3/3-a^3/3
```

```
(%i8) /* Nuevamente, si hay dudas,las pregunta */
      integrate(x^n,x,a,b);
Is n positive, negative or zero?p;
(%o8) b^(n+1)/(n+1)-a^(n+1)/(n+1)
```

```
(%i9) integrate(x^n,x,a,b);
Is n positive, negative or zero?n;
Is n equal to -1?n;
Is b positive, negative or zero?p;
Is a positive, negative or zero?p;
(%o9) b^(n+1)/(n+1)-a^(n+1)/(n+1)
```

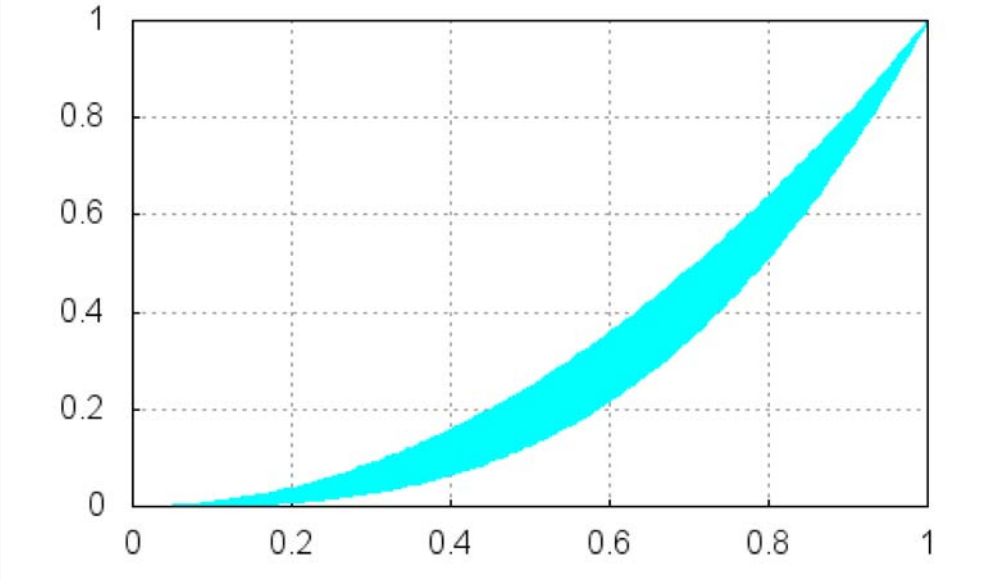
```
(%i10) /* Maxima puede calcular integrales numéricamente. Esto lo podemos
       hacer siempre y será necesario cuando la primitiva no exista */

       /* La función quad_wags hace esta operación dando cuatro salidas:
       la primera es la aproximación numérica de la integral,
       la segunda es el error estimado,la tercera es el número de evaluacion
        y la cuarta un código de error (cero indica que
       no ha habido problemas)*/

       quad_qags(x^2,x,0,1);
(%o10) [0.333333,3.70074 10^-15,21,0]

(%i11) /* Podemos hacer integrales dobles para calcular un área.
       Definamos los límites mediante dos funciones y dibujémoslas */
       kill(all)$
       f1(x):=x^3;
       f2(x):=x^2;
(%o1)  f1(x) := x^3
(%o2)  f2(x) := x^2

(%i3)  load(draw)$
       wxdraw2d(filled_func=f1(x),fill_color=cyan,
       explicit(f2(x),x,0,1),grid=true);
```




```
(%t4)

(%o4)

(%i5)  /* El área la calculamos integrando respecto de y la función unidad
       para un x determinado desde y=f1(x) a y=f2(x)
       y después evaluando la integral respecto de x, desde x=0 hasta x=1 */
       s:integrate(integrate(1,y,f1(x),f2(x)),x,0,1);
(%o5)  1/12
```

```
(%i6) /* Podemos evaluar el área de un cuarto de círculo de radio R en
      coordenadas Cartesianas, integraremos desde x cero a R el resultado d
      una primera integral para cada x que va desde y=0 hasta y=sqrt(R^2-x^

      ymax(x):=sqrt(R^2-x^2);
```

$$(\%o6)\quad \mathrm{ymax}(x) := \sqrt{R^2 - x^2}$$

```
(%i7) integrate(integrate(1,y,0,ymax(x)),x,0,R);
Is R positive, negative or zero?p;
```

$$(\%o7)\quad \frac{\pi\, R^2}{4}$$

```
(%i8) /* Podemos evitar que nos pregunte si R>0 */
      assume (R > 0)$
      integrate(integrate(1,y,0,ymax(x)),x,0,R);
```

$$(\%o9)\quad \frac{\pi\, R^2}{4}$$

```
(%i10) /* Podemos evaluarla en polares */
       assume (R > 0)$
       integrate(integrate(r,r,0,R),phi,0,%pi/2);
```

$$(\%o11)\quad \frac{\pi\, R^2}{4}$$