Primeros pasos en Maxima

Mario Rodríguez Riotorto

www.biomates.net

25 de marzo de 2006

Índice

1. Introducción **4**

2. Instalación **6**

3. Primera sesión con Maxima **11**

4. Operaciones aritméticas **16**

5. Números complejos **19**

6. Manipulaciones algebraicas **22**

7. Ecuaciones **27**

8. Matrices **32**

9. Funciones matemáticas **41**

10.Límites **45**

11.Derivadas **46**

12.Integrales **50**

13.Ecuaciones diferenciales **53**

14.Números aleatorios **57**

15.Gráficos **60**

16.Estadística descriptiva **71**

17.Interpolación **78**

18.Listas **83**

19.Manipulación de cadenas **87**

20.Operaciones con conjuntos **91**

21.Operaciones lógicas **93**

22.Programación en Maxima **95**

1. Introducción

Este es un manual introductorio al entorno de cálculo simbólico Maxima, directo sucesor del legendario MACSYMA.

El objetivo del manual es facilitar el acceso a este programa a todas aquellas personas que por vez primera se interesan por él.

Maxima es un programa cuyo objeto es la realización de cálculos matemáticos, tanto numéricos como simbólicos, capaz de manipular expresiones algebraicas, derivar e integrar funciones y realizar diversos tipos de gráficos.

Los orígenes de Maxima hay que buscarlos a partir del año 1967 en el MIT AI Lab (Laboratorio de Inteligencia Artificial del Instituto Tecnológico de Massachussets) como una parte del proyecto MAC (Machine Aided Cognition). El programa recibiría por aquel entonces el nombre de Macsyma (MAC's SYmbolic MAnipulator), del cual el MIT mandaría una copia en 1982 al DOE (US Department Of Energy), uno de los organismos que aportaron los fondos económicos para el desarrollo del proyecto; esta primera versión se la conoce como DOE-Macsyma. Posteriormente, el DOE concede la licencia de explotación del programa a la empresa Symbolics, que sigue desarrollando el proyecto durante unos años. En 1992 el programa es adquirido por una empresa que se llamaría precisamente Macsyma Inc, y el programa iría perdiendo fuelle progresivamente ante la presencia en el mercado de otros programas similares como Maple o Mathematica, ambos los dos inspirados en sus orígenes por el propio Macsyma.

Pero ocurrieron dos historias paralelas. Desde el año 1982, y hasta su fallecimiento en el 2001, William Schelter en la Universidad de Texas mantuvo una versión de este programa adaptada al estándar Common Lisp, la cual ya se conocía con el nombre de Maxima para diferenciarla de la versión comercial. En el año 1998 Schelter consiguió del DOE permiso para distribuir Maxima bajo la licencia GNU-GPL (www.gnu.org); con este paso, muchas más personas empezaron a dirigir su mirada hacia Maxima, justo en el momento en el que la versión comercial estaba prácticamente muerta. Actualmente, el proyecto está siendo liderado por un grupo de desarrolladores originarios de varios países, asistidos y ayudados por otras muchas personas interesadas en Maxima y que mantienen un cauce de comunicación a través de una lista de correo (maxima.sourceforge.net/maximalist.html).

Puesto que Maxima se distribuye bajo la licencia GNU-GPL, tanto el código fuente como los manuales son de libre acceso a través de la página web del proyecto (maxima.sourceforge.net).

Como profesor de matemáticas, no me resisto a traducir unas líneas del

capítulo introductorio del manual oficial del programa:

> Aquellos que se presten a utilizar el ordenador para hacer matemáticas, particularmente los estudiantes, deben estar sobre aviso de que estos entornos no son sustitutos del trabajo manual con las ecuaciones ni del esfuerzo por comprender los conceptos. Estos medios no ayudan a formar la intuición matemática ni a reforzar los conocimientos fundamentales... No se debe utilizar el ordenador como un sustituto de la formación básica.
>
> Sin embargo, el dominio del ordenador y de las herramientas matemáticas computacionales son cruciales a la hora de abordar el gran número de problemas que no pueden ser resueltos simplemente con lápiz y papel. En muchos casos, problemas que tardarían años en ser resueltos de forma manual pueden ser solventados en cuestión de segundos con un ordenador... Además, en caso de error, su corrección será más rápida y sencilla volviendo a ejecutar un código ya escrito, pero convenientemente modificado para subsanar el fallo.
>
> Si bien el ordenador puede corregir los errores humanos, el humano por su parte tampoco debe confiar en el ordenador de forma incuestionable. Todos estos sistemas tienen sus límites y cuando éstos son alcanzados, es posible obtener respuestas incorrectas... Así pues, no se puede uno olvidar de revisar los resultados que se obtienen. El ordenador no siempre dice la verdad, y si la dice, quizás no sea completa.
>
> *Ferrol-A Coruña*

2. Instalación

Es posible tener operativo Maxima tanto en Linux como en Windows. La información para la instalación en estos dos sistemas se puede encontrar en la página *web* del proyecto.

En lo que a Linux se refiere, el paquete básico permite tener operativo Maxima en el entorno de la consola de texto, Figura 1, pero también es posible la instalación de módulos o programas adicionales que permitan la utilización del programa a través de un entorno gráfico; aquí se pueden barajar varias opciones: xmaxima, Figura 2, basado en Tcl-Tk y descargable desde la propia página del proyecto Maxima; wxmaxima (wxmaxima.sourceforge.net), Figura 3, basado en wxWidgets y de aparición más reciente; también existe la posibilidad de acceder a Maxima desde una sesión del editor de texto WYSIWYG TeXmacs (www.texmacs.org), Figura 4, que al hacer uso de las fuentes TEX(www.tug.org/teTeX) aumenta considerablemente la calidad en la impresión de fórmulas y expresiones matemáticas. Un programa adicional que no debe faltar es gnuplot (www.gnuplot.info) para la representación de gráficos en 2D y 3D. Todos estos programas se pueden descargar gratuitamente desde Internet. Para la instalación de estos programas léase la documentación correspondiente a cada caso.

También, y para los más valientes, se puede descargar el código fuente y compilarlo, para lo que será necesario tener operativo un entorno Lisp en la máquina, como GCL (www.gnu.org/software/gcl), CLISP (clisp.cons.org) o CMUCL (www.cons.org/cmucl).

En cuanto a Windows, también desde la página del proyecto se puede descargar un ejecutable que instala xmaxima, con un aspecto similar al de la Figura 2. Si se prefiere wxmaxima, una vez instalado el paquete anterior, se hará uso del ejecutable para este sistema operativo que se encontrará en el sitio wxmaxima.sourceforge.net.

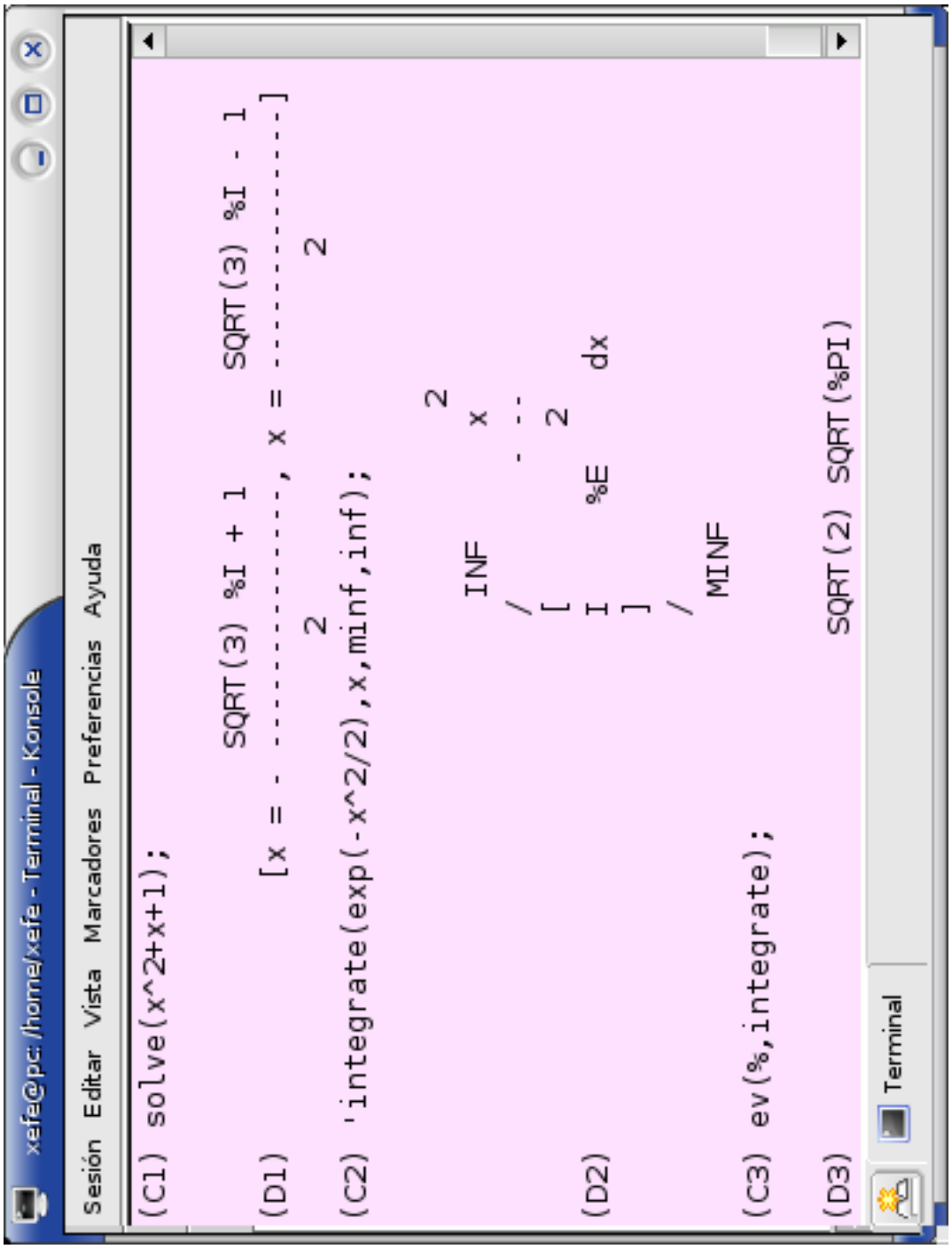


Figura 1: Maxima operando en la consola de texto.

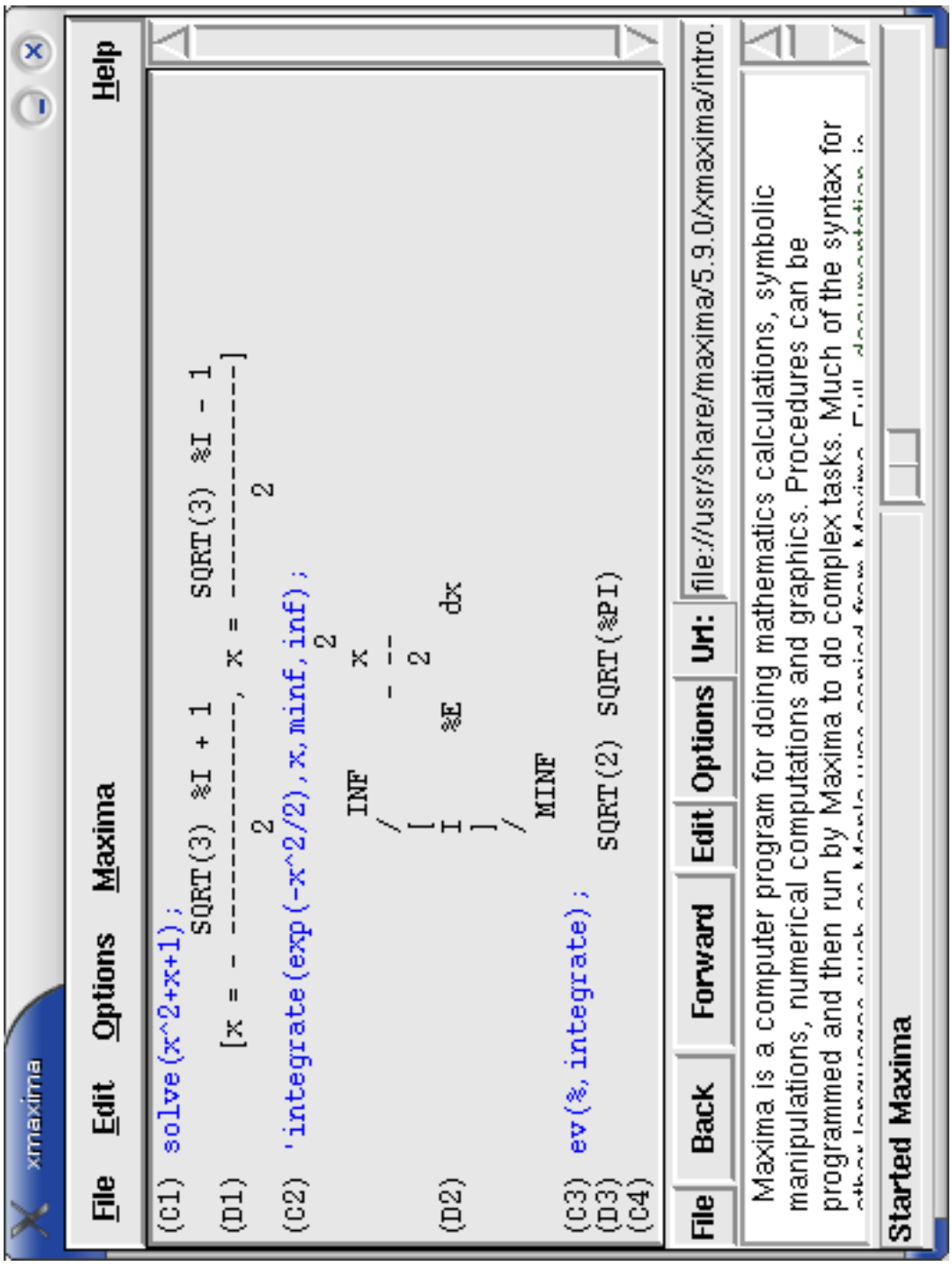


Figura 2: xmaxima: Maxima trabajando en el entorno gráfico Tcl-Tk.

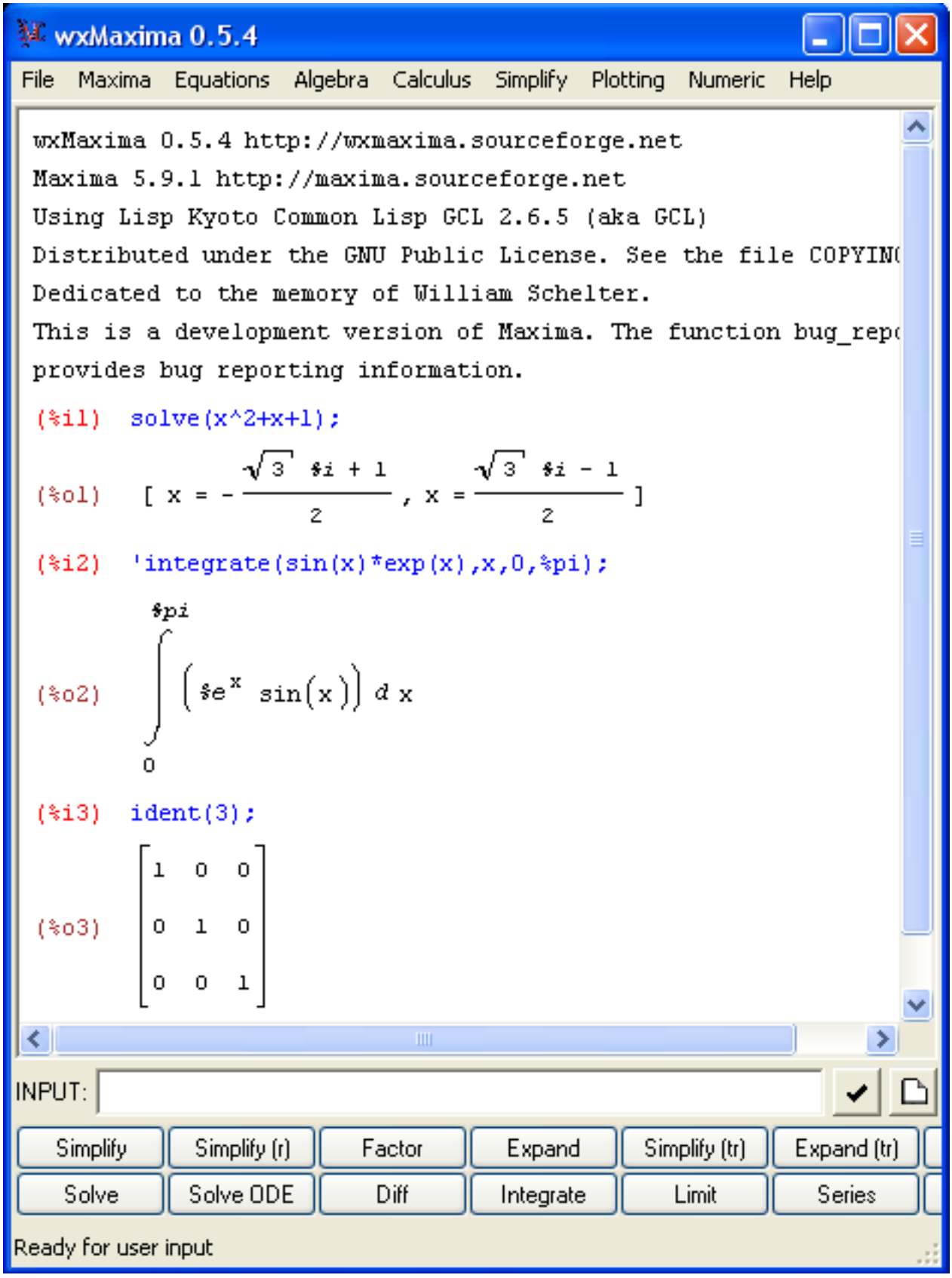


Figura 3: wxmaxima: Maxima ejecutándose en Windows.

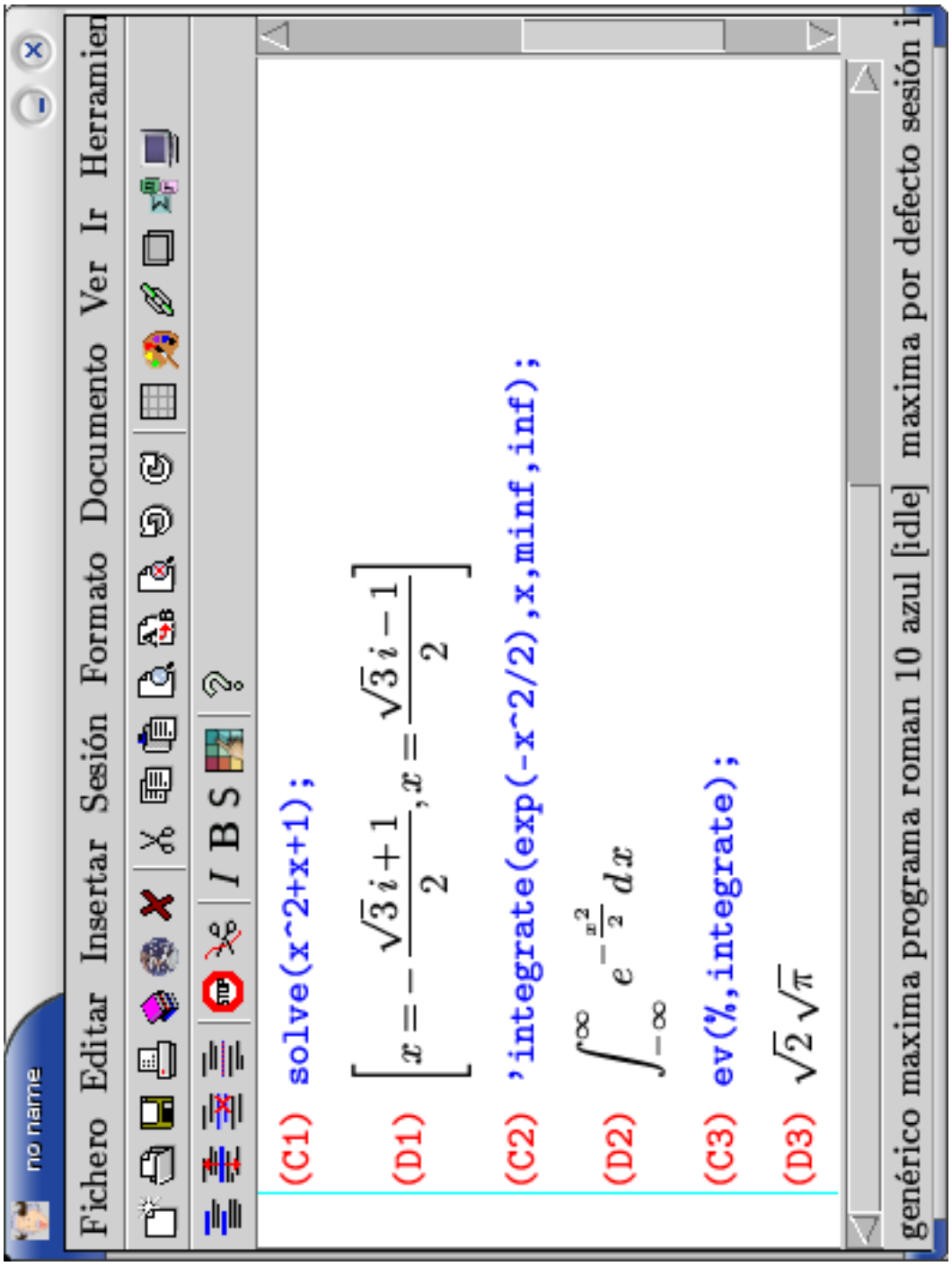


Figura 4: Maxima trabajando en TeXmacs.

3. Primera sesión con Maxima

Una vez accedemos a Maxima, lo primero que vamos a ver será la siguiente información:

```
Maxima 5.9.2 http://maxima.sourceforge.net
Using Lisp CLISP 2.33.2 (2004-06-02)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

Tras la información sobre la licencia, por supuesto GNU-GPL, nos informa sobre la versión con la que estamos trabajando y la página *web* del proyecto. A continuación, muy importante, aparece el indicador `(%i1)` esperando nuestra primera pregunta. Si queremos calcular una simple suma tecleamos la operación deseada seguida de un punto y coma (`;`) y una pulsación de la tecla retorno

```
(%i1) 45 + 23;
```

a lo que Maxima nos contestará

```
(%o1)                          68
(%i2)
```

indicando `(%i2)` que Maxima espera nuestra segunda instrucción.

El punto y coma actúa también como un separador cuando escribimos varias instrucciones en un mismos renglón. Nuestra siguiente operación consistirá en asignar el valor 34578 a la variable `x` y el 984003 a la `y`, solicitando luego su producto,

```
(%i2) x:34578; y:984003; x*y;
(%o2)                         34578
(%o3)                         984003
(%o4)                      34024855734
(%i5)
```

conviene prestar atención al hecho de que la asignación de un valor a una variable se hace con los dos puntos, no con el signo de igualdad, que se reserva para las ecuaciones.

Es posible que no deseemos los resultados intermedios que Maxima va calculando o, como en este caso, las asignaciones a las variables que va haciendo; en tales situaciones conviene hacer uso del delimitador `$`, que no devuelve a la consola los resultados que va calculando. Repitiendo el cálculo de la forma

```
(%i5) x:34578$ y:984003$ x*y;
(%o7)                  34024855734
```

podemos conseguir una salida más limpia. Las asignaciones a variables se mantienen activas mientras dure la sesión con Maxima, por lo que podemos restar las variables `x` e `y` anteriores

```
(%i8) x-y;
(%o8)                     - 949425
```

Esto tiene un riesgo; si queremos resolver la ecuación $x^2 - 3x + 1 = 0$,

```
(%i9) solve(x^2-3*x+1=0,x);
A number was found where a variable was expected -‘solve’
 -- an error.  Quitting.  To debug this try debugmode(true);
```

nos devuelve un mensaje de error, ya que donde se supone que hay una incógnita, lo que realmente encuentra es un número, en este caso 34578. El problema se resuelve vaciando el contenido de la variable `x` mediante la función `kill`,

```
(%i10) kill(x)$
(%i11) solve(x^2-3*x+1=0,x);
                   sqrt(5) - 3       sqrt(5) + 3
(%o11)      [x = - -----------, x = -----------]
                         2                 2
```

luego las soluciones de la ecuación forman el conjunto $\{-\frac{\sqrt{5}-3}{2}, \frac{\sqrt{5}+3}{2}\}$. Ya vemos que en Maxima la raíz cuadrada se calcula con la función `sqrt`.

Las etiquetas `(%ix)` y `(%ox)`, siendo `x` un número entero, pueden ser utilizadas a lo largo de una sesión a fin de evitar tener que volver a escribir expresiones; así, si queremos calcular el cociente $\frac{xy}{/}x - y$, con $x = 34578$ y $y = 984003$, podemos aprovechar los resultados `(%o7)` y `(%o8)` y hacer

```
(%i12) %o7/%o8;
                          11341618578
(%o12)                  - -----------
                             316475
```

Las etiquetas que indican las entradas y salidas pueden ser modificadas a voluntad haciendo uso de las variables `inchar` y `outchar`,

```
(%i13) inchar;
(%o13)                          %i
(%i14) outchar;
(%o14)                          %o
(%i15) inchar: "menda";
(%o15)                         menda
(menda15) outchar: "lerenda";
(lerenda15)                   lerenda
(menda16) 2+6;
(lerenda16)                      8
(menda17) inchar:"%i"$ outchar:"%o"$
(%i17)
```

En caso de que se pretenda realizar nuevamente un cálculo ya indicado deberá escribirse dos comillas simples (''), no comilla doble, junto con la etiqueta de la instrucción deseada

```
(%i18) ''%i9;
(%o18)                      x - 984003
```

Obsérvese que al haber dejado a `x` sin valor numérico, ahora el resultado es otro. Cuando se quiera hacer referencia al último resultado calculado por Maxima puede ser más sencillo hacer uso del símbolo `%`, de forma que si queremos restarle la `x` a la última expresión podemos hacer

```
(%i19) %-x;
(%o19)                       - 984003
```

Ciertas constantes matemáticas de uso frecuente tienen un símbolo especial en Maxima: la base de los logaritmos naturales (e), el cociente entre la longitud de una circunferencia y su diámetro (π), la unidad imaginaria ($i = \sqrt{-1}$) y la constante de Euler-Mascheroni ($\gamma = -\int_0^\infty e^{-x} \ln x dx$), se representarán por `%e`, `%pi`, `%i` y `%gamma`, respectivamente.

Es muy sencillo solicitar a Maxima que nos informe sobre alguna función de su lenguaje; la técnica nos servirá para ampliar información sobre cualquier instrucción a la que se haga referencia en este manual, por ejemplo,

```
(%i20) describe(sqrt);
```

```
 0: isqrt :(maxima.info)Definitions for Operators.
 1: sqrt :Definitions for Operators.
 2: sqrtdispflag :Definitions for Operators.
Enter space-separated numbers, ‘all’ or ‘none’: 1

Info from file /usr/local/info/maxima.info:
 - Function: sqrt (<x>)
     The square root of <x>. It is represented internally by
     ‘<x>^(1/2)’.  See also ‘rootscontract’.

     ‘radexpand’ if ‘true’ will cause nth roots of factors of a product
     which are powers of n to be pulled outside of the radical, e.g.
     ‘sqrt(16*x^2)’ will become ‘4*x’ only if ‘radexpand’ is ‘true’.

(%o20)                         false
```

Inicialmente muestra un sencillo menú en el que debemos seleccionar qué información concreta deseamos; en esta ocasión se optó por `1` que es el número asociado a la función que interesaba.

A fin de no perder los resultados de una sesión con Maxima, quizás con el objeto de continuar el trabajo en próximas jornadas, podemos guardar en un archivo aquellos valores que nos puedan interesar; supongamos que necesitamos almacenar el contenido de la variable `y`, tal como lo definimos en la entrada `(%i5)`, así como el resultado de la salida `(%o11)`, junto con la instrucción que la generó, la entrada `(%i11)`,

```
(%i21) save("mi.sesion",y,resultado=%o11,ecuacion=%i11)$
```

Compruébese que el primer argumento de `save` es el nombre del archivo, junto con su ruta si se considera necesario, que la variable `y` se escribe tal cual, pero que a las referencias de las entradas y salidas deben ir acompañadas de un nombre que las haga posteriormente reconocibles.

Por último, la forma correcta de abandonar la sesión de Maxima es mediante

```
(%i22) quit();
```

Abramos ahora una nueva sesión con Maxima y carguemos en memoria los resultados de la anterior sesión,

```
(%i1) load("mi.sesion")$
```

```
(%i2) y;
(%o2)                        984003
(%i3) ecuacion;
                          2
(%o3)              solve(x  - 3 x + 1 = 0, x)
(%i4) resultado;
                    sqrt(5) - 3      sqrt(5) + 3
(%o4)         [x = - -----------, x = -----------]
                          2                2
```

Si ahora quisiéramos recalcular las soluciones de la ecuación, modificándola previamente de manera que el coeficiente de primer grado se sustituyese por otro número, simplemente haríamos

```
(%i5) subst(5,3,ecuacion);
                          2
(%o5)              solve(x  - 5 x + 1 = 0, x)
(%i6) ev(%);
                    sqrt(21) - 5      sqrt(21) + 5
(%o6)        [x = - ------------, x = ------------]
                          2                2
```

4. Operaciones aritméticas

Los operadores aritméticos más comunes son:

`+`	suma
`-`	resta
`*`	producto
`/`	división
`^` o `**`	potencia

Maxima devuelve los resultados de forma exacta, sin aproximaciones decimales; así tenemos que para realizar el cálculo de la expresión

$$\left[\left(\frac{3^7}{4+\frac{3}{5}}\right)^{-1}+\frac{7}{9}\right]^3$$

haremos

```
(%i1) ((3^7/(4+3/5))^(-1)+7/9)^3;
                        620214013952
(%o1)                   -------------
                        1307544150375
```

obteniendo el resultado en forma de fracción simplificada.

De todos modos podemos solicitar el resultado en forma decimal; por ejemplo, si queremos la expresión decimal del último resultado,

```
(%i2) %,numer;
(%o2)                .4743350454163436
```

Maxima puede trabajar con precisión arbitraria. Para calcular el valor del cociente $\frac{e}{\pi}$ con 100 cifras decimales, deberemos especificar primero la precisión requerida asignándole a la variable `fpprec` el valor 100 y a continuación realizar el cálculo, solicitando la expresión decimal con una llamada a la función `bfloat`:

```
(%i3) fpprec:100$  bfloat(%e/%pi);
(%o4) 8.652559794322650872177747896460896174287446239085 15#
53945433028894804504457067705863192466251618451 73B-1
```

Nótese que cuando un resultado es devuelto en el formato `bfloat` se escribe la letra `B` en lugar de la clásica `E` para indicar el exponente.

En la instrucción `%i3` se establece la precisión deseada y luego se transforma el valor simbólico `%e/%pi` mediante la función `bfloat`. Recuérdese que el símbolo `$` sirve como delimitador entre instrucciones.

La factorización de un número entero en factores primos es un problema difícil de resolver para cantidades grandes; de hecho, algunos algoritmos de encriptación se basan en esta dificultad.

```
(%i5) factor(1303948800000);
                       12  3  5  3
(%o5)                 2   3  5  7  11
```

En la siguiente factorización hacemos uso de la variable global `showtime` para saber el tiempo que le lleva ejecutar el cálculo,

```
(%i6) showtime:true$    /* activamos el contador de tiempo */
Evaluation took 0.00 seconds (0.00 elapsed) using 80 bytes.
(%i7) factor(2^300-1);
Evaluation took 349.35 seconds (352.10 elapsed) using 21866187.852 KB.
        2  3
(%o7) 3  5  7 11 13 31 41 61 101 151 251 331 601 1201 1321
 1801 4051 8101 63901 100801 268501 10567201 13334701
 1182468601 1133836730401
(%i8) showtime:false$  /* desactivamos contador */
```

El texto que se escribe entre `/*` y `*/` son comentarios que Maxima ignora.

En relación con los números primos, para saber si un número lo es o no,

```
(%i9) primep(3^56-1);
(%o9)                       false
```

Y para solicitarle a Maxima que compruebe si un número es par o impar necesitamos echar mano de las funciones `evenp` or `oddp`, respectivamente,

```
(%i10) evenp(42);
(%o10)                       true
(%i11) oddp(31);
(%o11)                       true
```

Le solicitamos a Maxima que nos haga una lista con todos los divisores de un número,

```
(%i12) divisors(100);
(%o12)       {1, 2, 4, 5, 10, 20, 25, 50, 100}
```

En lo que concierne a la división, puede ser necesario conocer el cociente entero y el resto correspondiente; para ello se dispone de las funciones `quotient` y `remainder`,

```
(%i13) quotient(5689,214);
(%o13)                        26
(%i14) remainder(5689,214);
(%o14)                       125
```

5. Números complejos

Como ya se comentó más arriba, la unidad imaginaria $\sqrt{-1}$ se representa en Maxima mediante el símbolo `%i`,

```
(%i1) %i^2;
(%o1)                         - 1
```

Se soportan las operaciones básicas con números complejos,

```
(%i2) z1:3+5*%i$ z2:1/2-4*%i$  /*z1 y z2*/
(%i4) z1 + z2;     /*suma*/
                                 7
(%o4)                       %i + -
                                 2
(%i5) z1 - z2;     /*resta*/
                                  5
(%o5)                       9 %i + -
                                  2
(%i6) z1 * z2;     /*multiplicacion*/
                    1
(%o6)               (- - 4 %i) (5 %i + 3)
                    2
(%i7) z1 / z2;     /* division */
                           5 %i + 3
(%o7)                      --------
                           1
                           - - 4 %i
                           2
```

Es posible que estos dos últimos resultados nos parezcan frustrantes y los deseemos en otro formato, resultado de las operaciones indicadas; a tal efecto podemos pedir a Maxima que nos devuelva `(%o6)` y `(%o7)` en forma cartesiana,

```
(%i8) rectform(%o6);
                         43   19 %i
(%o8)                    -- - -----
                         2      2
(%i9) rectform(%o7);
```

```
                          58 %i   74
(%o9)                     ----- - --
                           65     65
```

Las funciones `realpart` y `imagpart` extraen del número complejo sus partes real e imaginaria, respectivamente. Podemos utilizarlas para comprobar que el resultado obtenido en `(%o9)` es el correcto,

```
(%i10) realpart(%o7);
                                74
(%o10)                        - --
                                65
(%i11) imagpart(%o7);
                               58
(%o11)                         --
                               65
```

Antes hemos optado por los resultados en formato cartesiano; pero también es posible solicitarlos en su forma polar,

```
(%i12) polarform(%o7);
                          %i (%pi - atan(29/37))
            2 sqrt(34) %e
(%o12)      ----------------------------------
                          sqrt(65)
(%i13) %,numer;   /*radio y argumento reales*/
                               2.4768181587724474 %i
(%o13)   1.4464811413591583 %e
```

La norma de un número complejo se calcula con la función `abs` y admite el argumento tanto en forma cartesiana como polar,

```
(%i14) abs(%o9);
                          2 sqrt(34)
(%o14)                    ----------
                           sqrt(65)
(%i15) abs(%o12);
                          2 sqrt(34)
(%o15)                    ----------
                           sqrt(65)
```

Por último, el conjugado de un número complejo se calcula con la función `conjugate`, pero su uso requiere la carga previa del archivo `eigen`. Como se ve en este ejemplo, el resultado dependerá del formato del argumento, cartesiano o polar,

```
(%i16) load(eigen)$
(%i17) conjugate(%o9);       /*forma cartesiana*/
                           58 %i   74
(%o17)                   - ----- - --
                            65     65
(%i18) conjugate(%o12);      /*forma polar*/
                             %i atan(29/37)
                 2 sqrt(34) %e
(%o18)         - ---------------------------
                           sqrt(65)
```

6. Manipulaciones algebraicas

Sin duda una de las capacidades más destacables de Maxima es su habilidad para manipular expresiones algebraicas. Desarrollemos un ejemplo que empieza por asignar a la variable `q` una expresión literal:

```
(%i1) q: (x+3)^5-(x-a)^3+(x+b)^(-1)+(x-1/4)^(-5);
             1            3          5      1
(%o1)      ----- - (x - a)  + (x + 3)  + --------
           x + b                                1 5
                                          (x - -)
                                               4
```

Se observa que en principio Maxima no realiza ningún cálculo. La función `expand` se encarga de desarrollar las potencias,

```
(%i2) expand(q);
                        1                          1      5
(%o2)  ------------------------------------ + ----- + x
                4      3      2                 x + b
        5   5 x    5 x    5 x    5 x    1
       x  - ---- + ---- - ---- + --- - ----
             4      8      32     256   1024
       4       3        2        2      2              3
 + 15 x  + 89 x  + 3 a x  + 270 x  - 3 a  x + 405 x + a
 + 243
```

No obstante es posible que no nos interese desplegar toda la expresión, entre otras cosas para evitar una respuesta farragosa y difícil de interpretar; en tal caso podemos utilizar `expand` añadiéndole dos argumentos y operar de la manera siguiente

```
(%i3) q,expand(3,2);
          1            5    3        2      2        1
(%o3)   ----- + (x + 3)  - x  + 3 a x  - 3 a  x + --------
        x + b                                          1 5
                                                 (x - -)
                                                      4
                                                          3
                                                       + a
```

Con el primer argumento indicamos que queremos la expansión de todas aquellas potencias con exponente positivo menor o igual a 3 y de las que teniendo el exponente negativo no excedan en valor absoluto de 2.

Dada una expresión con valores literales, podemos desear sustituir alguna letra por otra expresión; por ejemplo, si queremos hacer los cambios $a = 2$, $b = 2c$ en el último resultado obtenido,

```
(%i4) %,a=2,b=2*c;
          1             5    3      2             1
(%o4)   ------- + (x + 3)  - x  + 6 x  - 12 x + -------- + 8
        x + 2 c                                       1 5
                                                (x - -)
                                                      4
```

En estos dos últimos ejemplos (`%i3` y `%i4`) se presentaron sentencias en las que había elementos separados por comas (`,`). Esta es una forma simplificada de utilizar la función `ev`, que evalúa la primera expresión asignando los valores que se le van indicando a continuación; por ejemplo, `(%i3)` se podía haber escrito de la forma `ev(q,expand(3,2))` y `(%i4)` como `ev(%,a=2,b=2*c)`. El uso de la variante con `ev` está más indicado para ser utilizado dentro de expresiones más amplias. Obsérvese el resultado siguiente

```
(%i5) 3*x^2 + ev(x^4,x=5);
                            2
(%o5)                     3 x  + 625
```

donde la sustitución $x = 5$ se ha realizado exclusivamente dentro del entorno delimitado por la función `ev`.

De forma más general, la función `subst` sustituye subexpresiones enteras. En el siguiente ejemplo, introducimos una expresión algebraica y a continuación sustituimos todos los binomios `x+y` por la letra `k`,

```
(%i6) 1/(x+y)-(y+x)/z+(x+y)^2;
                    y + x           2     1
(%o6)             - ----- + (y + x)  + -----
                      z                y + x
(%i7) subst(k,x+y,%);
                          k    2   1
(%o7)                   - - + k  + -
                          z        k
```

No obstante, el siguiente resultado nos sugiere que debemos ser precavidos con el uso de esta función, ya que Maxima no siempre interpretará como subexpresión aquella que para nosotros sí lo es:

```
(%i8) subst(sqrt(k),x+y,(x+y)^2+(x+y));
(%o8)                    y + x + k
```

Como una aplicación práctica de `subst`, veamos cómo podemos utilizarla para obtener el conjugado de un número complejo,

```
(%i9) subst(-%i,%i,a+b*%i);
(%o9)                     a - %i b
```

La operación inversa de la expansión es la factorización. Expandamos y factoricemos sucesivamente un polinomio para comprobar los resultados,

```
(%i10) expand((a-2)*(b+1)^2*(a+b)^5);
           7      7      2  6        6      6       3  5
(%o10)  a b  - 2 b  + 5 a  b  - 8 a b  - 4 b  + 10 a  b
       2  5         5      5       4  4       2  4         4
 - 10 a  b  - 19 a b  - 2 b  + 10 a  b  - 35 a  b  - 10 a b
      5  3       4  3       3  3       2  3    6  2
 + 5 a  b  + 10 a  b  - 30 a  b  - 20 a  b  + a  b
      5  2       4  2       3  2      6      5         4
 + 8 a  b  - 10 a  b  - 20 a  b  + 2 a  b + a  b - 10 a  b
    6      5
 + a  - 2 a
(%i11) factor(%);
                                 2        5
(%o11)           (a - 2) (b + 1)  (b + a)
```

La función `ratsimp` simplifica cualquier expresión racional, así como las subexpresiones racionales que son argumentos de funciones cualesquiera. El resultado se devuelve como el cociente de dos polinomios. En ocasiones no es suficiente con una sola ejecución de `ratsimp`, por lo que será necesario aplicarla más veces, esto es lo que hace precisamente la función `fullratsimp`; concretemos esto con un ejemplo:

```
(%i12) (x^(a/2)-1)^2*(x^(a/2)+1)^2 / (x^a-1);
```

```
                  a/2     2   a/2     2
                 (x    - 1)  (x    + 1)
(%o12)           ----------------------
                           a
                          x  - 1
(%i13) ratsimp(%);  /* simplificamos una vez */
                        2 a      a
                       x    - 2 x  + 1
(%o13)                 ---------------
                            a
                           x  - 1
(%i14) ratsimp(%);  /* simplificamos otra vez */
                            a
(%o14)                     x  - 1
(%i15) fullratsimp(%o12);   /* simplificamos todo de una vez! */
                            a
(%o15)                     x  - 1
```

Dada una fracción algebraica, podemos obtener separadamente el numerador y el denominador,

```
(%i16) fr: (x^3-4*x^2+4*x-2)/(x^2+x+1);
                     3      2
                    x  - 4 x  + 4 x - 2
(%o16)              -------------------
                         2
                        x  + x + 1
(%i17) num(fr);
                     3      2
(%o17)              x  - 4 x  + 4 x - 2
(%i18) denom(fr);
                           2
(%o18)                    x  + x + 1
```

El máximo común divisor de un conjunto de polinomios se calcula con la función `gcd` y el mínimo común múltiplo con `lcm`

```
(%i19) p1: x^7-4*x^6-7*x^5+8*x^4+11*x^3-4*x^2-5*x;
         7      6      5      4       3      2
(%o19)  x  - 4 x  - 7 x  + 8 x  + 11 x  - 4 x  - 5 x
```

```
(%i20) p2: x^4-2*x^3-4*x^2+2*x+3;
                  4      3      2
(%o20)           x  - 2 x  - 4 x  + 2 x + 3
(%i21) gcd(p1,p2);
                       3    2
(%o21)                 x  + x  - x - 1
(%i22) load(functs)$
(%i23) lcm(p1,p2);
                                    2          3
(%o23)      (x - 5) (x - 3) (x - 1)  x (x + 1)
```

En `(%i19)` y `(%i20)` definimos los polinomios `p1` y `p2`, a continuación calculamos su máximo común divisor (mcd) en `(%i21)` y antes de pedir el mínimo común múltiplo en `(%i23)` cargamos el archivo `functs.mac` en el que se encuentra definida la función `lcm`. Es posible que deseemos disponer del mcd factorizado, por lo que hacemos

```
(%i24) factor(%o21);
                                     2
(%o24)                (x - 1) (x + 1)
```

7. Ecuaciones

Asignarle una etiqueta a una ecuación es tan simple como hacer

```
(%i1) ec: 3 * x = 1 + x;
(%o1)                  3 x = x + 1
```

A partir de ahí, aquellas operaciones en las que intervenga la etiqueta serán realizadas a ambos miembros de la igualdad; restamos `x` en los dos lados y a continuación dividimos lo que queda entre `2`,

```
(%i2) %-x;
(%o2)                      2 x = 1
(%i3) %/2;
                                1
(%o3)                       x = -
                                2
```

obteniendo de esta manera la solución de la ecuación como si hubiésemos operado manualmente.

Ni qué decir tiene que la ecuación anterior se pudo haber resuelto de un modo más inmediato,

```
(%i4) solve(ec);
                                1
(%o4)                      [x = -]
                                2
```

La instrucción `solve` puede admitir como segundo argumento la incógnita que se pretende calcular, lo que resultará de utilidad cuando en la ecuación aparezcan constantes literales,

```
(%i5) solve((2-a)/x-3=b*x+1/x,x);
              sqrt((4 - 4 a) b + 9) + 3
(%o5)  [x = - -------------------------,
                         2 b
                                  sqrt((4 - 4 a) b + 9) - 3
                             x = -------------------------]
                                           2 b
```

Las soluciones de las ecuaciones serán probablemente utilizadas en cálculos posteriores, por lo que nos interesará poder extraerlas de la lista anterior; a continuación tomamos el primer resultado calculado por Maxima mediante la función `part` y después asignamos a la variable `sol` el resultado numérico,

```
(%i6) part(%,1);
                     sqrt((4 - 4 a) b + 9) + 3
(%o6)          x = - -------------------------
                                2 b
(%i7) sol: rhs(%);
                    sqrt((4 - 4 a) b + 9) + 3
(%o7)             - -------------------------
                                2 b
```

La función `rhs` devuelve el miembro derecho de la igualdad, mientras que `lhs` haría lo propio con el miembro izquierdo.

Es posible resolver ecuaciones polinómicas de grado ≤ 4, pero desgraciadamente, como es de esperar, Maxima no dispone de un método algebraico que permita resolver ecuaciones polinómicas de grado mayor que cuatro,

```
(%i8) solve(x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0);
                  5      4      3      2
(%o8)       [0 = x  - 3 x  + 2 x  - 2 x  - x + 4]
```

por lo que `solve` devolverá la misma ecuación sin resolver.

Maxima dispone de otra función para resolver ecuaciones y sistemas, que en ocasiones será más util que `solve`. Veamos cómo trata la instrucción `algsys` la ecuación polinómica anterior,

```
(%i9) algsys([x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0],[x]);
(%o9) [[x = 2.478283086356668],
[x = .1150057557117294 - 1.27155810694299 %i],
[x = 1.27155810694299 %i + .1150057557117294],
[x = - .8598396689940523], [x = 1.151545166402536]]
```

Como se ve, al no ser capaz de resolverla algebraicamente, nos brinda la oportunidad de conocer una aproximación numérica de la solución. La función `algsys` reconoce la variable global `realonly`, que cuando toma el valor `true`, hará que se ignoren las soluciones complejas,

```
(%i10) realonly:true$
(%i11) ''%i9;  /* recalcula la entrada %i9 */
(%o12) [[x = 2.478283086356668], [x = 1.151545166402536],
                                  [x = - .8598396689940523]]
(%i13) realonly:false$  /* le devolvemos el valor por defecto */
```

Tratándose de ecuaciones polinómicas, para forzar el cálculo numérico de sus raíces se puede hacer uso también de la instrucción `allroots`, que obtiene tanto las raíces reales como complejas,

```
(%i14) allroots(x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0);
(%o14) [x = 1.151545173424091, x = - .8598397137271315,
x = 1.27155810694299 %i + .1150057557117294,
x = .1150057557117294 - 1.27155810694299 %i,
x = 2.478283028879582]
```

donde se recordará que el símbolo `%i` representa la unidad imaginaria $i = \sqrt{-1}$.

Vemos a continuación cómo resolver un sistema lineal de ecuaciones mediante `algsys`. Sean las ecuaciones

$$\begin{cases} 2x - 4y + 2z = -2 \\ \frac{1}{3}x + 2y + 9z = x + y \\ -4x + \sqrt{2}y + z = 3y \end{cases}$$

se hará

```
(%i15) algsys([ 2 * x - 4 * y + 2 * z = -2,
               1/3* x + 2 * y + 9 * z = x + y,
               -4 * x + sqrt(2) * y + z = 3 * y],
         [x,y,z]);
               27 sqrt(2) - 84                106
(%o15) [[x = - ---------------, y = - ----------------,
               29 sqrt(2) - 314        29 sqrt(2) - 314
                                       575 sqrt(2) - 2884
                                 z = - --------------------]]
                                       9106 sqrt(2) - 50139
```

nótese que las ecuaciones se encerrarán entre corchetes y se separarán por comas y que lo mismo se debe hacer con los nombres de las incógnitas.

También la función `algsys` permite la resolución de sistemas de ecuaciones no lineales como

$$\begin{cases} 3x^2 - y = 2y^2 + 4 \\ 5x + 7y = 1 \end{cases}$$

Podemos proceder como se indica a continuación,

```
(%i16) algsys([3*x^2-y=2*y^2+4, 5*x+7*y=1],[x,y]);
```

```
                7 sqrt(1685) + 55
(%o16) [[x = - -----------------,
                       194
    5 sqrt(5) sqrt(337) + 67
y = ------------------------],
               194
     7 sqrt(1685) - 55         5 sqrt(5) sqrt(337) - 67
[x = -----------------, y = - ------------------------]]
            194                          194
```

cuyo resultado es una lista con dos pares ordenados, soluciones ambos del sistema propuesto. Una manera alternativa de proceder en este caso podría consistir en solicitarle primero a Maxima que nos eliminase una de las variables y resolver a continuación para la otra, tal como indica el siguiente código,

```
(%i17) eliminate([3*x^2-y=2*y^2+4, 5*x+7*y=1],[y]);
                            2
(%o17)               [97 x  + 55 x - 205]
(%i18) algsys(%,[x]);
             7 sqrt(1685) - 55           7 sqrt(1685) + 55
(%o18) [[x = -----------------], [x = - -----------------]]
                    194                         194
```

En `(%i17)` pedimos que nos elimine la incógnita y, obteniendo como resultado una ecuación de segundo grado, la cual se resuelve a continuación. Nótese que en la expresión `(%o17)` falta una igualdad, lo que habrá de interpretarse como igualada a cero, siendo equivalente a $97x^2 + 55x - 205 = 0$.

Ya para terminar, resolvamos una ecuación indeterminada obteniendo la solución en forma paramétrica,

```
(%i19) algsys([3*x^2-5*y=x],[x,y]);
                                    2
                                3 %r1  - %r1
(%o19)          [[x = %r1, y = ------------]]
                                    5
(%i20) %, %r1:1;
                                    2
(%o20)                 [[x = 1, y = -]]
                                    5
```

Maxima nombra los parámetros siguiendo el esquema `%Rx`, siendo `x` un número entero positivo. En la entrada `%i20` pedimos que en `%o19` se sustituya el parámetro por la unidad.

8. Matrices

La definición de una matriz es extremadamente simple en Maxima,

```
(%i1) m1: matrix([3,4,0],[6,0,-2],[0,6,a]);
                       [ 3  4   0  ]
                       [           ]
(%o1)                  [ 6  0  - 2 ]
                       [           ]
                       [ 0  6   a  ]
```

También es posible definir una matriz de forma interactiva tal y como muestra el siguiente ejemplo,

```
(%i2) entermatrix(2,3);
Row 1 Column 1:
4/7;
Row 1 Column 2:
0;
Row 1 Column 3:
%pi;
Row 2 Column 1:
sqrt(2);
Row 2 Column 2:
log(3);
Row 2 Column 3:
-9;

Matrix entered.
                   [    4                 ]
                   [    -       0     %pi ]
(%o2)              [    7                 ]
                   [                      ]
                   [ sqrt(2)  log(3)  - 9 ]
```

Existe un tercer método para construir matrices que es útil cuando el elemento (i, j)-ésimo de la misma es función de su posición dentro de la matriz. A continuación, se fija en primer lugar la regla que permite definir un elemento cualquiera y luego en base a ella se construye una matriz de dimensiones 2×5

```
(%i3) a[i,j]:=i+j$
(%i4) genmatrix(a,2,5);
                     [ 2  3  4  5  6 ]
(%o4)                [               ]
                     [ 3  4  5  6  7 ]
```

Obsérvese que el símbolo de asignación para el elemento genérico es `:=`.

Podemos acceder a los diferentes elementos de la matriz haciendo referencia a sus subíndices, indicando primero la fila y después la columna:

```
(%i5) m1[3,1];
(%o5)                          0
```

Se puede extraer una submatriz con la función `submatrix`, teniendo en cuenta que los enteros que preceden al nombre de la matriz original son las filas a eliminar y los que se colocan detrás indican las columnas que no interesan; en el siguiente ejemplo, queremos la submatriz que nos queda de `m1` después de extraer la primera fila y la segunda columna,

```
(%i5) submatrix(1,m1,2);
                           [ 6  - 2 ]
(%o5)                      [        ]
                           [ 0   a  ]
```

Otro ejemplo es el siguiente,

```
(%i6) submatrix(1,2,m1,3);
(%o6)                       [ 0  6 ]
```

en el que eliminamos las dos primeras filas y la última columna, ¿se pilla el truco?

Al igual que se extraen submatrices, es posible añadir filas y columnas a una matriz dada; por ejemplo,

```
(%i7) addrow(m1,[1,1,1],[2,2,2]);
```

```
                        [ 3  4   0  ]
                        [           ]
                        [ 6  0  - 2 ]
                        [           ]
(%o7)                   [ 0  6   a  ]
                        [           ]
                        [ 1  1   1  ]
                        [           ]
                        [ 2  2   2  ]
(%i8) addcol(%,[7,7,7,7,7]);
                       [ 3  4   0   7 ]
                       [              ]
                       [ 6  0  - 2  7 ]
                       [              ]
(%o8)                  [ 0  6   a   7 ]
                       [              ]
                       [ 1  1   1   7 ]
                       [              ]
                       [ 2  2   2   7 ]
```

La matriz identidad es más fácil construirla mediante la función `ident`,

```
(%i9) ident(3);
                        [ 1  0  0 ]
                        [         ]
(%o9)                   [ 0  1  0 ]
                        [         ]
                        [ 0  0  1 ]
```

y la matriz con todos sus elementos iguales a cero,

```
(%i10) zeromatrix(2,4);
                       [ 0  0  0  0 ]
(%o10)                 [            ]
                       [ 0  0  0  0 ]
```

También, una matriz diagonal con todos los elementos de la diagonal principal iguales puede construirse con una llamada a la función `diagmatrix`,

```
(%i11) diagmatrix(4,%e);
```

```
                    [ %e  0   0   0  ]
                    [                ]
                    [ 0   %e  0   0  ]
(%o11)              [                ]
                    [ 0   0   %e  0  ]
                    [                ]
                    [ 0   0   0   %e ]
```

En todo caso, debe tenerse cuidado en que si la matriz no se construye de forma apropiada, Maxima no la reconoce como tal. Para saber si una expresión es reconocida como una matriz se utiliza la función `matrixp`; la siguiente secuencia permite aclarar lo que se pretende decir,

```
(%i12) matrix([[1,2,3],[4,5,6]]);  /* construccion correcta */
(%o12)            [ [1, 2, 3]  [4, 5, 6] ]
(%i13) matrixp(%);  /* es la anterior realmente una matriz? */
(%o13)                        TRUE
(%i14) [[7,8,9],[0,1,2]];    /* otra matriz */
(%o14)             [[7, 8, 9], [0, 1, 2]]
(%i15) matrixp(%);   /* sera una matriz? */
(%o15)                       FALSE
```

Casos particulares de submatrices son las filas y las columnas; los ejemplos se explican por sí solos:

```
(%i16) col(m1,3);
                           [  0  ]
                           [     ]
(%o16)                     [ - 2 ]
                           [     ]
                           [  a  ]
(%i17) row(m1,2);
(%o17)                 [ 6  0  - 2 ]
```

Con las matrices se pueden realizar múltiples operaciones. Empezamos por el cálculo de la potencia de una matriz:

```
(%i18) m1^^2;
```

```
                    [ 33  12     - 8   ]
                    [                  ]
(%o18)              [ 18  12    - 2 a  ]
                    [                  ]
                    [           2      ]
                    [ 36  6 a   a  - 12 ]
```

Nótese que se utiliza dos veces el símbolo ^ antes del exponente; en caso de escribirlo una sola vez se calcularían las potencias de cada uno de los elementos de la matriz independientemente, como se indica en el siguiente ejemplo,

```
(%i19) m2:m1^2;
                       [ 9   16  0  ]
                       [            ]
(%o19)                 [ 36  0   4  ]
                       [            ]
                       [          2 ]
                       [ 0   36  a  ]
```

Para la suma, resta y producto matriciales se utilizan los operadores +, - y ., respectivamente,

```
(%i20) m1+m2;
                      [ 12  20    0    ]
                      [                ]
(%o20)                [ 42  0     2    ]
                      [                ]
                      [          2     ]
                      [ 0   42  a  + a ]
(%i21) m1-m2;
                    [ - 6   - 12    0    ]
                    [                    ]
(%o21)              [ - 30   0     - 6   ]
                    [                    ]
                    [                  2 ]
                    [  0    - 30  a - a  ]
(%i22) m1.m2;
```

```
                    [ 171   48     16    ]
                    [                    ]
                    [                 2  ]
(%o22)              [ 54    24   - 2 a   ]
                    [                    ]
                    [             3      ]
                    [ 216  36 a  a  + 24 ]
```

Sin embargo, tanto el producto elemento a elemento de dos matrices, como la multiplicación por un escalar se realizan mediante el operador *, como indican los siguientes dos ejemplos,

```
(%i23) m1*m2;
                      [ 27   64    0  ]
                      [               ]
(%o23)                [ 216   0   - 8 ]
                      [               ]
                      [            3  ]
                      [  0   216  a   ]
(%i24) 4*m1;
                       [ 12  16   0  ]
                       [             ]
(%o24)                 [ 24  0   - 8 ]
                       [             ]
                       [ 0   24  4 a ]
```

Otros cálculos frecuentes con matrices son la transposición, el determinante, la inversión, el polinomio característico, así como los valores y vectores propios; para todos ellos hay funciones en Maxima:

```
(%i25) transpose(m1);    /*la transpuesta*/
                        [ 3   6   0 ]
                        [           ]
(%o25)                  [ 4   0   6 ]
                        [           ]
                        [ 0  - 2  a ]
(%i26) determinant(m1);  /*el determinante*/
(%o26)                  36 - 24 a
(%i27) invert(m1);       /*la inversa*/
```

```
         [     12            4 a           8     ]
         [  ---------   - ---------  - --------- ]
         [  36 - 24 a     36 - 24 a    36 - 24 a ]
         [                                       ]
         [      6 a         3 a           6      ]
(%o27)   [ - ---------   ---------    ---------  ]
         [   36 - 24 a   36 - 24 a    36 - 24 a  ]
         [                                       ]
         [     36            18           24     ]
         [  ---------   - ---------  - --------- ]
         [  36 - 24 a     36 - 24 a    36 - 24 a ]
(%i28) invert(m1),detout;  /*la inversa, con el determinante fuera*/
                  [  12    - 4 a  - 8  ]
                  [                    ]
                  [ - 6 a   3 a    6   ]
                  [                    ]
                  [  36    - 18   - 24 ]
(%o28)            ----------------------
                        36 - 24 a
(%i29) charpoly(m1,x); /*pol.  caract. con variable x*/
(%o29)     (3 - x) (12 - (a - x) x) - 24 (a - x)
(%i30) expand(%);     /*pol. caract. expandido*/
          3      2      2
(%o30) - x  + a x  + 3 x  - 3 a x + 12 x - 24 a + 36
```

Vamos a suponer ahora que a vale cero y calculemos los valores propios de la matriz,

```
(%i31) m1,a:0;
                      [ 3  4   0  ]
                      [           ]
(%o31)                [ 6  0  - 2 ]
                      [           ]
                      [ 0  6   0  ]
(%i32) eigenvalues(%o31);
           sqrt(15) %i + 3  sqrt(15) %i - 3
(%o32) [[- ---------------, ---------------, 6], [1, 1, 1]]
                  2                2
```

El resultado que se obtiene es una lista formada por dos sublistas, en la primera se encuentran los valores propios, que en este caso son $\lambda_1 = -\frac{3}{2} - \frac{\sqrt{15}}{2}i$, $\lambda_2 =$

$-\frac{3}{2}+\frac{\sqrt{15}}{2}i$ y $\lambda_3 = 6$, mientras que en la segunda sublista se nos indican las multiplicidades de cada una de las λ_i.

Para el cálculo de los vectores propios,

```
(%i33) eigenvectors(%o31);

            sqrt(15) %i + 3  sqrt(15) %i - 3
(%o33) [[[- ---------------, ---------------, 6],
                  2                2

                  sqrt(15) %i + 9
[1, 1, 1]], [1, - ---------------],
                         8

  3 sqrt(3) sqrt(5) %i - 21
- ------------------------]],
             8

    sqrt(15) %i - 9  3 sqrt(3) sqrt(5) %i + 21       3  3
[1, ---------------, ------------------------], [1, -, -]]
          8                     8                  4  4
```

Lo que se obtiene es, en primer lugar, los valores propios junto con sus multiplicidades, el mismo resultado que se obtuvo con la función `eigenvalues`; a continuación los vectores propios de la matriz asociados a cada uno de los valores propios. A veces interesa que los vectores sean unitarios, de norma 1, para lo que será de utilidad la función `uniteigenvectors`, que se encuentra definida en el paquete `eigen.lisp`, lo que significa que antes de hacer uso de ella habrá que ejecutar la orden `load(eigen)`.

Otros cálculos posibles con matrices,

```
(%i34) minor(%o31,2,1);  /* el menor de un elemento */
                         [ 4  0 ]
(%o34)                   [      ]
                         [ 6  0 ]
(%i35) rank(%o31);   /* el rango de la matriz*/
(%o35)                       3
(%i36) matrixmap(sin,%o31);
```

```
                [ sin(3)  sin(4)     0     ]
                [                          ]
(%o36)          [ sin(6)    0     - sin(2) ]
                [                          ]
                [   0     sin(6)     0     ]
```

En este último ejemplo, aplicamos la función seno a todos los elementos de la matriz.

9. Funciones matemáticas

En Maxima están definidas un gran número de funciones, algunas de las cuales se presentan en la Figura 5. A continuación se desarrollan algunos ejemplos sobre su uso.

Las funciones exponenciales siempre son positivas, independientemente del argumento x,

```
(%i1) limit(1/(x-1),x,1,minus);
(%o1)                         minf
(%i2) abs(3^-x);
                               1
(%o2)                          --
                                x
                               3
(%i3) signum(3^-x);
(%o3)                          1
```

La función `genfact(m,n,p)` es el factorial generalizado, de forma que `genfact(m,m,1)` coincide con $m!$,

```
(%i4) genfact(5,5,1)-5!;
(%o4)                          0
```

y `genfact(m,m/2,2)` es igual a $m!!$,

```
(%i5) genfact(5,5/2,2)-5!!;
(%o5)                          0
```

Maxima siempre devuelve resultados exactos, que nosotros podemos solicitar en formato decimal,

```
(%i6) asin(1);
                              %pi
(%o6)                         ---
                               2
(%i7) %,numer;
(%o7)                1.570796326794897
```

Recordemos que el formato decimal lo podemos pedir con precisión arbitraria,

```
(%i8) fpprec:50$ bfloat(%o6);
(%o9) 1.5707963267948966192313216916397514420985846996876B0
```

`abs(x)`	$\mathrm{abs}(x)$
`min(x1,x2,...)`	$\mathrm{mín}(x_1, x_2, \ldots)$
`max(x1,x2,...)`	$\mathrm{máx}(x_1, x_2, \ldots)$
`signum(x)`	$\mathrm{signo}(x) = \begin{cases} -1 & \text{si } x < 0 \\ 0 & \text{si } x = 0 \\ 1 & \text{si } x > 0 \end{cases}$
`x!`	$x!$
`x!!`	$x!!$
`binomial(m,n)`	$\binom{m}{n} = \frac{m(m-1)\ldots[m-(n-1)]}{n!}$
`genfact(m,n,p)`	$m(m-p)(m-2p)\ldots[m-(n-1)p]$
`sqrt(x)`	$\sqrt{x}$
`exp(x)`	e^x
`log(x)`	$\ln(x)$
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`csc(x)`	$\csc(x)$
`sec(x)`	$\sec(x)$
`cot(x)`	$\cot(x)$
`asin(x)`	$\arcsin(x)$
`acos(x)`	$\arccos(x)$
`atan(x)`	$\arctan(x)$
`atan2(x,y)`	$\arctan(\frac{x}{y}) \in (-\pi, \pi)$
`sinh(x)`	$\sinh(x) = \frac{1}{2}(e^x - e^{-x})$
`cosh(x)`	$\cosh(x) = \frac{1}{2}(e^x + e^{-x})$
`tanh(x)`	$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$
`asinh(x)`	$\mathrm{arcsinh}(x)$
`acosh(x)`	$\mathrm{arccosh}(x)$
`atanh(x)`	$\mathrm{arctanh}(x)$
`gamma(x)`	$\Gamma(x) = \int_0^\infty e^{-u}u^{x-1}du, \forall x > 0$
`beta(x,y)`	$\beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$
`erf(x)`	$\mathrm{erf}(x) = \int_0^x \frac{2}{\sqrt{\pi}}e^{-u^2}du$

Figura 5: Algunas funciones de Maxima.

La función de error está relacionada con la función de distribución de la variable aleatoria normal $X \sim \mathcal{N}(0,1)$ de la forma

$$\Phi(x) = \Pr(X \leq x) = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right),$$

por lo que la probabilidad de que esta variable tome un valor menor que 1,5 es

```
(%i10) 0.5+0.5*erf(1.5/sqrt(2)),numer;
(%o10)                0.9331927987311419
```

Una forma más elegante de hacer lo anterior es definir nuestra propia función de distribución a partir de la de error, para lo que se hace uso del símbolo `:=`,

```
(%i11) F(x):=1/2+erf(x/sqrt(2))/2$
(%i12) F(1.5),numer;
(%o12)                0.9331927987311419
```

No terminan aquí las funciones de Maxima; junto a las ya expuestas habría que incluir las funciones de Airy, elípticas y de Bessel, sobre las que se podrá obtener más información ejecutando la instrucción `describe` y utilizando como argumento `airy`, `elliptic` o `bessel`, según el caso. Por ejemplo,

```
(%i13) describe(airy);

 0: airy :(maxima.info)Definitions for Special Functions.
 1: airy_ai :Definitions for Special Functions.
 2: airy_bi :Definitions for Special Functions.
 3: airy_dai :Definitions for Special Functions.
 4: airy_dbi :Definitions for Special Functions.
Enter space-separated numbers, ‘all’ or ‘none’: 1

Info from file /usr/local/info/maxima.info:
 - Function: airy_ai (<x>)
     The Airy function Ai, as defined in Abramowitz and Stegun,
     Handbook of Mathematical Functions, Section 10.4.

     The Airy equation ‘diff (y(x), x, 2) - x y(x) = 0’ has two
     linearly independent solutions, ‘y = Ai(x)’ and ‘y = Bi(x)’.  The
     derivative ‘diff (airy_ai(x), x)’ is ‘airy_dai(x)’.
```

```
     If the argument ‘x’ is a real or complex floating point number,
     the numerical value of ‘airy_ai’ is returned when possible.

     See also ‘airy_bi’, ‘airy_dai’, ‘airy_dbi’.
(%o13)                       false
```

10. Límites

Sin más preámbulos, veamos algunos ejemplos de cómo calcular límites con la asistencia de Maxima. En primer lugar vemos que es posible hacer que la variable se aproxime al infinito ($x \to \infty$) haciendo uso del símbolo `inf`, o que se aproxime al menos infinito ($x \to -\infty$) haciendo uso de `minf`,

```
(%i1) limit(1/sqrt(x),x,inf);
(%o1)                         0
(%i2) limit((exp(x)-exp(-x))/(exp(x)+exp(-x)),x,minf);
(%o2)                        - 1
```

que nos permite calcular $\lim_{x\to\infty} \frac{1}{\sqrt{x}}$ y $\lim_{x\to-\infty} \frac{e^x-e^{-x}}{e^x+e^{-x}}$, respectivamente.

Los siguientes ejemplos muestran límites en los que la variable x se aproxima a puntos de discontinuidad,

```
(%i3) limit((x^2-x/3-2/3)/(5*x^2-11*x+6),x,1);
                               5
(%o3)                        - -
                               3
(%i4) limit(1/(x-1)^2,x,1);
(%o4)                        inf
```

donde hemos obtenido los resultados

$$\lim_{x\to 1} \frac{x^2 - \frac{x}{3} - \frac{2}{3}}{5x^2 - 11x + 6} = -\frac{5}{3}$$

y

$$\lim_{x\to 1} \frac{1}{(x-1)^2} = \infty.$$

Sin embargo, ciertos límites no se pueden resolver sin aportar información adicional, tal es el caso de $\lim_{x\to 1} \frac{1}{x-1}$, para el que podemos hacer

```
(%i5) limit(1/(x-1),x,1);
(%o5)                        und
```

donde Maxima nos responde con el símbolo `und` de *undefined* o indefinido. En tales situaciones podemos indicarle al asistente que la variable x se aproxima a 1 por la derecha ($x \to 1^+$) o por la izquierda ($x \to 1^-$),

```
(%i6) limit(1/(x-1),x,1,plus);
(%o6)                        inf
(%i7) limit(1/(x-1),x,1,minus);
(%o7)                        minf
```

11. Derivadas

Maxima controla el cálculo de derivadas mediante la instrucción `diff`. A continuación se presentan algunos ejemplos sobre su uso,

```
(%i1) diff(x^log(a*x),x);
                log(a x)  log(a x)   log(x)
(%o1)          x         (-------- + ------)
                                x         x
(%i2) diff(x^log(a*x),x,2); /*derivada segunda*/
       log(a x)  log(a x)   log(x) 2
(%o2) x         (-------- + ------)
                    x         x
                         log(a x)    log(a x)   log(x)   2
                   + x         (- -------- - ------ + --)
                                        2         2      2
                                       x         x      x
(%i3) factor(%);
       log(a x) - 2     2
(%o3) x             (log (a x) + 2 log(x) log(a x)
                                          2
                         - log(a x) + log (x) - log(x) + 2)
```

donde se han calculado la primera y segunda derivadas de la función $y = x^{\ln(ax)}$. Nótese que en la entrada `(%i3)` se le ha pedido al asistente que simplificase la salida `(%o2)`.

También se pueden calcular derivadas parciales, tal como se muestra a continuación,

```
(%i4) diff(x*cos(y)+y*sin(x*y),x,1,y,2);
                          2  2
(%o4) - 4 x y sin(x y) - x  y  cos(x y) + 2 cos(x y)
                                                    - cos(y)
(%i5) diff(exp(x)*sin(y)*tan(z),x,3,y,5,z,2);
                    x           2
(%o5)           2 %e  cos(y) sec (z) tan(z)
```

donde se han obtenido los resultados

$$\frac{\vartheta^3}{\vartheta x \vartheta y^2}\left(x\cos(y) + y\sin(xy)\right) = -4xy\sin(xy) - x^2y^2\cos(xy) + 2\cos(xy) - \cos(y)$$

y

$$\frac{\vartheta^{10}}{\vartheta x^3 \vartheta y^5 \vartheta z^2}\left(e^x \sin(y)\tan(z)\right) = 2e^x \cos(y)\sec^2(z)\tan(z).$$

Maxima también nos puede ayudar a la hora de aplicar la regla de la cadena en el cálculo de derivadas de funciones vectoriales con variable también vectorial. Supónganse que cierta variable z depende de otras dos x y y, las cuales a su vez dependen de u y v. Veamos cómo se aplica la regla de la cadena para obtener $\frac{\vartheta z}{\vartheta v}$, $\frac{\vartheta z^2}{\vartheta y \vartheta v}$ o $\frac{\vartheta z^2}{\vartheta u \vartheta v}$.

```
(%i6) depends(z,[x,y],[x,y],[u,v]);
(%o6)           [z(x, y), x(u, v), y(u, v)]
(%i7) diff(z,v,1);
                        dy dz   dx dz
(%o7)                   -- -- + -- --
                        dv dy   dv dx
(%i8) diff(z,y,1,v,1);
                           2         2
                       dy d z   dx  d z
(%o8)                  -- --- + -- -----
                       dv   2   dv dx dy
                          dy
(%i9) diff(z,u,1,v,1);
                2          2         2
       dy  dy d z    dx  d z      d y  dz
(%o9)  -- (-- --- + -- -----) + ----- --
       du  dv   2   dv dx dy     du dv dy
              dy
                                    2         2         2
                              dx  dx d z   dy  d z      d x  dz
                            + -- (-- --- + -- -----) + ----- --
                              du  dv   2   dv dx dy    du dv dx
                                     dx
```

En cualquier momento podemos solicitarle a Maxima que nos recuerde el cuadro de dependencias,

```
(%i10) dependencies;
(%o10)          [z(x, y), x(u, v), y(u, v)]
```

o también podemos eliminar dependencias,

```
(%i11) remove(x,dependency);
(%o11)                         done
(%i12) dependencies;
(%o12)                 [z(x, y), y(u, v)]
(%i13) diff(z,y,1,v,1);
                                  2
                              dy d z
(%o13)                        -- ---
                              dv   2
                                 dy
```

Veamos cómo deriva Maxima funciones definidas implícitamente. En el siguiente ejemplo, para evitar que `y` sea considerada una constante, le declararemos una dependencia respecto de `x`,

```
(%i14) depends(y,x)$
(%i15) diff(x^2+y^3=2*x*y,x);
                     2 dy             dy
(%o15)          3 y  -- + 2 x = 2 x -- + 2 y
                     dx               dx
```

Cuando se solicita el cálculo de una derivada sin especificar la variable respecto de la cual se deriva, Maxima utilizará el símbolo `del` para representar las diferenciales,

```
(%i16) diff(x^2);
(%o16)                       2 x del(x)
```

lo que se interpretará como $2xdx$. Si en la expresión a derivar hay más de una variable, habrá diferenciales para todas,

```
(%i17) diff(x^2+y^3=2*x*y);
          2               2 dy
(%o17) 3 y  del(y) + (3 y  -- + 2 x) del(x) =
                          dx
                                                dy
                          2 x del(y) + (2 x -- + 2 y) del(x)
                                                dx
```

Recuérdese que durante este cálculo estaba todavía activa la dependencia declarada en la entrada `(%i14)`.

Finalmente, para acabar esta sección, hagamos referencia al desarrollo de Taylor de tercer grado de la función

$$y = \frac{x \ln x}{x^2 - 1}$$

en el entorno de $x = 1$,

```
(%i18) taylor((x*log(x))/(x^2-1),x,1,3);
                           2          3
               1   (x - 1)    (x - 1)
(%o18)/T/      - - -------- + -------- + . . .
               2      12         12
(%i19) expand(%);
                       3    2
                      x    x    5 x   1
(%o19)                -- - -- + --- + -
                      12   3    12    3
```

A continuación un ejemplo de desarrollo multivariante de la función $y = \exp(x^2 \sin(xy))$ alrededor del punto $(2, 0)$ hasta grado 2 respecto de cada variable,

```
(%i20) taylor(exp(x^2*sin(x*y)),[x,2,2],[y,0,2]);
                          2
(%o20)/T/ 1 + 8 y + 32 y  + . . .
                 2
 + (12 y + 96 y  + . . .) (x - 2)
                  2                  2
 + (6 y + 120 y  + . . .) (x - 2)  + . . .
(%i21) expand(%);
              2  2           2         2       2
(%o21) 120 x  y  - 384 x y  + 320 y  + 6 x  y - 12 x y + 8 y
                                                          + 1
```

12. Integrales

La función de Maxima que controla el cálculo de integrales es `integrate`, tanto para las definidas como indefinidas; empecemos por estas últimas,

```
(%i1) integrate(cos(x)^3/sin(x)^4,x);
                            2
                       3 sin (x) - 1
(%o1)                  -------------
                              3
                         3 sin (x)
(%i2) integrate(a[3]*x^3+a[2]*x^2+a[1]*x+a[0],x);
                    4       3       2
                a  x    a  x    a  x
                 3       2       1
(%o2)           ----- + ----- + ----- + a  x
                  4       3       2      0
```

que nos devuelve los resultados

$$\int \frac{\cos^3 x}{\sin^4 x} dx = \frac{3\sin^2 x - 1}{3\sin^3 x}$$

y

$$\int (a_3 x^3 + a_2 x^2 + a_1 x + a_0) dx = \frac{a_3 x^4}{4} + \frac{a_2 x^3}{3} + \frac{a_1 x^2}{2} + a_0 x.$$

Además, este último ejemplo nos ofrece la oportunidad de ver cómo escribir coeficientes con subíndices.

Ahora un par de ejemplos sobre la integral definida,

```
(%i3) integrate(2*x/((x-1)*(x+2)),x,3,5);
            2 log(7) + log(4)   2 log(5) + log(2)
(%o3)   2 (----------------- - -----------------)
                    3                   3
(%i4) %,numer;   /*aproximacion decimal*/
(%o4)                  0.91072776920158
(%i5) integrate(asin(x),x,0,u);
Is  u  positive, negative, or zero?

positive;
                                        2
(%o5)           u asin(u) + sqrt(1 - u ) - 1
```

esto es,

$$\int_3^5 \frac{2x}{(x-1)(x+2)}dx \approx 0{,}91072776920158$$

y

$$\int_0^u \arcsin(x)dx = u\arcsin(u) + \sqrt{1-u^2} - 1, \forall u > 0.$$

Nótese en este último ejemplo cómo antes de dar el resultado Maxima pregunta si u es positivo, negativo o nulo; tras contestarle escribiendo `positive;` (punto y coma incluido) obtenemos finalmente el resultado.

La transformada de Laplace de una función $f(x)$ se define como la integral

$$L(p) = \int_0^\infty f(x)e^{-px}dx,$$

siendo p un número complejo. Así, la transformada de Laplace de $f(x) = ke^{-kx}$ es

```
(%i6) laplace(k*exp(-k*x),x,p);
                                k
(%o6)                         -----
                              p + k
```

y calculando la transformada inversa volvemos al punto de partida,

```
(%i7) ilt(%,p,x);
                                  - k x
(%o7)                        k %e
```

La transformada de Fourier de una función se reduce a la de Laplace cuando el argumento $p = -it$, siendo i la unidad imaginaria y $t \in \mathbb{R}$,

$$F(t) = \int_0^\infty f(x)e^{itx}dx.$$

De esta manera, la transformada de Fourier de $f(x) = ke^{-kx}$ es

```
(%i8) laplace(k*exp(-k*x),x,-%i*t);
                                k
(%o8)                        --------
                             k - %i t
```

Nótese que si $x > 0$, la $f(x)$ anterior es precisamente la función de densidad de una variable aleatoria exponencial de parámetro k, por lo que este último resultado coincide precisamente con la función característica de esta misma distribución. Téngase en cuenta que Maxima calcula la transformada de Laplace integrando en la semirecta positiva, por lo que el cálculo anterior no es válido para calcular funciones características de variables aleatorias que puedan tomar valores negativos, como por ejemplo la distribución normal. Una posible solución a esta situación es integrar directamente para calcular la función característica de la distribución normal $X \sim \mathcal{N}(0,1)$,

```
(%i9) integrate(1/sqrt(2*%pi)*exp(-x^2/2)*exp(%i*t*x),x,minf,inf);
                                2  2
                              %i  t
                              ------
                                2
(%o9)                        %e
(%i10) ratsimp(%);  /* Ojo: %i^2=-1 */
                                 2
                                t
                              - --
                                2
(%o10)                       %e
```

13. Ecuaciones diferenciales

Con Maxima se pueden resolver analíticamente algunas ecuaciones diferenciales ordinarias de primer y segundo orden mediante la instrucción `ode2`.

Una ecuación diferencial de primer orden tiene la forma general $F(x, y, y') = 0$, donde $y' = \frac{dy}{dx}$. Para expresar una de estas ecuaciones se hace uso de `diff`,

```
(%i1) /* ecuacion de variables separadas */
      ec:(x-1)*y^3+(y-1)*x^3*'diff(y,x)=0;
                 3         dy            3
(%o1)           x  (y - 1) -- + (x - 1) y = 0
                           dx
```

siendo obligatorio el uso del apóstrofo (') antes de `diff` al objeto de evitar el cálculo de la derivada, que por otro lado daría cero al no haberse declarado la variable `y` como dependiente de `x`. Para la resolución de esta ecuación tan solo habrá que hacer

```
(%i2) ode2(ec,y,x);
                    2 y - 1         2 x - 1
(%o2)               ------- = %c - -------
                        2               2
                     2 y             2 x
```

donde `%C` representa una constante, que se ajustará de acuerdo a la condición inicial que se le imponga a la ecuación. Supóngase que se sabe que cuando $x = 2$, debe verificarse que $y = -3$, lo cual haremos saber a Maxima a través de la función `ic1`,

```
(%i3) ic1(%o2,x=2,y=-3);
                                 2
                    2 y - 1     x  + 72 x - 36
(%o3)               ------- = - --------------
                        2               2
                     2 y             72 x
```

Veamos ejemplos de otros tipos de ecuaciones diferenciales que puede resolver Maxima,

```
(%i4) /* ecuacion homogenea */
      ode2(x^3+y^3+3*x*y^2*'diff(y,x),y,x);
```

```
                              3    4
                           4 x y  + x
(%o4)                      ---------- = %c
                              4
```

En este caso, cuando no se incluye el símbolo de igualdad, se da por hecho que la expresión es igual a cero.

```
(%i5) /* reducible a homogénea */
      ode2('diff(y,x)=(x+y-1)/(x-y-1),y,x);
             2    2                        x - 1
       log(y  + x  - 2 x + 1) + 2 atan(-----)
                                          y
(%o5)  ------------------------------------ = %c
                         4
(%i6) /* ecuacion exacta */
       ode2((4*x^3+8*y)+(8*x-4*y^3)*'diff(y,x),y,x);
                     4            4
(%o6)              - y  + 8 x y + x  = %c
(%i7) /* Bernoulli */
       ode2('diff(y,x)-y+sqrt(y),y,x);
(%o7)           2 log(sqrt(y) - 1) = x + %c
(%i8) solve(%,y);
                   x + %c        x/2 + %c/2
(%o8)       [y = %e       + 2 %e           + 1]
```

En este último caso, optamos por obtener la solución en su forma explícita.

Una ecuación diferencial ordinaria de segundo orden tiene la forma general $F(x, y, y', y'') = 0$, siendo y'' la segunda derivada de y respecto de x. Como ejemplo,

```
(%i9) 'diff(y,x)=x+'diff(y,x,2);
                              2
                        dy   d y
(%o9)                   -- = --- + x
                        dx     2
                             dx
(%i10) ode2(%,y,x);
```

```
                                  2
                              x   x  + 2 x + 2
(%o10)          y = %k1 %e  + ------------ + %k2
                                    2
```

Maxima nos devuelve un resultado que depende de dos parámetros, `%k1` y `%k2`, que para ajustarlos necesitaremos proporcionar ciertas condiciones iniciales; si sabemos que cuando $x = 1$ entonces $y = -1$ y $y' = \frac{dy}{dx}\big|_{x=1} = 2$, haremos uso de la instrucción `ic2`,

```
(%i11) ic2(%,x=1,y=-1,diff(y,x)=2);
                               2
                              x  + 2 x + 2   7
(%o11)                 y = ------------ - -
                                 2         2
```

En el caso de las ecuaciones de segundo orden, también es posible ajustar los parámetros de la solución especificando condiciones de contorno, esto es, fijando dos puntos del plano por los que pase la solución; así, si la solución obtenida en `(%o10)` debe pasar por los puntos $(-1, 3)$ y $(2, \frac{5}{3})$, hacemos

```
(%i12) bc2(%o10,x=-1,y=3,x=2,y=5/3);
                   x + 1    2                    3
              35 %e        x  + 2 x + 2   15 %e  + 20
(%o12) y = - ---------- + ------------ + -----------
                  3             2              3
              6 %e  - 6                    6 %e  - 6
```

Nótese que este cálculo se le solicita a Maxima con `bc2`.

La resolución de sistemas de ecuaciones diferenciales se hace con llamadas a la función `desolve`. En este contexto es preciso tener en cuenta que se debe utilizar notación funcional dentro de la expresión `diff`; un ejemplo aclarará este punto, resolviendo el sistema

$$\begin{cases} \frac{df(x)}{dx} &= 3f(x) - 2g(x) \\ \frac{dg(x)}{dx} &= 2f(x) - 2g(x) \end{cases}$$

```
(%i13) desolve(['diff(f(x),x)=3*f(x)-2*g(x),
                'diff(g(x),x)=2*f(x)-2*g(x)],
               [f(x),g(x)]);
```

```
                                  - x
                (2 g(0) - f(0)) %e
(%o13) [f(x) = --------------------
                          3
                        2 x
   (2 g(0) - 4 f(0)) %e
 - ----------------------, g(x) =
              3
                    - x                       2 x
(4 g(0) - 2 f(0)) %e      (g(0) - 2 f(0)) %e
---------------------- - --------------------]
           3                         3
```

Como se ve, las referecias a las funciones deben incluir la variable independiente y las ecuaciones estarán acotadas entre corchetes, así como los nombres de las funciones. Observamos en la respuesta que nos da Maxima la presencia de `f(0)` y `g(0)`, lo cual es debido a que se desconocen las condiciones de contorno del sistema.

En este último ejemplo, supongamos que queremos resolver el sistema de ecuaciones diferenciales

$$\begin{cases} \frac{df(x)}{dx} &= f(x)+g(x)+3h(x) \\ \frac{dg(x)}{dx} &= g(x)-2h(x) \\ \frac{dh(x)}{dx} &= f(x)+h(x) \end{cases}$$

bajo las condiciones $f(0) = -1$, $g(0) = 3$ y $f(0) = 1$. En primer lugar introduciremos estas condiciones con la función `atvalue`, para posteriormente solicitar la resolución del sistema,

```
(%i14) atvalue(f(x),x=0,-1)$
(%i15) atvalue(g(x),x=0,3)$
(%i16) atvalue(h(x),x=0,1)$
(%i17) desolve(['diff(f(x),x)=f(x)+g(x)+3*h(x),
         'diff(g(x),x)=g(x)-2*h(x),
         'diff(h(x),x)=f(x)+h(x)], [f(x),g(x),h(x)]);
                    2 x     2 x       - x
(%o17) [f(x) = x %e    + %e    - 2 %e   ,
                2 x       2 x     - x
g(x) = - 2 x %e    + 2 %e    + %e   ,
           2 x     - x
h(x) = x %e    + %e   ]
```

14. Números aleatorios

Si su argumento es un número entero positivo, la función `random(n)` genera un número seudoaleatorio con distribución uniforme discreta entre 0 y $n-1$, ambos inclusive; así, una simulación del lanzamiento de un dado sería

```
(%i1) random(6)+1;
(%o1)                          5
```

y una serie de 100 lanzamientos de una moneda,

```
(%i2) makelist(random(2),i,1,100);
(%o2) [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1,
1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1,
1, 1, 1]
```

Cuando el argumento es un número decimal positivo, la variable aleatoria que se simula es la uniforme continua, dando como resultado un número real perteneciente al intervalo $[0, r)$,

```
(%i3) random(6.0);
(%o3)                1.171107706786732
```

El algoritmo generador de los números seudoaleatorios es determinista, de manera que partiendo de una misma semilla o valor inicial, se generará la misma secuencia de números. Para controlar el valor de esta semilla disponemos de las funciones `make_random_state` y `set_random_state`; por ejemplo, para definir una semilla que se genere a partir del estado actual del reloj del sistema haremos

```
(%i4) nueva_semilla: make_random_state(true)$
```

Sin embargo, para que tal semilla se active en el generador, debemos indicarlos expresamente haciendo

```
(%i5) set_random_state(nueva_semilla)$
```

El argumento de la función `make_random_state` puede ser también un número entero, como se hace en el ejemplo de más abajo.

Veamos un caso de aplicación de todo esto. Supongamos que queremos simular diferentes series estocásticas, pero que todas ellas sean iguales. Si hacemos

```
(%i6) makelist(random(6),i,1,10);
(%o6)          [0, 2, 2, 4, 3, 0, 3, 3, 5, 3]
(%i7) makelist(random(6),i,1,10);
(%o7)          [5, 4, 4, 5, 0, 1, 3, 1, 3, 4]
(%i8) makelist(random(6),i,1,10);
(%o8)          [4, 4, 3, 4, 5, 2, 5, 5, 2, 3]
```

lo más probable es que obtengamos tres secuencias distintas, como en el ejemplo. Pero si hacemos

```
(%i9) semilla: make_random_state(123456789)$
(%i10) set_random_state(semilla)$ makelist(random(6),i,1,10);
(%o11)         [4, 4, 0, 1, 0, 3, 2, 5, 4, 4]
(%i12) set_random_state(semilla)$ makelist(random(6),i,1,10);
(%o13)         [4, 4, 0, 1, 0, 3, 2, 5, 4, 4]
(%i14) set_random_state(semilla)$ makelist(random(6),i,1,10);
(%o15)         [4, 4, 0, 1, 0, 3, 2, 5, 4, 4]
```

se verá que las tres secuencias son iguales, ya que antes de generar cada muestra aleatoria reiniciamos el estado del generador.

Otra variable aleatoria que suele interesar simular es la normal o gaussiana con media $\mu = m \in \mathbb{R}$ y desviación típica $\sigma = s > 0$, para lo cual se dispone de la función `gauss(m,s)`,

```
(%i16) gauss(31.45,1.23);
(%o16)                32.2873298461951
```

El generador de números aleatorios normales opera independientemente del mecanismo de semilla comentado más arriba.

Maxima dispone del paquete adicional `distrib` que permite simular muchas otras variables aleatorias, tanto discretas como continuas. A modo de ejemplo, pedimos sendas muestra de tamaño 5 de las variables aleatorias binomial $B(5, \frac{1}{3})$, Poisson $P(7)$, hipergeométrica $HP(15, 20, 7)$, exponencial $Exp(12{,}5)$ y Weibull $Wei(7, \frac{23}{3})$,

```
(%i17) load(distrib)$

(%i18) rbinomial(5,1/3,5);
(%o18)                [2, 1, 3, 2, 3]
(%i19) rpoisson(7,5);
(%o19)                [8, 2, 8, 6, 7]
```

```
(%i20) rhypergeo(15,20,7,5);
(%o20)                  [4, 3, 5, 4, 1]
(%i21) rexp(12.5,5);
(%o21) [.1419531364598988, .03965417848493739,
  .1127170185486312, .06784269182147352, 0.0163561427023252]
(%i22) rweibull(7,23/3,5);
(%o22) [8.161278429025327, 6.84800579381546,
     7.556572314785178, 8.253912037023762, 7.44772525941239]
```

Para más información sobre estas y otras funciones de simulación estocástica, tecléese `? distrib`.

15. Gráficos

Maxima no está habilitado para realizar él mismo gráficos, por lo que necesitará de un programa externo que realice esta tarea. Nosotros desde Maxima nos encargaremos de ordenar qué tipo de gráfico queremos y Maxima se encargará de comunicárselo a la aplicación gráfica que esté activa en ese momento, que por defecto será Gnuplot.

Veremos en primer lugar algunos ejemplos de cómo generar gráficos desde Maxima con Gnuplot y luego trataremos brevemente cómo podemos modificar algunas de las opciones por defecto del entorno gráfico.

Lo más sencillo es dibujar una simple función en el plano, por ejemplo $y = e^{-x^2}$, en un subdominio tal como $[-2, 5]$,

```
(%i1) plot2d(exp(-x^2),[x,-2,5])$
```

cuyo resultado se puede observar en el apartado *a)* de la Figura 6. En el apartado *b)* de la misma figura se puede ver cómo es posible representar varias funciones de una sola vez,

```
(%i2) plot2d([-x^2,1+x,7*sin(x)],[x,-2,5])$
```

Para la realización de funciones definidas paramétricamente necesitamos hacer uso del símbolo `parametric`, Figura 7,

```
(%i3) plot2d([parametric,t,t*sin(1/t),[t,0.01,0.2]])$
```

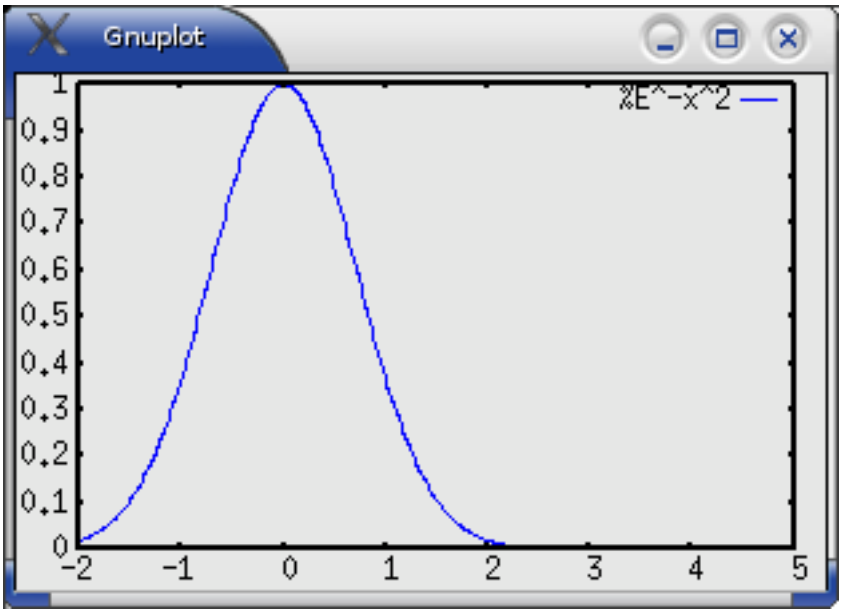
El resultado que se obtine es el del apartado *a)*. Maxima calcula por defecto un número fijo de puntos de la función que luego utilizará para representarla; como esta es una función que varía mucho cerca del origen, pediremos que nos haga el dibujo con un mayor número de puntos, lo que se hará mediante la opción `nticks`, tal como se indica a continuación,

```
(%i4) plot2d([parametric,t,t*sin(1/t),[t,0.01,0.2],[nticks,500]])$
```

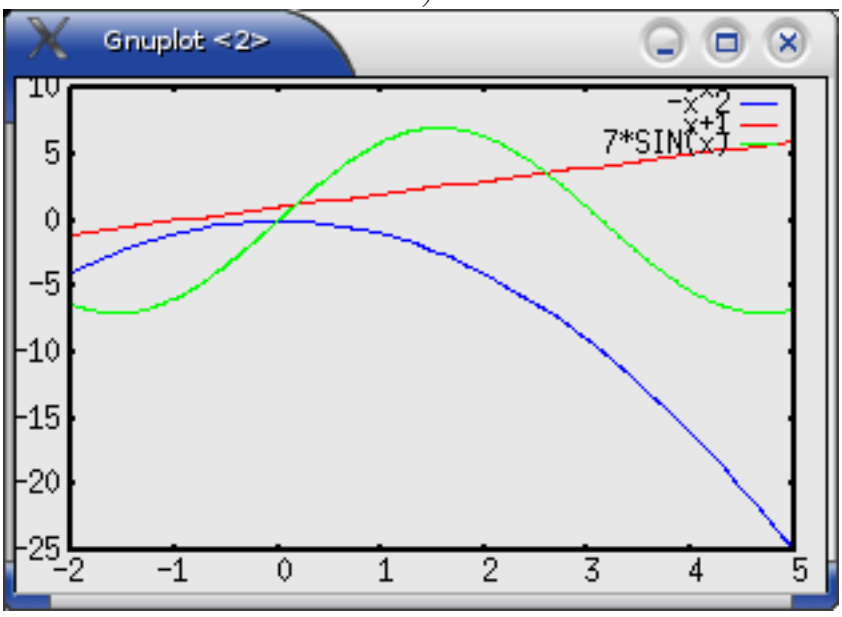
Se comprueba que el aspecto ha cambiado apreciablemente.

El siguiente ejemplo muestra la presencia en un mismo gráfico de la función $y = x^3 + 2$ junto con la circunferencia de radio unidad, expresada paramétricamente, que en el apartado *a)* de la Figura 8 se ve como una elipse debido a la diferencia de escalas en los ejes,

```
(%i5) plot2d([x^3+2,[parametric,cos(t),sin(t),[t,-5,5]]],
             [x,-1.3,1.3],[nticks,500])$
```

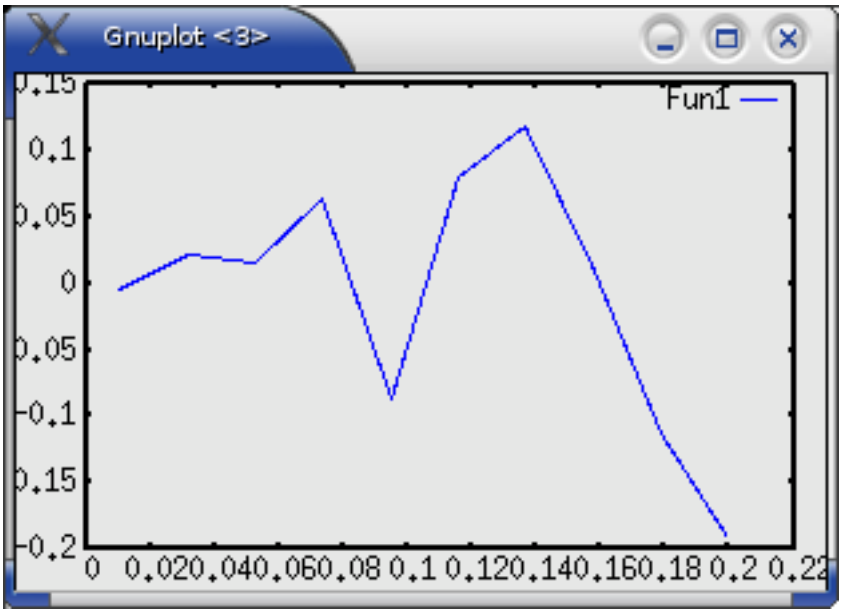


a)

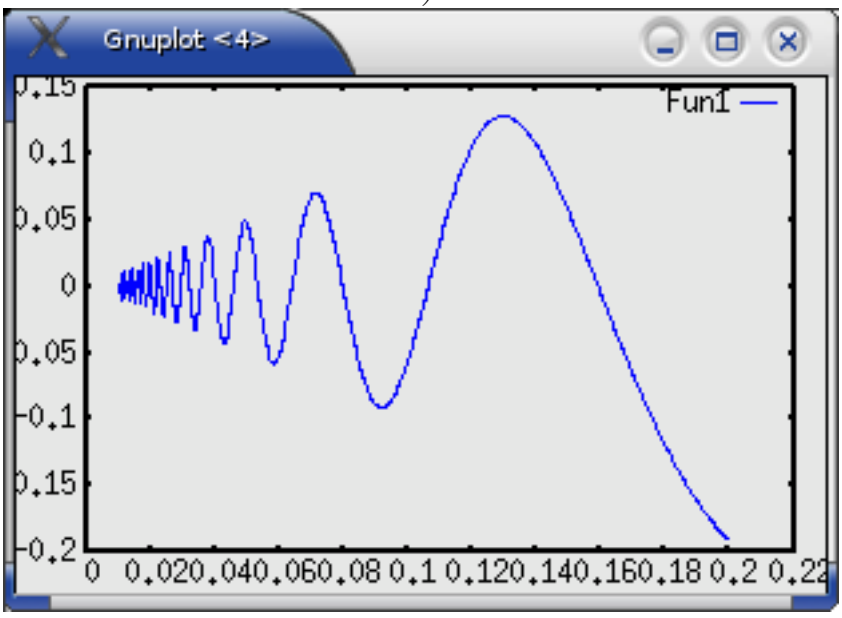


b)

Figura 6: Funciones en coordenadas cartesianas: *a)* una sola función; *b)* una familia de funciones.

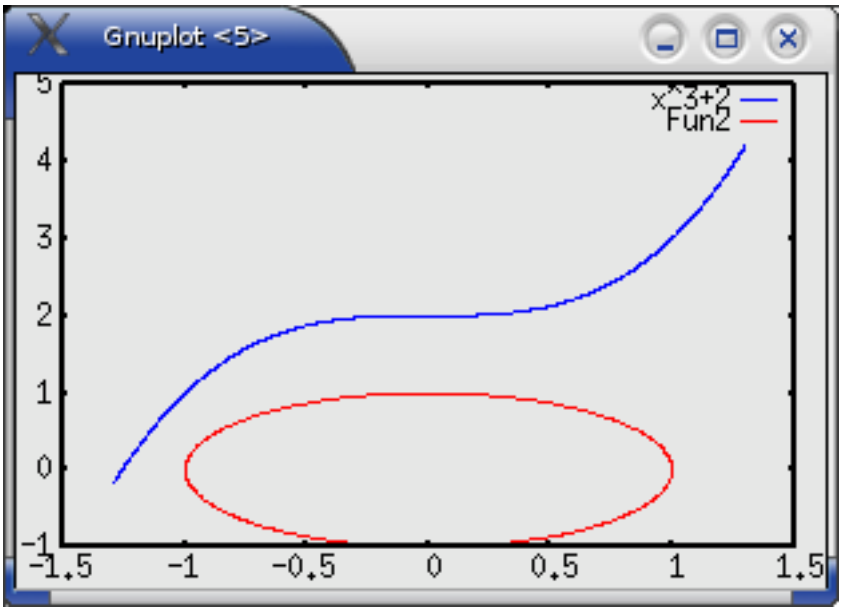


a)

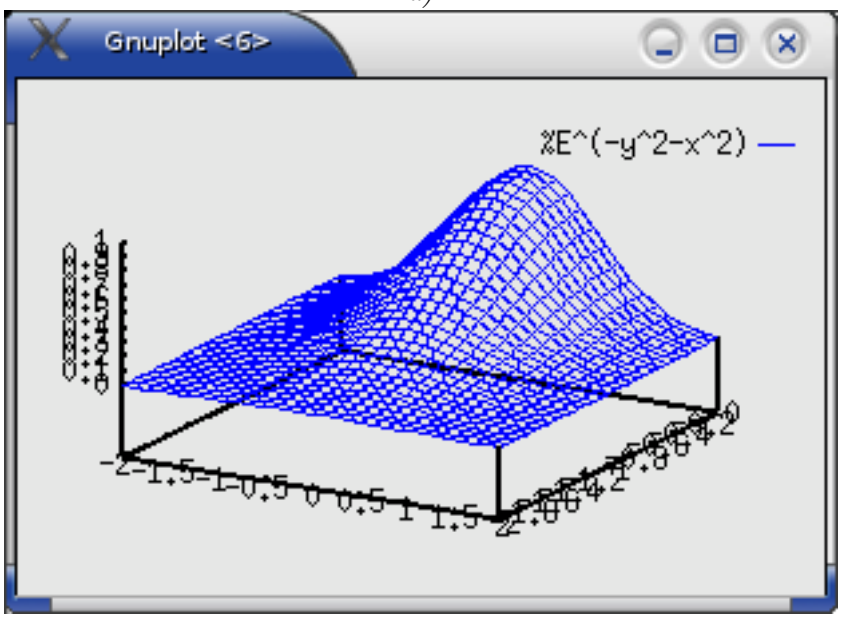


b)

Figura 7: Control del número de puntos: *a)* número de puntos por defecto; *b)* la misma función con 500 puntos.



a)



b)

Figura 8: Más tipos de gráficos: *a)* combinando funciones paramétrica y no paramétrica; *b)* superficie en 3D.

También es posible la generación de superficies en 3D definidas de la forma $z = f(x, y)$, apartado *b)* de la Figura 8,

```
(%i6) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0])$
```

Finalmente, del manual de Maxima se ha extraído la banda de Möbius como un ejemplo de superficie paramética, apartado *a)* de la Figura 9,

```
(%i7) plot3d([cos(x)*(3+y*cos(x/2)),
              sin(x)*(3+y*cos(x/2)),y*sin(x/2)],
              [x,-%pi,%pi],[y,-1,1],['grid,50,15]);
```

La función `plot3d` es capaz de generar una curva paramétrica en tres dimensiones, como lo prueba esta hélice, que se puede ver en el apartado *b)* de la Figura 9,

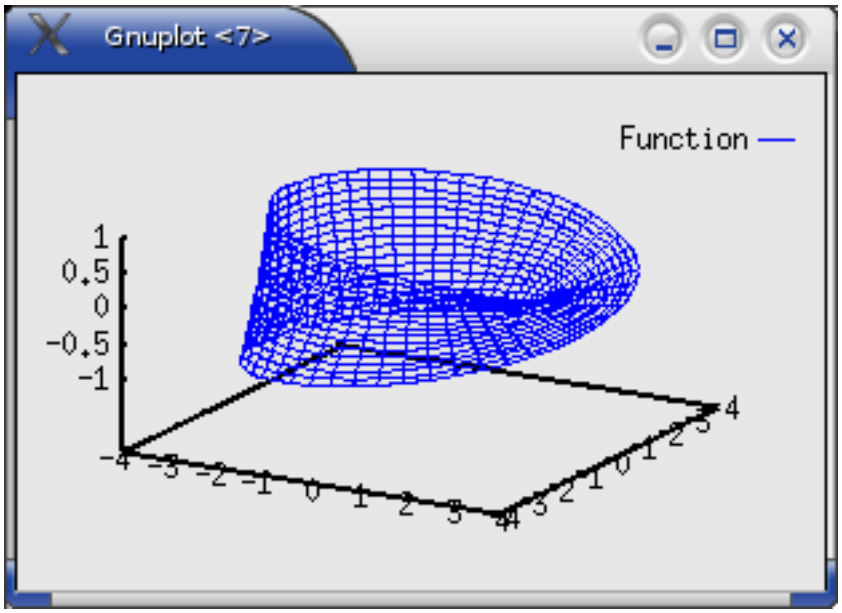
```
(%i8) plot3d([cos(t),sin(t),2*t],[t,-%pi,%pi],[u,0,1]);
```

El control de las opciones gráficas se consigue manipulando la variable global `plot_options`, cuyo estado por defecto es

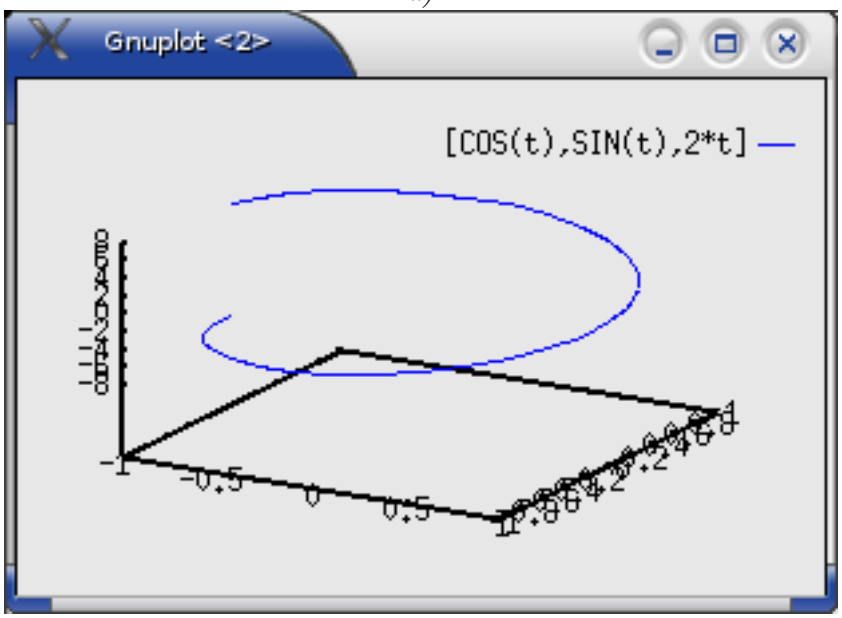
```
(%i9) plot_options;
(%o9) [[x, - 1.755559702014e+305, 1.755559702014e+305],
[y, - 1.755559702014e+305, 1.755559702014e+305],
[t, - 3, 3], [GRID, 30, 30], [VIEW_DIRECTION, 1, 1, 1],
[COLOUR_Z, FALSE], [TRANSFORM_XY, FALSE],
[RUN_VIEWER, TRUE], [PLOT_FORMAT, GNUPLOT],
[GNUPLOT_TERM, DEFAULT], [GNUPLOT_OUT_FILE, FALSE],
[NTICKS, 10], [ADAPT_DEPTH, 10], [GNUPLOT_PM3D, FALSE],
[GNUPLOT_PREAMBLE, ], [GNUPLOT_CURVE_TITLES, [DEFAULT]],
[GNUPLOT_CURVE_STYLES, [with lines 3, with lines 1,
with lines 2, with lines 5, with lines 4, with lines 6,
with lines 7]], [GNUPLOT_DEFAULT_TERM_COMMAND, ],
[GNUPLOT_DUMB_TERM_COMMAND, set term dumb 79 22],
[GNUPLOT_PS_TERM_COMMAND, set size 1.5, 1.5;set term postsc#
ript eps enhanced color solid 24]]
```

Para mayor información sobre el significado de cada uno de los elementos de esta lista sería aconsejable ejecutar el comando `describe(plot_options)`.

Ya se ha comentado que a menos que se le indique lo contrario, Maxima invocará al programa Gnuplot para la representación de un gráfico, pero quizás



a)



b)

Figura 9: Gráficos paramétricos en 3D: *a)* una superficie; *b)* una curva.

prefiramos el programa Openmath, que forma parte de la distribución de Maxima; en tal caso tendríamos que modificar previamente las opciones guardadas en `plot_options` y a continuación solicitar el gráfico deseado, como en este caso en el que se representa la función gamma y su inversa, cuyo resultado se observa en el apartado *a)* de la Figura 10

```
(%i10) set_plot_option([plot_format, openmath])$
(%i11) plot2d([gamma(x),1/gamma(x)],[x,-4.5,5],[y,-10,10])$
```

También Openmath es capaz de realizar gráficos en tres dimensiones definidos en coordenadas polares y que se representa en el apartado *b)* de la Figura 10. Nótese cómo desde la propia función es posible alterar las opciones gráficas; sin embargo, hecho de esta manera los cambios sólo tienen lugar para la realización del gráfico presente, sin alterar las opciones globales.

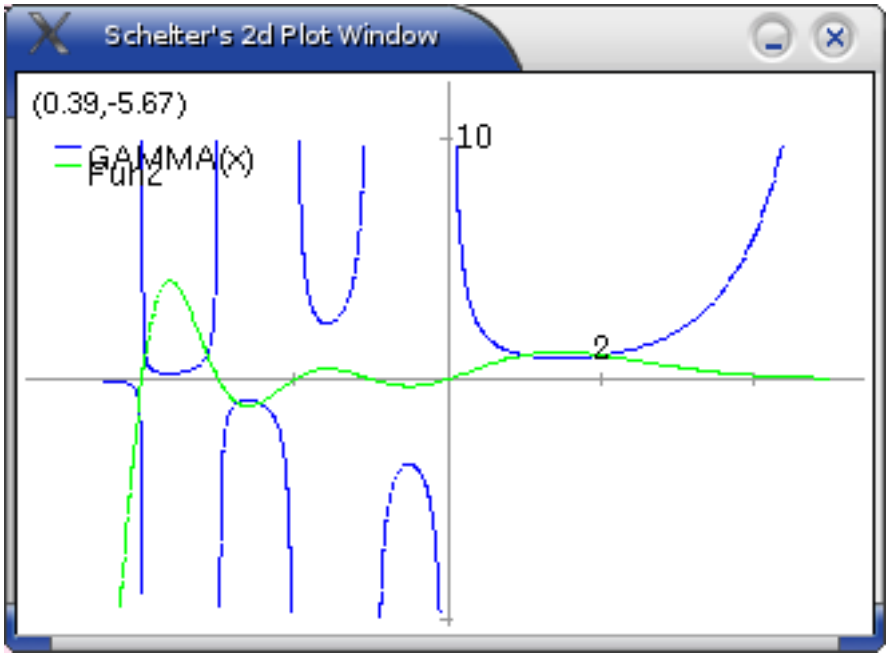
```
(%i12) plot3d(r^0.33*cos(t/3),[r,0,1],[t,0,6*%pi],
              ['grid,20,80],['transform_xy,polar_to_xy])$
```

Las Figuras que se han representado hasta ahora en esta sección son capturas de pantalla de la ventana generada por Gnuplot o por Openmath y guardadas en disco en formato PNG. Sin embargo, podemos hacer que el programa gráfico genere directamente un archivo PNG en lugar de mostrarlo en pantalla. Volvamos a Gnuplot y obtengamos directamente de este programa un archivo con este formato de uno de los gráficos que ya realizamos anteriormente,

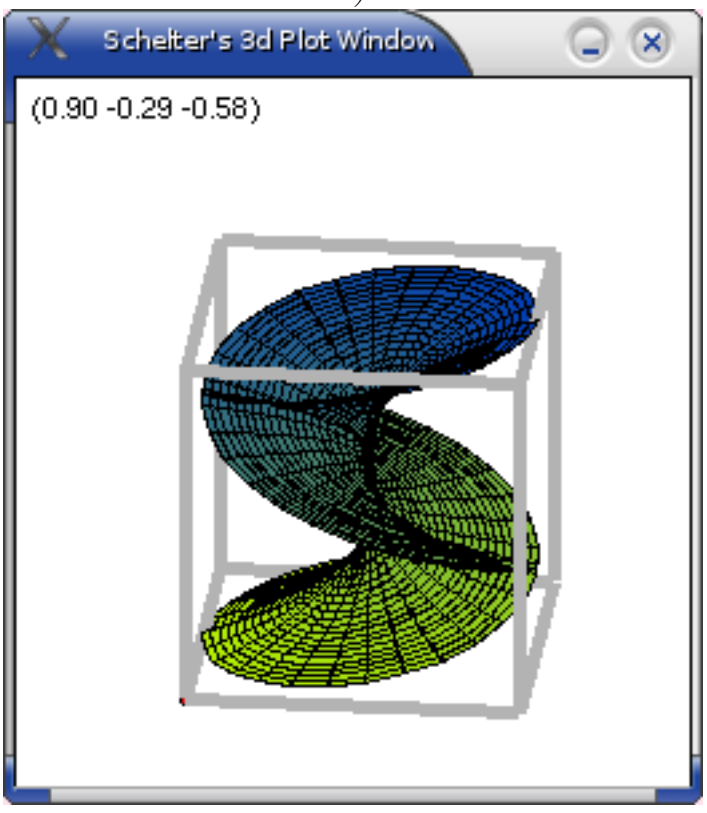
```
(%i13) set_plot_option([plot_format, gnuplot])$
(%i14) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0],
              [gnuplot_preamble,"set terminal png size 420,320;
                                 set out 'grafico.png'"])$
```

Después de cambiar globalmente el programa gráfico, volvemos a representar la superficie tridimensional que ya vimos en el apartado *b)* de la Figura 8. Véase cómo se ha cambiado el parámetro gráfico `gnuplot_preamble` para indicarle a Gnuplot que genere un archivo en formato PNG de ciertas dimensiones dadas y que lo guarde con el nombre `grafico.png`. El resultado lo vemos en el apartado *a)* de la Figura 11. Cuando se trabaja en Gnuplot, el parámetro `gnuplot_preamble` permite pasar a este programa una serie de comandos que afinan los detalles; estos comandos deben ir separados por puntos y comas y deben ser los propios del lenguaje de Gnuplot. Para un mejor dominio de estos detalles es aconsejable recurrir a la documentación de este programa (www.gnuplot.info).

Si se quiere un archivo gráfico en Postscript, repitamos el último ejemplo solicitando este formato,



a)



b)

Figura 10: Gráficos en Openmath: *a)* en el plano; *b)* en tres dimensiones.

```
(%i15) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0],
              [gnuplot_preamble,"set terminal postscript eps;
                                 set out 'grafico.eps'"])$
```

Los gráficos almacenados en formato Postscript, los que tienen extensión EPS, suelen ser necesarios cuando se planea crear un documento basado en TEX-LATEX.

Por último, veamos cómo generar un gráfico a partir de una lista de puntos dados por sus coordenadas.

```
(%i16) bs:[[64,-104], [64,-99], [61,-95], [57,-95], [55,-94],
           [55,-90], [72,-89], [80,-88], [77,-86], [80,-88],
           [82,-91], [80,-88], [72,-89], [55,-90], [47,-89],
           [39,-87], [33,-84], [32,-81], [35,-75], [36,-69],
           [42,-69], [45,-68], [42,-69], [36,-69], [35,-67],
           [35,-62], [35,-60], [39,-60], [45,-61], [51,-61],
           [53,-65], [56,-68], [61,-70], [67,-71], [73,-70],
           [78,-67], [80,-64], [81,-61], [78,-58], [75,-60],
           [75,-62], [78,-62], [81,-61], [81,-56], [79,-50],
           [75,-47], [70,-44], [62,-44], [58,-45], [53,-50],
           [52,-52], [49,-53], [48,-52], [48,-50], [49,-48],
           [51,-49], [52,-52], [51,-54], [51,-57], [51,-61],
           [45,-61], [39,-60], [35,-60], [32,-57], [32,-52],
           [32,-49], [35,-45], [39,-42], [43,-41], [48,-42],
           [51,-43], [55,-47], [51,-43], [48,-42], [43,-41],
           [44,-38], [47,-35], [54,-16], [56,-5], [60,-11],
           [64,-6], [67,-12], [72,-7], [74,-14], [80,-9],
           [80,-16], [86,-11], [87,-17], [92,-12], [93,-21],
           [99,-16], [100,-22], [106,-19], [107,-27], [115,-25],
           [110,-32], [94,-72], [91,-72], [88,-73], [87,-74],
           [88,-76], [91,-76], [91,-79], [89,-81], [91,-79],
           [91,-76], [94,-77], [94,-79], [94,-77], [91,-76],
           [88,-76], [87,-74], [88,-73], [91,-72], [94,-72],
           [97,-75], [98,-80], [97,-82], [94,-84], [91,-85],
           [87,-83], [84,-80], [87,-83], [91,-85], [91,-95],
           [92,-102], [85,-105], [69,-106], [64,-104]]$
(%i17) plot2d([discrete,bs],
              [gnuplot_curve_styles,["with lines"]],
              [gnuplot_preamble,"set size 0.4, 0.6;
                                 set terminal png;
```

```
                                    set out 'bart.png';
                                    set nokey"])$
```

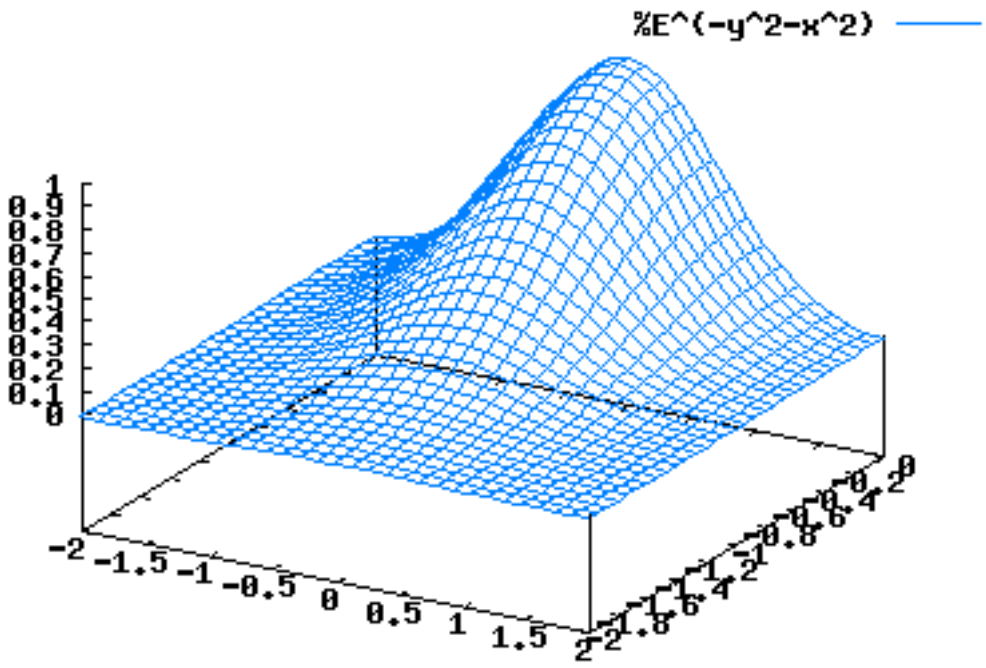
Puesto que estamos hablando de LaTeX, este es un buen lugar para hacer referencia a una función que transforma una expresión de Maxima a formato TeX, de manera que el resultado que se obtenga pueda ser incorporado fácilmente (copiar y pegar) a un archivo fuente de LaTeX; a modo de ejemplo, calculemos una derivada para posteriormente transformarla,

```
(%i18) 'diff(sin(x^x)*sqrt(log(x)),x)=
           diff(sin(x^x)*sqrt(log(x)),x);
       d                       x
(%o18) -- (sqrt(log(x)) sin(x )) =
       dx

              x
         sin(x )          x                                  x
     ---------------- + x  sqrt(log(x)) (log(x) + 1) cos(x )
     2 x sqrt(log(x))
(%i19) tex(%);

$${{d}\over{d\,x}}\,\left(\sqrt{\log x}\,\sin x^{x}\right)={{\sin x^{
 x}}\over{2\,x\,\sqrt{\log x}}}+x^{x}\,\sqrt{\log x}\,\left(\log x+1
 \right)\,\cos x^{x}$$
(%o19)                         false
```

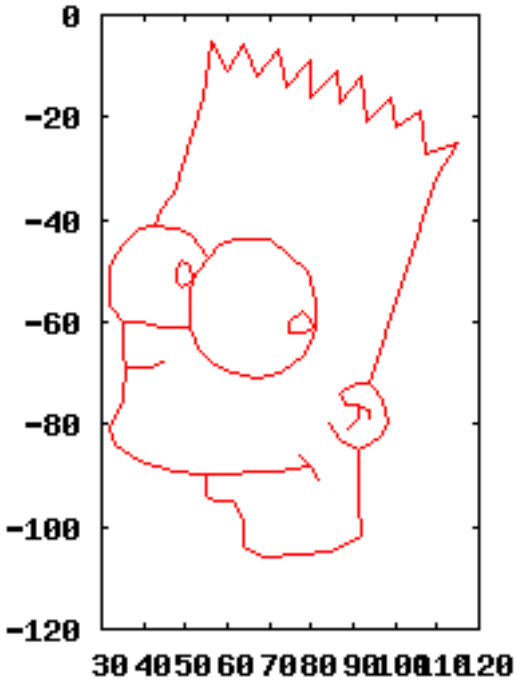
El apóstrofo que se coloca en la entrada `(%i17)` antes de la función `diff` sirve para devolver la expresión sin evaluarla, tal como aparece en el miembro izquierdo de la igualdad `(%o17)`. Una vez copiada y pegada la respuesta de la entrada `(%i18)` en una fuente LaTeX, su compilación dará como resultado la expresión

$$\frac{d}{d\,x}\,\left(\sqrt{\log x}\,\sin x^x\right)=\frac{\sin x^x}{2\,x\,\sqrt{\log x}}+x^x\,\sqrt{\log x}\,\left(\log x+1\right)\,\cos x^x$$

más fácil de interpretar por un humano.



a)



b)

Figura 11: Otros gráficos: *a)* superficie en 3D; *b)* puntos unidos por segmentos.

16. Estadística descriptiva

Aun siendo un sistema de cálculo simbólico, Maxima es capaz de llevar a cabo procesos numéricos y de procesamiento de datos. El paquete `descriptive` es el que nos puede ayudar en este tipo de tareas. Además, puesto que los datos procedentes de una muestra suelen almacenarse en un fichero de tipo texto, necesitaremos también del concurso de un segundo paquete que se encargará precisamente de leer las muestras para Maxima, nos referimos a `numericalio`.

Durante la instalación de Maxima se almacenan junto al fichero `descriptive.mac` tres muestras de datos: `pidigits.data`, `wind.data` y `biomed.data`, los cuales cargaremos en memoria con las funciones `read_list` y `read_matrix` del paquete `numericalio`.

Las muestras univariantes deben guardarse en listas, tal como se muestra a continuación

```
(%i1) s1:[3,1,4,1,5,9,2,6,5,3,5];
(%o1)       [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
```

y muestras multivariantes en matrices, como en

```
(%i2) s2:matrix([13.17, 9.29],[14.71, 16.88],[18.50, 16.88],
                [10.58,  6.63],[13.33, 13.25],[13.21, 8.12]);
                      [ 13.17  9.29  ]
                      [              ]
                      [ 14.71  16.88 ]
                      [              ]
                      [ 18.5   16.88 ]
(%o2)                 [              ]
                      [ 10.58  6.63  ]
                      [              ]
                      [ 13.33  13.25 ]
                      [              ]
                      [ 13.21  8.12  ]
```

En este caso, el número de columnas es igual a la dimensión de la variable aleatoria y el número de filas al tamaño muestral.

Los datos se pueden introducir manualmente, pero las muestras grandes se suelen guardar en ficheros de texto. Por ejemplo, el fichero `pidigits.data` contiene los 100 primeros dígitos del número π:

```
3
1
4
1
5
9
2
6
5
3 ...
```

Para cargar estos dígitos en Maxima,

```
(%i3) load("numericalio")$
(%i4) s1:read_list("ruta a/pidigits.data")$
(%i5) length(s1);
(%o5)                      100
```

El fichero `wind.data` contiene las velocidades medias del viento registradas diariamente en 5 estaciones meteorológicas en la República de Irlanda. Lo que sigue carga los datos,

```
(%i6) s2:read_matrix("ruta a/wind.data")$
(%i7) length(s2);
(%o7)                      100
(%i8) s2[%]; /* ultimo registro */
(%o8)         [3.58, 6.0, 4.58, 7.62, 11.25]
```

Algunas muestras contienen datos no numéricos. Como ejemplo, el fichero `biomed.data` contiene cuatro medidas sanguíneas tomadas en dos grupos de pacientes, `A` y `B`, de edades diferentes,

```
(%i9) s3:read_matrix("ruta a/biomed.data")$
(%i10) length(s3);
(%o10)                      100
(%i11) s3[1]; /* primer registro */
(%o11)        [A, 30, 167.0, 89.0, 25.6, 364]
```

El primer individuo pertenece al grupo `A`, tiene 30 años de edad y sus cuatro medidas sanguíneas fueron 167.0, 89.0, 25.6 y 364.

El paquete `descriptive` incluye dos funciones que permiten hacer recuentos de datos: `cfrec` y `dfrec`; la primera de ellas agrupa en intervalos los valores

de una lista de datos y hace el recuento de cuántos hay dentro de cada una de las clases,

```
(%i12) load("descriptive")$
(%i13) cfrec(s1,5);
(%o13) [[0, 1.8, 3.6, 5.4, 7.2, 9.0], [16, 24, 18, 17, 25]]
```

La primera lista contiene los límites de los intervalos de clase y la segunda los recuentos correspondientes: hay 16 dígitos dentro del intervalo $[0,1{,}8]$, eso es ceros y unos, 24 dígitos en $(1{,}8,3{,}6]$, es decir, doses y treses, etc. El segundo parámetro indica el número de clases deseadas, que por defecto es 10.

La función `dfrec` hace el recuento de las frecuencias absolutas en muestras discretas, tanto numéricas como categóricas. Su único argumento es una lista,

```
(%i14) dfrec(s1);
(%o14) [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
                        [8, 8, 12, 12, 10, 8, 9, 8, 12, 13]]
(%i15) dfrec(transpose(col(s3,1))[1]);
(%o15)               [[A, B], [35, 65]]
```

La primera lista devuelve los valores muestrales y la segunda sus frecuencias absolutas. Las instrucciones `? col` y `? transpose` ayudarán a comprender la última entrada.

Cuando las muestras tienen más datos de los que necesitamos, quizás porque sólo queramos analizar los datos de un subgrupo, hay que hacer uso de la función `subsample`. En cierto modo esta es una variante de la función `submatrix` de Maxima. El primer argumento es el nombre de la matriz de datos, el segundo es una expresión lógica sin evaluar (con apóstrofo `'`) y además admite opcionalmente argumentos adicionales con los números de las columnas deseadas. Su comportamiento se entenderá mejor con ejemplos,

```
(%i16) subsample(s2,'(%c[1]>18));
           [ 19.38  15.37  15.12  23.09  25.25 ]
           [                                    ]
           [ 18.29  18.66  19.08  26.08  27.63 ]
(%o16)     [                                    ]
           [ 20.25  21.46  19.95  27.71  23.38 ]
           [                                    ]
           [ 18.79  18.96  14.46  26.38  21.84 ]
```

Estos son los registros multivariantes en los cuales las velocidades del viento en la primera estación meteorológica fueron mayores que 18. Véase que en la

expresión lógica la i-ésima componente es referenciada como `%c[i]`. El símbolo `%c` se utiliza dentro de la función `subsample`, razón por la cual cuando se utiliza como variable categórica podemos confundir a Maxima. En el siguiente ejemplo, pedimos únicamente la primera, segunda y quinta componentes de aquellos registros con velocidades del viento mayores o iguales a 16 en la estación número 1 y menores que 25 nudos en la estación número 4,

```
(%i17) subsample(s2,'(%c[1]>=16 and %c[4]<25),1,2,5);
                  [ 19.38  15.37  25.25 ]
                  [                     ]
                  [ 17.33  14.67  19.58 ]
(%o17)            [                     ]
                  [ 16.92  13.21  21.21 ]
                  [                     ]
                  [ 17.25  18.46  23.87 ]
```

Aquí un ejemplo con las variables categóricas de `biomed.data`. Queremos los registros correspondientes a aquellos pacientes del grupo `B` mayores de 38 años,

```
(%i18) subsample(s3,'(%c[1]=B and %c[2]>38));
             [ B  39  28.0  102.3  17.1  146 ]
             [                               ]
             [ B  39  21.0  92.4   10.3  197 ]
             [                               ]
             [ B  39  23.0  111.5  10.0  133 ]
             [                               ]
             [ B  39  26.0  92.6   12.3  196 ]
(%o18)       [                               ]
             [ B  39  25.0  98.7   10.0  174 ]
             [                               ]
             [ B  39  21.0  93.2   5.9   181 ]
             [                               ]
             [ B  39  18.0  95.0   11.3  66  ]
             [                               ]
             [ B  39  39.0  88.5   7.6   168 ]
```

Probablemente, nuestro análisis estadístico se interesará únicamente por las mediciones sanguíneas,

```
(%i19) subsample(s3,'(%c[1]=B and %c[2]>38),3,4,5,6);
```

```
                  [ 28.0  102.3  17.1  146 ]
                  [                        ]
                  [ 21.0  92.4   10.3  197 ]
                  [                        ]
                  [ 23.0  111.5  10.0  133 ]
                  [                        ]
                  [ 26.0  92.6   12.3  196 ]
(%o19)            [                        ]
                  [ 25.0  98.7   10.0  174 ]
                  [                        ]
                  [ 21.0  93.2   5.9   181 ]
                  [                        ]
                  [ 18.0  95.0   11.3  66  ]
                  [                        ]
                  [ 39.0  88.5   7.6   168 ]
```

En cuanto al cálculo de parámetros muestrales, el paquete `descriptive` incluye, tanto para muestras univariantes como multivariantes, medias (`mean`), varianzas (`var`), desviaciones típicas (`std`), momentos centrales (`cmoment`) y no centrales (`ncmoment`), coeficientes de variación (`vc`), rangos (`range`), medianas (`median`), cuantiles (`quantile`), coeficientes de curtosis (`kurtosis`) y de asimetría (`skewness`), y otros. En los siguientes ejemplos, `s1` es una muestra univariante y `s2` multivariante:

```
(%i20) mean(s1);
                           471
(%o20)                     ---
                           100
(%i20) numer:true$   %;
(%o20)                     4.71
(%i21) mean(s2);  /* vector de medias */
(%o21)  [9.9485, 10.1607, 10.8685, 15.7166, 14.8441]
```

De ahora en adelante, todos los resultados se darán en notación decimal.

```
(%i22) var(s1);
(%o22)              8.425899999999999
(%i23) var(s2);  /* vector de varianzas */
(%o23) [17.22190675000001, 14.98773651, 15.47572875,
               32.17651044000001, 24.42307619000001]
```

```
(%i16) /* 1er y 3er cuartiles */
       [quantile(s1,1/4),quantile(s1,3/4)];
(%o16)                  [2.0, 7.25]

(%i24) range(s1);
(%o24)                       9
(%i25) range(s2);  /* vector de rangos */
(%o25)       [19.67, 20.96, 17.37, 24.38, 22.46]

(%i26) kurtosis(s1);

(%o26)               - 1.219988843897832
(%i27) kurtosis(s2); /* vector de coef. curtosis */
(%o27) [- .2715445622195385, 0.119998784429451,
       - .4275233490482866, - .6405361979019522,
       - .4952382132352935]
```

Un aspecto crucial en el análisis estadístico multivariante es la dependencia estocástica entre variables, para lo cual Maxima permite el cálculo de matrices de covarianzas (`cov`) y correlaciones (`cor`), así como de otras medidas de variabilidad global:

```
(%i28) fpprec:7$ /* cambia precision para mejorar salida */
(%i29) cov(s2); /* matriz de covarianzas */
       [ 17.22191  13.61811  14.37217  19.39624  15.42162 ]
       [                                                  ]
       [ 13.61811  14.98774  13.30448  15.15834  14.9711  ]
       [                                                  ]
(%o29) [ 14.37217  13.30448  15.47573  17.32544  16.18171 ]
       [                                                  ]
       [ 19.39624  15.15834  17.32544  32.17651  20.44685 ]
       [                                                  ]
       [ 15.42162  14.9711   16.18171  20.44685  24.42308 ]

(%i30) cor(s2); /* matriz de correlaciones */
       [   1.0     .8476339  .8803515  .8239624  .7519506 ]
       [                                                  ]
       [ .8476339    1.0     .8735834  .6902622  0.782502 ]
       [                                                  ]
(%o30) [ .8803515  .8735834    1.0     .7764065  .8323358 ]
```

```
[                                                          ]
[ .8239624   .6902622   .7764065    1.0      .7293848 ]
[                                                          ]
[ .7519506  0.782502   .8323358   .7293848    1.0     ]
```

La función `listvar` calcula algunas medidas globales de variación y `listdep` devuelve otras medidas de correlación; para más información, pídase ayuda a Maxima con el comando `describe` o con `?`.

Hay cuatro tipos de diagramas estadísticos programados en el paquete `descriptive`. La función `dataplot` permite la visualización directa de los datos muestrales, `histogram` se relaciona con el análisis de los patrones de las frecuencias en caso de observaciones continuas, `barsplot` con las frecuencias de datos categóricos o discretos y `boxplot` permite estudiar la propiedad homocedástica entre muestras. Todas estas funciones hacen uso del programa de representación gráfica `gnuplot` y admiten la posibilidad de dotarlas de opciones a fin de acomodar su comportamiento a nuestras necesidades. Para más información, hágase uso de la función `describe`.

17. Interpolación

El paquete `interpol` permite abordar el problema de la interpolación desde tres enfoques: lineal, polinomio de Lagrange y *splines* cúbicos.

A lo largo de esta sección vamos a suponer que disponemos de los valores empíricos de la siguiente tabla:

x	7	8	1	3	6
y	2	2	5	2	7

Nos planteamos en primer lugar el cálculo la función de interpolación lineal, para lo cual haremos uso de la función `linearinterpol`,

```
(%i1) load(interpol)$
(%i2) datos: [[7,2],[8,2],[1,5],[3,2],[6,7]]$
(%i3) linearinterpol(datos);
(%o3) - ((9 x - 39) charfun2(x, minf, 3)
 - 12 charfun2(x, 7, inf) + (30 x - 222) charfun2(x, 6, 7)
 + (18 - 10 x) charfun2(x, 3, 6))/6
(%i4) f(x):=''%;
(%o4) f(x) := - ((9 x - 39) charfun2(x, minf, 3)
 - 12 charfun2(x, 7, inf) + (30 x - 222) charfun2(x, 6, 7)
 + (18 - 10 x) charfun2(x, 3, 6))/6
```

Empezamos cargando el paquete que define las funciones de interpolación y a continuación introducimos los pares de datos en forma de lista. La función `linearinterpol` devuelve una expresión definida a trozos, en la que `charfun2(x,a,b)` devuelve 1 si el primer argumento pertence al intervalo $[a, b)$ y 0 en caso contrario. Por último, definimos cierta función `f` previa evaluación (dos comillas simples) de la expresión devuelta por `linearinterpol`. Esta función la podemos utilizar ahora tanto para interpolar como para extrapolar:

```
(%i5) map(f,[7.3,25/7,%pi]);
                      62    18 - 10 %pi
(%o5)             [2, --, - -----------]
                      21         6
(%i6) %,numer;
(%o6)   [2, 2.952380952380953, 2.235987755982988]
```

Unos comentarios antes de continuar. Los datos los hemos introducido como una lista de pares de números, pero también la función admite una matriz

de dos columnas o una lista de números, asignándole en este último caso las abscisas secuencialmente a partir de la unidad; además, la lista de pares de la variable `datos` no ha sido necesario ordenarla respecto de la primera coordenada, asunto del que ya se encarga Maxima por cuenta propia.

El polinomio de interpolación de Lagrange se calcula con la función `lagrange`; en el siguiente ejemplo le daremos a los datos un formato matricial y le indicaremos a Maxima que nos devuelva el polinomio con variable independiente `w`,

```
(%i7) datos2: matrix([7,2],[8,2],[1,5],[3,2],[6,7]);
                           [ 7  2 ]
                           [      ]
                           [ 8  2 ]
                           [      ]
(%o7)                      [ 1  5 ]
                           [      ]
                           [ 3  2 ]
                           [      ]
                           [ 6  7 ]
(%i8) lagrange(datos2,varname='w);
            4         3         2
        73 w  - 1402 w  + 8957 w  - 21152 w + 15624
(%o8)   ------------------------------------------
                             420
(%i9) g(w):=''%;
                    4         3         2
                73 w  - 1402 w  + 8957 w  - 21152 w + 15624
(%o9)  g(w) := ------------------------------------------
                                 420
(%i10) map(g,[7.3,25/7,%pi]), numer;
(%o10) [1.043464999999768, 5.567941928958183,
                                              2.89319655125692]
```

Disponemos en este punto de dos funciones de interpolación; representémoslas gráficamente junto con los datos empíricos,

```
(%i11) plot2d(['(f(x)),g(x),[discrete,datos]],[x,0,10],
          [gnuplot_curve_styles,
                   ["with lines",
                    "with lines",
```

```
                "with points pointsize 3"]],
        [gnuplot_preamble,
                "set size 0.6, 0.6;
                 set terminal png;
                 set out 'interpol1.png';
                 set nokey"])$
```

cuyo resultado se ve en el apartado *a)* de la Figura 12.

El método de los *splines* cúbicos consiste en calcular polinomios interpoladores de tercer grado entre dos puntos de referencia consecutivos, de manera que sus derivadas cumplan ciertas condiciones que aseguren una curva sin cambios bruscos de dirección. La función que ahora necesitamos es `cspline`,

```
(%i12) cspline(datos);
                3          2
(%o12) ((3477 x  - 10431 x  - 18273 x + 74547)
                                    3           2
 charfun2(x, minf, 3) + (- 15522 x  + 372528 x  - 2964702 x
 + 7842816) charfun2(x, 7, inf)
           3           2
 + (28290 x  - 547524 x  + 3475662 x - 7184700)
                                3          2
 charfun2(x, 6, 7) + (- 6574 x  + 80028 x  - 289650 x
 + 345924) charfun2(x, 3, 6))/9864
(%i13) s1(x):=''%$
(%i14) map(s1,[7.3,25/7,%pi]), numer;
(%o14) [1.438224452555094, 3.320503453379951,
                                          2.227405312429501]
```

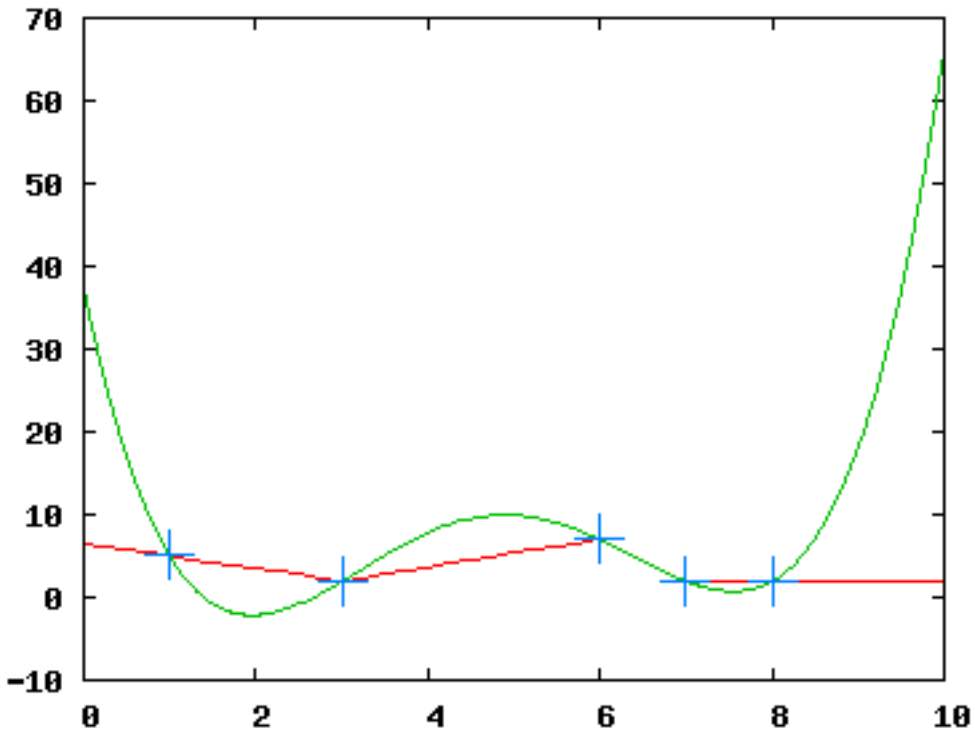
La función `cspline` admite, además de la opción `'varname` que ya se vió anteriormente, otras dos a las que se hace referencia con los símbolos `'d1` y `'dn`, que indican las primeras derivadas en las abscisas de los extremos; estos valores establecen las condiciones de contorno y con ellas Maxima calculará los valores de las segundas derivadas en estos mismos puntos extremos; en caso de no suministrarse, como en el anterior ejemplo, las segundas derivadas se igualan a cero. En el siguiente ejemplo hacemos uso de estas opciones,

```
(%i15) cspline(datos,'varname='z,d1=1,dn=0);
                3           2
(%o15) ((21051 z  - 130635 z  + 218421 z - 7317)
                                  3            2
```

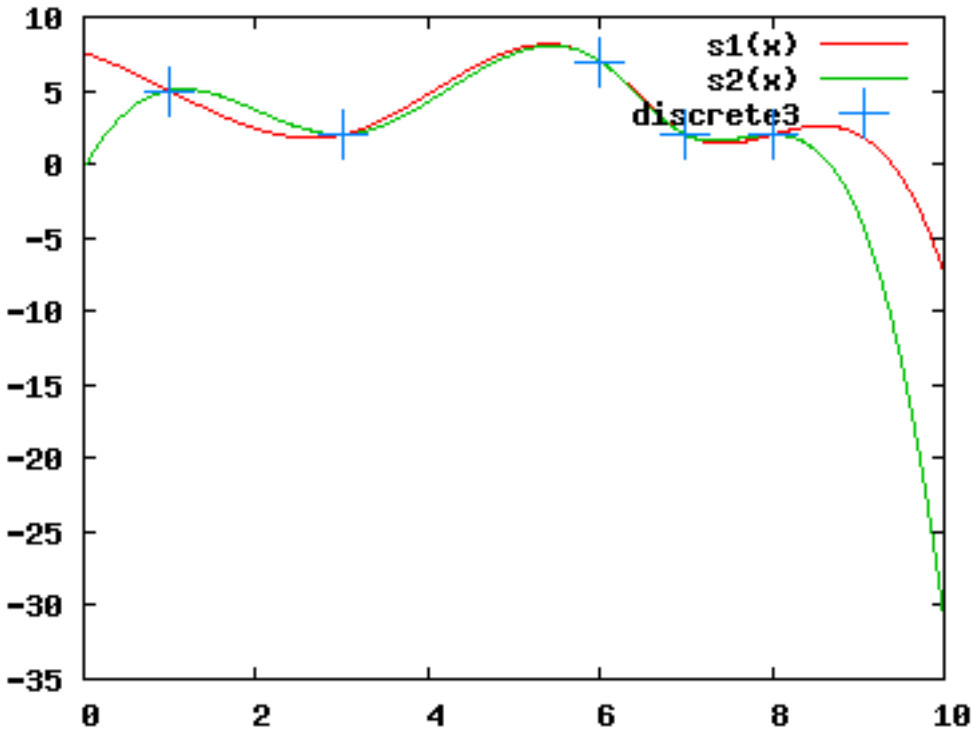
```
 charfun2(z, minf, 3) + (- 55908 z  + 1285884 z  - 9839808 z
 + 25087392) charfun2(z, 7, inf)
           3            2
 + (66204 z  - 1278468 z  + 8110656 z - 16797024)
                                3           2
 charfun2(z, 6, 7) + (- 16180 z  + 204444 z  - 786816 z
 + 997920) charfun2(z, 3, 6))/20304
(%i16) s2(z):=''%$
(%i17) map(s2,[7.3,25/7,%pi]), numer;
(%o17) [1.595228723404633, 2.881415315197325,
                                          2.076658794432381]
```

Con esto hemos obtenido dos interpoladores distintos por el método de los *splines* cúbicos; con el siguiente código pedimos su representación gráfica, cuyo resultado se observa en el apartado *b)* de la Figura 12.

```
plot2d(['(s1(x)),'(s2(x)),[discrete,datos]],[x,0,10],
           [gnuplot_curve_styles,
                    ["with lines",
                     "with lines",
                     "with points pointsize 3"]],
           [gnuplot_preamble,
                     "set size 0.6, 0.6;
                      set terminal png;
                      set out 'interpol2.png'"])$
```



a)



b)

Figura 12: Interpolación: *a)* lineal y de Lagrange; *b) Splines* cúbicos.

18. Listas

Las listas son objetos muy potentes 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*). Veamos cómo podemos ver la representación interna, esto es en Lisp, de una sencilla expresión tal como $1 + 3a$,

```
(%i1) :lisp #$1+3*a$
((MPLUS SIMP) 1 ((MTIMES SIMP) 3 $a))
```

Nótese que el formato general es de la forma `:lisp #$expr$`, siendo `expr` una expresión cualquiera en Maxima.

Pero al nivel del usuario que no está interesado en las interioridades de Maxima, también se puede trabajar con listas como las definidas a continuación, simpre encerradas entre corchetes,

```
(%i2) q:[b,5,a,d,1,3,7]$
(%i3) r:[1,[a,3],sqrt(3)/2,"Don Quijote"];
                          sqrt(3)
(%o3)        [1, [a, 3], -------, Don Quijote]
                            2
```

Vemos que los elementos de una lista pueden a su vez ser también listas, expresiones matemáticas o cadenas de caracteres incluidas entre comillas dobles, lo que puede ser aprovechado para la construcción y manipulación de estructuras más o menos complejas. Extraigamos a continuación alguna información de las listas anteriores,

```
(%i4) listp(r); /* es r una lista? */
(%o4)                         true
(%i5) first(r); /* primer elemento */
(%o5)                          1
(%i6) second(r); /* segundo elemento */
(%o6)                        [a, 3]
(%i7) third(r); /* ...hasta tenth */
                           sqrt(3)
(%o7)                      -------
                             2
(%i8) last(r); /* el ultimo de la lista */
(%o8)                   Don Quijote
```

```
(%i9) rest(r); /* todos menos el primero */
                         sqrt(3)
(%o9)          [[a, 3], -------, Don Quijote]
                           2
(%i10) part(r,3); /* pido el que quiero */
                          sqrt(3)
(%o10)                    -------
                            2
(%i11) length(r); /* cuantos hay? */
(%o11)                       4
(%i12) reverse(r); /* le damos la vuelta */
                           sqrt(3)
(%o12)       [Don Quijote, -------, [a, 3], 1]
                             2
(%i13) member(a,r); /* es a un elemento?*/
(%o13)                      false
(%i14) member([a,3],r); /* lo es [a,3]? */
(%o14)                       true
(%i15) sort(q); /* ordeno */
(%o15)             [1, 3, 5, 7, a, b, d]
(%i16) delete([a,3],r); /* borro elemento */
                     sqrt(3)
(%o16)           [1, -------, Don Quijote]
                       2
```

Nótese que en todo este tiempo la lista `r` no se ha visto alterada,

```
(%i17) r;
                         sqrt(3)
(%o18)       [1, [a, 3], -------, Don Quijote]
                           2
```

Algunas funciones de Maxima permiten añadir nuevos elementos a una lista, tanto al principio como al final,

```
(%i19) cons(1+2,q);
(%o19)            [3, b, 5, a, d, 1, 3, 7]
(%i20) endcons(x,q);
(%o20)            [b, 5, a, d, 1, 3, 7, x]
```

En este ejemplo observamos también que la lista `q` no ha cambiado; si lo que queremos es actualizar su contenido,

```
(%i26) q: endcons(x,cons(1+2,q))$
(%i27) q;
(%o27)          [3, b, 5, a, d, 1, 3, 7, x]
```

Es posible unir dos listas,

```
(%i28) append(r,q);
                    sqrt(3)
(%o28) [1, [a, 3], -------, Don Quijote, 3, b, 5, a, d, 1,
                       2

                                                      3, 7, x]
```

Cuando los elementos de una lista van a obedecer un cierto criterio de construcción, podemos utilizar la función `makelist`,

```
(%i29) s:makelist(2+k*2,k,0,10);
(%o29)    [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]
```

donde le hemos indicado a Maxima que nos construya una lista con elementos de la forma `2+2*k`, de modo que `k` tome valores enteros de 0 a 10.

La función `apply` permite suministrar a otra función todos los elementos de una lista como argumentos, así podemos sumar o multiplicar todos los elementos de la lista `s` recién creada,

```
(%i30) apply("+",s);
(%o30)                       132
(%i31) apply("*",s);
(%o31)                  81749606400
```

aunque estas dos operaciones hubiese sido quizás mejor haberlas realizado con las funciones `sum` y `product`.

A veces interesará aplicar una misma función a varios elementos de una lista de forma independiente, para lo que haremos uso de `map`; a continuación un ejemplo de cálculo de factoriales y otro trigonométrico,

```
(%i32) map("!",s);
(%o32) [2, 24, 720, 40320, 3628800, 479001600, 87178291200,
20922789888000, 6402373705728000, 2432902008176640000,
112400072777607680000]
(%i33) map(sin,s);
(%o33) [sin(2), sin(4), sin(6), sin(8), sin(10), sin(12),
                sin(14), sin(16), sin(18), sin(20), sin(22)]
```

Por último, las listas también pueden ser utilizadas en operaciones aritméticas,

```
(%i34) [1,2,3]+[a,b,c];
(%o34)             [a + 1, b + 2, c + 3]
(%i35) [1,2,3]*[a,b,c];
(%o35)                  [a, 2 b, 3 c]
(%i36) [1,2,3]/[a,b,c];
                          1  2  3
(%o36)                   [-, -, -]
                          a  b  c
(%i37) [1,2,3]-[a,b,c];
(%o37)             [1 - a, 2 - b, 3 - c]
(%i38) [1,2,3].[a,b,c]; /* producto escalar */
(%o38)                 3 c + 2 b + a
(%i39) [a,b,c]^3;
                          3   3   3
(%o39)                   [a , b , c ]
(%i40) 3^[a,b,c];
                          a   b   c
(%o40)                   [3 , 3 , 3 ]
```

Para que estas operaciones puedan realizarse sin problemas, la variable global `listarith` debe tomar el valor `true`, en caso contrario el resultado será bien distinto,

```
(%i41) listarith:false$
(%i42) [1,2,3]+[4,5,6];
(%o42)             [4, 5, 6] + [1, 2, 3]
(%i43) listarith:true$
```

Como ya se vió al comienzo de esta sección, una lista puede ser elemento de otra lista, si queremos deshacer todas las listas interiores para que sus elementos pasen a formar parte de la exterior,

```
(%i44) flatten([1,[a,b],2,3,[c,[d,e]]]);
(%o44)            [1, a, b, 2, 3, c, d, e]
```

19. Manipulación de cadenas

Una cadena se construye escribiendo un texto o cualquier otra expresión entre commillas dobles. El paquete `stringproc` dispone de varias funciones para procesar cadenas. Vamos a comenzar cargando el paquete y haciendo algunas construcciones básicas,

```
(%i1) load("stringproc")$
(%i2) xx: "Los ";
(%o2)                        Los
(%i3) yy: " cerditos";
(%o3)                      cerditos
(%i4) zz: sconc(xx,sqrt(9),yy);
(%o4)                  Los 3 cerditos
```

la función `sconc` admite varios parámetros, los cuales evalúa y luego concatena convirtiéndolos en una cadena.

La función `stringp` nos indica si su argumento es o no una cadena,

```
(%i5) stringp(zz);
(%o5)                         true
```

Para saber el número de caracteres de una cadena haremos uso de `slength`

```
(%i21) slength(zz);
(%o21)                          14
```

Una vez transformada en cadena una expresión de Maxima, deja de ser evaluable,

```
(%i6) sconc(sqrt(25));
(%o6)                           5
(%i7) % + 2;
(%o7)                       5 + 2
(%i8) stringp(%);
(%o8)                       false
```

el resultado no se simplifica porque uno de los sumandos es una cadena y no se reconoce como cantidad numérica.

La función `charat` devuelve el carácter que ocupa una posición determinada,

```
(%i9) charat(zz,7);
(%o9)                         c
```

También podemos solicitar que Maxima nos devuelva una lista con todos los caracteres o palabras que forman una cadena,

```
(%i10) charlist(zz);
(%o10)   [L, o, s,  , 3,  , c, e, r, d, i, t, o, s]
(%i11) tokens(zz);
(%o11)                [Los, 3, cerditos]
```

O podemos transformar letras minúsculas en mayúsculas y viceversa, todas las letras o sólo un rango de ellas,

```
(%i12) /* todo a mayúsculas */
       supcase(zz);
(%o12)                  LOS 3 CERDITOS
(%i13) /* a minúsculas entre la 9a y 12a */
       sdowncase(%,9,12);
(%o13)                  LOS 3 CErdiTOS
(%i14) /* a minúsculas de la 13a hasta el final */
       sdowncase(%,13);
(%o14)                  LOS 3 CErdiTos
(%i15) /* todo a minúsculas */
       sdowncase(%);
(%o15)                  los 3 cerditos
(%i16) /* a mayúsculas entre la 1a y 2a */
       supcase(%,1,2);
(%o16)                  Los 3 cerditos
```

Para comparar si dos cadenas son iguales disponemos de la función `sequal`

```
(%i17) tt:zz;
(%o17)                  Los 3 cerditos
(%i18) sequal(tt,zz);
(%o18)                       true
(%i19) tt: supcase(zz);
(%o19)                  LOS 3 CERDITOS
(%i20) sequal(tt,zz);
(%o20)                      false
```

Podemos buscar subcadenas dentro de otras cadenas mediante la función `ssearch`; los siguientes ejemplos pretenden aclarar su uso,

```
(%i21) zz;
(%o21)                 Los 3 cerditos
(%i22) /* busca la subcadena "cer" */
       ssearch("cer",zz);
(%o22)                       7
(%i23) /* busca la subcadena "Cer" */
       ssearch("Cer",zz);
(%o23)                     false
(%i24) /* busca "Cer" ignorando tamaño */
       ssearch("Cer",zz,'sequalignore);
(%o24)                       7
(%i25) /* busca sólo entre caracteres 1 y 5 */
       ssearch("Cer",zz,'sequalignore,1,5);
(%o25)                     false
```

Igual que buscamos subcadenas dentro de otra cadena, también podemos hacer sustituciones; en este caso la función a utilizar es `ssubst`,

```
(%i26) ssubst("mosqueteros","cerditos",zz);
(%o26)               Los 3 mosqueteros
```

Esta función tiene como mínimo tres argumentos, el primero indica la nueva cadena a introducir, el segundo es la subcadena a ser sustituída y el tercero la cadena en la que hay que realizar la sustitución. Es posible utilizar esta función con más argumentos; como siempre, para más información, `? ssubst`.

Con la función `substring` podemos obtener una subcadena a partir de otra, para lo cual debemos indicar los extremos de la subcadena,

```
(%i27) substring(zz,7);
(%o27)                     cerditos
(%i28) substring(zz,3,8);
(%o28)                      s 3 c
```

Como se ve, el segundo argumento indica la posición a partir de la cual queremos la subcadena y el tercero la última posición; si no se incluye el tercer argumento, se entiende que la subcadena se extiende hasta el final de la de referencia.

El paquete `eval_string` dispone de dos funciones que de algún modo se relacionan también con el procesamiento de cadenas y que ambas admiten como argumento una cadena que guarda una expresión sintácticamente correcta de Maxima; la función `eval_string` analiza la expresión y la evalúa, en tanto

que `parse_string` la analiza pero no la evalúa, limitándose a transformarla a una expresión nominal de Maxima,

```
(%i29) /* cargamos el paquete 'eval_string' */
       load(eval_string)$
(%i30) /* una cadena con una expresión de Maxima */
       expr: "integrate(x^5,x)";
(%o30)                integrate(x^5,x)
(%i31) /* se evalúa la cadena */
       eval_string(expr);
                              6
                             x
(%o31)                       --
                             6
(%i32) /* se transforma en una expresión nominal */
       parse_string(expr);
                                  5
(%o32)                integrate(x , x)
(%i33) ''%;
                              6
                             x
(%o33)                       --
                             6
```

Véase que el resultado de `parse_string` ya no es una cadena, sino una expresión correcta de Maxima sin evaluar, cuya evaluación se solicita a continuación con la doble comilla simple.

20. Operaciones con conjuntos

Se define a continuación un conjunto mediante la función `set`,

```
(%i1) c1:set(a,[2,k],b,sqrt(2),a,set(a,b),
             3,"Sancho",set(),b,sqrt(2),a);
(%o1)  {3, sqrt(2), {}, [2, k], a, {a, b}, b, Sancho}
```

Como se ve, se admiten objetos de muy diversa naturaleza como elementos de un conjunto: números, expresiones, el conjunto vacío (`{}`), listas, otros conjuntos o cadenas de caracteres. Cuando se trabaja con listas, puede ser de utilidad considerar sus componentes como elementos de un conjunto, luego se necesita una función que nos transforme una lista en conjunto,

```
(%i2) [[2,k],sqrt(2),set(b,a),[k,2],"Panza"];
(%o2)     [[2, k], sqrt(2), {a, b}, [k, 2], Panza]
(%i3) c2:setify(%);
(%o3)     {sqrt(2), [2, k], {a, b}, [k, 2], Panza}
```

el cambio en la naturaleza de estas dos colecciones de objetos se aprecia en la presencia de llaves frente a los corchetes. De igual manera, podemos transformar un conjunto en lista,

```
(%i4) listify(%o1);
(%o4)  [3, sqrt(2), {}, [2, k], a, {a, b}, b, Sancho]
```

Comprobemos de paso que `{}` representa al conjunto vacío,

```
(%i5) emptyp(%[3]);
(%o5)                         true
```

Recuérdese que `%` sustituye a la última respuesta dada por Maxima, que en este caso había sido una lista, por lo que `%[3]` hace referencia a su tercera componente.

Para comprobar si un cierto objeto forma parte de un conjunto hacemos uso de la instrucción `elementp`,

```
(%i6) elementp(sqrt(2),c1);
(%o6)                         true
```

Es posible extraer un elemento de un conjunto y luego añadirle otro distinto

```
(%i7) c1: disjoin(sqrt(2),c1);  /* sqrt(2) fuera */
(%o7)     {3, {}, [2, k], a, {a, b}, b, Sancho}
(%i8) c1: adjoin(sqrt(3),c1);  /* sqrt(3) dentro */
(%o8) {3, sqrt(3), {}, [2, k], a, {a, b}, b, Sancho}
```

La sustitución que se acaba de realizar se pudo haber hecho con la función `subst`,

```
(%i9) /* nuevamente a poner sqrt(2) */
      subst(sqrt(2),sqrt(3),c1);
(%o9) {3, sqrt(2), {}, [2, k], a, {a, b}, b, Sancho}
```

La comprobación de si un conjunto es subconjunto de otro se hace con la fución `subsetp`,

```
(%i10) subsetp(set([k,2],"Panza"),c2);
(%o10)                       true
```

A continuación algunos ejemplos de operaciones con conjuntos,

```
(%i11) union(c1,c2);
(%o11) {3, sqrt(2), sqrt(3), {}, [2, k], a, {a, b}, b,
                                        [k, 2], Panza, Sancho}
(%i12) intersection(c1,c2);
(%o12)                {[2, k], {a, b}}
(%i13) setdifference(c1,c2);
(%o13)         {3, sqrt(3), {}, a, b, Sancho}
(%i14) cardinality(%);
(%o14)                        6
```

Vemos aquí también cómo pedir el cardinal de un conjunto.

Igual que se ha visto como aplicar una función a todos los elementos de una lista, podemos hacer lo mismo con los de un conjunto,

```
(%i15) map(sin,set(1,2,3,4,5));
(%o15)    {sin(1), sin(2), sin(3), sin(4), sin(5)}
```

Por último ya, el producto cartesiano de tres conjuntos,

```
(%i16) cartesian_product(set(1,2),set(a,b,c),set(x,y));
(%o16) {[1, a, x], [1, a, y], [1, b, x], [1, b, y],
[1, c, x], [1, c, y], [2, a, x], [2, a, y], [2, b, x],
[2, b, y], [2, c, x], [2, c, y]}
```

21. Operaciones lógicas

Como cualquier otro intérprete o entorno de programación, Maxima dispone también de la capacidad de evaluar predicados lógicos, aquéllos que pueden ser o bien verdaderos (`true`), o bien falsos (`false`). Los predicados más simples son funciones que devuelven uno de estos dos valores de verdad; ejemplos de ellos ya han aparecido en secciones anteriores,

```
(%i1) listp([[1,2],[a,b]]);
(%o1)                         true
(%i2) matrixp([[1,2],[a,b]]);
(%o2)                        false
(%i3) evenp(2^3);
(%o3)                         true
```

Puesto que Maxima es un entorno de procesamiento matemático, es frecuente tener que comparar dos cantidades. Los operadores de comparación se resumen en la siguiente tabla,

=	...igual que...
#	...diferente de...
>	...mayor que...
<	...menor que...
>=	...mayor o igual que...
=<	...menor o igual que...

Si se quiere saber si un número es menor que otro podemos hacer uso de la función `is`,

```
(%i4) is(sqrt(7895)<85);
(%o4)                        false
(%i5) is(4^12#16777216);
(%o5)                        false
```

En el primer caso comprobamos si $\sqrt{7895}$ es estrictamente menor que 85 y en el segundo si la potencia 4^{12} es diferente de 16777216.

Junto con estos predicados simples, los conectores `and`, `or` y `not` permiten construir formas más complejas. La tabla adjunta resume el comportamiento de estos operadores,

p	q	p and q	p or q	not p
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

A la hora de escribir expresiones lógicas conviene tener en cuenta el orden de precedencia; primero se evalúa `not`, a continuación `and` y, finalmente, `or`; en caso de duda siempre se puede recurrir a los paréntesis. Un ejemplo,

```
(%i6) is(not 3<4 or 5<5);
(%o6)                         false
```

Aquí, primero se evalúa `not 3<4`, que da como resultado `false`; a continuación, se evalúa si es cierto o no que `5<5`, lo cual es también falso, finalmente se obtiene el resultado correspondiente a `false or false`.

El lector debería ser capaz de justificar las siguientes respuestas,

```
(%i7) is(3#4 and not 7<=2);
(%o7)                          true
(%i8) is((5<8 or 8<2) and oddp(3) and evenp(sqrt(16)));
(%o8)                          true
```

22. Programación en Maxima

Con vistas a la optimización de tiempo y esfuerzo será interesante poder definir de una sola vez nuestras propias funciones y poder luego reutilizarlas cuantas veces sea necesario.

La programación de funciones requiere de ciertas sentencias de control que son comunes, con más o menos matices en su sintaxis, en todos los lenguajes de programación.

Un elemento imprescindible en el control del flujo es la sentencia condicional `if`, cuya estructura es

```
if <cond> then <expr1> else <expr2>
```

tal como en

```
(%i1) x:30!$ y:exp(30)$
(%i3) if (x>y) then 0 else 1;
(%o3)                         0
```

La condición `<cond>` es una expresión lógica que admite los operadores `and`, `or` y `not`, siendo sus argumentos predicados lógicos, cuyo valor sólo puede ser *verdadero* o *falso*,

Otra sentencia que nunca falta en un lenguaje de programación es la que controla las iteraciones y los bucles. Maxima ofrece aquí varias posibilidades. En primer lugar,

```
for <var>:<val1> step <val2> thru <val3> do <expr>
```

El siguiente ejemplo escribe, haciendo uso de la función `print`, los cinco primeros cubos enteros positivos,

```
(%i4) for i:1 thru 5 do print(i^3);
1
8
27
64
125
(%o4)                         done
```

Otra posibilidad de controlar las iteraciones es con la versión

```
for <var>:<val1> step <val2> while <cond> do <expr>
```

que en el siguiente ejemplo se utiliza para calcular las quintas potencias de todos los números impares menores que 20; nótese cómo tras la sentencia `do` se pueden escribir varias expresiones separadas por comas (`,`) y encerradas entre paréntesis,

```
(%i5) for i:1 step 2 while i<20 do(j:i^5,print(j));
1
243
3125
16807
59049
161051
371293
759375
1419857
2476099
(%o5)                         done
```

También hay una versión de la sentencia `for` muy a propósito para trabajar con listas,

```
for <var> in <lista> do <expr>
```

como se muestra en el siguiente ejemplo, donde se imprimen todos los elementos de una muestra simulada de números aleatorios, aumentados en una unidad,

```
(%i7) m:makelist(random(4),i,1,10);
(%o7)          [2, 0, 3, 2, 2, 3, 0, 1, 1, 3]
(%i8) for i in m do print(i+1);
3
1
4
3
3
4
1
2
2
4
(%o8)                         done
```

En general, la definición de una nueva función en Maxima tiene la estructura

```
f(<arg1>,<arg2>,...):=<expr>
```

donde `<argi>` son los argumentos y `<expr>` es una expresión sintácticamente válida. Por ejemplo, ya que Maxima no dispone de la función logaritmo en base arbitraria, la podemos definir por nuestra cuenta,

```
(%i9) logb(x,b):=log(x)/log(b)$
(%i10) logb(234,10);
                          log(234)
(%o10)                    --------
                          log(10)
(%i11) %,numer;
(%o11)                    2.3692157
```

En ocasiones, la función es lo suficientemente compleja como para necesitar tanto de variables locales que guarden valores temporales, como de expresiones que los calculen; en tales casos habrá que echar mano del entorno de boque con la instrucción `block`, cuya estructura es

```
f(<arg1>,<arg2>,...):=block([<varloc1>,<varloc2>,...],
   <expr1>,
   <expr2>,
     ....
   <exprm> );
```

siendo el resultado devuelto el de la última expresión evaluada (`<exprm>`). Las variables locales a las que se ha hecho referencia se declaran entre corchetes (`<varloci>`) dentro del bloque, donde pueden ser inicializadas y cuya vida se extiende durante el tiempo que dure el cómputo del bloque; además, si una de estas variables temporales se llama igual que otra global de la sesión de Maxima, no interferirá con ella y cualquier referencia a la variable se considerará que es a la local, y en caso de que ésta no esté declarada, la referencia será a la externa a la función. A continuación, un ejemplo en el que se define una función que calcula la media de una lista de números,

```
(%i12) media(lista):=block([n:length(lista),suma],
   if not listp(lista)
      then return("Ojo: no es una lista"),
   suma: sum(lista[k],k,1,n),
```

```
   suma/n )$
(%i13) media([45,25,87,23,65,31]);
(%o13)                          46
(%i14) media([[2,3],[6,3],[4,7],[2,6]]);
                              7  19
(%o14)                       [-, --]
                              2  4
(%i15) media([1/3,sqrt(5),logb(5,2),x]);
                         log(5)             1
                    x + ------ + sqrt(5) + -
                         log(2)             3
(%o15)              ------------------------
                                4
(%i16) solve(%=10,x);  /* cuanto vale x para una media de 10? */
                  3 log(5) + (3 sqrt(5) - 119) log(2)
(%o16)  [x = - ----------------------------------]
                               3 log(2)
```

Este último cálculo no tiene nada que ver con lo que se comenta en esta sección, pero muestra cómo integrar nuestra función en una sesión rutinaria. Examinando el código de la función `media` reparamos en la presencia de la instrucción `return`, que es una forma alternativa de salir del contexto marcado por `block`; en este caso, el resultado que devuelve la función es el indicado por el argumento de `return`, es decir la cadena `"Ojo: no es una lista"`. También se observa que se declaran dos variables locales, `n` y `suma`, asignándole a la primera el número de elementos de la lista; junto con éstas existe otra variable local, la `k` de la instrucción `sum`, que sólo está activa durante el cálculo de esta suma, no siendo necesario declararla en todo el contexto del bloque.

A veces es necesario definir funciones cuyo número de argumentos no se conoce a priori. Por ejemplo, tal como está programada la función `gcd` en Maxima, la que calcula el máximo común divisor, no admite más de dos argumentos; podemos suplir esta carencia diseñando una función, que llamaremos `mcd`, y que admita un número arbitrario de números,

```
(%i17) mcd(a,b,[c]):=block([n],
   n: length(c),
   r: gcd(a,b),
   if n>0 then
      for i in c do r: gcd(r,i),
   r )$
```

```
(%i18) mcd(120,300,480,825);
(%o18)                        15
```

Por último ya, se añade un ejemplo en el que se define una función haciendo uso de algunas de las sentencias comentadas en esta sección. Como se ve, a la función se le asigna el nombre de `transafin` y su cometido es el de aplicar una transformación afín a una lista de puntos,

```
/*Asi se escriben los comentarios. */
/*pts debe ser una lista de pares: */
/* [[x1,y1],[x2,y2],[x3,y3],...]   */
transafin(pts, a, b, c, d, e, f):=
  block([pts2, n, i, x, y],
    pts2: copylist(pts),
    n: length(pts2),
    for i:1 thru n do(
      x: pts2[i][1],
      y: pts2[i][2],
      pts2[i][1]: a*x+b*y+c,
      pts2[i][2]: d*x+e*y+f ),
    return(pts2) )$
```

La función `transafin`, y cualquier otra dentro de esta sección, se puede escribir en Maxima y luego llamarla para que realice su cometido; sin embargo, quizás sea mejor escribirla directamente con un editor de texto cualquiera y guardarla en un archivo, pongamos de nombre `tf.mac`, para hacer uso de ella en el futuro; así, una vez dentro de Maxima ejecutaríamos la instrucción `batch("tf.mac")` y luego haríamos

```
(%i19) cuadrado:[[0,0],[1,0],[1,1],[0,1]]$
(%i20) transafin(cuadrado,-1,0,-1,0,1,1);
(%o20)      [[- 1, 1], [- 2, 1], [- 2, 2], [- 1, 2]]
```

obteniendo así el resultado de aplicarle a los vértices del cuadrado unidad una simetría respecto del eje de ordenadas seguida de una traslación a lo largo del vector $\vec{v} = (-1, 1)$.

Índice alfabético

; (punto y coma), 11
, (coma), 23, 96
$, 12
', 69
", 13
%, 13
%ix, 12
%ox, 12
%e, 13
%i, 13
%pi, 13
%gamma, 13
%Rx, 31
+, 16, 36, 85, 86
-, 16, 36, 86
*, 16, 37, 85, 86
., 36
/, 16, 36, 86
^ , 16, 36, 86
**, 16
^ ^ , 35
., 86
=, 93
#, 93
>, 93
<, 93
>=, 93
=<, 93
:=, 32, 43, 97
:lisp, 83
"...", 83
/*...*/, 17
[...], 83

abs, 20, 41
acos, 41
acosh, 41
addcol, 33
addrow, 33
adjoion, 91
airy, 43
algsys, 28, 29
allroots, 29
and, 93, 95
append, 85
apply, 85
asin, 41
asinh, 41
atan, 41
atan2, 41
atanh, 41
atvalue, 56

B, 16
barsplot, 77
bc2, 55
bessel, 43
beta, 41
bfloat, 16
binomial, 41
block, 97
boxplot, 77

cardinality, 92
cartesian_product, 92
cfrec, 72
charat, 87
charfun2, 78
charlist, 88
charpoly, 37
cmoment, 75
col, 35

conjugate, 21
cons, 84
cor, 76
cos, 41
cosh, 41
cot, 41
cov, 76
csc, 41
cspline, 80

dataplot, 77
del, 48
delete, 83
denom, 25
dependencies, 47
depends, 47
describe, 43
descriptive, 71
desolve, 55
determinant, 37
detout, 37
dfrec, 72
diagmatrix, 34
diff, 46, 53
disjoin, 91
distrib, 58
divisors, 17

E, 16
eigenvalues, 38
eigenvectors, 39
elementp, 91
eliminate, 30
elliptic, 43
emptyp, 91
endcons, 84
entermatrix, 32
erf, 41, 43
ev, 15, 23
eval_string, 89
evenp, 17
exp, 41
expand, 22, 24

factor, 17, 24, 26
false, 93
first, 83
flatten, 86
for, 95
fpprec, 16
fullratsimp, 24

gamma, 41
gauss, 58
gcd, 25, 98
genfact, 41
genmatrix, 32
gnuplot_preamble, 66

histogram, 77

i, 19
ic1, 53
ic2, 55
ident, 34
if-then-else, 95
ilt, 51
imagpart, 20
inchar, 13
inf, 45
integrate, 50
interpol, 78
intersection, 92
invert, 37
is, 93

kill, 12
kurtosis, 75

lagrange, 79
laplace, 51

last, 83
lcm, 25
length, 83
lhs, 28
limit, 41, 45
linearinterpol, 78
listarith, 86
listdep, 77
listify, 91
listp, 83
listvar, 77
load, 14
log, 41

make_random_state, 57
makelist, 57, 85
map, 85, 92
matrix, 32
matrixmap, 39
matrixp, 35
max, 41
mean, 75
median, 75
member, 83
min, 41
minf, 45
minor, 39
minus, 45

ncmoment, 75
negative, 50
not, 93, 95
num, 25
numer, 16
numericalio, 71

oddp, 17
ode2, 53
or, 93, 95
outchar, 13

parse_string, 90
part, 27, 83
plot2d, 60
plot3d, 64
plot_options, 64
plus, 45
polarform, 20
positive, 50
primep, 17
print, 95
product, 85

quantile, 75
quit, 14
quotient, 18

random, 57
range, 75
rank, 39
ratsimp, 24, 52
rbinomial, 58
read_list, 71
read_matrix, 71
realonly, 28
realpart, 20
rectform, 19
remainder, 18
remove, 47
rest, 83
return, 98
reverse, 83
rexp, 58
rhs, 28
rhypergeo, 58
row, 35
rpoisson, 58
rweibull, 58

save, 14
sconc, 87

sdowncase, 88
sec, 41
second, 83
sequal, 88
set, 91
set_plot_option, 66
set_random_state, 57
setdifference, 92
setify, 91
showtime, 17
signum, 41
sin, 41
sinh, 41
skewness, 75
slength, 87
solve, 27
sort, 83
sqrt, 12, 41
ssearch, 88
ssubst, 89
std, 75
stringp, 87
stringproc, 87
submatrix, 33, 73
subsample, 73
subsetp, 92
subst, 15, 23, 24, 92
substring, 89
sum, 85
supcase, 88

tan, 41
tanh, 41
taylor, 49
tex, 69
third, 83
tokens, 88
transpose, 37
true, 93

und, 45
union, 92
uniteigenvectors, 39

var, 75
vc, 75

zero, 50
zeromatrix, 34