Primeros pasos en Maxima

Mario Rodríguez Riotorto

www.telefonica.net/web2/biomates

11 de marzo de 2008

Índice

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 acercan a él.

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

Su nombre original fue Macsyma (*MAC's SYmbolic MAnipulation System*, donde MAC, *Machine Aided Cognition*, era el nombre del *Laboratory for Computer Science* del MIT durante la fase inicial del proyecto Macsyma). Se desarrolló en estos laboratorios a partir del año 1969 con fondos aportados por varias agencias gubernamentales norteamericanas (*National Aeronautics and Space Administration*, *Office of Naval Research*, *U.S. Department of Energy* y *U.S. Air Force*).

El concepto y la organización interna del programa están basados en la tesis doctoral que Joel Moses elaboró en el MIT sobre integración simbólica. Según Marvin Minsky, director de esta tesis, Macsyma pretendía automatizar las manipulaciones simbólicas que realizaban los matemáticos, a fin de entender la capacidad de los ordenadores para actuar de forma inteligente.

El año 1982 es clave. El MIT transfiere una copia de Macsyma a la empresa Symbolics Inc. para su explotación económica, haciendo el código propietario, y otra al Departamento de Energía, copia ésta que será conocida con el nombre de DOE-Macsyma. En 1992 la versión comercial de Macsyma sería adquirida 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 en el propio Macsyma.

Pero ocurrieron dos historias paralelas. Desde el año 1982, y hasta su fallecimiento en el 2001, William Schelter de 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 había conseguido 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 declinando. Actualmente, el proyecto está siendo mantenido 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 la lista de correo

http://maxima.sourceforge.net/maximalist.html

Además, desde el año 2006, se ha activado una lista de correos para usuarios de habla hispana en

http://sourceforge.net/mailarchive/forum.php?forum_id=50322

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,

http://maxima.sourceforge.net

Uno de los aspectos más relevantes de este programa es su naturaleza libre; la licencia GPL en la que se distribuye brinda al usuario ciertas libertades:

- libertad para utilizarlo,
- libertad para modificarlo y adaptarlo a sus propias necesidades,
- libertad para distribuirlo,
- libertad para estudiarlo y aprender su funcionamiento.

La gratuidad del programa, junto con las libertades recién mencionadas, hacen de Maxima una formidable herramienta pedagógica, de investigación y de cálculo técnico, accesible a todos los presupuestos, tanto institucionales como individuales.

No quiero terminar esta breve introducción sin antes recordar y agradecer a todo el equipo humano que ha estado y está detrás de este proyecto, desde el comienzo de su andadura allá por los años 70 hasta el momento actual; incluyo también a todas las personas que, aún no siendo desarrolladoras del programa, colaboran haciendo uso de él, presentando y aportando continuamente propuestas e ideas a través de las listas de correos. Un agradecimiento especial a Robert Dodier y James Amundson, quienes me brindaron la oportunidad de formar parte del equipo de desarrollo de Maxima y me abrieron las puertas a una experiencia verdaderamente enriquecedora.

Finalmente, un recuerdo a William Schelter. Gracias a él disponemos hoy de un Maxima que, paradójicamente, por ser libre, no pertenece a nadie pero que es de todos.

Ferrol-A Coruña

2. Despegando con Maxima

2.1. Instalación

Maxima funciona en Windows, Linux y Mac-OS.

En Windows, la instalación consiste en descargar el binario `exe` desde el enlace correspondiente en la página del proyecto y ejecutarlo.

En Linux, la mayoría de las distribuciones tienen ficheros precompilados en sus respectivos repositorios con las extensiones `rpm` o `deb`, según el caso. Sin embargo, en algunas distribuciones las versiones oficiales no suelen estar muy actualizadas, por lo que podría ser recomendable proceder con la compilación de las fuentes, tal como se expone a continuación.

Las siguientes indicaciones hacen referencia al sistema operativo Linux.

Si se quiere instalar Maxima en su estado actual de desarrollo, será necesario descargar los ficheros fuente completos del CVS de Sourceforge y proceder posteriormente a su compilación. Se deberá tener operativo un entorno Common Lisp en la máquina (`clisp`, `cmucl`, `sbcl` o `gcl` son todas ellas alternativas válidas), así como todos los programas que permitan ejecutar las instrucciones que se indican a continuación, incluidas las dependencias `tcl-tk` y `gnuplot`. *Grosso modo*, estos son los pasos a seguir para tener Maxima operativo en el sistema[1]:

1. Descargar las fuentes a un directorio local siguiendo las instrucciones que se indican en el enlace al CVS de la página del proyecto.

2. Acceder a la carpeta local de nombre `maxima` y ejecutar las instrucciones

   ```
   ./bootstrap
   ./configure --enable-clisp --enable-lang-es-utf8
   make
   make check
   sudo make install
   make clean
   ```

3. Para ejecutar Maxima desde la línea de comandos (Figura 1) se deberá escribir

   ```
   maxima
   ```

   si se quiere trabajar con la interfaz gráfica (Figura 2), teclear

   ```
   xmaxima
   ```

Otro entorno gráfico es Wxmaxima[2] (Figura 3), que se distribuye conjuntamente con el ejecutable para Windows. En Linux, una vez instalado Maxima, se podrá instalar este entorno separadamente.

[1] Se supone que se trabaja con clisp, con codificación unicode y que queremos el sistema de ayuda en inglés y español.

[2] http://wxmaxima.sourceforge.net

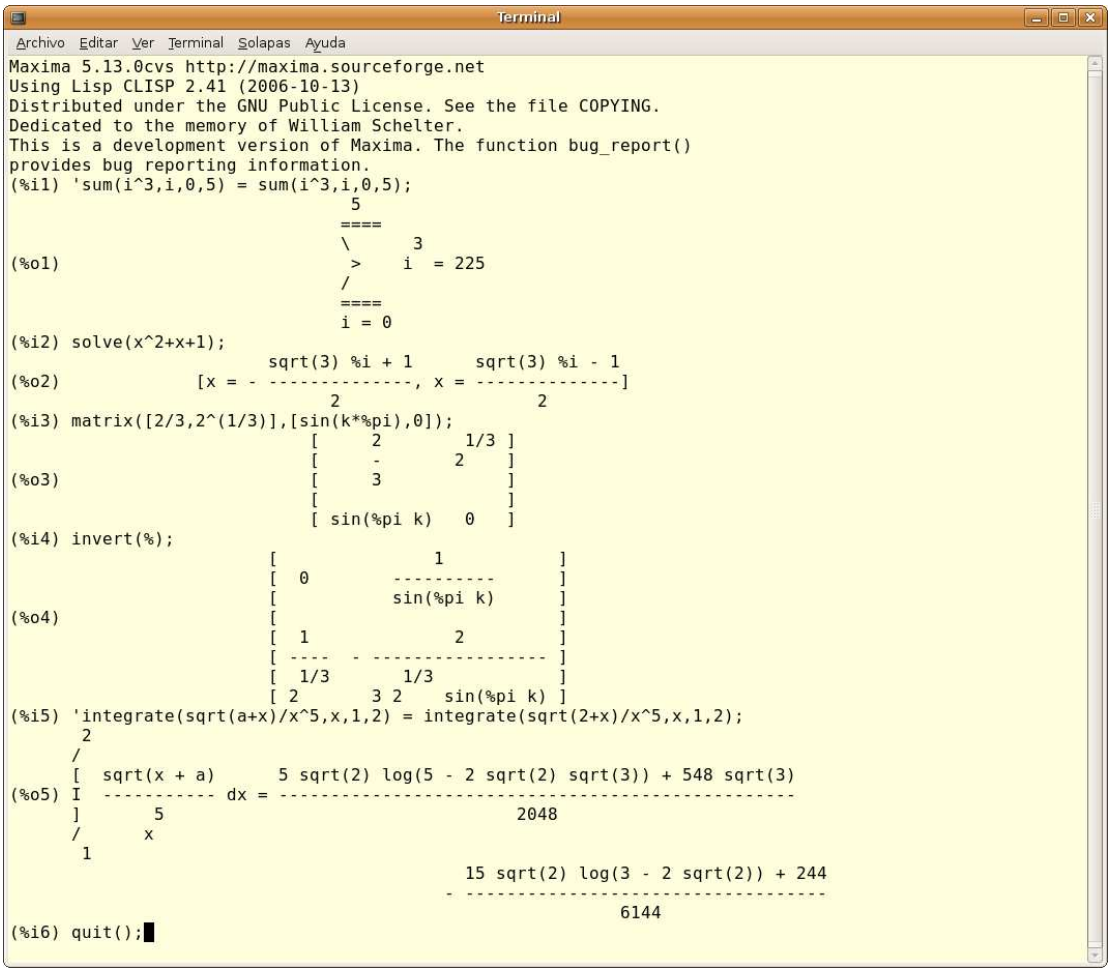


Figura 1: Maxima desde la línea de comandos.

Una cuarta alternativa, que resulta muy vistosa por hacer uso de TEX, es la de ejecutar Maxima desde el editor Texmacs[3] (Figura 4).

Emacs es un editor clásico en sistemas Unix. Aunque hay varios formas de ejecutar Maxima desde Emacs, con el modo Imaxima[4] se consigue que los resultados estén formateados en TEX.

También se han desarrollado en los últimos años entornos web para Maxima[5].

[3] http://www.texmacs.org
[4] http://members3.jcom.home.ne.jp/imaxima
[5] http://maxima.sourceforge.net/relatedprojects.shtml

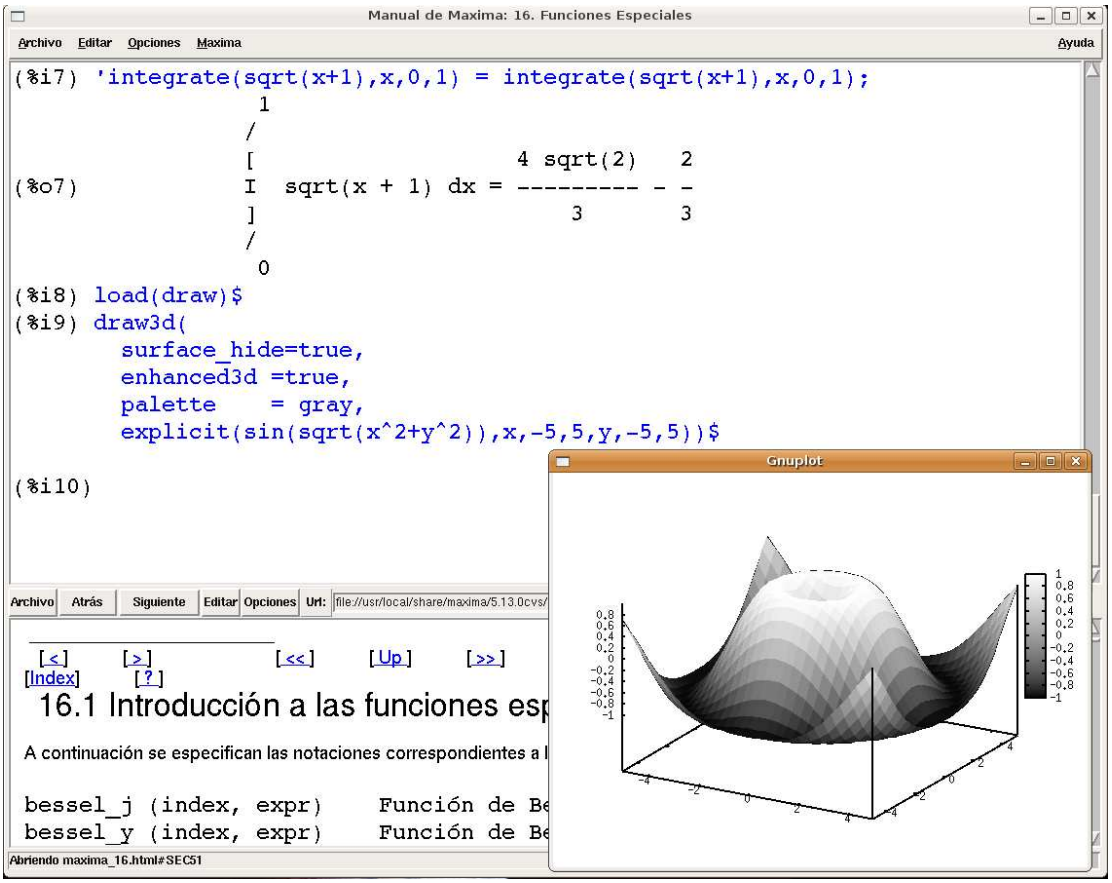


Figura 2: Xmaxima, el entorno gráfico basado en tcl-tk.

2.2. Una primera sesión

En esta sesión se intenta realizar un primer acercamiento al programa a fin de familiarizarse con el estilo operativo de Maxima, dejando sus habilidades matemáticas para más adelante.

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

```
Maxima 5.14.0 http://maxima.sourceforge.net
Using Lisp CLISP 2.38 (2006-01-24)
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)
```

En primer lugar nos informa sobre la versión de Maxima que se está ejecutando y la URL del proyecto, a continuación indica qué entorno Lisp está dándole soporte al programa

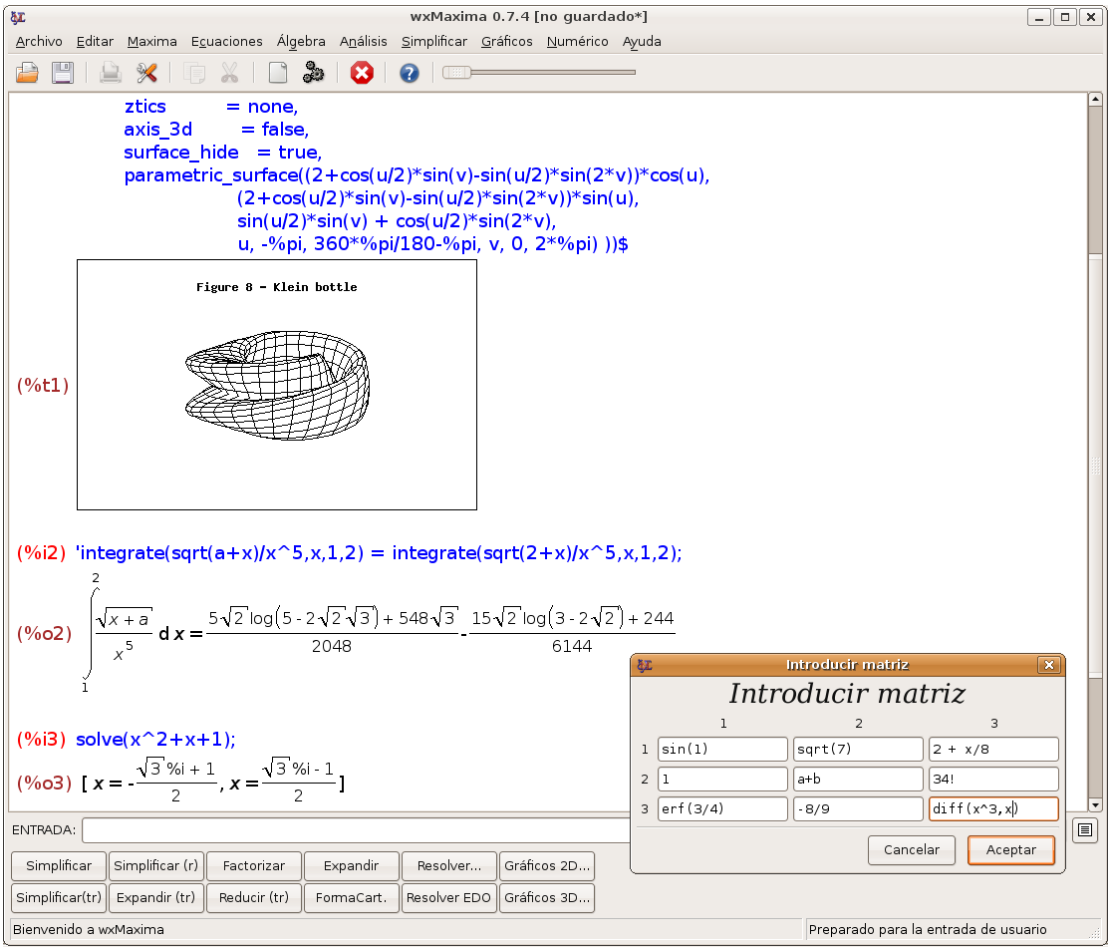


Figura 3: Wxmaxima, el entorno gráfico basado en wxwidgets.

y la nota legal sobre la licencia, por supuesto GNU-GPL. Finaliza con un recordatorio a Schelter y cómo debemos proceder si encontramos un fallo en el programa, ayudando así a su mejora. Tras esta cabecera, muy importante, aparece el indicador `(%i1)` esperando nuestra primera pregunta (`i` de *input*). 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;

(%o1)                          68

(%i2)
```

Vemos que el resultado de la suma está etiquetado con el símbolo `(%o1)` indicando que es la salida correspondiente a la primera entrada (`o` de *output*). A continuación `(%i2)` indica que Maxima espera nuestra segunda instrucción.

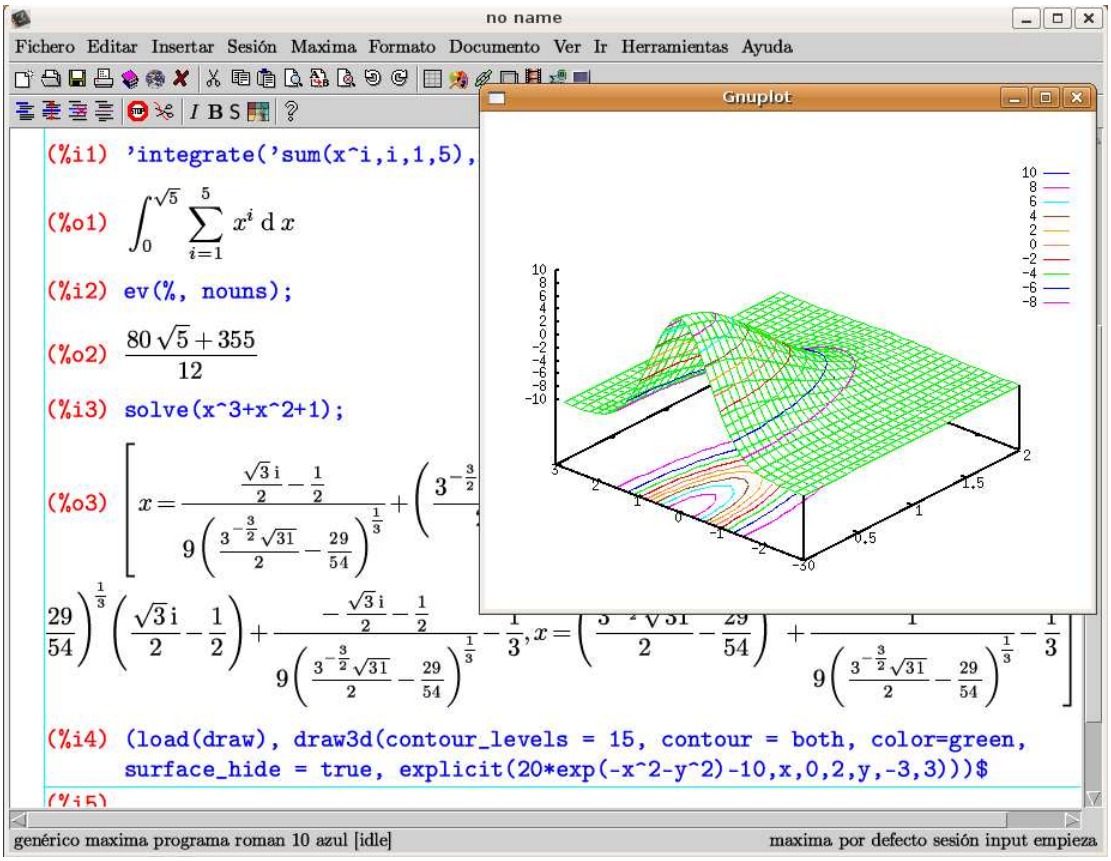


Figura 4: Maxima desde Texmacs.

El punto y coma actúa también como un separador cuando escribimos varias instrucciones en un mismo 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

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 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;
```

(%o13) −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. Hay dos formas de resolver esta situación; la primera consiste en utilizar el operador comilla simple (`'`), que evita la evaluación de las variables:

```
(%i9) solve('x^2-3*'x+1=0,'x);
```

(%o9) $$\left[x = -\frac{\sqrt{5}-3}{2}, x = \frac{\sqrt{5}+3}{2}\right]$$

```
(%i10) x;
```

(%o10) 34578

como se ve, `x` mantiene su valor original.

La segunda forma de evitar este problema consiste en vaciar el contenido de la variable `x` mediante la función `kill`,

```
(%i11) kill(x)$
(%i12) solve(x^2-3*x+1=0,x);
```

(%o12) $$\left[x = -\frac{\sqrt{5}-3}{2}, x = \frac{\sqrt{5}+3}{2}\right]$$

```
(%i13) x;
```

(%o13) x

en este caso, `x` ha perdido su valor.

Las etiquetas `(%in)` y `(%on)`, siendo `n` 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

```
(%i14) %o7/%o8;
```

(%o14) $$-\frac{11341618578}{316475}$$

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

```
(%i15) inchar;
```

(%o15) $\%i$

```
(%i16) outchar;
```

(%o16) $\%o$

```
(%i17) inchar: 'menda;
```

(%o17) menda

```
(menda18) outchar: 'lerenda;
```

(lerenda18) lerenda

```
(menda19) 2+6;
```

(lerenda19) 8

```
(menda20) inchar:'%i$ outchar:'%o$
```

```
(%i22)
```

Al final se han restaurado los valores por defecto.

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

```
(%i23) ''%i8;
```

(%o23) $x - 984003$

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

```
(%i24) %-x;
```

(%o24) -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 ($i = \sqrt{-1}$), la constante de Euler-Mascheroni ($\gamma = -\int_0^\infty e^{-x} \ln x dx$) y la razón aurea ($\phi = \frac{1+\sqrt{5}}{2}$), se representarán por `%e`, `%pi`, `%i`, `%gamma` y `%phi`, 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,

```
(%i25) describe(sqrt);

 -- Función: sqrt (<x>)
     Raíz cuadrada de <x>. Se representa internamente por '<x>^(1/2)'.
     Véase también 'rootscontract'.

     Si la variable 'radexpand' vale 'true' hará que las raíces
     'n'-ésimas de los factores de un producto que sean potencias de
     'n' sean extraídas del radical; por ejemplo, 'sqrt(16*x^2)' se
     convertirá en '4*x' sólo si 'radexpand' vale 'true'.


  There are also some inexact matches for 'sqrt'.
  Try '?? sqrt' to see them.

(%o25)                         true
```

nos instruye sobre la utilización de la función `sqrt`. Alternativamente, se puede utilizar para el mismo fin el operador interrogación (`?`),

```
(%i26) ? sqrt

 -- Función: sqrt (<x>)
     Raíz cuadrada de <x>. Se representa internamente por '<x>^(1/2)'.
     Véase también 'rootscontract'.
```

```
     Si la variable ‘radexpand’ vale ‘true’ hará que las raíces
     ‘n’-ésimas de los factores de un producto que sean potencias de
     ‘n’ sean extraídas del radical; por ejemplo, ‘sqrt(16*x^2)’ se
     convertirá en ‘4*x’ sólo si ‘radexpand’ vale ‘true’.


  There are also some inexact matches for ‘sqrt’.
  Try ‘?? sqrt’ to see them.

(%o26)                         true
```

Cuando no conocemos el nombre completo de la función sobre la que necesitamos información podremos utilizar el operador de doble interrogación (`??`) que nos hace la búsqueda de funciones que contienen en su nombre cierta subcadena,

```
(%i27) ?? sqrt

 0: isqrt  (Operadores generales)
 1: sqrt  (Operadores generales)
 2: sqrtdenest  (Funciones y variables para simplification)
 3: sqrtdispflag  (Operadores generales)
Enter space-separated numbers, ‘all’ or ‘none’: 3

 -- Variable opcional: sqrtdispflag
     Valor por defecto: ‘true’

     Si ‘sqrtdispflag’ vale ‘false’, hará que ‘sqrt’ se muestre con el
     exponente 1/2.


(%o27)                         true
```

Inicialmente muestra un sencillo menú con las funciones que contienen la subcadena `sqrt` y en el que debemos seleccionar qué información concreta deseamos; en esta ocasión se optó por `3` que es el número asociado a la variable opcional `sqrtdispflag`.

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 `(%o9)`, junto con la instrucción que la generó, la entrada `(%i9)`,

```
(%i28) save("mi.sesion",y,resultado=%o9,ecuacion=%i9)$
```

véase 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 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

```
(%i29) quit();
```

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

```
(%i1) load("mi.sesion")$
(%i2) y;
```

$$(\%o2) \qquad 984003$$

```
(%i3) ecuacion;
```

$$(\%o3) \qquad \text{solve}\left(x^2 - 3\,x + 1 = 0, x\right)$$

```
(%i4) resultado;
```

$$(\%o4) \qquad \left[x = -\frac{\sqrt{5}-3}{2}, x = \frac{\sqrt{5}+3}{2}\right]$$

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);
```

$$(\%o5) \qquad \text{solve}\left(x^2 - 5\,x + 1 = 0, x\right)$$

```
(%i6) ev(%, nouns);
```

$$(\%o6) \qquad \left[x = -\frac{\sqrt{21}-5}{2}, x = \frac{\sqrt{21}+5}{2}\right]$$

La función `save` guarda las expresiones matemáticas en formato Lisp, pero a nivel de usuario quizás sea más claro almacenarlas en el propio lenguaje de Maxima; si tal es el caso, mejor será el uso de `stringout`, cuyo resultado será un fichero que también se podrá leer mediante `load`.

Continuemos experimentando con esta segunda sesión de Maxima que acabamos de iniciar.

Maxima está escrito en Lisp, por lo que ya se puede intuir su potencia a la hora de trabajar con listas; en el siguiente diálogo, el texto escrito entre las marcas `/*` e `*/` son comentarios que no afectan a los cálculos,

```
(%i7) /* Se le asigna a la variable x una lista */
      x: [cos(%pi), 4/16, [a, b], (-1)^3, integrate(u^2,u)];
```

$$(\%o7) \qquad \left[-1, \frac{1}{4}, [a,b], -1, \frac{u^3}{3}\right]$$

```
(%i8) /* Se calcula el número de elementos del último resultado */
      length(%);
```

$$(\%o8) \qquad 5$$

```
(%i9) /* Se transforma una lista en conjunto, */
      /* eliminando redundancias. Los         */
      /* corchetes se transforman en llaves   */
      setify(x);
```

$$(\%o9) \qquad \left\{-1, \frac{1}{4}, [a,b], \frac{u^3}{3}\right\}$$

Vemos a continuación cómo se procede a la hora de hacer sustituciones en una expresión,

```
(%i10) /* Sustituciones a hacer en x */
       x, u=2, a=c;
```

$$(\%o10) \qquad \left[-1, \frac{1}{4}, [c,b], -1, \frac{8}{3}\right]$$

```
(%i11) /* Forma alternativa para hacer lo mismo */
       subst([u=2, a=c],x);
```

$$(\%o11) \qquad \left[-1, \frac{1}{4}, [c,b], -1, \frac{8}{3}\right]$$

```
(%i12) %o7[5], u=[1,2,3];
```

$$(\%o12) \qquad \left[\frac{1}{3}, \frac{8}{3}, 9\right]$$

En esta última entrada, `%o7` hace referencia al séptimo resultado devuelto por Maxima, que es la lista guardada en la variable `x`, de modo que `%o7[5]` es el quinto elemento de esta lista, por lo que esta expresión es equivalente a a `x[5]`; después, igualando `u` a la lista `[1,2,3]` este quinto elemento de la lista es sustituído sucesivamente por las tres cantidades. Tal como ya se ha comentado, Maxima utiliza los dos puntos (`:`) para hacer

asignaciones a variables, mientras que el símbolo de igualdad se reserva para la construcción de ecuaciones.

Un aspecto a tener en cuenta es que el comportamiento de Maxima está controlado por los valores que se le asignen a ciertas variables globales del sistema. Una de ellas es la variable `numer`, que por defecto toma el valor lógico `false`, lo que indica que Maxima evitará dar resultados en formato decimal, prefiriendo expresiones simbólicas y fraccionarias,

```
(%i13) numer;
```

(%o13) false

```
(%i14) sqrt(8)/12;
```

(%o14) $\frac{2^{-\frac{1}{2}}}{3}$

```
(%i15) numer:true$
(%i16) sqrt(8)/12;
```

(%o16) 0.2357022603955159

```
(%i17) numer:false$  /*se reinstaura valor por defecto */
```

Sin embargo, en ciertos contextos será preferible el uso de la función `float`,

```
(%i18) float(sqrt(8)/12);
```

(%o18) 0.2357022603955159

Las matrices también tienen su lugar en Maxima; la manera más inmediata de construirlas es con la instrucción `matrix`,

```
(%i19) A: matrix([sin(x),2,%pi],[0,y^2,5/8]);
```

(%o19) $\begin{pmatrix} \sin x & 2 & \pi \\ 0 & y^2 & \frac{5}{8} \end{pmatrix}$

```
(%i20) B: matrix([1,2],[0,2],[-5,integrate(x^2,x,0,1)]);
```

(%o20) $\begin{pmatrix} 1 & 2 \\ 0 & 2 \\ -5 & \frac{1}{3} \end{pmatrix}$

```
(%i21) A.B;  /* producto matricial */
```

(%o21)
$$\begin{pmatrix} \sin x - 5\,\pi & 2\,\sin x + \frac{\pi}{3} + 4 \\ -\frac{25}{8} & 2\,y^2 + \frac{5}{24} \end{pmatrix}$$

En cuanto a las capacidades gráficas de Maxima, se remite al lector a la sección correspondiente.

A fin de preparar publicaciones científicas, si se quiere redactar documentos en formato LaTeX, la función `tex` de Maxima será una útil compañera; en el siguiente ejemplo se calcula una integral y a continuación se pide la expresión resultante en formato TeX:

```
(%i22) 'integrate(sqrt(a+x)/x^5,x,1,2) = integrate(sqrt(2+x)/x^5,x,1,2)$

(%i23) tex(%);
$$\int_{1}^{2}{{{\sqrt{x+4}}\over{x^5}}\;dx}={{5\,\sqrt{2}\,\log
 \left(5-2\,\sqrt{2}\,\sqrt{3}\right)+548\,\sqrt{3}}\over{2048}}-{{15
 \,\sqrt{2}\,\log \left(3-2\,\sqrt{2}\right)+244}\over{6144}}$$
```

Al pegar y copiar el código encerrado entre los dobles símbolos de dólar a un documento LaTeX, el resultado es el siguiente:

$$\int_1^2 \frac{\sqrt{x+a}}{x^5}\,dx = \frac{5\,\sqrt{2}\,\log\left(5-2\,\sqrt{2}\,\sqrt{3}\right)+548\,\sqrt{3}}{2048} - \frac{15\,\sqrt{2}\,\log\left(3-2\,\sqrt{2}\right)+244}{6144}$$

Un comentario sobre la entrada `%i22`. Se observa en ella que se ha escrito dos veces el mismo código a ambos lados del signo de igualdad. La única diferencia entre ambos es el apóstrofo que antecede al primer `integrate`; éste es el operador de *comilla simple*, ya anteriormente aparecido, que Maxima utiliza para evitar la ejecución de una instrucción, de forma que en el resultado la primera integral no se calcula, pero sí la segunda, dando lugar a la igualdad que se obtiene como resultado final. Las expresiones que en Maxima se marcan con la comilla simple para no ser evaluadas reciben el nombre de *expresiones nominales*, en contraposición a las otras, conocidas como *expresiones verbales*.

Ya se ha comentado más arriba cómo solicitar ayuda de Maxima. Algunos usuarios encuentran engorroso que la ayuda se interfiera con los cálculos; esto se puede arreglar con un mínimo de bricolage informático. Al tiempo que se instala Maxima, se almacena también el sistema de ayuda en formato `html` en una determinada carpeta; el usuario tan sólo tendrá que buscar esta carpeta, en la que se encontrará la página principal `maxima.html`, y hacer uso de la función `system` de Maxima que ejecuta comandos del sistema, la cual admite como argumento una cadena alfanumérica con el nombre del navegador (en el ejemplo, `mozilla`) seguido de la ruta hacia el documento `maxima.html`,

```
(%i24)
system("(mozilla /usr/local/share/maxima/5.14.0/doc/html/es.utf8/maxima.html)&")$
```

Este proceso se puede automatizar si la instrucción anterior se guarda en un fichero de nombre `maxima-init.mac` y se guarda en una carpeta oculta de nombre `.maxima` en el directorio principal del usuario[6]. Así, cada vez que arranque Maxima, se abrirá el navegador con el manual de referencia del programa.

[6]En Linux

3. Números, estructuras de datos y sus operaciones

3.1. Operaciones aritméticas

Maxima puede trabajar con números enteros tan grandes como sea necesario,

```
(%i1) 200!;
```

(%o1)
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062351700858796178922222789623703897374720000000000000000000000000000000000000000000000000000

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

```
(%i2) ((3^7/(4+3/5))^(-1)+7/9)^3;
```

(%o2) $$\frac{620214013952}{1307544150375}$$

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,

```
(%i3) float(%);
```

(%o3) 0.4743350454163436

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`:

```
(%i4) fpprec:100$  bfloat(%e / %pi);
```

(%o5)
$$8.65255979432265087217774789646089617428744623908515539454330288948045$$
$$044570677058631924662516184517 3_B \times 10^{-1}$$

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 `%i4` 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.

Veamos cómo factorizar un números tal como 200! en sus factores primos,

```
(%i6) factor(%o1);
```

(%o6)
$$2^{197}\,3^{97}\,5^{49}\,7^{32}\,11^{19}\,13^{16}\,17^{11}\,19^{10}\,23^{8}\,29^{6}\,31^{6}\,37^{5}\,41^{4}\,43^{4}\,47^{4}\,53^{3}\,59^{3}\,61^{3}\,67^{2}\,71^{2}$$
$$73^{2}\,79^{2}\,83^{2}\,89^{2}\,97^{2}\,101\,103\,107\,109\,113\,127\,131\,137\,139\,149\,151\,157\,163\,167\,173$$
$$179\,181\,191\,193\,197\,199$$

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

```
(%i7) showtime:true$
Evaluation took 0.00 seconds (0.00 elapsed) using 80 bytes.
(%i8) factor(2^300-1);
Evaluation took 0.86 seconds (0.85 elapsed) using 48.571 MB.
```

(%o8)
$$3^{2}\,5^{3}\,7\,11\,13\,31\,41\,61\,101\,151\,251\,331\,601\,1201\,1321\,1801\,4051\,8101\,63901\,100801$$
$$268501\,10567201\,13334701\,1182468601\,1133836730401$$

```
(%i9) showtime:false$  /* desactivamos la información de los tiempos */
```

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

Además de la variable opcional `showtime`, otro mecanismo de control de los tiempos de ejecución es la función `time`,

```
(%i10) time(%o8);
```

(%o10) [1.7161081]

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

```
(%i11) primep(3^56-1);
```

(%o11) 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,

```
(%i12) evenp(42);
```

(%o12) true

```
(%i13) oddp(31);
```

(%o13) true

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

```
(%i14) divisors(100);
```

(%o14) $\{1, 2, 4, 5, 10, 20, 25, 50, 100\}$

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`,

```
(%i15) quotient(5689,214);
```

(%o15) 26

```
(%i16) remainder(5689,214);
```

(%o16) 125

3.2. 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*/
```

(%o4) $i + \frac{7}{2}$

```
(%i5) z1 - z2;     /*resta*/
```

(%o5) $$9\,i+\frac{5}{2}$$

```
(%i6) z1 * z2;      /*multiplicación*/
```

(%o6) $$\left(\frac{1}{2}-4\,i\right)\,(5\,i+3)$$

```
(%i7) z1 / z2;      /* división */
```

(%o25) $$\frac{5\,i+3}{\frac{1}{2}-4\,i}$$

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

```
(%i8) rectform(%o6);
```

(%o8) $$\frac{43}{2}-\frac{19\,i}{2}$$

```
(%i9) rectform(%o7);
```

(%o9) $$\frac{58\,i}{65}-\frac{74}{65}$$

Las funciones `realpart` e `imagpart` extraen del número complejo sus partes real e imaginaria, respectivamente,

```
(%i10) realpart(%o7);
```

(%o10) $$-\frac{74}{65}$$

```
(%i11) imagpart(%o7);
```

(%o11) $$\frac{58}{65}$$

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

```
(%i12) polarform(%o7);
```

(%o12) $$\frac{2\,\sqrt{34}\,e^{i\,\left(\pi-\arctan\left(\frac{29}{37}\right)\right)}}{\sqrt{65}}$$

```
(%i13) rectform(%);  /* forma rectangular */
```

$$(\%o13) \qquad \frac{58\sqrt{34}\,i}{\sqrt{65}\sqrt{2210}} - \frac{74\sqrt{34}}{\sqrt{65}\sqrt{2210}}$$

```
(%i14) float(%);  /* formato decimal */
```

$$(\%o14) \qquad 0.8923076923076924\,i - 1.138461538461538$$

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

```
(%i15) abs(%o9);
```

$$(\%o15) \qquad \frac{2\sqrt{34}}{\sqrt{65}}$$

```
(%i16) abs(%o12);
```

$$(\%o16) \qquad \frac{2\sqrt{34}}{\sqrt{65}}$$

Por último, el conjugado de un número complejo se calcula con la función `conjugate`. Como se ve en este ejemplo, el resultado dependerá del formato del argumento, cartesiano o polar,

```
(%i17) conjugate(%o9);        /*forma cartesiana*/
```

$$(\%o17) \qquad -\frac{58\,i}{65} - \frac{74}{65}$$

```
(%i18) conjugate(%o12);       /*forma polar*/
```

$$(\%o18) \qquad -\frac{2\sqrt{34}\,e^{i\,\arctan\left(\frac{29}{37}\right)}}{\sqrt{65}}$$

3.3. 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, siempre encerradas entre corchetes,

```
(%i2) r:[1,[a,3],sqrt(3)/2,"Don Quijote"];
```

$$(\%o2) \qquad \left[1, [a, 3], \frac{\sqrt{3}}{2}, \text{Don Quijote}\right]$$

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,

```
(%i3) listp(r); /* es r una lista? */
```

$$(\%o3) \qquad \text{true}$$

```
(%i4) first(r); /* primer elemento */
```

$$(\%o4) \qquad 1$$

```
(%i5) second(r); /* segundo elemento */
```

$$(\%o5) \qquad [a, 3]$$

```
(%i6) third(r); /* ...hasta tenth */
```

$$(\%o6) \qquad \frac{\sqrt{3}}{2}$$

```
(%i7) last(r); /* el último de la lista */
```

$$(\%o7) \qquad \text{Don Quijote}$$

```
(%i8) rest(r); /* todos menos el primero */
```

$$(\%o8) \qquad \left[[a, 3], \frac{\sqrt{3}}{2}, \text{Don Quijote}\right]$$

```
(%i9) part(r,3); /* pido el que quiero */
```

(%o9) $\frac{\sqrt{3}}{2}$

```
(%i10) length(r); /* cuantos hay? */
```

(%o10) 4

```
(%i11) reverse(r); /* le damos la vuelta */
```

(%o11) $\left[\text{Don Quijote}, \frac{\sqrt{3}}{2}, [a, 3], 1\right]$

```
(%i12) member(a,r); /* es a un elemento?*/
```

(%o12) false

```
(%i13) member([a,3],r); /* lo es [a,3]? */
```

(%o13) true

```
(%i14) sort(q: [7,a,1,d,5,3,b]); /* asigno valor a q y ordeno */
```

(%o14) $[1, 3, 5, 7, a, b, d]$

```
(%i15) delete([a,3],r); /* borro elemento */
```

(%o15) $\left[1, \frac{\sqrt{3}}{2}, \text{Don Quijote}\right]$

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

```
(%i16) r;
```

(%o16) $\left[1, [a, 3], \frac{\sqrt{3}}{2}, \text{Don Quijote}\right]$

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

```
(%i17) cons(1+2,q);
```

(%o17) $[3, 7, a, 1, d, 5, 3, b]$

```
(%i18) endcons(x,q);
```

(%o18) $[7, a, 1, d, 5, 3, b, x]$

```
(%i19) q;
```

(%o19) $[7, a, 1, d, 5, 3, b]$

En este ejemplo hemos observado también que la lista `q` no ha cambiado. Si lo que queremos es actualizar su contenido,

```
(%i20) q: endcons(x,cons(1+2,q))$
(%i21) q;
```

(%o21) $[3, 7, a, 1, d, 5, 3, b, x]$

Es posible unir dos listas,

```
(%i22) append(r,q);
```

(%o22) $\left[1, [a, 3], \frac{\sqrt{3}}{2}, \text{Don Quijote}, 3, 7, a, 1, d, 5, 3, b, x\right]$

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

```
(%i23) s:makelist(2+k*2,k,0,10);
```

(%o23) $[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]$

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,

```
(%i24) apply("+",s);
```

(%o24) 132

```
(%i25) apply("*",s);
```

(%o25) 81749606400

aunque estas dos operaciones hubiese sido quizás más apropiado 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,

```
(%i26) map("!",s);
```

$$(\%o26) \quad [2, 24, 720, 40320, 3628800, 479001600, 87178291200, 20922789888000, 6402373705728000, 2432902008176640000, 112400072777607680000]$$

```
(%i27) map(sin,s);
```

$$(\%o27) \quad [\sin 2, \sin 4, \sin 6, \sin 8, \sin 10, \sin 12, \sin 14, \sin 16, \sin 18, \sin 20, \sin 22]$$

A veces se quiere utilizar en `map` una función que no está definida y que no va a ser utilizada más que en esta llamada a `map`. En tales casos se puede hacer uso de las funciones *lambda*; supóngase que a cada elemento x de una lista se le quiere aplicar la función $f(x) = \sin(x) + \frac{1}{2}$, sin necesidad de haberla definido previamente,

```
(%i28) map(lambda([x], sin(x) + 1/2),[7,3,4,9]);
```

$$(\%o28) \quad \left[\sin 7 + \frac{1}{2}, \sin 3 + \frac{1}{2}, \sin 4 + \frac{1}{2}, \sin 9 + \frac{1}{2}\right]$$

Como se ve, la función *lambda* admite dos argumentos; el primero es una lista (no se puede evitar la redundancia) de argumentos, siendo el segundo la expresión que indica qué se hace con los elementos de la lista anterior.

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

```
(%i29) [1,2,3]+[a,b,c];
```

$$(\%o29) \quad [a+1, b+2, c+3]$$

```
(%i30) [1,2,3]*[a,b,c];
```

$$(\%o30) \quad [a, 2\,b, 3\,c]$$

```
(%i31) [1,2,3]/[a,b,c];
```

$$(\%o31) \quad \left[\frac{1}{a}, \frac{2}{b}, \frac{3}{c}\right]$$

```
(%i32) [1,2,3]-[a,b,c];
```

$$(\%o32) \quad [1-a, 2-b, 3-c]$$

```
(%i33) [1,2,3].[a,b,c]; /* producto escalar */
```

$$(\%o33) \quad 3\,c + 2\,b + a$$

```
(%i34) [a,b,c]^3;
```

(%o34) $$\left[a^3, b^3, c^3\right]$$

```
(%i35) 3^[a,b,c];
```

(%o35) $$\left[3^a, 3^b, 3^c\right]$$

En Maxima, las listas se pueden interpretar como vectores; para más información, acúdase a la Sección 5.6

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,

```
(%i36) listarith:false$
(%i37) [1,2,3]+[4,5,6];
```

(%o37) $$[4,5,6]+[1,2,3]$$

```
(%i38) 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,

```
(%i39) flatten([1,[a,b],2,3,[c,[d,e]]]);
```

(%o39) $$[1,a,b,2,3,c,d,e]$$

El cálculo del módulo de un vector se puede hacer mediante la definición previa de una función al efecto:

```
(%i40) modulo(v):=
         if listp(v)
            then sqrt(apply("+",v^2))
            else error("Mucho ojito: ", v, " no es un vector !!!!")$
(%i41) xx:[a,b,c,d,e]$
(%i42) yy:[3,4,-6,0,4/5]$
(%i43) modulo(xx-yy);
```

(%o43) $$\sqrt{\left(e-\frac{4}{5}\right)^2+d^2+(c+6)^2+(b-4)^2+(a-3)^2}$$

3.4. Arrays

Los arrays son estructuras que almacenan datos secuencial o matricialmente. Se diferencian de las listas en que éstas pueden alterar su estructura interna y los arrays no; pero la ventaja frente a las listas estriba en que el acceso a un dato cualquiera es directo, mientras que en las listas, al ser estructuras dinámicas, para acceder al n-ésimo dato, un puntero debe recorrer todos los elementos de la lista desde el que ocupa el primer lugar. Así pues, se recomienda el uso de arrays cuando se vaya a utilizar una secuencia larga de cantidades cuya estructura se va a mantener inalterable; en general, se mejorarán los tiempos de procesamiento.

Si queremos construir un array, lo debemos declarar previamente mediante la función `array`, indicando su nombre y dimensiones; a continuación le asignaríamos valores a sus elementos,

```
(%i1) array(arreglillo,2,2);
```

(%o1) *arreglillo*

```
(%i2) arreglillo[0,0]: 34$
(%i3) arreglillo[1,2]:x+y$
(%i4) arreglillo[2,2]:float(%pi)$
(%i5) listarray(arreglillo);
```

(%o5) $[34, \#\#\#\#\#, \#\#\#\#\#, \#\#\#\#\#, \#\#\#\#\#,$
$y + x, \#\#\#\#\#, \#\#\#\#\#, 3.141592653589793]$

En el ejemplo, hemos definido un array de dimensiones 3 por 3, ya que la indexación de los arrays comienza por el cero, no por el uno como en el caso de las listas; a continuación hemos asignado valores a tres elementos del array: un entero, una expresión simbólica y un número decimal en coma flotante. Finalmente, la función `listarray` nos permite ver el contenido del array, utilizando el símbolo `#####` para aquellos elementos a los que aún no se les ha asignado valor alguno.

Si ya se sabe de antemano que el array va a contener únicamente números enteros o números decimales, podemos declararlo al definirlo para que Maxima optimice el procesamiento de los datos. A continuación se declara un array de números enteros haciendo uso del símbolo `fixnum` (en el caso de números decimales en coma flotante se utilizaría `flonum`) y a continuación la función `fillarray` rellenará el array con los elementos previamente almacenados en una lista,

```
(%i6) lis: [6,0,-4,456,56,-99];
```

(%o6) $[6, 0, -4, 456, 56, -99]$

```
(%i7) array(arr,fixnum,5);
```

(%o7) arr

```
(%i8) fillarray(arr,lis);
```

(%o8) arr

```
(%i9) arr[4];
```

(%o9) 56

```
(%i10) listarray(arr);
```

(%o10) $[6, 0, -4, 456, 56, -99]$

Hasta ahora hemos hecho uso de los llamados array declarados. Si no hacemos la declaración pertinente mediante la función `array`, y en su lugar realizamos asignaciones directas a los elementos de la estructura, creamos lo que se conoce como array de claves (*hash array*) o, en la terminología de Maxima, array no declarado; estos arrays son mucho más generales que los declarados, puesto que pueden ir creciendo de forma dinámica y los índices no tienen que ser necesariamente números enteros,

```
(%i11) tab[9]:76;
```

(%o11) 76

```
(%i12) tab[-58]:2/9;
```

(%o12) $\frac{2}{9}$

```
(%i13) tab[juan]:sqrt(8);
```

(%o13) $2^{\frac{3}{2}}$

```
(%i14) tab["cadena"]: a-b;
```

(%o14) $a - b$

```
(%i15) tab[x+y]:5;
```

(%o15) 5

```
(%i16) listarray(tab);
```

(%o16) $$\left[\frac{2}{9}, 76, a-b, 2^{\frac{3}{2}}, 5\right]$$

```
(%i17) tab["cadena"];
```

(%o17) $$a-b$$

Esta generalidad en la estructura de los arrays no declarados tiene la contrapartida de restarle eficiencia en el procesamiento de los datos almacenados, siendo preferible, siempre que sea posible, la utilización de los declarados.

Ya por último, junto con los arrays declarados y no declarados mencionados, es posible trabajar a nivel de Maxima con arrays de lisp. En este caso su construcción se materializa con la función `make_array`,

```
(%i18) a: make_array(flonum,5);
```

(%o18) $$\{Array : \#(0.0\ \ 0.0\ \ 0.0\ \ 0.0\ \ 0.0)\}$$

```
(%i19) a[0]:3.4;
```

(%o19) 3.4

```
(%i20) a[4]:6.89;
```

(%o20) 6.89

```
(%i21) a;
```

(%o21) $$\{Array : \#(3.4\ \ 0.0\ \ 0.0\ \ 0.0\ \ 6.89)\}$$

Nótense aquí un par de diferencias respecto de la función `array`: el número 5 indica el número de elementos (indexados de 0 a 4) y no el índice máximo admisible; por otro lado, la función no devuelve el nombre del objeto array, sino el propio objeto.

3.5. Cadenas

Una cadena se construye escribiendo un texto o cualquier otra expresión entre comillas dobles. Vamos a hacer algunos ejemplos básicos,

```
(%i1) xx: "Los ";
```

(%o1) Los

```
(%i2) yy: " cerditos";
```

```
(%o2)                                cerditos
```

```
(%i3) zz: concat(xx,sqrt(9),yy);
```

```
(%o3)                              Los 3 cerditos
```

la función `concat` 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,

```
(%i4) stringp(zz);
```

```
(%o4)                                true
```

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

```
(%i5) slength(zz);
```

```
(%o5)                                14
```

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

```
(%i6) concat(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(%);
```

(%o72) los 3 cerditos

```
(%i16) /* a mayúsculas entre la 1a y 2a */
       supcase(%,1,2);
```

(%o74) 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.

En algunos contextos el usuario tiene almacenada una expresión sintácticamente correcta de Maxima en formato de cadena y pretende evaluarla, en cuyo caso la función `eval_string` la analizará y evaluará, en tanto que `parse_string` la analiza pero no la evalúa, limitándose a transformarla a una expresión nominal de Maxima,

```
(%i29) /* una cadena con una expresión de Maxima */
       expr: "integrate(x^5,x)";
```

(%o29) integrate(x^5,x)

```
(%i30) /* se evalúa la cadena */
       eval_string(expr);
```

(%o30) $$\frac{x^6}{6}$$

```
(%i31) /* se transforma en una expresión nominal */
       parse_string(expr);
```

(%o31) $$integrate\left(x^5, x\right)$$

```
(%i32) ''%;
```

(%o32) $$\frac{x^6}{6}$$

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.

3.6. 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);
```

$$(\%o1) \qquad \left\{3, \sqrt{2}, \{\}, [2,k], \text{Sancho}, a, \{a,b\}, b\right\}$$

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"];
```

$$(\%o2) \qquad \left[[2,k], \sqrt{2}, \{a,b\}, [k,2], \text{Panza}\right]$$

```
(%i3) c2:setify(%);
```

$$(\%o3) \qquad \left\{\sqrt{2}, [2,k], \text{Panza}, \{a,b\}, [k,2]\right\}$$

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

```
(%i4) listify(%o1);
```

$$(\%o4) \qquad \left[3, \sqrt{2}, \{\}, [2,k], \text{Sancho}, a, \{a,b\}, b\right]$$

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

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

$$(\%o5) \qquad \text{true}$$

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);
```

$$(\%o6) \qquad \text{true}$$

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

```
(%i7) c1: disjoin(sqrt(2),c1);  /* sqrt(2) fuera */
```

$$(\%o7) \qquad \{3, \{\}, [2,k], \text{Sancho}, a, \{a,b\}, b\}$$

```
(%i8) c1: adjoin(sqrt(3),c1);  /* sqrt(3) dentro */
```

$$(\%o8) \qquad \left\{3, \sqrt{3}, \{\}, [2,k], \text{Sancho}, a, \{a,b\}, b\right\}$$

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);
```

$$(\%o9) \qquad \left\{3, \sqrt{2}, \{\}, [2,k], \text{Sancho}, a, \{a,b\}, b\right\}$$

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

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

$$(\%o10) \qquad \text{true}$$

A continuación algunos ejemplos de operaciones con conjuntos,

```
(%i11) union(c1,c2);
```

$$(\%o11) \qquad \left\{3, \sqrt{2}, \sqrt{3}, \{\}, [2,k], \text{Panza}, \text{Sancho}, a, \{a,b\}, b, [k,2]\right\}$$

```
(%i12) intersection(c1,c2);
```

$$(\%o12) \qquad \{[2,k], \{a,b\}\}$$

```
(%i13) setdifference(c1,c2);
```

$$(\%o13) \qquad \left\{3, \sqrt{3}, \{\}, \text{Sancho}, a, b\right\}$$

```
(%i14) cardinality(%);
```

$$(\%o14) \qquad 6$$

Vemos aquí también cómo pedir el cardinal de un conjunto.

Igual que se ha visto cómo 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));
```

$$(\%o15) \qquad \{\sin 1, \sin 2, \sin 3, \sin 4, \sin 5\}$$

Por último ya, el producto cartesiano de tres conjuntos,

```
(%i16) cartesian_product(set(1,2),set(a,b,c),set(x,y));
```

$$(\%o16) \qquad \begin{array}{l}\{[1,a,x], [1,a,y], [1,b,x], [1,b,y], [1,c,x], [1,c,y],\\ [2,a,x], [2,a,y], [2,b,x], [2,b,y], [2,c,x], [2,c,y]\}\end{array}$$

3.7. Grafos

El soporte para el desarrollo de algoritmos en el ámbito de la teoría de grafos se encuentra en el módulo `graphs`. Se trata de un paquete de reciente creación que aporta la infraestructura básica para el trabajo con grafos, orientados o no. No se admiten bucles ni aristas múltiples.

```
(%i1) load(graphs)$
(%i2) g: create_graph(
            [1,2,3,4,5,6,7],
            [ [1,2],[2,3],[2,4],[3,4],
              [4,5],[5,6],[4,6],[6,7] ]);
```

(%o2) GRAPH

Como vemos, la respuesta de la función `create_graph` no es muy informativa, pero nos podemos hacer una idea de la estructura del grafo haciendo

```
(%i3) print_graph(g)$

Graph on 7 vertices with 8 edges.
Adjacencies:
  7 :  6
  6 :  7  4  5
  5 :  6  4
  4 :  6  5  3  2
  3 :  4  2
  2 :  4  3  1
  1 :  2
```

Vemos que, por ejemplo, el vértice número 3 comparte aristas (no orientadas) con los vértices 4 y 2.

En caso de que el grafo anterior fuese orientado (digrafo) su definición sería:

```
(%i4) dg: create_graph(
            [1,2,3,4,5,6,7],
            [ [1,2],[2,3],[2,4],[3,4],
              [4,5],[5,6],[4,6],[6,7] ],
            directed = true)$
(%i5) print_graph(dg)$

Digraph on 7 vertices with 8 arcs.
Adjacencies:
  7 :
  6 :  7
```

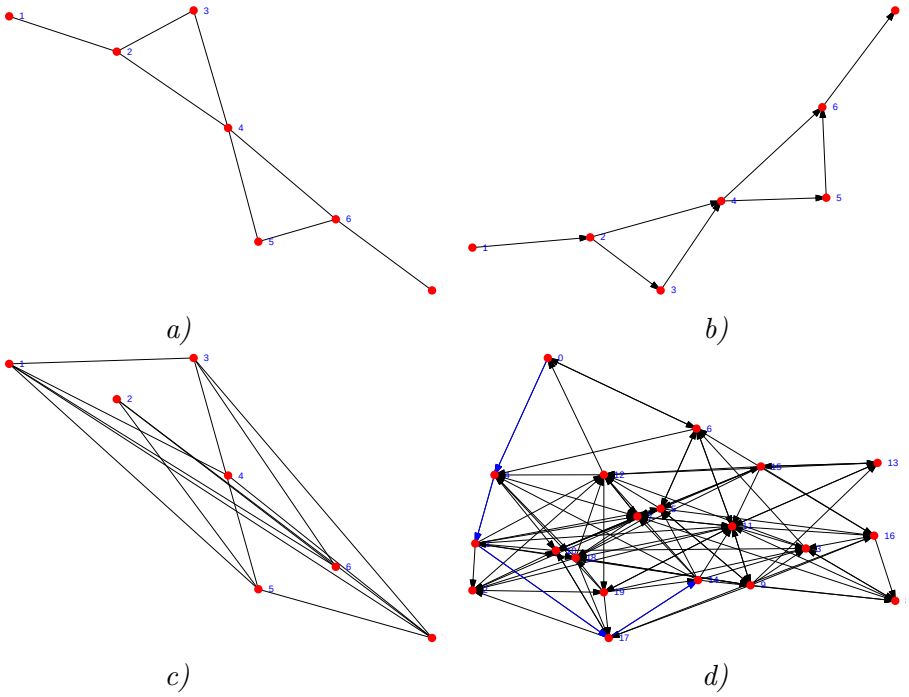


Figura 5: Grafos: *a)* no orientado; *b)* orientado; *c)* complemetario del no orientado; *d)* camino más corto.

```
5 :  6
4 :  6  5
3 :  4
2 :  4  3
1 :  2
```

En este caso, `print_graph` nos muestra los nodos hijos a la derecha de los padres. En particular, el nodo 7 no es origen de ningún arco, pero es hijo del nodo 6. Otra manera alternativa de mostrar la estructura de un grafo es mediante la función `draw_graph`

```
(%i6) draw_graph(g, show_id=true, terminal=eps)$
(%i7) draw_graph(dg, show_id=true, head_length=0.05, terminal=eps)$
```

Cuyos aspectos se pueden observar en los apartados *a)* y *b)* de la Figura 5.

Continuemos con el grafo no orientado `g` definido más arriba y veamos cómo extraer de él cierta información: el número cromático, matriz de adyacencia, grafo complementario, si es conexo, o planar, su diámetro, etc.

```
(%i8) chromatic_number(g);
```

(%o8) 3

```
(%i9) adjacency_matrix(g);
```

(%o9)
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
(%i10) gc:complement_graph(g);
```

(%o10) GRAPH

```
(%i11) print_graph(%)$

Graph on 7 vertices with 13 edges.
Adjacencies:
  1 :  3  4  5  6  7
  2 :  5  6  7
  3 :  1  5  6  7
  4 :  1  7
  5 :  1  2  3  7
  6 :  1  2  3
  7 :  1  2  3  4  5
```

Pedimos ahora a Maxima que nos represente gráficamente el complementario de `g`, cuyo aspecto es del apartado *c)* de la Figura 5.

```
(%i12) draw_graph(gc, show_id=true, terminal=eps)$
```

Continuamos con algunas otras propiedades:

```
(%i13) is_connected(g);
```

(%o13) true

```
(%i14) is_planar(g);
```

(%o14) true

```
(%i15) diameter(g);
```

(%o15) 4

Junto con la definición manual, también es posible generar determinados grafos mediante funciones, como el cíclico, el completo, el dodecaedro, un grafo aleatorio, etc.

```
(%i16) print_graph(g1: cycle_graph(5))$

Graph on 5 vertices with 5 edges.
Adjacencies:
  4 :  0  3
  3 :  4  2
  2 :  3  1
  1 :  2  0
  0 :  4  1


(%i17) print_graph(g2: complete_graph(4))$

Graph on 4 vertices with 6 edges.
Adjacencies:
  3 :  2  1  0
  2 :  3  1  0
  1 :  3  2  0
  0 :  3  2  1


(%i18) print_graph(g3: dodecahedron_graph())$

Graph on 20 vertices with 30 edges.
Adjacencies:
 19 : 14 13  1
 18 : 13 12  2
 17 : 12 11  3
 16 : 11 10  4
 15 : 14 10  0
 14 :  5 19 15
 13 :  7 19 18
 12 :  9 18 17
 11 :  8 17 16
 10 :  6 16 15
  9 :  7  8 12
  8 :  9  6 11
  7 :  5  9 13
```

```
  6 :  8  5 10
  5 :  7  6 14
  4 : 16  0  3
  3 : 17  4  2
  2 : 18  3  1
  1 : 19  2  0
  0 : 15  4  1
```

```
(%i19) /* 0.4 de probabilidad para cada arco */
       print_graph(g4: random_graph(10,0.4))$

Graph on 10 vertices with 15 edges.
Adjacencies:
  9 :  8  5  2  1  0
  8 :  9  6  3
  7 :  5  2  1
  6 :  8  3
  5 :  9  7
  4 :  2
  3 :  8  6  0
  2 :  9  7  4  0
  1 :  9  7  0
  0 :  9  3  2  1
```

Por último ya, un ejemplo de cálculo de la ruta más corta. Se genera un grafo orientado aleatorio, a continuación se calcula la ruta más corta entre dos vértices dados y finalmente se hace una representación gráfica del cálculo realizado, la cual se puede ver en el apartado *d)* de la Figura 5.

```
(%i20) r : random_digraph(20, 0.25)$
```

```
(%i21) /* Una lista de los vértices del digrafo */
       vertices(r);
```

$$(\%o21) \qquad [19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]$$

```
(%i22) sh: shortest_path(0,14,r);
```

$$(\%o22) \qquad [0, 4, 1, 17, 14]$$

```
(%i23) draw_graph(
          r,
          show_edges  = vertices_to_path(sh),
          show_id     = true,
          head_length = 0.05,
          terminal    = eps)$
```

El paquete `graphs` también permite el uso de etiquetas en los nodos y ponderaciones en las aristas y arcos. El lector interesado podrá consultar la documentación correspondiente[7].

[7] A partir de la versión 5.14 de Maxima

4. Álgebra

4.1. Transformaciones simbólicas

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);
```

$$(\%o1) \qquad \frac{1}{x+b} - (x-a)^3 + (x+3)^5 + \frac{1}{\left(x-\frac{1}{4}\right)^5}$$

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

```
(%i2) expand(q);
```

$$(\%o2) \qquad \frac{1}{x^5-\frac{5\,x^4}{4}+\frac{5\,x^3}{8}-\frac{5\,x^2}{32}+\frac{5\,x}{256}-\frac{1}{1024}} + \frac{1}{x+b} + x^5 + 15\,x^4 + 89\,x^3 + 3\,a\,x^2 + 270\,x^2 - 3\,a^2\,x + 405\,x + a^3 + 243$$

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);
```

$$(\%o3) \qquad \frac{1}{x+b} + (x+3)^5 - x^3 + 3\,a\,x^2 - 3\,a^2\,x + \frac{1}{\left(x-\frac{1}{4}\right)^5} + a^3$$

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;
```

$$(\%o4) \qquad \frac{1}{x+2\,c} + (x+3)^5 - x^3 + 6\,x^2 - 12\,x + \frac{1}{\left(x-\frac{1}{4}\right)^5} + 8$$

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))` o `expand(q,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);
```

(%o5) $$3\,x^2 + 625$$

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;
```

(%o6) $$-\frac{y+x}{z} + (y+x)^2 + \frac{1}{y+x}$$

```
(%i7) subst(k,x+y,%);
```

(%o44) $$-\frac{k}{z} + k^2 + \frac{1}{k}$$

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);
```

(%o10)
$$a\,b^7 - 2\,b^7 + 5\,a^2\,b^6 - 8\,a\,b^6 - 4\,b^6 + 10\,a^3\,b^5 - 10\,a^2\,b^5 - 19\,a\,b^5 - 2\,b^5 + 10\,a^4\,b^4 -$$
$$35\,a^2\,b^4 - 10\,a\,b^4 + 5\,a^5\,b^3 + 10\,a^4\,b^3 - 30\,a^3\,b^3 - 20\,a^2\,b^3 + a^6\,b^2 + 8\,a^5\,b^2 - 10\,a^4\,b^2 -$$
$$20\,a^3\,b^2 + 2\,a^6\,b + a^5\,b - 10\,a^4\,b + a^6 - 2\,a^5$$

```
(%i11) factor(%);
```

(%o11) $$(a-2)\,(b+1)^2\,(b+a)^5$$

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

```
(%i12) p1: x^7-4*x^6-7*x^5+8*x^4+11*x^3-4*x^2-5*x;
```

(%o12) $$x^7-4\,x^6-7\,x^5+8\,x^4+11\,x^3-4\,x^2-5\,x$$

```
(%i13) p2: x^4-2*x^3-4*x^2+2*x+3;
```

(%o13) $$x^4-2\,x^3-4\,x^2+2\,x+3$$

```
(%i14) gcd(p1,p2);
```

(%o14) $$x^3+x^2-x-1$$

```
(%i15) load(functs)$
(%i16) lcm(p1,p2);
```

(%o16) $$(x-5)\,(x-3)\,(x-1)^2\,x\,(x+1)^3$$

En `(%i12)` y `(%i13)` definimos los polinomios `p1` y `p2`, a continuación calculamos su máximo común divisor (mcd) en `(%i14)` y antes de pedir el mínimo común múltiplo en `(%i16)` cargamos el paquete `functs` en el que se encuentra definida la función `lcm`. Es posible que deseemos disponer del mcd factorizado, por lo que hacemos

```
(%i17) factor(%o14);
```

(%o17) $$(x-1)\,(x+1)^2$$

Si en algún momento queremos inhibir las simplificaciones, podemos hacer uso de la variable global `simp`, cuyo valor por defecto es `true`,

```
(%i18) simp: false$
(%i19) 2 + 2 + a + a;
```

(%o19) $$2+2+a+a$$

```
(%i20) simp: true$
(%i21) %o19;
```

(%o21) $$2\,a+4$$

Veamos ahora algunas transformaciones que se pueden hacer con fracciones algebraicas.

```
(%i22) expr:(x^2-x)/(x^2+x-6)-5/(x^2-4);
```

(%o22)
$$\frac{x^2-x}{x^2+x-6}-\frac{5}{x^2-4}$$

Podemos factorizar esta expresión,

```
(%i23) factor(expr);
```

(%o23)
$$\frac{(x-3)\,\left(x^2+4\,x+5\right)}{(x-2)\,(x+2)\,(x+3)}$$

La expansión devuelve el siguiente resultado,

```
(%i24) expand(expr);
```

(%o24)
$$\frac{x^2}{x^2+x-6}-\frac{x}{x^2+x-6}-\frac{5}{x^2-4}$$

Ahora descomponemos la fracción algebraica `expr` en fracciones simples,

```
(%i25) partfrac(expr,x);
```

(%o25)
$$-\frac{12}{5\,(x+3)}+\frac{5}{4\,(x+2)}-\frac{17}{20\,(x-2)}+1$$

Por último, transformamos la misma expresión a su forma canónica CRE (*Cannonical Rational Expression*), que es un formato que Maxima utiliza para reducir expresiones algebraicas equivalentes a una forma única,

```
(%i26) radcan(expr);
```

(%o26)
$$\frac{x^3+x^2-7\,x-15}{x^3+3\,x^2-4\,x-12}$$

La función `ratsimp` también 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:

```
(%i27) (x^(a/2)-1)^2*(x^(a/2)+1)^2 / (x^a-1);
```

(%o27)
$$\frac{\left(x^{\frac{a}{2}}-1\right)^2\,\left(x^{\frac{a}{2}}+1\right)^2}{x^a-1}$$

```
(%i28) ratsimp(%);  /* simplificamos una vez */
```

$$(\%o28) \qquad \frac{x^{2\,a} - 2\,x^a + 1}{x^a - 1}$$

```
(%i29) ratsimp(%);  /* simplificamos otra vez */
```

$$(\%o29) \qquad x^a - 1$$

```
(%i30) fullratsimp(%o27);   /* simplificamos todo de una vez! */
```

$$(\%o30) \qquad x^a - 1$$

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

```
(%i31) fr: (x^3-4*x^2+4*x-2)/(x^2+x+1);
```

$$(\%o31) \qquad \frac{x^3 - 4\,x^2 + 4\,x - 2}{x^2 + x + 1}$$

```
(%i32) num(fr);
```

$$(\%o32) \qquad x^3 - 4\,x^2 + 4\,x - 2$$

```
(%i33) denom(fr);
```

$$(\%o33) \qquad x^2 + x + 1$$

Maxima nunca puede adivinar a priori las intenciones del usuario para con las expresiones con las que está trabajando, por ello sigue la política de no tomar ciertas decisiones, por lo que a veces puede parecer que no hace lo suficiente o que no sabe hacerlo. Esta forma *vaga* de proceder la suple con una serie de funciones y variables globales que permitirán al usuario darle al motor simbólico ciertas directrices sobre qué debe hacer con las expresiones, cómo reducirlas o transformarlas. Sin ánimo de ser exhaustivos, siguen algunos ejemplos en los que se controlan transformaciones de tipo logarítmico y trigonométrico.

Las expresiones que contienen logaritmos suelen ser ambiguas a la hora de reducirlas. Transformamos a continuación la misma expresión según diferentes criterios,

```
(%i34) log(x^r)-log(x*y) + a*log(x/y);
```

$$(\%o34) \qquad -\log(x\,y) + a\,\log\left(\frac{x}{y}\right) + r\,\log x$$

En principio, ha hecho la transformación $\log x^r \;\rightarrow\; r \log x$. Esto se controla con la variable global `logexpand`, cuyo valor por defecto es `true`, lo cual sólo permite la reducción recién indicada. Si también queremos que nos expanda los logaritmos de productos y divisiones, le cambiaremos el valor a esta variable global, tal como se indica a continuación,

```
(%i35) logexpand: super$
(%i36) log(x^r)-log(x*y) + a*log(x/y);
```

$$(\%o36) \qquad -\log y + a\,(\log x - \log y) + r\,\log x - \log x$$

```
(%i37) /* pedimos que nos simplifique la expresión anterior */
       ratsimp(%);
```

$$(\%o37) \qquad (-a-1)\,\log y + (r+a-1)\,\log x$$

En caso de no querer que nos haga transformación alguna sobre la expresión de entrada, simplemente haremos

```
(%i38) logexpand: false$
(%i39) log(x^r)-log(x*y) + a*log(x/y);
```

$$(\%o39) \qquad -\log(x\,y) + a\,\log\left(\frac{x}{y}\right) + \log x^r$$

Devolvemos ahora la variable `logexpand` a su estado por defecto,

```
(%i40) logexpand: true$
```

La función `logcontract` es la que nos va a permitir compactar expresiones logarítmicas,

```
(%i41) logcontract(2*(a*log(x) + 2*a*log(y)));
```

$$(\%o41) \qquad a\,\log\left(x^2\,y^4\right)$$

Por defecto, Maxima contrae los coeficientes enteros; para incluir en este caso la variable a dentro del logaritmo le asignaremos la propiedad de ser un número entero,

```
(%i42) declare(a, integer)$
(%i43) ''%i41;  /* fuerza reevaluación de expresión nominal */
```

$$(\%o43) \qquad \log\left(x^{2\,a}\,y^{4\,a}\right)$$

Esta es una forma de hacerlo, pero Maxima permite un mayor refinamiento sobre este particular haciendo uso de la variable global `logconcoeffp`.

Toda esta casuística de diferentes formas de representar una misma expresión aumenta considerablemente con aquéllas en las que intervienen las funciones trigonométricas e hiperbólicas. Empecemos con la función `trigexpand`, cuyo comportamiento se controla con las variables globales `trigexpand` (que le es homónima), `trigexpandplus` y `trigexpandtimes`, cuyos valores por defectos son, respectivamente, `false`, `true` y `true`. Puestos a experimentar, definamos primero una expresión trigonométrica e interpretamos algunos resultados,

```
(%i44) expr: x+sin(3*x+y)/sin(x);
```

$$(\%o44) \qquad \frac{\sin\left(y+3\,x\right)}{\sin x}+x$$

```
(%i45) trigexpand(expr); /* aquí trigexpand vale false */
```

$$(\%o45) \qquad \frac{\cos\left(3\,x\right)\,\sin y+\sin\left(3\,x\right)\,\cos y}{\sin x}+x$$

Vemos que sólo se desarrolló la suma del numerador; ahora cambiamos el valor lógico de `trigexpand`,

```
(%i46) trigexpand(expr), trigexpand=true;
```

$$(\%o46) \qquad \frac{\left(\cos^3 x-3\,\cos x\,\sin^2 x\right)\,\sin y+\left(3\,\cos^2 x\,\sin x-\sin^3 x\right)\,\cos y}{\sin x}+x$$

Cuando la asignación de la variable global se hace como se acaba de indicar, sólo tiene efecto temporal durante esa ejecución, sin haberse alterado a nivel global. Podemos ordenar a Maxima que simplifique la expresión anterior,

```
(%i47) ratsimp(%);
```

$$(\%o47) \qquad -\frac{\left(3\,\cos x\,\sin^2 x-\cos^3 x\right)\,\sin y+\left(\sin^3 x-3\,\cos^2 x\,\sin x\right)\,\cos y-x\,\sin x}{\sin x}$$

Podemos inhibir el desarrollo de la suma,

```
(%i48) trigexpand(expr), trigexpandplus=false;
```

$$(\%o48) \qquad \frac{\sin\left(y+3\,x\right)}{\sin x}+x$$

Otra variable global de interés, que no tiene nada que ver con la función `trigexpand`, es `halfangles`, con valor `false` por defecto, que controla si se reducen los argumentos trigonométricos con denominador dos,

```
(%i49) sin(x/2);
```

$$(\%o49) \qquad \sin\left(\frac{x}{2}\right)$$

```
(%i50) sin(x/2), halfangles=true;
```

(%o50) $$\frac{\sqrt{1-\cos x}}{\sqrt{2}}$$

La función `trigsimp` fuerza el uso de las identidades fundamentales $\sin^2 x + \cos^2 x = 1$ y $\cosh^2 x - \sinh^2 x = 1$ para simplificar expresiones,

```
(%i51) 5*sin(x)^2 + 4*cos(x)^2;
```

(%o51) $$5\,\sin^2 x + 4\,\cos^2 x$$

```
(%i52) trigsimp(%);
```

(%o52) $$\sin^2 x + 4$$

Como un último ejemplo, veamos cómo reducir productos y potencias de funciones trigonométricas a combinaciones lineales,

```
(%i53) -sin(x)^2+3*cos(x)^2*tan(x);
```

(%o53) $$3\,\cos^2 x\,\tan x - \sin^2 x$$

```
(%i54) trigreduce(%);
```

(%o54) $$\frac{3\,\sin\left(2\,x\right)}{2} + \frac{\cos\left(2\,x\right)}{2} - \frac{1}{2}$$

Y si queremos transformar expresiones trigonométricas en complejas,

```
(%i55) exponentialize(%);
```

(%o55) $$\frac{e^{2\,i\,x} + e^{-2\,i\,x}}{4} - \frac{3\,i\,\left(e^{2\,i\,x} - e^{-2\,i\,x}\right)}{4} - \frac{1}{2}$$

Otras funciones útiles en este contexto y que el usuario podrá consultar en la documentación son: `triginverses`, `trigsign` y `trigrat`.

4.2. Ecuaciones

Almacenar una ecuación en una variable es tan simple como hacer

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

$$(\%o1) \qquad 3\,x = x + 1$$

A partir de ahí, aquellas operaciones en las que intervenga la variable 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;
```

$$(\%o2) \qquad 2\,x = 1$$

```
(%i3) %/2;
```

$$(\%o3) \qquad x = \frac{1}{2}$$

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);
```

$$(\%o4) \qquad \left[x = \frac{1}{2}\right]$$

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);
```

$$(\%o5) \qquad \left[x = -\frac{\sqrt{(4-4\,a)\;b+9}+3}{2\,b}, x = \frac{\sqrt{(4-4\,a)\;b+9}-3}{2\,b}\right]$$

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);
```

(%o6) $$x = -\frac{\sqrt{(4-4\,a)\;b+9}+3}{2\,b}$$

```
(%i7) sol: rhs(%);
```

(%o7) $$-\frac{\sqrt{(4-4\,a)\;b+9}+3}{2\,b}$$

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 general 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);
```

(%o8) $$\left[0 = x^5 - 3\,x^4 + 2\,x^3 - 2\,x^2 - x + 4\right]$$

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

También `solve` se puede utilizar para la resolución de sistemas, en cuyo caso las ecuaciones deben ir agrupadas en una lista, así como las incógnitas; nos planteamos la resolución del siguiente sistema no lineal de incógnitas x e y,

$$\begin{cases} 3*x^2 - y^2 &= 6 \\ x &= y + a \end{cases}$$

```
(%i9) solve([3*x^2-y^2=6,x=y+a],[x,y]);
```

(%o9) $$\left[\left[x = -\frac{\sqrt{3}\,\sqrt{a^2+4}+a}{2}, y = -\frac{\sqrt{3}\,\sqrt{a^2+4}+3\,a}{2}\right], \left[x = \frac{\sqrt{3}\,\sqrt{a^2+4}-a}{2}, y = \frac{\sqrt{3}\,\sqrt{a^2+4}-3\,a}{2}\right]\right]$$

Una alternativa al uso de `solve` es la función `algsys`. Veamos cómo `algsys` trata la resolución de la ecuación polinómica anterior `%o8`,

```
(%i10) algsys([x^5 - 3*x^4 + 2*x^3 -2*x^2 - x + 4 = 0],[x]);
```

(%o10) $$[[x = 2.478283086356668], [x = .1150057557117295 - 1.27155810694299\,i], [x = 1.27155810694299\,i + .1150057557117295], [x = -.8598396689940523], [x = 1.151545166402536]]$$

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,

```
(%i11) realonly:true$
(%i12) ''%i10;  /* recalcula la entrada %i10 */
```

(%o12)
$$[[x = 2.478283086356668], [x = 1.151545166402536], [x = -.8598396689940523]]$$

```
(%i13) realonly:false$  /* le devolvemos el valor por defecto */
```

Un sistema no lineal con coeficientes paramétricos y complejos

$$\begin{cases} 3u - av &= t \\ \frac{2+i}{u+t} &= 3v + u \\ \frac{t}{u} &= 1 \end{cases}$$

```
(%i14) algsys([3*u-a*v=t,(2+%i)/(u+t)=3*v+u,t/u=1],[u,v,t]);
```

(%o14)
$$\left[\left[u = \frac{\sqrt{\frac{i\,a}{a+6}+\frac{2\,a}{a+6}}}{\sqrt{2}}, v = \frac{2\,\sqrt{i+2}}{\sqrt{2}\,\sqrt{a}\,\sqrt{a+6}}, t = \frac{\sqrt{i+2}\,\sqrt{a}}{\sqrt{2}\,\sqrt{a+6}}\right], \left[u = -\frac{\sqrt{\frac{i\,a}{a+6}+\frac{2\,a}{a+6}}}{\sqrt{2}}, v = -\frac{2\,\sqrt{i+2}}{\sqrt{2}\,\sqrt{a}\,\sqrt{a+6}}, t = -\frac{\sqrt{i+2}\,\sqrt{a}}{\sqrt{2}\,\sqrt{a+6}}\right]\right]$$

Veamos cómo trata `algsys` las ecuaciones indeterminadas, devolviendo la solución en términos paramétricos,

```
(%i15) algsys([3*x^2-5*y=x],[x,y]);
```

(%o15)
$$\left[\left[x = \%r_1, y = \frac{3\,\%r_1{}^2 - \%r_1}{5}\right]\right]$$

```
(%i16) %, %r1:1;
```

(%o16)
$$\left[\left[x = 1, y = \frac{2}{5}\right]\right]$$

Maxima nombra los parámetros siguiendo el esquema `%rn`, siendo `n` un número entero positivo. En la entrada `%i16` pedimos que en `%o15` se sustituya el parámetro por la unidad.

En ocasiones conviene eliminar las variables de los sistemas de forma controlada; en tales casos la función a utilizar será `eliminate`,

```
(%i17) eliminate([3*u-a*v=t,(2+%i)/(u+t)=3*v+u,t/u=1],[u]);
```

(%o17)
$$\left[2\,t - a\,v, -\left(a^2 + 9\,a\right)\,v^2 - (5\,a + 36)\,t\,v - 4\,t^2 + 9\,i + 18\right]$$

Cuando de lo que se trata es de resolver un sistema de ecuaciones lineales, la mejor opción es `linsolve`, cuya sintaxis es similar a la de las funciones anteriores. En el siguiente ejemplo, el objetivo es

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

```
(%i18) linsolve(
         [ 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]);
```

(%o18)
$$\left[x = \frac{3021\sqrt{2} - 12405}{48457}, y = \frac{1537\sqrt{2} + 16642}{48457}, z = \frac{53\sqrt{2} - 2768}{48457}\right]$$

Cuando la matriz de coeficientes del sistema es dispersa, la función `fast_linsolve` será preferible, ya que aprovechará tal circunstancia para encontrar las soluciones de forma más rápida.

Si los coeficientes del sistema se tienen en formato matricial, quizás sea más apropiado el uso de la función `linsolve_by_lu`, tal como se indica en la sección dedicada a las matrices.

No todo es resoluble simbólicamente. Existen en Maxima varios procedimientos cuya naturaleza es estrictamente numérica. Uno de ellos es `realroots`, especializado en el cálculo de *aproximaciones racionales* de las raíces reales de ecuaciones polinómicas; el segundo parámetro, que es opcional, indica la cota de error.

```
(%i19) realroots(x^8+x^3+x+1, 5e-6);
```

(%o19)
$$\left[x = -1, x = -\frac{371267}{524288}\right]$$

En cambio, `allroots` obtiene aproximaciones en formato decimal de coma flotante de todas las raíces de las ecuaciones polinómicas, tanto reales como complejas,

```
(%i20) allroots(x^8+x^3+x+1);
```

(%o20)
$$\begin{array}{l} [x = .9098297401801199\,i + .2989522918873167,\ , \\ x = .2989522918873167 - .9098297401801199\,i\ , \\ x = -.7081337759784606, x = .9807253637807569\,i - .4581925662678885\ , \\ x = -.9807253637807569\,i - .4581925662678885,\ , \\ x = .5359278244124014\,i + 1.013307162369803\ , \\ x = 1.013307162369803 - .5359278244124014\,i, x = -1.000000000000001] \end{array}$$

Más allá de las ecuaciones algebraicas, `find_root` utiliza el método de bipartición para resolver ecuaciones en su más amplio sentido,

```
(%i21) f(x):=144 * sin(x) + 12*sqrt(3)*%pi - 36*x^2 - 12*%pi*x$
(%i22) find_root(f(z),z,1,2);
```

(%o22) 1.899889962840263

El uso del método de Newton requiere cargar en memoria el módulo correspondiente. Veamos como ejemplo la ecuación

$$2u^u - 5 = u$$

```
(%i23) load(mnewton)$  /* carga el paquete */
(%i24) mnewton([2*u^u-5],[u],[1]);
0 errors, 0 warnings
```

(%o14) $[[u = 1.70927556786144]]$

y el sistema

$$\begin{cases} x + 3\log(x) - y^2 &= 0 \\ 2x^2 - xy - 5x + 1 &= 0 \end{cases}$$

```
(%i25) mnewton([x+3*log(x)-y^2, 2*x^2-x*y-5*x+1],[x, y], [5, 5]);
```

(%o25) $[[x = 3.756834008012769, y = 2.779849592817897]]$

```
(%i26) mnewton([x+3*log(x)-y^2, 2*x^2-x*y-5*x+1],[x, y], [1, -2]);
```

(%o26) $[[x = 1.373478353409809, y = -1.524964836379522]]$

En los anteriores ejemplos, el primer argumento es una lista con la ecuación o ecuaciones a resolver, las cuales se suponen igualadas a cero; el segundo argumento es la lista de variables y el último el valor inicial a partir del cual se generará el algoritmo y cuya elección determinará las diferentes soluciones del problema.

4.3. Matrices

La definición de una matriz es extremadamente simple en Maxima,

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

(%o1) $$\begin{pmatrix} 3 & 4 & 0 \\ 6 & 0 & -2 \\ 0 & 6 & a \end{pmatrix}$$

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.
```

$$(\%o2) \qquad \begin{pmatrix} \frac{4}{7} & 0 & \pi \\ \sqrt{2} & \log 3 & -9 \end{pmatrix}$$

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);
```

$$(\%o4) \qquad \begin{pmatrix} 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

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];
```

$$(\%o5) \qquad 0$$

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,

```
(%i6) submatrix(1,m1,2);
```

(%o6) $$\begin{pmatrix} 6 & -2 \\ 0 & a \end{pmatrix}$$

Otro ejemplo es el siguiente,

```
(%i7) submatrix(1,2,m1,3);
```

(%o7) $$\begin{pmatrix} 0 & 6 \end{pmatrix}$$

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,

```
(%i8) addrow(m1,[1,1,1],[2,2,2]);
```

(%o8) $$\begin{pmatrix} 3 & 4 & 0 \\ 6 & 0 & -2 \\ 0 & 6 & a \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

```
(%i9) addcol(%,[7,7,7,7,7]);
```

(%o9) $$\begin{pmatrix} 3 & 4 & 0 & 7 \\ 6 & 0 & -2 & 7 \\ 0 & 6 & a & 7 \\ 1 & 1 & 1 & 7 \\ 2 & 2 & 2 & 7 \end{pmatrix}$$

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

```
(%i10) ident(3);
```

(%o10) $$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

y la matriz con todos sus elementos iguales a cero,

```
(%i11) zeromatrix(2,4);
```

(%o11) $$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

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

```
(%i12) diagmatrix(4,%e);
```

(%o12)
$$\begin{pmatrix} e & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & e & 0 \\ 0 & 0 & 0 & e \end{pmatrix}$$

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,

```
(%i13) matrix([[1,2,3],[4,5,6]]);  /* construcción correcta */
```

(%o13)
$$\begin{pmatrix} [1,2,3] & [4,5,6] \end{pmatrix}$$

```
(%i14) matrixp(%);  /* es la anterior realmente una matriz? */
```

(%o14) true

```
(%i15) [[7,8,9],[0,1,2]];    /* otra matriz */
```

(%o15) $[[7,8,9],[0,1,2]]$

```
(%i16) matrixp(%);   /* será una matriz? */
```

(%o16) false

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

```
(%i17) col(m1,3);
```

(%o17)
$$\begin{pmatrix} 0 \\ -2 \\ a \end{pmatrix}$$

```
(%i18) row(m1,2);
```

(%o18)
$$\begin{pmatrix} 6 & 0 & -2 \end{pmatrix}$$

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

```
(%i19) m1^^2;
```

(%o19)
$$\begin{pmatrix} 33 & 12 & -8 \\ 18 & 12 & -2\,a \\ 36 & 6\,a & a^2-12 \end{pmatrix}$$

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,

```
(%i20) m2:m1^2;
```

(%o38)
$$\begin{pmatrix} 9 & 16 & 0 \\ 36 & 0 & 4 \\ 0 & 36 & a^2 \end{pmatrix}$$

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

```
(%i21) m1+m2;
```

(%o21)
$$\begin{pmatrix} 12 & 20 & 0 \\ 42 & 0 & 2 \\ 0 & 42 & a^2+a \end{pmatrix}$$

```
(%i22) m1-m2;
```

(%o22)
$$\begin{pmatrix} -6 & -12 & 0 \\ -30 & 0 & -6 \\ 0 & -30 & a-a^2 \end{pmatrix}$$

```
(%i23) m1.m2;
```

(%o23)
$$\begin{pmatrix} 171 & 48 & 16 \\ 54 & 24 & -2\,a^2 \\ 216 & 36\,a & a^3+24 \end{pmatrix}$$

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,

```
(%i24) m1*m2;
```

(%o24)
$$\begin{pmatrix} 27 & 64 & 0 \\ 216 & 0 & -8 \\ 0 & 216 & a^3 \end{pmatrix}$$

```
(%i25) 4*m1;
```

(%o25)
$$\begin{pmatrix} 12 & 16 & 0 \\ 24 & 0 & -8 \\ 0 & 24 & 4\,a \end{pmatrix}$$

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:

```
(%i26) transpose(m1);     /*la transpuesta*/
```

(%o26)
$$\begin{pmatrix} 3 & 6 & 0 \\ 4 & 0 & 6 \\ 0 & -2 & a \end{pmatrix}$$

```
(%i27) determinant(m1);  /*el determinante*/
```

(%o27)
$$36 - 24\,a$$

```
(%i28) invert(m1);        /*la inversa*/
```

(%o28)
$$\begin{pmatrix} \frac{12}{36-24\,a} & -\frac{4\,a}{36-24\,a} & -\frac{8}{36-24\,a} \\ -\frac{6\,a}{36-24\,a} & \frac{3\,a}{36-24\,a} & \frac{6}{36-24\,a} \\ \frac{36}{36-24\,a} & -\frac{18}{36-24\,a} & -\frac{24}{36-24\,a} \end{pmatrix}$$

```
(%i29) invert(m1),detout;  /*la inversa, con el determinante fuera*/
```

(%o29)
$$\frac{\begin{pmatrix} 12 & -4\,a & -8 \\ -6\,a & 3\,a & 6 \\ 36 & -18 & -24 \end{pmatrix}}{36 - 24\,a}$$

```
(%i30) charpoly(m1,x); /*pol.  caract. con variable x*/
```

(%o30)
$$(3 - x)\ (12 - (a - x)\ x) - 24\ (a - x)$$

```
(%i31) expand(%);      /*pol. caract. expandido*/
```

(%o31)
$$-x^3 + a\,x^2 + 3\,x^2 - 3\,a\,x + 12\,x - 24\,a + 36$$

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

```
(%i32) m1,a=0;
```

(%o32)
$$\begin{pmatrix} 3 & 4 & 0 \\ 6 & 0 & -2 \\ 0 & 6 & 0 \end{pmatrix}$$

```
(%i33) eigenvalues(%);
```

$$(\%o33) \qquad \left[\left[-\frac{\sqrt{15}\,i+3}{2}, \frac{\sqrt{15}\,i-3}{2}, 6\right], [1,1,1]\right]$$

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,

```
(%i34) eigenvectors(%o32);
```

$$(\%o34) \qquad \begin{array}{l}\left[\left[\left[-\frac{\sqrt{15}\,i+3}{2}, \frac{\sqrt{15}\,i-3}{2}, 6\right], [1,1,1]\right],\\ \left[1, -\frac{\sqrt{15}\,i+9}{8}, -\frac{3\sqrt{3}\sqrt{5}\,i-21}{8}\right], \left[1, \frac{\sqrt{15}\,i-9}{8}, \frac{3\sqrt{3}\sqrt{5}\,i+21}{8}\right], \left[1, \frac{3}{4}, \frac{3}{4}\right]\right]\end{array}$$

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`, y 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)`. También podemos solicitar los vectores propios unitarios por medio de la función `uniteigenvectors`,

```
(%i35) uniteigenvectors(%o32);
```

$$(\%o35) \qquad \begin{array}{l}\left[\left[\left[-\frac{\sqrt{15}\,i+3}{2}, \frac{\sqrt{15}\,i-3}{2}, 6\right], [1,1,1]\right],\\ \left[\frac{\sqrt{2}}{\sqrt{23}}, -\frac{\sqrt{2}\sqrt{15}\,i+9\sqrt{2}}{8\sqrt{23}}, -\frac{3\sqrt{2}\sqrt{3}\sqrt{5}\,i-21\sqrt{2}}{8\sqrt{23}}\right],\\ \left[\frac{\sqrt{2}}{\sqrt{23}}, \frac{\sqrt{2}\sqrt{15}\,i-9\sqrt{2}}{8\sqrt{23}}, \frac{3\sqrt{2}\sqrt{3}\sqrt{5}\,i+21\sqrt{2}}{8\sqrt{23}}\right], \left[\frac{2\sqrt{2}}{\sqrt{17}}, \frac{3\sqrt{2}}{2\sqrt{17}}, \frac{3\sqrt{2}}{2\sqrt{17}}\right]\right]\end{array}$$

El rango, el menor de un elemento y una base del espacio nulo de cierta matriz pueden calcularse, respectivamente, con las funciones `rank`, `minor` y `nullspace`,

```
(%i36) rank(%o32);   /* el rango de la matriz*/
```

$$(\%o36) \qquad 3$$

```
(%i37) minor(%o32,2,1);  /* el menor de un elemento */
```

$$(\%o37) \qquad \begin{pmatrix} 4 & 0 \\ 6 & 0 \end{pmatrix}$$

```
(%i38) nullspace(%o32);  /* base del núcleo */
0 errors, 0 warnings
```

$$(\%o38) \qquad span\,()$$

que es la respuesta que se obtiene cuando el espacio nulo está formado por un único elemento.

De forma similar a como la instrucción `map` aplica una función a todos los elementos de una lista, `matrixmap` hace lo propio con los elementos de una matriz,

```
(%i39) matrixmap(sin,%o32);
```

$$(\%o39) \qquad \begin{pmatrix} \sin 3 & \sin 4 & 0 \\ \sin 6 & 0 & -\sin 2 \\ 0 & \sin 6 & 0 \end{pmatrix}$$

Avancemos un poco más en otros aspectos del álgebra lineal. Empezamos con el cálculo de la matriz de Jordan J de cierta matriz dada A, esto es, la matriz que verifica $A = SJS^{-1}$, para cierta S. Para ello es necesario cargar previamente el paquete `diag`, se introduce la matriz A y la función `jordan` se encargará de obtener lo que buscamos en un formato que luego utilizamos para obtener las matrices J y S. Finalmente, comprobamos los resultados:

```
(%i40) A: matrix([2,4,-6,0],[4,6,-3,-4],[0,0,4,0],[0,4,-6,2]);
```

$$(\%o40) \qquad \begin{pmatrix} 2 & 4 & -6 & 0 \\ 4 & 6 & -3 & -4 \\ 0 & 0 & 4 & 0 \\ 0 & 4 & -6 & 2 \end{pmatrix}$$

```
(%i41) load(diag)$  jl: jordan(A);
```

$$(\%o42) \qquad [[6,1],[2,2],[4,1]]$$

```
(%i43) J: dispJordan(jl);
```

$$(\%o43) \qquad \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

```
(%i44) S: ModeMatrix (A,jl);
```

(%o44)
$$\begin{pmatrix} 1 & 4 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & \frac{2}{3} \\ 1 & 4 & 1 & 0 \end{pmatrix}$$

```
(%i45) is(A = S.J.(S^^-1));
```

(%o45) true

Cuando los coeficientes de un sistema lineal se tengan en forma matricial y los términos independientes en un vector, el sistema se podrá resolver con la función `linsolve_by_lu`,

```
(%i46) A : matrix([sqrt(2),4],[a,4/5]);
```

(%o46)
$$\begin{pmatrix} \sqrt{2} & 4 \\ a & \frac{4}{5} \end{pmatrix}$$

```
(%i47) B : [1/3,a];
```

(%o47)
$$\left[\frac{1}{3}, a\right]$$

Resolvemos ahora $AX = B$ y luego simplificamos algo el resultado

```
(%i48) linsolve_by_lu(A,B);
```

(%o48)
$$\left[\begin{pmatrix} \frac{1}{3}-\frac{4\left(a-\frac{a}{3\sqrt{2}}\right)}{\frac{4}{5}-2\sqrt{2}a} \\ \frac{\sqrt{2}}{a-\frac{a}{3\sqrt{2}}} \\ \frac{4}{5}-2\sqrt{2}a \end{pmatrix}, \text{false}\right]$$

```
(%i49) ratsimp(%);
```

(%o49)
$$\left[\begin{pmatrix} \frac{15\sqrt{2}a-\sqrt{2}}{15\sqrt{2}a-6} \\ -\frac{\left(15\sqrt{2}-5\right)a}{60a-12\sqrt{2}} \end{pmatrix}, \text{false}\right]$$

El resultado que se obtiene es una lista que contiene la solución y el símbolo `false`. Cuando la resolución del sistema sea de índole numérica, esta segunda componente será sustituida por el número de condición, una medida de la bondad del resultado numérico. Veamos de aplicar esto a cierta ecuación matricial del estilo $MX = MM$

```
(%i50) M : matrix([sqrt(3),2/5],[3,sin(2)]);
```

(%o50) $$\begin{pmatrix} \sqrt{3} & \frac{2}{5} \\ 3 & \sin 2 \end{pmatrix}$$

```
(%i51) linsolve_by_lu(M,M.M,'floatfield);
```

(%o51) $$\left[\begin{pmatrix} 1.732050807568877 & .4000000000000005 \\ 3.000000000000001 & .9092974268256803 \end{pmatrix}, 49.33731560796254\right]$$

Veamos ahora cómo obtener algunas matrices especiales: la hessiana de una función, la de Hilbert, la de Toeplitz, la de Vandermone y la de Hankel,

```
(%i52) hessian(exp(x^2*y+a*z),[x,y,z]);
```

(%o52) $$\begin{pmatrix} 4\,x^2\,y^2\,e^{a\,z+x^2\,y}+2\,y\,e^{a\,z+x^2\,y} & 2\,x^3\,y\,e^{a\,z+x^2\,y}+2\,x\,e^{a\,z+x^2\,y} & 2\,a\,x\,y\,e^{a\,z+x^2\,y} \\ 2\,x^3\,y\,e^{a\,z+x^2\,y}+2\,x\,e^{a\,z+x^2\,y} & x^4\,e^{a\,z+x^2\,y} & a\,x^2\,e^{a\,z+x^2\,y} \\ 2\,a\,x\,y\,e^{a\,z+x^2\,y} & a\,x^2\,e^{a\,z+x^2\,y} & a^2\,e^{a\,z+x^2\,y} \end{pmatrix}$$

```
(%i53) hilbert_matrix(5);
```

(%o53) $$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix}$$

```
(%i54) toeplitz([1,2,3,4],[t,x,y,z]);
```

(%o54) $$\begin{pmatrix} 1 & x & y & z \\ 2 & 1 & x & y \\ 3 & 2 & 1 & x \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

```
(%i55) vandermonde_matrix([u,v,x,y,z]);
0 errors, 0 warnings
```

(%o55) $$\begin{pmatrix} 1 & u & u^2 & u^3 & u^4 \\ 1 & v & v^2 & v^3 & v^4 \\ 1 & x & x^2 & x^3 & x^4 \\ 1 & y & y^2 & y^3 & y^4 \\ 1 & z & z^2 & z^3 & z^4 \end{pmatrix}$$

```
(%i56) hankel ([v,x,y,z],[p,q,r,s]);
```

(%o56)
$$\begin{pmatrix} v & x & y & z \\ x & y & z & q \\ y & z & q & r \\ z & q & r & s \end{pmatrix}$$

El lector interesado puede obtener información sobre otras funciones matriciales accediendo a la documentación sobre el paquete `linearalgebra`.

4.4. Patrones y reglas

En un programa de cálculo simbólico, éste no sólo debe tener información sobre una función u operación a partir de su definición, sino que también habrá propiedades y reglas de transformación de expresiones de las que un programa como Maxima debe tener noticia.

Nos planteamos la siguiente situación: necesitamos trabajar con una función $G(x, y)$ que, independientemente de que esté definida o no, sabemos que es igual a la expresión $\frac{H(x,y)}{x}$ en todo su dominio, siendo $H(x, y)$ otra función; queremos que la primera expresión sea sustituída por la segunda y además queremos tener bajo control estas sustituciones. Todo ello se consigue trabajando con patrones y reglas.

Antes de definir la regla de sustitución es necesario saber a qué patrones será aplicable, para lo cual admitiremos que los argumentos de $G(x, y)$ pueden tener cualquier forma:

```
(%i1) matchdeclare ([x,y], true);
```

(%o1) done

En este caso, las variables patrón serán `x` e `y`, siendo el segundo argumento una función de predicado[8] que devolverá `true` si el patrón se cumple; en el caso presente, el patrón es universal y admite cualquier formato para estas variables. Si quisiésemos que la regla se aplicase sólo a números enteros, se debería escribir `matchdeclare ([x,y], integerp)`; si quisiésemos que la regla se aplicase siempre que `x` sea un número, sin imponer restricciones a `y`, escribiríamos `matchdeclare (x, numberp, y, true)`. Como se ve, los argumentos impares son variables o listas de variables y los pares las condiciones de los patrones.

Se define ahora la regla de sustitución indicada más arriba, a la que llamaremos `regla1`,

```
(%i2) defrule (regla1, G(x,y), H(x,y)/x);
```

(%o2)
$$regla_1 : G(x, y) \rightarrow \frac{H(x, y)}{x}$$

Aplicamos ahora la regla a las expresiones $G(f, 2 + k)$ y $G(G(4, 6), 2 + k)$,

[8] Se denominan así todas aquellas funciones que devuelven como resultado de su evaluación *true* o *false*.

```
(%i3) apply1(G(f,2+k),regla1);
```

$$(\%o3) \qquad \frac{H\left(f, k+2\right)}{f}$$

```
(%i4) apply1(G(G(4,6),2+k),regla1);
```

$$(\%o4) \qquad \frac{4\,H\left(\frac{H(4,6)}{4}, k+2\right)}{H\left(4,6\right)}$$

Como se ve en este último ejemplo, la regla se aplica a todas las subexpresiones que contengan a la función G. Además, `apply1` aplica la regla desde fuera hacia adentro. La variable global `maxapplydepth` indica a Maxima hasta qué nivel puede bajar en la expresión para aplicar la regla; su valor por defecto es 10000, pero se lo podemos cambiar,

```
(%i5) maxapplydepth;
```

$$(\%o5) \qquad 10000$$

```
(%i6) maxapplydepth:1;
```

$$(\%o6) \qquad 1$$

```
(%i7) apply1(G(G(4,6),2+k),regla1);
```

$$(\%o7) \qquad \frac{H\left(G\left(4,6\right), k+2\right)}{G\left(4,6\right)}$$

Quizás sea nuestro deseo realizar la sustitución desde dentro hacia fuera, controlando también a qué niveles se aplica; `applyb1` y `maxapplyheight` son las claves ahora.

```
(%i8) maxapplyheight:1;
```

$$(\%o8) \qquad 1$$

```
(%i9) applyb1(G(G(4,6),2+k),regla1);
```

$$(\%o9) \qquad G\left(\frac{H\left(4,6\right)}{4}, k+2\right)$$

Obsérvese que hemos estado controlando el comportamiento de las funciones G y H sin haberlas definido explícitamente, pero nada nos impide hacerlo,

```
(%i10) H(u,v):= u^v+1;
                                        v
(%o10)                     H(u, v) := u  + 1
(%i11) applyb1(G(G(4,6),2+k),regla1);
```

(%o11) $$G\left(\frac{4097}{4}, k+2\right)$$

Continuemos esta exposición con un ejemplo algo más sofisticado. Supongamos que cierta función F verifica la igualdad

$$F(x_1 + x_2 + \ldots + x_n) = F(x_1) + F(x_2) + \ldots + F(x_n) + x_1 x_2 \ldots x_n$$

y que queremos definir una regla que realice esta transformación. Puesto que la regla se aplicará sólo cuando el argumento de F sea un sumando, necesitamos una función de predicado que defina este patrón,

```
(%i12) esunasuma(expr):= not atom(expr) and op(expr)="+" $
(%i13) matchdeclare(z,esunasuma)$
```

En la definición de la nueva regla, le indicamos a Maxima que cada vez que se encuentre con la función F y que ésta tenga como argumento una suma, la transforme según se indica: `map(F, args(z))` aplica la función a cada uno de los sumandos de `z`, sumando después estos resultados con la función `apply`, finalmente a este resultado se le suma el producto de los sumandos de `z`,

```
(%i14) defrule(regla2,
               F(z),
               apply("+",map(F, args(z))) + apply("*", args(z)));
```

(%o14) $$regla_2 : F\,(z) \to apply\,(+, map\,(F, args\,(z))) + apply\,(*, args\,(z))$$

Veamos unos cuantos resultados de la aplicación de esta regla,

```
(%i15) apply1(F(a+b), regla2);
```

(%o15) $$F\,(b) + a\,b + F\,(a)$$

```
(%i16) apply1(F(a+b), regla2) - apply1(F(a+c), regla2) ;
```

(%o16) $$-F\,(c) - a\,c + F\,(b) + a\,b$$

```
(%i17) apply1(F(a+b+c), regla2);
```

(%o17) $$F(c) + a\,b\,c + F(b) + F(a)$$

```
(%i18) apply1(F(4), regla2);
```

(%o18) $$F(4)$$

```
(%i19) apply1(F(4+5), regla2);
```

(%o19) $$F(9)$$

```
(%i20) simp:false$  /* inhibe la simplificación de Maxima */
(%i21) apply1(F(4+5), regla2);
```

(%o21) $$4\,5 + (F(4) + F(5))$$

```
(%i22) simp:true$   /* restaura la simplificación de Maxima */
(%i23) %o21;
```

(%o23) $$F(5) + F(4) + 20$$

En los ejemplos recién vistos, hemos tenido que indicar expresamente a Maxima en qué momento debe aplicar una regla. Otro mecanismo de definición de reglas es el aportado por `tellsimp` y `tellsimpafter`; el primero define reglas a aplicar antes de que Maxima aplique las suyas propias, y el segundo para reglas que se aplican después de las habituales de Maxima.

Otra función útil en estos contextos es `declare`, con la que se pueden declarar propiedades algebraicas a nuevos operadores. A continuación se declara una operación de nombre `o` como *infija* (esto es, que sus operandos se colocan a ambos lados del operador); obsérvese que sólo cuando la operación se declara como conmutativa Maxima considera que `a o b` es lo mismo que `b o a`, además, no ha sido necesario definir qué es lo que hace la nueva operación con sus argumentos,

```
(%i24) infix("o");
```

(%o24) o

```
(%i25) is(a o b = b o a);
```

(%o25) false

```
(%i26) declare("o", commutative)$
(%i27) is(a o b = b o a);
```

(%o27) true

5. Cálculo

5.1. Funciones matemáticas

En Maxima están definidas un gran número de funciones, algunas de las cuales se presentan en la Figura 6. A continuación se desarrollan algunos ejemplos sobre su uso.

Empecemos por algunos ejemplos,

```
(%i1) log(%e);
```

$$(\%o1) \qquad 1$$

```
(%i2) abs(-3^-x);
```

$$(\%o2) \qquad \frac{1}{3^x}$$

```
(%i3) signum(-3^-x);
```

$$(\%o3) \qquad -1$$

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!;
```

$$(\%o4) \qquad 0$$

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

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

$$(\%o5) \qquad 0$$

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

```
(%i6) asin(1);
```

$$(\%o6) \qquad \frac{\pi}{2}$$

```
(%i7) float(%);
```

$$(\%o7) \qquad 1.570796326794897$$

Pero si damos el argumento en formato decimal, Maxima también devuelve el resultado en este mismo formato, sin necesidad de hacer uso de `float`,

`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)`	$\binom{m}{n} = \frac{m(m-1)...[m-(n-1)]}{n!}$
`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 6: Algunas funciones de Maxima.

```
(%i8) asin(1.0);
```

(%o8) 1.570796326794897

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

```
(%i9) fpprec:50$ bfloat(%o8);
```

(%o9) $1.5707963267948966192313216916397514420985846996876_B \times 10^0$

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 `??` y utilizando como argumento `airy`, `elliptic` o `bessel`, según el caso. Por ejemplo,

```
(%i13) ?? airy

 0: airy_ai  (Funciones y variables para las funciones especiales)
 1: airy_bi  (Funciones y variables para las funciones especiales)
 2: airy_dai  (Funciones y variables para las funciones especiales)
 3: airy_dbi  (Funciones y variables para las funciones especiales)
Enter space-separated numbers, ‘all’ or ‘none’: 0

 -- Función: airy_ai (<x>)
     Función Ai de Airy, tal como la definen Abramowitz y Stegun,
     Handbook of Mathematical Functions, Sección 10.4.
```

```
La ecuación de Airy 'diff (y(x), x, 2) - x y(x) = 0' tiene dos
soluciones linealmente independientes, 'y = Ai(x)' y 'y = Bi(x)'.
La derivada 'diff (airy_ai(x), x)' es 'airy_dai(x)'.

Si el argumento 'x' es un número decimal real o complejo, se
devolverá el valor numérico de 'airy_ai' siempre que sea posible.

Véanse 'airy_bi', 'airy_dai' y 'airy_dbi'.
```

Maxima reconoce los dominios en el plano complejo de las funciones; los siguientes ejemplos lo demuestran:

```
(%i14) sin(%i);
```

(%o14) $i \sinh 1$

```
(%i15) log(-3.0);
```

(%o15) $3.141592653589793\,i + 1.09861228866811$

```
(%i16) asin(2.0);
```

(%o16) $1.570796326794897 - 1.316957896924817\,i$

5.2. 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);
```

(%o3) $$-\frac{5}{3}$$

```
(%i4) limit(1/(x-1)^2,x,1);
```

(%o4) $$\infty$$

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) $$\infty$$

```
(%i7) limit(1/(x-1),x,1,minus);
```

(%o7) $$-\infty$$

5.3. 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);  /* primera derivada */
```

(%o1) $$x^{\log(a\,x)}\left(\frac{\log(a\,x)}{x} + \frac{\log x}{x}\right)$$

```
(%i2) diff(x^log(a*x),x,2); /*derivada segunda*/
```

(%o2) $$x^{\log(a\,x)}\left(\frac{\log(a\,x)}{x}+\frac{\log x}{x}\right)^2+x^{\log(a\,x)}\left(-\frac{\log(a\,x)}{x^2}-\frac{\log x}{x^2}+\frac{2}{x^2}\right)$$

```
(%i3) factor(%); /* ayudamos a Maxima a mejorar el resultado */
```

(%o3) $$x^{\log(a\,x)-2}\left(\log^2(a\,x)+2\,\log x\,\log(a\,x)-\log(a\,x)+\log^2 x-\log x+2\right)$$

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 programa que factorizase la salida `(%o2)`.

Pedimos ahora a Maxima que nos calcule el siguiente resultado que implica derivadas parciales,

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

```
(%i4) diff(exp(x)*sin(y)*tan(z),x,3,y,5,z,2);
```

(%o4) $$2\,e^x\,\cos y\,\sec^2 z\,\tan z$$

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 $\frac{\vartheta z^2}{\vartheta u\vartheta v}$.

```
(%i5) depends(z,[x,y],[x,y],[u,v]);
```

(%o5) $$[z(x,y),x(u,v),y(u,v)]$$

```
(%i6) diff(z,v,1);
```

(%o6) $$\frac{d}{d\,v}\,y\left(\frac{d}{d\,y}\,z\right)+\frac{d}{d\,v}\,x\left(\frac{d}{d\,x}\,z\right)$$

```
(%i7) diff(z,y,1,v,1);
```

(%o7) $$\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,y^2}\,z\right)+\frac{d}{d\,v}\,x\left(\frac{d^2}{d\,x\,d\,y}\,z\right)$$

```
(%i8) diff(z,u,1,v,1);
```

(%o8) $$\begin{array}{l}\frac{d}{d\,u}\,y\left(\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,y^2}\,z\right)+\frac{d}{d\,v}\,x\left(\frac{d^2}{d\,x\,d\,y}\,z\right)\right)+\frac{d^2}{d\,u\,d\,v}\,y\left(\frac{d}{d\,y}\,z\right)+\\ \frac{d}{d\,u}\,x\left(\frac{d}{d\,v}\,x\left(\frac{d^2}{d\,x^2}\,z\right)+\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,x\,d\,y}\,z\right)\right)+\frac{d^2}{d\,u\,d\,v}\,x\left(\frac{d}{d\,x}\,z\right)\end{array}$$

La función `depends` admite un número par de argumentos, los cuales pueden ser variables o listas de variables. Las que ocupan los lugares impares son las variables dependientes y las que están en las posiciones pares las independientes; denominación ésta que tiene aquí un sentido relativo, ya que una misma variable, como la `x` de la entrada `%i5`, puede actuar como independiente respecto de `z` y como dependiente respecto de `u` y `v`.

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

```
(%i9) dependencies;
```

(%o9) $$[z\left(x,y\right),x\left(u,v\right),y\left(u,v\right)]$$

También podemos eliminar algunas de la dependencias previamente especificadas,

```
(%i10) remove(x,dependency);
```

(%o10) done

```
(%i11) dependencies;
```

(%o11) $$[z\left(x,y\right),y\left(u,v\right)]$$

```
(%i12) diff(z,y,1,v,1);
```

(%o12) $$\frac{d}{d\,v}\,y\,\left(\frac{d^2}{d\,y^2}\,z\right)$$

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`,

```
(%i13) depends(y,x)$
```

```
(%i14) diff(x^2+y^3=2*x*y,x);
```

(%o14) $$3\,y^2\,\left(\frac{d}{d\,x}\,y\right)+2\,x=2\,x\,\left(\frac{d}{d\,x}\,y\right)+2\,y$$

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,

```
(%i15) diff(x^2);
```

(%o15) $$2\,x\,del\left(x\right)$$

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

```
(%i16) diff(x^2+y^3=2*x*y);
```

(%o16)
$$3\,y^2\,del\,(y)+\left(3\,y^2\,\left(\frac{d}{d\,x}\,y\right)+2\,x\right)\,del\,(x)=2\,x\,del\,(y)+\left(2\,x\,\left(\frac{d}{d\,x}\,y\right)+2\,y\right)\,del\,(x)$$

Recuérdese que durante este cálculo estaba todavía activa la dependencia declarada en la entrada `(%i13)`.

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 $x=1$,

```
(%i17) taylor((x*log(x))/(x^2-1),x,1,3);
```

(%o17)
$$\frac{1}{2}-\frac{(x-1)^2}{12}+\frac{(x-1)^3}{12}+\cdots$$

```
(%i18) expand(%);
```

(%o18)
$$\frac{x^3}{12}-\frac{x^2}{3}+\frac{5\,x}{12}+\frac{1}{3}$$

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,

```
(%i19) taylor(exp(x^2*sin(x*y)),[x,2,2],[y,0,2]);
```

(%o19)
$$1+8\,y+32\,y^2+\cdots+\left(12\,y+96\,y^2+\cdots\right)\,(x-2)+\left(6\,y+120\,y^2+\cdots\right)\,(x-2)^2+\cdots$$

```
(%i20) expand(%);
```

(%o20)
$$120\,x^2\,y^2-384\,x\,y^2+320\,y^2+6\,x^2\,y-12\,x\,y+8\,y+1$$

En ocasiones resulta necesario operar con funciones cuyas derivadas son desconocidas, quizás porque la propia función lo es. Esta situación puede llevar a que Maxima devuelva expresiones realmente complicadas de manipular. Un ejemplo es el siguiente, en el que trabajamos con una función f arbitraria

```
(%i21) taylor(f(x + x^2),x,1,1);
```

(%o21)
$$f\,(2)+\left(\frac{d}{d\,x}\,f\left(x^2+x\right)\Big|_{x=1}\right)\,(x-1)+\cdots$$

Podemos facilitar las cosas si le definimos una función derivada a f, a la que llamaremos df,

```
(%i22) gradef(f(x),df(x))$
(%i23) taylor(f(x+x^2),x,1,1);
```

(%o23)
$$f\,(2)+3\,df\,(2)\,(x-1)+\cdots$$

El paquete `pdiff`, que se encuentra en la carpeta `share/contrib/`, aporta una solución alternativa a este problema.

5.4. 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);
```

$$(\%o1) \qquad \frac{3\,\sin^2 x - 1}{3\,\sin^3 x}$$

```
(%i2) integrate(a[3]*x^3+a[2]*x^2+a[1]*x+a[0],x);
```

$$(\%o2) \qquad \frac{a_3\,x^4}{4} + \frac{a_2\,x^3}{3} + \frac{a_1\,x^2}{2} + a_0\,x$$

que nos devuelve, respectivamente, las integrales

$$\int \frac{\cos^3 x}{\sin^4 x} dx$$

y

$$\int (a_3x^3 + a_2x^2 + a_1x + a_0)dx.$$

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);
```

$$(\%o3) \qquad 2\left(\frac{2\,\log 7}{3} - \frac{2\,\log 5}{3} + \frac{\log 4}{3} - \frac{\log 2}{3}\right)$$

```
(%i4) %,numer;   /*aproximación decimal*/
```

$$(\%o4) \qquad 0.9107277692015807$$

```
(%i5) integrate(asin(x),x,0,u);
Is  u  positive, negative, or zero?

positive;
```

$$(\%o5) \qquad u\,\arcsin u + \sqrt{1-u^2} - 1$$

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. En previsión de situaciones como esta, podemos darle al sistema toda la información relevante sobre los parámetros utilizados antes de pedirle el cálculo de la integral,

```
(%i6) assume(u>0);
```

(%o6) $[u > 0]$

```
(%i7) integrate(asin(x),x,0,u);
```

(%o7) $u\ \arcsin u + \sqrt{1-u^2} - 1$

Cuando Maxima no puede resolver la integral, siempre queda el recurso de los métodos numéricos. El paquete `quadpack`, escrito inicialmente en Fortran y portado a Lisp para Maxima, es el encargado de estos menesteres; dispone de varias funciones, pero nos detendremos tan sólo en dos de ellas, siendo la primera la utilizada para integrales definidas en intervalos acotados,

```
(%i8) /* El integrador simbólico no puede con esta integral */
      integrate(exp(sin(x)),x,2,7);
```

(%o8) $$\int_2^7 e^{\sin x}\ dx$$

```
(%i9) /* Resolvemos numéricamente */
      quad_qag(exp(sin(x)),x,2,7,3);
```

(%o9) $\left[4.747336298073747, 5.27060206376023 \times 10^{-14}, 31, 0\right]$

La función `quad_qag` tiene un argumento extra, que debe ser un número entero entre 1 y 6, el cual hace referencia al algoritmo que la función debe utilizar para la cuadratura; la regla heurística a seguir por el usuario es dar un número tanto más alto cuanto más oscile la función en el intervalo de integración. El resultado que obtenemos es una lista con cuatro elementos: el valor aproximado de la integral, la estimación del error, el número de veces que se tuvo que evaluar el integrando y, finalmente, un código de error que será cero si no surgieron problemas.

La otra función de integración numérica a la que hacemos referencia es `quad_qagi`, a utilizar en intervalos no acotados. En el siguiente ejemplo se pretende calcular la probabilidad de que una variable aleatoria χ^2 de $n = 4$ grados de libertad, sea mayor que la unidad ($\Pr(\chi_4^2 > 1)$),

```
(%i10) n:4$
(%i11) integrate(x^(n/2-1)*exp(-y/2)/2^(n/2)*gamma(n/2),x,1,inf);
Integral is divergent
 -- an error.  To debug this try debugmode(true);

(%i12) quad_qagi(x^(n/2-1)*exp(-x/2)/2^(n/2)*gamma(n/2),x,1,inf);
```

$$(\%o12) \qquad \left[.9097959895689501, 1.913452127046495 \times 10^{-10}, 165, 0\right]$$

El integrador simbólico falla emitiendo un mensaje sobre la divergencia de la integral. La función `quad_qagi` ha necesitado 165 evaluaciones del integrando para alcanzar una estimación numérica de la integral, la cual se corresponde aceptablemente con la estimación que hacen los algoritmos del paquete de distribuciones de probabilidad (ver Sección 6.1).

Otras funciones de cuadratura numérica son `quad_qags`, `quad_qawc`, `quad_qawf`, `quad_qawo` y `quad_qaws`, cuyas peculiaridades podrá consultar el lector interesado en el manual de referencia.

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

```
(%i13) laplace(k*exp(-k*x),x,p);
```

$$(\%o13) \qquad \frac{k}{p+k}$$

y calculando la transformada inversa volvemos al punto de partida,

```
(%i14) ilt(%,p,x);
```

$$(\%o14) \qquad k\,e^{-k\,x}$$

La transformada de Fourier de una función se reduce a la de Laplace cuando el argumento p toma el valor $-it$, 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

```
(%i15) laplace(k*exp(-k*x),x,-%i*t);
```

$$(\%o15) \qquad \frac{k}{k-i\,t}$$

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.

5.5. 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) /* ecuación de variables separadas */
      ec:(x-1)*y^3+(y-1)*x^3*'diff(y,x)=0;
```

$$(\%o1) \qquad x^3\ (y-1)\left(\frac{d}{d\,x}\,y\right)+(x-1)\ y^3=0$$

siendo obligatorio el uso de la comilla simple (') 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);
```

$$(\%o2) \qquad \frac{2\,y-1}{2\,y^2}=\%c-\frac{2\,x-1}{2\,x^2}$$

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);
```

$$(\%o3) \qquad \frac{2\,y-1}{2\,y^2}=-\frac{x^2+72\,x-36}{72\,x^2}$$

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

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

$$(\%o4) \qquad \frac{4\,x\,y^3+x^4}{4}=\%c$$

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);
```

$$(\%o5) \qquad \frac{\log\left(y^2+x^2-2\,x+1\right)+2\ \arctan\left(\frac{x-1}{y}\right)}{4}=\%c$$

```
(%i6) /* ecuación exacta */
      ode2((4*x^3+8*y)+(8*x-4*y^3)*'diff(y,x),y,x);
```

$$(\%o6) \qquad -y^4 + 8\,x\,y + x^4 = \%c$$

```
(%i7) /* Bernoulli */
      ode2('diff(y,x)-y+sqrt(y),y,x);
```

$$(\%o7) \qquad 2\,\log\left(\sqrt{y} - 1\right) = x + \%c$$

```
(%i8) solve(%,y);
```

$$(\%o8) \qquad \left[y = e^{x+\%c} + 2\,e^{\frac{x}{2}+\frac{\%c}{2}} + 1\right]$$

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);
```

$$(\%o9) \qquad \frac{d}{d\,x}\,y = \frac{d^2}{d\,x^2}\,y + x$$

```
(%i10) ode2(%,y,x);
```

$$(\%o10) \qquad y = \%k_1\,e^x + \frac{x^2 + 2\,x + 2}{2} + \%k_2$$

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 $y' = \left.\frac{dy}{dx}\right|_{x=1} = 2$, haremos uso de la instrucción `ic2`,

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

$$(\%o11) \qquad y = \frac{x^2 + 2\,x + 2}{2} - \frac{7}{2}$$

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);
```

(%o12) $$y = -\frac{35\,e^{x+1}}{6\,e^3-6} + \frac{x^2+2\,x+2}{2} + \frac{15\,e^3+20}{6\,e^3-6}$$

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)]);
```

(%o13) $$\left[f\left(x\right) = \frac{\left(2\,g(0)-f(0)\right)e^{-x}}{3} - \frac{\left(2\,g(0)-4\,f(0)\right)e^{2\,x}}{3},\ g\left(x\right) = \frac{\left(4\,g(0)-2\,f(0)\right)e^{-x}}{3} - \frac{\left(g(0)-2\,f(0)\right)e^{2\,x}}{3}\right]$$

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)]);
```

(%o17) $$\left[f\left(x\right) = x\,e^{2\,x} + e^{2\,x} - 2\,e^{-x}, g\left(x\right) = -2\,x\,e^{2\,x} + 2\,e^{2\,x} + e^{-x}, h\left(x\right) = x\,e^{2\,x} + e^{-x}\right]$$

La función `desolve` también nos va a permitir resolver ecuaciones de orden mayor que dos. En el siguiente ejemplo, abordamos la resolución de la ecuación

$$\frac{d^3y(x)}{dx^3} + \frac{d^2y(x)}{dx^2} + \frac{dy(x)}{dx} + y(x) = 0$$

bajo las condiciones que se describen a continuación,

```
(%i18) atvalue('diff(y(x),x,2),x=0,v)$
(%i19) atvalue('diff(y(x),x),x=0,u)$
(%i20) atvalue(y(x),x=0,w)$
```

Ahora resolvemos,

```
(%i21) desolve('diff(y(x),x,3)+'diff(y(x),x,2)+'diff(y(x),x)+y(x)=0, y(x));
```

$$(\%o21) \qquad y(x) = \frac{(w+v+2\,u)\,\sin x}{2} + \frac{(w-v)\,\cos x}{2} + \frac{(w+v)\,e^{-x}}{2}$$

El paquete `plotdf` permite generar campos de direcciones, bien de ecuaciones diferenciales de primer orden

$$\frac{dy}{dx} = F(x,y),$$

bien de sistemas

$$\begin{cases} \frac{dx}{dt} = G(x,y) \\ \frac{dy}{dt} = F(x,y) \end{cases}$$

Los argumentos a pasar a la función `plotdf` son la función F, en el primer caso, y una lista con las funciones F y G en el segundo. Las variables serán siempre x e y. Como ejemplo, pidamos a Maxima que genere el campo de direcciones de la ecuación diferencial $\frac{dy}{dx} = 1 + y + y^2$

```
(%i22) load(plotdf)$
(%i23) plotdf(1 + y + y^2);
```

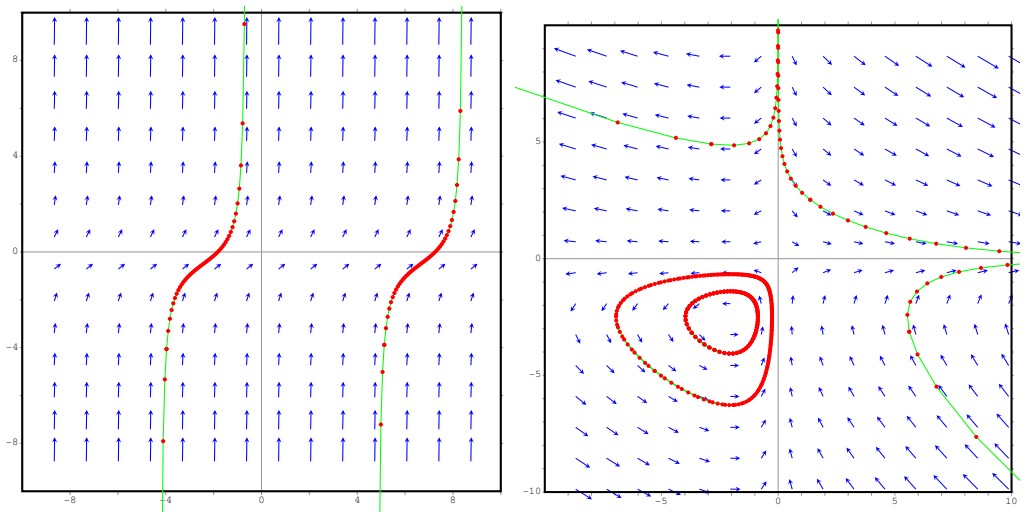
El gráfico que resulta es el de la Figura 7 *a)*, en el que además se observan dos trayectorias que se dibujaron de forma interactiva al hacer clic sobre dos puntos del plano.

La función `plotdf` admite varias opciones, algunas de las cuales aprovechamos en el siguiente ejemplo. Supongamos el modelo predador-presa de Lotka-Volterra, dependiente de dos parámetros h y k,

$$\begin{cases} \frac{dx}{dt} = 2x + hxy \\ \frac{dy}{dt} = -x + kxy \end{cases}$$

El siguiente código permite generar el campo de direcciones correspondiente, dándoles inicialmente a h y k los valores -1.2 y 0.9, respectivamente; además, aparecerán sobre la ventana gráfica dos barras de deslizamiento para alterar de forma interactiva estos dos parámetros y ver los cambios que se producen en el campo.

```
(%i24) plotdf([2*x+k*x*y, -y+h*x*y],
              [parameters,"k=-1.2,h=0.9"],
              [sliders,"k=-2:2,h=-2:2"]);
```



a) *b)*

Figura 7: Campos de direcciones creados con la función 'plotdf': *a)* campo de la ecuación $\frac{dy}{dx} = 1 + y + y^2$; *b)* campo correspondiente al modelo predador-presa.

En la Figura 7 *b)* se observa el gráfico obtenido despúes de pedir trayectorias concretas que pasan por varios puntos del plano (en el archivo gráfico no aparecen las barras de deslizamiento).

Cuando Maxima no es capaz de resolver la ecuación propuesta, se podrá recurrir al método numérico de Runge-Kutta, el cual se encuentra programado en el paquete `diffeq`. Como primer ejemplo, nos planteamos la resolución de la ecuación

$$\frac{dy}{dt} = -2y^2 + \exp(-3t),$$

con la condición $y(0) = 1$. La función `ode2` es incapaz de resolverla:

```
(%i25) ec: 'diff(y,t)+2*y^2-exp(-3*t)=0;
```

$$(\%o25) \qquad \frac{d}{d\,t}\,y + 2\,y^2 - e^{-3\,t} = 0$$

```
(%i26) ode2(ec,y,t);
```

(%o26) false

Abordamos ahora el problema con un enfoque numérico, para lo cual definimos la expresión

$$f(t, y) = \frac{dy}{dt} = -2y^2 + \exp(-3t)$$

en Maxima,

```
(%i27) load(diffeq)$
(%i28) f(t,y):= -2*y^2+exp(-3*t) $
(%i29) res: runge1(f,0,5,0.5,1);
```

(%o29)
[[0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0] ,
[1.0, .5988014211752297, .4011473182183033, .2915932807721147 ,
.2260784415237641, 0.183860316087325, .1547912058210609, .1336603954558797,
.1176289334565761, .1050518038293819, .0949194462538843 9] ,
[−1.0, −0.49399612385452, −.2720512734596094, −.1589442862446483 ,
−.0997441712669617 1, −.0670561472933142 6, −.0477972249949894 2, −.0357026661774945 7,
−.0276669877599098 8, −.0220703920165274 7, −.0180190966519675 9]]

```
(%i30) draw2d(
         points_joined = true,
         point_type    = dot,
         points(res[1],res[2]),
         terminal      = eps)$
```

La función `runge1` necesita cinco argumentos: la función derivada $\frac{dy}{dt}$, los valores inicial, t_0, y final, t_1, de la variable independiente, la amplitud de los subintervalos y el valor que toma y en t_0. El resultado es una lista que a su vez contiene tres listas: las abscisas t, las ordenadas y y las correspondientes derivadas. Al final del ejemplo anterior, se solicita la representación gráfica de la solución, cuyo aspecto es el mostrado por la Figura 8 *a)*. (Consúltese la sección 7 para una descripción de la función `draw2d`.)

Nos planteamos ahora la resolución de la ecuación diferencial de segundo orden

$$\frac{d^2y}{dt^2} = 0.2(1-y^2)\frac{dy}{dt} - y,$$

con $y(0) = 0.1$ y $y'(0) = 1$ para lo cual definimos en Maxima la función

$$g(t, y, y') = \frac{d^2y}{dt^2} = 0.2(1-y^2)y' - y,$$

```
(%i31) g(t,y,yp) := 0.2*(1-y^2)*yp - y $
(%i32) res: runge2(g,0,100,0.1,0.1,1)$
(%i33) midibujo(i,j):= draw2d(points_joined = true,
                              point_type    = dot,
                              points(res[i],res[j]),
                              terminal      = eps)$
(%i34) midibujo(1,2)$
(%i35) midibujo(2,3)$
(%i36) midibujo(2,4)$
```

La función `runge2` necesita seis argumentos: la función derivada $\frac{d^2y}{dt^2}$, los valores inicial, t_0, y final, t_1, de la variable independiente, la amplitud de los subintervalos y los valores

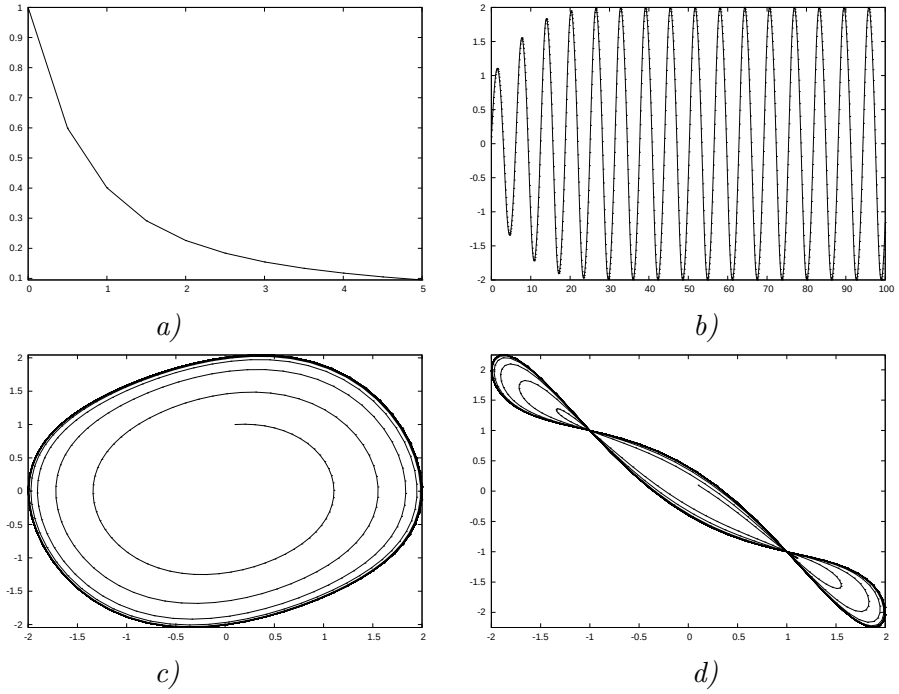


Figura 8: Resolución numérica de ecuaciones diferenciales con 'diffeq': *a)*, solución de la ecuación de primer orden $\frac{dy}{dt} = -2y^2 + \exp(-3t), y(0) = 1$; *b)*, solución de la ecuación de segundo orden $\frac{d^2y}{dt^2} = 0.2(1 - y^2)\frac{dy}{dt} - y$, con las condiciones $y(0) = 0.1$ y $y'(0) = 1$; *c)*, diagrama (y, y'); *d)*, diagrama (y, y'').

que toman y y su primera derivada en t_0. El resultado es una lista que a su vez contiene cuatro listas: las abscisas t, las ordenadas y, las correspondientes primeras derivadas y, por último, las segundas derivadas. Al final de este último ejemplo se solicitan algunos gráficos asociados a la solución, los formados con los pares (t, y), (y, y') y (y, y''), que son los correspondientes a los apartados *b)*, *c)* y *d)* de la Figura 8.

El paquete `dynamics` dispone de otra rutina para el método de Runge-Kutta, `rk`, que permite la resolución de sistemas de ecuaciones difereciales. Para resolver el sistema

$$\begin{cases} \frac{dx}{dt} = 4 - x^2 - 4y^2 \\ \frac{dy}{dt} = y^2 - x^2 + 1 \end{cases}$$

cargamos el paquete y ejecutamos la sentencia correspondiente, junto con el gráfico asociado que no mostramos.

```
(%i37) load(dynamics)$
(%i38) rk([4-x^2-4*y^2,y^2-x^2+1],[x,y],[-1.25,0.75],[t,0,4,0.02])$
(%i39) draw3d(
         points_joined = true,
```

```
        point_type    = dot,
        points(%),
        terminal      = eps)$
```

Para más información sobre la sintaxis de esta función, ejecútese `? rk`. El paquete `dynamics` contiene otras funciones relacionadas con los sistemas dinámicos y los fractales.

5.6. Vectores y campos vectoriales

En Maxima, los vectores se introducen como simples listas (Sección 3.3), siendo el caso que con ellas se pueden realizar las operaciones de adición, producto por un número y producto escalar de vectores,

```
(%i1) [1,2,3]+[a,b,c];
```

$$(\%o1) \qquad [a+1, b+2, c+3]$$

```
(%i2) s*[a,b,c];
```

$$(\%o2) \qquad [a\,s, b\,s, c\,s]$$

```
(%i3) [1,2,3].[a,b,c]; /* producto escalar */
```

$$(\%o3) \qquad 3\,c + 2\,b + a$$

El cálculo del módulo de un vector se puede hacer mediante la definición previa de una función al efecto:

```
(%i4) modulo(v):=
          if listp(v)
             then sqrt(apply("+",v^2))
             else error("Mucho ojito: ", v, " no es un vector !!!!")$
(%i5) xx:[a,b,c,d,e]$
(%i6) yy:[3,4,-6,0,4/5]$
(%i7) modulo(xx-yy);
```

$$(\%o7) \qquad \sqrt{\left(e-\frac{4}{5}\right)^2 + d^2 + (c+6)^2 + (b-4)^2 + (a-3)^2}$$

Los operadores diferenciales que son de uso común en el ámbito de los campos vectoriales están programados en el paquete `vect`, lo que implica que debe ser cargado en memoria antes de ser utilizado. Sigue a continuación una sesión de ejemplo sobre cómo usarlo.

Partamos de los campos escalares $\phi(x,y,z) = -x+y^2+z^2$ y $\psi(x,y,z) = 4x+\log(y^2+z^2)$ y demostremos que sus superficies de nivel son ortogonales probando que $\nabla\phi \cdot \nabla\psi = 0$, siendo ∇ el operador gradiente,

```
(%i8) /* Se carga el paquete */
      load(vect)$
(%i9) /* Se definen los campos escalares */
      phi: y^2+z^2-x$  psi:log(y^2+z^2)+4*x$
(%i11) grad(phi) . grad(psi);
```

$$(\%o11) \qquad grad\left(z^2+y^2-x\right)\cdot grad\left(\log\left(z^2+y^2\right)+4\,x\right)$$

Como se ve, Maxima se limita a devolvernos la misma expresión que le introducimos; el estilo de trabajo del paquete `vect` requiere el uso de dos funciones: `express` y `ev`, la primera para obtener la expresión anterior en términos de derivadas y la segunda para forzar el uso de éstas.

```
(%i12) express(%);
```

$$(\%o12) \qquad \begin{array}{c} \frac{d}{d\,z}\left(z^2+y^2-x\right)\left(\frac{d}{d\,z}\left(\log\left(z^2+y^2\right)+4\,x\right)\right)+ \\ \frac{d}{d\,y}\left(z^2+y^2-x\right)\left(\frac{d}{d\,y}\left(\log\left(z^2+y^2\right)+4\,x\right)\right)+ \\ \frac{d}{d\,x}\left(z^2+y^2-x\right)\left(\frac{d}{d\,x}\left(\log\left(z^2+y^2\right)+4\,x\right)\right) \end{array}$$

```
(%i13) ev(%,diff);
```

$$(\%o13) \qquad \frac{4\,z^2}{z^2+y^2}+\frac{4\,y^2}{z^2+y^2}-4$$

```
(%i14) ratsimp(%);
```

$$(\%o14) \qquad 0$$

Al final, hemos tenido que ayudar un poco a Maxima para que terminase de reducir la última expresión.

Sea ahora el campo vectorial definido por $\mathbf{F}=xy\mathbf{i}+x^2z\mathbf{j}-e^{x+y}\mathbf{k}$ y pidámosle a Maxima que calcule su divergencia, $(\nabla\cdot\mathbf{F})$, rotacional $(\nabla\times\mathbf{F})$y laplaciano $(\nabla^2\mathbf{F})$

```
(%i15) F: [x*y,x^2*z,exp(x+y)]$
(%i16) div (F);  /* divergencia */
```

$$(\%o16) \qquad div\left(\left[x\,y,x^2\,z,e^{y+x}\right]\right)$$

```
(%i17) express (%);
```

$$(\%o17) \qquad \frac{d}{d\,y}\left(x^2\,z\right)+\frac{d}{d\,z}\,e^{y+x}+\frac{d}{d\,x}\left(x\,y\right)$$

```
(%i18) ev (%, diff);
```

(%o18) $$y$$

```
(%i19) curl (F);  /* rotacional */
```

(%o19) $$curl\left(\left[x\,y, x^2\,z, e^{y+x}\right]\right)$$

```
(%i20) express (%);
```

(%o20) $$\left[\frac{d}{d\,y}\,e^{y+x} - \frac{d}{d\,z}\,\left(x^2\,z\right), \frac{d}{d\,z}\,\left(x\,y\right) - \frac{d}{d\,x}\,e^{y+x}, \frac{d}{d\,x}\,\left(x^2\,z\right) - \frac{d}{d\,y}\,\left(x\,y\right)\right]$$

```
(%i21) ev (%, diff);
```

(%o21) $$\left[e^{y+x} - x^2, -e^{y+x}, 2\,x\,z - x\right]$$

```
(%i22) laplacian (F);  /* laplaciano */
```

(%o22) $$laplacian\left(\left[x\,y, x^2\,z, e^{y+x}\right]\right)$$

```
(%i23) express (%);
```

(%o23) $$\frac{d^2}{d\,z^2}\,\left[x\,y, x^2\,z, e^{y+x}\right] + \frac{d^2}{d\,y^2}\,\left[x\,y, x^2\,z, e^{y+x}\right] + \frac{d^2}{d\,x^2}\,\left[x\,y, x^2\,z, e^{y+x}\right]$$

```
(%i24) ev (%, diff);
```

(%o24) $$\left[0, 2\,z, 2\,e^{y+x}\right]$$

Nótese en todos los casos el usos de la secuencia `express` - `ev`. Si el usuario encuentra incómoda esta forma de operar, siempre podrá definir una función que automatice el proceso; por ejemplo,

```
(%i25) milaplaciano(v):= ev(express(laplacian(v)), diff) $
(%i26) milaplaciano(F);
```

(%o26) $$\left[0, 2\,z, 2\,e^{y+x}\right]$$

Por último, el paquete `vect` incluye también la definición del producto vectorial, al cual le asigna el operador `~`,

```
(%i27) [a, b, c] ~ [x, y, z];
```

(%o27) $$[a, b, c] \sim [x, y, z]$$

```
(%i28) express(%);
```

(%o28) $$[b\,z - c\,y, c\,x - a\,z, a\,y - b\,x]$$

6. Análisis de datos

6.1. Probabilidad

El paquete *distrib* contiene la definición de las funciones de distribución de probabilidad más comunes, tanto discretas (binomial, de Poisson, de Bernoulli, geométrica, uniforme discreta, hipergeométrica y binomial negativa), como continuas (normal, t de Student, χ^2 de Pearson, F de Snedecor, exponencial, lognormal, gamma, beta, uniforme continua, logística, de Pareto, de Weibull, de Rayleigh, de Laplace, de Cauchy y de Gumbel). Para cada una de ellas se puede calcular la probabilidad acumulada, la función de densidad, los cuantiles, medias, varianzas y los coeficientes de asimetría y curtosis:

```
(%i1) load(distrib)$
```

```
(%i2) assume(s>0)$
```

```
(%i3) cdf_normal(x,mu,s);
```

$$(\%o3) \qquad \frac{\operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\,s}\right)}{2}+\frac{1}{2}$$

```
(%i4) pdf_poisson(5,1/s);
```

$$(\%o4) \qquad \frac{e^{-\frac{1}{s}}}{120\,s^5}$$

```
(%i5) quantile_student_t(0.05,25);
```

$$(\%o5) \qquad -1.708140543186975$$

```
(%i6) mean_weibull(3,67);
```

$$(\%o6) \qquad \frac{67\,\Gamma\left(\frac{1}{3}\right)}{3}$$

```
(%i7) var_binomial(34,1/8);
```

$$(\%o7) \qquad \frac{119}{32}$$

```
(%i8) skewness_rayleigh(1/5);
```

(%o8)
$$\frac{\frac{\pi^{\frac{3}{2}}}{4}-\frac{3\sqrt{\pi}}{4}}{\left(1-\frac{\pi}{4}\right)^{\frac{3}{2}}}$$

```
(%i9) kurtosis_gumbel (2,3);
```

(%o9)
$$\frac{12}{5}$$

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

```
(%i10) random(6)+1;
```

(%o10) 3

y una serie de 100 lanzamientos de una moneda,

```
(%i11) makelist(random(2),i,1,100);
```

(%o11)
$$[0,0,1,0,1,1,1,0,1,1,1,0,0,1,0,1,0,1,0,1,1,1,1,0,0,0,1,0,1,0,1,1,0,$$
$$0,1,1,1,1,0,0,0,1,0,1,0,1,0,1,0,0,1,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0,0,$$
$$1,1,1,0,1,1,1,0,1,0,0,0,0,1,1,0,0,1,0,1,0,1,1,0,0,1,1,0,0,1,0,0,1]$$

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 $[0, r)$,

```
(%i12) random(6.0);
```

(%o12) 0.373047098775396

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

```
(%i13) nueva_semilla: make_random_state(true)$
```

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

```
(%i14) 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

```
(%i15) makelist(random(6),i,1,10);
```

(%o15) $[3, 0, 0, 5, 0, 5, 1, 4, 4, 5]$

```
(%i16) makelist(random(6),i,1,10);
```

(%o16) $[0, 4, 0, 0, 5, 4, 0, 0, 1, 5]$

```
(%i17) makelist(random(6),i,1,10);
```

(%o17) $[3, 3, 3, 1, 3, 2, 1, 5, 2, 4]$

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

```
(%i18) semilla: make_random_state(123456789)$
(%i19) set_random_state(semilla)$ makelist(random(6),i,1,10);
```

(%o19) $[4, 4, 0, 1, 0, 3, 2, 5, 4, 4]$

```
(%i20) set_random_state(semilla)$ makelist(random(6),i,1,10);
```

(%o20) $[4, 4, 0, 1, 0, 3, 2, 5, 4, 4]$

```
(%i21) set_random_state(semilla)$ makelist(random(6),i,1,10);
```

(%o21) $[4, 4, 0, 1, 0, 3, 2, 5, 4, 4]$

se verá que las tres secuencias son iguales, ya que antes de generar cada muestra aleatoria reiniciamos el estado del generador. La función `random` y las otras funciones relacionadas con ella discutidas hasta aquí están disponibles sin necesidad de cargar el paquete `distrib`.

Sin embargo, el paquete adicional `distrib` también 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 $Wei(7, \frac{23}{3})$; puesto que ya se ha cargado más arriba el paquete, no es necesario ejecutar nuevamente `load(distrib)`.

```
(%i22) random_binomial(5,1/3,5);
```

```
(%o22)                          [3, 1, 2, 3, 2]
```

```
(%i23) random_poisson(7,5);
```

```
(%o23)                          [8, 3, 10, 5, 6]
```

```
(%i24) random_hypergeometric(15,20,7,5);
```

```
(%o24)                          [4, 2, 4, 3, 2]
```

```
(%i25) random_exp(12.5,5);
```

```
(%o25)  [.05865376074017901, .2604319923173137, .07552948674579418,
         .02948508382731128, .2117111885482312]
```

```
(%i26) random_weibull(7,23/3,5);
```

```
(%o26)  [6.35737206358163, 7.436207845095266, 8.101343432607079,
         7.835164678709573, 6.350884234996046]
```

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

6.2. Estadística descriptiva

Maxima es capaz de realizar ciertos tipos de procesamiento de datos. El paquete `descriptive`, junto con otros, es el que nos puede ayudar en este tipo de tareas.

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`.

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]);
```

(%o2)
$$\begin{pmatrix} 13.17 & 9.29 \\ 14.71 & 16.88 \\ 18.5 & 16.88 \\ 10.58 & 6.63 \\ 13.33 & 13.25 \\ 13.21 & 8.12 \end{pmatrix}$$

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) s1 : read_list (file_search ("pidigits.data"))$
(%i4) length(s1);
```

(%o4) 100

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

```
(%i5) s2:read_matrix(file_search ("wind.data"))$
(%i6) length(s2);
```

(%o6) 100

```
(%i7) s2[%]; /* último registro */
```

(%o7) $[3.58, 6.0, 4.58, 7.62, 11.25]$

Algunas muestras incluyen 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,

```
(%i8) s3:read_matrix(file_search ("biomed.data"))$
(%i9) length(s3);
```

(%o9) 100

```
(%i10) s3[1]; /* primer registro */
```

(%o10) $[A, 30, 167.0, 89.0, 25.6, 364]$

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: `continuous_freq` y `discrete_freq`; 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,

```
(%i11) load("descriptive")$
(%i12) continuous_freq(s1,5);
```

(%o12) $[[0, 1.8, 3.6, 5.4, 7.2, 9.0], [16, 24, 18, 17, 25]]$

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

```
(%i13) discrete_freq(s1);
```

(%o13) $[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [8, 8, 12, 12, 10, 8, 9, 8, 12, 13]]$

```
(%i14) discrete_freq(transpose(col(s3,1))[1]);
```

(%o14) $[[A, B], [35, 65]]$

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

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 (`central_moment`) y no centrales (`noncentral_moment`), coeficientes de variación (`cv`), 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:

```
(%i15) mean(s1);
```

$$(\%o15) \qquad \frac{471}{100}$$

```
(%i16) float(%);
```

$$(\%o16) \qquad 4.71$$

```
(%i17) mean(s2);  /* vector de medias */
```

$$(\%o17) \qquad [9.9485, 10.1607, 10.8685, 15.7166, 14.8441]$$

Cuando la muestra contiene datos en formato decimal, los resultados serán también en este formato, pero si la muestra contiene tan solo datos enteros, los resultados serán devueltos en formato racional. En caso de querer forzar que los resultados sean devueltos siempre en formato decimal, podemos jugar con la variable global `numer`.

```
(%i18) numer: true$
(%i19) var(s1);
```

$$(\%o19) \qquad 8.425899999999999$$

```
(%i20) var(s2);  /* vector de varianzas */
```

$$(\%o20) \qquad [17.22190675000001, 14.98773651, 15.47572875, 32.17651044000001, 24.42307619000001]$$

```
(%i21) /* 1er y 3er cuartiles */
       [quantile(s1,1/4),quantile(s1,3/4)];
```

$$(\%o21) \qquad [2.0, 7.25]$$

```
(%i22) range(s1);
```

$$(\%o22) \qquad 9$$

```
(%i25) range(s2);  /* vector de rangos */
```

$$(\%o25) \qquad [19.67, 20.96, 17.37, 24.38, 22.46]$$

```
(%i26) kurtosis(s1);
```

$$(\%o26) \qquad -1.273247946514421$$

```
(%i27) kurtosis(s2); /* vector de coef. curtosis */
```

$$(\%o27) \qquad [-.2715445622195385, 0.119998784429451, -.4275233490482866, -.6405361979019522, -.4952382132352935]$$

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. La variable global `fpprintprec` es utilizada para limitar el número de decimales a imprimir; lo cual no afecta a la precisión de los cálculos.

```
(%i28) fpprintprec:7$ /* número de dígitos deseados en la salida */
(%i29) cov(s2); /* matriz de covarianzas */
```

$$(\%o29) \qquad \begin{pmatrix} 17.22191 & 13.61811 & 14.37217 & 19.39624 & 15.42162 \\ 13.61811 & 14.98774 & 13.30448 & 15.15834 & 14.9711 \\ 14.37217 & 13.30448 & 15.47573 & 17.32544 & 16.18171 \\ 19.39624 & 15.15834 & 17.32544 & 32.17651 & 20.44685 \\ 15.42162 & 14.9711 & 16.18171 & 20.44685 & 24.42308 \end{pmatrix}$$

```
(%i30) cor(s2); /* matriz de correlaciones */
```

$$(\%o30) \qquad \begin{pmatrix} 1.0 & .8476339 & .8803515 & .8239624 & .7519506 \\ .8476339 & 1.0 & .8735834 & .6902622 & 0.782502 \\ .8803515 & .8735834 & 1.0 & .7764065 & .8323358 \\ .8239624 & .6902622 & .7764065 & 1.0 & .7293848 \\ .7519506 & 0.782502 & .8323358 & .7293848 & 1.0 \end{pmatrix}$$

Las funciones `global_variances` y `list_correlations` calculan, respectivamente, ciertas medidas globales de variación y correlación; para más información, pídase ayuda a Maxima con el comando `describe` o con `?`.

Hay varios tipos de diagramas estadísticos programados en el paquete `descriptive`: `scatterplot` para gráficos de dispersión, `histogram`, `barsplot`, `piechart` y `boxplot` para el análisis homocedástico. A continuación se presentan un par de ejemplos cuyos salidas se presentan en la Figura 9.

```
(%i31) /* Comparando variabilidades */
       boxplot(
          s2,
          terminal = eps,
          title    = "Velocidades del viento")$

(%i32) /* Diagramas de dispersión para las
          tres últimas estaciones meteorológicas */
       scatterplot(
          submatrix(s2,1,2),
          point_type = circle,
          terminal   = eps)$
```

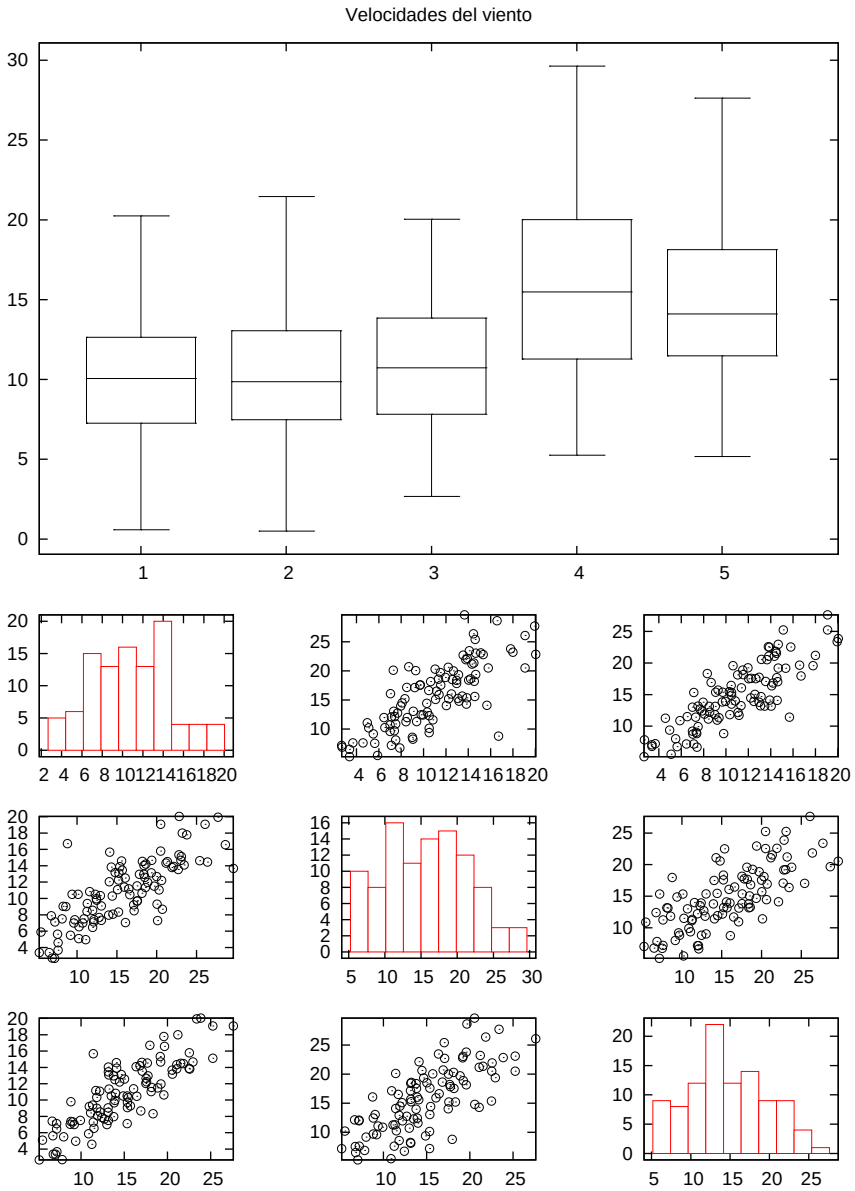


Figura 9: Gráficos para muestras multivariantes: *a)* diagramas de cajas; *b)* diagramas de dispersión para tres variables.

6.3. Estadística inferencial

El paquete `stats` contiene algunos procedimientos clásicos de estadística inferencial. Cada uno de estos procedimientos devuelve un objeto de tipo `inference_result`, el cual guarda todos los resultados obtenidos, aunque sólo muestre una selección de ellos por defecto, los que se consideran de más relevancia; el resto permanecen ocultos y sólo serán visibles a petición del usuario. Los siguientes ejemplos aclararán su uso.

Partamos de una muestra de tamaño 20 y tratemos de contrastar la hipótesis nula de que la media de la población de la cual procede es igual a 32, frente a la alternativa de que la media es diferente a este valor. Para aplicar el test de Student nos interesa saber si podemos considerar la población normal; la función `test_normality` nos puede ser de ayuda,

```
(%i1) load(stats)$
(%i2) x: [28.9,35.0,30.6,31.3,31.9,31.7,29.3,37.8,29.9,36.8,
          28.0,25.3,39.6,30.4,23.3,24.7,38.6,31.3,29.8,31.9]$
(%i3) test_normality(x);
```

$$(\%o3) \quad \left| \begin{array}{c} \text{SHAPIRO - WILK TEST} \\ statistic = 0.9508091 \\ p\_value = 0.3795395 \end{array} \right.$$

Según este resultado, no hay evidencias suficientes para rechazar la hipótesis de normalidad. Hagamos ahora el contraste sobre la media,

```
(%i4) z: test_mean(x,mean=32);
```

$$(\%o4) \quad \left| \begin{array}{c} \text{MEAN TEST} \\ mean\_estimate = 31.305 \\ conf\_level = 0.95 \\ conf\_interval = [29.21482, 33.39518] \\ method = \text{Exact t-test. Unknown variance.} \\ hypotheses = \text{H0: mean} = 32 \text{ , H1: mean} \neq 32 \\ statistic = 0.6959443 \\ distribution = [student\_t, 19] \\ p\_value = 0.4948895 \end{array} \right.$$

En la llamada a la función `test_mean` se ha indicado mediante la opción `mean` el valor de la media en la hipótesis nula. El objeto devuelto por la función consta, como se ve, de varios elementos: el título del objeto, la media muestral, el nivel de confianza, el intervalo de confianza, información sobre el método utilizado, hipótesis a contrastar, estadístico de contraste, su distribución y el p-valor correspondiente. Si ahora quisiésemos extraer la media muestral para utilizar en otros cálculos,

```
(%i5) take_inference(mean_estimate,z);
```

(%o5) 31.305

En caso de querer extraer más de un resultado,

```
(%i6) take_inference([p_value,statistic],z);
```

(%o6) $[0.4948895, 0.6959443]$

Para saber con certeza qué resultados guarda el objeto devuelto haremos uso de la función `items_inference`, con lo que comprobamos que la función `test_mean` no oculta ninguno de sus resultados,

```
(%i7) items_inference(z);
```

(%o7)
[*mean_estimate*, *conf_level*, *conf_interval*, *method*, *hypotheses*, *statistic*, *distribution*, *p_value*]

En caso de que la muestra no procediese de una población normal, y si el tamaño muestral fuese suficientemente grande, se podría hacer uso de la opción `asymptotic=true`, si la desviación típica de la población fuese conocida, por ejemplo $\sigma = 4.5$, podría añadirse la opción `dev=4.5`. En fin, las opciones que admite esta función permiten al usuario tener cierto control sobre el procedimiento; para más información consúltese el manual.

Si la población de referencia no es normal y la muestra es pequeña, siempre se puede acudir a un test no paramétrico para realizar un contraste sobre la mediana,

```
(%i8) signed_rank_test(x,median=32);
```

(%o8)
$$\left|\begin{array}{c} \text{SIGNED RANK TEST} \\ med\_estimate = 30.95 \\ method = \text{Asymptotic test. Ties} \\ hypotheses = \text{H0: med} = 32 \text{ , H1: med} \neq 32 \\ statistic = 75 \\ distribution = [normal, 104.5, 26.78152] \\ p\_value = 0.2706766 \end{array}\right|$$

Existe en Maxima la posibilidad de ajustar por mínimos cuadrados a datos empíricos los parámetros de curvas arbitrarias mediante las funciones del paquete `lsquares`. Vamos a simular una muestra de tamaño 100 de valores x en el intervalo $[0, 10]$. Con estas abscisas calculamos después las ordenadas a partir de la suma de una señal determinista (f) más un ruido gaussiano. Para hacer la simulación necesitamos cargar en primer lugar al paquete `distrib`.

```
(%i9) load(distrib)$
(%i10) abs: makelist(random_continuous_uniform(0,10),k,1,100)$
(%i11) f(x):=3*(x-5)^2$
(%i12) data: apply(matrix,makelist([x, f(x)+random_normal(0,1)],x,abs))$
```

Aunque la señal determinista obedece a una función polinómica de segundo grado, si no lo sabemos a priori intentamos ajustar un polinomio cúbico:

```
(%i13) load(lsquares)$
(%i14) param: lsquares_estimates(data,[x,y],
                                 y=a*x^3+b*x^2+c*x+d,[a,b,c,d]), numer;
```

(%o14) $$[[a = -0.002705800223881305, b = 3.018798873646606\ ,\ c = -29.94151342602112, d = 74.78603431944423]]$$

Vemos claramente que el término de tercer grado es supérfluo, por lo que reajustamos al de segundo grado,

```
(%i15) param: lsquares_estimates (data,[x,y],y=b*x^2+c*x+d,[b,c,d]),numer;
```

(%o15) $[[b = 2.979110687882263, c = -29.78353057922009, d = 74.64523259993118]]$

Ahora estamos en disposición de estudiar los residuos para valorar el ajuste. En primer lugar calculamos el error cuadrático medio del residuo, definido como

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{rhs}(e_i) - \mathrm{lhs}(e_i)\right)^2,$$

siendo n el tamaño de la muestra y $\mathrm{rhs}(e_i)$ y $\mathrm{lhs}(e_i)$ los miembros derecho e izquierdo, respectivamente, de la expresión a ajustar. Para el caso que nos ocupa, el valor calculado es

```
(%i16) lsquares_residual_mse(data, [x,y], y=b*x^2+c*x+d, first(param));
```

(%o16) 1.144872557335554

También podemos calcular el vector de los residuos, definidos como $\mathrm{lhs}(e_i) - \mathrm{rhs}(e_i)$, $\forall i$, para a continuación representarlos gráficamente junto con los datos y la curva ajustada, tal como se aprecia en la Figura 10.

```
(%i17) res: lsquares_residuals(data, [x,y], y=b*x^2+c*x+d, first(param))$
(%i18) load(draw)$ /* necesitaremos las rutinas gráficas */
(%i19) scene1: gr2d(points(data),
                    explicit(ev(b*x^2+c*x+d,first(param)),x,0,10))$
(%i20) scene2: gr2d(points_joined = true,
                    points(res))$
(%i21) draw(terminal = eps, scene1, scene2)$
```

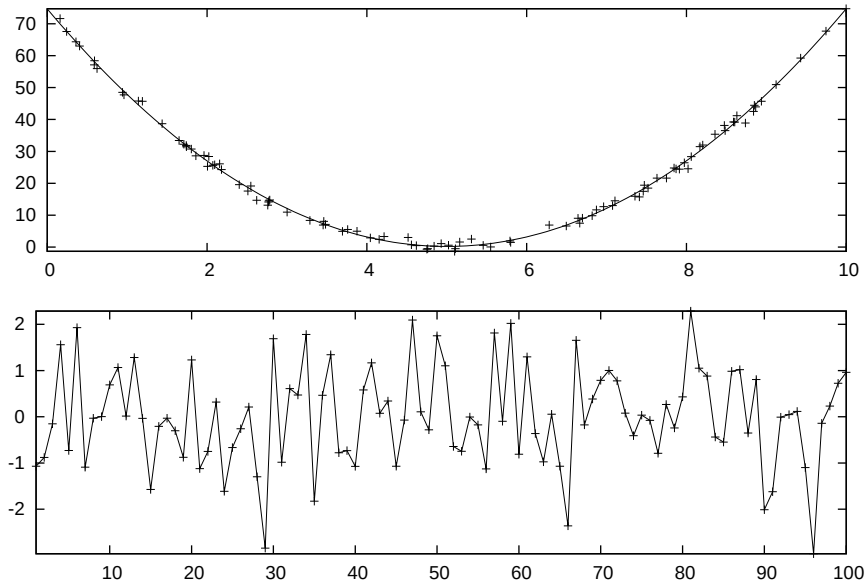


Figura 10: Resultados gráficos del ajuste por mínimos cuadrados.

Nosotros ya sabemos que el ruido del proceso simulado es gaussiano de esperanza nula. En una situación real, este dato lo desconoceremos y nos interesará contrastar la hipótesis de normalidad y la de la nulidad de la media de los residuos. El paquete `stats` tiene algunos procedimientos inferenciales, entre los cuales están las funciones `test_normality` y `test_mean`; primero cargamos el paquete y contrastamos la hipótesis de normalidad,

```
(%i22) load(stats)$
(%i23) test_normality(res);
```

(%o31)

SHAPIRO - WILK TEST
$statistic = 0.986785359052448$
$p\_value = 0.423354600782769$

El p-valor es lo suficientemente alto como para no rechazar la hipótesis nula de normalidad; ahora contrastamos la nulidad de la esperanza del ruido,

```
(%i24) test_mean(res);
```

(%o32)
$$\begin{array}{c} \text{MEAN TEST} \\ mean\_estimate = -1.002451472320587 \times 10^{-7} \\ conf\_level = 0.95 \\ conf\_interval = [-.2133782738324136, .2133780733421191] \\ method = \text{Exact t-test. Unknown variance.} \\ hypotheses = \text{H0: mean} = 0 \text{ , H1: mean} \neq 0 \\ statistic = 9.321855844715968 \times 10^{-7} \\ distribution = [student\_t, 99] \\ p\_value = .9999992583830712 \end{array}$$

Sin duda admitiríamos la hipótesis nula como válida.

Consúltese el manual de referencia de Maxima para ver qué otros tests están programados en el paquete `stats`, así como las opciones que se pueden utilizar en cada caso.

6.4. 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 de 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)
$$\left(\frac{13}{2}-\frac{3\,x}{2}\right)\ charfun_2\left(x,-\infty,3\right)+2\,charfun_2\left(x,7,\infty\right)+\left(37-5\,x\right)\ charfun_2\left(x,6,7\right)+\left(\frac{5\,x}{3}-3\right)\ charfun_2\left(x,3,6\right)$$

```
(%i4) f(x):=''%$
```

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]);
```

(%o5) $$\left[2, \frac{62}{21}, \frac{5\,\pi}{3} - 3\right]$$

```
(%i6) float(%);
```

(%o6) $$[2.0, 2.952380952380953, 2.235987755982989]$$

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]);
```

(%o7) $$\begin{pmatrix} 7 & 2 \\ 8 & 2 \\ 1 & 5 \\ 3 & 2 \\ 6 & 7 \end{pmatrix}$$

```
(%i8) lagrange(datos2,varname='w);
```

(%o8) $$\frac{73\,w^4}{420} - \frac{701\,w^3}{210} + \frac{8957\,w^2}{420} - \frac{5288\,w}{105} + \frac{186}{5}$$

```
(%i9) g(w):=''%$

(%i10) map(g,[7.3,25/7,%pi]), numer;
```

(%o10) $$[1.043464999999799, 5.567941928958199, 2.89319655125692]$$

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

```
(%i11) load(draw)$
```

```
(%i12) draw2d(
         key       = "Interpolador lineal",
         explicit(f(x),x,0,10),
         line_type = dots,
         key       = "Interpolador de Lagrange",
         explicit(g(x),x,0,10),
         key       = "Datos empiricos",
         points(datos),
         terminal  = eps)$
```

cuyo resultado se ve en el apartado *a)* de la Figura 11.

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`,

```
(%i13) cspline(datos);
```

$$
\begin{aligned}
(\%o13)\qquad & \left(\frac{1159\,x^3}{3288}-\frac{1159\,x^2}{1096}-\frac{6091\,x}{3288}+\frac{8283}{1096}\right)\ charfun_2\left(x,-\infty,3\right)+\\
& \left(-\frac{2587\,x^3}{1644}+\frac{5174\,x^2}{137}-\frac{494117\,x}{1644}+\frac{108928}{137}\right)\ charfun_2\left(x,7,\infty\right)+\\
& \left(\frac{4715\,x^3}{1644}-\frac{15209\,x^2}{274}+\frac{579277\,x}{1644}-\frac{199575}{274}\right)\ charfun_2\left(x,6,7\right)+\\
& \left(-\frac{3287\,x^3}{4932}+\frac{2223\,x^2}{274}-\frac{48275\,x}{1644}+\frac{9609}{274}\right)\ charfun_2\left(x,3,6\right)
\end{aligned}
$$

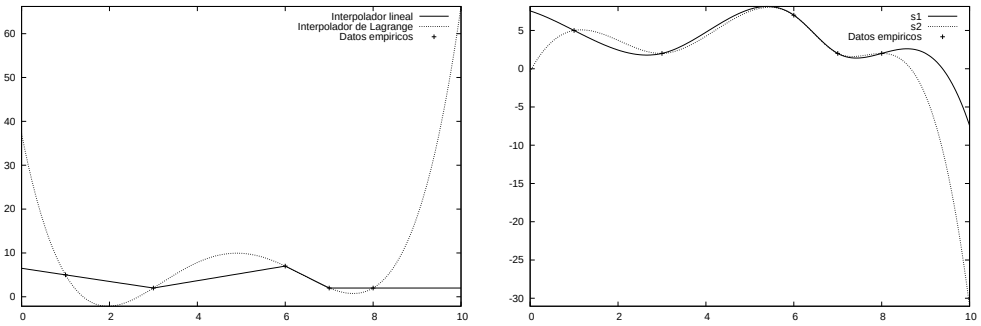
```
(%i14) s1(x):=''%$

(%i15) map(s1,[7.3,25/7,%pi]), numer;
```

$$(\%o15)\qquad [1.438224452554664, 3.320503453379974, 2.227405312429507]$$

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,

```
(%i16) cspline(datos,'varname='z,d1=1,dn=0);
```

$$
\begin{aligned}
(\%o16)\qquad & \left(\frac{2339\,z^3}{2256}-\frac{14515\,z^2}{2256}+\frac{24269\,z}{2256}-\frac{271}{752}\right)\ charfun_2\left(z,-\infty,3\right)+\\
& \left(-\frac{1553\,z^3}{564}+\frac{35719\,z^2}{564}-\frac{68332\,z}{141}+\frac{174218}{141}\right)\ charfun_2\left(z,7,\infty\right)+\\
& \left(\frac{613\,z^3}{188}-\frac{35513\,z^2}{564}+\frac{56324\,z}{141}-\frac{38882}{47}\right)\ charfun_2\left(z,6,7\right)+\\
& \left(-\frac{4045\,z^3}{5076}+\frac{1893\,z^2}{188}-\frac{5464\,z}{141}+\frac{2310}{47}\right)\ charfun_2\left(z,3,6\right)
\end{aligned}
$$



a) *b)*

Figura 11: Interpolación: *a)* lineal y de Lagrange; *b) Splines* cúbicos.

```
(%i17) s2(z):=''%$

(%i18) map(s2,[7.3,25/7,%pi]), numer;
```

(%o18) $[1.595228723404261, 2.88141531519733, 2.076658794432369]$

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 11.

```
(%i19) draw2d(
         key       = "s1",
         explicit(s1(x),x,0,10),
         line_type = dots,
         key       = "s2",
         explicit(s2(x),x,0,10),
         key       = "Datos empiricos",
         points(datos),
         terminal  = eps)$
```

7. Gráficos

Maxima no está habilitado para realizar él mismo gráficos, por lo que necesitará de un programa externo que realice esta tarea. Nosotros nos limitaremos a 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. La otra herramienta gráfica es Openmath, un programa Tcl-tk que se distribuye conjuntamente con Maxima.

7.1. El módulo "plot"

Las funciones `plot2d` y `plot3d` son las que se utilizan por defecto. Veamos un ejemplo de cada una de ellas.

```
(%i1) xy:[[10,.6], [20,.9], [30,1.1], [40,1.3], [50,1.4]]$
(%i2) plot2d([[discrete,xy], 2*%pi*sqrt(u/980)], [u,0,50],
              [style, [points,5,2,6], [lines,1,1]],
              [legend,"Datos experimentais","Predicion da teoria"],
              [xlabel,"Lonxitude do pendulo (cm)"], [ylabel,"periodo (s)"],
              [gnuplot_preamble,
                "set terminal postscript eps;set out 'plot2d.eps'"])$
```

Se ha definido en el ejemplo una lista de puntos empíricos a representar, junto con su función teórica, pidiendo su representación gráfica en el dominio $[0, 50]$. El resto de código hace referencia a diversas opciones: `style`, `legend`, `xlabel`, `ylabel` y `gnuplot_preamble`, esta última permite escribir código de Gnuplot para que éste lo ejecute; aquí se le pide que devuelva el gráfico en formato Postscript. El ejemplo siguiente genera una superficie explícita. Las dos salidas gráficas se representan en la Figura 12

```
(%i3) plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2],
        [gnuplot_preamble,
           "set terminal postscript eps;set out 'plot3d.eps'"])$
```

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

```
(%i4) plot_options;
(%o4) [[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,
```

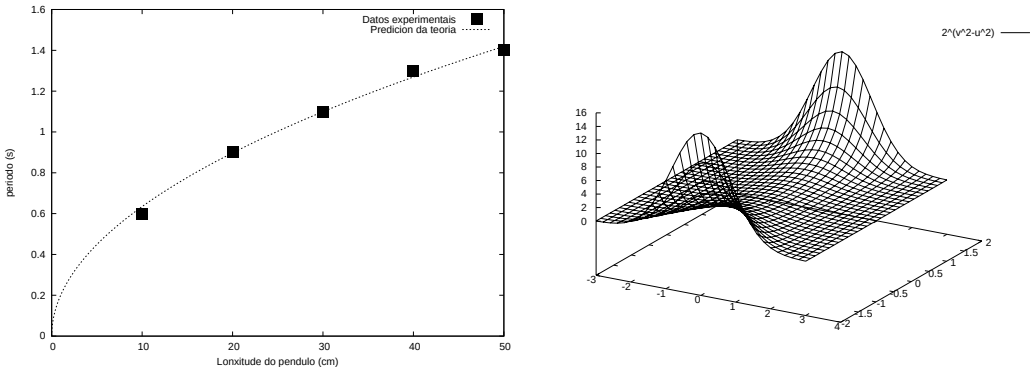


Figura 12: Gráficos generados por las funciones plot2d y plot3d.

```
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, así como de las funciones `plot2d` y `plot3d`, consúltese la documentación.

Por defecto, Maxima invocará al programa Gnuplot para la generación de gráficos, pero quizás 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.

```
(%i5) set_plot_option([plot_format, openmath])$
(%i6) plot2d([gamma(x),1/gamma(x)],[x,-4.5,5],[y,-10,10])$
```

El resto de esta sección lo dedicaremos a hacer una somera descripción del paquete `draw`, un proyecto consistente en el desarrollo de un interfaz que permita aprovechar al máximo las habilidades gráficas de Gnuplot.

7.2. El módulo "draw"

Aquí, las funciones a utilizar son `draw2d`, `draw3d` y `draw`, para escenas en 2d, 3d y para gráficos múltiples y animaciones, respectivamente. Puesto que se trata de un módulo adicional, será necesario cargarlo en memoria antes de hacer uso de él. Empecemos por un ejemplo comentado.

```
(%i1) load(draw)$
(%i2) draw2d(
        key           = "Cubic poly",
        explicit(%pi*x^3+sqrt(2)*x^2+10,x,0,2),
```

```
        color         = blue,
        key           = "Parametric curve",
        line_width    = 3,
        nticks        = 50,
        parametric(2*cos(rrr)+3, rrr, rrr, 0, 6*%pi),
        line_type     = dots,
        points_joined = true,
        point_type    = diamant,
        point_size    = 3,
        color         = red,
        line_width    = 1,
        key           = "Empiric data",
        points(makelist(random(40.0),k,1,5)),
        title         = "DRAWING CURVES",
        terminal      = eps )$
```

Los argumentos de `draw2d` se dividen en tres grupos: *objetos gráficos* (en el ejemplo, `explicit`, `parametric` y `points`), *opciones locales* (que afectan directamente a la representación de los objetos gráficos, como `key`, `color`, `line_width`, `nticks`, `line_type`, `points_joined` y `line_width`) y *opciones globales* (que hacen referencia a aspectos generales del gráfico, como `title` y `terminal`). Todos estos argumentos se interpretan secuencialmente, de forma que al asignar un cierto valor a una opción local, ésta afectará a todos los objetos gráficos que le sigan. En este ejemplo, `line_width` comienza teniendo valor 1, que es el asignado por defecto, luego se le da el valor 3 para la representación de la curva paramétrica y finalmente se le devuelve el valor original antes de representar los segmentos que unen los puntos aleatoriamente generados. En cambio, las dos opciones globales, `title` y `terminal`, aunque se colocaron al final, podrían haberse ubicado en cualquier otro lugar.

Siguiendo con los gráficos en dos dimensiones, el siguiente ejemplo muestra una escena en la que intervienen los objetos `ellipse`, `image`, `label`, `vector` y, ya lo conocemos, `explicit`. Antes de ejecutar esta instrucción es necesario leer el fichero gráfico `gatos.xpm`[9]

```
(%i3) cats: read_xpm("gatos.xpm")$
(%i4) draw2d(
        terminal    = eps,
        yrange      = [-4,10],
        ellipse(5,3,8,6,0,360),
        image(cats,0,0,10,7),
        line_width  = 2,
        head_length = 0.3,
        color       = blue,
        label(["This is Francisco",-1,-0.5]),
        vector([-1,0],[2,4]),
```

[9]El formato gráfico XPM es el único que puede leer Maxima.

```
        color       = green,
        vector([11,7],[-2,-1]),
        label(["This is Manolita",11,8]),
        explicit(sin(x)-2,x,-4,15) )$
```

Junto con los objetos gráficos introducidos en los ejemplos, cuyos resultados se pueden ver en los apartados *a)* y *b)* de la Figura 13, también existen `polygon`, `rectangle`, `polar`, `implicit` y `geomap`, este último para mapas cartográficos.

Mostramos a continuación algunas escenas tridimensionales. En primer lugar, el valor absoluto de la función Γ de Euler, junto con sus líneas de contorno.

```
(%i5) gamma2(x,y):=
        block([re,im,g:gamma(x+%i*y)],
          re:realpart(g),
          im:imagpart(g),
          sqrt(re^2+im^2))$
(%i6) draw3d(
        zrange         = [0,6],
        xu_grid        = 50,
        yv_grid        = 50,
        surface_hide   = true,
        contour        = surface,
        contour_levels = [0,0.5,6], /* de 0 a 6 en saltos de 0.5 */
        color          = cyan,
        terminal       = eps,
        explicit(gamma2(x,y),x,-4,4,y,-2,2))$
```

Una superficie paramétrica, que junto con la anterior escena, se pueden ver ambas en la Figura 13, apartados *c)* y *d)*.

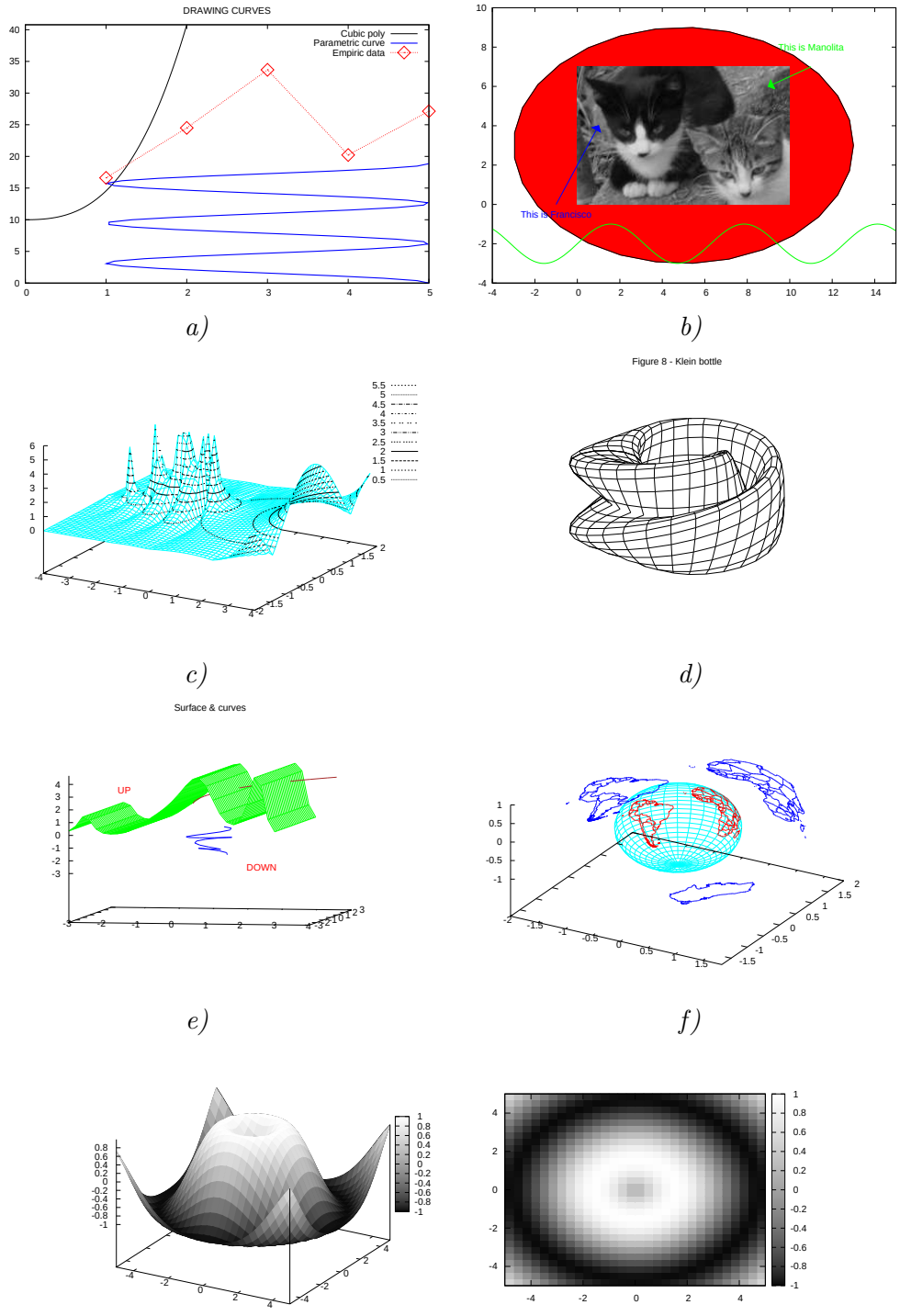
```
(%i7) draw3d(
        terminal       = eps,
        title          = "Figure 8 - Klein bottle",
        rot_horizontal = 360,
        xrange         = [-3.4,3.4],
        yrange         = [-3.4,3.4],
        zrange         = [-1.4,1.4],
        xtics          = none,
        ytics          = none,
        ztics          = none,
        axis_3d        = false,
        surface_hide   = true,
        parametric_surface((2+cos(u/2)*sin(v)-sin(u/2)*sin(2*v))*cos(u),
                           (2+cos(u/2)*sin(v)-sin(u/2)*sin(2*v))*sin(u),
                           sin(u/2)*sin(v) + cos(u/2)*sin(2*v),
                           u, -%pi, 360*%pi/180-%pi, v, 0, 2*%pi) )$
```

En el gráfico siguiente se combinan varios objetos: una superficie explícita, dos curvas paramétricas y dos etiquetas, cuyo aspecto es el mostrado en el apartado *e)* de la Figura 13.

```
(%i8) draw3d(
        color          = green,
        explicit(exp(sin(x)+cos(x^2)),x,-3,3,y,-3,3),
        color          = blue,
        parametric(cos(5*u)^2,sin(7*u),u-2,u,0,2),
        color          = brown,
        line_width     = 2,
        parametric(t^2,sin(t),2+t,t,0,2),
        surface_hide   = true,
        title          = "Surface & curves",
        color          = red,
        label(["UP",-2,0,3]),
        label(["DOWN",2,0,-3]),
        rot_horizontal = 10,
        rot_vertical   = 84,
        terminal       = eps )$
```

En el siguiente ejemplo hacemos una proyección esférica del hemisferio sur, que se ve en el apartado *f)* de la Figura 13. El paquete `worldmap` carga en memoria las coordenadas latitud–longitud de las líneas fronterizas y costeras de todos los países del mundo, así como algunas funciones necesarias para su manipulación y procesamiento,

```
(%i9) load(worldmap)$
(%i10) draw3d(
        surface_hide   = true,
        rot_horizontal = 60,
        rot_vertical   = 131,
        color          = cyan,
        parametric_surface(
               cos(phi)*cos(theta),
               cos(phi)*sin(theta),
               sin(phi),
               theta,-%pi,%pi,
               phi,-%pi/2,%pi/2),
        color          = red,
        geomap([South_America,Africa,Australia],
               [spherical_projection,0,0,0,1]),
        color          = blue,
        geomap([South_America,Africa,Australia],
               [cylindrical_projection,0,0,0,1,2]),
        terminal       = eps)$
```



a) *b)* *c)* *d)* *e)* *f)* *g)*

Figura 13: Gráficos generados con el paquete draw: *a)* y *b)* con draw2d; *c)*, *d)*, *e)* y *f)* con draw3d; *g)*, un gráfico múltiple.

Además de los objetos gráficos tridimensionales ya vistos, también se hayan definidos `points`, `vector` y, a partir de la versión 5.14 de Maxima, `implicit`.

También es posible generar múltiples gráficos en un mismo fichero o hacer animaciones en formato GIF o en la ventana gráfica. Para ver más ejemplos de gráficos generados con el paquete `draw`, se recomienda acceder a la dirección

http://www.telefonica.net/web2/biomates/maxima/gpdraw

o consultar el sistema de ayuda de Maxima.

Ya como colofón, un ejemplo de gráfico múltiple en el que se muestra también cómo dar sombreado a las superficies tridimensionales. El resultado en el apartado *g)* de la Figura 13.

```
(%i11) escena1:
         gr3d(surface_hide = true,
              enhanced3d   = true,
              palette      = gray,
              explicit(sin(sqrt(x^2+y^2)),x,-5,5,y,-5,5))$

(%i12) escena2:
         gr3d(surface_hide  = true,
              enhanced3d    = true,
              palette       = gray,
              user_preamble = "set pm3d map",
              explicit(sin(sqrt(x^2+y^2)),x,-5,5,y,-5,5))$

(%i13) draw(
        columns    = 2,
        terminal   = eps_color,
        eps_width  = 20,
        eps_height = 8,
        escena1, escena2)$
```

Se comienza definiendo las dos escenas a representar, la función explícita y el gráfico de densidades, en sendos objetos `gr3d`, ya que ambos son de naturaleza tridimensional. Estos objetos se pasan luego a la función `draw` como argumentos, junto con algunas opciones globales. La función `draw` es realmente la que ha generado también los gráficos anteriores, siendo `draw2d` y `draw3d` sinónimos de `draw(gr2d(...))` y `draw(gr3d(...))`, respectivamente.

8. Programando en Maxima

8.1. Programación a nivel de Maxima

Esencialmente, programar consiste en escribir secuencialmente un grupo de sentencias sintácticamente correctas que el intérprete pueda leer y luego ejecutar; la manera más sencilla de empaquetar varias sentencias es mediante paréntesis, siendo el resultado del programa la salida de la última sentencia:

```
(%i1) (a:3, b:6, a+b);
```

(%o1) 9

```
(%i2) a;
```

(%o2) 3

```
(%i3) b;
```

(%o3) 6

Como se ve en las salidas `%o2` y `%o3`, este método conlleva un peligro, que consiste en que podemos alterar desapercibidamente valores de variables que quizás se estén utilizando en otras partes de la sesión actual. La solución pasa por declarar variables localmente, cuya existencia no se extiende más allá de la duración del programa, mediante el uso de bloques; el siguiente ejemplo muestra el mismo cálculo anterior declarando `c` y `d` locales, además se ve cómo es posible asignarles valores a las variables en el momento de crearlas:

```
(%i4) block([c,d:6],
        c:3,
        c+d );
```

(%o4) 9

```
(%i5) c;
```

(%o5) c

```
(%i6) d;
```

(%o6) d

A la hora de hacer un programa, lo habitual es empaquetarlo como cuerpo de una función, de forma que sea sencillo utilizar el código escrito tantas veces como sea necesario sin más que escribir el nombre de la función con los argumentos necesarios; la estructura de la definición de una función necesita el uso del operador `:=`

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

donde `<expr>` es una única sentencia o un bloque con variables locales; véanse los siguientes ejemplos:

```
(%i7) loga(x,a):= float(log(x) / log(a)) $

(%i8) loga(7, 4);

(%o8)                          1.403677461028802

(%i9) fact(n):=block([prod:1],
          for k:1 thru n do prod:prod*k,
          prod )$

(%i10) fact(45);

(%o10)  119622220865480194561963161495657715064383733760000000000
```

En el primer caso (`loga`) se definen los logaritmos en base arbitraria (Maxima sólo tiene definidos los naturales); además, previendo que sólo los vamos a necesitar en su forma numérica, solicitamos que nos los evalúe siempre en formato decimal. En el segundo caso (`fact`) hacemos uso de un bucle para calcular factoriales. Esta última función podría haberse escrito recursivamente mediante un sencillo condicional,

```
(%i11) fact2(n):= if n=1 then 1
                         else n*fact2(n-1) $

(%i12) fact2(45);

(%o12)  119622220865480194561963161495657715064383733760000000000
```

O más fácil todavía,

```
(%i13) 45!;

(%o13)  119622220865480194561963161495657715064383733760000000000
```

Acabamos de ver dos estructuras de control de flujo comunes a todos los lenguajes de programación, las sentencias `if-then-else` y los bucles `for`. En cuanto a las primeras, puede ser de utilidad la siguiente tabla de operadores relacionales

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

He aquí un ejemplo de uso.

```
(%i14) makelist(is(log(x)<x-1), x, [0.9,1,1.1]);
```

(%o14) [true, false, true]

Los operadores lógicos que frecuentemente se utilizarán con las relaciones anteriores son `and`, `or` y `not`, con sus significados obvios

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`

```
(%i15) if sin(4)< 9/10  and 2^2 = 4 then 1 else -1 ;
```

(%o15) 1

```
(%i16) if sin(4)< -9/10  and 2^2 = 4 then 1 else -1 ;
```

(%o16) -1

Se ilustra en el ejemplo anterior (`%i14`) cómo se evalúa con la función `is` el valor de verdad de una expresión relacional ante la ausencia de un condicional o de un operador lógico; la siguiente secuencia aclara algo más las cosas:

```
(%i17) sin(4)< 9/10 ;
```

(%o17) $$\sin 4 < \frac{9}{10}$$

```
(%i18) is(sin(4)< 9/10);
```

(%o18) true

Pero ante la presencia de un operador lógico o una sentencia condicional, `is` ya no es necesario:

```
(%i19) sin(4)< 9/10  and 2^2 = 4;
```

(%o19) true

```
(%i20) if sin(4)< 9/10 then 1;
```

(%o20) 1

En cuanto a los bucles, `for` es muy versátil; tiene las siguientes variantes:

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

Cuando el incremento de la variable es la unidad, se puede obviar la parte de la sentencia relativa a `step`, dando lugar a los esquemas

```
for <var>:<val1> thru <val3> do <expr>
for <var>:<val1> while <cond> do <expr>
for <var>:<val1> unless <cond> do <expr>
```

Cuando no sea necesaria la presencia de una variable de recuento de iteraciones, también se podrá prescindir de los `for`, como en

```
while <cond> do <expr>
unless <cond> do <expr>
```

Algunos ejemplos que se explican por sí solos:

```
(%i21) for z:-5 while z+2<0 do print(z) ;
- 5
- 4
- 3
```

(%o21) done

```
(%i22) for z:-5 unless z+2>0 do print(z) ;
- 5
- 4
- 3
- 2
```

(%o22) done

```
(%i23) for cont:1 thru 3 step 0.5 do (var: cont^3, print(var)) ;
1
3.375
8.0
15.625
27.0
```

(%o23) done

```
(%i24) [z, cont, var];
```

(%o24) $[z, cont, 27.0]$

```
(%i25) while random(20) < 15 do print("qqq") ;
qqq
qqq
qqq
```

(%o25) done

Véase en el resultado %o24 cómo las variables z y cont no quedan con valor asignado, mientras que var sí; la razón es que tanto z como cont son locales en sus respectivos bucles, expirando cuando éste termina; esto significa, por ejemplo, que no sería necesario declararlas locales dentro de un bloque.

Otra variante de for, sin duda inspirada en la propia sintaxis de Lisp, es cuando el contador recorre los valores de una lista; su forma es

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

y un ejemplo:

```
(%i26) for z in [sin(x), exp(x), x^(3/4)] do print(diff(z,x)) $

cos(x)

  x
%e

  3
------
   1/4
4 x
```

Cuando una función debe admitir un número indeterminado de argumentos, se colocarán primero los que tengan carácter obligatorio, encerrando el último entre corchetes para indicar que a partir de ahí el número de argumentos puede ser arbitrario. La siguiente función suma y resta alternativamente las potencias n-ésimas de sus argumentos, siendo el exponente su primer argumento,

```
(%i27) sumdif(n,[x]):= x^n . makelist((-1)^(k+1), k, 1, length(x)) $

(%i28) sumdif(7,a,b,c,d,e,f,g);
```

$$(\%o28) \qquad g^7 - f^7 + e^7 - d^7 + c^7 - b^7 + a^7$$

```
(%i29) sumdif(%pi,u+v);
```

$$(\%o29) \qquad (v+u)^{\pi}$$

En la entrada `%i27`, dentro del cuerpo de la función, la variable `x` es la lista de los argumentos que podríamos llamar *restantes*; como tal lista, la función `length` devuelve el número de elementos que contiene. Recuérdese también que una lista elevada a un exponente devuelve otra lista con los elementos de la anterior elevados a ese mismo exponente. Por último, el operador `.` calcula el producto escalar.

En otras ocasiones, puede ser de interés pasar como argumento una función, en lugar de un valor numérico o una expresión. Esto no tiene problema para Maxima, pero debe hacerse con cierto cuidado. A continuación se define una función de Maxima que toma como argumento una función y la aplica a los cinco primeros números enteros positivos,

```
(%i30) fun_a(G):= map(G, [1,2,3,4,5]) $
```

Pretendemos ahora aplicar a estos cinco elementos la función seno, lo cual no da ningún problema,

```
(%i31) fun_a(sin);
```

$$(\%o31) \qquad [\sin 1, \sin 2, \sin 3, \sin 4, \sin 5]$$

Pero el problema lo vamos a tener cuando queramos aplicar una función algo más compleja,

```
(%i32) fun_a(sin(x)+cos(x));
Improper name or value in functional position:
sin(x) + cos(x)
#0: fun_a(g=sin(x)+cos(x))
 -- an error.  To debug this try debugmode(true);
```

lo cual se debe al hecho de que la función `map` no reconoce el argumento como función. En tal caso lo apropiado es definir la función objetivo separadamente, lo cual podemos hacer como función ordinaria o como función lambda:

```
(%i33) sc(z):= sin(z) + cos(z) $

(%i34) fun_a(sc);
```

(%o34) $[\sin 1 + \cos 1, \sin 2 + \cos 2, \sin 3 + \cos 3, \sin 4 + \cos 4, \sin 5 + \cos 5]$

```
(%i35) fun_a( lambda([z], sin(z)+cos(z)) );
```

(%o35) $[\sin 1 + \cos 1, \sin 2 + \cos 2, \sin 3 + \cos 3, \sin 4 + \cos 4, \sin 5 + \cos 5]$

Cuando se hace un programa, lo habitual es escribir el código con un editor de texto y almacenarlo en un archivo con extensión `mac`. Una vez dentro de Maxima, la instrucción

```
load("ruta/fichero.mac")$
```

leerá las funciones escritas en él y las cargará en la memoria, listas para ser utilizadas.

Si alguna de las funciones definidas en el fichero `fichero.mac` contiene algún error, devolviéndonos un resultado incorrecto, Maxima dispone del modo de ejecución de depurado (`debugmode`), que ejecuta el código paso a paso, pudiendo el programador chequear interactivamente el estado de las variables o asignarles valores arbitrariamente. Supongamos el siguiente contenido de cierto fichero `prueba.mac`

```
foo(y) :=
  block ([u:y^2],
    u: u+3,
    u: u^2,
    u);
```

Ahora, la siguiente sesión nos permite analizar los valores de las variables en cierto punto de la ejecución de la función:

```
(%i26) load("prueba.mac"); /* cargamos fichero */
```

(%o26) prueba.mac

```
(%i27) :break foo 3        /* declaro punto de ruptura al final de línea 3*/
Turning on debugging debugmode(true)
Bkpt 0 for foo (in prueba.mac line 4)
```

```
(%i27) foo(2);              /* llamada a función */
Bkpt 0:(prueba.mac 4)
prueba.mac:4::              /* se detiene al comienzo de la línea 4 */
(dbm:1) u;                  /* ¿valor de u? */
7
(dbm:1) y;                  /* ¿valor de y? */
2
(dbm:1) u: 1000;            /* cambio valor de u */
1000
(dbm:1) :continue           /* continúa ejecución */
```

(%o27) 1000000

8.2. Programación a nivel de Lisp

A veces se presentan circunstancias en las que es mejor hacer programas para Maxima directamente en Lisp; tal es el caso de las funciones que no requieran de mucho procesamiento simbólico, como funciones de cálculo numérico o la creación de nuevas estructuras de datos.

Sin dar por hecho que el lector conoce el lenguaje de programación Lisp, nos limitaremos a dar unos breves esbozos. Empecemos por ver cómo almacena Maxima internamente algunas expresiones:

```
(%i1) 3/4;
```

(%o1) $\frac{3}{4}$

```
(%i2) :lisp $%o1
((RAT SIMP) 3 4)

(%i2) :lisp ($num $%o1)
3
```

La expresión $\frac{3}{4}$ se almacena internamente como la lista `((RAT SIMP) 3 4)`. Una vez se ha entrado en modo Lisp mediante `:lisp`, le hemos pedido a Maxima que nos devuelva la expresión `%o1`; la razón por la que se antepone el símbolo de dólar es que desde Lisp, los objetos de Maxima, tanto variables como nombres de funciones, llevan este prefijo. En la segunda instrucción que introducimos a nivel Lisp, le indicamos que aplique a la fracción la función de Maxima `num`, que devuelve el numerador de una fracción.

Del mismo modo que desde el nivel de Lisp ejecutamos una instrucción de Maxima, al nivel de Maxima también se pueden ejecutar funciones de Lisp. Como ejemplo, veamos el comportamiento de las dos funciones de redondeo, la definida para Maxima y la propia del lenguaje Lisp:

```
(%i3) round(%pi);
```

(%o3) 3

```
(%i4) ?round(%pi);
Maxima encountered a Lisp error:


ROUND: $%PI is not a real number

Automatically continuing.
To reenable the Lisp debugger set *debugger-hook* to nil.
```

En el primer caso llamamos a la función de redondeo de Maxima y nos devuelve un resultado que reconocemos como correcto. En el segundo caso, al utilizar el prefijo `?`, estamos haciendo una llamada a la función `round` de Lisp y obtenemos un error; la razón es que esta función de Lisp sólo reconoce números como argumentos. En otras palabras, a nivel de Maxima se ha redefinido `round` para que reconozca expresiones simbólicas.

A efectos de saciar nuestra curiosidad, veamos cómo se almacenan internamente otras expresiones:

```
(%i5) x+y*2;
```

(%o5) $2\,y + x$

```
(%i6) [1,2];
```

(%o6) $[1, 2]$

```
(%i7) :lisp $%o5
((MPLUS SIMP) $X ((MTIMES SIMP) 2 $Y))

(%i7) :lisp $%o6
((MLIST SIMP) 1 2)
```

Otra forma de mostrar la representación interna de expresiones de Maxima es invocando la función `print` de Lisp. El siguiente ejemplo nos muestra que Maxima almacena internamente una expresión negativa como un producto por -1.

```
(%i8) ?print(-a)$

((MTIMES SIMP) -1 $A)
```

Para terminar, un ejemplo de cómo definir a nivel de Lisp una función que cambia de signo una expresión cualquiera que le pasemos como argumento:

```
(%i8) :lisp (defun $opuesto (x) (list '(mtimes) -1  x))
$OPUESTO

(%i9) opuesto(a+b);
```

$$(\%o9) \qquad -b-a$$

Se ha hecho un gran esfuerzo para que Maxima sea lo más universal posible en cuanto a entornos *Common Lisp*, de tal manera que en la actualidad Maxima funciona perfectamente con los entornos libres `clisp`, `gcl`, `cmucl` o `sbcl`.

También es posible llamar desde el nivel de Lisp funciones que previamente hayan sido definidas en Maxima, para lo cual será necesario utilizar la función Lisp `mfuncall`.

```
(%i10) cubo(x):= x^3$
(%i11) :lisp (mfuncall '$cubo 3)
27
```

Con `:lisp` sólo podemos ejecutar una única sentencia Lisp; pero si lo que queremos es abrir una sesión Lisp completa, haremos uso de la función `to_lisp`, terminada la cual volveremos a Maxima ejecutando `(to-maxima)`.

```
(%i12) to_lisp()$
Type (to-maxima) to restart, ($quit) to quit Maxima.

MAXIMA> (+ #c(3 4) #c(5 6/8)) ; suma complejos
#C(8 19/4)
MAXIMA> (if (< 2 0)
            "2 es negativo"
            "2 no es negativo") ; ejemplo de condicional
"2 no es negativo"
MAXIMA> (to-maxima)
Returning to Maxima
(%i13)
```

Índice alfabético