Software matemático básico: Maxima

Curso intensivo i-MATH de Software libre orientado a Ciencias e Ingeniería.

Vigo - Cádiz. Julio de 2008

Mario Rodríguez Riotorto
mario@edu.xunta.es

Índice general

Capítulo 1

Generalidades

Maxima es un programa de Matemáticas escrito en Lisp.

Su nombre original fue Macsyma (*MAC's SYmbolic MAnipulation System*). Se desarrolló en los laboratorios del MIT entre los años 1969 y 1982 con fondos aportados por varias agencias gubernamentales norteamericanas (*National Aeronautics and Space Administration*, *Office of Naval Research*, *U.S. Department of Energy* (DOE) 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 realizan los matemáticos, a fin de entender la capacidad de los ordenadores para actuar de forma inteligente.

El año 1982 es clave. El MIT da por finalizado el proyecto; una copia pasará al DOE (conocida como DOE-Macsyma) y otra a la empresa Symbolics Inc. para su explotación comercial, haciendo el código propietario. La copia del DOE fue mantenida por William Schelter de la Universidad de Texas y traducida a Common Lisp; parte de la comunidad científica podría utilizarla, pero no tendría derechos de redistribución. Es la época en la que aparecen en el mercado Maple y Mathematica. La versión comercial apenas se desarrolla, es superada por estos dos programas y deja de desarrollarse en 1999.

En 1998 Schelter obtiene permiso del Departamento de Energía para distribuir Maxima (así llamado ahora para diferenciarlo de la versión comercial) bajo la licencia GPL (*General Public License*) de la *Free Software Foundation*. En el año 2000 Maxima pasa a ser un proyecto hospedado en Sourceforge y en el 2001 fallece Schelter. Actualmente el proyecto se mantiene con el trabajo voluntario de un equipo internacional de programadores.

Toda la información relevante del proyecto está disponible en la URL

http://maxima.sourceforge.net

desde donde se puede descargar la documentación y los ficheros de instalación. La vía más directa y rápida de tomar contacto con el equipo de desarrollo y el resto de usuarios es a través de la lista de correo

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

Además, existe una lista de correos para usuarios de habla hispana en

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

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, accesible a todos los presupuestos, tanto institucionales como individuales.

Finalmente, tampoco es ajeno este programa al mundo empresarial e industrial (no en vano algunos de los desarrolladores actuales de Maxima son ingenieros), ya que junto a la reducción de los costes económicos en licencias hay que añadir el dinamismo de su desarrollo, pues no son pocas las ocasiones en las que una sugerencia o propuesta de mejora por parte de los usuarios del programa ha tenido una rápida respuesta por parte del equipo de desarrollo.

Capítulo 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 con las extensiones `rpm` o `deb`, según el caso.

Las siguientes indicaciones sobre su compilación 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` y `gcl` son todas ellas alternativas libres 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*, los pasos a seguir son los siguientes[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 se deberá escribir

   ```
   maxima
   ```

   si se quiere trabajar con la interfaz gráfica (ver sección siguiente), teclear

   ```
   xmaxima
   ```

[1] Se supone que se trabajará con clisp, siendo la codificación del sistema unicode.

2.2. Tomando contacto con el programa

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 iniciada la ejecución del programa, se nos presenta una cabecera:

```
Maxima 5.15.0cvs http://maxima.sourceforge.net
Using Lisp CLISP 2.41 (2006-10-13)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1)
```

donde el símbolo `(%i1)` nos indica que Maxima está esperando la primera entrada del usuario (`i` de *input*). Ahora se puede escribir una instrucción y terminarla con un punto y coma (`;`) para que el programa la ejecute; por ejemplo,

```
(%i1) diff(1-exp(-k*x),x);
```

(%o1) $$k\,e^{-k\,x}$$

```
(%i2)
```

donde le pedimos la derivada de la función $y = 1 - \exp(-kx)$ respecto de x, a lo que Maxima responde con ke^{-kx}, etiquetando este resultado con `(%o1)` (`o` de *output*). Nótese que tras la respuesta, la etiqueta `(%i2)` nos indica que Maxima espera una segunda instrucción.

El símbolo `%e` es la forma que tiene Maxima de representar la base de los logaritmos naturales, de igual forma que `%pi` representa al número π. Continuando con la sesión,

```
(%i2) %pi + %pi;
```

(%o2) $$2\,\pi$$

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

(%o3) $$6.283185307179586$$

Maxima hace simplificaciones algebraicas, como $\pi + \pi = 2\pi$, las cuales devuelve en forma simbólica. El símbolo `%`, cuando se utiliza aislado, representa la última respuesta dada por Maxima; así, en la entrada número tres se solicita que el último resultado se devuelva en formato decimal de coma flotante.

Maxima está escrito en Lisp, por lo que ya se puede intuir su potencia a la hora de trabajar con listas, las cuales deben finalizar y terminar con corchetes (`[a,c,...]`); en el siguiente diálogo, el texto escrito entre las marcas `/*` y `*/` son comentarios que no afectan a los cálculos, además se observa cómo se hace una asignación con el operador de dos puntos (`:`) a la variable `xx`,

```
(%i4) /* Se le asigna a la variable x una lista */
      xx: [cos(%pi), 4/16, [a, b], (-1)^3, integrate(u^2,u)];
```

(%o4) $$\left[-1, \frac{1}{4}, [a, b], -1, \frac{u^3}{3}\right]$$

```
(%i5) /* Se calcula el número de elementos del último resultado */
      length(%);
```

(%o5) 5

```
(%i6) /* Se transforma una lista en conjunto, */
      /* eliminando redundancias. Los         */
      /* corchetes se transforman en llaves   */
      setify(xx);
```

$$(\%o6) \qquad \left\{-1, \frac{1}{4}, [a,b], \frac{u^3}{3}\right\}$$

De la misma manera que `%` representa el último resultado, también se puede hacer referencia a una salida o entrada arbitraria, tal como muestra el siguiente ejemplo, donde también se observa cómo hacer sustituciones, lo que puede utilizarse para la evaluación numérica de expresiones:

```
(%i7) /* Sustituciones a hacer en x */
      xx, u=2, a=c;
```

$$(\%o7) \qquad \left[-1, \frac{1}{4}, [c,b], -1, \frac{8}{3}\right]$$

```
(%i8) /* Otra manera de hacer lo mismo */
      subst([u=2, a=c],xx);
```

$$(\%o8) \qquad \left[-1, \frac{1}{4}, [c,b], -1, \frac{8}{3}\right]$$

```
(%i9) %o4[5], u=[1,2,3];
```

$$(\%o9) \qquad \left[\frac{1}{3}, \frac{8}{3}, 9\right]$$

En esta última entrada, `%o4` hace referencia al cuarto resultado devuelto por Maxima, que es la lista guardada en la variable `xx`, de modo que `%o4[5]` es el quinto elemento de esta lista, por lo que esta expresión es equivalente a a `xx[5]`; después, igualando `u` a la lista `[1,2,3]` este quinto elemento de la lista es sustituído sucesivamente por estas tres cantidades. Maxima utiliza los dos puntos (`:`) para hacer asignaciones a variables, mientras que el símbolo de igualdad se reserva para la construcción de ecuaciones.

El final de cada instrucción debe terminar con un punto y coma (`;`) o con un símbolo de dólar (`$`); en el primer caso, Maxima muestra el resultado del cálculo y en el segundo lo oculta. En la entrada `%i11` se optará por el `$` para evitar una línea adicional que no interesa ver:

```
(%i10) a: 2+2;
```

(%o10) 4

```
(%i11) b: 6+6$
```

```
(%i12) 2*b;
```

(%o12) 24

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 de coma flotante, prefiriendo expresiones exactas:

```
(%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 el valor por defecto */
```

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

```
(%i18) A: matrix([sin(x),2,%pi],[0,y^2,5/8]);
```

(%o18) $\begin{pmatrix} \sin x & 2 & \pi \\ 0 & y^2 & \frac{5}{8} \end{pmatrix}$

```
(%i19) B: matrix([1,2],[0,2],[-5,integrate(x^2,x,0,1)]);
```

(%o19) $\begin{pmatrix} 1 & 2 \\ 0 & 2 \\ -5 & \frac{1}{3} \end{pmatrix}$

```
(%i20) A.B;  /* producto matricial */
```

(%o20) $\begin{pmatrix} \sin x - 5\,\pi & 2\,\sin x + \frac{\pi}{3} + 4 \\ -\frac{25}{8} & 2\,y^2 + \frac{5}{24} \end{pmatrix}$

En cuanto a gráficos, Maxima hace uso por defecto del programa externo `gnuplot`, pero también puede utilizar el programa `openmath`, escrito en `tcl-tk`, que se distribuye junto con Maxima. Las funciones a utilizar son `plot2d` y `plot3d`, para gráficos en dos y tres dimensiones, respectivamente. Sin embargo, últimamente se ha estado trabajando en el desarrollo del paquete `draw`, un interfaz para `gnuplot` que trata de aprovechar mejor las capacidades que se han incorporado a este programa en los últimos tiempos. Aunque posponemos para un capítulo posterior la descripción de este paquete, vaya aquí un botón de muestra en el que se describe una escena en tres dimensiones. El resultado se observa en la Figura 2.1

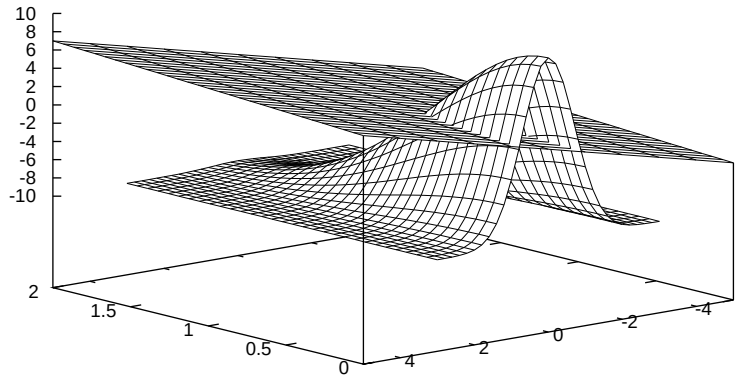


Figura 2.1: Un ejemplo de gráfico en tres dimensiones.

```
(%i21) load(draw)$

(%i22) draw3d(
         explicit(20*exp(-x^2-y^2)-10,x,0,2,y,-3,3),
         yv_grid        = 10,
         explicit(x+y,x,0,2,y,-5,5),
         rot_horizontal = 230,
         rot_vertical   = 70,
         surface_hide   = true,
         terminal       = eps)$
```

A fin de preparar publicaciones científicas, ya se ha visto cómo se pueden generar archivos gráficos en calidad Postscript; si se quiere redactar publicaciones 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:

```
(%i23) 'integrate(sqrt(a+x)/x^5,x,1,2) = integrate(sqrt(2+x)/x^5,x,1,2)$

(%i24) 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 símbolos de dólar a un documento LaTeX, el resultado es el siguiente[2]:

[2]Todos los cálculos de este documento se han realizado operando con Maxima desde la línea de comandos; tras reiteradas llamadas a la función tex se han obtenido los correspondientes códigos TeXdesde los que se han obtenido las expresiones matemáticas formateadas de este tutorial.

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

Un comentario sobre la entrada `%i23`. 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 la comilla simple que antecede al primer `integrate`; se trata de un operador (llamado precisamente así, de *comilla simple*) que Maxima utiliza para evitar la ejecución de una instrucción, de forma que en el resultado devuelto la primera integral no se evalúa, pero sí la segunda, dando lugar a la igualdad que se obtiene como resultado final.

Puede darse el caso de que llegue el momento en el que el usuario decida evaluar una expresión que previamente marcó con el operador comilla simple (tales expresiones reciben en Maxima el apelativo de *nominales*); el siguiente ejemplo muestra el uso del símbolo especial `nouns` para forzar el cálculo de estas expresiones,

```
(%i25) z: 'diff(tan(x),x);
```

(%o25) $$\frac{d}{d\,x}\,\tan x$$

```
(%i26) z+z;
```

(%o26) $$2\left(\frac{d}{d\,x}\,\tan x\right)$$

```
(%i27) z, nouns;
```

(%o27) $$\sec^2 x$$

Es posible que el usuario necesite trabajar sobre un mismo problema durante varias sesiones, por lo que le interesará guardar aquellas partes de la sesión actual que necesitará más adelante. Si se quiere almacenar la expresión guardada en `z`, junto con el resultado de su evaluación (salida `%o27`), podrá hacer uso de la función `save`:

```
(%i28) save("sesion", z, vz=%o27);
```

(%o28) sesion

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

Obsérvese que en primer lugar se escribe el nombre del fichero y a continuación los nombres de las variables a almacenar; como el resultado de la evaluación de `z` no se había asignado a ninguna variable, es necesario ponerle uno, en este caso `vz`. La última instrucción es la que termina la ejecución del programa. Una vez iniciada una nueva sesión, se cargará el fichero `sesion` con la función `load`:

```
(%i1) load("sesion");
```

(%o1) sesion

```
(%i2) z;
```

$$(\%o2) \qquad \frac{d}{d\,x}\,\tan x$$

```
(%i3) vz;
```

$$(\%o3) \qquad \sec^2 x$$

Se puede acceder a la ayuda relativa a cualquiera de las funciones que han aparecido hasta ahora, y otras que irán apareciendo, haciendo uso de la función `describe`, o equivalentemente, del operador `?`:

```
(%i4) ? 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.
```

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,

```
(%i5) ?? 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.
```

Inicialmente muestra un sencillo menú con las funciones que contienen en su nombre la cadena `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`.

Dependiendo de con qué interfaces se mantenga el diálogo con 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 la documentación 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`,

```
(%i6)
system("(mozilla /usr/local/share/maxima/5.15/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[3]. Así, cada vez que arranque Maxima, se abrirá el navegador con el manual de referencia del programa.

[3]En Linux

Capítulo 3

Matemáticas con Maxima

3.1. Números

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

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

(%o1)
$$\begin{aligned}&788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445621451664240254033\\&291864131227428294853277524242407573903240321257405579568660226031904170324062351700858796178922222789623703897374720000000000000000000000000000000000000000000000000000\end{aligned}$$

```
(%i2) factor(%);   /* factoriza el resultado anterior */
```

(%o2)
$$2^{197}\,3^{97}\,5^{49}\,7^{32}\,11^{19}\,13^{16}\,17^{11}\,19^{10}\,23^{8}\,29^{6}\,31^{6}\,37^{5}\,41^{4}\,43^{4}\,47^{4}\,53^{3}\,59^{3}\,61^{3}\,67^{2}\,71^{2}\,73^{2}\,79^{2}\,83^{2}\,89^{2}\,97^{2}\,101\,103\,107\,109\,113\,127\,131\,137\,139\,149\,151\,157\,163\,167\,173\,179\,181\,191\,193\,197\,199$$

Por defecto, no transforma las fracciones a su forma decimal, aunque sí las simplifica,

```
(%i3) zz: (3/12 - 3 * 8/5)^-5 ;
```

(%o3)
$$-\frac{3200000}{6240321451}$$

La expresión decimal de un número racional se puede obtener con una llamada a la función `float`, que devuelve el resultado como un número de coma flotante de doble precisión,

```
(%i4) float(%);
```

(%o4)
$$-5.127940964462986 \times 10^{-4}$$

Si se quiere más precisión, se puede recurrir a los números decimales grandes, para lo cual se indica en primer lugar la precisión deseada con la variable global `fpprec` y se hace la transformación con `bfloat` (*big float*),

```
(%i5) fpprec: 200$  bfloat(zz);
```

$$(\%o5)\quad \begin{array}{l} -5.12794096446298596293256880814488029168034600338090820571736601714365755675235967904307883385669518132648516346437807887855230937205128922131867273899509251418529208712713155391584330100241177463663 96_B \times 10^{-4} \end{array}$$

El subíndice B sólo es un recordatorio de que el resultado devuelto es un número decimal de precisión arbitraria.

Los números complejos se construyen haciendo uso de la constante imaginaria `%i`:

```
(%i6) %i^2;
```

$$(\%o6)\quad -1$$

```
(%i7) z1: 3+4*%i;
```

$$(\%o7)\quad 4\,i+3$$

```
(%i8) z2: exp(7*%pi/8*%i);
```

$$(\%o8)\quad e^{\frac{7\,i\,\pi}{8}}$$

```
(%i9) z1+z2;
```

$$(\%o9)\quad e^{\frac{7\,i\,\pi}{8}}+4\,i+3$$

```
(%i10) rectform(%);   /* forma cartesiana */
```

$$(\%o10)\quad i\left(\sin\left(\frac{7\,\pi}{8}\right)+4\right)+\cos\left(\frac{7\,\pi}{8}\right)+3$$

```
(%i11) polarform(%); /* forma polar */
```

$$(\%o11)\quad \sqrt{\left(\sin\left(\frac{7\,\pi}{8}\right)+4\right)^2+\left(\cos\left(\frac{7\,\pi}{8}\right)+3\right)^2}\;e^{i\,\arctan\left(\frac{\sin\left(\frac{7\,\pi}{8}\right)+4}{\cos\left(\frac{7\,\pi}{8}\right)+3}\right)}$$

```
(%i12) abs(%);  /* módulo */
```

$$(\%o12)\quad \sqrt{\sin^2\left(\frac{7\,\pi}{8}\right)+8\,\sin\left(\frac{7\,\pi}{8}\right)+\cos^2\left(\frac{7\,\pi}{8}\right)+6\,\cos\left(\frac{7\,\pi}{8}\right)+25}$$

Expandiendo el radio del resultado `(%o11)`, observamos que se trata precisamente del módulo:

```
(%i13) expand(part(%o11,1));
```

(%o13) $$\sqrt{\sin^2\left(\frac{7\,\pi}{8}\right)+8\,\sin\left(\frac{7\,\pi}{8}\right)+\cos^2\left(\frac{7\,\pi}{8}\right)+6\,\cos\left(\frac{7\,\pi}{8}\right)+25}$$

Por último, el módulo en formato decimal de coma flotante:

```
(%i14) float(%);
```

(%o14) 4.84955567695155

Prácticas

1. Calcúlese la expresión
$$\left[\left(\frac{3^7}{4+\frac{3}{5}}\right)^{-1}+\frac{7}{9}\right]^3$$
   a) en forma exacta,
   b) en forma decimal de doble precisión.
2. Calcúlese el número π con 5000 decimales.
3. Sea el número complejo $z=\frac{7}{3}-\frac{4}{5}i$. Haciendo uso de las funciones `conjugate` y `expand`, calcúlese su norma de acuerdo con la expresión $|z|=\sqrt{z\cdot\bar{z}}$. Compruébese el resultado con el devuelto por la función `abs`.

3.2. 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*). Desde el punto de vista del usuario medio, conviene saber que cualquier estructura de datos, por complicada que sea, siempre podrá reducirse a una lista más o menos elaborada.

En Maxima se definen las lista acotándolas con corchetes y separando sus elementos mediantes comas,

```
(%i1) r:[1,[a,3],sqrt(3)/2,"Don Quijote"];
```

(%o1) $$\left[1,\left[a,3\right],\frac{\sqrt{3}}{2},\text{Don Quijote}\right]$$

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,

```
(%i2) listp(r); /* es r una lista? */
```

(%o2) true

```
(%i3) first(r); /* primer elemento */
```

(%o3) 1

```
(%i4) second(r); /* segundo elemento */
```

(%o4) $[a, 3]$

```
(%i5) third(r); /* ...hasta tenth */
```

(%o5) $\frac{\sqrt{3}}{2}$

```
(%i6) last(r); /* el último de la lista */
```

(%o6) Don Quijote

```
(%i7) rest(r); /* todos menos el primero */
```

(%o7) $\left[[a, 3], \frac{\sqrt{3}}{2}, \text{Don Quijote}\right]$

```
(%i8) part(r,3); /* pido el que quiero */
```

(%o8) $\frac{\sqrt{3}}{2}$

```
(%i9) length(r); /* cuantos hay? */
```

(%o9) 4

```
(%i10) reverse(r); /* le damos la vuelta */
```

(%o10) $\left[\text{Don Quijote}, \frac{\sqrt{3}}{2}, [a, 3], 1\right]$

```
(%i11) member(a,r); /* es a un elemento?*/
```

(%o11) false

```
(%i12) member([a,3],r); /* lo es [a,3]? */
```

(%o12) true

```
(%i13) sort(q: [7,a,1,d,5,3,b]); /* asigno valor a q y ordeno */
```

(%o13) $[1, 3, 5, 7, a, b, d]$

```
(%i14) delete([a,3],r); /* borro elemento */
```

$$(\%o14) \qquad \left[1, \frac{\sqrt{3}}{2}, \text{Don Quijote}\right]$$

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

```
(%i15) r;
```

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

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

```
(%i16) cons(1+2,q);
```

$$(\%o16) \qquad [3, 7, a, 1, d, 5, 3, b]$$

```
(%i17) endcons(x,q);
```

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

```
(%i18) q;
```

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

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

```
(%i19) q: endcons(x,cons(1+2,q))$
(%i20) q;
```

$$(\%o20) \qquad [3, 7, a, 1, d, 5, 3, b, x]$$

Es posible unir dos listas,

```
(%i21) append(r,q);
```

$$(\%o21) \qquad \left[1, [a, 3], \frac{\sqrt{3}}{2}, \text{Don Quijote}, 3, 7, a, 1, d, 5, 3, b, x\right]$$

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

```
(%i22) s:makelist(2+k*2,k,0,10);
```

(%o22) $[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]$

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,

```
(%i23) apply("+",s);
```

(%o23) 132

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

(%o24) 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,

```
(%i25) map("!",s);
```

(%o25) $[2, 24, 720, 40320, 3628800, 479001600, 87178291200, 20922789888000,$
$6402373705728000, 2432902008176640000, 112400072777607680000]$

```
(%i26) map(sin,s);
```

(%o26) $[\sin 2, \sin 4, \sin 6, \sin 8, \sin 10, \sin 12, \sin 14, \sin 16, \sin 18, \sin 20, \sin 22]$

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,

```
(%i27) map(lambda([x], sin(x) + 1/2),[7,3,4,9]);
```

(%o27) $$\left[\sin 7 + \frac{1}{2}, \sin 3 + \frac{1}{2}, \sin 4 + \frac{1}{2}, \sin 9 + \frac{1}{2}\right]$$

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 y hacer uso de ellas como si fuesen vectores; para más información, acúdase a la Sección 3.7

Prácticas

1. Maxima extiende algunas operaciones numéricas a las listas. Guárdese en la variable `u` la lista formada por los números 7, 2, 6 y 9. Ejecútense a continuación e interprétense los resultados obtenidos por las siguientes sentencias: `u+10`, `u/9`, `9/u`, `u^7`, `7^u`, `sqrt(u)` y `log(u)`. Vemos que no siempre las cosas son como uno espera que sean. ¿Cómo haríamos para calcular los logaritmos de los elementos de una lista como `u`?

2. Guárdese en la variable `d` la lista de números $[5, 2, 9, 3, 1, 6, 8, 1, 9]$. Almacénse en la variable `p` el valor promedio de `d` haciendo uso de las funciones de Maxima que se han presentado en esta sección. Puesto que la varianza se definiría en este caso como $\sum_i (x_i - p)^2$, ¿qué expresión permitiría ahora calcular la desviación típica (esto es, la raíz cuadrada de la varianza)?

3. *a*) Fórmese una lista de ternas de la forma $[x, \sin(x), \exp(x)]$, donde x toma valores desde 0 hasta 1 en saltos de 0.1.
   b) Fórmese otra lista de ternas de la forma $[\cos(x), \log(x), x^2 - x]$, en la que x toma sucesivamente los valores 1.2, 1.5, 1.6, 1.7 y 1.9. (Convendrá leer lo que la documentación dice acerca de la función `makelist`.)
   c) Guárdese en la variable `t` una lista que sea la unión de las dos anteriores.
   d) Interpretando las ternas de `t` como puntos de $\mathbb{R}^3$, obténgase su proyección sobre el plano xy. Idem sobre el plano xz. (Véase `%i27`.)

3.3. Transformaciones algebraicas

Ya hemos visto cómo Maxima simplifica y reduce expresiones algebraicas automáticamente. Si en algún momento queremos inhibir las simplificaciones, podemos hacer uso de la variable global `simp`, cuyo valor por defecto es `true`,

```
(%i1) simp: false$

(%i2) 2 + 2 + a + a;
```

(%o2) $2 + 2 + a + a$

```
(%i3) simp: true$

(%i4) %o2;
```

(%o4) $2\,a + 4$

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, veamos algunas situaciones.

```
(%i5) expr:(x^2-x)/(x^2+x-6)-5/(x^2-4);
```

(%o5) $$\frac{x^2 - x}{x^2 + x - 6} - \frac{5}{x^2 - 4}$$

Podemos factorizar esta expresión,

```
(%i6) factor(expr);
```

$$(\%o6) \qquad \frac{(x-3)\,\left(x^2+4\,x+5\right)}{(x-2)\,(x+2)\,(x+3)}$$

La expansión devuelve el siguiente resultado,

```
(%i7) expand(expr);
```

$$(\%o7) \qquad \frac{x^2}{x^2+x-6}-\frac{x}{x^2+x-6}-\frac{5}{x^2-4}$$

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

```
(%i8) partfrac(expr,x);
```

$$(\%o9) \qquad -\frac{12}{5\,(x+3)}+\frac{5}{4\,(x+2)}-\frac{17}{20\,(x-2)}+1$$

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

```
(%i9) radcan(expr);
```

$$(\%o9) \qquad \frac{x^3+x^2-7\,x-15}{x^3+3\,x^2-4\,x-12}$$

Otra función que se puede utilizar para la simplificación de expresiones es `ratsimp`, cuyo uso se puede consultar en la documentación.

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

```
(%i10) log(x^r)-log(x*y) + a*log(x/y);
```

$$(\%o10) \qquad -\log\left(x\,y\right)+a\,\log\left(\frac{x}{y}\right)+r\,\log x$$

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,

```
(%i11) logexpand: super$

(%i12) log(x^r)-log(x*y) + a*log(x/y);
```

$$(\%o12) \qquad -\log y+a\,\left(\log x-\log y\right)+r\,\log x-\log x$$

```
(%i13) /* pedimos que nos simplifique la expresión anterior */
       ratsimp(%);
```

$$(\%o13) \qquad (-a-1)\,\log y + (r+a-1)\,\log x$$

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

```
(%i14) logexpand: false$

(%i15) log(x^r)-log(x*y) + a*log(x/y);
```

$$(\%o15) \qquad -\log(x\,y) + a\,\log\left(\frac{x}{y}\right) + \log x^r$$

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

```
(%i16) logexpand: true$
```

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

```
(%i17) logcontract(2*(a*log(x) + 2*a*log(y)));
```

$$(\%o17) \qquad a\,\log\left(x^2\,y^4\right)$$

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,

```
(%i18) declare(a, integer)$

(%i19) ''%i17;  /* fuerza evaluación de expresión nominal */
```

$$(\%o19) \qquad \log\left(x^{2\,a}\,y^{4\,a}\right)$$

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,

```
(%i20) expr: x+sin(3*x+y)/sin(x);
```

$$(\%o20) \qquad \frac{\sin(y+3\,x)}{\sin x} + x$$

```
(%i21) trigexpand(expr); /* aquí trigexpand vale false */
```

(%o21) $$\frac{\cos(3\,x)\,\sin y+\sin(3\,x)\,\cos y}{\sin x}+x$$

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

```
(%i22) trigexpand(expr), trigexpand=true;
```

(%o22) $$\frac{\left(\cos^3 x-3\,\cos x\,\sin^2 x\right)\,\sin y+\left(3\,\cos^2 x\,\sin x-\sin^3 x\right)\,\cos y}{\sin x}+x$$

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,

```
(%i23) ratsimp(%);
```

(%o23) $$-\frac{\left(3\,\cos x\,\sin^2 x-\cos^3 x\right)\,\sin y+\left(\sin^3 x-3\,\cos^2 x\,\sin x\right)\,\cos y-x\,\sin x}{\sin x}$$

Podemos inhibir el desarrollo de la suma,

```
(%i24) trigexpand(expr), trigexpandplus=false;
```

(%o24) $$\frac{\sin(y+3\,x)}{\sin x}+x$$

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,

```
(%i25) sin(x/2);
```

(%o25) $$\sin\left(\frac{x}{2}\right)$$

```
(%i26) sin(x/2), halfangles=true;
```

(%o26) $$\frac{\sqrt{1-\cos x}}{\sqrt{2}}$$

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,

```
(%i27) 5*sin(x)^2 + 4*cos(x)^2;
```

(%o27) $$5\,\sin^2 x+4\,\cos^2 x$$

```
(%i28) trigsimp(%);
```

(%o28) $$\sin^2 x + 4$$

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

```
(%i29) -sin(x)^2+3*cos(x)^2*tan(x);
```

(%o29) $$3\,\cos^2 x\,\tan x - \sin^2 x$$

```
(%i30) trigreduce(%);
```

(%o30) $$\frac{3\,\sin\left(2\,x\right)}{2} + \frac{\cos\left(2\,x\right)}{2} - \frac{1}{2}$$

Y si queremos transformar expresiones trigonométricas en complejas,

```
(%i31) exponentialize(%);
```

(%o31) $$\frac{e^{2\,i\,x} + e^{-2\,i\,x}}{4} - \frac{3\,i\,\left(e^{2\,i\,x} - e^{-2\,i\,x}\right)}{4} - \frac{1}{2}$$

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

Prácticas

1. El único logaritmo que Maxima conoce es el natural, al que llama `log`. Si quisiéramos trabajar en otras bases, podríamos definir una nueva función (se tratarán con más detalle en la sección 4):

   ```
   loga(a,x) := log(x)/log(a) ;
   ```

   Haciendo uso de esta nueva función, calcúlese la expresión $\log_{\sqrt{3}}(15)$ en forma decimal.

2. Haciendo uso de la función `trigsimp`, demuéstrense las siguientes identidades evaluando la diferencia de ambos miembros:

   a) $\frac{1+\cos(x)}{\sin(x)} = \frac{\sin(x)}{1-\cos(x)}$

   b) $\sec^2(x) - \cos^2(x) = \tan^2(x) + \sin^2(x)$

3. Repítase el ejercicio anterior con la igualdad

   $$1 - 2\cos(2x) = 4\sin^2(x) - 1.$$

   Téngase en cuenta que el coseno debe antes expandirse con la función `trigexpand` (véase entrada `%i24` más arriba).

3.4. Resolución de ecuaciones

La función `solve` será la encargada de resolver la mayor parte de nuestras ecuaciones, sin perjuicio de usar otras existentes en Maxima más especializadas para casos específicos.

Cuando la expresión que se pasa a `solve` no tiene igualdad, se interpreta que toda ella es igual a cero,

```
(%i1) solve(4*x^3-2*x^2+5*x -7);
```

(%o1)
$$\left[x = -\frac{3\sqrt{3}\,i+1}{4}, x = \frac{3\sqrt{3}\,i-1}{4}, x = 1\right]$$

La función `solve` es capaz de resolver algunas ecuaciones de tipo no algebraico; en este ejemplo se añade la especificación de la incógnita, la cual no es necesaria cuando la ecuación incorpora una única variable,

```
(%i2) solve (asin (cos (3*x))*(f(x) - 1), x);

'solve' is using arc-trig functions to get a solution.
Some solutions will be lost.
```

(%o2)
$$\left[x = \frac{\pi}{6}, f\left(x\right) = 1\right]$$

Pero la incógnita puede ser una expresión, no necesariamente un símbolo,

```
(%i3) solve (asin (cos (3*x))*(f(x) - 1), f(x));
```

(%o3)
$$\left[f\left(x\right) = 1\right]$$

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 de incógnitas x e y,

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

```
(%i4) solve([3*x^2-y^2=6,x=y+a],[x,y]);
```

(%o4)
$$\left[\left[x = -\frac{\sqrt{3}\,\sqrt{a^2+4}+a}{2}, y = -\frac{\sqrt{3}\,\sqrt{a^2+4}+3\,a}{2}\right], \left[x = \frac{\sqrt{3}\,\sqrt{a^2+4}-a}{2}, y = \frac{\sqrt{3}\,\sqrt{a^2+4}-3\,a}{2}\right]\right]$$

Sin embargo, la función `algsys` es específica para la resolución de sistemas de ecuaciones algebraicas. Admite como argumentos dos listas, en la primera se escribe la ecuación o ecuaciones del sistema, en las que, como en el caso de `solve`, si se omite la igualdad se interpretan como igualadas a cero; en la segunda lista se colocan los nombres de las incógnitas. Veamos algunos ejemplos:

Un sistema no lineal con coeficientes paramétricos y complejos

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

```
(%i5) algsys([3*u-a*v=t,(2+%i)/(u+t)=3*v+u,t/u=1],[u,v,t]);
```

$$(\%o5)\quad \left[\left[u=\frac{\sqrt{\frac{i\,a}{a+6}+\frac{2\,a}{a+6}}}{\sqrt{2}},v=\frac{2\,\sqrt{i+2}}{\sqrt{2}\,\sqrt{a}\,\sqrt{a+6}},t=\frac{\sqrt{i+2}\,\sqrt{a}}{\sqrt{2}\,\sqrt{a+6}}\right],\left[u=-\frac{\sqrt{\frac{i\,a}{a+6}+\frac{2\,a}{a+6}}}{\sqrt{2}},v=-\frac{2\,\sqrt{i+2}}{\sqrt{2}\,\sqrt{a}\,\sqrt{a+6}},t=-\frac{\sqrt{i+2}\,\sqrt{a}}{\sqrt{2}\,\sqrt{a+6}}\right]\right]$$

Cuando `algsys` no es capaz de resolver el sistema algebraicamente, recurre a métodos de aproximación numérica, como en el caso de la ecuación polinómica

$$x^5+x^4+x^3+x^2+x=0$$

```
(%i6) algsys([x^5+x^4+x^3+x^2+x],[x]);
```

$$(\%o6)\quad \begin{array}{l}[[x=0],[x=-.5877852522924731\,i-.8090169943749475],\\ [x=.5877852522924731\,i-.8090169943749475],\\ [x=.3090169943749475-.9510565162951535\,i],\\ [x=.9510565162951535\,i+.3090169943749475]]\end{array}$$

Sin embargo, los algoritmos de `solve` sí hubiesen podido con ella:

```
(%i7) solve([x^5+x^4+x^3+x^2+x],[x]);
```

$$(\%o7)\quad \begin{array}{l}\left[x=-2^{-\frac{3}{2}}\,5^{\frac{1}{4}}\,\sqrt{\sqrt{5}-1}\,i-\frac{\sqrt{5}}{4}-\frac{1}{4},x=2^{-\frac{3}{2}}\,5^{\frac{1}{4}}\,\sqrt{\sqrt{5}-1}\,i-\frac{\sqrt{5}}{4}-\frac{1}{4}\right.,\\ \left.x=-2^{-\frac{3}{2}}\,5^{\frac{1}{4}}\,\sqrt{\sqrt{5}+1}\,i+\frac{\sqrt{5}}{4}-\frac{1}{4},x=2^{-\frac{3}{2}}\,5^{\frac{1}{4}}\,\sqrt{\sqrt{5}+1}\,i+\frac{\sqrt{5}}{4}-\frac{1}{4},x=0\right]\end{array}$$

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

$$\left\{\begin{array}{rcrcrcl} 2x & - & 4y & + & 2z & = & -2 \\ \frac{1}{3}x & + & 2y & + & 9z & = & x+y \\ -4x & + & \sqrt{2}y & + & z & = & 3y \end{array}\right.$$

```
(%i8) 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]);
```

$$(\%o8)\quad \left[x=\frac{3021\,\sqrt{2}-12405}{48457},y=\frac{1537\,\sqrt{2}+16642}{48457},z=\frac{53\,\sqrt{2}-2768}{48457}\right]$$

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.

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.

```
(%i9) realroots(x^8+x^3+x+1, 5e-6);
```

(%o9) $$\left[x=-1, x=-\frac{371267}{524288}\right]$$

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

```
(%i10) allroots(x^8+x^3+x+1);
```

(%o10)
$[x = .9098297401801199\,i + .2989522918873167, x = .2989522918873167 - .9098297401801199\,i\ ,$
$x = -.7081337759784606, x = .9807253637807569\,i - .4581925662678885\ ,$
$x = -.9807253637807569\,i - .4581925662678885, x = .5359278244124014\,i + 1.013307162369803\ ,$
$x = 1.013307162369803 - .5359278244124014\,i, x = -1.000000000000001]$

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

```
(%i11) f(x):=144 * sin(x) + 12*sqrt(3)*%pi - 36*x^2 - 12*%pi*x$

(%i12) find_root(f(z),z,1,2);
```

(%o12) $$1.899889962840263$$

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

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

```
(%i13) load(mnewton)$  /* carga el paquete */

(%i14) mnewton([2*u^u-5=u],[u],[1]);
0 errors, 0 warnings
```

(%o14) $$[[u = 1.912564904116258]]$$

y el sistema

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

```
(%i15) mnewton([x+3*log(x)-y^2, 2*x^2-x*y-5*x+1],[x, y], [5, 5]);
```

(%o15) $$[[x = 3.756834008012769, y = 2.779849592817897]]$$

```
(%i16) mnewton([x+3*log(x)-y^2, 2*x^2-x*y-5*x+1],[x, y], [1, -2]);
```

(%o16) $[[x = 1.373478353409809, y = -1.524964836379522]]$

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.

Prácticas

1. La velocidad de un móvil obedece la ecuación $v(t) = t^2 - \frac{1}{t}$, donde el tiempo t viene dado en segundos y la velocidad en m/s. Determínese el instante exacto en el que el móvil alcanza la velocidad de $1m/s$.

2. Tomando ciertas hipótesis restrictivas, como que una población crece de forma continua en el tiempo y que en ella no hai mortalidad, el número de individuos en función del tiempo se puede modelar mediante
$$N(t) = N_0 e^{\lambda t} + \frac{\nu}{\lambda}\left(e^{\lambda t} - 1\right),$$
donde N_0 es la población inicial, ν un coeficiente que da cuenta de la inmigración absorbida en la unidad de tiempo y λ la constante de crecimiento poblacional. Obténgase el valor de λ en función de las demás variables y calcúlese su valor para una población que inicialmente tiene un millón de individuos y hacia la que emigran 435000 individuos cada año, si habiendo transcurrido el primer año la población asciende a 1564000.

3. En los cálculos sobre tuberías hidráulicas es necesario determinar cierto coeficiente F para establecer, por ejemplo, la bomba necesaria para crear un caudal determinado. La ecuación de Colebrook es la que se utiliza en estos casos, la cual toma la forma
$$F = -2\log_{10}\left(\frac{k}{3.7d} + \frac{2.51F}{R}\right),$$
donde R es el número de Reynolds, el cual es adimensional y normalmente conocido, d es el diámetro de la tubería en mm y k la rugosidad de la cara interna de la tubería, también medida en mm. Calcúlese el valor de F para una tubería con $R = 2 \cdot 10^5$, $d = 300mm$ y $k = 0.25mm$.

3.5. Límites, derivadas e integrales

Las funciones `limit`, `diff` e `integrate` son las que nos interesan ahora. Los siguientes ejemplos sobre límites son autoexplicativos:

```
(%i1) limit(1/sqrt(x),x,inf);
```

(%o1) 0

```
(%i2) limit((exp(x)-exp(-x))/(exp(x)+exp(-x)),x,minf);
```

(%o2) -1

Donde hemos calculado $\lim_{x\to\infty} \frac{1}{\sqrt{x}}$ y $\lim_{x\to-\infty} \frac{e^x-e^{-x}}{e^x+e^{-x}}$, respectivamente. Nótese que los símbolos `inf` y `minf` representan a ∞ y $-\infty$, respectivamente.

Algunos límites necesitan información adicional, como $\lim_{x\to1} \frac{1}{x-1}$, pudiendo solicitar los límites laterales,

```
(%i3) limit(1/(x-1),x,1);
```

(%o3) und

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

(%o4) ∞

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

(%o5) $-\infty$

El símbolo `und` hace referencia al término inglés *undefined*.

El cálculo de derivadas requiere el concurso de la función `diff`; a continuación algunos ejemplos

```
(%i6) diff(x^log(a*x),x);  /* primera derivada */
```

(%o6) $$x^{\log(a\,x)}\left(\frac{\log\left(a\,x\right)}{x}+\frac{\log x}{x}\right)$$

```
(%i7) diff(x^log(a*x),x,2); /* derivada segunda */
```

(%o7) $$x^{\log(a\,x)}\left(\frac{\log\left(a\,x\right)}{x}+\frac{\log x}{x}\right)^2+x^{\log(a\,x)}\left(-\frac{\log\left(a\,x\right)}{x^2}-\frac{\log x}{x^2}+\frac{2}{x^2}\right)$$

```
(%i8) factor(%); /* ayudamos a Maxima a mejorar la respuesta */
```

(%o8) $$x^{\log(a\,x)-2}\left(\log^2\left(a\,x\right)+2\,\log x\,\log\left(a\,x\right)-\log\left(a\,x\right)+\log^2 x-\log x+2\right)$$

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

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

```
(%i9) diff(exp(x)*sin(y)*tan(z),x,3,y,5,z,2);
```

(%o9) $$2\,e^x\,\cos y\,\sec^2 z\,\tan z$$

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

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

(%o10) $$[z\left(x,y\right),x\left(u,v\right),y\left(u,v\right)]$$

```
(%i11) diff(z,v,1);
```

(%o11) $$\frac{d}{d\,v}\,y\left(\frac{d}{d\,y}\,z\right)+\frac{d}{d\,v}\,x\left(\frac{d}{d\,x}\,z\right)$$

```
(%i12) diff(z,y,1,v,1);
```

(%o12) $$\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,y^2}\,z\right)+\frac{d}{d\,v}\,x\left(\frac{d^2}{d\,x\,d\,y}\,z\right)$$

```
(%i13) diff(z,u,1,v,1);
```

(%o13) $$\begin{array}{l}\frac{d}{d\,u}\,y\left(\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,y^2}\,z\right)+\frac{d}{d\,v}\,x\left(\frac{d^2}{d\,x\,d\,y}\,z\right)\right)+\frac{d^2}{d\,u\,d\,v}\,y\left(\frac{d}{d\,y}\,z\right)+\\ \frac{d}{d\,u}\,x\left(\frac{d}{d\,v}\,x\left(\frac{d^2}{d\,x^2}\,z\right)+\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,x\,d\,y}\,z\right)\right)+\frac{d^2}{d\,u\,d\,v}\,x\left(\frac{d}{d\,x}\,z\right)\end{array}$$

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

```
(%i14) dependencies;
```

(%o14) $$[z\left(x,y\right),x\left(u,v\right),y\left(u,v\right)]$$

También podemos eliminar dependencias,

```
(%i15) remove(x,dependency);
```

(%o15) done

```
(%i16) dependencies;
```

(%o16) $$[z\left(x,y\right),y\left(u,v\right)]$$

```
(%i17) diff(z,y,1,v,1);
```

(%o17) $$\frac{d}{d\,v}\,y\left(\frac{d^2}{d\,y^2}\,z\right)$$

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`,

```
(%i18) depends(y,x)$
```

```
(%i19) diff(x^2+y^3=2*x*y,x);
```

(%o19) $$3\,y^2\left(\frac{d}{d\,x}\,y\right)+2\,x=2\,x\left(\frac{d}{d\,x}\,y\right)+2\,y$$

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,

```
(%i20) diff(x^2);
```

(%o20) $$2\,x\,del\,(x)$$

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

```
(%i21) diff(x^2+y^3=2*x*y);
```

(%o21) $$3\,y^2\,del\,(y)+\left(3\,y^2\left(\frac{d}{d\,x}\,y\right)+2\,x\right)\,del\,(x)=2\,x\,del\,(y)+\left(2\,x\left(\frac{d}{d\,x}\,y\right)+2\,y\right)\,del\,(x)$$

Recuérdese que durante este cálculo está todavía activa la dependencia declarada en la entrada `(%i18)`.

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 $x=1$,

```
(%i22) taylor((x*log(x))/(x^2-1),x,1,3);
```

(%o22) $$\frac{1}{2}-\frac{(x-1)^2}{12}+\frac{(x-1)^3}{12}+\cdots$$

```
(%i23) expand(%);
```

(%o23) $$\frac{x^3}{12}-\frac{x^2}{3}+\frac{5\,x}{12}+\frac{1}{3}$$

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,

```
(%i23) taylor(exp(x^2*sin(x*y)),[x,2,2],[y,0,2]);
```

(%o23) $$1+8\,y+32\,y^2+\cdots+\left(12\,y+96\,y^2+\cdots\right)\,(x-2)+\left(6\,y+120\,y^2+\cdots\right)\,(x-2)^2+\cdots$$

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

(%o24) $$120\,x^2\,y^2-384\,x\,y^2+320\,y^2+6\,x^2\,y-12\,x\,y+8\,y+1$$

En ocasiones resulta necesario operar con una función cuya derivada se desconozca, quizás porque la propia función sea desconocida. 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

```
(%i25) taylor(f(x + x^2),x,1,1);
```

$$(\%o25) \qquad f(2) + \left(\frac{d}{d\,x} f\left(x^2 + x\right)\Big|_{x=1} \right) (x-1) + \cdots$$

Podemos facilitar las cosas si le definimos una función derivada a f, a la que llamaremos df,

```
(%i26) gradef(f(x),df(x))$

(%i27) taylor(f(x+x^2),x,1,1);
```

$$(\%o27) \qquad f(2) + 3\,df(2)\,(x-1) + \cdots$$

El paquete `pdiff`, que se encuentra en la carpeta `share/contrib/`, aporta una solución alternativa a este problema.

Las integrales indefinidas y definidas las controla la función `integrate`; siguen algunos ejemplos

```
(%i28) integrate(cos(x)^3/exp(x),x);
```

$$(\%o28) \qquad \frac{e^{-x}\,(3\sin(3\,x) - \cos(3\,x) + 15\sin x - 15\cos x)}{40}$$

```
(%i29) integrate(cos(x)^3/exp(x),x,a,2);
```

$$(\%o29) \qquad \frac{e^{-2}\,(3\sin 6 - \cos 6 + 15\sin 2 - 15\cos 2)}{40} - \frac{e^{-a}\,(3\sin(3\,a) - \cos(3\,a) + 15\sin a - 15\cos a)}{40}$$

En ocasiones, maxima puede hacernos preguntas antes de responder

```
(%i30) integrate(1/x^a,x);

Is  a - 1  zero or nonzero?

nonzero;
```

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

Aquí se ha visto que nos ha preguntado sobre la nulidad de $a-1$; una vez le hemos contestado escribiendo `nonzero;` y pulsado la tecla de retorno, ya nos da la solución. En previsión de situaciones como la anterior, podemos darle al sistema toda la información relevante sobre los parámetros utilizados antes de pedirle el cálculo de la integral

```
(%i31) assume(a>1);
```

$$(\%o31) \qquad [a > 1]$$

```
(%i32) integrate(1/x^a,x);
```

(%o32) $$\frac{x^{1-a}}{1-a}$$

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,

```
(%i33) /* El integrador simbólico no puede con esta integral */
       integrate(exp(sin(x)),x,2,7);
```

(%o33) $$\int_2^7 e^{\sin x}\,dx$$

```
(%i34) /* Resolvemos numéricamente */
       quad_qag(exp(sin(x)),x,2,7,3);
```

(%o34) $$\left[4.747336298073747, 5.27060206376023 \times 10^{-14}, 31, 0\right]$$

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 ($\Pr(\chi^2_4 > 1)$),

```
(%i35) n:4$

(%i36) 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);

(%i37) quad_qagi(x^(n/2-1)*exp(-x/2)/2^(n/2)*gamma(n/2),x,1,inf);
```

(%o37) $$\left[.9097959895689501, 1.913452127046495 \times 10^{-10}, 165, 0\right]$$

```
(%i38) load(distrib)$

(%i39) 1 - cdf_chi2(1,n),numer;
```

(%o39) .9097959895689502

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.

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.

Prácticas

1. En un experimento, la velocidad (en m/s) de cierta partícula al pasar frente a un detector depende del tiempo t (en s) durante el cual ha estado sometida a ciertas condiciones magnéticas previas según la expresión

$$v(t) = \frac{37t^3 - 42t}{\sqrt{15t^6 + 1} + 2t^3}$$

   a) Calcúlese la velocidad registrada cuando el tiempo de aplicación de dichas condiciones ha sido $t = 2s$.

   b) ¿Cuál era su aceleración en esas mismas condiciones?

   c) ¿Cuál hubiese sido la velocidad registrada en caso de haber estado sometida a esas condiciones durante un tiempo infinito?

   d) ¿Qué aceleración habría alcanzado entonces?

2. Calcúlese la ecuación de la recta que mejor se aproxime a la función

$$y = \frac{\sqrt[3]{x}}{x + 2} + 3e^{x^2}$$

   en el punto de abscisa $x = 1$

3. Con los datos del problema anterior, calcúlese la ecuación de la parábola que se ajuste a la función.

4. A la edad de 14 años, Karl Friedrich Gauss (1777-1855) conjeturó que la cantidad de números primos menores o iguales a x viene dada por la expresión

$$\text{Card}\{p : \ p \text{ es número primo y } p \leq x\} \approx \int_2^x \frac{dt}{\log(t)}.$$

   Este resultado, conocido como *Teorema del número primo*, no sería formalmente demostrado hasta 1896 por Jacques Salomon Hadamard. Haciendo uso de este teorema, aproxímese el número de primos menores o iguales a 10^7.

3.6. 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)
$$\begin{pmatrix} \frac{4}{7} & 0 & \pi \\ \sqrt{2} & \log 3 & -9 \end{pmatrix}$$

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)
$$\begin{pmatrix} 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

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

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

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

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

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

(%o36) 3

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

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

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

(%o38) $span\,()$

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

```
(%i39) columnspace(%o32);   /* base del espacio generado */
0 errors, 0 warnings
```

(%o39) $span\left(\begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 6 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \\ 6 \end{pmatrix}\right)$

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,

```
(%i40) matrixmap(sin,%o32);
```

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

Prácticas

1. Calcúlese el lugar geométrico de los puntos (x, y) del plano para los que la matriz

$$\begin{pmatrix} 3 & 2 & -\frac{8}{5} & x \\ -\frac{3}{5} & x & 9 & 1 \\ 6 & 3 & \frac{1}{9} & -2 \\ y & 2 & 3 & -4 \end{pmatrix}$$

   es singular.

2. Calcúlese la matriz X sabiendo que $A \cdot X \cdot B = C$, donde

$$A = \begin{pmatrix} 4 & -7 & 3 \\ 1 & 8 & -5 \\ 7 & 1 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 5 & 6 \\ -8 & 8 & 9 \end{pmatrix} \text{ y } C = \begin{pmatrix} -1 & 2 & -3 \\ 4 & -u & 6 \\ -7 & 8 & -9 \end{pmatrix}$$

3. Sea

$$A = \begin{pmatrix} 2 & -1 & 1 & 0 \\ -1 & 1 & 0 & -1 \\ -1 & 0 & 1 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

la matriz asociada a cierta aplicación lineal $f : \mathbb{R}^4 \longmapsto \mathbb{R}^4$.

a) Calcúlese la imagen del vector $(2, 5, 6, 8)^T$.

b) Calcúlese una base del espacio nulo de f. Compruébese, mediante el cálculo del rango, si el vector $(0, 3, 3, 0)^T$ pertenece a dicho espacio. (Puede ser de utilidad la función `args`.)

c) Calcúlese una base del espacio engendrado por f. ¿Cómo podríamos combinar las funciones `addcol`, `args` y `apply` para formar una matriz con el resultado que nos devuelve Maxima?

3.7. Vectores y campos

En Maxima, los vectores se introducen como simples listas, 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) $[a+1, b+2, c+3]$

```
(%i2) s*[a,b,c];
```

(%o2) $[a\,s, b\,s, c\,s]$

```
(%i3) [1,2,3].[a,b,c]; /* producto escalar */
```

(%o3) $3\,c + 2\,b + a$

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

Los operadores diferenciales que son de uso común en el ámbito de las funciones 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 líneas 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) $$grad\left(z^2+y^2-x\right)\cdot grad\left(\log\left(z^2+y^2\right)+4\,x\right)$$

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 cálculo de éstas.

```
(%i12) express(%);
```

(%o12) $$\begin{array}{l}\frac{d}{d\,z}\left(z^2+y^2-x\right)\left(\frac{d}{d\,z}\left(\log\left(z^2+y^2\right)+4\,x\right)\right)+\\ \frac{d}{d\,y}\left(z^2+y^2-x\right)\left(\frac{d}{d\,y}\left(\log\left(z^2+y^2\right)+4\,x\right)\right)+\\ \frac{d}{d\,x}\left(z^2+y^2-x\right)\left(\frac{d}{d\,x}\left(\log\left(z^2+y^2\right)+4\,x\right)\right)\end{array}$$

```
(%i13) ev(%,diff);
```

(%o13) $$\frac{4\,z^2}{z^2+y^2}+\frac{4\,y^2}{z^2+y^2}-4$$

```
(%i14) ratsimp(%);
```

(%o14) $$0$$

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) $$div\left(\left[x\,y, x^2\,z, e^{y+x}\right]\right)$$

```
(%i17) express (%);
```

$$(\%o17) \qquad \frac{d}{d\,y}\left(x^2\,z\right)+\frac{d}{d\,z}\,e^{y+x}+\frac{d}{d\,x}\left(x\,y\right)$$

```
(%i17) ev (%, diff);
```

$$(\%o17) \qquad y$$

```
(%i18) curl (F);  /* rotacional */
```

$$(\%o18) \qquad curl\left(\left[x\,y,x^2\,z,e^{y+x}\right]\right)$$

```
(%i19) express (%);
```

$$(\%o19) \qquad \left[\frac{d}{d\,y}\,e^{y+x}-\frac{d}{d\,z}\left(x^2\,z\right),\frac{d}{d\,z}\left(x\,y\right)-\frac{d}{d\,x}\,e^{y+x},\frac{d}{d\,x}\left(x^2\,z\right)-\frac{d}{d\,y}\left(x\,y\right)\right]$$

```
(%i20) ev (%, diff);
```

$$(\%o20) \qquad \left[e^{y+x}-x^2,-e^{y+x},2\,x\,z-x\right]$$

```
(%i21) laplacian (F);  /* laplaciano */
```

$$(\%o21) \qquad laplacian\left(\left[x\,y,x^2\,z,e^{y+x}\right]\right)$$

```
(%i22) express (%);
```

$$(\%o22) \qquad \frac{d^2}{d\,z^2}\,\left[x\,y,x^2\,z,e^{y+x}\right]+\frac{d^2}{d\,y^2}\,\left[x\,y,x^2\,z,e^{y+x}\right]+\frac{d^2}{d\,x^2}\,\left[x\,y,x^2\,z,e^{y+x}\right]$$

```
(%i23) ev (%, diff);
```

$$(\%o23) \qquad \left[0,2\,z,2\,e^{y+x}\right]$$

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,

```
(%i24) milaplaciano(v):= ev(express(laplacian(v)), diff) $

(%i25) milaplaciano(F);
```

$$(\%o25) \qquad \left[0,2\,z,2\,e^{y+x}\right]$$

Por último, el paquete `vect` incluye también la definición del producto vectorial, al cual le asigna el operador `~`,

```
(%i26) [a, b, c] ~ [x, y, z];
```

(%o26) $$[a, b, c] \sim [x, y, z]$$

```
(%i27) express(%);
```

(%o27) $$[b\,z - c\,y, c\,x - a\,z, a\,y - b\,x]$$

Cuando se trata de hacer cálculos generales en los que las expresiones de los campos, tanto escalares como vectoriales, son desconocidas, se deben declarar las dependencias de unas variables respecto de otras para que las derivadas funcionen correctamente. Como ejemplo, demostremos la identidad $\nabla \cdot (\nabla \times \mathbf{F}) = 0$, pero no sin antes definir un par de funciones que nos serán útiles,

```
(%i28) midivergencia(v):= ev(express(div(v)), diff) $

(%i29) mirotacional(v):= ev(express(curl(v)), diff) $

(%i30) F:[f1,f2,f3];
```

(%o30) $$[f_1, f_2, f_3]$$

```
(%i31) depends(F,[x,y,z]);
```

(%o31) $$[f_1\,(x, y, z), f_2\,(x, y, z), f_3\,(x, y, z)]$$

```
(%i32) midivergencia(mirotacional(F));
```

(%o32) $$0$$

Prácticas

1. Dada la espiral logarítmica $\mathbf{x}(t) = (e^t \sin(t), e^t \cos(t))$, demuéstrese que el ángulo entre los vectores $\mathbf{x}(t)$ y $\dot{\mathbf{x}}(t)$ se mantiene constante. (Recuérdese lo que se comentó en la sección 3.3 acerca de la función `trigsimp`.)
2. Calcúlense
   a) $\nabla \cdot (xe^y\mathbf{i} - \sin xy\mathbf{j} + z\mathbf{k})$
   b) $\nabla \times (y^2z\mathbf{i} + z^2x\mathbf{j} + x^2y\mathbf{k})$
3. Demuéstrense las identidades
   a) $\nabla \times \nabla\phi = \mathbf{0}$, siendo ϕ un campo escalar.
   b) $\nabla \times (\nabla \times \mathbf{F}) = \nabla(\nabla \cdot \mathbf{F}) - \nabla^2\mathbf{F}$, siendo $\mathbf{F}$ un campo vectorial.

3.8. Gráficos

Maxima no está habilitado para realizar gráficos de forma autónoma, 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.

Las funciones `plot2d` y `plot3d` son las rutinas gráficas que están definidas en el propio núcleo de Maxima, y con ellas es posible dar instrucciones tanto a Gnuplot como a Openmath. El módulo adicional `draw` está orientado exclusivamente a Gnuplot, por lo que hace un mejor aprovechamiento de él, a la vez que su sintaxis no exige por parte del usuario conocimiento alguno del lenguaje de este programa gráfico. Es al módulo `draw` al que dedicaremos esta sección.

Aquí, las funciones a utilizar son `draw2d`, `draw3d` y `draw`, para escenas en 2d, 3d y para gráficos múltiples, respectivamente. Puesto que se trata de un módulo adicional, será necesario cargarlo en memoria antes de hacer uso de él,

```
(%i1) load(draw)$
```

Empecemos por unos sencillos ejemplos de gráficos sobre el plano. En primer lugar dibujaremos una función definida de forma explícita,

```
(%i2) draw2d(explicit(u^3-3*u^2+5/8,u,-1,4))$
```

En este caso, `explicit` indica el objeto que queremos dibujar; el primer argumento es un polinomio cúbico de variable u, el segundo argumento es el nombre de esta misma variable, al que siguen los límites del subdominio que queremos representar, $(-1, 4)$. Podemos dibujar en un mismo gráfico más de un objeto, así añadimos al polinomio cúbico anterior una curva paramétrica,

```
(%i3) draw2d(
         explicit(u^3-3*u^2+5/8, u, -1, 4),
         parametric(t*3^t, 10*sin(t), t, -3/2*%pi, %pi/3))$
```

Junto con los objetos gráficos, también existen opciones que modifican o alteran las presentaciones por defecto. A continuación hacemos uso de algunas de estas opciones para mejorar el aspecto del último gráfico obtenido.

```
(%i4) draw2d(
         filled_func = true,
         fill_color  = cyan,
         line_width  = 3,
         explicit(u^3-3*u^2+5/8, u, -1, 4),
         color       = red,
         key         = "Esta es la curva parametrica",
         parametric(t*3^t, 10*sin(t), t, -3/2*%pi, %pi/3))$
```

Las opciones, que siguen todas la sintaxis `nombre_opción=valor`, se leen secuencialmente, permaneciendo sus valores activos hasta que éste se cambia por otro. No todas las opciones afectan a todos los objetos gráficos; en el ejemplo anterior, se ha hecho uso de la opción `filled_func`, que por defecto vale `false`, pero aunque durante toda la secuencia toma el valor `true`, afecta sólo al objeto `explicit`, pero no al `parametric`. Tanto los objetos gráficos como sus opciones están descritos, junto con ejemplos, en el sistema de ayuda de Maxima.

Algunas opciones son de carácter global, no afectando a objetos particulares, sino al gráfico en su conjunto. Veamos un ejemplo aplicado a una escena en tres dimensiones; aquí, `title` es global y puede colocarse en cualquier punto de la descripción de la escena, pero `color` es local y debe colocarse antes del objeto a dibujar.

```
(%i5) draw3d(
        title = "Ejemplo en tres dimensiones",
        color = green,
        explicit(exp(-x^2-y^2), x, -5, 5, y, 0, 5))$
```

Una vez presentado el estilo general del módulo `draw`, veamos algunos ejemplos más elaborados.

```
(%i6) 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 )$
```

Aquí los objetos gráficos son `explicit`, `parametric` y `points`, las opciones locales son `key`, `color`, `line_width`, `nticks`, `line_type`, `points_joined` y `line_width` y, finalmente, las opciones globales son `title` y `terminal`. 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. Como ya se ha comentado, 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`[1]

```
(%i7) cats: read_xpm("gatos.xpm")$

(%i8) draw2d(
        terminal    = eps,
        yrange      = [-4,10],
        ellipse(5,3,8,6,0,360),
        image(cats,0,0,10,7),
        line_width  = 2,
```

[1]El formato gráfico XPM es el único que puede leer Maxima.

```
        head_length = 0.3,
        color       = blue,
        label(["This is Francisco",-1,-0.5]),
        vector([-1,0],[2,4]),
        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 3.1, también existen `polygon`, `rectangle`, `polar`, `implicit` y `geomap`, este último para mapas cartográficos.

Desarrollamos a continuación algunas escenas tridimensionales. En primer lugar, el módulo de la función Γ de Euler, junto con sus líneas de contorno.

```
(%i9) gamma2(x,y):=
        block([re,im,g:gamma(x+%i*y)],
          re:realpart(g),
          im:imagpart(g),
          sqrt(re^2+im^2))$

(%i10) 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 3.1, apartados *c)* y *d)*.

```
(%i11) 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 3.1.

```
(%i12) 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 )$
```

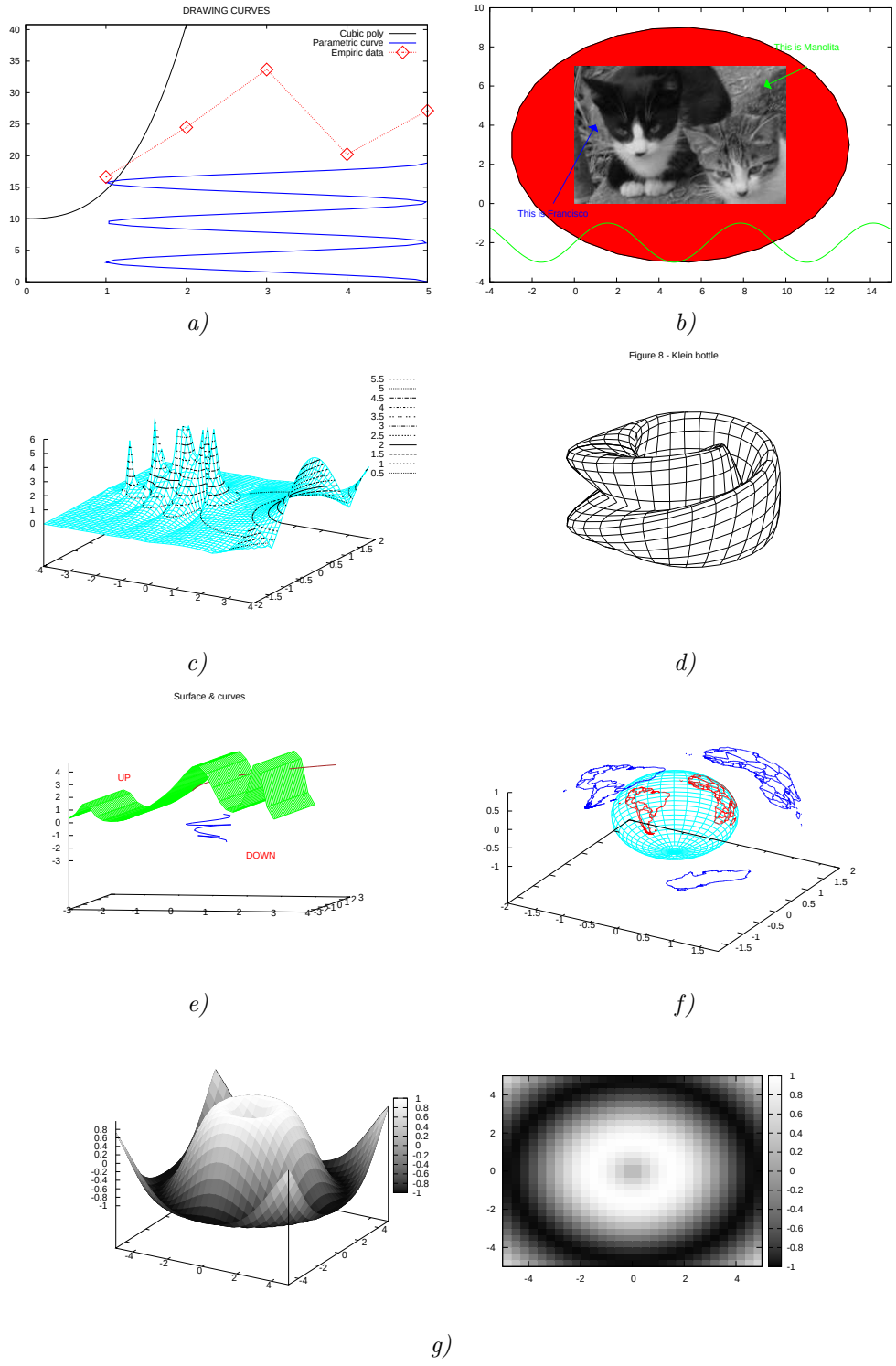
En el siguiente ejemplo hacemos una proyección esférica del hemisferio sur, que se ve en el apartado *f)* de la Figura 3.1. El paquete `worldmap` carga en memoria las coordenadas latitud–longitud de las líneas fronterizas y costeras de todos los países del mundo, así como algunas funciones necesarias para su manipulación y procesamiento,

```
(%i13) load(worldmap)$
```

```
(%i14) 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)$
```

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



a) *b)* *c)* *d)* *e)* *f)* *g)*

Figura 3.1: Gráficos generados con el paquete draw: *a)* y *b)* con draw2d; *c)*, *d)*, *e)* y *f)* con draw3d; *g)*, un gráfico múltiple.

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 3.1.

```
(%i15) escena1:
         gr3d(surface_hide = true,
              enhanced3d   = true,
              palette      = gray,
              explicit(sin(sqrt(x^2+y^2)),x,-5,5,y,-5,5))$

(%i16) 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))$

(%i17) 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.

Prácticas

1. *a)* Dibújese la función $y = x^3 - 2x^2 + 5$ en el subdominio $(-2, 5)$, conjuntamente con su recta tangente en el punto de abscisa $x = 3$.

   b) Amplíese el gráfico anterior añadiendo el polinomio de segundo grado que mejor se aproxime a la función en $x = 3$.

   c) Añádase el punto de abscisa $x = 3$ en formato de círculo relleno (opción `point_type`) y algo grueso para que destaque (opción `point_size`).

   d) Mejórese estéticamente el gráfico atendiendo a las siguientes indicaciones:

   - asignar diferentes colores/anchos a las curvas,
   - dotar de leyendas a las curvas,
   - añadir los ejes de coordenadas,
   - poner un título.

2. Vamos a construir un gráfico animado en pantalla, para lo cual necesitaremos de los bucles, que serán tratados con mayor profundidad en el capítulo de programación. Baste de momento saber que el código

```
for i:1 thru 5 step 0.5 do print(i);
```

imprimirá los valores que se le irá asignando a `i`, de 1 a 5, en pasos de 0.5.

a) Dibújese la función paramétrica $\mathbf{x}(t) = (t\cos(t), t\sin(t), t)$ para $t \in (0, 1)$

b) Idem para $t \in (0, 3)$ y $t \in (0, 5)$.

c) Si repetimos los gráficos anteriores haciendo variar de cada vez el límite superior del intervalo del parámetro t, crearemos una animación. Hágase de forma que este límite varíe de 0 a 10 en saltos de 0.1.

d) Se observará un extraño efecto debido a que cada gráfico utiliza unas escalas diferentes. Fíjense las escalas para evitar esto haciendo uso de las opciones `xrange`, `yrange` y `zrange`.

e) Un efecto que a veces puede ayudar al observador a apreciar mejor las formas tridimensionales consiste en modificar levemente el punto de observación entre fotograma y fotograma. Véase de incluir este efecto con las opciones `rot_verical` y `rot_horizontal`.

3.9. Ecuaciones diferenciales

Con Maxima se pueden resolver simbólicamente 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) /* ecuación 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) $$y = \frac{x^2 + 2\,x + 2}{2} - \frac{7}{2}$$

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 función `desolve` resuelve sistemas de ecuaciones diferenciales ordinarias lineales utilizando la transformada de Laplace. 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) $$y\left(x\right)=\frac{\left(w+v+2\,u\right)\,\sin x}{2}+\frac{\left(w-v\right)\,\cos x}{2}+\frac{\left(w+v\right)\,e^{-x}}{2}$$

Además de las funciones anteriores, el paquete `plotdf` permite representar 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 3.2 izquierda, 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.

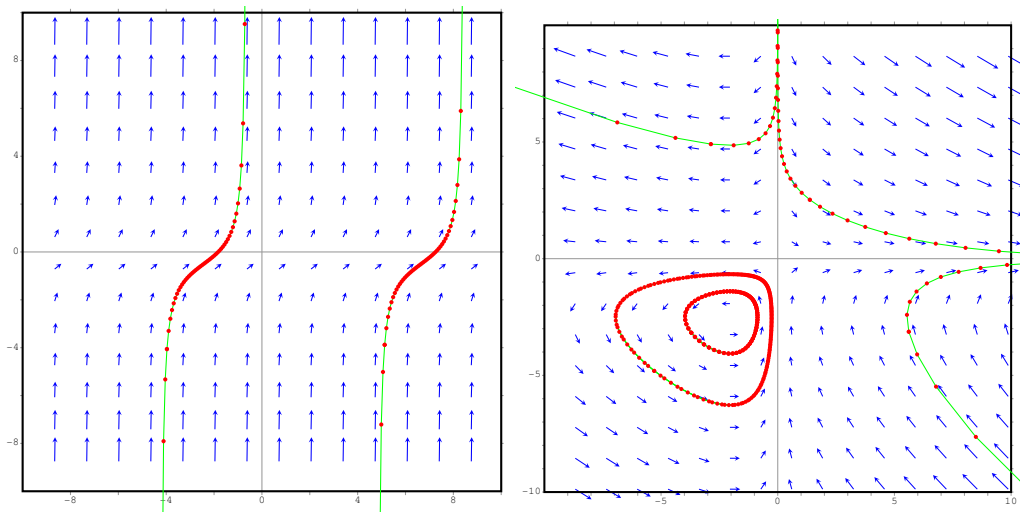


Figura 3.2: Campos de direcciones creados con la función 'plotdf': a la izquierda el campo de la ecuación $\frac{dy}{dx} = 1 + y + y^2$, a la derecha el correspondiente al modelo predador-presa.

```
(%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"]);
```

En la Figura 3.2 derecha 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 simbólicamente 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)
$$\frac{d}{d\,t}\,y + 2\,y^2 - e^{-3\,t} = 0$$

```
(%i26) ode2(ec,y,t);
```

(%o26) false

Abordamos ahora el problema con un enfoque numérico, para lo cual definimos la expresión

$$f(t, y) = \frac{dy}{dt} = -2y^2 + \exp(-3t)$$

```
(%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 ,
−.09974417126696171, −.06705614729331426, −.04779722499498942, −.03570266617749457,
−.02766698775990988, −.02207039201652747, −.01801909665196759]]

```
(%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 3.3 *a)*.

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 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.

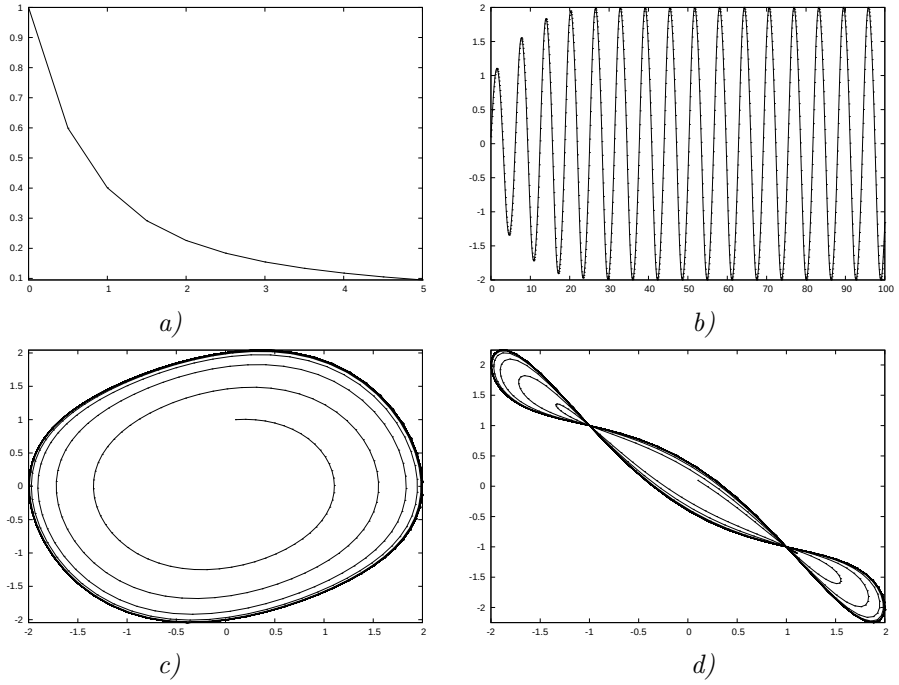


Figura 3.3: Resolución numérica de ecuaciones diferenciales con 'diffeq': *a)*, solución de la ecuación de primer orden $\frac{dy}{dt} = -2y^2 + \exp(-3t), y(0) = 1$; *b)*, solución de la ecuación de segundo orden $\frac{d^2y}{dt^2} = 0.2(1-y^2)\frac{dy}{dt} - y$, con las condiciones $y(0) = 0.1$ y $y'(0) = 1$; *c)*, diagrama (y, y'); *d)*, diagrama (y, y'').

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 3.3.

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) draw2d(
         points_joined = true,
         point_type    = dot,
         points(%),
         terminal      = eps)$
```

Para más información sobre la sintaxis de esta función, ejecútese `? rk`

Prácticas

1. Resuélvase la función $y(x)$ que verifica la ecuación diferencial

$$y' = \frac{(x^2+1)(1-y^2)}{xy}$$

y que pasa por el punto de coordenadas $(2,2)$. Utilícese el objeto gráfico `implicit` de la rutina `draw2d` para representar la solución en el rectángulo $[-3,3]\times[-3,3]$.

2. Resuélvase el problema

$$\begin{cases} x'''(t)+4x''(t)-5x(t)=0 \\ x(0)=1, x'(0)=0, x''(0)=0 \end{cases}$$

y analícese el comportamiento asintótico de la solución.

3. El *efecto mariposa* es la expresión que Edward Lorenz acuñó en 1963 para describir el comportamiento caótico y la fuerte dependencia a las condiciones iniciales del sistema de ecuaciones no lineales

$$\begin{cases} x'(t) = s(y-x) \\ y'(t) = rx - y - xz \\ z'(t) = xy - bz \end{cases}$$

donde $s=10$, $r=126.52$ y $b=8/3$. Aplíquese el método de Runge-Kutta (función `rk`) para obtener una solución numérica para los valores iniciales $x(0)=-7.69$, $y(0)=-15.61$ y $z(0)=90.39$, siendo $t\in[0,10]$, con incrementos de 0.005. Represéntese gráficamente la solución obtenida (Nótese que la solución son puntos en el espacio, por lo que la rutina gráfica a utilizar será `draw3d`; en lo demás, todo es similar a la entrada `%i39`).

3.10. Probabilidades y análisis de datos

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) $$\frac{e^{-\frac{1}{s}}}{120\,s^5}$$

```
(%i5) quantile_student_t(0.05,25);
```

(%o5) $$-1.708140543186975$$

```
(%i6) mean_weibull(3,67);
```

(%o6) $$\frac{67\,\Gamma\left(\frac{1}{3}\right)}{3}$$

```
(%i7) var_binomial(34,1/8);
```

(%o7) $$\frac{119}{32}$$

```
(%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) $$\frac{12}{5}$$

Por último, también es posible la simulación de muestras independientes de cualquiera de las distribuciones anteriores,

```
(%i10) random_negative_binomial(9,1/5,15);
```

(%o10) $$[23, 12, 39, 45, 47, 32, 56, 31, 28, 38, 29, 41, 37, 43, 34]$$

Para el análisis descriptivo de datos muestrales se dispone del paquete `descriptive`. La distribución de Maxima incluye algunos ficheros de datos muestrales de ejemplo; uno de ellos, `wind.data`, contiene una muestra multivariante de tamaño 100. En esta breve sesión, cargamos el paquete y la muestra para luego realizar algunos cálculos y representaciones gráficas:

```
(%i11) load (descriptive)$

(%i12) /* La muestra se guarda en una matriz */
       s2 : read_matrix (file_search ("wind.data"))$

(%i13) length(s2);  /* tamaño muestral */
```

(%o13) $$100$$

```
(%i14) /* Vector de medias */
       mean(s2);
```

(%o14) $[9.9485, 10.1607, 10.8685, 15.7166, 14.8441]$

```
(%i15) /* Dígitos significativos a imprimir */
       fpprintprec : 7$
```

```
(%i16) /* Matriz de correlaciones */
        cor(s2);
```

(%o16)
$$\begin{pmatrix} 1.0 & .8476339 & .8803515 & .8239624 & .7519506 \\ .8476339 & 1.0 & .8735834 & .6902622 & 0.782502 \\ .8803515 & .8735834 & 1.0 & .7764065 & .8323358 \\ .8239624 & .6902622 & .7764065 & 1.0 & .7293848 \\ .7519506 & 0.782502 & .8323358 & .7293848 & 1.0 \end{pmatrix}$$

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. Como ya tenemos en memoria las funciones de `distrib`, no necesitamos volver a cargar el paquete,

```
(%i19) abs: makelist(random_continuous_uniform(0,10),k,1,100)$
```

```
(%i20) f(x):=3*(x-5)^2$
```

```
(%i21) 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:

```
(%i22) load(lsquares)$
```

```
(%i23) param: lsquares_estimates (data,[x,y],y=a*x^3+b*x^2+c*x+d,[a,b,c,d]),numer;
```

(%o23)
$$[[a = -0.002705800223881305, b = 3.018798873646606\ ,\\ c = -29.94151342602112, d = 74.78603431944423]]$$

Vemos claramente que el término de tercer grado es supérfluo, por lo que reajustamos al de segundo grado,

```
(%i24) param: lsquares_estimates (data,[x,y],y=b*x^2+c*x+d,[b,c,d]),numer;
```

(%o24) $[[b = 2.979110687882263, c = -29.78353057922009, d = 74.64523259993118]]$

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

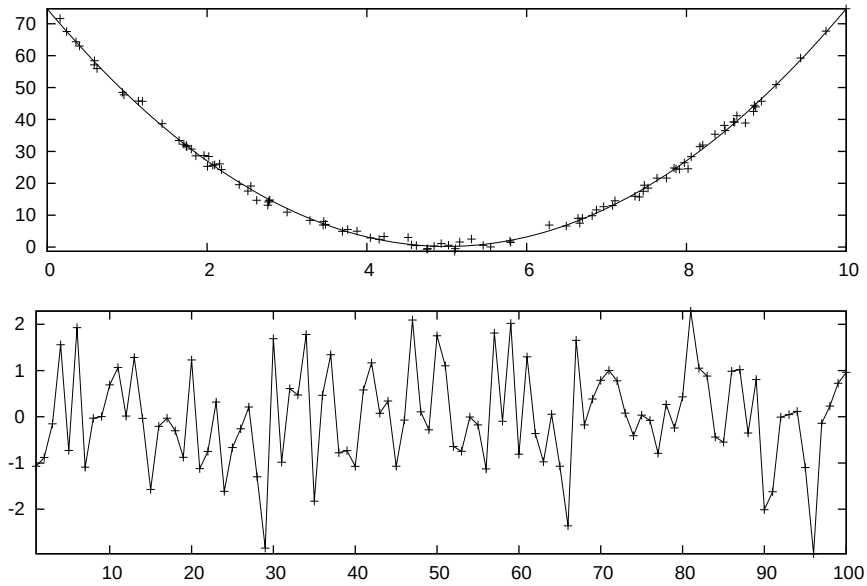


Figura 3.4: Resultados gráficos del ajuste por mínimos cuadrados.

```
(%i25) lsquares_residual_mse(data, [x,y], y=b*x^2+c*x+d, first(param));
```

(%o25) 1.144872557335554

También podemos calcular el vector de los residuos, definidos como $\text{lhs}(e_i) - \text{rhs}(e_i)$, $\forall i$, para a continuación representarlos gráficamente junto con los datos y la curva ajustada, tal como se aprecia en la Figura 3.4.

```
(%i26) res: lsquares_residuals(data, [x,y], y=b*x^2+c*x+d, first(param))$

(%i27) scene1: gr2d(points(data), explicit(ev(b*x^2+c*x+d,first(param)),x,0,10))$

(%i28) scene2: gr2d(points_joined=true, points(res))$

(%i29) draw(terminal = eps, scene1, scene2)$
```

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 que nos pueden ayudar en este tipo de análisis.

Prácticas

1. *a*) Simúlese una muestra de tamaño 100 de cierto fenómeno aleatorio normal de media nula y desviación típica 1 (*ruído gaussiano blanco*), guardando el resultado en la variable `m`.

   b) Calcúlese la media y desviación típica (`std`) muestrales de los datos recién simulados.

c) Hágase el histograma de la muestra con la función `histogram` del módulo `descriptive`.

2. Ajústese la función $y = a \cdot b^x$ a los siguientes datos empíricos por el método de los mínimos cuadrados:

$$\{(1, 4.97), (2, 4.85), (4, 6.80), (6, 9.04), (7, 12.48), (9, 16.40)\}.$$

¿Qué valores asignaríamos a $x = 3$, $x = 5$ y $x = 8$? Dibújense en un mismo gráfico los datos empíricos junto con la función estimada.

3.11. Interpolación numérica

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);
```

$$(\%o3) \quad \left(\frac{13}{2} - \frac{3\,x}{2}\right)\ charfun_2\,(x, -\infty, 3) + 2\ charfun_2\,(x, 7, \infty) + (37 - 5\,x)\ charfun_2\,(x, 6, 7) + \left(\frac{5\,x}{3} - 3\right)\ charfun_2\,(x, 3, 6)$$

```
(%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]);
```

$$(\%o5) \quad \left[2, \frac{62}{21}, \frac{5\,\pi}{3} - 3\right]$$

```
(%i6) float(%);
```

$$(\%o6) \quad [2.0, 2.952380952380953, 2.235987755982989]$$

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]);
```

$$(\%o7) \qquad \begin{pmatrix} 7 & 2 \\ 8 & 2 \\ 1 & 5 \\ 3 & 2 \\ 6 & 7 \end{pmatrix}$$

```
(%i8) lagrange(datos2,varname='w);
```

$$(\%o8) \qquad \frac{73\,w^4}{420} - \frac{701\,w^3}{210} + \frac{8957\,w^2}{420} - \frac{5288\,w}{105} + \frac{186}{5}$$

```
(%i9) g(w):=''%$

(%i10) map(g,[7.3,25/7,%pi]), numer;
```

$$(\%o10) \qquad [1.043464999999799, 5.567941928958199, 2.89319655125692]$$

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 3.5.

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);
```

(%o13)
$$\left(\frac{1159\,x^3}{3288}-\frac{1159\,x^2}{1096}-\frac{6091\,x}{3288}+\frac{8283}{1096}\right)\ charfun_2\left(x,-\infty,3\right)+\left(-\frac{2587\,x^3}{1644}+\frac{5174\,x^2}{137}-\frac{494117\,x}{1644}+\frac{108928}{137}\right)\ charfun_2\left(x,7,\infty\right)+\left(\frac{4715\,x^3}{1644}-\frac{15209\,x^2}{274}+\frac{579277\,x}{1644}-\frac{199575}{274}\right)\ charfun_2\left(x,6,7\right)+\left(-\frac{3287\,x^3}{4932}+\frac{2223\,x^2}{274}-\frac{48275\,x}{1644}+\frac{9609}{274}\right)\ charfun_2\left(x,3,6\right)$$

```
(%i14) s1(x):=''%$

(%i15) map(s1,[7.3,25/7,%pi]), numer;
```

(%o15) $[1.438224452554664, 3.320503453379974, 2.227405312429507]$

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);
```

(%o16)
$$\left(\frac{2339\,z^3}{2256}-\frac{14515\,z^2}{2256}+\frac{24269\,z}{2256}-\frac{271}{752}\right)\ charfun_2\left(z,-\infty,3\right)+\left(-\frac{1553\,z^3}{564}+\frac{35719\,z^2}{564}-\frac{68332\,z}{141}+\frac{174218}{141}\right)\ charfun_2\left(z,7,\infty\right)+\left(\frac{613\,z^3}{188}-\frac{35513\,z^2}{564}+\frac{56324\,z}{141}-\frac{38882}{47}\right)\ charfun_2\left(z,6,7\right)+\left(-\frac{4045\,z^3}{5076}+\frac{1893\,z^2}{188}-\frac{5464\,z}{141}+\frac{2310}{47}\right)\ charfun_2\left(z,3,6\right)$$

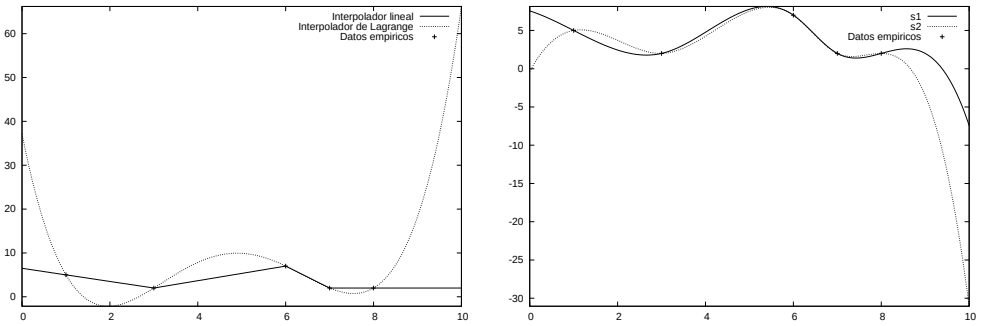
```
(%i17) s2(z):=''%$

(%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 3.5.

```
(%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)$
```



a) *b)*

Figura 3.5: Interpolación: *a)* lineal y de Lagrange; *b) Splines* cúbicos.

Prácticas

1. Los datos siguientes hacen referencia a la velocidad v alcanzada por un cohete transcurrido un tiempo t desde su lanzamiento.

t, en s	0	10	15	20	22.5	30
v, en m/s	0	227.04	362.78	517.35	602.97	901.67

   a) Calcúlese el polinomio de interpolación de Lagrange para estos datos.

   b) ¿Cuál es la expresión que estima la aceleración en el instante t? ¿Qué aceleración tiene el cohete en $t = 17.5s$?

   c) ¿Cuál es la expresión que estima el espacio recorrido después de transcurrido t segundos desde su lanzamiento? ¿Qué distancia lleva recorrida después de $t = 17.5s$?

2. Hágase una interpolación paramétrica (esto es, una función interpoladora para cada variable) por el método de los *splines* cúbicos de los siguientes puntos de $\mathbb{R}^3$.

x	1.08	-0.83	-1.98	-1.31	,0.57
y	2.61	2.82	0.44	-2.35	-2.97
z	2.50	5.00	7.50	10.00	12.50

   Represéntese la curva resultante junto con los datos muestrales. (Pueden consultarse en la documentación los objetos `points` y `parametric` de `draw3d`.)

Capítulo 4

Programación

A diferencia de los programas de Matemáticas privativos, cuyo código es cerrado e inaccesible para el usuario, Maxima se puede programar a dos niveles: en el lenguaje propio de Maxima, que es muy sencillo, y al que dedicaremos la próxima sección, y en el lenguaje en el que está diseñado todo el sistema, Lisp, al que dedicaremos algunos párrafos en la segunda sección de este capítulo.

4.1. Nivel 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) 11962222086548019456196316149565771506438373376000000000

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...

Los operadores lógicos que frecuentemente se utilizarán con las relaciones anteriores son `and`, `or` y `not`, con sus significados obvios.

```
(%i14) makelist(is(log(x)<x-1), x, [0.9,1,1.1]);
```

(%o14) [true, false, true]

```
(%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) $$g^7 - f^7 + e^7 - d^7 + c^7 - b^7 + a^7$$

```
(%i29) sumdif(%pi,u+v);
```

(%o29) $$(v+u)^{\pi}$$

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);
```

$$(\%o34) \qquad [\sin 1 + \cos 1, \sin 2 + \cos 2, \sin 3 + \cos 3, \sin 4 + \cos 4, \sin 5 + \cos 5]$$

```
(%i35) fun_a( lambda([z], sin(z)+cos(z)) );
```

$$(\%o35) \qquad [\sin 1 + \cos 1, \sin 2 + \cos 2, \sin 3 + \cos 3, \sin 4 + \cos 4, \sin 5 + \cos 5]$$

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 menmoria, 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

Ya para terminar, a modo de ejemplo de programación, una función que analiza los puntos críticos de una función diferenciable de variables independientes x e y.

```
/* Analiza los puntos críticos de una función de  */
/* variables independientes x e y.                */
/* Ejemplos de uso:                               */
/*     extremos(exp(x-y)*x*y);                    */
/*     extremos(y^2 + (x + 1)^2*y + (x + 1)^4);   */

load(vect)$

extremos(expr):=
  block([gr,cr,he,x,y,mn,de],

    /* vector gradiente de la función */
    gr: grad(expr),

    /* hessiano */
```

```
he: hessian(expr, [x,y]),

/* Resuelve para x e y el gradiente nulo */
cr: solve(ev(express(gr),diff), [x,y]),

/* s recorre las soluciones encontradas por solve */
for s in cr do (

  /* valor del hessiano para la solución s */
  mn: subst(s, he),

  /* determinante del hessiano */
  de: determinant(mn),

  /* controla el tipo de punto critico */
  if de > 0
    then if mn[1,1] > 0
           then print("Minimo relativo en ", s)
           else print("Maximo relativo en ", s)
    else if de < 0
           then print("Punto silla en ", s)
           else print("Determinante nulo en ", s) ) )$
```

Prácticas

1. La función `solve` da como resultado una lista de igualdades (véase `solve(x^3-x^2+x-5)`). Es frecuente entre los usuarios que a la vista del resultado, se quiera tomar una de las soluciones para seguir trabajando con ella. Escríbase una función de nombre `solucion` que admita dos parámetros: el primero sería una lista de igualdades como las devueltas por `solve` y el segundo indicando qué número de solución se quiere extraer. (Consúltese la documentación sobre la función `rhs`.)

2. Cuando se escribe una función en la que los parámetros deban cumplir determinadas condiciones, antes de proceder con los cálculos es necesario chequear que tales condiciones se cumplen y emitir mensajes descriptivos para ayudar al usuario a detectar y corregir sus errores. Haciendo uso conveniente de la función `error`, añádanse los siguientes controles de entrada a la función `solucion` anterior:

   a) el primer parámetro debe ser necesariamente una lista,

   b) todos los elementos de la lista deben ser igualdades (consúltense `op`, `atom` y `every`),

   c) el segundo parámetro debe ser un entero positivo (`integerp`) no mayor que el número de elementos del primer parámetro.

3. Dado el número complejo $c \in \mathbb{C}$, se define para cada $z \in \mathbb{C}$ la sucesión

$$\mathcal{J}_{c,z,n} = \begin{cases} z_0 = z \\ z_{n+1} = z_n^2 + c \end{cases}$$

Se llama entonces *conjunto de Julia* de parámetro c al conjunto

$$\mathcal{J}_c = \left\{ z \in \mathbb{C} : \lim_n \mathcal{J}_{c,z,n} < \infty \right\}.$$

Escríbase en Maxima una función de la forma

`conjulia(c,p1,p2,nr,ni)`

donde `c` es un número complejo, `p1` y `p2` son los vértices inferior izquierdo y superior derecho, respectivamente, de una ventana rectangular sobre el plano complejo, y `nr` y `ni` son el número de puntos en los que se discretizan la dirección real e imaginaria, respectivamente. Para cada $z \in \mathbb{C}$, se irán calculando los sucesivos términos de la sucesión $\mathcal{J}_{c,z,n}$, hasta un máximo de $n = 50$, de modo que si en algún momento se obtiene $|\mathcal{J}_{c,z,n}| \geq 100$ se entenderá que no hay convergencia, interrumpiéndose en ese momento el cálculo de la sucesión (esto es importante, para evitar *overflows*). La función devolverá una matriz en la que el elemento asociado al punto z guarde el módulo del último término calculado de la sucesión. (Esta matriz se podrá representar gráficamente con el objeto `image` de `draw2d`.)

4.2. Nivel 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;
```

$$(\%o5) \qquad 2\,y + x$$

```
(%i6) [1,2];
```

$$(\%o6) \qquad [1, 2]$$

```
(%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`.

Prácticas

1. En la sección anterior se escribió en Maxima la función `conjulia` para generar conjuntos de Julia. Al tratarse de un algoritmo fundamentalmente de carácter numérico, se puede ganar en rapidez (y mucho) si se escribe directamente en Lisp aprovechando su capacidad para operar directamente con números complejos. Hágase. (Es necesario que el lector vea previamente la estructura interna de una matriz de Maxima siguiendo el método expuesto en esta sección.)