Maxima: una herramienta de cálculo

Universidad de Cádiz - Diciembre, 2006

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*, donde MAC, *Machine Aided Cognition*, era el nombre del *Laboratory for Computer Science* del MIT durante la fase inicial del proyecto Macsyma). Se desarrolló en estos laboratorios 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 realizaban los matemáticos, a fin de entender la capacidad de los ordenadores para actuar de forma inteligente.

El año 1982 es clave. El MIT entrega Macsyma al DOE y éste a la empresa Symbolics Inc. para su explotación comercial, haciendo el código propietario. Debido a la presión que la comunidad científica ejerció sobre el MIT, éste editó otra versión recodificada ajena a la comercial, dando así nacimiento al denominado DOE-Macsyma. Esta política del MIT con respecto a los proyectos basados en Lisp y Macsyma coincide con la salida de Richard Stallman de la institución y la posterior creación de la *Free Software Foundation*.

A partir de 1982, William Schelter traduce DOE-Macsyma a Common Lisp. Parte de la comunidad científica podrá utilizarlo, pero no tendrá 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 23 personas.

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 es a través de la lista de correo

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

Además, desde mediados de septiembre de 2006, se ha activado 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.

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 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 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. Entornos de ejecución

Como queda comentado al final de la sección anterior, una vez instalado Maxima, se dispone de dos entornos de ejecución, uno basado en texto, ejecutándose desde la línea de comandos, Figura 2.1, y otro basado en el entorno gráfico `tcl-tk`, Figura 2.2. Ambos son partes constituyentes del programa Maxima.

Otro entorno gráfico es Wxmaxima, Figura 2.3, que aún no siendo parte del proyecto Maxima, se distribuye conjuntamente con el ejecutable para Windows. En Linux, una vez instalado Maxima, se podrá instalar este entorno separadamente[2].

Una cuarta alternativa, que resulta muy vistosa por hacer uso de TEX, es la de ejecutar Maxima desde el editor Texmacs[3], Figura 2.4.

Además de los citados, en los últimos tiempos han aparecido varios proyectos de entornos basados en la web; se puede acceder a ellos a través del enlace *Related Projects* de la página web de Maxima.

2.3. 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. El código que se muestra está copiado y pegado del entorno de texto.

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

```
Maxima 5.9.3.1cvs http://maxima.sourceforge.net
Using Lisp CLISP ()
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

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);
                                - k x
(%o1)                     k %e
(%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*). El símbolo `%e` es la forma que tiene Maxima de representar a la base de los logaritmos naturales, de igual forma que `%pi` representa al número π. Nótese que tras la respuesta, la etiqueta `(%i2)` nos indica que Maxima espera una segunda instrucción.

Continuando con la sesión,

```
(%i2) %pi + %pi;
(%o2)                       2 %pi
(%i3) float(%);
```

[2] http://wxmaxima.sourceforge.net
[3] http://www.texmacs.org

Figura 2.1: Maxima desde la línea de comandos.

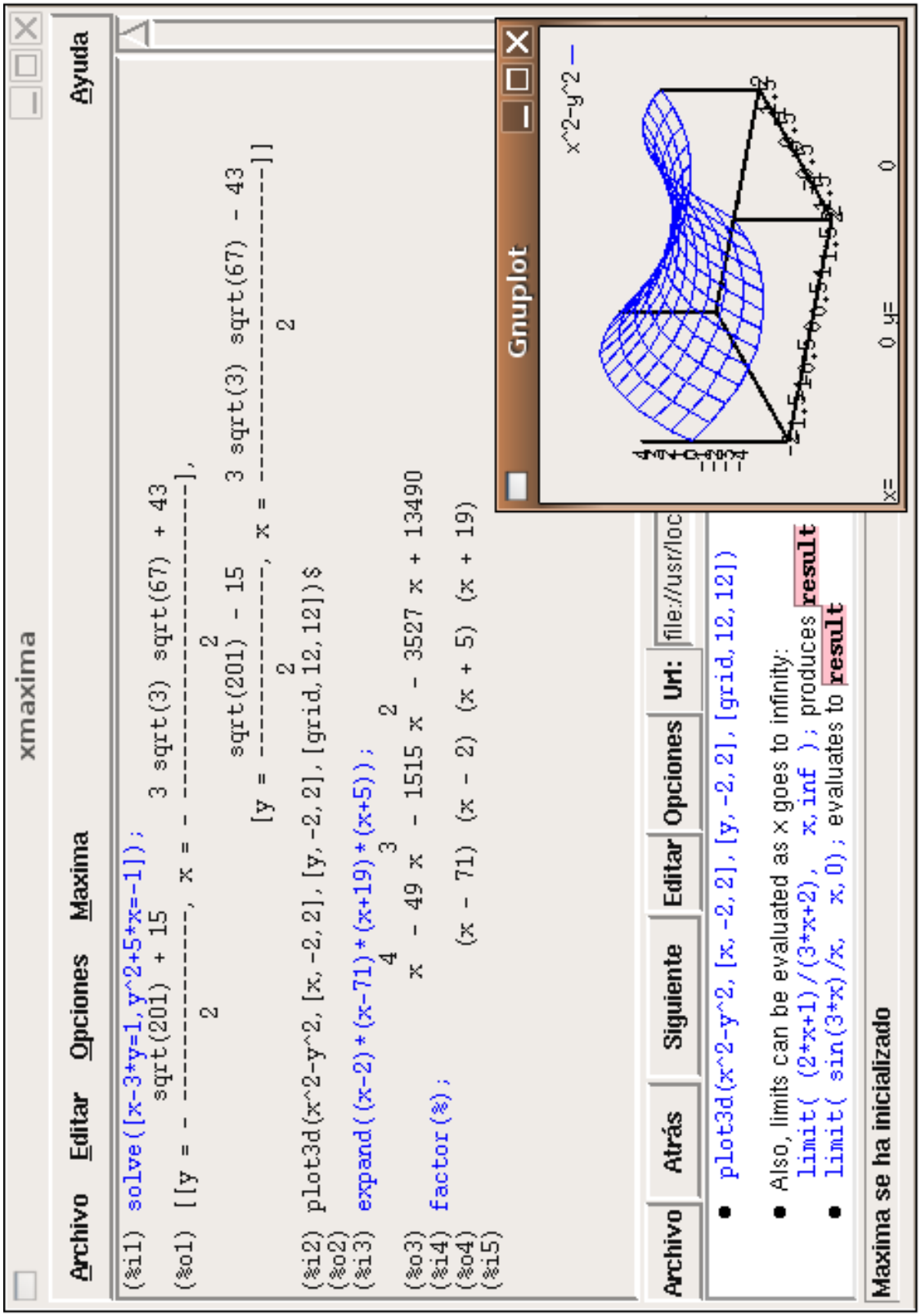


Figura 2.2: Xmaxima, el entorno gráfico de tcl-tk.

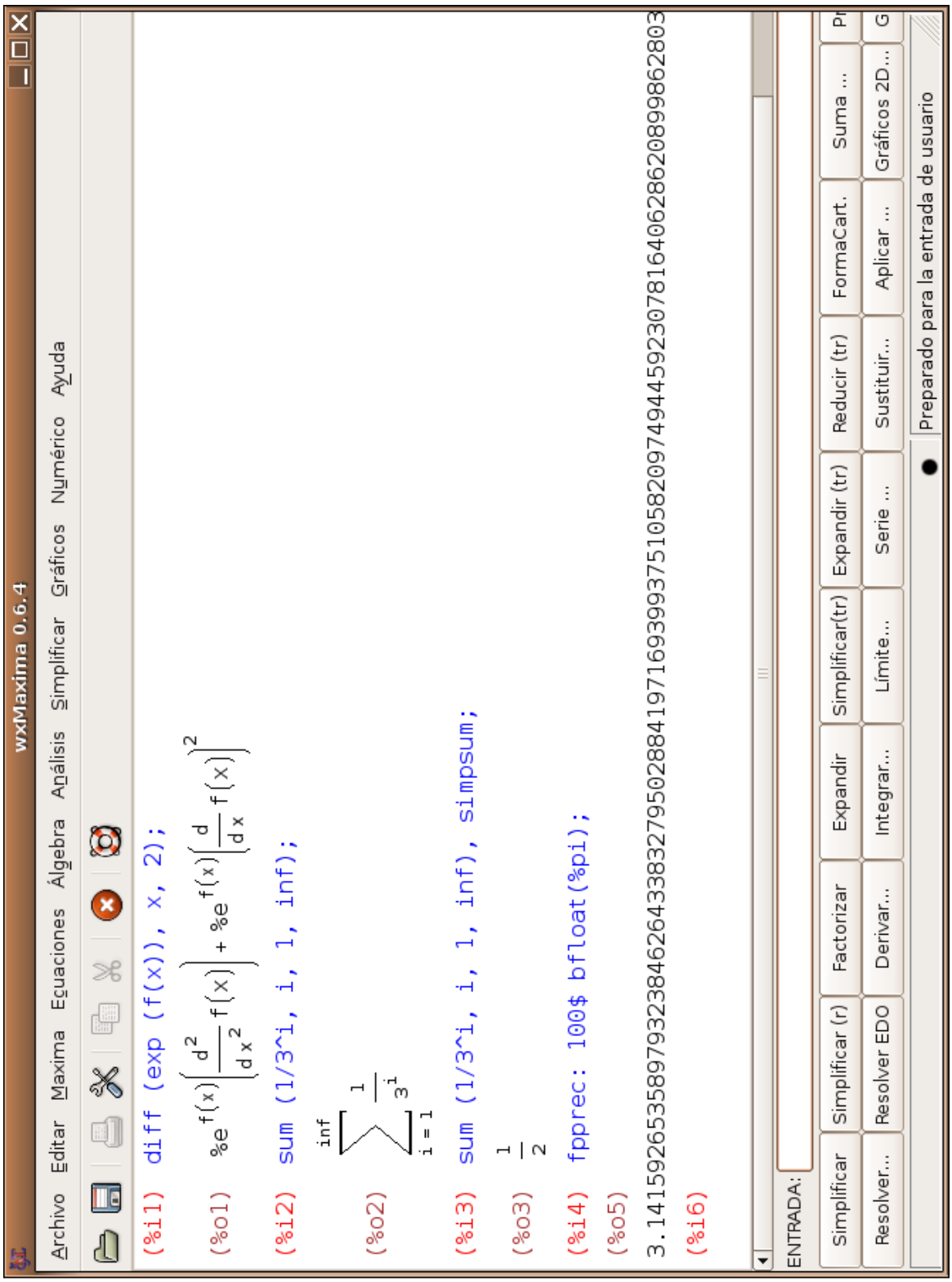


Figura 2.3: Wxmaxima, el entorno gráfico basado en wxwidgets.

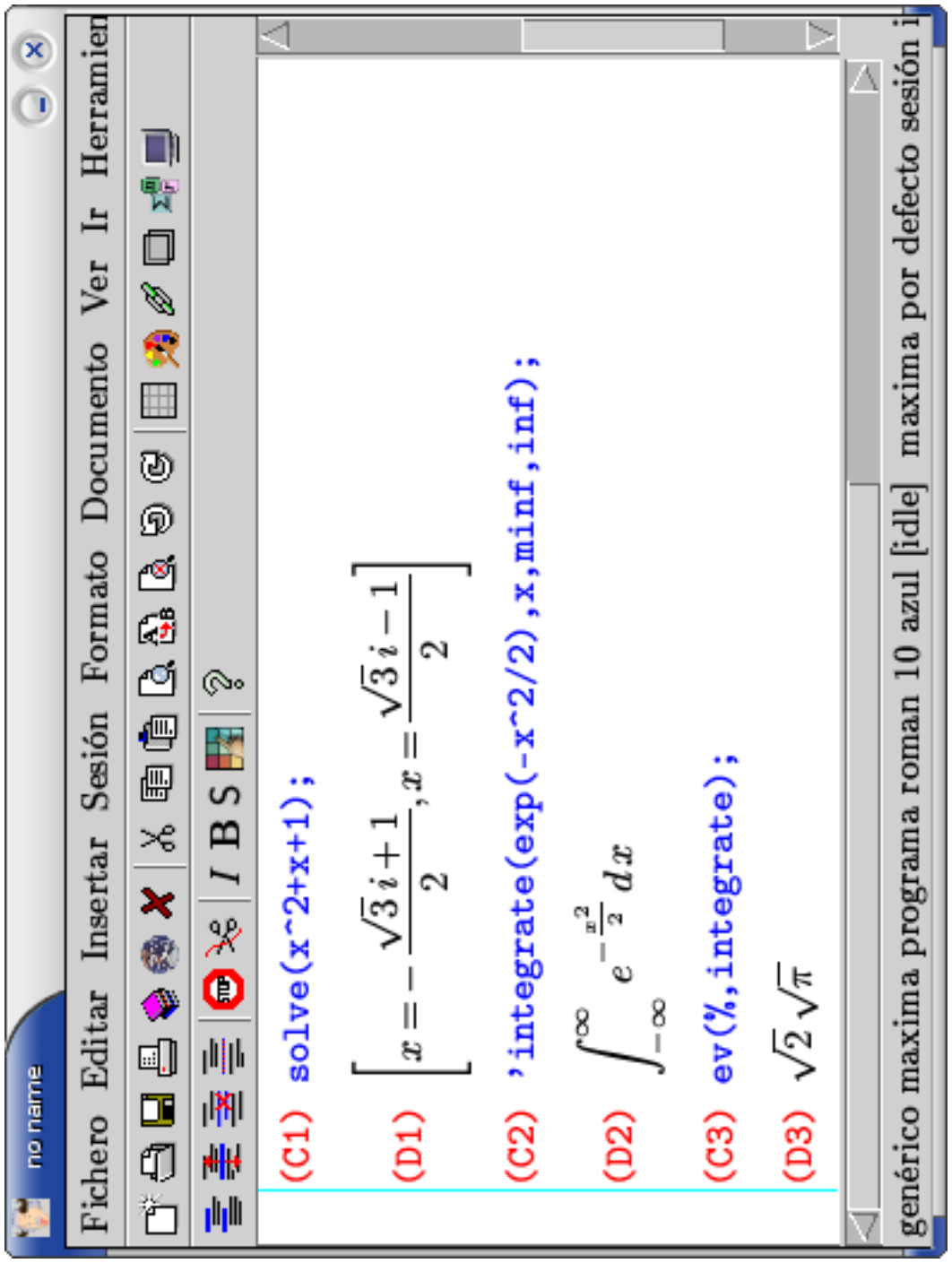


Figura 2.4: Maxima desde Texmacs.

```
(%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; en el siguiente diálogo, el texto escrito entre las marcas `/*` e `*/` 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)];
                                          3
                        1                 u
(%o4)             [- 1, -, [a, b], - 1, --]
                        4                 3
(%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);
                                       3
                           1           u
(%o6)                {- 1, -, [a, b], --}
                           4           3
```

De la misma manera que `%` representa al 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;
                          1                 8
(%o7)               [- 1, -, [c, b], - 1, -]
                          4                 3
(%i8) /* Forma alternativa para hacer lo mismo */
      xx, [u=2, a=c];
                          1                 8
(%o8)               [- 1, -, [c, b], - 1, -]
                          4                 3
(%i9) %o4[5], u=[1,2,3];
                           1  8
(%o9)                     [-, -, 9]
                           3  3
```

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 (`$`); en el primer caso, Maxima muestra el resultado del cálculo y en el segundo lo oculta. En la entrada `%i12` se optó por el `$` para evitar una línea adicional que no interesa ver:

```
(%i11) a: 2+2;
(%o11)                         4
(%i12) b: 6+6$
(%i13) 2*b;
(%o13)                         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, prefiriendo expresiones racionales:

```
(%i14) numer;
(%o14)                        false
(%i15) sqrt(8)/12;
                              - 1/2
                             2
(%o15)                       ------
                               3
(%i16) numer:true$
(%i17) sqrt(8)/12;
(%o17)                .2357022603955159
(%i18) numer:false$  /*se reinstaura valor por defecto */
```

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

```
(%i19) A: matrix([sin(x),2,%pi],[0,y^2,5/8]);
                    [ sin(x)  2   %pi ]
                    [                 ]
(%o19)              [          2   5  ]
                    [   0     y    -  ]
                    [              8  ]
(%i20) B: matrix([1,2],[0,2],[-5,integrate(x^2,x,0,1)]);
                          [  1   2 ]
                          [        ]
                          [  0   2 ]
(%o20)                    [        ]
                          [      1 ]
                          [ - 5  - ]
                          [      3 ]
(%i21) A.B;  /* producto matricial */
           [                                %pi     ]
           [ sin(x) - 5 %pi  2 sin(x) + --- + 4 ]
           [                                 3      ]
(%o21)     [                                       ]
           [        25              2   5          ]
```

```
          [      - --          2 y  + --      ]
          [        8                 24      ]
```

En cuanto a gráficos, Maxima hace uso por defecto del programa externo `gnuplot`. Las siguientes instrucciones generan diferentes tipos de gráficos, cuyas reproducciones[4] se pueden ver en la Figura 2.5, ordenadas de izquierda a derecha y de arriba hacia abajo.

```
(%i22) plot2d(exp(-x^2),[x,-2,5])$
(%i23) plot2d([-x^2,1+x,7*sin(x)],[x,-2,5])$
(%i24) plot2d([parametric,t,t*sin(1/t),[t,0.01,0.2]],[nticks,500])$
(%i25) plot3d(exp(-x^2-y^2),[x,-2,2],[y,-2,0])$
(%i26) plot3d([cos(t),sin(t),2*t],[t,-%pi,%pi],[u,0,1])$
(%i27) plot3d([cos(x)*(3+y*cos(x/2)),
             sin(x)*(3+y*cos(x/2)),y*sin(x/2)],
             [x,-%pi,%pi],[y,-1,1],['grid,50,15])$
(%i28) /* Un par de ejemplos de estadística descriptiva */
       load(descriptive)$ load (numericalio)$
(%i30) s2 : read_matrix (file_search ("wind.data"))$
(%i31) boxplot(s2,outputdev="eps")$
(%i32) dataplot(s2,outputdev="eps")$
```

Gnuplot es un programa gráfico muy potente; la forma que se tiene de aprovechar sus capacidades desde Maxima consiste en escribir dentro del apartado `gnuplot_preamble` (ver ejemplo a pie de página) todas las directrices que el usuario considere oportunas; sin embargo, tales directrices se deben hacer en el lenguaje de `scripts` de gnuplot, lo que puede llegar a ser embarazoso para el usuario medio. Maxima utiliza también el entorno `tcl-tk`, que permite más libertad a la hora de programar gráficos; aunque esta alternativa estuvo en hibernación en los últimos tiempos, está recobrando nuevos bríos en la actualidad. Ninguna de estas dos soluciones es completamente satisfactoria en su estado actual, pero está claro que antes o después habrá que tomar algunas decisiones sobre cómo generar todo tipo de gráficos desde Maxima.

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:

```
(%i36) 'integrate(sqrt(a+x)/x^5,x,1,2) = integrate(sqrt(2+x)/x^5,x,1,2);
        2
       /
       [  sqrt(x + 2)
(%o36) I  ----------- dx =
       ]       5
       /      x
        1
5 sqrt(2) log(5 - 2 sqrt(2) sqrt(3)) + 548 sqrt(3)
-------------------------------------------------
                      2048
```

[4] Reconozco que aquí hay algo de trampa. Los gráficos de la Figura 2.5 fueron creados con gnuplot en formato eps, para lo cual hubo que añadir cierto código a las sentencias que se describen; en particular, el primer fichero gráfico se generaría con el código

plot2d(exp(-x**2),[x,-2,5],[gnuplot_preamble,"set terminal postscript eps;set out 'grafico1.eps'"])

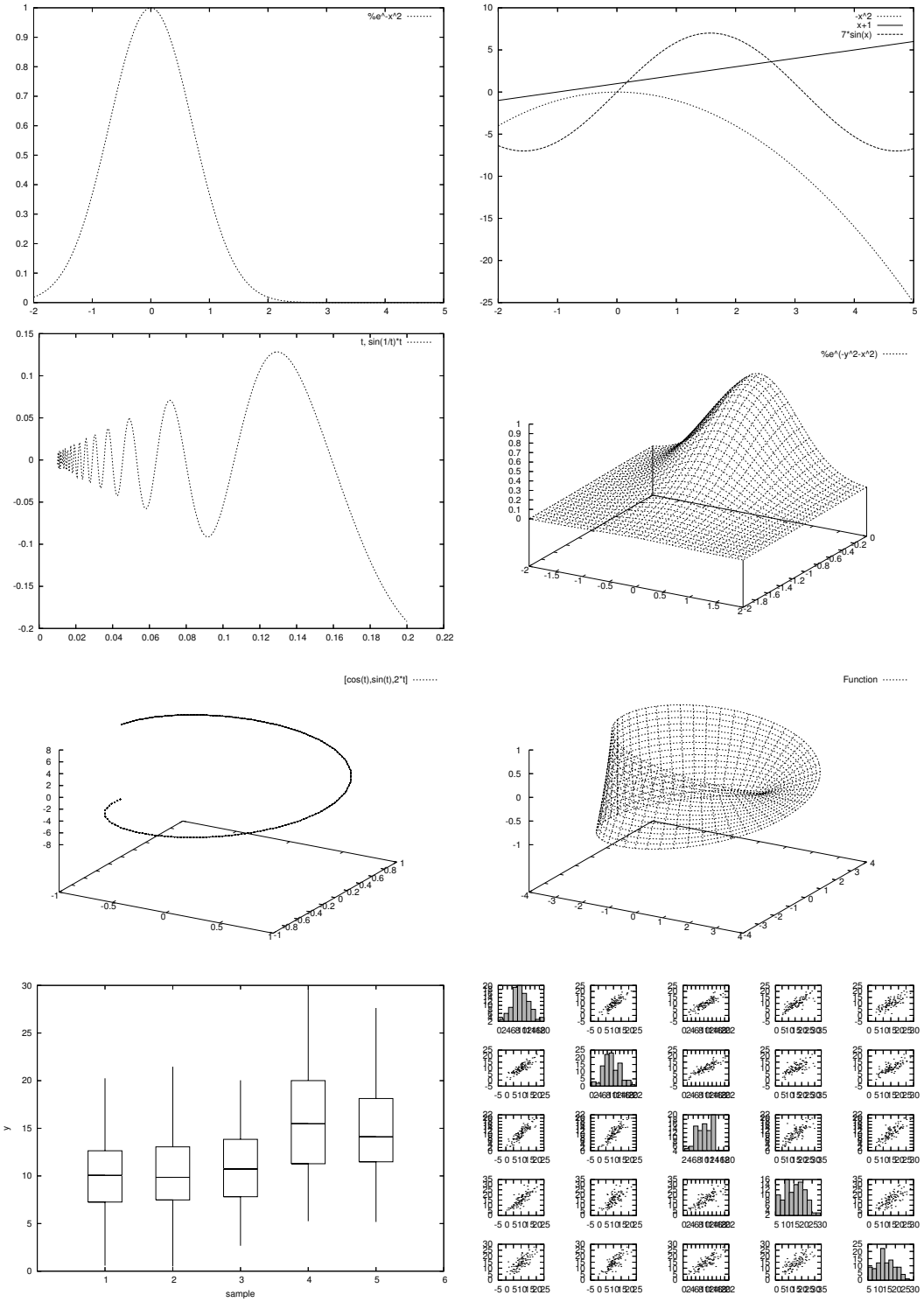


Figura 2.5: Ejemplos de la interacción entre Maxima y gnuplot.

```
   15 sqrt(2) log(3 - 2 sqrt(2)) + 244
 - ----------------------------------
                  6144
(%i37) tex(%);
$$\int_{1}^{2}{{{\sqrt{x+2}}\over{x^5}}\;dx}={{5\,\sqrt{2}\,\log
 \left(5-2\,\sqrt{2}\,\sqrt{3}\right)+548\,\sqrt{3}}\over{2048}}-{{15
 \,\sqrt{2}\,\log \left(3-2\,\sqrt{2}\right)+244}\over{6144}}$$
```

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

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

Un comentario sobre la entrada `%i36`. Se observa en ella que se ha escrito dos veces el mismo código a ambos lados del signo de igualdad. La única diferencia entre ambos es el apóstrofo que antecede al primer `integrate`; éste es un operador (llamado de *comilla simple*) que Maxima utiliza para evitar la ejecución de una instrucción, de forma que en el resultado la primera integral no se calcula, pero sí la segunda, dando lugar a la igualdad que se obtiene como resultado final.

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 (tales expresiones reciben en Maxima el apelativo de *nominales*); el siguiente ejemplo muestra el uso del símbolo especial `nouns`,

```
(%i38) z: 'diff(tan(x),x);
                            d
(%o38)                      -- (tan(x))
                            dx
(%i39) z+z;
                             d
(%o39)                    2 (-- (tan(x)))
                             dx
(%i40) z, nouns;
                                 2
(%o40)                         sec (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 `%o40`), podrá hacer uso de la función `save`:

```
(%i41) save("sesion", z, vz=%o40);
(%o41)                         sesion
(%i42) quit();
```

Obsérvese que en primer lugar se escribe el nombre del fichero y a continuación los nombres de las variables a almacenar; como el valor 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;
                            d
```

```
(%o2)                        -- (tan(x))
                             dx
(%i3) vz;
                                  2
(%o3)                          sec (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) ? diff

 0: (maxima.info)impdiff.
 1: Definiciones para impdiff.
 2: antidiff :Definiciones para Diferenciación.
 3: AntiDifference :Definiciones para zeilberger.
 4: covdiff :Definiciones para itensor.
 5: diff <1> :Definiciones para itensor.
 6: diff :Definiciones para Diferenciación.
 7: evundiff :Definiciones para itensor.
 8: extdiff :Definiciones para itensor.
 9: idiff :Definiciones para itensor.
 10: liediff :Definiciones para itensor.
 11: poisdiff :Definiciones para las Funciones Especiales.
 12: ratdiff :Definiciones para Polinomios.
 13: rediff :Definiciones para itensor.
 14: setdifference :Definiciones para los conjuntos.
 15: symmdifference :Definiciones para los conjuntos.
 16: undiff :Definiciones para itensor.
Enter space-separated numbers, 'all' or 'none': 5

 -- Función: diff (<expr>, <v_1>, [<n_1>, [<v_2>, <n_2>] ...])
     Se trata de la función de Maxima para la diferenciación, ampliada
     para las necesidades del paquete 'itensor'. Calcula la derivada de
     <expr> respecto de <v_1> <n_1> veces, respecto de <v_2> <n_2>
     veces, etc. Para el paquete de tensores,la función ha sido
     modificada de manera que <v_i> puedan ser enteros desde 1 hasta el
     valor que tome la variable 'dim'. Esto permite que la derivación
     se pueda realizar con respecto del <v_i>-ésimo miembro de la lista
     'vect_coords'. Si 'vect_coords' guarda una variable atómica,
     entonces esa variable será la que se utilice en la derivación. Con
     esto se hace posible la utilización de una lista con nombres de
     coordenadas subindicadas, como 'x[1]', 'x[2]', ...
```

En primer lugar, Maxima nos muestra una lista con todas las funciones y variables que contienen la cadena `diff`; una vez seleccionada la que nos interesa (5), se lee la información correspondiente. Algunos usuarios encuentran engorroso que la ayuda se interfiera con los cálculos; esto se puede arreglar con un mínimo de bricolage informático. Al tiempo que se instala Maxima, se almacena también el sistema de ayuda en formato `html` en una determinada carpeta; el usuario tan sólo tendrá que buscar esta carpeta, en la que se encontrará la página principal `maxima.html`, y hacer uso de la función `system` de Maxima que ejecuta comandos del sistema, la cual admite como

argumento una cadena alfanumérica con el nombre del navegador (en el ejemplo, `mozilla`) seguido de la ruta hacia el documento `maxima.html`,

```
(%i5) system("mozilla /usr/local/share/maxima/5.9.3/doc/html/es/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. Así, cada vez que arranque Maxima, se abrirá el navegador con el manual de referencia del programa.

Capítulo 3

Matemáticas con Maxima

3.1. Números

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

```
(%i1) 200!;
(%o1) 788657867364790503552363213932185062295135977687173263294742\
53324435944996340334292030428401198462390417721213891963883025764 2\
79024263710506192662495282993111346285727076331723739698894392244 5\
62145166424025403329186413122742829485327752424240757390324032125 7\
40557956866022603190417032406235170085879617892222278962370389737 4\
72000000000000000000000000000000000000000000000000
(%i2) factor(%);   /* factoriza el resultado anterior */
        197  97  49  32   19   16   11   10   8   6   6   5   4   4
(%o2)  2    3   5   7   11   13   17   19   23  29  31  37  41  43
   4   3   3   3   2   2   2   2   2   2   2
 47  53  59  61  67  71  73  79  83  89  97  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 ;
                              3200000
(%o3)                      - ----------
                             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)                - 5.127940964462986E-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`,

```
(%i5) fpprec: 200$  bfloat(zz);
(%o6) - 5.12794096446298596293256880814488029168034600338090820571\
73660171436575567523596790430788338566951813264851634643780788785 5\
```

```
230937205128922131867273899509251418529208712713155391584330100241\
17746366396b-4
```

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

```
(%i7) z1: 3+4*%i;
(%o7)                          4 %i + 3
(%i8) z2: exp(7*%pi/8*%i);
                                 7 %i %pi
                                 --------
                                    8
(%o8)                           %e
(%i9) z1+z2;
                            7 %i %pi
                            --------
                               8
(%o9)                     %e          + 4 %i + 3
(%i10) rectform(%);   /* forma cartesiana */
                           7 %pi              7 %pi
(%o10)           %i (sin(-----) + 4) + cos(-----) + 3
                           8                  8
(%i11) polarform(%); /* forma polar */
                  7 %pi      2        7 %pi      2
(%o11) sqrt((sin(-----) + 4)  + (cos(-----) + 3) )
                    8                   8
                                                        7 %pi
                                                  sin(-----) + 4
                                                        8
                                             %i atan(--------------)
                                                        7 %pi
                                                  cos(-----) + 3
                                                        8
                                           %e
(%i12) abs(%);  /* módulo */
                2 7 %pi          7 %pi       2 7 %pi          7 %pi
(%o12) sqrt(sin (-----) + 8 sin(-----) + cos (-----) + 6 cos(-----)
                    8              8             8              8
                                                                + 25)
(%i13) float(%);  /* módulo en formato decimal */
(%o13)                   4.84955567695155
```

3.2. Resolución de ecuaciones

La función `algsys` resuelve sistemas de ecuaciones algebraicas. Admite como argumentos dos listas, en la primera se escribe la ecuación o ecuaciones del sistema, en las que 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:

Ecuación polinómica de tercer grado

$$4x^3 - 2x^2 + 5x - 7 = 0$$

```
(%i1) algsys([4*x^3-2*x^2+5*x -7=0],[x]);
           3 sqrt(3) %i - 1          3 sqrt(3) %i + 1
(%o1) [[x = ----------------], [x = - ----------------], [x = 1]]
                  4                         4
```

Se piden las soluciones no complejas de la ecuación anterior

```
(%i2) realonly: true$
(%i3) algsys([4*x^3-2*x^2+5*x -7],[x]);
(%o3)                        [[x = 1]]
(%i4) realonly: false$  /* se restaura su valor por defecto */
```

Sistema no lineal

$$\begin{cases} 3*x^2-y^2 &= 6 \\ x &= y+9 \end{cases}$$

```
(%i5) algsys([3*x^2-y^2=6,x=y+9],[x,y]);
                sqrt(255) + 9
(%o5)  [[x = - -------------, y =
                      2
  sqrt(3) sqrt(5) sqrt(17) + 27
- -----------------------------],
                 2
    sqrt(255) - 9      sqrt(3) sqrt(5) sqrt(17) - 27
[x = -------------, y = -----------------------------]]
          2                          2
```

Un sistema no lineal con coeficientes paramétricos y complejos

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

```
(%i6) algsys([3*u-a*v=t,(2+%i)/(u+t)=3*v+u,t/u=1],[u,v,t]);
                  %i a     2 a
              sqrt(----- + -----)
                   a + 6   a + 6             2 sqrt(%i + 2)
(%o6)  [[u = --------------------, v = --------------------------,
                    sqrt(2)            sqrt(2) sqrt(a) sqrt(a + 6)
                                        %i a     2 a
                                     sqrt(----- + -----)
    sqrt(%i + 2) sqrt(a)                 a + 6   a + 6
t = --------------------], [u = - --------------------,
    sqrt(2) sqrt(a + 6)                  sqrt(2)
            2 sqrt(%i + 2)                sqrt(%i + 2) sqrt(a)
v = - --------------------------, t = - --------------------]]
      sqrt(2) sqrt(a) sqrt(a + 6)        sqrt(2) sqrt(a + 6)
```

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

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

```
(%i7) algsys([x^5+x^4+x^3+x^2+x],[x]);
(%o7) [[x = 0], [x = - .5877852522924731 %i - .8090169943749475],
[x = .5877852522924731 %i - .8090169943749475],
[x = .3090169943749475 - .9510565162951535 %i],
[x = .9510565162951535 %i + .3090169943749475]]
```

Más allá de las ecuaciones algebraicas, Maxima no dispone de algoritmos de resolución simbólica, aunque sí existe un módulo para la aplicación del método de Newton.

Resolución de la ecuación

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

```
(%i8) load("mnewton")$  /* carga el paquete correspondiente */
(%i9) mnewton([2*u^u-5],[u],[1]);
(%o9)                [[u = 1.70927556786144]]
```

Y del sistema

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

```
(%i10) mnewton([x+3*log(x)-y^2, 2*x^2-x*y-5*x+1],[x, y], [5, 5]);
(%o10)   [[x = 3.756834008012769, y = 2.779849592817897]]
(%i11) mnewton([x+3*log(x)-y^2, 2*x^2-x*y-5*x+1],[x, y], [1, -2]);
(%o11)  [[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.

3.3. Patrones y reglas

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

Antes de abordar este tema, convendrá reparar en dos funciones que suelen ser de utilidad en este contexto y que se relacionan con las listas; la función `apply`, que se utiliza para pasar como argumentos de una función a todos los elementos de una lista, y la función `map`, que se utiliza para aplicar una función a cada uno de los elementos de una lista,

```
(%i1) apply(min,[7,3,4,9,2,4,6]);
(%o1)                          2
(%i2) map(cos,[7,3,4,9,2,4,6]);
(%o2) [cos(7), cos(3), cos(4), cos(9), cos(2), cos(4), cos(6)]
```

Obsérvese que la función a aplicar en `apply` debe ser tal que admita múltiples argumentos; por ejemplo, la función suma `"+"` podría haberse utilizado sin problemas, no así `"-"`, que sólo admite un máximo de dos argumentos. Sin embargo, las funciones a utilizar por `map` deben ser de un único argumento.

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,

```
(%i3) map(lambda([x], sin(x) + 1/2),[7,3,4,9]);
                  1          1          1          1
(%o3)    [sin(7) + -, sin(3) + -, sin(4) + -, sin(9) + -]
                  2          2          2          2
```

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.

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

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

```
(%i4) matchdeclare ([x,y], true);
(%o4)                         done
```

En este caso, las variables patrón serán `x` e `y`, siendo el segundo argumento una función de predicado[1] que devolverá `true` si el patrón se cumple; en el caso presente, el patrón es universal y admite cualquier formato para estas variables. Si quisiésemos que la regla se aplicase sólo a números enteros, se debería escribir `matchdeclare ([x,y], integerp)`; si quisiésemos que la regla se aplicase siempre que `x` sea un número, sin imponer restricciones a `y`, escribiríamos `matchdeclare (x, numberp, y, true)`. Como se ve, los argumentos impares son variables o listas de variables y los pares las condiciones de los patrones.

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

```
(%i5) defrule (regla1, G(x,y), H(x,y)/x);
                                        H(x, y)
(%o5)               regla1 : G(x, y) -> -------
                                          x
```

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

```
(%i6) apply1(G(f,2+k),regla1);
                           H(f, k + 2)
(%o6)                      -----------
                                f
(%i7) apply1(G(G(4,6),2+k),regla1);
                           H(4, 6)
                      4 H(-------, k + 2)
                             4
(%o7)                 --------------------
                            H(4, 6)
```

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

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

```
(%i8) maxapplydepth;
(%o8)                          10000
(%i9) maxapplydepth:1;
(%o9)                            1
(%i10) apply1(G(G(4,6),2+k),regla1);
                         H(G(4, 6), k + 2)
(%o10)                   -----------------
                               G(4, 6)
```

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

```
(%i11) maxapplyheight:1;
(%o11)                           1
(%i12) applyb1(G(G(4,6),2+k),regla1);
                           H(4, 6)
(%o12)                   G(-------, k + 2)
                             4
```

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

```
(%i13) H(u,v):= u^v+1;
                                     v
(%o13)                   H(u, v) := u  + 1
(%i14) applyb1(G(G(4,6),2+k),regla1);
                           4097
(%o14)                   G(----, k + 2)
                             4
```

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

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

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

```
(%i15) esunasuma(expr):= not atom(expr) and op(expr)="+" $
(%i16) matchdeclare(z,esunasuma)$
```

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

```
(%i17) defrule(regla2, F(z), apply("+",map(F, args(z))) + apply("*", args(z)));
(%o17) regla2 : F(z) -> apply(+, map(F, args(z))) + apply(*, args(z))
```

Veamos unos cuantos resultados de la aplicación de esta regla,

```
(%i18) apply1(F(a+b), regla2);
(%o18)                   F(b) + a b + F(a)
(%i19) apply1(F(a+b), regla2) - apply1(F(a+c), regla2) ;
(%o19)                - F(c) - a c + F(b) + a b
(%i20) apply1(F(a+b+c), regla2);
(%o20)              F(c) + a b c + F(b) + F(a)
(%i21) apply1(F(4), regla2);
(%o21)                          F(4)
(%i22) apply1(F(4+5), regla2);
(%o22)                          F(9)
(%i23) simp:false$  /* inhibe la simplificación de Maxima */
(%i24) apply1(F(4+5), regla2);
(%o24)                   4 5 + (F(4) + F(5))
(%i25) simp:true$   /* restaura la simplificación de Maxima */
(%i26) %o24;
(%o26)                   F(5) + F(4) + 20
```

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

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

```
(%i27) infix("o");
(%o27)                           o
(%i28) is(a o b = b o a);
(%o28)                         false
(%i29) declare("o", commutative);
(%o29)                         done
(%i30) is(a o b = b o a);
(%o30)                         true
```

3.4. 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 $-\infty$.

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)                        inf
(%i5) limit(1/(x-1),x,1,minus);
(%o5)                        minf
```

El símbolo `und` hace referencia al término ingés *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 */
                log(a x)  log(a x)   log(x)
(%o6)          x         (-------- + ------)
                                x         x
(%i7) diff(x^log(a*x),x,2); /* derivada segunda */
       log(a x)  log(a x)   log(x) 2
(%o7) x         (-------- + ------)
                    x         x
                         log(a x)    log(a x)   log(x)   2
                   + x         (- -------- - ------ + --)
                                      2         2      2
                                     x         x      x
(%i8) factor(%); /* ayudamos a Maxima a mejorar el resultado */
       log(a x) - 2     2
(%o8) x             (log (a x) + 2 log(x) log(a x)
                                          2
                        - log(a x) + log (x) - log(x) + 2)
```

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);
                   x           2
(%o9)           2 %e  cos(y) sec (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(x, y), x(u, v), y(u, v)]
(%i11) diff(z,v,1);
                      dy dz   dx dz
(%o11)                -- -- + -- --
                      dv dy   dv dx
(%i12) diff(z,y,1,v,1);
                        2         2
                    dy d z    dx  d z
```

```
(%o12)                -- --- + -- -----
                      dv   2   dv dx dy
                         dy
(%i13) diff(z,u,1,v,1);
               2          2         2
       dy   dy d z    dx  d z      d y   dz
(%o13) -- (-- --- + -- -----) + ----- --
       du   dv   2    dv dx dy    du dv dy
              dy
                                    2          2         2
                              dx   dx d z    dy  d z      d x   dz
                            + -- (-- --- + -- -----) + ----- --
                              du   dv   2    dv dx dy    du dv dx
                                     dx
```

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

```
(%i14) dependencies;
(%o14)          [z(x, y), x(u, v), y(u, v)]
```

También podemos eliminar dependencias,

```
(%i15) remove(x,dependency);
(%o15)                       done
(%i16) dependencies;
(%o16)               [z(x, y), y(u, v)]
(%i17) diff(z,y,1,v,1);
                            2
                          dy d z
(%o17)                    -- ---
                          dv   2
                             dy
```

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

```
(%i18) depends(y,x)$
(%i19) diff(x^2+y^3=2*x*y,x);
                  2 dy             dy
(%o19)          3 y  -- + 2 x = 2 x -- + 2 y
                    dx             dx
```

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

```
(%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);
         2              2 dy
```

```
(%o21) 3 y  del(y) + (3 y  -- + 2 x) del(x) =
                         dx
                                              dy
                       2 x del(y) + (2 x -- + 2 y) del(x)
                                              dx
```

Recuérdese que durante este cálculo 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);
                          2          3
              1   (x - 1)    (x - 1)
(%o22)/T/     - - -------- + -------- + . . .
              2      12         12
(%i23) expand(%);
                      3    2
                     x    x    5 x   1
(%o23)               -- - -- + --- + -
                     12   3    12    3
```

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

```
(%i24) taylor(exp(x^2*sin(x*y)),[x,2,2],[y,0,2]);
                       2
(%o24)/T/ 1 + 8 y + 32 y  + . . .
              2
 + (12 y + 96 y  + . . .) (x - 2)
               2                  2
 + (6 y + 120 y  + . . .) (x - 2)  + . . .
(%i25) expand(%);
           2  2          2        2      2
(%o25) 120 x  y  - 384 x y  + 320 y  + 6 x  y - 12 x y + 8 y + 1
```

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

```
(%i26) integrate(cos(x)^3/sin(x)^4,x);
                             2
                          3 sin (x) - 1
(%o26)                    -------------
                                 3
                           3 sin (x)
(%i27) integrate(cos(x)^3/sin(x)^4,x,0,1);
Integral is divergent
 -- an error.  Quitting.  To debug this try debugmode(true);
(%i28) integrate(cos(x)^3/sin(x)^4,x,1,2);
               1          1         1          1
```

```
(%o28)        ------ - --------- - ------ + ---------
              sin(2)         3      sin(1)         3
                       3 sin (2)             3 sin (1)
(%i29) integrate(cos(x)^3/sin(x)^4,x,a,2);
Is  a - 2  positive, negative, or zero?

p;
                 1           1          1           1
(%o29)        - ------ + --------- + ------ - ---------
                sin(a)         3      sin(2)         3
                        3 sin (a)             3 sin (2)
```

La primera integral indefinida no requiere explicación; la segunda da un error debido a la singularidad en $x = 0$; la tercera es una integral definida y la última, siendo también definida, nos pide información sobre el parámetro 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,

```
(%i30) /* El integrador simbólico no puede con esta integral */
       integrate(exp(sin(x)),x,2,7);
                              7
                             /
                             [    sin(x)
(%o30)                       I  %e       dx
                             ]
                             /
                              2
(%i31) /* Resolvemos numéricamente */
       quad_qag(exp(sin(x)),x,2,7,3);
(%o31)   [4.747336298073747, 5.27060206376023E-14, 31, 0]
```

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

```
(%i32) n:4$
(%i33) integrate(x^(n/2-1)*exp(-y/2)/2^(n/2)*gamma(n/2),x,1,inf);
Integral is divergent
 -- an error.  Quitting.  To debug this try debugmode(true);
(%i34) quad_qagi(x^(n/2-1)*exp(-x/2)/2^(n/2)*gamma(n/2),x,1,inf);
(%o34)  [.9097959895689501, 1.913452127046495E-10, 165, 0]
(%i35) load(distrib)$
(%i36) 1 - cdf_chi2(1,n),numer;
```

```
(%o36)                    .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 (ver Sección 3.7).

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.

3.5. 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);
                4 2    2          2          2          2
(%o7)  sqrt((e - -)  + d  + (c + 6)  + (b - 4)  + (a - 3) )
                5
```

Los operadores diferenciales que son de uso común en el ámbito de los campos vectoriales están programados en el paquete `vect`, lo que implica que debe ser cargado en memoria antes de ser utilizado. Sigue a continuación una sesión de ejemplo sobre cómo usarlo.

Partamos de los campos escalares $\phi(x,y,z) = -x + y^2 + z^2$ y $\psi(x,y,z) = 4x + \log(y^2 + z^2)$ y demostremos que sus 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) /* Calculamos el producto escalar de los gradientes */
       grad(phi) . grad(psi);
                2    2                    2    2
(%o11)    grad (z  + y  - x) . grad (log(z  + y ) + 4 x)
```

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(%);
       d    2    2        d        2    2
(%o12) (-- (z  + y  - x)) (-- (log(z  + y ) + 4 x))
       dz                  dz
   d    2    2        d        2    2
 + (-- (z  + y  - x)) (-- (log(z  + y ) + 4 x))
   dy                  dy
   d    2    2        d        2    2
 + (-- (z  + y  - x)) (-- (log(z  + y ) + 4 x))
   dx                  dx
(%i13) ev(%,diff);
                           2         2
                        4 z       4 y
(%o13)                  ------- + ------- - 4
                         2    2    2    2
                        z  + y    z  + y
(%i7) ratsimp(%);
(%o7)                             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 $(\nabla^2\mathbf{F})$

```
(%i8) F: [x*y,x^2*z,exp(x+y)]$
(%i9) div (F);  /* divergencia */
                                2      y + x
(%o9)                 div [x y, x  z, %e     ]
(%i10) express (%);
               d    2      d     y + x    d
(%o10)         -- (x  z) + -- (%e     ) + -- (x y)
               dy          dz             dx
(%i11) ev (%, diff);
(%o11)                           y
(%i12) curl (F);  /* rotacional */
                                 2      y + x
(%o12)                curl [x y, x  z, %e     ]
(%i13) express (%);
       d     y + x    d    2     d          d     y + x
(%o13) [-- (%e     ) - -- (x  z), -- (x y) - -- (%e     ),
       dy             dz         dz         dx
                                             d    2      d
                                             -- (x  z) - -- (x y)]
                                             dx          dy
(%i14) ev (%, diff);
                  y + x    2      y + x
(%o14)         [%e      - x , - %e     , 2 x z - x]
(%i15) laplacian (F);  /* laplaciano */
```

```
                                    2      y + x
(%o15)            laplacian [x y, x  z, %e     ]
(%i16) express (%);
       2                              2
      d           2      y + x      d           2      y + x
(%o16) --- ([x y, x  z, %e     ]) + --- ([x y, x  z, %e     ])
        2                              2
       dz                             dy
                                            2
                                           d           2      y + x
                                         + --- ([x y, x  z, %e     ])
                                             2
                                           dx
(%i17) ev (%, diff);
                                     y + x
(%o17)                  [0, 2 z, 2 %e     ]
```

Nótese en todos los casos el usos de la secuencia `express - ev`. Por último, el paquete `vect` incluye también la definición del producto vectorial, al cual le asigna el operador `~`,

```
(%i18) [a, b, c] ~ [x, y, z];
(%o18)                 [a, b, c] ~ [x, y, z]
(%i19) express(%);
(%o19)           [b z - c y, c x - a z, a y - b x]
```

3.6. 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;
               3         dy            3
(%o1)         x  (y - 1) -- + (x - 1) y = 0
                         dx
```

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);
                  2 y - 1        2 x - 1
(%o2)             ------- = %c - -------
                     2              2
                   2 y            2 x
```

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

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

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

```
(%i4) /* ecuacion homogénea */
      ode2(x^3+y^3+3*x*y^2*'diff(y,x),y,x);
                           3    4
                       4 x y  + x
(%o4)                  ----------- = %c
                            4
```

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

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

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

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

```
(%i9) 'diff(y,x)=x+'diff(y,x,2);
                             2
                        dy   d y
(%o9)                   -- = --- + x
                        dx     2
                             dx
(%i10) ode2(%,y,x);
                              2
                         x   x  + 2 x + 2
(%o10)         y = %k1 %e  + ------------ + %k2
                                  2
```

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

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

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

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

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

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

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

```
(%i13) desolve(['diff(f(x),x)=3*f(x)-2*g(x),
                'diff(g(x),x)=2*f(x)-2*g(x)],
               [f(x),g(x)]);
                                  - x
                (2 g(0) - f(0)) %e
(%o13) [f(x) = ---------------------
                          3
                        2 x
   (2 g(0) - 4 f(0)) %e
 - ----------------------, g(x) =
              3
                    - x                       2 x
(4 g(0) - 2 f(0)) %e      (g(0) - 2 f(0)) %e
---------------------- - ---------------------]
          3                         3
```

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

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

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

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

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

Además de las funciones anteriores, el paquete `plotdf` permite generar campos de direcciones, bien de ecuaciones diferenciales de primer orden

$$\frac{dy}{dx} = F(x,y),$$

bien de sistemas

$$\begin{cases} \frac{dx}{dt} = G(x,y) \\ \frac{dy}{dt} = F(x,y) \end{cases}$$

Los argumentos a pasar a la función `plotdf` son la función F, en el primer caso, y una lista con las funciones F y G en el segundo. Las variables serán siempre x e y. Como ejemplo, pidamos a Maxima que genere el campo de direcciones de la ecuación diferencial $\frac{dy}{dx} = 1 + y + y^2$

```
(%i18) load(plotdf)$
(%i19) plotdf(1 + y + y^2);
```

El gráfico que resulta es el de la Figura 3.1 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.

```
(%i20) plotdf([2*x+k*x*y, -y+h*x*y],
              [parameters,"k=-1.2,h=0.9"],
              [sliders,"k=-2:2,h=-2:2"]);
```

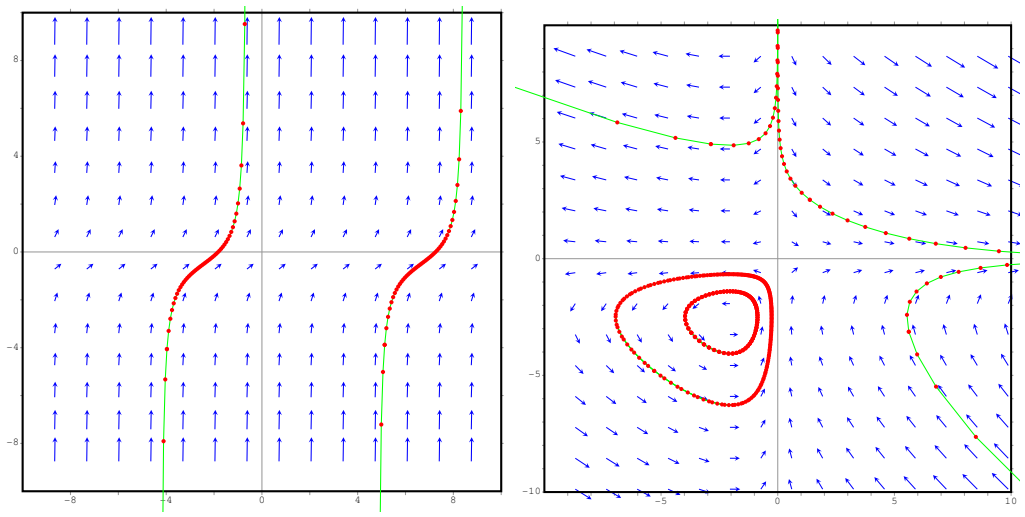


Figura 3.1: Campos de direcciones creados con la función 'plotdf': a la izquierda el campo de la ecuación $\frac{dy}{dx} = 1 + y + y^2$, a la derecha el correspondiente al modelo predador-presa.

En la Figura 3.1 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 la función `ode2` no es capaz de resolver la ecuación propuesta, se podrá recurrir al método numérico de Runge-Kutta, el cual se encuentra programado en el paquete `diffeq`. Como primer ejemplo, nos planteamos la resolución de la ecuación

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

con la condición $y(0) = 1$. La función `ode2` es incapaz de resolverla:

```
(%i21) ec: 'diff(y,t)+2*y^2-exp(-3*t)=0;
                      dy       2     - 3 t
(%o21)                -- + 2 y  - %e       = 0
                      dt
(%i22) ode2(ec,y,t);
(%o22)                            false
```

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

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

en Maxima,

```
(%i23) load(diffeq)$
(%i24) f(t,y):= -2*y^2+exp(-3*t) $
(%i25) res: runge1(f,0,5,0.5,1);
(%o25) [[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,
```

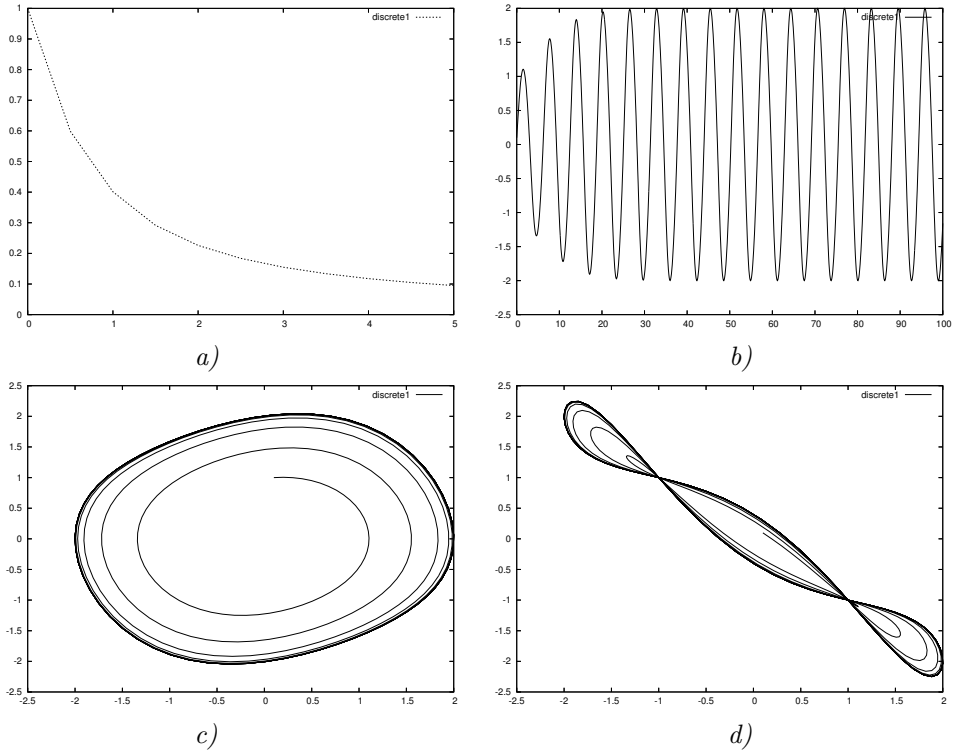


Figura 3.2: Resolución numérica de ecuaciones diferenciales con 'diffeq': *a)*, solución de la ecuación de primer orden $\frac{dy}{dt} = -2y^2 + \exp(-3t), y(0) = 1$; *b)*, solución de la ecuación de segundo orden $\frac{d^2y}{dt^2} = 0{,}2(1-y^2)\frac{dy}{dt} - y$, con las condiciones $y(0) = 0{,}1$ y $y'(0) = 1$; *c)*, diagrama (y, y'); *d)*, diagrama (y, y'').

```
.1336603954558797, .1176289334565761, .1050518038293819,
.09491944625388439], [- 1.0, - 0.49399612385452,
- .2720512734596094, - .1589442862446483, - .09974417126696171,
- .06705614729331426, - .04779722499498942, - .03570266617749457,
- .02766698775990988, - .02207039201652747, - .01801909665196759]]
(%i26) plot2d([discrete,res[1],res[2]])$
```

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 3.2 *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

$$g(t, y, y') = \frac{d^2y}{dt^2} = 0{,}2(1-y^2)y' - y,$$

```
(%i27) g(t,y,yp) := 0.2*(1-y^2)*yp - y $
(%i28) res: runge2(g,0,100,0.1,0.1,1)$
(%i29) plot2d([discrete,res[1],res[2]],
              [gnuplot_curve_styles,["with lines"]])$
(%i30) plot2d([discrete,res[2],res[3]],
              [gnuplot_curve_styles,["with lines"]])$
(%i31) plot2d([discrete,res[2],res[4]],
              [gnuplot_curve_styles,["with lines"]])$
```

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. 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 3.2.

3.7. Probabilidades y estadística

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 o los cuantiles,

```
(%i1) load(distrib)$
(%i2) assume(s>0)$
(%i3) cdf_normal(x,mu,s);
                            x - mu
                        erf(---------)
                            sqrt(2) s    1
(%o3)                       ------------- + -
                                 2         2
(%i4) pdf_poisson(5,1/s);
                                  - 1/s
                                 %e
(%o4)                            -------
                                     5
                                 120 s
(%i5) quantile_student_t(0.05,25);
(%o5)                    - 1.708140543186975
```

También se pueden calcular los momentos de primer y segundo orden,

```
(%i6) mean_weibull(3,67);
                                  1
                            67 gamma(-)
                                  3
(%o6)                       -----------
```

```
                                3
(%i7) var_binomial(34,1/8);
                              119
(%o7)                         ---
                              32
```

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

```
(%i8) random_negative_binomial(9,1/5,15);
(%o8) [65, 28, 44, 31, 19, 22, 28, 37, 42, 29, 47, 32, 53, 39, 57]
```

En cuanto a la estadística descriptiva, se remite al interesado a "Primeros pasos en Maxima".

En un futuro próximo, Maxima contendrá un paquete que incluye procedimientos inferenciales y de contraste estadístico de hipótesis, tanto paramétricos como no paramétricos. Sigue a continuación una sesión sobre regresión lineal simple[2]

```
(%i9) load("stats.mac")$
Warning: global variable 'numer' is set to 'true'.
Warning: global variable 'fpprintprec' is set to 7.
(%i10) datos:[[125,140.7],[130,155.1],[135,160.3],[140,167.2],[145,169.8]]$
(%i11) z:simple_linear_reg(datos,conflevel=0.99);
               [      SIMPLE LINEAR REGRESSION      ]
               [                                    ]
               [      model = 1.406 x - 31.19       ]
               [                                    ]
               [       correlation = .9611685       ]
               [                                    ]
               [      v_estimation = 13.57967       ]
               [                                    ]
(%o11)         [ b_conf_int = [.04469634, 2.767304] ]
               [                                    ]
               [ hypotheses = H0: b = 0 ,H1: b # 0  ]
               [                                    ]
               [        statistic = 6.032687        ]
               [                                    ]
               [   distribution = [student_t, 3]    ]
               [                                    ]
               [        p_value = .003805955        ]
```

Maxima nos muestra los resultados más importantes: la ecuación de la recta de regresión, el coeficiente de correlación, la estimación insesgada de la varianza de los errores aleatorios del modelo $y_i = a + bx_i + \epsilon_i$, con $\{\epsilon_i\}$ iid $N(0, \sigma^2)$. Además, obtenemos el intervalo de confianza para la pendiente de la recta de regresión, qué hipótesis se está contrastando sobre b (la nula y la alternativa), el estadístico del contraste, su distribución y el p-valor correspondiente; en este caso, hay suficiente evidencia (con nivel de significación 0.01) para rechazar $H_0 : b = 0$ en favor de $H_1 : b \neq 0$.

Veamos cómo hacer uso de estos resultados; en primer lugar, nos puede interesar una representación gráfica de los datos, junto con la recta estimada (Figura 3.3),

[2]Debido a que este paquete aún está en fase de desarrollo, es posible que en su versión final los resultados no se muestren tal cual se ve en estos ejemplos.

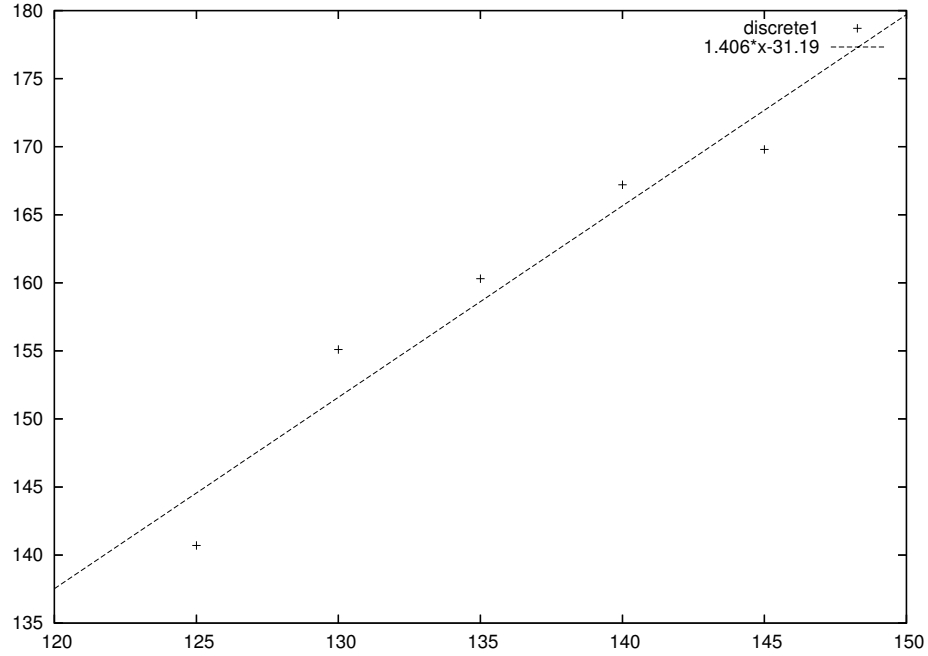


Figura 3.3: Ajuste del modelo de regresión lineal.

```
(%i12) plot2d([[discrete, datos], take_inference(model,z)],
              [x,120,150],
              [gnuplot_curve_styles, ["with points","with lines"]] )$
```

En caso de querer estimar la respuesta y para el valor $x = 133$,

```
(%i13) take_inference(model,z), x=133;
(%o13)                         155.808
```

Pero Maxima puede hacer más por nosotros. Además de los resultados mostrados, la variable `z` guarda otros que no muestra, porque entiende que no son de uso frecuente; esta variable almacena también las medias y varianzas de x e y, el coeficiente de determinación ajustado, los intervalos de confianza para los parámetros a y σ^2 del modelo, los intervalos de confianza para la media condicionada y para una nueva observación y, finalmente, los pares de residuos $(\hat{a}+\hat{b}x_i, y_i-\hat{a}-\hat{b}x_i)$, imprescindibles para el análisis de la bondad de ajuste del modelo y que se representan en la Figura 3.4 como resultado de la instrucción `%i16`. Veamos cómo seguir haciendo uso de la variable `z`:

```
(%i14) take_inference(means,z);
(%o14)                      [135.0, 158.62]
(%i15) /* Intervalo de confianza para nueva predicción */
       take_inference(new_pred_conf_int,z), x=133;
(%o15)                   [132.0729, 179.5431]
(%i16) plot2d([discrete,take_inference(residuals,z)])$
```

En fin, para saber qué resultados están almacenados en la variable `z`, no hay más que escribir

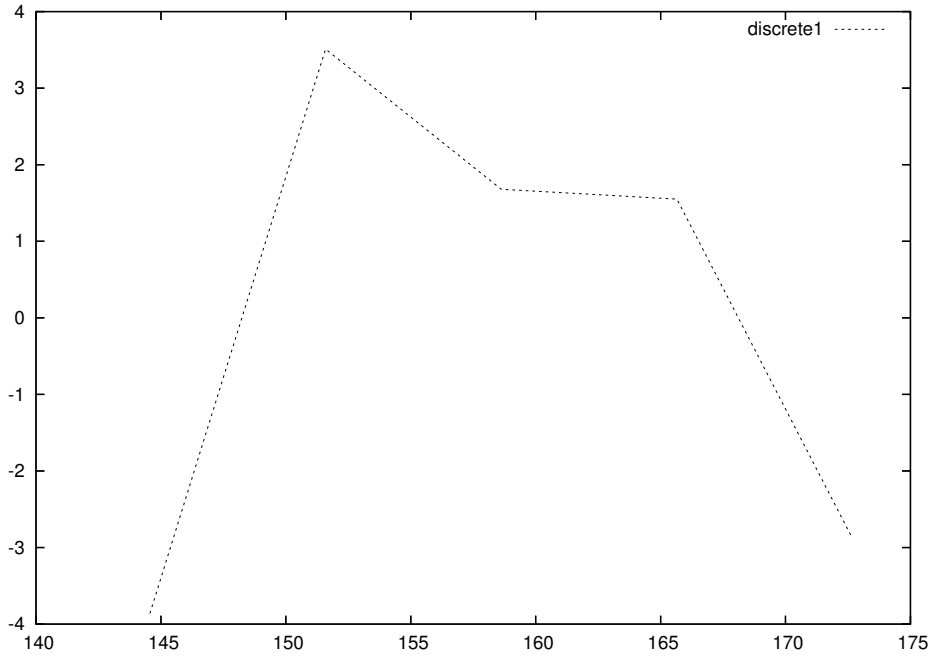


Figura 3.4: Predicciones y residuos: $(\hat{a} + \hat{b}x_i, y_i - \hat{a} - \hat{b}x_i)$.

```
(%i17) items_inference(z);
(%o17) [model, means, variances, correlation, adc, a_estimation,
a_conf_int, b_estimation, b_conf_int, hypotheses, statistic,
distribution, p_value, v_estimation, v_conf_int,
cond_mean_conf_int, new_pred_conf_int, residuals]
```

3.8. 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) - ((9 x - 39) charfun2(x, minf, 3)
```

```
 - 12 charfun2(x, 7, inf) + (30 x - 222) charfun2(x, 6, 7)
 + (18 - 10 x) charfun2(x, 3, 6))/6
(%i4) f(x):=''%;
(%o4) f(x) := - ((9 x - 39) charfun2(x, minf, 3)
 - 12 charfun2(x, 7, inf) + (30 x - 222) charfun2(x, 6, 7)
 + (18 - 10 x) charfun2(x, 3, 6))/6
```

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

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

Unos comentarios antes de continuar. Los datos los hemos introducido como una lista de pares de números, pero también la función admite una matriz de dos columnas o una lista de números, asignándole en este último caso las abscisas secuencialmente a partir de la unidad; además, la lista de pares de la variable `datos` no ha sido necesario ordenarla respecto de la primera coordenada, asunto del que ya se encarga Maxima por cuenta propia.

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

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

```
(%o10) [1.043464999999768, 5.567941928958183,
                                          2.89319655125692]
```

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

```
(%i11) plot2d(['(f(x)),g(x),[discrete,datos]],[x,0,10],
          [gnuplot_curve_styles,
                  ["with lines",
                   "with lines",
                   "with points pointsize 3"]])$
```

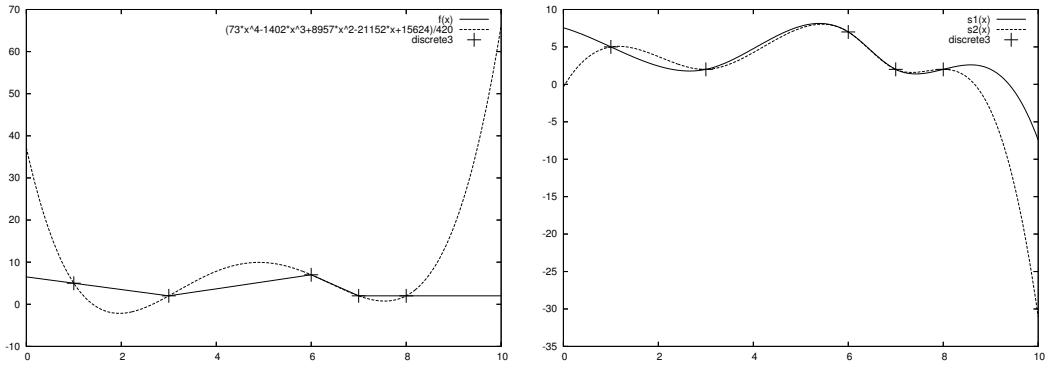
cuyo resultado se ve en el apartado *a)* de la Figura 3.5.

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`,

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

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

```
(%i15) cspline(datos,'varname='z,d1=1,dn=0);
                3          2
(%o15) ((21051 z  - 130635 z  + 218421 z - 7317)
                                    3           2
 charfun2(z, minf, 3) + (- 55908 z  + 1285884 z  - 9839808 z
 + 25087392) charfun2(z, 7, inf)
           3           2
 + (66204 z  - 1278468 z  + 8110656 z - 16797024)
                               3          2
 charfun2(z, 6, 7) + (- 16180 z  + 204444 z  - 786816 z
 + 997920) charfun2(z, 3, 6))/20304
```



a) *b)*

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

```
(%i16) s2(z):=''%$
(%i17) map(s2,[7.3,25/7,%pi]), numer;
(%o17) [1.595228723404633, 2.881415315197325,
                                        2.076658794432381]
```

Con esto hemos obtenido dos interpoladores distintos por el método de los *splines* cúbicos; con el siguiente código pedimos su representación gráfica, cuyo resultado se observa en el apartado *b)* de la Figura 3.5.

```
(%i18) plot2d(['(s1(x)),'(s2(x)),[discrete,datos]],[x,0,10],
          [gnuplot_curve_styles,
                  ["with lines",
                   "with lines",
                   "with points pointsize 3"]])$
```

Capítulo 4

Maxima como herramienta pedagógica

En los últimos tiempos se promueve el uso de las tecnologías informáticas en todos los niveles de la enseñanza. Centrándonos en los sistemas de cálculo simbólico, se pueden citar algunas ideas que, como todas las ideas, son por naturaleza opinables y sujetas a múltiples matices.

Las ventajas:

- Permite experimentar directamente sobre modelos reales, de mayor dificultad matemática, evitando la dispersión de esfuerzos en cálculos tediosos.
- Hay sesiones con los alumnos en los que se desea trabajar más el concepto que las habilidades manipulativas.
- El alumno aprenderá más rápidamente a utilizarlo si asiste a explicaciones donde el profesor lo usa junto con un cañón proyector.
- Ayuda a interpretar los aspectos geométricos de los fenómenos bajo estudio; los gráficos son inmediatos, al contrario que cuando se hacen a mano en la pizarra.
- Promoción del trabajo creativo frente al rutinario.
- La licencia libre de Maxima permite que los alumnos puedan disponer de él en casa; pudiendo diseñarse nuevas formas de trabajo personal.
- Puede fomentar la colaboración entre profesionales de la educación. Es bueno crear equipos que compartan experiencias educativas; lo que a uno se le está escapando, otro lo ve claro.
- Al disponer Maxima de un lenguaje muy sencillo, el alumno que tenga que programar algoritmos lo puede hacer más fácil, ir al grano, sin preocuparse de declaraciones de variables, herencias de clases, crear otras estructuras de datos previas, etc.
- Permite más tiempo para reflexionar sobre los modelos y los resultados.

Los inconvenientes:

- No siempre existe una disponibilidad razonable de los recursos informáticos y de hardware.
- Necesita más tiempo de dedicación por parte del profesor, no reconociéndose suficientemente a nivel administrativo su esfuerzo adicional.

- Las actividades y el proceso de enseñanza – aprendizaje deben ser diferentes a los planteamientos clásicos. Idear nuevas tareas, nuevas formas y nuevos estilos.
- Al profesor se le debe facilitar información sobre el software; si tiene que aprender por su cuenta habrá rechazo, ya que verá ante sí un doble trabajo: aprender a utilizar el programa y aprender a usarlo como herramienta pedagógica.
- El profesor debe adquirir ciertos conocimientos mínimos de programación, con el fin de poder diseñar actividades a conveniencia y explotar las posibilidades del programa.

Los riesgos:

- Evitar convertir una clase de Matemáticas en una clase de Maxima. Los errores sintácticos frenan la marcha de la clase; es imprescindible que los alumnos tengan que escribir lo menos posible.
- Evitar que el alumno abandone sus esfuerzos por adquirir habilidades y automatismos manuales.
- Riesgo de pérdida del sentido crítico: fe ciega en los resultados del ordenador.
- Los entornos gráficos pueden dispersar la atención del alumno (menús, opciones, botones, ventanas, más ventanas). El entorno de texto puede ayudar a centrar la atención.
- No pretender que esta sea la gran revolución de la pedagogía matemática. Maxima será útil en algunos casos, e inútil en otros. No forzar situaciones, pero hay contextos que lo piden a gritos.

Capítulo 5

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.

5.1. 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>` suele ser una única sentencia o un bloque con variables locales; véanse los siguientes ejemplos:

```
(%i7) loga(x,a):= ev(log(x) / log(a), numer) $
(%i8) loga(7, 4);
(%o8)                    1.403677461028802
(%i9) fact(n):=block([prod:1],
          for k:1 thru n do prod:prod*k,
          prod )$
(%i10) fact(45);
(%o10) 119622220865480194561963161495657715064383733760000000000
```

En el primer caso (`loga`) se definen los logaritmos en base arbitraria (Maxima sólo tiene definidos los naturales); además, previendo que sólo los vamos a necesitar en su forma numérica, solicitamos que nos los evalúe siempre en formato decimal. En el segundo caso (`fact`) hacemos uso de un bucle para calcular factoriales. Esta última función podría haberse escrito recursivamente mediante un sencillo condicional,

```
(%i11) fact2(n):= if n=1 then 1
                        else n*fact2(n-1) $
(%i12) fact2(45);
(%o12) 119622220865480194561963161495657715064383733760000000000
```

O más fácil todavía, `:-)`,

```
(%i13) 45!;
(%o13) 119622220865480194561963161495657715064383733760000000000
```

Acabamos de ver dos estructuras de control de flujo comunes a todos los lenguajes de programación, las sentencias `if-then-else` y los bucles `for`. En cuanto a las primeras, puede ser de utilidad la siguiente tabla de operadores relacionales

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

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 ;
                                    9
(%o17)                      sin(4) < --
                                    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>
```

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) [cont, var];
(%o24)                     [cont, 27.0]
```

Véase en este último resultado cómo la variable `cont` no queda con valor ninguno asignado, mientras que `var` sí; la razón es que `cont` es local al bucle, expirando cuando éste termina.

Otra variante de `for` es cuando el contador recorre los valores de una lista; su forma es

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

y un ejemplo:

```
(%i25) 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 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`

```
1: foo(y) :=
2:  block ([u:y^2],
3:    u: u+3,
4:    u: u^2,
5:    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        /* punto de ruptura al final de línea 3*/
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 final de 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
```

Para más información tecléese `? depura`.

5.2. 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 las expresiones matemáticas como listas Lisp:

```
(%i1) 3/4;
                                3
(%o1)                           -
                                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)`. Al entrar en modo Lisp mediante el símbolo `:lisp` le estamos pidiendo 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 qué pasa con la función de redondeo, la cual no existe en Maxima:

```
(%i3) round(4.897);
(%o3)                      round(4.897)
```

Pero sí hay una función en Lisp que realiza este cálculo; si queremos que Maxima la invoque, sólo necesitamos llamarla con el prefijo `?`, tal como indica este ejemplo:

```
(%i4) ?round(4.897);
(%o4)                           5
```

A efectos de saciar nuestra curiosidad, veamos cómo se almacenan internamente otras expresiones simbólicas:

```
(%i5) x+y*2;
(%o5)                        2 y + x
(%i6) [1,2];
(%o6)                        [1, 2]
(%i7) :lisp $%o5
((MPLUS SIMP) $X ((MTIMES SIMP) 2 $Y))
(%i7) :lisp $%o6
((MLIST SIMP) 1 2)
```

Para terminar, un ejemplo de cómo definir a nivel de maxima una función de redondeo,

```
(%i8) :lisp (defun $redondea (x) (round x))
$REDONDEA
(%i9) redondea(4.7777);
(%o9)                           5
```

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 compiladores libres como `clisp`, `gcl`, `cmucl` o `sbcl`.

Apéndice A

Ejemplos de programación

```
/*              COPYRIGHT NOTICE

Copyright (C) 2006 Mario Rodriguez Riotorto

This program is free software; you can redistribute
it and/or modify it under the terms of the
GNU General Public License as published by
the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it
will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details at
http://www.gnu.org/copyleft/gpl.html
*/


/*            INTRODUCCION
Ejemplos de programacion de funciones orientadas
a la docencia.

En los comentarios se ha evitado el uso de acentos
y caracteres especiales, a fin de evitar errores
de lectura en plataformas con diferentes
codificaciones de las fuentes.

Este conjunto de funciones esta almacenado en el
fichero cadiz.mac; para cargar en memoria el
paquete, ejecutese desde Maxima

load("ruta_a/cadiz.mac");

Para comentarios, sugerencias y aclaraciones,
mario ARROBA edu PUNTO xunta PUNTO es
*/
```

```
/* Transformaciones de la funcion seno. Opciones: */
/* 'amp: amplitud                                 */
/* 'fase: fase                                    */
/* 'frec: frecuencia                              */
/* 'desp: desplazamiento vertical                 */
/* Las opciones se separan por comas y se esta-   */
/* blecen con el formato:                         */
/*             opcion = valor                     */
/* Ejemplos de uso:                               */
/*     seno(amp=4);                               */
/*     seno(frec=2*%pi/3, amp=5);                 */
/*     seno(frec=%pi/2, amp=5, fase=5, desp=6);   */
/*     seno();                                    */
seno([select]):=
  block([options, defaults, aux, am, fa, fr, dv, ymin, ymax],
    /* nombres de opciones y sus valores por defecto */
    options  :  ['amp, 'fase, 'frec, 'desp],
    defaults :  [  1,     0,     1,     0 ],
    /* actualiza opciones del usuario */
    for i in select do(
      aux: ?position(lhs(i),options),
      if numberp(aux) and aux <= length(options) and aux >= 1
        then defaults[aux]: rhs(i)),

    /* actualiza variables locales */
    am: defaults[1], /* amplitud */
    fa: defaults[2], /* fase */
    fr: defaults[3], /* frecuencia */
    dv: defaults[4], /* desplazamiento vertical */
    ymin: min(-10,-abs(am)+dv),
    ymax: max(10,abs(am)+dv),

    /* representa el seno y su transformada,
       con 'gnuplot_preamble' se hacen los retoques
       necesarios para que aparezcan la malla, los
       ejes, el titulo y las dimensiones de la
       ventana grafica */
    plot2d([sin(x), dv + am * sin (fa + fr * x)], [x, -10, 10],
      [gnuplot_preamble,
        sconcat("set title 'Transformaciones del seno';",
                "set grid; set xzeroaxis lt -1; set yzeroaxis lt -1;",
                "set arrow 1 from -10, ", dv, " to 10, ", dv, " nohead lt 2;",
                if -fa > -10 and -fa < 10
                  then sconcat("set arrow 2 from ", -fa, ", ", ymin,
                               " to ", -fa, ", ", ymax, " nohead lt 2;")
                  else "",
                "set yrange [", ymin, ":", ymax, "]")]),
    /* muestra en terminal la funcion */
    print("Transformacion de la funcion sinusoidal:"),
    print("    Amplitud    =", am),
    print("    Fase        =", fa),
    print("    Frecuencia  =", fr),
    print("    Desp. vert. =", dv),
    'y = dv + am * sin (fa + fr * 'x)   )$
```

```
/* Definicion de la integral mediante las sumas   */
/* de Riemann. Permite seleccionar si se dibujan  */
/* los rectangulos inferiores o superiores.       */
/* Argumentos:                                    */
/* funcion: funcion a integrar                    */
/* rango: de la forma [x,-5,10]                   */
/* n: numero de rectangulos                       */
/* tipo: 'sup o 'inf                              */
/* Ejemplo de uso:                                */
/*     riemann(u^3,[u,0,5],10,'inf);              */
/*     riemann(exp(-v^2),[v,-2,2],10,'sup);       */
riemann(funcion,rango,n,tipo):=
   block([abscisas,ordenadas,d,numer:true,ratprint:false,int,suma],
      /* variables globales numer y ratprint se modifican temporalmente */
      d: (rango[3]-rango[2])/n,
      if not member(tipo,['sup,'inf])
         then error("Tipo de sumas incorrecto: debe ser 'sup o 'inf"),
      abscisas: makelist(rango[2]+k*d-d/2,k,1,n),
      if tipo = 'inf
         then ordenadas: makelist([k, min(ev(funcion,rango[1]=k-d/2),ev(funcion,rango[1]=k+d/2))],
                                  k,abscisas)
         else ordenadas: makelist([k, max(ev(funcion,rango[1]=k-d/2),ev(funcion,rango[1]=k+d/2))],
                                  k,abscisas),
      plot2d([[discrete,ordenadas],funcion],rango,
                 [gnuplot_curve_styles,
                    ["with boxes fs solid 0.3 border -1",
                     "with lines lt 3 lw 2"]],
                 [gnuplot_preamble,
                    ["set title 'Sumas de Riemann'; set nokey"]]),
      int: integrate(funcion,rango[1],rango[2],rango[3]),
      suma: apply("+",d*makelist(k[2],k,ordenadas)),
      print(""),
      print("Area bajo la curva ..=", int),
      print("Area rectangulos ....=", suma),
      print("Diferencia (int-sum) =", int-suma),
      print("") )$
```

```
/* Definicion de la integral mediante las sumas   */
/* de Riemann. Dibuja rectangulos inferiores y    */
/* superiores.                                    */
/* Argumentos:                                    */
/* funcion: funcion a integrar                    */
/* rango: de la forma [x,-5,10]                   */
/* n: numero de rectangulos                       */
/* Ejemplo de uso:                                */
/*     riemann2(u^3,[u,0,5],10);                  */
/*     riemann2(sin(x)+1,[x,0,5],5);              */
riemann2(funcion,rango,n):=
   block([abscisas,ordenadas,d,numer:true,ratprint:false,inf,sup,int,sumasup,sumainf],
      /* variables globales numer y ratprint se modifican temporalmente */
      d: (rango[3]-rango[2])/n,
      abscisas: makelist(rango[2]+k*d-d/2,k,1,n),
      inf: makelist([k, min(ev(funcion,rango[1]=k-d/2),ev(funcion,rango[1]=k+d/2))],k,abscisas),
      sup: makelist([k, max(ev(funcion,rango[1]=k-d/2),ev(funcion,rango[1]=k+d/2))],k,abscisas),
      plot2d([[discrete,sup],[discrete,inf],funcion],rango,
                 [gnuplot_curve_styles,
                    ["with boxes fs solid 0.3 border -1",
                     "with boxes fs solid 0.3 border -1",
                     "with lines lt 3 lw 2"]],
                 [gnuplot_preamble,
                    ["set title 'Sumas de Riemann'; set nokey"]]),

      int: integrate(funcion,rango[1],rango[2],rango[3]),
      sumasup: apply("+",d*makelist(k[2],k,sup)),
      sumainf: apply("+",d*makelist(k[2],k,inf)),
      print(""),
      print("Area bajo la curva ..=", int),
      /* a continuacion, diferencia de las sumas superior e inferior */
      print("Diferencia (sup-inf) =", sumasup-sumainf),
      print("") )$
```

```
/* Genera informacion relativa al calculo de los limites.  */
/* Argumentos:                                             */
/*   expr: expresion objetivo                              */
/*   var: nombre de la variable independiente              */
/*   val: valor hacia el que tiende var                    */
/*   xmin, xmax: subdominio para el grafico                */
/*   argumento opcional dir: indica si me aproximo         */
/*     por la izquierda (minus) o por la derecha (plus)    */
/* Ejemplos de uso:                                        */
/*     limite((z^3-z^2+5)/(z-4),z,7,5,9);                  */
/*     limite((z^3-z^2+5)/(z-4),z,7,5,9,'minus);           */
/*     limite((z^3-z^2+5)/(z-4),z,7,5,9,'plus);            */
npar:6$ /* variable global: controla numero de pares de la tabla */
limite(expr,var,val,xmin,xmax,[dir]):=
   block([xx,yy,tab,lim,side:0],
      /* asignando valor a side */
      if length(dir) > 0
         then if dir[1] = 'minus
                 then side: -1
                 else if dir[1] = 'plus
                         then side: 1,

      /* calculo de las abscisas */
      if side # 0
         then xx: float(makelist(val+side/2^k,k,0,npar))
         else xx: float(append(makelist(val-1/2^k,k,0,npar),
                               reverse(makelist(val+1/2^k,k,0,npar)))),

      /* achegamento tabular */
      yy: map (lambda ([u], subst(u,var,expr)), xx),
      tab: transpose(matrix(append([var],xx),
                            append([expr],yy))),
      print("     Estudio del limite"),
      print("     ================="),
      print(tab),

      /* calculo del limite */
      if side = -1
         then (lim: limit(expr,var,val,'minus),
               print(float(lim)=lim),
               print('limit(expr,var,val,'minus)= lim)  )
         else if side = 1
                 then (lim: limit(expr,var,val,'plus),
                       print(float(lim)=lim),
                       print('limit(expr,var,val,'plus)= lim)  )
                 else (lim: limit(expr,var,val),
                       print(float(lim)=lim),
                       print('limit(expr,var,val)= lim)  ),

      /* representacion grafica */
      plot2d([expr,[discrete, args(rest(tab))]],[var,xmin,xmax],
                  [gnuplot_curve_styles,
                              ["with lines","with impulses"]]),
      'fin  )$
```

```
/* Funciones para una actividade sobre transformacions no plano.          */
/* Se incluyen dos variables que contienen listas de puntos a transformar. */
/* Ejemplos de uso:                                                       */
/*      barco; simx(barco);                                               */
/*      dibuja(barco, simx(barco));                                       */
/*      dibuja(barco, simo(barco));                                       */
/*      dibuja(barco, traslacion(barco, 6, -7));                          */
/*      dibuja(bs, simy(bs), homotecia(bs,-2));                           */
/*      dibuja(bs, simy(bs), homotecia(bs,-2), homotecia(bs,5));          */

/* Simetria respecto del eje de abscisas */
simx(pts):=block([trans:[]],
   for i in pts do trans:endcons([i[1],-i[2]],trans),
   return(trans) )$
/* Simetria respecto del eje de ordenadas */
simy(pts):=block([trans:[]],
   for i in pts do trans:endcons([-i[1],i[2]],trans),
   return(trans) )$
/* Simetria respecto del origen de coordenadas */
simo(pts):=block([trans:[]],
   for i in pts do trans:endcons([-i[1],-i[2]],trans),
   return(trans) )$
/* Traslacion respecto del vector [v1,v2] */
traslacion(pts,v1,v2):=block([trans:[]],
   for i in pts do trans:endcons([i[1]+v1,i[2]+v2],trans),
   return(trans) )$
/* Homotecia respecto del origen con parametro k */
homotecia(pts,k):=block([trans:[]],
   for i in pts do trans:endcons(k*[i[1],i[2]],trans),
   return(trans) )$
/* dibuja numero arbitrario de listas de vertices */
dibuja([figs]):=
   apply('plot2d,
         append([map(lambda ([x], ['discrete, x]), figs)],
         [[gnuplot_preamble, "set grid;set zeroaxis linetype 6"]]))$

/* Vertices de dos objetos: barco y bs */
barco:
[[1,1],[1,3],[2,3],[2,6],[4,6],[4,3],[15,3],[12,1],[1,1]]$
bs:
[[64,-104],[64,-99],[61,-95],[57,-95],[55,-94],[55,-90],[72,-89],[80,-88],[77,-86],[80,-88],
 [82,-91],[80,-88],[72,-89],[55,-90],[47,-89],[39,-87],[33,-84],[32,-81],[35,-75],[36,-69],
 [42,-69],[45,-68],[42,-69],[36,-69],[35,-67],[35,-62],[35,-60],[39,-60],[45,-61],[51,-61],
 [53,-65],[56,-68],[61,-70],[67,-71],[73,-70],[78,-67],[80,-64],[81,-61],[78,-58],[75,-60],
 [75,-62],[78,-62],[81,-61],[81,-56],[79,-50],[75,-47],[70,-44],[62,-44],[58,-45],[53,-50],
 [52,-52],[49,-53],[48,-52],[48,-50],[49,-48],[51,-49],[52,-52],[51,-54],[51,-57],[51,-61],
 [45,-61],[39,-60],[35,-60],[32,-57],[32,-52],[32,-49],[35,-45],[39,-42],[43,-41],[48,-42],
 [51,-43],[55,-47],[51,-43],[48,-42],[43,-41],[44,-38],[47,-35],[54,-16],[56,-5],[60,-11],
 [64,-6],[67,-12],[72,-7],[74,-14],[80,-9],[80,-16],[86,-11],[87,-17],[92,-12],[93,-21],
 [99,-16],[100,-22],[106,-19],[107,-27],[115,-25],[110,-32],[94,-72],[91,-72],[88,-73],
 [87,-74],[88,-76],[91,-76],[91,-79],[89,-81],[91,-79],[91,-76],[94,-77],[94,-79],[94,-77],
 [91,-76],[88,-76],[87,-74],[88,-73],[91,-72],[94,-72],[97,-75],[98,-80],[97,-82],[94,-84],
 [91,-85],[87,-83],[84,-80],[87,-83],[91,-85],[91,-95],[92,-102],[85,-105],[69,-106],[64,-104]]$
```