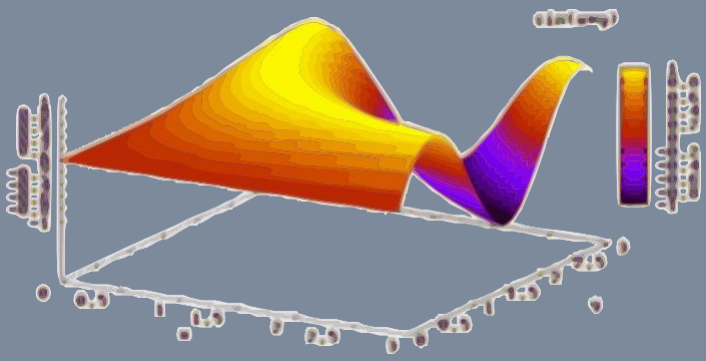
BREVE MANUAL DE

MAXIMA



R. Ipanaqué

Breve Manual de Maxima

Breve Manual de Maxima

R. Ipanaqué

Departamento de Ciencias - Área de Matemática
Universidad Nacional de Piura

publicaciones
eumed.net

Robert Ipanaqué Chero

Departamento de Ciencias - Area de Matemática
Universidad Nacional de Piura
Urb. Miraflores s/n, Castilla, Piura
PERÜ

http://www.unp.edu.pe/pers/ripanaque
robertchero©hotmail.com

La compos c ón de BREVE MANUAL DE MAX MA
se ha hecho en LaTeX, usando e ed tor bre TeXmaker 1 9 9.

Mayo 2010, publicado por el grupo eumed · net
Grupo de Investigación de la Universidad de Málaga, España
http://www.eumed.net

Hecho el depósito legal en la Biblioteca Nacional de España
con Resitro N°

ISBN

En memoria de mi padre)
Juan A. Ipanaqué Vargas

Índice

Prólogo v

1. Obtención de **Maxima** 1
   1.1. Descarga ... 1
   1.2. Instalación ... 2

2. Funcionamiento de **Maxima** 9
   2.1. Interfaz de cuaderno ... 9
   2.2. Interfaz basada en texto ... 10

3. Uso del sistema **Maxima** 12
   3.1. La estructura de Maxima ... 12
   3.2. Cuadernos como documentos ... 14
   3.3. Configuración de opciones y estilos ... 17
   3.4. Búsqueda de ayuda ... 19
   3.5. Reinicio ... 22
   3.6. Comentarios ... 23
   3.7. Paquetes en Maxima ... 23
   3.8. Advertencias y mensajes ... 24
   3.9. Interrupción de cálculos ... 25

4. Cálculos numéricos 26
   4.1. Aritmética ... 26
   4.2. Resultados exactos y aproximados ... 27
   4.3. Algunas funciones matemáticas ... 30
   4.4. Cálculos con precisión arbitraria ... 33
   4.5. Números complejos ... 35

5. Generación de cálculos 37
5.1. Uso de entradas y salidas previas 37
5.2. Definición de variables 39
5.3. Secuencia de operaciones 42
5.4. Impresión de expresiones sin evaluar 44

6. Cálculos algebraicos 47
6.1. Cálculo simbólico 47
6.2. Valores para símbolos 50
6.3. Transformación de expresiones algebraicas 54
6.4. Simplificación de expresiones algebraicas 56
6.5. Expresiones puestas en diferentes formas 59
6.6. Simplificación con asunciones 65
6.7. Selección de partes de expresiones algebraicas 66

7. Matemáticas simbólicas 68
7.1. Límites 68
7.2. Diferenciación 70
7.3. Integración 73
7.4. Sumas y Productos 78
7.5. Operadores relacionales y lógicos 81
7.6. Ecuaciones 84
7.7. Solución de Ecuaciones 85
7.8. Ecuaciones diferenciales ordinarias 88
7.9. Sistemas de ecuaciones diferenciales ordinarias lineales 90
7.10. Series de potencias 92
7.11. Transformada de Laplace 96

8. Matemáticas numéricas 97
8.1. Solución numérica de ecuaciones 97
8.2. Integrales numéricas 99

9. Funciones y programas 101
9.1. Definición de funciones 101
9.2. Reglas de transformación para funciones 111
9.3. Funciones definidas a partir de expresiones 114

10. Listas 117
10.1. Juntar objetos 117
10.2. Generación de listas 118
10.3. Elección de elementos de una lista 121

10.4. Prueba y búsqueda de elementos de una lista 124
10.5. Combinación de listas 126
10.6. Reordenamiento de listas 127
10.7. Agregar y quitar elementos de una lista 128
10.8. Reorganización de listas 129
10.9. Funciones adicionales para listas 130

11. Arrays 132

12. Matrices 136
12.1. Generación de Matrices 136
12.2. Elegir elementos de matrices 138
12.3. Operaciones matriciales 139
12.4. Funciones adicionales para matrices 144
12.5. Matrices asociadas a sistemas de ecuaciones 147
12.6. Autovalores y autovectores 148

13. Conjuntos 150
13.1. Generación de conjuntos 150
13.2. Conversiones entre conjuntos y listas 152
13.3. Elección de elementos de un conjunto 153
13.4. Prueba y búsqueda de elementos de un conjunto 154
13.5. Agregar y quitar elementos de un conjunto 156
13.6. Reorganización de conjuntos 157
13.7. Operaciones con conjuntos 157
13.8. Funciones adicionales para conjuntos 160

14. Gráficos 162
14.1. Gráficos básicos 163
14.2. Opciones 165
14.3. Gráficos de puntos y líneas 169
14.4. Gráficos paramétricos y polares 172
14.5. Combinación de gráficos 174
14.6. Gráficos de superficies tridimensionales 174
14.7. Gráficos de densidad y contornos 179
14.8. Gráficos animados 180

15. Gráficos con `draw` 183
15.1. Objetos gráficos bidimensionales 184
15.2. Opciones para los objetos gráficos bidimensionales 192
15.2.1. Opciones locales 192

15.2.2. Opciones locales genéricas 194
15.2.3. Opciones globales 195
15.2.4. Ejemplos ilustrativos 198
15.3. Objetos gráficos tridimensionales 203
15.4. Opciones para objetos gráficos tridimensionales 211
15.4.1. Opciones locales 211
15.4.2. Opciones locales genéricas 212
15.4.3. Opciones globales 212
15.4.4. Ejemplos ilustrativos 214
15.5. Fijación de valores para opciones 217
15.6. Gráficos múltiples 218
15.7. Gráficos animados 220

16. Archivos y operaciones externas 224
16.1. Generación de expresiones y archivos TEX 224
16.2. Generación de archivos HTML 228
16.3. Generación de expresiones Lisp y Fortran 228

17. Programación con **Maxima** 229
17.1. Operadores relacionales y lógicos 229
17.2. Operadores y argumentos 232
17.3. Programación funcional 235
17.4. Implementación del paquete: `ejemplo` 237

BIBLIOGRAFÍA 242

Prólogo

Este manual da una introducción al Software Libre Maxima v5.21.1, presentándolo como un potente Sistema Algebraico Computacional (Computer Algebraic System, o CAS) cuyo objeto es la realización de cálculos matemáticos, tanto simbólicos como numéricos; además de ser expandible, pues posee un lenguaje de programación propio.

Las razones para apostar por el uso de Software Libre pueden deducirse de las cuatro libertades asociadas a este tipo de Software: libertad de ejecutarlo) para cualquier propósito; libertad de estudiar cómo trabaja) y cambiarlo a voluntad de quien lo usa; libertad de redistribuir copias para ayudar al prójimo; y libertad de mejorarlo y publicar sus mejoras) y versiones modificadas en general) para que se beneficie toda la comunidad.

De hecho, las libertades asociadas a todo Software Libre y, en particular, al CAS Maxima hacen de éste una formidable herramienta pedagógica accesible a todos los presupuestos, tanto institucionales como individuales. No obstante, somos sinceros en señalar que no posee toda la versatilidad de sus símiles comerciales; pero el hecho que sea gratuito minimiza tal carencia. Hay que señalar, también, que cualquier actualización de un Software Libre puede obtenerse sin obstáculo alguno y así es posible contar inmediatamente con la última versión del mismo. Algo que no sucede con el Software Comercial, a menos que se tenga disponibilidad inmediata de dinero para pagar la actualización.

La idea de elaborar este manual surge de la necesidad de contar con bibliografía propia acerca de un CAS Libre para trabajar con alumnos de un curso de pregrado, los cuales ya estaban familiarizados con el uso de un CAS Comercial. La experiencia ha sido bastante satisfactoria y quedan en el tintero los borradores para la futura elaboración de un

libro en el que se plasmen los resultados obtenidos en tal curso.

Este manual se compone de diecisiete capítulos en los cuales se describen resumidamente las principales características y las funciones incorporadas en el núcleo de Maxima. Sin embargo, en el capítulo quince se hace una excepción y se aborda una breve descripción del paquete incorporado `draw`, pues éste amplia la capacidad gráfica que ofrecen las funciones incorporadas en el núcleo de Maxima. Además, en el último capítulo se dan los lineamientos generales para la elaboración de paquetes de funciones. Esto con la finalidad que el usuario obtenga el máximo provecho en el uso de Maxima.

R. lPANAQUÉ

Piura) Perú

Capítulo 1

Obtención de Maxima

Maxima puede funcionar en distintos sistemas operativos, entre ellos diversas variantes de Windows y de GNU/Linux. En este capítulo se tratará acerca de la descarga e instalación de Maxima en el sistema operativo Windows (95 o posterior). El lector interesado en utilizar Maxima en alguna variante de GNU/Linux, puede acceder a la sección Download de la web de Maxima y seguir las instrucciones que en ella se indican.

1.1 Descarga

Maxima se descarga gratuitamente desde la página de sourceforge que alberga a una gran cantidad de instaladores de softwares de código abierto[1]. Debemos destacar que por el hecho de ser gratuito no requiere de ningún password que siempre está asociada con el software comercial (también llamado software propietario o más acertadamente software privativo).

La dirección específica donde esta alojado Maxima es la siguiente:

`http://sourceforge.net/projects/maxima/files`

[1] Código abierto (en inglés open source) es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones morales y/o filosóficas las cuales destacan en el llamado software libre.

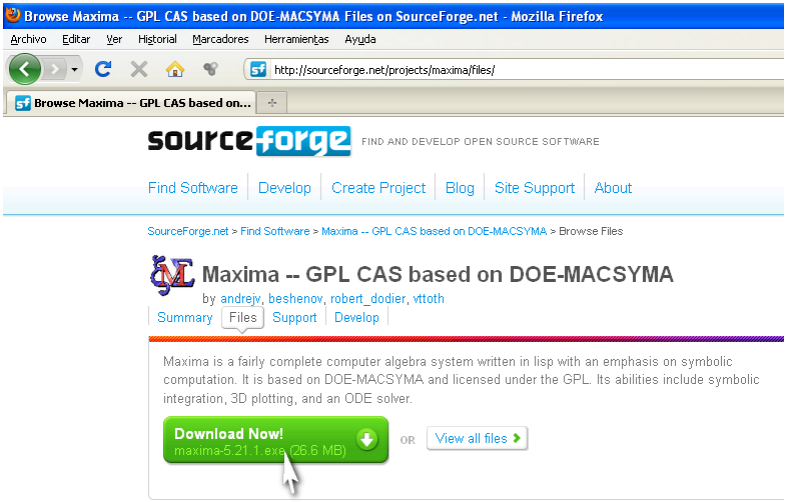


Figura 1.1: Porción de la página de descarga de `Maxima-5.21.1.exe`. El botón señalado permite la descarga directa de Maxima para Windows.

desde donde puede descargarse el archivo `Maxima-5.21.1.exe` que es el instalador de Maxima para Windows. Este instalador ocupa 25.3 MB de espacio en memoria.

Una vez descargado el instalador se verá un icono, como el que se aprecia en la figura 1.2, en la carpeta donde éste se haya descargado.

1.2 Instalación

Después de la descarga se procede a instalar Maxima, lo cual debe hacerse siguiendo los pasos que se detallan a continuación.

Figura 1.2: Icono del instalador de `Maxima-5.21.1.exe`.

1. Hacer doble clic sobre el icono del instalador.

2. Si aparece un cuadro como el de la figura 1.3, hacer clic sobre el botón Ejecutar.

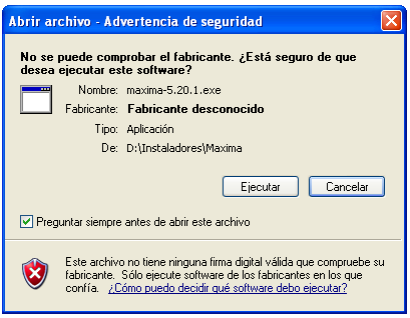


Figura 1.3: Cuadro de verificación.

3. Seleccionar el idioma (por defecto esta seleccionado Español) y hacer clic sobre el botón Aceptar (fig. 1.4).

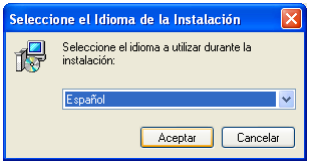


Figura 1.4: Cuadro para seleccionar el idioma.

4. Hacer clic sobre el botón Siguiente del cuadro Bienvenido al asistente de instalación de Maxima (fig. 1.5).

5. Seleccionar la opción Acepto e acuerdo del cuadro Acuerdo de Licencia. Luego hacer clic en el botón Siguiente del mismo cuadro (fig. 1.6).

6. Hacer clic en el botón Siguiente del cuadro Información (fig. 1.7).

7. Seleccionar la carpeta en la cual se quiere instalar Maxima (generalmente se deja la carpeta que aparece por defecto) y luego

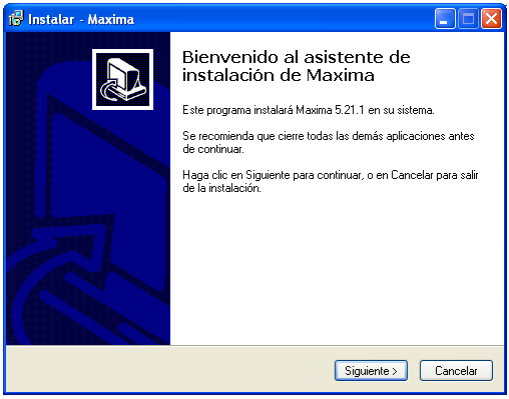


Figura 1.5: Cuadro Bienvenido al asistente de instalación de Maxima.

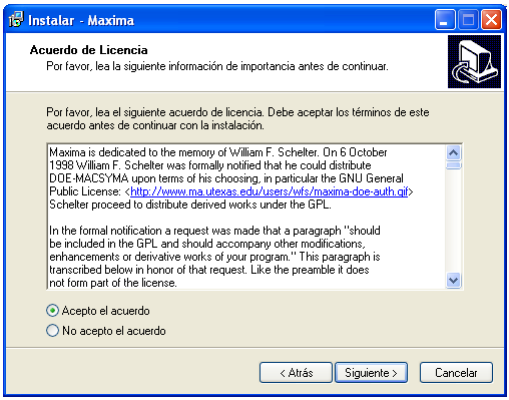


Figura 1.6: Cuadro Acuerdo de Licencia.

hacer clic en el botón Siguiente del cuadro Seleccione la Carpeta de Destino (fig. 1.8).

8. En el cuadro Seleccione los Componentes desmarcar las casillas Portugués y Portugués Brasileño ya que sólo utilizaremos el idioma Español. Esto permite, a su vez, el ahorro de memoria (fig. 1.9).

9. Seleccionar la carpeta del menú Inicio en la cual se quiere ubi-

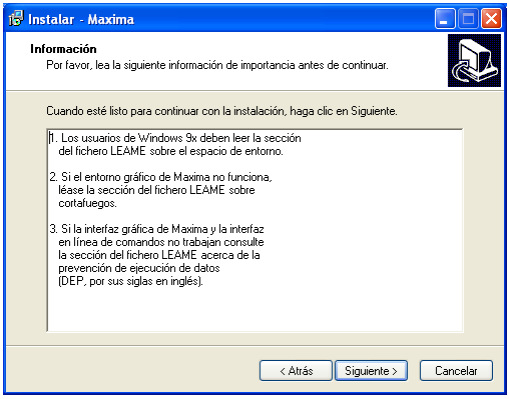


Figura 1.7: Cuadro Información.

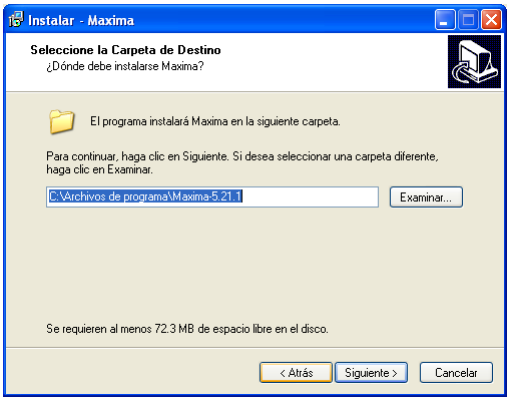


Figura 1.8: Cuadro Seleccione la Carpeta de Destino.

car el icono de acceso a Maxima (generalmente se deja la carpeta que aparece por defecto) y luego hacer clic en el botón Siguiente del cuadro Seleccione la Carpeta del Menú Inicio (fig. 1.10).

10. Hacer clic en el botón Siguiente del cuadro seleccione las Tareas Adicionales para que el asistente cree un icono de acceso directo a Maxima en el escritorio (fig. 1.11).

11. Hacer clic en el botón Instalar del cuadro Listo para Instalar

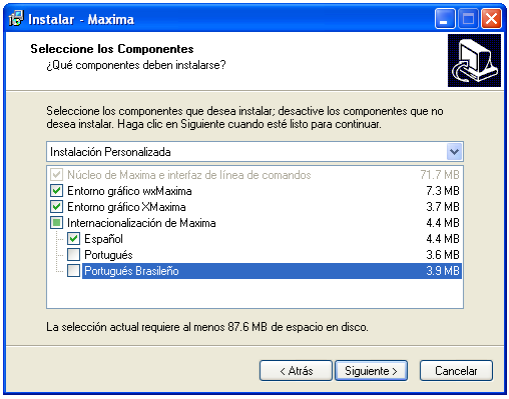


Figura 1.9: Cuadro Seleccione los Componentes.

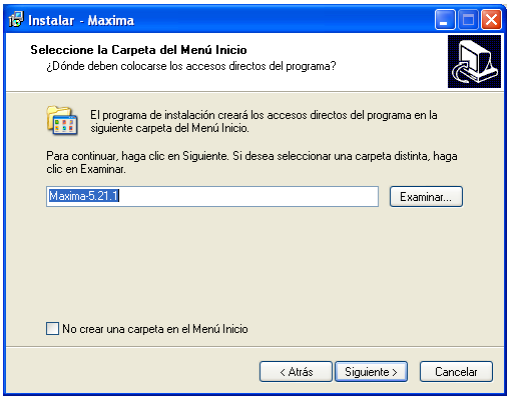


Figura 1.10: Cuadro Seleccione la Carpeta del Menú Inicio.

(fig. 1.12).

12. Hacer clic en el botón Siguiente del cuadro Información (fig. 1.13).

13. Por último, hacer clic en el botón Finalizar del cuadro Completando la Instalación de Maxima (fig. 1.14).

Después de seguir el procedimiento anterior deben haberse instalado: el núcleo de Maxima que es el responsable de todos los cálculos

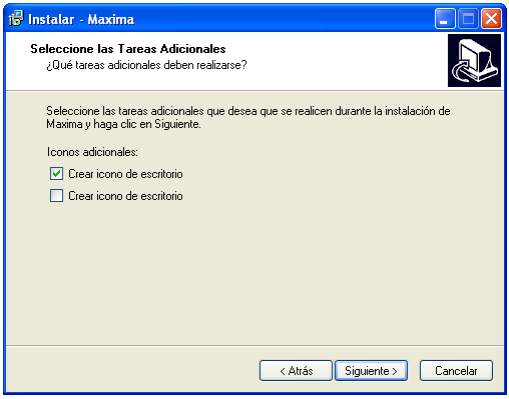


Figura 1.11: Cuadro Seleccione las Tareas Adicionales.

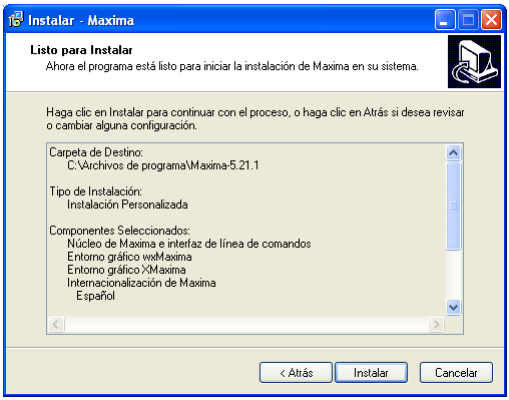


Figura 1.12: Cuadro Listo para Instalar.

y permite una interfaz de texto, el entorno gráfico wxMaxima que permite una interfaz gráfica (o interfaz de "cuaderno") bastante amigable, el entorno gráfico XMaxima que también permite una interfaz gráfica (aunque menos amigable que wxMaxima) y una aplicación para usuarios de habla hispana. Además, debe haberse creado automáticamente, en el escritorio de su ordenador (computadora), el icono de acceso directo al entorno gráfico wxMaxima (fig. 1.15).

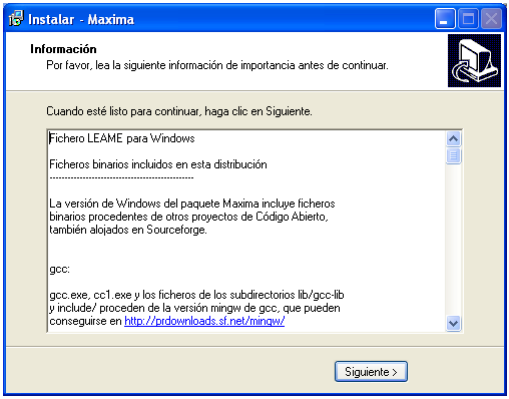


Figura 1.13: Cuadro Información.

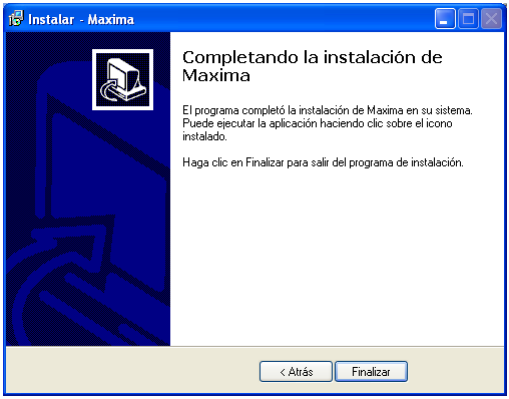


Figura 1.14: Cuadro Completando la Instalación de Maxima.

Figura 1.15: Icono de acceso directo a wxMaxima.

Capítulo 2

Funcionamiento de Maxima

2.1 Interfaz de cuaderno

utilice un icono o el menú de Inicio	formas gráficas de inicializar Maxima
finalizar texto con Shift + Enter	entrada para Maxima
elegir el ítem salida del menú	salir de Maxima

Funcionamiento de Maxima en una interfaz de cuaderno.

El acceso a una interfaz de "cuaderno" es factible en un ordenador usado vía una interfaz puramente gráfica (como Windows). En una interfaz de cuaderno, es posible interactuar con Maxima creando documentos interactivos. Para ello el usuario debe hacer doble clic en el icono de inicio de wxMaxima, después de lo cual se desplegará un cuaderno en blanco. En este cuaderno el usuario digita la entrada (input), luego presiona (en simultáneo) las teclas Shift + Enter para que Maxima procese su entrada. Shift + Enter indica a Maxima que el usuario ha finalizado su entrada.

Después de ejecutar una entrada en Maxima desde un cuaderno, Maxima etiquetará la entrada con (%i*n*) y después de presionar Shift + Enter, devolverá la correspondiente salida etiquetada con (%o*n*).

Maxima

El usuario digita 1+1, luego finaliza su entrada con Shift + Enter. Maxima procesa la entrada, después etiqueta la entrada con (%i1), y devuelve la respectiva salida etiquetada con (%o1).

```
(%i1)  1+1;
(%o1)  2
```

Debe recordarse que los cuadernos corresponden al entorno gráfico wxMaxima. El núcleo de Maxima es el que realiza realmente los cálculos (sección 3.1).

Para salir de wxMaxima, el usuario elige el ítem salida del respectivo menú en la interfaz de cuaderno.

2.2 Interfaz basada en texto

`maxima`	comando del sistema operativo para inicializar Maxima
finaliza r t con entrada	entrada para Maxima
; + Enter	
`quit();`	salir de Maxima

Funcionamiento de Maxima en una interfaz basada en texto.

Con una interfaz basada en texto, el usuario interactúa con su ordenador digitando texto mediante el teclado.

Para inicializar Maxima en una interfaz basada en texto, se digita el comando `maxima` en el prompt del sistema operativo.

Cuando Maxima ha inicializado, imprimirá el prompt `(%i1)`, esto significa que esta lista para que el usuario haga s u entrada. Éste puede entonces digitar su entrada, terminándola con ; + Enter.

Maxima procesa la entrada y genera un resultado, el mismo que etiquetará con `(%o1)`.

Obsérvese que la mayor parte de los diálogos dados en el libro muestran salidas en la forma que se obtendrían con una interfaz de

cuaderno de Maxima; la salida con una interfaz basada en texto luce similar, pero carece de características tales como caracteres especiales y cambio de tamaño de fuente.

Para salir de Maxima, debe digitarse `Quit();` en el prompt de la entrada.

Capítulo 3

Uso del sistema Maxima

3.1 La estructura de Maxima

Maxima	núcleo responsable de todos los cálculos
wxMaxima	interfaz de cuaderno que se ocupa de interactuar con el usuario (muy amigable)
XMaxima	interfaz gráfica que se ocupa de interactuar con el usuario (menos amigable que wxMaxima)

Partes básicas del Sistema Maxima.

Maxima es un potente motor de cálculo simbólico aunque, en su origen, no destacaba por tener una interfaz gráfica más amigable para los usuarios que la simple consola de texto. Con el tiempo este hecho ha ido cambiando y han aparecido distintos entornos de ejecución que intentan facilitar la interacción con los usuarios. Entre ellos, están XMaxima y wxMaxima.

XMaxima es la primera interfaz gráfica que fue desarrollada, es mantenida "oficialmente" por el equipo de desarrollo de Maxima. En Windows se instala automáticamente. Presenta algunas ventajas como la integración en formato HTML de manuales de ayuda. Sin embargo, también tiene algunas desventajas con respecto a otras inter-

faces más modernas.

wxMaxima[1], basada en la biblioteca gráfica wxwidgets, gracias a la cual existen versiones nativas tanto para sistemas operativos GNU/Linux como para Windows. Integra elementos específicos para la navegación de la ayuda, introducción de matrices, creación de gráficas, cálculo de límites, derivadas o integrales, etc. Actualmente también se instala automáticamente en Windows.

Maxima

Un cuaderno que mezcla texto, gráficos con entradas y salidas de Maxima.

1 Ejemplos de integrales

He aquí un ejemplo de una integral algebraica muy simple:

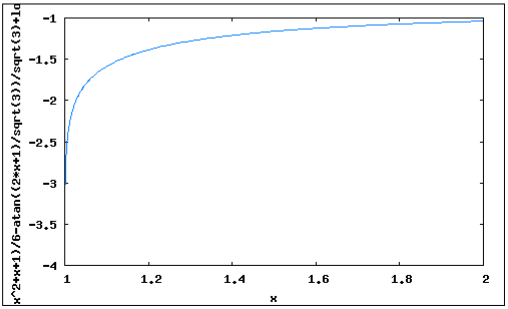
```
(%i1) integrate(1/(x^3-1),x);
```

$$(\%o1)\quad \frac{\log(x^2+x+1)}{6}-\frac{\operatorname{atan}\left(\frac{2x+1}{\sqrt{3}}\right)}{\sqrt{3}}+\frac{\log(x-1)}{3}$$

Y he aquí un gráfico de la función resultante:

```
(%i2) wxplot2d(%,[x,1,2]);
plot2d: expression evaluates to non-numeric value somewhere
```

(%t2)



(%o2)

interfaz de cuaderno con wxMaxima	documentos interactivos
interfaz basada en texto	texto desde el teclado

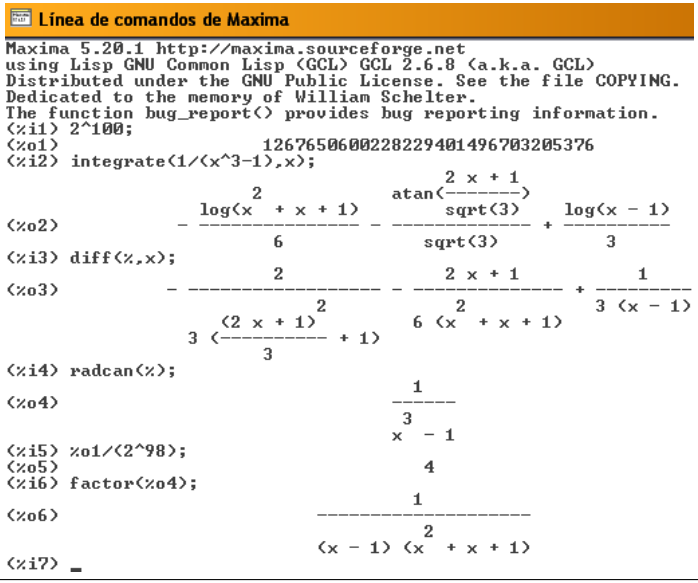
Tipos comunes de interfaz con Maxima.

[1] wxMaxima fue desarrollada por Andrej Vodopivec y está disponible en http //wxmaxima.sourceforge.net

En algunos casos, puede que el usuario no necesite usar la interfaz de cuaderno, y que desee en cambio interactuar directamente con el núcleo de Maxima. Es posible hacer esto usando la interfaz basada en texto, en la cual se digita el texto en el teclado y éste va directamente al núcleo.

Maxima

Un diálogo con Maxima usando una interfaz basada en texto.



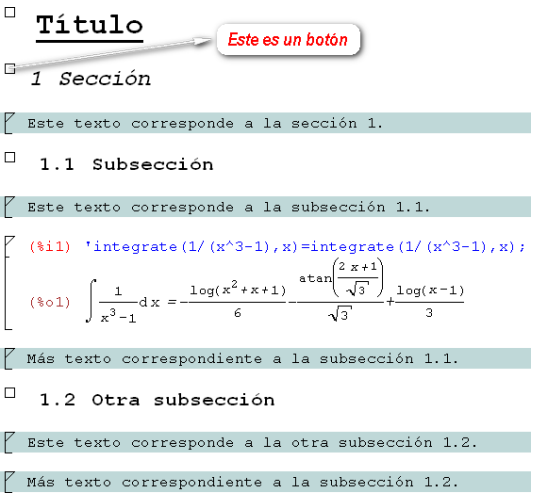
3.2 Cuadernos como documentos

Los cuadernos de wxMaxima permiten crear documentos que pueden verse interactivamente en la pantalla o imprimirse en papel. En los cuadernos extensos, es común tener los capítulos, secciones etc., representados cada uno en grupos de celdas. La extensión de estos grupos de celdas es indicada por el botón asociado a la celda dominante que es una celda de estilo título, sección o subsección.

Un grupo de celdas puede estar abierto o cerrado. Cuando está abierto se puede ver todas sus celdas explícitamente. Pero cuando está cerrado, sólo puede verse la celda que encabeza el grupo de celdas.

Maxima

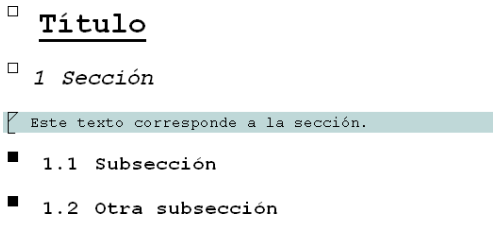
Un cuaderno de wxMaxima como documento.



Los cuadernos extensos son a menudo distribuidos con muchos grupos de celdas cerradas, para que cuando sean vistos por primera vez el cuaderno muestre solamente una lista de su contenido. Es posible abrir las partes en las que el usuario esté interesado haciendo clic sobre el botón apropiado.

Maxima

Haciendo clic sobre el botón que corresponde a la celda dominante se cierra el grupo, dejando sólo la primera celda visible.



Maxima

Cuando el grupo está cerrado, el botón que le corresponde aparece relleno en color negro. Haciendo clic sobre este botón se abre nuevamente el grupo.

Título

1 Sección

Este texto corresponde a la sección 1.

1.1 Subsección

Este texto corresponde a la subsección 1.1.

```
(%i1) 'integrate(1/(x^3-1),x)=integrate(1/(x^3-1),x);
```

$$(\%o1)\quad \int \frac{1}{x^3-1}\,dx = -\frac{\log(x^2+x+1)}{6} - \frac{\operatorname{atan}\left(\frac{2x+1}{\sqrt{3}}\right)}{\sqrt{3}} + \frac{\log(x-1)}{3}$$

Más texto correspondiente a la subsección 1.1.

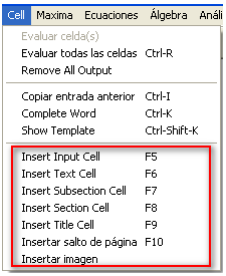
1.2 Otra subsección

A cada celda dentro de un cuaderno se le asigna un estilo en particular que indica su rol dentro del cuaderno.

La interfaz de wxMaxima provee menúes y métodos abreviados de teclas para insertar celdas con diferentes estilos todos ellos están disponibles en el último bloque del menú Ce .

Maxima

El recuadro muestra los menúes y métodos abreviados de teclas para insertar celdas con diferentes estilos.



Maxima

Esto muestra celdas en diferentes estilos. Los estilos no sólo definen el formato del contenido de las celdas, sino que también su ubicación y espaciado.

Esta celda está en estilo Título

1 Esta celda está en estilo Sección

1.1 Esta celda está en estilo Subsección

```
Esta celda está en estilo Texto.
```

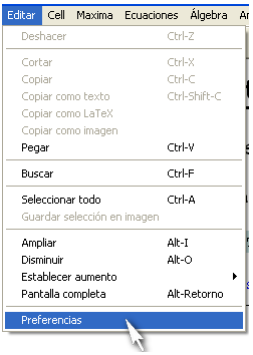
```
--> esta celda está en estilo Input
```

3.3 Configuración de opciones y estilos

Como se vio en la sección 3.2 los cuadernos pueden editarse a manera de documentos. wxMaxima incorpora una configuración predefinida para los estilos de los títulos, secciones, etc. Sin embargo, es posible cambiar algunos aspectos de dicha configuración haciendo clic en la opción Preferencias del menú Editar.

Maxima

Primer paso para configurar las opciones y estilos.



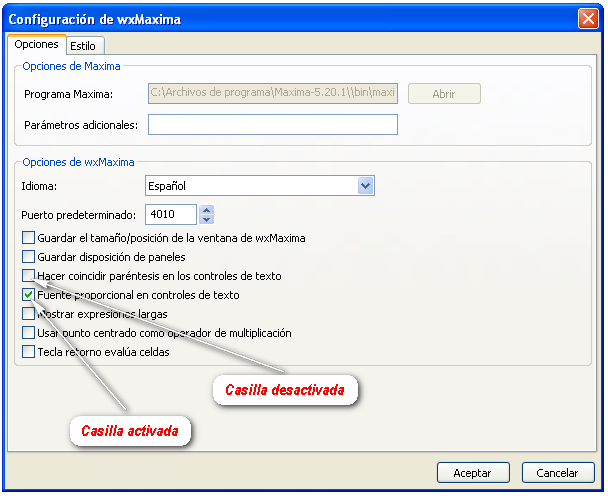
Después de hacer clic en la opción Preferencias se despliega la ventana Configuración de wxMaxima que incorpora dos pestañas: Opciones y Esti o.

Por ejemplo, cuando está activa la casilla de verificación de la opción Hacer coincidir os paréntesis en os contro es de texto (de la pestaña Opciones), Maxima cierra automáticamente cualquier paréntesis que se abra en una celda de estilo Input. Al desactivar esta casilla, y hacer clic en Aceptar, Maxima no volverá a cerrar automáticamente ningún paréntesis sino que esperará a que el usuario lo haga.

En la pestaña Esti o se presenta una lista de todos los estilos que pueden configurarse a gusto del usuario y la forma de hacerlo es bastante intuitiva.

Maxima

Activando o desactivando las casillas de verificación se cambia la configuración de las opciones.



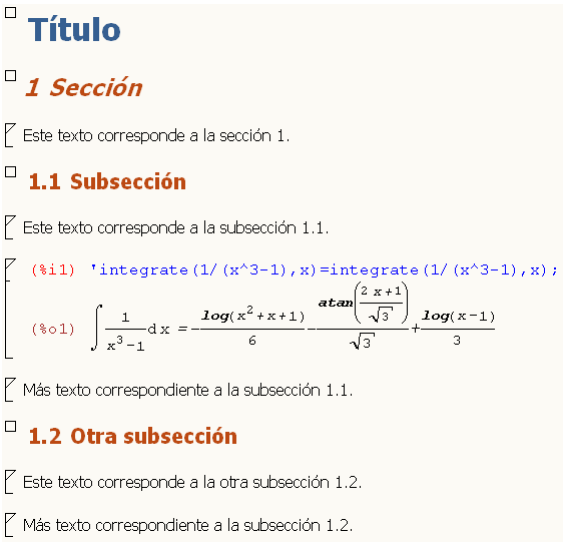
Por ejemplo, configurando los estilos con los valores indicados en las siguientes tablas se obtienen cuadernos con un aspecto elegante.

Fuentes	Tipo
Fuente predeterminada	Courier New (12)
Fuente matemática	Courier New (12)

Estilos	Color	Fuente	Aspecto	Tam.
Nombre de funciones	rgb(0,0,0)	Courier New	Gruesa, Itálica	12
Celda de texto	rgb(0,0,0)	Tahoma	Normal	12
Celda de subsección	rgb(l88,73,l8)	Tahoma	Gruesa	16
Celda de sección	rgb(l88,73,l8)	Tahoma	Gruesa	18
Celda de título	rgb(54,95,l45)	Tahoma	Gruesa	24
Fondo de celda de texto	rgb(252,250,245)			
Fondo	rgb(252,250,245)			

Maxima

Un cuaderno de wxMaxima como documento, después de haber editado la configuración de estilo.



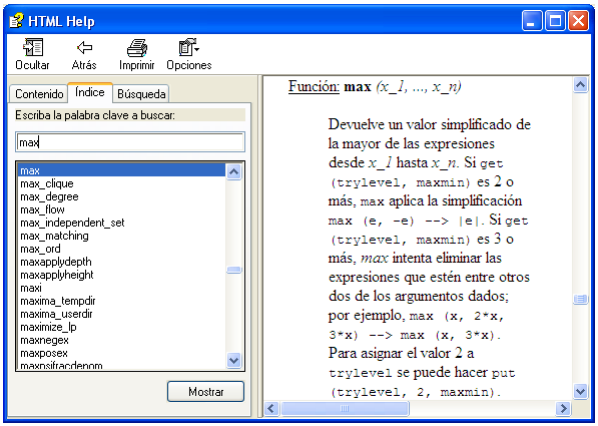
3.4 Búsqueda de ayuda

Todas las funciones incorporadas en Maxima están descritas en el manual en línea del usuario, el cual puede ser consultado en diferentes formas. La más usada es desde el menú Ayuda, de la barra de menúes, que da acceso a la opción Ayuda de Maxima la cual sirve como un

punto de entrada a la gran cantidad de documentación en línea para Maxima.

Maxima

Un ejemplo de búsqueda de información básica sobre una función en el Índice de la Ayuda de Maxima



Para buscar ayuda desde un cuaderno de trabajo puede utilizarse la función `describe`[2].

`describe`(string)	encuentra el elemento, si existe, cuyo título coincide exactamente con string (ignorando la diferencia entre mayúsculas y minúsculas)
`describe`(string)exact)	equivale a `describe`(string)
`describe`(string)inexact)	encuentra todos los elementos documentados que contengan string en sus títulos

Sintaxis de la función `describe`, la cual permite recibir información de las funciones de Maxima.

[2] `describe` no evalúa su argumento. La función `describe` devuelve `true` si encuentra la documentación solicitada y `false` en caso contrario.

?name	equivale a `describe("name")`
??name	equivale a `describe("name", inexact)`

Otras formas de recibir información.

Maxima

Esta sentencia da información de la función incorporada `max`.

```
(%i1) describe("max");
- -Función: max(< x_1 >, . . . , < x_n >)

    Devuelve un valor simplificado de la mayor de las expresiones
    desde < x_1 > hasta < x_n >. Si 'get(trylevel, maxmin)' es
    2 o más, 'max' aplica la simplificación 'max(e, -e) -> |e|'. Si
    'get(trylevel, maxmin)' es 3 o más, 'max' intenta eliminar las
    expresiones que estén entre dos de los argumentos dados; por
    ejemplo, 'max(x, 2*x, 3*x) -> max(x, 3*x)'. Para asignar el
    valor 2 a 'trylevel' se puede hacer 'put(trylevel, 2, maxmin)'.

There are also some inexact matches for 'max'.

Try '??max' to see them.

(%o1) true
```

Maxima

Esta sentencia encuentra todos los elementos documentados que contienen "plus" en sus títulos. No devuelve `true` o `false` hasta que el usuario seleccione las opciones que desee consultar (aquí las opciones disponibles son 0,l,2,3, all y none).

```
(%i2) describe("plus", inexact);
0: doscmxplus (Funciones y variables para las matrices y el álgebra
lineal).

1: poisplus (Series de Poisson)

2: region_boundaries_plus (Funciones y variables para worldmap)

3: trigexpandplus (Funciones y variables para trigonometría)

Enter space-separated numbers, 'all' or 'none':
```

Maxima

Una vez elegidas las opciones (en este caso 0 y 2) la sentencia devuelve `true`.

```
(%i2) describe("plus", inexact);
0: doscmxplus (Funciones y variables para las matrices y el álgebra
lineal).
1: poisplus (Series de Poisson)
2: region_boundaries_plus (Funciones y variables para worldmap)
3: trigexpandplus (Funciones y variables para trigonometría)
Enter space-separated numbers, 'all' or 'none': 0 2;
--Variable opcional: doscmxplus
    Valor por defecto: 'false'.
    Cuando 'doscmxplus' vale 'true', las operaciones entre escalares
    y matrices dan como resultado una matriz.
--Función: region_boundaries_plus(< x1 >, < y1 >, < x2 >, < y2 >)
    Detecta los segmentos poligonales almacenados en la variable
    global
    'boundaries_array' con al menos un vértice dentro del rec-
    tángulo definido por los extremos (< x1 >, < y1 >) -superior
    izquierdo- y (< x2 >, < y2 >) -inferior derecho-.
    Ejemplo.
        (%i1) load(worldmap)$;
        (%i2) region_boundaries(10.4,41.5,20.7,35.4);
        (%o2) 1846, 1863, 1864, 1881, 1888, 1894J
        (%i3) draw2d(geomap(%))$
(%o2)  true
```

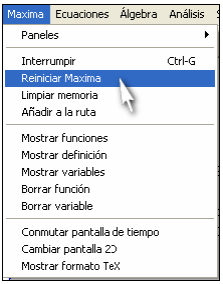
3.5 Reinicio

La forma brusca de reiniciar Maxima es saliendo de Maxima. No obstante, en muchos casos resulta útil reiniciar Maxima sin salir de

Maxima. Para reiniciar Maxima sin salir de Maxima se elige la opción Reiniciar Maxima del menú Maxima.

Maxima

Reiniciando Maxima en una interfaz de cuaderno.



3.6 Comentarios

Los comentarios son toda una serie de caracteres que no afectan los cálculos. En Maxima los comentarios se escriben entre las marcas /* y */.

/*comentario/*	con esta sintaxis comentario es interpretado como un comentario

Escribiendo comentarios.

Maxima

Aquí se muestra un cálculo y un comentario.

```
(%i3)  4+5 /*esto es una suma*/;
(%o3)  9
```

3.7 Paquetes en Maxima

Una de las características más importantes de Maxima es que es un sistema extensible, hay una cierta cantidad de funciones incorporadas

en Maxima pero, usando el lenguaje de programación de Maxima, siempre es posible añadir más funciones.

Para muchos tipos de cálculos, lo incorporado en la versión estándar de Maxima será suficiente. Sin embargo, si se trabaja en particular en un área especializada, es posible encontrarse en la necesidad de utilizar ciertas funciones no incorporadas en Maxima.

En tales casos, podría ser factible encontrar (o elaborar) un package (paquete) de funciones de Maxima que contenga las funciones que sean necesarias.

`load("paquete")`	lee un paquete de Maxima

Leyendo paquetes de Maxima.

Si el usuario quiere usar las funciones de un paquete en particular, primero debe inicializar el paquete en Maxima.

Maxima

Con estas sentencias se está inicializando y utilizando una función de un paquete en particular de Maxima.

```
(%i4) load("simplex");
(%i5) minimize_lp(x+1, *x+2* 2, x+4* );
```

(%o5) $[\frac{9}{10}, [y = \frac{7}{10}, x = \frac{1}{5}]]$

El hecho de que Maxima pueda extenderse usando paquetes significa que las posibilidades de Maxima son ilimitadas. En lo que al uso concierne, no hay en realidad ninguna diferencia entre las funciones definidas en paquetes y las funciones incorporadas en Maxima.

3.8 Advertencias y mensajes

Maxima "sigue" el trabajo del usuario silenciosamente, dando salida solamente cuando éste lo requiere. Sin embargo, si Maxima se percata de algo que se pretende hacer y que definitivamente no entiende, imprimirá un mensaje de advertencia.

Maxima

La función para calcular la raíz cuadrada debe tener solamente un argumento. Maxima imprime un mensaje para advertir que, en este caso, se ha errado en el número de argumentos.

```
(%i6) sqrt(4,5);
Wrong number of arguments to sqrt
- - an error. To debug this try: debugmode(true);
```

3.9 Interrupción de cálculos

Probablemente habrá veces en que el usuario desee detener Maxima en medio de un cálculo. Tal vez él se da cuenta que pidió a Maxima hacer un cálculo incorrecto. O quizás el cálculo tarda demasiado, y quiere saber que es lo que pasa. La forma en que se interrumpe un cálculo en Maxima depende de qué clase de interfaz está utilizando.

Formas de interrumpir cálculos en Maxima

Clic en el botón 	interfaz de cuaderno
Ctr + C	interfaz basada en texto

Capítulo 4

Cálculos numéricos

4.1 Aritmética

El usuario puede hacer aritmética con Maxima tal y como lo haría con una calculadora

Maxima

Aquí tenemos la suma de dos números.

```
(%i1)  5.6+.7;
(%o1)  9.
```

Maxima

Con ∗ indicamos el producto de dos números.

```
(%i2)  5.6*.7;
(%o2)  20.72
```

Maxima

Es posible digitar operaciones aritméticas haciendo uso de los paréntesis.

```
(%i3)  (2+)-4*(6+7);
(%o3)  7
```

x~y ó x**y	potencia
−x	menos
x/y	división
x*y*z	producto
x+y+z	suma

Operaciones aritméticas en Maxima.

Las operaciones aritméticas en Maxima se agrupan de acuerdo con las convenciones estándares de la matemática. Como es usual, `2+ /7`, por ejemplo, significa `2+( /7)`, y no `(2+ )/7`. El usuario siempre puede controlar la forma de agrupar explícitamente usando los paréntesis.

4.2 Resultados exactos y aproximados

Una calculadora electrónica hace todos sus cálculos con una precisión determinada, digamos de diez dígitos decimales. Con Maxima, en cambio, es posible obtener resultados exactos.

Maxima

Maxima da un resultado exacto para 2^{300}.

```
(%i4)  2  00;
(%o4)  20 70 5976 4486086268445688409 78161051468 9 6
       659 62506 6140449 54 8129976 670618 97 76
```

El usuario puede pedir a Maxima que devuelva un resultado aproximado, tal como lo daría una calculadora, para ello puede usar la función `float` o la variable `numer` o una combinación de ambos.

Maxima

Esto da un resultado numérico aproximado.

```
(%i5)  2 00, float;
(%o5)  2,0370359763344861 10^90
```

float(expr)	da un valor numérico aproximado para ciertas expr
expr, float	equivale a float(expr)
expr, numer	da un valor numérico aproximado para ciertas expr
float(expr), numer	da un valor numérico aproximado para cualquier expr que no sea una constante

Obteniendo aproximaciones numéricas.

Maxima

Esta forma también da un resultado numérico aproximado.

```
(%i6)  float(2^^90);
(%o6)  2,0370359763344861 10^90
```

Maxima

Para el cálculo previo la constante `numer` no es útil.

```
(%i7)  2^^90, numer;
(%o7)  20 70 5976 4486086268445688409 78161051468 9 6
       659 62506 6140449 54 8129976 670618 97 76
```

Maxima

Maxima puede dar resultados en términos de números racionales.

```
(%i8)  1/ +2/7;
(%o8)  13/21
```

Maxima

En este caso, tanto con `float` como con `numer`, se obtiene un resultado numérico aproximado.

```
(%i9)  1/ +2/7, float;
(%o9)  0. 61904761904762
```

```
(%i10)  1/ +2/7, numer;
(%o10)  0.61904761904762
```

Cuando el usuario digita un entero como 7, Maxima asume que es exacto. Si digita un número como 4.5, con un punto decimal explícito, Maxima asume que desea efectuar cálculo numérico aproximado.

Maxima

Esto es tomado como un número racional exacto, y es llevado a una fracción irreducible.

```
(%i11)  26/78;
(%o11)  1/3
```

Maxima

Cuando el usuario digita un número con un punto decimal explícito, Maxima produce un resultado numérico aproximado.

```
(%i12)  26.7/78;
(%o12)  0.  42  07692  0769
```

Maxima

Aquí, la presencia del punto decimal seguido del cero hace que Maxima dé un resultado numérico aproximado.

```
(%i13)  26.0/78;
(%o13)  0.
```

Maxima

Cuando cualquier número en una expresión aritmética es digitado con un punto decimal seguido del cero, el usuario obtiene un resultado numérico aproximado.

```
(%i14)  5.0+9/78-5/8;
(%o14)  4.490  84615  84615
```

4.3 Algunas funciones matemáticas

Maxima incluye una gran colección de funciones matemáticas. A continuación se mencionan las más comunes.

sqrt(x)	raíz cuadrada ($\sqrt{x}$)
exp(x)	exponencial (e^x)
log(x)	logaritmo neperiano ($\log_e x$)
sin(x), cos(x), tan(x), cot(x), sec(x), csc(x)	funciones trigonométricas (con argumentos en radianes)
asin(x), acos(x), atan(x), acot(x), asec(x), acsc(x)	funciones trigonométricas inversas
n!	factorial de n (producto de los enteros $1, 2, \ldots, n$)
n!!	$1 \times 3 \ldots \times n$ (n impar) ó $2 \times 4 \ldots \times n$ (n par)
abs(x)	valor absoluto
round(x)	redondeo
mod(n,m)	n módulo m (resto de la división de n entre m)
floor(x)	mayor entero de x
random(x)	número seudo aleatorio r, tal que $0 \leq r < x$, si $x \in \mathbb{N}$ ó $0 < r < x$, si $x \in \mathbb{R}^+$
max(x,y,...), min(x,y , . . .)	máximo, mínimo de x,y,. . .
ifactor(n)	factores primos de n

Algunas de las funciones matemáticas más comunes.

- Los argumentos de todas las funciones en Maxima se colocan entre paréntesis.
- Los nombres de las funciones incorporadas en Maxima empiezan con letra minúscula.

Dos puntos importantes acerca de funciones en Maxima.

Maxima

Esto da $\log_e 15{,}7$.

```
(%i15)  log(15.7);
(%o15)  2.75660712 54262
```

Maxima

Maxima no tiene definida una función para el logaritmo de base l0 u otras bases. Para salvar esta dificultad el usuario puede hacer uso de la igualdad matemática $\log_b x = \frac{\log_e x}{\log_e b}$. Así, por ejemplo, lo siguiente devuelve un resultado numérico para $\log_2 1024$.

```
(%i16)  log(1024)/log(2),numer;
(%o16)  10.0
```

Maxima

Esto devuelve $\sqrt{64}$ como un número exacto.

```
(%i17)  sqrt(64);
(%o17)  8
```

Maxima

Esto da un valor numérico aproximado para $\sqrt{6}$.

```
(%i18)  sqrt(6),numer;
(%o18)  2.44948974278 178
```

Maxima

La presencia explícita de un punto decimal seguido de un cero le indica a Maxima que dé un resultado numérico aproximado.

```
(%i19)  sqrt(6.0);
(%o19)  2.44948974278 178
```

Maxima

En este caso Maxima devuelve un número en forma simbólica exacta.

```
(%i20)  sqrt(6);
(%o20)  √6
```

Maxima

Aquí tenemos un resultado entero exacto para $40 \times 39 \times \ldots \times 1$.

```
(%i21)  40!;
(%o21)  81591528 2478977 4 45611269596115894272000000000
```

Maxima

Esto da un valor numérico aproximado del factorial.

```
(%i22)  float(40!);
(%o22)  8,1591528324789768 10^47
```

%e	$e \approx 2{,}718281828459045$
%i	$\sqrt{-1}$
inf	representa al infinito real positivo
minf	representa al infinito real negativo
infinity	representa al infinito complejo
und	representa un resultado indefinido
%pi	$\pi \approx 3{,}141592653589793$

Algunas constantes matemáticas comunes.

Maxima

Este es el valor numérico de π^2.

```
(%i23)  %pi^2, numer;
(%o23)  9.869604401089358
```

Maxima

Esto devuelve el valor exacto para sen($\pi/2$).

```
(%i24)  sin(%pi/2);
(%o24)  1
```

4.4 Cálculos con precisión arbitraria

Cuando el usuario utiliza `float`, `numer` o una combinación de ambos para obtener un resultado numérico, Maxima devuelve el resultado con un número fijo de cifras significativas. No obstante, es posible indicar a Maxima las cifras significativas con las que se desea operar. Esto permite obtener resultados numéricos en Maxima con cualquier grado de precisión.

fpprec :n$ **bfloat**(expr) ó **fpprec** :n$ expr, **bfloat**	valor numérico de expr calculado con n dígitos de precisión (el valor por defecto es 16)

Evaluación numérica con precisión arbitraria.

Maxima

Esto devuelve el valor numérico de π con un número fijo de cifras significativas.

```
(%i25)  float(%pi);
(%o25)  .14159265 58979
```

Maxima

Esto devuelve π con 50 dígitos.

```
(%i26)  fpprec : 50$ bfloat(%pi);
(%o27)  .14159265 58979 2 846264 8 279502884197169 99
          751b0
```

Cabe mencionar que el símbolo de dolar que aparece después del número que indica la cantidad de dígitos significativos se utiliza, en general, para finalizar una sentencia y, a diferencia del punto y coma, no permite que aparezca ninguna salida en pantalla (subsec. 5.3).

Al realizar cualquier tipo de cálculo numérico el usuario puede introducir pequeños errores de redondeo en sus resultados. Cuando se aumenta la precisión numérica estos errores se hacen más pequeños. Asegurarse que se obtiene la misma respuesta al aumentar la precisión numérica es a menudo una buena forma de verificar los resultados.

Maxima

La cantidad $e^{\pi\sqrt{163}}$ esta bastante próxima a ser entera. Para verificar que el resultado no es, de hecho, un entero, usted tiene que usar la precisión numérica suficiente.

```
(%i28)  fpprec : 40$ bfloat(exp( %pi*sqrt(163)));
(%o29)  2.625 741264076874 9999999999992500725972b17
```

El usuario que desee, por ejemplo, visualizar π con una precisión de 200 cifras significativas, o 2^{1000} con el total de las cifras, utilizando las sentencias aquí descritas encontrará que la salida se muestra truncada en la parte central, en la cual aparece un número que indica la cantidad de dígitos faltantes.

Maxima

Esto devuelve una salida de π con 200 dígitos, la cual ha sido truncada. El dato entre los corchetes indica que se han obviado 443 dígitos

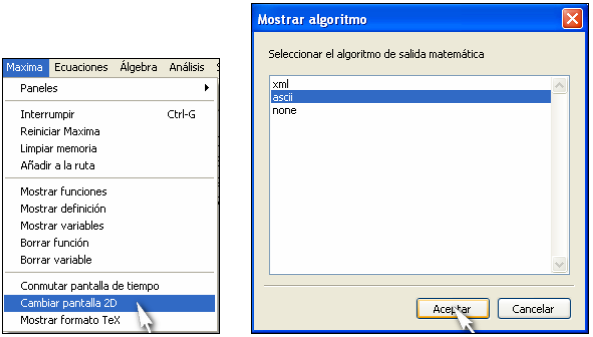
```
(%i30)  fpprec:200$bfloat(%pi);
(%o31)  .14159265 58979 2 846264 8 2 44 digits 88575
        27248912279 818 0119491b0
```

Para obtener una salida completa se selecciona ascii como el algoritmo de salida matemática de la opción Cambiar panta a 2D del menú Maxima y luego se ejecutan las sentencias de (%i30) .

Téngase presente que el algoritmo de salida matemática, previamente seleccionado (ascii), prevalecerá hasta que el usuario vuelva a seleccionar como algoritmo de salida matemática a xm .

Maxima

Seleccionando ascii como el algoritmo de salida.



Maxima

Después de cambiar el algoritmo de salida matemática se devuelve la salida π con 200 dígitos.

```
(%i32)  fpprec: 200$ bfloat( %pi);
(%o33)  .14159265 58979 2 846264  8 279502884197169  99
 75105820974944592 078164062862089986280 4825 4
211706798214808651 282 0664709 8446095505822  17
25 594081284811174502841027019  8521105559644622
9489549 0 82b0
```

4.5 Números complejos

Es posible ingresar números complejos en **Maxima** con sólo incluir la constante **%i**, igual a $\sqrt{-1}$.

Maxima

Esto devuelve como resultado el número imaginario $2i$.

```
(%i34)  sqrt(-4);
(%o34)  2 %i
```

Maxima

Esto devuelve la división de dos números complejos.

```
(%i35)  (8 + 4 * %i)/(-1 + %i), rectform;
(%o35)  -6 %i - 2
```

Maxima

Aquí tenemos el valor numérico de un exponencial complejo.

```
(%i36)  exp(11 + 5 * %i), numer;
(%o36)  16984,02989166794 - 57414,76791532402 %i
```

`x+%i*y`	el número complejo $x + i\,y$
`realpart(z)`	parte real
`imagpart(z)`	parte imaginaria
`conjugate(z)`	complejo conjugado z^* ó $\overline{z}$
`cabs(z)`	módulo de z
`carg(z)`	el argumento ϕ en $\lvert z\rvert e^{i\phi}$

Operaciones con números complejos.

Capítulo 5

Generación de cálculos

5.1 Uso de entradas y salidas previas

Al realizar cálculos, muchas veces se necesita usar expresiones previamente ingresadas u obtenidas. En Maxima, _ y % siempre hacen referencia a la última expresión de entrada y salida, respectivamente.

_	la última expresión de entrada
%	la última expresión de salida
%in	la expresión de la entrada %in
%on	la expresión de la salida %on
%th(i)	la expresión de la i-ésima salida anterior

Formas de hacer referencia a expresiones de entrada y salida previas.

Para efectos didácticos, a partir de esta sección se supone un reinicio de Maxima (sec. 3.5).

Maxima

Aquí se tienen las expresiones de la primera entrada y salida.

```
(%i1)  5   ;
(%o1)  125
```

Maxima

Esto agrega 6 a la expresión de la última salida.

```
(%i2)   %+6;
(%o2)   1  1
```

Maxima

Esto utiliza las dos expresiones de las salidas previas.

```
(%i3)   5+%o1+%;
(%o3)   261
```

Maxima

Esto suma las expresiones de las salidas **1** y 3.

```
(%i4)   %o1+%o3;
(%o4)   86
```

Maxima

Aquí se eleva al cuadrado la expresión de la salida 2.

```
(%i5)   %o2 2;
(%o5)   17161
```

Si se utiliza una interfaz basada en texto en Maxima, entonces las líneas sucesivas de entradas y salidas aparecerán siempre en orden. Sin embargo, si se utiliza una interfaz de cuaderno, varias líneas sucesivas de entradas y salidas no necesariamente aparecen en orden. Es posible, por ejemplo, "volver atrás" e insertar el cálculo siguiente dondequiera que se desee en el cuaderno. Téngase en cuenta que % siempre invoca el último resultado que Maxima generó. Éste puede o no ser el resultado que aparece inmediatamente encima de su actual posición en el cuaderno. Con una interfaz de cuaderno, la única manera de saber cuándo un resultado particular fue generado es mirar la etiqueta de (%o*n*) que tiene. Como es posible insertar y suprimir en

todas partes en un cuaderno, de acuerdo a la necesidad del usuario, el ordenamiento de los resultados, por lo general, no tiene ninguna relación con el orden en el cual los resultados fueron generados.

5.2 Definición de variables

Cuando se efectúan cálculos extensos, muchas veces es conveniente dar nombre a los resultados intermedios. De igual modo que en las matemáticas tradicionales o en otros lenguajes de programación, es posible hacer esto introduciendo variables con un nombre específico.

Maxima

Esto inicializa el valor de la variable x con 6.

```
(%i6)  x:6;
(%o6)  6
```

Maxima

Donde quiera que aparezca x, Maxima la reemplaza por su valor 6.

```
(%i7)  x  -25
(%o7)  191
```

Maxima

Esto asigna un nuevo valor para x.

```
(%i8)  x:11+5
(%o8)  16
```

Maxima

pi es inicializada con el valor numérico de π con 20 dígitos de exactitud.

```
(%i9)  fpprec:20$
(%i10)  pi : %pi, bfloat;
(%o10)  .14159265 58979 2 85b0
```

Maxima

Aquí esta el valor de sqrt(pi).

```
(%i11)  sqrt(pi)
(%o11)  1.77245 850905516027 b0
```

x : *valor*	asigna un valor a la variable *x*
x : *y* : *valor*	asigna un valor a las variable *x* e *y*
`kill(x)`	quita cualquier valor asignado a x
`kill(x , y)`	quita cualquier valor asignado a x e y
`values`	muestra las variables a las que se les ha asignado un valor

Manipulación de variables.

Es muy importante recordar que los valores asignados a las variables son permanentes. Una vez que el usuario ha asignado un valor a una variable en particular, el valor será almacenado hasta que éste lo remueva explícitamente. El valor, claro está, desaparecerá si el usuario inicia una nueva sesión con Maxima.

Olvidarse de las definiciones hechas es la causa más común de errores al usar Maxima. Si el usuario pusiera `x:5`, Maxima asume que éste siempre quiere que `x` tenga el valor `5`, hasta o a menos que se le indique explícitamente otra cosa. Para evitar los errores, deben quitarse los valores definidos en cuanto se haya terminado de usarlos.

- Quite valores que asigne a las variables en cuanto termine de usarlos.

Un principio útil al usar Maxima.

Maxima

Inicializando el valor de la variable y con 9.

```
(%i12)  :9;
(%o12)  9
```

Maxima

Aquí se muestran todas las variables que tienen un valor asignado.

```
(%i13)  values;
(%o13)  [x, π, %]
```

Maxima

Sentencia para quitar el valor asignado a la variable x.

```
(%i14)  kill(x);
(%o14)  done
```

Maxima

Sentencia para quitar el valor asignado a todas las variables.

```
(%i15)  kill(values);
(%o15)  done
```

Las variables que el usuario define pueden tener cualquier nombre. No hay límite para la longitud de sus nombres. Un inconveniente, sin embargo, es que los nombres de las variables nunca pueden empezar con números. Por ejemplo, x puede ser una variable, pero x corresponde a una sintaxis incorrecta en Maxima.

Maxima

He aquí cuatro variables diferentes.

```
(%i16)  EstoEsUnaVariable:4/ ;
(%o16)  4
(%i17)  Esto Es Una Variable:5 ;
(%o17)  125
(%i18)  estoesunavariable:8;
(%o18)  8
(%i19)  esto es una variable:sqrt(7);
(%o19)  √7
```

Maxima

Con `values` visualizamos las nuevas variables a las que se les ha asignado un valor.

```
(%i20) values;
(%o20) EstoEsUnaVariable, Esto Es Una Variable,
        estoesunavariable, esto es una variable
```

Maxima

Es posible realizar operaciones diversas con estas variables.

```
(%i21) (Esto Es Una Variable 2+esto es una variable*
       estoesunavariable)/EstoEsUnaVariable;
                    3(8√7 + 15625)
(%o21)              ──────────────
                           4
```

Maxima

Con `kill(all)` también se quita el valor asignado a todas las variables.

```
(%i22) kill(all);
(%o22) done
```

Maxima

Esta vez `values` no encuantra ninguan variable con valor asignado.

```
(%i23) values;
(%o23)
```

5.3 Secuencia de operaciones

Al realizar cálculos con Maxima, usualmente se lo hace mediante una secuencia de pasos. Si el usuario desea puede realizar cada paso en una línea separada. A veces, sin embargo, es conveniente ingresar varios pasos en una misma línea. Es posible hacer esto simplemente separando cada una de las partes ingresadas con punto y coma (si

quiere verse las salidas en pantalla) o con signo de dolar (si no quiere verse salida alguna).

$expr_1; expr_2; \ldots; expr_n;$	hace varias operaciones y da el resultado de todas ellas
$expr_1$ \$ $expr_2$ \$ . . . \$ $expr_n$ \$	hace varias operaciones y no muestra ningún resultado
$expr_1$ \$ $expr_2$ \$ . . . \$ $expr_n;$	hace varias operaciones y da el resultado de la última línea

Formas de hacer secuencias de operaciones en Maxima.

Maxima

Esto realiza tres operaciones en una misma línea y muestra todos los resultados.

```
(%i24)  x: ; :4; :x+ ;
(%o24)
(%o25)  4
(%o26)  7
```

Maxima

Esto realiza tres operaciones en una misma línea sin mostrar resultados.

```
(%i27)  x: $ :4 $ :x+ $
```

Maxima

Esto realiza tres operaciones en una misma línea y muestra el resultado de la última operación.

```
(%i30)  x: $ :4 $ :x+ ;
(%o32)  7
```

Si el usuario finaliza su entrada con un signo de dolar, esto es interpretado por Maxima como si estuviera ingresando una secuencia de operaciones con un signo de dolar al final; así que tiene el efecto

de hacer que Maxima calcule las operaciones especificadas, pero no muestre la salida.

expr $	realiza una operación, pero no muestra la salida

Inhibiendo la salida.

Maxima

Añadiendo un signo de dolar al final de la línea se le indica a Maxima que no muestre la salida.

```
(%i33)  x:47+5$
```

Maxima

Usando % se puede visualizar la salida anterior.

```
(%i34)  %
(%o34)  52
```

5.4 Impresión de expresiones sin evaluar

El operador comilla simple evita la evaluación. Aplicado a un símbolo, la comilla simple evita la evaluación del símbolo. Aplicado a la invocación de una función, la comilla simple evita la evaluación de la función invocada, aunque los argumentos de la función son evaluados (siempre y cuando la evaluación no se evite de otra manera). Aplicado a una expresión con paréntesis, la comilla simple evita la evaluación de todos los símbolos y llamadas a funciones que hayan en la expresión.

Maxima

Esto asigna valores a las variables a y b.

```
(%i35)  a:45$ b:56$
```

'a	evita la evaluación del símbolo a
'f (x)	evita la evaluación de la función f, pero no de sus argumentos
'(expr)	evita la evaluación de todos los símbolos y llamadas a funciones que hayan en la expresión expr

Evitando la evaluación.

Maxima

El operador comilla simple (') aplicado a la variable a evita la evaluación de ésta.

(%i37) 'a

(%o37) a

Maxima

El operador ' aplicado a la función `integrate` evita la evaluación de ésta.

(%i38) 'integrate(x^3, x);

(%o38) $\int x^3 dx$

Maxima

Un uso interesante del operador '.

(%i39) 'integrate(x^3, x)=integrate(x^3, x);

(%o39) $\int x^3 dx = \frac{x^4}{4}$

Maxima

El operador ' no evita la evaluación de los argumentos de una función.

(%i40) 'integrate((2+3)*x^3, x);

(%o40) $5 \int x^3 dx$

Maxima

El operador ' aplicado a una expresión.

(%i41)
```
'(2*sqrt(a)+b*integrate(x , x));
```

(%o41) $b\int x^3dx+2\sqrt{a}$

Maxima

Otro uso interesante del operador '.

(%i42)
```
'(2*sqrt(a)+b*integrate(x , x))=(2*sqrt(a)+
b*integrate(x , x));
```

(%o42) $b\int x^3dx+2\sqrt{a}=14x^4+6\sqrt{5}$

Capítulo 6

Cálculos algebraicos

6.1 Cálculo simbólico

Una de las características importantes de Maxima es que puede hacer cálculos simbólicos y numéricos. Esto significa que puede manejar fórmulas algebraicas así como números.

Maxima

He aquí un típico cálculo numérico.

```
(%i1)  4+36-1;
(%o1)  39
```

Maxima

Este es un cálculo simbólico.

```
(%i2)  7*x-3*x+6;
(%o2)  4 x+6
```

Cálculo numérico	$4 + 36 - 1 \rightarrow 39$
Cálculo simbólico	$7x - 3x + 6 \rightarrow 4x + 6$

Cálculo simbólico y numérico.

Maxima

El usuario puede digitar cualquier expresión algebraica en Maxima.

```
(%i3)  x^3+2*x-1;
(%o3)  x^3+2*x-1
```

Maxima

Maxima realiza automáticamente simplificaciones algebraicas básicas. Aquí combina a x^2 y a $-4x^2$ para dar $-3x^2$.

```
(%i4)  x^2+x-4*x^2;
(%o4)  x-3*x^2
```

Es posible digitar cualquier expresión algebraica usando los operadores enumerados en la sección 4.1. No debe olvidarse ingresar el asterisco para el producto, por ejemplo: x*y, de lo contrario Maxima asumirá que se trata de una sola variable.

Maxima

Maxima reordena y combina términos usando las reglas estándares del álgebra.

```
(%i5)  x*y+2*x^2*y+y^2*x^2-2*y*x;
(%o5)  x^2*y^2+2*x^2*y-x*y
```

Maxima

He aquí otra expresión algebraica.

```
(%i6)  (x+2*y+1)*(x-2)^2;;
(%o6)  (x-2)^2*(2*y+x+1)
```

Maxima

La función `expand` amplía productos y potencias.

```
(%i7)  expand(%);
(%o7)  2*x^2*y-8*x*y+8*y+x^3-3*x^2+4
```

Maxima

`factor` hace lo inverso de `expand`.

```
(%i8)  factor(%);
(%o8)  (x − 2)^2 (2y + x + 1)
```

Cuando se digita expresiones complicadas, es importante poner los paréntesis en los lugares correctos. Así, por ejemplo, debe dar la expresión x^{4y} en la forma `x^(4y)`. Si no colocan los paréntesis, se interpretará como $x^4 y$.

Maxima

He aquí una fórmula más complicada, que requiere de varios paréntesis.

```
(%i9)  sqrt(2)/9801 * (4 * n)! * (1103 + 26390 * n)/(n!^4 + 1);
                √2 (26390 n + 1103) (4 n)!
(%o9)  ---------------------------------------
                      9801 (n!^4 + 1)
```

Cuando el usuario digita una expresión, Maxima aplica automáticamente su gran repertorio de reglas para transformar las expresiones. Estas reglas incluyen las reglas estándares del álgebra, tales como $x - x = 0$, junto con reglas mucho más sofisticadas.

Maxima

Maxima utiliza reglas estándares del álgebra para sustituir $(\sqrt{x} + 1)^4$ por $(x+1)^2$.

```
(%i10)  sqrt(x+ 1)^4;
(%o10)  (x+ 1)^2
```

Maxima

Maxima no conoce ninguna regla para esta expresión, así que deja expresión en la forma original que usted le dio.

```
(%i11)  log(cos(x) + 1);
(%o11)  log(cos x + 1)
```

6.2 Valores para símbolos

Cuando Maxima transforma una expresión por ejemplo `x + x` en `2 x`, está tratando la variable `x` en forma puramente simbólica o formal. En tales casos, `x` es un símbolo que puede representar cualquier expresión.

A menudo, sin embargo, se necesita sustituir un símbolo como `x` por un "valor" determinado. Algunas veces este valor será un número; aunque puede que sea una expresión.

Para sustituir el símbolo `x`, que aparece en la expresión `1 + 2x`, con un valor determinado; el usuario puede utilizar la función `ev` o una sintaxis alternativa de la misma.

ev (expr) x=valor)	reemplaza x por valor en la expresión exp
ev (expr) x=valor) y=valor)	realiza varios reemplazos
expr) x=valor	reemplaza x por valor en la expresión exp
expr) x=valor) y=valor	realiza varios reemplazos

Sustitución de símbolos por valores en expresiones.

Maxima

Esto realiza la regla de sustitución x = 3 en la expresión 1 + 2x.

```
(%i12)  1 + 2*x, x = 3;
(%o12)  7
```

Maxima

También es posible sustituir x por cualquier expresión. Aquí cada ocurrencia de x es sustituida por 2 − y.

```
(%i13)  1 + x + x^2, x = 2 - y;
(%o13)  -y + (2 - y)^2 + 3
```

Maxima

Maxima trata las reglas de sustitución como cualquier otra expresión simbólica.

```
(%i14)  x = y + 3;
(%o14)  x = y + 3
```

Maxima

Esto aplica la regla de sustitución última a la expresión $x^2 - 9$.

```
(%i15)  x^2 - 9, %;
(%o15)  (y + 3)^2 - 9
```

Maxima

Es posible aplicar varias reglas de sustitución juntas.

```
(%i16)  (x + y) * (x - y)^2, x = 3, y = 1 - a;
(%o16)  (4 - a) (a + 2)^2
```

La función `ev` o su sintaxis alternativa, permiten aplicar reglas de sustitución a una expresión particular. A veces, sin embargo, se querrá definir reglas de sustitución que se apliquen siempre. Por ejemplo, puede ser que se desee sustituir `x` por `3` siempre que aparezca `x`. Según lo discutido en la sección 5.2, puede hacerse esto asignando el valor `3` a `x`, usando `x : 3`. Una vez que se haya hecho la asignación `x : 3`, `x` siempre será sustituido por `3`, cuando aparezca.

Maxima

Esto asigna el valor de 3 a x.

```
(%i17)  x :3;
(%o17)  3
```

Maxima

Ahora x será sustituido automáticamente por 3 dondequiera que aparezca.

```
(%i18)  x^2 - 1;
```

```
(%o18)  8
```

Maxima

Esto asigna la expresión 1 + a a x.

```
(%i19)  x : a + 1;
(%o19)  a + 1
```

Maxima

Ahora x es reemplazado por a + 1.

```
(%i20)  x^2 − 1;
(%o20)  (a + 1)^2 − 1
```

Es posible definir como valor de un símbolo a cualquier expresión, no solamente a un número. Debe recordarse que una vez que se haya dado tal definición, ésta continuará siendo utilizada siempre que aparezca el símbolo, hasta que el usuario la cambie o quite explícitamente. Para la mayoría de usuarios, el olvidarse quitar valores que han asignado a los símbolos es la causa más común de errores al usar Maxima.

x:valor	define un valor para x que será utilizado siempre
`kill(x)`	remueve cualquier valor definido para x

Asignando valores a símbolos.

Maxima

El símbolo x todavía tiene el valor que le asignó arriba.

```
(%i21)  x + 5 − 2 ∗ x;
(%o21)  −2 (a + 1) + a + 6
```

Maxima

Esto quita el valor que asignó a x.

```
(%i22)  kill(x)
(%o22)  done
```

Maxima

Ahora x no tiene ningún valor definido, así que puede ser utilizado como variable puramente simbólica.

```
(%i23)  x+5-2*x;
(%o23)  5 - x
```

Los lenguajes de programación tradicionales que no soportan el cálculo simbólico permiten que las variables sean utilizadas solamente como nombres para objetos, típicamente números, que se han asignado como valores para ellos. En Maxima, sin embargo, `x` se puede también tratar como variable puramente formal, a la cual se le puede aplicar varias reglas de transformación. Por supuesto, si el usuario da explícitamente una definición, por ejemplo `x : 3`, entonces `x` será sustituida siempre por 3, y no sirve más como variable formal.

Debe recordarse que las definiciones explícitas por ejemplo `x : 3` tienen un efecto global. Por otra parte, un reemplazo tal como

`expr , x = 3`

afecta solamente a la expresión específica expr.

Es posible mezclar siempre reemplazos con asignaciones. Con asignaciones, se puede dar nombres a las expresiones en las cuales se desea hacer reemplazos, o a las reglas que se desea utilizar para hacer los reemplazos.

Maxima

Esto asigna un valor al símbolo t.

```
(%i24)  t : x^2 + 1;
(%o24)  x^2 + 1
```

Maxima

Esto asigna un valor al símbolo **t**.

```
(%i25)  t : x = 2;
(%o25)  5
```

Maxima

Esto encuentra el valor de **t** para un valor diferente de x.

```
(%i26)  t, x = 5 * a;
(%o26)  25a^2 + 1
```

Maxima

Esto encuentra el valor de **t** cuando x es sustituido por %pi, y luego evalúa el resultado numéricamente.

```
(%i27)  t, x = %pi, numer;
(%o27)  10,86960440108936
```

6.3 Transformación de expresiones algebraicas

A menudo hay muchas formas diferentes de escribir la misma expresión algebraica. Como un ejemplo, la expresión $(x+1)^2$ puede ser escrita como $x^2 + 2x + 1$. Maxima proporciona una gran colección de funciones para hacer conversiones entre las diferentes formas de expresiones algebraicas.

Maxima

expand da la "forma expandida" de una expresión, con los productos y las potencias desarrolladas.

```
(%i28)  expand((x + 1)^2);
(%o28)  x^2 + 2 x + 1
```

`expand` (expr)	desarrolla productos y potencias, escribiendo el resultado como suma de términos
expr, `expand`	equivale a `expand` (expr)
`factor` (expr)	escribe expr como un producto de factores mínimos
expr, `factor`	equivale a `factor` (expr)

Dos funciones comunes para transformar expresiones algebraicas.

Maxima

`factor` recupera la forma original.

```
(%i29)  factor(%);
(%o29)  (x+1)^2
```

Maxima

Es fácil generar expresiones complicadas con `expand`.

```
(%i30)  (x+3*y+1)^4, expand;
(%o30)  81*y^4+108*x*y^3+108*y^3+54*x^2*y^2+108*x*y^2+
        54*y^2+12*x^3*y+36*x^2*y+36*x*y+12*y+x^4+4*x^3+
        6*x^2+4*x+1
```

Maxima

`factor` a menudo le da expresiones más simples.

```
(%i31)  %, factor;
(%o31)  (3*y+x+1)^4
```

Maxima

Hay algunos casos donde `factor` puede dar expresiones más complicadas.

```
(%i32)  x^8-1, factor;
(%o32)  (x-1)*(x+1)*(x^2+1)*(x^4+1)
```

Maxima

En este caso, `expand` da la forma "más simple".

```
(%i33)  %,expand;
(%o33)  x^8 - 1
```

6.4 Simplificación de expresiones algebraicas

En muchas situaciones el usuario desea escribir una expresión algebraica en la forma más simple posible. Aunque sea difícil saber exactamente lo que se entiende por la "forma más simple", un procedimiento práctico que vale la pena seguir es analizar varias formas diferentes de una expresión, y elegir la de menor número de partes

`rat`(expr)	convierte expr al formato canónico racional
`ratsimp`(expr)	simplifica la expresión expr y todas sus subexpresiones, incluyendo los argumentos de funciones no racionales
expr, `ratsimp`	equivale a `ratsimp`(expr)
`fullratsimp`(expr)	aplica repetidamente `ratsimp` a una expresión, seguida de simplificaciones no racionales, hasta que no se obtienen más transformaciones; entonces devuelve el resultado
expr, `fullratsimp`	equivale a `fullratsimp`(expr)

Simplificación de expresiones algebraicas.

Se puede utilizar, a menudo, `ratsimp` para "mejorar" expresiones complicadas que se obtienen como resultado de cálculos.

Maxima

He aquí la integral de $\frac{1}{x^4-1}$.

```
(%i34)  integrate(1/(x^4 - 1), x);
```

```
(%o34)  -log(x+1)/4 - arctan(x)/2 + log(x-1)/4
```

Maxima

Al derivar el último resultado debería volverse a la expresión original. En este caso, como es común, se obtiene una versión más complicada de la expresión original.

```
(%i35) diff(%,x);
(%o35) -1/(2*(x^2+1)) - 1/(4*(x+1)) + 1/(4*(x-1))
```

Maxima

`ratsimp` permite volver a la forma original de la expresión.

```
(%i36) ratsimp(%);
(%o36) 1/(x^4-1)
```

Las expresiones pueden incluir funciones no racionales y los argumentos de tales funciones son también racionalmente simplificados.

Maxima

He aquí una expresión que incluye funciones no racionales cuyos argumentos admiten ser racionalmente simplificados.

```
(%i37) sin(x/(x^2+x)) = exp((log(x)+1)^2-log(x)^2);
(%o37) sin(x/(x^2+x)) = %e^((log(x)+1)^2-log(x)^2)
```

Maxima

`ratsimp` simplifica los argumentos de tales funciones.

```
(%i38) %,ratsimp;
(%o38) sin(1/(x+1)) = %e^x^2
```

Ante expresiones no racionales, una llamada a `ratsimp` puede no ser suficiente para conseguir un resultado simplificado. En ocasiones serán necesarias más de una llamada a `ratsimp`, que es lo que hace precisamente `fullratsimp`.

Maxima

Esto define la variable `expresion`.

(%i39) `expresion : (x^(a/2) + 1)^2 *(x^(a/2) - 1)^2/(x^a - 1);`

(%o39) $$\frac{\left(x^{a/2}-1\right)^2\left(x^{a/2}+1\right)^2}{x^a-1}$$

Maxima

En general, `rat` no simplifica otras funciones que no sean la suma, resta, multiplicación, división y exponenciación de exponente entero.

(%i40) `rat(expresion);`

(%o40) $$\frac{\left(x^{a/2}\right)^4-2\left(x^{a/2}\right)^2+1}{x^a-1}$$

Maxima

Con `ratsimp` se consigue una "mejor" simplificación.

(%i41) `ratsimp(expresion);`

(%o41) $$\frac{x^{2a}-2x^a+1}{x^a-1}$$

Maxima

Con `fullratsimp` se consigue simplificar la expresión al máximo.

(%i42) `fullratsimp(expresion);`

(%o42) $x^a - 1$

6.5 Expresiones puestas en diferentes formas

Las expresiones algebraicas complicadas se pueden escribir generalmente en varias maneras. Maxima proporciona una variedad de funciones para convertir expresiones de una forma a otra. Los más comunes de estas funciones son `expand`, `factor` y `ratsimp`. Sin embargo, cuando se tiene expresiones racionales que contienen cocientes, puede ser necesario utilizar otras funciones.

`expandwrt` (expr)*var*1, . . . , *var*$_n$)	expande la expresión expr con respecto a las variables *var*1, . . . , *var*$_n$ y por defecto no expande los denominadores
`expand` (expr)	expande la expresión expr
`factor` (expr)	factoriza la expresión expr
`partfrac` (expr)var)	expande la expresión expr en fracciones parciales respecto de la variable principal var

Comandos para transformar expresiones algebraicas.

Maxima

He aquí una expresión racional, la cual puede ser escrita en varias formas diferentes.

(%i43) $e:(x-1)^2*(2+x)/((1+x)*(x-3)^2);$

(%o43) $$\frac{(x-1)^2\,(x+2)}{(x-3)^2\,(x+1)}$$

Maxima

`expandwrt` expande el numerador de la expresión, pero deja el denominador en forma factorizada.

(%i44) expandwrt(e, x);

(%o44) $$\frac{x^3}{(x-3)^2\,(x+1)}-\frac{3\,x}{(x-3)^2\,(x+1)}+\frac{2}{(x-3)^2\,(x+1)}$$

Maxima

`expand` expande todo, incluyendo el denominador.

```
(%i45) expand(e);
                  x^3                      3 x
(%o45) ------------------------ - ------------------------ +
       x^3 - 5 x^2 + 3 x + 9     x^3 - 5 x^2 + 3 x + 9
                  2
       ------------------------
       x^3 - 5 x^2 + 3 x + 9
```

Maxima

`partfrac` separa la expresión en términos con denominadores simples.

```
(%i46) partfrac( %,x);
          1          19           5
(%o46) ------- + ------- + ------- + 1
       4 (x+ 1)   4 (x - 3)   (x - 3)^2
```

Maxima

`factor` factoriza todo, en este caso reproduce la forma original.

```
(%i47) factor( %);
                  (x - 1)^2 (x + 2)
(%o47) ---------------------------------
                  (x - 3)^2 (x + 1)
```

`collectterms(expr)var)` agrupa juntas todos las potencias de var

Reordenamiento de expresiones en varias variables.

Maxima

He aquí una expresión algebraica en dos variables.

```
(%i48) v : expand((3 + 2 x)^2*(x + 2 y)^2);
(%o48) 16 x^2 y^2 + 48 x y^2 + 36 y^2 + 16 x^3 y + 48 x^2 y + 36 x y +
       4 x^4 + 12 x^3 + 9 x^2
```

Maxima

Esto agrupa los términos de `v` afectados por la misma potencia de `x`.

```
(%i49) collectterms(v,x);
(%o49) x (48 y^2 + 36 y) + x^2 (16 y^2 + 48 y + 9) +
       36 y^2 + x^3 (16 y + 12) + 4 x^4
```

Maxima

Esto agrupa juntas las potencias de `y`.

```
(%i50) collectterms(v, y);
(%o50) (16 x^2 + 48 x + 36) y^2 + (16 x^3 + 48 x^2 + 36 x) y +
       4 x^4 + 12 x^3 + 9 x^2
```

Como acaba de verse, cuando el usuario se limita a expresiones polinómicas racionales, hay muchas formas de escribir cualquier expresión particular. Si éste considera expresiones más complicadas, que incluyan, por ejemplo, funciones matemáticas trascendentes, la variedad de formas posibles llega a ser mayor. Por consiguiente, es imposible tener una función incorporada específico en Maxima para producir cada forma posible. Más bien, Maxima le permite construir sistemas arbitrarios de reglas de transformación para hacer diversas conversiones[1]. Sin embargo, hay algunas funciones incorporadas adicionales de Maxima para transformar expresiones.

`trigexpand(`expr`)`	expande funciones trigonométricas e hiperbólicas de sumas de ángulos y de múltiplos de ángulos presentes en expr
expr, `trigexpand`	equivale a `trigexpand(`expr`)`
`trigsimp(`expr`)`	utiliza las identidades $\text{sen}(x)^2 + \cos(x)^2 = 1$ y $\cosh(x)^2 - \text{senh}(x)^2 = 1$ para simplificar expresiones que contienen tan, sec, etc.

Algunas funciones más para transformar expresiones.

[1] Para más detalle al respecto consulte sobre las funciones `scsimp` y `defrule` en la ayuda de Maxima.

`trigreduce(`expr`)`var	combina productos y potencias de senos y cosenos trigonométricos e hiperbólicos de var, transformándolos en otros que son múltiplos de var
`trigreduce(`expr`)`	si no se introduce el argumento var, entonces se utilizan todas las variables de expr
expr, `trigreduce`	equivale a `trigreduce(`expr`)`
`trigrat(`expr`)`	devuelve una forma canónica simplificada cuasi-lineal de una expresión trigonométrica
`exponentiali e(`expr`)`	convierte las funciones trigonométricas e hiperbólicas de expr a exponenciales
expr, `exponentiali e`	equivale a `exponentiali e(`expr`)`
`demoivre(`expr`)`	convierte exponenciales complejos en expresiones equivalentes pero en términos de las funciones trigonométricas
expr, `demoivre`	equivale a `demoivre(`expr`)`
`rectform(`expr`)`	devuelve una expresión de la forma $a+b\,\%i$ equivalente a expr, con a y b reales
expr, `rectform`	equivale a `rectform(`expr`)`
`polarform(`expr`)`	devuelve una expresión de la forma $r\,\%e^{\%i\,\theta}$ equivalente a expr, con r y θ reales
expr, `polarform`	equivale a `polarform(`expr`)`
`radcan(`expr`)`	simplifica la expresión expr, que puede contener logaritmos, exponenciales y radicales, convirtiéndola a una forma canónica
expr, `radcan`	equivale a `radcan(`expr`)`
prod , `radexpand:all`	las raíces n-ésimas de los factores del producto prod, que sean potencias de n, se extraen del símbolo radical (p. ej., $\sqrt{4x^2} \rightarrow 2x$)

Algunas funciones más para transformar expresiones.

`logcontract(`**expr**`)`	analiza la expresión **expr** recursivamente, transformando subexpresiones de la forma `a1*log(b1)+a2*log(b2)+c` en expresiones de la forma `log(ratsimp(b1^a1*b2^a2))+c`
expr, `logcontract`	equivale a `logcontract(`**expr**`)`

Algunas funciones más para transformar expresiones.

Maxima

Esto expande la expresión trigonométrica, escribiéndola de modo que todas las funciones tengan argumento x.

```
(%i51) tan(x)*cos(3*x),trigexpand;
                        3           2
(%o51)                cos x - 3 cos x sin x
```

Maxima

Esto reduce la expresión usando ángulos múltiples.

```
(%i52) tan(x)*cos(2*x), trigreduce;
(%o52)                  tan x cos(3 x)
```

Maxima

Esto expande el seno asumiendo que `x` e `y` son reales.

```
(%i53) sin(x + %i*y), rectform;
(%o53)                %i cos x sinh y + sin x cosh y
```

Maxima

Con `logcontract` se "contrae" una expresión logarítimica.

```
(%i54) 2*(a*log(x) + 2^a*log(y)), logcontract;
                                 4
(%o54)                       a log(x  y )
```

Las transformaciones hechas por funciones como `expand` y `factor` siempre son correctas, independientemente del valor que puedan tener las variables simbólicas en las expresiones. A veces, sin embargo, es útil realizar transformaciones que sólo son correctas para algunos posibles valores de las variables simbólicas. Transformaciones como éstas las realizan `radcan` y `radexpand:all`.

Maxima

Maxima no expande automáticamente potencias no enteras de productos y cocientes.

```
(%i55) sqrt(x^5*y/w^3);
                                 5
                               sqrt(x  y)
(%o55)                         ----------
                                    3
                                   w
```

Maxima

`radcan` hace la expansión.

```
(%i56) %,radcan;
                                 5/2
                                x    sqrt(y)
(%o56)                          ------------
                                    3/2
                                   w
```

Maxima

En este caso Maxima aplica una equivalencia matemática.

```
(%i57) sqrt(x^6*y^2/w^10);
                                 3
                              abs(x)  abs(y)
(%o57)                        --------------
                                   5
                                abs(w)
```

Maxima

Utilizando la variable opcional `radexpand` con el valor asignado `all`, Maxima pasa por alto la equivalencia anterior.

```
(%i58) sqrt(x^6*y^2/w^10),radexpand:all;
                                 3
                                x  y
(%o58)                          ----
                                 5
                                w
```

6.6 Simplificación con asunciones

assume($pred_1, \ldots, pred_n$)	añade los predicados $pred_1, \ldots, pred_n$ al contexto actual
forget($pred_1, \ldots, pred_n$)	borra los predicados establecidos por `assume`

Asunción de predicados.

Maxima

`assume` devuelve una lista con los predicados que han sido añadidos al contexto.

```
(%i59)  assume(x > 0,y < 0);
(%o59)  [x > 0, y < 0]
```

Maxima

Maxima realiza simplificaciones asumiendo los predicados ingresados.

```
(%i60)  [sqrt(x^2), sqrt(y)];
(%o60)  x, √y
```

Maxima

Otra simplificación asumiendo los predicados ingresados.

```
(%i61)  sqrt(x^2 * y^2);
(%o61)  −xy
```

Maxima

`forget` borra los predicados previamente establecidos.

```
(%i62)  forget(x > 0, y < 0);
(%o62)  [x > 0,y < 0]
```

6.7 Selección de partes de expresiones algebraicas

`coeff`(expr)x)n)	devuelve el coeficiente de x^n en expr (el argumento n puede omitirse si es igual a la unidad)
`hipow`(expr)x)	devuelve el mayor exponente explícito de x en expr (si x no aparece en expr, `hipow` devuelve 0)
`part`(expr)$n_1, \ldots, n_k$)	devuelve la parte de expr que se especifica por los índices $n_1, \ldots, n_k$ (primero se obtiene la parte n_1 de expr, después la parte n_2 del resultado anterior, y así sucesivamente)

Comandos para seleccionar partes de polinomios.

Maxima

He aquí una expresión algebraica.

```
(%i63)  e : expand((1 + 3*x + 4*y^2)^2);
(%o63)  16*y^4+24*x*y^2+8*y^2+9*x^2+6*x+1
```

Maxima

Esto da el coeficiente de `x` en `e`.

```
(%i64)  coeff(e, x);
(%o64)  24*y^2+6
```

Maxima

`hipow(expr , y)` da la mayor potencia de `y` que aparece en expr.

```
(%i65)  hipow(e,y);
(%o65)  4
```

Maxima

Esto da el cuarto término en `e`.

```
(%i66)  part(e, 4);
(%o66)  9 x^2
```

`num(expr)`	devuelve el numerador de expr (si expr no es una fracción, devuelve expr)
`denom(expr)`	devuelve el denominador de expr (si expr no es una fracción, devuelve 1)

Comandos para seleccionar partes de expresiones racionales.

Maxima

He aquí una expresión racional.

```
(%i67)  r : (1 + x)/(2 * (2 - y));
                          x + 1
(%o67)                 ----------
                       2 (2 - y)
```

Maxima

`denom` selecciona el denominador.

```
(%i68)  denom(r);
(%o68)  2 (2 - y)
```

Maxima

`denom` da 1 para las expresiones que no son cocientes.

```
(%i69)  denom(1/x + 1/y);
(%o69)  1
```

Capítulo 7

Matemáticas simbólicas

7.1 Límites

`limit(f,x,x0)`	el límite $\lim_{x \to x_0} f$
`limit(f,x,x0,plus)`	el límite $\lim_{x \to x_0^+} f$
`limit(f,x,x0,minus)`	el límite $\lim_{x \to x_0^-} f$

Límites.

Maxima

He aquí la expresión $\frac{\operatorname{sen} x}{x}$.

```
(%i1)  f:sin(x)/x;
(%o1)  sin(x)/x
```

Maxima

Si se sustituye x por 0, la expresión se hace 0/0, y se obtiene un mensaje de error.

```
(%i2)  f,x=0;
Division by 0
- -an error. To debug this try: debugmode(true);
```

Maxima

Si se evalúa $\frac{\text{sen}(x)}{x}$ para un x próximo a 0, se consigue un resultado próximo a 1.

```
(%i3)  f, x= 0,01;
(%o3)  0,99998333341667
```

Maxima

Esto encuentra el límite de $\frac{\text{sen}(x)}{x}$ cuando x tiende a 0.

```
(%i4)  limit(f, x, 0);
(%o4)  1
```

`inf`	$+\infty$
`minf`	$-\infty$
`und`	indefinido
`ind`	indefinido pero acotado
`eroa`	infinitesimal mayor que cero
`erob`	infinitesimal menor que cero
`infinit`	infinito complejo

Símbolos especiales para límites.

La función `limit` con un solo argumento se utiliza frecuentemente para simplificar expresiones en las que aparecen los símbolos especiales para límites.

Maxima

Esto da un resultado para $1 - \infty$.

```
(%i5)  limit(1 - minf);
(%o5)  inf
```

Maxima

He aquí la simplificación de una expresión que incluye un infinitesimal mayor que cero.

```
(%i6)  limit(x + zeroa);
(%o6)  x
```

7.2 Diferenciación

`diff(f,x)`	devuelve la primera derivada de f respecto de la variable x
`diff(f,x,n)`	devuelve la n-esima derivada de f respecto de x
`diff(f,`$x_1, n_1, \ldots, x_m, n_m$`)`	devuelve la derivada parcial de f con respecto de $x_1, \ldots, x_m$ y equivale a `diff(...(diff(f,`x_m, n_m`...),`x_1, n_1`)`
`diff(f)`	devuelve el diferencial total de f

Diferenciación con Maxima.

Maxima

He aquí la derivada x^n con respecto a x.

```
(%i7)  diff(x^n, x);
(%o7)  n x^(n-1)
```

Maxima

Maxima conoce las derivadas de todas las funciones matemáticas estándar.

```
(%i8)  diff(atan(x), x);
                1
(%o8)  ----------
                2
               x  + 1
```

Maxima

La tercera derivada con respecto a x.

```
(%i9)  diff(x^n, x, 3);
(%o9)  (n − 2) (n − 1) n x^(n−3)
```

Si no se indica la variable, Maxima asume que se quiere calcular la diferencial total, en la cual todas las derivadas se asumen relacionadas. En notación matemática, `diff(f)x )` es como $\frac{d}{dx}f$, mientras `diff(f)` es como df.

Maxima

Esto da la diferencial total $d(x^n)$. `del(x)` y `del(n)` son los diferenciales dx y dn, respectivamente.

```
(%i10)  diff(x^n);
(%o10)  n x^(n−1) del(x) + x^n log x del(n)
```

Así como se trata variables simbólicamente, también es posible tratar funciones simbólicamente en Maxima. Así, por ejemplo, puede encontrarse fórmulas para las derivadas de `f(x)`, sin especificar una forma explícita para la función `f`. Para esto hay que indicar a Maxima la dependencia de la función, lo que se consigue con la función `depends`.

Maxima

Esto declara la dependencia $f(x^2)$.

```
(%i11)  depends(f,x^2);
(%o11)  [f(x^2)]
```

Maxima

Ahora Maxima puede utilizar la regla de cadena para simplificar la derivada.

```
(%i12)  diff(2 * x * f, x);
                  d
(%o12)  4  ------(f)  x^2 + 2 f
                   2
                  dx
```

depends(φ, ϕ)	declara la dependencia funcional $\varphi(\phi)$
depends(φ_1, ϕ_1, . . . , φ_n, ϕ_n)	declara la dependencia funcional $\varphi_1(\phi_1), \ldots, \varphi_n(\phi_n)$
depends([$\varphi_1, \ldots, \varphi_n$], ϕ)	declara la dependencia funcional $\varphi_1(\phi), \ldots, \varphi_n(\phi)$
depends(φ, [$\phi_1 \ldots, \phi_n$])	declara la dependencia funcional $\varphi(\phi_1, \ldots, \phi_n)$
depends([$\varphi_1, \ldots, \varphi_n$], [$\phi_1, \ldots, \phi_m$])	declara la dependencia funcional $\varphi_1(\phi_1, \ldots, \phi_m), \ldots, \varphi_n(\phi_1, \ldots, \phi_m)$
dependencies	lista de átomos que tienen algún tipo de dependencia funcional
remove(φ, dependenc)	elimina la dependencia funcional asociada con φ
remove([$\varphi_1, \ldots, \varphi_n$], dependenc)	elimina la dependencia funcional asociada con $\varphi_1, \ldots, \varphi_n$
remove(all, dependenc)	elimina la dependencia funcional de todos los átomos que la tengan

Declaración y remonición de dependencia funcional.

Maxima

Esto declara las dependencias $u(x)$ y $v(x)$.

(%i13) depends([u,v],x);

(%o13) [u(x), v(x)]

Maxima

Aquí se obtiene una fórmula para $\frac{d}{dx}(u^v)$, donde $u = u(x)$ y $v = v(x)$.

(%i14) diff(u^v,x), expand;

(%o14) $u^v \log u \left(\frac{d}{dx}v\right) + u^{v-1}\left(\frac{d}{dx}u\right) v$

Maxima

Esto permite apreciar todas las dependencias declaradas hasta el momento.

(%i15) dependencies;

(%o15) $[f(x^2), u(x), v(x)]$

Maxima

Con esta sentencia se borran todas las dependencias.

```
(%i16) remove(all, dependency);
(%o16) done
```

7.3 Integración

`integrate(f)x)`	la integral indefinida $\int f dx$
`integrate(f)x)a)b)`	la integral definida $\int_a^b f dx$
`integrate(f=g)x)`	la integral definida de una ecuación, equivale a $\int f dx = \int g dx$
`changevar('expr)`$\varphi(x, y)$, `y)x)`	hace el cambio de variable dado por $\varphi(x, y) = 0$ en la expresión expr que depende de x (la nueva variable será y)

Integración.

`integration constant`	variable del sistema cuyo valor por defecto es `%c`
`integration constant counter`	variable del sistema cuyo valor por defecto es 0

Constantes y contadores de constantes para integrar ecuaciones.

Maxima

Para calcular la integral $\int x^n dx$, Maxima, pregunta si $n + 1 = 0$ o $n + 1 \neq 0$. En este caso se le ha indicado la elección de la opción $n + 1 = 0$, es decir, $n \neq -1$.

```
(%i17) integrate(x^n, x);
Is n+1 zero or nonzero? n;
```

(%o17) $\frac{x^{n+1}}{n+1}$

Maxima

He aquí la integral $\int x^n dx$, cuando $n = -1$.

```
(%i18) integrate(x^n, x);
Is n+1  zero or nonzero? z;
(%o18) log(x)
```

Maxima

Este es un ejemplo ligeramente más complicado.

```
(%i19) integrate(1/(x^4 - a^4), x);
```

(%o19) $-\frac{\log(x+a)}{4a^3} + \frac{\log(x-a)}{4a^3} - \frac{\arctan\left(\frac{x}{a}\right)}{2a^3}$

Maxima

Recuérdese que `logcontract` "contrae" los logaritmos.

```
(%i20) integrate(1/(x^4 - a^4), x), logcontract;
```

(%o20) $-\frac{\log\left(\frac{x+a}{x-a}\right) + 2\arctan\left(\frac{x}{a}\right)}{4a^3}$

Maxima resuelve casi cualquier integral que puede ser expresada en términos de funciones matemáticas estándares. Pero debe comprenderse que aun cuando un integrando pueda contener sólo funciones simples, su integral puede implicar funciones mucho más complicadas, o puede no ser expresable en absoluto en términos de funciones matemáticas estándares.

Maxima

He aquí una integral simple.

```
(%i21) integrate(log(1 - x^2), x), logcontract;
```

```
(%o21)  x (log(1 - x^2) - 2) + log((x + 1)/(x - 1))
```

Maxima

Esta integral puede ser expresada sólo en términos de una función dilogarítmica[1].

```
(%i22)  integrate(log(1 - x^2)/x, x);
(%o22)  log(x) log(1 - x^2) + (li_2(1 - x^2))/2
```

Maxima

Esta integral involucra la función `erf`[2].

```
(%i23)  integrate(exp(1 _ x^2), x);
                        sqrt(%pi) %e erf(x)
(%o23)                  -------------------
                                2
```

Maxima

Esta integral simplemente no puede ser expresada en términos de funciones matemáticas estándares. Por consiguiente, Maxima la deja como está.

```
(%i24)  integrate(x^x, x);
                              1
(%o24)                       x^x dx
```

Maxima

He aquí la integral definida $\int_a^b \operatorname{sen}^2(x)\,dx$.

```
(%i25)  integrate(sin(x)^2, x, a, b);
                  sin(2 a) - 2 a     sin(2 b) - 2 b
(%o25)            -------------  _  -------------
                         4                 4
```

[1] La función polilogarítmica de orden s y argumento z está definida por la serie infinita $li_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s}$. En este caso $s = 2$ y $z = 1 - x^2$.

[2] `erf(z)` es la integral de la distribución gaussiana, dada por $erf(z) = \int_0^z e^{-t^2}\,dt$

Maxima

He aquí otra integral definida.

```
(%i26)  integrate(exp(-x^2),x,0,inf);
(%o26)  sqrt(%pi)/2
```

Maxima

Maxima no puede darle una fórmula para esta integral definida.

```
(%i27)  integrate(x^x,x,0,1);
                 1
                 /
(%o27)           [  x^x dx
                 ]
                 /
                 0
```

Maxima

Esto evalúa la integral múltiple $\int_0^1 \int_0^x (x^2 + y^2)\, dy\, dx$.

```
(%i28)  integrate(integrate(x^2+y^2,y,0,x),x,0,1);
(%o28)  1/3
```

Cuando una constante de integración se crea durante la integración definida de una ecuación, el nombre de la constante se construye concatenando `integration constant` y `integration constant counter`.

Maxima

Esto calcula la integral definida de una ecuación.

```
(%i29)  integrate(x^2=sin(x),x);
(%o29)  x^3/3=%c[1]-cos(x)
```

Maxima

A `integration_constant` se le puede asignar un símbolo cualquiera.

```
(%i30)  integration constant : 'K;
(%o30)  K
```

Maxima

Esto calcula la integral definida de una ecuación a la que se le ha reiniciado la constante de integración por defecto.

```
(%i31) integrate(x^2 = sin(x), x);
                    3
                   x
(%o31)             -- = K - cos(x)
                   3
```

Maxima

Es posible reiniciar el contador de `integration_constant_counter`.

```
(%i32) reset(integration constant counter);
(%o32) [integration constant counter]
```

Maxima

Aquí se aprecia que el contador ha sido reiniciado.

```
(%i33) integrate(x^2 = sin(x), x);
                    3
                   x
(%o33)             -- = K1 - cos(x)
                   3
```

Es posible realizar un cambio de variable en una integral indefinida o definida usando la función `changevar`. Para que Maxima pueda realizar esto debe mantenerse la integral sin evaluar (sección 5.4).

Maxima

He aquí una integral clásica sin evaluar.

```
(%i34) 'integrate(%e^sqrt(y),y);
                 1
                 /
                 [      sqrt(y)
(%o34)           I  %e       dy
                 ]
                 /
```

Maxima

Esto realiza un cambio de variable en una integral indefinida.

```
(%i35) assume(z > 0)$ changevar(%o28,y - z^2,z,y);
```

```
(%o36)   ∫ %e^√y dy
         1
```

Maxima

Esto realiza un cambio de variable en una integral definida.

```
(%i37)  'changevar(integrate(%e^sqrt(y),y,0,4),y-z^2,z,y);
(%o37)  -2 ∫ z %e^z dz
           1 0
           -2
```

7.4 Sumas y Productos

sum(f)i)i_{min}, i_{max})	la suma $\sum_{i_{min}}^{i_{max}} f$
lsum(f)i))	representa la suma de f para cada elemento i en

Sumas.

Maxima

Esto construye la suma $\sum_{i=1}^{7} \frac{x}{i}$.

```
(%i38)  sum(x^i/i,i,1,7);
                 7    6    5    4    3    2
(%o38)          x    x    x    x    x    x
                - + - + - + - + - + - + x
                7    6    5    4    3    2
```

Maxima

Esto devuelve la suma $\sum_{i=1}^{n} i^2$ sin realizar ningún cambio.

```
(%i39)  sum(i^2,i,1,n);
          n
          ∑   i^2
         i=1
(%o39)
```

Maxima

Agregando `simpsum` la suma es calculada simbólicamente como una función de `n`.

```
(%i40) sum(i^2, i, 1, n), simpsum;
                          3      2
(%o40)                   2 n  + 3 n  + n
                         ---------------
                                6
```

Maxima

Combinando `simpsum` con `factor` se obtiene un resultado factorizado.

```
(%i41) sum(i^2, i, 1, n), simpsum, factor;
                          n (n + 1) (2 n + 1)
(%o41)                    -------------------
                                  6
```

Maxima

Maxima también puede dar un resultado exacto para esta suma infinita.

```
(%i42) sum(1/i^4, i, 1, inf), simpsum;
                           4
                          %pi
(%o42)                    ----
                           90
```

Maxima

Esta es la suma múltiple $\sum_{i=1}^{3}\sum_{j=1}^{i} x^i y^j$.

```
(%i43) sum(sum(x^i * y^j, j, 1, i), i, 1, 3);
                3 3    3 2    2 2    3      2      3    2
(%o43)         x  y  + x  y  + x  y  + x  y + x  y + x  y + x y
```

Maxima

Esta es una suma para la cual los valores del índice de variación no están equi-incrementados.

```
(%i44) lsum(x^i, i, [1, 2, 7]);
                          7    2
(%o44)                   x  + x  + x
```

product(f (i))i)i_{min},i_{max}) el producto $\prod_{i_{min}}^{i_{max}} f(i)$

Productos.

Maxima

Los productos se obtienen en forma similar a las sumas.

```
(%i45)  prod(x+i,i,1,4);
(%o45)  (x+1) (x+2) (x+3) (x+4)
```

Maxima

Agregando `simpproduct` la suma es calculada simbólicamente como una función de `n`.

```
(%i46)  product(k,k,1,n),simpproduct;
(%o46)  n!
```

Cabe mencionar que la función `changevar` también se puede utilizar para cambiar los índices de una suma o producto. Sin embargo, debe tenerse en cuenta que cuando se realiza un cambio en una suma o producto, el mismo debe expresarse en términos de sumas, como $i = j + \ldots$, no como una función de mayor grado.

Maxima

He aquí una suma finita con índice *j*.

```
(%i47)  sum(f(j+2)*x^j,j,-2,n);
                 n
(%o47)           ====
                 \      f(j + 2) x^j
                 /
                 ====
                 j=-2
```

Maxima

Esto realiza el cambio $j = i - 2$ en la suma anterior.

```
(%i48)  changevar(%,j-i+2,i,j);
```

```
(%o48)  sum(f(i)*x^(i-2),i,0,n+2)
```

Maxima

Aquí se hace el cambio $i = k - 2$ en un producto infinito.

```
(%i49)  product(f(i+2)*x^(i+2),i,-2,inf);
(%o49)  sum(f(i+2)*x^(i+2),i,-2,inf)
(%i50)  changevar(%,i-k+2,k,i);
(%o50)  sum(f(k)*x^k,k,0,inf)
```

7.5 Operadores relacionales y lógicos

=	igual (por sintaxis)
#	desigual (por sintaxis)
>	mayor que
>=	mayor o igual que
<	menor que
<=	menor o igualque
`equal`	igual (por valor)
`notequal`	desigual (por valor)

Operadores relacionales.

`compare(x,y)`	compara x e y y devuelve <, <=, >, >=, =, #, `notcomparable` ó `unknown`, según sea el caso

Función para obtener operadores relacionales.

Maxima

Esto prueba si l0 es menor que 7.

```
(%i51)  10<7,pred;
(%o51)  false
```

Maxima

Esto prueba si l0 es diferente que 7.

```
(%i52)  10#7,pred;
(%o52)  true
```

Maxima

Ya que los entes involucrados son numéricas, Maxima puede determinar que esto es verdadero.

```
(%i53)  %pi^%e < %e^%pi,pred;
(%o53)  true
```

Maxima

Puesto que los entes a comparar son constantes, Maxima no sabe si esto es verdadero o falso.

```
(%i54)  x > y,pred;
(%o54)  x > y
```

Maxima

He aquí dos comparaciones.

```
(%i55)  compare(1, 2);
(%o55)  <
(%i56)  compare(1/x, 0);
(%o56)  #
```

Maxima

Una diferencia notable entre = y `equal`.

```
(%i57)  (x + 1)^2 = x^2 + 2*x + 1, pred;
(%o57)  false
(%i58)  equal((x + 1)^2, x^2 + 2*x + 1), pred;
(%o58)  true
```

El usuario debe tener presente que, los operadores relacionales son todos operadores binarios. Maxima no reconoce expresiones del estilo $a < b < c$.

Maxima

Al pretender evaluar una desigualdad como la siguiente, Maxima devuelve un mensaje de error.

```
(%i59)  2 < 3 < 4, pred;
Incorrect syntax: Found logical expression where algebraic expression
expected
2<3<4
   ~
```

`not`	negación
`and`	conjunción
`or`	disyunción
`if` cond `then` *expr*$_1$ `else` *expr*$_2$	devuelve *expr*$_1$ si cond es `true`, y *expr*$_2$ si cond es `false`

Operadores lógicos.

Maxima

Una forma de ingresar expresiones del estilo $a < b < c$, en Maxima, es usando el operador lógico `and`.

```
(%i60)  2 < 3 and 3 <4;
(%o60)  true
```

Maxima

Mientras no se hayan hecho asignaciones adecuadas a p y q, Maxima no sabe si esto es verdadero o falso.

```
(%i61)  p and q;
(%o61)  p ∧ q
```

7.6 Ecuaciones

En Maxima, una ecuación consiste de dos expresiones vinculadas con el símbolo =.

$expr_1$ = $expr_2$	representa una ecuación
lhs ($expr_1$ = $expr_2$)	devuelve $expr_1$
rhs ($expr_1$ = $expr_2$)	devuelve $expr_2$

Ecuaciones en Maxima.

Maxima

Una ecuación por sí misma no es evaluada.

```
(%i62)  x^4 - 5*x^2 - 3 = x;
(%o62)  x = x^4 - 5 x^2 - 3
```

Es muy importante que el usuario no confunda `x:` con `x=` (ver sección 5.2). Mientras que `x:` es una declaración imperativa que origina una asignación, `x=` no causa ninguna acción explícita.

Maxima

Con `lhs`[3] se selecciona el miembro izquierdo de la ecuación.

```
(%i63)  lhs(x = x^4 - 5*x^2 - 3);
(%o63)  x
```

[3] Las funciones `lhs` y `rhs` también pueden ser aplicadas en los casos cuando el vínculo es cualquier otro operador relacional.

Maxima

Con `rhs` se selecciona el miembro derecho de la ecuación.

```
(%i64)  rhs(x=x^4-5*x^2-3);
(%o64)  x^4-5*x^2-3
```

7.7 Solución de Ecuaciones

`solve`(ecu)x)	resuelve la ecuación algebraica ecu de incógnita x

Solución de ecuaciones.

`solve` siempre trata de dar fórmulas explícitas para las soluciones de ecuaciones. Sin embargo, para ecuaciones suficientemente complicadas, no pueden darse fórmulas algebraicas explícitas. Si se tiene una ecuación algebraica en una variable, y la potencia más alta de la variable es menor que cinco, entonces Maxima siempre puede dar fórmulas para las soluciones. Sin embargo, si la potencia más alta es cinco o más, puede ser matemáticamente imposible dar fórmulas algebraicas explícitas para todas las soluciones.

Maxima

Maxima siempre puede solucionar ecuaciones algebraicas en una variable cuando la potencia más alta es menor que cinco.

```
(%i65)  solve(x^4-5*x^2-3=0,x);
(%o65)  x=-sqrt(sqrt(37)+5)/2,x=sqrt(sqrt(37)+5)/2,x=-sqrt(sqrt(37)-5)*%i/2,
        x=sqrt(sqrt(37)-5)*%i/2
```

Maxima

También puede solucionar algunas ecuaciones que involucran potencias más altas.

```
(%i66)  solve(x^6=1,x);
```

(%o66) $x = \frac{\sqrt{3}\,\%i+1}{2}, x = \frac{\sqrt{3}\,\%i-1}{2}, x = -1, x = -\frac{\sqrt{3}\,\%i+1}{2}, x = -\frac{\sqrt{3}\,\%i-1}{2}, x = 1$

Maxima

Hay algunas ecuaciones, sin embargo, para las cuales es matemáticamente imposible encontrar fórmulas explícitas para las soluciones.

```
(%i67) solve(2 - 4 * x + x^5 = 0, x);
(%o67) [0 = x^5 - 4 x + 2]
```

Además de la capacidad de solucionar ecuaciones puramente algebraicas, Maxima también puede solucionar algunas ecuaciones que implican otras funciones.

Maxima

Hay algunas ecuaciones, sin embargo, para las cuales es matemáticamente imposible encontrar fórmulas explícitas para las soluciones. Después de la impresión de una advertencia, Maxima devuelve una solución para esta ecuación.

```
(%i68) solve(sin(x) = a, x);
solve: using arc-trig functions to get a solution.
Some solutions will be lost.
(%o68) [x = asin(a)]
```

Es importante comprender que una ecuación tal como sen(x) = a en realidad tiene un número infinito de soluciones posibles, en este caso que se diferencian por múltiplos de 2π. Sin embargo, `solve` por defecto da sólo una solución, pero imprime un mensaje que le dice que pueden existir otras soluciones.

Maxima

No hay ninguna solución explícita para una ecuación trascendental como esta.

```
(%i69) solve(cos(x) = x, x);
(%o69) [x = cos(x)]
```

`solve`([$ecu_1, \ldots, ecu_n$] , [$x_1, \ldots, x_n$])	resuelve un sistema de ecuaciones polinómicas simultáneas (lineales o no)

Solución de sistemas ecuaciones.

También puede usarse **Maxima** para solucionar conjuntos de ecuaciones simultáneas. Simplemente se da la lista de ecuaciones, y especifica la lista de variables.

Maxima

He aquí una lista de dos ecuaciones simultáneas, para que sean resueltas en las variables x e y.

(%i70) solve ([a * x + y = 0, 2 * x + (1 - a) * y = 1], [x, y]) ;

(%o70) $$x = \frac{1}{a^2 - a + 2},\ y = -\frac{a}{a^2 - a + 2}$$

Maxima

He aquí algunas ecuaciones simultáneas más complicadas.

(%i71) solve([x^2 + x * y + y^2 = 4, x + x * y + y = 2], [x, y]);

(%o71) $$[x = 2, y = 0],\ \left[x = -\frac{\sqrt{11}\,\%i + 3}{2}, y = -\frac{\sqrt{11}\,\%i + 7}{\sqrt{11}\,\%i + 1}\right],\ \left[x = \frac{\sqrt{11}\,\%i - 3}{2}, y = -\frac{\sqrt{11}\,\%i - 7}{\sqrt{11}\,\%i - 1}\right],\ [x = 0, y = 2]$$

Maxima

Con `rectform` se obtiene una mejor presentación.

(%i72) %, rectform;

(%o72) $$[x = 2, y = 0],\ \left[x = -\frac{\sqrt{11}\,\%i}{2} - \frac{3}{2}, y = \frac{\sqrt{11}\,\%i}{2} - \frac{3}{2}\right],\ \left[x = \frac{\sqrt{11}\,\%i}{2} - \frac{3}{2}, y = -\frac{\sqrt{11}\,\%i}{2} - \frac{3}{2}\right],\ [x = 0, y = 2]$$

Una variable para `solve` puede ser también una expresión. Esto se aprecia, por ejemplo, al aplicar esta función para obtener la derivada de una función dada en forma implícita.

Maxima

Esto asigna la ecuación $y^3 - 3\,x\,y + x^3 = 0$ a la variable e.

(%i73) e:x^3-3*x*y+y^3=0;

(%o73) $y^3 - 3\,x\,y + x^3 = 0$

Maxima

Aquí se define la dependencia $y(x)$.

(%i74) depends(y,x);

(%o74) [y (x)]

Maxima

Ahora se deriva e

(%i75) diff(e,x);

(%o75) $3\,y^2\left(\frac{d}{d\,x}\,y\right) - 3\,x\left(\frac{d}{d\,x}\,y\right) - 3\,y + 3\,x^2 = 0$

Maxima

Finalmente se resuelve la ecuación para la variable $\frac{d}{dx}y$

(%i76) solve(%,diff(y,x));

(%o76) $\frac{d}{d\,x}\,y = \frac{y - x^2}{y^2 - x}$

7.8 Ecuaciones diferenciales ordinarias

ode2 (ecu)dvar)ivar)	resuelve la ecuación diferencial ordinaria ecu, de variable dependiente dvar y variable independiente ivar, de primer orden y segundo orden

Solución de ecuaciones diferenciales ordinarias.

`ic11` (sol)$x = x_0$, $y = y_0$)	resuelve el problema del valor inicial en ecuaciones diferenciales ordinarias de primer orden
`ic12` (sol)$x = x_0$, $y = y_0$, ' $diff(y, x) = dy_0$)	resuelve el problema del valor inicial en ecuaciones diferenciales ordinarias de segundo orden
`bc2` (sol)$x = x_0$, $y = y_0$, $x = x_1$, $y = y_1$)	resuelve el problema del valor en la frontera para ecuaciones diferenciales ordinarias de segundo orden

Solución de ecuaciones diferenciales ordinarias.

Maxima

He aquí la solución de la ecuación diferencial $y'(x) = ay(x) + 1$. En esta solución $\%c$ es una constante que debe ser determinada a partir de valores iniciales.

(%i77) ode2(' diff(y, x) = a * y + 1, y, x);

(%o77) $y = \left(\%c - \frac{e^{-a\,x}}{a}\right) e^{a\,x}$

Maxima

Si se incluye una condición inicial apropiada, entonces no hay ninguna constante en la solución.

(%i78) ic1(%, x = 0, y = 0);

(%o78) $y = \frac{e^{a\,x} - 1}{a}$

Maxima

He aquí la solución de la ecuación diferencial $y''(x) + y\,y'(x) = 0$. En esta solución $\%k_1$ y $\%k_2$ son constantes a ser determinadas a partir de valores iniciales o valores de frontera.

(%i79) ode2(' diff(y, x, 2) + y * ' diff(y, x)^3 = 0, y, x);

(%o79) $\frac{y^3 + 6\ \%k_1 y}{6} = x + \%k_2$

Maxima

Si se incluyen las condiciones iniciales apropiadas desaparecen las constantes.

```
(%i80)  ic2(%,x=0,y=0,'diff(y,x)=1);
                          3
(%o80)                   y  - 3 y (y^2 - 1)
                         ------------------ = x
                                 6
```

Maxima

Si se incluyen los valores de frontera apropiados también desaparecen las constantes.

```
(%i81)  bc2(%o3,x=0,y=0,x=1,y=1);
                          3
(%o81)                   y  + 5 y
                         ------- = x
                            6
```

7.9 Sistemas de ecuaciones diferenciales ordinarias lineales

La función `desolve` resuelve sistemas de ecuaciones diferenciales ordinarias lineales utilizando la transformada de Laplace. La dependencia funcional respecto de una variable independiente debe indicarse explícitamente, tanto en las variables como en las derivadas. Por ejemplo, para la primera derivada, a diferencia de la forma `'diff(y,x)` (usada en la sección 7.8), debe usarse la forma `'diff(y(x),x)`.

`desolve(ecu,y(x))`	resuelve una ecuación diferencial para `y(x)`, tomando a `x` como variable independiente
`desolve([`$ecu_1,\ldots,ecu_n$`],[`$y_1(x),\ldots,y_n(x)$`])`	resuelve sistemas de ecuaciones diferenciales ordinarias lineales
`atvalue(`$\varphi(x), x = x_0, val$`)`	permite añadir la condición inicial $\varphi(x_0) = val$ a un determinado sistema de ecuaciones diferenciales ordinarias lineales

Solución de sistemas de ecuaciones diferenciales ordinarias lineales.

Maxima

He aquí la solución de la ecuación diferencial $y''(x) + y(x) = 2x$.

(%i82) desolve('diff(y(x),x,2) + y(x) = 2 * x, y(x));

(%o82) $y(x) = \sin(x)\left(\left.\frac{d}{d\,x}y(x)\right|_{x=0} - 2\right) + y(0)\cos(x) + 2\,x$

Maxima

Esto resuelve el sistema $\begin{cases} x'(t) = y(t) \\ y'(t) = x(t) \end{cases}$.

(%i83) desolve(['diff(x(t),t) = y(t), 'diff(y(t),t) = x(t)], [x(t), y(t)]);

(%o83) $x(t) = \frac{(y(0)+x(0))\,\%e^{t}}{2} - \frac{(y(0)-x(0))\,\%e^{-t}}{2},\; y(t) = \frac{(y(0)+x(0))\,\%e^{t}}{2} + \frac{(y(0)-x(0))\,\%e^{-t}}{2}$

Maxima

De esta manera se añaden las las condiciones iniciales $x(0) = 1$, $y(0) = 0$ al sistema anterior.

(%i84) atvalue(x(t), t = 0, 1);

(%o84) 1

(%i85) atvalue(y(t),t=0,0);

(%o85) 0

Maxima

Al volver a resolver el sistema se obtienen las soluciones con las condiciones iniciales dadas.

(%i86) desolve(['diff(x(t),t) = y(t), 'diff(y(t),t) = x(t)], [x(t), y(t)]);

(%o86) $x(t) = \frac{\%e^{t}}{2} + \frac{\%e^{-t}}{2},\; y(t) = \frac{\%e^{t}}{2} - \frac{\%e^{-t}}{2}$

7.10 Series de potencias

`ta lor`(*expr*, *x*, *a*, *n*)	expande la expresión *expr* en un desarrollo de Taylor o de Laurent respecto de la variable *x* alrededor del punto *a*, con términos hasta $(x-a)^n$
`ta lor`(*expr*, [x_1, x_2, ...], *a*, *n*)	devuelve la serie en potencias truncada de grado *n* en todas las variables x_1, x_2, ... alrededor del punto $(a, a, \ldots)$
`ta lor`(*expr*, [x_1, a_1, n_1], [x_2, a_2, n_2], ...)	devuelve la serie en potencias truncada en las variables x_1, x_2, ... alrededor del punto $(a_1, a_2, \ldots)$; el truncamiento se realiza, respectivamente, en los grados $n_1, n_2, \ldots$
`ta lor`(*expr*, [x_1, x_2, ...], [$a_1, a_2, \ldots$], [$n_1, n_2, \ldots$], ...)	equivale a `ta lor`(*expr*, [x_1, a_1, n_1], [x_2, a_2, n_2], ...)
`ta lor`(*expr*, [*x*, *a*, *n*, `' as mp`])	desarrolla *expr* en potencias negativas de $x-a$ (el término de mayor orden es $(x-a)^{-n}$)

Obtención de series de potencias.

Las operaciones matemáticas de las que se ha hablado hasta ahora son exactas. Considerando la entrada exacta, sus resultados son fórmulas exactas. En muchas situaciones, sin embargo, no se necesita un resultado exacto. Puede ser suficiente, por ejemplo, encontrar una fórmula aproximada que es válida, digamos, cuando la cantidad x es pequeña.

Maxima

Esto da una aproximación en serie de potencias para alrededor de 0, hasta los términos de orden 3.

```
(%i87)  taylor((1 + x)^n, x, 0, 3);
```

$$(\%o87)\quad 1 + n\,x + \frac{\left(n^2 - n\right) x^2}{2} + \frac{\left(n^3 - 3\,n^2 + 2\,n\right) x^3}{6} + \cdots$$

Maxima

Maxima conoce las expansiones en serie de potencias para una gran cantidad funciones matemáticas.

```
(%i88)  taylor(exp(-a*t)*(1+sin(2*t)),t,0,4);
```

$$(\%o88)\quad 1+(-a+2)\,t+\frac{(a^2-4a)\,t^2}{2}-\frac{(a^3-6a^2+8)\,t^3}{6}+\frac{(a^4-8a^3+32a)\,t^4}{24}+\ldots$$

Maxima

Si se da una función no conocida, `taylor` escribe la serie de potencias en términos de derivadas.

```
(%i89)  taylor(1+f(t),t,0,3);
```

$$(\%o89)\quad 1+f(0)+\left(\frac{d}{dt}f\right)_{t=0}t+\frac{\left(\frac{d^2}{dt^2}f(t)\right)_{t=0}t^2}{2}+\frac{\left(\frac{d^3}{dt^3}f(t)\right)_{t=0}t^3}{6}+\ldots$$

Las series de potencias son fórmulas aproximadas que juegan el mismo papel con respecto a las expresiones algebraicas como los números aproximados con las expresiones numéricas.

Maxima permite realizar operaciones en series de potencias y en todos los casos mantiene el orden apropiado o el "grado de precisión" para las series de potencias resultantes.

Maxima

He aquí una serie de potencias simple, de orden 3.

```
(%i90)  taylor(exp(x),x,0,3);
```

$$(\%o90)\quad 1+x+\frac{x^2}{2}+\frac{x^3}{6}+\ldots$$

Maxima

Cuando se hacen operaciones en una serie de potencias, el resultado es calculado sólo en el orden apropiado en x.

```
(%i91)  %^2 * (1 + %);
```

$$(\%o91)\quad 2 + 5\,x + \frac{13\,x^2}{2} + \frac{35\,x^3}{6} + \cdots$$

Maxima

Al copiar los tres primeros términos del desarrollo anterior se tiene una expresión ordinaria.

```
(%i92)  2 + 5 * x + (13 * x^2)/2 + (35 * x^3)/6;
```

$$(\%o92)\quad \frac{35\,x^3}{6} + \frac{13\,x^2}{2} + 5\,x + 2$$

Maxima

Ahora el cuadrado es calculado exactamente.

```
(%i93)  %^2;
```

$$(\%o93)\quad \left(\frac{35\,x^3}{6} + \frac{13\,x^2}{2} + 5\,x + 2\right)^2$$

Maxima

Al aplicar `expand` se obtiene un resultado con once términos.

```
(%i94)  %, expand;
```

$$(\%o94)\quad \frac{1225\,x^6}{36} + \frac{455\,x^5}{6} + \frac{1207\,x^4}{12} + \frac{265\,x^3}{3} + 51\,x^2 + 20\,x + 4$$

Maxima

He aquí una serie de potencias doble de orden 3.

```
(%i95)  taylor(sin(y + x), [x, 0, 3], [y, 0, 3]);
```

$$(\%o95)\quad \left(y - \frac{y^3}{6} + \cdots + \right)\left(1 - \frac{y^2}{2} + \cdots\right) x + \left(-\frac{y}{2} + \frac{y^3}{12} + \cdots\right) x^2 + \left(-\frac{1}{6} + \frac{y^2}{12} + \cdots\right) x^3 + \cdots$$

Un detalle interesante para mencionar es que Maxima, en los casos que sea posible, permite obtener la fórmula general del desarrollo en serie de potencias de una función.

`powerseries`(*expr*, *x*, *a*)	devuelve la forma general del desarrollo en serie de potencias de *expr* para la variable *x* alrededor del punto *a*

Obtención de series de potencias.

Maxima

He aquí una conocida fórmula.

```
(%i96) powerseries(sin(x), x, 0);
```

$$(\%o96)\quad \sum_{i_1=0}^{\infty} \frac{(-1)^{i_1}\, x^{2\,i_1+1}}{(2\,i_1+1)!}$$

Maxima

Esta es una fórmula más elaborada.

```
(%i97) powerseries(cos(x^2-1), x, 0);
```

$$(\%o97)\quad \sin(1)\sum_{i_2=0}^{\infty} \frac{(-1)^{i_2}\, x^{2\,(2\,i_2+1)}}{(2\,i_2+1)!} + \cos(1)\sum_{i_2=0}^{\infty} \frac{(-1)^{i_2}\, x^{4\,i_2}}{(2\,i_2)!}$$

Maxima

Una forma de eliminar los s'Ubíndices de los índices, en las sumas, es usando la función `niceindices`.

```
(%i98) niceindices(powerseries(cos(x^2-1), x, 0));
```

$$(\%o98)\quad \sin(1)\sum_{i=0}^{\infty} \frac{(-1)^{i}\, x^{2\,(2\,i+1)}}{(2\,i+1)!} + \cos(1)\sum_{i=0}^{\infty} \frac{(-1)^{i}\, x^{4\,i}}{(2\,i)!}$$

7.11 Transformada de Laplace

`laplace`(*expr*, *t*, *s*)	calcula la transformada de Laplace de *expr* con respecto de la variable *t* y parámetro de transformación *s*
`ilt`(*expr*, *s*, *t*)	calcula la transformada inversa de Laplace de *expr* con respecto de *s* y parámetro *t*.

Transformadas de Laplace.

Maxima

Esto calcula la transformada de Laplace.

```
(%i99)  laplace(t^3 * exp(a * t), t, s);
                              6
(%o99)                  ------------
                                4
                        (s - a)
```

Maxima

He aquí la transformada inversa.

```
(%i100)  ilt(%, s, t);
                   3  a t
(%o100)           t  %e
```

Capítulo 8

Matemáticas numéricas

8.1 Solución numérica de ecuaciones

`algs s`([$ecu_1, \ldots, ecu_m$], [$x_1, \ldots, x_n$])	resuelve el sistema de ecuaciones polinómicas $ecu_1, \ldots, ecu_m$ para las variables $x_1, \ldots, x_n$
`allroots`(ecu, x)	calcula aproximaciones numéricas de las raíces reales y complejas de la ecuación polinómica ecu para la variable x
`find root`(ecu, x, a, b)	calcula una raíz de la ecuación ecu en el intervalo de aislamiento [a, b]

Búsqueda de raíces numéricas.

Maxima

`solve` devuelve la ecuación ingresada.

```
(%i1)  solve(2 - 4 * x + x^5 = 0, x);
(%o1)  [0 = x^5 - 4 x + 2]
```

Maxima

`algsys` proporciona una lista con soluciones aproximadas.

```
(%i2)  algsys([2 - 4 * x + x^5 = 0], [x]);
```

```
(%o2)  [[x= 1,243596445373759] ,
        [x = −1,438447695329177 %i − 0,11679186122298],
        [x = 1,438447695329177 %i − 0,11679186122298] ,
        [x = −1,518512140520062], [x = 0,50849947534103]]
```

Maxima

La opción `realonly:true` proporciona únicamente las aproximaciones reales.

```
(%i3)  algsys([2 − 4 x + x^5 = 0], [x]), realonly :true;
(%o3)  [[x= 1,243596445373759],[x=0,50849947534103],
        [x = −1,518512140520062]]
```

Si las ecuaciones involucran sólo funciones lineales o polinómicas, entonces puede usarse `algs s` para obtener aproximaciones numéricas de todas las soluciones. Sin embargo, cuando las ecuaciones involucran funciones más complicadas, no hay en general ningún procedimiento sistemático para obtener todas las soluciones, aún numéricamente. En tales casos, puede usarse `find root` para buscar soluciones.

Téngase presente que `find root` espera que la función en cuestión tenga signos diferentes en los extremos del intervalo de aislamiento.

Maxima

Aquí Maxima devuelve la misma sentencia de entrada, pues `find_root` evalúa los extremos del intervalo de aislamiento [0, 2]; y la función log que, en este caso, forma parte de la ecuación no está definida en cero.

```
(%i4)  find_root(3 ∗ cos(x) = log(x), x, 0, 2);
(%o4)  find_root(3 ∗ cos(x) = log(x), x, 0,0, 2,0)
```

Maxima

Variando el extremo izquierdo del intervalo de aislamiento se obtiene una solución aproximada.

```
(%i5)  find_root(3 cos(x) = log(x), x, 0,00001, 2);
(%o5)  1,447258617277903
```

Maxima

La ecuación tiene varias soluciones. Si se da un intervalo de aislamiento diferente, `find_root` puede devolver una solución diferente.

```
(%i6)  find_root(3*cos(x) = log(x), x, 12, 15);
(%o6)  13,10638768062491
```

8.2 Integrales numéricas

`quad_qags`(f, x, a, b)	calcula numéricamente la integral $\int_a^b f\,dx$
`quad_qagi`(f, x, a, inf)	calcula numéricamente la integral $\int_a^\infty f\,dx$
`quad_qagi`$(f, x, minf, b)$	calcula numéricamente la integral $\int_{-\infty}^b f\,dx$
`quad_qagi`$(f, x, minf, inf)$	calcula numéricamente la integral $\int_{-\infty}^\infty f\,dx$

Integrales numéricas.

Las funciones `quad_qags` y `quad_qagi` devuelven una lista de cuatro elementos:

1. la aproximación a la integral,
2. el error absoluto estimado de la aproximación,
3. el número de evaluaciones del integrando,
4. un código de error.

El código de error puede tener los siguientes valores:

0 si no ha habido problemas;

1 si se utilizaron demasiados intervalos;

2 si se encontró un número excesivo de errores de redondeo;

3 si el integrando ha tenido un comportamiento extraño frente a la integración;

4 fallo de convergencia;

5 la integral es probablemente divergente o de convergencia lenta;

6 si los argumentos de entrada no son válidos.

Maxima

`quad_qags` puede manejar singularidades en los puntos finales de la región de integración.

```
(%i7)  quad_qags(1/sqrt(x*(1-x)),x,0,1);
(%o7)  [3.141592653589849,6.2063554295832546*10^-10,567,
        0]
```

Maxima

Para resolver integrales numéricas sobre regiones infinitas se usa `quad_qagi`.

```
(%i8)  quad_qagi(exp(-x^2),x,minf,inf);
(%o8)  [1.772453850905516,1.420263678183091*10^-8,270,0]
```

Capítulo 9

Funciones y programas

9.1 Definición de funciones

Hasta aquí, se ha visto muchos ejemplos de funciones incorporadas en Maxima. En esta sección, se mostrará la forma en que el usuario puede añadir sus propias funciones a Maxima.

Maxima

Esto define la función f.

```
(%i1)  f(x) := x^2;
(%o1)  f(x) := x²
```

Maxima

f eleva al cuadrado su argumento.

```
(%i2)  f(a + 1);
(%o2)  (a + 1)²
```

Maxima

El argumento puede ser un número.

```
(%i3)  f(4);
(%o3)  16
```

Maxima

O puede ser una expresión más complicada.

(%i4) f(x^2 + 3 * x);

(%o4) $(x^2 + 3\,x)^2$

Maxima

Puede usarse f en un cálculo.

(%i5) expand(f(x + y + 1));

(%o5) $y^2 + 2\,x\,y + 2\,y + x^2 + 2\,x + 1$

Maxima

Esto muestra la definición hecha para f.

(%i6) dispfun(f);

(%t6) $f(x) := x^2$

(%o6) [%t38]

f(x) := x^2	define la función f
dispfun(f)	muestra la definición de f
remfunction(f)	borra todas las definiciones de f
functions	es una variable que contiene los nombres de las funciones definidas por el usuario

Definición de una función en Maxima.

Maxima

Las funciones en Maxima pueden tener cualquier número de argumentos

(%i7) hump(x, xmax) := (x - xmax)^2/xmax;

(%o7) $\mathrm{hump}(x, xmax) := \frac{(x - xmax)^2}{xmax}$

Maxima

Puede usarse la función `hump` tal como cualquiera de las funciones predefinidas.

```
(%i8)  2 + hump(x, 3,5);
(%o8)  0,28571428571429 (x – 3,5)^2 + 2
```

Maxima

Esto da una nueva definición para `hump`, que sobrescribe la anterior.

```
(%i9)  hump(x,xmax) := (x – xmax)^4;
(%o9)  hump(x,xmax) := (x – xmax)^4
```

Maxima

Sólo es mostrada la nueva definición.

```
(%i10)  dispfun(hump);
(%t10)  hump(x,xmax) := (x– xmax)^4
(%o10)  [%t42]
```

Maxima

Esto limpia todas las definiciones para `hump`.

```
(%i11)  remfunction(hump);
(%o11)  [hump]
```

`dispfun`($f_1, \ldots, f_n$)	muestra las definiciones de $f_1, \ldots, f_n$
`remfunction`($f_1, \ldots, f_n$)	borra todas las definiciones de $f_1, \ldots, f_n$
`dispfun`(*all*)	muestra las definiciones de todas la funciones definidas por el usuario
`remfunction`(*all*)	borra todas las definiciones de todas la funciones definidas por el usuario

Vista y borrado de varias funciones.

Maxima

Ahora se define la función g.

```
(%i12)  g(x) := sqrt(1 - x);
(%o12)  g(x) := sqrt(1 - x)
```

Maxima

Estro muestra la definición de todas las funciones (en este caso f y g).

```
(%i13)  dispfun(all);
(%t13)  f(x) := x^2
(%t14)  g(x) := sqrt(1 - x)
(%o14)  [%t13, %t14]
```

Maxima

Esto borra la definición de todas las funciones (en este caso f y g).

```
(%i15)  remfunction(all);
(%o15)  [f, g]
```

Maxima

Ya no hay funciones definidas por el usuario.

```
(%i16)  dispfun(all);
(%o16)  [ ]
```

Cuando se ha terminado con una función particular, es bueno limpiar las definiciones que se haya hecho para ella. Si no, podría incurrirse en un conflicto al usar la misma función para un propósito diferente en la sesión de Maxima.

Maxima también permite definir funciones con una cantidad variable de argumentos (funciones de argumento variable) y funciones que se aplican directamente a listas (funciones array).

$F\,[L] := \varphi(L)$	define la función F cuyo número de argumentos es variable
$F\,[x_1, \ldots, x_n] := \varphi(x_1, \ldots, x_n)$	define la función array F cuyo argumento es una lista
$F\,[x_1, \ldots, x_n] := [\varphi_1(x_1, \ldots, x_n), \ldots, \varphi_m(x_1, \ldots, x_n)]$	define la función array F cuyo argumento es una lista y que devuelve una lista

Más definiciones de funciones en Maxima.

Maxima

Aquí se define la función F que admite un número variable de argumentos.

```
(%i17)  F([x]) := x^2 + 4;
(%o17)  F([x]) := x^2 + 4
```

Maxima

Esto evalúa F en un solo argumento.

```
(%i18)  F(2);
(%o18)  8
```

Maxima

Esto evalúa F en dos argumentos.

```
(%i19)  F(2, 3);
(%o19)  [8, 13]
```

Maxima

He aquí la definición de la función array G.

```
(%i20)  G[x,y] := x^2 + y^2;
(%o20)  G_x,y := x^2 + y^2
```

Maxima

Aquí se evalúa `G` en la lista [2, 3]

```
(%i21)  G[2,3];
(%o21)  13
```

Maxima

Esto define la función array `H` que devuelve una lista.

```
(%i22)  H[x,y] := [2 * x, 3 - y, x * y];
(%o22)  H_x,y := [2 * x, 3 - y, x * y]
```

Maxima

Aquí se evalúa la función `H` en la lista [4, 3].

```
(%i23)  H[4,3];
(%o23)  [8, 0, 12]
```

Para visualizar la definición de este tipo de funciones puede usarse la función `dispfun`, pero para borrar las respectivas definiciones debe usarse la función `remarra` .

`remarra` (F)	borra la función array F
`remarra` $(F_1, \ldots, F_n)$	borra las funciones array $F_1, \ldots, F_n$
`remarra` (all)	borra todas las funciones array definidas por el usuario

Borrado de funciones array.

Maxima

Esto la definición de todas las funciones array, hasta ahora, definidas.

```
(%i24)  dispfun(all);
(%t24)  F([x]) := x^2 + 4
(%t25)  G_x,y := x^2 +y^2
```

```
( %t26)   H_x,y := [2x, 3 _ y, xy]
( %o26)   [ %t24, %t25, %t26]
```

Maxima

Esto borra la definición de la función `F` definida en (%i17) .

```
( %i27)   remfunction(F);
( %o27)   [F]
```

Maxima

Esto borra la definición de todas las funciones array definidas por el usuario.

```
( %i28)   remarray(all);
( %o28)   [G, H]
```

Además, Maxima permite definir las llamadas funciones anónimas que son de mucha utilidad en algunos contextos en los que sea necesario un nombre de función y que generalmente está vinculado con la programación. La función para definirlas es lambda.

`lambda`([*x*], *expr*)	define una función anónima
`lambda`([$x_1, \ldots, x_m$], $expr_1, \ldots, expr_n$)	define una función anónima de argumentos múltiples

Función anónima.

Maxima

Una función anónima puede asignarse a una variable y ser evaluada como si fuese una función ordinaria.

```
( %i29)   f : lambda([x], x^2);
( %o29)   lambda([x], x^2)
( %i30)   f(a);
( %o30)   a^2
```

Maxima

O puede aplicarse directamente.

```
( %i31)  lambda([x],x^2)(a);
( %o31)  a^2
```

Maxima

No obstante, no es reconocida por `dispfun`.

```
( %i32)  dispfun(all);
( %o32)  [ ]
```

En muchas clases de cálculos, el usuario puede encontrarse digitando la misma entrada a Maxima muchas veces. Es posible ahorrarse mucha digitación definiendo una función que contiene todas las sentencias de entrada.

Maxima

Esto construye un producto de tres términos, y expande el resultado.

```
( %i33)  expand(product(x+i,i,1,3));
( %o33)  x^3+6*x^2+11*x+6
```

Maxima

Esto hace lo mismo, pero con cuatro términos.

```
( %i34)  expand(product(x+i,i,1,4));
( %o34)  x^4+10*x^3+35*x^2+50*x+24
```

Maxima

Esto define una función `exprod` que construye un producto de `n` términos, luego lo expande.

```
( %i35)  exprod(n) := expand(product(x+i,i,1,n)) $
```

Maxima

Siempre que usa la función, ejecutará las operaciones `product` y `expand`.

```
(%i36)  exprod(5);
(%o36)  x^5 + 15*x^4 + 85*x^3 + 225*x^2 + 274*x + 120
```

Las funciones que se definen en Maxima son esencialmente procedimientos que ejecutan las sentencias dadas por el usuario. Es posible incluir varios pasos en los procedimientos, separados por comas.

Maxima

El resultado que se obtiene de la función es simplemente la última expresión en el procedimiento. Note que debe ponerse paréntesis alrededor del procedimiento cuando se define.

```
(%i37)  cex(n,i) := (t:exprod(n),
        coeff(t, x^i) )$
```

Maxima

Esto "corre" el procedimiento.

```
(%i38)  cex(5,3);
(%o38)  85
```

(*expr*1, *expr*2, . . .)	una secuencia de expresiones para evaluar (procedimiento)
`block`([*a*, *b*, . . .], *proc*)	un procedimiento con variables locales *a*, *b*, . . .

Construcción de procedimientos.

Una buena idea es declarar, como locales, las variables que se usan dentro de los procedimientos, de modo que no interfieran con cálculos fuera de éstos. Puede hacerse esto estableciendo los procedimientos como **bloques**, en los cuales se da una lista de variables para que sean tratadas como locales.

Maxima

La función `cex` definida en (%i37) no es un bloque, así que el valor de `t` "escapa", y existe incluso después de la evaluación de la función.

```
(%i39)  t;
(%o39)  x^5 + 15 x^4 + 85 x^3 + 225 x^2 + 274 x + 120
```

Maxima

Esta función es definida como un bloque con variable local `u`.

```
(%i40)  ncex(n, i) :=
        block([u] ,
        u : exprod(n),
        coeff(u, x^i) ) $
```

Maxima

La función devuelve el mismo resultado que la anteriormente definida.

```
(%i41)  ncex(5,3);
(%o41)  85
```

Maxima

Ahora, sin embargo, el valor de u no se escapa de la función.

```
(%i42)  u
(%o42)  u
```

Es posible asignar un valor inicial a una o más variables locales dentro de la lista de las mismas. También se puede definir una función como bloque sin declarar variables locales, para lo cual debe omitirse dicha lista.

Un bloque puede aparecer dentro de otro bloque. Las variables locales se inicializan cada vez que se entra dentro de un nuevo bloque. Las variables locales de un bloque se consideran globales dentro de otro anidado dentro del primero. Si una variable es no local dentro

de un bloque, su valor es el que le corresponde en el bloque superior. Este criterio se conoce con el nombre de "alcance dinámico".

9.2 Reglas de transformación para funciones

Maxima permite al usuario definir sus propias reglas de transformación y patrones de comparación.

Maxima

Definición de la regla de transformación que reemplaza `x` por 3.

```
(%i43)  defrule(regtran1,x,3);
(%o43)  regtran1 : x → 3
```

Maxima

Aplicación de la regla de transformación `regtran1`

```
(%i44)  apply1(1+f(x)+f(y),regtran1);
(%o44)  f(y) + f(3) + 1
```

Maxima

Puede definirse una regla de transformación para `f(x)`. Esta regla no afecta a `f(y)`.

```
(%i45)  defrule(regtran2, f(x), p);
(%o45)  regtran2 : f(x) → p
(%i46)  apply1(1+f(x)+f(y),regtran2);
(%o46)  f(y) + p + 1
```

Maxima

Esto define un patrón `f(t)` que admite cualquier argumento para f.

```
(%i47)  matchdeclare(t, true);
(%o47)  done
(%i48)  defrule(patron, f(t), t^2);
(%o48)  patron : f(t) → t^2
```

Maxima

Aquí se aplica el patrón previamente definido.

```
(%i49)  apply1(1+f(x)+f(y),patron);
(%o49)  y^2 + x^2 + 1
```

Maxima

Esto muestra todas las reglas, hasta aquí, definidas.

```
(%i50)  disprule(all);
(%t51)  regtran1 : x → 3
(%t52)  regtran2 : f(x) → p
(%t53)  patron : f(t) → t^2
(%o53)  [%t51, %t52, %t53]
```

Maxima

Con esta sentencia se borran todas las definiciones de las reglas `regtran1`, `regtran2` y `patron`.

```
(%i54)  clear rules();
(%o54)  false
```

Maxima

Esto indica que todas las definiciones de las reglas han sido borradas.

```
(%i55)  disprule(all);
(%o55)  [ ]
```

Probablemente, el aspecto más potente de las reglas de transformación en Maxima es que ellas pueden involucrar expresiones no sólo literales, sino también patrones. Un patrón es una expresión genérica `f(t)` para la cual se ha declarado el tipo de variable `t` con `matchdeclare`. Así, una regla de transformación para `f(t)` especifica cómo debería ser transformada la función `f` con el tipo de argumento especificado. Nótese que, en contraste, una regla de transformación

para `f(x)` sin realizar la declaración de la variable, especifica sólo la transformación de la expresión literal `f(x)`, y por ejemplo, no dice nada sobre la transformación de `f( )`.

Siempre que se cumpla con la declaración especificada, es posible establecer reglas de transformación para expresiones de cualquier forma.

Maxima

Esto define una función de predicado que define el patrón de producto para un argumento.

```
(%i56) prodp(expr):=not atom(expr) and op(expr)="*";
(%o56) prodp(expr):=¬ atom(expr) ∧ op(expr) = "*"
(%i57) matchdeclare(p, prodp);
(%o57) done
```

Maxima

Ahora se define una regla que transforme una función aplicada a un producto como la suma de los factores a los cuales se les ha aplicado la función.

```
(%i58) defrule(reg, f(p), apply("+", map(f, args(p))));
(%o58) reg : f(p) → apply(+, map(f, args(p)))
```

Maxima

Luego, al aplicar la regla recién definida se aprecia el efecto producido.

```
(%i59) apply1(f(a*b)+f(c*d), reg);
(%o59) f(d)+f(c)+f(b)+f(a)
```

Maxima

Esto aplica la regla sin limitar el número de los factores en el argumento de `f`.

```
(%i60) apply1(f(a)+f(a*b*c)+f(c), reg);
(%o60) 2f(c)+f(b)+2f(a)
```

Maxima

Esto combina la regla definida por el usuario con la función `expand` de Maxima.

```
(%i61)  apply1(f(a) f(a b c) f(c), reg), expand;
(%o61)  f(a) f²(c) + f(a) f(b) f(c) + f²(a) f(c)
```

9.3 Funciones definidas a partir de expresiones

En muchos casos, sobre todo en programación, resulta bastante útil poder definir una función a partir de una expresión dada para luego poder invocarla con facilidad y así evitar el uso reiterado de la función `ev` (ver sección 6.2).

`define`($f(x_1,\ldots,x_n)$, *expr*)	define una función de nombre f con argumentos $x_1,\ldots,x_n$ y cuerpo *expr*
`define`($f[x_1,\ldots,x_n]$, *expr*)	define una función array de nombre f con argumentos $x_1,\ldots,x_n$ y cuerpo *expr*

Definición de funciones a partir de expresiones.

Maxima

Esto almacena la expresión $x^2 + 1$ en la variable *ex*.

```
(%i62)  ex : x^2 + 1;
(%o62)  x² + 1
```

Maxima

Aquí se pretende definir la función *f* a partir de *ex*. No obstante, al realizar una evaluación no se obtiene el resultado esperado.

```
(%i63)  f(x) := ex$
(%i64)  f(8);
(%o64)  x² + 1
```

Maxima

Por otra parte, al utilizar `define` para definir la función g a partir de ex si se obtiene un resultado satisfactorio.

```
(%i65)  define(g(x), ex)$
(%i66)  g(8);
(%o66)  65
```

Una gran utilidad de `define` se aprecia al definir funciones a partir de las derivadas de otras, las cuales serán evaluadas en valores numéricos.

Maxima

He aquí la definición de la función $f = x^2 + 1$.

```
(%i67)  f(x) := x^2 + 1$
```

Maxima

Ahora, a partir de f, se define $''f'$ como $''f' = \frac{df}{dx}$.

```
(%i68)  ''f'(x) := diff(f(x), x)$
```

Maxima

La evaluación sombólica de $''f'$ no presenta problema alguno.

```
(%i69)  ''f'(t);
(%o69)  2t
```

Maxima

No obstante, la evaluación numérica no esta permitida para dicha función.

```
(%i70)  ''f'(8);
diff: second argument must be a variable; found 8
#0: f'(x 8)
- - an error. To debug this try: debugmode(true);
```

Maxima

Aquí se vuelve a definir $''f'$, pero en este caso se utiliza la función `define` (recuerde que esta nueva definición anula a la anterior).

```
(%i71)  define(''f'(x), diff(f(x), x)) $
```

Maxima

La evaluación simbólica es idéntica.

```
(%i72)  ''f'(t);
(%o72)  2t
```

Maxima

Sin embargo, ya no hay dificultad en la evaluación numérica.

```
(%i73)  ''f'(8);
(%o73)  16
```

Capítulo 10

Listas

10.1 Juntar objetos

Las listas son bloques de construcción básica para Maxima y isp[1]. Todos los tipos de datos diferentes a los arreglos, tablas mixtas o números son representados como listas en isp.

Puede decirse que, en un nivel básico, lo que esencialmente hace una lista en Maxima es proporcionar una forma de agrupar varias expresiones de cualquier clase.

Maxima

He aquí una lista de números.

```
(%i1)  [2,3,4];
(%o1)  [2,3,4]
```

Maxima

Esto da una lista de expresiones simbólicas.

```
(%i2)  x^% - 1;
(%o2)  [x^2 - 1, x^3 - 1, x^4 - 1]
```

[1] Lisp (List Processing) es el lenguaje de programación de alto nivel en el que está programado Maxima.

Maxima

Es posible derivar estas expresiones.

```
(%i3)  diff(%, x);
(%o3)  [2x, 3x^2, 4x^3]
```

Maxima

Y luego encontrar los valores cuando x es reemplazado por 3.

```
(%i4)  %, x = 3;
(%o4)  [6, 27, 108]
```

La mayoría de funciones matemáticas incorporadas en Maxima han sido implementadas "listables" de modo que actúen separadamente sobre cada elemento de una lista.

10.2 Generación de listas

Maxima permite crear listas de acuerdo a una cierta regla definida por el usuario. Para ello cuenta con la función incorporada `create list`.

`create list`(*f*, *i*, i_{min}, i_{max})	da una lista de valores con *i* variando de i_{min} a i_{max}
`create list`(*f*, *i*, *list*)	da una lista de valores con *i* variando de acuerdo a cada elemento de *list*
`create list`(*f*, *i*, i_{min}, i_{max}, *j*, j_{min}, j_{max}, . . .)	genera una lista con cada índice $(i, j, \ldots)$ variando del valor mínimo al valor máximo que le han sido asociados
`create list`(*f*, *i*, $list_1$, *j*, $list_2$, . . .)	genera una lista con cada índice $(i, j, \ldots)$ variando de acuerdo a cada elemento de la lista que se le ha asociado

Funciones para generar de listas.

Maxima

Esto da una lista de los valores de i^2, con i variando de 1 a 6.

```
(%i5)  create list(i^2, i, 1, 6);
(%o5)  [1, 4, 9, 16, 25, 36]
```

Maxima

He aquí una lista de los valores de $\text{sen}\left(\frac{n}{5}\right)$ para n variando de 0 a 4.

```
(%i6)  create list(sin(n/5), n, 0, 4);
(%o6)  [0, sin(1/5), sin(2/5), sin(3/5), sin(4/5)]
```

Maxima

Esto da los valores numéricos de la tabla generada en (%i6).

```
(%i7)  %, numer;
(%o7)  [0, 0, 0.19866933079506, 0.38941834230865,
        0.56464247339504, 0.71735609089952]
```

Maxima

También puede hacerse tablas de fórmulas.

```
(%i8)  create list(x^i + 2*i, i, 1, 5);
(%o8)  [x + 2, x^2 + 4, x^3 + 6, x^4 + 8, x^5 + 10]
```

Maxima

`create_list` usa exactamente la misma notación para el iterador que las funciones `sum` y `product` que fueron mencionadas en la sección 7.4.

```
(%i9)  product(x^i + 2*i, i, 1, 5);
(%o9)  (x + 2) (x^2 + 4) (x^3 + 6) (x^4 + 8) (x^5 + 10)
```

Maxima

Esto hace una lista con valores de x variando desde 0 hasta 1 en pasos de 0,25

```
(%i10) create list(i/4, i, 1, 4), numer;
(%o10) [0,25, 0,5, 0,75, 1]
(%i11) create list(sqrt(x), x, %);
(%o11) [0,5, 0,70710678118655, 0,86602540378444, 1]
```

Maxima

Es posible realizar otras operaciones en las listas que se obtienen con `create_list`.

```
(%i12)  %^2 +3;
(%o12) [3,25, 3,5, 3,75, 4]
```

Maxima

Esto hace una lista de $x^i + y^j$ con i variando desde 1 hasta 3 y j variando desde 1 hasta 2.

```
(%i13) create list(x^i + y^j, i, 1, 3, j, 1, 2);
(%o13) [y + x, y^2 + x, y + x^2, y^2 + x^2, y + x^3, y^2 + x^3]
```

Maxima

Esto crea una lista conteniendo cuatro copias del símbolo x.

```
(%i14) create list(x, i, 1, 4);
(%o14) [x, x, x, x]
```

Maxima

Esto da una lista de cuatro números seudo aleatorios.

```
(%i15) create list(random(1,0), i, 1, 4);
(%o15) [0,25223887668538,  0,41556272272148,
         0,61257388925382,  0,84181265533592]
```

10.3 Elección de elementos de una lista

Los elementos de las listas son numerados en orden ascendente, empezando en 1. Por lo cual, es posible elegir un elemento de una lista en Maxima mediante su "índice".

Para realizar la mencionada elección debe utilizarse adecuadamente la función `part`.

`part`(*list*, *i*) ó *list* [*i*]	el *i*-ésimo elemento de *list* (el primer elemento es *list* [1])
`part`(*list*, [*i*, *j*, . . .])	una lista formada por los elementos *i*, *j*, . . . de *list*
`part`(*list*, $i_1, \ldots, i_k$)	obtiene la parte de *list* que se especifica por los índices $i_1, \ldots, i_k$ (primero se obtiene la parte i_1 de *list*, después la parte i_2 del resultado anterior, y así sucesivamente)
`first`(*list*), `second`(*list*), . . . , `tenth`(*list*)	devuelve el primer, segundo, ..., décimo elemento de *list*
`last`(*list*)	devuelve el último elemento de *list*

Operaciones con elementos de una lista.

Maxima

Esto extrae el tercer elemento de la lista.

```
( %i16)  part([4,-1, 8, 6] , 3);
( %o16)  8
```

Maxima

Esto extrae una lista de elementos.

```
( %i17)  part([4,-1,8,-6],[2,3,1,2,3,4,1]);
( %o17)  [-1,8,4,-1,8,-6,4]
```

Maxima

Esto asigna una lista en la variable u.

```
(%i18)  u : [7, 2, 4, 6];
(%o18)  [7, 2, 4, 6]
```

Maxima

Puede extraerse elementos de u.

```
(%i19)  u[3];
(%o19)  4
```

Maxima

Pueden realizarse operaciones diversas con u.

```
(%i20)  (u + 1)/(u - 6);
(%o20)  -3/4, -1/6, -6, 8
```

Es posible crear listas multidimensionales mediante la anidación de listas en listas.

Maxima

Esto crea una lista 2×2, y le da el nombre m.

```
(%i21)  m : create list(create list(i - j, j, 1, 2),
        i, 1, 2);
(%o21)  [[0, -1], [1, 0]]
```

Maxima

Esto extrae la primera sublista de la lista de listas, m

```
(%i22)  m[1];
(%o22)  [0, -1]
```

Maxima

Esto extrae el segundo elemento de aquella sublista.

```
(%i23)  %[2];
(%o23)  -1
```

Maxima

Esto hace las dos operaciones juntas.

```
(%i24)  part(m,1,2);
(%o24)  -1
```

Maxima

Esto genera una lista tridimensional de $2 \times 2 \times 2$. Es una lista de listas de listas.

```
(%i25)  create list(create list(create list(i *j^2 *k^3,
        k,1,2),j,1,2),i,1,2);
(%o25)  [[[1,8],[4,32]],[[2,16],[8,64]]]
```

Maxima

Si no se anida `create_list` se obtiene una lista unidimensional.

```
(%i26)  create list(i *j^2 *k^3,i,1,2,j,1,2,k,1,2);
(%o26)  [1,8,4,32,2,16,8,64]
```

Al asignar una lista a una variable, puede usarse las listas en Maxima de modo similar a los "arrays" en otros lenguajes de programación. De esta manera, por ejemplo, es posible resetear un elemento de una lista asignando un valor a `u [ i ]`.

`part(u, i)` ó `u [i]`	extrae el *i*-ésimo elemento de la lista `u`
`u [i] : valor`	resetea el i-ésimo elemento de la lista `u`

Operando listas como arrays.

Maxima

Aquí hay una lista.

```
(%i27)  u : [2, 1, 5, 7];
(%o27)  [2, 1, 5, 7]
```

Maxima

Esto resetea el segundo elemento de la lista.

```
(%i28)  u [2] : 0;
(%o28)  0
```

Maxima

Ahora la lista asignada a `u` se ha modificado.

```
(%i29)  u;
(%o29)  [2, 0, 5, 7]
```

10.4 Prueba y búsqueda de elementos de una lista

Maxima también tiene funciones que buscan y prueban elementos de listas, basándose en los valores de dichos elementos.

`sublist`(*L*, *P*)	devuelve la lista de elementos *x* de la lista *L* para los cuales el predicado *P* (*x*) es verdadero
`sublist_indices`(*L*, *P*)	devuelve los índices de los elementos *x* de la lista *L* para la cual el predicado *P* (*x*) es verdadero
`?position`(*elem*, *L*)	devuelve el índice que indica la posición que ocupa *elem* en la lista *L*
`member`(*form*, *list*)	prueba si *form* es un elemento de *list*

Funciones para la prueba y búsqueda de elementos de una lista.

`freeof`(*form*, *list*)	prueba si *form* no ocurre en ninguna parte *list*
`freeof`(*form*$_1$, ..., *form*$_n$, *list*)	equivale a `freeof`(*form*$_1$, *list*) `and` ... `and` `freeof`(*form*$_n$, *list*)

Funciones para la prueba y búsqueda de elementos de una lista.

Maxima

Esto da una lista de todos los elementos, de la lista [*a*, 1, 3, *b*, 7], que son de tipo simbólico.

```
(%i30)  sublist([a,1,3,b,7],symbolp);
(%o30)  [a,b]
```

Maxima

Esto da una lista de todas posiciones, de la lista [*a*, 1, 3, *b*, 7], en las que se encuentran elementos de tipo simbólico.

```
(%i31)  sublist_indices([a,1,3,b,7],symbolp);
(%o31)  [1,4]
```

Maxima

En este caso se emplea una función anónima para definir un predicado, el cual indica que se quiere obtener una lista de todos los elementos, de la lista [1, 2, 4, 7, 6, 2], que son mayores que 2.

```
(%i32)  sublist([1,2,4,7,6,2],lambda([h],h>2));
(%o32)  [4,7,6]
```

Maxima

Esto muestra que 4 es un elemento de [11, 1, 2, 4, 5, 5].

```
(%i33)  member(4,[11,1,2,4,5,5]);
(%o33)  true
```

Maxima

Por otra parte 3, no lo es.

```
(%i34)  member(3,[11,1,2,4,5,5]);
(%o34)  false
```

Maxima

Esto muestra que 0 ocurre en alguna parte de [[1, 2] , [3, 0]].

```
(%i35)  freeof(0,[[1,2],[3,0]]);
(%o35)  false
```

10.5 Combinación de listas

Maxima permite combinar dos listas alternando sus elementos.

`join`(*list*$_1$, *list*$_2$)	crea una nueva lista con los elementos de las listas *list*$_1$ y *list*$_2$ alternados

Combinar listas.

Maxima

He aquí dos listas, de igual longitud, combinadas.

```
(%i36)  join([1,2],[a,b]);
(%o36)  [1,a,2,b]
```

Maxima

He aquí la combinación de dos listas de diferente longitud (`join` ignora los elementos sobrantes de la lista de mayor longitud).

```
(%i37)  join([1,2],[a,b,c,d]);
(%o37)  [1,a,2,b]
```

10.6 Reordenamiento de listas

El ordenamiento de listas también es permitido por Maxima.

`sort`(*list*)	ordena los elementos de *list* en forma ascendente
`sort`(*list*, *P*)	ordena los elementos de *list* de acuerdo con el predicado *P*
`reverse`(*list*)	invierte el orden de los elementos de *list*
`permutations`(*list*)	devuelve un conjunto con todas las permutaciones distintas de los miembros de *list*
`unique`(*list*)	devuelve *list* sin redundancias, es decir, sin elementos repetidos

Ordenamiento de listas.

Maxima

Esto ordena los elementos de una lista en forma estándar. En casos simples como éste el ordenamiento es alfabético o numérico.

```
(%i38)  sort([d,b,c,a]);
(%o38)  [a, b, c, d]
```

Maxima

Esto ordenada los elementos de la lista en forma descendente.

```
(%i39)  sort([11, 1, 2, 4, 5, 5], " > ");
(%o39)  [11, 5, 5, 4, 2, 1]
```

Maxima

Esto invierte el orden de los elementos de la lista.

```
(%i40)  reverse([d,b,c,a]);
(%o40)  [a, c, b, d]
```

Maxima

Aquí se tienen todas las permutaciones de [*a, b, c*].

```
(%i41)  permutations([a, b,c]);
(%o41)  {[a,b,c],[a,c,b],[b,a,c],[b,c,a],[c,a,b],[c,b,a]}
```

Maxima

De esta forma se eliminan los elementos repetidos en una lista.

```
(%i42)  unique([11,1,2,2,4,5,5]);
(%o42)  [1,2,4,5,11]
```

10.7 Agregar y quitar elementos de una lista

`append`(*list*$_1$, *list*$_2$, . . . , *list*$_n$)	devuelve una lista cuyos elementos son los de la lista *list*$_1$ seguidos de los de *list*$_2$, . . . , *list*$_n$
`cons`(*elem*, *list*)	agrega *elem* al inicio de la lista *list*
`endcons`(*elem*, *list*)	agrega *elem* al final de la lista *list*
`delete`(*elem*, *list*)	borra *elem* de la lista *list*
`rest`(*list*, *n*)	borra los primeros *n* elementos ($n > 0$) o los últimos *n* elementos ($n < 0$) de la lista *list*
`deleten`(*list*, *n*)	borra el n-ésimo elemento de la lista *list*

Agregar y quitar elementos de una lista.

Maxima

Aquí se genera una lista con los elementos de dos listas dadas.

```
(%i43)  append([1,2,3],[a,b,c]);
(%o43)  [1,2,3,a,b,c]
```

Maxima

Aquí se inserta `x` al inicio de la lista dada.

```
(%i44)  cons(x,[a,b,c]);
(%o44)  [x,a,b,c]
```

Maxima

Esto devuelve una lista dada con los dos primeros elementos borrados.

```
(%i45)  rest([a,g,f,c,x],2);
(%o45)  [f,c,x]
```

Maxima

Esto devuelve una lista dada con los dos últimos elementos borrados.

```
(%i46)  rest([a,g,f,c,x],-2);
(%o46)  [a,g,f]
```

10.8 Reorganización de listas

`flatten`(*list*)	elimina todos los niveles que pueda tener la lista *list*
`partition`(*list*, *x*)	particiona la lista *list* en dos listas; una que contiene los elementos que no dependen de *x*, y otra, los que si dependen de *x*

Funciones para reorganizar listas anidadas.

Maxima

He aquí una lista a la que se le eliminan todos los niveles.

```
(%i47)  flatten([7,2,[3,[5]],4]);
(%o47)  [7,2,3,5,4]
```

Maxima

Esto particiona la lista [*a, b, a*^2 + 1, *c*].

```
(%i48)  partition([a,b,a^2+1,c],a);
(%o48)  [[b,c],[a,a^2+1]]
```

10.9 Funciones adicionales para listas

`listp`(*expr*)	verifica si *expr* es una lista
`empt p`(*list*)	verifica si *list* es una lista vacía
`length`(*list*)	devuelve la longitud de *list*
`appl` (f, $[x_1,\ldots,x_n]$)	evalúa $f(x_1,\ldots,x_n)$
`map`(f, $[a_1,\ldots,a_n]$)	devuelve la lista $[f(a_1),\ldots,f(a_n)]$
`map`(f, $[a_1,\ldots,a_n]$, $[b_1,\ldots,b_n],\ldots$)	devuelve la lista $[f(a_1,b_1,\ldots),\ldots, f(a_n,b_n,\ldots)]$
`outermap`(f, *list*$_1$, $\ldots$, *list*$_n$)	aplica la función f a cada uno de los elementos del producto cartesiano *list*$_1$ $\times$ *list*$_2$ $\ldots$ $\times$ *list*$_n$

Algunas funciones adicionales para listas.

Maxima

Con la función `length` es fácil obtener la longitud de una lista.

```
(%i49)  length([7,9,0]);
(%o49)  3
```

Maxima

Aquí se usa la función `apply` para encontrar el elemento de mayor valor.

```
(%i50)  apply(max,[11,1,2,4,5,5]);
(%o50)  11
```

Maxima

He aquí un ejemplo en el que se usa la función `map`.

```
(%i51) map(first,[[a,b],[c,d]]);
(%o51) [a, c]
```

Maxima

Este es otro ejemplo en el que se usa la función `map`. Para usos como este debe recordarse que las listas deben tener la misma cantidad de elementos.

```
(%i52) map("+",[1,2,9],[_1,3,7]);
(%o52) [0, 5, 16]
```

Maxima

He aquí un ejemplo en el que se usa la función `outermap`.

```
(%i53) outermap("^",[a,b],[c,d,e]);
                  c  d  e  c  d  e
(%o53) [[a , a , a ], [b , b , b ]]
```

Capítulo 11

Arrays

`arra` (*nombre*, dim_1, ..., dim_n)	crea un array de dimensión *n*, que debe ser menor o igual que 5
`arra` (*nombre*, *tipo*, dim_1, ..., dim_n)	crea un array con sus elementos del tipo especificado (el tipo puede ser `fixnum` o `flonum`)

Creación de arrays.

`listarra` (*A*)	devuelve una lista con los elementos del array *A*
`arra info`(*A*)	devuelve información acerca del array *A*
`fillarra` (*A*, *B*)	rellena el array *A* con los valores de *B*, que puede ser una lista o, también, otro array

Algunas funciones para arrays.

Maxima

Esto define el array `u` de dimensión 2.

```
(%i1)  array(u,2);
(%o1)  u
```

Maxima

Esto indica que el array no tiene valores asignados.

```
(%i2)  listarray(u);
(%o2)  [#####, #####, #####]
```

Maxima

Esto asigna valores, a partir de una lista, al array.

```
(%i3)  fillarray(u, [1, -7, 8]);
(%o3)  u
```

Maxima

Esto permite visualizar los valores asignados.

```
(%i4)  listarray(u);
(%o4)  [1, -7, 8]
```

Maxima

Aquí se define el array φ.

```
(%i5)  array(phi,1,2);
(%o5)  φ
```

Maxima

Esto asigna valores a φ.

```
(%i6)  phi 0,0 :-1$ phi 0,1 :1$ phi 0,2 :4$ phi 1,0 :4$
       phi 1,1 :7$ phi 1,2 :5$
```

Maxima

Aquí se tiene una lista con los valores asignados a φ.

```
(%i12)  listarray(phi);
```

```
(%o12)  [-1,1,4,4,7,5]
```

Maxima

Esto da información acerca de φ

```
(%i13)  arrayinfo(phi);
(%o13)  [declared,2,[1,2]]
```

Si el usuario asigna un valor a una variable subindicada antes de declarar el array correspondiente, entonces se construye un array no declarado.

Los arrays no declarados, también conocidos por el nombre de "arrays de claves" (hashed arrays), son más generales que los arrays declarados. El usuario no necesita declarar su tamaño máximo y pueden ir creciendo de forma dinámica. Los subíndices de los arrays no declarados no necesariamente deben ser números.

Maxima

Esto asigna un valor a la variable `t` subindicada con `hola`.

```
(%i14)  t[hola]:a;
(%o14)  a
```

Maxima

Esto asigna un valor a la variable `t` subindicada con `adios`.

```
(%i15)  t[adios]:b;
(%o15)  b
```

Maxima

Ahora se muestran los valores asignados al array no declarado `t`.

```
(%i16)  listarray(t);
(%o16)  [b,a]
```

Maxima

Y también se muestra información con respecto a t.

```
(%i17) arrayinfo(t);
(%o17) [hashed, 1, [adios], [hola]]
```

Capítulo 12

Matrices

12.1 Generación de Matrices

Las matrices en Maxima son expresiones que comprenden el operador `matrix` y cuyos argumentos son listas (ver sección 17.2). Esto indica que Maxima tiene establecida una diferencia entre una matriz y una lista de listas.

`matrix`([a_{11}, a_{12}, ..., a_{1n}] , ..., [a_{m1}, a_{m2}, ..., a_{mn}])	devuelve una matriz de orden $m \times n$
`genmatrix`(f, m, n)	devuelve una matriz $m \times n$ generada a partir de f
`entermatrix`(m, n)	devuelve una matriz de orden $m \times n$, cuyos elementos son leidos de forma interactiva
`ident`(n)	devuelve la matriz identidad de orden n
`eromatrix`(m, n)	devuelve una matriz rectangular $m \times n$ con todos sus elementos iguales a cero
`diagmatrix`(n, *elem*)	devuelve una matriz diagonal de orden n con los elementos de la diagonal todos ellos iguales a *elem*

Funciones para generar matrices.

`diag matrix`$(d_1, d_2, \ldots, d_n)$	devuelve una matriz diagonal con los elementos de la diagonal iguales a $d_1, d_2, \ldots, d_n$
`vandermonde matrix` $([x_1, \ldots, x_n])$	devuelve una matriz n por n, cuya i-ésima fila es $[1, x_i, x_i^2, \ldots, x_i^{(n-1)}]$
`hilbert matrix` (n)	devuelve la matriz de Hilbert n por n (si n no es un entero positivo, emite un mensaje de error)

Funciones para generar matrices.

Maxima

He aquí la definición de una matriz.

```
(%i1) matrix([3,1],[2,4]);
(%o1)                 3 1
                      2 4
```

Maxima

Una lista de listas no es interpretada como matriz.

```
(%i2) [[3,1],[2,4]];
(%o2)                [[3,1],[2,4]]
```

Maxima

Con `apply` puede convertirse una lista de listas a matriz.

```
(%i3) apply(matrix,[[3,1],[2,4]]);
(%o3)                 3 1
                      2 4
```

Maxima

Aquí se genera una matriz $(a_{i,j}) \in M_{3\times3} / a_{i,j} = i - 2j$.

```
(%i4) a[i,j]:=i-2*j$
(%i5) genmatrix(a,3,3);
```

```
                [ -1  -3  -5 ]
(%o5)           [  0  -2  -4 ]
                [  1  -1   3 ]
```

Maxima

Esto genera una matriz de Vandermonde simbólica.

```
(%i6) vandermonde_matrix([x[0], x[1], x[2]]);
                [ 1  x0  x0^2 ]
(%o6)           [ 1  x1  x1^2 ]
                [ 1  x2  x2^2 ]
```

Maxima

Esto genera una matriz de Vandermonde numérica.

```
(%i7) vandermonde_matrix([2,3,5]);
                [ 1  2   4  ]
(%o7)           [ 1  3   9  ]
                [ 1  5  25  ]
```

Maxima

He aquí una matriz de Hilbert de orden 3.

```
(%i8) hilbert_matrix(3);
                [  1   1/2  1/3 ]
(%o8)           [ 1/2  1/3  1/4 ]
                [ 1/3  1/4  1/5 ]
```

12.2 Elegir elementos de matrices

M [i] ó `part`(M, i)	devuelve la i-ésima fila de la matriz M
M [i, j] ó `part`(M, i, j)	devuelve el elemento $M_{i,j}$ de la matriz M

Funciones para elegir elementos de matrices.

Maxima

Esto genera la matriz A mediante una función anónima.

```
(%i9)  A:genmatrix(lambda([i,j],i-j),3,3);
                                [  0   -1   -2 ]
                                [              ]
(%o9)                           [  1    0   -1 ]
                                [              ]
                                [  2    1    0 ]
```

Maxima

Aquí se selecciona el elemento $A_{1,2}$.

```
(%i10)  A[1,2];
(%o10)  -1
```

Maxima

Esto devuelve una matriz identidad de orden 3.

```
(%i11)  ident(3);
                                [ 1  0  0 ]
                                [         ]
(%o11)                          [ 0  1  0 ]
                                [         ]
                                [ 0  0  1 ]
```

Maxima

Esto selecciona la segunda fila de la matriz identidad.

```
(%i12)  %[2];
(%o12)  [0,1,0]
```

12.3 Operaciones matriciales

$A.B$	producto matricial
`invert`(M)	devuelve la inversa de la matriz M

Algunas operaciones matriciales.

$M\hat{}\hat{}p$	exponenciación matricial no conmutativa
`determinant`(M)	devuelve el determinante de la matriz M
`mat trace`(M)	devuelve la traza de la matriz cuadrada M
`transpose`(M)	devuelve la transpuesta de la matriz M
`minor`(M, i, j)	devuelve el menor (i, j) de la matriz M
`adjoint`(M)	devuelve la matriz adjunta de la matriz M
`addcol`$(M, list_1, \ldots, list_n)$	añade la(s) columna(s) dada(s) por la(s) lista(s) (o matrices) a la matriz M
`addrow`$(M, list_1, \ldots, list_n)$	añade la(s) fila(s) dada(s) por la(s) lista(s) (o matrices) a la matriz M
`col`(M, i)	devuelve la i-ésima columna de la matriz M. El resultado es una matriz de una sola columna
`row`(M, i)	devuelve la i-ésima fila de la matriz M. El resultado es una matriz de una sola fila
`rank`(M, i, j)	calcula el rango de la matriz M
`setelmx`(x, i, j, M)	asigna el valor x al (i,j)-ésimo elemento de la matriz M y devuelve la matriz actualizada
`submatrix`$(i_1, \ldots, i_m, M, j_1, \ldots, j_n)$ `submatrix`$(i_1, \ldots, i_m, M)$ `submatrix`$(M, j_1, \ldots, j_n)$	devuelve una nueva matriz formada a partir de la matriz M pero cuyas filas $i_1, \ldots, 1_m$ y columnas $j_1, \ldots, j_n$ han sido eliminadas
`echelon`(M)	devuelve la forma escalonada de la matriz M, obtenida por eliminación gaussiana
`triangulari e`(M)	devuelve la forma triangular superior de la matriz M, obtenida por eliminación gaussiana

Algunas operaciones matriciales.

`rowop`(M, i, j, λ)	devuelve la matriz que resulta de relizar la transformación $F_i \leftarrow F_i - \lambda * F_j$ con las filas F_i y F_j de la matriz M
`rowswap`(M, i, j)	intercambia las filas i y j de la matriz M
`columnop`(M, i, j, λ)	devuelve la matriz que resulta de relizar la transformación $C_i \leftarrow C_i - \lambda * C_j$ con las columnas C_i y C_j de la matriz M
`columnswap`(M, i, j)	intercambia las columnas i y j de la matriz M
`kronecker product`(A, B)	devuelve el producto de Kronecker de las matrices A y B

Algunas operaciones matriciales.

Maxima

He aquí una matriz 2×2 de variables simbólicas.

```
(%i13) A : matrix([a, b], [c, d]);
(%o13)                  [ a  b ]
                        [ c  d ]
```

Maxima

He aquí la transpuesta de A.

```
(%i14) transpose(A);
(%o14)                  [ a  c ]
                        [ b  d ]
```

Maxima

Esto da la inversa de A en forma simbólica.

```
(%i15) invert(A);
(%o15)                  [  d     - b    ]
                        [ ad - bc  ad - bc ]
                        [                  ]
                        [   c      a   ]
                        [ - ad - bc  ad - bc ]
```

Maxima

Esto da el determinante de A.

```
(%i16)  determinant(A);
(%o16)  ad − bc
```

Maxima

He aquí una matriz racional 3×3.

```
(%i17)  B : genmatrix(lambda([i,j], 1/(i + j − 1)), 3, 3);
```

$$(\%o17)\quad \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

Maxima

Esto da su inversa.

```
(%i18)  invert(B);
```

$$(\%o18)\quad \begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{bmatrix}$$

Maxima

Multiplicando la inversa con la matriz original se obtiene como resultado la matriz identidad.

```
(%i19)  %.B;
```

$$(\%o19)\quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Maxima

Aquí se define una matriz M de orden 4×5.

```
(%i20)  M : matrix([2, −1, 2, −1, −8], [2, −2, 3, −3, −20],
        [1, 1, 1, 0, −2], [1, −1, 4, 3, 4]);
```

```
(%o20)  matrix([2, -1, 2, -1, -8],
                [2, -2, 3, -3, -20],
                [1, 1, 1, 0, -2],
                [1, -1, 4, 3, 4])
```

Maxima

Esto efectúa el intercambio de las filas 2 y 1 de la matriz M y devuelve la matriz resultante.

```
(%i21)  rowswap(M, 2, 1);
(%o21)  matrix([2, -2, 3, -3, -20],
                [2, -1, 2, -1, -8],
                [1, 1, 1, 0, -2],
                [1, -1, 4, 3, 4])
```

Maxima

Esto realiza la transformación $F_2 \leftarrow F_2 + F_1$ en la matriz M y devuelve la matriz resultante.

```
(%i22)  rowop(M, 2, 1, -1);
(%o22)  matrix([2, -1, 2, -1, -8],
                [4, -3, 5, -4, -28],
                [1, 1, 1, 0, -2],
                [1, -1, 4, 3, 4])
```

Maxima

He aquí el producto de Kronecker de dos matrices.

```
(%i23)  kronecker_product(matrix([a, b], [c, d]),
        matrix([1, 2, 3], [4, 5, 6], [7, 8, 9]) );
(%o23)  matrix([a, 2*a, 3*a, b, 2*b, 3*b],
                [4*a, 5*a, 6*a, 4*b, 5*b, 6*b],
                [7*a, 8*a, 9*a, 7*b, 8*b, 9*b],
                [c, 2*c, 3*c, d, 2*d, 3*d],
                [4*c, 5*a, 6*a, 4*d, 5*d, 6*d],
                [7*c, 8*c, 9*c, 7*d, 8*d, 9*d])
```

12.4 Funciones adicionales para matrices

`matrix size`(*M*)	devuelve una lista con dos elementos que constituyen el número de filas y columnas de la matriz *M* , respectivamente
`matrixp`(*expr*)	verifica si *expr* es una matriz
`mat norm`(*M*, *p*)	devuelve la p-norma de la matriz *M* . Los valores admisibles para p son 1, `inf` y `frobenius`
`addmatrices`(*f*, *A*, *B*, ...)	devuelve la matriz $(f\,(a_{i,j}, b_{i,j}, \ldots))_{m\times n}$, donde $m \times n$ es el orden común de las matrices *A*, *B*, ...
`mat unblocker`(*M*)	si *M* es una matriz de bloques, deshace los bloques de un nivel
`outermap`(*f*, $M_1, \ldots, M_n$)	aplica la función *f* a cada uno de los elementos del producto cartesiano $M_1 \times M_2 \ldots \times M_n$

Más funciones para matrices.

Maxima

He aquí la matriz *D*.

```
(%i24)  d[i,j] := 1 + (-1)^(i + j)$
(%i25)  D: genmatrix(d, 3, 4);

                [ 2  0  2  0 ]
(%o25)          [ 0  2  0  2 ]
                [ 2  0  2  0 ]
```

Maxima

Esto devuelve una lista con el número de filas y columnas de la matriz *D*.

```
(%i26)  matrix size(D);
(%o26)  [3, 4]
```

Maxima

Utilizando la función `matrixp` es posible comprobar que B es una expresión matricial.

```
(%i27) matrixp(D);
(%o27) true
```

Maxima

He aquí la norma de frobenius de la matriz B.

```
(%i28) mat_norm(D, frobenius);
(%o28) 2*sqrt(6)
```

Maxima

A continuación se definen las matrices A, B y C con elementos simbólicos indexados.

```
(%i29) A : genmatrix(a, 2, 3);
```

$$\begin{matrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{matrix}$$

```
(%i30) B: genmatrix(b, 2, 3);
```

$$\begin{matrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{matrix}$$

```
(%i31) C: genmatrix(c, 2, 3);
```

$$\begin{matrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \end{matrix}$$

Maxima

Este es el resultado obtenido al aplicar f mediante la función `addmatrices` a las matrices A, B y C.

```
(%i32) addmatrices(f, A, B, C);
```

$$\begin{matrix} f(a_{1,1},b_{1,1},c_{1,1}) & f(a_{1,2},b_{1,2},c_{1,2}) & f(a_{1,3},b_{1,3},c_{1,3}) \\ f(a_{2,1},b_{2,1},c_{2,1}) & f(a_{2,2},b_{2,2},c_{2,2}) & f(a_{2,3},b_{2,3},c_{2,3}) \end{matrix}$$

Maxima

Si f es "+", el resultado es $A + B + C$.

```
(%i33) addmatrices("+", A, B, C);
```

$$(\%o33)\quad \begin{matrix} c_{1,1}+b_{1,1}+a_{1,1} & c_{1,2}+b_{1,2}+a_{1,2} & c_{1,3}+b_{1,3}+a_{1,3} \\ c_{2,1}+b_{2,1}+a_{2,1} & c_{2,2}+b_{2,2}+a_{2,2} & c_{2,3}+b_{2,3}+a_{2,3} \end{matrix}$$

Maxima

También puede utilizarse una función anónima para f. En este caso el resultado obtenido es el mismo que el que se obtuvo al aplicar "+". No obstante, en casos como éste es mejor el uso de "+" ya que, por ejemplo, si hubiese sólo dos matrices habría que cambiar la función anónima por `lambda`([x, y], x + y).

```
(%i34) addmatrices(lambda([x,y,z],x+y+z), A, B, C);
```

$$(\%o34)\quad \begin{matrix} c_{1,1}+b_{1,1}+a_{1,1} & c_{1,2}+b_{1,2}+a_{1,2} & c_{1,3}+b_{1,3}+a_{1,3} \\ c_{2,1}+b_{2,1}+a_{2,1} & c_{2,2}+b_{2,2}+a_{2,2} & c_{2,3}+b_{2,3}+a_{2,3} \end{matrix}$$

Maxima

Con `outermap` y `mat_unblocker` también es posible obtener el producto de Kronecker de dos matrices.

```
(%i35) outermap("*", matrix([a, b], [c,d]),
matrix([1, 2, 3], [4, 5, 6], [7, 8, 9]) );
```

$$(\%o35)\quad \begin{matrix} a & 2a & 3a & b & 2b & 3b \\ 4a & 5a & 6a & 4b & 5b & 6b \\ 7a & 8a & 9a & 7b & 8b & 9b \\ c & 2c & 3c & d & 2d & 3d \\ 4c & 5c & 6c & 4d & 5d & 6d \\ 7c & 8c & 9c & 7d & 8d & 9d \end{matrix}$$

```
(%i36) mat_unblocker( %);
```

$$(\%o36)\quad \begin{matrix} a & 2a & 3a & b & 2b & 3b \\ 4a & 5a & 6a & 4b & 5b & 6b \\ 7a & 8a & 9a & 7b & 8b & 9b \\ c & 2c & 3c & d & 2d & 3d \\ 4c & 5a & 6a & 4d & 5d & 6d \\ 7c & 8c & 9c & 7d & 8d & 9d \end{matrix}$$

12.5 Matrices asociadas a sistemas de ecuaciones

`coefmatrix`([$eq_1, \ldots, eq_m$], [$x_1, \ldots, x_n$])	devuelve la matriz de coeficientes para las variables $x_1, \ldots, x_n$ del sistema de ecuaciones lineales $eq_1, \ldots, eq_m$
`augcoefmatrix`([$eq_1, \ldots, eq_m$], [$x_1, \ldots, x_n$])	devuelve la matriz aumentada de coeficientes para las variables $x_1, \ldots, x_n$ del sistema de ecuaciones lineales $eq_1, \ldots, eq_m$

Funciones para generar matrices asociadas a sistemas de ecuaciones.

Maxima

He aquí la matriz de coeficientes del sistema de ecuaciones lineales, dado.

```
(%i37) coefmatrix([2*x-3*y=1,x+y=3],[x,y]);
                [ 2  -3 ]
(%o37)          [       ]
                [ 1   1 ]
```

Maxima

Esta es la matriz aumentada del mismo sistema anterior.

```
(%i38) augcoefmatrix([2*x-3*y=1,x+y=3],[x,y]);
                [ 2  -3   1 ]
                [           ]
                [ 1   1   3 ]
```

Maxima

Aquí se hace la transformación $F_2 \leftarrow F_2 - \frac{1}{2}F_1$ en la matriz aumentada anterior. A partir de éste último resultado se deduce que la solución del sistema de ecuaciones lineales, dado, será $y = 1$ y $x = 2$.

```
(%i39) rowop(%,2,1,1/2);
                [ 2   -3    -1 ]
                [              ]
                [     5     5  ]
(%o39)          [ 0   -   -    ]
                [     2     2  ]
```

12.6 Autovalores y autovectores

`charpol` (M, x)	calcula el polinomio característico vinculado a la matriz M, en términos de la variable x
`eigenvalues`(M)	devuelve una lista con dos sublistas, la primera de éstas conteniendo los valores propios de la matriz M y la segunda, conteniendo sus correspondientes multiplicidades
`eigenvectors`(M)	devuelve una lista con dos elementos; el primero está formado por `eigenvalues`(M), el segundo es una lista de listas de vectores propios, una por cada valor propio, pudiendo haber uno o más vectores propios en cada lista

Obtención de autovalores y autovectores asociados a una matriz dada.

Maxima

He aquí una matriz 3×3.

```
(%i40)  c[i,j] := i + j + 1;
(%o40)  c_{i,j} := i + j + 1
(%i41)  C: genmatrix(c, 3, 3);

                [ 3  4  5 ]
(%o41)          [ 4  5  6 ]
                [ 5  6  7 ]
```

Maxima

Éste es el polinomio característico, en términos de la variable x, asociado a la matriz C.

```
(%i42)  charpoly(C, x), expand;
                     3      2
(%o42)              - x  + 15 x  + 6 x
```

Maxima

Aquí se tienen los valores propios, y su respectiva multiplicidad, asociados a la matriz C.

```
(%i43)  eigenvalues(C);
(%o43)  [-(sqrt(249-15)/2, sqrt(249+15)/2, 0), [1, 1, 1]]
```

Maxima

Finalmente, se muestran los valores propios, con su respectiva multiplicidad, y vectores propios de la matriz C.

```
(%i44)  eigenvectors(C);
(%o44)  [-(sqrt(249-15)/2, sqrt(249+15)/2, 0), [1, 1, 1],
          1, -(sqrt(249-19)/28), -(3 sqrt(83-5)/14),
          1, sqrt(249+19)/28, (3-sqrt(83+5))/14], [[1, -2, 1]]]
```

Capítulo 13

Conjuntos

Los conjuntos en Maxima son expresiones que comprenden el operador `set` y cuyos argumentos son los elementos del mismo (ver sección 17.2). Esto indica que Maxima hace una diferencia entre conjuntos y listas, lo que permite trabajar con conjuntos cuyos elementos puedan ser también conjuntos o listas.

Maxima dispone de funciones para realizar operaciones con conjuntos, como la intersección o la unión. No obstante, debe tenerse en cuenta que los conjuntos deben ser finitos y definidos por enumeración.

13.1 Generación de conjuntos

`set`$(a_1, \ldots, a_n)$	construye un conjunto cuyos elementos son $a_1, \ldots, a_n$
$\{a_1, \ldots, a_n\}$	equivale a `set`$(a_1, \ldots, a_n)$
`set`()	genera un conjunto vacío
{}	equivale a `set`()
`makeset`(*f*, *vars*, *s*)	genera un conjunto cuyos miembros se generan a partir de f, siendo *vars* una lista de variables de f y s un conjunto o lista de listas

Funciones para generar conjuntos.

Maxima

He aquí un conjunto formado por los elementos *a, b, c, d*.

```
(%i1) {a, b, c, d};
(%o1) {a, b, c, d}
```

Maxima

Al ingresar un conjunto cuyos elementos están desordenados y repetidos, éstos son ordenados y unificados de forma automática.

```
(%i2) {c, e, f, f, a, b, c, d, a};
(%o2) {a, b, c, d, e, f}
```

Maxima

La función `makeset` permite generar un conjunto. En este caso se considera una función de dos variables, la cual es aplicada a una lista de listas.

```
(%i3) makeset(i/j, [i, j], [[1, a], [2, b], [3, c], [4, d]]);
                  1  2  3  4
(%o3)             -, -, -, -
                  a  b  c  d
```

Maxima

He aquí otro conjunto generado con `makeset`. Ahora se considera una función de una variable, la cual es aplicada a un conjunto de listas.

```
(%i4) makeset(x/2, [x], {[1], [2], [3], [4], [5]});
                  1  1  3
(%o4)             -, -, -, 2, 2
                  2  2  2
```

Maxima

Las operaciones aritméticas son "listables".

```
(%i5) x^[1, 2, 3];
                          2  3
(%o5)                [x, x , x ]
```

Maxima

Sin embargo, las operaciones aritméticas no son "conjuntables".

```
(%i6)  x^{1,2,3};
(%o6)  x^{1,2,3}
```

Maxima

La sustitución si es "conjuntable", aunque siempre respetando la unificación.

```
(%i7)  {x,x^2,x^3,x+1},x=1;
(%o7)  {1,2}
```

13.2 Conversiones entre conjuntos y listas

`setif` (*list*)	forma un conjunto a partir de los miembros de la lista *list*
`fullsetif` (*list*)	sustituye los operadores de lista presentes en *conj* por operadores de conjuntos

Conversiones de listas a conjuntos.

`listif` (*conj*)	forma un conjunto a partir de los miembros del conjunto *conj*
`full listif` (*conj*)	sustituye los operadores de conjunto presentes en *conj* por operadores de listas

Conversiones de conjuntos a listas.

Maxima

Esto convierte una lista en conjunto.

```
(%i8)  setify([a,c,d,e]);
(%o8)  {a,c,d,e}
```

Maxima

Esto convierte un conjunto en lista.

```
(%i9)  listify( {a, b, c, d});
(%o9)  [a, b, c, d]
```

Maxima

Esto sustituye todos los operadores de conjuntos a operadores de listas.

```
(%i10)  fullsetify([a, [b, c] , [e, [f, g]] , k]);
(%o10) {a,{b, c},{e,{f, g}} , k}
```

Maxima

Esto sustituye todos los operadores de listas a operadores de conjuntos.

```
(%i11)  full   listify( %);
(%o11)  [a,[b,c],[e,[f,g]],k]
```

Considerando que, cualquier lista puede convertirse a conjunto, entonces puede usarse `create list` para generar una lista y luego convertir la lista obtenida a conjunto. De esta manera se tiene una forma indirecta para generar conjuntos.

13.3 Elección de elementos de un conjunto

`part`(*conj*, *i*)	devuelve el i-ésimo elemento del conjunto *conj* respetando el ordenamiento y la unificación internos
`first`(*conj*), `second`(*conj*), . . . , `tenth`(*conj*)	devuelve el primer, segundo, ..., décimo elemento de *conj*
`last`(*conj*)	devuelve el último elemento de *conj*

Selección de partes de un conjunto.

Maxima

Aquí se obtiene el primer elemento de un conjunto que ha sido ordenado y unificado internamente.

```
(%i12)  part({b, c, c, d, a}, 1);
(%o12)  a
```

Maxima

El quinto elemento no existe.

```
(%i13)  part({b, c, c, d, a} , 5);
part: fell off the end.
- - an error. To debug this try: debugmode(true);
```

13.4 Prueba y búsqueda de elementos de un conjunto

Existe una función específica para averiguar si un elemento pertenece a un conjunto, no obstante hay funciones genéricas alternativas para ello.

`subset`(*conj*, *P*)	extrae todos los elementos de un conjunto *conj* que satisfacen el predicado *P*
`subsetp`($conj_1$, $conj_2$)	devuelve **true** si y sólo si $con_1 \subseteq conj_2$
`elementp`(*x*, *conj*)	devuelve **true** si y sólo si *x* es elemento del conjunto *conj*
`member`(*x*, *conj*)	devuelve **true** si y sólo si *x* es miembro del conjunto *conj*
`freeof`(*x*, *conj*)	prueba si *x* no ocurre en ninguna parte *conj*
`freeof`($x_1 \ldots, x_n$, *conj*)	equivale a *freeof* (x_1; *conj*) and ... and *freeof* (x_n; *conj*)

Prueba de elementos de un conjunto.

Maxima

Aquí se extrae, del conjunto {1, 2, 3, 4, 5, 7}, el subconjunto de todos los elementos t tales que $t < 4$.

```
(%i14)  subset({1, 2, 3, 4, 5, 7},
        lambda([t], is(t < 4)));
(%o14)  {1, 2, 3}
```

Maxima

Aquí se extrae, del conjunto {1, 2, 3, 4, 5, 7}, el subconjunto de todos los elementos t tales que $mod(t, 3) = 1$.

```
(%i15)  subset({1, 2, 3, 4, 5, 7},
        lambda([t], is(mod(t, 3) = 1)));
(%o15)  {1, 4, 7}
```

Maxima

Esto verifica una contención de conjuntos.

```
(%i16)  subsetp({1, 2, 5}, {1, 2, 3, 5, 6, 7, 9});
(%o16)  true
```

Maxima

Esto verifica que el elemento 2 pertenece al conjunto {2, 4, 5, 7}.

```
(%i17)  elementp(2, {2, 4, 5, 7});
(%o17)  true
```

Maxima

La función genérica *member* realiza los mismo.

```
(%i18)  member(2, {2, 4, 5, 7});
(%o18)  true
```

Maxima

La función genérica *freeof* indica que 2 forma parte del conjunto {2, 4, 5, 7}.

```
(%i19)  freeof(2, {2, 4, 5, 7});
(%o19)  false
```

Maxima

La función genérica *freeof* indica que 6 y 5 no forman parte del conjunto {1, 2, 3}.

```
(%i20)  freeof(6, 5, {1, 2, 3});
(%o20)  true
```

13.5 Agregar y quitar elementos de un conjunto

Es posible agregar y/o quitar elementos de un conjunto dado mediante dos funciones específicas de Maxima.

`adjoin`(*x*, *conj*)	devuelve el conjunto *conj* con el elemento *x* insertado
`disjoin`(*x*, *conj*)	devuelve el conjunto *conj* sin el elemento *x*

Funciones para agregar y quitar elementos de un conjunto.

Maxima

Aquí se unen {2} y {1, 4, 7}.

```
(%i21)  adjoin(2,{1,4,7});
(%o21)  {1,2,4,7}
```

Maxima

Esto elimina el elemento 4 de {1, 4, 7}.

```
(%i22)  disjoin(4,{1,4,7});
(%o22)  {1,7}
```

13.6 Reorganización de conjuntos

`flatten`(*conj*)	elimina todos los niveles en *conj*
`set partitions`(*conj*)	devuelve el conjunto de todas las particiones de *conj*
`set partitions`(*conj*, *n*)	devuelve un conjunto con todas las descomposiciones de *conj* en *n* conjuntos no vacíos disjuntos

Reorganización de conjuntos anidados.

Maxima

He aquí un conjunto al que se le eliminan todos los niveles.

```
(%i23) flatten({4,{1,3},{4,{7,8}},10});
(%o23) {1,3,4,7,8,10}
```

Maxima

Esto da todas las particiones del conjunto {*a, b, c*}.

```
(%i24) set partitions(a, b, c);
(%o24) {{{a},{b},{c}},{{a},{b,c}},{{a,b},{c}},
        {{a, b, c}} , {{a, c} , {b}}}
```

Maxima

He aquí todas las descomposiciones del conjunto {*a, b, c*} en dos conjuntos no vacíos disjuntos.

```
(%i25) set partitions(a, b, c,2);
(%o25) {{{a},{b,c}},{{a,b},{c}},{{a,c},{b}}}
```

13.7 Operaciones con conjuntos

Las más comunes operaciones matemáticas con conjuntos están implementadas en Maxima.

`union`($conj_1, \ldots, conj_n$)	devuelve la unión de los conjuntos $conj_1, \ldots, conj_n$
`intersection`($conj_1, \ldots, conj_n$)	devuelve la intersección de los conjuntos $conj_1, \ldots, conj_n$
`setdifference`($conj_1$, $conj_2$)	devuelve la diferencia de los conjuntos $conj_1$, $conj_2$
`powerset`($conj$)	devuelve el conjunto potencia de $conj$
`subsetp`($conj_1$, $conj_2$)	devuelve `true` si y sólo si el conjunto $conj_1$ es un subconjunto de $conj_2$
`setequalp`($conj_1$, $conj_2$)	devuelve `true` si y sólo los conjuntos $conj_1$ y $conj_2$ son iguales
`s mmdifference`($conj_1$, $conj_2$)	devuelve la diferencia simétrica de los conjuntos $conj_1$, $conj_2$
`cartesian product`($conj_1, \ldots, conj_n$)	devuelve el producto cartesiano de los conjuntos $conj_1, \ldots, conj_n$ en forma de listas
`disjointp`($conj_1$, $conj_2$)	devuelve `true` si y sólo si los conjuntos $conj_1$ y $conj_2$ son disjuntos
`equiv classes`($conj, F$)	devuelve el conjunto de las clases de equivalencia del conjunto $conj$ respecto de la relación de equivalencia F

Operaciones con conjuntos.

Maxima

Aquí se definen los conjuntos A y B.

```
(%i26)  A : {a, b, c, d}$
(%i27)  B : {c, d, e, f, g}$
```

Maxima

Esto da la unión de A y B.

```
(%i28)  union(A, B);
(%o28)  {a, b, c, d, e, f, g}
```

Maxima

Y esto la intersección.

```
(%i29)  intersection(A,B);
(%o29)  {c, d}
```

Maxima

La diferencia $A - B$ se calcula así.

```
(%i30)  setdifference(A, B);
(%o30)  {a, b}
```

Maxima

Esto devuelve la diferencia simétrica $B \triangle A$.

```
(%i31)  symmdifference(B, A);
(%o31)  {a, b, e, f, g}
```

Maxima

He aquí el conjunto potencia del conjunto A.

```
(%i32)  powerset(A);
(%o32)  {{},{a},{a, b},{a, b, c},{a, b, c, d},{a, b, d},{a, c},
        {a, c, d},{a, d},{b},{b, c},{b, c, d},{b, d},{c},{c, d},
        {d}}
```

Maxima

Esto da el producto cartesiano $A \times B$.

```
(%i33)  cartesian product(A, B);
(%o33)  {[a, c], [a, d], [a, e], [a, f], [a, g], [b, c], [b, d], [b, e], [b, f],
        [b, g], [c, c], [c, d], [c, e], [c, f], [c, g], [d, c], [d, d], [d, e],
        [d, f] , [d, g]}
```

Maxima

Esto indica que los conjuntos A y B no son disjuntos.

```
(%i34) disjointp(A,B);
(%o34) false
```

Maxima

En este ejemplo, las clases de equivalencia son números que difieren en un múltiplo de 3.

```
(%i35) equiv_classes({1,2,3,4,5,6,7},lambda([s,t],
       mod(s - t,3) = 0 ));
(%o35) {{1,4,7},{2,5},{3,6}}
```

13.8 Funciones adicionales para conjuntos

`setp`(*expr*)	devuelve `true` si y sólo si *expr* es un conjunto de Maxima
`empty_p`(*conj*)	devuelve `true` si y sólo si *conj* es el conjunto vacío
`cardinality`(*conj*)	devuelve el número de elementos del conjunto *conj*
`map`($f, \{a_1, \ldots, a_n\}$)	devuelve $\{f(a_1), \ldots, f(a_n)\}$
`map`($f, \{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_n\}, \ldots$)	devuelve el conjunto $\{f(a_1, b_1, \ldots), \ldots, f(a_n, b_n, \ldots)\}$, siempre que los conjuntos estén bien ordenados

Algunas funciones adicionales.

Maxima

He aquí el cardinal de un conjunto.

```
(%i36) cardinality( {a, b, c, a});
(%o36) 4
```

Maxima

Para conjuntos bien ordenados, al aplicar la función `map`, se obtiene un resultado acorde con el ordenamiento de tales conjuntos.

```
(%i37)  map(f,{a,b,c},{x,y,z});
(%o37)  {f(a,x),f(b,y),f(c,z)}
```

Maxima

En este caso primero son ordenados los conjuntos y luego devuelve un resultado que coincide con el de (%o37) .

```
(%i38)  map(f,{c,a,b},{y,x,z});
(%o38)  {f(a,x),f(b,y),f(c,z)}
```

Capítulo 14

Gráficos

Maxima no ha sido programado para elaborar sus propios gráficos, de modo que alternativamente utiliza un programa externo que realiza esta tarea. El usuario se limita a ordenar el tipo de gráfico deseado y Maxima se encarga de comunicárselo al programa gráfico que esté activo en ese momento, que por defecto será Gnuplot. La otra aplicación gráfica que utiliza Maxima es Openmath[1].

Las funciones `plot2d` y `plot d`[2] son funciones para obtener gráficos que están definidas en el propio núcleo de Maxima (téngase presente que existen otras funciones similares definidas en un paquete llamado `draw`[3]).

Lo antes mencionado ocasiona que los gráficos generados con las funciones `plot2d` y `plot d` se muestren en una ventana aparte del cuaderno de trabajo actual. No obstante, en contraparte, wxMaxima incorpora funciones alternativas, cuyos nombres se obtienen anteponiendo `wx` a `plot2d` y `plot d`, respectivamente. La diferencia con las anteriores es que éstas funciones devuelven todos los gráficos en el cuaderno de trabajo actual.

[1] Tanto Gn'Uplot como Openmath son softwares libres y una copia de cada uno de ellos es distribuida conjuntamente con Maxima.

[2] Con `plot2d` y `plot3d` es posible dar instrucciones tanto a Gn'Uplot como a Openmath sin que sea necesario tener conocimiento alguno de la sintaxis de estos programas.

[3] El paquete `draw` (ver el capítulo l5) está orientado exclusivamente a Gn'Uplot. Éste incorpora las funciones `draw2d` y `draw3d`, las cuales dan resultados similares a `plot2d` y `plot3d`. Además incorpora funciones para graficar funciones definidas implícitamente.

14.1 Gráficos básicos

Por comodidad, la mayoría de los ejemplos de esta sección se ejecutarán con las funciones incorporadas en wxMaxima; sin embargo, todos estos ejemplos pueden ser ejecutados sin ningún problema con las funciones estándar de Maxima.

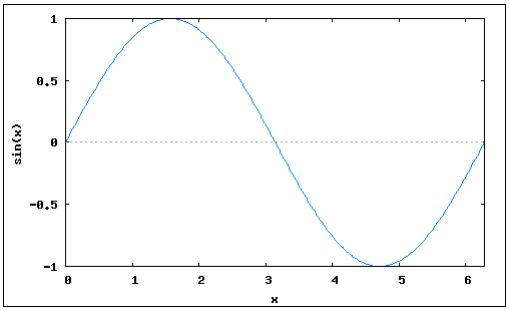
`plot2d`(f, [x, x_{min}, x_{max}])	muestra un gráfico de $y = f(x)$, con $x_{min} \leq x \leq x_{max}$, en una ventana independiente
`plot2d`([$f_1, \ldots, f_n$], [x, x_{min}, x_{max}])	muestra un gráfico de $y = f_1(x)$, $\ldots$, $y = f_n(x)$, con $x_{min} \leq x \leq x_{max}$, en una ventana independiente
`wxplot2d`(f, [x, x_{min}, x_{max}])	muestra un gráfico de $y = f(x)$, con $x_{min} \leq x \leq x_{max}$, en el cuaderno de trabajo actual
`wxplot2d`([$f_1, \ldots, f_n$], [x, x_{min}, x_{max}])	muestra un gráfico de $y = f_1(x)$, $\ldots$, $y = f_n(x)$, con $x_{min} \leq x \leq x_{max}$, en el cuaderno de trabajo actual

Trazado básico de funciones.

Maxima

Esto traza un gráfico de sen(x) como una función de x. Para x variando desde 0 hasta 2π.

```
(%i1) wxplot2d(sin(x),[x,0,2*%pi]);
(%t1)
```



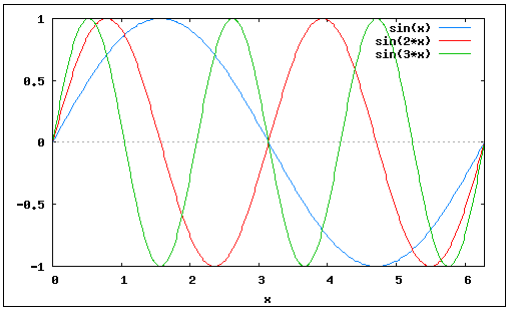

```
(%o1)
```

Maxima

Puede darse una lista de funciones para que sean trazadas. Por defecto, a cada función se le asigna un color específico.

```
(%i2) wxplot2d([sin(x), sin(2 * x), sin(3 * x)],
      [x, 0, 2 * %pi]);
```

(%t2)



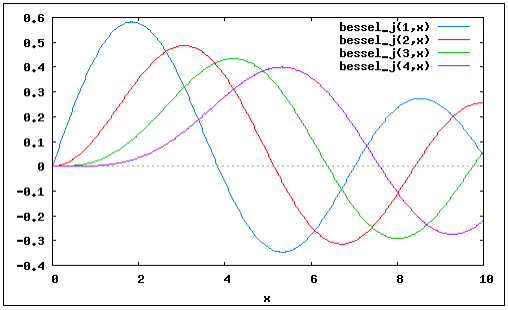
(%o2)

Maxima

Aquí se utiliza la función `create_list` para generar una lista de las funciones de Bessel, $J_n(x)$, con n variando desde 1 hasta 4. Luego se devuelve la gráfica de dichas funciones.

```
(%i3) wxplot2d(create list(bessel j(n, x), n, 1, 4),
      [x, 0, 10]);
```

(%t3)



(%o3)

14.2 Opciones

Hay muchas opciones para escoger en el trazado de gráficos. La mayoría de las veces, la aplicación gráfica invocada por Maxima probablemente tomará opciones bastante buenas. Sin embargo, si se quiere obtener los mejores dibujos posibles para objetivos particulares, debería ayudarse a la aplicación en particular en la elección de algunas de sus opciones.

Hay un mecanismo general para especificar opciones en las funciones de Maxima. Cada opción tiene un nombre definido. Como últimos argumentos de una función, como `plot2d` (`wxplot2d`), se puede incluir una secuencia de la forma [*nombre, valor*] para especificar los valores de varias opciones. A cualquier opción para la cual no se indique un valor explícito se le asigna su valor por defecto.

`plot2d`(*f*, [*x*, x_{min}, x_{max}] , [*opcion, valor*])	devuelve un gráfico, especificando un valor particular para una opción, en otra ventana

Elección de una opción para el trazado de un gráfico.

Opción	Val. por defecto	Descripción
	no presenta	rango vertical del gráfico
`nticks`	29	número inicial de puntos utilizados por el procedimiento adaptativo para la representación de funciones
`adapt depth`	10	número máximo de particiones utilizado por el algoritmo adaptativo de representación gráfica
`xlabel`	x	etiqueta del eje horizontal en gráficos 2d
`label`	nombre de la función f, dada	etiqueta del eje vertical en gráficos 2d

Algunas de las opciones de `plot2d` (`wxplot2d`).

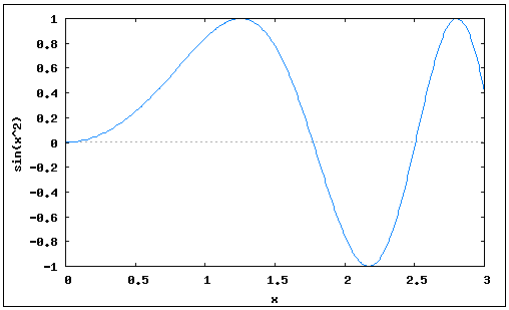
Opción	Val. por defecto	Descripción
`logx`	no presenta	hace que el eje horizontal en los gráficos 2d se dibuje en la escala logarítmica (no necesita de parámetros adicionales)
`log`	no presenta	hace que el eje vertical en los gráficos 2d se dibuje en la escala logarítmica (no necesita de parámetros adicionales)
`legend`	nombre de la función f, dada	etiquetas para las expresiones de los gráficos 2d. Si hay más expresiones que etiquetas, éstas se repetirán
`st le`	`lines, 1, 1`	estilos a utilizar para las funciones o conjuntos de datos en gráficos 2d
`gnuplot preamble`	`" "`	introduce instrucciones de gnuplot antes de que se haga el dibujo
`plot format`	`gnuplot`	determina qué programa gráfico se va a utilizar
`gnuplot term default`	selecciona el terminal a utilizar por Gnuplot; algunos valores posibles para el terminal son: dumb, png, jpg, eps, pdf, gif, etc. (la opción `gnuplot term` únicamente puede utilizarse con `plot2d`)	
`plot realpart`	`false`	si `plot realpart` vale `true`, se representará la parte real de un valor complejo
`run viewer`	`true`	controla si el visor apropiado para la salida gráfica debe ejecutarse o no

Algunas de las opciones de `plot2d` (`wxplot2d`).

Maxima

He aquí un gráfico para el cual todas las opciones tienen asignados sus valores por defecto.

```
(%i4) wxplot2d(sin(x^2),[x,0,3]);
(%t5)
```

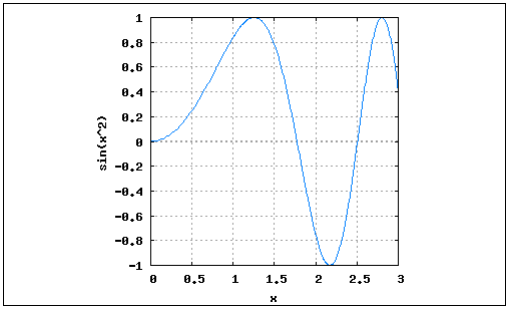



```
(%o4)
```

Maxima

La instrucción `set size ratio 1` iguala las escalas para los ejes `x` e `y`; y la instrucción `set grid` añade una rejilla al gráfico. Ambas instrucciones corresponden a la opción `gnuplot_preamble`.

```
(%i5) wxplot2d(sin(x^2),[x,0,3],[gnuplot_preamble,
      "set size ratio 1; set grid"]);
(%t5)
```



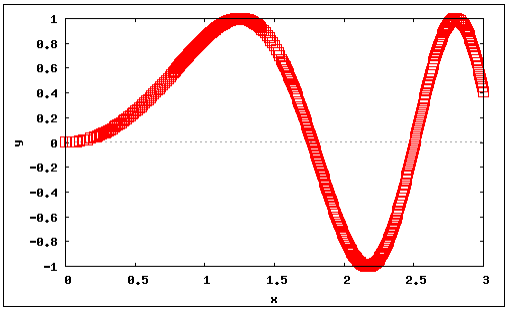

```
(%o5)
```

Maxima

Esto especifica el estilo del gráfico y la etiqueta para el eje y.

```
(%i6) wxplot2d(sin(x^2),[x,0,3],[style,[points,3,2,7]],
      [ylabel, y]);
```

(%t6)



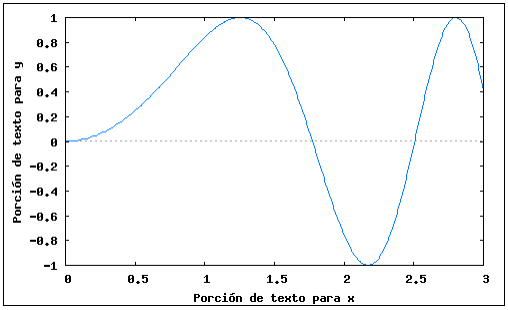
(%o6)

Maxima

Las expresiones que se dan como etiquetas puede ser cualquier porción de texto, pero ésta debe ponerse entre comillas.

```
(%i7) wxplot2d(sin(x^2),[x,0,3],
      [xlabel,"Porción de texto para x"],
      [ylabel,"Porción de texto para "]);
```

(%t7)

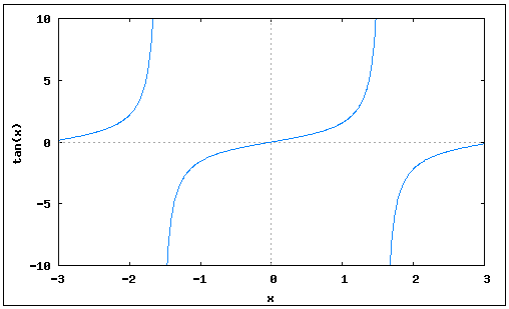


(%o7)

Maxima

Es posible trazar funciones que tienen singularidades. Para este efecto resulta bastante útil la opción `y` que permite indicar el rango.

```
(%i8) wxplot2d(tan(x),[x,-3,3],[y,-10,10]);
plot2d: some values were clipped.
(%t8)
```




```
(%o8)
```

14.3 Gráficos de puntos y líneas

Con `plot2d` (`wxplot2d`) pueden graficarse puntos del plano (aislados) o poligonales que unen puntos del plano.

`plot2d`([discrete, $[[x_1,y_1],\ldots,[x_n,y_n]]$], [style, points])	devuelve el gráfico de los puntos aislados $(x_1,y_1),\ldots(x_n,y_n)$
`plot2d`([discrete, $[x_1,\ldots,x_n],[y_1,\ldots,y_n]$], [style, points])	devuelve el gráfico de los puntos aislados $(x_1,y_1),\ldots(x_n,y_n)$
`plot2d`([discrete, $[[x_1,y_1],\ldots,[x_n,y_n]]$], [style, lines])	devuelve el gráfico de una poligonal que une los puntos $(x_1,y_1),\ldots(x_n,y_n)$

Gráficos de puntos aislados y poligonales.

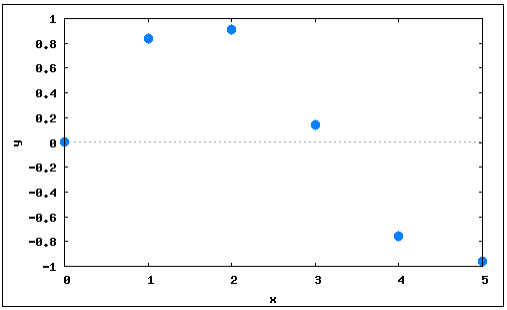
`plot2d`([discrete, $[x_1,\ldots,x_n]$, $[y_1,\ldots,y_n]$], [style, lines])	devuelve el gráfico de una poligonal que une los puntos $(x_1,y_1),\ldots(x_n,y_n)$
`plot2d`([discrete, $[[x_1,y_1],\ldots,[x_n,y_n]]$])	devuelve el gráfico de una poligonal que une los puntos $(x_1,y_1),\ldots(x_n,y_n)$
`plot2d`([discrete, $[x_1,\ldots,x_n]$, $[y_1,\ldots,y_n]$])	devuelve el gráfico de una poligonal que une los puntos $(x_1,y_1),\ldots(x_n,y_n)$

Gráficos de puntos aislados y poligonales.

Maxima

De esta forma se grafica una lista de puntos aislados.

```
(%i9)  pts : create_list([i, sin(i)], i, 0, 5);
(%o9)  [[0, 0], [1, sin(1)], [2, sin(2)], [3, sin(3)], [4, sin(4)],
        [5, sin(5)]]
(%i10) wxplot2d([discrete, pts], [style, points]);
(%t10)
```




```
(%o10)
```

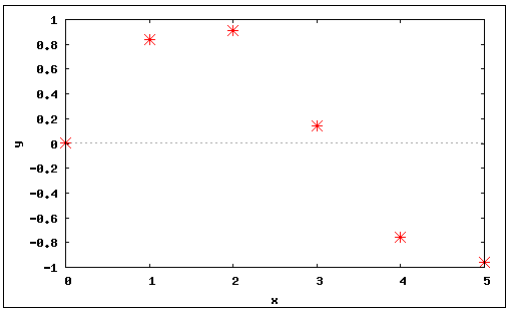
Cabe destacar que los estilos `points` y `lines` también aceptan opciones. Éstas deben ingresarse como una secuencia de tres números enteros positivos de la forma [points n_1, n_2, n_3] (en el caso de `points`), o dos números enteros positivos de la forma [lines n_1, n_2] (en el caso de `lines`). En ambos casos el primer valor, n_1, corresponde al tamaño de los puntos o grosor de la línea, según sea el caso; el segundo, n_2, al color (1 ≡ azul, 2 ≡ rojo, 3 ≡ verde, 4 ≡ violeta, 5 ≡ negro, 6 ≡ celeste). Finalmente para `points` está el tercer valor, n_3,

que corresponde a la forma en que se visualiza el punto (1 ≡ disco, 2 ≡ círculo, 3 ≡ cruz, 4 ≡ aspa, 5 ≡ asterisco, 6 ≡ cuadrado relleno, 7 ≡ cuadrado sin rellenar, 8 ≡ triángulo relleno, 9 ≡ triángulo sin rellenar, 10 ≡ triángulo relleno invertido, 11 ≡ triángulo sin rellenar invertido, 12 ≡ rombo relleno, 13 ≡ rombo sin rellenar).

Maxima

Aquí se muestran los puntos de la salida %o9 con opciones que cambian el tamaño, color y forma de los mismos.

```
(%i11) wxplot2d([discrete,pts],[style,[points,3,2,5]]);
(%t11)
```

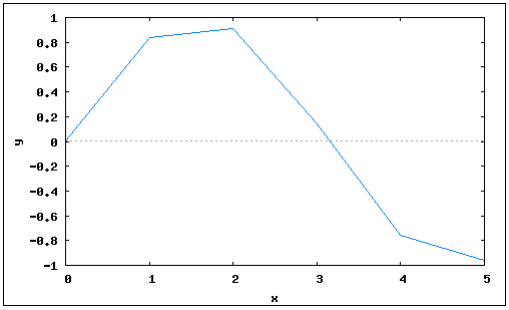



```
(%o11)
```

Maxima

Aquí se muestran los puntos de la salida %o9 unidos por una línea quebrada.

```
(%i12) wxplot2d([discrete, pts] , [style, lines]);
(%t12)
```




```
(%o12)
```

14.4 Gráficos paramétricos y polares

plot2d([parametric, f_x, f_y, [t, t_{min}, t_{max}]])	traza un gráfico paramétrico
plot2d([[parametric, f_x, f_y, [t, t_{min}, t_{max}]] , [parametric, g_x, g_y, [s, s_{min}, s_{max}]] , . . .])	traza varias curvas paramétricas juntas

Gráficos de curvas definidas en forma paramétrica.

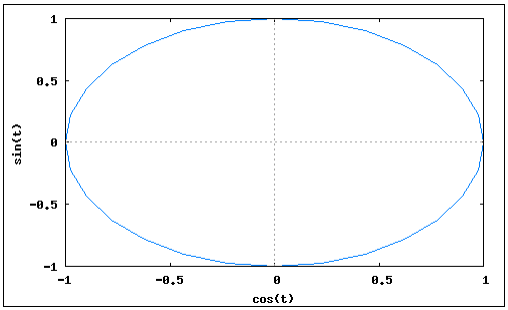
plot2d($f(t)$, [t, t_{min}, t_{max}] , [gnuplot_preamble, "set polar"])	traza un gráfico polar
plot2d([$f(t)$, $g(t)$, . . .] [t, t_{min}, t_{max}] , [gnuplot_preamble, "set polar"])	traza varias curvas polares juntas

Gráficos de curvas definidas en forma polar.

Maxima

Esto devuelve la gráfica de la curva $t \to (\cos(t), \text{sen}(t))$ para $t \in [0, 2\pi]$.

```
(%i13) wxplot2d([parametric,cos(t),sin(t),[t,0,2* %pi]]);
(%t13)
```



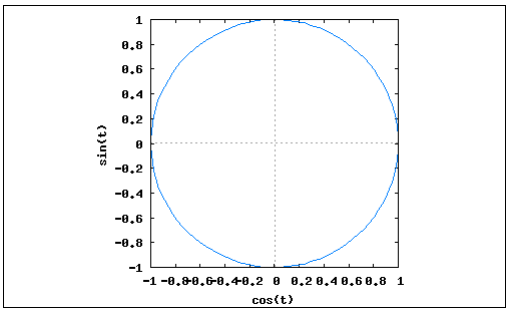
(%o13)

Maxima

Utilizando las opciones adecuadas es posible mejorar la presentación de la salida %o12.

```
(%i14) wxplot2d(
       [parametric, cos(t), sin(t), [t,0,2 * %pi]],
       [gnuplot preamble, "set size ratio 1"],
       [nticks, 100]
       );
```

(%t14)



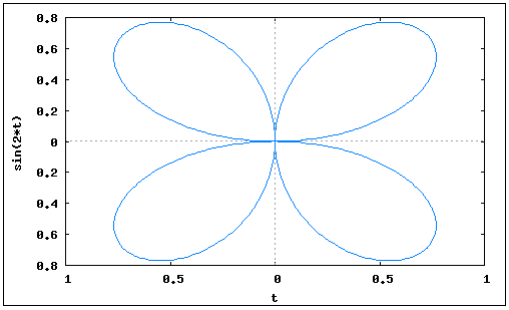
(%o14)

Maxima

Esto devuelve la gráfica de $\rho = \text{sen}(2t)$.

```
(%i15) wxplot2d(sin(2 * t), [t, 0, 2 * %pi],
       [gnuplot preamble, "setpolar"], [x, -1, 1]);
```

(%t15)



(%o15)

14.5 Combinación de gráficos

Adicionalmente, la función `plot2d` (`wxplot2d`) permite combinar varios tipos de gráficos (salvo los polares) y presentarlos en un mismo sistema coordenado.

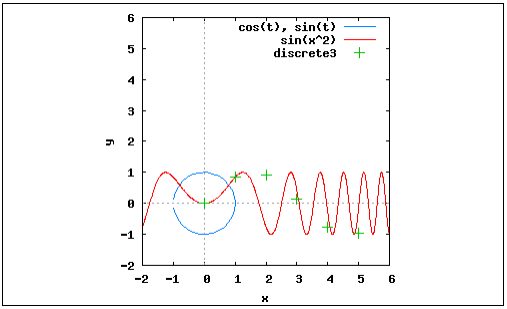
Maxima

Aquí se combinan varios tipos de gráficos.

```
(%i16) pts : create list([i, sin(i)] , i, 0, 5) $
(%i17) wxplot2d([
       [parametric, cos(t), sin(t)] , [t, 0, 2 * %pi] ,
       sin(x^2),
       [discrete, pts]] ,
       [x, 2, 6] , [y, 2, 6] ,
       [style, lines, lines, points] ,
       [gnuplot preamble, "set size ratio 1"]);
```

plot2d: expression evaluates to non-numeric value everywhere in plotting range.

(%t17)



(%o17)

14.6 Gráficos de superficies tridimensionales

Para graficar superficies en R^3 se utiliza la función `plot d` (`wxplot d`). Es preciso mencionar que, al utilizar la función `wxplot d` el gráfico resultante será mostrado en el cuaderno de trabajo actual; no obstante, se pierde la interacción en tiempo real con el gráfico. Esto no sucede si se utiliza la función `plot d`, ya que en este caso basta con

hacer clic sobre la figura y, sin soltar el botón del mouse, arrastrarlo para que la superficie gire en tiempo real.

`plot d`(f, [x, x_{min}, x_{max}], [y, y_{min}, y_{max}])	muestra un gráfico de $z = f(x,y)$, con $x_{min} \leq x \leq x_{max}$ y $y_{min} \leq y \leq y_{max}$, en una ventana independiente; en esta ventana es posible interactuar en tiempo real con dicho gráfico
`wxplot d`(f, [x, x_{min}, x_{max}], [y, y_{min}, y_{max}])	muestra un gráfico de $z = f(x,y)$, con $x_{min} \leq x \leq x_{max}$ y $y_{min} \leq y \leq y_{max}$, en el cuaderno de trabajo actual y no hay interacción en tiempo real con el gráfico

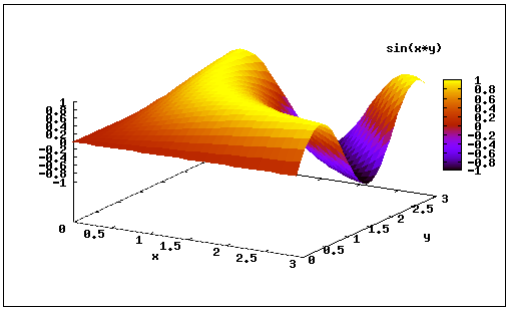
Trazado básico de funciones 3D.

Al igual que `plot2d` (`wxplot2d`), la función `plot d` (`wxplot d`) también incluye una serie de opciones para obtener los mejores dibujos posibles.

Maxima

Esto traza un gráfico de la función $f(x, y) = \text{sen}(xy)$.

```
(%i18) wxplot3d(sin(x*y),[x,0,3],[y,0,3]);
(%t18)
```




```
(%o18)
```

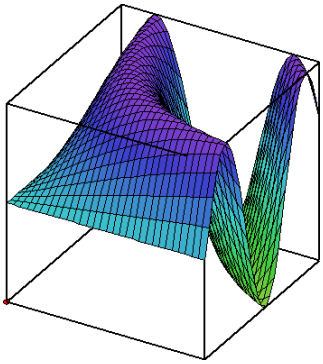
Opción	Val. por defecto	Descripción
`grid`	30, 30	establece el número de puntos de la retícula a utilizar en las direcciones x e y
`transform x`	`false`	permite que se realicen transformaciones en los gráficos de tres dimensiones
`gnuplot term`	`default`	selecciona el terminal a utilizar por Gnuplot; algunos valores posibles son: dumb, png, jpg, eps, pdf, gif, etc. (la opción `gnuplot term` sólo funciona con `plot d`)
`gnuplot preamble`	`" "`	introduce instrucciones de gnuplot antes que se haga el dibujo
`plot format`	`gnuplot`	determina qué programa gráfico se va a utilizar

Algunas de las opciones de `plot3d` (`wxplot3d`)

Maxima

Esto traza un gráfico de $f(x, y) = \text{sen}(xy)$ con el programa gráfico `openmath`.

```
( %i19)  plot3d(sin(x*y), [x, 0, 3], [y, 0, 3],
         [plot format, openmath]);
( %o19)
```

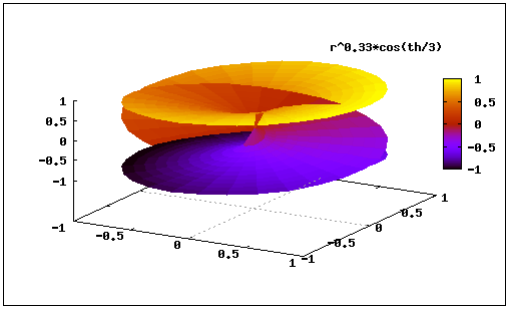


Maxima

Esto traza un gráfico tridimensional de la superficie definida por $r^{0.33}\cos(\frac{th}{3})$ en coordenadas cilíndricas[4]

```
(%i20)  wxplot3d(r^.33*cos(th/3),
                 [r,0,1],[th,0,6* %pi],
                 [grid, 12, 80],
                 [transform xy, polar to xy]);
```

(%t20)



(%o20)

`plot d` (`wxplot d`) también permite trazar la gráfica de una superficie definida en forma paramétrica; sin embargo, no permite graficar curvas en $\mathbb{R}^3$.

`plot d`$([x(u,v), y(u,v), z(u,v)], [u, u_{min}, u_{max}], [v, v_{min}, v_{max}])$	traza el gráfico de una superficie paramétrica en una ventana independiente, siendo posible la interacción en tiempo real con dicho gráfico
`wxplot d`$([x(u,v), y(u,v), z(u,v)], [u, u_{min}, u_{max}], [v, v_{min}, v_{max}])$	traza el gráfico de una superficie paramétrica en el cuaderno de trabajo actual, siendo imposible la interacción en tiempo real con dicho gráfico

Trazado básico de superficies definidas en forma paramétrica.

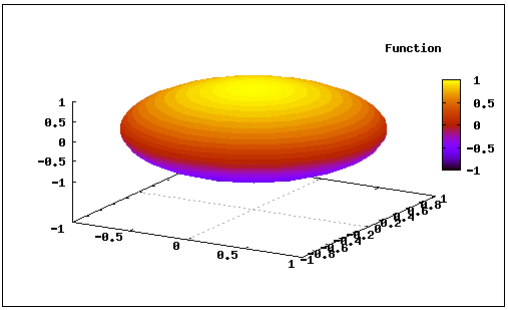
[4] En este caso se ha realizado una transformación de coordenadas. El usuario puede definir sus propias transformaciones usando la función `make_transform`$(vars, f_x, f_y, f_z)$. Por ejemplo, aquí se ha usado la transformación polar_to_xy : make_transform([r, th, z] , r * cos(th), r * sin(th), z).

Maxima

Esto devuelve la gráfica de la esfera unitaria definida paramétricamente mediante $(u, v) \to (\text{sen } u \text{ sen } v, \cos u \text{ sen } v, \cos v)$, $0 \leq u \leq 2\pi$, $-\frac{\pi}{2} \leq v \leq \frac{\pi}{2}$.

```
(%i21) wxplot3d([cos(u)*cos(v), sin(u)*cos(v), sin(v)],
[u, 0, 2*%pi], [v, -%pi/2, %pi/2]);
```

(%t21)



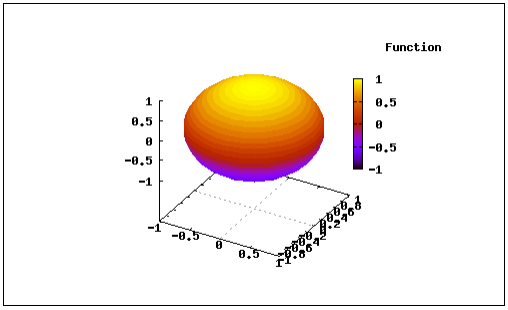
(%o21)

Maxima

Utilizando las opciones adecuadas es posible mejorar la presentación de la salida %o21.

```
(%i22) wxplot3d([cos(u)*cos(v), sin(u)*cos(v), sin(v)],
[u, 0, 2*%pi], [v, -%pi/2, %pi/2],
[gnuplot_preamble, "set size ratio 1"]);
```

(%t22)



(%o22)

14.7 Gráficos de densidad y contornos

Un gráfico de contorno le da esencialmente un "mapa topográfico" de una función. Los contornos unen puntos sobre la superficie que tienen la misma altura. El valor por defecto permite obtener contornos correspondientes a una secuencia de valores de z igualmente espaciados.

`plot d`$(f(x,y),$ $[x, x_{min}, x_{max}],$ $[y, y_{min}, y_{max}],$ [`gnuplot preamble`, `"set pm d map"`])	traza un gráfico de densidad de $f(x,y)$ en el rectángulo $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$
`contour plot`$(f(x,y),$ $[x, x_{min}, x_{max}],$ $[y, y_{min}, y_{max}])$	dibuja las curvas de nivel de $f(x,y)$ en el rectángulo $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ (cualesquiera otros argumentos adicionales se tratan como en `plot d`)

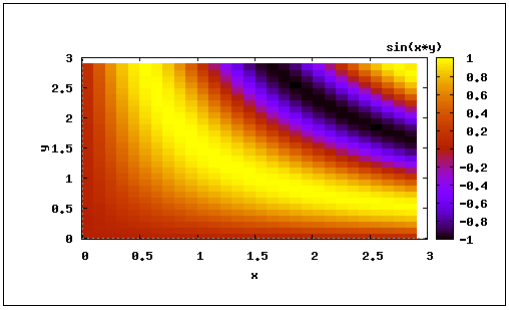
Gráficos de contornos.

Maxima

Esto traza un gráfico de densidad de la función $f(x, y) = \text{sen}(xy)$ en el rectángulo $[0, 3] \times [0, 3]$.

```
(%i23)  wxplot3d(sin(x*y),[x,0,3],[y,0,3],
        [gnuplot preamble, "set pm3d map"]);
```

(%t23)

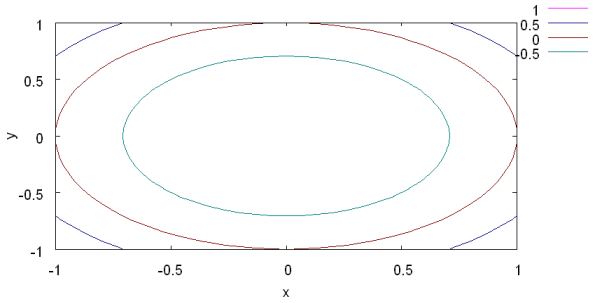


(%o23)

Maxima

He aquí un gráfico de contorno de $f(x, y) = x^2 + y^2 - 1$.

```
(%i24) contour_plot(x^2 + y^2 - 1, [x, -1, 1], [y, -1, 1]);
(%o24)
```



14.8 Gráficos animados

También es posible crear animaciones, aunque únicamente en el entorno de wxMaxima. Para tal fin se utiliza la función `with_slider`[5] con la cual se genera un gráfico idéntico al que se genera con `wxplot2d`, sin embargo dicho gráfico puede ser animado selecionándolo y pulsando luego el botón Comenzar animación, del cuadro de controles, de la barra de herramientas.

Maxima

Cuadro de controles, ubicado en la barra de herramientas, para la animación de gráficos.

`with_slider`	función para animar gráficos de `wxplot2d`

Obtención de gráficos animados.

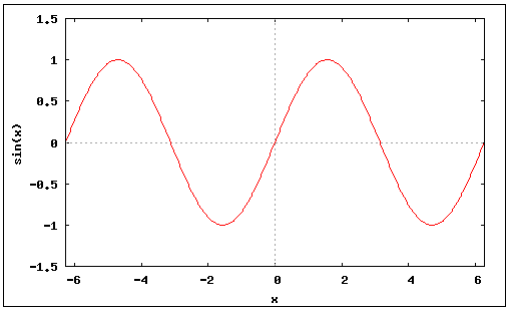
[5] Es importante señalar que adicionalmente estan incluidas las funciones `with_slider_draw` y `with_slider_draw3d` relacionadas con las funciones `draw` y `draw3d` del paquete `draw`.

Maxima

Esto genera un gráfico listo para ser animado. Para conseguir la animación primero se selecciona el gráfico y luego se pulsa el botón ▷, del cuadro de controles, y automáticamente ésta es generada. Para detenerla basta pulsar el botón □, del mismo cuadro. También es posible navegar a través de cada cuadro de la animación con tan sólo arrastrar a voluntad el botón, después de haber seleccionado el gráfico.

```
(%i25) with_slider(a,[0,1,2,3,4,5],sin(x+a),
       [x,-2*%pi,2*%pi],[y,-1,5,1,5],[color,red]);
```

(%t25)



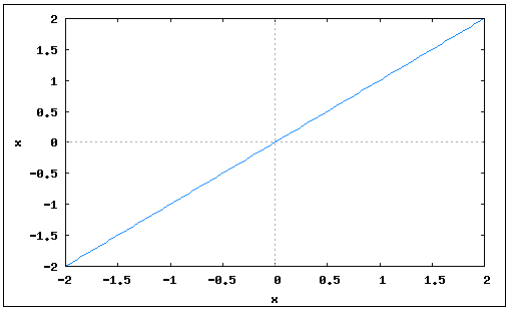
(%o25)

Maxima

He aquí otro gráfico listo para animar.

```
(%i26) with_slider(f,[x,x^2,x^3,x^4],f,[x,2,2]);
```

(%t26)



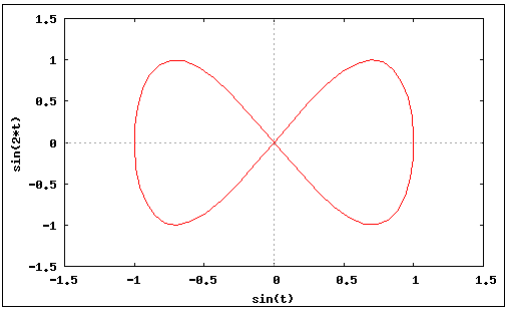
(%o26)

Maxima

Este es un gráfico más listo para animar.

```
(%i27) with slider(a,[0,1,2,3,4,5],
       [parametric,sin(t),sin(2*(t+a)),[t,0,2*%pi]],
       [x,-1·5,1·5],[y,-1·5,1·5],[color,red],[nticks,50]);
```

(%t27)



(%o27)

Capítulo 15

Gráficos con `draw`

El paquete `draw` se distribuye conjuntamente con Maxima y constituye una interfaz que comunica de manera muy eficiente Maxima con Gnuplot. Este paquete incorpora una considerable variedad de funciones y opciones que permiten obtener la representación de un amplio número de objetos gráficos bidimensionales y tridimensionales.

Para poder utilizar el paquete `draw` es preciso cargarlo en la memoria, y para ello se utiliza la función `load` (ver sección 3.7).

`load(draw)` $	"carga" (inicializa) el paquete `draw`
`draw`(gr2d, . . . , gr3d,. . . , opciones)	representa gráficamente una serie de escenas; sus argumentos son objetos gr2d y/o gr3d, junto con algunas opciones. Por defecto, las escenas se representan en una columna
`draw2d`(opciones, objeto_gráfico, . . .)	esta función es un atajo para `draw`(gr2d(opciones, objeto_gráfico, . . .))
`draw d`(opciones, objeto_gráfico, . . .)	esta función es un atajo para `draw`(gr3d(opciones, objeto_gráfico, . . .))

Inicialización del paquete `draw` y descripción de sus tres funciones principales.

15.1 Objetos gráficos bidimensionales

points([[x_1, y_1], [x_2, y_2], ...])	puntos bidimensionales
points([x_1, x_2, ...], [y_1, y_2, ...])	puntos bidimensionales
points([y_1, y_2, ...])	equivale a points([[1, y_1], [2, y_2], ...])
points(matrix([x_1, y_1], [x_2, y_2], ...)	puntos bidimensionales
pol gon([[x_1, y_1], [x_2, y_2], ...])	polígono
pol gon([x_1, x_2, ...], [y_1, y_2, ...])	polígono
rectangle([x_1, y_1], [x_2, y_2])	rectángulo de vértices opuestos (x_1, y_1) y (x_2, y_2)
vector([p_1, p_2], [v_1, v_2])	vector (v_1, v_2) con punto de aplicación (p_1, p_2)
label([cadena, x, y], ...)	etiquetas para gráficos bidimensionales
ellipse(x_c, y_c, a, b, ang_1, ang_2)	sector elíptico de centro (x_c, y_c) con semiejes horizontal y vertical a y b, respectivamente, comenzando en el ángulo ang_1 y trazando un arco de amplitud igual al ángulo ang_2
explicit($f(x)$, x, x_{min}, x_{max})	función explícita f cuya variable x toma valores desde x_{min} hasta x_{max}
implicit($E(x, y)$, x, x_{min}, x_{max}, y, y_{min}, y_{max})	expresión implícita E a ser representada en el rectángulo $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$
parametric(f_x, f_y, t, t_{min}, t_{max})	curva paramétrica bidimensional, cuyo parámetro t toma valores desde t_{min} hasta t_{max}
polar($r(\theta)$, θ, θ_{min}, θ_{max})	función polar r cuya variable θ toma valores desde θ_{min} hasta θ_{max}

Principales objetos gráficos bidimensionales incorporados en draw.

Las funciones `draw`, `draw2d` y `draw d` devuelven las salidas gráficas en una ventana de Gnuplot, aparte de la ventana de trabajo actual. No obstante el entorno gráfico wxMaxima permite utilizar las funciones `wxdraw`, `wxdraw2d` y `wxdraw d` que devuelven las salidas en el cuaderno de trabajo actual. Debe tenerse presente que al utilizar la función `wxdraw d`, el punto de vista de la gráfica obtenida no puede ser cambiado en tiempo real.

En este capítulo se van a mostrar los resultados mediante las funciones `wxdraw`, `wxdraw2d` y `wxdraw d`, pero todos los ejemplos mostrados pueden ser ejecutados, sin ningún problema, con las funciones `draw`, `draw2d` y `draw d`, según sea el caso.

Maxima

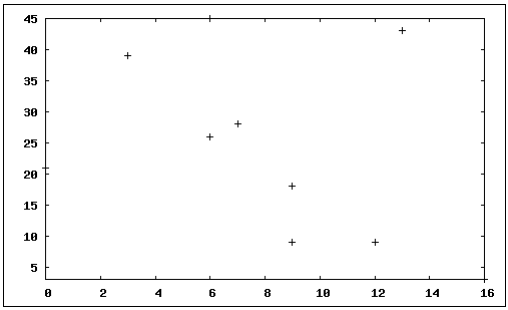
Aquí se genera una lista de listas. Los elementos de la misma, generados aleatoriamente, representan las coordenadas de puntos del plano.

```
(%i1) p:create list([random(20),random(50)],k,1,10);
(%o1) [[9,9],[3,39],[6,26],[16,3],[0,21],[12,9],[9,18],[6,45],
      [13,43],[7,28]]
```

Maxima

Después de aplicar la función `points` a la variable *p*, en la que se han almacenado los puntos, se obtiene un objeto gráfico el cual puede graficarse con `draw2d`.

```
(%i2) wxdraw2d( points(p));
(%t2)
```



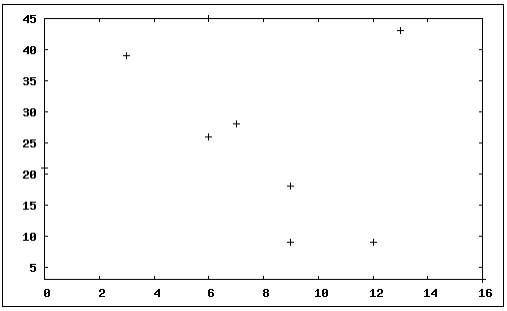

```
(%o2) [gr2d(points)]
```

Maxima

Con gr2d y draw siempre se obtendrá el mismo resultado que con draw2d.

```
( %i3)  wxdraw( gr2d(points(p)) );
```

(%t3)



(%o3) [gr2d(points)]

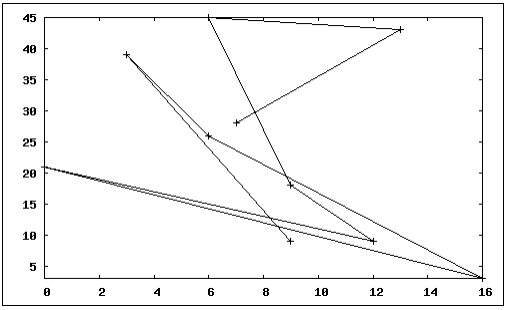
draw no cuenta con una función específica, como line, para definir el objeto geométrico línea; simplemente se asigna el valor true a la opción points joined para unir los puntos dados mediante segmentos.

Maxima

Gráfica de los puntos *p* unidos mediante segmentos.

```
( %i4)  wxdraw2d(
        points joined = true,
        points(p) );
```

(%t4)



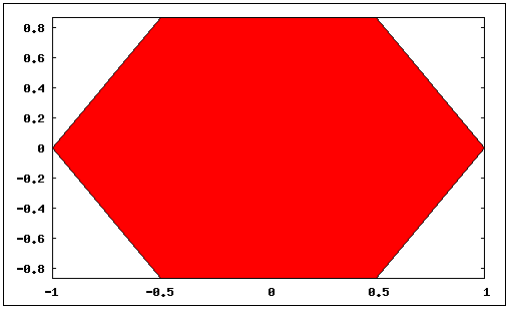
(%o4) [gr2d(points)]

Maxima

Así se visualiza la gráfica de un hexágono.

```
( %i5)  wxdraw2d(
         polygon([
        [1/2, sqrt(3)/2], [-1/2, sqrt(3)/2], [-1, 0],
        [-1/2, -sqrt(3)/2], [1/2, -sqrt(3)/2], [1, 0]
        ])
```

(%t5)



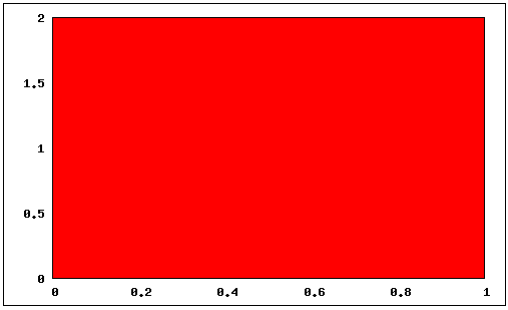
(%o5) [gr2d(polygon)]

Maxima

Y así, la gráfica de un rectángulo.

```
( %i6)  wxdraw2d(
        rectangle([0, 0], [1, 2])
        );
```

(%t6)



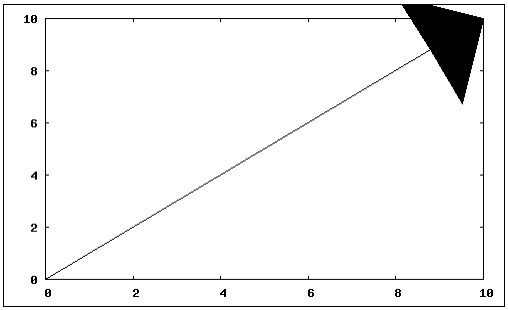
(%o6) [gr2d(rectangle)]

Maxima

De esta manera se obtiene la gráfica del vector (10, 10), cuyo punto de aplicación es el origen.

```
( %i7)  wxdraw2d(
        vector([0, 0], [10, 10])
        );
```

(%t7)



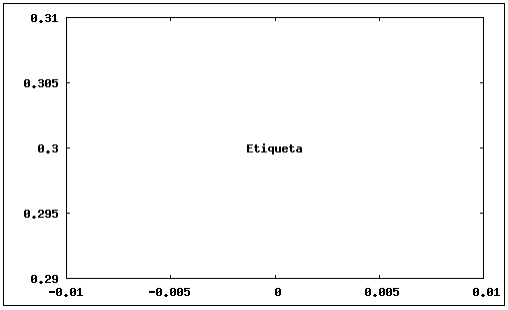
(%o7) [gr2d(vector)]

Maxima

Para insertar texto en un punto cualquiera de un gráfico se utiliza la función `label`. En este caso el punto de inserción es (0, 0 · 3).

```
( %i8)  wxdraw2d(
        label(["Etiqueta", 0, 0 3])
        );
```

(%t8)



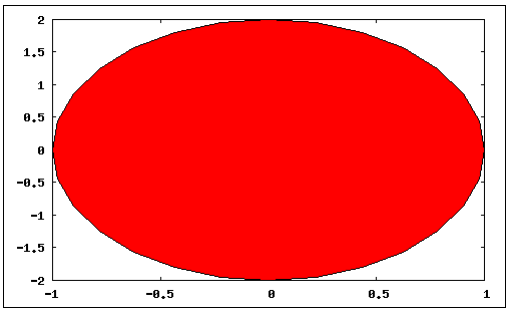
(%o8) [gr2d(label)]

Maxima

Esto devuelve la gráfica de un sector elíptico, de 0° a 360°, cuyo centro es el origen, su semieje horizontal es 1, su semieje vertical es 2.

```
(%i9)  wxdraw2d(
       ellipse(0, 0, 1, 2, 0, 360)
       );
```

(%t9)



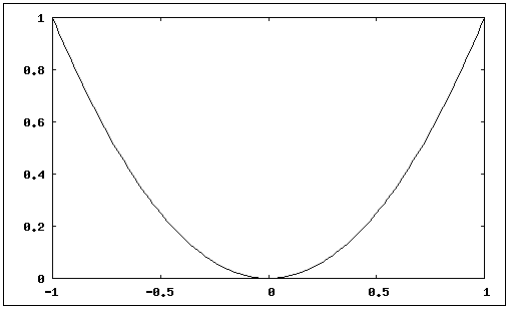
(%o9) [gr2d(ellipse)]

Maxima

De esta manera se obtiene la gráfica de la función $x \to x^2$, con $-1 \leq x \leq 1$.

```
(%i10)  wxdraw2d(
        explicit(x^2, x, -1, 1)
        );
```

(%t10)



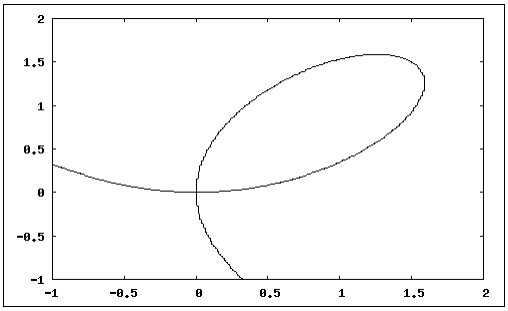
(%o10) [gr2d(explicit)]

Maxima

Esto muestra la gráfica de la ecuación $x^3 + y^3 - 3xy = 0$ en el rectángulo $[-1, 2] \times [-1, 2]$.

```
(%i11) wxdraw2d(
       implicit(x^3 + y^3 - 3 * x * y = 0, x, -1, 2, y, -1, 2)
       );
```

(%t11)



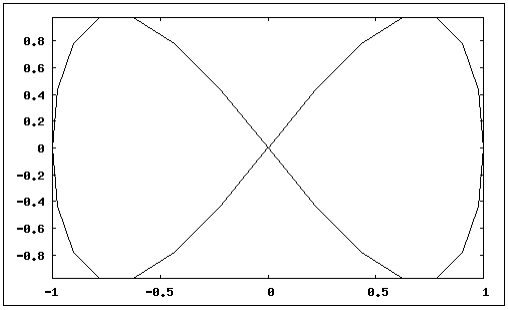
(%o11) [gr2d(implicit)]

Maxima

Aquí se muestra la gráfica de la curva definida en forma paramétrica mediante $t \to (sin(t), sin(2t))$, con $0 \le t \le 2\pi$.

```
(%i12) wxdraw2d(
       parametric(sin(t), sin(2*t), t, 0, 2 * %pi)
       );
```

(%t12)



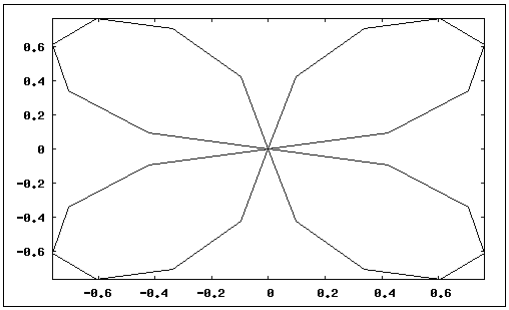
(%o12) [gr2d(parametric)]

Maxima

Aquí se muestra la gráfica de la función definida en coordenadas polares mediante $t \to \text{sen}(t)$, tal que $0 \leq t \leq 2\pi$.

```
( %i13) wxdraw2d(
        polar(sin(2*t), t, 0, 2*%pi)
        );
```

(%t13)



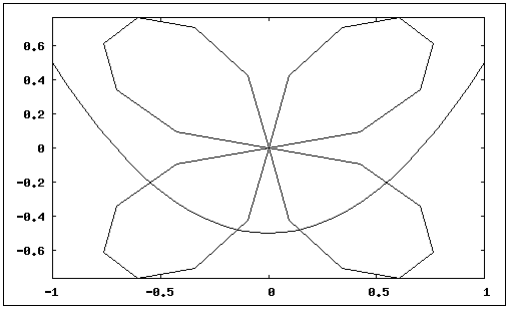
(%o13) [gr2d(polar)]

Maxima

Combinar objetos gráficos bidimensionales es sencillo con `draw2d`.

```
( %i14) wxdraw2d(
        parametric(t, t^2 - 1/2, t, -1, 1),
        polar(sin(2* t), t,0, 2* %pi)
        );
```

(%t14)



(%o14) [gr2d(parametric, polar)]

15.2 Opciones para los objetos gráficos bidimensionales

15.2.1 Opciones locales

Las opciones locales sólo son relevantes para objetos gráficos específicos y para tal efecto deben digitarse antes de dicho objeto. Si una opción gráfica es digitada antes de un objeto gráfico al cual no corresponde, no se produce ningún mensaje de error, simplemente la gráfica se muestra sin presentar alteración alguna.

Opción	Val. por defecto	Descripción
`point si e`	1	establece el tamaño de los puntos dibujados (debe ser un número no negativo)
`point t pe`	1	indica la forma que tendrán los puntos aislados
`point joined`	`false`	indica si los puntos aislados se unen mediante segmentos o no

Opciones de `draw2d` para el objeto gráfico `points`.

Opción	Val. por defecto	Descripción
`nticks`	29	indica el número de puntos a utilizar para generar las gráficas
`border`	`true`	especifica si debe dibujarse el borde o no
`transparent`	`false`	establece si el polígono debe rellenarse o no
`fill color`	`red`	especifica el color del relleno del polígono

Opciones de `draw2d` para los objetos gráficos bidimensionales `polygon` y `rectangle`.

Opción	Val. por defecto	Descripción
head both	false	indica si el vector será bidireccional o no
head length	2	indica, en las unidades del eje x, la longitud de las flechas de los vectores
head angle	45	indica el ángulo, en grados, entre la flecha y el segmento del vector
head t pe	filled	especifica cómo se habrán de dibujar las flechas de los vectores
unit vectors	false	especifica si los vectores se dibujan con módulo unidad

Opciones de draw2d para el objeto gráfico vector.

Opción	Val. por defecto	Descripción
label alignment	center	especifica el color que tendrá el vector
label orientation	hori ontal	especifica el color que tendrá el vector

Opciones de draw2d para el objeto gráfico label.

Opción	Val. por defecto	Descripción
nticks	29	indica el número de puntos a utilizar para generar las gráficas
adapt depth	10	indica el número máximo de particiones utilizadas por la rutina gráfica adaptativa

Opciones de draw2d para el objeto gráfico explicit.

Opción	Val. por defecto	Descripción
filled func	false	establece cómo se van a rellenar las regiones limitadas por las gráficas
fill color	red	especifica el color para rellenar las regiones limitadas por las gráficas

Opciones de draw2d para el objeto gráfico explicit.

Opción	Val. por defecto	Descripción
ip grid	[50, 50]	establece la rejilla del primer muestreo para la gráfica
ip grid in	[5, 5]	establece la rejilla del segundo muestreo para la gráfica

Opciones de draw2d para el objeto gráfico implicit.

Opción	Val. por defecto	Descripción
nticks	29	indica el número de puntos a utilizar para generar las gráficas

Opciones de draw2d para los objetos gráficos bidimensionales parametric y polar.

15.2.2 Opciones locales genéricas

Las opciones locales genéricas son opciones comunes a todos los objetos gráficos bidimensionales. Aunque hay excepciones que cabe destacar, por ejemplo las opciones line width y line t pe que, por razones obvias, únicamente no son relevantes con los objetos gráficos point y label. Tampoco la opción color es relevante con el objeto gráfico label, esto constituye una falla que será corregida cuando se incorpore, a Maxima, la versión 4.3 de Gnuplot.

Opción	Val. por defecto	Descripción
color	black	especifica el color para dibujar líneas, puntos y bordes de polígonos
line width	1	indica el ancho de las líneas a dibujar
line t pe	solid	indica cómo se van a dibujar las líneas (valores posibles son solid y dots)
ke	solid	indica la clave del gráfico en la leyenda

Opciones locales genéricas de draw2d.

15.2.3 Opciones globales

Las opciones globales se caracterizan porque afectan a toda la escena y su posición dentro de la descripción de ésta no reviste importancia.

Opción	Val. por defecto	Descripción
proportional axes	none	indica si una escena bidimensional se dibujará con los ejes proporcionales a sus longitudes relativas (valores posibles: none y x)
xrange	auto	permite especificar un intervalo para x (valores posibles: auto y $[x_{min}, x_{max}]$)
range	auto	permite especificar un intervalo para y (valores posibles: auto y $[y_{min}, y_{max}]$)
logx	false	permite especificar si el eje x se dibujará en la escala logarítmica

Algunas opciones globales de draw2d.

Opción	Val. por defecto	Descripción
log	false	permite especificar si el eje y se dibujará en la escala logarítmica
terminal	screen	selecciona el terminal a utilizar por Gnuplot (valores posibles son: screen, png, jpg, eps, eps color, pdf, pdfcairo, gif, animated gif, wxt y aquaterm)
file name maxima out		indica el nombre del fichero en el que los terminales png, jpg, eps, eps color, pdf, pdfcairo, etc. guardarán el gráfico
font	""	permite seleccionar el tipo de fuente a utilizar por el terminal
font	12	permite seleccionar el tamaño de la fuente a utilizar por el terminal
grid	false	permite especificar si se dibujará una rejilla sobre xy
title	""	almacena una cadena con el título de la escena
xlabel	""	almacena una cadena con la etiqueta del eje x
label	""	almacena una cadena con la etiqueta del eje y
xtics	auto	controla la forma en la que se dibujarán las marcas del eje x (valores posibles: auto, none, [*inicio, inc, fin*], {$n_1, n_2, \ldots$} y también {["label1", n_1], ["label1", n_1], ...})

Algunas opciones globales de draw2d.

Opción	Val. por defecto	Descripción
`xtics axis`	`false`	indica si las marcas y sus etiquetas se dibujan sobre el propio eje x, o se colocan a lo largo del borde del gráfico
`tics`	`auto`	similar que `xtics` pero con el eje y
`tics axis`	`false`	similar que `xtics axis` pero con el eje y
`xaxis`	`false`	especifica si se dibujará el eje x
`xaxis width`	1	indica el ancho del eje x
`xaxis t pe`	`dots`	indica cómo se debe dibujar el eje x (valores admisibles: `solid` y `dots`)
`xaxis color`	`black`	indica el color para el eje x
`axis`	`false`	especifica si se dibujará el eje y
`axis width`	1	indica el ancho del eje y
`axis t pe`	`dots`	similar que `xaxis t pe` pero con el eje y
`axis color`	`black`	indica el color para el eje y
`file bgcolor`	`"xffffff"`	establece el color de fondo (en código hexadecimal rgb) para los terminales `png`, `jpg` y `gif`
`dela`	5	establece el retraso en centésimas de segundo entre imágenes en los ficheros `gif` animados
`eps width`	12	indica el ancho (en cm) del archivo Postscript generado por los terminales `eps` y `eps color`
`eps height`	12	indica el largo (en cm) del archivo Postscript

Algunas opciones globales de `draw2d`.

Opción	Val. por defecto	Descripción
`pdf width`	21 · 0	especifica el ancho (en cm) del documento PDF generado por los terminales `pdf` y `pdfcairo`
`pdf height`	29 · 7	especifica el largo (en cm) del documento PDF
`pic width`	640	especifica la anchura del fichero de imagen de bits generado por los terminales `png` y `jpg`
`pic height`	640	especifica el largo del fichero de imagen de bits
`user preamble`	`""`	inserta código Gnuplot

Algunas opciones globales de `draw2d`.

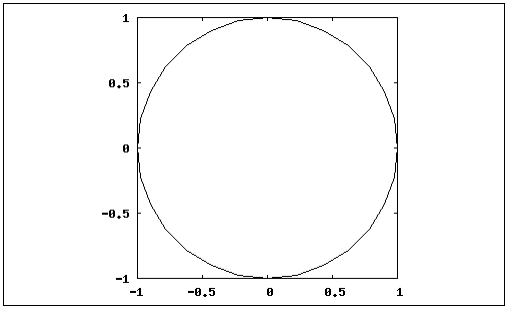
15.2.4 Ejemplos ilustrativos

Maxima

Aquí se usa `user_preamble` para insertar código Gn'Uplot. El resultado, en este caso, equivale a asignar el valor `xy` a la opción `proportional_axes`.

```
(%i15)  wxdraw2d(
        parametric(cos(t),sin(t),t,0,2* %pi),
        user preamble = "set size ratio 1"
        );
```

(%t15)



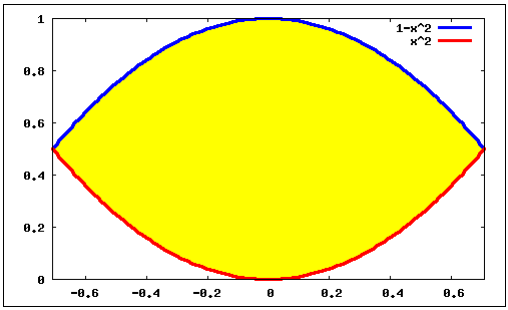
(%o15) [gr2d(parametric)]

Maxima

He aquí la región encerrada por las parábolas $y = x^2$ y $y = 1 - x^2$.

```
(%i16) wxdraw2d(
       filled_func = x^2,
       fill_color = yellow,
       explicit(1 - x^2, x, -sqrt(2)/2, sqrt(2)/2),
       filled_func = false,
       line_width = 3,
       key = "1 - x^2",
       color = blue,
       explicit(1 - x^2, x, -sqrt(2)/2, sqrt(2)/2),
       key = "x^2",
       color = red,
       explicit(x^2, x, -sqrt(2)/2, sqrt(2)/2)
       );
```

(%t16)



(%o16) [gr2d(explicit, explicit, explicit)]

Maxima

Aquí se resuelve un sistema de ecuaciones para encontrar los puntos de intersección de las curvas definidas por $x^2 + y^2 = 1$ y $y - 2x^2 + \frac{3}{2} = 0$.

```
(%i17) sol : solve([x^2 + y^2 = 1, y - 2*x^2 + 3/2 = 0], [x, y]);
```

(%o17)
$$x = -\frac{\sqrt{\sqrt{5}+5}}{2^{\frac{3}{2}}}, y = \frac{\sqrt{5}-1}{4}, x = \frac{\sqrt{\sqrt{5}+5}}{2^{\frac{3}{2}}}, y = \frac{\sqrt{5}-1}{4},$$
$$x = -\frac{\sqrt{5-\sqrt{5}}}{2^{\frac{3}{2}}}, y = -\frac{\sqrt{5}+1}{4}, x = \frac{\sqrt{5-\sqrt{5}}}{2^{\frac{3}{2}}}, y = -\frac{\sqrt{5}+1}{4}$$

Maxima

Esto almacena las coordenadas de los puntos de intersección, previamente calculados, en la variable pts.

```
(%i18) pts : map(lambda([h], subst(h, [x, y])), sol);
```

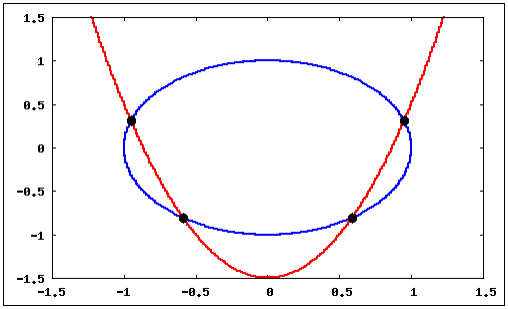
$$(\%o18)\quad \left[\left[x=-\frac{\sqrt{5+\sqrt{5}}}{2^{\frac{3}{2}}},\ y=\frac{\sqrt{5}-1}{-4}\right],\ \left[x=\frac{\sqrt{5+\sqrt{5}}}{2^{\frac{3}{2}}},\ y=\frac{\sqrt{5}-1}{4}\right],\ \left[x=-\frac{\sqrt{5-\sqrt{5}}}{2^{\frac{3}{2}}},\ y=\frac{\sqrt{5}+1}{4}\right],\right.$$
$$\left.\left[x=\frac{\sqrt{5-\sqrt{5}}}{2^{\frac{3}{2}}},\ y=-\frac{\sqrt{5}+1}{4}\right]\right]$$

Maxima

He aquí un gráfico de las curvas definidas por $x^2 + y^2 = 1$ y $y - 2x^2 + \frac{3}{2} = 0$ y los respectivos puntos de intersección.

```
(%i19) wxdraw2d(
       line width = 2,
       color = blue,
       implicit(x^2 + y^2 = 1, x, -1.5, 1.5, y, -1.5, 1.5),
       color = red,
       implicit(y - 2*x^2 + 3/2 = 0, x, -1.5, 1.5,
         y, -1.5, 1.5)
       color = black
       point size = 1,5,
       point type = 7,
       points(pts),
       );
```

(%t19)



(%o19) [gr2d(parametric)]

Maxima

Aquí se define la curva paramétrica a.

```
(%i20)  a(t) := [t, sin(t)]$
```

Maxima

Esto define el campo vectorial tangente asociado a la curva a.

```
(%i21)  define("a'"(t), diff(a(t), t))$
```

Maxima

Aquí se construyen la lista T de vectores tangentes a la curva a y la lista P de puntos de aplicación de éstos. Los valores t_0, para tal construcción, son tomados de la lista t0.

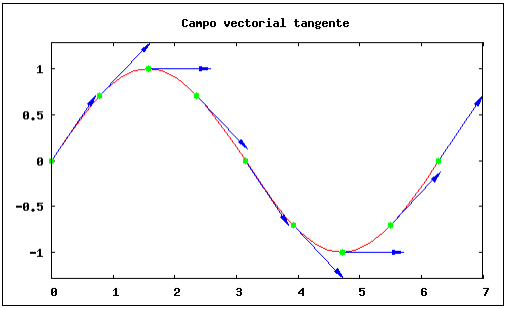
```
(%i22)  t0 : create list(i * %pi/4, i, 0, 8)$
(%i23)  T : create list(vector(a(i), "a'"(i)), i, t0)$
(%i24)  P : map(a, t0)$
```

Maxima

Este gráfico muestra la curva a, un conjunto de vectores unitarios tangentes a ésta y los puntos de aplicación de los mismos.

```
(%i25)  wxdraw2d(
        title = "Campo vectorial tangente",
        color = red,
        parametric(t, sin(t), t, 0, 2 * %pi),
        color = blue,
        head length = 0.2,
        head angle = 10,
        unit vectors = true,
        T[1], T[2], T[3], T[4], T[5], T[6], T[7], T[8], T[9],
        color = green,
        point type = 7,
        points(P)
        );
```

(%t25)



(%o25) [gr2d(parametric, vector, vector, vector, vector, vector, vector, vector, vector, vector, points)]

Maxima

A continuación se construye la lista N de vectores normales a la curva a.

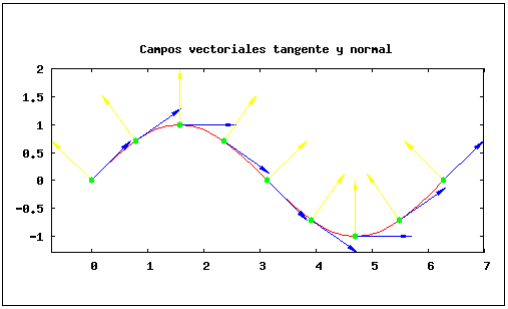
```
( %i26) J(v) := [-last(v), first(v)] $
( %i27) N : create_list(vector(a(i), J("a'"(i))), i, t0) $
```

Maxima

Este gráfico muestra la curva a, un conjunto de vectores unitarios tangentes a la curva, un conjunto de vectores unitarios normales a la misma y los puntos de aplicación de los vectores.

```
( %i28) wxdraw2d(
        title = "Campos vectoriales tangente y normal",
        color = red,
        parametric(t, sin(t), t, 0, 2 * %pi),
        color = blue, head_length = 0.2, head_angle = 10,
        unit_vectors = true,
        T[1], T[2], T[3], T[4], T[5], T[6], T[7], T[8], T[9],
        color = yellow,
        N[1], N[2], N[3], N[4], N[5], N[6], N[7], N[8], N[9],
        color = green, point_type = 7,
        points(P),
        proportional_axes = xy
        ) $
```

(%t28)

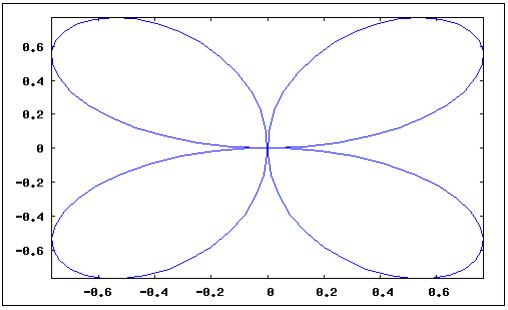


Maxima

He aquí un gráfico polar mejorado.

```
(%i29) wxdraw2d( color = blue, nticks = 100,
       polar(sin(2 * t), t, 0, 2 * %pi) );
```

(%t29)



(%o29) [gr2d(polar)]

15.3 Objetos gráficos tridimensionales

`points`([[x_1, y_1, z_1], [x_2, y_2, z_2], ...])	puntos tridimensionales
`points`([x_1, x_2, ...], [y_1, y_2, ...], [z_1, z_2, ...])	puntos tridimensionales

Principales objetos gráficos tridimensionales incorporados en `draw`.

points(matrix([x_1, y_1, z_1], [x_2, y_2, z_2], ...)	puntos tridimensionales
vector([p_1, p_2, p_3], [v_1, v_2, v_3])	vector (v_1, v_2, v_3) con punto de aplicación (p_1, p_2, p_3)
label([cadena, x, y, z], ...)	etiquetas para gráficos tridimensionales
explicit($f(x,y)$, x, x_{min}, x_{max}, y, y_{min}, y_{max})	función explícita f cuyas variables satisfacen $x_{min} \leq x \leq x_{max}$ y $y_{min} \leq y \leq y_{max}$
implicit($E(x,y,z)$, x, x_{min}, x_{max}, y, y_{min}, y_{max}) z, z_{min}, z_{max})	expresión implícita E a ser representada en el paralelepípedo $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$
parametric(f_x, f_y, f_z t, t_{min}, t_{max})	curva paramétrica tridimensional, cuyo parámetro t satisface $t_{min} \leq t \leq t_{max}$
parametric surface(f_x, f_y, f_z, u, u_{min}, u_{max}, v, v_{min}, v_{max})	superficie definida paramétricamente, cuyos parámetros satisfacen $u_{min} \leq u \leq u_{max}$ y $v_{min} \leq v \leq v_{max}$
c lindrical($r(z,\theta)$, z, z_{min}, z_{max}, θ, θ_{min}, θ_{max})	función cilíndrica r cuyas variables satisfacen $z_{min} \leq z \leq z_{max}$ y $\theta_{min} \leq \theta \leq \theta_{max}$
spherical($r(\varphi,\theta)$, φ, φ_{min}, φ_{max}, θ, θ_{min}, θ_{max})	función esférica r cuyas variables satisfacen $\varphi_{min} \leq \varphi \leq \varphi_{max}$ y $\theta_{min} \leq \theta \leq \theta_{max}$
mesh(m, x_0, y_0, *ancho*, *largo*)	define un gráfico tridimensional de la matriz m (los valores z se toman de m, las abscisas van desde x_0 hasta x_0 + *ancho* y las ordenadas desde y_0 hasta y_0 + *largo*)
tube(f_x, f_y, f_z, f_r, t, t_{min}, t_{max})	superficie tubular de radio f_r, generada a partir de una curva paramétrica tridimensional, cuyo parámetro t satisface $t_{min} \leq t \leq t_{max}$

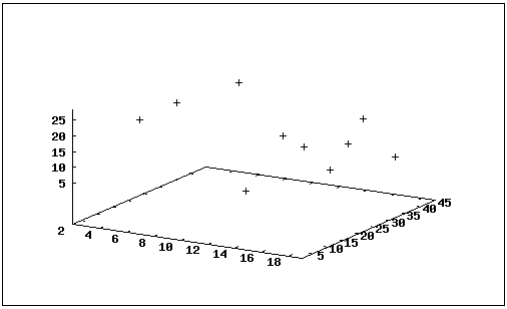
Principales objetos gráficos tridimensionales incorporados en draw.

Maxima

Aquí se genera, en forma aleatoria, un conjunto de puntos tridmensionales. Luego se muestra la gráfica de los dichos puntos.

```
( %i30)  p:create list(map(random,[20,50,30]),k,1,10)$
( %i31)  wxdraw3d( points( p ) );
```

(%t31)



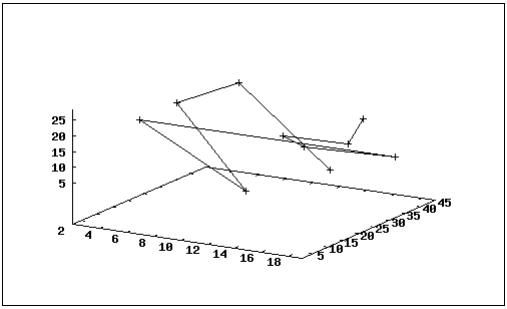
(%o31) [gr3d(points)]

Maxima

Asignando el valor `true` a la opción `plot_joined` se obtiene la poligonal que une los puntos almacenados en `p`.

```
( %i32)  wxdraw3d(
         points joined =true,
         points( p )
         );
```

(%t32)



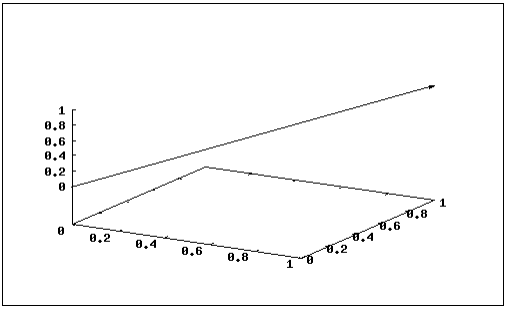
(%o32) [gr3d(points)]

Maxima

Aquí se muestra la gráfica del vector cuyo punto de aplicación es el origen y cuya parte vectorial es (1, 1, 1).

```
(%i33) wxdraw3d(
       vector([0, 0, 0], [1, 1, 1])
       );
```

(%t33)



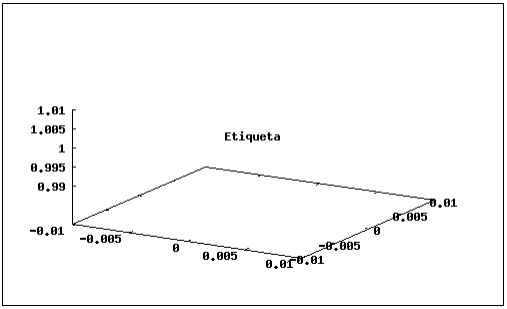
(%o33) [gr3d(vector)]

Maxima

He aquí un texto insertado en el punto (0, 0, 1).

```
(%i34) wxdraw3d(
       label(["Etiqueta", 0, 0, 1])
       );
```

(%t34)



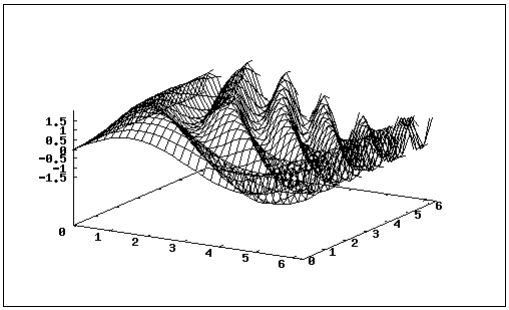
(%o34) [gr3d(label)]

Maxima

Esto muestra la gráfica de la función $(x, y) \to \text{sen } x + \text{sen}(xy)$, con $0 \le x \le 2\pi$ y $0 \le y \le 2\pi$.

```
(%i35) wxdraw3d(
       explicit(sin(x)+sin(x*y),x,0,2*%pi,y,0,2*%pi)
       );
```

(%t35)



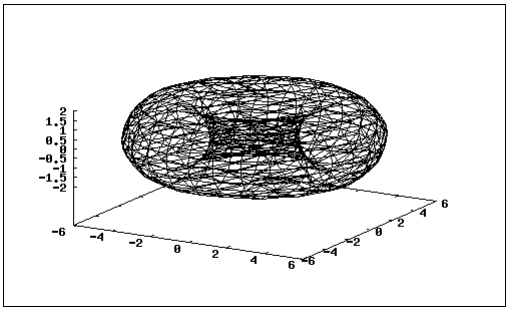
(%o35) [gr3d(explicit)]

Maxima

Aquí se muestra la gráfica de la superficie generada a partir de la ecuación $(\sqrt{x^2 + y^2} - 4)^2 + z^2 = 4$, con $-6 \le x \le 6$, $-6 \le y \le 6$ y $-2 \le z \le 2$.

```
(%i36) wxdraw3d(
       implicit((sqrt(x^2+y^2)-4)^2+z^2=4,x,-6,6,
       y,-6,6,z,-2,2) );
```

(%t36)



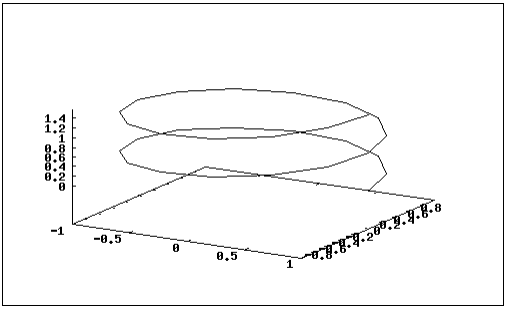
(%o36) [gr3d(implicit)]

Maxima

Esta es la gráfica de la curva $t \to (\cos t, \text{sen}\, t, \frac{t}{8})$, con $0 \le t \le 4\pi$.

```
(%i37) wxdraw3d(
       parametric(cos(t), sin(t), t/8, t, 0, 4 * %pi)
       );
```

(%t37)



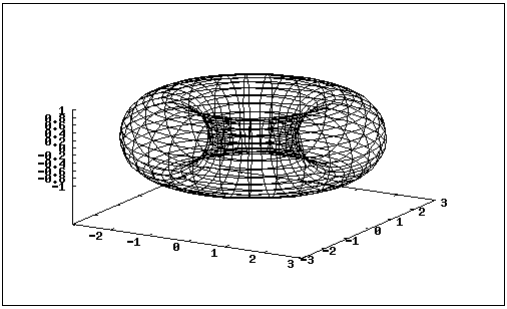
(%o37) [gr3d(parametric)]

Maxima

Aquí se muestra la gráfica de la superficie definida por $(u, v) \to ((2+\cos v)\cos u, (2+\cos v)\sin u, \sin v)$, con $0 \le u \le 2\pi$ y $0 \le v \le 2\pi$.

```
(%i38) wxdraw3d(
       parametric surface((2 + cos(v)) * cos(u),
       (2 + cos(v)) * sin(u), sin(v),
       u, 0, 2 * %pi, v, 0, 2 * %pi) );
```

(%t38)



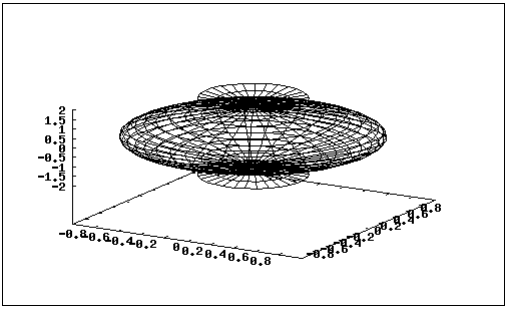
(%o38) [gr3d(parametric surface)]

Maxima

Esta es la gráfica de la superficie definida en coordenadas cilíndricas mediante $(z,\ t) \rightarrow \cos z$, con $-2 \leq z \leq 2$ y $0 \leq t \leq 2\pi$.

```
(%i39) wxdraw3d(
cylindrical(cos(z), z, -2, 2, t, 0, 2 * %pi)
);
```

(%t39)



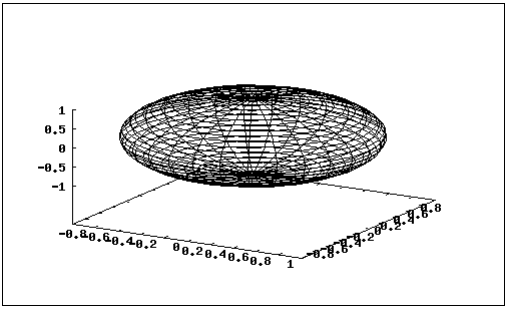
(%o39) [gr3d(cylindical)]

Maxima

He aquí la gráfica de la superficie definida en coordenadas esféricas mediante $(a,\ t) \rightarrow 1$, con $0 \leq a \leq 2\pi$ y $0 \leq t \leq \pi$.

```
(%i40) wxdraw3d(
spherical(1, a, 0, 2 * %pi, t, 0, %pi)
);
```

(%t40)



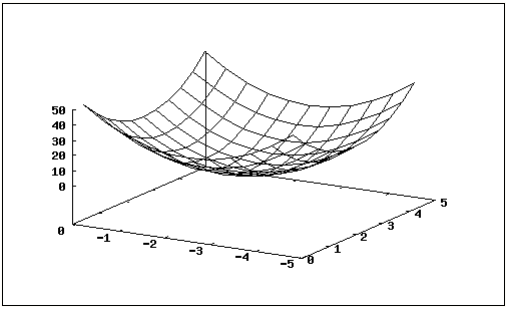
(%o40) [gr3d(spherical)]

Maxima

Esta es la gráfica de un objeto geométrico tridimensional de tipo mesh.

```
(%i41) m : apply(matrix, create list(create list(k^2 + i^2,
         k, -5, 5), i, -5, 5)) $
(%i41) wxdraw3d( mesh(m, 0, 0, -5, 5) );
```

(%t41)



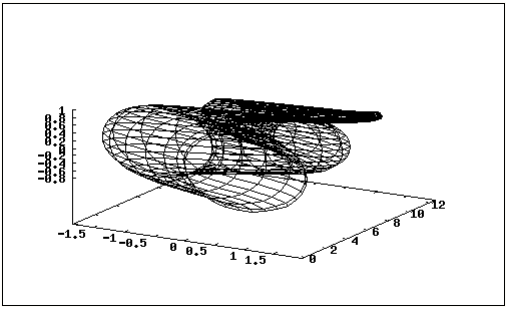
(%o41) [gr3d(mesh)]

Maxima

He aquí la gráfica de la superficie tubular, de radio $\cos\left(\frac{t}{10}\right)^2$, generada por la curva $t \to (\cos t,\ t,\ 0)$, con $0 \le t \le 4\pi$.

```
(%i42) wxdraw3d(
       tube(cos(t), t, 0, cos(t/10)^2, t, 0, 4 * %pi)
       );
```

(%t42)



(%o42) [gr3d(tube)]

15.4 Opciones para objetos gráficos tridimensionales

15.4.1 Opciones locales

Para los objetos gráficos `points` y `label` las opciones locales son las mismas en `draw2d` y `draw d`. En cambio para el objeto gráfico `vector` las opciones `head length` y `head angle` únicamente son relevantes en `draw2d` (vea la subsección 15.2.1). Por este motivo, en esta sección, no se presentan tablas de opciones para los tres objetos gráficos mencionados.

Opción	Val. por defecto	Descripción
`xu grid`	30	indica el número de coordenadas de x para formar la rejilla de puntos muestrales
`v grid`	30	indica el número de coordenadas de y para formar la rejilla de puntos muestrales

Opciones de `draw3d` para `explicit`, `parametric_surface`, `cylindrical`, `spherical` y `tube`.

Opción	Val. por defecto	Descripción
`x voxel`	10	indica el número de voxels en la dirección x a utilizar por el algoritmo implementado
`voxel`	10	indica el número de voxels en la dirección y a utilizar por el algoritmo implementado
`voxel`	10	indica el número de voxels en la dirección z a utilizar por el algoritmo implementado

Opciones de `draw3d` para `implicit`.

Opción	Val. por defecto	Descripción
nticks	29	indica el número de puntos a utilizar para generar las gráficas

Opciones de draw3d para parametric.

15.4.2 Opciones locales genéricas

Vea la subsección 15.2.2.

15.4.3 Opciones globales

A excepción de la opción proportional axes todas las opciones globales de draw2d son las mismas de draw d (vea la subsección 15.2.3). Además draw d incorpora opciones que complentan, de forma natural, las incorporadas en draw2d. Por ejemplo, estan las opciones range, log , label, tics, axis, axis t pe y axis color. Otras opciones se describen a continuación.

Opción	Val. por defecto	Descripción
x plane	false	coloca el plano xy en escenas tridimensionales (para el valor false, el plano xy se coloca automáticamente; en cambio, si toma un valor real, éste intersectará con el eje z a ese nivel)
rot vertical	60	indica el ángulo (en grados) de la rotación vertical (alrededor del eje x) para situar el punto del observador en las escenas tridimensionales (el ángulo debe pertenecer al intervalo [0, 180])

Algunas opciones globales de draw3d.

Opción	Val. por defecto	Descripción
`rot hori ontal`	30	indica el ángulo (en grados) de la rotación horizontal (alrededor del eje z) para situar el punto del observador en las escenas tridimensionales (el ángulo debe pertenecer al intervalo [0, 360])
`axis d`	`true`	indica si los ejes x, y y z, tradicionales, permanecerán visibles
`palette`	`color`	es un vector de longitud tres con sus componentes tomando valores enteros en el rango desde −36 a +36; cada valor es un índice para seleccionar una fórmula que transforma los niveles numéricos en las componentes cromáticas rojo, verde y azul (`palette = gra` y `palette = color` son atajos para `palette = ,  ,` y `palette = 7, 5, 15` , respectivamente)
`enhanced d`	`false`	si `enhanced d` vale `true`, los objetos gráficos se colorearán activando el modo pm3d de Gnuplot. Si se da una expresión a `enhanced d` (excepto en `implicit`), ésta se utilizará para asignar colores de acuerdo con el valor de `palette`; las variables de esta expresión deben ser las mismas que luego se utilicen para la descripción de la superficie

Algunas opciones globales de `draw3d`.

Opción	Val. por defecto	Descripción
surface hide	false	establece si las partes ocultas se mostrarán o no en las superficies
contour	none	permite decidir dónde colocar las líneas de nivel (valores posibles: none, base, surface, both y map)
contour levels	5	controla cómo se dibujarán las líneas de nivel (valores posibles: *n*, [*inicio*, *inc*, *f in*] y $\{n_1, n_2, \ldots\}$

Algunas opciones globales de draw3d.

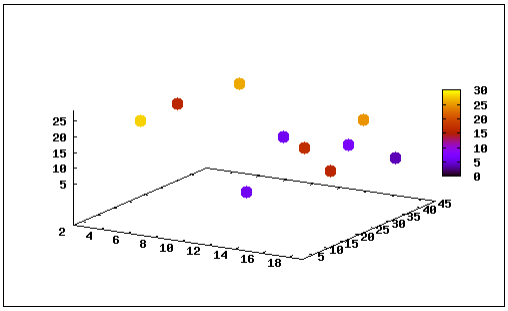
15.4.4 Ejemplos ilustrativos

Maxima

Aquí se han cambiado los valores por defecto de dos opciones de draw3d. El resultado se aprecia al mostrar los puntos generados en (%i30) .

```
( %i43)  wxdraw2d(
         enhanced3d = true,
         point type = 7, point size = 2,
         points( p)
         );
```

(%t43)



(%o43) [gr3d(points)]

Maxima

Esto define una curva, algunos puntos sobre ésta y algunos vectores tangentes a la misma.

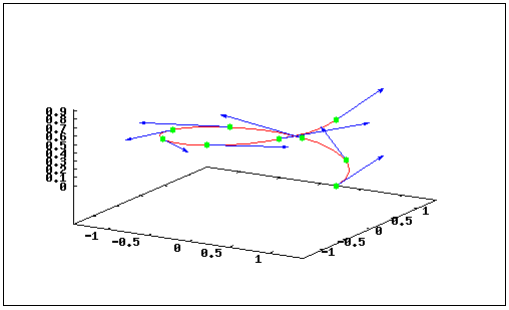
```
(%i44) a(t) := [cos(t), sin(t), t/8]$
(%i45) define("a'"(t), diff(a(t), t))$
(%i46) t0 : create list(i * %pi/4, i, 0, 8)$
(%i47) T : create list(vector(a(i), "a'"(i)), i, t0)$
(%i48) P : map(a, t0)$
```

Maxima

He aquí la gráfica de todos los objetos gráficos tridimensionales previamente definidos.

```
(%i49) wxdraw3d(
       color = red,
       parametric(t, sin(t), t/8, t, 0, 2 * %pi),
       color = blue,
       head angle = 10,
       unit vectors = true,
       T[1], T[2], T[3], T[4], T[5], T[6], T[7], T[8], T[9],
       color = green,
       point type = 7,
       points(P)
       );
```

(%t49)



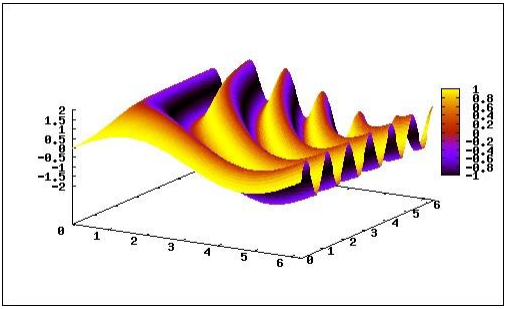
(%o49) [gr3d(parametric, vector, vector, vector, vector, vector, vector, vector, vector, vector, points)]

Maxima

Esta es la gráfica de (%t35) después de cambiar algunas opciones.

```
( %i50)  wxdraw3d( xu grid = 150, yv grid = 150,
           enhanced3d = cos(x * y),
         explicit(sin(x) + sin(x * y), x, 0, 2 * %pi, y, 0, 2 * %pi)
           );
```

(%t50)



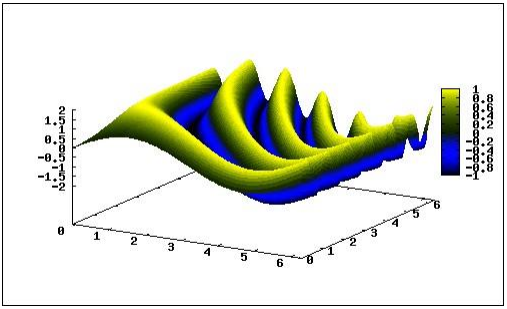
```
( %o50)  [gr3d(explicit)]
```

Maxima

Esta es la gráfica de (%t35) después de más cambios de opciones.

```
( %i51)  wxdraw3d( xu grid = 150, yv grid = 150,
           palette = [6, 5, 15], enhanced3d = sin(x * y),
         explicit(sin(x) + sin(x * y), x, 0, 2 * %pi, y, 0, 2 * %pi)
           );
```

(%t51)



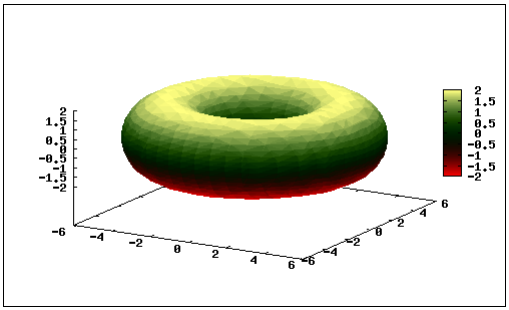
```
( %o51)  [gr3d(explicit)]
```

Maxima

He aquí la superficie obtenida en (%t36) .

```
( %i52)  wxdraw3d(
         x voxel = 17,
         y voxel = 17,
         z voxel =15,
         palette = [12, 5, 27],
         enhanced3d = true,
         implicit((sqrt(x^2 + y^2) − 4)^2 + z^2 = 4, x, −6, 6,
           y, −6, 6, z, −2, 2)
         );
```

(%t52)



(%o52) [gr3d(implicit)]

15.5 Fijación de valores para opciones

En algunas ocasiones se requiere dibujar varios gráficos con las mismas opciones y para evitar escribirlas en cada escena puede optarse por fijar, inicialmente, los valores deseados para dichas opciones. Para hacer factible esto `draw` cuenta con una función específica.

`set draw defaults(` **opción gráfica,. . . ,** **opción gráfica,. . .)**	fija los valores para las opciones gráficas del usuario (llamando a la función sin argumentos se borran las opciones fijadas por el usuario)

Fijando valores para opciones.

Maxima

Aquí se fijan los valores de algunas opciones.

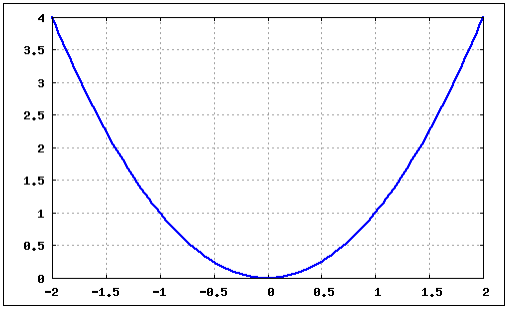
```
(%i53) set draw defaults(
surface hide = true,
color = blue,
grid = true,
line width = 2
);
```

Maxima

A continuación se grafica una parábola sin especificaciones para ninguna opción.

```
(%i54) wxdraw2d(
explicit(x^2, x, 2, 2)
);
```

(%t54)



(%o54) [gr2d(explicit)]

15.6 Gráficos múltiples

Opción	Val. por defecto	Descripción
columns	1	indica el número de columnas a considerar cuando se realizan gráficos múltiples

Opción para gráficos múltiples.

Maxima

Esto define dos escenas una bidimensional y otra tridmensional.

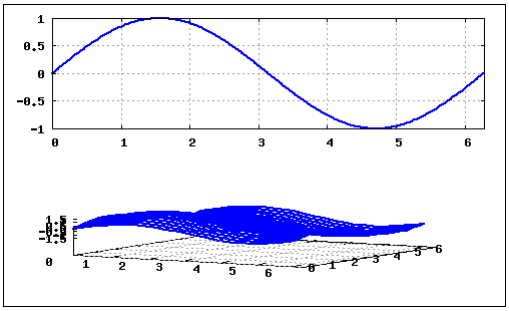
```
( %i55)  f1 : gr2d(explicit(sin(x), x, 0, 2 * %pi)) $
( %i56)  f2 : gr3d(explicit(sin(x) + sin(y), x, 0, 2 * %pi,
           y, 0, 2 * %pi)) $
```

Maxima

Con `draw` la dos escenas previas se presentan por defecto en una sola columna.

```
( %i57)  wxdraw(f1,f2)$
```

(%t57)

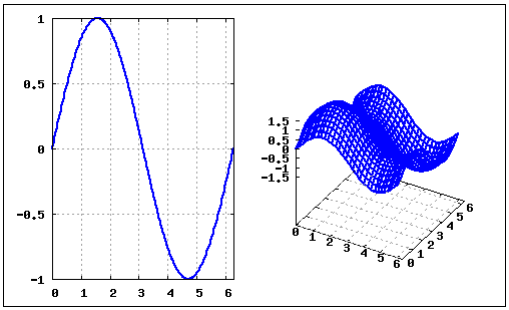


Maxima

Con la opción `columns` es posble cambiar el número de columnas.

```
( %i58)  wxdraw(f1,f2,columns=2)$
```

(%t58)



Maxima

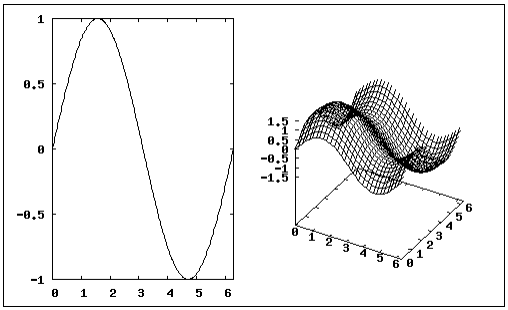
De esta manera se borran las opciones que fueron fijadas en (%i51) .

```
( %i59)  set_draw_defaults() $
```

Maxima

El aspecto de las gráficas luce ahora diferente, pues se han restituido los valores por defecto de las opciones.

```
( %i60)  wxdraw(f1,f2,columns=2)$
( %t60)
```



15.7 Gráficos animados

Del mismo modo que en la sección 14.8 se pueden crear animaciones únicamente en el entorno de wxMaxima. En este caso se utilizan las funciones `with_slider_draw` y `with_slider_draw d` con las cuales se generan gráficos idénticos a los que se generan con `wxplot2d` y `wxplot d`, respectivamente, sin embargo tales gráficos pueden ser animados después de seleccionarlos y pulsar, luego, el botón Comenzar animación, del cuadro de controles, de la barra de herramientas.

Maxima

Cuadro de controles, ubicado en la barra de herramientas, para la animación de gráficos.

`with slider draw`	función para animar gráficos de `wxdraw2d`
`with slider draw d`	función para animar gráficos de `wxdraw d`

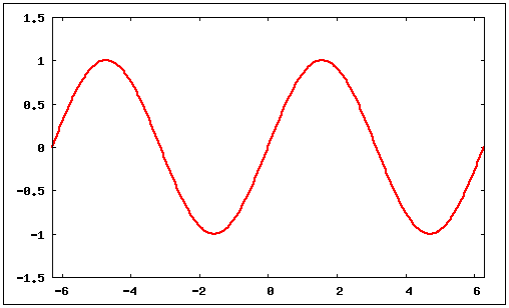
Obtención de gráficos animados con `draw`.

Maxima

Esto genera un gráfico listo para ser animado. Para conseguir la animación primero se selecciona el gráfico y luego se pulsa el botón ▷, del cuadro de controles, y automáticamente ésta es generada. Para detenerla basta pulsar el botón □, del mismo cuadro. También es posible navegar a través de cada cuadro de la animación con tan sólo arrastrar a voluntad el botón, después de haber seleccionado el gráfico.

```
(%i61) with_slider_draw(
       a, [0, 1, 2, 3, 4, 5],
       color = red, line_width = 2, yrange = [-1.5, 1.5],
       explicit(sin(x + a), x, -2 * %pi, 2 * %pi)
       );
```

(%t61)



(%o61)

Maxima

A continuación se define la función infija "/*" para calcular el producto vectorial.

```
(%i62) infix("/*")$
(%i63) "/*"(a, b) := [a[2] * b[3] - b[2] * a[3],
         b[1] * a[3] - a[1] * b[3], a[1] * b[2] - b[1] * a[2]]$
```

Maxima

Esto define la curva a, su primera derivada y su segunda derivada.

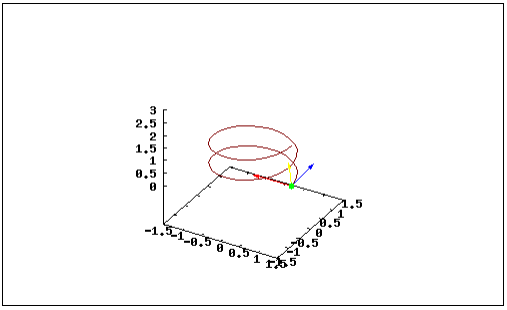
```
(%i64) a(t) := [cos(t), sin(t), t/8]$
(%i65) define("a'"(t), diff(a(t), t))$
(%i66) define("a''"(t), diff(a(t), t, 2))$
```

Maxima

Al animar la siguiente secuencia de cuadros se visualiza el tiedro de Frenet recorriendo la curva a.

```
(%i67) with_slider_draw3d(
       t, create_list(i * %pi/4, i, 0, 16),
       color = dark_red, nticks = 50,
       parametric(cos(u), sin(u), u/8, u, 0, 4 * %pi),
       unit_vectors = true,
       color = blue,
       vector(a(t), "a'"(t)),
       color = yellow,
       vector(a(t), "a'"(t)/ * "a''"(t)),
       color = red,
       vector(a(t), ("a'"(t)/ * "a''"(t))/ * "a'"(t)),
       color = green, point_type = 7,
       points([a(t)]),
       xrange = [-1,5, 1,5], yrange = [-1,5, 1,5],
       zrange = [0, 3],
       user_preamble = "set size ratio 1");
```

(%t67)



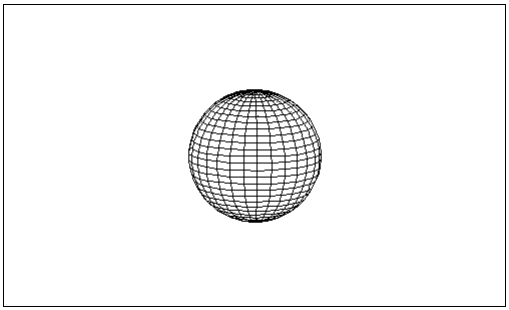
(%o67)

Maxima

Esta es la animación de un morfismo entre la esfera y el toro.

```
(%i68) with_slider_draw3d(
       a, create_list(i/8, i, 0, 8),
       surface_hide= true,
       user_preamble = "set size ratio 1",
       xtics = false, ytics = false, ztics = false,
       axis_3d = false,
       rot_vertical = 80, rot_horizontal = 100,
       parametric_surface(
          (1 - a) * cos(t) * sin(u/2) - (a * sin(u))/3,
          (a * sin(t) * (2 - cos(u)))/3+
             (1 - a) * sin(t) * sin(u/2),
          (a * cos(t) * (2 - cos(u)))/3 + (1 - a) * cos(u/2),
          t, 0, 2 * %pi, u, 0, 2 * %pi)
       );
```

(%t68)



(%o68)

Capítulo 16

Archivos y operaciones externas

16.1 Generación de expresiones y archivos TEX

Si el usuario quiere combinar su trabajo con material existente en TEX, puede encontrar conveniente usar la función `tex` para convertir expresiones de Maxima en forma conveniente de entrada para TEX. El resultado que así se obtiene es un fragmento de código que puede incluirse en un documento mayor, pero que no puede ser procesado aisladamente.

`tex`(expr)	imprime en la consola la representación en TEX de expr
`tex`(expr, `false`)	devuelven el código TEX en formato de cadena
`tex`(expr)destino)	añade la salida al archivo destino

Salidas de Maxima para TEX

Maxima

He aquí una expresión, impresa en forma estándar de Maxima.

```
(%i1)  (x+y)^2/sqrt(x*y);
                                 (y + x)^2
(%o1)                            ---------
                                  sqrt(x y)
```

Maxima

He aquí la expresión previa en forma de entrada para TEX.

```
(%i2)  tex(%);
$${{\left(y+x\right)^2}\over{\sqrt{x\,y}}}$$
(%o2)  false
```

Maxima

Esto añade la expresión $\frac{(y+x)^2}{\sqrt{x\,y}}$, traducida a TEX, al archivo `ejemplo.tex` ubicado en `d:/maximatex/`.

```
(%i3)  tex((x + y)^2/sqrt(x * y),
       "d : /maximatex/ejemplo.tex");
(%o3)  false
```

`texput(a)f )`	establece el formato, en TEX, del átomo a, el cual puede ser un símbolo o el nombre de un operador
`get tex environment(op)`	devuelve el entorno TEX que se aplica al operador op. Si no se ha asignado ningún entorno, devolverá el que tenga por defecto
`set tex environment(op)antes) después)`	asigna el entorno TEX al operador op
`get tex environment default()`	devuelve el entorno TEXque se aplica a expresiones para las cuales el operador de mayor rango no tiene entorno TEXasignado
`set tex environment default(antes) después)`	asigna el entorno TEXpor defecto

Salidas de Maxima para TEX

Maxima

De esta forma se asigna código TEXpara una variable.

```
(%i4)  texput(s, "\\sqrt{2}");
(%o4)  \sqrt{2}
```

Maxima

Ahora puede usarse la asignación anterior para generar más código TEX.

```
(%i5)  tex(s + 1/2);
$$\sqrt{2}+{{1}\over{2}}$$
(%o5)  false
```

Maxima

El entorno TEX aplicado, por defecto, a expresiones, en Maxima, es $$ $$.

```
(%i6)  tex(3/4);
$${{3}\over{4}}$$
(%o6)  false
```

Maxima

Con la función set_tex_environment_default es posible cambiar el entorno TEX. En este caso se ha anulado todo entorno.

```
(%i7)  set_tex_environment_default(" ", " ");
(%o7)
```

Maxima

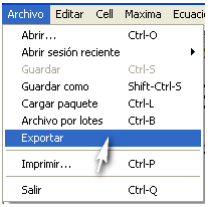
He aquí el resultado del nuevo código TEX.

```
(%i8)  tex(3/4);
{{3}\over{4}}
(%o8)  false
```

Además de traducir expresiones individuales a TEX, Maxima también traduce cuadernos completos a documentos pdf LATEX. Para ello el usuario debe digitar el nombre que asignará al archivo resultante, así como la respectiva extensión, `tex`, en la casilla Nombre de la ventana Exportar que aparece después de elegir la opción Exportar del menú Archivo.

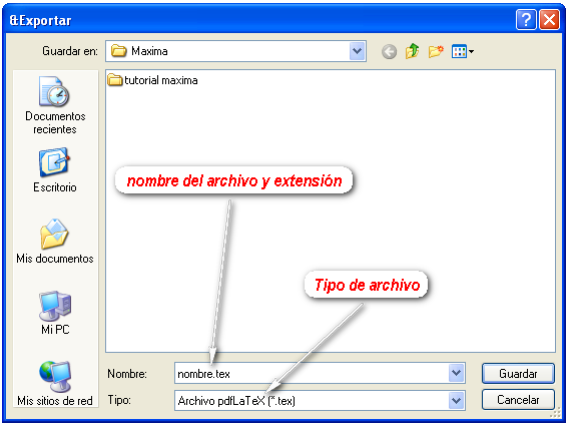
Maxima

Primer paso para exportar un cuaderno.



Maxima

Exportando un cuaderno como documento pdf LATEX



16.2 Generación de archivos HTML

Adicionalmente, Maxima brinda capacidades para convertir cuadernos completos a páginas web. El proceso que debe seguirse para la traducción es el mismo que se siguió en la página 227, pero en este caso la extensión será `html`.

16.3 Generación de expresiones Lisp y Fortran

Si el usuario tiene programas escritos en Lisp o Fortran, puede ser que quiera tomar fórmulas que ha generado en Maxima e insertarlas en el código original de sus programas. Maxima permite convertir expresiones matemáticas en expresiones Lisp y Fortran.

`:lisp $%in`	escribe para Lisp la expresión a la que hace referencia la etiqueta `%in`
`fortran`(expr)	escribe expr para Fortran

Salidas de Maxima para Lisp y Fortran.

Maxima

Aquí se obtienen expresiones para Lisp y Fortran.

```
(%i9)  x^2 + 2*x + 1;
(%o9)  x^2 + 2 x + 1
(%i10)  :lisp $%i7;
((MPLUS) ((MEXPT) $X 2) ((MTIMES) 2 $X) 1)
(%i11)  fortran(x^2 + 2 * x + 1);
      x**2 + 2*x + 1
(%o11)  done
```

Capítulo 17

Programación con Maxima

La elaboración de programas con el lenguaje de programación de Maxima permite al usuario definir sus propias funciones. De esta manera se hace posible la automatización de secuencias de operaciones que son útiles para abordar la solución de un determinado tipo de problema.

Además, es posible implementar varias funciones, relacionadas con cierto tema, y guardarlas en un solo archivo que luego se pueda ejecutar sin necesidad de visualizar todo el código elaborado. A tal archivo se le conoce como **paquete**[1] de funciones.

17.1 Operadores relacionales y lógicos

De manera similar que cualquier lenguaje de programación Maxima incluye operadores relacionales y lógicos, así como estructuras de control (que se utilizan para controlar el flujo del programa en una rutina).

En Maxima, los operadores lógicos pueden ser **infijos** o **prefijos**. Un operador recibe el nombre de infijo cuando éste debe escribirse entre los operandos, por ejemplo el operador `and` cuya sintaxis es p `and` q, para ciertos operandos p y q. Por otra parte, un operador prefijo es

[1] Del inglés package. Un paquete es almacenado en forma automática por Maxima como archivo de extensión `lisp` y el código es convertido internamente al lenguaje de programación `Lisp`.

aquel que debe escribirse antes del operando, por ejemplo el operador `not` cuya sintaxis es `not` *p*, para cierto operando *p*.

Cabe destacar que, en Maxima, casi todos los operadores lógicos son infijos y únicamente hay un operador lógico prefijo.

Operador	Símbolo	Tipo
menor que	<	operador relacional infijo
menor o igual que	<=	operador relacional infijo
igualdad (sintáctica)	=	operador relacional infijo
negación de =	#	operador relacional infijo
igualdad (por valor)	`equal`	operador relacional infijo
negación de `equal`	`notequal`	operador relacional infijo
mayor o igual que	>=	operador relacional infijo
mayor que	>	operador relacional infijo
y	`and`	operador lógico infijo
o	`or`	operador lógico infijo
no	`not`	operador lógico prefijo

Operadores relacionales y lógicos.

Controlador	Descripción
`if`	permite, mediante una condición, que se ejecute o no se ejecute determinada tarea o línea de código
`for`	es utilizado para generar una repetición de instrucciones entre un número inicial y un número final que deben ser indicados o, también, entre un conjunto de elementos de una lista
`while`	repetirá sin detenerse un determinado código mientras se cumpla una condición
`unless`	repetirá sin detenerse un determinado código hasta que se cumpla una condición
`do`	se utiliza para realizar iteraciones con `for`, `while` y `unless`

Principales estructuras de control.

Maxima

He aquí un ejemplo sencillo en el que se muestra la sintaxis de `if`.

```
(%i1)  if 2 = 3 then 1;
(%o1)  false
(%i2)  if 3 = 3 then 1;
(%o2)  1
```

Maxima

En este ejemplo se añade el resultado ha obtener en caso de que la condición sea falsa.

```
(%i3)  if 2 = 3 then 1 else 3;
(%o3)  3
```

Téngase en cuenta que las expresiones consideradas en los ejemplos de las entradas (%i1) y (%i2) equivalen a las expresiones:
if 2 = 3 **then** 1 **else false** y **if** 3 = 3 **then** 1 **else false**, respectivamente.

Maxima

Aquí se obtiene el valor ligado a la expresión verdadera más próxima (en este caso 3 = 3). Si todas las expresiones fuesen falsas, entonces se obtendría el último valor (en este caso 4).

```
(%i4)  if 2 = 3 then 1 elseif 3 = 3 then 5 else 4;
(%o4)  5
```

Maxima

Mediante este otro ejemplo se muestra la sintaxis de `for`.

```
(%i5)  for i : 1 thru 5 step 2 do print(i);
1

5
(%o5)  done
```

Maxima

Si no se indica el paso (incremento) se asume uno, por defecto.

```
(%i6)  for i : 1 thru 5 do print(i);
1
2

(%o6)  done
```

Maxima

Para inicializar el contador en `while` se utiliza `for`.

```
(%i7)  for i : 1 while i <= 3 do print(i);
1
2

(%o7)  done
```

Maxima

Para inicializar el contador en `unless` también se utiliza `for`.

```
(%i8)  for i : 1 unless i >= 4 do print(i);
1
2

(%o8)  done
```

17.2 Operadores y argumentos

Casi todo en Maxima es un objeto de la forma

$$\mathrm{f'un}(a_1, \ldots, a_n),$$

es decir, una expresión que comprende un operador como f'un y los argumentos de éste, $a_1, \ldots, a_n$. Las funciones `op` y `args` permiten averiguar la estructura de las expresiones.

`op`(*expr*)	permite obtener el operador de la expresión *expr*
`args`(*expr*)	permite obtener una lista cuyos elementos son los argumentos de la expresión *expr*

Obtención del operador y de los argumentos de una expresión.

Maxima

Aquí se define la expresión $a + b$, la cual es almacenada en la variable `expr`.

```
(%i9)  expr:a+b;
(%o9)  b+a
```

Maxima

A continuación, con la primera operación se obtiene el operador de la expresión previamente definida; y en la segunda, se obtiene una lista con los argumentos de dicha expresión.

```
(%i10)  op(expr);
(%o10)  +
(%i11)  args(expr);
(%o11)  [b,a]
```

Maxima

Es posible "reconstruir" la expresión usando la función `apply`.

```
(%i12)  apply(op(expr),args(expr));
(%o12)  b+a
```

Maxima

Esto define otra expresión.

```
(%i13)  expr:ejemplo(x,y,z);
(%o13)  ejemplo(x,y,z)
```

Maxima

Aquí, nuevamente, se obtienen el operador y los argumentos de la expresión.

```
(%i14) op(expr);
(%o14) ejemplo
(%i15) args(expr);
(%o15) [x, y, z]
```

Maxima

También es posible "reconstruir" la última expresión usando la función `apply`.

```
(%i16) apply(op(expr), args(expr));
(%o16) ejemplo(x, y,z)
```

Maxima

Algunas expresiones (`plot2d`, `integrate`, etc.) requieren de una comilla simple.

```
(%i17) expr : ' plot2d(x^2, [x, -1, 1]);
(%o17) plot2d(x^2, [x, -1, 1])
```

Maxima

Ahora si es posible obtener el operador y los argumentos de la expresión.

```
(%i18) op(expr);
(%o18) plot2d
(%i19) args(expr);
(%o19) [x^2, [x, -1, 1]]
```

Maxima

También las listas y los conjuntos son objetos de la forma **fun**$(a_1, \ldots, a_n)$. Aquí se almacena una lista cualquiera en la variable `expr`.

```
(%i20) expr : [a, b, c, d] ;
(%o20) [a, b, c, d]
```

Maxima

Del mismo modo que en los ejemplos anteriores se obtienen el operador y una lista con los argumentos.

```
(%i21)  op(expr);
(%o21)  [
(%i22)  args(expr);
(%o22)  [a, b, c, d]
```

17.3 Programación funcional

La programación funcional es la programación que pone énfasis en el uso de funciones. Maxima incluye las funciones predefinidas `appl` , `map`, `lambda` que permiten apreciar la potencia de la programación funcional.

`appl` (f, [$expr_1, \ldots,$ $expr_n$])	construye y evalúa la expresión $f(arg_1, \ldots, arg_n)$
`map`(f, $expr_1, \ldots, expr_n$)	devuelve una expresión cuyo operador principal es el mismo que aparece en las expresiones $expr_1, \ldots, expr_n$ pero cuyas subpartes son los resultados de aplicar f a cada una de las subpartes de las expresiones
`lambda`([$x_1, \ldots, x_m$] , $expr_1, \ldots, expr_n$)	define y devuelve una expresión lambda (es decir, una función anónima) con argumentos $x_1, \ldots, x_m$; la cual devuelve el valor $expr_n$
`lambda`([[L]] , $expr_1, \ldots, expr_n$)	define y devuelve una expresión lambda con argumento opcional L; la cual devuelve el valor $expr_n$
`lambda`([$x_1, \ldots, x_m$, [L]] , $expr_1, \ldots, expr_n$)	define y devuelve una expresión lambda con argumentos $x_1, \ldots, x_m$, argumento opcional L; la cual devuelve el valor $expr_n$

Funciones predefinidas para programación funcional.

Maxima

Aquí se define la función *G*, la cual es aplicada luego a una lista cuyos elementos pasan a ser los argumentos de *G*.

```
(%i23) G(x,y,z) := x^2 + y^2 + z^2 $
(%i24) apply(G, [x - y, a + b, u]);
(%o24) (x - y)^2 + u^2 + (b + a)^2
```

Maxima

En este caso se aplica la función predefinida `min`.

```
(%i25) apply(min, [7, 9, 3, 4]);
(%o25) 3
```

Maxima

Esto define la función *F* y luego la mapea en una lista.

```
(%i26) F(x) := x^3 - 1 $
(%i27) map(F, [2, 3, 5, a]);
(%o27) [7, 26, 124, a^3 - 1]
```

Maxima

Aquí se muestra la definición de una función lambda *f*, la misma que posee dos argumentos.

```
(%i28) f: lambda([x,y], x+y) $
(%i29) f(a, b);
(%o29) b + a
```

Maxima

Ahora se define una función lambda con argumento opcional.

```
(%i30) f: lambda([ [x] ], x^2) $
(%i31) f(p);
(%o31) [p^2]
```

```
(%i32)  f(p, q, r, s, t);
(%o32)  [p², q², r², s², t²]
```

Maxima

En este ejemplo se define una función lambda con dos argumentos y un argumento opcional. Luego esta función es evaluada en tres argumentos y se obtiene una lista con el resultado esperado, no obstante al evaluar la función en más argumentos se obtiene una lista con tantos elementos como argumentos adicionales hay.

```
(%i33)  f : lambda([ x, y, [z] ], x* z + y) $
(%i34)  f(p, q, r);
(%o34)  [p r + q]
(%i35)  f(p, q, r, s, t, u, v, w);
(%o35)  [p r + q, p s + q, p t + q, p u + q, p v + q, p w + q]
```

17.4 Implementación del paquete: `ejemplo`

En esta sección se implementará el paquete `ejemplo`, que incorporará las funciones `triangulo` y `circunferencia`.

Maxima

Aquí se define la función `triangulo`. Esta función permite calcular el área de un triángulo y el baricentro de un conjunto de puntos de R^2 , dados. Además, si los puntos son colineales, devuelve el mensaje Los p'Untos son colineales.

```
(%i36)  triangulo(pts) :=
        block([f : lambda([h], endcons(1, h)), pts1, M, d, a],
          pts1 : map(f, pts),
          M : apply(matrix, pts1),
          d : determinant(M),
          a : abs(d/2),
          if d = 0 then string("Los puntos son colineales")
            else
              (b : apply("  +  ", pts)/3,
               [sconcat(Area, " : ", a, "  ", u^2),
               sconcat(Baricentro, " : ", b)])
        ) $
```

Maxima

Dado el conjunto de puntos del plano $\{(1, 2), (3, -1), (2, 3)\}$, no colineales, la función `triangulo` devuelve el área y las coordenadas del baricentro del triángulo definido por los puntos dados.

```
(%i37)  p : [ [1, 2], [3, -1], [2, 3] ] $
(%i38)  triangulo(p);
(%o38)  [ Area :  5/2 u^2,  Baricentro :  [2, 4/3] ]
```

Maxima

Para visualizar los gráficos se utilizará el paquete `draw`.

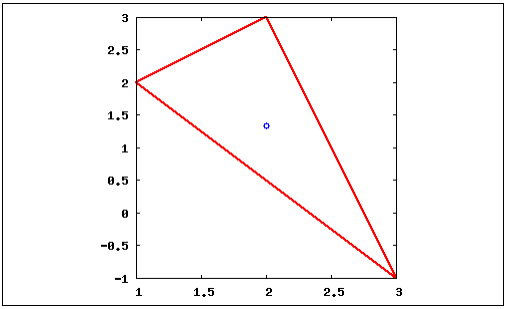
```
(%i39)  load(draw) $
```

Maxima

Esto muestra la gráfica del triángulo previamente analizado, conjuntamente con el punto que corresponde al baricentro del mismo.

```
(%i40)  wxdraw2d(
        color = red, fill color = white, line width = 2,
        polygon(p),
        point type = 6, color = blue,
        points([ [2, 4/3] ]),
        user preamble = "set size ratio 1"
        );
```

(%t40)




```
(%o40)  gr2d(pol gon, points)
```

Maxima

Puesto que, en este caso, los puntos del conjunto {(1, 1), (2, 2), (3, 3)} son colineales, la función `triangulo` devuelve un mensaje indicando éste hecho.

```
(%i41)  p : [ [1, 1], [2, 2], [3, 3] ] $
(%i42)  triangulo(p);
(%o42)  "Los puntos son colineales"
```

Maxima

Aquí se define la función `circunferencia` que permite obtener la ecuación de la circunferencia definida por tres puntos dados.

```
(%i43)  circunferencia(pts) :=
        block([f : lambda([h], endcons(1, h)),
               g : lambda([h], cons(h[1]^2 + h[2]^2, h)),
               pts1, M, d, eq],
          pts1 : map(f, pts),
          M : apply(matrix, pts1),
          d : determinant(M),
          if d = 0 then string("Los puntos son colineales")
           else
             (aux:map(g,cons([x,y,1],pts1)),
              M : apply(matrix,aux),
              d : determinant(M),
              eq : expand(d),
              expand(eq/coeff(eq, x^2)) = 0)
        ) $
```

Maxima

Dado el conjunto de puntos del plano {(1, 2), (3, −1), (2, 3)}, no colineales, la función `circunferencia` devuelve la ecuación de la circunferencia definida por estos puntos.

```
(%i44)  p : [ [1, 2], [3, -1], [2, 3] ] $
(%i45)  circunferencia(p);
```

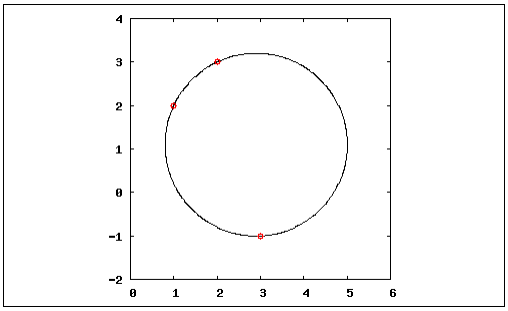
$$(\%o45)\quad y^2 - \frac{11\,y}{5} + x^2 - \frac{29\,x}{5} + \frac{26}{5} = 0$$

Maxima

Esto muestra la gráfica de la circunferencia previamente analizada, conjunatmente con los puntos que la definen.

```
( %i46) wxdraw2d(
        implicit(circunferencia(p),x,0,6,y,-2,4),
        point_type = 6,color = red,point_size = 1,
        points(p),
        user_preamble = "set size ratio 1" );
```

(%t46)




```
(%o46) gr2d(pol_gon,points)
```

Maxima

Igual que con la función `triangulo`, para el conjunto de puntos del plano {(1, 1), (2, 2), (3, 3)}, la función `circunferencia` devuelve un mensaje indicando que éstos son colineales.

```
( %i47) p : [ [1, 1], [2, 2], [3, 3] ] $
( %i48) circunferencia(p);
( %o48) "Los puntos son colineales"
```

Hasta aquí se han definido, y se ha verificado el correcto funcionamiento de, las funciones `triangulo` y `circunferencia`. Seguidamente se guardará la definición de estas funciones en un fichero de nombre ejemplo y de extensión lisp, el cual constituirá un ejemplo de un paquete de funciones definido por el usuario.

Naturalmente, si el usuario desea puede copiar solamente la definición de las citadas funciones y luego guardarlas sin necesidad de

copiar y ejecutar los ejemplos.

Maxima

Esto guarda todas las funciones definidas por el usuario en el archivo ejemplo.lisp. El directorio donde se guardará el fichero es indicado por el usario, en este caso éste es d:/maximapackages.

```
(%i49)  save("d:/maximapackages/ejemplo.lisp",
        functions)$
```

Una vez que se ha guardado la definición de las funciones en el mencionado fichero es posible inicializarlo como si se tratara de cualquier paquete incorporado en Maxima.

En este caso se asume que el usuario ha cerrado el cuaderno de trabajo actual de Maxima y luego ha abierto un nuevo cuaderno desde el cual inicializará el paquete ejemplo.lisp.

Maxima

Esto inicializa el paquete ejemplo.lisp desde el directorio en el que se guardó.

```
(%i1)  load("d:/maximapackages/ejemplo.lisp")$
```

Maxima

Ahora es posible ejecutar cualquiera de las funciones incorporadas en el paquete ejemplo.lisp sin necesidad de exponer el código del mismo.

```
(%i2)  p:[[0,0],[2,0],[1,sqrt(3)]]$
(%i3)  triangulo(p);
(%o3)  [Area: sqrt(3) u^2, Baricentro: [1,1/sqrt(3)]]
```

Bibliografía

1J Fokker, J. PROGRAMACIÓN FUNCIONAL. http://people.cs.uu.nl/jeroen/courses/fpsp.pdf (1996).

2J Rodríguez, J. R. MAXIMA CON WXMAXIMA: SOFTWARE LIBRE EN EL AULA DE MATEMATICAS. http://knuth.uca.es/repos/maxima (2007).

3J Rodríguez, M. y Villate, J. MANUAL DE MAXIMA ver. 5.18. http://maxima.sourceforge.net/es/documentation.html (2009).

4J Rodríguez, M. PRIMEROS PASOS EN MAXIMA. www.telefonica.net/web2/biomates/maxima/max.pdf (2008).

5J Rodríguez, M. SOFTWARE MATEMATICO BASICO: MAXIMA. www.telefonica.net/web2/biomates/maxima/i-math.pdf (2008).

6J Rodríguez, M. MAXIMA: UNA HERRAMIENTA DE CALCULO. http://softwarelibre.uca.es/cursos/maxima/cadiz.pdf (2006).