

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/303913874>

Maxima con wxMaxima: software libre en el aula de matemáticas

Book · January 2007

CITATIONS

5

READS

2,519

1 author:



[J. Rafael Rodríguez Galván](#)
Universidad de Cádiz

38 PUBLICATIONS 136 CITATIONS

SEE PROFILE

Maxima con *wxMaxima*: software libre en el aula de matemáticas

J. Rafael Rodríguez Galván

Departamento de Matemáticas de la Universidad de Cádiz

Oficina de Software Libre de la Universidad de Cádiz

Febrero de 2007

Versión 0.1

Copyright © 2007, J. Rafael Rodríguez Galván.

Este documento es libre. Se otorga permiso para copiarlo, distribuirlo y/o modificarlo bajo los términos de la licencia FDL (GNU Free Documentation License) versión 1.2 o posterior, publicada por la Fundación de Software Libre¹. No contiene secciones invariantes, texto de portada ni de respaldo.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Las fuentes L^AT_EX pueden conseguirse en <https://forja.rediris.es/projects/guia-wxmaxima/>

Los contenidos de la sección 1.3 están directamente basados en el manual “Primeros pasos en Maxima”, de Mario Rodríguez Riotorto. Copyright © 2006, Mario Rodríguez Riotorto. En los apéndices A y B se ha utilizado el manual “Introducción a Maxima” de Victoria Redondo Neble y Rafael Rodríguez Galván.

¹<http://www.gnu.org/licenses/fdl.html>

Índice general

1. Generalidades	9
1.1. Introducción al software libre	9
1.1.1. Comunidades científicas de conocimiento y el origen del software libre	10
1.1.2. El software libre: definición y significado	11
1.1.3. El sistema operativo GNU/Linux	13
1.2. El software libre y las matemáticas en el aula	15
1.2.1. Software libre y educación	15
1.2.2. Software matemático	16
1.3. Breve historia de <i>Maxima</i>	19
2. Primer contacto con <i>Maxima</i> y <i>wxMaxima</i>	21
2.1. Instalación	21
2.1.1. <i>Maxima</i>	21
2.1.2. <i>wxMaxima</i>	22
2.2. Entornos de ejecución: la consola <i>Maxima</i> y <i>wxMaxima</i>	23
2.3. Primeros pasos y exploración del entorno <i>wxMaxima</i>	27
2.3.1. Abrir <i>Maxima</i> en modo consola	27
2.3.2. Primer contacto con <i>wxMaxima</i>	28
2.3.3. Sacar partido a las posibilidades de <i>wxMaxima</i>	32
2.3.4. Un ejemplo algo más avanzado	37
2.3.5. Una panorámica de <i>wxMaxima</i>	42
3. Aritmética	47
3.1. Trabajando con distintos tipos de números	47
3.1.1. Números enteros	47
3.1.2. Números racionales	48
3.1.3. Números reales	49
3.2. Potencias, radicales y logaritmos. Otras funciones predefinidas	53
3.3. Divisibilidad	58
3.4. Actividades	58

4. Álgebra	61
4.1. Listas	61
4.2. Polinomios y fracciones algebraicas	64
4.3. Ecuaciones, sistemas de ecuaciones e inecuaciones	69
4.4. Matrices y determinantes	73
5. Cálculo Infinitesimal	81
5.1. Sucesiones y límite	81
5.2. Funciones	85
5.3. Representación de gráficas	89
5.3.1. Gráficas de funciones de una variable (en coordenadas cartesianas)	89
5.3.2. Gráficas en coordenadas polares y paramétricas	93
5.3.3. Gráficas de curvas poligonales	95
5.3.4. Otras cuestiones relacionadas con los gráficos	96
5.4. Derivadas, integrales y aplicaciones	98
6. Programación y adaptación de <i>Maxima</i>	103
6.1. Almacenando sesiones y datos	103
6.2. <i>Maxima</i> como lenguaje de programación	109
6.2.1. Bucles	111
6.2.2. Funciones	113
A. Problema resuelto 1: estudio y representación de una función	117
B. Problema resuelto 2: estudio de una función definida a trozos	125
C. Creación de un módulo: “inecuaciones”	131
D. Un ejemplo: geometría de tortuga	137
Referencias	143

Índice de figuras

2.1.	<i>Maxima</i> ejecutándose en entorno consola	23
2.2.	<i>wxMaxima</i> como interfaz de <i>Maxima</i>	24
2.3.	<i>Maxima</i> ejecutándose bajo la interfaz <i>xmaxima</i>	25
2.4.	<i>Maxima</i> ejecutándose en una sesión <i>TEXmacs</i>	26
2.5.	Un fichero maxima (<i>buffer</i> izquierdo) y una sesión imaxima (<i>buffer</i> derecho). Ambos, interactuando en <i>Emacs</i>	26
2.6.	Distintas áreas de <i>wxMaxima</i>	28
2.7.	Utilización del área de entrada	29
2.8.	Resultados obtenidos en el área de salida	29
2.9.	Ventana de diálogo <i>Resolver</i>	36
2.10.	Ventana de diálogo <i>Limite</i>	39
2.11.	Ventana de diálogo <i>Derivar</i>	40
2.12.	Gráfica de la función $f(x) = \frac{\sqrt{x}}{1+x^2}$	42
4.1.	Introducción del contenido de una matriz 3×3 en <i>wxMaxima</i>	74
5.1.	Algunas funciones predefinidas en <i>Maxima</i>	85
5.2.	Ventana de diálogo de Gráficos 2D en <i>wxMaxima</i>	89
5.3.	Gráficas de $\cos(x^2)$ y del par $[x^2, \sqrt{x}]$	90
5.4.	Gráfica de x^2 con las restricciones $x \in [0, 3]$, $y \in [0, 4]$	90
5.5.	Gráfica de x^2 , ampliada a todo $x \in [0, 3]$, $y \in [0, 4]$	91
5.6.	Algunas gráficas en coordenadas cartesianas	92
5.7.	Gráficas polares	94
5.8.	Curva paramétrica $(\cos(t), \sin(t))$, $t \in [0, 2\pi]$	94
5.9.	Poligonales (opción “discrete” de <i>Maxima</i>)	95
5.10.	Ficheros gráficos postscript generados por <i>Maxima</i>	97
5.11.	$f(x) = x^4 - x$	101
6.1.	Personalización de gráficos	109
A.1.	A. oblicua	120
A.2.	R. tangente	122
B.1.	Gráfica de $f(x)$	130

D.1. Una curva generada por la ruta de la “tortuga virtual”	138
D.2. Algunos resultados de <i>poligono(lado,angulo)</i>	139
D.3. Curvas de Koch para distintos niveles	141
D.4. Un copo de nieve realizado como unión de tres curvas de Koch	141

Prólogo

El presente manual ha sido elaborado expresamente para el curso de formación al profesorado de los C.E.P. de Cádiz, Villamartín y Jerez que tendrá lugar en Febrero de 2007. La versión actual contiene una amplia guía de introducción a los programas *Maxima* y *wxMaxima* (situados en el contexto del software libre y las matemáticas) que deberá ser suficiente para utilizar con un cierto grado de pericia estas herramientas. A medida que avance el curso, se añadirán sucesivos capítulos que tendrán un carácter más específico, desarrollando completamente el programa de estos cursos. Los apuntes estarán estarán disponibles en versión PDF a través del campus virtual del curso y en versión L^AT_EX en <http://knuth.uca.es/repos/maxima>.

1

Generalidades

Capítulo

1.1. Introducción al software libre

La mayor parte de los usuarios de informática están habituados a utilizar programas de ordenador con licencias privativas, sin ser conscientes de lo que esto significa. Es para ellos algo usual que el proceso de instalación de los programas incluya una etapa en la que se muestra la licencia y se pide al usuario que decida si la aceptan. Cosa que deben hacer si desean utilizarlo, aun siendo conscientes de que la aceptación de esta licencia puede significar la privación para realizar acciones que podrían ser tan normales como instalar el programa en un segundo ordenador, prestarlo o regalar una copia a otra persona. Es importante señalar que estas restricciones no son de tipo tecnológico (las nuevas tecnologías hacen que sea trivial la copia y difusión de información) sino que se basan en la elección de una licencia muy restrictiva por parte del autor y en la aceptación de ésta por parte del usuario.

Pero es más: si, teniendo suficientes conocimientos, detectamos un fallo en el programa y deseamos estudiarlo para poder solucionarlo por nosotros mismos, nos volveremos a encontrar con la prohibición legal de hacerlo, impuesta exclusivamente por la licencia que el autor eligió y que nosotros aceptamos. Por supuesto, no se puede trasladar esta situación a otras industrias diferentes a la del software. Los resultados serían paradójicos. Así, en la industria automovilística, la analogía sería la de una persona con suficientes conocimientos, por ejemplo un mecánico de profesión, que adquiere un coche. Años después el coche se para y éste decide abrir el capó para estudiar la avería y poder aplicar sus conocimientos para arreglarlo. Pero, descubre que esto es imposible pues, en el momento de la compra, el mecánico firmó un contrato que le prohíbe abrir el capó o realizar cualquier tipo de modificación técnica.

A este tipo de programas, cuyas licencias privan del derecho a estudiarlo, modificarlo o redistribuirlo, se les conoce como “software propietario” o, con más propiedad, “software privativo”.

Pero existen cada vez más programadores que, en ejercicio de sus legítimos derechos de propiedad intelectual, deciden dotar a su trabajo de una licencia distinta, mucho menos restrictiva, devolviéndole al propietario final sus derechos para copiarlo a cuantas personas desee, estudiarlo hasta el último detalle, y modificarlo sin restricciones para poder mejorarlo o adaptarlo. Los motivos para tomar esta decisión son muy variados: desde el mero altruismo hasta el más duro capitalismo, pasando por otras consideraciones, entre ellas políticas, empresariales o de simple eficacia.

Pero en última instancia, por muy novedoso que pueda parecer este concepto, es siempre conveniente tener en cuenta lo siguiente: cuando hablamos de software libre no

estamos haciendo más que trasladar al campo de la informática el método científico que, en los últimos siglos, ha dado excelentes resultados a la humanidad. En particular, para un matemático debería resultar natural la idea de aplicar a la informática (un área de conocimiento tan cercana) la forma en que las matemáticas se desarrollan y progresan.

1.1.1. Comunidades científicas de conocimiento y el origen del software libre

El modelo de una comunidad de personas que producen e intercambian información, criticándola y mejorándola para obtener, actualizar y publicar conocimiento, no es nuevo y tiene su paradigma en la comunidad científica y en las matemáticas.

En efecto, si bien en algún momento de la historia de las matemáticas éstas han podido inclinarse hacia el oscurantismo, apartándose del camino de la libre publicación y revisión por pares, el transcurrir de los siglos ha derivando inexorablemente hacia esta última senda. En la Europa del siglo XVII, como consecuencia de la revolución científica, se confirmó definitivamente este modelo. Y en el siglo XX, la irrupción de las tecnologías de la información y las comunicaciones, en particular de Internet, ha abierto posibilidades insospechadas para las matemáticas, la ciencia y en general para todas las comunidades basadas en el conocimiento, al facilitar la comunicación y la transmisión de información de forma instantánea a cualquier punto del planeta.

El concepto de software libre, tal y como lo conocemos hoy, se puede entender como el fruto de una concepción científica del conocimiento informático y de la generalización de las nuevas tecnologías y de Internet en las sociedades desarrolladas.

En la primera mitad del siglo XX y hasta la década de 1960, los ordenadores eran máquinas de grandes dimensiones instaladas en centros gubernamentales y grandes corporaciones y muchas de las personas encargadas de manipularlos formaban parte de la comunidad matemática y científica. En esta época, cuando una gran institución, pública o privada, adquiría un ordenador, el software venía como un acompañante y en este sentido se trataba como un todo con el hardware adquirido. Mientras se pagase el contrato de mantenimiento, se tenía acceso al catálogo de software que ofrecía el fabricante [3]. Además, no era común la idea de que los programas fueran algo separado desde un punto de vista comercial. En esta época, el software se solía distribuir junto a su código fuente y en general no había restricciones prácticas para usarlo, compartirlo o modificarlo. Podría decirse que el software, en su origen, fue libre y los informáticos constituían una comunidad científica que se podría comparar a la actual “comunidad de software libre”.

Pero en los años 70, las compañías de hardware comenzaron a vender parte de su software por separado. Con la percepción del software como un valor intrínseco, nació la preocupación por limitar (tanto técnica como legalmente) el acceso a los programas y restringir las posibilidades de los usuarios para compartirlo o modificarlo. Esto desembocó en la situación que todavía hoy es habitual, en la que el software con licencia privativa es mayoritario en el sector de la informática doméstica (aunque no en otros

sectores, como el de los servidores).

Aunque durante esta década siguieron apareciendo productos que hoy consideramos libres (como T_EX) o comunidades que compartían y desarrollaban el software de una forma abierta (por ejemplo, las comunidades que se formaron entorno al sistema operativo UNIX, que fueron semilla de los actuales sistemas libres de tipo BSD), no fue hasta principios de la década de 1980 cuando aparecieron, de forma organizada y consciente, los primeros proyectos para la creación de sistemas compuestos de software libre, y lo que probablemente es más importante: los fundamentos éticos, legales y hasta económicos, que luego se continuarían desarrollando hasta el día de hoy. De esta época procede también el propio término software libre, ambos formulados por primera vez por Richard Stallman a principios de los años 80.

Richard Stallman, empleado en el laboratorio de inteligencia artificial del prestigioso MIT (Instituto de Tecnología de Massachussets), se consideraba un *hacker*, en el sentido de una persona que disfruta ejercitando su inteligencia y compartiendo con sus colegas sus inquietudes tecnológicas y el código de sus programas. Pero en los primeros años 80 veía con desagrado cómo, progresivamente, sus compañeros de trabajo comenzaban a firmar contratos de exclusividad y no compartición de código para la elaboración de software privativo. Su negativa a hacerlo le estaban convirtiendo en un extraño en su propio mundo y, ante con la imparable extensión del software propietario en su entorno, se sentía limitado e impotente ante situaciones que antes podía solventar fácilmente.

En esta situación, tomó la decisión de abandonar su trabajo para poderse dedicar sin ataduras legales a un nuevo proyecto: la elaboración de un sistema de propósito general completamente libre [5], de tipo Unix, al que llamó GNU [2] (acrónimo recursivo que significa “GNU’s Not Unix” es decir “GNU No es Unix”).

1.1.2. El software libre: definición y significado

El software libre fue definido por R. Stallman como todo aquél que garantice las siguientes libertades:

0. Libertad para ejecutar el programa en cualquier lugar, en cualquier momento y con cualquier propósito.
1. Libertad de estudiar cómo funciona el programa, y adaptarlo a nuestras necesidades (requisito: acceso al código fuente).
2. Libertad para redistribuir copias a cualquier persona.
3. Libertad para mejorar el programa y publicar las mejoras (requisito: acceso al código fuente)

Esta definición es muy cercana a las características de las comunidades de conocimiento científico, en las que los avances se basan en la existencia de canales para el intercambio de conocimiento, la revisión por pares y la publicación de mejoras.

En la práctica, estas cuatro libertades se suelen concretar de acuerdo con la legalidad vigente por medio de una licencia, en la que se también se suelen plasmar algunas restricciones compatibles con ellas, como la obligación de dar crédito a los autores originales cuando redistribuyamos el trabajo. Algunos de los tipos de licencias libres más comunes:

- **BSD:** Estas licencias imponen muy pocas restricciones, estando muy cercanas al dominio público. Tanto es así que permiten incluso el que puedan existir derivaciones que no sean libres (este es, por ejemplo, el caso del sistema privativo Mac OS X, derivado de los BSD).
- **GPL:** Con el fin de proteger las cuatro libertades anteriores, se impone una restricción adicional, compatible con éstas: los trabajos derivados tienen que mantener la misma licencia libre que el trabajo original.

El mecanismo genérico que utilizan las licencias tipo GPL para conseguir estas garantías fue llamado *copyleft*, en un ingenioso juego de palabras, que hoy día es el nombre de una gran familia de licencias de software libre.

Es fundamental entender que con el concepto de software libre no estamos hablando simplemente de software gratuito: el software libre se puede vender si se desea (y en muchas ocasiones puede haber quien esté interesado en pagarlo). En este sentido, el software libre también puede ser “software comercial” y, de hecho, parte del modelo de negocio de algunas empresas (por ejemplo, distribuidoras de GNU/Linux como RedHat o SuSE) se centra en la venta de software libre. Aunque quien lo adquiriera debe ser consciente de que (debido a la tercera libertad) podrá redistribuirlo cuando desee y como lo desee, por ejemplo sin pedir dinero a cambio ni permiso a nadie.

También es conveniente distinguir el software libre de otros conceptos, como *freeware* (software gratuito, pero sin libertad de estudio o modificación) o *shareware* (programas que se pueden evaluar de forma gratuita pero durante un tiempo, uso o características limitadas).

A nivel práctico, asociaremos el concepto de software libre con el de “software de fuente abierta” o de “código abierto” (*open source*), pues la única distinción es el enfoque que normalmente desean transmitir quienes utilizan esta denominación, mucho más pragmático, menos centrado en la ética y en la defensa de los derechos de los usuarios.

Existen decenas de miles de programas que pueden considerarse libres, desde pequeños juguetes de sólo unas líneas de código hasta sistemas operativos completos, como GNU/Linux, FreeBSD, etc, incluyendo a muchos de los programas más difundidos para servicios críticos en empresas e instituciones de todo el mundo. Algunos casos de éxito:

- **Mozilla Firefox.** Derivado del antiguo Netscape, Mozilla Firefox es el mejor navegador web existente en el mercado. Estable, seguro e integrado con los estándares de Internet, las características avanzadas que incorpora (uso de pestañas, filtros para ventanas emergentes, etc) marcan tendencia y, años más tarde, son incorporadas por el resto de los navegadores.

- *OpenOffice.org*, una completa *suite* ofimática (proceso de textos, hoja de cálculos, presentaciones...) que utiliza de forma nativa el estándar OpenDocument (ISO-26300) y es altamente compatible con MS Office.
- *Apache*, un servidor web con licencia libre que constituye el programa utilizado por el 70 % de los sitios web en el mundo, entre ellos servicios críticos de grandes corporaciones.
- En la lista que *Top500.org* publica semestralmente con los 500 supercomputadores más potentes del mundo. GNU/Linux ocupa el 73,40 % de los puestos (por ejemplo, Microsoft ocupa el 0,4 %, concretamente solamente solo hay dos ordenadores, que se sitúan en los lugares 132 y 472 del ranking).

1.1.3. El sistema operativo GNU/Linux

Aunque desde el principio el proyecto GNU, fundado en los años 80 para la creación de un sistema operativo completamente libre, incluyó en su sistema software que ya estaba disponible con una licencia que se podía considerar libre (como T_EX, o más adelante, el sistema X Window), había mucho que construir. Richard Stallman comenzó por escribir un editor (Emacs) y un compilador de C (GCC), ambos aún en uso (y muy populares) hoy día y más gente se unió a él.

A principios de la década de 1990, unos seis años después de su nacimiento, el proyecto GNU estaba muy cerca de tener un sistema completo similar a Unix. Sus productos ya gozaban de una merecida reputación de estabilidad y calidad y eran muy populares entre los usuarios de las distintas variantes de Unix, por aquella época el sistema operativo más usado en las empresas. Así, el proyecto GNU había conseguido ser relativamente conocido entre los profesionales informáticos y muy especialmente entre los que trabajaban en universidades. Aun así, hasta ese momento, todavía no había producido la pieza necesaria para culminar el rompecabezas: el núcleo del sistema, la pieza que se relaciona con el hardware y permite que todo funcione.

En julio de 1991 Linus Torvalds (por entonces, estudiante finés de 21 años) anunció por primera vez su proyecto de crear un núcleo de tipo Unix para sistemas Intel PC 386, al que llamó Linux. En septiembre libera la primera versión (0.01) y cada pocas semanas aparecen nuevas versiones. En marzo de 1994 apareció la primera versión que fue denominada estable (1.0), pero el núcleo desarrollado por Linus era usable desde bastantes meses antes. Durante este periodo, literalmente cientos de desarrolladores se habían volcado en su desarrollo y en su integración con el software de GNU y con muchos otros programas libres, como el sistema gráfico X-Windows, dando lugar al sistema operativo que hoy es conocido como GNU/Linux o, a veces, simplemente como Linux. Con el paso de los años este sistema operativo se extendió a arquitecturas distintas de Intel, como Sparc, ó AMD64.

Con GNU/Linux nacieron distintas *distribuciones*: selecciones de aplicaciones libres reunidas, configuradas y coordinadas que facilitan el proceso de instalación del sistema

para usuarios no expertos, a la vez que simplifican la gestión del software. Para ello, suelen utilizar paquetes de software, ficheros que contienen programas, documentación, etc. y que facilitan su instalación/desinstalación y configuración. Existen decenas de distribuciones, algunas con un carácter marcadamente comercial y otras manteniendo un espíritu más cercano al concepto de comunidad de intercambio de conocimiento. Entre las más conocidas: Debian, RedHat, Suse, Mandriva...

A finales de los años 1990, a la vez que otros sistemas operativos de escritorio (como Windows 95 y 98) se estaban extendiendo en los hogares, el sistema GNU/Linux continuaba creciendo y madurando. Entre sus bazas estaba el permitir convertir cualquier ordenador personal en una estación de trabajo de tipo Unix, segura, multiusuario, estable y orientada a Internet, lo que propició su implantación en universidades y en centros científicos, en ordenadores para cálculo numérico y en servidores, donde hoy goza de una posición privilegiada.

La aparición de sistemas de escritorio como GNOME y KDE, que forman una capa homogénea sobre X-Windows para interactuar con ventanas, menús, iconos, etc, ha permitido su acercamiento al gran público, así como la aparición de iniciativas políticas como la de la Junta de Extremadura y de Andalucía, que incentivan su uso en las escuelas y en la sociedad aprovechando, sus ventajas: ahorro de costes, impulso del desarrollo tecnológico local, reducción de la brecha digital frente a las grandes multinacionales del software, etc. En el mercado de los servidores, GNU/Linux es un “terreno neutral”, atrayendo las inversiones de compañías de hardware (y software) de la talla de HP, Sun ó IBM.

En el caso de Andalucía, se optó por elaborar una distribución específica, a la que se llamó Guadalinux, que inicialmente estuvo basada directamente en Debian GNU/Linux y que ahora se basa en Ubuntu.

Debian es una distribución desarrollada a través de la colaboración de miles de voluntarios de todo el mundo. Se caracteriza por su fidelidad a la filosofía del software libre y por la diversidad de arquitecturas de ordenadores soportadas (entorno a 15 desde las derivadas del Intel 386 hasta las utilizadas en super-ordenadores). Pero, especialmente, por la calidad de su sistema de paquetes de software y por su abundancia (actualmente, en torno a 15.000).

Debido a estas características, suele tener el problema de su poca agilidad (poco frecuente actualización de sus versiones estables) y su dificultad de uso para los nuevos usuarios. Ubuntu es una distribución basada en Debian que, heredando muchas de sus ventajas, intenta aportar algunas nuevas, como facilidad de uso, introducción de las últimas versiones disponibles de los programas y ciclo regular de publicación de versiones del sistema (dos al año). Estas características motivaron que, a partir de la versión V3, Guadalinux utilizara a Ubuntu como base.

1.2. El software libre y las matemáticas en el aula

El uso de las tecnologías de la información y de la comunicación aumenta día a día y en todos los ámbitos. En particular, el uso del ordenador en las aulas de asignaturas de matemáticas está abriendo las puertas en nuestros días a nuevas oportunidades y a terrenos inexplorados. Nuestros alumnos constituyen una generación habituada a los contenidos audiovisuales y el uso de los ordenadores en el aula (obviamente sin ser la panacea que pueda resolver las dificultades del sistema educativo) puede constituir un recurso de tanta utilidad como la pizarra, el libro y el mapa.

1.2.1. Software libre y educación

La elección de los programas usados en el aula se debería guiar por una serie de consideraciones, de las cuales, calidad del producto y su adecuación a la asignatura son, por supuesto, requisitos previos. Pero existe una tercera cuestión que, en la práctica, tendrá una importancia crucial: la licencia de los productos elegidos y las connotaciones que este factor implica.

Una cuestión de precio La más evidente de estas consideraciones es simplemente una cuestión económica: para que un programa con licencia privativa pueda ser usado en las aulas, es necesaria la adquisición de suficientes licencias para cubrir los puestos de trabajo en las aulas de informática.

La copia ilegal La dependencia de programas privativos conlleva problemas éticos añadidos, puesto que de forma irremisible provoca en el alumnado y, en general, en toda la comunidad educativa, la *seducción por una marca cuyo precio hará que, en la mayoría de los casos, no pueda ser adquirida legalmente*, incitando su copia ilegal. Consciente de la actitud ilícita que provoca a sus alumnos, al profesor, que no puede ser guardián de los intereses de una empresa, le caben solamente dos opciones: o bien simular no ser consciente de esta situación, o bien apoyar abiertamente el que sus alumnos realicen actividades ilegales, como la copia de programas de ordenador cuyos autores no lo permiten.

Mejor con software libre En el otro extremo se sitúa el caso del software libre permitiendo que sea instalado en tantos ordenadores como sea conveniente y que el profesor comparta con sus alumnos, con toda legalidad, las herramientas utilizadas (quizás, acompañadas de material docente propio), facilitándoles reproducir en sus hogares el entorno de trabajo del aula. Más aún, al usar en el aula una herramienta con licencia libre, el profesor cuenta con ventajas adicionales a la hora de la planificación y el desarrollo de la asignatura, derivadas de *tener la garantía de que los programas podrán ser instalados y usado por los alumnos en su propio domicilio*, y además podrán ser instalados y usados por el profesor en tantos puestos como sea necesario.

Experimentar con varios programas El software con licencia libre permite y de hecho fomenta el *disponer de varias herramientas a la vez, complementarias o capaces de interactuar entre sí*, cada una de las cuales contará con sus puntos fuertes y sus debilidades. Aunque el profesor se decante por una de ellas, siempre podrá ofrecer a sus alumnos la enriquecedora posibilidad de experimentar con otras, de resolver un mismo problema desde distintas perspectivas y de saciar su curiosidad a aquellos que cuenten con mayores inquietudes.

Saciar la curiosidad Incluso, aunque sea muy poco el porcentaje de nuestros alumnos a los que esta cuestión les pueda interesar (de la misma manera que muy pocos de ellos se dedicarán a la física, a la literatura o al periodismo), el software libre ofrece la interesante posibilidad de *saciar la curiosidad de aquellos alumnos (¡y profesores!) interesados en saber cómo funcionan los programas de ordenador que están utilizando*. No solo esto, de *poder contribuir a mejorar los programas utilizados por sus compañeros o discípulos*. Y para ello no es necesario tener la categoría de gran experto: diseñar y enviar a los autores un nuevo icono, corregir o ampliar la documentación, proponer un ejemplo didáctico, sugerir un nuevo algoritmo... El poder observar y complementar un programa que es utilizado por miles de personas, profesionales y estudiantes de todo el mundo, constituye una experiencia tremendamente gratificante, de gran valor docente, como refuerzo y motivación.

Formación neutral Tengamos en cuenta que la velocidad con la que evoluciona la sociedad de la información puede hacer que tecnologías que hoy son hitos incuestionables sufran mañana severos cambios e incluso lleguen a ser superadas y olvidadas en cuestión de años: *¿qué procesador de textos estábamos usando hace una década?*. Por tanto *una formación basada en la excesiva dependencia de una única herramienta comercial, puede llegar, con el tiempo, a ser obsoleta*. Los estudiantes deberían estar formados en habilidades generales, en conocimiento neutral, y no en los productos concretos de una sola marca comercial. Sólo de esta manera se garantizará el carácter universal de los conocimientos adquiridos y se evitará que la no disponibilidad de un producto o sus carencias evidencien las lagunas del proceso formativo.

Valores éticos Y en último lugar, uno de los argumentos más importantes pero, con frecuencia, no suficientemente valorado, debido quizás al desconocimiento del software libre y a la asimilación social de los valores que conlleva el software privativo: impulsando el software en el aula y con él los valores éticos asociados, *estaremos basando la educación en pilares como la libertad, el conocimiento, la solidaridad y la colaboración*.

1.2.2. Software matemático

Para terminar este capítulo, introduciremos un listado de algunas de las aplicaciones con licencia libre que pueden resultar de utilidad para un profesor de matemáticas:

Geometría

- Dr. GEO (Geometría dinámica e interactiva). En Guadalinex
<http://www.offset.org/drgeo>
- Kig (Geometría interactiva). En Guadalinex
<http://edu.kde.org/kig/>
- Kseg (Geometría interactiva). En Guadalinex
<http://www.mit.edu/~ibaran/kseg.html>
- Eukleides (Lenguaje para construcción de figuras geométricas). En Guadalinex
<http://www.eukleides.org/>
- Geomview (Visor de objetos 3D interactivo). En Guadalinex
<http://www.geomview.org/>
- PyGeo (Marco para la creación de construcciones geométricas dinámicas, utilizando el lenguaje Python).
<http://pw1.netcom.com/~ajs/>

Aritmética

- Kcalcul (Operaciones aritméticas: suma, resta, multiplicación y división)
<http://website.lineone.net/~a-m.mahfouf/kalcul.html>
- Kpercentage (Cálculo de porcentajes). En Guadalinex
<http://edu.kde.org/kpercentage/>
- Xabacus y Xmabacus (Operaciones con un ábaco). En Guadalinex
<http://ftp.tux.org/pub/tux/bagleyd/xabacus>

Representación gráfica

- GNUpot (Funciones y tablas de valores, 2d y 3d). En Guadalinex
<http://gnuplot.info/>
- Kmplot (Funciones en 2D). En Guadalinex
<http://edu.kde.org/kmplot/>
- Geg (Funciones en 2D). En Guadalinex
<http://www.infolaunch.com/~daveb/>

- Superficie (Superficies en 3d).

<http://superficie.sourceforge.net/>

Fractales

- Xaos (Visor interactivo de fractales en tiempo real). En Guadalinex

<http://xaos.sourceforge.net/>

Juegos

- Tux, of Math Command (tuxmath). (Dispara a las naves espaciales y resuelve operaciones matemáticas) En Guadalinex

<http://www.newbreedsoftware.com/tuxmath/>

- MathWar (Resolver operaciones matemáticas). En Guadalinex

<http://webpages.charter.net/stuffle/linux/software.html#mathwar>

Estadística

- R (excelente programa de cálculo estadístico). <http://www.r-project.org/>

Cálculo simbólico y numérico

- Octave (Excelente programa de cálculo matricial y numérico). En Guadalinex

<http://www.octave.org>

- Yacas (Cálculo simbólico). En Guadalinex

<http://yacas.sourceforge.net/>

- Axiom (Cálculo simbólico). En Guadalinex

<http://www.axiom-developer.org/>

- PARI/GP (Teoría de números). En Guadalinex

<http://www.parigp-home.de/>

- Y por supuesto... **Maxima** (Cálculo simbólico). En Guadalinex

<http://maxima.sourceforge.net/>

1.3. Breve historia de *Maxima*

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

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

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

Puesto que Maxima se distribuye bajo la licencia GNU-GPL, tanto el código fuente como los manuales son de libre acceso a través de la página web del proyecto (<http://maxima.sourceforge.net>). *Maxima* es, por tanto, 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 (y que se enumerarán en la sección 2.2). Entre ellos, este curso se centrará en *wxMaxima*, desarrollado por Andrej Vodopivec y disponible en <http://wxmaxima.sourceforge.net>.

2 Capítulo

Primer contacto con *Maxima* y *wxMaxima*

2.1. Instalación

En este apartado se detallará el proceso e instalación de *Maxima* y *wxMaxima*, prestando especial atención al caso de Guadalinex (y, por tanto, de Ubuntu y sistemas derivados de Debian).

2.1.1. *Maxima*

Maxima puede funcionar en distintos sistemas operativos, entre ellos diversas variantes de Windows y de GNU/Linux. En el resto de esta sección nos centraremos en el caso de Guadalinex (V3 o superior aunque, en particular, todo lo que se comentará es aplicable de forma directa a Ubuntu o a cualquier otra distribución basada en Debian).

El lector interesado en utilizar *Maxima* en alguna otra variante de GNU/Linux o en Windows, puede acceder a la sección *Download* de la web de *Maxima* y seguir las instrucciones que en ella se indican.

Para utilizar este programa desde Guadalinex, podemos actuar de la forma habitual: abrimos un gestor de paquetes, por ejemplo *Synaptic* (menú "Sistema" → "Administración"), buscamos el paquete¹ "maxima", lo marcamos para instalar². Por último, aplicamos los cambios que hemos marcado³.

Además de "maxima", es recomendable instalar algunos otros paquetes complementarios:

- "maxima-share" Extensiones de *Maxima*, muchas de las cuales introducirán nuevas bibliotecas y funcionalidades que nos resultarán de gran utilidad.
- "maxima-doc" Documentación sobre *Maxima*, en particular el manual "oficial" del programa, que por defecto se instala en `/usr/share/doc/maxima/` y que puede ser abierto desde *wxMaxima*.
- "gnuplot-x11": representación de gráficos 2D y 3D.

¹Pinchando en el icono "Buscar", o bien el menú "Editar" → "Buscar..." o bien tecleando CTRL+F

²Ya sea haciendo doble *click* sobre él, o pulsando el botón derecho del ratón, o usando la entrada de menú "Paquete" → "Marcar para instalación" o bien pulsando CTRL+I

³Para ello, deberemos hacer *click* en el botón "Aplicar" o bien en el menú "Editar" → "Aplicar cambios marcados" o bien teclear CTRL+F

Debemos tener en cuenta que, puesto que no hemos instalado ninguna interfaz gráfica de usuario, para comenzar a utilizar *Maxima* será necesario abrir una consola (menú "Aplicaciones" → "Accesorios" → "Terminal") y teclear en ella el comando "maxima".

2.1.2. *wxMaxima*

La instalación de *wxMaxima* en Guadalinex, en Ubuntu o en cualquier sistema derivado de Debian, sigue el mismo proceso que se comentó en el apartado anterior:

- Abrir un gestor de paquetes, por ejemplo *Synaptic*
- Buscar el paquete "wxmaxima"
- Marcarlo para instalar
- Aplicar los cambios marcados

Una vez instalado en el sistema, contaremos con una nueva entrada en el menú (del tipo de "Aplicaciones" → "Otros" → "wxMaxima"), que es el camino que utilizaremos habitualmente para arrancar el programa.

Sin embargo, en Guadalinex V3 (no en V4 y posteriores) se nos presenta una dificultad: *wxMaxima* no se encuentra entre los paquetes disponibles, ya que en el momento del lanzamiento de su lanzamiento, el paquete *wxMaxima* aún no había sido incluido en Ubuntu, su distribución base.

La solución es simple⁴: se trata de crear un paquete de *wxMaxima* específicamente para Guadalinex V3. Esta es la tarea que se ha llevado a cabo en la Oficina de Software Libre de la Universidad de Cádiz, dando como resultado una serie de paquetes disponibles en <http://osl.uca.es/debian/breezy-backports/>.

Para instalar estos paquetes, es suficiente con añadir la dirección web anterior a la lista de repositorios de Guadalinex. Si usamos *Synaptic*, basta acceder al "Configuración" → "Repositorios" → "Añadir" → "Personalizado", a continuación introducimos en "Línea de APT" lo siguiente:

```
deb http://softwarelibre.uca.es/debian breezy-backports/
```

A partir de ese momento, tendremos disponible el paquete "wxmaxima" (versión 0.7.0), que podrá ser instalado de la forma habitual. Al hacerlo, también será instalada la versión 5.10 de *Maxima*. Además, se podrá contar con la versión 5.10 del resto de los paquetes complementarios.

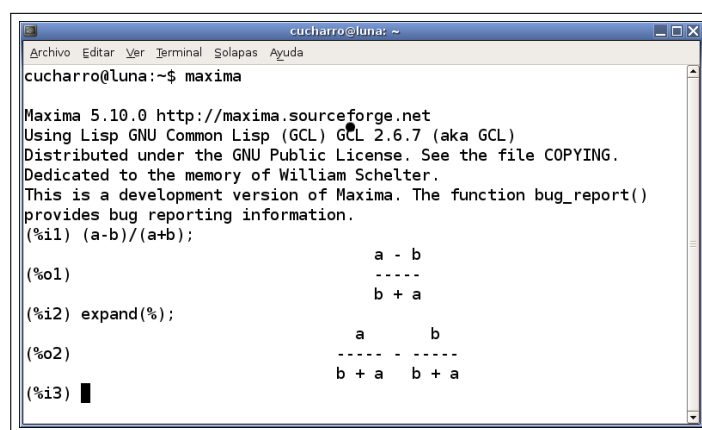
Para la instalación de *wxMaxima* en otras variantes de GNU/Linux o de Windows, se puede consultar wxmaxima.sourceforge.net. En el caso de Windows, resultará

⁴Aunque, técnicamente, para ponerla en práctica es necesario tener conocimientos sobre la creación de paquetes Debian, además ha sido necesario sortear algunos problemas de conexión entre *Maxima* y *wxMaxima* que han sido solucionados con la versión 5.10 del primero, lo que ha hecho necesario empaquetar también esta versión de *Maxima* y crear el resto de paquetes complementarios

necesaria una tarea adicional de post-instalación: indicar a *wxMaxima* cuál es la localización concreta del programa ejecutable de *Maxima* dentro del árbol de carpetas de Windows. Para ello, se debe utilizar el menú "Editar" → "Preferencias" y rellenar el campo "Programa Maxima:"

2.2. Entornos de ejecución: la consola *Maxima* y *wxMaxima*

Como se ha comentado anteriormente, *Maxima* es un potente motor de cálculo simbólico (y numérico). El paquete básico permite utilizar sus funcionalidades a través de una consola de texto (figura 2.1), preparada para que el usuario comience a introducir órdenes en el lenguaje de *Maxima*.



```

cucharro@luna: ~
Archivo Editar Ver Terminal Solapas Ayuda
cucharro@luna:~$ maxima

Maxima 5.10.0 http://maxima.sourceforge.net
Using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (aka GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1) (a-b)/(a+b);

              a - b
              ----
              b + a

(%o1)

(%i2) expand(%);

              a      b
              ---- - ----
              b + a  b + a

(%o2)

(%i3) █
  
```

Figura 2.1: *Maxima* ejecutándose en entorno consola

Aunque puede resultar demasiado espartano para personas nuevas en *Maxima*, este entorno permite acceder a todas sus posibilidades y muchos de los usuarios más avanzados pueden preferir, en ocasiones, la claridad y velocidad que proporciona el acceso a las funcionalidades sin necesidad de navegar entre árboles de menús con el ratón.

Pero además, existen una serie de programas, entre ellos *wxMaxima*, que actúan como entornos gráficos, permitiendo al usuario el ejecutar *Maxima* de forma indirecta e interactuar con él mediante filosofías más "visuales" (figura fig:interfaz-wxmaxima). Estos entornos están dotados de licencia libre y se pueden instalar de forma complementaria a *Maxima*, siguiendo un proceso similar a lo que se comentó en el apartado anterior. Cada una de ellas tiene unas características propias que la pueden hacer más adecuada para unos usuarios u otros.

En este manual se ha optado por elegir desde el principio una de estas interfaces para fijar concretamente los contenidos y la forma de interactuar con el programa. La interfaz elegida es *wxMaxima*, debido a que se trata, quizás, de aquella que puede resultar más amable para un profesor o un estudiante que se enfrente por vez primera al uso de

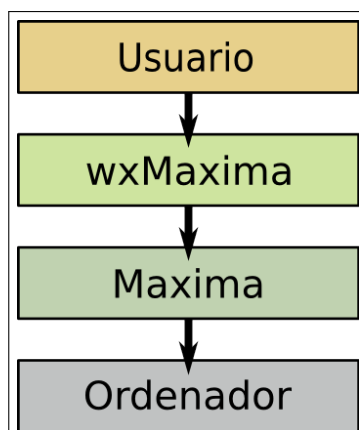


Figura 2.2: *wxMaxima* como interfaz de *Maxima*

Maxima dentro del aula de matemáticas. A partir de la próxima sección se comenzará a estudiar este programa con detalle.

Aun así, se ha creído conveniente ofrecer una breve panorámica de los entornos existentes, con la intención de ofrecer suficientes elementos de juicio como para que el lector interesado pueda explorar la riqueza que proporciona a *Maxima* el contar con tal diversidad de interfaces:

- *xmaxima* (figura 2.3), la primera interfaz gráfica que fue desarrollada, es mantenida “oficialmente” por el equipo de desarrollo de *Maxima*. De hecho, en algunos sistemas como Windows, se instala automáticamente y es arrancada por defecto. Presenta algunas ventajas, como la integración, en formato HTML, de manuales de ayuda. Sin embargo, también tiene algunas desventajas con respecto a otras interfaces más modernas como *wxMaxima* y presenta menos ventanas y menús de apoyo que ésta.
- *TeXmacs* (figura 2.4) es un proyecto iniciado en 1998 el C.N.R.S (Instituto Nacional Francés para la Investigación Científica), con el propósito de crear una aplicación para redactar textos matemáticos de forma sencilla. Su nombre tiene raíces en el sistema \TeX , en el que se basa, y en el conocido editor de textos Emacs, del que toma parte de su filosofía.

El resultado es un editor de textos de tipo WYSWYG⁵ que permite la creación de documentos matemáticos y científicos con la calidad que aporta \LaTeX , utilizado como motor de maquetado.

Pero el verdadero potencial de *TeXmacs* estriba en la posibilidad de incorporar y utilizar con comodidad sesiones interactivas de numerosos motores de cálculo, entre ellos *Maxima*. En este sentido, *TeXmacs* es una “interfaz universal” para programas matemáticos, aportando a los mismos ventajas adicionales, como un modo

⁵What You See is What You Get, lo que ves (en pantalla) es lo que obtienes (al imprimir)

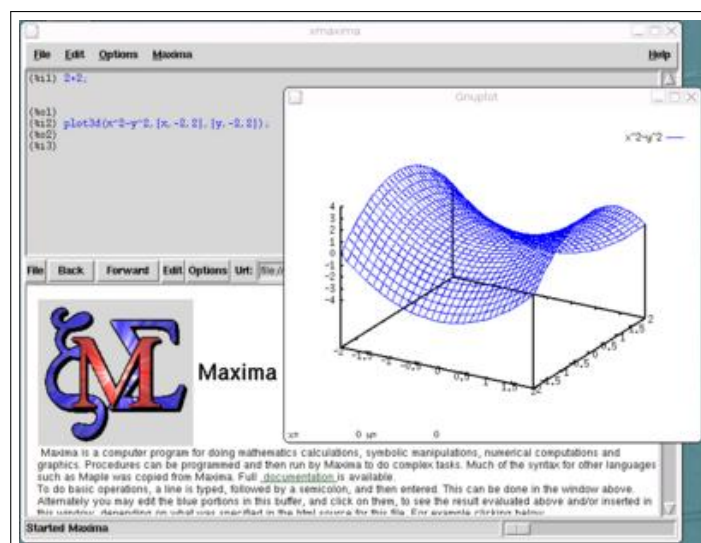


Figura 2.3: *Maxima* ejecutándose bajo la interfaz *wxMaxima*

(algo limitado) para la introducción de expresiones con el ratón y, en especial la mejora del aspecto de las salidas matemáticas.

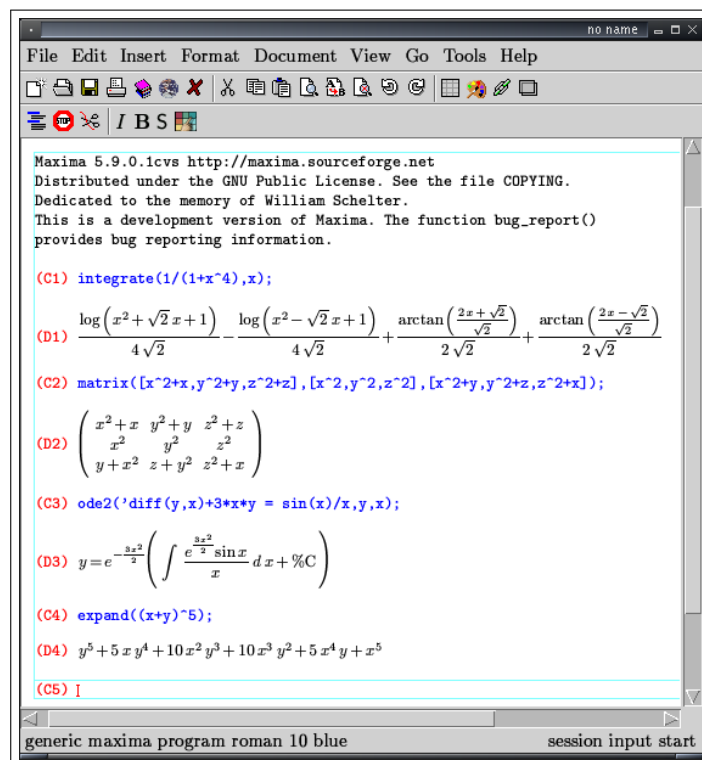
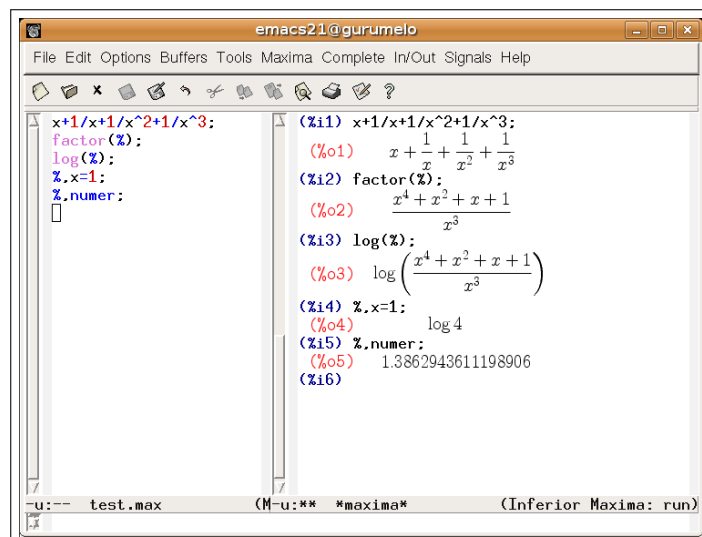
Pero, por otra parte, el carácter genérico de *T_EXmacs* hace que no incluya acceso mediante menús o ventanas a las funciones específicas de *Maxima*.

- *Emacs* (figura 2.5) es un editor de textos completamente configurable que constituye una plataforma de desarrollo con modos específicos para decenas de lenguajes de programación (entre ellos, el lenguaje de *Maxima*) y con soporte para la edición de ficheros *T_EX/L_AT_EX*.

Aunque cuente con un modo que permite ejecutar una sesión *Maxima* (para ello, debemos teclear M-X y escribir “maxima”), los usuarios más avanzados podrán encontrar más interesante la posibilidad de editar ficheros en lenguaje *Maxima* (con reconocimiento y coloreado de sintaxis y aprovechando las ventajas de un editor tan avanzado). Estos ficheros pueden ser enviados a *Maxima* para su evaluación, completamente o línea a línea (para ello, en *wxMaxima* se puede utilizar el menú “Archivo” → “Archivo de lotes”).

Por último, un par de modos que pueden resultar interesantes a la hora de usar *Maxima* en *Emacs*: Por un lado, “imaxima”, que se puede utilizar para que *Emacs* use *T_EX* para embellecer las salidas de *Maxima* (como se puede ver en la ventana derecha de la figura 2.5). Y por otro lado, “emaxima”, ideado para la redacción de documentos *T_EX/L_AT_EX* en los que haya que incluir sesiones o comandos de *Maxima*. Este es el modo se ha utilizado para la redacción del presente manual.

- ... y, por supuesto, *wxMaxima*, basada en la biblioteca gráfica *wxwidgets*, gracias a la cual existen versiones nativas tanto para sistemas operativos GNU/Linux co-

Figura 2.4: *Maxima* ejecutándose en una sesión *T_EXmacs*Figura 2.5: Un fichero maxima (*buffer* izquierdo) y una sesión imaxima (*buffer* derecho). Ambos, interactuando en *Emacs*

mo para Windows. Integra elementos específicos para la navegación de la ayuda, introducción de matrices, creación de gráficas, cálculo de límites, derivadas o integrales, etc. A partir de la próxima sección se estudiará con más detalle.

2.3. Primeros pasos y exploración del entorno *wxMaxima*

2.3.1. Abrir *Maxima* en modo consola

La experiencia de un primer contacto con *Maxima* puede resultar completamente diferente según la interfaz que empleemos.

En el caso de utilizar una consola de texto (y, en mayor medida, en otras interfaces, como *xmaxima* o *TEXmacs*), contaremos básicamente con una pantalla de bienvenida similar a la siguiente:

```
Maxima 5.10.0 http://maxima.sourceforge.net
Using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (aka GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

A través de estas líneas, *Maxima* nos ofrece diversos datos, entre ellos la versión con la que estamos trabajando y la dirección web del proyecto. La clave está en el indicador que aparece en la última línea, con la etiqueta `(%i1)` seguida del cursor en indicación de que se encuentra esperando a la primera entrada (en inglés *input*, de donde proviene la letra “i” de la etiqueta) para interaccionar con el programa.

Podemos, por ejemplo, calcular una suma muy sencilla, tecleando la operación deseada seguida de un punto y coma (“;”) y una pulsación de la tecla RETORNO. *Maxima* procesará la operación y nos devolverá el resultado, precedido de la etiqueta `(%o1)` (del inglés *output* 1). Por ejemplo:

```
(%i1) 2+2;
(%o1)                                     4
(%i2)
```

La etiqueta `(%i2)` irá seguida del cursor, en espera de una segunda instrucción.

A partir de este momento, podremos acceder a todas las posibilidades que encierra *Maxima*, eso sí, siempre que contemos con un manual adecuado. En este sentido, se puede recomendar el de Mario Rodríguez Riotorto (Primeros Pasos en Maxima) [4] y, por supuesto, el manual “oficial” [1].

2.3.2. Primer contacto con *wxMaxima*

Por otro lado, cuando accedemos a *Maxima* a través de *wxMaxima* (pinchando en el menú de aplicaciones de nuestro entorno de ventanas), encontraremos una ventana amigable, con numerosos botones y menús. La podemos considerar dividida en distintas secciones (ver la figura 2.6):

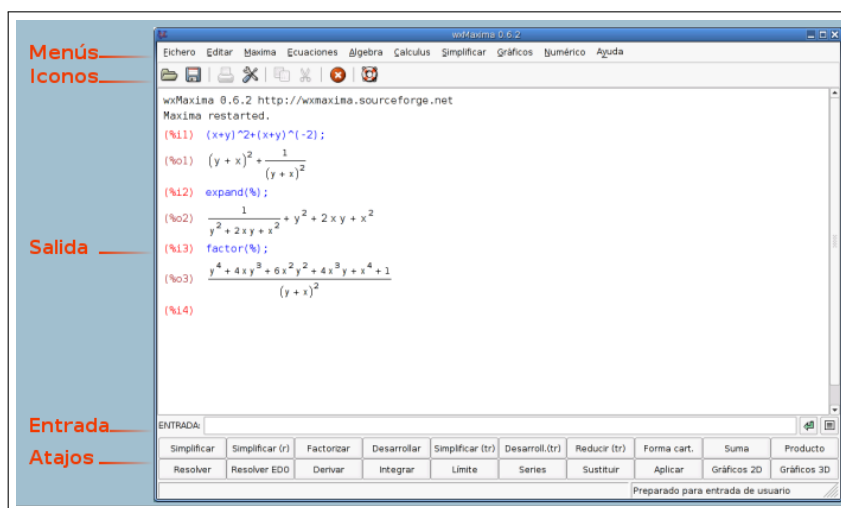


Figura 2.6: Distintas áreas de *wxMaxima*

1. *Barra de menús*: Nos permite acceder al motor de cálculo simbólico *Maxima*
2. *Barra de iconos*: Acceso rápido a algunas de las opciones de la barra de menús
3. *Área de salida o consola*: En ella se muestran los resultados
4. *Área de entrada*: Para teclear comandos
5. *Área de botones o atajos*: Otro punto de acceso rápido a los comandos de *Maxima*

Inicialmente, el *área de salida* contiene una información similar a la que antes apreciábamos, con información sobre el programa:

```
wxMaxima 0.6.4 http://wxmaxima.sourceforge.net
Maxima restarted.
(%i1)
```

Como antes, comenzaremos realizando una simple suma. Para ello, en esta ocasión nos debemos situar en el *área de entrada* (con el ratón o pulsando la tecla F4) y teclear la operación que deseemos, por ejemplo, “44+77” (ver figura 2.7). A continuación pulsamos la tecla de retorno.

ENTRADA:

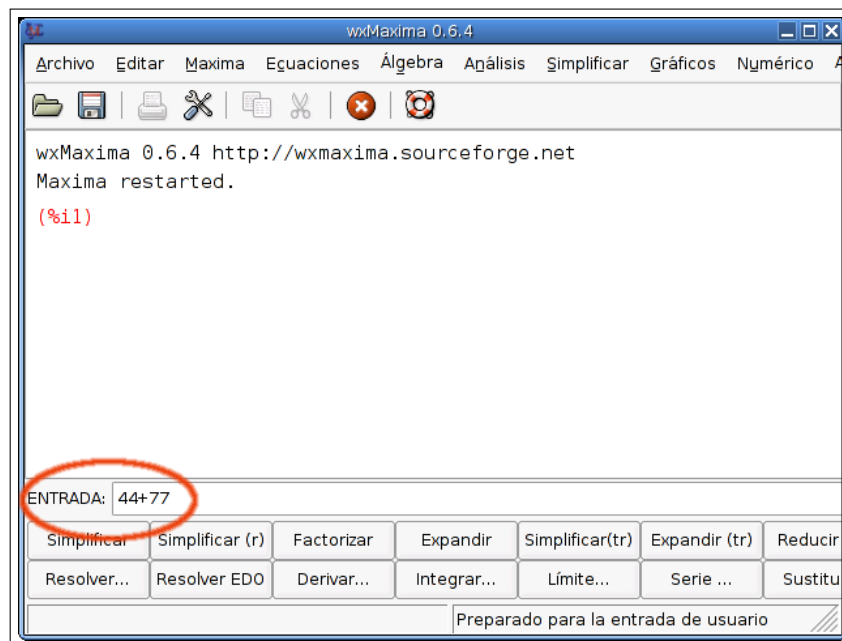


Figura 2.7: Utilización del área de entrada

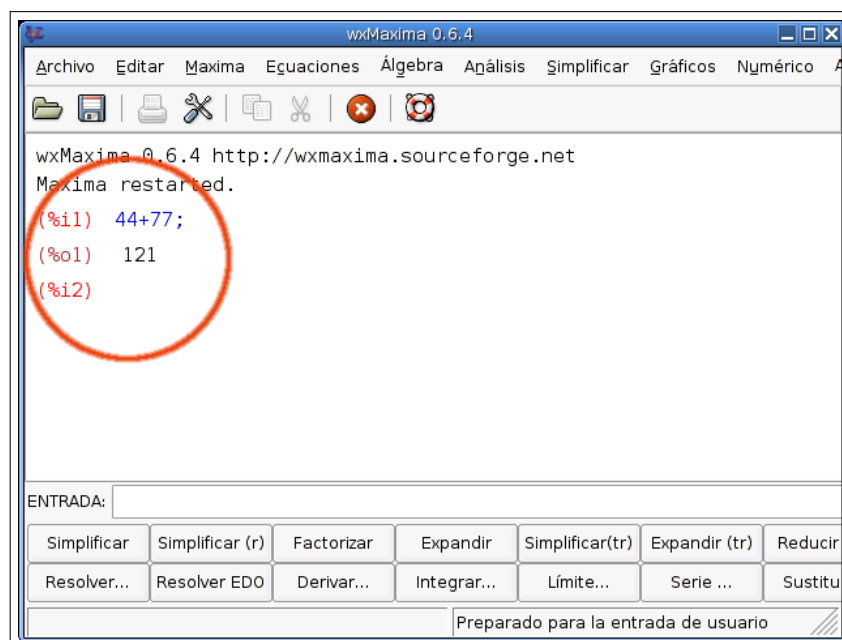


Figura 2.8: Resultados obtenidos en el área de salida

wxMaxima responderá mostrando, en el área de consola, el resultado de la operación anterior (ver figura 2.8).

En el diseño del área de entrada se han incluido algunas características destinadas a facilitar la introducción de expresiones del lenguaje *Maxima*. Por ejemplo, ya no es necesario que las entradas utilicen el carácter punto y coma para indicar el fin de línea, como se puede apreciar en el ejemplo anterior. Pero el punto y coma en *Maxima* actúa también como un separador cuando escribimos varias instrucciones seguidas. Nuestro siguiente ejemplo consistirá en asignar el valor 32123 a la variable *x*, el 1234321 a *y*, para luego solicitar su producto. Para ello, tecleamos:

ENTRADA: `x:32123; y:1234321; x*y`

y obtenemos una salida similar a la siguiente:

```
(%i1)  x:32123;
(%o1)

32123

(%i2)  y:1234321;
(%o2)

1234321

(%i3)  x*y;
(%o3)

39650093483
```

Como se puede apreciar, en el lenguaje de *Maxima* se utiliza el símbolo de dos puntos (":") para asignar un valor a una variable. El símbolo de igualdad ("=") queda reservado para las ecuaciones.

En ocasiones, no nos interesará que aparezcan en pantalla los valores de algunas operaciones, por ejemplo la asignación de valores a variables intermedias, para lo cual podemos utilizar el carácter "\$" como sustituto del separador ";". Por ejemplo, si escribimos:

ENTRADA: `x:321123$ y:123321$ x/y`

obtenemos una salida similar a la siguiente, mucho más limpia que en ejemplo anterior y similar a la siguiente:

```
(%i4)  x:321123$
(%i5)  y:123321$
(%i6)  x/y;
(%o6)

263
101
```

Como vemos, *Maxima* opera con aritmética racional y, por defecto, nos devuelve una fracción como resultado. Si añadimos una coma (",") seguida de la orden "numer", se obtendrá una expresión numérica, por defecto, con 16 cifras decimales. Por ejemplo, realizaremos la siguiente resta:

ENTRADA:

```
(%i7) 5-2/3;
(%o7)
```

$$\frac{13}{3}$$

y pediremos cuál es el valor numérico de la salida anterior:

ENTRADA:

```
(%i8) % , numer;
(%o8)
```

4.333333333333333

En este ejemplo hemos utilizado por primera vez el operador "%", que se emplea para hacer referencia a la última salida. Se pueden usar las etiquetas "%iN" y "%oN" para acceder, respectivamente, al valor de la entrada y la salida N-ésima (por ejemplo, "%o3" es una referencia a la salida número 3).

Por último, *Maxima* puede trabajar, no solamente con números y variables, sino también con expresiones simbólicas. Por ejemplo, escribimos:

ENTRADA:

```
(%i9) a+b+a/b;
(%o9)
```

$$b + \frac{a}{b} + a$$

Y a continuación usamos la función de *Maxima* "ratsimp", que simplifica expresiones racionales, expresándolas de una forma canónica.

ENTRADA:

```
(%i10) ratsimp(%);
(%o10)
```

$$\frac{b^2 + a b + a}{b}$$

En este caso, debemos ser cuidadosos, pues si las variables utilizadas tienen algún valor, obtendremos un resultado numérico. Por ejemplo, puesto que en un ejemplo anterior habíamos definido x e y , tendríamos:

ENTRADA: $x+y+x/y$

(%i11) $x+y+x/y;$

(%o11)

$$\frac{44889107}{101}$$

Para poder emplear estas variables de forma simbólica tendremos que eliminar su valor:

ENTRADA: $\text{kill}(x, y)$

(%i12) $\text{kill}(x, y);$

(%o12)

done

ENTRADA: $x+y+x/y$

(%i13) $x+y+x/y;$

(%o13)

$$y + \frac{x}{y} + x$$

En *wxMaxima*, la entrada " $\text{kill}(x, y)$ " se puede introducir mediante la entrada de menú "*Maxima*" → "*Borrar Variable*". Y existen muchas otras ventajas similares que se comenzarán a introducir a partir del siguiente apartado.

2.3.3. Sacar partido a las posibilidades de *wxMaxima*

Hasta este momento, nos hemos limitado a utilizar *wxMaxima* como un marco para introducir las órdenes de *Maxima*, sin utilizar más elementos que las áreas de entrada y salida. Pero el entorno gráfico de *wxMaxima* tiene muchas más funcionalidades, algunas de las cuales se mostrarán en los siguientes párrafos.

Por ejemplo, planteemos la siguiente relación de problemas (escogidos sin mayor finalidad didáctica que el ilustrar el funcionamiento de *wxMaxima*):

Problema:

- (a) Descomponer el valor de $10!$ en producto de factores primos
- (b) Factorizar el polinomio $x^6 - 1$
- (c) Multiplicar los factores obtenidos y comprobar que el resultado coincide con el polinomio anterior
- (d) Simplificar la siguiente fracción algebraica:

$$\frac{x^6 - 1}{x^2 + x + 1}$$

- (e) Resolver la ecuación

$$\frac{x^6 - 1}{x^2 + x + 1} = 0$$

Una persona con suficientes conocimientos del lenguaje de *Maxima* los podría haber resuelto mediante la siguiente secuencia de instrucciones:

```
(%i14) factor(10!);
(%o14)
```

$$2^8 3^4 5^2 7$$

```
(%i15) factor(x^6 - 1);
(%o15)
```

$$(x - 1) (x + 1) (x^2 - x + 1) (x^2 + x + 1)$$

```
(%i16) expand(%);
(%o16)
```

$$x^6 - 1$$

```
(%i17) ratsimp((x^6-1)/(x^2+x+1));
(%o17)
```

$$x^4 - x^3 + x - 1$$

```
(%i18) solve(%=0);
(%o18)
```

$$\left[x = 1, x = -1, x = -\frac{\sqrt{3}i - 1}{2}, x = \frac{\sqrt{3}i + 1}{2} \right]$$

La hipotética persona que resolvió el ejercicio anterior habría podido utilizar cualquiera de los entornos de ejecución de *Maxima* que se estudiaron en la sección 2.2.

Entre ellos, podría haber optado por usar la línea de entrada de *wxMaxima* para introducir cada una de las órdenes anteriores y conseguir, de esa manera, factorizar números y polinomios, simplificar fracciones, resolver ecuaciones, etc. Pero sacar realmente partido del entorno *wxMaxima* significa saber utilizar el gran número de utilidades, menús, botones y ventanas específicas que este programa pone a nuestra disposición para facilitarnos el trabajo. La idea es conseguir que, mediante el uso del ratón, los nuevos usuarios puedan realizar un acercamiento a *Maxima* tan amable como sea posible⁶ y los usuarios avanzados puedan aprovechar aquellas características que le puedan ser de utilidad.

De esta forma, aunque no conociéramos las funciones “`factor()`”, “`expand()`”, “`ratsimp()`”, “`solve()`” que fueron utilizadas anteriormente para resolver el problema, podríamos utilizar *wxMaxima* y actuar de la siguiente forma:

- (a) Introducimos el número 10!:

ENTRADA:

```
(%i19) 10!;
```

```
(%o19)
```

3628800

Como podemos ver, *wxMaxima* nos devuelve el valor del factorial.

- (b) Pulsamos el botón [Factorizar] situado en la barra de botones inferior (o utilizamos el menú “Simplificar” → “Factorizar expresión”). *Maxima* nos devuelve el siguiente resultado,

```
(%i20) factor(%);
```

```
(%o20)
```

$2^8 3^4 5^2 7$

- (c) Para factorizar el polinomio $x^6 - 1$ procedemos de la misma forma:

- a) Introducimos el polinomio:

ENTRADA:

⁶Aunque, a medida que aumente su experiencia, puede que muchos usuarios encuentren más eficiente el conocer en profundidad las expresiones en el lenguaje *Maxima*, y utilizar los menús solamente en los casos en los que les resulten realmente necesarios.

- b) Pulsamos el botón [Factorizar] (o utilizamos el menú "Simplificar" → "Factorizar expresión")

Obtendremos el siguiente resultado en el área de consola:

```
(%i21) x^6-1;
```

```
(%o21)
```

$$x^6 - 1$$

```
(%i22) factor(%);
```

```
(%o22)
```

$$(x - 1) (x + 1) (x^2 - x + 1) (x^2 + x + 1)$$

- (d) Una vez factorizado, podemos volver a desarrollar el polinomio sin más que pulsar el botón [Expandir] (o el menú "Simplificar" → "Expandir expresión"), obteniendo:

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

```
(%o23)
```

$$x^6 - 1$$

- (e) Para simplificar la fracción

$$\frac{x^6 - 1}{x^2 + x + 1}$$

podemos actuar de forma análoga: tecleamos

ENTRADA:

```
(%i24) (x^6-1)/(x^2+x+1);
```

```
(%o24)
```

$$\frac{x^6 - 1}{x^2 + x + 1}$$

y pulsamos el botón [Simplificar]

```
(%i25) ratsimp(%);
```

```
(%o25)
```

$$x^4 - x^3 + x - 1$$

(f) A continuación, podemos resolver la ecuación

$$\frac{x^6 - 1}{x^2 + x + 1} = 0$$

o, equivalentemente,

$$x^4 - x^3 + x - 1 = 0.$$

Para ello, no tenemos más que pulsar el botón [Resolver...] (o el menú "Ecuaciones" → "Resolver..."), tras lo cual se abrirá una ventana de diálogo (ventana *Resolver*, figura 2.9) en la que se nos preguntará cuáles son la ecuación y las variables para las que queremos resolver.



Figura 2.9: Ventana de diálogo *Resolver*

Por defecto, estará seleccionada la expresión anterior⁷ (símbolo %) para la variable *x*. Si estamos de acuerdo, no tenemos más que pulsar en el botón [Aceptar] para obtener el resultado:

```
(%i26) solve( [%], [x] );
```

```
(%o26)
```

$$\left[x = 1, x = -1, x = -\frac{\sqrt{3}i - 1}{2}, x = \frac{\sqrt{3}i + 1}{2} \right]$$

Algunas observaciones:

- En el ejemplo anterior, al utilizar el botón [Resolver], la expresión que estamos resolviendo (representada por la variable %) no es una ecuación, sino un polinomio ($x^4 - x^3 + x - 1$). En tal caso, *Maxima* trata de resolver la ecuación homogénea ($x^4 - x^3 + x - 1 = 0$).

⁷Aunque si hubiéramos seleccionado con el ratón alguna expresión del área de salida o si la hubiéramos tecleado en la línea de entrada, sería esta la que aparecería automáticamente como ecuación a resolver.

- Como se puede apreciar, *wxMaxima* introduce entre corchetes (“[...]”) tanto las ecuaciones como las variables y las soluciones. Los corchetes representan listas de datos y es la estructura sobre la que descansa el lenguaje de programación Lisp y, por tanto, *Maxima*. Se volverá a hablar de ellos cuando estudiemos características avanzadas de este programa.
- Como veremos, *Maxima* utiliza listas de ecuaciones y de variables para resolver sistemas de ecuaciones lineales. En *wxMaxima* se puede utilizar para ello el menú “Ecuaciones” → “Resolver sistema lineal...”.

Problema:

Como ejercicio, se propone resolver el siguiente sistema de ecuaciones:

$$\left. \begin{array}{rcl} x + y & = & 0 \\ 2x + 3y & = & -1 \end{array} \right\}$$

2.3.4. Un ejemplo algo más avanzado

Planteemos ahora un problema que nos puede servir como un modelo más completo, relacionado con la representación gráfica de funciones.

Problema:

Dada la función

$$f(x) = \frac{\sqrt{x}}{1+x^2}$$

- Estudiar su dominio, puntos de corte y asíntotas.
- Calcular su función derivada primera. ¿Está dicha función definida en $x = 0$? Hallar la derivada en el punto $x = 1$.
- Determinar sus intervalos de crecimiento y decrecimiento, así como sus máximos y mínimos relativos.
- Representar su gráfica.

Por supuesto, *Maxima* tiene instrumentos para la representaciones gráficas 2D y 3D pero, antes de utilizarlas, nos interesaremos por realizar un estudio analítico con la simple intención de mostrar la forma de proceder con *wxMaxima*:

- Al estudiar el dominio, la raíz cuadrada nos indica que debe ser $x \geq 0$. Pero también deberíamos plantearnos si es posible que el denominador sea cero. Por supuesto, esto es imposible, pues $1 + x^2$ siempre toma valores estrictamente positivos. Pero si hubiera alguna duda, se podría utilizar *Maxima*, para comprobar que $1 + x^2$ no tiene ninguna raíz real (todas sus soluciones son números imaginarios puros). Escribimos la ecuación (obsérvese que utilizamos para ello el símbolo “=”):

ENTRADA: $1+x^2=0$

y a continuación pulsamos el botón [Resolver]

```
(%i1) 1+x^2=0;
```

```
(%o1)
```

$$x^2 + 1 = 0$$

```
(%i2) solve(%);
```

```
(%o2)
```

$$[x = -i, x = i]$$

Un inciso: si tuviéramos más conocimientos de *Maxima*, podríamos haber utilizado la orden “is” (que intenta comprobar si una expresión es cierta). Así, tecleamos lo siguiente (si deseamos escribir lo menos posible podemos copiar y pegar o bien escribir “is (%o4) ”):

ENTRADA:

y obtenemos la expresión “false” (falso):

```
(%i3) is (1+x^2=0);
```

```
(%o3)
```

false

Podemos incluso concretar más: si preguntáramos

ENTRADA:

obtendríamos la respuesta “true” (cierto):

```
(%i5) f(x):=sqrt(x)/(1+x^2);
```

```
(%o5)
```

$$f(x) := \frac{\sqrt{x}}{1+x^2}$$

Se puede observar que hemos empleado la función “sqrt” para la raíz cuadrada. A continuación, podemos estudiar el punto de corte con el eje de ordenadas, obteniendo el punto (0,0):

ENTRADA:

```
(%i6) f(0);
```

```
(%o6)
```

0

Para obtener los puntos de corte con el eje de abscisas, podemos pulsar el botón [Resolver] y escribir la ecuación “ $f(x)=0$ ” (o simplemente “ $f(x)$ ”), obteniendo de nuevo el punto $x = 0$:

```
(%i7) solve([f(x)=0], [x]);
```

```
(%o7)
```

$[x = 0]$



Figura 2.10: Ventana de diálogo *Límite*

Para estudiar las asíntotas horizontales (no existen verticales), podemos utilizar el botón [Límite...], que abrirá una ventana de diálogo (figura 2.10) en la que teclearemos la expresión “ $f(x)$ ” y el valor al que tiende la variable x . Si lo deseamos, podemos utilizar el botón [Especial] (contenido en la ventana *Límite*) para acceder a los valores de $\pm\infty$ (“Infinity” y “- Infinity”). Obtenemos así:

```
(%i8) limit(f(x), x, inf);
```

```
(%o8)
```

0

```
(%i9) limit(f(x), x, minf);
```

```
(%o9)
```

0

- (b) *wxMaxima* nos pone fácil el cálculo de la derivada, a través del menú "Análisis" → "Derivar..." . Si tenemos el panel de botones completo (menú "Editar" → "Preferencias") tendremos también disponible el botón [Derivar...]. Se abrirá una ventana de diálogo (figura 2.11) en la que introduciremos la expresión a derivar ("f (x)"), la variable (x) y el orden de la derivada (1). El resultado es:



Figura 2.11: Ventana de diálogo *Derivar*

```
(%i10) diff(f(x),x);
```

```
(%o10)
```

$$\frac{1}{2\sqrt{x}(x^2+1)} - \frac{2x^{\frac{3}{2}}}{(x^2+1)^2}$$

expresión que queda más compacta si sumamos las dos fracciones, por ejemplo pulsado el botón [Factorizar]:

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

```
(%o11)
```

$$-\frac{3x^2-1}{2\sqrt{x}(x^2+1)^2}$$

En el cociente anterior se aprecia perfectamente que la función no está definida en $x = 0$, pues en tal caso estaríamos dividiendo por cero. Esto se ve con claridad si sustituimos $x = 0$ de la siguiente forma (también podíamos haber utilizado el menú "Simplificar" → "Sustituir..."):

ENTRADA:

El programa nos advierte de una división por cero (*Division by 0*).

Para hallar la derivada en el punto $x = 1$, podemos seleccionar con el ratón la expresión de la derivada y pulsar en el menú "Simplificar" → "Sustituir...", sustituyendo x por 1 y obteniendo:

```
(%i12) subst(1, x, -(3*x^2-1)/(2*sqrt(x)*(x^2+1)^2));
```

```
(%o12)
```

$$-\frac{1}{4}$$

Otra posibilidad habría sido escribir simplemente

ENTRADA: %o16, x=1

suponiendo que (%o16) sea la etiqueta correspondiente a la expresión de la derivada

- (c) Para estudiar el crecimiento y extremos relativos de $f(x)$ estudiaremos sus puntos críticos. Comenzamos definiendo una función $df(x) := f'(x)$ (lo que nos resultará más fácil si seleccionamos la expresión de la derivada (pulsamos "Editar" → "Selección a la entrada" o utilizamos cortar y pegar o la etiqueta %oN con N adecuado).

```
(%i13) df(x) := -(3*x^2-1)/(2*sqrt(x)*(x^2+1)^2);
```

```
(%o13)
```

$$df(x) := \frac{-(3x^2 - 1)}{2\sqrt{x}(x^2 + 1)^2}$$

Usando el botón [Resolver], obtenemos

```
(%i14) solve([df(x)=0], [x]);
```

```
(%o14)
```

$$\left[x = -\frac{1}{\sqrt{3}}, x = \frac{1}{\sqrt{3}} \right]$$

El primero de estos puntos críticos no está en el dominio de $f(x)$. Ahora podemos dar valores a $f'(x)$ en distintos intervalos, por ejemplo:

```
(%i15) 1/sqrt(3), numer;
```

```
(%o15)
```

0,57735026918962584

```
(%i16) df(0.1);
```

```
(%o16)
```

1,5034846242345494

```
(%i17) df(1);
```

```
(%o17)
```

$$-\frac{1}{4}$$

Así, $f'(x) > 0$ en $(0, 1/\sqrt{3})$ (f creciente) y $f'(x) < 0$ en $(1/\sqrt{3}, +\infty)$ (f decreciente). Por supuesto, tenemos un máximo relativo (y, en este caso, global) en $x = 1/\sqrt{3}$.

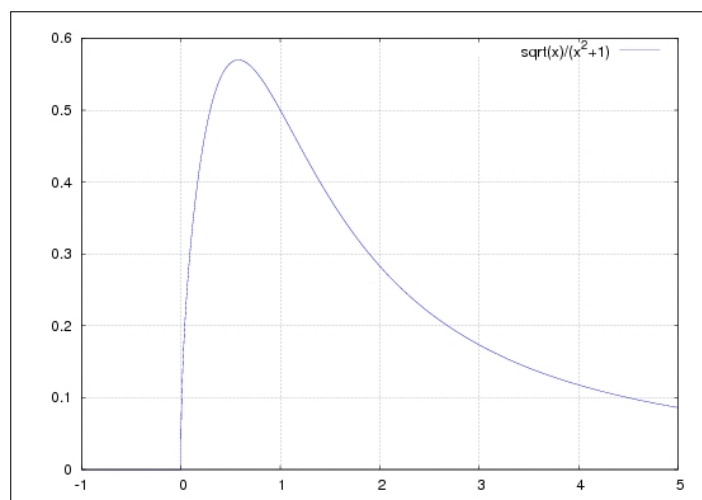


Figura 2.12: Gráfica de la función $f(x) = \frac{\sqrt{x}}{1+x^2}$

- (d) Para representar su gráfica, utilizaremos el botón [Gráficos 2D...]. Escribimos la expresión $f(x)$, el intervalo x que sea de nuestro interés (el intervalo y es opcional) y seleccionamos aquellas opciones que deseemos. El resultado será similar al de la figura 2.12.

2.3.5. Una panorámica de *wxMaxima*

Los ejemplos que se han tratado en el apartado anterior resumen buena parte de la filosofía de *wxMaxima*: por un lado, utilizar el área de entrada (o ventanas de diálogo del tipo *Resolver*) para introducir expresiones algebraicas, utilizando para ello una notación propia del lenguaje *Maxima* que es similar a la de otros lenguajes como *Matlab* o \TeX y que se detallará en próximas secciones. Y por otro lado, emplear los menús y botones disponibles para realizar operaciones sobre estas expresiones.

Como se ha comentado en la sección 2.3.2, éstos se agrupan en tres zonas a las que podemos llamar *área de menús*, *de iconos* y *de botones* ó *atajos*. Los dos últimos constituyen simplemente accesos rápidos a las entradas de menú más utilizadas.

Existen diez menús que contienen alrededor de un centenar de entradas, muchas de las cuales (en concreto las situadas en los menús "Ecuaciones", "Álgebra", "Análisis", "Simplificar", "Gráficos") tienen un carácter específico y serán detalladas en otros capítulos. En las próximas líneas, nos centraremos en estudiar los menús de propósito general relacionadas con el funcionamiento de *Maxima* y *wxMaxima*: "Archivo", "Editar", "Maxima", "Numérico", "Ayuda".

Menú "Archivo"

En él se puede acceder a distintas posibilidades relacionadas con el almacenamiento de información en el disco duro. Algunas de éstas son:

"Abrir sesión" Se puede utilizar para recuperar el trabajo que se haya realizado en una sesión anterior y se haya almacenado con la opción "Guardar Sesión"

"Guardar sesión" Almacenar en el disco duro todo el trabajo que se haya realizado (incluyendo variables, funciones, etc), tal y como se puede observar en el área de salida. El fichero resultante no contiene expresiones en lenguaje *Maxima*, sino en lenguaje Lisp.

"Archivo de lotes" Leer un fichero y procesar sucesivamente las expresiones en lenguaje *Maxima* contenidas en él. Este fichero se puede haber creado manualmente (con un editor de textos apropiado) o mediante el menú "Crear archivo de Lotes"

"Crear archivo de lotes" Volcar el trabajo que se haya realizado hacia un fichero de texto en lenguaje *Maxima*, apto para ser procesado posteriormente.

"Exportar a HTML" Almacenar la sesión en un fichero apropiado para su acceso mediante navegadores web.

"Salir" Abandonar la sesión

Menú "Editar"

"Copiar" Copiar la zona del área de salida (la consola) que esté seleccionada

"Copiar texto" Igual que la opción anterior, pero incluyendo también caracteres de salto de línea.

"Selección a entrada" Copiar en la línea de entrada el texto de la consola que esté seleccionado. Esto mismo se puede conseguir copiando y pegando en la entrada (teclas CTRL+C y CTRL+V).

"Limpiar la pantalla" Borrar los contenidos de la consola (no las variables, funciones, etc).

"Ampliar" y **"Disminuir"** Modificar el tamaño de la fuente que se utiliza para dibujar el contenido de la consola.

"Ir a la entrada" (F4) Situar el cursor en la línea de entrada y seleccionar su contenido (de forma que, si empezamos a escribir, se borrará todo éste).

"Preferencias" Configurar distintas características de *wxMaxima*, entre ellas la forma de ejecutar *Maxima*, el tipo de letra utilizado y el tipo de panel de botones: "Desactivado", "Básico" o "Completo"

Menú "Maxima"

Interactuar con el programa *Maxima* que subyace ejecutándose en el fondo y que *wxMaxima* utiliza para cualquier tipo de cálculo.

"Interrumpir" Detener los cálculos que se estén realizando en este momento

"Reiniciar Maxima" Reiniciar el programa *Maxima* subyacente.

"Limpiar memoria" Eliminar todas las variables que se hayan definido por parte del usuario

"Mostrar funciones" Mostrar las funciones que hayan sido definidas por el usuario

"Mostrar definición" Mostrar la definición de una función de usuario

"Mostrar variables" Mostrar las variables que hayan sido definidas por el usuario

"Borrar función" Eliminar una función de usuario que haya sido previamente definida

"Borrar variable" Eliminar una variable de usuario

"Conmutar pantalla de tiempo" Mostrar, para cada nueva orden, el tiempo que ésta haya tardado en ejecutarse

"Cambiar pantalla 2D" Cambia el algoritmo utilizado para representar expresiones matemáticas en la consola

"Mostrar formato T_EX" Mostrar el código T_EX asociado a la expresión anterior

Menú "Ayuda"

"Ayuda de Maxima" Muestra los libros de ayuda de *Maxima* que estén disponibles en el sistema. Estos libros se pueden navegar de forma interactiva y en cada uno de ellos se pueden realizar búsquedas de palabras clave.

"Describir" Muestra la descripción de cualquier función del sistema *Maxima* (como "factor", "solve", etc)

"Ejemplo" Muestra un ejemplo de uso de cualquier función de *Maxima*

"A propósito" Muestra funciones de *Maxima* similares a una palabra

"Mostrar sugerencia" Muestra una idea relacionada con el uso de *wxMaxima*

"Información de compilación" / "de error" Muestra información técnica relacionada con el funcionamiento del programa.

3 Aritmética

Capítulo

3.1. Trabajando con distintos tipos de números

3.1.1. Números enteros

Maxima (y *wxMaxima*) pueden trabajar con números enteros con cualquier número de dígitos. Pero por defecto, cuando la representación decimal de un número es demasiado larga como para que quepa en una sola línea, *wxMaxima* corrige esta representación eliminando la mayor parte de los dígitos intermedios. Por ejemplo, ante la entrada

ENTRADA:

wxMaxima nos devolvería el siguiente resultado:

```
(%o1) 933262154439441526816992388562[98 digits]916864000000000000000000000000
```

Esto no significa que los 98 dígitos intermedios se hayan perdido, simplemente no aparecen en pantalla. Si realmente estamos interesados en obtener todos los dígitos, simplemente tenemos que seleccionar "Maxima" → "Cambiar pantalla 2D" y elegir el algoritmo "ascii" para las salidas numéricas. De esta forma, el formato de salida será exactamente el mismo que el utilizado por *Maxima* y por otras interfaces gráficas como *xmaxima*, en particular se presentarán todos los dígitos intermedios, obteniendo en el ejemplo anterior:

ENTRADA:

```
(%o2) 933262154439441526816992388562667004907159682643816214685929638952175999
932299156089414639761565182862536979208272237582511852109168640000000000000000
00000000
```

La función "is" se puede utilizar para comprobar si una proposición lógica es cierta (*true*) o falsa (*false*). Por ejemplo:

```
(%i1) is(10! = 9! * 10);
(%o1)
```

true

Ejercicios resueltos y actividades

Se propondrá una tarea en el campus virtual consistente en realizar ejercicios o actividades didácticas relacionadas con este apartado.

Un ejemplo muy sencillo podría ser el siguiente ejercicio:

Pregunta: Con las operaciones suma y producto, obtener el número 100 empleando las nueve cifras 1, 2, 3, 4, 5, 6, 7, 8, 9, sin omitir ni repetir ninguna.

Solución:

```
(%i2) 2*(3*5+1+4+6+7+8+9);
(%o2)
```

100

Seguramente hay más soluciones...

3.1.2. Números racionales

Como se ha comentado, *Maxima* es un programa de cálculo simbólico, capaz de operar con aritmética racional:

```
(%i1) 95!/100!;
(%o1)
```

$$\frac{1}{9034502400}$$

Como se aprecia en el ejemplo anterior, *Maxima* simplifica de forma automática las fracciones numéricas.

Podemos operar utilizando los operadores +, -, *, /, pero siempre teniendo en cuenta que las dos producto y división tienen mayor prioridad que la suma y la resta. En aquellas ocasiones en que deseemos efectuar sumas o restas antes que divisiones o productos, es necesario utilizar los paréntesis, como se muestra en los siguientes ejemplos:

```
(%i2) 3+1/2;
(%o2)
```

$$\frac{7}{2}$$

```
(%i3) (3+1)/2;
(%o3)
```

2

```
(%i4) 1/2+3;
(%o4)
```

$$\frac{7}{2}$$

```
( %i5)  1/(2+3);
( %o5)
```

$$\frac{1}{5}$$

```
( %i6)  1+2*3+4;
( %o6)
```

$$11$$

```
( %i7)  1+2*(3+4);
( %o7)
```

$$15$$

```
( %i8)  (1+2)*(3+4);
( %o8)
```

$$21$$

Ejercicios resueltos y actividades

Se propondrá una tarea en el campus virtual consistente en realizar ejercicios o actividades didácticas relacionadas con este apartado.

3.1.3. Números reales

Existen algunos números reales predefinidos como “%pi”, “%e”, otros números irracionales se obtienen como resultado de funciones como “sqrt” (raíz cuadrada).

```
( %i1)  (%pi+sqrt(2))/%e+1;
( %o1)
```

$$e^{-1} \left(\pi + \sqrt{2} \right) + 1$$

Existen dos formas de representarlos decimalmente:

Números en coma flotante de precisión fija ("float")

Para obtener una representación decimal de un número real¹ podemos optar por una de las siguientes posibilidades:

- Utilizar la función "float"
- Añadir una coma (",") seguida de la orden "numer".
- En el menú de *wxMaxima*, seleccionar "Numérico" → "A real (float)"

El número resultante tendrá 16 cifras decimales:

```
(%i2)  %pi-%e;
(%o2)
```

$$\pi - e$$

```
(%i3)  float(%);
(%o3)
```

0,42331082513074803

```
(%i4)  %pi/6,numer;
(%o4)
```

0,52359877559829882

Si alguno de los números que interviene en una operación se encuentra en representación decimal, el resultado se convertirá automáticamente a este formato:

```
(%i5)  22/7-3.0;
(%o5)
```

0,14285714285714279

Del mismo modo, si a una función se le pasa un argumento en formato de coma flotante, devolverá el resultado en el mismo formato:

```
(%i6)  cos(1);
(%o6)
```

cos 1

```
(%i7)  cos(1.0);
```

¹En realidad, obtenemos un número de coma flotante y precisión fija (IEE 754) en doble precisión (64 bits)

```
(%o7)
```

```
0,54030230586813977
```

Números en coma flotante de precisión no fija (real largo o “bfloat”)

Existen números reales de precisión arbitraria, que podemos fijar con la variable “fpprec” (*float point precision*, por defecto fpprec=16) o mediante el menú “Numerico” → “Establecer precisión...”. Para representar un número real en este formato, podemos optar por una de las siguientes posibilidades:

- Utilizar la función “bfloat”
- En el menú de *wxMaxima*, seleccionar “Numerico” → “A real grande (bigfloat)”

Estos números terminan con los caracteres “bN”, siendo N un número entero que indica que los dígitos anteriores se deben multiplicar por 10^N . Ejemplos:

```
(%i8) fpprec:50;
```

```
(%o8)
```

```
50
```

```
(%i9) bfloat(%pi);
```

```
(%o9)
```

```
3,1415926535897932384626433832795028841971693993751b0
```

```
(%i10) 1000+cos(1b0);
```

```
(%o10)
```

```
1,0005403023058681397174009366074429766037323104206b3
```

Ejercicios resueltos y actividades

Se propondrá una tarea en el campus virtual consistente en realizar ejercicios o actividades didácticas relacionadas con este apartado.

Números complejos

En *Maxima* se puede acceder a la unidad imaginaria a través de la constante “%i”. El resto de los números complejos se representan mediante expresiones de la forma

$$a + b * \%i,$$

donde a y b son números reales. Las funciones “realpart” e “imagpart” nos devuelven, respectivamente, la parte real e imaginaria de un número complejo. A estas funciones (y a otras relacionadas con los números complejos, como “demoivre”), se puede acceder a través del menú “Simplificar” → “Simplificación compleja”

Ejemplos

```
(%i1)  c: 1/2+3*%i;
```

```
(%o1)
```

$$3i + \frac{1}{2}$$

```
(%i2)  realpart(c);
```

```
(%o2)
```

$$\frac{1}{2}$$

```
(%i3)  imagpart(c);
```

```
(%o3)
```

$$3$$

```
(%i4)  exp(c);
```

```
(%o4)
```

$$e^{3i + \frac{1}{2}}$$

```
(%i5)  demoivre(%);
```

```
(%o5)
```

$$\sqrt{e} (i \sin 3 + \cos 3)$$

Ejercicios resueltos y actividades

Se propondrá una tarea en el campus virtual consistente en realizar ejercicios o actividades didácticas relacionadas con este apartado.

3.2. Potencias, radicales y logaritmos. Otras funciones predefinidas

Para calcular potencias en *Maxima* podemos utilizar el operador \wedge , que se puede obtener:

- o bien pulsando las teclas Mayúsculas + \wedge y a continuación Espacio,
- o bien manteniendo pulsada la tecla Mayúsculas y pulsando dos veces la tecla \wedge .

En el caso de potencias de base e , podemos utilizar la función “exp”.

```
(%i1) 1/2+(1/2)^2+(1/2)^3;
(%o1)

$$\frac{7}{8}$$


(%i2) is(10^7=5^7*2^7);
(%o2)
true

(%i3) is(10^7=5^7+5^7);
(%o3)
false

(%i4) (9/2)^3;
(%o4)

$$\frac{729}{8}$$


(%i5) is((a/b)^3=a^3/b^3);
(%o5)
true

(%i6) exp(2)-%e^2;
(%o6)
0
```

Las raíces cuadradas se pueden escribir utilizando la función “sqrt” o bien elevando a $\frac{1}{2}$. En general, para calcular cualquier raíz n -ésima, elevamos a $\frac{1}{n}$.

```
(%i7) r:sqrt(2);
```

```
(%o7)

$$\sqrt{2}$$


(%i8)  fpprec:25;
(%o8)
25

(%i9)  bfloat(r);
(%o9)
1,41421356237309504880168960

(%i10)  2197^(1/3);
(%o10)
13

(%i11)  is(sqrt(4)=sqrt(2)+sqrt(2));
(%o11)
false

(%i12)  is(sqrt(4)=sqrt(2)*sqrt(2));
(%o12)
true
```

Maxima mantiene una filosofía muy conservadora desde el punto de vista de la simplificación de expresiones, en el sentido de que solamente en los casos más evidentes se aplicarán las propiedades de las operaciones aritméticas, potencias, raíces, etc. En el resto de los casos, el usuario deberá solicitar explícitamente el tipo de acción que desee efectuar. En *wxMaxima*, el menú "Simplificar" contiene varias de estas acciones, entre ellas "Simplificar expresión", "Simplificar radicales" (que como indica su nombre, aplicará las propiedades de potencias y radicales), "Factorizar expresión" y "Expandir expresión", a las que también existen accesos rápidos mediante en los botones [Simplificar], [Simplificar (r)], [Factorizar] y [Expandir]. En el siguiente ejemplo, se define una expresión (a la que se llama "expr"), que se simplifica mediante los dos algoritmos (Simplificar y simplificar radicales) y después se vuelve a expandir. Se puede observar en este ejemplo cómo el segundo algoritmo descompone la raíz de a/b en \sqrt{a} y \sqrt{b} (pero el primero no).

```
(%i13)  expr:a/b+sqrt(a/b);
(%o13)

$$\frac{a}{b} + \sqrt{\frac{a}{b}}$$

```

```
(%i14) ratsimp(expr);
(%o14)
```

$$\frac{\sqrt{\frac{a}{b}}b + a}{b}$$

```
(%i15) radcan(expr);
(%o15)
```

$$\frac{\sqrt{a}\sqrt{b} + a}{b}$$

```
(%i16) expand(expr);
(%o16)
```

$$\frac{a}{b} + \sqrt{\frac{a}{b}}$$

Otro ejemplo:

```
(%i17) (a^b)^c;
(%o17)
```

$$(a^b)^c$$

```
(%i18) ratsimp(%);
(%o18)
```

$$(a^b)^c$$

```
(%i19) radcan(%);
(%o19)
```

$$a^{bc}$$

Para calcular logaritmos neperianos disponemos de la función “log”.

```
(%i20) x:%e^10;
(%o20)
```

$$e^{10}$$

```
(%i21) log(x);
(%o21)
```

$$10$$

Maxima no dispone de ninguna función para calcular logaritmos en base arbitraria, aunque la podemos definir de la siguiente forma:

```
(%i22) logb(x,b):=log(x)/log(b)$
(%i23) /* Calculamos \log_{10}(1000) */
      logb(256,2);
(%o23)
```

$$\frac{\log 256}{\log 2}$$

```
(%i24) bfloat(%);
(%o24)
```

8,060

Como se puede observar, en el ejemplo anterior hemos utilizado el operador “:=” para definir una nueva función. Esta es la forma en la que, en *Maxima*, se crean funciones simples definidas por los usuarios.

Esta función sería una buena candidata para ser incorporada en nuestra “biblioteca de funciones particular”, de manera que la tengamos disponible en todas las sesiones de Maxima. Esto se estudiará con más detalle en el tema 6. También nos puede resultar interesante contar con una función específica para logaritmos en base 10 o en base 2:

```
(%i25) log2(x):=logb(x,2)$
(%i26) log10(x):=logb(x,10)$
(%i27) log10(100)+log2(4),numer;
(%o27)
```

4,0

En el contexto de los logaritmos, nos pueden resultar útiles las siguientes entradas del menú de *wxMaxima*: “Simplificar” → “Expandir logaritmos” y “Simplificar” → “Contraer logaritmos”, cuyos resultados se ilustran en el siguiente ejemplo:

```
(%i28) log(a*b);
(%o28)
```

$\log(ab)$

```
(%i29) /* Resultado de expandir logaritmos */
      %, logexpand=super;
(%o29)
```

$\log b + \log a$

```
(%i30) /* Resultado de contraer logaritmos */
      logcontract(%);
(%o30)
```

$$\log(ab)$$

Maxima cuenta con las funciones trigonométricas habituales: “sin”, “cos”, “tan” (seno, coseno, tangente), “asin”, “acos”, “atan” (arco seno, arco coseno, arco tangente), “csc”, “sec”, “cot” (cosecante, secante, cotangente), etc.

```
(%i31) cos(%pi/2);
(%o31)
```

$$0$$

```
(%i32) acos(%);
(%o32)
```

$$\frac{\pi}{2}$$

wxMaxima dispone, dentro del menú “Simplificar” → “Simplificación trigonométrica”, de varias entradas relacionadas con estas funciones: “Simplificar trigonometría”, “Reducir trigonometría”, “Expandir trigonometría” y “Forma canónica”. Estas funciones se encuentran también en los botones [Simplificar (tr)], [Expandir (tr)], [Reducir (tr)], y [FormaCart].

Otras funciones interesantes:

- “abs”, valor absoluto.

- “binomial”:

$$\text{binomial}(m, n) = \binom{m}{n}$$

- ...

Ejercicios resueltos y actividades

Se propondrá una tarea en el campus virtual consistente en realizar ejercicios o actividades didácticas relacionadas con este apartado.

3.3. Divisibilidad

La entrada de menú "Simplificar" → "Factorizar expresión" (también accesible a través del botón [Factorizar]) se puede utilizar para descomponer números enteros como producto de números primos. Se puede utilizar la función "primep" para comprobar si un entero es primo o no. He aquí un ejemplo del resultado de utilizar estas funciones:

```
(%i33) factor(83421);
(%o33)                                     32 13 23 31
```

```
(%i34) primep(521);
(%o34)                                     true
```

Ejercicios resueltos y actividades

Se propondrá una tarea en el campus virtual consistente en realizar ejercicios o actividades didácticas relacionadas con este apartado.

3.4. Actividades

1. La expresión

$$(\%o25) \quad \frac{a - 3b}{(2b - 2a)^3 + (b + a)^3}$$

se puede simplificar como

$$(\%o26) \quad \frac{na + nb}{3b^2 - 6ab + 7a^2}$$

con $x \in R$ adecuado. Indicar el valor de x .

2. El término "googol", que significa 10^{100} , fue acuñado en 1938 por un niño de nueve años, sobrino del matemático estadounidense Edward Kasner, que introdujo este concepto en su libro "Las matemáticas y la imaginación". El motor de búsqueda "Google" fue llamado así debido a este número.

- a) Calcular un número natural, n_1 tal que

$$(n_1 - 1)! < 1\text{googol} < n_1!$$

- ★ **Indicación:** Definir una variable `googol` como 10^{100} y utilizar la función `is` con distintos valores de n hasta, mediante prueba y error, encontrar el número natural que verifique la desigualdad anterior. Desde la línea de entrada de *wxMaxima*, puede ser útil el utilizar la tecla “flecha hacia arriba” para recuperar y modificar la entrada anteriormente tecleada.

b) Calcular un segundo número natural, n_2 tal que

$$e_2^n - 1! < 1\text{googol} < e_2^n!$$

Como se puede observar, n_2 es del orden de las centenas, mucho mayor que n_1 , que se mide en decenas. Por supuesto, esto se debe a que el factorial crece mucho más rápidamente que la función exponencial.

4 Álgebra

Capítulo

4.1. Listas

Como hemos comentado, *Maxima* está escrito en el lenguaje de programación *Lisp*. Este es un lenguaje de programación orientado a listas (de hecho, el nombre *Lisp* proviene de de “List Processing”, proceso de listas). Como resultado, *Maxima* hereda la potencia y versatilidad en el manejo de listas, que se utilizan de forma nativa como bloques de construcción de conjuntos de valores numéricos o simbólicos.

En *Maxima* las listas se pueden definir como series de expresiones separadas por comas y encerradas entre corchetes.

```
(%i1) mi_lista:[0, 1+3/2, x+y^2];  
(%o1)
```

$$\left[0, \frac{5}{2}, y^2 + x\right]$$

Se puede acceder a los elementos de una lista mediante las funciones “first”, “second”, ..., “tenth” y “last”.

```
(%i2) first(mi_lista);  
(%o2)
```

$$0$$

```
(%i3) second(mi_lista);  
(%o3)
```

$$\frac{5}{2}$$

```
(%i4) last(mi_lista);  
(%o4)
```

$$y^2 + x$$

O bien, mediante corchetes se puede indexar el elemento n -ésimo:

```
(%i5) mi_lista[1];  
(%o5)
```

$$0$$

```
(%i6)  mi_lista[2];
(%o6)
```

$$\frac{5}{2}$$

```
(%i7)  mi_lista[3];
(%o7)
```

$$y^2 + x$$

Las operaciones aritméticas y básicas (incluyendo potencias y radicales) pueden actuar sobre listas, aunque no así funciones trascendentes como logaritmos o funciones trigonométricas.

```
(%i8)  l:[1,2,3,4]$
(%i9)  s:[1,-2,3,-4]$
(%i10) l+s;
(%o10)
```

$$[2, 0, 6, 0]$$

```
(%i11) sqrt(2^s);
(%o11)
```

$$\left[\sqrt{2}, \frac{1}{2}, 2\sqrt{2}, \frac{1}{4} \right]$$

```
(%i12) sin(s);
(%o12)
```

$$\sin [1, -2, 3, -4]$$

Las listas se pueden manipular de muchas formas, por ejemplo la función “delete” devuelve la lista resultante de eliminar un elemento y “cons” devuelve la que resulta al añadir un elemento al principio de una lista. Se pueden unir dos listas mediante la función “append”.

```
(%i13) l1:delete(5/2,mi_lista);
(%o13)
```

$$[0, y^2 + x]$$

```
(%i14) l2:cons(%pi,mi_lista);
(%o14)
```

$$\left[\pi, 0, \frac{5}{2}, y^2 + x \right]$$

```
(%i15) append(l1,l2);
```

```
(%o15)
```

$$\left[0, y^2 + x, \pi, 0, \frac{5}{2}, y^2 + x\right]$$

Aunque no profundizaremos en este asunto (pues existe mucha más información en la web de *Maxima* o a través del sistema de ayuda de *wxMaxima* "Ayuda" → "Ayuda de Maxima", desde donde se puede acceder al manual de *Maxima* en Español que contiene un capítulo dedicado a listas), a continuación estudiaremos algunas otras funciones que pueden ser interesantes.

Mediante la función "makelist". Podemos construir una listas formadas por un número finito de expresiones que se ajusten a un término general.

```
(%i16) makelist(2^k,k,0,10);
```

```
(%o16)
```

[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

Esta función está disponible en *wxMaxima* a través del menú "Algebra" → "Construir lista..." . Como se aprecia, sus parámetros son, respectivamente, el término general, la variable utilizada como índice y dos números enteros que expresan el rango en que varía este índice de forma consecutiva¹.

Mediante "apply" (o, en *wxMaxima*, mediante el menú "Algebra" → "Aplicar a lista..."), podemos aplicar una determinada función a todos los elementos de una lista:

```
(%i17) f(x):=x^2$
```

```
(%i18) l:makelist(f(k),k,1,20);
```

```
(%o18)
```

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]

```
(%i19) apply(min,l);
```

```
(%o19)
```

1

```
(%i20) apply(max,l);
```

```
(%o20)
```

400

¹Existe una variante de "makelist" en la que, después del término general y la variable, se le pasa una lista en la que se indican de forma explícita los valores (no necesariamente consecutivos, ni enteros) que tomará esta variable índice.


```
(%i21)  apply("+",1);
(%o21)
```

2870

Por supuesto, para poder aplicar una función a todos los elementos de una lista, esa función debe admitir todos esos parámetros de forma simultánea, por ejemplo la siguiente expresión es errónea porque la función coseno admite un único argumento (no tiene sentido $\cos(1, 2, 3, 4)$):

```
(%i22)  l:[1,2,3,4]$
(%i23)  apply(cos, l);
Wrong number of arguments to cos
- an error.  Quitting.  To debug this try debugmode(true);
```

Por último, dada una lista, la función “map” (disponible en *wxMaxima* a través del menú “Algebra” → “Corresponder a lista”) se utiliza para construir una nueva lista formada por el valor de una función sobre cada uno de los elementos de la lista original.

```
(%i24)  x:makelist(k*%pi/2,k,1,10);
(%o24)
```

$$\left[\frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi, \frac{5\pi}{2}, 3\pi, \frac{7\pi}{2}, 4\pi, \frac{9\pi}{2}, 5\pi \right]$$

```
(%i25)  map(cos,x);
(%o25)
```

$$[0, -1, 0, 1, 0, -1, 0, 1, 0, -1]$$

La función “map” es muy adecuada para construir tablas de valores en aquellos casos en los que la función no esté diseñada para actuar directamente sobre listas.

4.2. Polinomios y fracciones algebraicas

Podemos definir polinomios y utilizar con ellos las operaciones habituales:

```
(%i1)  p1:x^2-2;
(%o1)
```

$$x^2 - 2$$

```
(%i2)  p2:2*x^3+x^2-3*x-1;
(%o2)
```

$$2x^3 + x^2 - 3x - 1$$

```
(%i3) 2*p1+p2;
```

```
(%o3)
```

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

```
(%i4) p1*p2;
```

```
(%o4)
```

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

La función “divide” (a la que se puede acceder desde el menú “Análisis” → “Dividir polinomios...” de *wxMaxima*) devuelve una lista formada por el cociente y el resto de la división de dos polinomios. Las funciones “gcd” y “lcm” (siglas de las expresiones *greatest common divisor* y *least common multiple*) devuelven, respectivamente, el máximo común divisor y el mínimo común múltiplo de dos polinomios (o de dos números). En *wxMaxima*, se puede acceder a las mismas a través de “Análisis” → “Máximo común divisor...” y “Análisis” → “Mínimo común múltiplo...”

```
(%i5) divide(p2,p1);
```

```
(%o5)
```

$$[2x + 1, x + 1]$$

Para dar valores a un polinomio, existen varias posibilidades: podemos pedir directamente a *Maxima* que sustituya la variable por el valor deseado, mediante el operador coma (“,”) o mediante la función “subst”, que está disponible en *wxMaxima* mediante el menú “Simplificar” → “Sustituir...”). O bien podemos definir una función polinómica (mediante el operador “:=”) y darle valores como a cualquier otra función.

```
(%i6) p2,x=1;
```

```
(%o6)
```

$$-1$$

```
(%i7) subst(1,x,p2);
```

```
(%o7)
```

$$-1$$

```
(%i8) f(x):=2*x^3+x^2-3*x-1;
```

```
(%o8)
```

$$f(x) := 2x^3 + x^2 + (-3)x - 1$$

```
(%i9) f(1);
```

```
(%o9)
```

$$-1$$

```
(%i10) map(f, [-2, -1, 0, 1, 2]);
```

```
(%o10)
```

$$[-7, 1, -1, -1, 13]$$

Para calcular ceros de un polinomio, podemos utilizar la función “solve” (a la que se puede acceder desde *wxMaxima* mediante el menú “Ecuaciones” → “Resolver...” o mediante el botón [Resolver...]), proporcionando la ecuación y la variable sobre la que queremos resolver (en caso de que exista una única variable, esta puede omitirse). Obtendremos una lista formada por todos los ceros. Si, en lugar de una ecuación, introducemos alguna otra expresión, se intentará resolver la ecuación que resulta al igualarla a cero.

```
(%i11) p:x^3+x^2+x+1;
```

```
(%o11)
```

$$x^3 + x^2 + x + 1$$

```
(%i12) solve(p);
```

```
(%o12)
```

$$[x = -i, x = i, x = -1]$$

Podemos contar también con las funciones “realroots” y “allroots” (accesibles desde las entradas “Ecuaciones” → “Raíces reales” y “Ecuaciones” → “Raíces de un polinomio” del menú de *wxMaxima*). La primera de ellas devuelve solamente las raíces reales de un polinomio. La segunda las calcula de forma numérica las raíces reales y complejas.

```
(%i13) allroots(p);
```

```
(%o13)
```

$$[x = 1,0 i + 2,6237694926444074 \times 10^{-17}, x = 2,6237694926444074 \times 10^{-17} - 1,0 i, x = -1,0]$$

```
(%i14) realroots(p);
```

```
(%o14)
```

$$[x = -1]$$

Aunque el cálculo de raíces de forma numérica pueda significar la introducción de (pequeños) errores, esto es inevitable (obviamente) en la mayor parte de los polinomios de grado mayor que cuatro.

Para factorizar polinomios, como cualquier otra expresión, contamos con la función “factor” (disponible en *wxMaxima* a través del menú “Simplificar” → “Factorizar expresión” o en el botón [Factorizar]). Inversamente, podemos utilizar la función “expandir” (menú “Simplificar” → “Expandir expresión” o botón [Expandir]) para desarrollar un polinomio como suma de monomios. Existen muchas otras funciones relacionadas con los polinomios, que se pueden consultar en la documentación de *Maxima* (que es accesible, en español, desde la ayuda de *wxMaxima*).

```
(%i15) factor(p);
```

```
(%o15)
```

$$(x + 1) (x^2 + 1)$$

```
(%i16) expand((x+a)^5);
```

```
(%o16)
```

$$x^5 + 5 a x^4 + 10 a^2 x^3 + 10 a^3 x^2 + 5 a^4 x + a^5$$

```
(%i17) expand((x-a)*(x-b)*(x-c));
```

```
(%o17)
```

$$x^3 - c x^2 - b x^2 - a x^2 + b c x + a c x + a b x - a b c$$

```
(%i18) solve(%,x);
```

```
(%o18)
```

$$[x = c, x = b, x = a]$$

En *Maxima*, obtendremos una fracción algebraica cuando utilicemos el operador de división (“/”) entre dos polinomios. La mayor parte de las funciones que hemos visto anteriormente seguirán siendo aplicables y nos resultarán de utilidad. Además, nos pueden ser de utilidad las funciones “ratsimp” y “radcan”, que se estudiaron en el capítulo anterior (disponibles, respectivamente, en los menús “Simplificar” → “Simplificar expresión”, “Simplificar” → “Simplificar radicales” y en los botones [Simplificar] y [Simplificar (r)] de *wxMaxima*).

```
(%i1) r: (x^3+x^2+x+1)/(x+1);
```

```
(%o1)
```

$$\frac{x^3 + x^2 + x + 1}{x + 1}$$

```
(%i2) ratsimp(%);
```

```

(%o2)

$$x^2 + 1$$


(%i3)  % - x^2/(x^2-9);
(%o3)

$$-\frac{x^2}{x^2 - 9} + x^2 + 1$$


(%i4)  s:ratsimp(%);
(%o4)

$$\frac{x^4 - 9x^2 - 9}{x^2 - 9}$$


(%i5)  factor(s);
(%o5)

$$\frac{x^4 - 9x^2 - 9}{(x - 3)(x + 3)}$$


(%i6)  denom(s);
(%o6)

$$x^2 - 9$$


(%i7)  num(s);
(%o7)

$$x^4 - 9x^2 - 9$$


(%i8)  %, x=a+b;
(%o8)

$$(b + a)^4 - 9(b + a)^2 - 9$$


(%i9)  expand(%);
(%o9)

$$b^4 + 4ab^3 + 6a^2b^2 - 9b^2 + 4a^3b - 18ab + a^4 - 9a^2 - 9$$


(%i10) s, x=3;
      Division by 0
      - an error.  Quitting.  To debug this try debugmode(true);

```

Como se puede apreciar, se han utilizado las funciones “num” y “denom” para acceder al numerador y denominador de las fracciones anteriores. Y, por supuesto, si intentamos dividir por cero *Maxima* nos advertirá de nuestro error.

4.3. Ecuaciones, sistemas de ecuaciones e inecuaciones

Como sabemos, las ecuaciones se crean en Maxima utilizando el operador igual ("="). Para acceder al primer y al segundo miembro se utilizan, respectivamente, las funciones "lhs" y "rhs" (en inglés, iniciales de "left" y "right hand side", que podemos traducir como lado izquierdo y derecho). Pero también se pueden utilizar las funciones "first" y "second" para este propósito².

```
(%i1) expand((x-1)*(x^2+7));
```

```
(%o1)
```

$$x^3 - x^2 + 7x - 7$$

```
(%i2) ec:x^2+7=(7+x^2)/x;
```

```
(%o2)
```

$$x^2 + 7 = \frac{x^2 + 7}{x}$$

```
(%i3) first(ec);
```

```
(%o3)
```

$$x^2 + 7$$

```
(%i4) second(ec);
```

```
(%o4)
```

$$\frac{x^2 + 7}{x}$$

```
(%i5) is(lhs(ec)=first(ec));
```

```
(%o5)
```

true

A las ecuaciones se les pueden sumar y multiplicar expresiones simbólicas (y también se pueden sumar ecuaciones entre sí):

```
(%i6) ec*x;
```

```
(%o6)
```

$$x(x^2 + 7) = x^2 + 7$$

²Como ya sabemos, estas funciones se utilizan para obtener elementos de una lista. En realidad, su uso en el contexto de las ecuaciones es coherente con lo anterior, pues Maxima almacena internamente las ecuaciones como una lista compuesta por sus dos miembros. Por el mismo motivo, si utilizamos las funciones "first" y "second" sobre una fracción, obtendremos el numerador y el denominador, al utilizarlo sobre un polinomio que esté expresado como suma de monomios, se obtendrán éstos a partir del de mayor grado, etc.

```
(%i7)  %-(x^2+7);
```

```
(%o7)
```

$$x(x^2 + 7) - x^2 - 7 = 0$$

```
(%i8)  expand(%);
```

```
(%o8)
```

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

Como sabemos, para su resolución podemos utilizar la función “solve” (menú “Ecuaciones, Resolver...” o botón [Resolver] en *wxMaxima*). De cualquier manera, existirán muchas ocasiones en las que no es posible obtener una solución de forma simbólica. En estos casos, podemos optar por la resolución numérica, por ejemplo mediante la función “find_root” (a la que podemos acceder a través de la entrada “Ecuaciones” → “Resolver numéricamente...”, introduciendo como parámetro la ecuación y la variable respecto a la cual resolver, además de una cota inferior y superior de la solución³

```
(%i9)  ec:exp(-x)=x;
```

```
(%o9)
```

$$e^{-x} = x$$

```
(%i10) solve(ec);
```

```
(%o10)
```

$$[x = e^{-x}]$$

```
(%i11) find_root("ec,x,-1,1);
```

```
(%o11)
```

$$0,56714329040978384$$

Los sistemas de ecuaciones se escriben, simplemente, como listas de ecuaciones y, en el caso de sistemas lineales, se pueden resolver mediante la orden “solve” (teniendo en cuenta que, en caso de ambigüedad, se debe explicitar la lista de las variables a resolver):

```
(%i12) s1:[a*x+y=a, x+a*y=a];
```

```
(%o12)
```

$$[y + ax = a, ay + x = a]$$

³Según el manual de *Maxima*, el método que se aplica es una búsqueda binaria en el rango especificado por los dos últimos argumentos. Cuando el algoritmo considera que la función está lo suficientemente cerca de ser lineal, comienza a utilizar interpolación lineal.

```
(%i13) solve(s1,[x,y]);
```

```
(%o13)
```

$$\left[\left[x = \frac{a}{a+1}, y = \frac{a}{a+1} \right] \right]$$

```
(%i14) solve([x+y-z=0, x-2*y+3*z=0, 2*x - y + 2*z = 0]);
```

```
Dependent equations eliminated: (3)
```

```
(%o14)
```

$$[[z = -3 \%r1, y = -4 \%r1, x = \%r1]]$$

Como se puede observar, los sistemas indeterminados se resuelven mediante la introducción de un parámetro, que se denota de la forma $\%rN$, siendo N un número natural que lo identifica de forma única.

En cuanto a ecuaciones y sistemas no lineales, “solve” intenta resolverlos de forma simbólica, pero cuando no es posible, entra en funcionamiento (sin que el usuario sea consciente de ello) una función llamada “algsys”, que trata de obtener de forma numérica las soluciones. Cuando los sistemas (y ecuaciones) no lineales son demasiado complicados, “algsys” no consigue su objetivo y devolverá un mensaje de error quejándose de su fracaso. Los tres ejemplos siguientes ilustran todas las posibilidades (resolución simbólica, numérica y no resolución):

```
(%i15) solve([x^4-y^2=1, x^2+y=2]);
```

```
(%o15)
```

$$\left[\left[y = \frac{3}{4}, x = -\frac{\sqrt{5}}{2} \right], \left[y = \frac{3}{4}, x = \frac{\sqrt{5}}{2} \right] \right]$$

```
(%i16) solve([x^6-y^3=1, x^2+y^2=2]);
```

```
(%o16)
```

$$[[y = 0,89565802760013469, x = -1,0944391179290509], [y = 0,89565802760013469, x = 1,0944391179290509]]$$

```
(%i17) solve([sin(x)+y=0, x+y^2=1]);
```

```
`algsys' cannot solve - system too complicated.
```

```
- an error. Quitting. To debug this try debugmode(true);
```

Por último, aunque no existe una función específica para la resolución de inecuaciones, las funciones que ya conocemos nos pueden ser, con un poco de perspicacia, de utilidad. Por ejemplo, supongamos que queremos resolver una sencilla inecuación de segundo grado:

$$x^2 - 8x + 15 > 0$$


```
(%i1) p(x):=x^2-8*x+15;
```

```
(%o1)
```

$$p(x) := x^2 - 8x + 15$$

```
(%i2) solve(p(x)=0);
```

```
(%o2)
```

$$[x = 3, x = 5]$$

```
(%i3) p([2,4,6]);
```

```
(%o3)
```

$$[3, -1, 3]$$

Según lo anterior, podemos deducir que $x^2 - 8x + 15 > 0$ en $(-\infty, 3) \cup (5, +\infty)$ y $x^2 - 8x + 15 < 0$ en $(3, 5)$.

Si queremos profundizar más, podemos utilizar las funciones “assume”, “facts” y “forget”. La primera de ellas informa a *Maxima* de una condición lógica (o varias) que asumiremos como válida. La segunda de ella, nos lista las condiciones lógicas que se han asumido hasta el momento y la tercera elimina una de ellas.

```
(%i4) p(x):=x^2-8*x+15;
```

```
(%o4)
```

$$p(x) := x^2 - 8x + 15$$

```
(%i5) is(p(x)>0);
```

```
Maxima was unable to evaluate the predicate:
```

```
** error while printing error message **
```

```
Maxima was unable to evaluate the predicate:~%~M
```

```
- an error. Quitting. To debug this try debugmode(true);
```

```
(%i6) solve(p(x)=0);
```

```
(%o6)
```

$$[x = 3, x = 5]$$

```
(%i7) assume(x<3);
```

```
(%o7)
```

$$[x < 3]$$

```
(%i8) is(p(x)>0);
```

```
(%o8)
```

true

```
(%i9) facts();
```

```
(%o9)
```

$$[3 > x]$$

```
(%i10) forget(x<3);
(%o10)
```

$$[x < 3]$$

```
(%i11) assume(3<x, x<5) $
(%i12) is(p(x)<0);
(%o12)
```

true

```
(%i13) forget(3<x, x<5) $
(%i14) assume(x>5) $
(%i15) is(p(x)>0);
(%o15)
```

true

```
(%i16) forget(x>5) $
(%i17) facts();
(%o17)
```

$$[]$$

4.4. Matrices y determinantes

Para introducir Matrices, en *Maxima* se utiliza la función “`matrix`”, que recibe como argumentos una serie de listas que definen las filas de la matriz. Para acceder a esta función, *wxMaxima* nos ofrece una interfaz muy atractiva, a la que se puede acceder a través del menú “Algebra” → “Introducir matriz...”. *wxMaxima* nos preguntará el número de filas y columnas, así como el formato de la matriz resultante (de tipo general, diagonal, simétrico o antisimétrico) y a continuación nos ofrecerá una ventana (figura 4.4) en la que podremos introducir sus elementos.

Para acceder a los elementos de una matriz, utilizaremos (al igual que las listas) los corchetes.

```
(%i1) A:matrix([1,0,-2], [2,3,2], [-2,0,1]);
(%o1)
```

$$\begin{pmatrix} 1 & 0 & -2 \\ 2 & 3 & 2 \\ -2 & 0 & 1 \end{pmatrix}$$

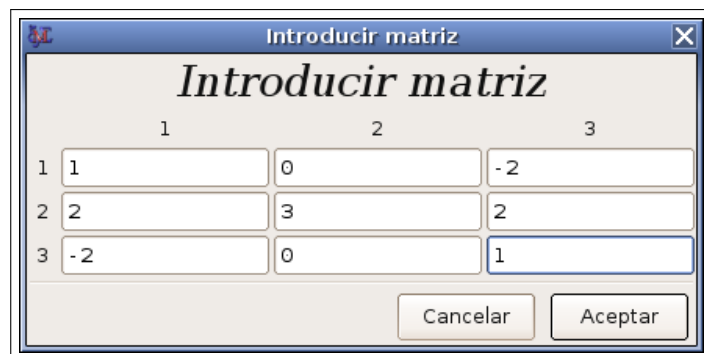


Figura 4.1: Introducción del contenido de una matriz 3×3 en *wxMaxima*

```
(%i2) A[2,3];
(%o2)
```

2

Es bueno recordar que siempre que deseemos etiquetar con un nombre, a una matriz creada a través de la interfaz gráfica de *wxMaxima* (como, en el ejemplo anterior, *A*), podemos escribir en la línea de entrada

ENTRADA: A:

y a continuación pulsar en el menú, en este caso "Álgebra" → "Introducir matriz..." .

Existe una segunda forma de definir matrices cuyos elementos se ajusten a una regla predefinida. Para ello debemos predefinir esta regla, de forma que defina una "tabla", que asocie a cada fila y columna (entre corchetes) un número real, como en el siguiente ejemplo:

```
(%i3) T[i,j] := i+j-1;
(%o3)
```

$$T_{i,j} := i + j - 1$$

Obsérvese que, como la definición de funciones, hemos utilizado el operador " := "

A continuación, podemos usar la función "genmatrix", a la que se puede acceder a través de "Álgebra" → "Generar matriz..." en *wxMaxima*. Esta función nos pedirá el nombre de la tabla, así como el número de filas y columnas de la matriz resultante, por ejemplo:

```
(%i4) B:genmatrix(T,3,5);
(%o4)
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

Utilizando “`submatrix`” se obtienen submatrices, para lo cual introducimos las filas a eliminar (si existe alguna), a continuación el nombre de la matriz y por último las columnas a eliminar (en su caso). Las funciones “`row`” y “`col`” nos permiten acceder, respectivamente, a filas y columnas que deseemos. Se pueden añadir filas y columnas nuevas con “`addrow`” y “`addcol`”, todo lo cual muestran los siguientes ejemplos:

```
(%i5) submatrix(1,B,4,5);
```

```
(%o5)
```

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

```
(%i6) row(B,1);
```

```
(%o6)
```

$$(1 \ 2 \ 3 \ 4 \ 5)$$

```
(%i7) col(B,5);
```

```
(%o7)
```

$$\begin{pmatrix} 5 \\ 6 \\ 7 \end{pmatrix}$$

```
(%i8) addcol(B,[10,11,12]);
```

```
(%o8)
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 10 \\ 2 & 3 & 4 & 5 & 6 & 11 \\ 3 & 4 & 5 & 6 & 7 & 12 \end{pmatrix}$$

Puesto que Maxima es un programa de cálculo simbólico, las matrices no se limitan (como ocurre en programas más orientados al cálculo numérico y matricial como Octave, también con licencia libre) a valores numéricos, sino que estos pueden venir dados por cualquier tipo de expresión:

```
(%i9) C:matrix([x^2, 1+x/y], [sqrt(x), x^2-y=0]);
```

```
(%o9)
```

$$\begin{pmatrix} x^2 & \frac{x}{y} + 1 \\ \sqrt{x} & x^2 - y = 0 \end{pmatrix}$$

```
(%i10) solve(C[2,2],x);
```

```
(%o10)
```

$$[x = -\sqrt{y}, x = \sqrt{y}]$$

```
(%i11) C,x=2,y=4;
```

```
(%o11)
```

$$\begin{pmatrix} 4 & \frac{3}{2} \\ \sqrt{2} & 0 \end{pmatrix}$$

Todas las operaciones algebraicas están disponibles para matrices, aunque debemos tener en cuenta que el operador asterisco ("*") se interpreta como producto elemento a elemento, mientras que el producto matricial viene dado a través del operador punto ("."). Además, el operador "^" significa calcular las potencias de los elementos de una matriz, mientras que "^^" se utiliza para calcular potencias de matrices.

```
(%i12) A:matrix([a,b,c],[d,e,f]);
```

```
(%o12)
```

$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$

```
(%i13) B:matrix([u,v,w],[x,y,z]);
```

```
(%o13)
```

$$\begin{pmatrix} u & v & w \\ x & y & z \end{pmatrix}$$

```
(%i14) A+B;
```

```
(%o14)
```

$$\begin{pmatrix} u+a & v+b & w+c \\ x+d & y+e & z+f \end{pmatrix}$$

```
(%i15) 2*A;
```

```
(%o15)
```

$$\begin{pmatrix} 2a & 2b & 2c \\ 2d & 2e & 2f \end{pmatrix}$$

```
(%i16) A*B;
```

```
(%o16)
```

$$\begin{pmatrix} au & bv & cw \\ dx & ey & fz \end{pmatrix}$$

```
(%i17) C:submatrix(B,3);
```

```
(%o17)
```

$$\begin{pmatrix} u & v \\ x & y \end{pmatrix}$$

```
(%i18) C.A;
```

```
(%o18)
```

$$\begin{pmatrix} dv+au & ev+bu & fv+cu \\ dy+ax & ey+bx & fy+cx \end{pmatrix}$$

```
(%i19) A^n;
```

```
(%o19)
```

$$\begin{pmatrix} a^n & b^n & c^n \\ d^n & e^n & f^n \end{pmatrix}$$

```
(%i20) %, n=1/2;
```

```
(%o20)
```

$$\begin{pmatrix} \sqrt{a} & \sqrt{b} & \sqrt{c} \\ \sqrt{d} & \sqrt{e} & \sqrt{f} \end{pmatrix}$$

```
(%i21) C^2;
```

```
(%o21)
```

$$\begin{pmatrix} vx + u^2 & vy + uv \\ xy + ux & y^2 + vx \end{pmatrix}$$

Otras funciones aplicables sobre matrices:

- “diagmatrix” y “zeromatrix”, se pueden utilizar para construir, respectivamente matrices diagonales (con todos sus elementos diagonales iguales entre sí) y matrices nulas.
- “transpose”, devuelve la matriz traspuesta (disponible en *wxMaxima* a través de “Algebra” → “Trasponer matriz”)
- “determinant”, calcula el determinante de una matriz cuadrada (menú “Algebra” → “Determinante”)
- “rank”, calcula el rango
- “invert”, devuelve la matriz inversa (menú “Algebra” → “Invertir matriz”)
- “triangularize”, devuelve una matriz triangular superior resultante de aplicar el método de Gauss a una matriz dada
- “eigenvalues”, devuelve dos listas, la primer formada por los autovalores de una matriz y la segunda por sus multiplicidades (accesible desde *wxMaxima* en “Algebra” → “Valores propios”)
- “eigenvectors”, devuelve una lista formada por los autovalores junto a una serie de listas representando a sus autovectores asociados (menú “Algebra” → “Vectores propios” de *wxMaxima*)

Ejemplos:

```
(%i22) diagmatrix(3,%pi^2);
```

```
(%o22)
```

$$\begin{pmatrix} \pi^2 & 0 & 0 \\ 0 & \pi^2 & 0 \\ 0 & 0 & \pi^2 \end{pmatrix}$$

```
(%i23) zeromatrix(3,4);
```

```
(%o23)
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
(%i24) A:matrix([1,0,-2], [2,3,2], [-2,0,1]);
```

```
(%o24)
```

$$\begin{pmatrix} 1 & 0 & -2 \\ 2 & 3 & 2 \\ -2 & 0 & 1 \end{pmatrix}$$

```
(%i25) determinant(A);
```

```
(%o25)
```

$$-9$$

```
(%i26) invert(A);
```

```
(%o26)
```

$$\begin{pmatrix} -\frac{1}{3} & 0 & -\frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & \frac{2}{3} \\ -\frac{2}{3} & 0 & -\frac{1}{3} \end{pmatrix}$$

```
(%i27) triangularize(A);
```

```
(%o27)
```

$$\begin{pmatrix} 1 & 0 & -2 \\ 0 & 3 & 6 \\ 0 & 0 & -9 \end{pmatrix}$$

```
(%i28) I:diagmatrix(3,1);
```

```
(%o28)
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
(%i29) M:A-x*I;
```

```
(%o29)
```

$$\begin{pmatrix} 1-x & 0 & -2 \\ 2 & 3-x & 2 \\ -2 & 0 & 1-x \end{pmatrix}$$

```
(%i30) solve(determinant(M)=0) /* Autovalores */;
```

```
(%o30)
```

$$[x = 3, x = -1]$$

```
(%i31) M, x=3;
```

```
(%o31)
```

$$\begin{pmatrix} -2 & 0 & -2 \\ 2 & 0 & 2 \\ -2 & 0 & -2 \end{pmatrix}$$

```
(%i32) rank(%);
```

```
(%o32)
```

$$1$$

```
(%i33) eigenvalues(A) /* [autovalores], [multiplicidades] */;
```

```
(%o33)
```

$$[[3, -1], [2, 1]]$$

```
(%i34) eigenvectors(A) /* [autovalores], [v1], [v2], [v3] */;
```

```
(%o34)
```

$$[[[3, -1], [2, 1]], [1, 0, -1], [0, 1, 0], [1, -1, 1]]$$

5

Capítulo

Cálculo Infinitesimal

5.1. Sucesiones y límite

En *Maxima*, las sucesiones se definen, a través de su término general o mediante una expresión de recurrencia, utilizando el operador “:=”, e introduciendo la variable índice entre corchetes (“[]”). En las definiciones por recurrencia, deberemos utilizar el operador “:” para asignar valores a los primeros elementos.

```
(%i1) a[n]:=1/(2*n-1);  
(%o1)
```

$$a_n := \frac{1}{2n - 1}$$

```
(%i2) a[1];  
(%o2)
```

1

```
(%i3) makelist(a[k],k,1,10);  
(%o3)
```

$$\left[1, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}, \frac{1}{11}, \frac{1}{13}, \frac{1}{15}, \frac{1}{17}, \frac{1}{19}\right]$$

```
(%i4) b[1]:1;  
(%o4)
```

1

```
(%i5) b[n]:=2*b[n-1];  
(%o5)
```

$$b_n := 2 b_{n-1}$$

```
(%i6) makelist(b[k],k,1,10);  
(%o6)
```

$$[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]$$

Otro ejemplo: definición de la sucesión de Fibonacci mediante recurrencia y mediante su término general¹

```
(%i7) fib1[1]:1;
```

```
(%o7) 1
(%i8) fib1[2]:1;
```

```
(%o8) 1
(%i9) fib1[n]:=fib1[n-1]+fib1[n-2];
```

```
(%o9) fib1_n := fib1_{n-1} + fib1_{n-2}
(%i10) l1:makelist(fib1[n],n,1,10);
```

```
(%o10) [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
(%i11) phi:(1+sqrt(5))/2 /* RAZON AUREA */;
```

```
(%o11) (sqrt(5) + 1) / 2
(%i12) fib2[n]:=radcan((phi^n-(1-phi)^n)/sqrt(5));
```

```
(%o12) fib2_n := radcan((phi^n - (1 - phi)^n) / sqrt(5))
(%i13) l2:makelist(fib2[n],n,1,10);
```

```
(%o13) [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Para Calcular límites usamos la función “limit”, que está disponible en el menú “Análisis” → “Calcular límite...” de *wxMaxima*. Como parámetros de esta función, tenemos que teclear la expresión de una sucesión, el nombre de la variable respecto de

¹Aunque esta sucesión está predefinida en *Maxima* mediante la función “fib(n)”

la cual deseamos calcular el límite y el valor hacia el que ésta tiende. En *Maxima* existen dos variables especiales, “inf” y “minf”, que representan respectivamente a $+\infty$ y $-\infty$.

```
(%i14) a[n] := (n-sqrt(n^2-4))/(n+1);
```

```
(%o14) a := 
$$\frac{n - \sqrt{n^2 - 4}}{n + 1}$$

```

```
(%i15) limit(a[n],n,inf);
```

```
(%o15) 0
```

```
(%i16) limit(n/sqrt(n),n,inf);
```

```
(%o16) inf
```

```
(%i17) limit(1/(sqrt(n)-sqrt(n+1)),n,inf);
```

```
(%o17) minf
```

```
(%i18) limit(((n-2)/(n-3))^(2*n),n,inf);
```

```
(%o18) %e2
```

```
(%i19) limit((-1)^n,n,inf);
```

```
(%o19) ind
```

```
(%i20) limit(n*(-1)^n,n,inf);
```

```
(%o20) und
```

Como se puede observar, ante el cálculo del límite de $(-1)^n$, *Maxima* devuelve la palabra clave “ind”, que se puede interpretar como indefinido pero acotado, mientras que ante el límite $n(-1)^n$ responde “und” (de *undefined*) para indicar que el límite es indefinido y no necesariamente acotado.

En algunas ocasiones, *Maxima* no tendrá suficiente información para realizar el cálculo requerido y nos solicitará información adicional, que tendremos que teclear:

```
(%i21) limit(a*n^2,n,inf);
      Is a positive, negative, or zero?
      negative;
(%o21)
```

$$-\infty$$

Todo lo anterior es aplicable para el cálculo de límites en un punto. La función “limit” admite, además, un cuarto parámetro que podemos utilizar para calcular límites laterales. Si utilizamos el menú “Análisis” → “Calcular límite...” de *wxMaxima*, podemos utilizar el ratón para seleccionar límite desde la derecha o desde la izquierda. Si preferimos teclear directamente la entrada, deberemos utilizar las palabras clave “plus” (+) o “minus” (−) para hallar límites desde la derecha y desde la izquierda, respectivamente.

```
(%i22) limit(sin(x)/x,x,0);
(%o22)
```

$$1$$

```
(%i23) limit(x/(x-1)^2,x,1);
(%o23)
```

$$\infty$$

```
(%i24) limit(1/(x-2),x,2);
(%o24)
```

$$\text{und}$$

```
(%i25) limit(1/(x-2),x,2,plus);
(%o25)
```

$$\infty$$

```
(%i26) limit(1/(x-2),x,2,minus);
(%o26)
```

$$-\infty$$

FUNCIÓN <i>Maxima</i>	DESCRIPCIÓN
<code>abs(x)</code>	Valor absoluto de x
<code>acos(x)</code>	Arco coseno de x
<code>acosh(x)</code>	Arco coseno hiperbólico de x
<code>asin(x)</code>	Arco seno de x
<code>asinh(x)</code>	Arco seno hiperbólico de x
<code>atan(x)</code>	Arco tangente de x
<code>atanh(x)</code>	Arco tangente hierbólica de x
<code>binomial(m,n)</code>	Número combinatorio $\binom{a}{b} = \frac{m!}{n!(m-n)!}$
<code>csc(x)</code>	Cosecante de x
<code>cos(x)</code>	Coseno de x
<code>cosh(x)</code>	Coseno hiperbólico de x
<code>cot(x)</code>	Cotangente de x
<code>exp(x)</code>	Función exponencial, e^x
<code>floor(x)</code>	Parte entera de x ($p \in \mathbf{Z} / p \leq x < p+1$)
<code>log(x)</code>	Logaritmo neperiano de x
<code>max(x1,x2,x3,...)</code>	Máximo de $x_1, x_2, x_3...$
<code>min(x1,x2,x3,...)</code>	Mínimo de $x_1, x_2, x_3...$
<code>signum(x)</code>	Signo de x (1 si $x > 0$, -1 si $x < 0$, 0 si $x = 0$)
<code>sin(x)</code>	Seno de x
<code>sinh(x)</code>	Seno hiperbólico de x
<code>sqrt(x)</code>	Raíz cuadrada de x
<code>tan(x)</code>	Tangente de x
<code>tanh(x)</code>	Tangente hiperbólica de x
<code>x!</code>	Factorial de x

Figura 5.1: Algunas funciones predefinidas en Maxima

5.2. Funciones

Como sabemos, en Maxima podemos definir funciones mediante el operador “:=”. Además, existen decenas de funciones predefinidas, algunas de las cuales se listan en la tabla 5.2. Podemos componer funciones utilizando la sintaxis habitual, $f(g(x))$ y definir funciones a trozos mediante una secuencia “if”...“then”...“else”.

En el siguiente ejemplo, definimos una función compuesta y una función a trozos, dada por:

$$f(x) = \begin{cases} \text{sen}(x) & \text{si } x \leq 0 \\ \log(x+1) & \text{si } x > 0 \end{cases}$$

```
(%i1) f(x):=1-x/(x-1);
(%o1)
```

$$f(x) := 1 - \frac{x}{x-1}$$

```
(%i2)  g(x):=cos(x);
(%o2)
```

$$g(x) := \cos x$$

```
(%i3)  h(x):=f(g(x));
(%o3)
```

$$h(x) := f(g(x))$$

```
(%i4)  h(x);
(%o4)
```

$$1 - \frac{\cos x}{\cos x - 1}$$

```
(%i5)  ratsimp(h(x));
(%o5)
```

$$-\frac{1}{\cos x - 1}$$

```
(%i6)  f(x):= if x<=0 then sin(x) else log(x+1);
(%o6)
```

$$f(x) := \text{if } x \leq 0 \text{ then } \sin x \text{ else } \log(x + 1)$$

```
(%i7)  [f(-%pi/2), f(0), f(3)];
(%o7)
```

$$[-1, 0, \log 4]$$

Las funciones definidas a trozos tienen el problema de que no es posible emplearlas directamente en *Maxima* para calcular límites, derivadas, integrales, etc. (aunque sí es posible representarlas gráficamente, como se comentará en el capítulo 5.3). Por ejemplo:

```
(%i8)  limit(f(x), x, 0, minus);
Maxima was unable to evaluate the predicate:
** error while printing error message **
Maxima was unable to evaluate the predicate:~%~M
#0: f(x=x)
- an error. Quitting. To debug this try debugmode(true);
```

Una posible solución es definir por separado cada una de las ramas que forman la función a trozos, así resultará algo más cómodo el cálculo de límites laterales, derivadas, etc.

```
(%i9)  f1(x):=sin(x);
```

```
(%o9)
f1(x) := sin x

(%i10) f2(x) := log(x+1);
(%o10)
f2(x) := log(x + 1)

(%i11) f(x) := if x <= 0 then f1(x) else f2(x);
(%o11)
f(x) := if x ≤ 0 then f1(x) else f2(x)

(%i12) l1:limit(f1(x),x,0,minus);
(%o12)
0

(%i13) l2:limit(f2(x),x,0,plus);
(%o13)
0

(%i14) is(l1=l2 and l2=f(0));
(%o14)
true
```

En el ejemplo anterior, se utilizó la función “is” para resaltar el hecho de que f es continua en $x = 0$. Un segundo ejemplo:

$$f(x) = \begin{cases} \frac{\sqrt{x^2+1}}{x} & \text{si } x \leq -1 \\ x^2 + x - 1 & \text{si } x \in [-1, 1] \\ 2x - 1 & \text{si } x > 1 \end{cases}$$

```
(%i15) f1(x) := sqrt(x^2+1)/x;
(%o15)
f1(x) :=  $\frac{\sqrt{x^2+1}}{x}$ 

(%i16) f2(x) := x^2+x-1;
(%o16)
f2(x) := x2 + x - 1
```



```
(%i17) f3(x):=2*x-1;
(%o17)
```

$$f_3(x) := 2x - 1$$

```
(%i18) f(x):= if x<-1 then f1(x) else if x<=1 then f2(x) else
f3(x);
(%o18)
```

$$f(x) := \text{if } x < -1 \text{ then } f_1(x) \text{ else (if } x \leq 1 \text{ then } f_2(x) \text{ else } f_3(x))$$

```
(%i19) l1:limit(f1(x),x,-1,minus);
(%o19)
```

$$-\sqrt{2}$$

```
(%i20) l2:limit(f2(x),x,-1,plus);
(%o20)
```

$$-1$$

```
(%i21) l3:limit(f2(x),x,1,minus);
(%o21)
```

$$1$$

```
(%i22) l4:limit(f3(x),x,1,plus);
(%o22)
```

$$1$$

```
(%i23) is(l1=l2 and l2=f(-1));
(%o23)
```

false

```
(%i24) is(l3=l4 and l4=f(1));
(%o24)
```

true

5.3. Representación de gráficas

Para la representación de gráficas, *Maxima* y *wxMaxima* cuentan con algunas opciones que superan las pretensiones del presente documento. Aun así, aunque en alguna ocasión se supere lo estrictamente necesario, en la presente sección se ha optado por presentar todas aquellas facetas relacionadas con los gráficos cuya realización resulte especialmente sencilla desde *wxMaxima*, como gráficas polares y paramétricas. Además, con el ánimo de mostrar las posibilidades más representativas de *Maxima* en gráficas 2D, se muestran algunos ejemplos de gráficas discretas (poligonales) y se muestran algunos sobre cómo configurar el aspecto de las gráficas. Sin embargo, aunque *wxMaxima* también facilita la creación de gráficas 3D, éstas superan excesivamente los objetivos de este manual y no se tratarán en lo que sigue.

5.3.1. Gráficas de funciones de una variable (en coordenadas cartesianas)

Para dibujar gráficas de funciones de una variable *Maxima* utiliza el comando “`plot2d`”, al que se puede acceder a través de la entrada “Gráficos” → “Gráficos 2D” de *wxMaxima* (figura 5.2). Como se puede apreciar en la figura, existen numerosos parámetros pero,



Figura 5.2: Ventana de diálogo de Gráficos 2D en *wxMaxima*

antes de discutirlos, es conveniente detenerse a entender la forma en que *Maxima* genera los gráficos: cuando recibe una orden del usuario, ésta es automáticamente procesada y enviada a un programa externo, sobre el que *Maxima* delegará la tarea de generar la gráfica resultante y mostrarla en pantalla (o enviarla hacia un fichero, a un dispositivo, etc). Por defecto, el programa utilizado es “`Gnuplot`”, una potente (y veterana) aplica-

ción de propósito general con licencia libre, aunque existen otras posibilidades, que se detallarán más adelante.

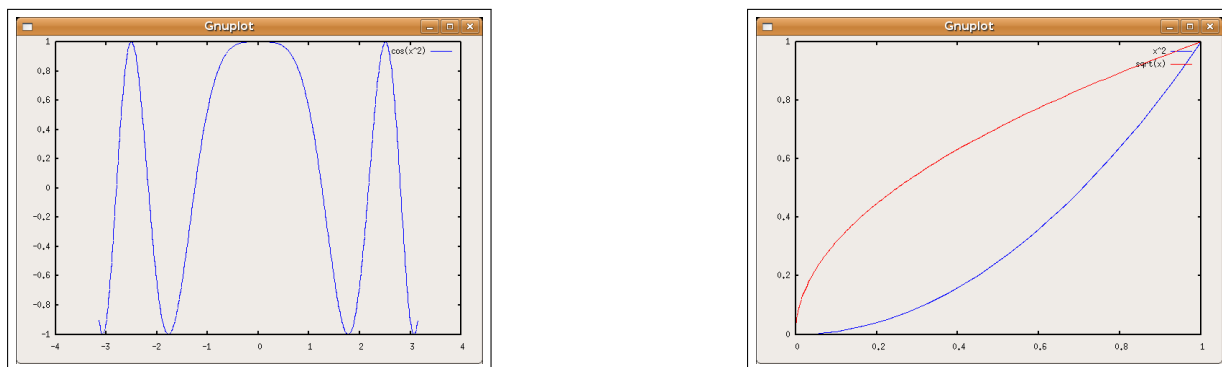


Figura 5.3: Gráficas de $\cos(x^2)$ y del par $[x^2, \sqrt{x}]$

Como mínimo, “plot2d” toma dos parámetros, el primero de los cuales es la función (o lista de funciones) que deben ser representadas y el segundo, el intervalo de valores que tomará la variable independiente. En *wxMaxima*, se abrirá una nueva ventana en la que aparecerá el resultado (figura 5.3).

```
(%i1) plot2d(cos(x^2), [x, -%pi, %pi])$
(%i2) plot2d([x^2, sqrt(x)], [x, 0, 1])$
```

Como tercer parámetro, podemos pasar un intervalo que indica el máximo rango de valores de la variable dependiente que deben ser representados. Si es necesario, se reducirá el dominio de valores de la variable independiente, de forma que su imagen mediante f no sobrepase este intervalo. Por ejemplo la siguiente sesión producirá el resultado que se puede observar en la figura 5.4

```
(%i3) f(x) := x^2$
(%i4) plot2d(f(x), [x, 0, 3], [y, 0, 4])$
```

En esta figura, *Maxima* ha reducido el intervalo de valores para x , tomando $[0, 2]$, pues

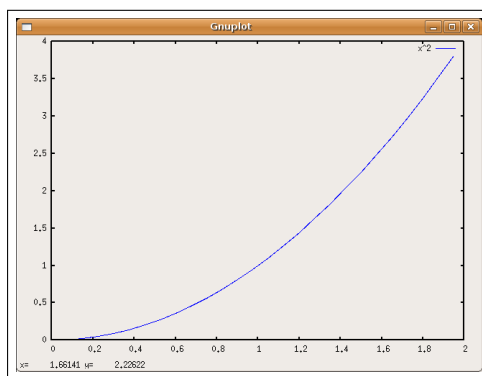


Figura 5.4: Gráfica de x^2 con las restricciones $x \in [0, 3]$, $y \in [0, 4]$

si $x > 2$ los valores de y desbordarían a $[0, 4]$.

Sin embargo, en algunas ocasiones nos puede interesar obtener una gráfica sobre un marco más amplio que el de los valores que tome la función. En este caso, podemos introducir un cuarto argumento en el que pedimos explícitamente a Gnuplot que realice esta tarea, como se observa en el siguiente ejemplo. En *wxMaxima* este proceso se simplifica, pues solamente deberemos marcar la opción “Enviar rango a Gnuplot” en la ventana de diálogo de “Gráficos 2D”.

```
(%i5) plot2d([x^2], [x,0,3], [y,0,4], [gnuplot_preamble, "set
xrange [0:3]; set yrange [0:4];"])$
```

El resultado se puede observar en la figura 5.5

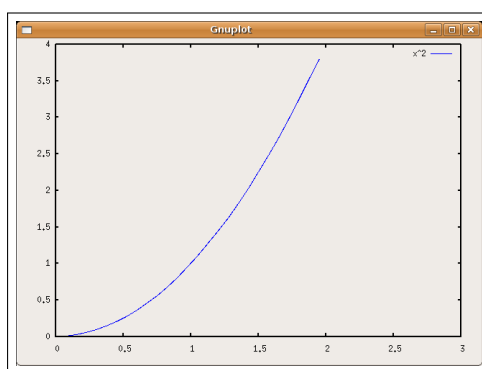


Figura 5.5: Gráfica de x^2 , ampliada a todo $x \in [0, 3]$, $y \in [0, 4]$

De la misma forma en que hemos visto, podemos utilizar la opción “gnuplot_preamble” para pasar a Gnuplot cualquier serie de comandos que deseemos sean ejecutados por este programa antes de representar la gráfica y que modificarán el aspecto final de la misma. Esto transfiere a *Maxima* toda la potencia de Gnuplot, pero también toda su complejidad. Por suerte, *wxMaxima* contiene, dentro de la ventana de diálogo “Gráficos 2D”, una serie de órdenes para Gnuplot que, en muchos de los casos más frecuentes, nos permitirán realizar esta tarea a golpe de ratón. Éstas se encuentran dentro de la pestaña “Opciones” de dicha ventana de diálogo, y algunas de ellas son:

- “set zeroaxis;”. Dibujar los ejes de coordenadas.
- “set size ratio 1; set zeroaxis;”. Dibujar los ejes de coordenadas, de forma que cada unidad en OY es el doble de grande que en OX .
- “set grid;”. Dibujar una rejilla.
- “set polar; set zeroaxis;”. Dibujar una gráfica en coordenadas polares. Por defecto, se supone que la variable independiente (el ángulo) es “ph”.

A continuación, se muestran algunos ejemplos, cuyos resultados se pueden observar en las figuras 5.6:

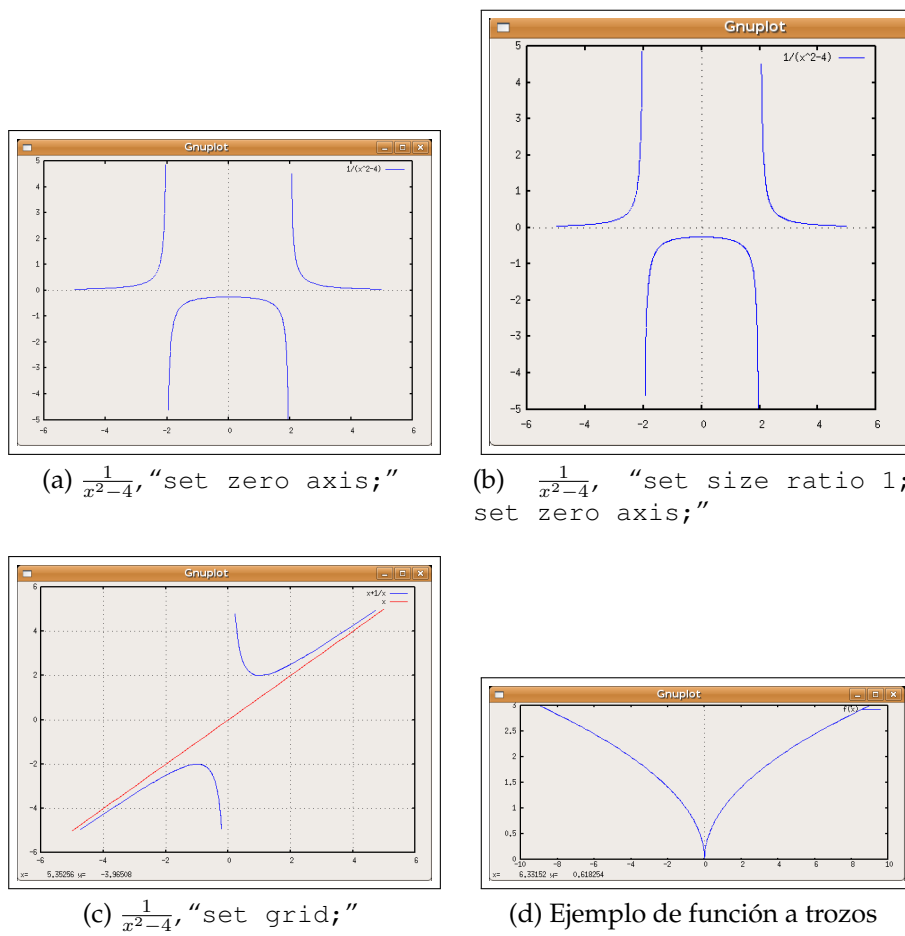


Figura 5.6: Algunas gráficas en coordenadas cartesianas

```
(%i6) plot2d([1/(x^2-4)], [x,-5,5], [y,-5,5], [gnuplot_preamble,
"set zeroaxis;"])$
(%i7) plot2d([1/(x^2-4)], [x,-5,5], [y,-5,5], [gnuplot_preamble,
"set size ratio 1; set zeroaxis;"])$
(%i8) plot2d([x+1/x,x], [x,-5,5], [y,-5,5], [gnuplot_preamble,
"set grid;"])$
(%i9) f(x):=if x<0 then sqrt(-x) else sqrt(x)$
(%i10) plot2d('(f(x))', [x,-9,9], [gnuplot_preamble, "set zeroaxis;"])$
```

En el último de los ejemplos anteriores, es destacable ver cómo las funciones definidas a trozos se pueden representar en *Maxima* con ayuda de los paréntesis y del operador “'” (apóstrofe, que en el teclado Español está situado en la misma tecla que la interrogación “?”).

5.3.2. Gráficas en coordenadas polares y paramétricas

En las siguientes líneas se muestran algunos ejemplos sobre la representación en *wxMaxima* de gráficas en coordenadas polares, utilizando la opción que se comentó en el apartado anterior. Los resultados se pueden ver en la figura 5.7. En la segunda espiral (figura 5.7b) se ha utilizado el parámetro “*nticks=30*”, con la intención de introducir más puntos para suavizar así la poligonal que representa a la espiral, consiguiendo el efecto visual de una línea curva. En *wxMaxima*, este parámetro se puede regular desde el botón [Graduaciones], situado en la ventana de gráficos.

```
(%i1) plot2d([ph], [ph,0,4*%pi], [gnuplot_preamble, "set polar;
set zeroaxis;"])$
(%i2) plot2d([ph], [ph,0,4*%pi], [gnuplot_preamble, "set polar;
set zeroaxis;"], [nticks,30])$
(%i3) plot2d([2+cos(5*phi)], [phi, 0, 2*%pi], [gnuplot_preamble,
"set polar; set zeroaxis;"])$
(%i4) r(ph):=exp(cos(ph))-2*cos(4*ph)+sin(ph/12)**5$
(%i5) /* Mariposa */
      plot2d(r(ph), [ph,0,2*%pi], [gnuplot_preamble, "set polar; set
zeroaxis;"])$
```

Además de las polares, *wxMaxima* facilita la representación de curvas paramétricas, a través del botón [Parametrica] situado en la parte superior derecha de la ventana de diálogo de “Gráficos 2D”. A continuación, se presenta un ejemplo cuyo resultado se puede observar en la figura 5.8.

```
(%i6) plot2d ([parametric, cos(t), sin(t)], [t, 0, 2*%pi], [nticks,50])
```

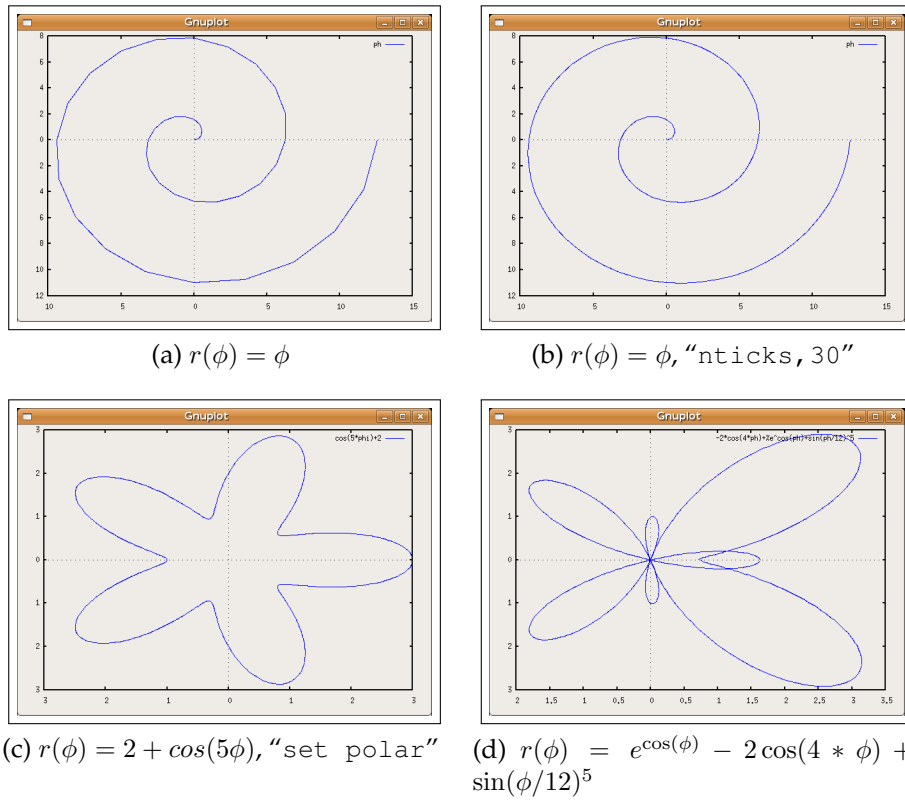
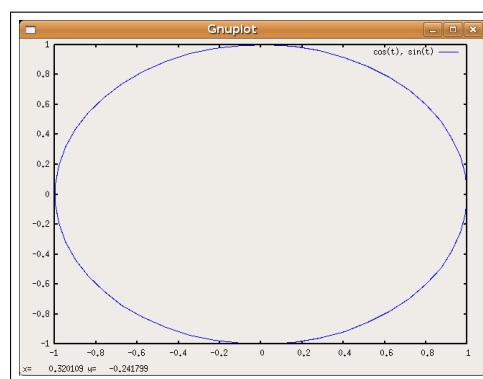
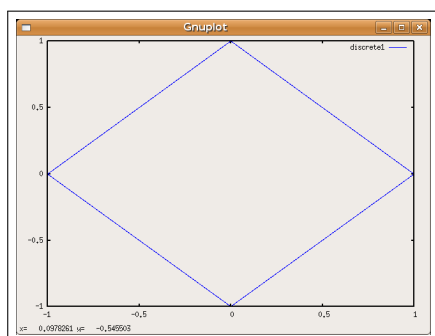
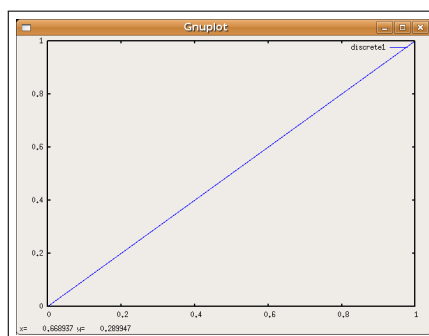


Figura 5.7: Gráficas polares

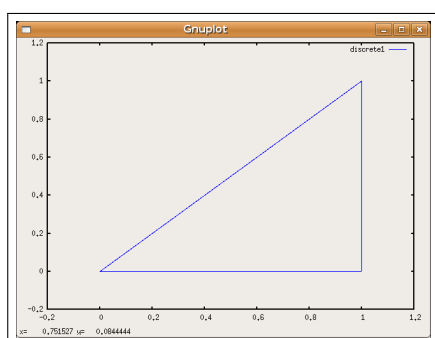
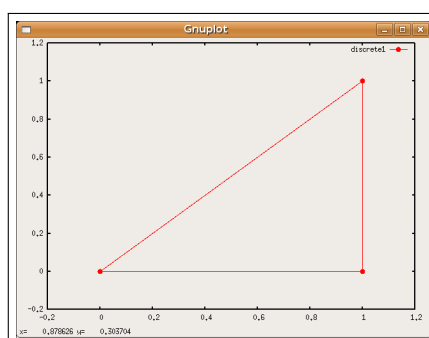
Figura 5.8: Curva paramétrica $(\cos(t), \sin(t))$, $t \in [0, 2\pi]$



(a) Rombo



(b) Triángulo 1 (dos lados ocultos)

(c) Triángulo 2 (fijando rangos x e y)

(d) Triángulo 3 (dibujando vértices)

Figura 5.9: Poligonales (opción “discrete” de *Maxima*)

5.3.3. Gráficas de curvas poligonales

Para terminar con este repaso a los distintos tipos de curvas 2D que pueden ser representados en *Maxima*, se mostrará a continuación un ejemplo relacionado con la forma de representar gráficos a partir de tablas discretas de valores. Aunque este tipo de curvas ha sido introducido en la ventana de diálogo “Gráficos 2D” de *wxMaxima* a partir de la versión 0.7.0 (a través del botón “Especial”), las personas que estén utilizando versiones algo más antiguas no podrán disfrutar de esta posibilidad y deberán teclear directamente la orden “discrete” en la línea de *Maxima*, como se puede observar en el siguiente ejemplo.²

```
(%i1) lista_x: [-1, 0, 1, 0, -1]$
(%i2) lista_y: [0, -1, 0, 1, 0]$
(%i3) plot2d([discrete, lista_x, lista_y])$
```

En el ejemplo anterior, se dibuja la poligonal (en este caso, cerrada) resultante de unir los puntos $(-1, 0)$, $(0, -1)$, $(1, 0)$, $(0, 1)$ y $(-1, 0)$, cuyas coordenadas x e y vienen dadas en dos listas. El resultado se puede ver en la figura 5.9a. Por supuesto, el estilo de

²Por ejemplo, los sistemas operativos Guadalinex V4 y Ubuntu 6.10 incluyen *wxMaxima* versión 0.6.X y no podrán disfrutar de esta opción en la ventana de gráficos de *wxMaxima*.

dibujo se puede afinar mucho más utilizando opciones de Gnuplot.

En el siguiente ejemplo, al representar un triángulo rectángulo, dos de sus lados quedan ocultos por el marco que utiliza Gnuplot para mostrar las coordenadas (figura 5.9b). Por ese motivo, utilizamos los comandos de Gnuplot “set xrange” y “set yrange” (a los que se puede acceder desde la ventana de “Gráficos 2D” de *wxMaxima* a través de la opción “Enviar rango a Gnuplot”). El resultado se puede ver en la figura 5.9c). Además, se muestra un ejemplo sobre la forma de cambiar las propiedades de la curva y de dibujar los vértices de la poligonal, produciendo la gráfica 5.9d).

```
(%i4) lista2_x:[0,1,1,0];
```

```
(%o4)
```

[0, 1, 1, 0]

```
(%i5) lista2_y:[0,0,1,0];
```

```
(%o5)
```

[0, 0, 1, 0]

```
(%i6) plot2d([discrete,lista2_x,lista2_y])$
```

```
(%i7) caja_gnuplot: [gnuplot_preamble, "set xrange [-0.2:1.2];  
set yrange [-0.2:1.2];"]$
```

```
(%i8) plot2d([discrete,lista2_x,lista2_y], caja_gnuplot)$
```

```
(%i9) estilo_gnuplot: [gnuplot_curve_styles, ["with linespoints  
pointsize 2 pointtype 7"]]$
```

```
(%i10) plot2d([discrete,lista2_x,lista2_y], caja_gnuplot, estilo_gnupl
```

5.3.4. Otras cuestiones relacionadas con los gráficos

wxMaxima nos permite grabar, de forma sencilla, un gráfico en un fichero. Para ello, simplemente debemos introducir el nombre del fichero en el campo “Gráfico al archivo” de la ventana de diálogo de “Gráficos 2D”. Automáticamente, *wxMaxima* generará las órdenes adecuadas, utilizando Gnuplot y obteniendo resultados similares al que se puede observar en el siguiente ejemplo:

```
(%i1) f(x):=sin(x)*cos(x/3);
```

```
(%o1)
```

$$f(x) := \sin x \cos\left(\frac{x}{3}\right)$$

```
(%i2) plot2d([f(x)], [x,-5,5], [gnuplot_term, ps], [gnuplot_out_file,  
"grafica.eps"])$
```

```
Output file "grafica.eps".
```

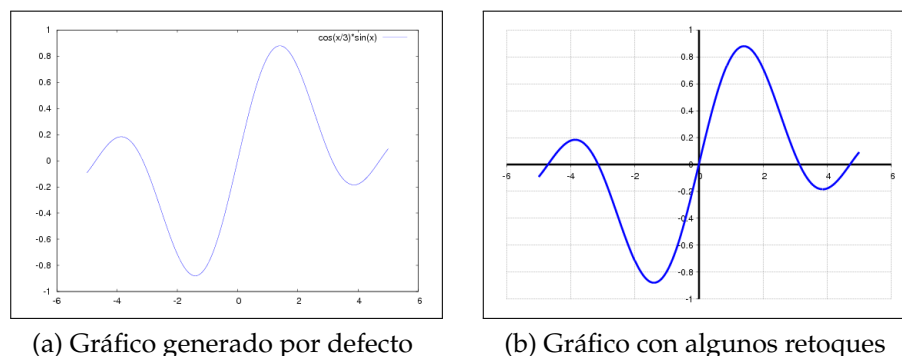


Figura 5.10: Ficheros gráficos postscript generados por *Maxima*

El resultado, que se puede ver en la figura 5.10a, es un fichero en formato “postscript” encapsulado (al que, en el ejemplo anterior se ha llamado “grafica.eps”). Este tipo de ficheros, que son prácticamente un estándar para la codificación de figuras gráficas científicas, describen a éstas mediante un lenguaje de programación propio (llamado, asimismo, “postscript”) y pueden ser escalados tanto como sea necesario, sin perder calidad. Si se desea, pueden ser convertidos a algún otro tipo de fichero gráfico, como jpg ó png, utilizando programas adecuados, como el excelente *gimp* o bien *convert* (ambos con licencia libre y con versiones tanto para GNU/Linux como para Windows)³.

Y para terminar con la representación de gráficas con *Maxima* y Gnuplot, se mostrará a continuación un ejemplo mucho más avanzado que puede servir como indicación de hasta qué punto se pueden personalizar las gráficas, utilizando la técnica de retocar las opciones de Gnuplot. Como se puede ver en la figura 5.10b, se consigue un resultado bastante fino, aunque puede resultar muy engorroso el tener que introducir toda la línea de opciones de Gnuplot que son necesarias y que aparecen más abajo. En el próximo capítulo, veremos como se puede configurar a *Maxima*, personalizando entre otras cosas el aspecto que tendrán las gráficas generadas con Gnuplot sin necesidad de teclear tantas órdenes.

```
(%i27) graph_options:"set xzeroaxis linetype -1 linewidth 5;
set yzeroaxis linetype -1 linewidth 5; set xtics axis nomirror;
set ytics axis nomirror; unset x2tics; set grid; unset border;
unset key;"$
(%i28) curve_options:"with lines linetype 3 linewidth 10"$
(%i29) plot2d([f(x)], [x,-5,5], [gnuplot_preamble, graph_options],
[gnuplot_curve_styles, curve_options], [gnuplot_term, ps], [gnuplot_out_
"grafica.eps"])$
Output file "grafica.eps".
```

³Por ejemplo, las figuras 5.10a y 5.10b han sido convertidas a formato jpg escribiendo en una consola de GNU/Linux la orden “convert grafica.eps grafica.jpg”

No entraremos a describir cada una de las opciones anteriores. Desgraciadamente, para profundizar más en la modificación de las propiedades de las curvas (por ejemplo, tal y como antes, modificar grosor, tipo, color, etc.), las versiones actuales de *wxMaxima* no son de gran ayuda y es necesario tener algunos conocimientos del lenguaje de Gnuplot. En su página web⁴ y en general en Internet existe abundante información al respecto. Además, *Maxima* almacena las opciones gráficas en la variable “plot_options” y modificando ésta se pueden conseguir distintos efectos, que persistirán de forma global, para el resto de los gráficos que realicemos en la sesión actual de *Maxima*. Más información en la documentación de *Maxima* y en el menú de ayuda de *wxMaxima*.

Aunque hasta ahora nos hemos centrado en Gnuplot, *Maxima* puede utilizar distintos programas para la representación de gráficos, a algunos de los cuales (como “openmath”) se puede acceder desde el menú de “Gráficos 2D” de *wxMaxima* y a otros no (como “geomview”). Además, a las versiones de *wxMaxima* 0.7.X, existe un modo “inline” de representación de gráficos que hace que las figuras aparezcan dentro del área de salida, junto al resto de las entradas y salidas *Maxima*. Cada una de estas posibilidades tiene sus ventajas e inconvenientes (por ejemplo, “openmath” y “geomview” tienen la posibilidad de retocar de forma interactiva algunas características de las gráficas que ha generado *Maxima*, en particular el último de estos programas permite manipular gráficas 3D (trasladar, girar, hacer zoom, cambiar color, textura y luces de las superficies, etc.). Pero todo ello excede los objetivos del presente manual.

5.4. Derivadas, integrales y aplicaciones

Para calcular derivadas se utiliza la función “diff” (accesible a través del menú “Análisis” → “Derivar...” de *wxMaxima*), que toma como argumentos la función a derivar, la variable con respecto a la cual hacerlo y, opcionalmente, el orden de derivación. Para calcular integrales, se usa la función “integrate” (que es accesible en “Análisis” → “Integrar...”) que, además de la función y la variable de integración, toma opcionalmente como argumentos los extremos del intervalo para el cálculo de integrales definidas. Podemos calcular desarrollos de Taylor mediante la función “taylor”, que toma como argumentos la función, la variable, el centro y el orden.

```
(%i1) f(x):=x^a * %e^x;
(%o1)
```

$$f(x) := x^a e^x$$

```
(%i2) diff(f(x), x);
(%o2)
```

$$x^a e^x + a x^{a-1} e^x$$

⁴<http://www.gnuplot.info/>

```
(%i3) diff(f(x),x,2), a=0 /* Segunda derivada para a=0 */;
(%o3)
```

$$e^x$$

```
(%i4) g(x):=x/(x^2+1);
(%o4)
```

$$g(x) := \frac{x}{x^2 + 1}$$

```
(%i5) diff(g(x),x);
(%o5)
```

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

```
(%i6) ratsimp(%);
(%o6)
```

$$-\frac{x^2 - 1}{x^4 + 2x^2 + 1}$$

```
(%i7) integrate(%,x);
(%o7)
```

$$\frac{x}{x^2 + 1}$$

```
(%i8) integrate(9*x*cos(3*x),x,0,%pi);
(%o8)
```

$$-2$$

```
(%i9) h:=(e^x-1)/x;
(%o9)
```

$$\frac{e^x - 1}{x}$$

```
(%i10) taylor(h,x,0,4);
(%o10)
```

$$1 + \frac{x}{2} + \frac{x^2}{6} + \frac{x^3}{24} + \frac{x^4}{120} + \cdots$$

Utilizando la primera y la segunda derivada, será fácil calcular el crecimiento/decrecimiento, puntos críticos y extremos relativos, etc. Por ejemplo, estudiaremos los extremos relativos de la función

$$f(x) = x^4 - x;$$

A continuación se muestra la secuencia de órdenes en *Maxima* junto a sus respectivas salidas, por supuesto, muchos de estos pasos (calcular derivadas, resolver ecuaciones, dibujar gráficas) pueden ser realizados utilizando los menús y botones de *wxMaxima*.

```
(%i11) f(x):=x^4-x;
```

```
(%o11)
```

$$f(x) := x^4 - x$$

```
(%i12) diff(f(x),x) /* Primera derivada */;
```

```
(%o12)
```

$$4x^3 - 1$$

```
(%i13) D1f(x):="(%) /* La llamamos D1f(x) */;
```

```
(%o13)
```

$$D1f(x) := 4x^3 - 1$$

```
(%i14) solve(D1f(x),x) /* Puntos críticos */;
```

```
(%o14)
```

$$\left[x = \frac{\sqrt{3}i - 1}{2 \cdot 4^{\frac{1}{3}}}, x = -\frac{\sqrt{3}i + 1}{2 \cdot 4^{\frac{1}{3}}}, x = \frac{1}{4^{\frac{1}{3}}} \right]$$

```
(%i15) last(%);
```

```
(%o15)
```

$$x = \frac{1}{4^{\frac{1}{3}}}$$

```
(%i16) a:rhs(%) /* a: pto crítico */;
```

```
(%o16)
```

$$\frac{1}{4^{\frac{1}{3}}}$$

```
(%i17) diff(f(x),x,2) /* Segunda derivada */;
```

```
(%o17)
```

$$12x^2$$

```
(%i18) D2f(x):="(%) /* La llamamos D2f(x) */;
```

```
(%o18)
```

$$D2f(x) := 12x^2$$

```
(%i19) D2f(a) /* Evaluamos en el pto crítico */;
```

```
(%o19)
```

$$\frac{12}{4^{\frac{2}{3}}}$$

```
(%i20) if "(%)>0 then print("Mínimo relativo!!!") else print("Máximo  
%relativo!!!");
```

```
(%o20)
```

```
Mínimo relativo!!!
```

```
(%i21) plot2d(f(x),[x,-5,5], [y,-5,5], [gnuplot_preamble, "set  
zeroaxis;"]);
```

```
(%o21)
```

Algunos comentarios: para manejar la primera derivada de $f(x)$, se definió una nueva función, $D1f(x)$. Para ello, fue necesario preceder la expresión de la salida anterior (%) por dos apóstrofes, "" (cada uno de los cuales se obtiene pulsando la tecla ?). No confundir con las comillas, situadas en la tecla 2). La misma técnica se ha utilizado para definir la segunda derivada como $D2f(x)$ y para utilizar la salida anterior dentro de una sentencia "if". La gráfica se muestra en la figura 5.11.

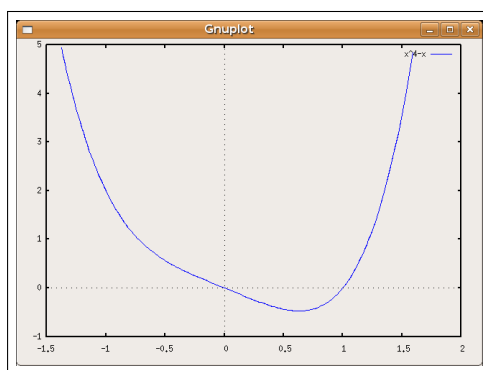


Figura 5.11: $f(x) = x^4 - x$

A estas alturas, sabemos que *Maxima* es un sistema de cálculo simbólico muy potente, que tiene una gran cantidad de funciones predefinidas y que puede resultar un valioso instrumento si es usado apropiadamente por matemáticos y científicos, en concreto para la docencia en enseñanzas medias o en universidad. Pero además, como veremos en este capítulo, podremos sacarle un mayor provecho si somos capaces de definir nuevas funcionalidades y hacer sencillo el acceso a éstas.

6.1. Almacenando sesiones y datos

Si en *Maxima* no existe ninguna función para el cálculo de áreas, tenemos una solución es muy sencilla: podemos definir todas aquellas funciones que necesitemos. Por ejemplo, introducimos las siguientes funciones y comprobamos que todo va bien:

```
(%i1) area_tri(base,altura):=base*altura/2;
```

```
(%o1)
```

$$\text{area_tri}(\text{base}, \text{altura}) := \frac{\text{base altura}}{2}$$

```
(%i2) area_rec(base,altura):=base*altura;
```

```
(%o2)
```

$$\text{area_rec}(\text{base}, \text{altura}) := \text{base altura}$$

```
(%i3) area_cua(lado):=lado^2;
```

```
(%o3)
```

$$\text{area_cua}(\text{lado}) := \text{lado}^2$$

```
(%i4) area_cir(radio):=%pi*radio^2;
```

```
(%o4)
```

$$\text{area_cir}(\text{radio}) := \pi \text{radio}^2$$

```
(%i5) area_tri(2,1) /* Probando area_tri() */;
```

```
(%o5)
```



```
(%i6) area_rec(2,a) /* Probando area_rec() */;
(%o6)
```

2a

```
(%i7) area_cua(sqrt(2)) /* Probando area_cua() */;
(%o7)
```

2

```
(%i8) area_cir(5) /* Probando area_cua() */;
(%o8)
```

25π

El problema es que si en este momento salimos de *Maxima*, la próxima vez que entremos todo lo anterior se habrá perdido y será necesario volver a teclear estas definiciones. En *wxMaxima*, existe una solución muy evidente: antes de salir, podemos pulsar la entrada de menú "Archivo" → "Guardar" o "Archivo" → "Guardar como", de esta forma el programa nos preguntará un nombre de fichero (podemos llamarlo, por ejemplo, `areas.mac` (utilizando la extensión `.mac`, que proviene del nombre *Macsyma*). En este fichero se almacenará toda la sesión actual. La próxima vez que abramos el programa, si lo abrimos mediante "Archivo" → "Abrir", se repetirán todas las acciones efectuadas en la sesión anterior y en particular volveremos a contar con las funciones que habíamos definido.

La posibilidad de guardar una sesión completa de *wxMaxima* para que sea recuperada posteriormente puede ser interesante si deseamos posponer el trabajo que estamos realizando o bien puede ser útil para fines didácticos. En el lenguaje de *Maxima*, se puede utilizar la orden "batch" con la misma finalidad, leer una serie de expresiones de *Maxima* desde un fichero y ejecutarlas. Estas expresiones se encuentran en formato de texto, inteligible por una persona. Para crear el fichero, podemos utilizar cualquier editor de texto¹. También podemos utilizar cualquier fichero que haya sido guardado desde *wxMaxima*. Existen algunas variantes: la orden "demo" ejecutará el fichero paso a paso, deteniéndose después de cada orden hasta que pulsemos la tecla INTRO, mientras que la orden "batchload" evalúa todas las órdenes de un fichero sin mostrar ningún tipo de entradas ni las salida.

Por ejemplo, si escribimos la orden

```
demo("areas.mac");
```

¹Existen editores, como *Emacs* (con licencia libre), que poseen modos de funcionamiento especiales para *Maxima*, incluyendo funcionalidades como resaltado de sintaxis, auto-completado o ejecución automática comandos

se producirá el mismo efecto que si abrimos este fichero desde el menú de *wxMaxima*, pero a través ejecución paso a paso².

Otro ejemplo: en un sistema Guadalinex (o, en general en Ubuntu u otro sistema derivado de Debian GNU/Linux) en el que se haya instalado la versión 5.10.0 de *Maxima*, la orden

```
batch("/usr/share/maxima/5.10.0/demo/demo.dem");
```

provocaría la ejecución del contenido del fichero `demo.dem`, situado en la ruta de directorios anterior. La ejecución del fichero consiste en una demostración de las posibilidades de *Maxima*³, Mientras que la orden

```
demo("/usr/share/maxima/5.10.0/demo/demo.dem");
```

producirá el mismo resultado, aunque deteniéndose en cada paso.

Sin embargo, en muchas ocasiones no necesitaremos almacenar toda una sesión, sino solamente recordar los valores de determinadas variables, funciones o en general de determinadas expresiones. En el ejemplo anterior, nos podría interesar crear un paquete en el que se almacenaran los nombres de las funciones que hemos creado, para utilizarlas cuando deseemos calcular áreas. Para este fin, podemos utilizar las órdenes “*save*” y “*loadfile*” de *Maxima*. La primera de ellas toma como parámetros el nombre del fichero y el de las variables o las funciones que deseamos grabar, que son almacenadas en dicho fichero. El segundo, simplemente el nombre del fichero cuyos datos deseamos cargar en la sesión actual de *Maxima*.

Una particularidad destacable “*save*” no utiliza el lenguaje habitual de *Maxima* para almacenar los datos, sino que crea un fichero en lenguaje “*lisp*” en el que, como veremos más adelante, está programado *Maxima*. En cualquier caso, esto es transparente al usuario que, de hecho, solamente notará una mayor velocidad de acceso a los datos almacenados en el caso de que estos sean muy numerosos.

Por ejemplo, supongamos que, después de probar que nuestras funciones, todo va bien y queremos almacenarlas para un uso posterior. Podemos utilizar la orden:

```
(%i9) area_cua(3);
(%o9)
```

9

```
(%i10) save("areas.lisp", area_tri, area_rec, area_cua, area_cir);
```

²Como se comenta más abajo, para que esta orden localice el fichero, es necesario que éste se encuentre en alguno de los directorios en los que la función “*demo*” realiza las búsquedas (es decir, en alguno de los que se listan en la variable “*file_search_demo*”)

³Este fichero contiene numerosas e interesantes instrucciones en *Maxima* que pueden servir como ilustración de sus posibilidades. En algunas de ellas, *Maxima* devuelve un mensaje de error, pues el fichero conserva la sintaxis del antiguo *Macsyma* y no ha sido actualizado. En el mismo directorio se pueden encontrar más ejemplos.

```
(%o10)
areas.lisp
```

En este caso, podríamos haber abreviado y utilizar la variable especial “functions”, que contiene una lista con todas las funciones actualmente definidas⁴:

```
(%i11) save("areas.lisp", functions);
(%o11)
areas.lisp
```

O bien, podríamos haber escrito lo siguiente:

```
(%i12) save("areas.lisp", all);
(%o12)
areas.lisp
```

en cuyo caso se habrían almacenado todas las funciones, pero también todas las variables (en caso de que hubiéramos definido alguna) e incluso los valores de las etiquetas %i1, %i2, etc. Además, si deseamos almacenar solamente las variables que se hayan definido podemos utilizar la variable especial⁵ “values”.

Cuando guardamos o cargamos un fichero mediante las órdenes “save” y “loadfile”, este se almacenará en aquella carpeta desde la que hayamos arrancado *Maxima*. En los sistemas GNU/Linux esta es, por defecto, nuestro directorio de inicio, es decir, un directorio del tipo:

```
/home/usuario/,
```

donde “usuario” es nuestro nombre de usuario en el sistema

Al día siguiente, volvemos a abrir *Maxima*, cargamos el fichero y volvemos a contar con las funciones que habíamos definido:

```
(%i1) loadfile("areas.lisp");
(%o12)
areas.lisp
```

```
(%i13) area_cir(r);
(%o13)
```

$$\pi r^2$$

⁴A esta lista de funciones se puede acceder desde el menú “Maxima” → “Mostrar funciones” de *wxMaxima*)

⁵A la que en *wxMaxima* se puede acceder desde “Maxima” → “Mostrar variables”

De forma análoga a “loadfile”, se puede utilizar la función “load” para recuperar los contenidos de un fichero. La diferencia es que esta última función lo buscará en una lista de directorios específicos, en los que se almacenan ficheros del sistema. Por ejemplo, aunque *Maxima* no esté orientado al cálculo estadístico, dispone de paquetes que pueden utilizarse para tal fin, por ejemplo “descriptive”, creado por el profesor Mario Rodríguez Ríotorto, al que podemos acceder mediante la función “load”

```
(%i1) load(descriptive);
(%o1)
```

```
/usr/share/maxima/5.10.0/share/contrib/descriptive/descriptive.mac
```

```
(%i2) data:[1,2,3,1,4,2,1,5,3,5,1,4,2,5,1,3,2]$
(%i3) mean(data) /* Media */;
(%o3)
```

$$\frac{45}{17}$$

```
(%i4) var(data) /* Varianza */;
(%o4)
```

$$\frac{610}{289}$$

```
(%i5) median(data) /* Mediana*/;
(%o5)
```

$$2$$

En el manual de *Maxima*⁶ se puede consultar un amplio listado de los paquetes disponibles.

La lista de directorios en los que “load” busca los paquetes se almacena en la variable “file_search_maxima”. Si tenemos funciones o variables que vayamos a utilizar con frecuencia y las almacenamos mediante “save”, podrá ser de utilidad la posibilidad de copiar los ficheros resultantes hacia alguno de los directorios de esta lista, con el fin de que en el futuro podamos cargarlos, de la misma forma que el resto de los paquetes del sistema, mediante la función “load”. En los sistemas GNU/Linux, se incluye entre estos directorios a la carpeta

```
/home/usuario/.maxima/,
```

⁶<http://maxima.sourceforge.net/docs/manual/es/maxima.html>

por lo que éste constituye un lugar muy apto para guardar nuestros paquetes⁷. Por supuesto, si tenemos permiso de super-usuario, podremos copiar el paquete en alguno de los directorios del sistema listados en "file_search_maxima", lo que sería aconsejable si distintos usuarios de un ordenador desearan utilizar dicho paquete. Y, en cualquier caso, podemos añadir a la lista el directorio que deseemos, utilizando una orden similar a la siguiente:

```
(%i6) append(file_search_maxima, ["/home/usuario/mi_maxima/"])$
```

Por ejemplo, supongamos que contamos con un aula de informática dotada de ordenadores con sistema GNU/Linux y deseamos poner a disposición de los alumnos el paquete `areas.lisp` que hemos creado. Simplemente, debemos crear en cada ordenador el directorio `/home/alumno/.maxima/` (suponiendo que `alumno` sea el nombre de usuario con el que acceden al sistema nuestros estudiantes) y copiar al mismo el fichero `areas.lisp`. A partir de ese momento, los alumnos podrán acceder a todas las funciones, variables y en general a las expresiones definidas en este paquete a través de la orden:

```
(%i7) load(areas);
(%o12)

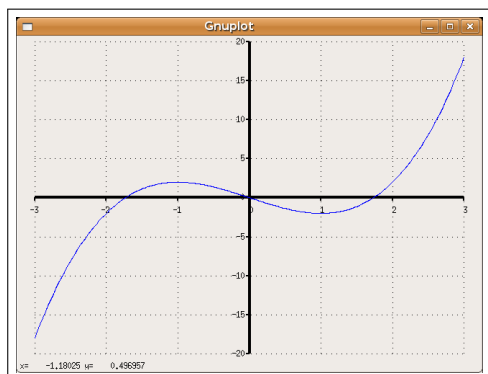
/home/alumno/.maxima/areas.lisp
```

Existe incluso la posibilidad de cargar, de forma automática, expresiones, definir funciones, y en realizar configurar cualquier tipo de parámetro que determinará el modo de funcionamiento de *Maxima*. Para ello podemos usar el fichero "maxima-init.mac", que se carga automáticamente cada vez que se empieza a ejecutar *Maxima* y por tanto se puede utilizar para personalizar el entorno. Este fichero debe estar alojado en cualquier directorio que esté en "file_search_maxima", para asegurarnos de que sea hallado y ejecutado por *Maxima*.

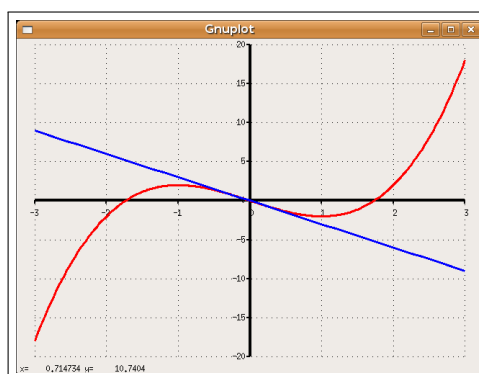
Por ejemplo, podemos definir variables que nos permitan adaptar fácilmente el formato de los gráficos según nuestras preferencias. Para ello, introducimos (utilizando cualquier editor de textos) el siguiente código *Maxima* en el fichero "maxima-init.mac".

```
estilo_graficos_1:"set xzeroaxis linetype -1 linewidth 2;
set yzeroaxis linetype -1 linewidth 2; set xtics axis nomirror;
set ytics axis nomirror; unset x2tics; set grid; unset border;
unset key;"$
estilo_curva_1:"with lines linetype 1 linewidth 3"$
estilo_curva_2:"with lines linetype 3 linewidth 3"$
```

⁷Obsérvese que, puesto que su nombre comienza por un punto, esta es en sistemas GNU/Linux una carpeta oculta; podremos visualizarla si marcamos la opción "Mostrar ficheros ocultos" en nuestro navegador de archivos



(a) Usando un estilo de gráficos personalizado



(b) Y usando estilos de curvas personalizados

Figura 6.1: Personalización de gráficos

De esta forma, en cualquier sesión de *Maxima* (o *wxMaxima*) que ejecutemos, las siguientes órdenes producirán los resultados que se muestran en las figuras 6.1a y 6.1b.

```
(%i16) f(x):=x^3-3*x;
```

```
(%o16)
```

$$f(x) := x^3 - 3x$$

```
(%i17) plot2d(f(x), [x,-3,3], [gnuplot_preamble, estilo_graficos_1]);
```

```
(%o17)
```

```
(%i18) plot2d([f(x),-3*x], [x,-3,3], [gnuplot_preamble, estilo_grafico_1,
[gnuplot_curve_styles, [estilo_curva_1,estilo_curva_2]]);
```

```
(%o18)
```

6.2. *Maxima* como lenguaje de programación

Como hemos visto, podemos crear programas en el lenguaje de *Maxima*, normalmente editándolos desde un fichero de texto, aunque también existe la posibilidad de grabar una sesión interactiva, por ejemplo utilizando el menú "Archivo" → "Guardar como" de *wxMaxima*. En lo que sigue, analizaremos con mayor profundidad este lenguaje.

Éste se trata de un lenguaje interpretado, en el sentido de que (a diferencia de los lenguajes compilados, como C, C++, Fortran, etc) los programas no pueden ejecutarse directamente y necesitan que *Maxima* actúe como intérprete. Esto tiene algunas ventajas (por ejemplo, el ciclo de desarrollo se simplifica al no necesitar de una etapa de compilación) pero también algunos inconvenientes (por ejemplo, *Maxima* no puede competir en velocidad de ejecución con lenguajes compilados).

En (el lenguaje) *Maxima* existen distintos tipos de expresiones y el proceso de programación consiste en la creación y manipulación de éstas para obtener nuestros objetivos. Ya conocemos gran parte de éstas, por ejemplo expresiones con operadores...

```
(%i1)  expr : 1 + %pi + %e + x + a;
```

```
(%o1)
```

$$x + a + \pi + e + 1$$

```
(%i2)  expr, a--(x+1);
```

```
(%o2)
```

$$\pi + e$$

expresiones con funciones...

```
(%i3)  f(x):=x^2+1;
```

```
(%o3)
```

$$f(x) := x^2 + 1$$

```
(%i4)  f(x)+f(1/x);
```

```
(%o4)
```

$$x^2 + \frac{1}{x^2} + 2$$

```
(%i5)  ratsimp(%);
```

```
(%o5)
```

$$\frac{x^4 + 2x^2 + 1}{x^2}$$

expresiones con listas, condicionales...

```
(%i6)  lista:[["Juan","V"], ["María","M"], ["Antonio","V"]];
```

```
(%o6)
```

```
[[Juan,V],[María,M],[Antonio,V]]
```

```
(%i7)  saludar(persona):=if(second(persona)="V" ) then print("Hola,
Sr.", first(persona)) else print("Hola, Sra.", first(persona))$
```

```
(%i8)  map(saludar,lista)$
```

```
Hola, Sr. Juan
Hola, Sra. María
Hola, Sr. Antonio
```

La función “print”, utilizada en el ejemplo anterior, evalúa y muestra en pantalla los argumentos que recibe.

En el resto de este capítulo, analizaremos más en profundidad de estas expresiones, como las funciones, y estudiaremos la forma de definir otras, como bucles que nos permitan realizar una tarea repetidamente.

6.2.1. Bucles

El tipo de bucles más usuales, similares al de otros lenguajes de programación como C, C++, Fortran, Python, etc, se construyen mediante la palabra clave “for”. Para ello, se define el valor inicial de una variable que actuará como índice y que será incrementada en cada ciclo del bucle hasta que ésta alcance o supere un valor final, especificado mediante la palabra “thru”. En cada ciclo, podemos realizar una tarea indicada mediante la palabra clave “do”. Veamos el siguiente ejemplo:

```
(%i1) for i:1 thru 5 do print(i, "al cuadrado es igual a", i^2);
1 al cuadrado es igual a 1
2 al cuadrado es igual a 4
3 al cuadrado es igual a 9
4 al cuadrado es igual a 16
5 al cuadrado es igual a 25
(%o1)
done
```

La cantidad que debe ser incrementada la variable índice en cada etapa se puede determinar mediante la palabra clave “step”:

```
(%i2) for a:0 thru -5 step -2 do display(a);
\mathrm{a}=0
\mathrm{a}=-2
\mathrm{a}=-4
(%o2)
done
```

Como se ve, se utilizó la función “display” para mostrar el valor de la variable. Los bucles “for” tienen dos formas equivalentes, que se construyen utilizando las palabras clave “while” (mientras que) y “unless” (salvo que), en lugar de “thru”.

```
(%i3) s:0;
(%o3)
```



```
(%i4) for i:5 while i>0 step -1 do s:s+i;
(%o4)
done
```

```
(%i5) s;
(%o5)
```

15

```
(%i6) for i:1 unless i^3>100 do display(i^3);
1^3=1
2^3=8
3^3=27
4^3=64
(%o6)
done
```

A la hora de construir bucles más complejos, nos será muy útil la posibilidad de ejecutar varias expresiones dentro del cuerpo de los mismos:

```
(%i7) f(x):=%e^x$
(%i8) poli_taylor:0$
(%i9) for n:0 thru 5 do (
    derivada: diff(f(x), x, n),
    derivada_en_0: subst (x=0, derivada),
    poli_taylor: poli_taylor + derivada_en_0/n! * x^n
)$
(%i10) poli_taylor;
(%o10)
```

$$\frac{x^5}{120} + \frac{x^4}{24} + \frac{x^3}{6} + \frac{x^2}{2} + x + 1$$

```
(%i11) is(poli_taylor = taylor(f(x),x,0,5)) /* Usamos la función
taylor() de Maxima para comprobar que todo ha ido bien */;
(%o11)
```

true

Como se puede apreciar en el ejemplo anterior, el cuerpo del bucle se introduce entre paréntesis, separando por comas con las distintas expresiones contenidas en él.

Si lo deseamos, podemos omitir tanto las condiciones de inicialización como las de finalización, quedando únicamente en este caso la palabra “do” y seguida del cuerpo

del bucle. Si no queremos entrar en un bucle infinito, debemos utilizar la palabra clave “return” como puerta de salida, lo cual se ilustra en el siguiente ejemplo, en el que se calcula una fracción que constituye una aproximación del número e , como límite de la sucesión:

$$\lim_{n \rightarrow +\infty} \left(1 + \frac{1}{n}\right)^n.$$

```
(%i12)  n:1$ x:1$ epsilon:10^(-3)$
(%i13)  do (
          y:(1+1/n)^n,
          if abs(x-y)<epsilon then return(y) else (x:y, n:n+1)
        );
(%o13)

28313468473157736370011296127792731029354930758695159595008
-----
10555134955777783414078330085995832946127396083370199442517

(%i14)  %,numer;
(%o14)

2,6824354773085304
```

Como se puede apreciar, partiendo de $x = 1$, se realizan iteraciones hasta que dos términos consecutivos de la sucesión se encuentran a una distancia menor que epsilon, en cuyo caso la orden “return” provoca la salida del bucle y éste devuelve el valor alcanzado. En el ejemplo anterior, es interesante observar cómo, de nuevo, utilizamos los paréntesis para poder introducir dos expresiones en el cuerpo de la sentencia “else”.

Como se puede comprobar en la documentación de *Maxima*, todavía existen más tipos de bucles. Terminaremos la sección con el siguiente ejemplo:

```
(%i15)  colores:["amarillo","azul","rojo","verde"]$
(%i16)  for c in colores do print("Me gusta el color",c)$
Me gusta el color amarillo
Me gusta el color azul
Me gusta el color rojo
Me gusta el color verde
```

6.2.2. Funciones

En las funciones que hemos definido hasta ahora, solamente había una expresión a la derecha del operador “:=”

```
(%i17)  f(x):=x/y;
(%o17)
```

$$f(x) := \frac{x}{y}$$

Pero si deseamos utilizar las funciones para definir procedimientos mucho más elaborados, que engloben secuencias de expresiones, podemos contar con la orden “block”, por ejemplo:

```
(%i18) f(x,y):=block([d:x-y],
    if(d>0) then print("Diferencia positiva") else
    if(d<0) then print("Diferencia negativa") else print("Iguales"),
    d)$
(%i19) f(2,5);
Diferencia negativa
(%o19)
-3

(%i20) f(1,1);
Iguales
(%o20)
0
```

```
(%i21) f(3,-2)$
Diferencia positiva
```

Como se puede observar, dentro de la estructura “block” hemos definido una variable local (v) que hemos inicializado con un valor ($x-y$). En general, la una estructura “block” tiene la siguiente forma:

$$f(x_1, x_2, \dots, x_N) := \text{block}([v_1, v_2, \dots, v_N], \text{expr}_1, \text{expr}_2, \dots, \text{expr}_N)$$

Se puede entender como una secuencia de expresiones precedida por una serie de variables (entre corchetes) que serán locales al bloque, no teniendo ningún tipo de efecto fuera del mismo. Cualquier otra variable utilizada dentro del bloque se considera global. El valor devuelto por las funciones así definidas es igual al de la última expresión, salvo que alguna sentencia “return” situada en el cuerpo del bloque devuelva un valor diferente.

En el siguiente ejemplo, se define una función que calculará, mediante el método de bipartición, un cero, c , de una función en un intervalo $[a, b]$, deteniéndose cuando $|f(c)| < \epsilon$:

```
(%i22) biparticion(f_objetivo,a,b,epsilon) := block(
    [ ],
    if( sign(f_objetivo(a)) = sign(f_objetivo(b))) then (
        print("ERROR, f tiene el mismo signo en a y en b"),
        return(false)
    ) else
    do (
        c: (a+b)/2,
```

```
        if(abs(f_objetivo(c))<epsilon) then return(c),  
        if( sign(f_objetivo(a)) = sign(f_objetivo(c)) ) then a:c  
else b:c  
    )  
    )$
```

```
(%i23) f(x):=(2*x)^2-exp(-x)$  
(%i24) biparticion(f,0,1,0.01), numer;  
(%o24)
```

0,40625

```
(%i25) g(x):=x^2-x-2$  
(%i26) biparticion(g,0,10,10^-6), numer;  
(%o26)
```

1,9999998807907104

```
(%i27) h(x):=1-x^2*atan(x)$  
(%i28) biparticion(h,-5,5,10^-6), numer;  
(%o28)
```

1,0966706275939941

Problema resuelto 1: estudio y representación de una función

Se presenta a continuación un problema que puede resumir una buena parte de lo que se ha estudiado en el capítulo dedicado al Cálculo Infinitesimal. A lo largo de las siguientes páginas, se mostrará solamente la salida de *Maxima*, sin detallar el uso de los menús de *wxMaxima*. No obstante, se recomienda el uso de estos menús siempre que los mismos puedan ser de utilidad para el cálculo de límites, derivadas, integrales, representación de funciones, etc.

Problema:

Sea

$$f(x) = \frac{x^2 - 9}{x}$$

- Halla los puntos de corte de $f(x)$ con los ejes de coordenadas
- Presenta una lista con los valores de $f(n)$, $n = 100, 200, 300, \dots, 1500$. ¿Cuál piensas que es el límite de $f(x)$ en $+\infty$? Confírmalo con *wxMaxima*.
- Estudia las asíntotas verticales, horizontales y oblicuas de $f(x)$.
- Representa de forma conjunta la gráfica de $f(x)$ y de la recta $y = x$.
- Halla la función derivada de $f(x)$ y la recta tangente a $f(x)$ en el punto $a = 3$. Representar de forma conjunta a la gráfica de $f(x)$ y la de esta recta tangente.
- Calcula una primitiva de $f(x)$.
- Sea A el área limitada por la gráfica de $f(x)$ y el eje OX entre $x = 3$ y $x = 4$. ¿Es A mayor que uno?

Puntos de corte

Comenzamos definiendo $f(x)$:

```
(%i1) f(x) := (x^2-9)/x;
```

```
(%o1)
```

$$f(x) := \frac{x^2 - 9}{x}$$

Sus puntos de corte con OX verifican $f(x) = 0$:

```
(%i2) solve(f(x)=0,x);
(%o2)
```

$[x = -3, x = 3]$

así que tenemos $(-3, 0)$ y $(3, 0)$. Los puntos de corte con OY verifican $x = 0$, pero

```
(%i3) f(0);
Division by 0
#0: f(x=0)
- an error. Quitting. To debug this try debugmode(true);
```

Evidentemente, el punto $x = 0$ no está en el dominio, pues el denominador se hace cero (y *Maxima* nos da el error anterior).

Valores de $f(n)$

Podríamos calcular uno a uno los valores de $f(n)$ pero es más rápido, como vimos en el tema anterior, utilizar la función “map”, que es especialmente apropiada para crear tablas de valores:

```
(%i4) lista:makelist(100*n,n,1,15);
(%o4)
```

$[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500]$

```
(%i5) tabla:map(f,lista);
(%o5)
```

$\left[\frac{9991}{100}, \frac{39991}{200}, \frac{29997}{100}, \frac{159991}{400}, \frac{249991}{500}, \frac{119997}{200}, \frac{489991}{700}, \frac{639991}{800}, \frac{89999}{100}, \frac{999991}{1000}, \frac{1209991}{1100}, \frac{479997}{400}, \frac{16899}{130} \right]$

```
(%i6) %,numer;
(%o6)
```

$[99,9099999999999997, 199,9550000000000001, 299,9700000000000003, 399,9775000000000002, 499,9820000000000000]$

Parece que

$$\lim_{x \rightarrow +\infty} f(x) = +\infty,$$

pero es mejor confirmarlo con *Maxima*. Efectivamente:

```
(%i7) limit(f(x),x,inf);
(%o7)
```

$$\infty$$

Asíntotas

Como vemos a continuación, $f(x)$ tiene una asíntota vertical en $x = 0$, de hecho

$$\lim_{x \rightarrow 0^+} f(x) = -\infty \quad \lim_{x \rightarrow 0^-} f(x) = +\infty$$

```
(%i8) limit(f(x),x,0);
(%o8)
```

$$\text{und}$$

```
(%i9) limit(f(x),x,0,plus);
(%o9)
```

$$-\infty$$

```
(%i10) limit(f(x),x,0,minus);
(%o10)
```

$$\infty$$

Por otra parte, $f(x)$ no tiene asíntotas horizontales en $+\infty$ (como vimos, el límite es $+\infty$) y tampoco en $-\infty$, como vemos ahora:

```
(%i11) limit(f(x),x,minf);
(%o11)
```

$$-\infty$$

Y, por otro lado, puesto que

```
(%i12) limit(f(x)/x,x,inf);
(%o12)
```

$$1$$

$f(x)$ tiene una asíntota oblicua, de la forma $y = 1 \cdot x + a$, donde a viene dado por el siguiente límite:


```
(%i13) limit(f(x)-1*x,x,inf);
(%o13)
```

0

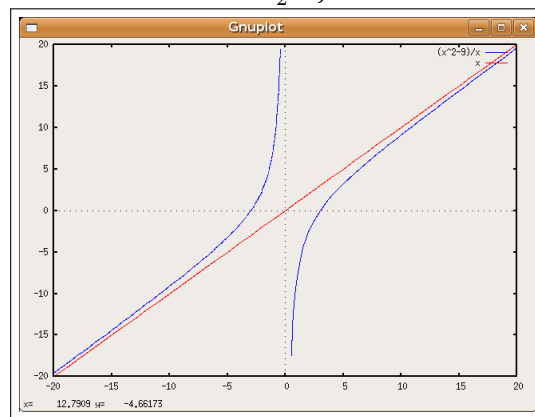
Por tanto, la asíntota oblicua es $y = x$.

Gráfica

Representaremos $f(x)$ junto con su asíntota, $y = x$. A base de prueba y error, se encontrarán los intervalos de x e y adecuados, en este caso se tomará $x \in [-20, 20]$, $y \in [-20, 100]$:

```
(%i14) plot2d([f(x),x],[x,-20,20],[y,-20,20],[gnuplot_preamble,"set
zeroaxis;"]);
(%o14)
```

Figura A.1: Gráfica de la función $\frac{x^2-9}{2}$ junto a su asíntota oblicua $y = x$



Derivada y recta tangente

A continuación, calculamos la derivada de $f(x)$, definimos el punto $a = 3$ y hallamos $f'(a)$.

```
(%i15) diff(f(x),x);
(%o15)
```

$$2 - \frac{x^2 - 9}{x^2}$$

```
(%i16) ratsimp(%);
(%o16)
```

$$\frac{x^2 + 9}{x^2}$$

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

$$3$$

```
(%i18) "(diff(f(x),x)),x=a;
(%o18)
```

$$2$$

Obsérvese que (como se comentó en los apuntes de este tema) antes de sustituir el valor $x = a$ en la derivada, es necesario utilizar un doble operador comilla ("), que se obtiene pulsando dos veces en la tecla ?. Este operador se puede interpretar como «el resultado de evaluar la siguiente expresión» (en este caso, la derivada de $f(x)$). En general, la última entrada se puede interpretar como «sustituir $x = a$ en el resultado de calcular la derivada de $f(x)$ ».

También podríamos haber utilizado este operador para definir una función, a la que podríamos llamar $Df(x)$, que representaría a la primera derivada de $f(x)$. Esto lo haremos más abajo, como ejemplo. Y como se puede ver aprovecharemos para simplificar la derivada (con la función "ratsimp" antes de definir $Df(x)$). Así, la definición se puede leer como « $Df(x)$ se define como el resultado (*doble comilla*) de simplificar (*ratsimp*) la derivada (*diff*) de $f(x)$ »

Una vez que conocemos la pendiente, $m = f'(a) = 2$, podemos escribir la recta tangente como

$$y - f(a) = m * (x - a)$$

```
(%i19) Df(x):="(ratsimp(diff(f(x),x)));
(%o19)
```

$$Df(x) := \frac{x^2 + 9}{x^2}$$

```
(%i20) m:Df(a);
(%o20)
```

$$2$$

```
(%i21) y-f(a)=m*(x-a);
(%o21)
```

$$y = 2(x - 3)$$

```
(%i22) expand(%);
(%o22)
```

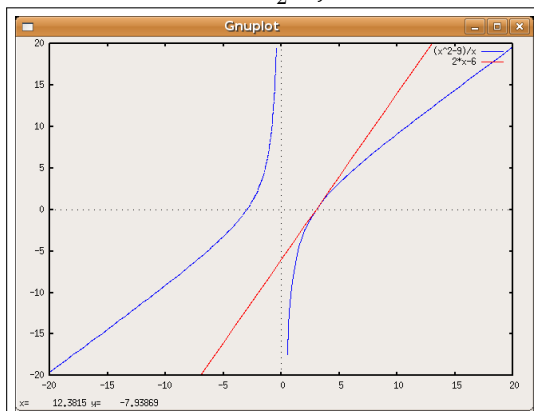
$$y = 2x - 6$$

Y (utilizando el menú de *wxMaxima*, si así lo deseamos), podemos representar la gráfica:

```
(%i23) plot2d([f(x), 2*x-6], [x, -20, 20], [y, -20, 20], [gnuplot_preamble, "set
zeroaxis;"]);
(%o23)
```

El resultado se puede apreciar en la figura A.2

Figura A.2: Gráfica de la función $\frac{x^2-9}{2}$ junto su recta tangente en $x = 3$



Primitiva y área

Calcular una primitiva es fácil. Recurramos o no al sistema de menús de *wxMaxima*, tendremos:

```
(%i24) integrate(f(x), x);
(%o24)
```

$$\frac{x^2}{2} - 9 \log x$$

Para el cálculo de áreas, observamos que $f(x) > 0$ en $[3, +\infty)$, pues hemos visto que $f(3) = 0$ y si $x > 3$ el numerador y el denominador de $f(x)$ son estrictamente positivos

(en *Maxima* esto se puede comprobar con las órdenes “assume” e “is”, como vemos más abajo). Por lo tanto, el área del recinto coincide con la siguiente integral definida:

$$\int_3^4 f(x)dx$$

```
(%i25) f(x);
```

```
(%o25)
```

$$\frac{x^2 - 9}{x}$$

```
(%i26) numerador:num(f(x));
```

```
(%o26)
```

$$x^2 - 9$$

```
(%i27) denominador:denom(f(x));
```

```
(%o27)
```

$$x$$

```
(%i28) assume(x>3);
```

```
(%o28)
```

$$[x > 3]$$

```
(%i29) is(numerador>0);
```

```
(%o29)
```

true

```
(%i30) is(denominador>0);
```

```
(%o30)
```

true

```
(%i31) forget(x>3);
```

```
(%o31)
```

$$[x > 3]$$

```
(%i32) integrate(f(x),x,3,4);
```

```
(%o32)
```

$$-9 \log 4 + 9 \log 3 + \frac{7}{2}$$

```
(%i33) %,numer;
```

(%o33)

0,910861347933972

Como se puede observar, el área del recinto es menor que uno.

Ejercicio

- Representar gráficamente el recinto en el intervalo $x \in [3, 4]$
- Colorear la región del plano situada bajo la curva $f(x)$. *[Difícil! Habría que utilizar opciones «avanzadas» de Gnuplot (ver documentación en internet)]*

B

Apéndice

Problema resuelto 2: estudio de una función definida a trozos

El siguiente ejemplo es muy similar y, si se complica con respecto al anterior, es solamente por el hecho de que se trata de una función a trozos.

Problema:

Dada la función

$$f(x) = \begin{cases} \frac{1}{1+x^2} & \text{si } x \leq 1 \\ 1 + \ln x & \text{si } x > 1 \end{cases}$$

1. Estudiar su dominio, puntos de corte y asíntotas.
2. Analizar su continuidad y su derivabilidad, calcular su función derivada primera.
3. Determinar sus intervalos de crecimiento y decrecimiento, así como sus máximos y mínimos relativos.
4. Representar su gráfica.

Dominio, puntos de corte, asíntotas

Definimos:

```
(%i1) g(x):=1/(1+x^2);  
(%o1)
```

$$g(x) := \frac{1}{1+x^2}$$

```
(%i2) h(x):=1+log(x);  
(%o2)
```

$$h(x) := 1 + \log x$$

El dominio de g es todo \mathbb{R} pues, evidentemente, $1+x^2 \neq 0 \forall x \in \mathbb{R}$. Pero, por si hubiera alguna duda, el alumno podría utilizar *Maxima* para comprobar que no existe ninguna solución real (todas sus soluciones son imaginarios puros):

```
(%i3) solve(1+x^2=0);
```

```
( %o3)
```

$$[x = -i, x = i]$$

```
( %i4) solve(1+x^2=0);
```

```
( %o4)
```

$$[x = -i, x = i]$$

Maxima es incluso capaz de asegurarnos que $1 + x^2$ es estrictamente positivo (independientemente del valor de x) si utilizamos la función `is`:

```
( %i5) is(1+x^2>0);
```

```
( %o5)
```

true

Por otra parte, la función $h(x)$ solamente está bien definida para aquellos valores de x para los que tiene sentido $\log(x)$, es decir, el dominio de h es $\{x \in \mathbf{R} / x > 0\}$. Luego

El dominio de la función $f(x)$ es todo \mathbf{R} , pues cuando $x \leq 1$ es igual a g , que está perfectamente definida, y cuando $x > 1$ coincide con $h(x)$, que no tiene ningún problema para estos valores de x .

Para estudiar los puntos de corte con el eje de las abscisas, planteamos

```
( %i7) solve(g(x)=0,x);
```

```
( %o7)
```

$$[]$$

```
( %i8) solve(h(x)=0,x);
```

```
( %o8)
```

$$[x = e^{-1}]$$

El único punto de corte es $x = \frac{1}{e}$. En cuanto a posibles puntos de corte con el eje vertical, cuando $x = 0$ nuestra función toma el valor $f(0) = g(0) = 1$:

```
( %i9) g(0);
```

```
( %o9)
```

$$1$$

En definitiva:

Los puntos de corte de $f(x)$ son $(0, 1)$ y $(\frac{1}{e}, 0)$.

En nuestro caso, la función $g(x)$ no tiene ninguna asíntota vertical, porque su denominador es siempre distinto de cero. La función $h(x)$ tendría una asíntota vertical en $x = 0$, debido al logaritmo:

```
(%i10) limit(h(x), x, 0, plus);
(%o10)
```

$-\infty$

Pero esto no afecta a $f(x)$ ya que, en los alrededores de $x = 0$, esta función no toma los valores de $h(x)$, sino de $g(x)$. Por lo tanto, f no tiene ninguna asíntota vertical.

Con respecto a asíntotas horizontales, tendremos que estudiar límites de $f(x)$ cuando $x \rightarrow -\infty$ (en cuyo caso $f = g$) y cuando $x \rightarrow +\infty$ (en cuyo caso $f = h$),

```
(%i11) limit(g(x), x, minf);
(%o11)
```

0

```
(%i12) limit(h(x), x, inf);
(%o12)
```

∞

Por lo tanto, podemos concluir que:

$f(x)$ no tiene ninguna asíntota vertical y tiene una asíntota horizontal (la recta $y = 0$) cuando $x \rightarrow -\infty$.

Continuidad, derivabilidad

Las funciones $g(x)$ y $h(x)$ son continuas dentro de sus respectivos dominios, por lo tanto f es continua salvo, eventualmente, en $x = 1$, el punto que divide las regiones donde f toma los valores de g y de h . Pero los límites laterales de f en este punto son distintos, pues

```
(%i13) limit(g(x), x, 1, minus);
(%o13)
```

$\frac{1}{2}$

```
(%i14) limit(h(x), x, 1, plus);
(%o14)
```

1

en consecuencia, f no es continua en $x = 1$ (tendrá una discontinuidad de salto).

Además, f no es derivable en $x = 1$ (por no ser continua) pero sí en el resto de R (pues tanto g como h lo son para valores de x que estén dentro de sus respectivos dominios). Puesto que

```
(%i15) diff(g(x), x);
(%o15)
```

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

```
(%i16) diff(h(x), x);
(%o16)
```

$$\frac{1}{x}$$

podemos concluir que la función derivada de f es:

$$f'(x) = \begin{cases} -\frac{2x}{(x^2 + 1)^2} & \text{si } x < 1 \\ \frac{1}{x} & \text{si } x > 1 \end{cases}$$

Crecimiento, extremos relativos

El crecimiento de f depende del signo de la función derivada, $f'(x)$, calculada en el apartado anterior. Cuando $x > 1$, obviamente, $\frac{1}{x} > 0$ y por lo tanto, $f'(x) > 0$. Aunque es un caso tan sencillo que no merece la pena recurrir al ordenador, puede servir como ilustración la forma en que se podría comprobar lo anterior, utilizando los comandos “assume” e “is”:

```
(%i22) assume(x>1);
(%o22)
```

$$[x > 1]$$

```
(%i23) is(1/x>0);
(%o23)
```

true

cuando $x < 1$, el asunto es diferente, pues el signo de $f'(x)$ dependerá de la expresión

$$-\frac{2x}{(x^2 + 1)^2}, \quad (\text{B.1})$$

que depende del signo de $-2x$ (puesto que el denominador es siempre estrictamente positivo), siendo, negativo cuando $x > 0$ y positivo en caso contrario.

Por lo tanto

f es creciente en $(-\infty, 0) \cup (1, +\infty)$ y decreciente en $(0, 1)$.

Para hallar máximos y mínimos relativos, calculemos los puntos críticos de f :

```
(%i24) solve( diff(g(x), x)=0, x);
```

```
(%o24)
```

$$[x = 0]$$

```
(%i25) solve( diff(h(x), x)=0, x);
```

```
(%o25)
```

$$[]$$

Esto es, h no tiene puntos críticos y g tiene un punto crítico, $x = 0$. Cuando $x = 0$ la función f es igual a g , luego éste es un punto crítico de f . Para saber si se trata de un máximo o un mínimo, podemos estudiar el signo de la segunda derivada:

```
(%i26) diff(g(x), x, 2);
```

```
(%o26)
```

$$\frac{8x^2}{(x^2 + 1)^3} - \frac{2}{(x^2 + 1)^2}$$

```
(%i27) %, x=0;
```

```
(%o27)
```

$$-2$$

Por lo tanto, $f''(0) = g''(0) = -2 < 0$, es decir, f tiene un máximo relativo en $x = 0$.

Representación gráfica

Para representar la gráfica de f , podemos definir:

```
(%i33) f(x) := if(x<1) then g(x) else h(x);
```

```
(%o33)
```

$$f(x) := \text{if } x < 1 \text{ then } g(x) \text{ else } h(x)$$

El problema es que *Maxima* no está preparado para representar directamente este tipo de funciones, siendo necesario encapsular $f(x)$ en una expresión de la forma $((f(x)))$.

```
(%i34) plot2d(f(x), [x, -5, 5]);
```

```
Maxima was unable to evaluate the predicate:
```

```
** error while printing error message **
```

```
Maxima was unable to evaluate the predicate:~%~M
```

```
#0: f(x=x)
```

```
- an error. Quitting. To debug this try debugmode(true);
```

```
(%i35) plot2d('f(x)', [x, -5, 5], [gnuplot_preamble, "set zeroaxis"],  
[gnuplot_term, ps], [gnuplot_out_file, "grafica.eps"]);  
(%o35)
```

El resultado se puede observar en la figura B.1.

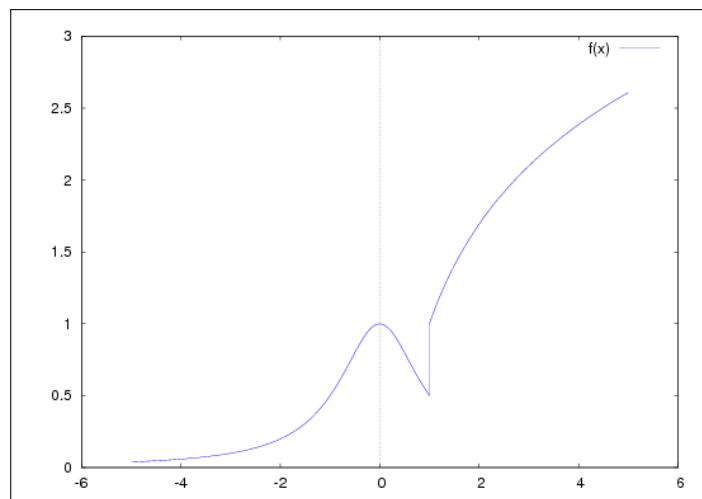


Figura B.1: Gráfica de $f(x)$

Observaciones:

- Como se puede observar, se ha optado en este caso por grabar la gráfica hacia un fichero `.eps` (postscript encapsulado) mediante la opción “Gráfico al archivo” de *wxMaxima*.
- El algoritmo que ha empleado *Maxima* para la representación gráfica de la función a trozos $f(x)$ tiene un problema: erróneamente, representa una línea vertical en $x = 1$ que oculta la existencia de una discontinuidad de salto en este punto. El alumno debe ser suficientemente astuto como para interpretar esa línea vertical como una discontinuidad de salto.

El módulo “inecuaciones” define las dos funciones siguientes en *Maxima*:

- `signo(f, x)`. Estudia el signo de una función racional. Devuelve dos listas:

1. La primera contiene una lista de intervalos abiertos, representados a través de una de pares $[a, b]$, por ejemplo el conjunto de intervalos

$$\{(-\infty, -1), (-1, 1), (1, +\infty)\}$$

se representaría como

$$[[\text{minf}, -1], [-1, 1], [1, \text{inf}]].$$

Por ejemplo:

```
(%i194) signo(x^2-9, x);
```

```
(%o194)
```

$$[[[-\infty, -3], [-3, 3], [3, \infty]], [+ , - , +]]$$

2. La segunda define el signo de f en cada uno de los intervalos anteriores, denotado a través de una lista de signos “+” ó “-”. Así, en el ejemplo anterior, la lista $[+, -, +]$ significaría “ $f > 0$ en $(-\infty, 1) \cup (1, +\infty)$ y $f < 0$ en $(-1, 1)$ ”. Por ejemplo:

```
(%i196) resolver_inecuac(x^2-9>0, x);
```

```
(%o196)
```

$$[[[-\infty, -3], [3, \infty]]]$$

- `resolver_inecuac(inecuacion, x)`. Devuelve los intervalos en los que se verifica una inecuación. En principio, ésta debe ser de tipo racional. El estudio de los extremos de los intervalos no está hecho

Más ejemplos

```
(%i197) load(inecuaciones)$
```

```
(%i198) signo(1-x, x);
```

```

(%o198)

$$[[[-\infty, 1], [1, \infty]], [+,-]]$$


(%i199) signo((x-1)/x,x);
(%o199)

$$[[[-\infty, 0], [0, 1], [1, \infty]], [+,-,+]]$$


(%i200) f(x):=(x-1)/(x^3-3)$
(%i201) signo(f(x),x);
(%o201)

$$\left[ \left[ [-\infty, 1], \left[ 1, 3^{\frac{1}{3}} \right], \left[ 3^{\frac{1}{3}}, \infty \right] \right], [+,-,+] \right]$$


(%i202) resolver_inecuac(x^2-1<0,x);
(%o202)

$$[[-1, 1]]$$


(%i203) resolver_inecuac((x^4-16)/(x^2-1)>0,x);
(%o203)

$$[[-\infty, -2], [-1, 1], [2, \infty]]$$


```

Código módulo "inecuaciones"

```
/*          COPYRIGHT NOTICE
```

Copyright (C) 2007 Rafa Rodríguez Galván

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

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

```
*/
```

```
/*
```

```
signo(f,x)
```

```
Estudia el signo de una función racional, f, respecto de la variable
x.
```

```
Devuelve dos listas:
```

- La primera contiene una lista de intervalos abiertos, representados como $[\text{minf}, -1]$, $[-1, 1]$, $[1, \text{inf}]$
- La segunda contiene el signo en cada intervalo, como $["+", "-", "+"]$

```
*/
```

```
signo(f,x):=block(
```

```
[
```

```
intervalos:[], pts_test:[], signos:[],
```

```
numerador, denominador, raices, raices_numer, raices_denom,
```

```
algexact:true, realonly:true /* Para que algsys() llame a solve() [real
],
```

```
/* Función de apoyo, devuelve true si expr es una constante */
```

```
es_constante(expr):=block(
```

```
if(subst(x=1,expr)=expr) then return(true) else return(false)
```

```
),
```

```
/* Función de apoyo, devuelve una lista de las raíces de un polinomio *
```

```
lista_raices(polinomio):=block(
```

```
[raices:[],
```

```
if(not es_constante(polinomio)) then (
```

```
raices:algsys([polinomio],[x]), /* -> Por ejemplo: [[x=-1], [x=
```

```
raices:map(lambda([lista], rhs(first(lista))), raices) /* -> [-1,
```

```
),
```

```
return(raices)
```

```
),
```

```
/* Hallar las raíces del numerador y del denominador */
```

```
numerador:num(f),
```

```
denominador:denom(f),
```

```
raices_numer:lista_raices(numerador),
```

```
raices_denom:lista_raices(denominador),
```

```
raices:sort( append(raices_numer, raices_denom) ),
```

```
/* Construir lista de intervalos y puntos-test para signos */
```

```
if(raices=[]) then (
```

```
intervalos:[[minf,inf]], pts_test:[0]
```

```
)
```

```

else (
  extremo_derecho:first(raices),
  intervalos:[[minf,extremo_derecho]],
  pts_test:[extremo_derecho-1],
  extremo_izquierdo:extremo_derecho,
  for i:2 thru length(raices) do (
    extremo_derecho:raices[i],
    intervalos:append(intervalos, [[extremo_izquierdo,extremo_derecho]]),
    pts_test:append(pts_test, [(extremo_izquierdo+extremo_derecho)/2]),
    extremo_izquierdo:extremo_derecho
  ),
  intervalos:append(intervalos, [[extremo_izquierdo,inf]]),
  pts_test:append(pts_test, [extremo_izquierdo+1])
),

/* Evaluar el signo en cada intervalo (en el punto-test) */
for punto in pts_test do (
  if(subst(x=punto, f) > 0)
    then signos:append(signos, ["+"])
    else signos:append(signos, ["-"])
  ),
return([intervalos, signos])
)$

/*
resolver_inecuac(inecuacion,x)

Devuelve los intervalos en los que se verifica una inecuación. En
principio, ésta debe ser de tipo racional. El estudio de los extremos
de los intervalos no está todavía hecho.
*/
resolver_inecuac(inecuacion,x):=block(
  [operador:op(inecuacion), solucion:[],
  f, resultado, intervalos, signos, objetivo],
  if(rhs(inecuacion) # 0) then (
    print ("Esta función sólo admite inecuaciones del tipo  $f(x)<0$  ó  $f(x)$ 
    return(false)
  ),
  f:lhs(inecuacion),
  resultado:signo(f,x),
  intervalos:resultado[1], signos:resultado[2],
  if(operador="<") then objetivo:"- " else objetivo:"+ ",
  for i:1 thru length(intervalos) do (

```

```
        if(signos[i]=objetivo) then solucion:append(solucion, [intervalos[i]]
    ),
    return(solucion)
)$
```


D

Apéndice

Un ejemplo: geometría de tortuga

En este ejemplo, se diseña en Maxima una versión (muy particular) de un subconjunto del lenguaje de programación Logo.

Según la wikipedia española “Logo es un lenguaje de alto nivel en parte funcional en parte estructurado, de muy fácil aprendizaje, razón por la cual suele ser el lenguaje de programación preferido para trabajar con niños y jóvenes. Fue diseñado con fines didácticos por Danny Bobrow, Wally Feurzeig y Seymour Papert, los cuales se basaron en la sintaxis del lenguaje Lisp. A pesar de que Logo no fue creado con la finalidad de usarlo para enseñar programación, puede usarse para enseñar la mayoría de los principales conceptos de la programación, ya que proporciona soporte para manejo de listas, archivos y entrada/salida. Logo es uno de los pocos lenguajes de programación con instrucciones en español en algunas de sus versiones. Además, existen varios intérpretes de Logo en español (y más de 160 en total, según constan en el proyecto “Logo Tree”), entre ellos: FMSLogo, LogoWriter, WinLogo, Logo Gráfico, XLogo, MSWLogo y LogoEs. XLogo, MSWLogo y LogoES tienen la particularidad de ser además software libre.”

La característica más destacada de Logo es que nos permite mirar las figuras geométricas del plano como trazos hechos sobre la pantalla por una “tortuga virtual” cuyos movimientos son controlados por el usuario, a través de algunos comandos básicos:

- `avanza(distancia)` Hace que la tortuga avance la distancia indicada
- `derecha(angulo)` Hace que la tortuga gire de rumbo un determinado ángulo a la derecha
- `izquierda(angulo)` Cambia el rumbo un determinado ángulo a la izquierda
- `pinta()` Dibuja el rumbo seguido por la tortuga
- `reinicia()` Reinicia la posición y el rumbo de la tortuga, situándose en el punto (0,0) con rumbo de 90 grados (mirando hacia arriba).

Se ha optado por que las funciones `izquierda` y `derecha` utilicen ángulos sexagesimales, aunque habría sido fácil dejarlos en radianes. Este sencillo esta versión particular de Logo se puede definir de la siguiente forma (la idea sería guardar las siguientes órdenes en un fichero llamado “logo.mac”, para que sea accesible siempre que deseemos):

```
posicion:[0,0]$
ruta:[posicion]$
```

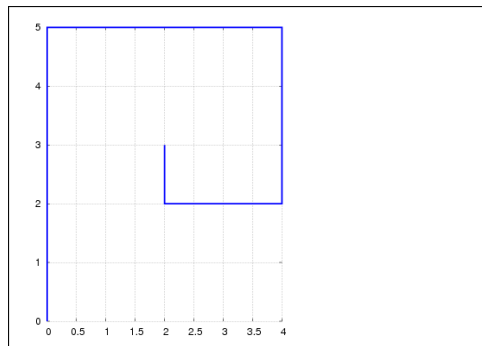


Figura D.1: Una curva generada por la ruta de la “tortuga virtual”

```

rumbo:%pi/2$
a_radianes(angulo_grados):=angulo_grados*%pi/180$
reinicia() := block( posicion:[0,0], ruta:[posicion], rumbo:%pi/2)$
avanza(distancia) := block(
    posicion:posicion+distancia*[cos(rumbo),sin(rumbo)],
    ruta:cons(float(posicion),ruta)
)$
derecha(angulo) := rumbo: rumbo - a_radianes(angulo)$
izquierda(angulo) := rumbo: rumbo + a_radianes(angulo)$
dibuja() := plot2d([discrete, ruta],
    [gnuplot_preamble, "set grid; unset border; unset key; set size ratio -

```

Un ejemplo de su uso: la siguiente sesión produciría el cuadrado que se puede apreciar en la figura D.1a

```

(%i1) load(logo)$
(%i2) avanza(5)$
(%i3) derecha(90)$
(%i4) avanza(4)$
(%i5) derecha(90)$
(%i6) avanza(3)$
(%i7) derecha(90)$
(%i8) avanza(2)$
(%i9) derecha(90)$
(%i10) avanza(1)$
(%i11) dibuja()$

```

Lo anterior se podría haber realizado de forma más rápida utilizando un bucle como el siguiente:

```

(%i12) reinicia()$
(%i13) for i:5 thru 1 step -1 do ( avanza(i), derecha(90) )$
(%i14) dibuja()$

```

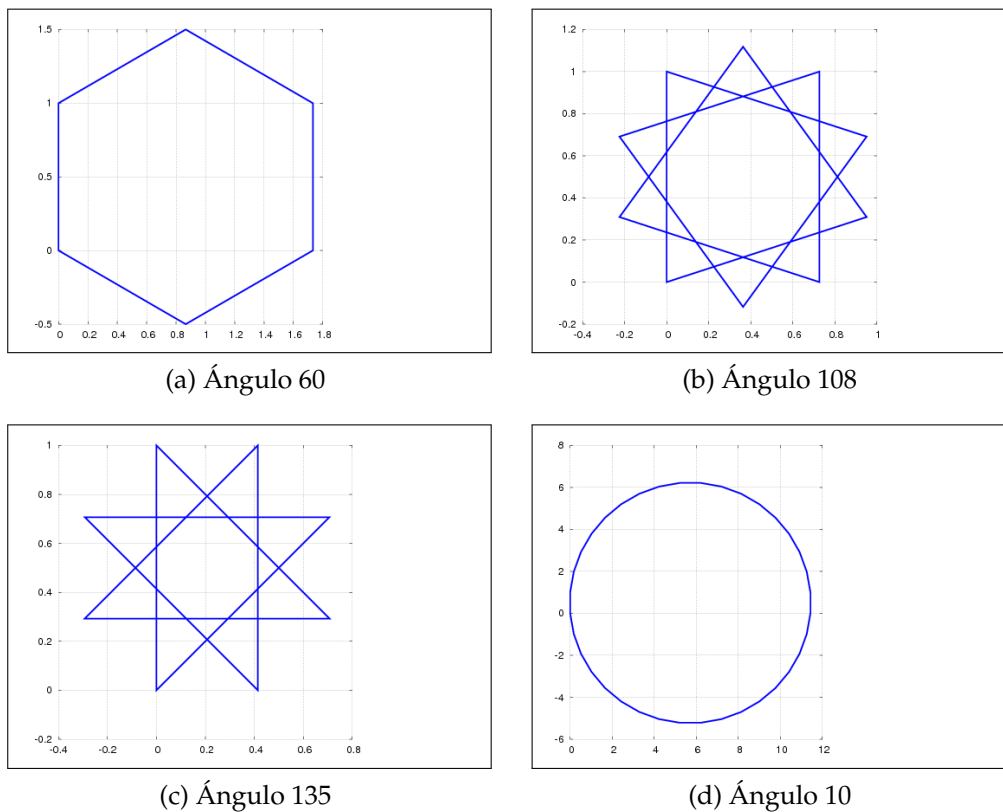


Figura D.2: Algunos resultados de *poligono(lado, angulo)*

Podemos definir funciones, como en el siguiente ejemplo. Los resultados se muestran en la figura D.2:

```
(%i15) poligono(lado, angulo) := block(
    [num_iteraciones:36],
    for i:1 thru num_iteraciones do (avanza(lado), derecha(angulo))
) $
(%i16) reinicia() $
(%i17) poligono(1, 60) $
(%i18) dibuja() $
(%i19) reinicia() $
(%i20) poligono(1, 108) $
(%i21) dibuja() $
(%i22) reinicia() $
(%i23) poligono(1, 135) $
(%i24) dibuja() $
(%i25) reinicia() $
(%i26) poligono(1, 10) $
```

```
(%i27)  dibuja()$
```

En lo que sigue, un ejemplo más avanzado, en el que se define una función que genera la curva de Koch. La sucesión dada por estas curvas converge hacia un objeto no derivable y de longitud infinita.

La función que definimos a continuación tiene la particularidad de llamarse recursivamente a sí misma hasta que el nivel es igual a cero, momento en el que la tortuga avanza y la función retorna al nivel inmediatamente superior.

```
(%i28)  curva_koch(talla,nivel):=block(
        if(nivel=0) then (avanza(float(talla)), return(true)),
        curva_koch(talla/3,nivel-1),
        izquierda(60),
        curva_koch(talla/3,nivel-1),
        derecha(120),
        curva_koch(talla/3,nivel-1),
        izquierda(60),
        curva_koch(talla/3,nivel-1)
    )$
(%i29)  reinicia()$
(%i30)  curva_koch(1,1)$
(%i31)  dibuja()$
(%i32)  reinicia()$
(%i33)  curva_koch(1,2)$
(%i34)  dibuja()$
(%i35)  reinicia()$
(%i36)  curva_koch(1,3)$
(%i37)  dibuja()$
(%i38)  reinicia()$
(%i39)  curva_koch(1,4)$
(%i40)  dibuja()$
```

Por último, utilizamos la función que hemos definido para dibujar un copo de nieve:

```
(%i41)  copo_de_nieve(talla,nivel):=block(
        thru 3 do (curva_koch(talla, nivel), derecha(120))
    )$
(%i42)  reinicia()$
(%i43)  copo_de_nieve(1,5)$
(%i44)  dibuja()$
```

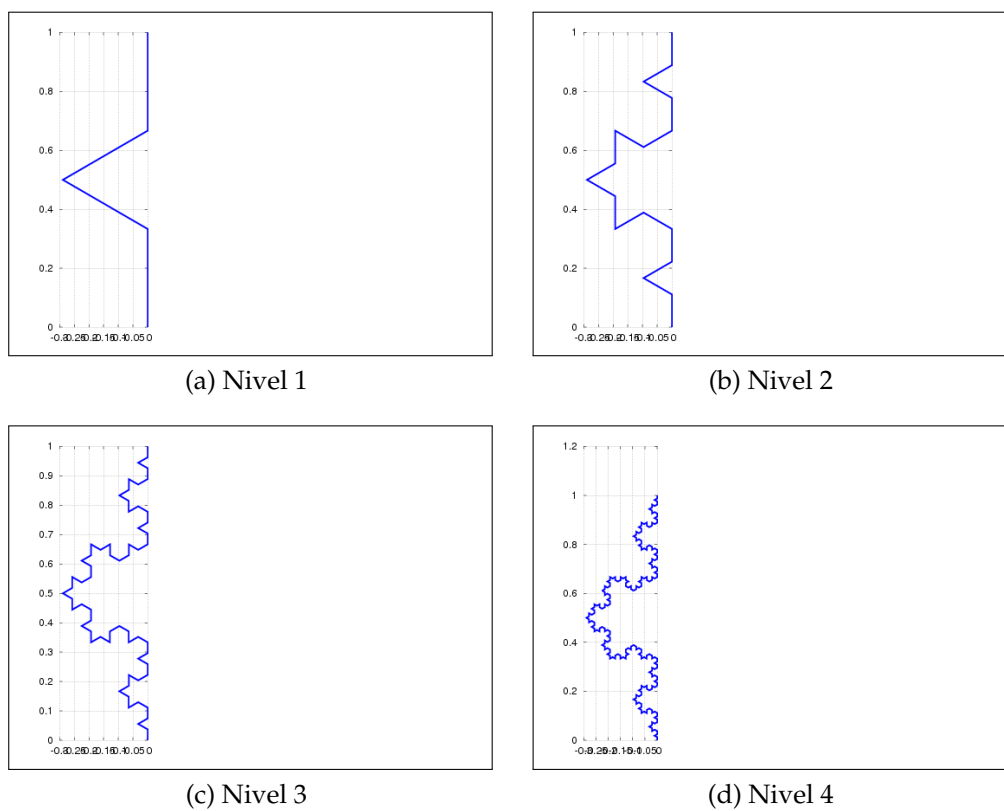


Figura D.3: Curvas de Koch para distintos niveles

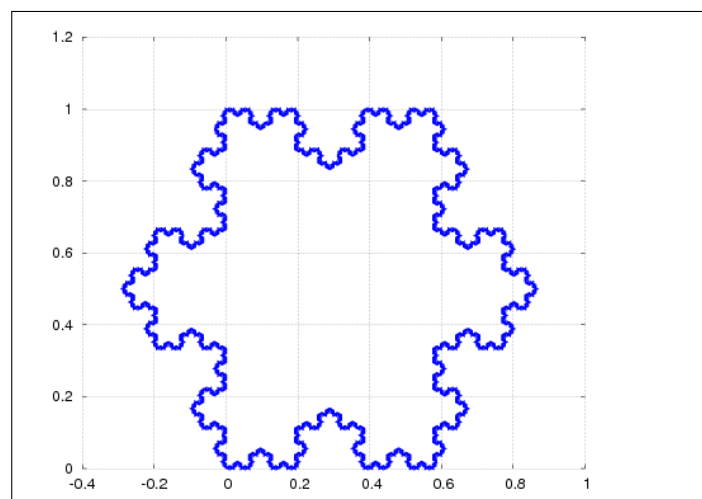


Figura D.4: Un copo de nieve realizado como unión de tres curvas de Koch

Bibliografía

- [1] Equipo de desarrollo de Maxima. *Manual de Maxima (versión en español)*.
- [2] The gnu operating system. <http://www.gnu.org>.
- [3] J. González Barahona, J. Seoane Pascual, and G Robles. *Introducción al Software Libre*. Universitat Oberta de Catalunya, 2004.
- [4] Mario Rodríguez Riotorto. Primeros pasos en maxima. <http://www.telefonica.net/web2/biomates/maxima/max.pdf>, 2008.
- [5] Richard M. Stallman. Software libre para una sociedad libre. http://osl.uca.es/jornadas/josluca06/web/cd/Contenidos/libros_y_articulos.html#software-libre-para-una-sociedad-libre.