Tema 2

Introducción al Maxima

En la primera mitad de esta asignatura vamos a aprender a programar en MAXIMA, esto significa que no vamos a conformarnos con aprender a usar muchas de las numerosas funciones que forman parte de este lenguaje de programación para cálculo simbólico y numérico, sino que aprenderemos a programar nuestras propias funciones. El objetivo central de esta parte de la asignatura es que cada uno de nosotros aprenda a generar su propia biblioteca de funciones, orientadas a resolver los problemas más o menos avanzados que se va encontrando a medida que progresa en sus estudios.

Para conseguir este fin debemos cubrir tres etapas: familiarizarnos con la sintaxis, aprender a usar el manual, aprender a cargar archivos de texto con instrucciones en MAXIMA en sesiones de trabajo con WXMAXIMA. Una vez hecho esto ya estaremos en condiciones de programar en MAXIMA.

En este capítulo nos familiarizaremos con las peculiaridades de la sintaxis de este lenguaje de programación, como por ejemplo el uso de "`:`" para hacer asignaciones, o el uso de un carácter especial para indicar el final de una instrucción: "`;`" o "`$`" dependiendo de que queramos, o no, visualizar el output producido por dicha instrucción.

Como es habitual en otros sistemas de álgebra computacional, en MAXIMA se suele trabajar por medio de una interfaz gráfica, denominada WXMAXIMA, la cual viene provista de un *manual* con información sobre las diversas (muy numerosas) funciones que ya vienen programadas en MAXIMA. El manual al que nos referimos está disponible en los menús:

```
Ayuda/Ayuda de Maxima

Ayuda/Ejemplo
```

de la ventana de la interfaz gráfica de WXMAXIMA. Este manual es extremadamente útil, en él se explica la finalidad y sintaxis de todas las funciones del lenguaje MAXIMA, además tiene buscadores que nos permiten localizar rápidamente cualquier función. El lenguaje MAXIMA está provisto de un número enorme de funciones; ningún programador, por experto que sea, las conoce todas, de forma que el uso del manual para consultar sintaxis u opciones de funciones ya conocidas, o también para localizar otras funciones que no conocíamos, es algo rutinario para todos los programadores y usuarios de este lenguaje. Por tanto es conveniente que desde el primer momento nos acostumbremos a consultar el manual.

En particular, a lo largo de estas notas sobre MAXIMA se hace referencia a diversos comandos o funciones del MAXIMA, indicando su finalidad y (en algunas ocasiones) la

sintaxis correspondiente. En todos estos casos se sobreentiende que la información aportada en estos apuntes debe complementarse con la información disponible en el manual, donde la sintaxis de estos comandos o funciones del MAXIMA viene descrita con todo detalle, junto con las diversas opciones disponibles y algunos ejemplos de uso.

2.1. ¿Por qué MAXIMA?

Si buscamos un sistema de álgebra computacional disponible en todos los sistemas operativos habituales, con capacidad para cálculo tanto simbólico como numérico, capacidad para diversos tipos de representaciones gráficas, y que además sea gratuito y de código abierto, la mejor opción es MAXIMA.

> MAXIMA es un *sistema de álgebra computacional* para la manipulación de expresiones matemáticas tanto simbólicas como numéricas, con capacidad para representaciones gráficas de funciones y datos en dos y tres dimensiones. MAXIMA produce resultados con alta precisión usando expresiones exactas en el cálculo simbólico y representaciones con aritmética de coma flotante de precisión arbitraria en el cálculo numérico.
>
> MAXIMA incluye herramientas para todas las operaciones de cálculo habituales: operaciones con vectores y matrices, números complejos, diferenciación, integración, desarrollos de Taylor y Fourier, transformadas integrales (Laplace, Fourier), resolución de ecuaciones algebraicas por métodos analíticos y trascendentes por métodos numéricos, así como diversas herramientas para ecuaciones diferenciales ordinarias, sistemas de ecuaciones lineales, etc.
>
> MAXIMA es un programa gratuito de código abierto. Su código fuente puede ser compilado sobre varios sistemas incluyendo Windows, Linux y MacOS X. El código fuente para todos los sistemas y los binarios precompilados para Windows y Linux están disponibles en el Administrador de archivos de SourceForge (`http://maxima.sourceforge.net`).

Actualmente existen otros sistemas de álgebra computacional, entre los que incluyen todas las características anteriores. Los más extendidos son: MATHEMATICA, MATLAB y MAPLE, todos ellos considerablemente más potentes que MAXIMA, pero con el inconveniente de no ser gratuitos.

MAXIMA es un descendiente de MACSYMA, el legendario sistema de álgebra computacional desarrollado a finales de 1960 en el Instituto Tecnológico de Massachusetts (MIT). Es un sistema basado en el esfuerzo voluntario de muchos programadores y de una comunidad de usuarios activa, gracias a la naturaleza del *open source*. El programa está en constante actualización, corrigiendo errores (*bugs*) y mejorando el código y la documentación. La mayor parte de la discusión se hace por medio de listas de correo. A continuación incluimos algunos enlaces con información útil sobre sistemas de álgebra computacional en general y sobre MAXIMA en particular:

- Página central de MAXIMA y documentación

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

http://andrejv.github.io/wxmaxima/

http://andrejv.github.io/wxmaxima/help.html

https://en.wikipedia.org/wiki/Maxima_(software)

- Sistemas de álgebra computacional

https://en.wikipedia.org/wiki/Computer_algebra_system

https://en.wikipedia.org/wiki/List_of_computer_algebra_systems

2.1.1. Interfaz gráfica (*front end*)

Existen tres formas posibles en las que el programa MAXIMA se presenta al usuario: la ventana de *línea de comandos* básica y dos interfaces gráficas llamadas WXMAXIMA y XMAXIMA; aparte de estos existen otros programas, como el TEXmacs, que también permiten enviar instrucciones al MAXIMA.

⋆ ¿Qué interfaz se recomienda?

En general la interfaz más recomendada es WXMAXIMA, especialmente para usuarios nuevos, que estén comenzando con este lenguaje, ya que tiene numerosos menús que facilitan aplicar las funciones del MAXIMA en las tareas más comunes, aparte de un menú de ayuda muy completo. Además esta interfaz permite combinar texto, cálculos y gráficas en un documento único, que se puede grabar de forma permanente, y utilizar como gestor del trabajo realizado. Para estos nuevos usuarios, como es obvio, el uso de los menús y botones les permiten un aprendizaje más rápido de la sintaxis del MAXIMA. No obstante, no debe olvidarse que este modo de trabajo no es exhaustivo, ya que los menús y botones del interfaz gráfico no contienen todos los comandos del MAXIMA que podrían necesitarse en una aplicación práctica concreta. En particular, si nos limitamos a usar MAXIMA por medio los menús y botones del WXMAXIMA exclusivamente, no tendremos la posibilidad de programar nuestras propias funciones, que es el elemento fundamental de esta parte del curso

Muchos usuarios expertos tienden a trabajar al unísono en las dos interfaces habituales: WXMAXIMA y XMAXIMA, cambiando a XMAXIMA cuando ya conocen los nombres de las funciones que necesitan y prefieren una interfaz estable, sin distracciones. Por último, es importante conocer que la interfaz XMAXIMA se mantiene bastante invariable a lo largo de las nuevas versiones del MAXIMA, mientras que el WXMAXIMA se está desarrollando de forma activa y suele cambiar frecuentemente su apariencia y contenido en las sucesivas versiones del programa.

En cualquier caso, a modo de resumen

para este curso recomendamos el uso de WXMAXIMA.

En lo sucesivo, salvo que se diga lo contrario se da por entendido que estamos trabajando con WXMAXIMA, que es la interfaz de trabajo más habitual.

2.1.2. Descarga e Instalación

En entornos Linux la instalación del programa MAXIMA es muy sencilla, suponiendo que queremos usar el *front end* WXMAXIMA lo único que tenemos que hacer es abrir una terminal como root y escribir:

`dnf install wxmaxima` (si estamos usando Fedora)

`apt install wxmaxima` (si estamos usando Debian o Ubuntu)

El programa de instalación se encargará automáticamente de revisar qué bibliotecas adicionales necesitamos para instalar el WXMAXIMA, nos informará de toda la lista de paquetes que vamos a instalar, las descargará de Internet y las instalará. Finalizado este proceso (unos pocos minutos si nuestra conexión de red es razonablemente rápida) ya estaremos en condiciones de empezar a trabajar con MAXIMA.

2.2. Sintaxis de MAXIMA

2.2.1. Primera sesión con Maxima

Para esta primera sesión utilizaremos MAXIMA en línea de comandos. Como hemos comentado antes la forma habitual de trabajar es por medio del interfaz WXMAXIMA, pero también es interesante observar, al menos una vez, cómo funciona este programa al nivel más básico. sin interfaz gráfica. En cualquier caso, las instrucciones que incluimos a continuación nos servirán para comenzar a familiarizarnos con la sintaxis de este lenguaje.

Abrimos entonces una terminal de *línea de comandos* (como usuario normal, no como root) y en ella tecleamos `maxima` + enter. Una vez accedemos a MAXIMA lo primero que vamos a ver será el siguiente mensaje:

```
Maxima 5.37.2 http://maxima.sourceforge.net
using Lisp SBCL 1.2.15-1.fc23
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1)
```

Tras la información sobre la licencia GNU-GPL, nos informa sobre la versión con la que estamos trabajando y la página web del proyecto. A continuación, ya aparece el indicador `(%i1)`, donde `i` es abreviatura de *input*, y el `1` indica su número, esperando nuestra primera acción. Si queremos calcular una simple suma tecleamos la operación deseada seguida de un punto y coma (;) y una pulsación de la tecla de retorno enter.

```
(%i1) 2.5+4;
```

a lo que MAXIMA da la primera respuesta en la forma

```
(%o1) 6.5
(%i2)
```

donde la `o` en `(%o1)` es la abreviatura de *output*. A continuación del *output* observamos `(%i2)`, indicando que MAXIMA espera nuestra segunda instrucción.

Recordamos que en la notación anglosajona, la que sigue el MAXIMA, la separación entre la parte entera y la parte decimal de un número se indica mediante un punto (.), mientras que la coma (,) se reserva para separar elementos de una lista (componentes de un vector, argumentos de una función, etc.), de tal forma que la siguiente acción da un resultado de “error”

```
(%i2) 2,5+4;
Improper argument to ev:9
- an error. To debug this try: debugmode(true);
```

El programa numera el input recibido, independientemente de que exista o no, un output. Como se ve en este ejemplo, los mensajes de advertencia y error no son considerados como verdaderos resultados de cálculo (output). El punto y coma actúa también como un separador cuando escribimos varias instrucciones en un misma línea. Nuestra siguiente operación consistirá en asignar el valor 125 a la variable x y 46 a la y, solicitando luego su producto,

```
(%i3) x:125; y:46; x*y;
(%o3) 125
(%o4) 46
(%o5) 5750
```

Conviene prestar atención al hecho de que la asignación de un valor a una variable se hace con los dos puntos (:), no con el signo de igualdad (=), que en MAXIMA se reserva para definir ecuaciones. Como vemos, el MAXIMA va notificando al usuario cada uno de los cálculos que realiza, en este caso, la asignación del valor 125 a x, con el correspondiente *output* `(%o3)`, la asignación del valor 46 a y, con el correspondiente `(%o4)`, y el producto xy con valor 5750, y correspondiente `(%o5)`. En general no necesitamos que nos confirme todos los pasos intermedios de cálculo realizados; para eso puede usarse el separador "$" en lugar de ";" al final de una expresión, indicando a MAXIMA que no debe mostrar el output correspondiente. Como ejemplo, el producto anterior tiene la expresión más directa

```
(%i3) x:125$ y:46$ x*y;
(%o3) 5750
```

Obsérvese que el número de inputs necesarios para realizar la operación es el mismo que antes, aunque no todos sean explícitos. Analicemos ahora brevemente el uso del comando "$=$" en este ejemplo. Si hubiéramos cometido el error de utilizar "$=$", en lugar de la asignación ":" en y, el resultado es el siguiente

```
(%i3) x:125; y=46; x*y;
(%o3) 125
(%o4) y=46
(%o5) 125 y
```

Esto quiere decir que MAXIMA admite la ecuación $y = 46$ genérica, pero no tiene orden de asignar ningún valor a y, por lo que el producto xy para MAXIMA es el producto de 125 e y.

Reiniciemos ahora el MAXIMA por medio de la instrucción `kill(all);`, con esto limpiamos la memoria (eliminamos todas las asignaciones a variables realizadas) y el programa se reinicia solicitando el primer input. Retomemos entonces al caso anterior con la asignación de 125 a x y 46 a y. Las asignaciones a variables se mantienen activas mientras dure la sesión con MAXIMA, por lo que podemos restar las variables x e y, con el resultado

```
(%i1) x:125$ y:46$
(%i3) x-y;
(%o3) 79
```

Es importante puntualizar aquí que MAXIMA no admite la expresión xy, como el producto de x por y, sino que considera la expresión xy como una sola variable. Por eso, un input del tipo $2y$ daría un mensaje de error, ya que debemos escribir `2*y`.

Uno de los errores más comunes que se comenten con los programas de cálculo simbólico es evaluar una expresión o proceso dependiente de una variable (p. ej. x) cuando ya existe una asignación previa para ese nombre (x) (este error es especialmente habitual en sesiones de trabajo largas, para evitarlo se puede usar el menú: `Maxima/Reiniciar Maxima`). Por ejemplo, supongamos que queremos resolver la ecuación algebraica $x^2 = 2$, con soluciones $x = \pm\sqrt{2}$. Para ello debemos escribir la ecuación `x^2=2` empleando como variable un símbolo para el que no se haya asignado antes un valor. Aunque no se haya estudiado todavía, la función de MAXIMA que resuelve ecuaciones algebraicas es `solve`, y su sintaxis es fácil de entender: `solve(`ecuación`,` variable`)`. Ejecutando esta acción, encontramos que

```
(%i4) solve(x^2=2,x);
A number was found where a variable was expected - 'solve'
- an error. To debug this try: debugmode(true);
```

nos da un error, la variable x no es tal variable, es un número (igual, en este caso, a 125). Para resolver este problema sin necesidad de reiniciar el programa basta eliminar de la memoria la asignación a x. El comando a utilizar para eliminar una asignación es `kill`, en la forma

```
(%i5) kill(x);
(%o5) done
```

y de nuevo, al ejecutar `solve`, obtenemos la respuesta correcta

```
(%i6) solve(x^2=2,x);
(%o6) [x = -sqrt(2), x = sqrt(2)]
```

En este caso, la resolución de la ecuación desde el punto de vista del MAXIMA simplemente consiste en dar ecuaciones más sencillas, equivalentes a la original, donde la variable x está despejada. Es importante destacar que esta solución no es una asignación, no tiene el signo de puntuación ":", y por tanto, la variable x sigue siendo una variable,

```
(%i7) x;
(%o7) x
```

La ecuación anterior nos permite también introducir el símbolo empleado en MAXIMA para denotar la operación "elevar a una potencia": `^` (por ejemplo, `2^10` es 1024, `2^(-1/2)` es $1/\sqrt{2}$). Es muy importante darse cuenta de que (al menos en la mayoría de los sistemas operativos) para que aparezca el operador `^` debemos pulsar la tecla `^` seguida de un espacio, de lo contrario el siguiente número aparecerá en WXMAXIMA como un superíndice que no se interpreta como una potencia. Hay que tener especial cuidado al usar el comando `^` en el *frontend* WXMAXIMA, a fin de evitar confusión entre *exponentes* y *superíndices*. En este sentido es bastante desafortunado el modo en que el WXMAXIMA interpreta el resultado de teclear `^`. Generalmente, cuando uno introduce `^` tecleando este símbolo "una sola vez" WXMAXIMA interpreta lo que va a continuación como un superíndice; posteriormente, *dependiendo del valor de dicho superindice*, WXMAXIMA lo interpreta bien como un exponente o bien sencillamente como un superíndice. Por ejemplo, en la versión (15.08.2) de WXMAXIMA y con un sistema operativo Linux Fedora 23 obtenemos estos resultados tecleando "una sola vez" el símbolo `^`:

```
(%i1) 2^2;
(%o1) 4
(%i1) 2^3;
(%o1) 8
```

```
(%i1) 2^4;
incorrect syntax: ^ is not an infix operator
2^4;
```

(dependiendo de la versión del WXMAXIMA empleada y del sistema operativo estos resultados pueden variar). Todas las expresiones anteriores darían como resultado un "error" en MAXIMA en línea de comandos, y el hecho de que WXMAXIMA las evalúe de forma distinta dependiendo del valor del superíndice genera mucha confusión cuando se está aprendiendo. Evidentemente este comportamiento del WXMAXIMA es muy desafortunado. La manera de evitar esta confusión entre superíndices y exponentes es la siguiente:

- Cuando queramos introducir un exponente en WXMAXIMA hay que teclear el símbolo `^` seguido de un espacio (igual que en MAXIMA),

de esta forma el símbolo `^` aparece de manera explícita en nuestra expresión, indicando de manera inequívoca que lo que va a continuación es un exponente, no un superíndice. Operando de esta manera se resuelve el problema que aparecía antes con el exponente 4

```
(%i5) 2^4;
(%o5) 16;
```

o en general con cualquier exponente

```
(%i6) 2^a;
(%o6)
```

Hasta ahora no hemos hecho uso de la numeración de los inputs y outputs, que en principio podría parecer que sólo sirven para el control de los cálculos. Sin embargo, dichas entradas (`(%i`n`)`) quedan grabadas en la sesión, y podemos en cualquier momento, recalcular su expresión, sin tener que copiarlo explícitamente. Así por ejemplo, al repetir el primer input obtenemos

```
(%i8) %i1;
(%o8) x:125
```

Con la expresión obtenida en el output vemos que sólo se ha transcrito el input 1, para comprobar si esta acción se ha ejecutado de nuevo basta solicitar el valor de x,

```
(%i9) x;
(%o9) x
```

Vemos entonces que, realmente, lo que hemos realizado es una copia del input sin ejecutarlo. Si quisiéramos que se ejecutara, tendríamos que colocar dos comillas simples (no una comilla doble) antes del input a realizar. De esta forma,

```
(%i10) ''%i1;
(%o10) 125
(%i11) x;
(%o11) 125
```

Antes de terminar esta primera sesión vamos a hacer uso del programa de ayuda básico para buscar información sobre las funciones definidas en MAXIMA. El comando se denomina `describe`, y como ejemplo utilizamos la función de raíz cuadrada `sqrt` que hemos visto anteriormente,

```
(%i12) describe(sqrt);
- Función: sqrt (<x>)
Raíz cuadrada de <x>. Se representa internamente por '<x>^(1/2)'.
Véase también 'rootscontract'.
```

```
(%o12) true
```

Para no perder los resultados y cálculos desarrollados en la sesión, podemos guardar en un archivo los valores que nos interesen, de variables, instrucciones y resultados. Por ejemplo, para grabar en el archivo sesion1, los valores de x, y, y la instrucción del %i6 con el nombre de "resolucion", la instrucción sería

```
(%i13) save("sesion1",x,y,resolucion=%i6)$
```

Comprúebese que el primer nombre es el archivo (junto con su ruta de acceso, si es necesario), que las variables x e y se escriben tal cual, pero que las referencias a inputs y outputs deben ser referenciadas con nombres de cadena, para poder ser reconocibles por el MAXIMA al abrir este archivo, ya que tanto %in como %on van variando con el desarrollo del cálculo. Una vez terminada la sesión, la forma correcta de abandonar la sesión es mediante el comando

```
(%i14) quit();
```

Abramos ahora una nueva sesión con el WXMAXIMA. Para cargar el archivo anterior desde el wxMaxima, ejecutamos la sentencia

```
(%i1) load(sesion1);
```

A diferencia del MAXIMA en línea de comandos, en el WXMAXIMA para ejecutar una instrucción debemos presionar shift + enter (presionando solamente enter lo que obtenemos es un "salto de línea"). Ejecutando el anterior (%i1) comprobamos que obtenemos los resultados grabados, para x e y

```
(%i2) x; y;
(%o2) 125;
(%o3) 46
```

y también podemos comprobar que hemos recuperado la instrucción que resuelve la ecuación en x,

```
(%i4) resolucion;
(%o4) solve(x^2=2,x)
```

y si queremos evaluarla, bastaría con la secuencia de instrucciones que hemos visto más arriba:

```
(%i4) kill(x)$ ''resolucion;
(%o5) [x = -sqrt(2), x = sqrt(2)]
```

2.3. Trabajando con wxMaxima

Según muestra la página web del WXMAXIMA:

http://wxmaxima.sourceforge.net/wiki/index.php/Main_Page

las ventajas de esta interfaz respecto al XMAXIMA, son, entre otras:

- creación de documentos, con texto y cálculos matemáticos, que permiten grabar y cargar las sesiones de trabajo,
- ventanas de diálogo cuando se utilizan funciones del MAXIMA con más de un argumento, que permiten introducir los datos sin recordar la sintaxis exacta,
- sistema de menús con la mayoría de las funciones del MAXIMA ordenadas por secciones y aplicaciones,
- **menú de ayuda con mucha información sobre comandos del** MAXIMA, incluyendo ejemplos de uso.

junto con mejoras en la visualización de las expresiones matemáticas de salida, en la transcripción de comandos, animaciones, etc. Desde esta misma página, se pueden descargar tutoriales que contienen archivos en el formato propio de la aplicación (archivos de tipo `wxm`), que pueden ser cargados en una sesión del WXMAXIMA. El archivo `10minute.zip` contiene un tutorial de uso básico de WXMAXIMA para realizar en 10 minutos, y el archivo `usingmaxima.zip` ofrece información general básica sobre la aplicación incluyendo, entre otras cosas, la "estructura de celdas" empleada en WXMAXIMA.

2.3.1. Estructura de celdas

De forma distinta a la típica línea de comandos del MAXIMA, que trabaja con un sistema de input-output único, WXMAXIMA utiliza un concepto dinámico de evaluación, en el que se puede unir texto, cálculos y gráficas. Cada documento del WXMAXIMA consiste en un número de las llamadas *celdas*. La celda es un conjunto básico de instrucciones, y está visualmente identificada por un borde superior e inferior. Hay celdas de distintos tipos, p. ej. hay celdas de título, de sección, de texto, etc., la más importante para nosotros será la celda de entrada o input.

La celda de entrada o input es el tipo de celda básico, empleado para escribir instrucciones que queremos evaluar por medio del programa MAXIMA. Para obtener la primera celda de input en una sesión de WXMAXIMA podemos pulsar enter directamente, o bien pedir a WXMAXIMA que aporte esta celda por medio del menú: `Celda/Nueva celda de entrada`. Una vez escritas las instrucciones a evaluar en la celda de entrada, que aparece con bordes rojos antes de su evaluación, ésta se evalúa al pulsar en el teclado la combinación de teclas shift + enter. En ese momento el código completo escrito en la celda es enviado al MAXIMA, que lo verifica, y comprueba que cada línea de código termine en ";" o en "$" (si no es así el WXMAXIMA añade el símbolo ";" al final de cada línea de forma automática) y posteriormente ejecuta de forma secuencial las instrucciones que hayamos escrito en la celda.

Hacemos notar que en WXMAXIMA la celda de código no tiene por qué limitarse a una línea de texto. Una vez corregida la sintaxis, MAXIMA procede a la evaluación de la celda, y en la salida van apareciendo los sucesivos (%in) y (%on) que son indicación del progreso del cálculo. El programa permite tener distintos colores para los datos de entrada y salida, y los *input* y *output* asociados, para mejor visualización del contenido de la celda. Una vez evaluada la celda, para acceder a la siguiente, se debe pulsar la tecla enter; entonces aparecerá ésta en el color asignado a las celdas sin evaluar (rojo, por defecto). Las celdas ya evaluadas se pueden modificar y recalcular, se pueden copiar y borrar, tras resaltarlas haciendo *click* a la izquierda de la línea de código. La copia de la celda sólo contiene las instrucciones de entrada de la original, en espera de su evaluación.

La sesión de trabajo con WXMAXIMA puede grabarse en un archivo (p. ej. desde el menú: `Archivo/Guardar` o `Archivo/Guardar como`); en este sentido hay dos posibilidades. En el formato básico `wxm` (el único disponible en versiones antiguas de WXMAXIMA) sólo serán grabados los inputs de cada celda de la sesión de trabajo, pero los outputs no son grabados, de forma que al cargar el archivo que contiene la sesión debemos evaluar todas las celdas para volver a obtener los resultados esperados. Por el contrario, en el formato más avanzado `wxmx` (disponible en versiones nuevas) se grabará la sesión de trabajo completa (*inputs* y *outputs*).

2.3.2. Configuración del wxMaxima

Desde el menú: `Maxima/Paneles` se pueden visualizar los paneles de comandos básicos: Matemáticas generales y Estadística, así como el panel de Historia, que contiene los últimos comandos utilizados, o el panel de Insertar celda, entre otros. El tipo, color y tamaño de letra, junto con otras opciones básicas, se configura en el menú: `Editar/Preferencias`. Una vez realizados los cambios oportunos, pueden guardarse las especificaciones en el archivo `style.ini`, para poder cargarlas posteriormente en el inicio de sesión.

La mayoría de los comandos comunes en MAXIMA aparecen al seleccionar los menús directores: `Ecuaciones`, `Álgebra`, `Análisis`, `Simplificar`, `Gráficos` y `Numérico`. En el momento de seleccionar uno de ellos, WXMAXIMA lo actualiza con el dato que figura en la celda activa. Si no existiera este dato (no existe una celda activa), WXMAXIMA calcula el resultado operando sobre el símbolo "`%`", que es una *variable del sistema* cuyo valor está dado por la última expresión empleada. Por tanto, si queremos utilizar esta forma directa de escritura, debemos escribir primero los datos en la celda activa, antes de seleccionar el comando de MAXIMA. Cuando el comando necesite de más de un dato de entrada, como puede ser el comando `solve` que necesita una ecuación y una variable, al ser seleccionado, MAXIMA abre una ventana de diálogo para introducir estos datos adicionales, junto con el dato obtenido de la celda activa.

Un aspecto muy interesante del uso de estos menús es que no solamente generan el *output* de la operación matemática solicitada, sino también el correspondiente *input* en MAXIMA, lo cual es muy interesante para irse familiarizando con este lenguaje de programación.

2.3.3. Control del cálculo y memoria

Como otras características importantes, desde el menú de comandos del WXMAXIMA se permite interrumpir en cualquier momento el cálculo de una celda activa, evitando así demoras si el usuario decide que la ejecución tarda demasiado. El WXMAXIMA también permite borrar el contenido de la memoria, aplicando el comando `kill(all)` por medio del menú correspondiente, y también existe un menú para reiniciar MAXIMA. Asimismo el menú `Maxima/Conmutar pantalla de tiempo` desactiva el control del tiempo que ha sido empleado en cada evaluación de celda activa.

Un último apunte dentro de esta sección, para indicar con un ejemplo cómo aplicar los comandos de MAXIMA sólo a una parte de la celda activa. El menú: `Simplificar` contiene comandos que permiten encontrar expresiones más sencillas ('simplificadas') que las dadas. El termino 'simplificado' no tiene una acepción clara y evidente, puesto que MAXIMA no tiene forma de conocer qué es una expresión *más sencilla* para el usuario. No obstante, desde el punto de vista matemático, existen comandos bastante completos para la simplificación racional y trigonométrica, para la factorización y expansión de expresiones. Del que nos vamos a ocupar brevemente aquí es del comando de simplificación racional `ratsimp`, con el caso

```
(%i1) (x^2-1)/(x-1)+log(x);
(%o1) (x^2-1)/(x-1)+log(x)
```

P. ej. si quisiéramos simplificar la primera parte de la fórmula anterior, bastaría resaltar la expresión $(x^2-1)/(x+1)$, y aplicar desde el menú la simplificación racional. Entonces, se abre una celda nueva con este resultado parcial,

```
(%i2) ratsimp(x^2-1)/(x-1));
(%o2) x+1
```

resultado de la simplificación realizada.

2.3.4. Manual de wxMaxima

Desde el menú: `Ayuda` se accede a toda la información sobre el programa MAXIMA, incluidos tutoriales, ejemplos y sugerencias para el usuario. Asimismo desde la celda activa, una vez resaltado un comando, la tecla F1 abre el Manual de MAXIMA para mostrar la entrada de esa función (principalmente su significado y sintaxis completa).

Como ya se ha mencionado anteriormente, se sobreentiende que a medida que se estudian estos apuntes se debe consultar en el manual la información completa sobre cada uno de los comandos o funciones a los que se hace referencia, y por supuesto también se debe practicar un poco el uso de estos comandos, a fin de asegurarse que hemos comprendido correctamente su funcionamiento.

2.3.5. Input y output

Como ejemplo de entrada de datos mostramos cómo calcular en una celda el área total y volumen de un cilindro de radio $R = 2$ y altura $H = 3$ en unidades arbitrarias. Matemáticamente el área total del cilindro es la suma del área de cada base circular (πR^2) más el área de la pared vertical ($2\pi RH$), resultando $A = 2\pi R^2 + 2\pi RH = 2\pi R(R+H)$, mientras que el volumen es el producto base $\times$ altura, igual a $V = \pi R^2 H$. Antes de realizar este cálculo recordamos que:

- el número $\pi = 3{,}141592\ldots$ es una constante matemática predefinida en MAXIMA por medio del símbolo `%pi`,
- debemos introducir `enter` tras cada línea de texto para acceder a la siguiente,
- MAXIMA diferencia entre mayúsculas y minúsculas y no tiene en cuenta los espacios en blanco entre las variables y los signos, dentro de las expresiones.

Las instrucciones necesarias para este cálculo pueden escribirse en la celda activa como

```
-> (R : 2, H : 3)$ A : 2*%pi*R*(R + H); V :  %pi*R^2*H;
```

Hasta ahora no hemos enviado la celda activa al MAXIMA, observamos que el corchete de la celda (a la izquierda de las líneas de texto) está en rojo, que es el color por defecto, y tenemos el cursor parpadeando en algún espacio de línea en el interior de la celda. Si ahora pulsamos `shift` + `enter`, todos los comandos se ejecutan y se muestran los *outputs* a medida que MAXIMA los obtiene, en el mismo orden que se encuentran los *inputs* (es decir, de manera *secuencial*). Además en la salida de inputs sólo aparecerá numerado el input primero de la secuencia. En el caso anterior la respuesta sería

```
(%i1) (R : 2, H : 3)$
      A : 2*%pi*R*(R + H); /*área total*/
      V :%pi*R^2*H; /*volumen*/
(%o2) 20 π
(%o3) 12 π
```

En el input anterior hemos introducido texto no ejecutable mediante la sintaxis `/*` texto `*/`, como *comentarios* para facilitar la descripción del contenido. Una observación, la primera línea del `(%i1)` contiene varias instrucciones en un solo paréntesis, separadas por comas (,), por eso el MAXIMA lo considera como un único input.

Aunque también tiene capacidad para cálculo numérico, el MAXIMA está pensado para ser un programa de cálculo simbólico. Por este motivo, siempre que sea posible mantendrá la notación para las constantes predefinidas sin evaluarlas numéricamente, de esta forma en las expresiones anteriores el número π aparece como un símbolo, en lugar de aparecer un valor numérico con precisión finita. Si deseamos obtener el valor numérico del área y volumen podemos utilizar el comando `numer` de la siguiente forma:

```
->%pi, numer;   [A, V], numer;
```

con el resultado

```
(%i4)%pi, numer;   [A, V], numer;
(%o4) 3.141592653589793
(%o5) [62.83185307179586, 37.69911184307752]
```

Otra posibilidad para forzar la evaluación numérica de un resultado es por medio del comando `float`, en la forma `float(A); float(V);`, que produce los mismos resultados numéricos para A y V mostrados en `(%o5)`.

El número de decimales empleado en las evaluaciones numéricas depende de la precisión del cálculo (por defecto es de 16 dígitos), que puede establecerse por el usuario de manera arbitraria en el menú: `Numérico/Establecer precisión`.

2.3.5.1. Expresiones y funciones

La forma en que hemos calculado el área y volumen de un cilindro más arriba tiene el inconveniente de dejar las variables `R` y `H` evaluadas a los valores numéricos de R y H correspondientes a un cilindro concreto (en este caso $R = 2$ y $H = 3$). Esto es poco conveniente si, p. ej., queremos considerar diferentes valores para estas variables en una misma sesión, o si posteriormente a este cálculo queremos resolver una ecuación para la variable R, en cuyo caso deberíamos eliminar la asignación realizada por medio de `kill(R);`.

Una forma sencilla de evitar esto, sin necesidad de emplear `kill`, es por medio del comando de sustitución `subst`. La instrucción `subst(a, b, c)` sustituye en la expresión que hayamos introducido en `c`, la variable `b` por lo que hayamos introducido en `a` (`a` podrá ser un valor concreto, el nombre de otra variable, una expresión, etc.). Así, podemos definir el área y volumen del cilindro por medio de las expresiones que hemos empleado mas arriba

```
(%i1) A : 2*%pi*R*(R + H); /*área total*/
      V :%pi*R^2*H; /*volumen*/
```

pero conservando las variables `R` y `H` como *símbolos* (no evaluados a valores numéricos), de esta forma `A` y `V` quedan evaluadas a las correspondientes *expresiones simbólicas*:

```
(%o1) 2 π R (R + H)
      π H R²
```

A continuación, si queremos calcular los valores de A y V correspondientes a los valores concretos de R y H de antes, pero sin asignar estas variables a ningún valor,

basta con emplear `subst` para que sustituya estos valores en las variables y muestre el resultado

```
(%i2) subst(3, H, subst(2, R, A));
(%i3) subst(3, H, subst(2, R, V));
```

lo cual genera los valores de `A` y `V` correspondientes a los valores de `R` y `H` indicados (2 y 3 respectivamente).

Como veremos en este curso hay multitud de ocasiones en las que resulta muy práctico emplear el comando de sustitución `subst`, pero en el caso que nos ocupa es más práctico definir una *función* para el área y otra para el volumen. Para definir funciones en MAXIMA se emplea el símbolo "`:=`", en el caso de funciones *sencillas* la sintaxis es la siguiente:

$$\boxed{\text{"nombre de la función"}\left(\begin{array}{c}\text{lista de argumentos}\\ \text{separados por comas}\end{array}\right) := \begin{array}{c}\text{expresión que}\\ \text{define la función}\end{array};}$$

Siguiendo esta sintaxis, para definir las funciones `A(R, H)` y `V(R, H)` (respectivamente: "área y volumen, dependientes de las variables radio, `R`, y altura, `H`"), escribimos (después de eliminar las posibles asignaciones previas a `A`, `V`, `R` y `H`):

```
(%i1) A(R, H) :=  2*%pi*R*(R + H); /*área total*/
      V(R, H) :=  %pi*R^2*H; /*volumen*/
```

Pulsando shift + enter obtenemos el output

```
(%o1) A(R, H) := 2 π R (R + H);
(%o2) V(R, H) := π R² H;
```

por medio del cual MAXIMA nos informa que ha incorporado a la memoria de la sesión actual las definiciones de estas dos nuevas funciones, de acuerdo a las expresiones que aparecen en `(%o1)` y `(%o2)`. Si no queremos visualizar la expresión que define la función introducida basta con emplear el carácter "$", en lugar de ";", para finalizar la instrucción de definición de la función.

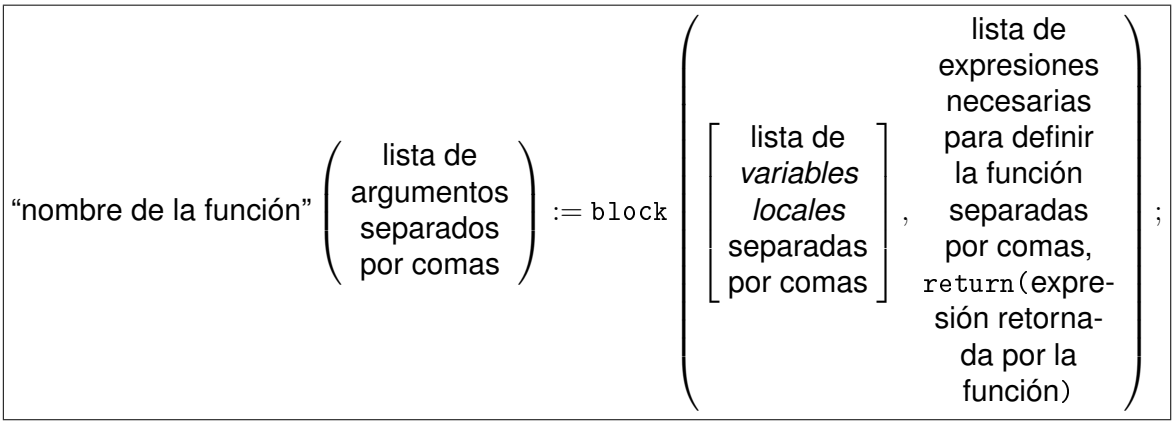
Una vez hemos definido estas dos nuevas funciones, para visualizar el área y volumen del cilindro del ejemplo anterior, sin asignar valores fijos a las variables `R` o `H`, basta con hacer

```
(%i3) A(2, 3);
(%i4) V(2, 3);
```

Pulsando shift + enter MAXIMA devuelve el valor que se obtiene al sustituir `R` y `H` por los valores suministrados como argumentos de las funciones `A(R, H)` y `V(R, H)` en `(%i3)` e `(%i4)`.

Como comentábamos más arriba, la sintaxis de definición de funciones que hemos mostrado es válida para definir *funciones sencillas*, es decir, "funciones que pueden definirse por medio de **una única** expresión". El trabajo a realizar a lo largo de este curso consiste no obstante, en la inmensa mayoría de los casos, en definir *funciones complicadas*, es decir, funciones para cuya definición es necesario evaluar no una, sino varias expresiones intermedias, antes de poder obtener el resultado retornado por la función. En ese caso para definir una función se emplea el comando `block`, y la

sintaxis es la siguiente:



Por ejemplo, la función anterior para calcular el área del cilindro puede definirse también como

```
(%o1) A(R, H) := block( [Atapa, Alado],
                        Atapa : %pi*R^2,
                        Alado : 2*%pi*R*H,
                        return(2*Atapa + Alado) )$
```

que nos sirve como ejemplo de uso de la función `block`. Evidentemente, para una función tan sencilla como esta basta con la definición que habíamos dado antes, pero como veremos a lo largo de este curso, en MAXIMA pueden programarse funciones considerablemente más complicadas, para las que la función `block` resulta más que recomendable.

La definición de funciones usando el comando `block` será un tema central en esta asignatura. En particular, la Prueba de Evaluación Continua (PEC) de MAXIMA consistirá en la programación de diversas funciones más o menos complicadas empleando la sintaxis que acabamos de mostrar. A lo largo de los siguiente temas veremos numerosos ejemplos de definiciones de funciones por medio de `block`, de modo que en este capítulo de introducción al MAXIMA no adelantaremos más información a este respecto.

2.4. Expresiones en MAXIMA

La unidad básica de información de MAXIMA es la expresión. En la sección anterior ya hemos visto algunos ejemplos sencillos de expresiones. Una expresión es una combinación de variables, números, operadores y constantes.

- Las variables empleadas en expresiones (p. ej. `x`, `area`) deben tener un nombre que no coincida con palabras reservadas u operadores predefinidos del lenguaje MAXIMA (p. ej. `if`, `then`, `else`, `for`, `thru`, `sin`, `cos`, `exp`, etc.), y no necesita declararse su *tipo* (los tipos habituales son: entero, real, complejo, de cadena (*string*), etc.).

- Los operadores son los comandos o funciones del MAXIMA, o cualquier otra función definida por el usuario.

- Las constantes predefinidas en MAXIMA son las constantes matemáticas habituales, que incluyen, entre otras, la base de los logaritmos naturales e (definida en MAXIMA como `%e`), la unidad compleja imaginaria $i = \sqrt{-1}$ (`%i`), el número π (`%pi`), el infinito real positivo $+\infty$ (`%inf`) y el infinito real negativo $-\infty$ (`%minf`).

  En general MAXIMA tiende a tratar estas constantes matemáticas como símbolos, y para obtener sus valores numéricos puede emplearse el operador `float`, o `numer`:

  ```
  (%i2) float([%pi,%e,%i]);
  (%o2) [3.141592653589793, 2.718281828459045,%i]
  ```

  La constante `%i`, al ser compleja no es evaluada numéricamente. El producto `%i*%i` daría el valor numérico -1.0, en coma flotante.

En cuanto a las palabras reservadas, una forma rápida y sencilla de evitar estos conflictos en la definición es comprobar su posible existencia en el índice del manual de MAXIMA en el momento de seleccionar un nombre de variable nuevo. Como hemos comentado antes el MAXIMA diferencia entre mayúsculas y minúsculas, de modo que, dado que los comandos del MAXIMA siempre van en minúsculas, una buena práctica es definir las funciones del usuario, y en su caso las variables, con mayúsculas.

2.4.1. Operaciones aritméticas

Los operadores aritméticos más comunes son suma ($+$), resta ($-$), multiplicación ($*$), división ($/$), y potencia (`^`). MAXIMA por defecto devuelve los resultados de forma exacta, sin aproximaciones ni cálculos numéricos. Así podemos obtener resultados sencillos de expresiones complicadas como

$$\left(\frac{2^5}{1+\frac{1}{(2/3)^3+(3/2)^2}}\right)^{-3}$$

en forma exacta

```
(%i1) (2^5/(1+1/((2/3)^3+(3/2)^2)))^(-3);
(%o1) 56181887/681472000000
```

Para evaluar este número en coma flotante basta aplicar el operador `numer`, `float` o equivalentemente `bfloat`

```
(%i2) %, numer;
(%o2) 8.2441959464218638 10^-5
```

Recordamos que la precisión de esta operación puede variarse por el usuario desde el menú: `Numérico/Establecer precisión`.

Maxima puede trabajar con precisión arbitraria. Por ejemplo, para calcular la potencia π^e con 50 cifras decimales tenemos dos opciones: modificar la precisión desde el menú anterior asignando 50 dígitos, con lo cual cada cálculo posterior se realizará con esa precisión (lo cual puede ser muy lento), o directamente desde la línea de código, asignando a la variable interna de precisión `fpprec` para grandes flotantes (`bfloat`) el valor 50, con lo cual sólo será válida esa precisión para la celda activa. De esta forma obtenemos:

```
(%i3) fpprec:50$ bfloat(%pi^%e);
(%o3) 22.459157718361045473427152204543735027589315133997b1
```

Nótese que cuando utilizamos grandes números de coma flotante con MAXIMA el exponente se escribe empleando la letra "b", en lugar de la usual "e", es decir, $x\texttt{b}n$ quiere decir $x \times 10^n$.

2.4.2. Teoría elemental de números con MAXIMA

MAXIMA tiene una serie de comandos que resultan muy útiles para verificar algunos teoremas de la teoría de números, que además nos sirven para poner en práctica parte de lo aprendido hasta ahora. Tenemos `primep(`n`)` que verifica si el número n dado es primo o no, las respuestas pueden ser `true` o `false`. En este sentido conviene recordar que por convención se considera que la unidad no es un número primo. Otras funciones interesantes son `next_prime(`n`)`, que retorna el primer número primo correlativo mayor que el número n dado, `gcd(`n_1, n_2`)`, da el máximo común divisor de los dos números dados n_1 y n_2, y finalmente `ifactors(`n`)`, que factoriza el número n dado en producto de sus factores primos elevados a las potencias correspondientes.

Utilizando algunos de estos comandos vamos a calcular de forma sencilla cuántos número primos hay entre 1 y 200. La forma más directa es ejecutar el comando `next_prime(i)` sucesivamente desde el valor inicial $i = 1$ hasta que i supere el valor 200, apuntando el número de veces N que ejecutamos esta instrucción. De esta forma el número de primos que buscamos estará dado por $N - 1$ (hay que restar una unidad porque el número primo obtenido en la última iteración ya es mayor que 200, de modo que no cuenta). Para poner esto en práctica comenzamos asignando el valor inicial $i = 1$ en la primera celda:

```
(%i1) i : 1$
```

En la segunda celda aplicamos el comando `next_prime`

```
(%i2) i : next_prime(i);
(%o2) 2
```

que opera sobre el valor de i anterior, evaluando su primer número primo mayor que i, y asignando el valor encontrado de nuevo a i, este último resultado es lo que se muestra como output de la celda. Ejecutando sucesivamente esta misma celda iremos asignando los valores de i a números primos cada vez mayores, hasta que alcancemos un número primo mayor que 200. El número final de veces que se ha ejecutado la celda será el valor N que buscamos. El resultado es $N = 47$, lo que significa que hay 46 números primos menores que 200.

Nos preguntamos ahora cuántos de estos primos son de la forma $2^n - 1$, siendo n un número entero. Para ello, basta introducir el comando `ifactors` en la celda principal de cálculo, con una nueva línea de instrucción

```
(%i2) i: next_prime(i);
(%i3) ifactors(i+1);
(%o2) 2
(%o3) [[3,1]]
```

Así, con la factorización en números primos de $i + 1$ podemos distinguir qué factorización es de la forma 2^n, y por tanto qué números primos son de la forma $i = 2^n - 1$. La salida `(%o3) [[3,1]]` indica que la primera factorización en números primos de $i + 1$ da el resultado 3^1. Ejecutando sucesivamente la celda obtenemos los siguientes números primos en la forma buscada, $1 = 2^1 - 1$, $3 = 2^2 - 1$, $7 = 2^3 - 1$, $31 = 2^5 - 1$ y $127 = 2^7 - 1$. Viendo esta secuencia de números da la impresión que los números primos se corresponden con valores primos del exponente n en la expresión $2^n - 1$,

de esta forma MAXIMA nos ha servido para intuir una regularidad en el conjunto de los números primos. Veamos a continuación si esta regularidad se cumple para un número primo n arbitrario grande

```
(%i-) (n:1000, n:next_prime(n), i:2^n-1)$   primep(i);
(%o3) false
```

lo que nos indica que la regularidad que intuíamos no se cumple siempre. Los números de la forma $M_p = 2^p - 1$ se denominan *números de Mersenne*, y como hemos visto *no todos ellos* son primos. Hasta el día de hoy se conocen 45 números primos de Mersenne, el mayor de ellos fue descubierto en 2008 y tiene más de 12 millones de cifras.

2.4.3. Simplificación, expansión y factorización

Al utilizar programas de cálculo simbólico es necesario en multitud de ocasiones simplificar expresiones que, al haberse generado de manera automática, adoptan formas demasiado complicadas o extensas que, sin embargo, son equivalentes a expresiones analíticas más sencillas y manejables. El MAXIMA dispone de numerosos comandos para simplificar expresiones (de forma racional, trigonométrica, con números complejos), expandir o desarrollar funciones (logaritmos, potencias), y factorizar números o términos en ecuaciones. Los comandos más comunes son `ratsimp` para la simplificación de expresiones racionales, `trigsimp` simplifica expresiones trigonométricas, `factor` que factoriza todo tipo de expresiones, y `expand` que desarrolla todos los términos que resultan de una expresión.

Uno de los problemas que tienen los programas de cálculo simbólico es que con frecuencia no hacen de manera automática simplificaciones que a nosotros nos resultan evidentes, sólo cuando el usuario lo solicita se llevan a cabo estas operaciones de simplificación. Veamos el siguiente ejemplo en MAXIMA:

```
(%i1) (x^2-1)/(x+1);
```

(%o1) $\frac{x^2-1}{x+1}$

Evidentemente esta expresión puede simplificarse dividiendo numerador y denominador por el factor común $x+1$, es decir, factorizando $x^2-1 = (x+1)(x-1)$ en el numerador y cancelando a continuación factores comunes. Sin embargo MAXIMA no simplifica esta expresión a no ser que el usuario lo precise, en cuyo caso obtenemos el resultado esperado

```
(%i1) factor((x^2-1)/(x+1));
(%o1) x-1
```

Simplificar expresiones matemáticas no es una tarea trivial y el éxito del MAXIMA en este sentido (o en general de cualquier paquete de álgebra computacional) está condicionado por el grado de experiencia del usuario. Por otra parte, para que MAXIMA nos dé los resultados esperados es necesario conocer las técnicas que utiliza para manipular expresiones. Por ejemplo, MAXIMA no es capaz de simplificar la expresión $\frac{x^2+1}{x+i}$ (siendo i la unidad imaginaria) por medio del comando `factor`

```
(%i1) factor((x^2+1)/(x+%i));
```

(%o1) $\frac{x^2+1}{x+i}$

ya que este comando solo trabaja con expresiones reales, en este caso es preciso emplear el comando para factorización compleja `rectform`

```
(%i2) rectform(%);
```

(%o2) $x-i$

Resolvemos ahora un ejemplo sencillo de identificación de fórmulas trigonométricas. Para ello hacemos uso de los comandos usuales de funciones trigonométricas: `sin` y `cos`, donde el argumento se escribe en radianes (por ejemplo, `cos(%pi) = -1`). Queremos determinar si la fórmula

$$2\sin 2x\left(\cos^2 x-\sin^2 x\right)=\cos 3x\sin x+\sin 3x\cos x$$

es correcta. Dado que todos los argumentos que aparecen son múltiplos enteros de x, una opción es realizar las expansiones trigonométricas de los senos y cosenos de $2x$ y $3x$, verificando que resultan expresiones idénticas en ambos miembros de la igualdad (matemáticamente basta con verificar que su resta es nula). Realizamos los cálculos en distintas celdas y a cada uno de los output le asignamos un nombre de variable relacionado con lo que se calcula. Para el $\sin 2x$, obtenemos el resultado conocido del ángulo doble

```
(%i1) sen2x: trigexpand(sin(2*x));
```
(%o1) $2\sin(x)\cos(x)$

y para el ángulo triple obtenemos

```
(%i2) sen3x: trigexpand(sin(3*x)); cos3x: trigexpand(cos(3*x));
```
(%o2) $3\cos^2(x)\sin(x)-\sin^3(x)$

(%o3) $\cos^3(x)-3\cos(x)\sin^2(x)$

Una vez hemos realizado estos cálculos verificamos la veracidad de la fórmula propuesta, utilizando para ello las variables que acabamos de definir:

```
(%i4) 2*sin2x*(cos(x)^2 - sin(x)^2) - (sin3x*cos(x) + cos3x*sin(x));
```
(%o4) $-\cos(x)\sin(3x)-\sin(x)\cos(3x)+2\left(\cos(x)^2-\sin(x)^2\right)\sin(2x)$

Como era de esperar MAXIMA no da un resultado simplificado, ya que no se lo hemos pedido. Las únicas acciones que ha realizado es asignar los ordenes de aparición de los términos entre paréntesis. Por tanto, debemos aplicar una simplificación posterior para poder confirmar que la fórmula trigonométrica es correcta. Para que MAXIMA simplifique esta fórmula necesitaríamos pedirle primero que expanda las razones trigonométricas de ángulos dobles que hemos introducido, para posteriormente simplificar el resultado por medio de `factor`, al hacer esto MAXIMA encontrará un factor 0, de modo que la diferencia "lado izquierdo $-$ lado derecho" es nula. Esto mismo puede hacerse con el comando directo de simplificación `ratsimp` o con el de simplificación trigonométrica `trigsimp`, el resultado es

```
(%i5) trigsimp(trigexpand(%));
(%o4) 0
```

que confirma que la fórmula era correcta, puesto que tenemos un desarrollo de $\sin 4x$ en cada término.

Analizamos ahora un ejemplo de fracciones continuas, uno de los casos más típicos es

$$1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\ldots}}}$$

donde el proceso se extiende al infinito. Queremos verificar si esta fracción continua tiene un valor finito y calcularlo, en particular nos preguntamos si este valor depende o no de la variable x, definida como indicamos a continuación. Tomamos la variable x como variable independiente de la fracción continua

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

Lógicamente como x no tiene valor asignado, MAXIMA lo reconoce como tal. Aplicamos ahora la primera asignación

```
(%i2) fraccion:1+1/%;
(%o2) 1+1/x
```

y ya tenemos el desarrollo de la fracción continua en su primer término, al tomar como input el output (%o1). Aplicando de nuevo esta asignación obtenemos

```
(%i3) fraccion:1+1/%;
(%o3) 1+1/(1+1/x)
```

y así sucesivamente (recordar que % es una variable reservada del MAXIMA, cuyo valor es la última expresión evaluada). Vemos entonces que podemos construir la fracción continua mediante la sucesiva aplicación de una asignación muy simple. Esto mismo puede resolverse también fácilmente con el uso de funciones. Para ello basta definir una función sobre los enteros n, en la forma $f(n) = 1 + 1/f(n-1)$ y exigir que el término correspondiente a $n = 0$ sea igual a la semilla inicial x. Así, la fracción con n términos está dada por $f(n)$. Retomemos nuestro cálculo, suponiendo que hemos aplicado la asignación 15 veces, si queremos determinar el valor de la suma (por ahora en términos de x) le pedimos a MAXIMA que simplifique la fracción, con el resultado

```
(%i17) ratsimp(%);
(%o17) (610+987x)/(610x+377)
```

Este es el valor de la fracción continua con 16 términos. Si quisiéramos calcular numéricamente su valor para un x dado basta con sustituir en esta expresión x por el valor a considerar. Por ejemplo, para $x = 0$, obtenemos

```
(%i18) x:0$
(%i19) subst(0, x, %), numer;
(%o19) 1.618037135278515
```

Se puede demostrar que el límite de esta fracción continua cuando el número de términos tiende a infinito es único e independiente de x, con el valor $1{,}61803398874989$.

2.4.4. Sustituciones

Ya hemos mostrado el comando de sustitución dependiente de tres argumentos `subst(`variable y, variable x, expresión`)`, que aplica la sustitución $x \to y$ sobre el argumento que aparezca en "expresión". Una posible aplicación de este comando es la creación de funciones compuestas, $f[g(x)]$. Como ejemplo de uso de `subst` vamos a verificar si una función de dos variables dada es *homogénea de grado* p. Para ello comprobaremos si $f(\lambda x, \lambda y) = \lambda^p f(x, y)$ para un determinado exponente p que dependerá de la función escogida. Consideremos la expresión dependiente de x e y:

```
(%i1) (x^3 + y^3)/(x^2*y^2) + 2*x^2*y/(2*y^4 - x^3*y - 3*x^2*y^2);
(%o1) 2*x^2*y/(2*y^4-3*x^2*y^2-x^3*y)+(x^3+y^3)/(x^2*y^2)
```

Para analizar si esta expresión define una función homogénea realizamos las sustituciones sucesivas $x \to \lambda x$, $y \to \lambda y$, y verificamos si la expresión final cumple la relación $f(\lambda x, \lambda y) = \lambda^p f(x, y)$ para algún valor de p. Primero realizamos la sustitución en x

```
(%i2) subst(lambda * x,x, %o1);
(%o2) 2*x^2*y*λ^2/(-x^3*y*λ^3-3*x^2*y^2*λ^2+2*y^4)+(x^3*λ^3+y^3)/(x^2*y^2*λ^2)
```

A continuación, para la sustitución en y, elegimos %o2 como expresión, en la que se sustituye y por λy

```
(%i3) subst(lambda*y,y, %o2);
```

```
(%o3)
```
$\frac{2\,x^2\,y\,\lambda^3}{2\,y^4\,\lambda^4-3\,x^2\,y^2\,\lambda^4-x^3\,y\,\lambda^4}+\frac{y^3\,\lambda^3+x^3\,\lambda^3}{x^2\,y^2\,\lambda^4}$

Nos queda verificar si la expresión final es igual a la expresión inicial multiplicada por una cierta potencia de λ. Para ello hacemos la comparación

```
(%i3) factor(%o3 /%o1);
(%o3)
```
$\frac{1}{\lambda}$

Por tanto la función $f(x,y)$ es homogénea de grado -1.

2.5. Definición de funciones con Maxima

En MAXIMA existen dos tipos de asignaciones: el comando ":" se usa para asignar *valores fijos* a variables o parámetros (p. ej. `x:%pi`), mientras que para definir funciones se emplea la sintaxis "función(variables) := expresión", tal y como hemos comentado más arriba. Por ejemplo, la instrucción empleada para definir la función $f(x)=x^2$ es:

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

Para comprobar que realmente hemos definido la función $f(x)$ de la forma indicada podemos verificar que $f(a)$ devuelve a^2, sea cual sea el valor de a y sea también cual sea el tipo de variable (número real, entero, complejo, ...).

```
(%i1) f(a);
(%o1)
```
a^2

Para definir funciones *sencillas* dependientes de varias variables lo único que debemos hacer es introducir en el lado izquierdo de la asignación la lista de variables independientes separadas por comas, de acuerdo a la sintaxis que ya hemos comentado anteriormente

"nombre de la función" (lista de argumentos separados por comas) := expresión que define la función ;

donde con *sencillas* nos referimos a funciones que pueden definirse por medio de una única expresión (o instrucción) no demasiado larga. Por ejemplo, con

```
(%i1) f(x, y, z) := sqrt(x^2 + y^2 + z^2);
```

definimos una función que nos da la distancia euclídea entre el origen y el punto con coordenadas (x,y,z).

Por el contrario, cuando es necesario evaluar una secuencia de instrucciones (o expresiones) previas antes de poder evaluar la instrucción (o expresión) que define la función, se emplea el comando `block` y la sintaxis es:

"nombre de la función" (lista de argumentos separados por comas) := `block` ([lista de *variables locales* separadas por comas] , lista de expresiones necesarias para definir la función separadas por comas, `return(`expresión retornada por la función`)`) ;

A lo largo del curso veremos muchos ejemplos de definición de funciones por medio del comando `block` ya que, como también comentábamos antes, el objetivo principal de esta parte de la asignatura es aprender a programar funciones *no triviales* en MAXIMA.

Antes de continuar conviene aclarar que las *funciones* que pueden programarse en MAXIMA (o en cualquier otro entorno de álgebra computacional) no se restringen a *funciones matemáticas*, consistentes en realizar un cierto cálculo (de forma simbólica o numérica) como p. ej. `f(x) := x^2;`, sino que también pueden programarse en forma de funciones cualquier secuencia de instrucciones que opere sobre una serie de argumentos, esto también incluye operaciones como "abrir un archivo y leer su contenido", "escribir el resultado de una cierta operación en un archivo", "consultar un dato en una página web", "generar una gráfica derivada de un cálculo", etc.

Por ejemplo, la función `graficaderivadaEPS(f, x, a, b, filename)` calcula la primera ($'$) y segunda ($''$) derivada del primer argumento, `f`, respecto del segundo, `x`, a continuación muestra la gráfica de f, f' y f'' para valores de la variable `x` entre `a` y `b`, guarda dicha gráfica en formato `eps` en un archivo con nombre indicado en el último argumento `filename` (de tipo "cadena" o *string*), y finalmente devuelve como output una lista con los resultados obtenidos para f' y f'':

```
(%i1) graficaderivadaEPS(f, x, a, b, filename) := block( [aux],
          aux : [f, diff(f, x, 1), diff(f, x, 2)],
          plot2d( aux, [x, a, b],
                [gnuplot_term, ps], [gnuplot_out_file, filename] ),
          wxplot2d( aux, [x, a, b] ),
          return( [aux[2], aux[3]] )
          )$
```

para ver lo que hace esta función evaluamos, p. ej.

```
(%i2) graficaderivadaEPS(x^3, x, -1, 1, "mi-grafica.eps");
```

al presionar shift + enter la función ejecuta de manera secuencial la lista de instrucciones anterior: en primer lugar calculamos la primera y segunda derivada de x^3 por medio de `diff(f, x, 1)` y `diff(f, x, 2)` respectivamente, y asignamos a la *variable local* `aux` una lista cuyos elementos son f, f' y f''. En la siguiente instrucción la función `plot2d` genera la gráfica de esta lista de funciones (contenida en la variable local `aux`) para valores de `x` en el intervalo $[a, b]$ indicado, en este caso: $[-1, 1]$. En esta instrucción el argumento `[gnuplot_term, ps]` de `plot2d` indica que en lugar de ver la gráfica en pantalla (resultado "por defecto") lo que queremos es generar el código *postscript* (`ps`) de dicha figura, y por medio de la instrucción `[gnuplot_out_file, filename]` indicamos que queremos guardar dicho código *postscript* en el archivo indicado en el argumento `filename`. En este caso el nombre del archivo es `mi-grafica.eps`, para indicar este nombre de archivo el último argumento va entre *comillas dobles*, `"mi-grafica.eps"`, esto es necesario para que WXMAXIMA interprete este argumento no como el nombre de una variable matemática, sino como una *cadena de caracteres*, es decir, una variable de tipo *string*, que es lo que necesitamos para generar el nombre de un archivo en el disco. En la siguiente instrucción `wxplot2d( aux, [x, a, b] )` genera la misma gráfica de antes, pero esta vez la muestra por pantalla dentro de la sesión de trabajo de WXMAXIMA. Finalmente, por medio de `return( [aux[2], aux[3]] )` la función "retorna" una lista con las expresiones simbólicas obtenidas para f' y f'', en este caso $[3x^2, 6x]$.

Con este ejemplo también hemos aprovechado para introducir los temas de repre-

sentaciones gráficas y escritura de archivos, sobre los que volveremos en capítulos posteriores.

2.5.1. Variables locales y globales

En esta función `graficaderivadaEPS(f, x, a, b, filename)` también hemos visto el uso de *variables locales*. Cuando se programan funciones un poco complicadas es necesario realizar cierto número de cálculos intermedios (u otras operaciones) necesarios para obtener el resultado final deseado (es decir, el *output* producido por la función), pero que no nos son de utilidad una vez la función ha concluido su evaluación. Cuando se programan funciones, lo más cómodo para trabajar de una manera ordenada es asignar a *variables locales* (dentro de la función) estos resultados intermedios de cálculo.

En cálculos avanzados es normal operar de manera simultánea con un número elevado de funciones ciertamente complicadas. En este caso, debido al elevado número de variables que entra en juego, la tarea de buscar nombres de variables "que no estén en uso por otras funciones o celdas de la sesión de trabajo" se volvería muy engorrosa si todas las variables empleadas fuesen visibles de manera simultánea desde todas las funciones en uso. Para resolver este problema, que es común a todos los lenguajes de programación, se han inventado **dos categorías** de variables:

- ***variables globales***
- ***variables locales***

Las variables globales son las que uno crea en una sesión de trabajo con la finalidad de que puedan usarse posteriormente en dicha sesión. Por ejemplo, cuando evaluamos en una celda de wxMAXIMA la instrucción `a : 1;` creamos la variable global `a`, y asignado a ella el valor `1`. Las variables globales se caracterizan por que sus valores son visibles desde cualquier celda, expresión o función a la que llamemos en la sesión de trabajo donde se han definido. Que sus valores sean visibles desde cualquier función implica dos cosas: por un lado desde cualquier función podemos *usar* estas variables con sus valores correspondientes pero, por otra parte, desde cualquier función podemos *modificar* los valores de estas variables.

Por el contrario, las variables locales no se crean en la sesión de trabajo, sino dentro de las funciones, y se crean con la finalidad de ser empleadas exclusivamente dentro de dicha función, como variables "auxiliares" para pasos intermedios de cálculo (u otras operaciones intermedias que sean precisas). La principal característica de las variables locales es que solo son visibles *desde dentro* de la propia función donde se han definido, pero no son visibles desde otras funciones o desde la sesión de trabajo. En el ejemplo anterior, al escribir el argumento `aux` dentro de los corchetes en el comando `block` indicamos a MAXIMA que la variable `aux` es una variable local de esta función. Esto implica que esta variable existe dentro de la función `graficaderivadaEPS` mientras se ejecuta dicha función, y una vez ejecutada la función y obtenido el *output* correspondiente la variable local *desaparece*.

De esta forma ya no hay que preocuparse de buscar nombres de variables nuevos para las variables auxiliares que con frecuencia es necesario definir dentro de las funciones, ya que aunque dichos nombres de variables estén en uso en la sesión de

trabajo (como variables globales) o en otras funciones (como variables locales) estas definiciones no entrarán en conflicto unas con otras.

En el caso de la función `graficaderivadaEPS`, si en la sesión de trabajo desde donde llamamos a esta función no existe una variable con el nombre `aux`, después de llamar a esta función la variable `aux` seguirá sin estar asignada. De la misma manera, si en la sesión de trabajo desde donde llamamos a `graficaderivadaEPS` existiese una variable `aux` con un cierto valor asignado, dicha asignación no se modificará al llamar a `graficaderivadaEPS`.

En todos los lenguajes de programación en los que se pueden definir funciones existe la posibilidad de definir *variables locales*, con la finalidad que acabamos de indicar. Aprender a programar funciones usando correctamente las variables locales que sean necesarias es uno de los ingredientes fundamentales de esta parte de la asignatura.

Hay un detalle importante que conviene poner de manifiesto. Si nos fijamos atentamente en el código que define la función `graficaderivadaEPS` vemos que todas las variables que entran en juego en dicha función, o bien se reciben como argumentos (caso de: `f`, `x`, `a`, `b` y `filename`), o bien son variables locales (caso de: `aux`). Es decir, en esta función *no se hace uso de ninguna variable global* que pudiera estar definida en la sesión de trabajo en wxMAXIMA en curso. Esta es la forma correcta en que deben escribirse las funciones, ya que así es como garantizamos su correcto funcionamiento en *cualquier* sesión de trabajo, independientemente de las variables globales que podamos haber definido en una sesión de trabajo en concreto. Por ejemplo, si eliminásemos la `f` de la lista de argumentos de `graficaderivadaEPS`, el correcto funcionamiento de esta función estaría condicionado a que exista en la sesión de trabajo una variable global, llamada `f`, cuyo valor sea la expresión matemática $f(x)$ sobre la que queremos operar, pero no funcionará correctamente en caso contrario. Al fin y al cabo cuando escribimos una función y la guardamos en un archivo, lo hacemos con la finalidad de poder emplear dicha función posteriormente, en otras sesiones de trabajo, que posiblemente serán muy distintas de la sesión de trabajo en curso, y por tanto tendrán variables globales totalmente distintas.

Veamos otro ejemplo de programación de funciones con el comando `block`. En este capítulo hemos estudiado el uso de `next_prime(`n`)` para calcular cuántos números primos existen con valores menores que un número dado; para ello comenzábamos definiendo una variable global `i:1;`, cuyo valor posteriormente íbamos asignando a los sucesivos números primos por medio de `i:next_prime(i)`, contando el número de veces que es necesario realizar esta asignación para superar el valor del número dado fijado al principio. Evidentemente, es poco práctico tener que contar el número de asignaciones realizadas "a mano", especialmente si queremos calcular cuántos números primos existen por debajo de un número x elevado (p. ej. 10^6). Lo más razonable es programar en una función todas las operaciones anteriores, de modo que sea el MAXIMA quien se encargue de contar. La forma más sencilla de programar esta operación es por medio de un *bucle*, empleando para ello la instrucción `for ... while ... do` (cuyo funcionamiento puede consultarse en la ayuda). El código que implementa esta función es el siguiente:

```
numprimosmenoresque(x) := block( [N, i, contador],
      contador : 0,
      i : 1,
      for N : 1 step 1 while next_prime(i) < x do (
```

```
        contador : N, i : next_prime(i) ),
 return(contador)
)$
```

Empleando esta función se puede ver que exiten 78498 números primos menores que 10^6, y 148933 menores que 2×10^6. ¿Hubiéramos podido calcular esto contando?

Resumiendo, en esta sección hemos visto la **sintaxis** empleada para definir **funciones** en MAXIMA, hemos visto la diferencia existente entre **variables locales** y **globales**, y finalmente hemos visto que cuando definimos una función, en general, esta debe recibir como argumentos a **todas** las variables necesarias para obtener el output deseado, y debe tener definidas como **variables locales** a **todas** las variables auxiliares necesarias para los pasos intermedios que sean precisos.

2.6. Aprendiendo MAXIMA

2.6.1. Niveles de uso de MAXIMA

Cualquier sistema de álgebra computacional, como el MAXIMA, se puede usar a distintos niveles, tal y como resumimos a continuación.

En nuestro caso el nivel más básico consiste en abrir una sesión de WXMAXIMA, escribir en ella algunas expresiones y posteriormente manipularlas por medio de las funciones que la ventana de WXMAXIMA nos ofrece en sus menús: `Ecuaciones`, `Álgebra`, `Análisis`, `Simplificar`, `Gráficos` y `Numérico`. Este nivel de *usuario básico* nos ofrece una capacidad de computación similar a la de una calculadora científica muy avanzada, con capacidad para hacer gráficas y con la posibilidad adicional de guardar la sesión de trabajo en un archivo, lo que nos permite volver a abrir en otro momento nuestra sesión de trabajo y tomarla como punto de partida para cálculos posteriores. En este sentido tenemos las opciones "guardar como archivo de tipo `wxmx`" para conservar la sesión de trabajo completa (input y output), o "guardar como archivo de tipo `wxm`" para guardar solo el input.

Para acceder a este nivel de uso basta, por tanto, con tener el programa instalado en el ordenador, escribir alguna expresión y posteriormente explorar un poco las posibilidades de cálculo, representación gráfica y manipulación de expresiones matemáticas que nos ofrecen los menús del WXMAXIMA, es decir, basta con abrir el programa WXMAXIMA y comenzar a explorar los menús, sin necesidad de tener conocimientos previos sobre este lenguaje de programación. Por supuesto que todos hemos sido *usuarios básicos* cuando hemos empezado a usar este lenguaje, no hay nada malo en ello.

El siguiente nivel, que podríamos denominar de *usuario medio*, consiste en no limitarse a manipular expresiones por medio de los comandos disponibles en los menús del WXMAXIMA, sino en emplear comandos o funciones adicionales del lenguaje MAXIMA no disponibles en estos menús. Al cabo de unas pocas sesiones de trabajo, cualquier usuario básico con un poco de curiosidad se convierte sin darse cuenta, de manera natural, en un usuario medio. Para ello basta con explorar un poco el menú **Ayuda** del WXMAXIMA. Una forma muy práctica de ir aprendiendo es la siguiente:

- Sin necesidad de tener conocimientos previos de MAXIMA seleccionamos alguna función que nos interese de las disponibles en los menús del WXMAXIMA. Por ejemplo, si nos interesa factorizar polinomios seleccionamos el menú:

`Ecuaciones/Raices de un polinomio`. Al hacer esto aparece una celda con el comando `allroots(%)`.

- Lo primero que podemos hacer para ver cómo se usa este comando es visalizar un ejemplo de uso. Para ello seleccionamos con un doble click la función `allroots` y nos vamos al menú: `Ayuda/Ejemplo`. Esto nos genera una celda en la sesión de trabajo con la instrucción `example(allroots);`, que nos ofrece un ejemplo sencillo de uso de esta función.

- Para aprender un poco más sobre la función `allroots` la seleccionamos con un doble click y posteriormente nos vamos al menú: `Ayuda/Ayuda de Maxima`. Esto nos lleva al correspondiente capítulo de la documentación del MAXIMA, donde está toda la información sobre el tema seleccionado. Pulsando el botón "Buscar", en la barra lateral izquierda de la ventana de ayuda, accedemos directamente a la información específica sobre la función seleccionada (`allroots`). Leyendo esta información específica aprenderemos exactamente qué es lo que hace dicha función, qué limitaciones tiene, qué opciones de uso existen, etc.

  Además, explorando un poco el capítulo de ayuda localizaremos inmediatamente otros comandos o funciones del MAXIMA, no disponibles en los menús del WXMAXIMA, relacionados con el comando que habíamos seleccionado al principio. De esta forma aprenderemos rápidamente, en unas pocas sesiones de trabajo, a usar otras muchas funciones del MAXIMA y opciones de uso de estas, aparte de las que vienen en los menús.

- Evidentemente la forma de aprender MAXIMA es usándolo, y para que esta tarea no nos resulte demasiado ardua lo que debemos hacer es usarlo para resolver problemas que nos parezcan interesantes, explorando en la documentación (menú: `Ayuda/Ayuda de Maxima`) qué herramientas nos ofrece este lenguaje de programación para resolver el problema en que estemos trabajando.

  También hay que recordar que además de la ayuda del MAXIMA existe una cantidad muy considerable de documentación adicional en la web, sobre todo en lengua inglesa, incluyendo foros de usuarios avanzados, documentos, vídeos con tutoriales, etc. todos ellos fácilmente localizables por medio de cualquier buscador.

Estos son los pasos que hemos seguido todos para pasar de *usuarios básicos* a *usuarios medios*.

El siguiente nivel de uso de MAXIMA, y objetivo de esta parte de la asignatura, es el de *usuarios avanzados*. Ser un usuario avanzado no consiste en saberse de memoria cientos de comandos de MAXIMA, sino en saber programar funciones con este lenguaje de programación. Para ello es imprescindible tener algunos conocimientos básicos sobre sintaxis y funciones del MAXIMA y sobre todo hay que saber buscar la información necesaria para localizar las herramientas de este lenguaje que podemos necesitar en un problema concreto. Como decíamos antes, ningún usuario de MAXIMA conoce todas las funciones disponibles en este lenguaje, lo que caracteriza a un usuario avanzado es que es capaz de localizar las funciones que necesita usar, y es capaz de escribir sus propias funciones cuando lo que quiere hacer no está programado en ninguna función o comando propio del lenguaje MAXIMA.

2.6.2. Comandos e instrucciones de uso frecuente

En los ejemplos disponibles en los apuntes y PECs resueltas de cursos anteriores pueden encontrarse diversos ejemplos de uso de bucles y otras instrucciones empleadas frecuentemente en programación con MAXIMA. A modo de introducción, a continuación mencionamos un breve listado de comandos e instrucciones de uso frecuente. Por supuesto que es en la ayuda del MAXIMA donde debe consultarse la información completa sobre uso, sintaxis, opciones, ejemplos, etc. acerca de estas instrucciones:

- Operaciones básicas
  - Para las operaciones básicas se emplea: `+`, `-`, `*`, `/`. Para el producto matricial se emplea `.` (las matrices que multiplicamos deben tener dimensiones compatibles, de lo contrario MAXIMA no podrá realizar el producto y nos informará de un error). Para la operación "elevar a una potencia" se emplea el símbolo `^` (o `^^` en el caso de potencias de matrices). Es importante que el símbolo `^` aparezca realmente escrito en el código (para ello es necesario pulsar la tecla seguida de un espacio), de lo contrario lo que hacemos es introducir un superíndice que, con frecuencia, MAXIMA no interpretará como un exponente. Para las funciones matemáticas habituales MAXIMA cuenta con las funciones habituales `sin`, `cos`, `log`, etc. En la ayuda de MAXIMA podremos encontrar sin ninguna dificultad cualquiera de estas funciones a medida que nos vayan haciendo falta.
- Asignaciones
  - MAXIMA es distinto de la inmensa mayoría de los lenguajes de programación en lo referente a asignar valores a variables. En MAXIMA, para asignar un valor a una variable se emplea el símbolo "`:`". El símbolo "`=`" en MAXIMA se emplea para definir ecuaciones. Por ejemplo, si queremos asignar el valor 3 a la variable `x` hacemos `x:3;`. Si queremos que la variable `EQ` contenga la ecuación `LHS=RHS` hacemos `eq:LHS=RHS`.
  - Aparte de asignar valores a variables, una operación que emplearemos constantemente es **definir funciones**. Para definir funciones en MAXIMA se emplea el símbolo `:=`, y con frecuencia es conveniente usar el comando `block`.
- Bucles

  En programación los bucles son una de las instrucciones más habituales y útiles. En MAXIMA para hacer un bucle hacemos, en general, lo siguiente:

  `for` índice `:` valor inicial `thru` valor final `step` incremento `do` instrucciones`;`

  Por ejemplo, el siguiente bucle escribe los primeros 50 múltiplos de 3:

  ```
  for i : 1 thru 50 step 1 do print(3*i);
  ```

  El mismo resultado se obtiene con este otro bucle

  ```
  for i : 3 thru 150 step 3 do print(i);
  ```

  Aparte de "`thru` valor final" la instrucción que define cuándo debemos interrumpir la ejecución del bucle también puede programarse por medio de una condición, empleando para ello la instrucción `while`, por ejemplo

```
for N : 1 step 1 while (N <= 10 - N) do print("N = ", N);
```

Además de esto, similarmente a todos los lenguajes de programación MAXIMA dispone del comando `if`, para introducir condiciones, con la sintaxis habitual `if ... then ... else`, o también `if ... then... elseif ...`, etc.

- Listas

  Con mucha frecuencia trabajaremos con listas de objetos, que podrán ser números, matrices, ecuaciones, vectores, cadenas de texto, gráficas, otras listas, etc. La instrucción para definir una lista es:

  `makelist(` elemento, índice, valor inicial, valor final, incremento `);`

  Por ejemplo, esta instrucción genera una lista que contiene los números pares de 0 a 10:

  ```
  makelist(i, i, 0, 10, 2);
  ```

  Equivalentemente lo mismo se consigue con

  ```
  makelist(2*i, i, 0, 5, 1);
  ```

2.6.3. ¿Cómo preparar esta parte de la asignatura?

La forma en que está diseñada esta parte de la asignatura es la siguiente. En estos apuntes *no* se reproduce la información disponible en el manual del MAXIMA (menú: `Ayuda/Ayuda de Maxima`). Estos apuntes contienen diversos ejemplos de programación de funciones, unas muy sencillas, otras un poco más complicadas, por medio de las cuales vamos a ir explorando algunas de las muchísimas posibilidades que nos ofrece este lenguaje de programación. En todo momento hemos intentado que los problemas matemáticos sobre los que versan estos ejemplos resulten interesantes y útiles. Por este motivo los ejemplos están relacionados con problemas de matemáticas de interés en física.

La manera de trabajar con estos apuntes es la siguiente.

- De manera secuencial (uno a uno y en orden) vamos cargando las funciones programadas en los ejemplos en una sesión de trabajo.

- Leemos cuidadosamente los comentarios que aparecen en dichas funciones, donde se explica qué es lo que hace cada línea de código.

- Ponemos en uso estas funciones en una sesión de trabajo y de esta forma vemos cómo funcionan.

- En las funciones que aparecen como ejemplos en estos apuntes se usan diversos comandos o funciones propios del lenguaje MAXIMA:

  - Es fundamental estudiar en la documentación del MAXIMA la información disponible sobre estas funciones, no para aprendérselas de memoria, sino para entender qué es lo que hace cada línea de código de las funciones que ponemos como ejemplo y también para aprender qué otras opciones nos ofrece este lenguaje de programación.

- Para adquirir *experiencia* suficiente es fundamental recorrer todos los ejemplos disponibles en todos los apuntes, independientemente de que estén relacionados, o no, con los problemas propuestos en la PEC del curso actual. No es recomendable esperar a que se publique la PEC y posteriormente intentar resolverla consultando solo el capítulo relacionado con la PEC del año en curso.
- Es muy importante no olvidarse de trabajar también los ejemplos disponibles en el capítulo de "Exámenes resueltos de cursos anteriores".

Esto último nos lleva al tema de la evaluación de la asignatura. Aunque toda la información que ponemos a continuación viene en la Guía del Curso, lo cierto es que todos los años se pregunta también en los foros de la asignatura (con respuesta: "lea la Guía del Curso"). La evaluación de esta asignatura se basa en una PEC para cada parte (programación en MAXIMA y en C) más un examen presencial.

- Aunque la mayor parte de la nota se basa en las dos PECs, para aprobar la asignatura es imprescindible tener aprobadas las dos PECs y el examen presencial.
- En cada curso hay dos convocatorias, junio y septiembre, tanto para las PECs como para el examen presencial.
- Se guarda para septiembre la calificación obtenida en junio en cualquiera de estas tres pruebas (PEC de MAXIMA, PEC de C y examen presencial), de forma que **en septiembre solo hay que presentarse a la parte (o partes) no aprobadas en junio**.
- Si después de las dos convocatorias (junio y septiembre) aún queda alguna prueba sin aprobar, entonces es necesario repetir la asignatura completa. Es decir, los aprobados de las PECs y examen presencial se guardan de junio para septiembre, pero no se guardan de un curso para el siguiente.
- Obviamente, a todos los efectos las PEC de MAXIMA y C tienen categoría de **examen**, por tanto deben realizarse de manera individual.
- En caso de localizarse intentos de *plagio* entre compañeros se opera de manera similar a como se hace en un examen presencial, es decir, se notifica al servicio de inspección y se aplican las sanciones correspondientes (que incluyen suspender la asignatura y en algunos casos pérdida de matrícula u otras sanciones).
- En el caso particular de La PEC de MAXIMA:
  - La PEC de MAXIMA consiste en la programación de varias funciones cuya finalidad se explica con todo detalle en el enunciado (ver capítulo de "Exámenes resueltos de cursos anteriores").
  - El enunciado de la PEC de MAXIMA de cada curso, en la convocatoria de junio, se publica en el curso virtual aproximadamente un mes antes de la fecha final de entrega pedida. Para la convocatoria de septiembre el enunciado se publica a primeros de julio.
  - Las funciones pedidas en la PEC de MAXIMA son algo avanzadas, para poder programarlas es necesario haber adquirido cierta experiencia previamente, trabajando con los ejemplos aportados en los apuntes.

  - Para entregar la PEC de MAXIMA hay que subir al curso virtual (Menú: `Entrega de trabajos`) un **archivo de texto plano** con el código en MAXIMA que define las funciones pedidas. Esto quiere decir que en la PEC de MAXIMA **no hay que entregar una sesión de trabajo en** WXMAXIMA, sino el archivo de texto plano con el código mencionado.
  - El **nombre** de dicho archivo debe contener los *dos apellidos* del alumno.
  - La **extensión** de dicho archivo puede ser `.mac`, o `.mc`, o `.m`, o incluso `.txt`: **da exactamente lo mismo**. De todas formas, a los archivos de código en MAXIMA es costumbre ponerles la extensión `.mac`, o `.mc`.
  - Las aclaraciones, explicaciones o comentarios sobre las funciones programadas que cada estudiante quiera incorporar a su PEC deben ir codificadas en el interior del mencionado archivo de código, escritas como *comentarios*, delimitadas por los caracteres que se emplean en lenguaje MAXIMA para escribir comentarios: `/* ...*/`.

- Para calificar esta PEC el equipo docente procederá a cargar en una sesión de MAXIMA el código aportado por cada estudiante, verificando el correcto funcionamiento de las funciones que se pedían. Posteriormente el equipo docente abrirá, por medio de un editor de textos, el archivo de texto plano subido al curso virtual, y se examinará el código aportado. En este sentido se tendrá en cuenta la claridad del código y los comentarios introducidos por el estudiante en el mismo para facilitar su lectura.

En cualquier caso, a modo de resumen, las PECs de MAXIMA de cada año son similares, en contenido y dificultad, a las de años precedentes. Lo que se pide es algo similar a los archivos de código (`*.mc`) disponibles en la carpeta "PECS de Maxima" del curso virtual. Una aclaración más, cada uno de estos archivos `*.mc` es un archivo de *texto plano*; al bajarlos de la carpeta del curso virtual donde están alojados, algunos navegadores les añaden la extensión `.bin`, esto no afecta de ninguna manera al contenido del archivo. Si al bajar del curso virtual un archivo `.mc` obtiene un archivo con la extensión adicional `.bin`, sencillamente, acceda al menú de "cambiar nombre de archivo" de su SO y elimine dicha extensión adicional.

2.6.4. Programando funciones y cargando archivos de código desde sesiones de trabajo

Cuando se escribe una función, una subrutina o cualquier otro ingrediente de un programa de ordenador en un cierto lenguaje de programación, lo que se está escribiendo es un **código fuente**, o sencillamente un **código**. Este código que uno escribe se guarda en un tipo de archivo denominado **archivo de texto plano**, que se caracteriza por contener, solamente, *texto plano*.

En el capítulo de introducción hemos explicado qué es un *archivo de texto plano*, de modo que no vamos a repetir aquí dicha información. Antes de continuar con el estudio del lenguaje de programación MAXIMA es absolutamente fundamental tener claro este concepto.

En el capítulo de introducción también hemos mencionado que los programas de ordenador que se emplean para leer y escribir archivos de texto plano son los *editores de texto*, y hemos aportado enlaces a diversas páginas de Wikipedia donde se explica

este concepto con mayor profundidad y donde se expone un listado muy completo (y actualizado) de editores de texto, incluyendo información sobre cuáles son gratuitos y cuáles no y sobre cuáles están disponibles dependiendo del sistema operativo en uso. Los editores de texto son una herramienta tan fundamental en informática que todos los sistemas operativos (por malos que sean) incorporan al menos uno. Sin embargo, en algunos sitemas operativos el editor de texto que viene *por defecto* es **tan malo** (nos referimos, por supuesto, al *notepad* de MS Windows) que recomendamos encarecidamente el uso de *cualquier otro*. Por tanto, es fundamental antes de continuar explorar un poco qué editores de texto (por supuesto gratuitos) tenemos disponibles para nuestro sistema operativo, y escoger uno que nos guste. Por ejemplo, el KWrite, el Kate o el Geany son bastante recomendables

`https://en.wikipedia.org/wiki/KWrite`

`https://kate-editor.org/`

`https://www.geany.org/`

pero en cualquier caso hay total libertad para que cada uno emplee el que considere más conveniente.

Cuando se trabaja en WXMAXIMA una posibilidad es usar la propia ventana del WXMAXIMA como editor de texto. Para ello escribimos la función que queremos guardar en una celda del WXMAXIMA y posteriormente vamos al menú: `Archivo/Exportar`, seleccionamos en el menú desplegable de la esquina inferior derecha la opción "maxima batch file (*.mac)", escogemos el directorio y nombre de archivo donde queremos guardar nuestro código y pulsamos el botón "Guardar". Al hacer esto hemos generado un archivo de texto plano en el directorio seleccionado, con el nombre de archivo que hayamos escrito y con la extensión `.mac`. El contenido de dicho archivo es el *input* de todas las celdas de la sesión de trabajo en WXMAXIMA desde donde lo hayamos salvado. Si en la sesión de trabajo en uso tenemos multitud de celdas y solo queremos guardar el código de una de ellas una posibilidad es *copiar* la celda a guardar, *pegarla* en una sesión nueva de WXMAXIMA (menú: `Archivo/Nuevo`), y exportarla desde esta sesión nueva (hay que recordar evaluar esta celda antes de exportarla, para que WXMAXIMA interprete la celda como input y la exporte correctamente).

La anterior forma de trabajar es una posibilidad que **desaconsejamos**. Es infinitamente más cómodo, más práctico y más sencillo usar un editor de texto *de verdad* para escribir el código, y usar la ventana de WXMAXIMA solo para las sesiones de trabajo. Aparece entonces la cuestión

- ⋆ ¿qué parte del trabajo va en el archivo de texto plano *externo* al WXMAXIMA y qué parte va en la sesión de WXMAXIMA?

vamos a respoder esta cuestión con un ejemplo sencillo.

Cuando se hace un código para resolver un problema concreto es muy conveniente hacerlo de forma que pueda reutilizarse en el futuro, si es que tenemos que resolver el mismo problema u otro similar. Para ello es importante que el código esté *bien documentado*, es decir, que esté escrito de forma que si lo leemos dentro de varios meses (o años) entendamos rápidamente qué es lo que hace, de modo que podamos usarlo o bien directamente o bien modificándolo si es necesario.

Supongamos que para un problema concreto necesitamos visualizar por pantalla la gráfica de una función $f(x)$ junto con su primera y segunda derivadas, y al mismo tiempo guardar en un archivo `.eps` esta gráfica para incorporarla posteriormente a un documento. En ese caso lo más práctico sería programar una función como la `graficaderivadaEPS(f, x, a, b, filename)` del ejemplo anterior, y guardarla en un archivo de texto plano. Posteriormente, para cada uno de los casos de estudio donde apliquemos esta función abriríamos una sesión de WXMAXIMA, desde donde cargamos la función y la empleamos para generar los resultados correspondientes al caso concreto que estemos estudiando (por ejemplo: $f(x) = x^3$ con $x \in [-1, 1]$.

Las funciones de *propósito más o menos general* que vamos escribiendo van formando nuestra *biblioteca de funciones*, y deben guardarse en un árbol de directorios con cierto orden, de forma que (cuando tengamos muchas) no resulte difícil localizar una de ellas.

De esta forma iremos generando poco a poco una biblioteca de funciones que podremos usar en el futuro para resolver rápidamente problemas similares a cosas que hayamos resuelto en el pasado. Cuando se trabaja de esta manera es impresionante la cantidad de trabajo que se puede realizar en poco tiempo. Por el contrario si somos desordenados y no hacemos las cosas bien, cada vez que tengamos que resolver cualquier problema tendremos que partir de cero, con lo que será realmente difícil que lleguemos demasiado lejos. Además, si el código está bien ordenado y bien documentado será más fácil identificar los posibles errores (*bugs*) que hayamos cometido al escribirlo, localizar un *bug* en un código desordenado o mal estructurado puede ser prácticamente imposible.

La siguiente cuestión que aparece de manera lógica es

⋆ ¿cómo se cargan estas funciones desde WXMAXIMA?

Una posibilidad que funciona pero que **desaconsejamos** por ser incómoda es ir al menú: `Archivo/Archivo por lotes`, seleccionar el archivo que contiene la función que queramos cargar y presionar el botón “Abrir” de la esquina inferior derecha.

La manera cómoda de cargar archivos de texto plano con código, evaluando su contenido, desde WXMAXIMA es por medio de la instrucción

```
batchload(filename);
```

el argumento `filename` de esta función es una *cadena de caracteres* (o variable de tipo *string*) con el nombre completo del archivo a abrir (extensión incluida). En caso de no suministrar la localización de este archivo en el disco (es decir, el *path*) WXMAXIMA lo buscará en los directorios definidos por defecto, y dará un error en caso de no encontrarlo. Los directorios en los que WXMAXIMA busca por defecto están definidos en la variable `file_search_maxima`. Para visualizar el contenido de esta variable podemos evaluar la instrucción

```
file_search_maxima;
```

Para *añadir* más directorios (p. ej., `mi-directorio`) a la anterior lista podemos usar la función `append`, que sirve para combinar listas

```
append(file_search_maxima, ["mi-directorio"]);
```

Para que la instrucción anterior funcione correctamente la cadena de caracteres `"mi-directorio"` debe contener la dirección completa del directorio indicado en el disco.

De todas formas, en lugar de modificar el contenido de `file_search_maxima` la forma de trabajar que nos parece más recomendable es indicar a `batchload` la ruta completa del archivo a cargar.

Por ejemplo, supongamos que estamos resolviendo problemas para la asignatura de mecánica. Para un problema concreto necesitamos trabajar con la ecuación de Newton

$$\boldsymbol{F} = m\,\boldsymbol{a}$$

y con las ecuaciones de Euler-Lagrange

$$\frac{\mathrm{d}}{\mathrm{d}t}L_v(t, q(t), q'(t)) - L_x(t, q(t), q'(t)) = 0$$

para las cuales hemos programado diversas funciones guardadas en los archivos `ec-Newton.mc` y `ecs-Euler-Lagrange.mc`. Si trabajamos de una manera más o menos organizada, tendremos definido un cierto directorio (p. ej. `bib`) dentro de nuestro directorio personal (`/home/usuario` o `C:/Users/usuario`, en Windows), donde alojaremos nuestra biblioteca de funciones, de forma que la localización de los archivos anteriores en el disco será algo parecido a (p. ej.):

```
/home/usuario/bib/Maxima/mecanica/ec-Newton.mc
/home/usuario/bib/Maxima/mecanica/ecs-Euler-Lagrange.mc
```

Además, con toda probabilidad el directorio `/home/usuario/bib/Maxima/mecanica/` será también el depositario de cualquier otro archivo con funciones de interés para mecánica programadas en MAXIMA. En este caso una forma cómoda de trabajar para cargar estos archivos desde una sesión de WXMAXIMA es asignar una variable de tipo *string* al directorio donde están las funciones que queremos cargar. Para ello, dentro de la sesión de trabajo de WXMAXIMA hacemos

```
path_mec : "/home/usuario/bib/Maxima/mecanica/";
```

(obsérvese el uso de comillas dobles, a fin de que la variable `path_mec` sea de tipo *string*) y posteriormente empleamos la función de *concatenación* `concat` para generar el path completo de los archivos a cargar

```
batchload( concat( path_mec, "/ec-Newton.mc") );
batchload( concat( path_mec, "/ecs-Euler-Lagrange.mc") );
```

operando de manera similar con otros archivos que queramos cargar.

2.6.5. Errores frecuentes

- Tanto si se trabaja con SO MS Windows o con sistemas tipo UNIX, como el Linux, cuando especificamos la ruta o path de un archivo en el disco en WXMAXIMA hay que indicar los sub-directorios por medio del carácter empleado en UNIX: "`/`", en lugar del *backslash* empleado en Windows.

- No se deben usar caracteres no estándar en los nombres de archivos y/o directorios, esto incluye

  - caracteres con tilde,
  - espacios,
  - letra "ñ",
  - etc.

  Aunque el sistema operativo nos permita generar nombres de archivo y/o directorios empleando caracteres no estándar, luego es una fuente de problemas

para muchos programas. Por ejemplo, la instrucción `batchload` que acabamos de mostrar no funcionará si en el nombre del archivo o en el path existe algún carácter no estándar. Véase la información a este respecto en el capítulo de introducción.

- Verificar muy cuidadosamente la sintaxis de las funciones que se programan. Cualquier error sintáctico, por pequeño que sea, convierte un código en algo totalmente incomprensible para el MAXIMA. Los errores sintácticos más frecuentes son

  - Paréntesis sin cerrar o mal cerrados (por ejemplo, cerramos con "`]`" un paréntesis abierto con "`(`", en cualquier lenguaje de programación estos caracteres tienen significados y finalidades distintos).
  - Falta una coma ",", o se ha sustituido por otro signo de puntuación (`.`, `;`, ...). Igual que antes, en cualquier lenguaje de programación los distintos signos de puntuación son caracteres reservados con significados y finalidades distintos.
  - "Hay una errota (falta una ltra, se ha sustituido una letri por utra, ...)."

    Aunque los seres humanos somos capaces de darnos cuenta instantáneamente de este tipo de errores, y podemos comprender sin dificultad el significado de una señal con ruido, como un texto lleno de errores, las máquinas no son tan flexibles. Un ordenador lee y ejecuta exactamente lo que pone en el código, si el código tiene un error el programa o bien no funcionará o, peor aún, funcionará haciendo algo diferente de lo que creemos que hace.

Todos estos errores son sencillos de cometer si se trabaja demasiado deprisa, y también es fácil localizarlos si se examina el código con atención. La claridad y limpieza del código es fundamental en este sentido.

Como regla general, cuando algo que hayamos programado no funcione, o WXMAXIMA se demore demasiado en generar una respuesta, lo primero que debemos sospechar es que quizá hay un error sintáctico en alguna parte.