Prácticas de Matemáticas I con Maxima.

Jose Salvador Cánovas Peña.
Departamento de Matemática Aplicada.

Contents

¿Qué es Maxima?

Maxima es un programa que permite hacer cálculos matemáticos complicados con gran rapidez. Para entendernos, es como una especie de calculadora gigante a la que no sólo podemos pedirle que haga cálculos numéricos, sino que también hace derivadas, cálculo de primitivas, representación gráfica de curvas y superficies, factorización de polinomios etc.

Abordamos en esta práctica una iniciación a Maxima partiendo desde cero, e intentaremos poner de manifiesto su utilidad a la hora de trabajar con expresiones matemáticas complicadas, permitiendo hacer éstas con poco coste de tiempo.

Será necesaria por parte del alumno una lectura previa de esta práctica antes de empezar a trabajar con el programa. Esta lectura previa tiene por objeto el conocimiento de ciertas sentencias clave que permiten el manejo del programa. Al igual que al aprender el manejo de una calculadora científica es necesario leer las instruciones de la misma, estas notas pueden ser útiles para aprender el manejo de Maxima.

Por otra parte, a pesar de la potencia evidente del programa, hemos de hacer notar que es necesario por parte del alumno un conocimiento matemático teórico de todas las funciones y sentencias que vamos a usar. Por ejemplo, aunque una calculadora multiplica números con suma facilidad, sólo nos percatamos de su potencia en cuanto conocemos dicha operación y somos capaces de realizarla de un modo mucho más lento. Con Maxima ocurre lo mismo. Sólo conociendo teóricamente las operaciones que Maxima realiza nos percataremos de su indudable utilidad.

Chapter 1

Preliminares

Cuando se arranca Maxima, aparece una pantalla blanca vacía. En ella podemos escribir aquellas operaciones que queremos que realice. Una vez tecleada la operación, hemos de pulsar las teclas *shift + enter* para obtener el resultado. Por ejemplo, supongamos que queremos hacer la operación 2+2. Teclearemos entonces

$$2 + 2$$

en la pantalla. A continuación pulsamos *shift + enter* y aparecerá lo siguiente en pantalla:

$$\begin{array}{ll} (\%i1) & 2 + 2; \\ (\%o1) & 4 \end{array}$$

siendo %i1 la entrada uno, proporcionando la salida %o1.

Además se pueden realizar las siguiente operaciones algebraicas:

$$\begin{array}{rcl} x + y & \rightarrow & suma \\ x - y & \rightarrow & resta \\ x/y & \rightarrow & división \\ x * y & \rightarrow & producto \\ x\hat{}y & \rightarrow & potencia. \end{array}$$

Actividad 1 *Realizar las siguientes operaciones con Maxima:*

(a) $3.75 + 8.987 =$

(b) $(2-3.1)^{23} =$

(c) $\frac{2.4+3^2}{4*7.2^2} =$

(d) $2 \times 10^2 + 3 \times 10^{-3} =$

(e) $\left(\frac{2.3*4}{2-4.5^2}\right)^{56} =$

A la hora de trabajar con Maxima, hemos de tener en cuenta que hay dos modalidades admisibles. Tomemos por ejemplo la operacion

$$2 \times 10^2 + 3 \times 10^{-3}.$$

Si introducimos la sentencia

$$2 * 10^2 + 3 * 10^{-3}$$

obtendremos al ejecutarla

$$(\%i1) \quad 2 * 10^2 + 3 * 10^{-3};$$
$$(\%o1) \quad \frac{200003}{1000}$$

mientras que si escribimos

$$2.0 * 10^2 + 3 * 10^{-3}$$

obtendremos

$$(\%i2) \quad 2.0 * 10^2 + 3 * 10^{-3};$$
$$(\%o2) \quad 200.003$$

Como vemos, en la primera obtenemos una fracción, mientras en la segunda obtenemos un número decimal. Pero las diferencias van más allá de esta apariencia: en la primera el programa ha trabajado con precisión infinita y el resultado que presenta es exacto. En la segunda se ha trabajado (el programa lo hará automáticamente al detectar un número decimal, 2.0 en este caso) con precición finita y por lo tanto con errores de redondeo. Estos errores no afectan a esta operación, pero sí lo hacen a la operación

$$(1/10)\hat{}4,$$

que en precisión infinita se obtiene

$$(\%i1)\ \ (1/10)\hat{}4;$$
$$(\%o1)\ \ \frac{1}{10000}$$

mientras que en finita tenemos

$$(\%i2)\ \ (1.0/10)\hat{}4;$$
$$(\%o2)\ \ 1.0000000000000005\ 10^{-4}$$

La función **float** se puede utilizar para convertir un número en decimal de doble precisión, que es la que por defecto utiliza el programa. Así por ejemplo

$$(\%i3)\ \ \text{float}((1/10)\hat{}4);$$
$$(\%o3)\ \ 1.0\ \ 10^{-4}$$

Igualmente la operación

$$(\%i4)\ 2\hat{}100;$$
$$(\%o4)\ 1267650600228229401496703205376$$

pero si escribimos

$$(\%i5)\ \text{float}(2\hat{}100);$$
$$(\%o5)\ 1.2676506002282294\ 10^{30}$$

Los números de la forma 3.05×10^{-3} en coma flotante pueden expresarse escribiendo el exponente como "f", "d" o "e". Por ejemplo

$$3.05 \times 10^{-3},\ 3.05e^{-3},\ 3.05d^{-3},\ 3.05f^{-3}$$

Para saber si el programa está trabajando en precisión finita o infinita tenemos la sentencia **numer**. Esta sentencia, si la ejecutamos nos devuelve el valor por defecto **false**, lo que significa que el programa está trabajando en precisión infinita. Si introducimos

$$(\%i6)\ \text{numer:true};$$
$$(\%o6)\ true$$

pasamos a trabajar con precisión finita. Para activar la precisión infinita debemos teclear

$$(\%i7)\ \text{numer:false;}$$
$$(\%o7)\ false$$

Los números complejos se introducen teniendo en cuenta que la unidad imaginaria i se escribe %i. Así, el número complejo $2+3i$ se escribe $2+3*\%i$. Las operaciones con números complejos son análogas a las de con números reales, disponiendose además de las siguientes funciones propias de la naturaleza de los complejos:

- realpart -> Parte real del número complejo.
- imagpart -> Parte imaginaria del número complejo.
- rectform -> Devuelve la expresión rectangular del número complejo.
- polarform -> Devuelve la expresión polar del número complejo.
- cabs -> Devuelve el módulo del número complejo.
- carg -> Devuelve el argumento del número complejo.
- conjugate -> Devuelve el conjugado del número complejo.

No obstante, maxima no ejecuta multiplicaciones y divisiones si no las introducimos dentro de la sentencia **expand**, que se utiliza para desarrollar ciertas operaciones. Por ejemplo

$$(\%i1)\ (2+\%i)*(2-\%i);$$
$$(\%o2)\ (2+\%i)*(2-\%i)$$

mientras que

$$(\%i1)\ \exp and((2+\%i)*(2-\%i));$$
$$(\%o2)\ 5$$

Actividad 2 *Realizar las siguientes operaciones con Maxima:*

(a) $(3+4i)\cdot(8+i) =$

(b) $(3+4i)/(8+i) =$

(c) $(3+4i)\cdot(8+i) - i =$

(d) Obtener los módulos, argumentos, formas polar y rectangular, conjugados y partes real e imaginaria de los números obtenidos en las operaciones (a)–(c) anteriores.

1.1 Sobre las constantes

Las principales constantes matemáticas elementales se teclean en Maxima del siguiente modo

$$\begin{aligned}
\%e- &> e \approx 2.718281828459045d0\\
\%i- &> \sqrt{-1}\\
ind\ o\ und- &> \text{Representa un valor indeterminado}\\
\inf- &> +\infty\\
\min f- &> -\infty\\
\inf inity- &> \infty \text{ complejo (polo norte de la esfera de Reimann)}\\
\%phi- &> \frac{1+\sqrt{5}}{2} \text{ (razón aúrea)}\\
\%pi- &> \pi \approx 3.141592653589793d0
\end{aligned}$$

Aparte de estas constantes, pueden declarar otras que se puedan necesitar en un momento dado. Por ejemplo, si queremos asignar la constante de la gravitación universal $G = 6.67 \times 10^{-11}$ debemos teclear

$$\begin{aligned}
&(\%i1)\ G : 6.67 \times e - 11;\\
&(\%o1)\ 6.67\ 10^{-11}
\end{aligned}$$

Nótese que usamos : en vez del símbolo de igualdad =. Si tecleamos G y pulsamos shift +enter obtendremos en pantalla

$$\begin{aligned}
&(\%i2)\ G;\\
&(\%o2)\ 6.67\ 10^{-11}
\end{aligned}$$

Maxima distingue entre mayúsculas y minúsculas por lo que si escribimos g, esta no es igual a G. Para eliminar el valor asignado a G tenemos la función

kill. Tecleando

$$(\%i3)\ kill(G);$$
$$(\%o3)\ done$$

desposeemos a G de su valor. Si tecleamos como antes obtenemos en pantalla

$$(\%i4)\ G;$$
$$(\%o4)\ G$$

dado que hemos borrado la variable de entre las definidas.

Por otra parte, si en una misma línea queremos definir varias variables, o escribir varias expresiones debemos separar estas con ";". Por ejemplo

$$(\%i5)\ x:1;y:2;z:x+y$$
$$(\%o5)\ 1$$
$$(\%o6)\ 2$$
$$(\%o7)\ 3$$

que como vemos proporciona 3 salidas. Si queremos borrar todos estas variables declearemos

$$(\%i8)\ kill(all);$$
$$(\%o8)\ done$$

Si no deseamos escribir la variable en una salida basta escribir un "$" al final. Por ejemplo

$$(\%i1)\ x:1;y:2;z:x+y\$$$
$$(\%o1)\ 1$$
$$(\%o2)\ 2$$

y no proporciona la tercera salida de la variable z como anteriormente ocurría.

1.2 Sobre las funciones

Una primera apreciación sobre las funciones en maxima es que estas van definidas en minúsculas con los argumentos entre paréntesis. Las nociones matemáticas más notables las presentamos en la siguiente tabla:

$$\begin{aligned}
\text{sqrt}(x) &= \sqrt{x} \\
\exp(x) &= e^x \\
\log(x) &= \log x \\
\sin(x) &= \sin x \\
\cos(x) &= \cos x \\
\tan(x) &= \tan x \\
\text{asin}(x) &= \arcsin x \\
\text{acos}(x) &= \arccos x \\
\text{atan}(x) &= \arctan x \\
n! &= \textit{factorial de } n \\
\text{abs}(x) &= |x| \\
\text{entier}(x) &= \textit{parte entera de } x
\end{aligned}$$

Así, si escribimos

```
(%i1) sqrt(16);
(%o1) 4
(%i2) Sqrt(2);
(%o2) sqrt(2)
(%i3) float(sqrt(2));
(%o3) 1.414213562373095
```

Actividad 3 *Calcular los siguientes valores:*

(a) $\sin \pi/4 + \cos \pi/7 =$

(b) $\log_2 256 =$

(c) $\left|\arcsin 0.98 + \frac{\log 2}{\sqrt{2}}\right| =$

(d) $e^{10!} =$

(e) $\sqrt{\log 34 + e^{12}} =$

Aparte de las funciones que Maxima tenga integradas, podemos introducir nuevas funciones con la sentencia **define**. Su estructura es

$$define(f(x\_1, ..., x\_n), expr),$$

que definirá una función de nombre f dependiente de las variables $x\_1, ..., x\_n$ según la expresión $expr$, que puede ser escalar o vectorial. Por ejemplo, si queremos definir la función $f(x, y) = xy$, escribimos

$$(\%i1)\ define(f(x, y), x * y);$$
$$(\%o1)\ f(x, y) := x * y$$

con lo que la función estará introducida. Si ahora tecleamos

$$(\%i2)\ f(2, 3);$$
$$(\%o2)\ 6$$

Las funciones también pueden definirse de una forma más cómoda con la expresión

$$(\%i3)\ f(x, y) := x * y$$

1.3 Aprovechando cálculos anteriores

A veces, es posible que tengamos que hacer una sucesión de cálculos consecutivos de manera que cada nueva operación se basa en la anterior. Parece necesaria entonces algo que nos remita a resultados anteriores. Esto se realiza con maxima simplemente llamando a la entrada o salida correspondiente anteponiendo siempre el símbolo %. Por ejemplo, si queremos calcular $\cos(\sin \pi/7)$ tendríamos que calcular primero $\sin \pi/7$, para despues calcular el coseno de dicha cantidad. Esta operación podríamos hacerla del modo siguiente:

$$(\%i1)\ \sin(\%pi/7);$$
$$(\%o1)\ \sin\left(\frac{\pi}{7}\right)$$
$$(\%i2)\ \cos(\%o1)$$
$$(\%o2)\ \cos\left(\sin\left(\frac{\pi}{7}\right)\right)$$
$$(\%i3)\ \text{float}(\%o2)$$
$$(\%o3)\ 0.90733988115078$$

Obviamente, este ejemplo es bastante sencillo ya que la operación en cuestión podría haberse hecho en una sola línea de comandos, pero ilustra bien el modo de proceder cuando se estén realizando operaciones y cálculos más complejos. Finalmente, para llamar a la salida anterior basta usar el símolo %. Así, la operación anterior podría haberse escrito como

$$
\begin{aligned}
&(\%i1)\ \sin(\%pi/7);\\
&(\%o1)\ \ \sin\left(\frac{\pi}{7}\right)\\
&(\%i2)\ \cos(\%)\\
&(\%o2)\ \ \cos\left(\sin\left(\frac{\pi}{7}\right)\right)\\
&(\%i3)\ \ \mathrm{float}(\%)\\
&(\%o3)\ 0.90733988115078
\end{aligned}
$$

Chapter 2

Algebra lineal con Maxima

Una vez introducidas ciertas generalidades sobre el programa, vamos a ver cómo podemos utilizar el éste para agilizar algunos cálculos relativos a los contenidos explicados en la parte teórica de la asignatura. Empezamos por álgebra lineal.

2.1 Vectores y matrices

Para escribir vectores de $\mathbb{K}^n$ debemos de introducir cada una de las componentes del mismo entre corchetes, separándolas por comas. Así, el vector $v = (1, 2, -1, 5)$ de $\mathbb{R}^4$ se tecleará

$$(\%i1)\ v : [1, 2, -1, 5];$$
$$(\%o1)\ [1, 2, -1, 5]$$

Las matrices se introducirán por filas como un vector de vectores, ayudados de la sentencia **matrix**. Así la matriz $M = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \end{pmatrix}$ se escribirá como

$$(\%i2)\ M : matrix([1, 2, -1], [0, 3, 1]);$$
$$(\%i2)\ \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \end{bmatrix}$$

Para la matriz identidad de $M_{n\times n}(\mathbb{K})$ tenemos asignado el nombre

$$ident(n),$$

mientras que tecleando

$$diagmatrix(n, x)$$

introducimos una matriz diagonal de tamaño $n \times n$ con el valor x en cada elemento de la diagonal principal. Si tenemos instroducida la matriz M, las sentencia

$$row(M, i)$$

devuelve la fila i-esima (con formato de matriz), mientras que

$$\text{col}(M, i)$$

devuelve la columna i–esima. Por ejemplo

$$\begin{array}{ll} (\%i1) & M : matrix([1, 2, -1], [0, 3, 1]); \\ (\%i1) & \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \end{bmatrix} \\ (\%i2) & \text{col}(M, 2); \\ (\%o2) & \begin{bmatrix} 2 \\ 3 \end{bmatrix} \\ (\%i3) & \text{row}(M, 1); \\ (\%o3) & [1\ 2\ -1] \end{array}$$

Otra sentencia interesante es

$$entermatrix(n, m)$$

con la que se introduce de forma interactiva una matriz $n \times m$. Si $n = m$ (matriz cuadrada), la función pregunta primero si la matriz que se va a introducir es diagonal, simétrica, antisimétrica o general, y una vez contestado se van introduciendo los valores de la misma.

2.2 Operaciones con vectores

Dados dos vectores v_1 y v_2 y un número real λ, se definen con Mathematica las siguientes operaciones:

$v_1 + v_2$	suma de vectores
$\lambda * v_1$	multiplicación del vector v_1 por λ
$v_1.v_2$	producto escalar de v_1 por v_2

Por ejemplo, si introducimos los vectores de $\mathbb{R}^3$ $v = (1, 2, 1)$ y $u = (0, 1, 0)$ podemos calcular la suma de ambos, $2v$ y el producto escalar de ambos vectores del siguiente modo:

$(\%i1)$ $v : [1, 2, 1]; u : [0, 1, 0];$
$(\%o1)$ $[1, 2, 1]$
$(\%o2)$ $[0, 1, 0]$
$(\%i3)$ $u + v;$
$(\%o3)$ $[1, 3, 1]$
$(\%i4)$ $2 * v;$
$(\%o4)$ $[2, 4, 2]$
$(\%i5)$ $u.v;$
$(\%o5)$ 2

Actividad 4 *Dados los vectores* $v = (1, 2, 3, 4)$, $u = (3, -1, 0, 2)$ *y* $w = (2, 0, 0, 1)$ *calcular:*

(a) $(w.v)u$ *(b)* $u - 2v + 3w$ *(c)* $(2u + v).w$

(d) $\|w\|$ *(e)* $\|u - v\|$ *(f)* $u - v + w$

2.3 Operaciones con matrices

Dadas dos matrices A, B definidas en Mathematica, podemos realizar las siguientes operaciones con ellas, siempre que éstas puedan hacerse:

$A + B$	suma de las matrices
$A.B$	producto de las matrices
$\lambda * A$	producto del número λ por la matriz A
transpose(A)	traspuesta de A
determinant(A)	determinante de A
invert(A)	inversa de A

Si por ejemplo tenemos las matrices

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

y

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

podemos calcular su suma, producto, A^t, $|A|$ y A^{-1} de la siguiente forma:

$(\%i1)$ $A : matrix([1,2,1],[1,0,1],[0,1,1])\$;$
$(\%i2)$ $B : matrix([2,1,-1],[0,-1,0],[1,2,3])\$;$
$(\%i3)$ $A + B;$
$(\%o3)$ $\begin{bmatrix} 3 & 3 & 0 \\ 1 & -1 & 1 \\ 1 & 3 & 4 \end{bmatrix}$
$(\%i4)$ $transpose(A)$
$(\%o4)$ $\begin{bmatrix} 1 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
$(\%i5)$ $determinant(A)$
$(\%o5)$ -2
$(\%i6)$ $invert(A)$
$(\%o6)$ $\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -1 \\ \frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}$

Actividad 5 *Dados las matrices* $A = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 3 & -1 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$, $C = \begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}$ *y* $D = \begin{pmatrix} 0 & -1 & 0 \\ 2 & 0 & 5 \\ 2 & 1 & 0 \end{pmatrix}$ *calcular:*

(a) CAB *(b)* $A(B+C)$ *(c)* $(B^t + C)$

(d) CAB^tD^{-1} *(e)* AD^tB^{-1} *(f)* $B^{-1}D$

Actividad 6 *Calcular el determinante de las siguientes matrices cuadradas*

$$(a)\begin{vmatrix} 3-i & 5 & 7 & 2 \\ 2-6i & 4 & 1 & 1 \\ -2 & 0 & 0 & 0 \\ 1 & 1-i & 3 & 4 \end{vmatrix} \quad (b)\begin{vmatrix} 1 & \cos x & \cos 2x \\ \cos x & \cos 2x & \cos 3x \\ \cos 2x & \cos 3x & \cos 4x \end{vmatrix} \quad (c)\begin{vmatrix} x & a & b & c \\ a & x & 0 & 0 \\ b & 0 & x & 0 \\ c & 0 & 0 & x \end{vmatrix}$$

Actividad 7 *Calcular el rango de las siguientes matrices*

$$(a)\begin{pmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 8 & 9 & 0 \\ 1 & 2 & 3 \end{pmatrix} \quad (b)\begin{pmatrix} 2 & 2 & 3 \\ 3 & 0 & 1 \\ 10 & 4 & 8 \end{pmatrix} \quad (c)\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 0 & 3 & 4 \\ 2 & 3 & 0 & 4 \\ 2 & 3 & 4 & 0 \end{pmatrix}$$

2.4 Diagonalización de matrices cuadradas

Dada una matriz cuadrada $A \in M_{n\times n}(\mathbb{R})$ es posible a veces encontrar una matriz diagonal $D \in M_{n\times n}(\mathbb{R})$ de manera que existe una matriz invertible $P \in M_{n\times n}(\mathbb{R})$ de forma que

$$D = PAP^{-1}.$$

Los elementos de la matriz diagonal D se llaman vectores propios de A, y las matrices P y P^{-1} son matrices de cambio de base que nos pasan de la matriz A a la D. La ventaja de tener matrices diagonales es que éstas son muy fáciles de manejar. Hemos de remarcar que no siempre existe la forma diagonal de una matriz.

El polinomio caracteristico de A se puede obtener con la sentencia

$$charpoly(A, x)$$

donce x indica la variable en la que se construirá.

Maxima dispone de sentencias que permiten calcular rápidamente la forma diagonal de la matriz, en caso de que ésta exista, proporcionando además una base de vectores propios. También dispone de una sentencia para calcular la forma diagonal, y en caso de que ésta no exista, una forma canónica lo más parecido posible a la forma diagonal. Consideremos una matriz cuadrada arbitraria A. Las sentencias que vamos a usar son las siguientes:

- eigenvalues(A). Con esta sentencia se calculan los valores propios de la matriz A. El resultado ofrece dos listas, la primera con los valores propios y la segunda con sus respectivas multiplicidades.

- eigenvectors(A). Calcula los vectores propios de la matriz A. El resultado devuelto es una lista con dos elementos; el primero está formado por dos listas, la primera con los valores propios de A y la segunda con sus respectivas multiplicidades, el segundo elemento es una lista de listas de vectores propios, una por cada valor propio, pudiendo haber uno o más vectores propios en cada lista.

Por ejemplo, si consideramos la matriz $A = \begin{pmatrix} 3 & -2 & -1 \\ -1 & 4 & 1 \\ 1 & 2 & 5 \end{pmatrix}$ podemos calcular

$$(\%i1)\ A : matrix([3,-1,-1],[-1,4,1],[1,2,5])\$;$$
$$(\%i2)\ eigenvalues(A);$$
$$(\%o2)[[4,2,6],[1,1,1]]$$
$$(\%i3)\ eigenvectors(A);$$
$$(\%o3)\ [[[4,2,6],[1,1,1]],[[[1,-1,1]],[[1,1,-1]],[[1,-1,-1]]]]$$

Actividad 8 *Calcular los vectores y valores propios, y en caso de existir, la forma diagonal de las matrices B y D del ejercicio 5.*

2.5 Resolución de sistemas de ecuaciones lineales

Si tenemos que resolver un sistema de ecuaciones lineales de la forma $AX = b$ donde A es una matriz de n filas por m columnas, X es una vector columna incógnita de n componentes y b es un vector de m componentes, disponemos de la sentencia

$$linsolve([expr\_1,...,expr\_m],[x\_1,...,x\_n]),$$

donde expr_1,...,expr_m son las m ecuaciones lineales que definen el sistema y x_1,...,x_n son las incógnitas del mismo. Para introducir las ecuaciones

utilizamos el signo de igualdad =, como es usual. Por ejemplo, si queremos resolver el sistema

$$\left.\begin{array}{l} x+y+z=1 \\ x+2y=1 \end{array}\right\}$$

escribiremos

$$(\%i1)\ lin\,\text{solve}([x+y+z=1, x+2*y=1],[x,y,z]);$$
$$(\%o1)[x=1-2*\%r1, y=\%r1, z=\%r1]$$

con lo que la solución del sistema depende del parámetro $r1$ y presenta la forma paramétrica

$$\left\{\begin{array}{ll} x=2t, & \\ y=t & \text{donde } t\in\mathbb{R}. \\ z=t, & \end{array}\right.$$

Dentro de la resolución de ecuaciones existen una serie de alternativas que a continuación pasamos a describir. Las alternativas vienen dados por funciones que tienen asignados el valor true si están activadas y false si no lo están. Según esten activadas o no dichas funciones el programa efectúa las operaciones y presenta resultados de una u otra forma. Para cambiar la asignación true o false basta con teclear

$$(i\%1)\ nombre\_funcion : true;$$
$$(o\%1)\ true$$

que asigna a la funcion el valor true y el correspondiente

$$(i\%1)\ nombre\_funcion : false;$$
$$(o\%1)\ false$$

para false. Para ver qué valor tiene una función en un determinado momento basta con ejecutarla. Veamos los parámetros o funciones más relevantes.

- **linsolve_params** Valor por defecto: true. Si linsolve_params es true, la función linsolve genera símbolos %r para representar parámetros arbitrarios para representar la solución de forma paramétrica. Si vale false, el resultado devuelto para un sistema es indeterminado elimina

las ecuaciones dependientes y se expresa con la forma general. Por ejemplo

$$(\%i1)\ lin\,\text{solve}\ \_params : false;$$
$$(o\%1)\ false$$
$$(\%i2)\ lin\,\text{solve}([x+y+z=1, x+2*y=1],[x,y,z]);$$
$$(\%o2)[x=1-2*z, y=z]$$

- **globalsolve** Valor por defecto: false. Si se activa a true, al resolver el sistema asigna a las incógnitas el valor de las soluciones, de igual forma que se introducen las constantes. Por ejemplo

$$(\%i1)\ global\,\text{solve} : true;$$
$$(o\%1)\ true$$
$$(\%i2)\ lin\,\text{solve}([x+y+z=1, x+2*y=1],[x,y,z]);$$
$$(\%o2)[x : 1-2*\%r1, y : \%r1, z : \%r1]$$

- **programmode** Valor por defecto: true. Si cambiamos a false, linsolve muestra la solución con etiquetas de expresiones intermedias (%t). Por ejemplo

$$(\%i1)\ programmode : false;$$
$$(o\%1)\ false$$
$$(\%i2)\ lin\,\text{solve}([x+y+z=1, x+2*y=1],[x,y,z]);$$
$$(\%t2)\ x=1-2*\%r1$$
$$(\%t3)\ y=\%r1$$
$$(\%t4)\ z=\%r1$$
$$(\%o2)[\%t2,\%t3,\%t4]$$

Finalemente, se admiten diferentes combinaciones de las funciones anteriores, es decir, algunas de ellas con valor false y otras true, lo que da lugar a diferentes maneras de representar las soluciones y ejecutar la funcion linsolve.

Actividad 9 *Resolver los siguientes sistemas de ecuaciones lineales:*

$$(a)\ \begin{cases} x-y=-1 \\ -x+y=1 \\ 2x-2y=-2 \end{cases} \qquad (b)\ \begin{cases} 2x+y+4z=0 \\ x-y+2z=4 \\ 2x+y-z=14 \\ 3x+z=18 \end{cases}$$

$$(c)\ \begin{cases} 2x+2y-3z=2 \\ -x+5y-4z=4 \\ x+7y-7z=7 \end{cases} \qquad (d)\ \begin{cases} x+2y+3z=0 \\ 2x+2y+3z=0 \\ 3x+2y+z=0 \end{cases}$$

$$(e)\ \begin{cases} x+2y-3z+16t=4 \\ y+2z-3t=6 \\ -x-y+z+9t=-2 \end{cases} \qquad (f)\ \begin{cases} x-2y+3z=0 \\ 2x+5y+6z=0 \end{cases}$$

2.6 Optimización lineal

Maxima permite resolver problemas de optimización lineal mediante el algoritmo del simplex. Para ello, hay que cargar un paquete especial, **simplex**, desarrollado para este menester, ejecutando la sentencia (añadimos $ al final para que no proporcione salida)

$$(i\%1)\ load(\text{"}simplex\text{"})\$$$

Una vez caragdo, disponemos de los siguientes comandos:

- **linear_program.** Este comando tiene la sintaxis

  $$linear\_program(\mathbf{A}, \mathbf{b}, \mathbf{c})$$

  donde $\mathbf{A}$ es una matriz $n \times m$, y $\mathbf{b}$ y $\mathbf{c}$ son vectores de dimensiones n y m, respectivamente. Dicha sentencia minimiza la función objetivo

  $$\mathbf{c} \cdot \mathbf{x}$$

  bajo las restricciones

  $$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

  y si $\mathbf{x} = (x_1, ..., x_m)$, entonces $x_i \geq 0$, $i = 1, ..., m$. Por ejemplo, si tecleamos

  $$(i\%2)\ A : matrix([1,1,-1,0],[2,-3,0,-1],[4,-5,0,0])\$$$
  $$(i\%3)\ b : [1,1,6]\$$$
  $$(i\%4)\ c : [1,-2,0,0]\$$$
  $$(\%i5)\ linear\_program(A,b,c);$$
  $$(\%o5)\ [[\frac{13}{2}, 4, \frac{19}{2}, 0], \frac{3}{2}]$$

que nos minimiza la función

$$f(x_1, x_2, x_3, x_4) = x_1 - 2x_2$$

con las restricciones anteriormente comentadas. El mínimo, con valor $\frac{3}{2}$, se alcanzará en el punto $(\frac{13}{2}, 4, \frac{19}{2}, 0)$.

- **minimize_lp**. Esta función permite encontrar los mínimos de la función objetivo (obj), bajo unas condiciones o restricciones lineales (cond), con argumentos opcionales (pos), según la siguiente sintaxis

$$\min imize\_lp(obj, cond, [pos]).$$

Dará las salidas "Problem not bounded!" si el problema es no acotado y "Problem not feasible!" si es no factible. Al contrario que en la sentencia anterior, no se suponen que todos los valores son no negativos. Si queremos que así sea, utilizaremos la sentencia opcional **nonegative_lp** asignándole el valor true.

Por ejemplo

$(\%i6)\ minimize\_lp(x+y, [3*x+y=0, x+2*y>2]);$
$(\%o6)\ [4/5, [y=6/5, x=-2/5]]$

$(\%i7)\ nonegative\_lp : true;$
$(\%o7)\ true$
$(\%i8)\ minimize\_lp(x+y, [3*x+y>0, x+2*y>2])$
$(\%o8)\ [1, [y=1, x=0]]$

$(\%i9)\ minimize\_lp(x+y, [3*x+y=0, x+2*y>2]);$
$(\%o9)$ "Problem not feasible!"

$(\%i10)\ nonegative\_lp : false;$
$(\%o11)\ false$
$(\%i12)\ minimize\_lp(x+y, [3*x+y>0])$
$(\%o12)$ "Problem not bounded!"

- **maximize_lp**. Es análoga a la función anterior para obtener máximos.

Actividad 10 *Encontrar la solución de los siguientes problemas:*

$$(a)\begin{cases} \min imizar & x+y+z \\ suejto\ a & x+2*y>2 \\ & x,y,z\geq 0 \end{cases} \qquad (b)\begin{cases} \max imizar & x+2y+z \\ suejto\ a & x+y-z>2 \end{cases}$$

$$(c)\begin{cases} \min imizar & 2x+y \\ suejto\ a & x+2*y=2 \\ & x-y\geq 0 \end{cases} \qquad (d)\begin{cases} \max imizar & x+2y+z \\ suejto\ a & x+y-z>2 \\ & x,y,z\leq 0 \end{cases}$$

$$(e)\begin{cases} \min imizar & x+y+z \\ suejto\ a & x+2*y=2 \\ & x-y-z=0 \end{cases} \qquad (f)\begin{cases} optimizar & x+2y-z \\ suejto\ a & x-z>2 \\ & x\leq 0 \end{cases}$$

Chapter 3

Cálculo en una y varias variables

Maxima dispone de herramientas para hacer de forma rápida sumas de series numéricas, derivadas de funciones de una y varias variables, cáculo de primitivas de funciones de una variable, desarrollos de Taylor de funciones de una variable, desarrollos en series de Fourier de funciones de una variable, resolución númerica de ecuaciones, representaciones gráficas de funciones de una y dos variables, así como curvas en el espacio etc...Vamos a ver como hacer todas estas cosas con Maxima.

3.1 Cálculo de límites

Maxima tiene también una sentencia, **limit**, que permite el cálculo de límites funcionales. Suele tener la sintaxis

$$\lim it(\exp r, x, val)$$

Por ejemplo, si queremos calcular $\lim_{x\to 0} \frac{\sin x}{x}$, debemos escribir.

$$(\%i1)\ \text{limit}(\sin(x)/x, x, 0);$$
$$(\%i1)\ 1$$

Se pueden calcular límites en $-\infty$ (minf) y en $+\infty$ (inf). El valor de salida puede ser inderminado, distinguiéndose si es acotado (ind) o no acotado

(und). Por ejemplo $\lim_{\times\to+\infty} x\sin x$ es indeterminado y no acotado, por lo que al teclear

$$(\%i1)\ \text{limit}(x*\sin(x), x, \text{inf});$$
$$(\%i1)\ \textit{und}$$

Actividad 11 *Calcular los siguientes límites:*

(a) $\lim_{x\to 0}\frac{1-\cos 2x}{\sin x^2} =$

(b) $\lim_{x\to\infty}\frac{\log x}{x} =$

(c) $\lim_{x\to 1}\frac{1-x^2}{1-x} =$

(d) $\lim_{x\to 0}\frac{1}{x} =$

3.2 Cálculo de suma y productos

Supongamos que queremos sumar o multiplicar números que vienen dados por una cierta función, como por ejemplo calcular la suma $\sum_{i=1}^{10} i^2$ o el producto $\prod_{i=1}^{10} i^2$. Maxima dispone de comandos que realizan estas tareas con poco coste de tiempo. Estos comandos son **sum** para la suma y **product** para el producto, y el modo de empleo es el siguiente. Para sumar $\sum_{i=1}^{10} i^2$ escribimos

$$(\%i1)\ \text{sum}(i\hat{}2, i, 1, 10);$$
$$(\%o1)\ 385$$

y para multiplicar $\prod_{i=1}^{10} i^2$ debemos escribir

$$(\%i2)\ \text{product}(i\hat{}2, i, 1, 10);$$
$$(\%o2)\ 13168189440000$$

También se pueden hacer las sumas n–ésimas, así como la suma de series numéricas, es decir, expresiones como $\sum_{i=1}^{n} i^2$ y $\sum_{i=1}^{\infty} i^2$. Para trabajar con sumas tenemos la opción **simpsum**, que vale false como valor por defecto, pero una vez que hemops cambiado su valor a true permite simplificar sumas y calcular los valores de algunas series numéricas. Por ejemplo, si tecleamos

$$(\%i3)\ \text{sum}(i\hat{}2, i, 1, n);$$
$$(\%o3)\ \sum_{i=1}^{n} i^2$$

vemos que no obtenemos ninguna simplificación. Si activamos previamente tecleando debemos proceder a hacer modificaciones en las expresiones anteriores del siguiente modo:

$$(\%i4)\ simpsum : true;$$
$$(\%o4)\ true$$
$$(\%i5)\ \mathrm{sum}(i\hat{}2, i, 1, n);$$
$$(\%o5)\ \frac{2n^3 + 3n^2 + n}{6}$$

que como vemos si proporciona una simplificación. De igual forma, con simpsum activado como true, si tecleamos

$$(\%i6)\ \mathrm{sum}(1/i\hat{}2, i, 1, \mathrm{inf});$$
$$(\%o6)\ \frac{\pi^2}{6}$$

Asímismo, es posible escribir expresiones algebraicas como los polinomios $\sum_{i=1}^{4} x^i$ y $\prod_{i=1}^{4} (x+i)$ con facilidad. Basta teclear:

$$(\%i7)\ \mathrm{sum}(x\hat{}i, i, 1, 4);$$
$$(\%o7)\ 1 + x^2 + x^3 + x^4$$
$$(\%i8)\ \mathrm{product}(x + i, i, 1, 4);$$
$$(\%o8)\ (1+x)(2+x)(3+x)(4+x)$$

A la hora de trabajar con polinomios y cocientes de éstos suelen ser útiles las sentencias **expand**, que desarrolla los mismos, y **factor**, que los agrupa en factores comunes. Por ejemplo, si queremos poner el polinomio de la salida 8 en la forma extendida tecleamos

$$(\%i9)\ \exp and(\%o8);$$
$$(\%o9)\ x^4 + 10x^3 + 35x^2 + 50x + 24$$

mientras que para volver a escribirlo como producto de sus factores irreducibles tecleamos

$$(\%i10)\ \mathrm{factor}(\%o9);$$
$$(\%o10)\ (1+x)(2+x)(3+x)(4+x)$$

Actividad 12 *Calcular las siguientes sumas y productos con Mathematica.*

(a) $\sum_{i=1}^{34} \frac{1}{i^2}$.

(b) $\prod_{i=1}^{10} \left(\frac{i+i^2}{i+1}\right)$.

(c) $\sum_{i=1}^{n} i^3$.

(d) $\sum_{i=0}^{25} \frac{(x+i)^i}{i+1}$.

(e) $\prod_{i=1}^{10} (a+i)^i$.

Actividad 13 *Calcular la suma de las siguientes series numéricas.*

(a) $\sum_{n=1}^{\infty} \frac{1}{n^2}$.

(b) $\sum_{n=1}^{\infty} \frac{3^n}{n!}$.

(c) $\sum_{n=1}^{\infty} \frac{1}{n(n-1)}$.

(d) $\sum_{n=1}^{\infty} \frac{\log n}{n^2}$.

(e) $\sum_{n=1}^{\infty} \frac{(-1)^n}{n}$.

3.3 Derivadas

Supongamos que tenemos una función de una variable $f(x)$ o de varias variables $f(x_1, x_2, ..., x_n)$ a la que queremos calcular su derivada o derivada parcial respecto de alguna de sus variables. El comando que realiza ese cáculo con Maxima es **diff**, que realiza diversos cálculos. Por ejemplo si queremos calcular la derivada de $f(x) = \sin x$ escribiremos

$$(\%i1)\ diff(\sin(x), x);$$
$$(\%o1)\ \cos(x)$$

especificando tanto la función como la variable respecto de la cual vamos a derivar. Así para calcular la derivada parcial con respecto a la variable y de la función $f(x, y) = \sin(x + y)$ debemos escribir

$$(\%i2)\ diff(\sin(x + y), y);$$
$$(\%o2)\ \cos(x + y)$$

Para calcular la derivada n–ésima de $f(x)$, hemos de proceder con el comando $diff(f, x, n)$. Así la segunda derivada de $f(x) = \sin x$ se calcula

$$(\%i3)\ diff(\sin(x), x, 2);$$
$$(\%o3)\ -\sin(x)$$

y $\frac{\partial^3 f}{\partial y^3}$ de la función $f(x, y) = \sin(x + y)$ sería

$$(\%i4)\ diff(\sin(x + y), y, 3);$$
$$(\%o4)\ -\cos(x + y)$$

Si ahora queremos calcular derivadas parciales cambiando la variable con la que derivamos. Por ejemplo calcular $\frac{\partial^2 f}{\partial x \partial y}$ debemos escribir

$$(\%i5)\ diff(\sin(x + y), x, 1, y, 1);$$
$$(\%o5)\ -\sin[x + y]$$

indicando que derivamos una vez respecto de x y otra respecto de y. Por ejemplo $\frac{\partial^4 f}{\partial x^3 \partial y}$ se calcula tecleando

$$(\%i6)\ diff(\sin(x + y), x, 3, y, 1);$$
$$(\%o6)\ \sin[x + y]$$

Actividad 14 *Calcula las derivadas de las siguientes funciones:*

(a) $f(x) = \log(\sin x)$.

(b) $f(x) = \frac{\log(\cos x)\arcsin x}{e^x \log_4(x^2+10)}$.

(c) $f(x) = 1 + \left(\frac{3x+e^x}{x^2+\tan\sqrt{x}}\right)$.

Actividad 15 *Calcula la derivada cuarta y sexta de cada una de las funciones del ejercicio 14.*

Actividad 16 *Calcula* $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial^2 f}{\partial x \partial y}$, $\frac{\partial^2 f}{\partial x^2}$, $\frac{\partial^3 f}{\partial x \partial y \partial x}$ *y* $\frac{\partial^4 f}{\partial x^4}$ *de cada una de las siguientes funciones:*

(a) $f(x, y) = \cos(\sin x - y)$.

(b) $f(x, y) = \arcsin\left(\frac{1}{x^2+y^2}\right)$.

(c) $f(x, y) = \frac{1+\log(x+e^y)}{\sqrt{x^2-\tan xy}}$.

Actividad 17 *Calcular la diferencial de las funciones del ejercicio 16.*

3.4 Cálculo de primitivas e integral definida

Maxima también posee sentencias para calcular primitivas de funciones de una variable. El comando que se utiliza para calcular la primitiva de una función $f(x)$ es $integrate(f, x)$, indicando la variable respecto la cual se integra. Por ejemplo, para calcular una primitiva de $f(x) = \sin x$ procedemos del siguiente modo.

$$(\%i1)\ \text{integrate}(\sin(x), x);$$
$$(\%o1)\ -\cos(x)$$

Si lo que pretendemos es calcular la integral definida $\int_a^b f(x)\,dx$, el comando que debemos usar es $integrate(f, x, a, b)$. Entonces $\int_0^1 xdx$ se calcularía del modo siguiente.

$$(\%i2)\ \text{integrate}(x, x, 0, 1);$$
$$(\%o2)\ \frac{1}{2}$$

Actividad 18 *Calcular las primitivas de las siguientes funciones:*

(a) $f(x) = \cos x \sin 2x \cos 3x \tan 4x$

(b) $f(x) = \frac{x+x^2}{1+x^{10}}$

(c) $f(x) = \cos xe^{10x}$

(d) $f(x) = shxch2xsh3x.$

Actividad 19 *Calcular para cada una de las funciones del ejercicio 18 la integral* $\int_0^1 f(x)\,dx$ *dando el resultado exacto y con cifras decimales.*

3.5 Polinomio de Taylor

Maxima incorpora sentencias para obtener polinomios de Taylor de funciones reales. Si tenemos una función f de la cual queremos obtener su desarrollo de Taylor de grado n en el punto x_0 mediante la función **taylor** escribiendo

$$\text{taylor}(f(x), x, x_0, n),$$

indicando la función y la variable respecto de la cual hacemos el desarrollo. Así, para calcular el desarrollo de Taylor de grado 5 en el punto 0 de la función e^x escribimos

$$(\%i1)\ \text{taylor}(\exp(x), x, 0, 5);$$
$$(\%o1)\ /T/\ 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + ...$$

Es posible escribir también los polinomios de Taylor de funciones reales de varias variables. Para ello hemos de indicarle las variables, los puntos y los grados del polinomio en cada variable. Por ejemplo, si queremos obtener el polinomio de Taylor de la función $f(x, y) = e^{x+y}$ centrado en $(0, 0)$ y de grado 1 en x y tres en y tecleamos

$$(\%i1)\ \text{taylor}(\exp(x + y), [x, 0, 1], [y, 0, 3]);$$
$$(\%o1)\ /T/\ 1 + y + \frac{y^2}{2} + \frac{y^3}{6} + ... + \left(1 + y + \frac{y^2}{2} + \frac{y^3}{6} + ...\right) x + ...$$

Actividad 20 *Calcular el polinomio de Taylor de grado 8 en los punto 0 y 1 de las siguientes funciones*

(a) $f(x) = \log(1 + x)$ *(b)* $f(x) = e^{x^2+1}$ *(c)* $f(x) = \sin(\cos x)$

(d) $f(x) = e^x \log(1 + x^2)$ *(e)* $f(x) = \frac{2x}{1+x^2}$ *(f)* $f(x) = \sin(e^x)$

3.6 Representación gráfica de funciones, curvas y superficies

Maxima permite hacer representaciones gráficas de funciones de una y varias variables. Para ello hemos de darle tanto la función, como el dominio de definición de ésta.

Para representar funciones reales de variable real, tenemos el comando **plot2d**, al que habrá que introducirle la función que deseamos representar así como la variable independiente y el dominio de ésta de forma [var,liminf,limsup]. Así, para representar la función $f(x) = \sin x$ en el dominio $[0, 2\pi]$ escribimos

$$(\%i1)\ \text{plot2d}(\sin(x), [x, 0, 2 * \%pi]);$$
$$(\%o1)$$

Al ejecutarlo nos saldrá una nueva ventana con la gráfica en cuestión, pues se utiliza un programa externo a maxima llamado Gnuplot. Esta gráfica podemos guardarla en extensión "emf". Si queremos representar varias funciones a la vez, hemos de escribir

$$(\%i2)\ \text{plot2d}([\sin(x), \sin(2*x)], [x, 0, 2*\%pi]);$$
$$(\%o2)$$

expresión que hará una representación gráfica simultánea de las funciones $\sin x$ y $\sin 2x$. Si queremos indicar cual será el rango de representación de las funciones, teclearemos

$$(\%i3)\ \text{plot2d}([\sin(x), \sin(2*x)], [x, 0, 2*\%pi], [y, 0, 1]);$$
$$(\%o3)$$

que nos dará la representación de las funciones entre 0 y 1, no representando nada cuándo la imagen se halle fuera de dicho intervalo.

La representación gráfica de funciones de dos variables se hace mediante el comando **plot3d**. Por ejemplo, si queremos representar la función $f(x, y) = \sin(xy)$ en el dominio $[0, 3] \times [0, 3]$ hemos de escribir

$$(\%i4)\ \text{plot3d}(\sin(x*y), [x, 0, 3], [y, 0, 3]);$$
$$(\%o4)$$

pero si queremos representar varias funciones a la vez, debemos escribir (nótese la diferencia con la sentencia plot2d anterior)

$$(\%i5)\ \text{plot3d}([\sin(x*y), x-y, [x, 0, 3], [y, 0, 3]]);$$
$$(\%o5)$$

que realizará la representación gráfica conjunta de las funciones $\sin(xy)$ y $x - y$ en el dominio $[0, 3]^2$. Al igual que en el caso bidimensional, podemos elegir el rango en el que queremos hacer la representación gráfica.

Para la representación de curvas de nivel de funciones reales del plano tenemos la sentencia **contour_plot**, cuya sintaxis es idéntica a plot3d. Si tecleamos

$$(\%i6)\ \textit{contour\_plot}(\sin(x*y), [x, 0, 3], [y, 0, 3]);$$
$$(\%o6)$$

obtenemos las curvas de nivel de la función $\sin(xy)$.

El comando empleado para representar curvas parametrizadas en el plano es plot2d, añadiendole la opción **parametric**, e indicándole el número de puntos que vamos a usar en la representación que introducimos con la expresion **nticks**. Por ejemplo, para representar la curva $\begin{cases} x(t) = \sin t \\ y(t) = \sin 2t \end{cases}$ en el dominio $[0, 2\pi]$ debemos teclear

$$(\%i7)\ \text{plot2d}([parametric, \sin(t), \sin(2*t), [t, 0, 2*\%pi], [nticks, 1000]])\$$$

que nos dará la representación deseada con una malla de 1000 puntos. Al aumentar el número de puntos mejoramos la representación pero aumentamos el tamaño del fichero y el tiempo de computación.

Existen numerosas alternativas y opciones para la representación gráfica según sean nuestras necesidades. Estas pueden consultarse en el menú de ayuda del programa.

Actividad 21 *Representar gráficamente las siguientes funciones de una variable:*

(a) $f(x) = \frac{1+x}{1-x^2}$ *en el dominio* $[-2, 2]$.

(b) $f(x) = e^{x^2}\frac{1+x}{1-x^2}$ *en el dominio* $[-2, 2]$.

(c) $f(x) = \sin\left(\frac{1+x}{1-x^2}\right)$ *en el dominio* $[-2, 2]$.

(d) $f(x) = e^{x\cos x}$ *en el dominio* $[-5, 5]$.

(e) $f(x) = \frac{e^x}{\cos x}$ *en el dominio* $[-\pi, \pi]$.

Actividad 22 *Representar gráficamente las siguientes funciones de varias variables:*

(a) $f(x, y) = \log(x^2 + y^2)$ *en el dominio* $[-2, 2] \times [-2, 2]$.

(b) $f(x, y) = x^2 + y^2$ *en el dominio* $[-2, 2] \times [-2, 2]$.

(c) $f(x, y) = e^{\frac{1}{x^2+y^2}}$ *en el dominio* $[-2, 2] \times [-2, 2]$.

(d) $f(x, y) = (x^2 + y^2)\sin\frac{1}{x^2+y^2}$ *en el dominio* $[-1, 1] \times [-1, 1]$.

Actividad 23 *Representar gráficamente las siguientes curvas en el plano y el espacio:*

(a) $\begin{cases} x(t) = \cos 2t \\ y(t) = \sin t \end{cases}$ *en el dominio* $[0, 2\pi]$.

(b) $\begin{cases} x(t) = \frac{1}{t} \\ y(t) = t^2 \end{cases}$ *en el dominio* $[-1, 1]$.

(c) $\begin{cases} x(t) = t^3 \\ y(t) = \sqrt{t} \end{cases}$ *en el dominio* $[0, 2]$.

(d) $\begin{cases} x(t) = \cos t^2 \\ y(t) = \sin t^2 \\ z(t) = \sqrt{t} \end{cases}$ *en el dominio* $[0, 2\pi]$.

(e) $\begin{cases} x(t) = t^2 \\ y(t) = t \end{cases}$ *en el dominio* $[0, 2]$.

3.7 Resolución numérica de ecuaciones

Supongamos que queremos resolver la ecuación polinómica $x^2 + 2x - 4 = 0$. El comando para resolverlas en Maxima es **solve**, de forma que para resolver la ecuación anterior escribiremos

$$(\%i1)\ \text{solve}(x\hat{}2 + 2 * x - 4 = 0, x);$$
$$(\%o1)\ [x = -\sqrt{5} - 1, x = \sqrt{5} - 1]$$

Este comando ya se introdujo al estudiar la resolución de sistemas de ecuaciones lineales y puede usarse la función **programmode** explicada anteriormente.

Si lo que buscamos son soluciones numéricas aproximadas, tanto reales como complejas, tenemos la sentencia **allroots**, de forma que al teclear

$$(\%i2)\ \text{allroots}(x\hat{}2 + 2 * x - 4 = 0, x);$$
$$(\%o2)\ [x = 1.23606797749979, x = -3.23606797749979]$$

obtenemos las soluciones aproximadas.

La sentencia solve puede usarse para resolver sistemas de ecuaciones polinómicas, introduciéndolas en una lista junto con otra lista para las incógnitas. Por ejemplo, el sistema

$$\begin{cases} x+y=1 \\ xy+y\hat{}2=5 \end{cases}$$

se resolvería de forma exacta tecleando

$$(\%i3)\ solve([x+y=1, x*y+y\hat{}2=5],[x,y]);$$
$$(\%o3)\ [[x=-4, y=5]]$$

Alternativamente, puede usarse la sentencia **algsys**, con sintaxis

algys([eq_1,..., eq_n],[x_1,...,x_n])

resuelve las ecuaciones polinómicas eq_1,..., eq_n. Por ejemplo, tecleando

$$(\%i1)\ eq1 : x\hat{}2-y\hat{}2=0;$$

$$(\%i2)\ eq1 : -1-y+2*y\hat{}2-x+x\hat{}2=0\$$$

(%i3) algsys([eq1,eq2],[x,y])

produce la salida

$$(\%o4)[[x=-\frac{1}{\sqrt{3}}, y=\frac{1}{\sqrt{3}}], [x=\frac{1}{\sqrt{3}}, y=-\frac{1}{\sqrt{3}}], [x=-\frac{1}{3}, y=-\frac{1}{3}], [x=1, y=1]]$$

Si es una ecuación no polinómica, Maxima utiliza métodos iterativos de resolución numérica de ecuaciones como el algoritmo de Newton. La función que permite obtener las soluciones es **find_root**, a la que habrá que introducir la ecuación, incógnita e intervalo donde queremos obtener la solución. Por ejemplo, para resolver la ecuación $e^{-x}=x$ con Maxima debemos teclear

$$(\%i4)\ find\_root(exp(-x)=x, x, 0, 1);$$
$$(\%o4)\ 0.56714329040978$$

donde 0 y 1 indican que buscamos una solución en el intervalo $(0,1)$. Si buscáramos la solución en otro intervalo disjunto con $(0,1)$ veríamos como Maxima no es capaz de dar ninguna solución ya que esta no existe.

Además, podemos cargar el paquete **newton1** que incorpora el método de Newton para resolver ecuaciones como la anterior con error prefijado por nosotros. Para ello hemos de teclear

$$(\%i5)\ load(newton1);$$
$$(\%o5)\ "C:/PROGRA\tilde{}2/MAXIMA\tilde{}1.0-2/share/maxima/5.28.0-2/share/numeric/newton1.mac"$$

Una vez cargado el paquete newton1, podemos utilizar la sentencia **newton** con la sintaxis

$$newton(\exp r, var, pi, err),$$

que proporcionará la solución aproximada de la ecuación $\exp r = 0$, con eroor err prefijado respecto de la variable var, con condición inicial pi. Por ejemplo, la ecuación anterior la podemos resolver tecleando

$$(\%i6)\ newton(exp(-x)-x, x, 0.5, 0.00001);$$
$$(\%o6)\ 0.56714316503486$$

con error 0.0001.

Si tenemos sistemas de ecuaciones, podemos usar el paquete **mnewton**, que permite trabajar con sistemas al igual que hicimos con ecuaciones. Para usarlos cargamos el paquete

$$(\%i7)\ load("mnewton")\$$$

La sintaxis es de la forma

$$mnewton([\exp r_1, ...\exp r_n], [x_1, ..., x_n], [pi_1, ...pi_n]),$$

y resolverá el sistema $\exp r_1 = 0, ...\exp r_n = 0$. Por ejemplo

$$(\%i8)\ mnewton([x+3*log(x)-y\hat{}2, 2*x\hat{}2-x*y-5*x+1], [x,y], [5,5]);$$

$$(\%o8)\ [[x = 3.756834008012769, y = 2.779849592817897]]$$

Obviamente, lo complicado a la hora de usar estas sentencias es obtener las condiciones iniciales desde las que partir. Puede ser útil hacer representaciones gráficas a fin de ver si las soluciones existen y obtener un entorno de las mismas.

Actividad 24 *Calcular las raíces de los siguientes polinomios, dando el resultado exacto si lo hubiere, y un resultado aproximado.*

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

(b) $x^{10} - 1$.

(c) $5x^4 - x^2 + x - 1$.

(d) $x^5 - x^3 + 1$.

Actividad 25 *Resolver las siguientes ecuaciones numéricamente:*

(a) $\cos x = \log x$.

(b) $\log x = x$.

(c) $e^{-x} = x^2$.

(d) $\sin x^2 = x^3$.

Actividad 26 *Resolver los sitemas de ecuaciones siguientes:*

(a) $x^2 + y^2 = 1;\ x + y = 0.5$.

(b) $x + 3y - z = 1;\ x - y = 0;\ 2x + z = 7$.

(c) $x + 3y = 0;\ 2x + 6y = 1$.

3.8 Programación en Maxima

Para programar un método numérico en Maxima tenemos que conocer la construcción

for variable:valor_inicial **step** incremento **thru** límite **do** (expr)

Vamos con un ejemplo sencillo cómo programar un bucle. Por ejemplo, vamos a sumar los 50 primeros números pares, mediante el siguiente programa

$$\begin{aligned} (\%i1)\ s &: 0\$ \\ for\ i &: 2\ step\ 2\ thru\ 100\ do \\ s &: s + i; \end{aligned}$$

$$(\%o2)\ done$$

Si ahora tecleamos

$$(\%i3)\ s$$
$$(\%o3)\ 2550$$

obtenemos el valor.

Si queremos hacer varias operaciones, por ejemplo sumar los 50 primeros pares los 50 primeros impares, escribimos las operaciones que queremos hacer entre paréntesis después del for separadas por comas. Por ejemplo

$$\begin{aligned}(\%i4)\ s1 &: 0\$\\ s2 &: 0\$\\ for\ i &: 1\ step\ 1\ thru\ 50\ do\\ (s1 &: s1+2*i,\\ s2 &: s2+2*(i-1)+1)\end{aligned}$$

$$(\%o6)\ done$$

Aquí, si el paso es uno podría suprimirse.

Una vez sabemos como construir bucles, vamos a programar el método de integración numérica del trapecio cuya fórmula es

$$\int_a^b f(x)dx = \frac{h}{2}\left(f(a)+2\sum_{i=1}^{n-1} f(a+ih)+f(b)\right)$$

donde $h = (b-a)/n$. Este sencillo programa nos aproxima la integral $\int_{-1}^{1} e^x dx$, que vamos a tomar como ejemplo.

$$\begin{aligned}
\text{numer} &: false\$ \\
f(x) &: = exp(x)\$ \\
a &: -1\$ \\
b &: 1\$ \\
n &: 100\$ \\
h &: (b-a)/n\$ \\
s &: f(a)+f(b)\$ \\
for\ i &: 1\ step\ 1\ thru\ n-1\ do \\
s &: s+2*f(a+i*h); \\
it &: s*h/2;
\end{aligned}$$

e **it** nos da el valor aproximado 2.350402395122275.

Actividad 27 *Resolver numéricamente las siguientes integrales mediante el método del trapecio*

(a) $\int_0^1 x^6+1dx$

(b) $\int_{-3}^1 \cos(2x)dx$

(c) $\int_0^3 e^{-x}dx$

(d) $\int_0^1 (x^2+e^{-x^2})dx$.

Actividad 28 *Resolver numéricamente las siguientes integrales mediante el método de Simpson 1/3*

(a) $\int_0^1 x^6+1dx$

(b) $\int_{-3}^1 \cos(2x)dx$

(c) $\int_0^3 e^{-x}dx$

(d) $\int_0^1 (x^2+e^{-x^2})dx$.

Actividad 29 *Programar el método de Newton para resolver ecuaciones algebraicas y resolver con él las siguientes ecuaciones numéricamente:*

(a) $\cos x = \log x$.

(b) $e^{-x} = x^2$.

(c) $\sin x^2 = x^3$.

3.9 Resolución de ecuaciones diferenciales

3.9.1 El método de Euler

Consideramos un problema de condiciones iniciales de la forma

$$\begin{cases} \mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases} \tag{3.1}$$

donde la función $\mathbf{f} : \Omega \subseteq \mathbb{R}^{m+1} \to \mathbb{R}^m$ es suficiente regular para que dicho problema tenga solución única. Por ejemplo, $\mathbf{f}$ y $\frac{\partial \mathbf{f}}{\partial y_i}$, $i = 1, ..., m$ continuas. Sin embargo, dada un problema de condiciones iniciales arbitrario, es muy posible que no sepamos cómo hallar dicha solución. Basta considerar el problema

$$\begin{cases} y' = e^{y^2}, \\ y(0) = 4. \end{cases}$$

Es por ello importante determinar métodos que permitan obtener aproximaciones de dichas soluciones, que existen, pero son desconocidas.

En esencia, dado el problema (3.1), denotemos su solución por $\mathbf{y}(t)$ y buscamos cómo aproximar el valor de $\mathbf{y}(t_f)$, para un cierto $t_f > t_0$ (análogamente se haría para $t_f < t_0$). Los métodos que vamos a estudiar consisten en generar una sucesión $\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_n$ de manera que $\mathbf{y}_n$ sea un valor aproximado de $\mathbf{y}(t_f)$.

El método más sencillo para hacer la aproximación de la solución viene dado por el método de Euler, que se basa en construir una suceción de la forma la forma

$$\mathbf{y}(t_1) \approx \mathbf{y}_1 = \mathbf{y}_0 + \frac{1}{1!}\mathbf{f}(t_0, \mathbf{y}_0)h, \tag{3.2}$$

y tiene un claro significado geométrico. Imaginemos que $m = 1$, es decir, se trata de una ecuación diferencial. Entonces la recta tangente de la solución $y(t)$ para $t = t_0$ tiene la forma

$$y - y(t_0) = y'(t_0)(t - t_0),$$

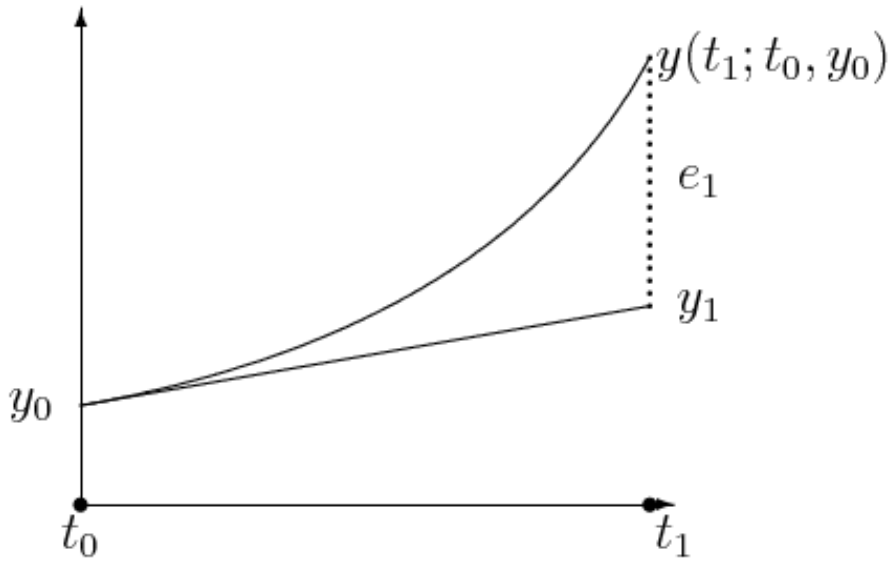

Figure 3.1: El método de Euler. El error e_1 es $|y_1 - y(t_1; t_0, y_0)|$.

y sustituyendo cada elemento de la expresión anterior por su valor obtenemos

$$y - y_0 = f(t_0, y_0)(t - t_0).$$

Si sustituimos t por t_1 en la recta anterior obtenemos

$$y(t_1) \approx y_1 = y_0 + f(t_0, y_0)h,$$

que es la expresión (3.2) para ecuaciones de dimensión uno. La figura 3.1 nos muestra gráficamente el método.

Veamos cómo funciona el método de Euler con un ejemplo. Consideremos el problema de condiciones iniciales

$$\begin{cases} y' = y, \\ y(0) = 1, \end{cases}$$

que como sabemos, tiene por solución $y(t; 0, 1) = e^t$. Tomemos $t_1 = 0.1$, y estimemos por el método de Euler $y(0.1; 0, 1)$. Como $h = 0.1$, entonces

$$y_1 = y_0 + y_0 h = 1 + 0.1 = 1.1.$$

Como vemos, el error cometido

$$e_1 = |y(0.1; 0, 1) - y_1| = |e^{0.1} - 1.1| \approx 0.00517092.$$

Si ahora, tomamos $t_1 = 1$, entonces $h = 1$ e

$$y_1 = y_0 + y_0 h = 1 + 1 = 2,$$

y el error

$$e_1 = |y(1; 0, 1) - y_1| = |e - 2| \approx 0.718282,$$

esto es, el error aumenta considerablemente.

Esto se debe a que estamos tomando aproximaciones locales. Para reducir el error se procede de la siguiente manera. Tomamos una partición $\mathcal{P}$ del intervalo $[t_0, t_f]$, esto es $\mathcal{P} = t_0 < t_1 < t_2 < ... < t_{n-1} < t_n = t_f$. Definimos $h_i = t_{i+1} - t_i$, $i = 0, 1, ..., n - 1$. Construimos la sucesión $\mathbf{y}_n$ de la siguiente manera

$$\mathbf{y}_1 = \mathbf{y}_0 + \mathbf{f}(t_0, \mathbf{y}_0)h_0.$$

Ahora bien, $\mathbf{y}_1$ es una aproximación de $\mathbf{y}(t_1; t_0, \mathbf{y}_0)$. Para construir $\mathbf{y}_2$, tomamos la aproximación mediante el método de Euler del problema

$$\begin{cases} \mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \\ \mathbf{y}(t_1) = \mathbf{y}_1, \end{cases}$$

dado por

$$\mathbf{y}_2 = \mathbf{y}_1 + \mathbf{f}(t_1, \mathbf{y}_1)h_1,$$

y de forma recurrente para $i = 1, ..., n$,

$$\mathbf{y}_i = \mathbf{y}_{i-1} + \mathbf{f}(t_{i-1}, \mathbf{y}_{i-1})h_{i-1}.$$

En general, suele tomarse $h_i = h$, $i = 0, 1, ..., n - 1$, cantidad que suele llamarse tamaño de paso y n el número de pasos. En este caso el método de Euler queda como

$$\mathbf{y}_i = \mathbf{y}_{i-1} + \mathbf{f}(t_{i-1}, \mathbf{y}_{i-1})h = \mathbf{y}_{i-1} + \mathbf{f}(t_0 + (i-1)h, \mathbf{y}_{i-1})h,$$

para $i = 1, ..., n$.

En el ejemplo anterior, tomamos $h = 0.1$ y calculamos

$$\begin{aligned}
y_1 &= y_0 + f(0, y_0)h = 1 + 1 \cdot 0.1 = 1.1, \\
y_2 &= y_1 + f(h, y_1)h = 1.1 + 1.1 \cdot 0.1 = 1.21, \\
y_3 &= y_2 + f(2h, y_2)h = 1.21 + 1.21 \cdot 0.1 = 1.331, \\
y_4 &= y_3 + f(3h, y_3)h = 1.331 + 1.331 \cdot 0.1 = 1.4641, \\
y_5 &= y_4 + f(4h, y_4)h = 1.4641 + 1.4641 \cdot 0.1 = 1.61051, \\
y_6 &= y_5 + f(5h, y_5)h = 1.61051 + 1.61051 \cdot 0.1 = 1.77156, \\
y_7 &= y_6 + f(6h, y_6)h = 1.77156 + 1.77156 \cdot 0.1 = 1.94872, \\
y_8 &= y_7 + f(7h, y_7)h = 1.94872 + 1.94872 \cdot 0.1 = 2.14359, \\
y_9 &= y_8 + f(8h, y_8)h = 2.14359 + 2.14359 \cdot 0.1 = 2.35795, \\
y_{10} &= y_9 + f(9h, y_9)h = 2.35795 + 2.35795 \cdot 0.1 = 2.59374,
\end{aligned}$$

y ahora los errores son

$$e_i = |e^{i*0.1} - y_i|,$$

para $i = 1, ..., 10$, que nos da la siguiente tabla aproximada

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
0.005	0.011	0.019	0.028	0.038	0.051	0.065	0.082	0.102	0.125

Como vemos, el error ha disminuido notablemente, a pesar de que en los pasos intermedios la aproximación del método de Euler no coincide en su condición inicial con la solución del problema de condiciones original. Vemos no obstante que los errores se van acumulando desde e_1 hasta e_{10}, de manera que estos van creciendo. Sin embargo, si disminuimos el tamaño de paso, vemos en la siguiente tabla como los errores al final dismuyen

$h = 1$	$h = 0.1$	$h = 0.01$	$h = 0.001$	$h = 0.0001$	$h = 0.00001$
0.718	0.125	0.013	0.001	0.00014	0.000014

Como vemos, al dividir el tamaño de paso h por diez, el error final aproximadamente también hace esta operación. Veremos posteriormente una explicación a este hecho.

3.9.2 El método de Euler con Maxima

El siguiente programa, bastante sencillo, nos dice cómo programar el método de Euler con Maxima

$$
\begin{aligned}
numer &: true\$ \\
f(t,x) &: = t * x\$ \\
x &: 1\$ \\
t0 &: 0\$ \\
tf &: 1\$ \\
n &: 10\$ \\
h &: (tf - t0)/n\$ \\
lista &: [[t0, x]]\$ \\
for\ i &: 1\ step\ 1\ thru\ n\ do \\
(x &: x + h * f(t0 + i * h, x), \\
lista &: append(lista, [[t0 + i * h, x]])); \\
&\quad plot2d([discrete, lista]);
\end{aligned}
$$

que nos dibujará con $plot2d([discrete, lista])$ la solución aproximada dibujada uniendo los puntos que hemos calculado e introducido en la lista (nota: buscar append y $plot2d([discrete, lista])$ en el menú de ayuda) que hemos creado con rectas.

Actividad 30 *Resolver numéricamente las siguientes ecuaciones diferenciales para el tiempo final* $t_f = 2$.

(a) $y' = y + t,\ y(1) = 4.$

(b) $y' = yt,\ y(-1) = 4.$

(c) $y' = y^2 + t,\ y(0) = 2.$

(d) $y' = y^2 + yt,\ y(1) = -1.$