



Universidad Politécnica de Cartagena
Departamento de Matemática Aplicada y Estadística

Asignatura: Matemática Aplicada
Ingeniería de Edificación

Prácticas con MÁXIMA

CURSO 2016-2017

PROFESOR: PEDRO J. MARTÍNEZ APARICIO

Índice general

Introducción a MÁXIMA y a las prácticas	1
Breve introducción a MÁXIMA y a las prácticas	1
1.1. ¿Qué es MÁXIMA ?	1
1.2. Puesta en marcha. Operaciones algebraicas	1
1.3. Funciones matemáticas de interés	4
1.4. Definición de variables y funciones	6
1.4.1. Variables	6
1.4.2. Funciones	7
1.5. Aprovechando cálculos anteriores	8
1.6. La ayuda de MÁXIMA	9
1.7. Resolución numérica de ecuaciones	9
1.8. Cálculo de límites	11
1.9. Condicionales y bucles	12
1.10. Manejo de expresiones simbólicas	13
Álgebra lineal	19
2.1. Operaciones básicas con matrices	19
Cálculo diferencial e integral	27
3.1. Introducción	27
3.2. Descripción de comandos	27
3.2.1. Derivadas de funciones	27
3.2.2. Cálculo de primitivas e integral definida	29
3.2.3. Representación gráfica de funciones, curvas y superficies .	30
3.2.4. Curvas y superficies paramétricas	33
3.3. Un ejercicio resuelto con MÁXIMA	36

Introducción a MÁXIMA y a las prácticas

1.1. ¿Qué es MÁXIMA ?

MÁXIMA es un programa que permite hacer cálculos matemáticos complicados con gran rapidez. Tiene dos modalidades de uso:

- mediante ventanas de iconos o
- por sentencias predefinidas.

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, etcétera.

Abordamos en estas prácticas una iniciación a MÁXIMA 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 estas prácticas 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 instrucciones de la misma, estas notas pueden ser útiles para aprender el manejo de MÁXIMA .

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 MÁXIMA ocurre lo mismo. Sólo conociendo teóricamente las operaciones que MÁXIMA realiza nos percataremos de su indudable utilidad.

1.2. Puesta en marcha. Operaciones algebraicas

Cuando se arranca MÁXIMA 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:

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

```
(%o1) 4
```

Una cosa que hay que tener en cuenta es que (% i1) y (% o1) los escribe el ordenador para ir numerando las órdenes que vamos poniendo.

Con MÁXIMA podemos realizar las siguiente operaciones algebraicas:

$x + y$	\rightarrow	<i>suma</i>
$x - y$	\rightarrow	<i>resta</i>
x/y	\rightarrow	<i>división</i>
$x * y$	\rightarrow	<i>producto</i>
x^y ó $x * y$	\rightarrow	<i>potencia x^y</i>
$x!$	\rightarrow	<i>factorial de x</i>

Ejemplo 1.1. (%i1) 2+2;

```
(%o1) 4
```

```
(%i73) 2**3;
```

```
(%o73)8
```

```
(%i74) 3**2;
```

```
(%o74)9
```

```
(%i75) 3*2;
```

```
(%o75)6
```

```
(%i82) 2**500;
```

```
(%o82)
327339060789614187001318969682[91digits]217256545885393053328527589376
```

Como vemos en el anterior resultado no nos ponen todos los dígitos del resultado. Nos dice que, en este caso, hay 91 dígitos que no esta mostrando. Podemos saber cuáles son los dígitos omitidos yendo al menu Maxima→Cambiar pantalla 2D y escogemos ascii. Por ultimo, repetimos la operación y obtenemos:

```
(%i84) 2**500;
```

```
(%o84)32733906078961418700131896968275991522166420460430647894832
913680961337964046745548832700923259041571508866841275600710092172
56545885393053328527589376
```

MÁXIMA es un programa de cálculo simbólico y hará las operaciones que le encomendemos de manera exacta, por ejemplo la suma de dos fracciones será una fracción y no cambiará raíces cuadradas por su valor numérico salvo que se lo pidamos usando las siguientes sentencias.

- `float(número)` : da la expresión decimal de número (por defecto usa 16 dígitos)
- `número,numer:` da la expresión decimal de número
- `bfloat(número):` da la expresión decimal larga de número. El resultado acabará con b seguido de un número n , lo que significa multiplicar por 10^n . La precisión que nos brinda el programa se puede modificar cambiando los valores de `fpprec` como ponemos de manifesto en los ejemplos.

`float`

`bfloat`

Ejemplo 1.2. `(%i1) %pi;`

```
(%o1) 
$$\pi$$

```

```
(%i2) 2^100;
```

```
(%o2) 1267650600228229401496703205376
```

```
(%i3) 2/3+5/8;
```

```
(%o3) 
$$\frac{31}{24}$$

```

```
(%i4) float(%pi);
```

```
(%o4) 3.141592653589793
```

```
(%i5) float(2**100);
```

```
(%o5) 1.2676506002282294 10+30
```

```
(%i6) float(2/3+5/8);
```

```
(%o6) 1.291666666666667
```

```
(%i7) bfloat(%pi);
```

```
(%o7) 3.141592653589793b0
```

```
(%i8) fpprec:200;
```

```
(%o8) 200
```

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

```
(%o9) 3.1415926535897932384626433832[142digits]8521105559644622948954930382b0
```

```
(%i10) set_display('ascii)$
```

```
(%i11) bfloat(%pi);
```

```
(%o11) 3.14159265358979323846264338327950288419716939937510582097494459
```

```
2307816406286208998628034825342117067982148086513282306647093844609550
```

```
582231725359408128481117450284102701938521105559644622948954930382b0
```

1.3. Funciones matemáticas de interés

Las principales funciones matemáticas elementales se teclean en MÁXIMA del siguiente modo:

■ $\text{sqrt}(x) = \sqrt{x}$	■ $\text{atan}(x) = \text{arcotangente de } x$
■ $\exp(x) = e^x$	■ $\sinh(x) = \text{seno hiperbólico de } x$
■ $\log(x) = \log x$ (logaritmo neperiano)	■ $\cosh(x) = \text{coseno hiperbólico de } x$
■ $\log(x)/\log(b) = \log_b x$	■ $\tanh(x) = \text{tangente hiperbólico de } x$
■ $\sin(x) = \text{sen } x$	■ $\text{asinh}(x) = \text{arcoseno hiperbólico de } x$
■ $\cos(x) = \text{cos } x$	■ $\text{acosh}(x) = \text{arcocoseno hiperbólico de } x$
■ $\tan(x) = \text{tangente de } x$	■ $\text{atanh}(x) = \text{arcotangente hiperbólica de } x$
■ $\sec(x) = \text{secante de } x$	■ $\text{abs}(x) = x $
■ $\csc(x) = \text{cosecante de } x$	
■ $\cot(x) = \text{cotangente de } x$	
■ $\text{asin}(x) = \text{arcoseno de } x$	
■ $\text{acos}(x) = \text{arcocoseno de } x$	

funciones
matemáticas

Funciones relativas al cálculo con números naturales

■ $n! = \text{factorial de } n$	■ $\text{divisors}(n)$ nos da los divisores del número n .
■ $\text{primep}(n)$ nos dice si el número natural n es primo.	■ $\text{remainder}(n, m)$ nos da el resto de la división de n entre m .
■ $\text{oddp}(n)$ nos dice si el número natural n es impar.	■ $\text{quotient}(n, m)$ nos da el cociente de la división de n entre m .
■ $\text{evenp}(n)$ nos dice si el número natural n es par.	■ $\text{binomial}(m, n) = \binom{m}{n}$
■ $\text{factor}(n) =$ da la descomposición en factores primos del número natural n	

cálculo con
naturales

Funciones relativas al cálculo con números complejos

Empezamos aclarando que $\%i$ representa al número imaginario unidad $i = \sqrt{-1}$. Las operaciones aritméticas básicas se realizan como se han definido antes. Además podemos utilizar las siguientes funciones:

■ $\text{rectform}(z)$ nos da la forma binómica del número complejo z .	■ $\text{polarform}(z)$ nos da la forma polar del número complejo z .
■ $\text{realpart}(z)$ devuelve la parte real de z .	■ $\text{conjugate}(z) =$ devuelve el conjugado del número z
■ $\text{imagpart}(z)$ devuelve la parte imaginaria del número complejo z .	■ $\text{abs}(z)$ nos da el módulo de z .

cálculo con
complejos

Constantes matemáticas

%phi

%e

%inf

%i

$$\begin{aligned} \%pi &= \pi \simeq 3.14159 \\ \%e &= e \simeq 2.71828 \\ \%inf &= \infty \\ \%i &= i = \sqrt{-1} \\ \%phi &= \phi = \frac{1 + \sqrt{5}}{2} \text{ (número de oro)} \end{aligned}$$

1.4. Definición de variables y funciones

1.4.1. Variables

Supongamos que tenemos que hacer una serie de cálculos en los cuales interviene repetidamente una constante, como por ejemplo la constante de la gravitación universal, o en los que interviene repetidamente la función $f(x) = (1.45)^x$. Se hace necesario entonces definir variables y funciones con estos valores a la hora de realizar cálculos. Las variables y las funciones pueden ser designadas por letras o por sucesiones de letras.

Veamos cómo se hace esto con MÁXIMA. Supongamos que queremos definir la constante de la gravitación universal $G = 6.67 \times 10^{-11}$ con MÁXIMA. Entonces deberíamos hacer

```
(%i54) G:6.67*10**(-11);
```

```
(%o54) 6.67 10-11
```

Si ahora tenemos dos cuerpos de masa $3Kg$ separados a una distancia de $10m$ la fuerza con la que se atraen es

```
(%i55) G*3**2/100;
```

```
(%o55) 6.0030000000000002 10-12
```

Algunos comandos relacionados con la gestión de variables son los que siguen.

kill

remvalue
values

- **kill(a1,a2,...)** elimina los valores de las variables a1, a2, ...
kill(all) borra todos los valores
- **remvalue(var1,var2,...)** borra los valores de las variables pasadas como argumento
- **values** muestra las variables con valor asignado

Si en una misma línea queremos definir varias variables, o escribir varias expresiones debemos separar estas con ”;”.

1.4.2. Funciones

Abordemos ahora la definición de funciones. Para definir la función de una variable $f(x) = (1.45)^x$ se tiene que introducir la orden:

```
(%i61) f(x):=1.45**x;
```

```
(%o61) 
$$f(x) := 1.45^x$$

```

Una vez definida la función se puede utilizar como cualquier otra definida en MÁXIMA :

```
(%i66) f(.3);
```

```
(%o66) 1.11791916279868
```

Tienes que tener en cuenta el siguiente detalle a la hora de definir una función:

- El “igual” de la definición para MÁXIMA es la combinación “:=” más “=”.

También se pueden definir funciones que dependan de más de una variable. Por ejemplo:

```
(%i64) AreaRectangulo(base,altura):=base*altura;
```

```
(%o64) 
$$AreaRectangulo(base, altura) := base \, altura$$

```

```
(%i65) AreaRectangulo(2,4);
```

```
(%o65) 8
```

Funciones recurrentes

Una de las ventajas de los lenguajes de programación es la definición de funciones por recurrencia. Aunque MÁXIMA tiene definida una función para calcular el factorial de un número bien la podríamos haber definido como sigue:

```
(%i68) calculafactorial(n):= if n=1 then 1 else n*calculafactorial(n-1);
```

```
(%o68) 
$$calculafactorial(n) := if n = 1 then 1 else n \, calculafactorial(n - 1)$$

```

```
(%i71) calculafactorial(7);
```

(%o71) 5040

Quizás hace falta explicar la secuencia de control:

if condición then acción si condición es verdadera else acción si condición es falsa

Finalmente si queremos borrar funciones que hayamos definido nosotros usaremos el comando `remfunction` como describimos.

`remfunction`

- `remfunction (f1, ..., fn)`: desliga las definiciones de función de sus símbolos `f1, ..., fn`.
- `remfunction (all)`: desliga todas las definiciones de funciones.

1.5. 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 una sentencia que nos remita a resultados anteriores. Dicha sentencia es `%`. Por ejemplo, si queremos calcular $\cos(\sin 20^\circ)$ tendríamos que calcular primero $\sin 20^\circ$, para después calcular el coseno de dicha cantidad. Esta operación podríamos hacerla del modo siguiente:

(%i16) `grados: 2*%pi/360;`

(%o16) $\frac{\pi}{180}$

(%i17) `sin(20*grados);`

(%o17) $\sin\left(\frac{\pi}{9}\right)$

(%i18) `cos(%);`

(%o18) $\cos\left(\sin\left(\frac{\pi}{9}\right)\right)$

(%i19) `bfloat(%);`

(%o19) $9.420790505828131b - 1$

Para remitirnos a un resultado obtenido en el paso n debemos escribir `%on`.

1.6. La ayuda de MÁXIMA

El entorno MÁXIMA permite acceder a la amplia ayuda incluida con Maxima de una manera gráfica. Desde la línea de comandos tenemos algunas órdenes que nos pueden ser útiles.

- `describe(expr)` ó `?expr` nos da información sobre `expr`
- `example(expr)` devuelve un ejemplo de `expr`
- `apropos("expr")` nos dice qué comandos están relacionados con `expr`
- `??expr` nos devuelve los comandos que contienen `expr`

`describe`

`example`
`apropos`
`??`

1.7. Resolución numérica de ecuaciones

Supongamos que queremos resolver la ecuación polinómica $x^2 + 2x - 7 = 0$. En primer lugar, en MÁXIMA dicha expresión debe escribirse $x^2 + 2*x - 7 = 0$. Una vez que sabemos como escribir ecuaciones, el comando para resolverlas en MÁXIMA es `solve`. Así, para resolver la ecuación anterior escribiremos

```
(%i20) solve(x^2+2*x-7=0,x);
```

```
(%o20) [x = -2*sqrt(2) - 1, x = 2*sqrt(2) - 1]
```

Utilizando el comando `float` obtenemos soluciones numéricas aproximadas a partir de las expresiones exactas sencillas $-1 - 2\sqrt{2}$ y $-1 + 2\sqrt{2}$.

```
(%i21) float(%);
```

```
(%o21) [x = -3.82842712474619, x = 1.82842712474619]
```

El comando `solve` admite diferente usos. Los comentamos ahora.

- `solve(expr, x)`
- `solve(expr)`
- `solve ([eqn 1, ..., eqn n], [x 1, ..., x n])`
- `multiplicities`: es una variable que define el sistema cuando resolvemos una ecuación y que contiene las multiplicidades de las soluciones encontradas en la ultima ejecución de `solve` o `realroots`.

`solve`

Este comando resuelve la ecuación algebraica `expr` de incógnita `x` y devuelve una lista de igualdades con la `x` despejada. Si `expr` no es una igualdad, se supone que se quiere resolver la ecuación `expr = 0`. El argumento `x` puede ser una función (por ejemplo, `f(x)`), puede omitirse `x` si `expr` contiene solamente una variable. El argumento `expr` puede ser una expresión racional y puede contener funciones trigonométricas, exponenciales, etcétera.

A la variable `multiplicities` se le asignará una lista con las multiplicidades de las soluciones individuales devueltas por `solve`.

```
(%i27) solve(x^6-17*x^5+115*x^4-391*x^3+692*x^2-592*x+192=0,x);
```

```
(%o27) [x = 3, x = 1, x = 4]
```

```
(%i28) multiplicities;
```

```
(%o28) [1, 2, 3]
```

Con estas órdenes vemos que el polinomio al que le hemos calculado las raíces se factoriza así:

$$x^6 - 17x^5 + 115x^4 - 391x^3 + 692x^2 - 592x + 192 = (x-3)(x-1)^2(x-4)^3$$

La llamada `solve([eqn 1, ..., eqn n], [x 1, ..., x n])` resuelve un sistema de ecuaciones polinómicas simultáneas (lineales o no) llamando a `linsolve` o `algsys` y devuelve una lista de listas con soluciones para las incógnitas. En caso de haberse llamado a `linsolve` esta lista contendrá una única lista de soluciones. La llamada a `solve` tiene dos listas como argumentos. La primera lista tiene las ecuaciones a resolver y la segunda son las incógnitas cuyos valores se quieren calcular. Si el número de variables en las ecuaciones es igual al número de incógnitas, el segundo argumento puede omitirse.

Por ejemplo el sistema $x + y = 2$, $x - 3y + z = 3$, $x - y + z = 0$ puede resolverse del siguiente modo

```
(%i29) solve([x+y=2,x-3*y+z=3,x-y+z=0]);
```

```
(%o29) [[z = -5, y = -3/2, x = 7/2]]
```

```
(%i30) solve([x+y=2,x-3*y+z=3,x-y+z=0],[x,y,z]);
```

```
(%o30) [[x = 7/2, y = -3/2, z = -5]]
```

Ejercicio. Factoriza el polinomio

$$x^8 - 14x^7 + 89x^6 - 336x^5 + 820x^4 - 1328x^3 + 1412x^2 - 912x + 288,$$

para ello se pide:

1. Calcular las raíces de $p(x)$ y sus multiplicidades.
2. Dar la factorización.
3. Dar el producto de todas las raíces del polinomio (teniendo en cuenta sus multiplicidades).

1.8. Cálculo de límites

MÁXIMA tiene también una sentencia para calcular límites funcionales. Este comando es `limit` y tiene diferentes usos:

- `limit (expr, x, val, dir)`
- `limit (expr, x, val)`
- `limit (expr)`

`limit`

La función `limit` calcula el límite de `expr` cuando la variable real x se aproxima al valor `val` desde la dirección `dir`. El argumento `dir` puede ser el valor *plus* para un límite por la derecha, *minus* para un límite por la izquierda o simplemente se omite para indicar un límite en ambos sentidos.

La función `limit` utiliza los símbolos especiales siguientes: *inf* (más infinito) y *minf* (menos infinito). En el resultado también puede hacer uso de *und* (indefinido), *ind* (indefinido pero acotado) y *infinity* (infinito complejo).

La variable `lhospitallim` guarda el número máximo de veces que la regla de L'Hopital es aplicada en la función `limit`, evitando bucles infinitos al iterar la regla en casos como `limit (cot(x)/csc(x), x, 0)`. Si la variable `tlimswitch` vale `true`, hará que la función `limit` utilice desarrollos de Taylor siempre que le sea posible. La variable `limsubst` evita que la función `limit` realice sustituciones sobre formas desconocidas, a fin de evitar fallos tales como que `limit (f(n)/f(n+1), n, inf)` devuelva 1. Dándole a `limsubst` el valor `true` se permitirán tales sustituciones. La función `limit` con un solo argumento se utiliza frecuentemente para simplificar expresiones constantes, como por ejemplo `limit (inf-1)`. La instrucción `example(limit)` muestra algunos ejemplos.

`inf`
`minf`
`ind`
`unf`
`infinity`

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

```
(%i36) limit(sin(x)/x,x,0);
```

```
(%o36) 1
```

Así que, el límite $\lim_{x \rightarrow 0} \frac{\sin^3(x^2)}{1-\cos(x^3)}$ que apareció en el examen de febrero de 2006 y que requería hacer un desarrollo de Taylor de orden 6 se calcularía con MÁXIMA como:

```
(%i42) limit(sin(x^2)^3/(1-cos(x^3)),x,0);
```

```
(%o42) 2
```

Otros ejemplos fáciles de entender con la explicación del comando:

```
(%i37) limit(sin(x)/x,x,inf);
```

```
(%o37) 0
```

```
(%i38) limit(1/x,x,0);
```

(%o38) $infinity$

(%i39) limit(1/x,x,0,plus);

(%o39) ∞

(%i40) limit(1/x,x,0,minu);

(%o40) $-\infty$

1.9. Condicionales y bucles

La base de la programación de los algoritmos numéricos son los condicionales y los bucles. Una breve explicación del uso de `if` la puedes encontrar en la página 7.

Para los bucles disponemos de `for`, que tiene las siguientes variantes:

`for`

- `for <var>:<val1> step <val2> thru <val3> do <expr>`
- `for <var>:<val1> step <val2> while <cond> do <expr>`
- `for <var>:<val1> step <val2> unless <cond> do <expr>`

Cuando el incremento de la variable es la unidad, se puede obviar la parte de la sentencia relativa a `step`, dando lugar a

- `for <var>:<val1> thru <val3> do <expr>`
- `for <var>:<val1> while <cond> do <expr>`
- `for <var>:<val1> unless <cond> do <expr>`

Cuando no sea necesaria la presencia de una variable de recuento de iteraciones, también se podrá prescindir de los `for`, como en:

`while`
`unless`

- `while <cond> do <expr>`
- `unless <cond> do <expr>.`

(%i1) for k:0 thru 8 do (angulo:k*pi/4,print(
"El valor del seno de ",angulo, "es",sin(angulo)));

El valor del seno de 0 es 0

El valor del seno de $\frac{\pi}{4}$ es $\frac{1}{\sqrt{2}}$

El valor del seno de $\frac{\pi}{2}$ es 1

El valor del seno de $\frac{3\pi}{4}$ es $\frac{1}{\sqrt{2}}$

El valor del seno de π es 0

El valor del seno de $\frac{5\pi}{4}$ es $-\frac{\sqrt{2}}{2}$

El valor del seno de $\frac{3\pi}{2}$ es -1

El valor del seno de $\frac{7\pi}{4}$ es $-\frac{\sqrt{2}}{2}$

El valor del seno de 2π es 0

```
(%o1)                                     done
```

```
(%i3) for k:0 while k<3 do print(k);
```

```
(%o3)                                     0 1 2 done
```

```
(%i4) for k:0 unless k>3 do print(k);
```

```
(%o4)                                     0 1 2 3 done
```

Si queremos sumar los cuadrados de los 10 primeros números naturales con un bucle `while .. do` escribiremos:

```
(%i13) suma:0;
indice:0;
while indice<=10 do (suma:suma+indice**2,indice:indice+1);
print(suma);
```

```
(%o16)                                     0 0done 385
```

Para realizar sumatorios, como no podía ser de otra forma, MÁXIMA tiene implementada la función `sum`.

```
sum(expr, i, i0, i1)
```

`sum`

Representa la suma de los valores de *expr* según el índice *i* varía de *i0* hasta *i1*. Si la diferencia entre los límites superior e inferior es un número entero, el sumando *expr* se evalúa para cada valor del índice *i*, siendo el resultado una suma en forma explícita. En caso contrario, el rango del índice no está definido, aplicándose entonces algunas reglas que permitan simplificar la suma. Cuando la variable global `simplsum` valga `true`, se aplicarán reglas adicionales.

```
(%i17) sum(i**2,i,1,10);
```

```
(%o17)                                     385
```

Ejercicio. 1. Calcular la suma $7+9+11+13+\dots+131 =$.

1.10. Manejo de expresiones simbólicas

MÁXIMA dispone de unos comandos para manipular algebraicamente expresiones de los tipos que siguen:

1. Polinomios en una y varias variables.
2. Fracciones cuyo numerador y denominador son polinomios.
3. Expresiones trigonométricas.
4. Expresiones que contienen exponenciales y logaritmos.

Expresiones racionales

Supongamos que tenemos una expresión o fórmula de los tipos 1 ó 2. Los comandos que MÁXIMA posee para manipular estas expresiones son los siguientes.

- **expand(arg)**: multiplica y realiza potencias en la expresión arg
- **expand(arg,n,m)**: multiplica y realiza potencias para grados entre -m y n en la expresión arg
- **partfrac(frac,var)** descompone en fracciones simples la fracción frac respecto a la variable var
- **factor(arg)**: escribe la expresión arg como producto de factores más sencillos
- **ev(arg,cond1,...,condn)**: evalúa la expresión arg asignando los valores que se le van indicando a continuación;
- **simp**: es una variable que por defecto toma el valor **true**. Si la definimos a **false** evitaremos todas las simplificaciones.
- **radcan(arg)**: transforma arg a su forma canónica, que es un formato que Maxima utiliza para reducir expresiones algebraicas equivalentes a una forma única.
- **ratsimp(arg)**: también simplifica cualquier expresión racional. El resultado se devuelve como el cociente de dos polinomios. En ocasiones no es suficiente con una sola ejecución de **ratsimp**, por lo que será necesario aplicarla más veces, esto es lo que hace **fullratsimp**.

expand

partfrac

factor

ev

simp

radcan

ratsimp

fullratsimp

```
(%i132) q: (x+3)^5-(x-a)^3+(x+b)^(-1)+(x-1/4)^(-5);
```

$$(\%o132) \quad \frac{1}{x+b} - (x-a)^3 + (x+3)^5 + \frac{1}{\left(x-\frac{1}{4}\right)^5}$$

```
(%i133) expand(q);
```

$$\frac{1}{x^5 - \frac{5x^4}{4} + \frac{5x^3}{8} - \frac{5x^2}{32} + \frac{5x}{256} - \frac{1}{1024}}$$

$$(\%o133) \quad + \frac{1}{x+b} + x^5 + 15x^4 + 89x^3 + 3ax^2 + 270x^2 - 3a^2x + 405x + a^3 + 243$$

```
(%i134) /*con esta orden no desarrollamos todas las potencias*/
expand(q,3,2);
```

$$(\%o134) \quad \frac{1}{x+b} + (x+3)^5 - x^3 + 3ax^2 - 3a^2x + \frac{1}{\left(x-\frac{1}{4}\right)^5} + a^3$$

```
(%i135) ev(q,a:3,b:2);
```

```
(%o135) 
$$(x+3)^5 + \frac{1}{x+2} + \frac{1}{\left(x-\frac{1}{4}\right)^5} - (x-3)^3$$

```

```
(%i141) simp:false;
```

```
(%o141) 
$$false$$

```

```
(%i142) 2+3+5;
```

```
(%o142) 
$$2+3+5$$

```

```
(%i143) simp:true;
```

```
(%o143) 
$$true$$

```

```
(%i144) %o142;
```

```
(%o144) 
$$10$$

```

```
(%i145) expr:(x^2-x)/(x^2+x-6)-5/(x^2-4);
```

```
(%o145) 
$$\frac{x^2-x}{x^2+x-6} - \frac{5}{x^2-4}$$

```

```
(%i146) factor(%);
```

```
(%o146) 
$$\frac{(x-3)(x^2+4x+5)}{(x-2)(x+2)(x+3)}$$

```

```
(%i149) expand(expr);
```

```
(%o149) 
$$\frac{x^2}{x^2+x-6} - \frac{x}{x^2+x-6} - \frac{5}{x^2-4}$$

```

```
(%i150) partfrac(expr,x);
```

```
(%o150) 
$$-\frac{12}{5(x+3)} + \frac{5}{4(x+2)} - \frac{17}{20(x-2)} + 1$$

```

```
(%i151) (x^(a/2)-1)^2*(x^(a/2)+1)^2 / (x^a-1);
```

```
(%o151) 
$$\frac{\left(x^{\frac{a}{2}} - 1\right)^2 \left(x^{\frac{a}{2}} + 1\right)^2}{x^a - 1}$$

```

```
(%i152) fullratsimp(%);
```

```
(%o152) 
$$x^a - 1$$

```

Expresiones logarítmicas

MÁXIMA dispone de algunas variables globales que permiten al usuario darle al motor simbólico ciertas directrices sobre qué debe hacer con las expresiones, cómo reducirlas o transformarlas. Algunas de ellas controlan las transformaciones de logaritmos y de trigonometría. La variable **logexpand** puede tomar los valores false, true (por defecto) y super. La primera opción es la más restrictiva y no realiza transformaciones logarítmicas, true realiza algunas y super todas las que el programa tiene definidas.

```
(%i160) log(x^r)-log(x*y) + a*log(x/y);
```

```
(%o160) 
$$-\log(xy) + a \log\left(\frac{x}{y}\right) + r \log(x)$$

```

```
(%i162) expand(%i160);
```

```
(%o162) 
$$-\log(xy) + a \log\left(\frac{x}{y}\right) + r \log(x)$$

```

```
(%i163) logexpand:false;
```

```
(%o163) 
$$false$$

```

```
(%i164) expand(%i160);
```

```
(%o164) 
$$-\log(xy) + a \log\left(\frac{x}{y}\right) + \log(x^r)$$

```

```
(%i169) logexpand:true;
```

```
(%o169) 
$$true$$

```

```
(%i170) expand(%i160);
```

```
(%o170) 
$$-\log(xy) + a \log\left(\frac{x}{y}\right) + r \log(x)$$

```

```
(%i171) logexpand:super;
```

```
(%o171) 
$$super$$

```

```
(%i172) expand(%i153);
```

```
(%o172) 
$$-a \log(y) - \log(y) + r \log(x) + a \log(x) - \log(x)$$

```

logcontract

logcontract es una función que permite compactar expresiones logarítmicas

```
(%i174) exprlog:2*(a*log(x) + 2*a*log(y));
```

```
(%o174) 
$$2 (2 a \log(y) + a \log(x))$$

```

```
(%i175) expand(%);
```

```
(%o175) 
$$4 a \log(y) + 2 a \log(x)$$

```

```
(%i176) logcontract(exprlog);
```

```
(%o176) 
$$a \log(x^2 y^4)$$

```

Expresiones trigonométricas

MÁXIMA conoce las identidades trigonométricas y puede usarlas para simplificar expresiones en las que aparezcan dichas funciones. En lugar de **expand** y **factor**, utilizaremos las órdenes:

trigreduce

trigsimp

trigreduce

- **trigexpand**(expresion): expande las expresiones trigonométricas e hipérbolicas pasadas como argumento. El comportamiento de **trigexpand** se controla con las variables globales **trigexpand**, **trigexpandplus** y **trigexpandtimes**, cuyos valores por defectos son, respectivamente, **false**, **true** y **true**.
- **trigsimp**(expresion) simplifica las funciones trigonométricas e hipérbolicas del argumento
- **trigreduce**(expresion) simplifica las funciones trigonométricas e hipérbolicas

Álgebra lineal

2.1. Operaciones básicas con matrices

Vimos en la teoría que existen muchas clases de matrices, en los primeros temas hicimos una sustancial distinción entre dos tipos: las que tienen determinante no nulo y en consecuencia son invertibles y las que tienen determinante igual a 0 y son por lo tanto no invertibles. MÁXIMA nos permite saber cuando una matriz es invertible o no, calcular su determinante y su inversa. Vamos a ir introduciendo los comandos que nos permiten realizar tales operaciones.

El comando `matrix`

■ `matrix` (fila 1, ..., fila n)

`matrix`

La función de MÁXIMA `matrix` devuelve una matriz rectangular con las filas *fila 1*, ..., *fila n*. Cada fila es una lista de expresiones. Todas las filas deben tener el mismo número de miembros. Las operaciones + (suma), - (resta), * (multiplicación) y / (división), se llevan a cabo **elemento a elemento**¹ cuando los operandos son dos matrices, un escalar y una matriz o una matriz con un escalar. La operación ^ (exponenciación, equivalente a **) se lleva cabo también **elemento a elemento** si los operandos son un escalar y una matriz o una matriz y un escalar, pero no si los operandos son dos matrices. El **producto matricial** se representa con el operador de multiplicación no conmutativa .. El correspondiente operador de exponenciación no conmutativa es ^^ . Dada la matriz A , $A.A = A^{..2}$ y $A^{..-1}$ es la **inversa** de A , si existe. Algunas variables controlan la simplificación de expresiones que incluyan estas operaciones: `doallmxops`, `domxexpt`, `dommxops`, `doscmxops` y `doscmxplus`. Hay otras opciones adicionales relacionadas con matrices: `lmxchar`, `rmxchar`, `ratmx`, `listarith`, `detout`, `scalarmatrix` y `sparse`. Hay también algunas funciones que admiten matrices como argumentos o que devuelven resultados matriciales:

¹Mucho cuidado con esto porque con * no conseguiremos la multiplicación usual de matrices y / da una división que en ningún caso hemos definido en nuestra asignatura

eigenvalues

eigenvectors

determinant

charpoly

addcol

addrow

copymatrix

nullspace

transpose

echelon

rank

diagmatrix

invert

zeromatrix

- **eigenvalues** devuelve los valores propios de la matriz A
- **eigenvectors** (A): devuelve una lista de listas, la primera de las cuales es la salida de eigenvalues y las siguientes son los vectores propios de la matriz asociados a los valores propios correspondientes.
- **determinant** (A): nos da el determinante de la matriz A
- **charpoly** (A, x): calcula el polinomio característico de A usando la variable x
- **addcol** (A , lista 1, ..., lista n): añade la/s columna/s dada/s por la/s lista/s (o matrices) a la matriz A .
- **addrow** (A , lista 1, ..., lista n): añade la/s fila/s dada/s por la/s lista/s (o matrices) a la matriz A .
- **copymatrix** (A): devuelve una copia de la matriz A independiente de ésta (cualquier modificación en A no alterará la copia que hemos hecho de A)
- **transpose** (A): nos da la transpuesta de A .
- **nullspace** (A): si A es una matriz, devuelve $\text{span}(v_1, \dots, v_n)$, siendo v_1, \dots, v_n la base del espacio solución de $Ax = \mathbf{0}$. Si el resultado contiene sólo a $\mathbf{0}$, devuelve $\text{span}()$.

- **echelon** (A): devuelve la forma escalonada de la matriz A , obtenida por eliminación gaussiana, también se puede obtener con la función **triangularize**.
- **rank** (A): calcula el rango de A .
- **diagmatrix** (n, x): devuelve una matriz diagonal de orden n con los elementos de la diagonal todos ellos iguales a x . La llamada **diagmatrix**($n, 1$) devuelve una matriz identidad (igual que **ident**(n)).
- **invert** (A): da la inversa de la matriz M , calculada por el método del adjunto.
- **zeromatrix** (m, n): devuelve una matriz rectangular m por n con todos sus elementos iguales a cero.

Vamos a resolver unos problemas de álgebra lineal usando MÁXIMA .

Ejemplo 2.3. Sea $f : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ una aplicación lineal tal que $M_{\beta_e^3 \beta_e^3}(f) = \begin{pmatrix} 3 & 2 & 1 \\ 4 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix}$ y considera la base $\equiv \{(1, 1, 1), (1, 1, 0), (1, 0, 0)\}$. Responde a las siguientes cuestiones:

1. Calcula $M_{\beta\beta}(f)$.

Solución. *Calculamos*

$$M_{\beta\beta}(f) = M_{\beta\beta_c^3} M_{\beta_c^3\beta_c^3}(f) M_{\beta_c^3\beta}$$

Así que:

$$M_{\beta\beta}(f) = \begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & 3 \\ -3 & -2 & -1 \end{pmatrix}$$

Las órdenes introducidas en MÁXIMA han sido las que siguen:

```
(%i1) m:matrix([3,2,1],[4,3,2],[1,1,1]);
```

```
(%o1) 
$$\begin{pmatrix} 3 & 2 & 1 \\ 4 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

```

```
(%i2) rank(m);
```

```
(%o2) 2
```

```
(%i4) mcb:transpose(matrix([1,1,1],[1,1,0],[1,0,0]));
```

```
(%o4) 
$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

```

```
(%i5) mbc:mcb^^-1;
```

```
(%o5) 
$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & 0 \end{pmatrix}$$

```

```
(%i6) mccf:matrix([3,2,1],[4,3,2],[1,1,1]);
```

```
(%o6) 
$$\begin{pmatrix} 3 & 2 & 1 \\ 4 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

```

```
(%i7) mbbf:mbc.mccf.mcb;
```

$$(\%o7) \quad \begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & 3 \\ -3 & -2 & -1 \end{pmatrix}$$

2. Calcula las ecuaciones de $\text{Ker } f$ respecto de la base β y da una base de $\text{Ker } f$ expresando las coordenadas de los vectores que aparezcan en la base β .

Solución. Como $M_{\beta\beta}(f)$ tiene rango 2:

$$\begin{aligned} \text{Ker } f &= \{(x, y, z)_{\beta} : M_{\beta\beta}(f)(x, y, z)^t = (0, 0, 0)^t\} = \{(x, y, z)_{\beta} : 3x + 2y + z \\ &= 6x + 5y + 3z = 0\} \end{aligned}$$

$$\beta_{\text{Ker } f} = \{(1, -3, 3)_{\beta}\}$$

Las órdenes realizadas en MÁXIMA han sido las que siguen:

(%i9) nullspace(mbbf);

$$(\%o9) \quad \text{span} \left(\begin{pmatrix} 1 \\ -3 \\ 3 \end{pmatrix} \right)$$

3. Calcula las ecuaciones de $\text{Im } f$ respecto de la base β_c^3 y da una base de $\text{Im } f$ expresando las coordenadas de los vectores que aparezcan en la base β_c^3 .

Solución. Usando que $3 = \dim \mathbb{R}^3 = \dim \text{Ker } f + \dim \text{Im } f$ se saca que $\dim \text{Im } f = 2$. Ahora usando que las columnas de $M_{\beta_c^3 \beta_c^3}(f)$ generan $\text{Im } f$ se tiene que $\text{Im } f = \{(3, 2, 1), (2, 3, 1), (1, 2, 1)\}$ y

$$\beta_{\text{Im } f} = \{(2, 3, 1), (1, 2, 1)\}.$$

Ahora la ecuación de la imagen es

$$\begin{vmatrix} 2 & 3 & 1 \\ 1 & 2 & 1 \\ x & y & z \end{vmatrix} = x - 1y + 1z = 0$$

Los cálculos con MÁXIMA :

(%i11) determinant(matrix([2,3,1],[1,2,1],[x,y,z]));

$$(\%o11) \quad 2(2z - y) - 3(z - x) + y - 2x$$

```
(%i12) expand(%);
```

```
(%o12) 
$$z - y + x$$

```

4. Encuentra bases β_1 y β_2 tales que $M_{\beta_1\beta_2}(f) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.

Solución. Basta con tomar $\beta_1 = \{(1, 0, 0), (0, 0, 1), (1, -2, 1)\}$ y $\beta_2 = \{(3, 4, 1), (-2, -2, 0), (1, 0, 0)\}$.

Hemos calculado en MÁXIMA así:

```
(%i14) /*buscamos un vector de Ker f para tomarlo como tercer
vector de la base b1*/ nullspace(mccf);
```

```
(%o14) 
$$\text{span} \left( \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \right)$$

```

```
(%i16) /*calculamos las imágenes de los otros dos vectores
de b1 que hemos tomado y que no pertenecen a Ker f*/ mcf.[1,0,0];
```

```
(%o16) 
$$\begin{pmatrix} 3 \\ 4 \\ 1 \end{pmatrix}$$

```

```
(%i18) mcf.[0,0,1];
```

```
(%o18) 
$$\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

```

```
(%i20) %o18-%o16;
```

```
(%o20) 
$$\begin{pmatrix} -2 \\ -2 \\ 0 \end{pmatrix}$$

```

Ejemplo 2.4. Estudia si la matriz

$$M = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 18 & 19 & 12 & 6 \\ -34 & -30 & -19 & -10 \\ 14 & 12 & 8 & 5 \end{pmatrix}$$

es o no diagonalizable

Solución.

- Empezamos definiendo la matriz M en MÁXIMA (la llamaremos m):

```
(%i107) m:matrix([3,0,0,0],[18,19,12,6],[-34,-30,-19,-10],  
[14,12,8,5]);
```

$$(\%o107) \quad \begin{pmatrix} 3 & 0 & 0 & 0 \\ 18 & 19 & 12 & 6 \\ -34 & -30 & -19 & -10 \\ 14 & 12 & 8 & 5 \end{pmatrix}$$

- Calculamos el polinomio característico

```
(%i110) charpoly(m,x);
```

$$((-x-19)(5-x)+80)(19-x)-12$$

$$(\%o110) \quad (120-30(5-x))+6(-12(-x-19)-240)(3-x)$$

- Lo ponemos en forma de suma de monomios con el comando `expand`:

```
(%i111) expand(%);
```

$$(\%o111) \quad x^4 - 8x^3 + 22x^2 - 24x + 9$$

- Calculamos las raíces del polinomio característico (son 1 y 3):

```
(%i112) solve(%=0,x);
```

$$(\%o112) \quad [x=3, x=1]$$

- Obtenemos las multiplicidades de las raíces del polinomio característico (ambas tienen multiplicidad 2):

```
(%i113) multiplicities;
```

$$(\%o113) \quad [2, 2]$$

- Calculamos una base de $V_3 = \{x \in \mathbb{R}^4 : (M - 3I_4)x = 0\}$ y como tiene dos vectores $\dim V_3 = 2 = m(3)$. Lo mismo le ocurre a V_1 , así que la matriz será diagonalizable.

```
(%i117) nullspace(m-3*ident(4));
```

$$(\%o117) \quad \text{span} \left(\begin{pmatrix} 0 \\ 12 \\ -20 \\ 8 \end{pmatrix}, \begin{pmatrix} 8 \\ -12 \\ 4 \\ 0 \end{pmatrix} \right)$$

```
(%i119) nullspace(m-1*ident(4));
```

$$(\%o119) \quad \text{span} \left(\begin{pmatrix} 0 \\ -24 \\ 36 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 12 \\ -24 \end{pmatrix} \right)$$

- Ponemos las bases obtenidas en el paso anterior por columnas para construir la matriz de paso que llamamos p :

```
(%i120) p:transpose(matrix([0,12,-20,8],[8,-12,4,0],
[0,-24,36,0],[0,0,12,-24]));
```

$$(\%o120) \quad \begin{pmatrix} 0 & 8 & 0 & 0 \\ 12 & -12 & -24 & 0 \\ -20 & 4 & 36 & 12 \\ 8 & 0 & 0 & -24 \end{pmatrix}$$

- Calculamos la inversa de p :

```
(%i125) p^(-1);
```

$$(\%o125) \quad \begin{pmatrix} \frac{7}{8} & \frac{3}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{8} & 0 & 0 & 0 \\ \frac{3}{8} & \frac{1}{3} & \frac{1}{4} & \frac{1}{8} \\ \frac{7}{24} & \frac{1}{4} & \frac{1}{6} & \frac{1}{24} \end{pmatrix}$$

- Comprobamos que todo ha sido correcto porque multiplicando $p^{-1}mp$ da la matriz diagonal que tiene la secuencia 3,3,1,1 en la diagonal:

```
(%i126) p^(-1).m.p;
```

$$(\%o126) \quad \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cálculo diferencial e integral

3.1. Introducción

MÁXIMA dispone de herramientas para hacer de forma rápida derivadas de funciones de una y varias variables, cálculo de primitivas de funciones de una variable, integrales definidas, desarrollos de Taylor de funciones de una variable, resolución numérica de ecuaciones, representaciones gráficas de funciones de una y dos variables, así como curvas en el espacio, etcétera. Vamos a centrarnos en esta práctica en cuestiones relativas a la integración de funciones, veremos los comandos con los que MÁXIMA nos permite resolver problemas en los que interviene la integral.

3.2. Descripción de comandos

3.2.1. Derivadas de funciones

Supongamos que tenemos una función de una variable $f(x)$ o de varias variables $f(x_1, x_2, \dots, x_n)$ a la que queremos calcular su derivada o derivada parcial respecto de alguna de sus variables. El comando que realiza ese cálculo con MÁXIMA es `diff`.

`diff`
`derivabbrev`

La función `diff` puede llamarse usando cualquiera de las formas siguientes:

- `diff(expr, x1, n1, ..., xm, nm)`: devuelve la derivada parcial de `expr` con respecto de `x1` `n1` veces, ..., `xm` `nm` veces.
- `diff(expr, x,n)`: devuelve la n -ésima derivada de `expr` respecto de `x`.
- `diff(expr, x)`: devuelve la primera derivada de `expr` respecto de la variable `x`.
- `diff(expr)`: devuelve el diferencial total de `expr`, esto es, la suma de las derivadas de `expr` respecto de cada una de sus variables, multiplicadas por el diferencial del de cada una de ellas. La forma nominal de `diff` es necesaria en algunos contextos, como para definir ecuaciones diferenciales. En tales casos, `diff` puede ir precedida de un apóstrofo (como `'diff`) para evitar el cálculo de la derivada.
- Si `derivabbrev` vale `true`, las derivadas se muestran como subíndices. En otro caso, se muestran en la notación

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

```
(%i6) g(x,y):=sin(x+y);
```

```
(%o6) g(x,y) := sin(x+y)
```

```
(%i7) diff(g(x,y),y);
```

```
(%o7) cos(y+x)
```

La segunda derivada de $f(x) = \sin x$ se calcula

```
(%i5) diff(sin(x),x,2);
```

```
(%o5) -sin(x)
```

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


```
(%i8) diff(g(x,y),x,3);
```

```
(%o8) 
$$-\cos(y+x)$$

```

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

```
(%i10) diff(g(x,y),x,1,y,1);
```

```
(%o10) 
$$-\sin(y+x)$$

```

3.2.2. Cálculo de primitivas e integral definida

MÁXIMA 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`.

La función `integrate` puede llamarse usando cualquiera de las formas siguientes:

- `integrate(expr, x)`: calcula simbólicamente la integral indefinida de `expr` respecto de `x`.
- `integrate(expr, x, a, b)`: resuelve una integral definida con límites de integración `a` y `b`. Los límites no pueden contener a `x`.

Por ejemplo, para calcular una primitiva de $f(x) = \sin x$ procedemos del siguiente modo.

```
(%i11) integrate(sin(x),x);
```

```
(%o11) 
$$-\cos(x)$$

```

$\int_0^1 x dx$ se calcularía del modo siguiente:

```
(%i13) integrate(x,x,0,1);
```

```
(%o13) 
$$\frac{1}{2}$$

```

Ejercicio. Calcular las primitivas de dos de las siguientes funciones:

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

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

(c) $f(x) = \cos x e^{cx}$

(d) $f(x) = sh(x)ch(cx)sh(3x)$.

3.2.3. Representación gráfica de funciones, curvas y superficies

MÁXIMA 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 los comandos `plot2d` y `wxplot2d` (ambos son similares pero el primero imprime las gráficas en una ventana aparte y permite definir opciones adicionales). Así, para representar la función $f(x) = \sin x$ en el dominio $[0, 2\pi]$ escribimos

```
(%i3) wxplot2d(sin(x),[x,0,2*%pi]);
```

```
(%o3) << Graphics >>
```

y obtenemos la imagen de la figura 3.1.

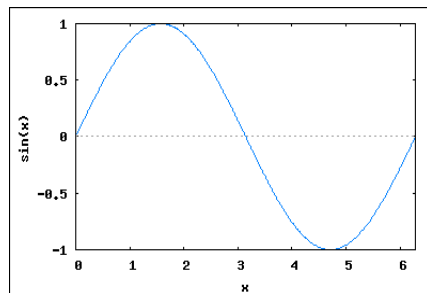


Figura 3.1: Gráfica de la función $f(x) = \sin x$ en el intervalo $[0, 2\pi]$

Si queremos representar varias funciones a la vez, hemos de escribir

```
(%i4) wxplot2d([sin(x),sin(2*x)],[x,0,2*%pi]);
```

```
(%o4) << Graphics >>
```

expresión que hará una representación gráfica simultánea de las funciones $\sin x$ y $\sin 2x$.

MÁXIMA nos facilita el trabajo pudiendo acceder a un menú gráfico pinchando en el botón denominado *Gráficos 2D*. Al pulsar aparece una ventana con varios campos para completar:

- **Expresión.** Es la función o funciones que queramos dibujar. Por defecto, wxMaxima rellena este espacio con `%` para referirse a la salida anterior.
- **Variable x.** es el intervalo de la variable x elegido para representar la función.
- **Variable y.** Nos sirve para elegir un intervalo de la variable y para representar la función.

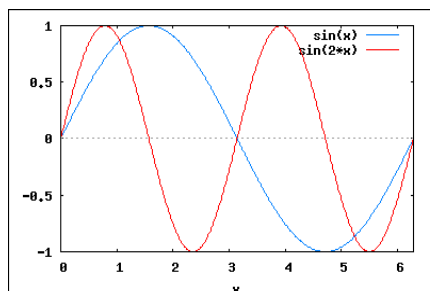


Figura 3.2: Gráficas de las funciones $f(x) = \sin x$ y $g(x) = \sin(2x)$

- **Graduaciones.** Es el número de puntos en los que el programa evalúa una función para su representación en polares.
- **Formato.** Maxima realiza por defecto la gráfica con un programa auxiliar. Si seleccionamos en línea, dicho programa auxiliar es wxMaxima y obtendremos la gráfica en una ventana alineada con la salida correspondiente. Hay dos opciones más y ambas abren una ventana externa para dibujar la gráfica requerida: gnuplot es la opción por defecto que utiliza el programa Gnuplot para realizar la representación; también está disponible la opción openmath que utiliza el programa XMaxima.
- **Opciones.** Aquí podemos seleccionar varias opciones, por ejemplo para que:
 - Dibuje los ejes de coordenadas ("set zeroaxis;");
 - Dibuje los ejes de coordenadas, de forma que cada unidad en el eje Y sea igual que el eje X ("set size ratio 1; set zeroaxis;").
 - Dibuje una cuadrícula ("set grid;") o dibuje una gráfica en coordenadas polares ("set polar; set zeroaxis;").
- **Gráfico al archivo.** Guarda el gráfico en un archivo con formato Postscript.

Otra forma de elegir las opciones para dibujar las gráficas es mediante la función `set_plot_option (opción)` que asigna un valor a una de las variables globales que controlan los gráficos. El argumento *opción* se especifica como una lista de dos o más elementos, en la que el primero es el nombre de una de las opciones de la lista `plot_options`. La función `set_plot_option` evalúa sus argumentos y devuelve `plot_options` tal como queda después de la actualización.

La variable `plot_options` consta de una lista de elementos que establecen las opciones por defecto para los gráficos. Si una opción está presente en una llamada a `plot2d` o a `plot3d`, este valor adquiere prevalencia sobre las opciones por defecto. En otro caso se utilizará el valor que tenga en `plot_options`. Las opciones por defecto se asignan mediante la función `set_plot_option` comentada en el párrafo anterior. Cada elemento de `plot_options` es una lista de dos o más elementos, el primero de los cuales es el nombre de la opción, siendo los siguientes los valores de aquella. En algunos casos el valor asignado es a su

vez una lista, que puede contener varios elementos. Las opciones gráficas que reconocen `plot2d` y `plot3d` son:

- **Opción: y.** Nos sirve para establecer el rango vertical del gráfico.
Ejemplo: `[y, - 3, 3]` establece el rango vertical como `[-3, 3]`.
- **Opción: nticks.** En `plot2d`, es el número inicial de puntos utilizados por el procedimiento adaptativo para la representación de funciones. También es el número de puntos a ser calculados en los gráficos paramétricos.
Ejemplo: `[nticks, 20]` El valor por defecto para `nticks` es 10.
- **Opción: ylabel.** Etiqueta del eje vertical en gráficos 2d.
Ejemplo: `[ylabel, "Temperature"]`
- **Opción: legend.** Etiquetas para las expresiones de los gráficos 2d. Si hay más expresiones que etiquetas, éstas se repetirán. Por defecto se pasarán los nombres de las expresiones o funciones, o las palabras `discrete1`, `discrete2`, ..., para gráficos de puntos. Ejemplo: `[legend, "Set 1", "Set 2", "Set 3"]`
- **Opción: style.** Estilos a utilizar para las funciones o conjuntos de datos en gráficos 2d. A la palabra `style` debe seguirle uno o más estilos. Si hay más funciones o conjuntos de datos que estilos, éstos se repetirán. Los estilos que se admiten son: `lines` para segmentos lineales, `points` para puntos aislados, `linespoints` para segmentos y puntos, `dots` para pequeños puntos aislados. Gnuplot también acepta el estilo `impulses`. Los estilos se pueden escribir como elementos de una lista, junto con algunos parámetros adicionales. `lines` acepta uno o dos números: el ancho de la línea y un entero que identifica el color. Los códigos de color por defecto son: 1, azul; 2, rojo; 3, magenta; 4, naranja; 5, marrón; 6, verde lima; 7, aguamarina.

`points` acepta uno, dos o tres parámetros; el primer parámetro es el radio de los puntos, el segundo es un entero para seleccionar el color, con igual codificación que en `lines` y el tercer parámetro sólo es utilizado por Gnuplot y hace referencia a varios objetos para representar los puntos. Los tipos de objetos disponibles son: 1, círculos rellenos; 2, circunferencias; 3, +; 4, x; 5, *; 6, cuadrados rellenos; 7, cuadrados huecos; 8, triángulos rellenos; 9, triángulos huecos; 10, triángulos rellenos invertidos; 11, triángulos huecos invertidos; 12, rombos rellenos; 13, rombos huecos.

`linesdots` acepta hasta cuatro parámetros: ancho de línea, radio de los puntos, color y tipo de objetos para representar puntos.
Ejemplo: `[style,[lines,2,3],[points,1,4,3]]` En este ejemplo se representará la primera (tercera, quinta, etc.) expresión con segmentos rectilíneos magenta de ancho 2, la segunda (cuarta, sexta, etc.) expresión con símbolos de suma naranja de tamaño 1 (círculos naranja en el caso de Openmath).
El estilo por defecto es `lines` de ancho 1 y diferentes colores.
- **Opción: grid.** Establece el número de puntos de la retícula a utilizar en las direcciones x e y en los gráficos de tres dimensiones.

Ejemplo: `[grid, 50, 50]` establece la retícula en 50 por 50 puntos. El valor por defecto es 30 por 30.

La representación gráfica de funciones de dos variables se hace mediante el comando `wxplot3d` o su análogo `plot3d` que dibujará los objetos en una ventana nueva y permitirá opciones como girar el gráfico. Por ejemplo, si queremos representar la función $f(x, y) = \sin(xy)$ en el dominio $[0, 3] \times [0, 3]$ hemos de escribir

```
(%i13) wxplot3d(sin(x*y), [x, 0, 3], [y, 0, 3]);
```

```
(%o13) << Graphics >>
```

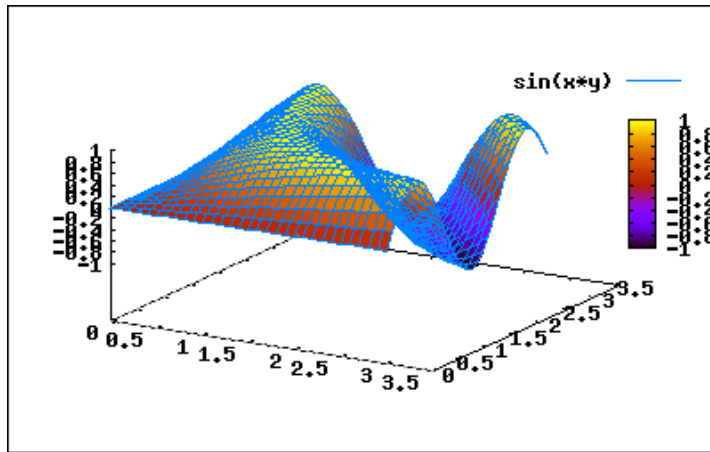


Figura 3.3: Gráfica de $\sin(xy)$

3.2.4. Curvas y superficies paramétricas

El comando empleado para representar curvas parametrizadas en el plano es `plot2d([parametric,x(t),y(t),[t,a,b]])`, la ejecución de este comando dibuja la gráfica de la curva $(x(t), y(t))$ en el intervalo $[a, b]$. 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

```
(%i16) wxplot2d([parametric,sin(t),sin(2*t),[t,0,2*%pi]]);
```

```
(%o16) << Graphics >>
```

Para representar curvas (paramétricas) y superficies (paramétricas) en el espacio usaremos el módulo `draw` que habrá que cargar previamente mediante

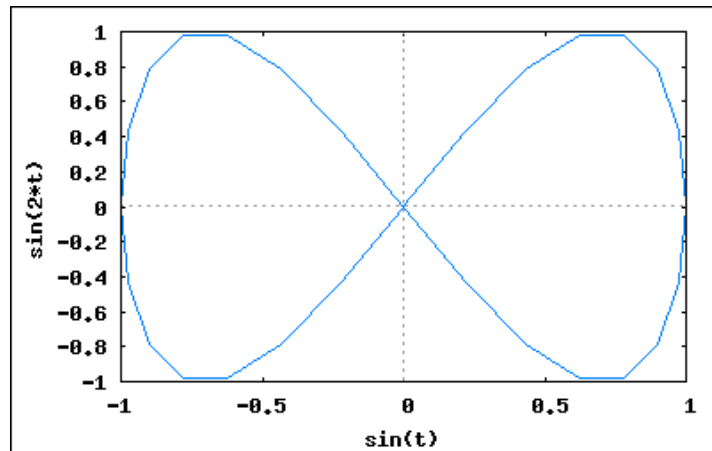


Figura 3.4: Gráfica de la curva $(\sin(t), \sin(2t))$

la sentencia `load(draw)`. Este módulo ofrece, entre otras, las funciones `draw2d`, `draw3d` y `draw`, que permiten dibujar escenas en 2d, 3d y gráficos múltiples y animaciones, respectivamente.

Resumimos las funciones más importantes del módulo

- `gr2d(opciones, objeto gráfico,...)` gráfico dos dimensional
- `gr3d(opciones, objeto gráfico,...)` gráfico tres dimensional
- `draw(opciones, objeto gráfico,...)` dibuja un gráfico
- `draw2d(opciones, objeto gráfico,...)` dibuja gráfico dos dimensional
- `draw3d(opciones, objeto gráfico,...)` dibuja gráfico tres dimensional

Exponemos un primer ejemplo:

```
(%i3) load(draw)$

(%i12) draw2d(
key = "polinomio de tercer grado",
explicit(x^3+x^2+10,x,0,4),
color = blue,
key = "curva paramétrica (cos(t)+3,t)",
line_width = 3,
nticks = 50,
parametric(cos(t)+3, t, t, 0, 20*%pi),
line_type = dots,
points_joined = true,
point_type = diamant,
point_size = 3,
color = red,
line_width = 1,
key = "Datos aleatorios",
```

`gr2d`

`gr3d`

`draw`

`draw2d`

`draw3d`

```
points(makelist(random(25.0),k,1,5)),
title = "Ejemplos de dibujo de curvas")$
```

El gráfico de salida será:

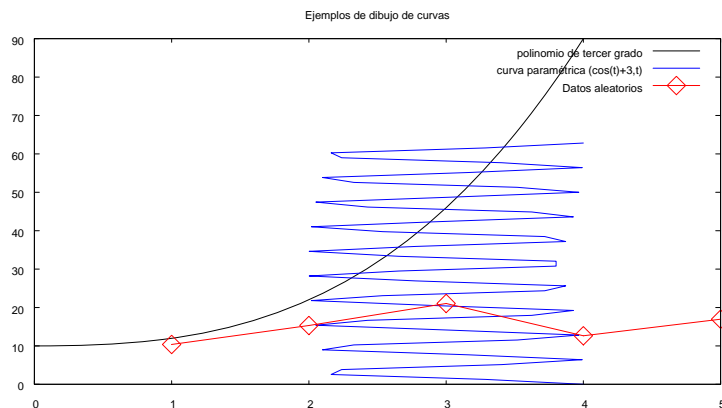


Figura 3.5: Salida del uso del comando `draw2d`

Por ejemplo, para representar la curva $\begin{cases} x(t) = \sin t \\ y(t) = \cos t \\ z(t) = t/3 \end{cases}$ en el dominio $[0, 15]$

y una esfera (asando coordenadas paramétricas, que en este caso son sus coordenadas polares) hemos de escribir:

```
(%i32) esfera:gr3d(key="esfera",
color=royalblue,
surface_hide=false,
parametric_surface(cos(u)*sin(v),sin(v)*sin(u),cos(v),
u,0,2*%pi,v,0,%pi)
);
```

```
gr3d(key = esfera,color = royalblue,surface_hide =
(%o32)
false,parametric_surface(cos(u) sin(v),sin(u) sin(v),cos(v),u,0,2 pi,v,0,pi))
```

```
(%i28) espiral:gr3d(key="espiral",
color=red,
line_width=2,
parametric(sin(t),cos(t),t/3,t,0,15)
);
```

```
(%o28)
gr3d(key = espiral,color = red,line_width = 2,parametric(sin(t),cos(t),t/3,t,0,15))
```

```
(%i33) draw(espiral, esfera);
```

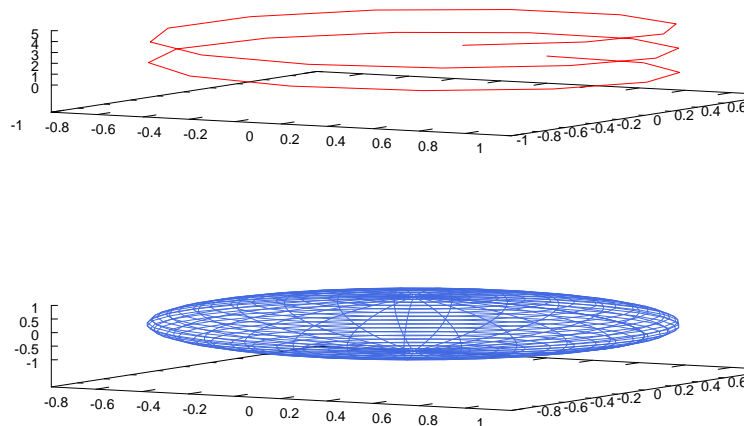


Figura 3.6: Espiral y esfera usando `draw` y `gr3d`

3.3. Un ejercicio resuelto con MÁXIMA

Ejemplo 3.3.1

Halla el área del recinto limitado por las curvas de ecuaciones $y = x^3 - 12x$ e $y = x^2$.

Para resolver este problema es necesario encontrar los puntos de corte de ambas gráficas y hacer una representación gráfica de las funciones para saber qué áreas estamos calculando:

```
(%i34) solve(x^3-12*x=x^2,x);
```

```
(%o34) [x = -3, x = 4, x = 0]
```

```
(%i42) draw2d(
```

```
filled_func=x^2,
```

```
fill_color=navy,
```

```
explicit(x^3-12*x,x,-3,4));
```

```
(%o42) [gr2d(explicit)]
```

Fíjate que con `draw2d` se consigue dibujar las dos gráficas y rellenar el espacio que queda entre ambas haciendo uso de la opción `filled_func`. Es fácil ver que la cúbica va por encima, si no tendríamos que haber dibujado las dos funciones antes de hacer el gráfico del relleno.

Por la interpretación geométrica de la integral sabemos que el área es:

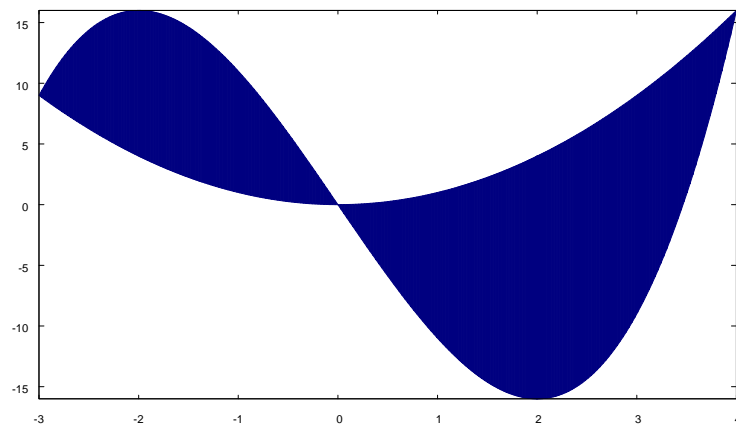


Figura 3.7: Gráfica de las funciones $f(x) = x^3 - 12x$ y $g(x) = x^2$ en el intervalo $[-3, 4]$

$$A = \int_{-3}^4 |x^3 - 12x - x^2| dx = \int_{-3}^0 x^3 - 12x - x^2 dx + \int_0^4 x^2 - (x^3 - 12x) dx$$

Así que la calculamos con MÁXIMA mediante las órdenes:

```
(%i44) cubica:x^3-12*x;
cuadratica:x^2;
```

```
(%o45) 
$$x^3 - 12xx^2$$

```

```
(%i46) integrate(cubica-cuadratica,x,-3,0)+integrate(cuadratica-cubica,x,0,4);
```

```
(%o46) 
$$\frac{937}{12}$$

```

```
(%i47) float(%);
```

```
(%o47) 78.08333333333333
```