Prácticas de Fundamentos Matemáticos

Departamento de Matemática Aplicada

12 de abril de 2010

Índice general

Consideraciones previas

Introducción a MÁXIMA y a las prácticas

1. ¿Qué es MÁXIMA ?

MÁXIMA es un programa que permite hacer cálculos matemáticos complicados con gran rapidez. La versión 0.8.1 con la que vamos a trabajar 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 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 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.

2. Observaciones para realizar las prácticas

Cada práctica se debe entregar en un fichero diferente y a ser posible todas en el mismo disco. Los nombres de los ficheros deben ser:

- Practica1NombreYApellidosDelAlumno.wxm,
- Practica2NombreYApellidosDelAlumno.wxm,
- etcétera.

Dentro de cada fichero, cuando empieces un ejercicio se debe quedar bien claro, para eso, si vas hacer el ejercicio 1.5 tienes que indicarlo poniendo en una celda *Ejercicio 1.5*. Después marcas dicha celda y le pones un tamaño mayor de letra (esta operación la puedes realizar usando el menú de MÁXIMA , elige Format, Style, Subtitle o directamente usando la combinación de teclas *Alt-2*).

En muchos de los ejercicios que se plantean en lo que sigue aparecen los parámetros a, b, c, d, e, f, g, h. Para dichos parámetros tendrás que elegir los siguientes valores: a será la última cifra de tu DNI, b la penúltima, c la antepenúltima, etcétera. Si alguna de estas cifras es cero, cámbiala por un 1. En el disco que entregues con las prácticas debes señalar en una etiqueta tu nombre, dos apellidos, titulación y D.N.I. Además deberás entregar este cuadernillo de prácticas relleno.

3. 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:

$x+y$	$\rightarrow$	*suma*
$x-y$	$\rightarrow$	*resta*
x/y	$\rightarrow$	*división*
$x*y$	$\rightarrow$	*producto*
$x\hat{}y$ ó $x**y$	$\rightarrow$	*potencia* x^y
$x!$	$\rightarrow$	factorial de x

Ejemplo .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)
$$327339060789614187001318969682[91 digits]217256545885393053328527589376$$

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)327339060789614187001318969682759915221664204604306478948329136809613379640467455488327009232590415715088668412756007100921725654588539305332852758937 6

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 manifiesto en los ejemplos.

`float`
`bfloat`

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

(%o1) π

```
(%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) $1.26765060022822940\,10^{+30}$

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

(%o6) 1.291666666666667

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

(%o7) $3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196b0$

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

(%o8) 200

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

(%o9)
$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196b0$

```
(%i10) set_display('ascii)$
(%i11) bfloat(%pi);
```

(%o11)3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196b0

4. Funciones matemáticas de interés

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

funciones matemáticas

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

Funciones relativas al cálculo con números naturales

cálculo con naturales

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

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:

- `rectform`(z) nos da la forma binómica del número complejo z.
- `realpart`(z) devuelve la parte real de z.
- `imagpart`(z) devuelve la parte imaginaria del número complejo z.
- `polarform`(z) nos da la forma polar del número complejo z.
- `conjugate`(z) = devuelve el conjugado del número z
- `abs`(z) nos da el módulo de z.

cálculo con complejos

Constantes matemáticas

`%pi`	=	$\pi \simeq 3.14159$
`%e`	=	$e \simeq 2.71828$
`inf`	=	∞
`%i`	=	$i = \sqrt{-1}$
`%phi`	=	$\phi = \dfrac{1+\sqrt{5}}{2}$ (número de oro)

`%phi`
`%e`
`%inf`
`%i`

5. Definición de variables y funciones

5.1. 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(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

`kill`
`remvalue`
`values`

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

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

`remfunction`

6. 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 (sen 20°) tendríamos que calcular primero sen 20°, 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.

7. 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
??

8. 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 $x\hat{}2+2*x-7=0$. 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`. La instrucción `apropos(solve)` hará que se muestren las variables optativas que de algún modo afectan al comportamiento de solve. Se podrá luego utilizar la función `describe` para aquellas variables cuyo objeto no esté claro.

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

$$(\%o27) \qquad [x = 3, x = 1, x = 4]$$

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

$$(\%o28) \qquad [1, 2, 3]$$

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

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

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) $$[[z=-5, y=-\frac{3}{2}, x=\frac{7}{2}]]$$

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

(%o30) $$[[x=\frac{7}{2}, y=-\frac{3}{2}, z=-5]]$$

Ejercicio 1. Factoriza el polinomio

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

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).

9. 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

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

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.

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

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

$$(\%o36) \qquad 1$$

Así que, el límite $\lim_{x\to 0} \frac{\operatorname{sen}^3(x^2)}{1-\cos(x^3)}$ 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);
```

$$(\%o42) \qquad 2$$

Otros ejemplos fáciles de entender con la explicación del comando:

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

$$(\%o37) \qquad 0$$

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

$$(\%o38) \qquad infinity$$

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

$$(\%o39) \qquad \infty$$

```
(%i40) limit(1/x,x,0,minus);
```

$$(\%o40) \qquad -\infty$$

10. 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 11.

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

- `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>

`for`

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

`while`
`unless`

```
(%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 $2\,\pi$ 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 0*done* 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 `simpsum` valga true, se aplicarán reglas adicionales.

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

(%o17) 385

Ejercicio 2. 1. Calcular la suma 7+9+11+13+...+131 = $\square$.

2. Calcula, mediante un bucle for, la suma siguiente: $\sum_{n=0}^{50} \frac{1}{n^{a+b+1}} = \square$

11. La sucesión de Fibonacci y el número de oro

En matemáticas, la sucesión de Fibonacci es la siguiente sucesión infinita de números naturales:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 \ldots$$

11.1. Historia de la sucesión

Antes de que Fibonacci escribiera su trabajo, la sucesión de los números de Fibonacci había sido descubierta por matemáticos indios tales como Gopala (antes de 1135) y Hemachandra (c. 1150), quienes habían investigado los patrones rítmicos que se formaban con sílabas o notas de uno o dos pulsos. El número de tales ritmos (teniendo juntos una cantidad n de pulsos) era f_{n+1}, que produce explícitamente los números 1, 2, 3, 5, 8, 13, 21, etc.

La sucesión fue descrita por Fibonacci como la solución a un problema de la cría de conejos: "Cierto hombre tenía una pareja de conejos juntos en un lugar cerrado y uno desea saber cuántos son creados a partir de este par en un año cuando es su naturaleza parir otro par en un simple mes, y en el segundo mes los nacidos parir también".

De esta manera Fibonacci presentó la sucesión en su libro Liber Abaci, publicado en 1202. Muchas propiedades de la sucesión de Fibonacci fueron descubiertas por Édouard Lucas, responsable de haberla denominado como se la conoce en la actualidad.

También Kepler describió los números de Fibonacci, y el matemático escocés Robert Simson descubrió en 1753 que el cociente entre dos números de Fibonacci sucesivos se acerca al número de oro. Esta serie ha tenido popularidad en el siglo XX especialmente en el ámbito musical, en el que compositores con tanto renombre como Béla Bartók, Olivier Messiaen u Delia Derbyshire la han utilizado para la creación de acordes y de nuevas estructuras de frases musicales.

11.2. Algunas propiedades para verificarlas con MÁXIMA

Para verificar con MÁXIMA que se cumplen las propiedades que vamos a ir enunciando empezaremos por el siguiente ejercicio.

Ejercicio 3. Define una función llamada `fibonacci` que dependa de un argumento (número real) de manera que al ejecutar `fibonacci`(n) devuelva el término n-ésimo de la sucesión de Fibonacci.

Propiedades

1. La razón o cociente entre un término y el inmediatamente anterior varía continuamente, pero se estabiliza en el número áureo. Es decir:

$$\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \varphi,$$

donde φ es el número de oro, es decir $\frac{1+\sqrt{5}}{2}$ (una de las soluciones de $x^2-x-1=0$)

Ejercicio 4. *a)* Calcula los cocientes f_{100}/f_{99}, f_{100}/f_{99}, f_{1000}/f_{999}, f_{10000}/f_{9999} y la diferencia que existe con el número áureo.

b) Calcula un número n para el que f_n/f_{n-1} difiera del número áureo menos que 10^{-20}.

c) Calcula con la función `limit` el límite de la sucesión f_n/f_{n-1}.

2. La sucesión puede expresarse mediante otra fórmula explícita llamada forma de Binet (de Jacques Binet). Si $\alpha = \frac{1+\sqrt{5}}{2}$ y $\beta = \frac{1-\sqrt{5}}{2}$, entonces

$$f_n = \frac{\alpha^n - \beta^n}{\alpha - \beta} y f_n \approx \frac{\alpha^n}{\sqrt{5}}$$

Ejercicio 5.

Comprueba mediante un bucle `for` que se verifica la igualdad anterior para valores de n entre 1 y 50.

Dar el error que se comete en la aproximación para $n = 10000$.

3. Cada número de Fibonacci es el promedio del término que se encuentra dos posiciones antes y el término que se encuentra una posición después. Es decir

$$f_n = \frac{f_{n-2} + f_{n+1}}{2}$$

Ejercicio 6.

Comprueba que se verifica la igualdad anterior para $n = 10000$.

4. Lo anterior también puede expresarse así: calcular el siguiente número a uno dado es 2 veces éste número menos el número 2 posiciones más atrás: $f_{n+1} = 2f_n - f_{n-2}$

5. La suma de los n primeros números es igual al número que ocupa la posición $n+2$ menos uno. Es decir

$$f_0 + f_1 + f_2 + \cdots + f_n = f_{n+2} - 1$$

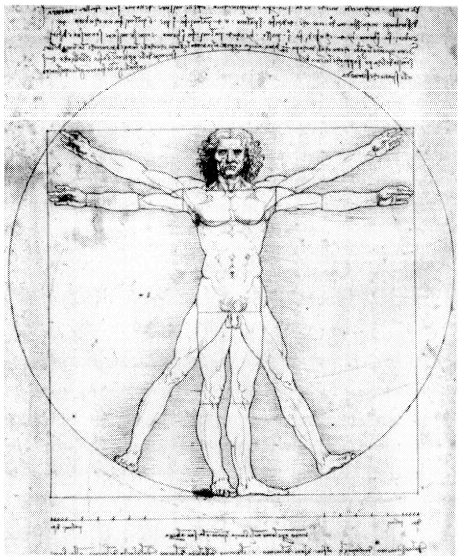
Ejercicio 7.

Comprueba que se verifica la igualdad anterior para $n \in \{1, 2, \ldots, 50\}$.

A ti, maravillosa disciplina,
media, extrema razón de hermosura
que claramente acata la clausura
viva en la malla de tu ley divina.

A ti, cárcel feliz de la retina,
áurea sección, celeste cuadratura,
misteriosa fontana de mesura
que el universo armónico origina.

A ti, mar de los sueños angulares,
flor de las cinco formas regulares,
dodecaedro azul, arco sonoro.
Luces por alas un compás ardiente.
Tu canto es una esfera transparente.
A ti, divina proporción de oro.



Rafael Alberti

12. 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 formula 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)
$$\frac{1}{x+b}-(x-a)^3+(x+3)^5+\frac{1}{\left(x-\frac{1}{4}\right)^5}$$

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

(%o133)
$$\frac{1}{x^5-\frac{5\,x^4}{4}+\frac{5\,x^3}{8}-\frac{5\,x^2}{32}+\frac{5\,x}{256}-\frac{1}{1024}}+\frac{1}{x+b}+x^5+15\,x^4+89\,x^3+3\,a\,x^2+270\,x^2-3\,a^2\,x+405\,x+a^3+243$$

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

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

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

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

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

$$(\%o141) \qquad false$$

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

$$(\%o142) \qquad 2+3+5$$

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

$$(\%o143) \qquad true$$

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

$$(\%o144) \qquad 10$$

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

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

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

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

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

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

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

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

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

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

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

$$(\%o152) \qquad x^a-1$$

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.

`logexpand`

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

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

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

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

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

$$(\%o163) \qquad false$$

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

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

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

$$(\%o169) \qquad true$$

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

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

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

$$(\%o171) \qquad super$$

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

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

`logcontract` es una función que permite compactar expresiones logarítmicas

logcontract

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

$$(\%o174) \qquad 2\,\left(2\,a\,log\,(y) + a\,log\,(x)\right)$$

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

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

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

$$(\%o176) \qquad a\,log\left(x^2\,y^4\right)$$

Expresiones trigonométricas

Maxima 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:

- `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`.

trigreduce

- `trigsimp`(expresion) simplifica las funciones trigonométricas e hiperbólicas del argumento

trigsimp

- `trigreduce`(expresion) simplifica las funciones trigonométricas e hiperbólicas

trigreduce

Práctica 2

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, $A.A = A\hat{}\hat{}\ 2$ y $A\hat{}\hat{}-1$ es la **inversa** de A, si existe. Algunas variables controlan la simplifcación de expresiones que incluyan

[1] Mucho 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

estas operaciones: `doallmxops`, `domxexpt`, `domxmxops`, `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:

- `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 vectorios propios de la matriz asociados a los valores propios correspondientes.
- `determinant` (A): nos de 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 span$(v_1, ..., v_n)$, siendo $v_1, ..., v_n$ la base del espacio solución de A x= $\mathbf{0}$. Si el resultado contiene sólo a $\mathbf{0}$, devuelve span().

`eigenvalues`
`eigenvector`

`determinant`
`charpoly`
`addcol`
`addrow`
`copymatrix`
`nullspace`
`transpose`

- `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.

`echelon`
`rank`
`diagmatrix`
`invert`
`zeromatrix`

Vamos a resolver unos problemas de álgebra lineal usando MÁXIMA .

Ejemplo 2.1. Sea $f : \mathbb{R}^3 \to \mathbb{R}^3$ una aplicación lineal tal que $M_{\beta_c^3\beta_c^3}(f) = \begin{pmatrix} 3 & 2 & 1 \\ 4 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix}$ y considera la base $\beta = \{(1,1,1),(1,1,0),(1,0,0)\}$. Responde a las siguientes cuestiones:

1. Calcula $M_{\beta\beta}(f)$ *(0.5 puntos)*

   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);
```

$$(\%o2) \qquad 2$$

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

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

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

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

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

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

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

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

2. Calcula las ecuaciones de Ker f respecto de la base β y da una base de Ker f expresando las coordenadas de los vectores que aparezcan en la base β *(0.5 puntos)*

Solución.

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

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

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

Las órdenes realizadas en MÁXIMA han sido las que siguen:

```
(%i9) nullspace(mbbf);
```

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

3. Calcula las ecuaciones de Im f respecto de la base β_c^3 y da una base de Im f expresando las coordenadas de los vectores que aparezcan en la base β_c^3 *(0.5 puntos)*

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 Im f se tiene que 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) $$2\ (2\,z - y) - 3\ (z - x) + y - 2\,x$$

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

$$(\%o12) \qquad z - y + x$$

4. Encuentra bases β_1 y β_2 tales que $M_{\beta_1\beta_2}(f) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ *(0.5 puntos).*

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);
```

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

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

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

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

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

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

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

Ejemplo 2.2. 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]);
```

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

- Calculamos el polinomio característico

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

$$\text{(\%o110)} \qquad \begin{array}{c} (((-x-19)\,(5-x)+80)\,(19-x)-12 \\ (120-30\,(5-x))+6\,(-12\,(-x-19)-240))\,(3-x) \end{array}$$

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

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

$$\text{(\%o111)} \qquad x^4 - 8\,x^3 + 22\,x^2 - 24\,x + 9$$

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

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

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

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

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

(%o113) $[2, 2]$

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

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

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

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

- Calculamos la inversa de p:

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

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

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

2.2. Ejercicio para resolver con MÁXIMA

Ejercicio 8.

En este ejercicio consideraremos las siguientes bases de $\mathbb{R}^3$:

$$\beta_P = \{(1, b, c), (0, 1, a), (0, 0, 1)\},$$

$$\beta_Q = \{(1, 0, 1), (0, 1, 1), (1, 0, 0)\},$$

$$\beta_R = \{(1, 0, 2), (0, 1, 0), (1, 0, 1)\}$$

y la aplicación lineal $f : \mathbb{R}^3 \to \mathbb{R}^3$ definida por:

$$M_{\beta_P\beta_P}(f) = \begin{pmatrix} 1 & 0 & -ab+c \\ 0 & 1 & a \\ 0 & 0 & 0 \end{pmatrix}$$

Indicación a tener en cuenta.

Define las matrices siguientes:

$$P = \begin{pmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ c & a & 1 \end{pmatrix}, \quad Q = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}, A = M_{\beta_P\beta_P}(f)$$

y ten en cuenta que $P = M_{\beta_c^3\beta_P}$, $Q = M_{\beta_c^3\beta_Q}$,$R = M_{\beta_c^3\beta_R}$.

1. Calcula las siguientes matrices:

$$M_{\beta_P\beta_Q}, M_{\beta_P\beta_R}, M_{\beta_R\beta_P}, M_{\beta_Q\beta_P}.$$

Indicación a tener en cuenta.

$$M_{\beta_P\beta_Q} = M_{\beta_P\beta_c^3}M_{\beta_c^3\beta_Q}; \quad M_{\beta_P\beta_R} = M_{\beta_P\beta_c^3}M_{\beta_c^3\beta_R};$$
$$M_{\beta_R\beta_P} = M_{\beta_R\beta_c^3}M_{\beta_c^3\beta_P}; \quad M_{\beta_Q\beta_P} = M_{\beta_Q\beta_c^3}M_{\beta_c^3\beta_P};$$

Solución.

$$M_{\beta_P\beta_Q} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}, M_{\beta_P\beta_R} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix},$$

$$M_{\beta_R\beta_P} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}, M_{\beta_Q\beta_P} = \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}.$$

2. Calcula las coordenadas de los vectores $u = (d, e, f)_{\beta_R}$, $v = (a, 2, 1)_{\beta_P}$ y $w = (0, 1, b)_{\beta_Q}$ en las bases β_R, β_P, β_Q y β_c^3.

Indicación a tener en cuenta.

Para obtener las coordenadas de u en la base β_Q debes utilizar la relación $(x, y, z)^t_{\beta_Q} = M_{\beta_Q\beta_R}(d, e, f)^t_{\beta_R}$. Relaciones similares debes utilizar para los demás cambios de base.

Solución.

$u = \left(\square, \square, \square\right)_{\beta_c^3}$ $v = \left(\square, \square, \square\right)_{\beta_c^3}$

$w = \left(\square, \square, \square\right)_{\beta_c^3}$

$u = \left(\square, \square, \square\right)_{\beta_P}$ $v = \left(\square, \square, \square\right)_{\beta_P}$

$w = \left(\square, \square, \square\right)_{\beta_P}$

$u = \left(\square, \square, \square\right)_{\beta_Q}$ $v = \left(\square, \square, \square\right)_{\beta_Q}$

$w = \left(\square, \square, \square\right)_{\beta_Q}$

$u = \left(\square, \square, \square\right)_{\beta_R}$ $v = \left(\square, \square, \square\right)_{\beta_R}$

$w = \left(\square, \square, \square\right)_{\beta_R}$

3. Calcula $M_{\beta_Q\beta_Q}(f)$, $M_{\beta_P\beta_P}(f)$, $M_{\beta_R\beta_R}(f)$, $M_{\beta_c^3\beta_c^3}(f)$ $M_{\beta_P\beta_Q}(f)$, $M_{\beta_Q\beta_P}(f)$, $M_{\beta_P\beta_R}(f)$, $M_{\beta_R\beta_P}(f)$, $M_{\beta_Q\beta_R}(f)$, $M_{\beta_R\beta_Q}(f)$.

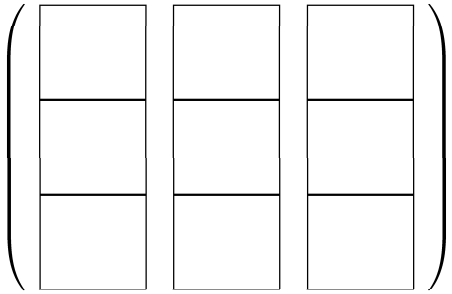
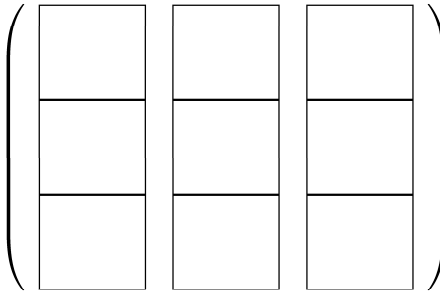
Indicación a tener en cuenta.

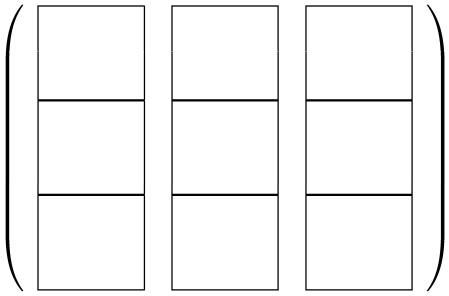
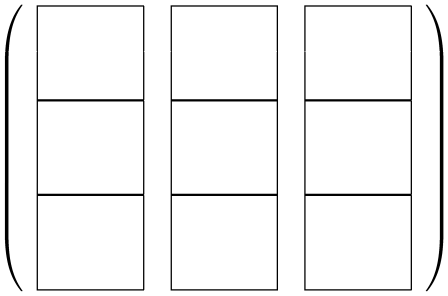
Para uno de los casos te será útil la fórmula

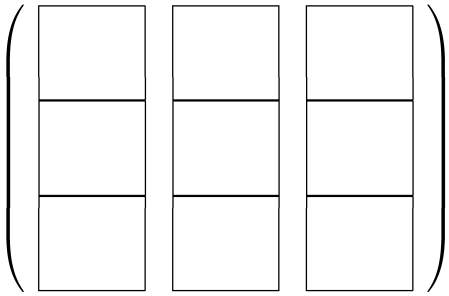
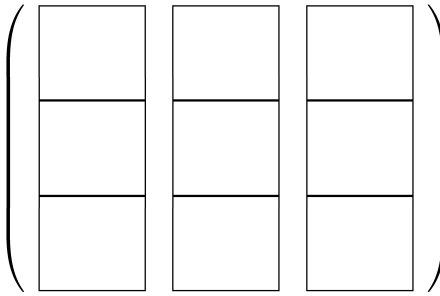
$$M_{\beta_Q\beta_Q}(f) = M_{\beta_Q\beta_P} M_{\beta_P\beta_P}(f) M_{\beta_P\beta_Q}.$$

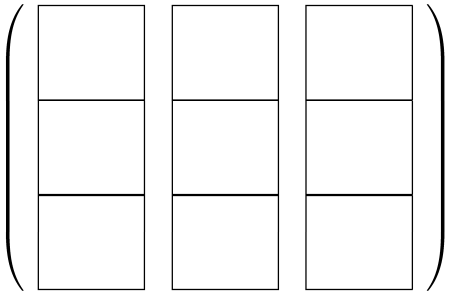
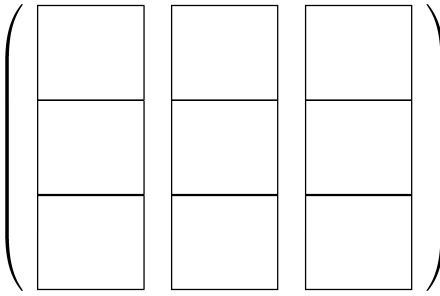
Debes adaptar esta igualdad para los demás casos.

Solución.

$M_{\beta_Q\beta_Q}(f) =$  , $M_{\beta_P\beta_P}(f) =$ 

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

$M_{\beta_P\beta_Q}(f) =$  , $M_{\beta_Q\beta_P}(f) =$ 

$M_{\beta_P\beta_R}(f) =$  , $M_{\beta_R\beta_P}(f) =$ 

$$M_{\beta_Q\beta_R}(f) = \left(\begin{array}{ccc} & & \\ & & \\ & & \end{array}\right), M_{\beta_R\beta_Q}(f) = \left(\begin{array}{ccc} & & \\ & & \\ & & \end{array}\right)$$

4. Usa la función de MÁXIMA Solve para encontrar una base de Ker f y exprésala en todas las bases manejadas en este ejercicio.

Solución.

$$\beta_{\operatorname{Ker} f} = \left\{ \left(\quad , \quad , \quad \right)_{\beta_P} \right\} = \left\{ \left(\quad , \quad , \quad \right)_{\beta_Q} \right\}$$
$$= \left\{ \left(\quad , \quad , \quad \right)_{\beta_R} \right\} = \left\{ \left(\quad , \quad , \quad \right)_{\beta_c^3} \right\}$$

5. Calcula el polinomio $p_A(x)$.

Solución.

$$p_A(x) = \boxed{}$$

6. Calcula σ_A. A partir de ahora l y m denotarán a los dos valores propios de A.

Solución.

$$\sigma_A = \left\{ l = \boxed{}, m = \boxed{} \right\}$$

7. Da bases de V_l y de V_m (expresadas respecto de la base canónica).

Solución.

$$\beta_{V_l} = \left\{ \boxed{} \right\}$$

$$\beta_{V_m} = \left\{ \boxed{} \right\}$$

8. Calcula la matriz diagonal D y las matrices de paso, T y T^{-1}, que diagonalizan a A.

Solución.

$$D = \left(\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \right)$$

$$T = \left(\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \right) \qquad T^{-1} = \left(\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \right)$$

9. Comprueba que $D = T^{-1}AT$, para ello se piden las matrices AT y $T^{-1}AT$.

Solución.

$$AT = \left(\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \right) \qquad T^{-1}(AT) = \left(\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \right)$$

2.3. El teorema de Cayley-Hamilton

Introducimos el teorema de Cayley-Hamilton y veremos las importantes aplicaciones que tiene a lo largo de toda esta práctica.

Teorema 1 (Cayley-Hamilton)**.** Dada una matriz cuadrada A con polinomio característico $p_A(x) = a_n x^n + a_{n-1}x^{n-1} + a_2 x^2 + a_1 x + a_0$, se tiene que:

$$p_A(A) = a_n A^n + a_{n-1}A^{n-1} + a_2 A^2 + a_1 A + a_0 I_n = 0_{n\times n}.$$

Ejercicio 9. Comprueba que se verifica el teorema de Cayley-Hamilton para la matriz

$$H = \begin{pmatrix} 2+a & 3+a+b & 3+a+b+c \\ 1+a & 2+a & 2+a+c \\ 1 & 2 & 2+c \end{pmatrix}$$

siguiendo los siguientes pasos:

1. Calcula el polinomio característico $p_H(x)$ usando la función *textttdeteminant*, es decir deberás calcular textttdeteminant(H-x* `ident`(3)).
2. Define una función pH(M):=... que actuando sobre una matriz M devuelva como resultado $p_H(M)$.
3. Verifica que pH(H) devuelve la matriz nula.

Solución.

$p_H(x) =$

$pH(M) :=$

$pH(H) =$

2.4. Inversa de una matriz

Teorema 2. Una matriz A es invertible si y sólo si no tiene a 0 como valor propio.

Demostración. Veamos primero que si A es invertible entonces 0 no puede ser un valor propio. Supongamos lo contrario, es decir, 0 es un valor propio de la matriz invertible A. Esto quiere decir que:

$$0 = p_A(0) = |A - 0I_n| = |A|,$$

y esto es una contradicción porque el determinante de $|A|$ no puede ser 0 por ser A invertible.

En segundo lugar suponemos que 0 no es un valor propio de A y veremos que A es invertible. Por no ser 0 un valor propio de A se tiene que su polinomio característico debe ser de la forma:

$$p_A(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

con $a_0 \neq 0$.

Ahora usamos el teorema de Cayley-Hamilton y obtenemos:

$$\begin{aligned} 0_{n\times n} = p_A(A) &= a_n A^n + a_{n-1}A^{n-1} + \cdots + a_1 A + a_0 I_n \\ &\Rightarrow \frac{-1}{a_0}\left(a_n A^n + a_{n-1}A^{n-1} + \cdots + a_1 A\right) = I_n \\ \Rightarrow A\frac{-1}{a_0}&\left(a_n A^{n-1} + a_{n-1}A^{n-2} + \cdots + a_2 A + a_1 I_n\right) = I_n \\ \Rightarrow \frac{-1}{a_0}&\left(a_n A^{n-1} + a_{n-1}A^{n-2} + \cdots + a_2 A + a_1 I_n\right) A = I_n \end{aligned}$$

Así que hemos demostrado que la inversa de A es

$$\frac{-1}{a_0}\left(a_n A^{n-1} + a_{n-1}A^{n-2} + \cdots + a_2 A + a_1 I_n\right)$$

y por lo tanto hemos concluido la demostración.

□

2.4.1. Método para construir la inversa de una matriz

La demostración del teorema anterior nos da un método para construir la inversa de una matriz A, el método consiste en:

1. Construir el polinomio característico de la matriz A:

$$p_A(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0.$$

2. Obtener la inversa de A mediante los siguientes cálculos:

$$A^{-1} = \frac{-1}{a_0}\left(a_n A^{n-1} + a_{n-1}A^{n-2} + \cdots + a_2 A + a_1 I_n\right).$$

Ejercicio 10. Usando este método se pide calcular la inversa de las siguientes matrices:

$$A = \begin{pmatrix} 2+a & 3+a+b & 3+a+b+c & 3+a+b+c+d \\ 1+a & 2+a & 2+a+c & 2+a+c+d \\ 1 & 2 & 2+c & 2+c+d \\ 1 & 1+b & 1+b & 1+b+d \end{pmatrix}$$

$$B = \begin{pmatrix} 2+a & 1+b & c & d \\ 2 & 1+b & c & d \\ 1 & 1 & c & d \\ 0 & 0 & 0 & d \end{pmatrix} \quad C = \begin{pmatrix} 5+2a & 3+2b & 3c & 4d \\ 3+a & 2+b & 2c & 3d \\ 2 & 1+b & c & 2d \\ 1 & b & 0 & d \end{pmatrix}$$

Además se exige calcular el determinante, los polinomios característicos de cada una de las matrices y las siguientes operaciones:

- $A^2 + B(A+C)$;
- $e^{\operatorname{sen}|A|+\cos|B|+\sqrt{|A+B+C|^2+1}}$ con 4 cifras decimales.

Indicaciones a tener en cuenta.

- Recuerda que si un número de tu DNI involucrado en el ejercicio es 0 lo debes sustituir por un 1.
- Antes de calcular la inversa debes comprobar que la matriz es invertible calculando su determinante.
- Debes calcular el polinomio característico de cada una de las matrices con el comando Det de MÁXIMA . Por ejemplo el polinomio característico de la matriz A se calcula con las órdenes: Det[A-xIdentityMatrix[4]].
- Debes comprobar que el resultado obtenido con el método de Cayley-Hamilton es correcto. Para ello multiplica cada matriz por la inversa obtenida y verifica que da I_4.
- En los resultados no deben aparecer los parámetros a,b,c,d,....

Solución.

$$A = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \qquad |A| = \boxed{}$$

$$p_A(x) = \boxed{}$$

$$A^{-1} = \frac{\boxed{}}{\boxed{}} \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

$$B = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix} \qquad |B| = \boxed{}$$

$p_B(x) =$

$B^{-1} =$

$C =$ $\quad |C| =$

$p_C(x) =$

$C^{-1} =$

$$A^2 + B(A + C) = \left(\begin{array}{cccc} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{array} \right)$$

$$e^{\operatorname{sen}|A|+\cos|B|+\sqrt{|A+B+C|^2+1}} = \boxed{}$$

Práctica 3

Cálculo 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, por un lado veremos los comandos con los que MÁXIMA nos permite resolver problemas en los que interviene la integral y por otro lado veremos cómo se programan los métodos de integración numérica compuestos de Simpson y del trapecio.

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 $f(x_1, x_2, ..., x_n)$ 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`.

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

`diff`
`derivabbrev`

Por ejemplo si queremos calcular la derivada de $f(x) = \operatorname{sen} x$ escribiremos

```
(%i1) diff(sin(x),x);
```

$$(\%o1) \qquad cos\,(x)$$

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 $f(x,y) = \operatorname{sen}\,(x+y)$ debemos escribir

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

$$(\%o6) \qquad g\,(x,y) := sin\,(x+y)$$

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

$$(\%o7) \qquad cos\,(y+x)$$

La segunda derivada de $f(x) = \operatorname{sen} x$ se calcula

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

$$(\%o5) \qquad -sin\,(x)$$

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

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

$$(\%o8) \qquad -cos\,(y+x)$$

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);
```

$$(\%o10) \qquad -sin\,(y+x)$$

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 $f\,(x) = \text{sen}\,x$ procedemos del siguiente modo.

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

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

$\int_0^1 xdx$ se calcularía del modo siguiente:

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

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

Ejercicio 11. Calcular las primitivas de dos de las siguientes funciones:

(a) $f\,(x) = \cos(x)\text{sen}\,(2x)\cos(3x)\tan(cx)$

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

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

(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 $f\,(x) = \text{sen}\,x$ 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.

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

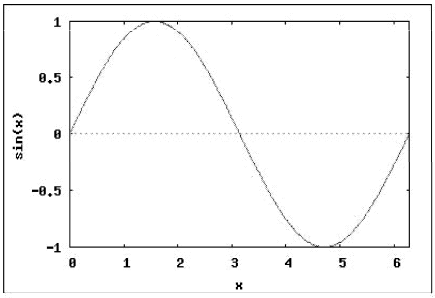


Figura 3.1: Gráfica de la función $f(x) = \operatorname{sen} x$ en el intervalo $[0, 2\pi]$

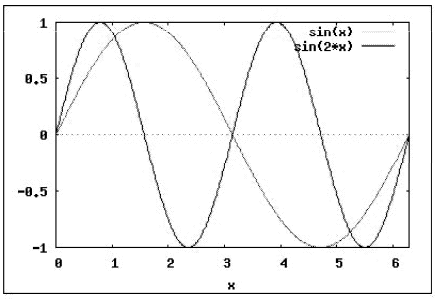


Figura 3.2: Gráficas de las funciones $f(x) = \operatorname{sen} x$ y $g(x) = \operatorname{sen}(2x)$

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

(%o4) *<< Graphics >>*

expresión que hará una representación gráfica simultánea de las funciones $\operatorname{sen} x$ y $\operatorname{sen} 2x$.

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.
- **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.

`set_plot_op`

La variable `plot_options` consta de un 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 aquélla. 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:

`plot_option`

- **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 $f(x, y) = \text{sen}\,(xy)$ en el dominio $[0, 3] \times [0, 3]$ hemos de escribir

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

(%o13) *<< Graphics >>*

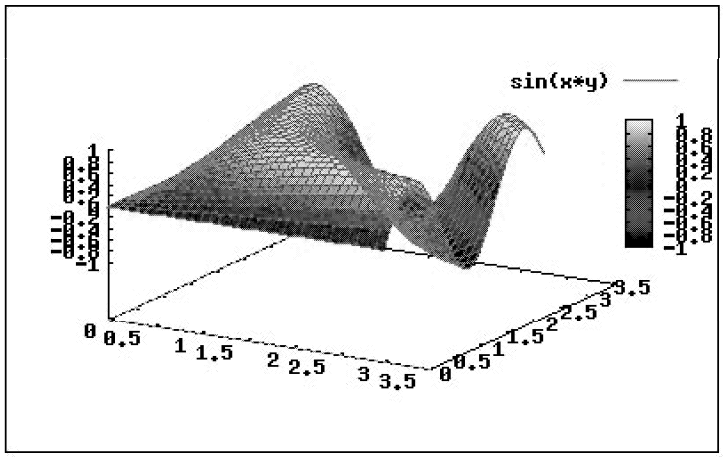


Figura 3.3: Gráfica de $\operatorname{sen}(xy)$

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

$$\begin{cases} x(t) = \operatorname{sen} t, \\ y(t) = \operatorname{sen} 2t, \end{cases}$$

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 la sentencia `load(draw)`. Este módulo ofrece, entre otras, las funciones `draw2d`, `draw3d` y

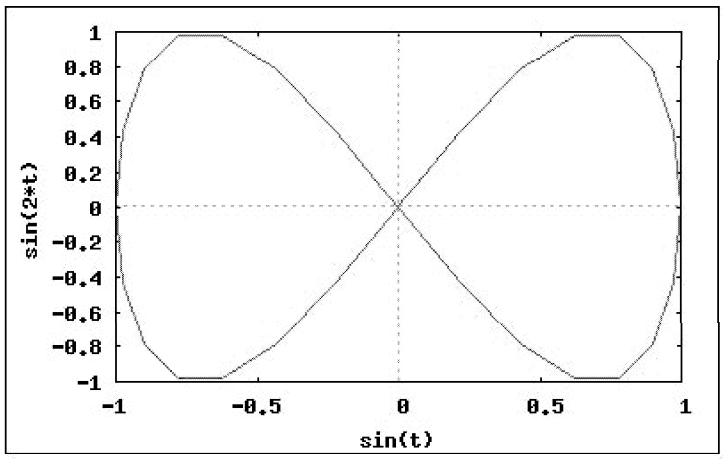


Figura 3.4: Gráfica de la curva $(\operatorname{sen}(t), \operatorname{sen}(2t))$

`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

`gr2d`
`gr3d`

`draw`
`draw2d`
`draw3d`

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

El gráfico de salida será:

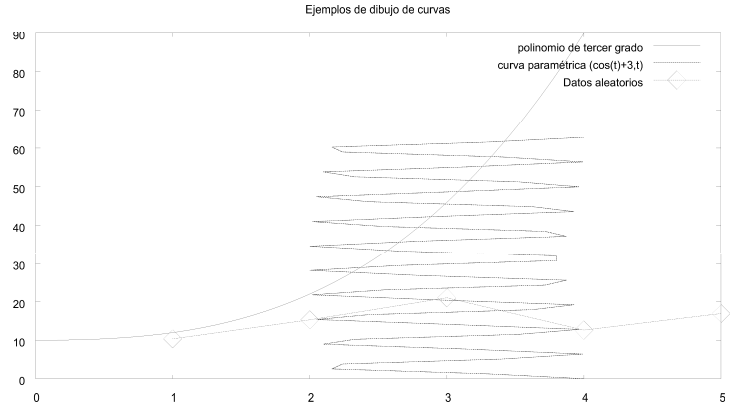


Figura 3.5: Salida del uso del comando `draw2d`

Por ejemplo, para representar la curva $\begin{cases} x(t) = \operatorname{sen} t \\ y(t) = \cos t \\ z(t) = t/3 \end{cases}$ 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)
);
```

$(\%o32)$
$gr3d(key = esfera, color = royalblue, surface\_hide = false, parametric\_surface(cos(u)\ sin(v)$

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

(%o28)

$$gr3d\left(key = espiral, color = red, line\_width = 2, parametric\left(sin\left(t\right), cos\left(t\right), \frac{t}{3}, t, 0, 15\right)\right)$$

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

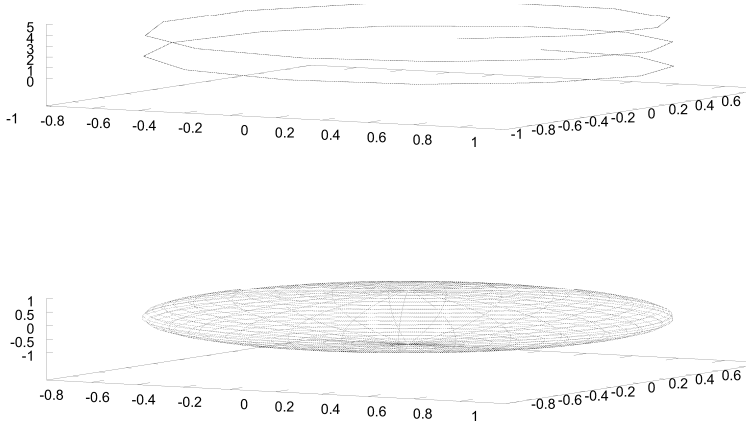


Figura 3.6: Espiral y esfera usando `draw` y `gr3d`

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.

Así que el área que hay que calcular es la que hemos sombreado en la figura **??**. Por la interpretación geométrica de la integral sabemos que el área es:

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

Así que la calculamos con MÁXIMA mediante las órdenes:

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

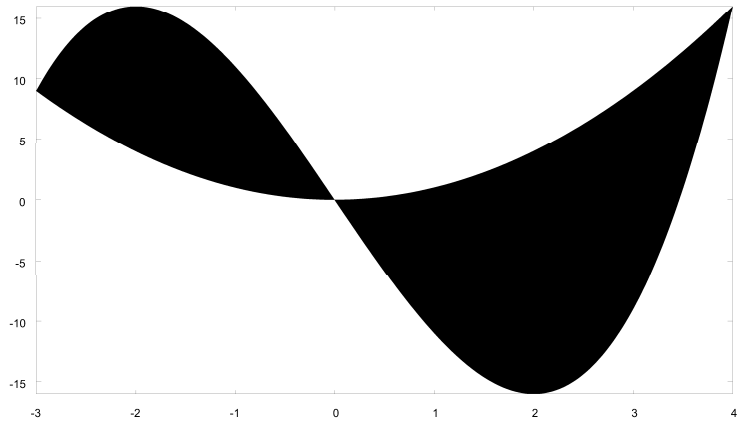


Figura 3.7: Gráfica de las funciones $f(x) = x^3 - 12x$ y $g(x) = x^2$ en el intervalo $[-3, 4]$

(%o45) $$x^3 - 12\,x x^2$$

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

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

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

(%o47) $$78.08333333333333$$

3.4. Ejercicios propuestos relativos a la interpretación geométrica de la integral

Ejercicio 12. Calcula el área comprendida entre las gráficas de las funciones $r(x) = 1/(a+b+c+d+4)^2(-1)^b(x-a)(x-a-b-1)(x-a-b-c-2)(x-a-b-c-d-3)+(a+b+c+d+10)$
y
$s(x) = 1/(a+b+c+d+4)^2(-1)^(b+6a+1)(x-a-1)(x-a-b-2)(x-a-b-c-3)(x-a-b-c-d-4)+(a+b+c+d+10)$

Para ello se pide dar los siguientes pasos:

1. Define la función r(x):=1/(a+b+c+d+4)2(-1)b (x-a).....
2. Define la función s(x):=1/(a+b+c+d+4)2(-1)(b+6a+1) .....
3. Encuentra las soluciones de la ecuación $r(x) = s(x)$.
4. Dibuja las gráficas de r y s en un rango en el que se vea la región comprendida entre ambas.
5. Finalmente escribe aquí la solución:

Solución.

$$A = \boxed{}$$

Ejercicio 13. Calcula el volumen del cuerpo generado al girar la región anterior alrededor del eje OX.

Solución.

$$V_{OX} = \boxed{}$$

Ejercicio 14. Calcula la superficie del cuerpo generado al girar la región anterior alrededor del eje OX.

Solución.

$$A_{OX} = \boxed{}$$

3.5. Reglas del trapecio y de Simpson

3.5.1. Desarrollo teórico

A pesar de que la regla de Barrow permite calcular el valor de integrales definidas de funciones que admiten primitivas, no siempre es fácil calcular dicha primitiva. Por ello es necesario disponer de métodos alternativos para calcular el valor de las integrales, la alternativa la dan los métodos numéricos. Estudiaremos dos de ellos: *la regla del trapecio* y *el método de Simpson*

La regla del trapecio Dada una función $f : [a, b] \to \mathbb{R}$ integrable, la regla del trapecio consiste en aproximar la integral definida $\int_a^b f(x)dx$ por $\int_a^b P_1(x)dx$, donde $P_1(x)$ es el único polinomio de grado 1 (recta) que pasa por los puntos $(a, f(a))$ y $(b, f(b))$. Así quc:

$$\int_a^b f(x)dx \approx \int_a^b P_1(x)dx = \frac{b-a}{2}\left(f(a) + f(b)\right)$$

y el error que cometemos en dicha aproximación, si la función f es de clase C^2, es:

$$E = -\frac{1}{12}(b-a)^3 f''(c),$$

donde c es un punto del intervalo (a, b).

El error anterior puede reducirse si utilizamos la *regla del trapecio compuesta*, ésta consiste en dividir el intervalo $[a, b]$ en n subintervalos de longitud $h = \frac{b-a}{n}$ y aplicar la regla del trapecio simple a cada uno de los intervalos $[a + jh, a + (j + 1)h]$ con $j \in \{0, 1, \ldots, n - 1\}$. Este método proporciona la aproximación:

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

para esta aproximación el error que se comete, si f es de clase C^2, es:

$$E = -\frac{1}{12}(b-a)h^2 f''(c),$$

siendo c un punto del intervalo (a, b).

La regla de Simpson. La idea de esta regla es aproximar la función $f : [a, b] \to \mathbb{R}$ a integrar por el polinomio de grado 2 (único) que pasa por los puntos $(a, f(a))$, $(\frac{a+b}{2}, f(\frac{a+b}{2}))$ y $(b, f(b))$. De esta manera, se obtiene la aproximación:

$$\int_a^b f(x)dx \approx \int_a^b P_2(x)dx = \frac{b-a}{6}\left(f(a) + 4f(\frac{a+b}{2}) + f(b)\right).$$

Además, si la función es de clase C^4, existe $c \in (a, b)$ tal que, el error que se comete en la aproximación es:

$$E = -\frac{1}{2880}(b-a)^5 f^{(iv)}(c).$$

Al igual que en la regla del trapecio, si subdividimos el intervalo $[a, b]$ en n partes **(con n número par)** y aplicamos a cada una de ellas la regla de Simpson, se obtiene una mejor aproximación de la integral:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(a)+2\sum_{i=2}^{n/2} f(a+2(i-1)h) +4\sum_{i=1}^{n/2} f\left(a+(2i-1)h\right)+f(b)\right),$$

con un error, si f es de clase C^4, dado por:

$$E=-\frac{1}{180}(b-a)h^4 f^{(iv)}(c),$$

estando c en (a,b).

3.5.2. Regla del trapecio. Construcción

Recordamos la regla del trapecio para calcular un valor aproximado de la integral de una función $f:[a,b]\to\mathbb{R}$ haciendo n particiones de dicho intervalo. Definiendo h como $(b-a)/n$, dicha regla se basa en aproximar el valor de la integral $\int_a^b f(x)dx$ por la suma:

$$\frac{h}{2}f(a)+h\sum_{i=1}^{n-1} f(a+ih)+\frac{h}{2}f(b).$$

Por fin damos la implementación de la regla del trapecio para un número n de particiones en el intervalo $[a,b]$. A la función IntegraTrapecio se le pasa como argumento la función que queremos integrar, el extremo inferior del intervalo de integración, el extremo superior y el número de particiones que haremos en el intervalo. El resultado que da la función es un conjunto de 4 números, el primero es el valor exacto de la integral calculado con el comando Integrate, el segundo es el valor calculado por el método del trapecio, el tercero es el error y el cuarto el número de particiones que se han hecho en el intervalo. Conviene que sepas que la orden ev(f,x=a), que se usa a continuación, hace que MÁXIMA evalúe la función f en el punto a

```
(%i3) declare([integral,h,valorexacto,errora,valorexacto],real);
```

(%o3) *done*

```
(%i4) kill(integral,h,valorexacto,errora,valorexacto);
```

(%o4) *done*

```
(%i5) integral:0;
h:0;
valorexacto:0;
errora:0;
valorexacto:0;
```

(%o9) 00000

```
(%i14) IntegraTrapecio(f,a,b,n):=(
kill(errora,integral,h,valorexacto),
valorexacto:integrate(f,x,a,b),
h:(b-a)/n,
integral:h/2*(ev(f,x=a)+ev(f,x=b)),
for j:1 thru n-1 do integral:integral+h*ev(f,x=a+j*h),
valorexacto:integrate(f,x,a,b),
errora:valorexacto-integral,
print("exacto=",float(valorexacto),". Trapecio=",float(integral),
". Error=",float(errora),". n=",n)
);
```

```
(%i19) IntegraTrapecio(sin(x),0,1,1000);
```

(%o19)
$exacto = 0.45969769413186.Trapecio = 0.45969765582372.Error = 3.8308140889759413\,10^{-8}.n = 1$

Ejemplo 3.1. IntegraTrapecio[sen [x],0,π, 5]

{{2.,1.93377,0.0662344,5}}

3.5.3. Regla de Simpson. Construccción

Ejercicio 15. Construye la función IntegraSimpson similar a IntegraTrapecio pero que devuelva el valor aproximado de la integral calculado con la regla de Simpson. Ten en cuenta que el método de Simpson sólo se puede aplicar haciendo un número par de subdivisiones del intervalo. Constata que la aplicación de dicho método a la función seno en el intervalo [0,Pi] da como resultado el conjunto de valores {{2., 2.00456, 0.00455975, 4}}.

Ahora se pide que des los siguientes resultados:

1. IntegraSimpson(sin(a x),0, %pi,10)= .

2. IntegraSimpson(cos(b x),0, %pi,10)= .

3. IntegraSimpson(exp(c x),0, %pi,10)= .

Índice alfabético