Introducción a wxMaxima

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

José Manuel Cascón
Facultad de Economía y Empresa
2013

Índice general

Capítulo 1

Introducción

1.1. Sobre este manual

`Maxima` es un programa que realiza cálculos matemáticos de forma tanto simbólica como numérica. Se encuentra disponible bajo licencia GNU GPL, y puede descargarse desde

http://maxima.sourceforge.net/es/

El objetivo de esta guía es iniciar al alumno en el manejo de `Maxima` orientado a las matématicas de un primer curso de grado en Economía o Administración de Empresas. El contenido de este documento debe servir como apoyo y complemento a la asignaturas de AÍgebra y Análisis Matemático.

1.2. Descarga e instalación

El programa puede descargarse desde

http://sourceforge.net/projects/maxima/files/.

Selecciona tu sistema operativo, descarga y ejecuta la versión correspondiente. De este modo, se instalará el entorno *wxMaxima*, que es una interfaz gráfica que facilita el uso de *Maxima*.

1.3. Inicio

La información en *Maxima* se organiza en celdas. Se debe terner en cuenta:

- Las celdas de entrada de datos quedarán numeradas ($\%in$). Donde n especifica la entrada n-ésima. Este número es automáticamente generado por el sistema. En este documento reprentaremos la celda de entrada activa como:

  ```
  (%i)
  ```

- Las celdas de salida de datos son numeradas con ($\%on$). La celda de salida activa se representará:

  ```
  (%o)
  ```

- Para ejecutar una instrucción se debe escribir seguida de ";" y pulsar la combinación **Mayúscula**+**Enter**. La versión del interfaz de usuario que utilizamos añade el ";" sin necesidad de escribirlo.

- A la ejecución de una celda de *Maxima* corresponde una celda de salida donde se muestra el resultado. En ocasiones, puede resultar conveniente omitir esa salida. Esto se consigue situando al final de la celda el símbolo $.
- *Maxima* distingue entre mayúsculas y minúsculas.
- Para realizar una **asignanción estática** en *Maxima* se escribe:

  ```
  (%i) a: 2;
  ```

  Esta instrucción asigna a la la variable a el valor 2.
- La **asignación dinámica** permite definir funciones (o sucesiones) dependientes de uno o varios parámetros.

  ```
  (%i) f(x):= x^2+2;
  ```

  En esta ocasión *Maxima* entide que se define una función, cuyo nombre es f de una única variable. En cada llamada a $f(a)$, *Maxima* se remitira a la regla y sustituira el valor de x por el valor del argumento, en este caso a.
- En ocasiones es conveniente liberar la memoria de *Maxima*. Para ello se utiliza el comando `kill()`. Si se desea elimiar todo el contenido de la memoria se evalua:

  ```
  (%i) kill(all);
  ```

  Si por el contrario se desea borrar el contenido de una variable particular, por ejemplo la variable a:

  ```
  (%i) kill(a);
  ```
- Los archivos de *Maxima* tiene extensión *.wxm. Para guardar un documento en *Maxima* se selecciona el menú archivo y dentro de él la opción *Save*. En el documento quedarán grabadas todas las entradas que se hayan realizado.

1.4. Operaciones básicas

A continuación presentamos los símbolos asociados a las operaciones básicas y funciones usuales:

$+$	suma
$*$	producto
$/$	división
$\hat{}$	potencia
sqrt()	raíz cuadrada
exp()	exponencial
sin(), cos(), tan()	funciones trigonométricas

Por defecto *Maxima* trabaja en forma simbólida, si queremos obtener una aproximación numérica se debe usar la función *float()*. Por ejemplo, para obtener una aproximación numérica de $\sqrt{5}$:

```
(%i) float(sqrt(5));
(%o) 2.23606797749979
```

Maxima incorpora constantes matemáticas. Algunas de ellas son:

`%pi`	número π
`%e`	número e
`%i`	unidad imaginaria
`%phi`	razón áurea $\frac{1+\sqrt{5}}{2}$

1.5. Ayuda

El manual del programa es accesible en la web:

http://maxima.sourceforge.net/es/documentation.html

En la misma página también es posible encontrar otros manuales más básicos.

El entorno de *wxMaxima* permite acceder al manual a través del botón *Help/Ayuda*. Además también es posible acceder a la ayuda desde la ventana de comandos. Algunos de los más útiles son:

`?? expr`	comandos que contiene `expr`
`describe(expr)`	ayuda sobre `expr`
`example(expr)`	ejemplo de `expr`

Capítulo 2

Álgebra

2.1. Vectores y Matrices

Maxima permite trabajar con vectores y matrices. Para introducir un **vector**, los elementos se separan por comas, y se delimitan por corchetes, por ejemplo:

```
(%i) v1:[1,-7.3];
(%i) v2:[2,0,3];
```

Los vectores pueden sumarse $(+)$, restarse $(-)$, multiplicarse término a término $(*)$, o multiplicarae escasamente $(\cdot)$:

```
(%i) v1 +v2;
(%o) [3,-7,6]

(%i) v1 *v2;
(%o) [2,0,9]

(%i) v1 .v2;
(%o) 11
```

Para introducir una **matriz** se utiliza el comando `matrix`. Las filas de la matriz son tratadas como vectores. Por ejemplo, para introducir la matriz

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

```
(%i) matrix( [1, 0 , 3],[2, 0 ,6 ]);
```

La suma y resta de matrices se indica como es usual:

```
(%i) A:matrix([1,2,3],[-1,0,3],[2,1,-1]);
(%i) B:matrix([-1,1,1],[1,0,0],[-3,7,2]);
(%i) A+B;
(%i) A-B;
```

En cambio, el producto de matrices se indica con un punto, $(\cdot)$, como ya vimos con vectores. El operador $(*)$ multiplica los elementos de la matriz término a término:

```
(%i) A.B;
```

$$\text{(\%o)} \quad \begin{bmatrix} -8 & 22 & 7 \\ -8 & 20 & 5 \\ 2 & -5 & 0 \end{bmatrix}$$

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

$$\text{(\%o)} \quad \begin{bmatrix} -1 & 2 & 3 \\ -1 & 0 & 0 \\ -6 & 7 & -2 \end{bmatrix}$$

Con las potencias ocurre algo parecido: `^^` n eleva la matriz a n, es decir, multiplica la matriz consigo misma n veces,

```
(%i) A^^2;
```

$$(\%o)\quad \begin{bmatrix} 5 & 5 & 6 \\ 5 & 1 & -6 \\ -1 & 3 & 10 \end{bmatrix}$$

y `^` n eleva cada término de la matriz a n.

```
(%i) A^2;
```

$$(\%o)\quad \begin{bmatrix} 1 & 4 & 9 \\ 1 & 0 & 9 \\ 4 & 1 & 1 \end{bmatrix}$$

Para el producto de una matriz por un vector se utilizar el punto $(\cdot)$

```
(%i) A.v1;
(%o) [-4,8,-8]
```

La multiplicación producto de una matriz o un vector por un escalar se realiza con el operador de multiplicación $(*)$:

```
(%i) 3*A;
```

$$(\%o)\quad \begin{bmatrix} 3 & 6 & 9 \\ -3 & 0 & 9 \\ 6 & 3 & -3 \end{bmatrix};$$

Otras operaciones que *Maxima* permite realizar con matrices son:

`A[i,j]`	devuelve el elmento i,j, de la matriz `A`
`matrix_size(matrix)`	dimensiones de la matriz
`rank(matriz)`	rango de la matriz
`col(matriz,i)`	columna i de la matriz
`row(matriz, j)`	fila j de la matriz
`minor(matriz, i, j)`	menor de la matriz obtenida al eliminar la fila i y la columna j
`submatrix(f1, f2,..., matriz, c1,c2,...)`	matriz obtenida al eliminar las filas y columnas mencionadas
`determinant(matriz)`	determinante
`invert(matriz)`	matriz inversa
`transpose(matriz)`	matriz transpuesta

Por ejemplo, para extraer el elemento $(2,3)$ de la matriz B

```
(%i) B[2,3];
(%o) 0
```

para determinar el dimensiones de la matriz A:

```
(%i) matriz_size(A);
(%o) [3,3]
```

para calcular el rango de la matriz:

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

```
(%i) rank(matrix( [1, 0 , 3],[2, 0 ,6 ])); (%o) 1
```

La segunda columna de la matriz B:

```
(%i) col(B,2);
(%o)
```
$$\begin{bmatrix} 1 \\ 0 \\ 7 \end{bmatrix}$$

La primera fila de la matriz A:

```
(%i) row(A,1);
(%o) [1,2,3]
```

El menor $(1,2)$, de la matriz A:

```
(%i) minor(A,1,2);
(%o)
```
$$\begin{bmatrix} -1 & 3 \\ 2 & -1 \end{bmatrix}$$

La matriz que resulta al eliminar las filas 1 y 2 y la columna 3 de la matriz B:

```
(%i) submatrix(1,2,B,3);
(%o) [-3,7]
```

El determinante de A:

```
(%i) determinant(A);
(%o) 4
```

La inversa de B:

```
(%i) invert(B);
(%o)
```
$$\begin{bmatrix} 0 & 1 & 0 \\ -\frac{2}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{7}{5} & \frac{4}{5} & -\frac{1}{5} \end{bmatrix}$$

Por último, la matriz transpuesta de

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

viene dada por:

```
(%i) transpose(matrix([1,0,3],[2,0,6]));
(%o)
```
$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \\ 3 & 6 \end{bmatrix}$$

2.2. Sistema de Ecuaciones Lineales

Para resolver sistemas de ecuaciones lineales *Maxima* dispone de la función `linsolve ()` . A la función se le pasan dos argumentos, en primer lugar las ecuaciones, separadas por comas, y delimitadas por []. El segundo argumento es el vector de incógnitas. Por ejemplo para resolver el sistema:

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

```
(%i) linsolve([x+y+z=1,y-z=2,x-y-2*z=0],[x,y,z]);
(%o) [x=1/5, y=7/5, z= -3/5]
```

Si el sistema es incompatible *Maxima* devuelve el vector vacio []:

```
(%i) linsolve([x+y=1, 2*x + 2*y=4],[x,y]);
(%o) []
```

Por otro lado si el sistema es compatible indeterminado, *Maxima* introduce las variables `%ri`, la `i` denota el número de la variable. Este índice crece a lo largo de la sesión. Por ejemplo:

```
(%i) linsolve([x+y+z=1, x + y=1],[x,y,z]);
(%o) [x=1 -%r1, y=%r1, z=0]
```

Lo que estaría indicando:

$$x = 1 - \lambda$$
$$y = \lambda$$
$$z = 0$$

2.3. Espacios Vectoriales

Las herramientas que incorpora *Maxima* para el tratamiento de matrices nos permiten resolver muchos de los problemas que se han planteado en el tema de espacios vectoriales. Por ejemplo:

- Dependencia e Independencia Lineal: utilizando la función `rana` es muy sencillo si un conjunto de vectores son o no linealmente independientes. Por ejemplo, para decidir si los vectores

  $$\{(1, 20, 1), (2, 4, 1, 2), (0, 0, 1, 0)\}$$

  son linealmente independientes, escribimos

  ```
  (%i) rank(matrix([1,2,0,1],[2,4,1,2],[0,0,1,0]));
  (%o) 2
  ```

  De donde deducimos que no lo son. Para decidir cuales lo son basta encontrar una submatriz de rango dos, por ejemplo

  ```
  (%i) rank(matrix([1,2,0,1],[2,4,1,2]);
  (%o) 2
  ```

- Cáculo de ecuaciones paramétricas a partir de las implícitas: basta con resolver el sistema. Por ejemplo, para hallar las ecuaciones implícitas del subespacio:

  $$V = \{(x, y, z, t) : x + y + z = 0, z - t = 0\}$$

  ```
  (%i) linsolve([x+y+z=0, z-t= 0],[x,y,z,t]);
  (%o) [x=%r1,y=-%r2-%r1,z=%r2,t=%r2]
  ```

  Es decir, teneríamos las ecuaciones:

  $$x = \lambda, \quad y = -\mu - \lambda, \quad z = \mu, \quad t = \mu$$

- Cáculo de bases de un subespacio dado por ecuaciones implícitas: Para ello se resuelve el sistema, como en el caso anterior, y se da valores a los parámetros correspondientes. Por ejemplo en el caso anterior se obtendría,

  $$V = \langle (1, -1, 0, 0), (0, -1, 1, 1) \rangle.$$

  Otra posibilidad es usar la función `nullspace` que se aplica a la matriz asociada al sistema:

```
(%i) nullspace(matrix([1,1,1,0],[0,0,1,-1]));
```

$$\texttt{(\%o) span}\left[\begin{pmatrix}0\\-1\\1\\1\end{pmatrix},\begin{pmatrix}1\\-1\\0\\0\end{pmatrix}\right]$$

La palabra `span`, indica que el espacio V está generado por los vectores que aparecen a continuación.

- Cáculo de las ecuaciones implícitas a partir de una base. Para ello se deben calcular los correspondientes determinantes. Por ejemplo para calcular las ecuaciones implícitas del subespecie:

$$V = \langle (1,1,0,1), (0,1,-1,2) \rangle.$$

En primer lugar buscamos un menor de orden dos distinto de cero:

```
(%i) rank(matrix([1,1],[0,1]); (%o)2
```

Ahora ampliamos, y calculamos los correspondientes determinantes:

```
(%i) determinant(matrix([x,y,z],[1,1,0],[0,1,-1]);
(%o) z+y-x

(%i) determinant(matrix([x,y,t],[1,1,1],[0,1,2]);
(%o) -2y+x+t
```

De donde tenemos:

$$V = \{(x,y,z,t) : -x+y+z = 0, \quad x-2y+t = 0\}$$

- Bases de Suma de subespacios. Basta con calcular bases de cada espacio y usar la función `rank`.
- Bases de Intersección de subespacios. Se calculan las ecuaciones implícitas de la intersección y después se usa `nullspace`.
- Suma directa, suplementario. Se reducen a la resolución de problemas ya tratados.

Para ilustrar estos últimos puntos resolvemos el siguiente ejercicio:

Sea

$U = \langle (1,1,0), (-1,1,1), (0,2,1) \rangle \qquad V = \{(x,y,z) \in \mathbb{R}^3 : 2x-y+z = 0\}$

a) Calcular una base de $U+V$.
b) Calcular una base de $U \cap V$.
c) Comprobar si $U \oplus V$.
d) Calcular un suplementario de U.

Solución:

a) En primer lugar determinaremos bases de U y V. Comenzamos con U:

```
(%i) rank(matrix([1,1,0],[-1,1,1],[0,2,1]));
(%o) 2
```

De modo que la dimensión de U será 2. Como

```
(%i) rank(matrix([1,1,0],[-1,1,1]));
(%o) 2
```

podemos escoger como base de U $\{(1,1,0),(-1,1,1)\}$. En cuanto a V utilizamos la función `nullspace` para determinar una base

```
(%i) nullspace(matrix([2,-1,1));
```

(%o) span $\left[\begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}1\\2\\0\end{pmatrix}\right]$

la base de V será $\{(0,1,1),(1,2,0)\}$;

El subespacio $U+V$ está generado

$$U+V=\langle(1,1,0),(-1,1,1),(0,1,1),(1,2,0)\rangle$$

para generar una base basta con encontrar los que son linealmente independientes:

```
(%i) rank(matrix([1,1,0],[-1,1,1],[0,1,1],[1,2,0]));
(%o) 3
```

de donde resulta que $U+V=\mathbb{R}^3$, y por tanto cualquier base de $\mathbb{R}^3$ será base de $U+V$, por ejemplo la canónica: $\{(1,0,0),(0,1,0),(0,0,1)\}$.

b) Usando la fórmula de Grassman podemos deducir que la dimensión de la intesección $U\cap V$ es 1. Para obtener una base de la intersección, en primer lugar calculamos las ecuaciónes implícitas de U,

```
(%i) determinat(matrix([x,y,z],[1,1,0],[-1,1,1]));
(%o) 2z-y+x
```

La intersección $U\cap V$ viene dada por:

$$U\cap V=\{(x,y,z)\in\mathbb{R}^3 : x-y+2z=0, 2x-y+z=0\}$$

y calculamos una base usando `nullspace`

```
(%i) nullspace(matrix([1,-1,2],[2,-1,1]));
```

(%o) span $\left[\begin{pmatrix}1\\3\\1\end{pmatrix}\right]$

c) Puesto que $U\cap V\neq\{0\}$, deducimos que los espacios U y V no están en posición de suma directa.

d) Completamos una base de U hasta tener una base de $\mathbb{R}^3$:

```
(%i) rank(matrix([1,1,0],[-1,1,1],[1,0,0]));
(%o) 3
```

por tanto, podemos escoger como suplementario de U al subespacio $\langle(1,0,0)\rangle$.

2.4. Aplicaciones

Usando *Maxima* podemos definir una aplicación lineal. Por ejemplo

$$f(x, y, z) = (2x + 3y - z, 2y - 5z)$$

se introduce

```
(%i) f(x,y,z):= [2*x +3*y -z, 2*y-5z];
```

La matriz asociada en la base canóncia vendrá dada por:

```
(%i) A:transpose(matrix(f(1,0,0),f(0,1,0),f(0,0,1)));
```

$$\text{(\%o)} \begin{bmatrix} 2 & 3 & -1 \\ 0 & 2 & -5 \end{bmatrix}$$

Para calcular la dimensión de la imagen podemos hallar el rango de A:

```
(%i) rank(A);
(%o) 2
```

Para calcular una base de la $Im\ f$ basta con elegir dos columnas de A linealmente independientes. Es decir una submatriz cuadrada de A con rango 2, o determinante distinto de cero. Por ejemplo si eliminamos la tercera columna de A:

```
(%i) rank(submatrix(A,3));
(%o) 2
```

ó

```
(%i) determinat(submatrix(A,3));
(%o) 4
```

De modo que $Im\ f = \langle (2, 0), (3, 2) \rangle$.

Para calcular el núcleo de la aplicación, podemos resolver el sistema $f(x, y, z) = (0, 0)$,

```
(%i) linsolve([f(x,y,z)[1] = 0, f(x,y,z)[2] = 0],[x,y,z]);
```

$$\text{(\%o)} \left[-\frac{13\,\%r1}{4}, y = \frac{5\,\%r1}{2}, z = \%r1 \right]$$

si asignamos $\%r1 = 4$, obtenemos que el núcleo esta generado por el vector:

$$\text{Ker } f = \langle (-13, 10, 4) \rangle$$

También es posible obtener directamente la base del núcleo usando el comando `nullspace` sobre la matriz asociada a la aplicación:

```
(%i) nullspace(A);
```

$$\texttt{(\%o) span} \left(\begin{bmatrix} -13 \\ 10 \\ 4 \end{bmatrix} \right)$$

La palabra `span` indica que el núcleo de la aplicación está generado por $(-13, 10, 4)$.

En otros casos la aplicación lineal puede venir definida por medio de una matriz. Considerar el siguiente ejercicio:

Sea $f : \mathbb{R}^3 \to \mathbb{R}^2$ la aplicación lineal que tiene asociada la matriz

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

a) Hallar $f(x, y, z)$.

b) Hallar una base de $\ker f$ y una base de Im f.

c) Hallar la matriz asociada a f en las bases $B_1 = \{(1, 1, -1), (0, 0, 1), (2, 1, 1)\}$ de $\mathbb{R}^3$ y $B_2 = \{(-1, 1), (0, 1)\}$ de $\mathbb{R}^2$.

Solución: Introducimos la matriz A en *Maxima*:

```
(%i) A : matrix([2,-1,1],[-1,1,0]);
(%o)  [  2  -1  1 ]
      [ -1   1  0 ]
```

a) Para calcular la imagen de $f(x, y, z)$, basta con multiplicar la matriz A por el vector (x, y, z):

```
(%i) A.[x,y,z]
(%o)  [ z - y + 2x ]
      [ y - x      ]
```

b) Para hallar una base del núcleo:

```
(%i) nullspace(A);
(%o) span([  1 ])
          [  1 ]
          [ -1 ]
```

La base del la imagen, podemos calcular igual que el caso anterior. Primero el rango de A:

```
(%i) rank(A);
(%o) 2
```

ahora como el rango de la submatriz:

```
(%i) rank(submatrix(A,3));
(%o) 2
```

resulta que $Im\ f = \langle(2, -1), (-1, 1)\rangle$.

c) Introducimos las matrices, y utilizamos la fórmula de cambio de base:

```
(%i) B:matrix([1, 0, 2],[1,0,2],[-1,1,1]); C:matrix([-1,0],[1,1]);
(%i) invert(C).A.B ;
(%o)  [ 0  -1  -4 ]
      [ 0   1   3 ]
```

2.5. Diagonalización

El comando más útil para la diagonalización de una matriz A es `eigenvectors`, que devuelve los autovalores de A y los autovectores asociados. Permite, por tanto, decidir si A es diagonalizable o no. En caso afirmativo, calcular P y D tales que $A = P \cdot D \cdot P^{-1}$.

Por ejemplo:

```
(%i) A: matrix([4,0,0],[0,2,2],[0,2,2]);
(%i) eigenvectors(A);
(%o) [[[0,4],[1,2]],[[[0,1,-1]],[[1,0,0],[0,1,1]]]]
```

La salida se interpreta del siguiente modo:

- La primera parte corresponde a los valores propios. Aparecen dos vectores, el primero índica los valores propios. El segundo vector son las multiplicidades de los valores propios, es decir en este caso:
  - 0 es valor propio de A de multiplicidad 1.
  - 4 es valor propio de A de multiplidadad 2.
- La segunda parte corresponde a los valores propios. Aparecerán tantas listas como valores propios. Cada lista contendrá los vectores propios asociados a cada valor propio. En este caso:
  - El valor propio 0 tiene el vector propio asociado $(0, 1, -1)$.
  - El valor propio 4 tiene los vectores propios asociados $(1, 0, 0), (0, 1, 1)$.

En este caso como la multiplicidad aritmética y geométrica de los autovalores coincide se pude concluir que la matriz A es diagonalizable, donde:

$$D = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix} \qquad P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

Puede comprobarse con *Maxima* que $A = P * D * P^{-1}$.

Tambiés es posible calcular el **polinomio característico**. Para ello se usa la función `charpoly()`. A esta función se le pasan dos argumentos: la matriz y la variable del polinomio:

```
(%i) charpoly(A,x);
(%o) ((2-x)^2-4) (4-x)
```

Este polinomio puede ser factorizado con la función `factor()`

```
(%i) factor(charpoly(A,x));
(%o) -(x-4)^2 x
```

Las raíces del polinomio pueden calcularse con la función `allroots`

```
(%i) allroots(charpoly(A,x));
(%o) [x=0.0, x=4.0, x=4.0]
```

Sin embargo, calcular los autovalores de una matriz de este modo es un procedimiento poco recomendable, por ser un procedimiento muy inestable. Siempre es mejor opción usar el comando `eigenvectors()`, o si solo se precisa conocer los autovalores el comando `eigenvalues`.

```
(%i) eigenvalues(A);
(%o) [[0,4],[1,2]]
```

Como en el caso anterior *Maxima* nos informa que hay dos autovalores 0 y 4, de multiplicidad 1 y 2 respectivamente.

2.6. Ejercicios

2.6.1. Vectores y Matrices

1. Considerar los vectores $a = (1, 2, -1)$, $b = (0, 2, 3/4)$, c $=(1, 1, 0)$ y $d = (0, 0, 1)$. Realizar las siguientes operaciones

a) $a + b$

b) $3c + 2b$

c) $c \cdot d$

d) $b \cdot d + 3a \cdot c$

2. Considerar las matrices:

$$A = \begin{pmatrix} 1 & -2 & 0 \\ 2 & 5 & 3 \\ -3 & 1 & -4 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & -2 & 6 \\ 12 & 2 & 0 \\ -1 & -1 & 3 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 2 & 0 & -5 \\ -4 & -2 & 1 & 0 \\ 3 & 2 & -1 & 3 \\ 5 & 4 & -1 & -5 \end{pmatrix} \qquad D = \begin{pmatrix} -1 & 2 & 3 & 0 \\ 12 & -5 & 0 & 3 \\ -6 & 0 & 0 & 1 \end{pmatrix}$$

a) Calcular $A \cdot B$, $A + B$, $D \cdot C$

b) Extraer la segunda fila de A, la tercera de C y el elemento $(3, 3)$ de D

c) Calcular $det(A)$, $det(B)$ y $det(C)$. Para las matrices cuyo determinante sea no nulo, calcular su inversa.

d) Calcular el rango de las matrices, A, B, C, D, $D \cdot C$, y $A + B$

2.6.2. Sistemas de Ecuaciones Lineales

1. Resolver los siguientes sistemas de ecuaciones:

$$\left.\begin{array}{rrrr} x & +5y & +2z & = 8 \\ 3x & -y & -2z & = 8 \\ 2x & & -z & = 6 \end{array}\right\} \qquad \left.\begin{array}{rrrrr} 2x & +y & -z & +t & = 3 \\ & & z & +2t & = 3 \end{array}\right\}$$

$$\left.\begin{array}{rrrrr} x & +y & +z & +t & = 5 \\ 2x & -y & +z & +2t & = 11 \\ x & -y & +2z & -2t & = 0 \\ x & +2y & & +3t & = 8 \end{array}\right\}$$

2.6.3. Espacios Vectoriales

1. En $\mathbb{R}^4$ se consideran los vectores: $a = (1, 0, 0, 1)$, $b = (0, -1, 2, 0)$, $c = (1, -2, 2, 1)$, $d = (2, 3, -6, 2)$ $e = (1, 1, 0, 0)$.

   a) Hallar una base del subespacio generado por a, b, c.

   b) Hallar una base del subespacio generado por d, e.

   c) Hallar una base del subespacio intersección de los dos anteriores.

2. En $\mathbb{R}^5$ se consideran los siguientes subespacios: S el subespacio generado por $a = (1, 1, 1, 1, 1)$, $b = (1, -1, 1, -1, 1)$, S' el subespacio generado por $a' = (0, 0, 1, 0, 1)$, $b' = (1, 0, 0, 0, 0)$, $c' = (1, 1, 0, 0, 1)$.

   a) ¿Pertenece el vector $(1, 2, 2, 1, 4)$ al subespacio $S \cap S'$?

   b) Calcular una base de la intersección $S \cup S'$.

3. Sean A, B y C los siguientes subespacios de $\mathbb{R}^3$:

   $A = \{(x, y, z); x + y + z = 0\}$

$B = \{(x, y, z); x = z\}$

$C = \{(x, y, z); x = y = 0\}$

a) Probar que:

1) $\mathbb{R}^3 = A + B$
2) $\mathbb{R}^3 = A + C$
3) $\mathbb{R}^3 = B + C$

b) ¿En qué caso las sumas del apartado anterior son directas?

4. En $\mathbb{R}^3$ se considera el subespacio $U = \{(x, y, z) : x + y + z = 0\}$. Hallar el suplementario de U y descomponer el vector $(1, 2, 1)$ según U y su suplementario.

5. Sean $U = \{(a, b, c) \in \mathbb{R}^3 : a = b = c\}$ y $V = \{(a, b, c) \in \mathbb{R}^3 : a = 0\}$. Probar que $\mathbb{R}^3$ es suma directa de U y V.

6. Estudiar si $\mathbb{R}^4 = U \oplus V$ donde

$U = \{(x, y, z, t) \in \mathbb{R}^4 : x - y = 0,\ z = 0\}, \qquad V = \{(x, y, z, t) \in \mathbb{R}^4 : x = 0,\ y + z = 0\}$

2.6.4. Aplicaciones

1. Sea $f : \mathbb{R}^3 \to \mathbb{R}^2$ la aplicación lineal definida por $f(x, y, z) = (x + 2y - z, y - 3z)$.

a) Hallar la matriz asociada a f en la bases canónicas de $\mathbb{R}^3$ y $\mathbb{R}^2$.

b) Hallar una base de $\ker f$ y una base de Im f.

c) Hallar $f(V)$ donde $V = \{(x, y, z) \in \mathbb{R}^3 : x - z = 0, x + y = 0\}$.

d) Hallar la matriz asociada a f en las bases $B_1 = \{(1, 0, 0), (1, 1, 0), (1, 1, 1)\}$ de $\mathbb{R}^3$ y $B_2 = \{(2, -1), (1, 1)\}$ de $\mathbb{R}^2$.

2. Sea el homomorfismo $T : \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ definido por: $T(x, y) = (x + y, x - y, 2x)$

a) Calcular su matriz asociada respecto de las bases canónicas de $\mathbb{R}^2$ y $\mathbb{R}^3$.

b) Calcular su matriz asociada respecto de las bases $e_1 = (1, 1)$, $e_2 = (0, -1)$ de $\mathbb{R}^2$ y $u_1 = (0, 1, 1)$, $u_2 = (1, 0, 1)$, $u_3 = (1, 1, 0)$ de $\mathbb{R}^3$.

c) Qué relación existe entre ellas.

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

$$\begin{array}{rcl} f(1,0,0) & = & (1,1,0) \\ f(1,1,0) & = & (2,0,0) \\ f(1,1,1) & = & (2,0,2) \end{array}$$

a) Hallar una base de $\ker f$ y una base de Im f.

b) Hallar $f(V)$ donde $V = \langle (1, 0, 0), (1, -1, 0) \rangle$.

c) Hallar la matriz asociada a f en la base canónica de $\mathbb{R}^3$.

d) Hallar la matriz asociada a f en la base $\{(0, 1, 1), (1, 0, 1), (1, 1, 0)\}$ de $\mathbb{R}^3$.

2.6.5. Diagonalización

1. Estudiar si es posible la diagonalización de las siguientes matrices cuadradas reales

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad \begin{pmatrix} -1 & 0 & 0 \\ 3 & 2 & 0 \\ 1 & 4 & -1 \end{pmatrix}, \quad \begin{pmatrix} -1 & 0 & 0 \\ 3 & 2 & 0 \\ 1 & 4 & -1 \end{pmatrix},$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, \quad \begin{pmatrix} -1 & -2 & 3 & 3 \\ 0 & 1 & 0 & 1 \\ -2 & -2 & 4 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. Calcular

$$\begin{pmatrix} 4 & 0 & -2 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}^n, \quad \begin{pmatrix} 2 & -1 & 3 \\ 0 & 1 & -1 \\ 0 & 0 & 3 \end{pmatrix}^n, \quad \begin{pmatrix} 3 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 3 \end{pmatrix}^n$$

3. Estudiar para qué valores de los parámetros $a, b, c \in \mathbb{R}$, el endomorfismo $T : \mathbb{R}^3 \to \mathbb{R}^3$ que en la base canónica de $\mathbb{R}^3$ tiene asociada la matriz:

$$\begin{pmatrix} 1 & a & b \\ 0 & 2 & c \\ 0 & 0 & 1 \end{pmatrix}$$

es diagonalizable.

Capítulo 3

Listas y Sucesiones

3.1. Introducción

El objetivo de este capítulo es el manejo de las sucesiones con `wxMaxima`. Para ello en primer lugar estudiaremos cómo opera `wxMaxima` con listas.

3.2. Listas

Una lista es una enumeración, en cierto modo se puede identificar con un vector. El objetivo de esta sección es ilustar como trabaja *Maxima* con listas. Todo ello servirá como base para el manejo de sucesiones.

Existen diferentes formas de definir una lista en *Maxima*:

- Explícitamente. Basta con introducir los elementos de la lista entre corchetes ([ ]) separados por comas, como haciamos con los vectores:

```
(%i) L: [a, sin(b), c!, d-1];
```

- Comando `makelist`. Permite crear una lista dada una ley de formación. La sintaxis es la siguiente:

```
makelist (<expr>, <i>, <i_0>, <i_max>, <step>)
```

donde:

`expr` : ley de formacion

`i` : parámetro de la ley de formación.

`i_0` : valor inicicial del parámetro.

`i_max` : valor final del parámetro.

`step` : paso de avance del parametro. Si no se indica se supone 1.

Por ejemplo, si queremos generar los primeros 10 términos de una lista cuya ley de formación es $\frac{1}{n}$, escribiríamos:

```
(%i) makelist(1/n, n, 1,10,1);
```

o simplemente

```
(%i) makelist(1/n, n, 1,10);
```

en ambos casos obtendríamos:

```
(%o)
```

$$[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}]$$

A menudo, es reconendable definir previamente la ley de formación. Esto puede hacerse del siguiente modo:

```
(%i) a[n]:=1/n;
(%i) makelist(a[n], n, 1,10);
```

Por último, para dibujar los puntos que forman una lista se deben crear una nueva lista en la que cada elemento es un par: $[n, a_n]$

```
(%i) makelist([n,a[n]], n, 1,10);
(%o)
```

$$[[1,1],[2,\frac{1}{2}],[3,\frac{1}{3}],[4,\frac{1}{4}],[5,\frac{1}{5}],[6,\frac{1}{6}],[7,\frac{1}{7}],[8,\frac{1}{8}],[9,\frac{1}{9}],[10,\frac{1}{10}]]$$

- Comando `create_list`. La sintaxis es muy parecida a la anterior. En este caso la lista de valores que toma el parámetro se pasa de forma explicita. Además permite crear listas que dependan de más de una variable.

  ```
  create_list (<form>, <x_1>, <list_1>, ..., <x_n>,<list_n>)
  ```

  donde:

  `form` : ley de formacion
  `x_1` : primer parámetro de la ley de formación.
  `list_1` : lista de valores del primer par'ametro.
  ...
  `x_n` : valor final del parámetro.
  `list_n` : lista de valores del n-ésimo par'ametro.

  Por ejemplo:

  ```
  (%i) create_list (i^ 3 , i,[1,3,5,7])
  ```

  produce

  ```
  (%o)
  ```

  $$[1, 27, 125, 343]$$

  Como comentabamos, también es posible generar listas que dependan de dos parámetros:

  ```
  (%i) create_list ([i, j], i, [a, b], j, [e, f, h]);
  (%o)[[a,e],[a,f],[a,h],[b,e],[b,f],[b,h]]
  ```

Maxima proporciona funciones que permiten manipular listas. Algunas de las más importantes se muestran a continuación:

`L[i]`	muestra el elemento i-ésimo e la lista L
`append (L1,L2)`	concatena las listas L1 y L2
`join(L1,L2)`	genera una lista entrelazando los elementos de L1 y L2
`push(x,L)`	introduce el elemento x al inicio de la lista L (paquete basic)
`pop(L)`	elimina el primer elemento de la lista L
`unique(L)`	elimina los elementos repetidos en la lista L
`+,-,*,/`	operaciones algebraicas con listas. Deben tener la misma longitud
`apply(f,L)`	applica la función f a la lista L
`map(f,L)`	applica la función f a cada uno de los elementos de la lista L

Por ejemplo, dadas las listas:

```
(%i) L1:makelist(3*n, n,1,5);
(%o) [3,6,9,12,15]
(%i) L2:makelist(2*n-1,n,1,5);
(%o) [1,3,5,7,9]
```

El elemento 3 de la lista L1 se obtiene:

```
(%i) L1[3];
(%o) 9
```

La concatenación de las dos listas resulta:

```
(%i) L3:append(L1,L2);
(%o) [3,6,9,12,15,1,3,5,7,9]
```

Si mezclamos los elementos resulta:

```
(%i) join(L1,L2);
(%o) [3,1,6,3,9,5,12,7,15,9]
```

Podemos añadir el elemento 0 a la lista L1,

```
(%i) push(0,L1);
(%o) [0,3,1,6,3,9]
```

y después eliminarlo:

```
(%i) pop(L1);
(%o) 0
(%i) L1
(%o) [3,6,9,12,15]
```

Si queremos ordenar la lista L3 y eliminar los elementos repetidos podemos utilizar:

```
(%i) unique(L3);
(%o) [1,3,5,6,7,9,12,15]
```

La multiplicación de los elementos de las listas L1 y L2 se indica:

```
(%i) L1*L2;
(%o) [3,18,45,84,135]
```

Podemos sumar todos los elementos de la lista L1, utilizando la función `map`:

```
(%i) apply( ''+'',L1);
(%o) 45
```

o calcular el máximo de la lista L2:

```
(%i) apply(max,L1);
(%o) 15
```

Por último, si queremos calcular el coseno de cada uno de los elementos de la lista L2, usamos:

```
(%i) map(cos,L2);
(%o) [cos(3),cos(6),cos(9),cos(12),cos(15)]
```

3.2.1. Ejercicios

1. Generar las siguientes listas utilizando *Maxima*:
   - Los 100 primeros números impares.
   - Todos los múltiplos de 3 menores de 100.
   - Los cubos de los 50 primeros números naturales.
   - Todos los múltiplos de 9 ó 7 menores de 200.
2. La sucesión de Fibonacci viene definida por:

$$a_1 = a;$$
$$a_2 = b;$$
$$a_n = a_{n-1} + a_{n-2};$$

   - Utiliza *Maxima* para generar los 20 primeros términos de la sucesión si $a = 1$, $b = 1$;
   - Calcular los 19 primeros términos de la sucesión $\frac{a_{n+1}}{a_n}$. Utiliza el comando float, para generar un resultado numérico.
   - Cúal es el límite de la sucesión anterior. Sabrías decir de qué número se trata.
   - Qué ocurre si se cambia el valor de a y b.
3. Calcula el mínimo y máximo de las siguientes listas.

$$\{n^2-3n+5, n, 1, 10\} \quad ,\{\cos(n)^3\sin(n), n, 1, 200\} \quad ,\{\sin(\log(n^2+3)), n, 1, 100\}$$

4. Calcula las siguientes operaciones:

$$\sum_{i=1}^{100} i \quad ,\sum_{n=1}^{100} \frac{1}{n} \quad ,\sum_{n=1}^{64} 2^n, \quad \prod_{n=1}^{20} 2n-1$$

3.3. Sucesiones

Un sucesión de números reales $\{a_n\}_{n=1}^{\infty}$ es una aplicación del conjunto de los números naturales, $\mathbb{N}$, en los números reales, $\mathbb{R}$:

$$a : \mathbb{N} \longrightarrow \mathbb{R}$$
$$n \longrightarrow a_n$$

Usualmente la sucesión se identifica con el denominado término general a_n.

Una sucesión a_n puede venir definida por medio de una expresión explícita o ley de formación, por recurrencia, o por una defínción explícita.

- Expresión explícita. Es el caso más habitual, y *Maxima* permite definir la sucesión de forma automática. Por ejemplo, la sucesión $a_n = \frac{1}{n^2}$:

```
(%i) a[n]:= 1/n^ 2
```
(%o) $a_n := \frac{1}{n^2}$

  Para acceder a uno de los términos de la sucesión, por ejemplo a_5 basta con sustituir n por 5:

```
(%i) a[5];
(%o) 1/25
```

Para mostrar los primeros términos de la sucesión podemos utilizar los comandos que han sido descritos en la sección anterior:

```
(%i) makelist(a[n], n,1,10);
(%o) [1, 1/4, 1/9, 1/16, 1/25, 1/36, 1/49, 1/64, 1/81, 1/100]
```

- Ley de recurrencia. En este caso la fórmula general del término n-ésimo de la sucesión depende de términos anteriores. El ejercico 2 de la sección 3.2.1 es un ejemplo de tales sucesiones. El primer o primeros términos son definidios de forma expícita y a continuación se introduce la ley de recurrencia.

  Por ejemplo para definir la sucesión,

  $$c_1 = 1; \qquad c_n = \sqrt{1 + c_{n-1}};$$

  se procede del siguiente modo:

  ```
  (%i) c[1] : 1;
  (%i) c[n]:= sqrt(1 + c[n-1]);
  ```

  Observar que al definir el valor de c_1 se utiliza **asignación estática**, y en el segundo caso la **asignación es dinámica**. En el primer caso *Maxima* simplemente asigna el valor 1 a c_1, sin embargo en la segunda expresión *Maxima* invocará la fórmula siempre que $n \neq 1$.

  Igual que el caso anterior, podemos acceder a un término concreto de la sucesión o generar los primeros términos de la misma:

  ```
  (%i) c[4];
  (%o) sqrt(sqrt(sqrt(2)+1)+1)

  (%i) makelist(float(c[n]), n,1,4);
  (%o)[1.0,1.414,1.553,1.598]
  ```

  En el ejemplo anterior, hemos utilizado el comando float() para mostar las expresiones decimales de los términos de la sucesión.

- Definición explícita. Corresponde a este tipo las sucesiones que no encajan en los casos anteriores. Son aquellas para las que no existe una 'fórmula".

  Por ejemplo son sucesiones de este tipo las cifras que forman la expresión decimal del número π, o la sucesión de números primos.

  No obstante, usando las funciones que proporciona *Maxima* en ocasiones es posible generar sucesiones de este tipo. Por ejemplo, la sucesión de números primos se puede definir de modo recurrente del siguiente modo:

  ```
  (%i) p[1]:2;
  (%i) p[n]:=next_prime(p([n-1]));
  ```

  Podemos obtener los 10 primeros números primos del siguiente modo:

  ```
  (%i) makelist(p[n],n,1,10);
  (%o) [2,3,5,7,11,13,17,19,23,29]
  ```

3.3.1. Gráficas

Una de las grandes ventajas que ofrece *Maxima* es su soporte gráfico. En esta sección describiremos las herramientas que permiten dibujar los términos de una sucesión.

Existen dos comandos para dibujar:

`plot2d` : *Maxima* despliega una nueva ventana donde incluye el gráfico.

`wxplot2d` : *Maxima* incluye el gráfico en la ventana de comandos.

La sintaxis para ambos comandos es la misma. Se trata de comandos de gran versatilidad, y que permiten dibujar no sólo sucesiones, si no también funciones, y con una enorme cantidad de opciones. En esta sección nos centraremos en sucesiones, y dejaremos para más adelante otras funcionalidades de los comandos `plot2d`, `wxplot2d`. En lo que sigue siempre escribiremos `wxplot2d`, aunque como hemos dicho todo sigue siendo válido para `plot2d`.

Para dibujar los términos de una sucesión, se deben seguir los siguientes pasos:

1. Definir término general de la sucesión. Por ejemplo si $a_n = \left(1 + \frac{1}{n}\right)^n$

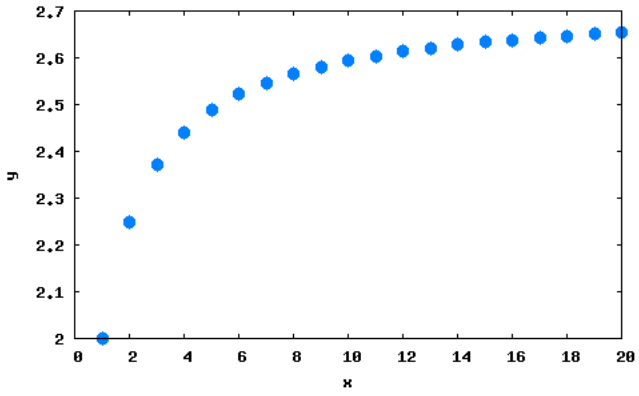
```
(%i) a[n]:= (1 + 1/n)^ n
```

2. Generar los términos de la sucesión que deseamos dibujar. Por ejemplo si queremos dibujar los 20 primeros términos de la sucesion a_n:

```
(%i) La: makelist([n,a[n]], n,1,20]$
```

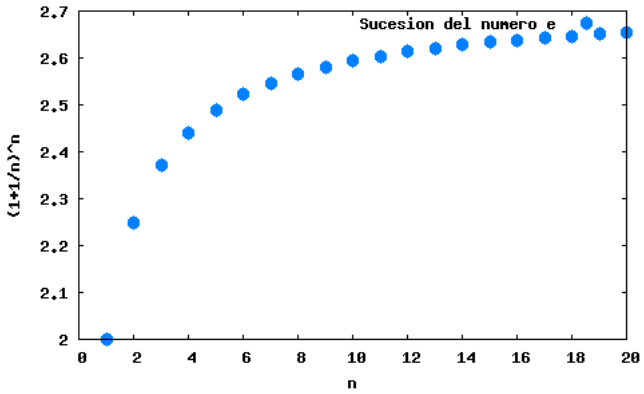
3. Llamar al comando `wxplot2d`;

```
(%i) wxplot2d([discrete, La],[style, points])
```



4. Formatear el grafico, incluyendo leyendas o etiquetas si se desea:

```
(%i) wxplot2d([discrete, La],[style, points],[legend,"Sucesion
del numero e"],[xlabel, "n"], [ylabel, "(1+1/n)^ n")
```



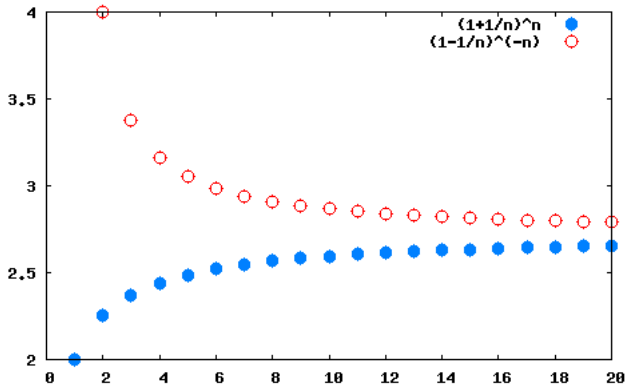
Algun comentario en cuanto a la sintaxis figura a continuación. El comando *discrete*, indica que el dato a dibujar es una lista finita que aparece a continuación. Más adelante veremos que se omite al dibujar funciones. El comando *style* acompañado de *points*, indica que el gráfico aparecera como una nube de puntos. Por defecto *Maxima* une los puntos con una polilínea.

También es posible dibujar dos sucesiones sobre el mismo gráfico. Por ejemplo, definimos la sucesión $b_n = \left(1 - \frac{1}{n}\right)^{-n}$ y la dibujamos con la anterior:

```
(%i) b[n]:= (1 + 1/n)^ n $
(%i) Lb: makelist([n,b[n]], n,2,20]$
```

A continuación las dibujamos juntas:

```
(%i) wxplot2d([[discrete,La],[discrete,Lb]],[style,points],[legend,
["(1+1/n)^ n","(1-1/n)^ (-n)"]]);
```



3.3.2. Límites

En esta sección utilizaremos *Maxima* para el cálculo de límites de sucesiones. El comando es:

```
limit (<expr>, <n>, <val>)
```

donde:

`expr` : término general de la sucesión

`n` : variable de la sucesión.

`val` : en el caso de sucesiones es siempre `inf` que denota infinito (∞).

Por ejemplo, para calcuar el límite de la sucesión $a_n = \left(1 + \frac{1}{n}\right)^n$, escribiríamos:

```
(%i) limit((1+1/n)^ n,n,inf);
(%o) %e
```

Maxima resuelve gran cantidad de indeterminaciones:

- $\infty - \infty$:

  ```
  (%i) limit(sqrt(n^ 2+1)-n ,n, inf);
  (%o) 0
  ```

- $\frac{\infty}{\infty}$:

  ```
  (%i)limit((n^ 2-3*n)/(n + 3),n,inf);
  (%o) ∞
  ```

- $\frac{0}{0}$:

  ```
  (%i)limit(sin(1/n)/log(1 + 2/n), n, inf);
  (%o) 1/2
  ```

- ∞^0:

  ```
  (%i) limit((n^ 3-1)^ (1/(3*n)), n, inf);
  (%o)1
  ```

- 0^0:

  ```
  (%i)limit(sin(1/n)^ (1 - cos(1/n)),n, inf);
  (%o)1
  ```

- 1^∞:

  ```
  (%i) limit((1-3/n)^ (2*n),n,inf);
  (%o)e^-6.
  ```

Sin embargo, tiene problemas con el cálculo de límites dados por formulas de recurrencia:

$$c_1 = 1; \qquad c_n = c_{n-1}/(1 + c_{n-1});$$

```
(%i) c[1]:1 $ c[n]:=c[n-1]/(1+c[n-1]) $
(%i) limit(c[n],n,inf);
```

Observaras, que la evaluación anterior devuelve un mensaje de error, o no termina.

Maxima también tiene problemas con límites que requieren el uso del criterio de Stolz:

$$\lim_{n\to\infty} \frac{\sqrt{1} + \sqrt{2} + \ldots + \sqrt{n}}{n\sqrt{n}}$$

```
(%i) limit(sum(sqrt(i),i,1,n)/(n*sqrt(n)),n,1,inf);
(%o)limit: direction must be either 'plus' or 'minus'; found:
inf - an error. To debug this try: debugmode(true);
```

3.3.3. Ejercicios

1. Considerar la sucesion $a_n = \frac{n+(-1)^n}{n-(-1)^n}$.
   - Dibujar los primeros 20 términos de la sucesión. Cúal parece ser si límite.
   - Calcula el límite.
2. Considerar la sucesión $a_n = \left(\frac{3n-1}{3n+2}\right)^{n+3}$
   - Dibujar los primeros 50 términos y deducir que la sucesión es acotada y monótona creciente.
   - Calcular el límite.
3. Considerar la sucesión definida por $x_1 = 3$; $x_n = \sqrt{2 + x_{n-1}}$.
   - Definir la sucesión $y_n = x_{n+1} - x_n$. Utilizar *Maxima* para comprobar que todos sus términos son negativos. Lo que implica que $x_{n+1} \leq x_n$, es decir la sucesión x_n es decreciente.
   - Dibujar algunos términos de la sucesión x_n y deducir que es acotada.
   - Deducir de los apartados anteriores que x_n es convergente. Cúal parece ser el límite.
   - El límite si existe, debe ser la solución de la ecuación:

     $$L = \sqrt{2 + L}$$

     Comprobar que el valor de L coincide con el del apartado anterior.
   - Repetir el ejercicio para otros valores de x_1. Por ejemplo $x_1 = 0$, $x_1 = 1$, $x_1 = 2$, $x_1 = 8$. Cúal es el valor del límite en estos casos. Es monótona la sucesión y_n en estos casos.
4. Utilizar *Maxima* para calcular los siguientes límites:

$$\lim_{n\to\infty} \sqrt[n]{\frac{(2n)!}{n!n^n}} \qquad \lim_{n\to\infty} \frac{3/n}{\log(\frac{n}{n-1})}$$

$$\lim_{n\to\infty} \frac{\sqrt[n]{n!}}{n} \qquad \lim_{n\to\infty} \frac{n\sin(n!)}{n^2+1}$$

$$\lim_{n\to\infty} \left(1 - n^{\frac{1}{n}}\right) \qquad \lim_{n\to\infty} \left(\frac{n+4}{n-1}\right)^{(3n+1)}$$

Capítulo 4

Series

4.1. Introducción

Dada una sucesión de números reales $\{a_n\}$ se denomina **suma parcial n-ésima** de a_n a

$$S_n = a_1 + a_2 + \ldots + a_n$$

Se denomina **serie** de término general a_n, a la sucesión formada por las sumas parciales n-ésimas de $\{S_n\}$. Al límite

$$\lim_{n\to\infty} S_n \quad \text{lo denotamos} \quad \sum_{n=1}^{\infty} a_n.$$

El carácter de la serie viene determinado por el caracter de la sucesión $\{S_n\}$. Así diremos que la serie es **convergente** si el límite anterior es un número real, es **divergente** si el límite es ∞, o es **oscilante** si el límite no existe.

Que una serie sea convergente no implica que se pueda calcular el valor del límite $\sum_{n=1}^{\infty} a_n$, si no tan sólo que el límite es finito. Se denominan series **sumables** a aquellas series para la que es posible calcular $\sum_{n=1}^{\infty} a_n$.

El objetivo de esta práctica es presentar las herramientas de las que dispone `Maxima` para trabajar con series. Comenzaremos describiendo los comandos que permiten sumar series (calcular el valor del límite $\sum_{n=1}^{\infty} a_n$). Como avanzabamos en el parrafo anterior, calcular la suma de una serie no siempre es posible. A continuación abordaremos la convergencia de series. Primero construiremos la sucesión de sumas parciales y la respresentaremos gráficamente. Esto nos permitirá obtener algunas conclusiones sobre el comportamiento de la serie, aunque en ningún momento será una garantía de éxito. Finalmente, estudiaremos la convergencia de series, desde un punto de vista analítico usando los criterios de convergencia.

4.2. Suma de Series

`Maxima` permite obtener la suma de algunas series, entre ellas:

- Series **geométricas**: $\sum r^n$, con $r \in \mathbb{R}$, $0 < r < 1$.
- Series **aritmético geométricas**: $\sum P(n)r^n$, con $r \in \mathbb{R}$, $0 < r < 1$ con $P(n)$ un polinomio.
- Series **armónicas**: $\sum \frac{1}{n^p}$, con $p \in \mathbb{R}$, con $p > 1$.
- Series **telescópicas**: $\sum a_n$, con $a_n = b_{n+m} - b_n$, con $\lim_{n\to\infty} b_n = 0$.

Para ello, `Maxima` dispone de los siguientes comandos:

`sum`	función elemental para la suma de series
`nusum`	emplea algoritmos más eficaces que `sum` en expresiones racionales
`symply_sum`	es el comando más potente de `Maxima` para la suma de series. Requiere cargar el paquete `symply_ sum`

4.2.1. Comando `sum`

La sintaxis de `sum` es la siguiente

```
sum (<expr>, n, i_0, i_max)
```

donde:

`expr` : término general de la serie.

`n` : parámetro del término general.

`i_0` : valor inicicial del parámetro.

`i_max` : valor final del parámetro, puede ser `infty` (∞).

Este comando permite calcular sumas parciales:

```
(%i) a[n]:1/3^ n $
(%i) sum(a[n],n,1,100);
```
(%o) $\frac{25768876036600566551823056488281063635105376 1000}{51537752073201133103646112976562127270210752 2001}$

incluso obtener reglas de sumación:

```
(%i) b[n]: = n^ 2;
(%i) sum(b[i],,1,n),simpsum;
```
(%o) $\frac{2\,n^3+3\,n^2+n}{6}$

Notar que esta ocasipón se ha añadido el operador `simpsum` que índica a `Maxima` que debe simplicar la expresión resultante. Es **recomendable** utilizar este comando siempre que se utilice la instrucción `sum`.

En cuanto a la suma de series, este comando permite obtener algunas resultados:

```
(%i) sum(a[n],n,1,inf),simpsum;
```
(%o) $\frac{1}{2}$

Incluso puede proporciona información sobre series no convergentes:

```
(%i) c[n] :=1/n;
(%i) sum(c[n],n,1,inf),simpsum;
(%o) sum: sum is divergent.
- an error. To debug this try: debugmode(true);
```

Sin embargo, en ocasiones no proporciona información relevante:

```
(%i) d[n] :=n/3^ n;
(%i) sum(d[n],n,1,inf),simpsum;
```
(%o) $\sum_{n=1}^{\infty} \frac{n}{3^n}$

4.2.2. Comando `nusum`

La sintaxis de `nusum` es la misma que la del comando `nusum`. La principal diferencia es que el último utiliza técnicas de simplificación más sofisticadas.

```
nusum (<expr>, n, i_0, i_max)
```

donde:

`expr` : término general de la serie.

`n` : parámetro del término general.

`i_0` : valor inicicial del parámetro.

`i_max` : valor final del parámetro, puede ser `infty` (∞).

Por ejemplo, la suma de la serie ármónica $a_n = \frac{1}{n^3}$, procedemos:

```
(%i) kill(a);
(%i) a[n] :=1/n^ 3
(%i) sum(a[n],n,1,inf),simpsum;
(%o) ζ (3)
```

La función $\zeta(\cdot)$ se denomina funcion *zeta* y podemos obtener una aproximación numérica del siguiente modo:

```
(%i) float(%);
(%o) 1,202056903159594
```

4.2.3. Comando `simplify_sum`

Esta instrucción utiliza las reglas mas sofisticadas que posee `Maxima` para la simplificación de la suma de series. Forma parte del paquete `symply_ sum`. De modo que para utilizar esta instrucción en primer lugar es necesario cargar el paquete:

```
(%i) load(simplify_sum);
```

Su sintaxis es la siguiente:

```
simplify_sum (<serie>)
```

donde:

`serie` : es la expresión de la serie con el comando `sum` o `nusum`.

Si tratamos de obtener la suma de la serie $b[n] = \frac{2n^2}{n!}$

```
(%i) kill(b);
(%i) b[n]:=2*n^ 2/n!;
(%i) sum(b[n],n,1,inf);
(%o) 2 ∑_{n=1}^{∞} n^2/n!
```

Sin embargo si utilizamos el comando `simplify_sum` (recuerda que debes cargar el paquete):

```
(%i) simplify_sum(sum(b[n],n,1,inf));
(%o) 4e
```

4.3. Convergencia de Series

En esta sección estudiaremos la convergencia de series. Primero consturyendo y dibujando la suma de series parciales y después mediante el empleo de los criterios de convergencia. Por supuesto el primer método nos proporcionará información sobre la serie pero nunca nos permitirá decidir su carácter de modo riguroso.

4.3.1. Representación gráfica

Las herramientas que proporciona `Maxima` nos permiten de modo sencillo calcular la sucesión de sumas parciales, y estudiar sus propiedades.

Por ejemplo, vamos a construir las sucesiones de sumas parciales de las series:

$$\sum_{n=1}^{\infty} \frac{1}{n}, \qquad \sum_{n=1}^{\infty} \frac{1}{n^2}$$

y después las dibujaremos y calcularemos aproximaciones de su suma (para el segundo caso).

```
(%i) a1[n] := 1/n
(%i) suma1[n]:=sum(a1[k],k,1,n),simpsum;


(%i) a2[n] := 1/n^2
(%i) suma2[n]:=sum(a2[k],k,1,n),simpsum;
```

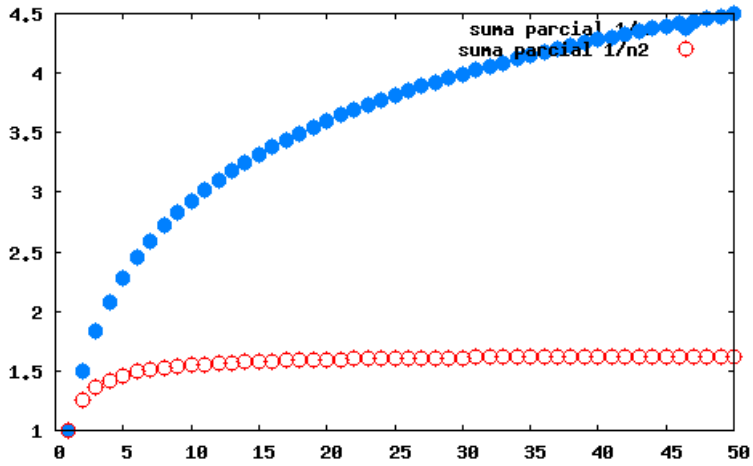
Notar que `suma1[n]` y `suma2[n]` son las sumas parciales n-ésimas de las series anteriores. Podemos calcular una lista de términos significativos. En primer lugar cambiamos a modo númerico:

```
(%i) numer:true;


(%i) create_list(float(suma1[n]),n,[10,100,1000,10000,100000]);
(%o) [2.928968253968254,5.187377517639621,7.485470860550345,
9.787606036044382 12.09014612986334]


(%i) create_list(float(suma2[n]),n,[10,100,1000,10000,100000]);
(%o) [1.549767731166541,1.634983900184892,1.643934566681561,
1.644834071848065, 1.644924066898242]
```

Se puede observar que en el primer caso la suma de la serie no parece estabilizarse (sabemos que la serie es divergente). Sin embargo en el segundo caso se observa que la sucesión e se estabiliza.

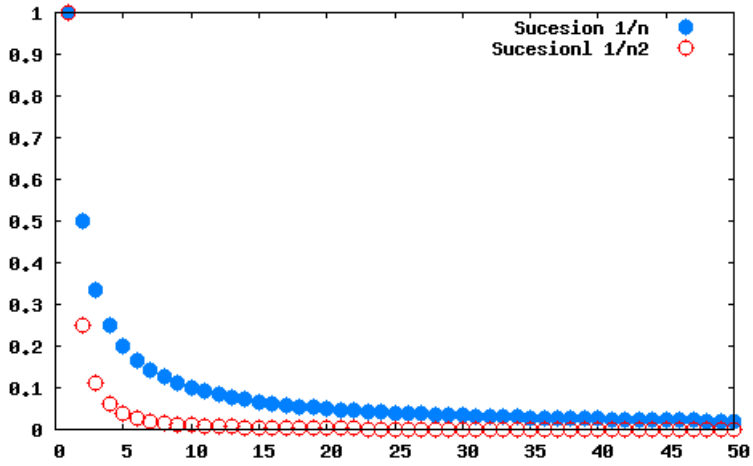
También podemos dibujar términos de ambas sucesiones:

```
(%i) termino1:makelist([n,a1[n]],n,1,50)$
(%i) termino2:makelist([n,a2[n]],n,1,50)$
(%i) wxplot2d([[discrete,termino1],[discrete,termino2]],[style,points,points],
[legend,"Sucesion 1/n", "Sucesion 1/n2"]);
```



O los térmnos generales de la serie:

```
(%i) lista1:makelist([n,suma1[n]],n,1,50)$
(%i) lista2:makelist([n,suma2[n]],n,1,50)$
(%i) wxplot2d([[discrete,lista1],[discrete,lista2]],[style,points,points],
[legend,"Suma parcial 1/n", "Suma parcial 1/n2"]);
```



En este aso observamos que el término general de ambas sucesiones tiene límite cero. Por último, dado que es conocido que $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$, podemos calcular el error cometido al aproximar la suma de la serie por su sucesión de series parciales:

```
(%i) create_list(%pi^2/6 - suma2[k],k,[10,100,1000,10000]);
(%o)[.09516633568168564,.009950166663333482,
9.995001666665004*10e-4, 9.999500016655283*e-5]
```

4.3.2. Criterios de convergencia

Los criterios de convergencia permiten determinar el carácter de una serie a partir del de series ya conocidas, o mediante el cálculo de determinados límites que dependen el término general de la serie. A continuación Los más importantes son los siguientes:

- **Comparación por paso al límite:** Dadas dos series

$$\sum_{n=1}^{\infty} a_n \qquad \text{y} \qquad \sum_{n=1}^{\infty} b_n$$

de términos positivos. Si existe el límite:

$$\lim_{n\to\infty} \frac{a_n}{b_n} = \lambda$$

resulta que:

  - Si $\lambda \in \mathbb{R}^+$; $\lambda \neq 0$ las series tienen el mismo caracter (ambas convergen o ambas divergen).
  - Si $\lambda = 0$, si la serie $\sum_{n=1}^{\infty} b_n$ converge también converge la serie $\sum_{n=1}^{\infty} a_n$. O equivalentemente, si la serie $\sum_{n=1}^{\infty} a_n$ diverge también lo hace la serie $\sum_{n=1}^{\infty} b_n$
  - Si $\lambda = \infty$, si la serie $\sum_{n=1}^{\infty} a_n$ converge también lo hará la serie $\sum_{n=1}^{\infty} b_n$. Equivalentemente, si la serie $\sum_{n=1}^{\infty} b_n$ diverge también diverge la serie $\sum_{n=1}^{\infty} a_n$.

- **Criterio del cociente** Dada la serie $\sum_{n=1}^{\infty} a_n$ de términos positivos. Sea

$$\lim_{n\to\infty} \frac{a_{n+1}}{a_n} = \lambda$$

entonces:

  - Si $\lambda < 1$, la serie converge.
  - Si $\lambda = 1$, el criterio no decide.
  - Si $\lambda > 1$, la serie diverge.

- **Criterio de Raabe** Dada la serie $\sum_{n=1}^{\infty} a_n$ de términos positivos. Sea

$$\lim_{n\to\infty} n\left(1 - \frac{a_{n+1}}{a_n}\right) = \lambda$$

entonces:

  - Si $\lambda < 1$, la serie diverge.
  - Si $\lambda = 1$, el criterio no decide.
  - Si $\lambda > 1$, la serie converge.

- **Criterio de la raíz** Dada la serie $\sum_{n=1}^{\infty} a_n$ de términos positivos. Sea

$$\lim_{n\to\infty} \sqrt[n]{a_n} = \lambda$$

entonces:

  - Si $\lambda < 1$, la serie converge.
  - Si $\lambda = 1$, el criterio no decide.
  - Si $\lambda > 1$, la serie diverge.

Para mostrar como utilizar estos criterios con `Maxima` resolvermos el siguiente ejercicio:

Ejercicio. Estudiar el carácter de las siguientes series:

$$a)\sum_{n=1}^{\infty}\sin\frac{1}{n} \qquad b)\sum_{n=1}^{\infty}\frac{n+1}{(n+2)n!}$$

$$c)\sum_{n=1}\frac{4^n n!}{((n+1)(n+2)\dots(2n+1)} \qquad d)\sum_{n=2}^{\infty}\frac{1}{n\log n}$$

a) Aplicamos el criterio de comparación por paso al límite y comparamos con $\sum_{n=1}^{\infty}\frac{1}{n}$ que sabemos que es divergente:

```
(%i) limit(sin(1/n) / (1/n), n, infty)
(%o) 1
```

Por tanto deducimos que $\sum_{n=1}^{\infty}\frac{1}{n}$ es divergente.

b) Utilizamos el criterio del cociente:

```
(%i) kill(a)$
(%i) a[n]:=(n+1)/((n+2)*n!);
(%i) limit(a[n+1]/a[n], n, infty)
(%o) 0
```

Por tanto la serie es convergente.

c) Escribimos la serie utilizando la instrucción `product` cuya sintaxis es la misma que la de `sum`:

```
(%i) kill(a)$
(%i) a[n]:= (4^ n n!)/(product(k+1,k,n,2*n)$
```

Utilizando la instrucción `simplify_product` simplificamos la expresión a_{n+1}/a_n (es necesario haber cargado el paquete `simplify_sum`)

```
(%i) c: simplify_product(a[n+1]/a[n]);
(%o) 4 (n+1)!^2 (2 n+1)!/(n!^2 (2 n+3)!)
```

Utilizano el criterio del cociente:

```
(%i) limit(c,n,inf)
(%o) 1
```

no obtenemos ninguna conclusión. Utilizando el criterio de Raabe:

```
(%i) limit(n*(1-c),n,inf)
(%o) 1/2
```

por tanto, la serie diverge.

d) Utilizamos el criterio de la raíz:

```
(%i) kill(a)$
(%i) a[n]:=1/(n *log n);
(%i) limit(a[n]^ (1/n), n, infty)
(%o) 0
```

Luego la serie convergente.

4.4. Ejercicios

1. Comprueba la siguiente identidad:

$$1^3 + 2^3 + \ldots + n^3 = (1 + 2 + 3 + \ldots + n)^2$$

2. Determinar el caracter de las siguientes series:

a) $\displaystyle\sum_{n=1}^{\infty} \frac{3n+1}{3^n}$ b) $\displaystyle\sum_{n=1}^{\infty} \log\left(1 + \frac{1}{n^2}\right)$

c) $\displaystyle\sum_{n=1}^{\infty} \frac{n^2}{\sqrt{2^n}}$ d) $\displaystyle\sum_{n=1}^{\infty} \frac{n!2^n}{(n+1)(n+2)\ldots(2n+1)}$,

g) $\displaystyle\sum_{n=1}^{\infty} \frac{3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2n-1)}{2 \cdot 4 \cdot 6 \cdot \ldots \cdot (2n)}$ h) $\displaystyle\sum_{n=1}^{\infty} \left(\frac{2n+1}{3n+1}\right)^n$

3. Estudiar la convergencia y sumar, cuando sea posible, las siguientes series:

a) $\displaystyle\sum_{n=1}^{\infty} \frac{2^n + 3^n}{6^n}$ b) $\displaystyle\sum_{n=1}^{\infty} \left(\frac{e}{3}\right)^n$

c) $\displaystyle\sum_{n=1}^{\infty} \frac{2}{n^2 + 5n + 6}$ d) $\displaystyle\sum_{n=1}^{\infty} \frac{2n+3}{n(n+1)} \frac{1}{3^n}$

4. En 1735, Euler escribió con gran alegría: *Contra todo pronóstico, he encontrado una expresión elegante para la suma de la serie*

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \ldots$$

.

Efectivamente, el resultado era fascinante: una suma de números racionales, en cuyo cálculo habían fracasado los mejores matemáticos del siglo XVII, dependía de la cuadratura del círculo. En concreto, el resultado era $\frac{\pi^2}{6}$.

- Utiliza Maxima para calcular la suma de los 1000 primeros términos.
- Utiliza el resultado anterior para calcular una aproximación de π utilizando el resultado de Euler.

5. Hallar el carácter de las siguientes series:

$$a) \sum_{n=1}^{\infty} \left[\left(\frac{n+1}{n}\right)^n - \frac{2n}{n+1}\right]^{-n} \qquad b) \sum_{n=1}^{\infty} \frac{n^4[\sqrt{3} + (-1)^n]^n}{5^n}$$

6. Sumar la siguientes series:

$$a) \sum_{n=2}^{\infty} \frac{2n+1}{10^n} \qquad b) \sum_{n=1}^{\infty} \frac{2^{n+3}}{3^n}$$

$$c) \sum_{n=4}^{\infty} \frac{1}{4n^2 - 1} \qquad d) \sum_{n=4}^{\infty} \frac{1}{(4n-1)(4n+3)}$$

Capítulo 5

Funciones de una variable

5.1. Introducción

Dedicaremos esta práctica a funciones reales de una variable real.

En la primera sección se introducirán los comandos que `wxmaxima` incorpora para la representación gráfica de funciones. A continuación, revisaremos las funciones elementales: exponencial, logaritmo y funciones trigonométricas. Finalizaremos la primera sección con algunos comentarios sobre simetrías y traslación de funciones.

En la segunda sección, veremos las herramientas de las que dispone `wxmaxima` para derivar funciones. Como aplicación calcularemos los extremos relativos (máximos y mínimos) de funciones de una variable.

Por último utilizaremos `wxmaxima` para calcular desarrollos de Taylor.

5.2. Gráficas de funciones

Recordemos en primer lugar, que para definir una función en `wxmaxima` se procede del siguiente modo:

```
nombre(variable) : = expresión
```

Por ejemplo, para definir la función $f(x) = x^3 - x + 3$, escribiríamos:

```
(%i) f(x) := x^ 3-x+3
```

`wxMaxima` dispone de dos funciones para representar funciones de una variable: `wxplot2d` y `wxdraw2d`. La primera de ellas es básica y permiter representar funciones que aparezcan en forma explicita, $y = f(x)$, la segunda opción es mas compleja y permite representar funciones dadas en forma paramétrica , $(x(t), y(t))$, o implícita, $f(x, y) = 0$.

Nota: Recordar que el prefijo `wx` inserta el objeto gráfico en el documento de `wxmaxima`, mientras que si se prescinde de él, el gráfico se desplegará en una ventana adicional.

5.2.1. Comando `wxplot2d`/`plot2d`

La sintaxis del comando `wxplot2d` es la siguiente:

```
wxplot2d(f(x),[x, a, b])
```

donde

`f(x)`: función a dibujar dependiente de x.

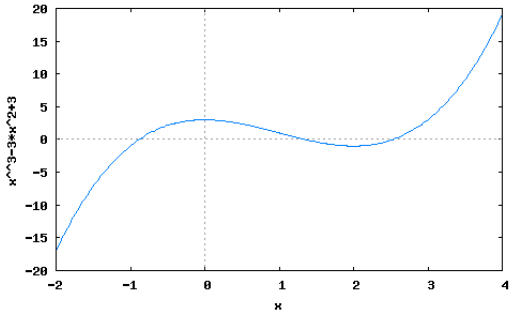
`x` : variable independiente de la función.

`a` : extremo inferior del intervalo.

`b` : extremo superior del intervalo.

Por ejemplo, para dibujar la función $y(x) = x^3 - 3x^2 + 3$, en el intervalo $[-2, 4]$ procedemos:

```
(%i) f(x) := x^3-3*x^2 +3$
(%i) wxplot2d(f(x),[x,-2,4]);
```

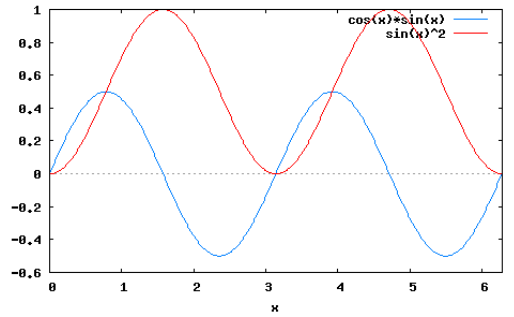


También es posible dibujar varias funciones sobre el mismo intervalo. En este caso la sintasis seria:

```
wxplot2d([f1(x),f2(x),f3[x],... ] ,[x, a, b])
```

Por ejemplo, si queremos dibujar las funciones, $y = \cos(x)\sin(x)$, $y = \sin^2(x)$, en el intervalo $[0, 2\pi]$:

```
(%i) wxplot2d([cos(x)*sin(x), sin(x)^2],[x, 0, 2*%pi])
```



5.2.2. Comando `wxdraw2d`/ `draw2d`

La sintaxis del comando `wxdraw2d` es la siguiente:

```
draw2d(opciones, objeto)
```

donde

Opciones : comandos que modifican el gráfico. Todas ellas tienen valores por defecto. Algunas de las más útiles son las siguientes.

- **color**: color de la gráfica.

- **line_width**: grosor con el que se dibujan las líneas.
- **nticks**: es el número de puntos para dibujar la gráfica. Por defecto es 30. Es conveniente modificar este valor, si se quiere aumentar la precisión de la gráfica.
- **title**: título de la ventana.

Objeto : el objeto gráfico puede ser:

- **explicit(f(x), x, a, b)** : comando para dibujar una función explícita $y = f(x)$ en el intervalo $[a, b]$.
- **parametric(x(t), y(t), param, a, b)**: expresión para dibujar una función $(x(t), y(t))$ donde el parámetro t varia en el intervalo $[a, b]$.
- **implicit(F(x,y)=0, x, a, b, y, c, d)**: comando para dibujar una función con ecuación ímplicita $F(x, y) = 0$, donde $x \in [a, b]$, e $y \in [c, d]$.

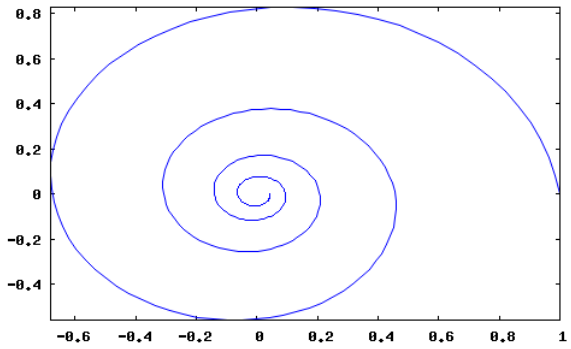
No comentaremos el objeto `explicit`, pues este tipo de curvas se pueden dibujar con el comando `wxplot2d`.

Para dibujar la espiral, dada por las ecuaciones paramétricas:

$$x(t) = e^{-t/8}\cos(x) \qquad y(t) = e^{-t/8}\sin(x) \qquad t \in [0, 2 * pi]$$

escribiriamos:

```
(%i) x(t):=%e^(-t/8)*cos(x)$
(%i) y(t):=%e^(-t/8)*sin(x)$
(%i) wxdraw2d(nticks=500, parametric(x(t),y(t),t,0,8*%pi));
```

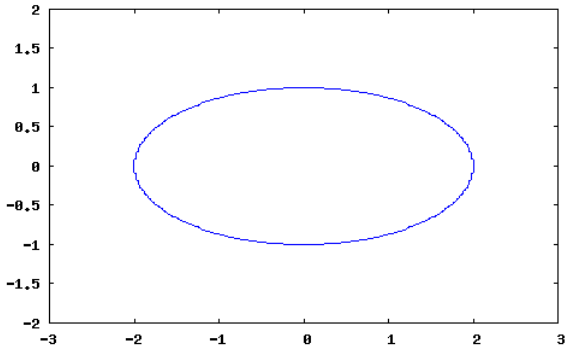


Por otro lado, para dibujar la curva (elipse) dada de forma implícita:

$$\frac{x^2}{4} + y^2 = 1$$

se procede:

```
(%i) kill(x)$ kill(xy)$
(%i) wxdraw2d(nticks=500,implicit(x^2/4 + y^2 = 1, x, -3,3,y,-2,2));
```



5.2.3. Funciones elementales

Revisaremos en esta sección la función exponencial, la función logaritmo, y las funciones trigonométricas seno y coseno.

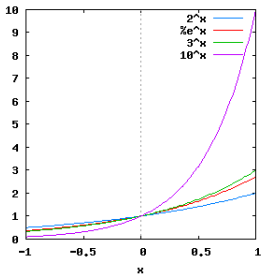
Función exponencial

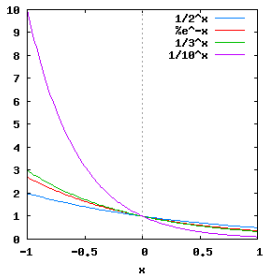
Si $a > 0$ la función $f : \mathbb{R} \to \mathbb{R}$ definida por $f(x) = a^x$ se denomina función exponencial.

Propiedades:

- Dominio: $\mathbb{R}$.
- Imagen: $(0; \infty)$ si $a \neq 1$ y $\{1\}$ si $a = 1$.
- La función exponencial es creciente si $a > 1$ y decreciente si $0 < a < 1$.
- La función exponencial es continua.
- $a^{x+y} = a^x a^y$, $x, y \in \mathbb{R}$.
- $(a^x)^y = a^{xy}$, $x, y \in \mathbb{R}$.
- $$\lim_{x\to\infty} a^x = \begin{cases} +\infty & \text{si } a > 1 \\ 0 & \text{si } a < 1 \\ 1 & \text{si } a = 1 \end{cases} \qquad \lim_{x\to-\infty} a^x = \begin{cases} 0 & \text{si } a > 1 \\ +\infty & \text{si } a < 1 \\ 1 & \text{si } a = 1 \end{cases}$$
- La función $f(x) = a^x$ con $a > 0, a \neq 1$ es biyectiva, por tanto tiene inversa.

Su respresentación gráfila es la siguiente:





Función logaritmo

El logaritmo es base a, $(a > 0, a \neq 1)$, de un número x es el único número $y \in \mathbb{R}$ tal que $a^x = y$:

$$\log_a x = y \Leftrightarrow a^x = y$$

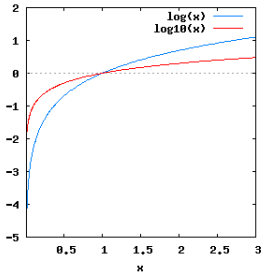
Si $a = e$ se denomina **logaritmo neperiano o natural** y lo denotaremos por $\log x$.

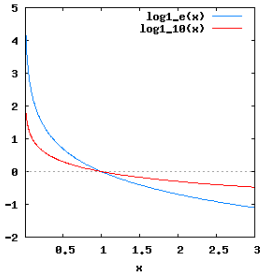
Propiedades

- Dominio $(0, +\infty)$.
- Imagen: $\mathbb{R}$.
- La función logaritmo es creciente si $a > 1$ y decreciente si $a < 1$.
- La función logaritmo es continua.
- $\log_a(xy) = \log_a x + \log_a y$.
- $\log_a(x/y) = \log_a x - \log_a y$.
- $\log_a x^y = y \log_a x$.
- $\log_a 1 = 0$
-

$$\lim_{x \to 0^+} \log_a x = \begin{cases} +\infty & \text{si } a > 1 \\ -\infty & \text{si } a < 1 \end{cases} \qquad \lim_{x \to +\infty} \log_a x = \begin{cases} +\infty & \text{si } a > 1 \\ -\infty & \text{si } a < 1 \end{cases}$$

A continuación figuran las respresentaciones gráficas de $y = \log(x)$ e $y = \log_{10}(x)$ a la izquierda e $y = \log_{1/e}(x)$, e $y = \log_{1/10} x$ a la derecha.





Nota: `wxMaxima` sólo proporciona la función logaritmo neperiano o natural (`log`). Para trabajar con otro logaritimo se puede emplear la equivalencia:

$$\log_a(x) = \frac{\log(x)}{\log(a)}$$

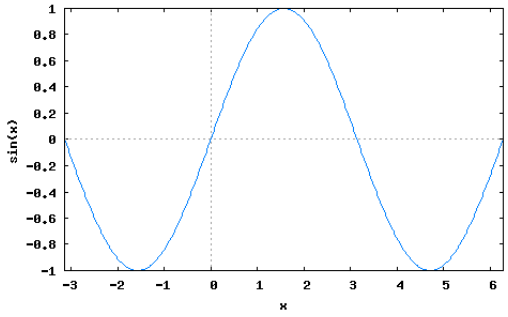
Funciones trigonométricas

Función seno: Denotaremos por $f(x) = \sin(x)$ a la función seno, que tiene las siguientes propiedades:

- Dominio $\mathbb{R}$.
- Imagen $[-1, 1]$
- Es periódica de periodo 2π, es decir $\sin(x + 2\pi) = \sin(x)$.

- Es una función impar, es decir $\sin(x) = -\sin(-x)$.
- Es creciente $(0, \frac{\pi}{2}) \cup (\frac{3\pi}{2}, 2\pi)$ y decreciente en $(\frac{\pi}{2}, \frac{3\pi}{2})$.
- Es una función continua.

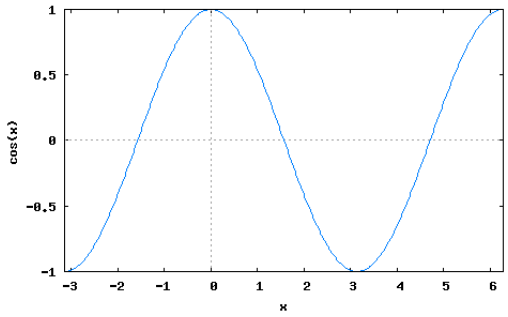
Su representación gráfica es la siguiente:



Función coseno: Denotaremos por $f(x) = \cos(x)$ a la función coseno, que tiene las siguientes propiedades:

- Dominio $\mathbb{R}$.
- Imagen $[-1, 1]$
- Es periódica de periodo 2π, es decir $\cos(x + 2\pi) = \cos(x)$.
- Es una función par, es decir $\cos(x) = \cos(-x)$.
- Es creciente $(\pi, 2\pi)$ y decreciente en $(0, \pi)$.
- Es una función continua.

Su representación gráfica es la siguiente:



5.2.4. Simetrías y traslaciones

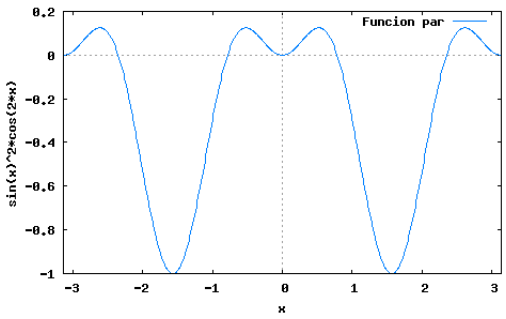
Simetrías: Dada una función $f(x) : \mathbb{R} \to \mathbb{R}$, se denomina

- Función simétrica respecto al **Eje OX** a la función $-f(x)$.
- Función simétrica respecto al **Eje OY** a la función $f(-x)$.
- Función simétrica respecto al **Origen** a la función $-f(x)$.

Una función se dice **par** si es simétrica respecto al eje OX, es decir

$$f(x) = f(-x).$$

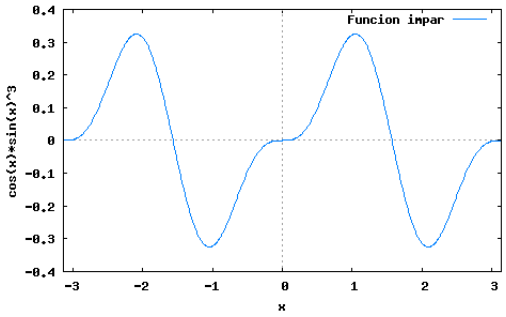
Un ejemplo de función par sería:



Una función se dice **impar** si es simétrica respecto al origen, es decir

$$f(x) = -f(-x).$$

Un ejemplo de función impar sería:



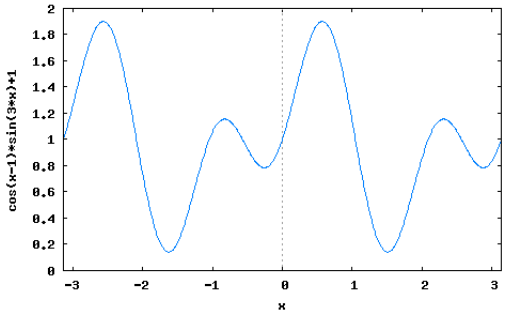
Ejemplo: Dada la función $f(x) = 1 + sin(3x)cos(x-1)$, utilizar `wxmaxima` para calcular las funciones simétricas de $f(x)$ respecto al eje OX, al eje OY y al origen en el intervalo $[-\pi, \pi]$.

En primer lugar definimos la función en `wxmaxima`:

```
(%i) f(x):= 1 +sin(3*x)*cos(x-1) ;
```

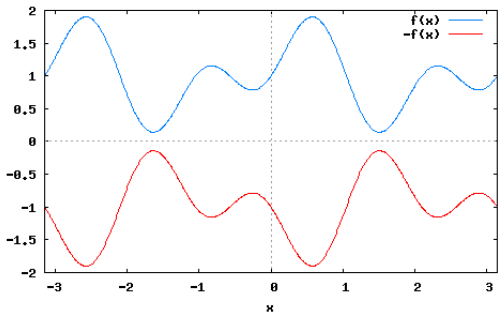
A continuacion, dibujamos la función

```
(%i) wxplot2d(f(x),[x,-%pi,%pi]);
```



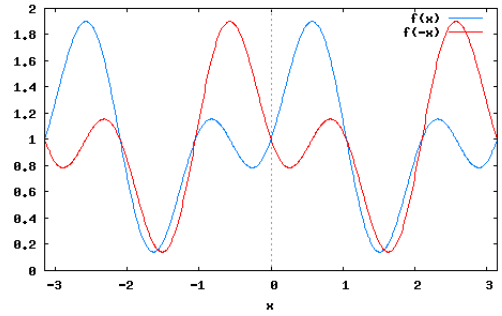
La función junto con su simetrica con respecto al eje OX:

```
(%i) wxplot2d([f(x),-f(x)],[x,-%pi,%pi]);
```



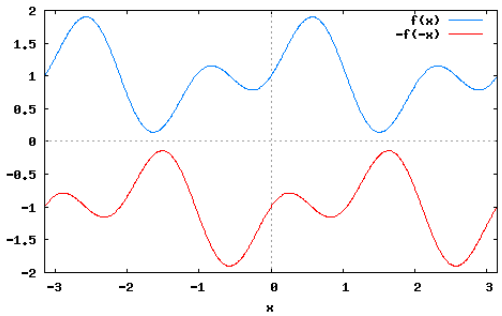
La función junto con su simétrica con respecto al eje OY:

```
(%i) wxplot2d([f(x),f(-x)],[x,-%pi,%pi]);
```



La función junto con su simétrica con respecto al origen:

```
(%i) wxplot2d([f(x),-f(-x)],[x,-%pi,%pi]);
```



Traslaciones: Dada una función $f(x) : \mathbb{R} \to \mathbb{R}$,

- la función $f(x - a)$ es una **traslación vertical** de $f(x)$ de a unidades.
- la función $a + f(x)$ es una **traslación horizontal** de $f(x)$ de a unidades.

Ejemplo. Dada la función $f(x) = \log(x^2 + 1)$:

a) Calcular traslaciones verticales de $f(x)$ de 1 y 2 unidades.

b) Calcular traslaciones horizontales de $f(x)$ de 1 y 2 unidades.

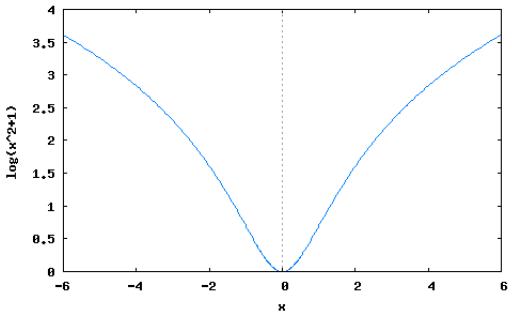
c) Calcular la traslación de $f(x)$ de 2 unidad en dirección vertical, y -3 unidades en la dirección horizontal.

Como en el caso anterior, en primer lugar, introducimos la función en `wxMaxima`:

```
(%i) kill(f)$
(%i) f(x):=log(x^2+1) ;
```

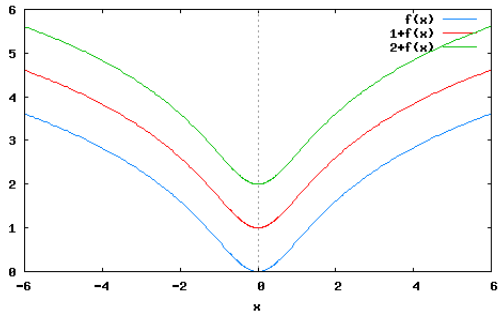
Dibujamos la función

```
(%i) wxplot2d(f(x),[x,-5,5]);
```



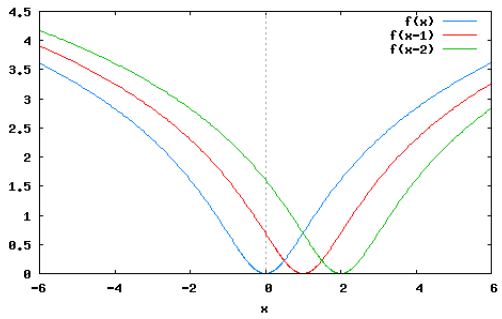
Para resolver el apartado a) basta con dibujar las funciones $1+f(x)$ y $2+f(x)$:

```
(%i) wxplot2d([f(x),1 +f(x),2 + f(x)],[x,-6,6], [legend,["f(x)","1+f(x)",
"2+f(x)"]]);
```



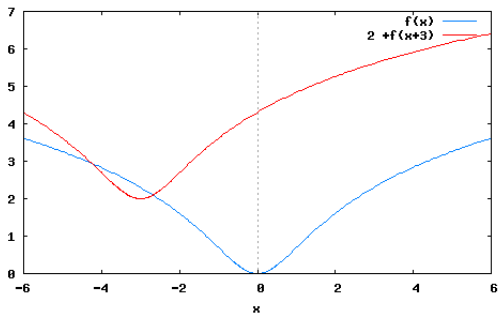
Para resolver el apartado b) basta con dibujar las funciones $f(x+1)$ y $f(x+2)$:

```
(%i) wxplot2d([f(x),f(x-1),f(x-2)],[x,-6,6], [legend,["f(x)","f(x-1)",
"f(x-2)"]]);;
```



Por último, la solución del apartado c) será: $2 + f(x - (-3)) = 2 + f(x + 3)$

```
wxplot2d([f(x),2 + f(x+3)],[x,-6,6], [legend,["f(x)","2 +f(x+3)"]]);
```



5.3. Derivadas

En esta sección describiremos el comando del que dispone `wxMaxima` para calcular derivadas: `diff`. El objetivo es resolver problemas de optimización, por tanto, precisamos calcular ceros de funciones. Para ello, utilizaremos los comandos: `solve`, `find_root`.

5.3.1. Comando `diff`

La sintaxis de este comando es la siguiente:

```
diff(expr, variable, n)
```

donde

`expr` : función a derivar.

`variable` : variable con respecto a la que se deriva.

`n` : orden de la derivada. Por defecto este valor es 1.

Por ejemplo, para calcular la derivada de la función $y = \frac{x}{x^2+1}$

```
(%i) diff(x/(x^2+1),x);
(%o)
```

$$\frac{1}{x^2+1} - \frac{2\,x^2}{\left(x^2+1\right)^2}$$

Si deseamos simplificar la expresión, podemos aplicar la función `ratsimp`:

```
(%i) ratsimp(diff(x/(x^2+1),x));
(%o)
```

$$-\frac{x^2-1}{x^4+2\,x^2+1}$$

`wxMaxima` permite calcular derivadas de cualquier orden. Por ejemplo, para calcular la derivada octava de la función $y = e^{x^2}$:

```
(%i) diff(%e^(x^2),x,8);
(%o)
```

$256\,x^8\,e^{x^2} + 3584\,x^6\,e^{x^2} + 13440\,x^4\,e^{x^2} + 13440\,x^2\,e^{x^2} + 1680\,e^{x^2}$

5.3.2. Comandos `solve` / `find_root`/ `allroots`

La sintaxis del comando es la siguiente:

```
solve(ecuación, incógnita)
```

donde

`ecuación` : ecuación a resolver.

`incógnita` : incógnita de la ecuación.

Por ejemplo, para resolver la ecuación $x^2 - 3x + 1 = 0$ escribiremos:

```
(%i) solve(x^2 - 3*x+1= 0, x);
(%o)
```

$$[x = -\frac{\sqrt{5}-3}{2}, x = \frac{\sqrt{5}+3}{2}]$$

`wxMaxima` puede resolver ecuaciones que dependen de parámetros:

```
(%i) solve(a*x^2 + b*x + c= 0, x);
(%o)
```

$$[x = -\frac{\sqrt{b^2 - 4\,a\,c}+b}{2\,a}, x = \frac{\sqrt{b^2 - 4\,a\,c}-b}{2\,a}]$$

Sin embargo, este comando que trabaja en forma simbólica, no siempre proporciana los ceros de las funciones. Por ejemplo, si tratamos de calcular las raíces de $x5 + x4 - 11x3 - 9x2 + 18x + 10$ obtenemos:

```
(%i) kill(f)$
(%i) f(x):=x^5+x^4-11*x^3-9*x^2+18*x+10$
(%i) solve(f(x)= 0, x);
(%o)
```

$$[0 = x^5 + x^4 - 11\,x^3 - 9\,x^2 + 18\,x - 10]$$

En estas ocasiones, es necesario recurrir a funciones que implementan procedimientos numéricos. Por ejemplo, la función `find_root`, que requiere además como argumentos, los extremos del intervalo donde se busca la solución:

```
find_root(ecuación, incognita, inf, sup)
```

donde

`ecuación` : ecuación a resolver.

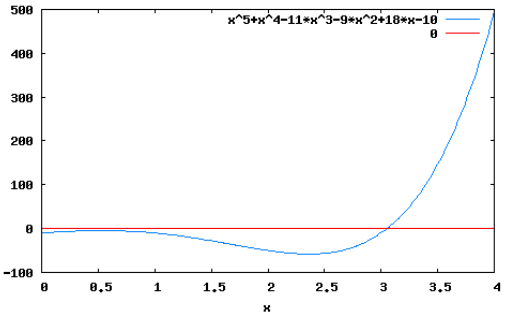
`incognita` : incognita de la ecuación.

`inf` : extremo inferior del intervalo.

`sup` : extremo superior del intervalo.

En el caso anterior, en primer lugar dibujamos la función para localizar las raíces:

```
(%i)wxplot2d([f(x),0],[x,0,4]);
```



De donde deducimos, que hay una raíz en el intervalo $[2, 4]$:

```
(%i)find_root([f(x)=0,x,[2,4]);
(%o)3.052211666355731
```

Cuando se trata de calcular **raíces de polinomios** la opción más rápida es utilizar el comando `allroots`:

```
allroots(polinomio)
```

donde

`polinomio` : polinomio del que se requiren las raíces

Aplicado al ejemplo anterior:

```
(%i)allroots([f(x));
(%o)[x=.4295082511125006*%i+.5567660971265102,
x=.5567660971265102-.4295082511125006*%i,
x=-2.370119928219864,x=-2.795623932388887,x=3.05221166635573]
```

5.3.3. Cálculo de máximos y mínimos

Para calcular los extremos relativos de una función $f(x)$, se deben localizar los puntos críticos:

$$f'(x) = 0 \quad \Rightarrow x_1, x_2, \ldots, x_r$$

y a continuación evaluar esos puntos en la segunda derivada:

$$f''(x_i) = \begin{cases} > 0 & \text{mínimo } (x_i, f(x_i) \\ < 0 & \text{máximo } (x_i, f(x_i) \end{cases}$$

En este tipo de problemas, la función derivada se reutiliza para evaluar, por ello es conveniente utilizar la función `define`, para definir las derivadas:

```
define(función, definicón)
```

Por ejemplo, para calcular los extremos relativos de la función $x^5 + x^4 - 11x^3 - 9x^2 + 18x + 10$ procedemos del siguiente modo:

Definimos la función y sus derivadas:

```
(%i) kill(f)$
(%i) f(x):=x^5+x^4-11*x^3-9*x^2+18*x+10;
(%o)
```

$$x^5 + x^4 - 11\,x^3 - 9\,x^2 + 18\,x - 10$$

```
(%i) define(df(x), diff(f(x),x));
(%o)
```

$$\mathrm{df}\,(x) := 5\,x^4 + 4\,x^3 - 33\,x^2 - 18\,x + 18$$

```
(%i) define(d2f(x),diff(f(x),x,2));
(%o)
```

$$\mathrm{d2f}\,(x) := 20\,x^3 + 12\,x^2 - 66\,x - 18$$

Calculamos las raíces de la derivada (por ser un polinomio utilizamos `allroot`) y las almacenamos en la variable `sol`:

```
(%i) sol:allroot(df(x));
(%o) [x=.5338613537430753,x=-1.096856508567827,
x=2.363816272329865,x=- 2.600821117505113]
```

Evaluamos cada una de las cuatro soluciones, y decidimos el carácter del punto crítico:

```
(%i) df2(x),sol[1]
(%o)-46.77165945967596

(%i) df2(x),sol[2]
(%o)42.43722588782085

(%i) df2(x),sol[3]
(%o)157.2021444450348

(%i) df2(x),sol[4]
(%o)-117.0277108731798
```

Por tanto:

- (0,533, f(0,533)) y (2,600, f(2,600)) son máximos relativos.
- ($-$1,096, $f(-1{,}096)$) y (2,363, f(2,363) son mínimos relativos.

Nota: Para evaluar la derivada segunda hemos utilizado una regla de transformación. Observar que cada una de las componentes del vector solución `sol` es de la forma, $x = val$. Al escribir

```
(%i) df2(x),sol[4]
(%o)-117.0277108731798
```

estamos diciendo a `wxMaxima` que primero evalúe $df2(x)$, y después sustituya el valor de x, por el valor de val.

Otra posibilidad es evaluar la derivada segunda directamente en el punto:

```
(%i) df2(-2.600821117505113)
(%o)-117.0277108731798
```

5.4. Desarrollos de Taylor

5.4.1. Comando `taylor`

El desarrollo de Taylor de una función $f(x)$ en el punto x_0 de orden n es un polinomio de grado n que se define como

$$T_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f^2(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^n(x_0)}{n!}(x - x_0)^n$$

En `wxMaxima` se incluye el comando `taylor` para calcular desarrollos de Taylor:

```
taylor(expr, x, x0, n)
```

donde

`expr` : función de la que se calculará el desarrollo.

`x` : variable de la función.

`x0` : punto entorno al cual se realiza el desarrollo.

`n` : orden del desarrollo.

Por ejemplo, para calcular el desarrollo de Taylor de orden 9 de la función $y = \sin(x)$ en el punto $x_0 = 0$:

```
(%i) taylor(sin(x),x, 0, 9)
(%o) /T/ x - x^3/6 + x^5/120 - x^7/5040 + x^9/362880 + ...
```

`wxMaxima` devuelve el polinomio de Taylor en un formato especial (observar /T/). Si se quiere convertir a polinomio es necesario utilizar el comando `taytorat`

```
(%i) taytorat(taylor(sin(x),x, 0, 9) )
(%o)
```

$$\frac{x^9 - 72\,x^7 + 3024\,x^5 - 60480\,x^3 + 362880\,x}{362880}$$

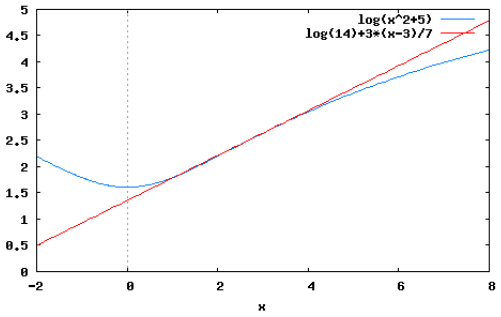
La recta tangente a la curva $y = f(x)$ en el punto x_0 coincide con el polinomio de Taylor de grado $n = 1$ el punto x_0. Por ejemplo, la recta tangente a la curva $y = \log(x^2 + 5)$ en el punto $x = 3$ se puede calcular:

```
(%i) taylor(log(x^2 +5) ,x, 3, 1)
(%o)
```

$$\log(14) + \frac{3\,(x-3)}{7}$$

Podemos dibujar esta recta junto con la curva original:

```
(%i) wxplot2d([log(x^2+5), taylor(log(x^2 +5) ,x, 3, 1)],[x,0,5])
```



Para finalizar vamos a calcular sucesivas aproximaciones de Taylor de la función $y = 1 + \cos^2(x)$ en el punto $x = 0$:

```
(%i) f(x):=1-cos(x)^2$
```

```
(%i) define(t2(x),taylor(f(x),x,0,2));
(%o)
```

$$\mathrm{t2}\,(x) := x^2$$

```
(%i) define(t4(x),taylor(f(x),x,0,4));
(%o)
```

$$\mathrm{t4}\,(x) := x^2 - \frac{x^4}{3}$$

```
(%i) define(t6(x),taylor(f(x),x,0,6));
(%o)
```

$$\mathrm{t6}\,(x) := x^2 - \frac{x^4}{3} + \frac{2\,x^6}{45}$$

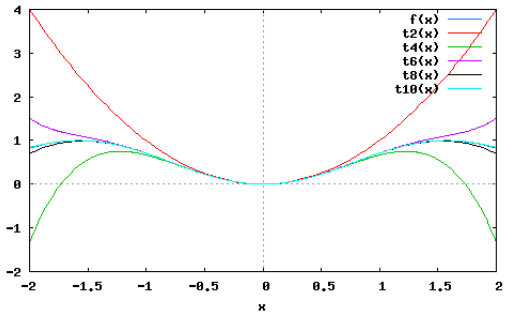
```
(%i) define(t8(x),taylor(f(x),x,0,8));
(%o)
```

$$\mathrm{t8}\,(x) := x^2 - \frac{x^4}{3} + \frac{2\,x^6}{45} - \frac{x^8}{315}$$

```
(%i) define(t10(x),taylor(f(x),x,0,10));
(%o)
```

$$\mathrm{t10}\,(x) := x^2 - \frac{x^4}{3} + \frac{2\,x^6}{45} - \frac{x^8}{315} + \frac{2\,x^{10}}{14175}$$

```
(%i) wxplot2d([f(x),t2(x),t4(x),t6(x),t8(x),t10(x)],[x,-2,2],
[legend,["f(x)","t2(x)","t4(x)","t6(x)","t8(x)","t10(x)"]]);
```



5.5. Integración

En esta sección describiremos las herramientas de las que dispone `wxMaxima` para el calculo de integrales: `integrate`. Finalizaremos esta apartado con breve comentario sobre integración numérica.

5.5.1. Integrales indefinidas: `integrate`

El comando que permite calcular primitivas en `wxMaxima` es `integrate`, cuya sintaxis es la siguiente:

```
integrate(expr, variable)
```

donde

`expr` : función a integrar.

`variable` : variable con respecto a la que se integra.

Por ejemplo, para calcular una primitiva de la función $f(x) = x^2 \sin(x)$

```
(%i) integrate(x^2 *sin(x), x);
(%o)
```

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

La función `integrate` funciona aplicando reglas de transformación, de forma similar a como funciona `diff`. Sin embargo, así como para la derivación existen principios generales, para la integración no existe un procedimiento sistemático de resolución. Podemos encontrarnos además con funciones para las cuales la primitiva no puede ser expresada en términos de funciones elementales. `wxMaxima` resuelve dos tipos de integrales:

- Integrales cuyo integrando es combinación de funciones elementales y que pueden ser resueltas en términos de funciones elementales. Entendemos por funciones elementales: funciones racionales, exponenciales, logaritmos, funciones trigonométricas e inversas de funciones trigonométricas.

- Integrales que, por su importancia en la economía, estadísitica, física o en otras ciencias, se expresan en términos de funciones especiales (tales como las funciones error, gamma, beta, ...).

En general `wxMaxima` resuelve la mayoría de las integrales que aparecen en los libros de tablas. Es difícil encontrar funciones para las que nosotros podamos calcular una primitiva y `wxMaxima` a no sea capaz de hacerlo. A continuación vemos unos ejemplos de cálculo de primitivas haciendo un recorrido por los tipos habituales:

- Integración inmediata:

```
(%i) integrate(x^ 3, x);
(%o)
```

$$\frac{x^4}{4}$$

Este tipo de integrales se resuelven mediante una tabla de integración, que es exactamente una tabla de derivadas que se lee en sentido contrario. Alguna de las reglas más básicas son las siguientes:

$$\int [f(x)+g(x)]dx = \int f(x)dx + \int g(x)dx = F(x)+G(x)+K$$
$$\int cf(x)dx = c\int f(x)dx = cF(x)+K$$

$$\int x^n dx = \frac{x^{n+1}}{n+1} + K \quad (n \neq -1) \qquad \int f(x)^n f'(x) dx = \frac{1}{n+1} f(x)^{n+1} + K$$

$$\int \frac{1}{x} dx = \log|x| + K \qquad \int \frac{f'(x)}{f(x)} dx = \log|f(x)| + K$$

$$\int e^x dx = e^x + K \qquad \int f'(x) e^{f(x)} dx = e^{f(x)} + K$$

$$\int a^x dx = \frac{1}{\log a} a^x + K \qquad \int f'(x) a^{f(x)} dx = \frac{1}{\log a} a^{f(x)} + K$$

$$\int \sin x dx = -\cos x + K \qquad \int f'(x) \sin f(x) dx = -\cos f(x) + K$$

$$\int \cos x dx = \sin x + K \qquad \int f'(x) \cos f(x) dx = \sin f(x) + K$$

- Integración por sustitución o cambio de variable:

```
(%i) integrate((3*x^2+1)*%e^(x^3+x), x);
(%o)
```

$$e^{x^3+x}$$

En este tipo de integrales se realiza un cambio de variable

$$x = g(t), \qquad dx = g'(t)dt$$

con el objetivo de simplificar el integrando y obtener una expresión que se corresponda con una integral inmediata o al menos más sencilla:

$$\int f(x)dx = \int f(g(t))g'(t)dt = F(t) + K = F(g^{-1}(x)) + K$$

Los tipos que figuran a continuación no forman parte del temario del curso, en el sentido que no se exigirá resolver integrales de los siguientes tipos sobre papel, sin embargo los enumeramos para mostrar la potencia de `wxMaxima`:

- Integración por partes:

```
(%i) integrate(log(x)^2, x);
(%o)
```

$$x\left(\log(x)^2 - 2\log(x) + 2\right)$$

- Integración de funciones racionales:

```
(%i) integrate((x^5+2*x+1)/(x^4+2*x^2+1), x);
(%o)
```

$$-\log\left(x^2+1\right) + \frac{\operatorname{atan}(x)}{2} + \frac{x-3}{2x^2+2} + \frac{x^2}{2}$$

- Integracion de funciones trigonométricas:

```
(%i) integrate(sin(x)^5 *cos(x)^2, x);
(%o)
```

$$-\frac{15\cos(x)^7 - 42\cos(x)^5 + 35\cos(x)^3}{105}$$

- Integración de funciones trigonométricas racionales.

```
(%i) integrate((4*cos(x)^3-3*cos(x))/sin(x), x);
(%o)
```

$$\frac{\log(\cos(x)+1)}{2} + \frac{\log(\cos(x)-1)}{2} + 2\cos(x)^2$$

- Integrales irracionales:

```
(%i) integrate(sqrt(1-x^2), x);
(%o)
```

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

```
(%i) integrate(1/sqrt(2-x-x^2), x);
(%o)
```

$$-\operatorname{asin}\left(\frac{-2\,x-1}{3}\right)$$

No obstante, es conveniente recordar que existen funciones cuyas primitivas no se pueden expresar en términos de funciones elementales. En estos casos, la respuesta de `wxMaxima` es la siguiente:

```
(%i) integrate((3*x^2+x)*%e^(x^3+x), x);
(%o)
```

$$\int \left(3\,x^2+x\right)\,e^{x^3+x}dx$$

En otros casos, las primitivas por su importancia en la economía o ciencia tiene nombre especial. Por ejemplo, si integramos

$$\int e^{-x^2}dx$$

la respuesta de `wxMaxima` es:

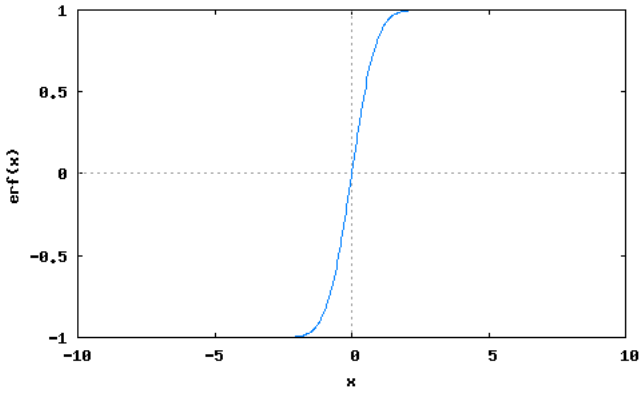
```
(%i) integrate(%e^(-x^2), x);
(%o)
```

$$\frac{\sqrt{\pi}\,\operatorname{erf}(x)}{2}$$

En la ayuda se puede consultar que la función `erf` es la función de error:

$$\operatorname{erf}(x)=\int_0^x \frac{2}{\sqrt{\pi}}e^{-t^2}dt$$

su representación es la siguiente:



Por último, `wxMaxima` permite integrar cuando en las fórmulas aparecen parámetros. Si el resultado de la integral está condicionado por el valor del parámetro, es posible que `wxMaxima` solicite alguna información adicional. Por ejemplo:

```
(%i) integrate(x^n, x);
Is n+1 zero or nonzero? zero;
```
(%o) $\log(x)$

o por el contrario:

```
(%i) integrate(x^n, x);
Is n+1 zero or nonzero? nonzero;
```
(%o) $\frac{x^{n+1}}{n+1}$

5.5.2. Integrales definidas: `integrate`

El comando `integrate` también permite calcular integrales definidas. En este caso la sintaxis es la siguiente:

```
integrate(expr, variable, a,b)
```

donde

`expr` : función a integrar.

`variable` : variable con respecto a la que se integra.

`a` : extremo inferior del intervalo.

`b` : extremo superior del intervalo.

Por ejemplo, el cálculo de

$$\int_0^3 (x^3 + 5x^2 - 2)dx$$

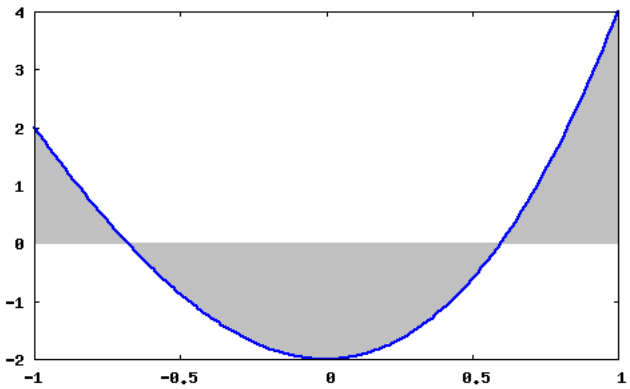
se haría del siguiente modo:

```
(%i) integrate(x^3+5*x^2 - 2, x, 0 ,3);
```
(%o) $\frac{237}{4}$

Para el cáclulo de la integral definida `wxMaxima` primero calcula una primitiva y después utiliza la regla de Barrow, es decir:

$$\int_a^b f(x) = F(b) - F(a),$$

Si bien la derivada de una función esta relacionada con la recta tangente a la curva, la integral definida se interpreta geométricamente como el área (con signo) comprendida entre la función $f(x)$ y el eje OX, en el intervalo $[a, b]$. El área se considera positiva si está por encima del eje OX y negativa si está por debajo. En el ejemplo anterior, el valor $\frac{237}{4}$ corresponde al área con signo, de la región en gris:



Nota: El gráfico anterior se puede obtener del siguiente modo:

```
(%i) f(x):=x^3+5*x^2-2$
(%i) wxdraw2d(filled_func=0, fill_color=grey,
explicit(f(x),x,-1,1), filled_func=false, color=blue, line_width=2,
explicit(f(x),x,-1,1), xaxis=true);
```

5.5.3. Integrales impropias: `integrate`

Si el intervalo $[a, b]$ es acotado y la función f es continua en $[a, b]$, es posible demostrar que la integral $\int_a^b f(x)dx$ toma un valor finito. Cuando el intevalo $[a, b]$ es no acotado o la función f no es continua se habla de integrales impropias. En este caso, es necesario un estudio de convergencia que tiene ciertas analogías con el desarrollo que se hizo con las series.

En general, el cálculo de una integral impropia combina el cálculo de primitivas y límites. Con `wxMaxima` el cálculo es directo utilizando el comando `integrate`, que admite $\pm\infty$ como extremos de integración. Es responsabilidad del usuario interpretar el resultado proporcionado por `wxMaxima`. Veamos algunos ejemplos:

- Integrales impropias de primera especie (intervalo no acotada).

  Si tratamos de calcular la integral:

  $$\int_1^\infty \frac{1}{x}dx$$

  ```
  (%i) integrate(1/x, x, 1 ,inf);
  (%o) defint: integral is divergent.
  - an error. To debug this try: debugmode(true);
  ```

  `wxMaxima` nos indica que la integral es divergente.

  En cambio, si calculamos la integral

  $$\int_1^\infty \frac{1}{x^2}dx$$

  ```
  (%i) integrate(1/x^2, x, 1 ,inf);
  (%o) 1
  ```

  `wxMaxima` devuelve un valor numérico, indicando en particular, que es una integral convergente.

- Integrales impropias de segunda especie (función discontinua).

  Corresponden a este tipo por ejemplo integrales del tipo:

  $$\int_0^1 \frac{1}{x}dx$$

  Notar que $\lim_{x\to 0^+} f(x) = \infty$. La instrucción en `wxMaxima`

  ```
  (%i) integrate(1/x, x, 0,1);
  (%o) defint: integral is divergent.
  - an error. To debug this try: debugmode(true);
  ```

  que de nuevo es divergente. En cambio

  $$\int_0^1 \frac{1}{x^{1/2}}dx$$

```
(%i) integrate(1/x^(1/2), x, 0,1);
(%o) 2
```

es convergente.

Una integral impropia que suele aparecer en Estadística es la conocida como función gamma:

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}dx$$

Si α es un número entero se verifica $\Gamma(\alpha) = (\alpha - 1)!$. En otro caso, es preciso consultar tablas o utilizar una aplicación de características similares a `wxMaxima`:

$$\int_0^\infty x^{5/2}e^{-6x}dx$$

```
(%i) integrate(x^(5/2) *%e^(-6*x),x,0,inf);;
(%o) 15 sqrt(%pi)/(8 6^(7/2))
```

5.5.4. Integración numérica: `quad_qags`

En ocasiones el cálculo directo de primitivas es muy difícil o incluso imposible, lo cual no hace viable el uso de la regla de Barrow. No obstante, el cálculo de integrales definidas sigue siendo posible mediante procedimientos numéricos que se basan el la aproximación del área que delimita la curva.

`wxMaxima` incorpora algunas herramientas que permiten aproximar numéricamente las integrales. Entre ellos se encuentra el comando `quad_qags` cuya sintaxis es la siguiente:

```
quad_qags(expr, variable, a, b)
```

donde

`expr` : función a integrar.

`variable` : variable con respecto a la que se integra.

`a` : extremo inferior del intervalo.

`b` : extremo superior del intervalo.

Esta función proporciona una lista con cuatro valores:

a) la aproximación de la integral

b) el error absoluto estimado de la aproximación

c) el número de evaluaciones del integrando

d) el código de error (que puede ir desde 0 hasta 6) y significa

- 0 : no ha habido problemas;
- 1 : se utilizaron demasiados intervalos;
- 2 : hubo una acumulación de errores de redondeo que puede dañar la aproximación;
- 3 : el integrando ha tenido un comportamiento extraño frente a la integración;

4 : fallo de convergencia;

5 : la integral es divergente o de convergencia lenta;

6 : los argumentos de entrada no son válidos;

Por ejemplo, dada la siguiente integral

$$\int_1^3 x^3 \sin(x) e^{x^{-3}} dx$$

el comando `integrate` proporciona la siguiente salida:

```
(%i) integrate(x^3 *sin(x) *%e^(x^(-3)),x, 1, 3);
(%o) ∫_1^3 e^(x^-3) x^3 sin(x) dx
```

lo que significa que `wxMaxima` no es capaz de calcular una primitiva. Sin embargo, siempre podremos calcular una aproximación numérica de la integral:

```
(%i) quad_qags(x^3 *sin(x) *%e^(x^(-3)), x, 0, 3);
(%o) [,1411190520026566,2,180849270669424*10^-15,63,0]
```

5.5.5. Aplicaciones geométricas

Una de las aplicaciones del cálculo integral es la determinación de áreas. Dadas dos curvas $f, g : [a, b] \to \mathbb{R}$ el área que encierran sus gráficas viene dada por:

$$\int_a^b |f(x) - g(x)| dx$$

Observar que la expresión anterior incluye un valor absoluto. En general `wxMaxima` no es capaz de resolver integrales que incluyan el valor absoluto, y es responsabilidad del usuario determinar la posición relativa entre las curvas f y g.

Para calcular el área entre dos curvas necesitaremos:

1. Conocer los puntos de intersección de las curvas, así como los cortes de éstas con el eje OX. Para estos cálculos podemos utilizar los comandos de resolución ecuaciones.

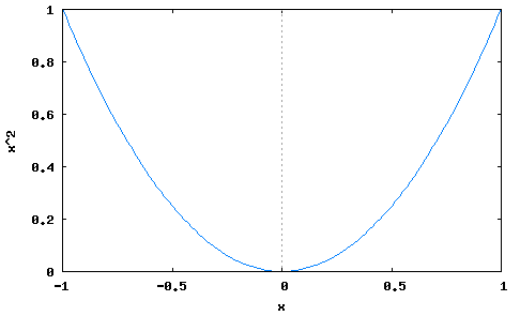
2. Es conveniente contar con un gráfico de ambas curvas.

Ejemplos

1. Calcular el área entre la curva $y = x^2$ y el eje OX en el intervalo $[-1, 1]$.

   Dibujamos la curva $y = x^2$:

   ```
   (%i) wxplot2d(x^2,[x,-1,1]);
   ```



y dado que la curva siempre esta por encima del eje OX, el área se calculará:

```
(%i) integrate(x^2,x,-1,1)
(%o) 2/3
```

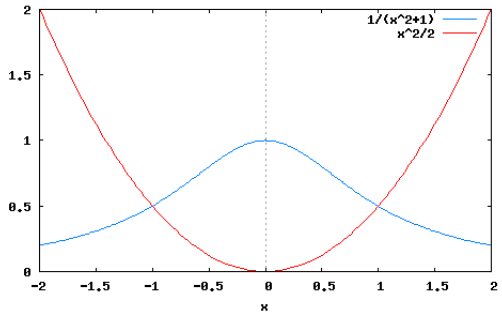
2. Calcular el área de la figura limitada por las curvas $y = \frac{1}{1+x^2}$ e $\frac{x^2}{2}$

   En primer lugar definimos ambas funciones:

```
(%i) f(x):= 1/(1+x^2)$
(%i) g(x):= x^2/2$
```

   Dibujamos ambas curvas:

```
(%i) wxplot2d([f(x),g(x)],[x,-2,2]);
```



   obsevando que la función $f(x)$ esta situada por encima de $g(x)$

   Para estar seguros de los límites de integración resolvemos la ecuación $f(x) = g(x)$

```
(%i) solve(f(x)=g(x),x);
(%o) [x = -sqrt(2) i, x = sqrt(2) i, x = -1, x = 1]
```

   sólo nos interesan las soluciones reales que indican que los límites de integración son $x = -1$ y $x = 1$. Finalmente calculamos el área:

```
(%i) integrate(f(x) - g(x), x, -1,1);
(%o) (3 %pi-2)/6
```

5.6. Ejercicios

1. Dibujar las gráficas de las siguientes funciones:
   - $x(t) = t - \sin(t), \quad y(t) = 1 - \cos(t), \quad t \in [0, 4\pi]$.
   - $x^{2/3} + y^{2/3} = 1$
2. La función de utilidad de un consumidor que adquiere dos bienes en cantidades x e y es $u(x, y) = \log xy$. Dibujar las curvas de indiferencia para valores $u = 3$, $u = 4$ y $u = 10$.
3. Determina si las siguientes funciones tienen simetría par o impar.
   - $y = \frac{\sin(x)}{x^2+1}$
   - $y = \frac{x^3-x}{x^4+x^2+1}$
   - $y = \log x^2 + x + 1$

- $y = \cos(3x)\sin(x^3)$;

4. Dada la función $y = x^3 + 2x - 3$:

   a) Dibuja su simétrica respecto del eje OX.

   b) Dibuja su simétrica respecto del eje OY.

   c) Dibuja la translación de la función 3 unidades en la dirección vertical, y 2 unidades en la dirección horizontal.

5. Su supone que la evolución del precio de una acción sigue una ley:

$$y(t) = t^2 - 8t + 20 + 5/(t+1)\sin(10t) - \cos(3t);$$

   donde t representa el tiempo en días. ¿Cuál es el momento óptimo para efectuar su compra?

6. Una fábrica de plásticos recibe del Ayuntamiento de la ciudad un pedido de 8000 tablas flotadoras para el programa de natación del verano. La fábrica posee 10 máquinas, cada una de las cuales produce 50 tablas por hora. El coste de preparar las máquinas para hacer el trabajo es de 800 euros por máquina. Una vez que las máquinas están preparadas, la operación es automática y puede ser supervisada por una sola persona, que gana 35 euros/hora.

   a) ¿Cuántas máquinas hay que usar para minimizar el coste de producción?

   b) Si se usa el número óptimo de máquinas, ¿cuánto ganará el supervisor durante el proceso si cada tabla flotadora tiene un precio de 1,5 euros?

7. Dada la curva $f(x) = 2e^x + e^{-x}$, calcular la tangente en el punto $x = 1$, y representar ambas gráficas en la misma figura.

8. Considerar la función $f(x) = \log(x+1)$, calcula los polinomios de Taylor de orden 2,3,4,5 y 6 alrededor de $x_0 = 0$ y representa sus correspondientes gráficas en una única figura.

9. Calcular las siguientes primitivas

a) $\int (x^3 + 2x - 4)dx$

b) $\int 4e^{3x}dx$

c) $\int (e^x + \frac{1}{3x} + x^2 + 2 - \sin x + \cos 3x)dx$

d) $\int \frac{1}{x-1}dx$

e) $\int \frac{\sin x}{\cos x}dx$

f) $\int x2^{x^2}dx$

g) $\int (3^x + x^3)dx$

h) $\int \sqrt{x-1}dx$

i) $\int \frac{3x^2 + 2x - 2}{x^3 + x^2 - 2x}dx$

j) $\int \cos(x)e^{sin(x)}dx$

k) $\int (x+2)^{-1/4}dx$

l) $\int (2x+3)(x^2+3x)^{\frac{2}{3}}dx$

m) $\int \frac{1}{\sqrt{x}}dx$

n) $\int \frac{x^3 + 4x^2 - x + 3}{x^2}dx$

o) $\int (\sqrt{a} - \sqrt{x})^2 dx$

p) $\int \left(1 + \frac{1}{x}\right)^3 dx$

10. Calcula las siguientes primitivas

$$\int \frac{x^2+1}{x-1}dx \qquad \int (2^{2/3}-x^{2/3})^3 dx$$
$$\int 2\sin(5x)-\cos(5x)dx \qquad \int \log(x)dx$$
$$\int e^x \sin(x)dx \qquad \int x^3 e^{3x}dx$$
$$\int \frac{x^2-5x+9}{x^3-5x^2+4x}dx \qquad \int \sin^2 x\cos^2 x dx$$
$$\int \frac{\sin(2x)}{1+\sin^2(x)}dx \qquad \int \frac{x^3}{\sqrt{x-1}}dx$$
$$\int \frac{x^5}{\sqrt{1-x^2}}dx \qquad \int \frac{1}{1+e^x}dx$$

11. Calcula las siguientes integrales definidas:

a) $\int_1^e \frac{1}{x}dx$ b) $\int_0^\pi \cos x dx$

c) $\int_0^5 (x^2+3x+1)dx$ d) $\int_{-2}^2 e^{2x}dx$

e) $\int_1^2 \frac{1}{x+1}dx$ f) $\int_0^1 xe^{-3x^2}dx$

g) $\int_0^\infty e^{-3x}dx$ h) $\int_1^\infty \frac{1}{x^a}dx, \quad a>1$

12. Calcula las siguientes integrales definidas:

$$\int_0^1 \frac{1}{1+e^x} \qquad \int_0^{1/2} \frac{1}{\sqrt{20+8x-x^2}}dx$$
$$\int_0^3 \frac{1}{\sqrt{9-x^2}}dx \qquad \int_0^1 \frac{x^2}{\sqrt{1-x^6}}dx$$
$$\int_0^\infty \frac{x}{3+x^4}dx \qquad \int_{-\infty}^\infty \frac{1}{e^x+e^{-x}}dx$$
$$\int_0^\infty e^{-\alpha x}\cos(\beta x)dx, \quad \alpha>0, \beta\in\mathbb{R} \qquad \int_0^\infty e^{-\alpha x}\sin(\beta x)dx, \quad \alpha>0, \beta\in\mathbb{R}$$

13. Si $X \sim N(\mu,\sigma)$ sigue una distribución normal de media μ y varianza σ, su funciones de densidad y distribución vienen dadas por:

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(x-\mu)^2}{2\sigma^2}} \qquad F(x)=\int_{-\infty}^x f(t)dt$$

Suponer que $\mu=10$, $\sigma=5$ y responder a las siguientes cuestiones:

a) Dibujar la función de densidad.

b) Dibujar la función de distribución.

c) Comprobar que

$$F(\infty)=\int_{-\infty}^\infty f(t)dt=1$$

d) Utilizar integración numérica para calcular:

$$P(X \leq 6) = F(6) \qquad P(X \geq 3) = 1 - F(3) \qquad P(X \leq 10) = F(10)$$

14. Hallar el área comprendida entre el eje de abcisas y la curva $y = x^3 - 6x + 8x$.

15. Hallar el área comprendida entre las parábolas $y = 6x - x^2$, $y = x^2 - 2x$.

Capítulo 6

Interpolación y Aproximación de Raíces

6.1. Introducción

En esta práctica utilizaremos `wxMaxima` para realizar interpolación polinomial y aproximación de raíces.

En la sección 2, presentamos dos funciones que permiten calcular el polinomio interpolador mediante la forma de Lagrange y Newton, respectivamente. Concluíremos esta sección describiendo el paquete `interpol` que es la herramienta que incluye `wxMaxima` y que además de la interpolación clásica, permite interpolación lineal a trozos y mediante splines.

En la sección 3, se han implementado en `wxMaxima` los algoritmos del método de la bisección y método de Newton.

Aunque no es parte del temario de este curso, se incorpora un apéndice con una introducción a la programación en `wxMaxima` que incluye los comandos usados en las secciones anteriores.

6.2. Interpolación

6.2.1. Polinomio interpolador de Lagrange

La siguiente función proporciona el polinomio interpolador. Para su cálculo usa la forma de Lagrange.

```
interLagrange(puntos):=block( [X, Y, L, P, n, num, den, i, j],
  n:length(puntos),
  X:makelist(puntos[i][1],i,1,n),
  Y:makelist(puntos[i][2],i,1,n),
  for i:1 step 1 thru n do(
    num:1,
    den:1,
    for j:1 step 1 thru n do(
      if i#j then (
      num: num * (x -X[j]),
      den: den * (X[i] - X[j]))
    ),
    L[i]:num/den
  ),
```

```
    P:0,
    for i:1 step 1 thru n do(
      P : P + Y[i]*L[i]
    ),
    ratsimp(P) )$
```

La sintaxis es la siguiente:

```
interLagrange(puntos)
```

donde

`puntos` : es la lista de coordenadas de los puntos a interpolar.

La función proporciona como resultado el polinomio interpolador como función de x.

Ejemplo. Calcular el polinomio interpolado que pasa por los puntos:

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

Sol. En primer lugar, definimos la variable puntos:

```
(%i) puntos:[[1,3],[2,4],[3,5],[4,2],[5,5]];
```

A continuación llamamos a la función `InterLagrange`:

```
(%i) interLagrange(puntos);
(%o)
```

$$\frac{7\,x^4-78\,x^3+293\,x^2-426\,x+240}{12}$$

Si queremos que el polinomio interpolador sea una función dependiente de x, para usarlo posteriormente, escribiríamos:

```
(%i) p(x):=interLagrange(puntos);
```

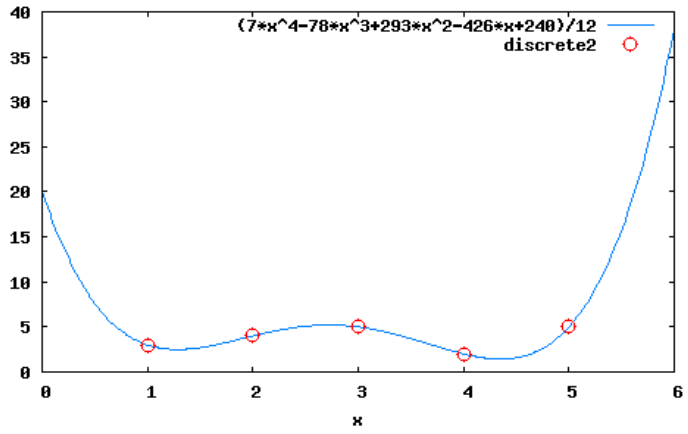
En ese caso para obtener el polinomio, evaluamos la función $p(x)$:

```
(%i) p(x)
(%o)
```

$$\frac{7\,x^4-78\,x^3+293\,x^2-426\,x+240}{12}$$

Podemos dibujar el polinomio interpolador junto con los puntos de interpolación del siguiente modo:

```
(%i) wxplot2d([p(x),[discrete,puntos]],[x,0,6],[style,lines,points]);
```

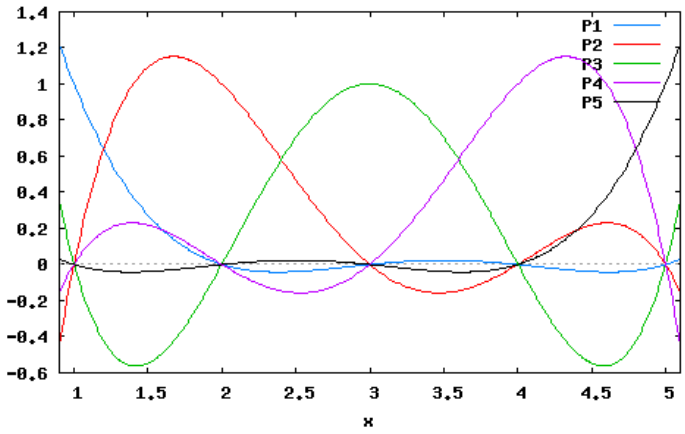


Por último, vamos a dibujar los 5 polinomios de Lagrange asociados a los puntos de interpolación. En primer lugar construímos los polinomios de Lagrange: para ello, la abcisa asociada al polinomio se fija con valor 1 y el resto de abcisas con valor 0

```
(%i) puntos1:[[1,1],[2,0],[3,0],[4,0],[5,0]]$
(%i) P1(x):=interLagrange(puntos1)$

(%i) puntos2:[[1,0],[2,1],[3,0],[4,0],[5,0]]$
(%i) P2(x):=interLagrange(puntos1)$

(%i) puntos3:[[1,0],[2,0],[3,1],[4,0],[5,0]]$
(%i) P3(x):=interLagrange(puntos3)$

(%i) puntos4:[[1,0],[2,0],[3,0],[4,1],[5,0]]$
(%i) P4(x):=interLagrange(puntos4)$

(%i) puntos5:[[1,0],[2,0],[3,0],[4,0],[5,1]]$
(%i) P5(x):=interLagrange(puntos5)$
```

A continuación utilizamos la instrucción `wxplot2d`:

```
wxplot2d([P1(x),P2(x),P3(x),P4(x),P5(x)],[x,0.9,5.1],
        [legend,"P1","P2","P3","P4","P5"]);
```



6.2.2. Polinomio interpolador de Newton

La siguiente función devuelve la tabla de diferencias divididas asociada a un conjunto de puntos `puntos`:

```
tdd(puntos):=block([X, Y, D, n, i, j],
  n:length(puntos),
  X:makelist(puntos[i][1],i,1,n),
  Y:makelist(puntos[i][2],i,1,n),
  D:zeromatrix(n,n),
  for i:1 step 1 thru n do (
    D[i,1]: Y[i]
  ),
  for i:2 step 1 thru n do (
    for j:1 step 1 thru (n+1-i) do (
     D[j,i]:(D[j+1,i-1] - D[j,i-1])/(X[j+i-1]-X[j])
    )
  ),
  D )$
```

La función `interNewton` proporciona el polinomio interpolador asociado a una serie de `puntos`. Para su cálculo usa la forma de Newton. Observar que en la segunda linea hay una llamada a la función `tdd`.

```
interNewton(puntos):=block([X, Y, D,P,aux],
  D:tdd(puntos),
  n:length(puntos),
  X:makelist(puntos[i][1],i,1,n),
  Y:makelist(puntos[i][2],i,1,n),
  P:D[1,1],
  for i:1 step 1 thru n-1 do(
    aux:product(x-X[k],k,1,i),
    P : P + D[1,i+1]*aux
  ),
  ratsimp(P) )$
```

Ejemplo. Calcular la tabla de diferencias divididas asociada a los puntos:

x	-2	0	2	4	6
y	1	-4	-5	2	3

Sol. Basta, con definir la variable `puntos` y a continuación llamar a la función `tdd`

```
(% i) tdd(puntos);
(%o)
```

$$\begin{pmatrix} 1 & -\frac{5}{2} & \frac{1}{2} & \frac{1}{12} & -\frac{3}{64} \\ -4 & -\frac{1}{2} & 1 & -\frac{7}{24} & 0 \\ -5 & \frac{7}{2} & -\frac{3}{4} & 0 & 0 \\ 2 & \frac{1}{2} & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Ejemplo. Dada la función $y = \text{sen}(x)$:

- Calcular el polinomio interplador en los nodos $0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi$ usando la forma de Newton.
- Comprobar que el resultado es el mismo si se usa la forma de Lagrange.
- Dibujar el polinomio interpolador junto con la función y los puntos de interpolación.
- Dibuja la función de error: $|p(x) - \text{sen}(x)|$.
- Estima el valor de sen(1,0) con el polinomio interpolador, y calcula el error cometido.

Sol. Definimos los puntos de interpolación:

```
(%i) puntos:[[0,sin(0)],[%pi/2,sin(%pi/2)],[%pi, sin(%pi)],
[3*%pi/2,sin(3*%pi/2)],[2*%pi,sin(2*%pi)]] $
```

Calculamos el polinomio interpolador usando la forma de Newton:

```
(%i) interNewton(puntos);
(%o)
```

$$\frac{8\,x^3 - 24\,\pi\,x^2 + 16\,\pi^2\,x}{3\,\pi^3}$$

Ahora, usamos la forma de Lagrange y podemos comprobar que el resultado es el mismo:

```
(%i) interLagrange(puntos);
(%o)
```

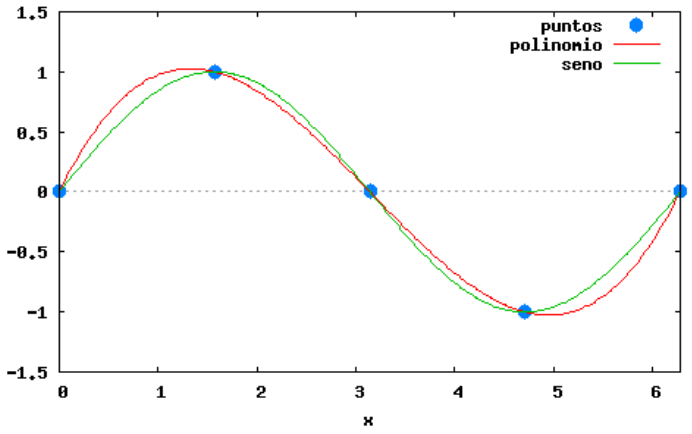
$$\frac{8\,x^3 - 24\,\pi\,x^2 + 16\,\pi^2\,x}{3\,\pi^3}$$

Para los siguientes apartados, definimos el polinomio interpolador como función de x como hicimos en la sección anterior:

```
(%i) p(x):=interNewton(puntos)$
```

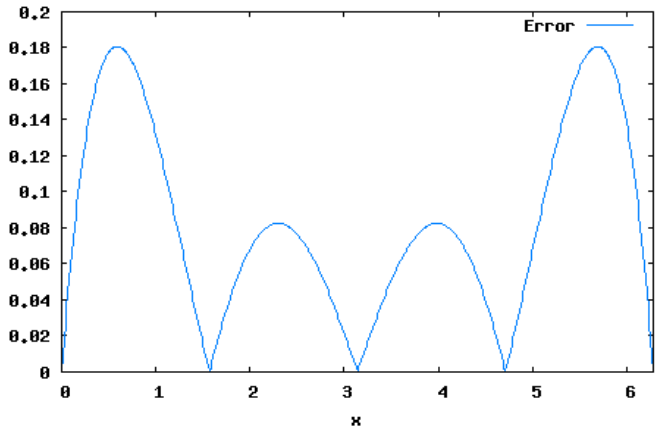
A continuación, usamos el comando `wxplot2d`

```
(%i) wxplot2d([[discrete,puntos],p(x),sin(x)],[x,0,2*%pi],
  [style,points,lines,lines],[legend,"puntos","polinomio","seno"]);
```



Para dibujar el error, usamos la función valor absoluto: `abs`

```
(%i)wxplot2d(abs(p(x) - sin(x)),[x,0,2*%pi], [legend,‘‘Error’’]);
```



Por último para estimar el valor de sen(1,0) simplemente evaluamos:

```
(%i)p(1);
(%o).9730873489967129
```

y para calcular el error:

```
(%i)sin(1) - p(1);
(%o) - .1316163641888164
```

6.2.3. Paquete `interpol`

`wxMaxima` incorpora el paquete `interpol` que contiene interpolación de Lagrange, interpolación lineal e interpolación por splines cúbicos. Para usar estas funciones, en primer lugar, se debe cargar el paquete `interpol`:

```
(%i) load(interpol);
```

La sintaxis de estas funciones es la siguiente:

`lagrange(puntos)`	calcula el polinomio interpolador asociado al conjunto `puntos`
`linearinterpol(puntos)`	calcula la interpolación lineal a trozos del conjunto `puntos`
`cspline(puntos)`	caclula la interpolación mediante splines cúbicos del conjunto `puntos`

Nota. Un spline cúbico es una función definida a trozos por polinomios de grado 3 que interpolan un conjunto de puntos. La diferencia con respecto a la interpolación a trozos es que un spline cúbico es diferenciable en todo punto (es una función suave).

Ejemplo. Dados los puntos:

x	-2	0	2	4	6	8	10
y	1	-4	-5	2	3	5	6

- Calcular el polinomio interpolador.
- Calcular la interpolación lineal a trozos.
- Calcular la interplación con un spline cúbico.

Sol. Como en casos anteriores, definimos la variable puntos:

```
(%i) puntos:[[-2,1],[0,-4],[2,-5],[4,2],[6,3],[8,5],[10,6]]$
```

A continuación llamamos a las funciones `lagrange, linearinterpol, cspline` del paquete `interpol`:

```
(%i) lagrange(puntos);
(%o)
```

$$\frac{(x-8)\ (x-6)\ (x-4)\ (x-2)\ x\ (x+2)}{7680}$$
$$-\frac{(x-10)\ (x-6)\ (x-4)\ (x-2)\ x\ (x+2)}{1536}$$
$$+\frac{(x-10)\ (x-8)\ (x-4)\ (x-2)\ x\ (x+2)}{1024}$$

$$-\frac{(x-10)\ (x-8)\ (x-6)\ (x-2)\ x\ (x+2)}{1152}$$
$$-\frac{5\ (x-10)\ (x-8)\ (x-6)\ (x-4)\ x\ (x+2)}{3072}$$
$$+\frac{(x-10)\ (x-8)\ (x-6)\ (x-4)\ (x-2)\ (x+2)}{1920}$$
$$+\frac{(x-10)\ (x-8)\ (x-6)\ (x-4)\ (x-2)\ x}{46080}$$

Por defecto, la función `lagrange` no simpifica la expresión. Si anteponemos la función `ratsimp` obtenemos la simplifiación de la expresión anterior:

```
(%i) ratsimp(lagrange(puntos));
(%o)
```

$$-\frac{23\,x^6-570\,x^5+4580\,x^4-10040\,x^3-26368\,x^2+72320\,x+61440}{15360}$$

```
(%i) linearinterpol(puntos);
(%o)
```

$$\left(-\frac{5\,x}{2}-4\right)\ \text{charfun2}\,(x,-\infty,0)+\left(\frac{x}{2}+1\right)\ \text{charfun2}\,(x,8,\infty)$$
$$+(x-3)\ \text{charfun2}\,(x,6,8)+\frac{x\,\text{charfun2}\,(x,4,6)}{2}$$
$$+\left(\frac{7\,x}{2}-12\right)\ \text{charfun2}\,(x,2,4)+\left(-\frac{x}{2}-4\right)\ \text{charfun2}\,(x,0,2)$$

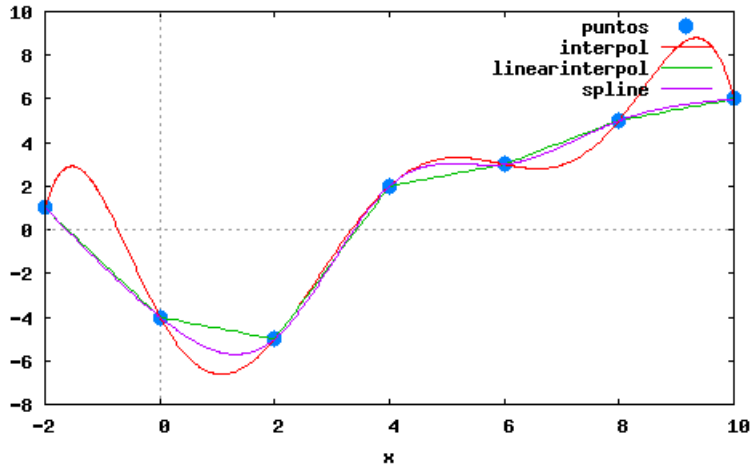
```
(%i) cspline(puntos); (%o)
```

$$\left(\frac{293\,x^3}{6240}+\frac{293\,x^2}{1040}-\frac{1657\,x}{780}-4\right)\ \text{charfun2}\,(x,-\infty,0)$$
$$+\left(\frac{383\,x^3}{6240}-\frac{383\,x^2}{208}+\frac{14561\,x}{780}-\frac{753}{13}\right)\ \text{charfun2}\,(x,8,\infty)$$
$$+\left(-\frac{227\,x^3}{1248}+\frac{4157\,x^2}{1040}-\frac{21871\,x}{780}+\frac{4331}{65}\right)\ \text{charfun2}\,(x,6,8)$$
$$+\left(\frac{2597\,x^3}{6240}-\frac{7039\,x^2}{1040}+\frac{28511\,x}{780}-\frac{4066}{65}\right)\ \text{charfun2}\,(x,4,6)$$
$$+\left(-\frac{3793\,x^3}{6240}+\frac{5741\,x^2}{1040}-\frac{9829\,x}{780}+\frac{194}{65}\right)\ \text{charfun2}\,(x,2,4)$$
$$+\left(\frac{331\,x^3}{1248}+\frac{293\,x^2}{1040}-\frac{1657\,x}{780}-4\right)\ \text{charfun2}\,(x,0,2)$$

La función `charfun2` permite a `wxMaxima` definir funciones a trozos.

Para comprobar las tres aproximaciones las podemos dibujarlas. En primer las definimos las tres aproximaciones como funciones

```
(%i) p(x):= lagrange(puntos)$
(%i) l(x):= linearinterpol(puntos)$
(%i) s(x):= cspline(puntos)$
(%i) wxplot2d([[discrete,puntos],p(x),l(x),s(x)],[x,-2,10],
 [style,points,lines,lines,lines],
 [legend,"puntos",ïnterpol","linearinterpol","spline"]);
```



Ejemplo. Dada la función $f(x) = e^{-x^2}$ en el intervalo $[-5, 5]$ comparar los distintos métodos de interpolación usando nodos equiespaciados a distancia 1.

Sol. Introducimos la función $f(x)$ en `wxMaxima`:

```
(%i) f(x):=%e^ (-x^ 2);
```

Generamos la variable puntos con la función `makelist`:

```
(%i) puntos:makelist([i,f(i),i,-5,5,1);
(%o)
```

$$[[-5, e^{-25}], [-4, e^{-16}], [-3, e^{-9}], [-2, e^{-4}], [-1, e^{-1}], [0, 1],$$

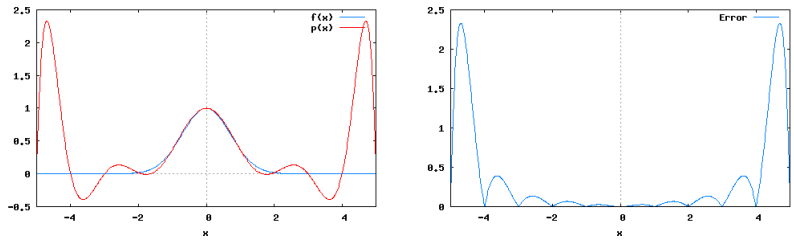
$$[1, e^{-1}], [2, e^{-4}], [3, e^{-9}], [4, e^{-16}], [5, e^{-25}]]$$

Calculamos cada una de las interpolaciones:

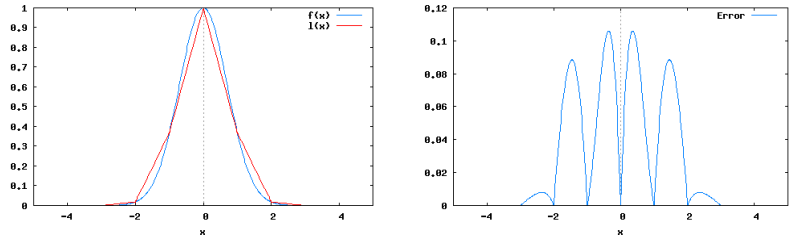
```
(%i) p(x):=lagrange(puntos)$
(%i) l(x):=linearinterpol(puntos)$
(%i) s(x):=cspline(puntos)$
```

A continuación, dibujamos cada aproximación junta a la función $f(x)$ y su respectivo error:

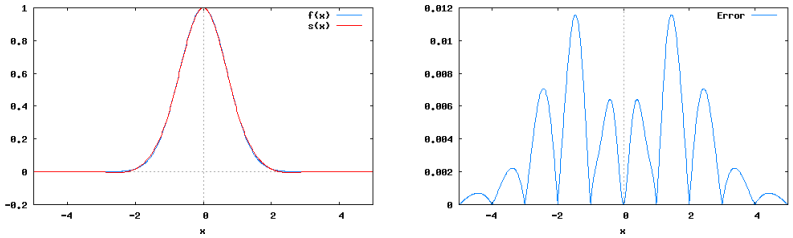
```
(%i) wxplot2d([f(x),p(x)],[x,-5,5]);
(%i) wxplot2d(abs(f(x)-p(x)),[x,-5,5]);
```




```
(%i) wxplot2d([f(x),l(x)],[x,-5,5]);
(%i) wxplot2d(abs(f(x)-l(x)),[x,-5,5]);
```

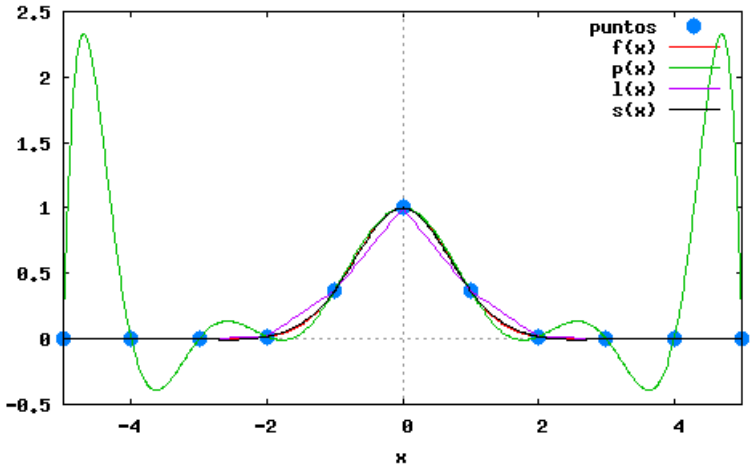



```
(%i) wxplot2d([f(x),s(x)],[x,-5,5]);
(%i) wxplot2d(abs(f(x)-s(x)),[x,-5,5]);
```



También podemos dibujar todas las aproximaciones junto con la función:

```
(%i) wxplot2d([[discrete,puntos],f(x),p(x),l(x),s(x)],[x,-5,5],
 [style,points,lines,lines,lines,lines],
 [legend,"puntos","f(x)","p(x)","l(x)","s(x)"]);
```



Nota El ejercicio muestra que la interpolación de alto orden (en el ejercicio estamos utilizando un polinomio de grado 10) con nodos equiespaciados es desaconsejable, *peligrosa* y da lugar a oscilaciones en los extremos del intervalo de interpolación. Este comportamiento se conoce como fenómeno de Runge y puede evitarse si los nodos se sitúan de forma apropiada (nodos de Chebyshev).

Los nodos de Chebyshev en un intervalo centrado en el 0 vienen dados por la expresión:

$$x_k = A\cos\left(\frac{(2k+1)\pi}{2n}\right), \quad k = 0, \ldots, n-1,$$

donde A es la semiamplitud del intervalo, y n el número de puntos de interpolación.

Calculemos los puntos de Chebyshev del ejemplo anterior y su correspondiente polinomio interpolador. En primer lugar vamos a pasar a modo numérico, para facilitar los cálculos a `wxMaxima`, y calcular los ndos de Chebyshev.

```
(%i) numer:true$
(%i) nodos: makelist(5*cos(((2*k+1)*%pi)/(2*11)),k,0,10)$
```

Ahora hallamos los puntos de interpolación:

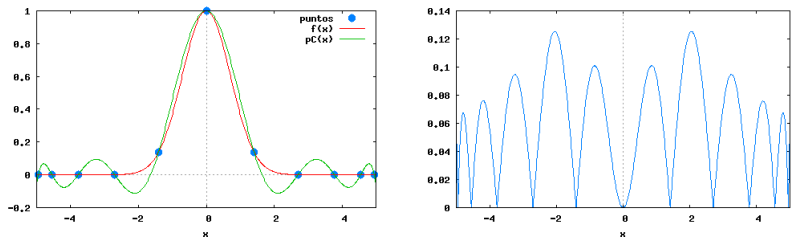
```
(%i) puntosCheb:makelist([nodos[i],f(nodos[i])],i,1,11)$
```

A continuación calculamos el polinomio interpolador,

```
(%i) pC(x):=interLagrange(puntosCheb)$
```

Finalmente, dibujamos la aproximación de Chebyshev junto con los puntos de interpolación y la función inicial, y el error de aproximación:

```
(%i) wxplot2d([[discrete,puntosCheb],f(x),pC(x)],[x,-5,5],
 [style,points,lines,lines],[legend,"puntos","f(x)","pC(x)"])$
(%i) wxplot2d(abs(f(x)-pC(x)),[x,-5,5])$
```



Observar, que el error ha disminuido con respecto a la interpolación realizada con nodos equiespaciados, pero aún aparecen oscilaciones en los extremos.

6.3. Aproximación de raíces

6.3.1. Método de la Bisección

La siguiente función implementa el método de la bisección para el cálculo de raíces.

```
solveBiseccion(f,a,b,tol, iterMax):=block([c:0,iter:0,e],
  numer:true,
  if (f(a)*f(b))<0 then (
  do(
    e:(b-a)/2,
    c: a + e,
    display([iter, a,b,c]),
    if e <tol then return(c),
    if f(b)*f(c)<0 then a:c else b:c,
    if iter>iterMax then return(print(" ERROR: Alcanzado el numero
        maximo de iteraciones sin obtener convergencia")),
    iter:iter+1
    )
  )
  else(
    print(" ERROR: la funcion toma el mismo signo en los extremos")
  ),
  c )$
```

La sintaxis de la función es la siguiente:

```
solveBiseccion(f,a,b,tol,iterMax)
```

donde

`f`: función de una variable.

`a`: extremo inferior del intervalo

`b`: extremo superior del intervalo

`tol`: tolerancia impuesta en la aproximación de la raiz

`iterMax`: número máximo de iteraciones

La función proporciona como resultado la aproximación de la raíz con la tolerancia prefijada, o un mensaje de error si la función toma valores con el mismo signo en los extremos del intervalo `[a,b]`.

Ejemplo. Calcular la solución de la ecuación:

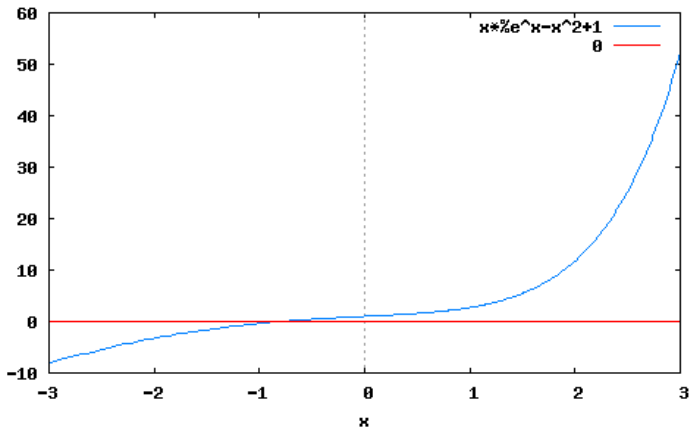
$$xe^x - x^2 + 1 = 0$$

Sol. Definimos la función en `wxMaxima`

```
(%i) f(x):=x*%e^ x -x^2+1;
```

Dibujamos la función junto con la función nula ($y = 0$) para localizar el intervalo donde utilizar el método de la bisección:

```
(%i) wxplot2d([f(x),0],[x,-3,3]);
```



Fijamos la tolerancia a 10^{-6}, el número máximo de iteraciones a 100, y lanzamos el método, para obtener tras 20 iteraciones:

```
(%i) solveBiseccion(f,-2,0,1.e-6,100);
(%o) -.8003168106079102
```

6.3.2. Método de Newton

La siguiente función implementa el método de Newton para el cálculo de soluciones de una ecuación algebraica.

```
solveNewton (f, x0,tol,iterMax):= block([iter, y, df, dfx,x1],
  df: diff (f ('x0), 'x0),
  numer:true,
  iter:0,
  display([iter,x0]),
  do (
    iter:iter+1,
    dfx0: ev(df),
    x1: x0 - f(x0)/dfx0,
    display([iter, x1]),
    if abs (x0-x1) <tol then return (x1) else x0:x1,
    if iter>iterMax then return(print('' ERROR: Alcanzado el numero
        maximo de iteraciones sin obtener convergencia"))
  ),
  x1 )$
```

La sintaxis de la función es la siguiente:

```
solveBiseccion(f,a,b,tol,iterMax)
```

donde

`f`: función de una variable.

`x0`: punto inicial.

`tol`: tolerancia impuesta en la aproximación de la raiz

`iterMax`: número máximo de iteraciones

La función proporciona como resultado la aproximación de la raíz con la tolerancia prefijada, o un mensaje de error si el número máximo de iteraciones es alcanzado sin obtener convergencia. El criterio de convergencia se basa en medir la distancia entre dos aproximaciones consecutivas. Cuando este valor es menor que la tolerancia prefijada la función devuelve la aproximación de la raíz.

Ejemplo. Calcular la raíz de la ecuación:

$$x^2 - 5 = 0$$

Sol. Definimos la función:

```
(%i) f(x):= x^ 2 - 5$
```

Fijamos x0=3, tol=10^{-10} y maxIter=100, y utilizamos la función `solveNewton`

```
(%i) solveNewton(f,3,1.e-10,100);
(%o) 2.23606797749979
```

Tan sólo tras 5 iteraciones obtenemos la solución. El método de Newton, en condiciones ideales ($x0$ "cerca" de la raíz, $f(x)$ función suave y raíz sencilla), converge con velocidad cuadrática. De un modo sencillo, podríamos decir que el número de cifras exactas se duplica tras cada iteración.

Sin embargo, esta convergencia se puede deteriorar si la raíz es múltiple:

Ejemplo. Utilizar el método de Newton para calcular la raíz positiva de la ecuación

$$x^4 - 4x^2 + 4 = 0$$

con un error de 10^{10} **Sol.** Definimos la función:

```
(%i) f(x):= x^4 - 4*x^2 + 4$
```

Tras dibujar la función, decidimos fijar x0=2, tol=10^{-10} y maxIter=100, y utilizamos la función `solveNewton`

```
(%i) solveNewton(f,2,1.e-10,100);
(%o) 1.414213572611696
```

Observamos que en este caso Newton requiere de 31 iteraciones para obtener la solución. La velocidad del método se ha deteriorado por qué la raíz tiene multiplicidad 2. Dicho de otro modo, la solución que hemos obtenido es raíz de la función y también de su derivada.

Por otro lado, el método de Newton es local, esto quiere decir que para obtener convergencia la aproximación inicial debe estar "suficientemente" próxima a la raíz. Observar el siguiente ejemplo.

Ejemplo. Usar el método de Newton para aproximar el cero de la función.

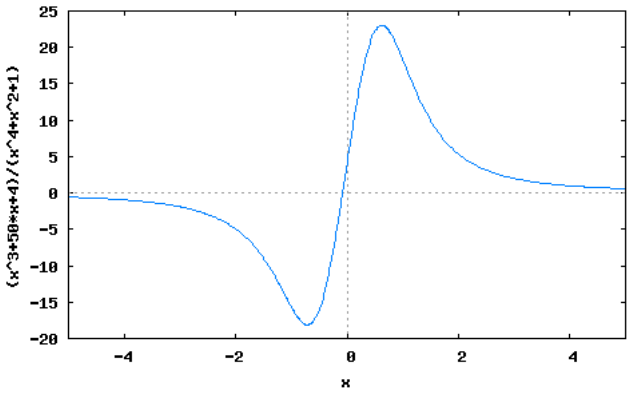
$$\frac{x^3 + 50x + 4}{x^4 + x^2 + 1}$$

Sol. Definimos la función:

```
(%i) f(x):= (x^3 + 50*x + 4) /(x^4 + x^2 +1)$
```

Dibujamos la función,

```
(%i)wxplot2d(f(x),[x,-5,5]);
```




```
(%i) solveNewton(f,2,1.e-10,100);
(%o) 1.414213572611696
```

Si iniciamos el método en x0=1, obtenemos:

```
(%i) solveNewton(f,1,1.e-6,20);
''ERROR: Alcanzado el numero maximo de iteraciones sin obtener
convergencia''
(%o) 305390.5775795507
```

El método diverge y se aleja de la solución. Si iniciamos en x0=0,5

```
(%i) solveNewton(f,1,0.5.e-6,20);
‘‘ERROR: Alcanzado el numero maximo de iteraciones sin obtener
convergencia’’
(%o) -179990.6969444858
```

De nuevo, el método ha fracasado. Si fijamos x0=0,25

```
(%i) solveNewton(f,1,0.25.e-6,20);
(%o) -.07998976393013375
```

Nota: En el ejercicio anterior, bastaba con haber usado el numerador. Sin embargo, hemos trabajado con la función completa para ilustar que el método de Newton es local.

6.3.3. Comandos `find_root, allroots`

`wxMaxima` incorpora funciones especificas para el cálculo de raíces. Algunas de ellas como `find_root` y `allroots` ya fueron descritas en la práctica anterior. A continuación recordamos su sintaxis:

```
find_root(ecuación, incognita, inf, sup)
```

donde

`ecuación` : ecuación a resolver.

`incognita` : incognita de la ecuación.

`inf` : extremo inferior del intervalo.

`sup` : extremo superior del intervalo.

```
allroots(polinomio)
```

donde

`polinomio` : polinomio del que se requiren las raíces

La función `find_root` combina el método de la bisección con una interpolación lineal. La función `allroots` utiliza un procedimiento de separación de raíces combinado con un método de Newton adaptado a ecuaciones algebraicas (algoritmo de Jenkins).

6.4. Apéndice: Programación en wxMaxima

En esta sección presentamos una pequenña introducción a la programación en `wxMaxima`. Dado que no es uno de los objetivos del curso, nos limitaremos a describir los comandos que han aparecido a lo largo de la práctica.

6.4.1. Operadores relacionales

`a = b`	operador de **comparación**: devuelve valor cierto si las dos expresiones `a` y `b` son iguales y falso en otro caso.
`a # b`	operador **distinto**: devuelve valor cierto si las dos expresiones `a` y `b` son distintas y falso en otro caso.

6.4.2. Sentencia alternativa simple

if `cond` **then(**
`expr1)`
else(
`expr2)`
	Si la condición `cond` es cierta se evalúa la expresión o grupo de expresiones `expr1` en caso contrario se evalúa la expresión o grupo de expresiones `expr2`

6.4.3. Sentencias repetitivas (bucles)

`for i:i0 step s thru n do(`
`expr...,`
`)`
	Se evalúa el grupo de sentencias `expr` para los valores de `i` que van desde el valor `i0` hasta el valor `n` a incrementos `s`. Si el incremento se omite, vale `s = 1`.

`while cond do(`
`expr...,`
`)`
	Mientras la condición `cond` sea cierta se evalúa el grupo de sentencias `expr`. El valor de `cond` debe modificarse en el grupo de sentencias `expr.`

`do(`
`expr... ,`
`return(var)`
`)`
	Evaluar repetidamente el grupo de sentencias `expr`. La salida de este bucle se efectúa por medio de la instrucción `return` que devuelve el valor de la variable `var`. La insturcción `return` suele ir combinada con una instrucción `if .. then .. else`.

6.4.4. Otras sentencias

`display(var)`	muestra por pantalla la variable `var`
`,`	operador coma, separa varias instrucciones.
`block([var], sentecias`	estructura que agrupa sentencias, separadas por comas. Las variables definidas al inicio son locales al bloque.

6.5. Ejercicios

1. Dada la tabla:

x_i	-1	0	1	2
y_i	-2	-2	0	4

   a) Hallar el polinomio interpolador usando la fórmula de Lagrange.

   b) Hallar la tabla de diferencias divididas.

   c) Hallar el polinomio interpolador usando la fórmula de Newton.

   d) Utilizando el polinomio interpolador estimar el valor de y para $x = 1{,}5$.

   e) Actualizar el polinomio interpolador si se añaden los datos $x_5 = 3$, e $y_5 = 5$.

2. Una empresa presenta el balance de algunos de sus últimos ejercicios, en los que se han producido los siguientes beneficios en miles de euros:

Año	2006	2008	2010	2012
Beneficios	80	120	60	40

   Utilizar interpolación lineal, cuadrática y cúbica, para estimar el beneficio en el año 2011.

3. Dada la función,

$$y = e^x.$$

a) Calcular el polinomio interpolador de grado 3, con nodos equiespaciados en el intervalo [0,2].

b) Calcular el polinomio interpolador de grado 5, con nodos equiespaciados en el intervalo [0,2].

c) Dibujar ambos polinomios con la función. Dibujar las gráficas de error y comentar los resultados.

4. Determinar todas las raíces de la siguiente ecuación con un error 10^{-8}.

$$x^3 - 4x^2 + x + 1 = 0$$

Para ello, usar los métodos de la bisección y Newton que se han estudiado en esta práctica.

5. Calcular las tres raíces positivas más pequeñas de la función

$$f(x) = x \tan x$$

6. Usar el método de Newton para hallar las soluciones de los siguientes problemas con una precisión de 10^{-8}

a) $e^x + 2^{-x} + 2\cos(x) - 6 = 0$, para $1 \leq x \leq 2$.

b) $(x-2)^2 - \log x = 0$, para $1 \leq x \leq 2$ y $e \leq x \leq 4$.

7. El crecimiento de grandes poblaciones se puede modelizar para periodos cortos de tiempo, suponiendo que la población crece de forma continua con el tiempo a una velocidad proporcional al número de individuos presentes. Si $N(t)$ representa el número de individuos presentes a un tiempo t y λ el porcentaje de nacimientos menos defunciones, la función:

$$N(t) = N_0 e^{\lambda t}$$

modela la población en el tiempo t, siendo N_0 la población inicial.

Este modelo es válido si se trata de una población aislada, es decir, no hay inmigración, en caso contrario, si existe inmigración a razón de v individuos por año, la fórmula que rige el comportamiento de la población es:

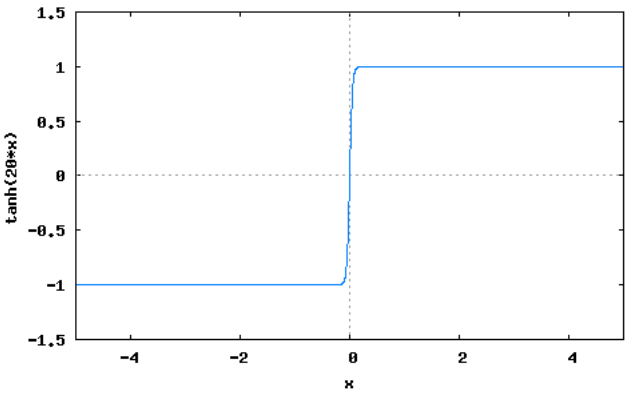
$$N(t) = N_0 e^{\lambda t} + \frac{v}{\lambda}(e^{\lambda t} - 1)$$

Supóngase que cierta población tiene inicialmente 100,000, que inmigran el primer año 4,211 y que al final de ese año hay 103,542. Determinar el porcentaje λ con una precisión de 10^{-4}. Utilizar ese porcentaje para calcular la población al final del segundo año.

8. Considerar la función (tangente hiperbólica),

$$f(x) = \tanh 20x$$

Utilizar los distintos tipos de interpolación que se han explicado en la práctica (interpolación polinómica, interpolación lineal a trozos e interpolación por splines) para aproximar la función anterior en el intervalo $[-3, 3]$ (usar 10, 15, 20 nodos), dibujar gráficas de error y comentar los resultados.


1.5
1
0.5
0
-0.5
-1
-1.5
tanh(20*x)
-4
-2
0
2
4
x

Capítulo 7

Funciones de varias variables

7.1. Introducción

Dedicaremos esta práctica al análisis de funciones de varias variables con `wxMaxima`. Comenzaremos estudiando como se definen en `wxMaxima`.

Las funciones de varias variables se definen de forma análoga a las de una varable. Tan sólo se debe distinguir si se trata de funciones a valores reales o vectoriales. Por ejemplo, para definir la función:

$$\begin{aligned} f : \mathbb{R}^3 &\to \mathbb{R} \\ (x, y, z) &\to e^{xy} + \cos(z) \end{aligned}$$

escribiríamos en `wxMaxima`:

```
(%i) f(x,y,z):=%e^(x*y) + cos(z);
```

Si en cambio, queremos definir una función a valores vectoriales, por ejemplo

$$\begin{aligned} f : \mathbb{R}^2 &\to \mathbb{R}^2 \\ (x, y) &\to (x^2 + y^2, \log xy) \end{aligned}$$

se haría:

```
(%i) h(x,y):= [x^2 + y^2, log(x*y)];
```

Notar, que los se han usado los operadores $[\cdot]$ para indicar que la función devuelve un vector.

En la sección 1 presentaremos las herramientas que permiten dibujar funciones de dos variables a valores reales. A continuación en la sección 2 calcularemos los límites reiterados y direccionales de funciones de dos variables. La sección 3, estará dedicada al cálculo diferencial. Por último en la sección 4 presentaremos algunas aplicaciones del cálculo diferencial como son el polinomio de Taylor y el cálculo del plano tangente.

7.2. Gráficas de funciones de dos variables

Al igual que con una variable, tenemos dos opciones `wxplot3d` y `wxdraw3d`, el último de ellos precisa la carga del paquete `draw`.

Nota: Recordar que el prefijo `wx` inserta los gráficos en el documento de `maxima`, si se prescinde de él, el gráfico se genera en una ventana auxiliar.

7.2.1. Gráficos con `wxplot3d` (`plot3d`)

La sintaxis del comando `wxplot3d` es la siguiente:

```
wxplot3d(f(x,y), [x, a, b], [y, c, d])
```

donde

`f(x,y)`: función a dibujar dependiente de x e y.

`x, y` : variables independientes de la función.

`a, b` : extremos que determinan la variación de la primera variable .

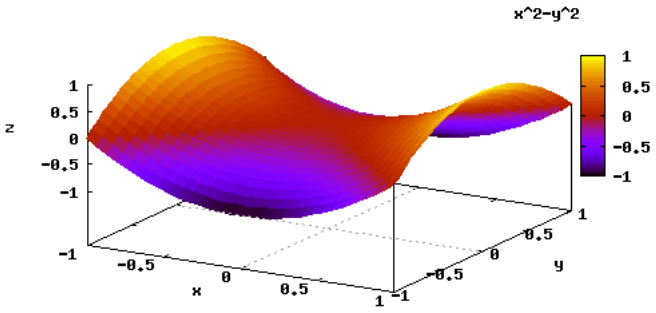
`c, d` : extremos que determinan la variación de la segunda variable.

Por ejemplo si quieremos dibujar la función:

$$f(x,y) = x^2 - y^2$$

en la región $[-1,1] \times [-1,1]$ escribiríamos:

```
(%i) f(x,y) := x^2 - y^2$
(% i) wxplot3d(f(x,y), [x, -1, 1], [y, -1, 1])
```



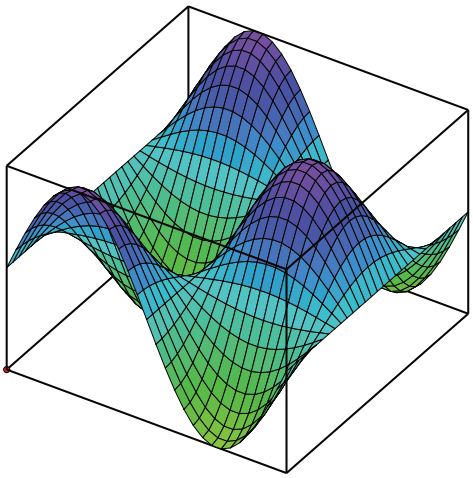
Nota: La función anterior se denomina habitualmente *silla de montar*. El punto $(0,0)$ es un punto característico que se denomina **punto silla**. Observar que en la dirección del eje x es un mínimo, sin embargo en la dirección del eje y, es un máximo.

En ocasiones es útil poder girar la gráfica, para ello podemos usar el comando `plot3d`, al que anadiremos la opción

```
[plot_format, openmath]
```

Por ejemplo, para dibujar la función: $f(x,y) = \sin(x)\cos(y)$ en el intervalo $[-\pi,\pi] \times [-\pi,\pi]$ y tener opción de mover la gráfica con el ratón, escribiríamos:

```
(%i) g(x,y) := sin(x)*cos(y)$
(%i) plot3d(g(x,y), [x, -%pi,%pi], [y, -%pi,%pi],[plot_format,
openmath]);
```



También es posible gestionar los gr'aficos 3d a través del menú `Plot->Plot3d`.

Por último, comentar que es posible dibujar varias funciones sobre el mismo gráfico, en ese caso la sintaxis es:

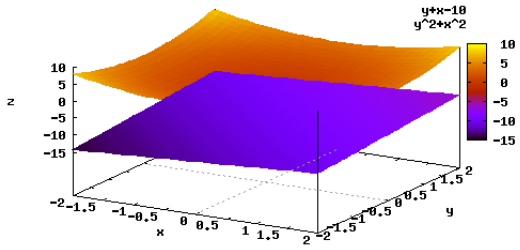
```
wxplot3d([f1(x,y),f2(x,y),f3(x,y),... ,[x, a, b], [y, c, d] ])
```

Por ejemplo, para dibujar las funciones ,

$$f_1(x,y) = x^2 + y^2 \qquad f_2(x,y) = x + y - 10$$

en el intervalo $[-2,2] \times [-2,2]$:

```
(%i) f1(x,y):=x^2 + y^2$
(%i) f2(x,y):=x+y-10$
(%i) wxplot3d([f1(x,y),f2(x,y),[x, -2, 2],[y,-2,2] ])
```



Sin embargo, como se observa, el resultado no es muy bueno.

7.2.2. Gráficos con `wxdraw3d` (`draw`)

En primer lugar es necesario cargar el paquete `draw`.

```
(%i) load(draw);
```

La sintaxis del comando `wxdraw2d` es la siguiente:

```
wxdraw3d(opciones, objeto)
```

donde

Opciones : comandos que modifican el gráfico. Todas ellas tienen valores por defecto. Algunas de las más útiles son las siguientes.

- **enhanced3d**: si su valor es `true` se dibuja la superficie, en otro caso se muestra una rejilla de la superficie.
- **surface_hide**: si vale `true` , las partes ocultas de la superficie no se muestran en las escenas 3d. Por defecto su valor es `false`
- **contour**: Sirve para dibujar lines de nivel y su valor puede ser:
  - **none**: no se dibujan las lineas de nivel.
  - **base**: las líneas de nivel se dibujan en el plano xy.
  - **surface**: las lineas de nivel se dibujan sobre la propia superficie.
  - **both**: se dibujan sobre la superficie y sobre el plano xy.
- **contour_levels**: sirve para indicar las lineas de nivel que se desea dibujar. Puede ser:
  - **Un número entero**: `n`. Entonces se dibujarán `n` líneas de nivel.
  - **Una lista de numeros enteros**: `[n1, n2, ...]`. Se dibujan las isolíneas correspondientes a los valores indicados.
  - **Una lista de tres números**: `[inf, p, sup]`. Se dibujan las isolineas que van desde el valor `inf` hasta el valor `sup` a distancia `p`.
- **user_preamble**: En este caso es apropiado fijar su valor a ‘‘`set size ratio 1`’’, para que la escala sea la misma en ambos ejes.
- **color**: color de la gráfica.
- **line_width**: grosor con el que se dibujan las líneas.
- **title**: título de la ventana.

Objeto : el objeto gráfico puede ser:

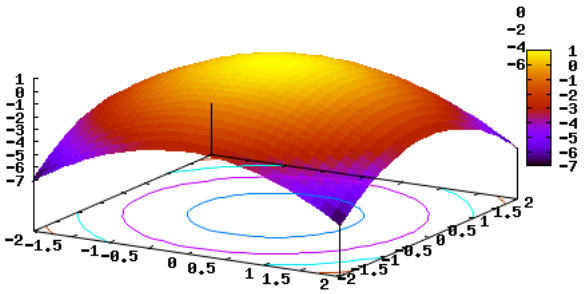
- **explicit(f(x,y), x, a, b, y, c, d)** : comando para dibujar una función explícita $z = f(x, y)$ en el intervalo $[a, b] \times [c, d]$.
- **parametric_surface(x(u,v), y(u,v), z(u,v), u, umin, umax, v, vmin, vmax)**: expresión para dibujar una superficie $(x(u, v), y(u, v), z(u, v))$ donde los parámetros (u, v) varian en $[umin, umax] \times [vmin, vmax]$.
- **implicit(F(x,y,z), x, xmin, ymax, y, ymin, ymax,z, zmin, zmax)**: comando para dibujar una función con ecuación ímplicita $F(x, y, z) = 0$, donde $x \in [xmin, xmax]$, $y \in [ymin, ymax]$, y $z \in [zmin, zmax]$.

Por ejemplo para dibujar la superficie explícita:

$$f(x, y) = 1 - x^2 - y^2$$

en el intervalo $[-2, 2], \times [-2, 2]$, escribiriamos:

```
(%i) f(x,y) := 1 - x^2 - y^2;
(%i) wxdraw3d( enhanced3d = true,
surface_hide = true,
contour = base,
explicit(f(x,y),x,-2,2,y,-2,2));
```

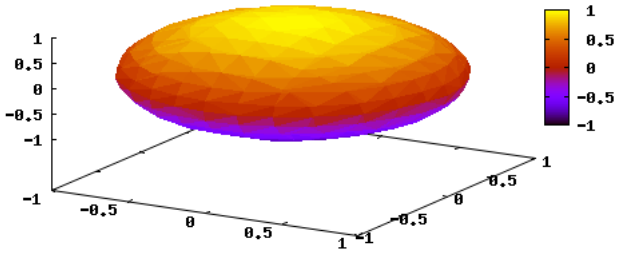


Para dibujar la esfera, que viene dada por la ecuación

$$x^2 + y^2 + z^2 = 1$$

escribiriamos:

```
(%i) wxdraw3d(enhanced3d= true,
implicit(x^2+y^2+z^2 -1, x,-1,1,y,-1,1,z,-1,1));
```



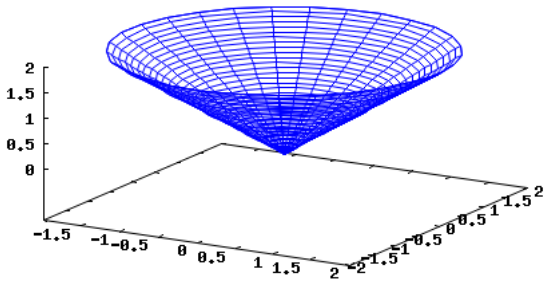
Nota: La esfera parece visualmente un elipsoide, porque no se ha respectado la escala en los ejes.

Por último si se desea representar un cono dado por las ecuaciones paramétricas:

$$x = r\cos(\alpha) \qquad y = r\sin(\alpha) \quad z = r$$

con $r \in= [0, 2]$ y $\alpha \in [0, 2\pi]$, escribiríamos:

```
(%i) wxdraw3d(enhanced3d=false, parametric_surface(r*cos(alpha),
r*sin(alpha), r, r,0,2, alpha, 0, 2*%pi));
```



7.3. Cálculo de límites

En esta sección veremos como calcular límites reiterados y direccionales. Como en ambos casos, al final, se trata de realizar un límite en una variable, basta con utilizar la instrucción `limit`, cuya sintaxis es:

```
limit(f(t),t,t0)
```

donde

`f(t)`: función a la que se cálcula el límite.

`t` : variable de la que depende el límite.

`t0` : valor hacía el que tiende el límite.

7.3.1. Límites reiterados

Veamos como se pueden calcular los límites reiterados en `wxMaxima`. Supongamos que se requiere calcular los límites reiterados de la siguiente función en el punto $(0,0)$:

$$f(x,y) = \frac{x^2 - y^3}{x^2 + y^2}$$

En primer lugar definimos la función:

```
(%i) f(x,y):= (x^2 - y^3)/(x^2+y^2)$
```

y a continuación calculamos los límites reiterados. Para calcular

$$\lim_{x\to 0}\lim_{y\to 0} f(x,y)$$

escribiríamos:

```
limit(limit(f(x,y),y,0),x,0);
(%o) 1
```

y para calcular

$$\lim_{x\to 0}\lim_{y\to 0} f(x,y)$$

usaríamos:

```
(%i) limit(limit(f(x,y),x,0),y,0);
(%o) 0
```

De los resultados anteriores, podemos concluir que la función f(x,y) no podría definirse de forma continua en el punto $(0,0)$.

7.3.2. Límites direccionales

Sea $f(x, y)$ la función definida por:

$$f(x, y) = \frac{2xy^2 - x^3}{x^2 + y^4}$$

para calcular los limites direccionales según la familia de rectas $y = mx$ en el punto $(0, 0)$, se procede del siguiente modo:

```
(%i) f(x,y):= (2x*y^2 - x^3)/(x^2+y^4)$
(%i) limit(f(x,m*x),x,0) (%o) 0
```

Del resultado anterior no es posible decidir si la función $f(x, y)$ puede ser o no continua. Sin embargo, si calculamos el límite según la curva $x = y^2$ obtenemos,

```
(%i) limit(f(y^2,y),y,0);
(%o) 1
```

y por tanto concluimos que la función no puede ser continua (si lo fuera el límite debería ser 0).

7.4. Cálculo diferencial

En esta sección describiremos las diferentes herramientas de las que dispone `wxMaxima`.

7.4.1. Derivadas parciales

Para el cálculo de las derivadas parciales disponemos de la instrucción `diff` cuya sintaxis es la siguiente:

```
diff(f(x1,x2,...,xk), x1, n1, x2, n2, ..., xk, nk)
```

donde

`f(x1,...,xk)`: es una función de k variables.

`xi` : es la variable i-ésima .

`ni` : es el número de derivadas con respecto a la variable i-ésima.

Por ejemplo dada la función:

$$f(x, y, z) = x\cos(xyz) - 3z^2\,\mathrm{sen}(xz);$$

vamos a calcular:

$$\frac{\partial f}{\partial x}, \qquad \frac{\partial^2 f}{\partial x \partial y}, \qquad \frac{\partial^5 f}{\partial x^2 \partial y \partial z}$$

Para ello, en primer lugar definimos la función en `wxMaxima`

```
(%i) f(x,y,z):=x *cos(x*y*z) - 3*z^2* sin (x*z)$
```

y calculamos respectivamente:

```
(%i) diff(f(x,y,z),x,1);
(%o)
```

$$-x\,y\,z\sin\left(x\,y\,z\right) + \cos\left(x\,y\,z\right) - 3\,z^3\cos\left(x\,z\right)$$

```
(%i)diff(f(x,y,z),x,1,y,1);
(%o)
```

$$-2\,x\,z\sin\left(x\,y\,z\right)-x^2\,y\,z^2\cos\left(x\,y\,z\right)$$

```
(%i)diff(f(x,y,z),x,2,y,1,z,2);
(%o)
```

$$-x^4\,y^4\,z^3\sin\left(x\,y\,z\right)+24\,x^2\,y^2\,z\sin\left(x\,y\,z\right)+10\,x^3\,y^3\,z^2\cos\left(x\,y\,z\right)-12\,x\,y\cos\left(x\,y\,z\right)$$

7.4.2. Gradiente/Jacobiano/Diferencial

Los tres conceptos que trataremos en esta sección están relacionados como las derivadas primeras de una función, y se calculan del mismo modo. El nombre en realidad depende de como se interpreten, asi:

- **Gradiente**: En el caso de funciones a valores reales es un vector que índica la dirección de mayor variación. Si $f:\mathbb{R}^n\to\mathbb{R}$:

$$\nabla f(a_1,\ldots,a_n)=\left(\frac{\partial f}{\partial x_1},\ldots,\frac{\partial f}{\partial x_n}\right)_{(a_1,\ldots,a_n)}$$

- **Jacobiano**: Es la matriz asociada a la aplicación (lineal) diferencial. Si $f:\mathbb{R}^n\to\mathbb{R}^m$:

$$J_f(a_1,\ldots,a_n)=\left(\begin{array}{ccc}\frac{\partial f_1}{\partial x_1}&\cdots&\frac{\partial f_1}{\partial x_n}\\ \cdots&\cdots&\cdots\\ \frac{\partial f_m}{\partial x_1}&\cdots&\frac{\partial f_1}{\partial x_m}\end{array}\right)_{(a_1,\ldots,a_n)}$$

- **Diferencial**: es la aplicación diferencial. Si $f:\mathbb{R}^n\to\mathbb{R}^m$:

$$\begin{aligned}d_{(a_1,\ldots,a_n)}f(x_1,\ldots,x_n)&=J_f(a_1,\ldots,a_n)\cdot(x_1,\ldots,x_n)^t\\&=\left(\begin{array}{ccc}\frac{\partial f_1}{\partial x_1}&\cdots&\frac{\partial f_1}{\partial x_n}\\ \cdots&\cdots&\cdots\\ \frac{\partial f_m}{\partial x_1}&\cdots&\frac{\partial f_1}{\partial x_m}\end{array}\right)_{(a_1,\ldots,a_n)}\left(\begin{array}{c}x_1\\ \cdots\\ x_n\end{array}\right)\end{aligned}$$

El comando que usaremos en `wxMaxima` para el cálculo de los operadores anteriores es `jacobian` cuya sintaxis es la siguiente:

```
jacobian([f1(x1,...,xn),...fm(x1,...,xn)], [x1,...,xn])
```

donde

1. `[f1(x1,...,xn),...fm(x1,...,xn)],`: es una función escrita en componentes. Si es una función a valores reales: $f:\mathbb{R}^n\to\mathbb{R}$ sólo tendra una componente, que **deberá ir entre corchetes** `[·]` .
2. `[x1,...,xn]`: vector de variable de las que depende la función.

En general es necesario evaluar el jacobiano/gradiente/diferencial en un determinado punto para ello, y como ya vimos en la práctica 3, es conveniente utilizar el comando `define`:

```
define(Jf(x1,...,xn), jacobian(...))
```

Veamos como funcionan estos comandos con varios ejemplos:

- Dada la función $f(x,y,z) = x^2 - 3y + z - \log xyz$, calcular el ∇f en un punto génerico, y $\nabla f(1,1,1)$.

  Definimos en primer lugar el vector gradiente utilizando los comandos `jacobian` y `define`:

```
(%i) f(x,y,z):=x^2 -3*y + z - log (x*y*z)$
(%i) define(gradf(x,y,z), jacobian([f(x,y,z)],[x,y,z]);
(%o)
```

$$\text{gradf}\,(x,y,z) := \begin{pmatrix} 2\,x - \frac{1}{x}, & -\frac{1}{y} - 3, & 1 - \frac{1}{z} \end{pmatrix}$$

  Para obtener el gradiente en un punto genérico evaluamos en el punto (x,y,z) la función `gradf`:

```
(%i) gradf(x,y,z);
(%o)
```

$$\begin{pmatrix} 2\,x - \frac{1}{x}, & -\frac{1}{y} - 3, & 1 - \frac{1}{z} \end{pmatrix}$$

  Para obtener el gradiente en el punto $(1,1,1)$, evaluamos de nuevo `gradf`:

```
(%i) gradf(1,1,1);
(%o)
```

$$\begin{pmatrix} 1 & -4 & 0 \end{pmatrix}$$

- Sea $f : \mathbb{R}^2 \to \mathbb{R}^3$ la función:

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

  1. Calcular la matriz jacobiana de f en el punto (x_0, y_0): $J_f(x_0, y_0)$
  2. Hallar la diferencial de f en $(\pi, \pi/2)$: $df_{(\pi,\pi/2)}$.
  3. Calcular $df_{(\pi,\pi/2)}(3,2)$

  Definimos la función y calculamos su jacobiano:

```
(%i) f1(x,y):= x * cos(y)$
(%i) f2(x,y):= x * sin(y)$
(%i) f3(x,y):= x * cos(y) * sin(y)$
(%i) define(Jf(x,y), jacobian([f1(x,y),f2(x,y),f3(x,y)],[x,y]));
(%o)
```

$$\text{Jf}\,(x,y) := \begin{pmatrix} \cos(y) & -x\sin(y) \\ \sin(y) & x\cos(y) \\ \cos(y)\,\sin(y) & x\cos(y)^2 - x\sin(y)^2 \end{pmatrix}$$

  Para calcular el Jacobiano en el punto (x_0, y_0), evaluamos:

```
(%i) Jf(x0,y0);
(%o)
```

$$\begin{pmatrix} \cos(y0) & -x0\sin(y0) \\ \sin(y0) & x0\cos(y0) \\ \cos(y0)\,\sin(y0) & x0\cos(y0)^2 - x0\sin(y0)^2 \end{pmatrix}$$

  Para obtener la diferencial $df_{(\pi,\pi/2)}$, multiplicamos el Jacobiano en el punto $(\pi, \pi/2)$ por el vector (x,y) (Usamos la `define`)

```
(%i) define(df(x,y), Jf(%pi,%pi/2) . [x,y] ;
(%o)
```

$$\mathrm{df}\,(x,y) := \begin{pmatrix} -\pi\, y \\ x \\ -\pi\, y \end{pmatrix}$$

Finalmente, para calcular $df_{(\pi,\pi/2)}(3,2)$, evaluamos la función anterior:

```
(%i) df(3,2);
(%o)
```

$$\begin{pmatrix} -2\,\pi \\ 3 \\ -2\,\pi \end{pmatrix}$$

7.4.3. Derivada direccional

Si la función es diferenciable la derivada direccional según un vector $(v_1, \ldots, v_n)$ puede calcularse mediante la fórmula:

$$D_{(v_1,\ldots,v_m)} f(x_1, \ldots, x_n) = \nabla f(x_1, \ldots x_n) \cdot (v_1, \ldots, v_n)$$

De modo, que de nuevo la instrucción `jacobian` nos permite calcular la derivada direccional de una función diferenciable:

Por ejemplo, si $f(x,y) = x^2 - 3y + log(xy)$, para calcular la derivada direccional de f según el vector $(1,1,)$ en el punto $(1,2)$ escribiríamos:

```
(%i) f(x,y) :=x^ 2 -3*y + log(x*y)$
(%i) define(gradf(x,y), jacobian([f(x,y)],[x,y])$
(%i) gradf(1,2) . [1,1];
(%o) 1/2
```

7.4.4. Hessiano

Si $f : \mathbb{R}^n \to \mathbb{R}$ el hessiano de f en un punto $(x_1, \ldots, x_n)$ viene dado por:

$$\begin{pmatrix} \frac{\partial f^2}{\partial x_1^2} & \cdots & \frac{\partial f^2}{\partial x_n \partial x_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial f^2}{\partial x_1 \partial x_n} & \cdots & \frac{\partial f^2}{\partial x_n^2} \end{pmatrix}$$

El hessiano es de vital importancia para la determinación de máximos y mínimos locales, como veremos en la siguiente práctica. Por ahora sólo nos interesa su cálculo, para ello, `wxMaxima` dispone de la instrucción `hessian`, cuya sintaxis es la siguiente:

```
hessian(f(x1,..., xn), [x1,...,xn])
```

donde

`f(x1,...,xn)` : es la función. Notar que en este caso la función no va delimitada por corchetes, `[ · ]`.

`[x1,...,xn]` : es el vector de variables.

Por ejemplo, si queremos calcular el hessiano de la función: $f(x,y,z) = \sin(x)\cos(y)e^z$ en el punto $(\pi/2, \pi, 0)$ procederíamos del siguiente modo:

```
(%i) f(x,y,z) = sin(x) *cos(y) *%e^z$
(%i) define(Hf(x,y,z), hessian(f(x,y,z),[x,y,z]);
(%o)
```

$$\mathrm{Hf}\,(x,y,z) := \begin{pmatrix} -\sin(x)\,\cos(y)\,e^z & -\cos(x)\,\sin(y)\,e^z & \cos(x)\,\cos(y)\,e^z \\ -\cos(x)\,\sin(y)\,e^z & -\sin(x)\,\cos(y)\,e^z & -\sin(x)\,\sin(y)\,e^z \\ \cos(x)\,\cos(y)\,e^z & -\sin(x)\,\sin(y)\,e^z & \sin(x)\,\cos(y)\,e^z \end{pmatrix}$$

y a continuación evaluamos en el punto:

```
(%i) Hf(%pi/2,%pi,0);
(%o)
```

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

7.5. Aplicaciones

En esta sección presentaremos dos aplicaciones del cálculo diferencial en varias variables como son el polinomio de Taylor y el plano tangente a una superficie.

7.5.1. Polinomio de Taylor

El polinomio de Taylor permite obtener aproximaciónes polinomicas de una función $f(x_1,\ldots,x_k)$ en un entorno del punto estudiado. La función que permite su cálculo en `wxMaxima` es `taylor` cuya sintaxis ya habíamos estudiado en el caso de funciones de una variable, y que ahora recordamos:

```
taylor(f(x1,...,xn), [x1,...,xk], [a1, ..., ak], [n1,..., nk])
```

donde: donde

`f(x1,...,xk)` : es la función.

`[x1,...,xk]` : es el vector de variables.

`[a1,...,ak]` : son las coordenadas del punto donde se cálcula el polinomio.

`[n1,...,nk]` : son los órdenes de aproximación en cada una de las variables.

Por ejemplo, para calcular el polinomio de Taylor de la función

$$f(x,y) = \sin(x)\cos(y)$$

en el punto $(0,0)$ de orden 3, escribíriamos:

```
(%i) f(x,y) := sin(x) * cos(y)$
(%i) taylor(f(x,y),[x,y],[0,0],[3,3]);
(%o) x - (x^3+3y^2x)/6
```

7.5.2. Plano Tangente

Dada una superfice en forma explícita $z = f(x,y)$ en esta sección veremos como calcular el plano tangente a la superficie en un punto (x_0,y_0). Este plano viene dado por la ecuación:

$$\left(\frac{\partial f}{\partial x}\right)_{(x_0,y_0)}(x-x_0) + \left(\frac{\partial f}{\partial y}\right)_{(x_0,y_0)}(y-y_0) - (z-z_0) = 0$$

y coincide con el desarrollo de Taylor de la función $f(x, y)$ en el punto (x_0, y_0) de orden 1. De modo, que su cálculo es directo si utilizamos la instrucción `taylor`.

Para ilustrarlo, vamos a calcular el plano tangente a la superficie:

$$z = f(x, y) = 1 - x^2 - y^2$$

en el punto $(1, 1)$:

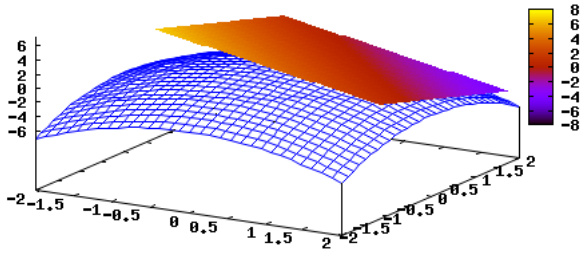
```
(%i) f(x,y):= 1- x^2 - y^2 $
(%i) taylor(f(x,y),[x,y],[1,1],[1,1]);
(%o) −1 + (−2(x − 1) − 2(y − 1));
```

De donde deducimos que el plano tangente tendrá ecuación:

$$z = -1 + (-2(x-1) - 2(y-1)).$$

Para terminar, vamos a dibujar función y plano sobre el mismo gráfico usando `wxdraw3d`:

```
(%i) plano(x,y):=taylor(f(x,y),[x,y],[1,1],[1,1])$
(%i) wxdraw3d(
surface_hide= true,
explicit(f(x,y),x,-2,2,y,-2,2),
enhanced3d=true,
explicit(plano(x,y),x,-1,2,y,-1,2));
(%o)
```



7.6. Ejercicios

1. Dibujar las siguientes superficies:
   - $z = x^3y^2$ en $[-1, 1] \times [-1, 1]$.
   - $z = xy$ en $[0, 2] \times [0, 2]$.
   - $x^2 + y^2 - z^2 = 0$ en $[0, 4] \times [0, 4] \times [0, 2]$.
   - $x = \cos(\alpha), \quad , y = \sin(\alpha), \quad z = r$ donde $\alpha \in [0, 2\pi]$ y $r \in [0, 3]$.
2. Dada la función $f(x, y) = x^2 - 3y + \log xy$:
   a) Calcular el gradiente de f en el punto $(1, 2)$: $\nabla f(1, 2)$.
   b) Calcular la derivada de f según el vector $(1, 1)$ en el punto $(1, 2)$:

3. Hallar el vector gradiente de las siguientes funciones reales en un punto (x, y, z) de su domino:

$$f(x, y, z) = x^2 - yz + z^2$$
$$f(x, y, z) = xy^2 e^{3x+y-z} + 2z$$
$$f(x, y, z) = z^2 \log(\sqrt{x} + y^2 + 1)$$

4. Sea $f : \mathbb{R}^2 \to \mathbb{R}^3$ la función:

$$f(x, y, z) = (x \cos y, x \operatorname{sen} y, x \cos y \operatorname{sen} y, \sin(z))$$

   a) Calcular la matriz jacobiana de f en el punto (x_0, y_0, z_0): $J_f(x_0, y_0, z_0)$
   b) Hallar la diferencial de f en $(\pi, \pi/2, 0)$: $df_{(\pi,\pi/2,0)}$.
   c) Calcular $df_{(\pi,\pi/2,0)}(3, 2, 1)$

5. Sean:

$$f(x, y) = (x/y, xy)$$
$$g(x, y) = x^2 - 2xy$$

   a) Calcular $J_f(2, 1)$.
   b) Calcular $J_g(2, 2)$
   c) Calcular $J_{(g \circ f)}(2, 1)$

6. Sean

$$f(x, y, z) = (\operatorname{sen}(xy + z), 1 + x^2)$$
$$g(u, v) = (u + e^v, v)$$

   Calcular $\left(\frac{\partial(g \circ f)}{\partial x}\right)_{(1,-1,1)}$, $\left(\frac{\partial(g \circ f)}{\partial y}\right)_{(1,-1,1)}$, $\left(\frac{\partial(g \circ f)}{\partial z}\right)_{(1,-1,1)}$.

7. Sea $f(x, y, z) = (x - y)(y - z)(z - x)$. Comprobar que:

$$\frac{\partial f}{\partial x}(x, y, z) + \frac{\partial f}{\partial y}(x, y, z) + \frac{\partial f}{\partial z}(x, y, z) = 0$$

8. Hallar la matriz hessiana de las siguientes matrices en el punto (x, y)

$$f(x, y) = \cos(xy) + e^y$$
$$f(x, y) = 3xy + 2y^2$$
$$f(x, y) = 3xy^2 - 2y + 5x^2y^2$$

9. Sea $f(x, y) = x^3y + x^2y^2 - xy^2 + 3$. Hallar el desarrollo de Taylor de grado 2 de f en el punto $(1, 2)$.

10. Sea $f(x, y) = \cos x \cos y$. Hallar el desarrollo de Taylor de grado 2 de f en el punto (π, π).

11. Desarrollar $f(x, y, z) = xy - xz + 2yz - z^2 + xy^2 + xyz$ en potencias de $x - 1$, y e $z + 2$.

12. Dada la superficie $z = (x - 3)^2 + y^2$ calcular el plano tangente en el punto $(0, 0)$. A continuación, dibujar en la misma gráfica función y plano.

Capítulo 8

Optimización en funciones de varias variables

8.1. Introducción

En esta última práctica utilizaremos `wxMaxima` para resolver problemas de optimización de varias variables. El proceso requiere de varias etapas en las que será necesario utilizar comandos que ya han sido descritos en prácticas anteriores.

Dada la función objetivo, en primer lugar, se deberá calcular su diferencial o jacobiano (`jacobian`). A continuación para determinar los puntos críticos será preciso resolver un sistema de ecuaciones (`algsys`). Recordar que este sistema de ecuaciones se ha denominado *condición de primer orden* (CPO).

Tras calcular los puntos críticos, se procede a analizar cada uno de ellos utilizando la *condición de segundo orden* (CSO). Es decir, se cálcula el hessiano (`hessian`) de la función objetivo en cada punto crítico y se decide en función de su carácter como forma cuadrática (definido positivo, definido negativo, indefinido). Para determinar el carácter del hessiano, se pueden calcular sus valores propios directamente (`eigenvalues`), o bien a través de su polinomio característico (`charpoly`, `allroots`).

Lo descrito anteriormente corresponde a la optimización sin restricciones, si además el problema incorpora restricciones, puede ser necesario restringir la forma cuadrática al núcleo del jacobiano de la rectricción. En ese caso el comando `nullspace`, es útil pues permite calcular el núcleo de una aplicación lineal.

Con el objetivo de hacer está práctica autocontenida, dedicaremos la sección 2 a recordar los comandos que requeriremos.

En las siguientes secciones estudiaremos la optimización sin restricciones (o en abiertos), y con restricciones.

8.2. Comandos útiles

8.2.1. Jacobiano: `jacobian`

La sintaxis del comando que nos permite calcular el jacobiano en `wxMaxima` es la siguiente:

```
jacobian([f1(x1,...,xn),...,fm(x1,...,xn)], [x1,...,xn])
```

donde

- `[f1(x1,...,xn),..., fm(x1,...,xn)],`: es una función escrita en componentes. Si es una función a valores reales: $f : \mathbb{R}^n \to \mathbb{R}$ sólo tendra una componente, que **deberá ir entre corchetes** `[·]` .
- `[x1,...,xn]`: vector de variables de las que depende la función.

8.2.2. Resolución de sistemas de ecuaciones `algsys`

A lo largo del curso hemos estudidado distintos comandos para la resolución de ecuaciones: `solve, find_root, allroots`. El primero trabaja en forma análitica, el segundo de forma numérica, y el tercero es específico para polinomios.

En esta práctica, introduciremos un comando más: `algsys`. En realidad es un comando que llama a los anteriores en función de las características de las ecuaciones. Básicamente, primero trata de resolver el sistema de forma analítica (usa el comando `solve`) y si no es posible utiliza procedimientos numéricos.

Nuestro objetivo es la busqueda de soluciones analíticas siempre que sea posible, para indicar al comando `algsys` que agote esta vía se debe iniciarse la variable `algexact` a `true` antes de la primera llamada a `algsys`:

```
(%i) algexact:true;
```

La sintaxis de `algsys` es la siguiente:

```
algsys([eq1, eq2, ..., eqm], [x1, x2, ..., xn])
```

donde

- `[eq1, eq2, ..., eqm]`: es la lista de ecuaciones. La ecuación `eq1` equivale a `eq1 = 0`, es decir no es necesario escribir el segundo término de la ecuación si este es cero.
- `[x1, x2, ..., xn]`: es la lista de variables. El número de variable y ecuaciones no tiene que ser el mismo.

La función `algsys` devuelve una lista de soluciones, cada una de las cuales consistente a su vez en una lista de ecuaciones asociando valores a las variables x1, ..., xn que satisfacen el sistema. Si `algsys` no puede encontrar soluciones devuelve la lista vacía [...].

Por ejemplo, si queremos resolver el sistema:

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

procederíamos:

```
(%i) eq1:x^2 - y^2$
(%i) eq2:x^2+2*y^2-x-y-1$
(%i) algsys([eq1, eq2],[x,y]);
(%o)
```

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

Lo que nos daría 4 soluciones. Notar que se ha utilizado el operador de asignación `:` para definir las ecuaciones, y que en la segunda ecuación el término independiente: 1 se ha colocado en el primer miembro para convertir la ecuación en homogénea (igualada a cero).

8.2.3. Hessiano: `hessian`

`wxMaxima` dispone de la instrucción `hessian`, para determinar el hessiano:

```
hessian(f(x1,..., xn), [x1,...,xn])
```

donde

- `f(x1,...,xn)`: es la función. Notar que en este caso la función no va delimitada por corchetes, `[ · ]`.
- `[x1,...,xn]`: es el vector de variables.

8.2.4. Valores propios: `eigenvalues`

Para el cálculo de los valores propios `wxMaxima` dispone del comando `eigenvalues` cuya sintaxis es:

```
eigenvalues(matriz)
```

donde

- `matriz`: es la matriz cuadrada, de la que se desean calcular los valores propios. Es conveniente recordar que para definir matrices se utiliza el comando `matrix`:

  ```
  matrix([a11,a12,..., a1n],...,[am1,am2,...,amn])
  ```

  La función `eigenvalues` devuelve una lista con sublistas, la primera de ellas son los valores propios y la segunda las multiplicidades. En esta práctica, sólo nos interesa la primera sublista. Por ejemplo, para calcular los valores propios de la matriz:

  $$A = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{pmatrix}$$

  procederíamos del siguiente modo:

  ```
  (%i) A: matrix([4,0,0],[0,2,2],[0,2,2])$
  (%i) eigenvalues(A);
  (%o) [[0,4],[1,2]]
  ```

  El resultado quiere decir que la matriz A tiene dos valores propios: 0 con multiplicidad 1 y 4 con multiplicidad 2. Sin embargo, en esta práctica sólo nos interesa saber, que un valor propio es cero y el otro positivo, por tanto la forma cuadrática es semidefinido positiva.

8.2.5. Polinomio característico: `charpoly`

Recordar que toda matriz simétrica real (es el caso de una matriz que corresponda a la evaluación del hessiano en un punto) tiene todos sus valores propios reales y diagonaliza. En ocasiones el comando `eigenvalues`, no funciona correctamente (por ejemplo proporciona valores propios imaginarios para matrices simétricas) y es necesario utilizar un procedimiento alternativo. Una posibilidad es calcular el polinomio característico. Para ello disponemos del comando `charpoly` cuya sintaxis es la siguiente:

```
charpoly(matriz, var)
```

donde

- `matriz`: es una matriz cuadrada.
- `var`: es la variable del polinomio característico. Por ejemplo, `x`.

Por ejemplo, el polinomio característico de la matriz A se calcularía:

```
(%i) charpoly(A,x);
(%o) ((2 - x)^2 - 4)(4 - x)
```

al que se puede anteponer el comando `expand` si se desea obtener el polinomio simplificado:

```
(%i) expand(charpoly(A,x));
(%o) -x^3 + 8x^2 - 16x
```

Utilizando la regla de Descartes, se puede deducir que el polinomio anterior tiene dos raíces reales positvas (dos cambios de signo), y una raíz nula (no existe término independiente). Lo que nos proporcionaría la misma información que en la sección anterior: la matriz A es semidefinido positiva.

8.2.6. Raíces de un polinomio: `allroots`

En lugar de usar la regla de Descartes, también se pueden calcular directamente las raíces mediante el comando `allroots` cuya sintaxis es:

```
allroots(polinomio)
```

donde

- `polinomio`: es un polinomio. Notar que esta función **sólo** funciona con polinomios.

Por ejemplo, para calcular las raíces del polinomio anterior se procedería:

```
(%i) allroots(-x^3+8*x^2-16*x);
(%o) [x=0.0,x=4.0,x=4.0]
```

8.2.7. Definiciones: `define`

Como ya vimos en las prácticas anteriores el comando `define` es útil para definir funciones que requieren el uso de operadores de diferenciación (`diff, jacobian, hessian`). Su sintaxis es la siguiente:

```
define(nombre, defincón)
```

donde

- `nombre`: es el nombre de la función incluyendo las variables de las que depende.
- `definición`: es la definición de la función.

Por ejemplo, si se desea definir el hessiano de la función $f(x, y)$ definida previamente escribiríamos:

```
(%i) define(Hf(x,y), hessian(f(x,y), [x,y]) );
```

8.2.8. Base del núcleo de una aplicación: `nullspace`

Si A es la matriz asociada a una aplicación, el comando `nullspace` proporciona una base del núcleo de la apliación.

Por ejemplo, si $f(x, y, z) = xyz$, para calcular una base del núcleo de la apliación diferencial en el punto $(1, 1, 1)$, se procede del siguiente modo.

- Se definen función y jacobiano:

```
(%i) f(x,y,z):= x*y*z;
(%i) define(Jf(x,y,z), jacobian([f(x,y,z)],[x,y,z]));
```

- Se calcula el núcleo:

```
(%i) nullspace(Jf(1,1,1))
(%o)
```

$$\text{span}\left(\begin{pmatrix}-1\\1\\0\end{pmatrix},\begin{pmatrix}0\\1\\-1\end{pmatrix}\right)$$

Lo que quiere decir que el subespacio núcleo esta generado por los vectores $(-1, 1, 0), (0, 1, -1)$. En esta práctica será conveniente almacenar estos vectores en una matriz:

```
(%i) BJ : matrix([-1,0], [1,1],[0,-1]);
(%o)
```

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

8.3. Optimización sin restricciones

8.3.1. Problema y esquema de resolución

El problema que debemos resolver en este caso es:

Calcular los extremos locales de la función $f(x_1, \ldots, x_n)$

Para ello procedemos del siguiente modo:

1. Defínción de la función objetivo: f.
2. Cálculo del Jacobiano: $Jf = \left(\frac{\partial f}{\partial x_1}\ \frac{\partial f}{\partial x_2}\ \ldots\ \frac{\partial f}{\partial x_n}\right)$.
3. Cálculo de los puntos críticos. Para ello aplicamos la condición de primer orden (CPO), es decir se resuelve el sistema:

$$\begin{aligned}\frac{\partial f}{\partial x_1} &= 0 = Jf[1,1]\\ \frac{\partial f}{\partial x_2} &= 0 = Jf[1,2]\\ &\ldots\\ \frac{\partial f}{\partial x_n} &= 0 = Jf[1,n]\end{aligned}$$

Lo que figura en la parte derecha, es la expresión que utilizaremos en `wxMaxima` para invocar las parciales. `wxMaxima` devuelve una matriz al usar el comando

`jacobian` y es por ello que los elementos deben recuperarse como miembros de una matriz.

Tras resolver el sistema obtendremos una lista de puntos críticos:

$$\{(x_1^i, x_2^i, \ldots, x_n^i)\}_{i=1}^N$$

`wxMaxima` proporcionará soluciones reales e imaginarias. De esa lista se deben eliminar las soluciones imaginarias pues no corresponden al dominio de la función que estamos considerando.

4. Clasificación de los puntos críticos. Para ello utilzaremos la condición de segundo orden (CSO).

   En primer lugar se calcula el hessiano de la función f: Hf. A continuación para cada punto crítico: $\{(x_1^i, x_2^i, \ldots, x_n^i)\}$

   a) Calcular el hessiano de f en el punto crítico: $Hf((x_1^i, x_2^i, \ldots, x_n^i))$.

   b) Clasificación de $Hf((x_1^i, x_2^i, \ldots, x_n^i))$ como forma cuadrática

   - Si es **definida positva** el punto $(x_1^i, \ldots, x_n^i)$ es **mínimo local**.
   - Si es **definida negativa** el punto $(x_1^i, \ldots, x_n^i)$ es **máximo local**.
   - Si es **indefinida** el punto $(x_1^i, \ldots, x_n^i)$ es **punto silla**.
   - Si es **semidefinida** la CSO **no decide**.

8.3.2. Ejemplo

Aplicaremos el esquema anterior para resolver el siguiente problema:

Calcular los extremos relativos de $f(x, y) = x^3 + 3x^2y - 2y^2(1 + y^2)$

1. Defínción de la función objetivo:

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

2. Definición y cálculo del Jacobiano:

```
(%i) define(Jf(x,y), jacobian([f(x,y)],[x,y]));
```

3. Cálculo de los puntos críticos:

```
(%i) algexact:true;
(%i) algsys( [ Jf(x,y)[1,1], Jf(x,y)[1,2] ], [x,y] );
(%o)
```

$$[[x = 0, y = -\frac{i}{\sqrt{2}}], [x = 0, y = \frac{i}{\sqrt{2}}], [x = 0, y = 0], [x = -1, y = \frac{1}{2}], [x = -2, y = 1]]$$

Hemos obtenido 5 soluciones, de las que descartamos las dos primeras por ser algunas de sus componentes números imaginarios. Los puntos críticos serán:

$$(0, 0), (-1, \frac{1}{2}), (-2, 1)$$

4. Clasificación de puntos críticos. Definimos el hessiano:

```
(%i) define( Hf(x,y), hessian( f(x,y), [x,y] ) );
```

y a continuación analizamos cada punto crítico:

- Punto $(0, 0)$: Calculamos el Hessiano y sus valores propios:

```
(%i) Hf(0,0);
(%o)
```

$$\begin{pmatrix} 0 & 0 \\ 0 & -4 \end{pmatrix}$$

```
(%i) eigenvalues(Hf(0,0));
(%o)[[-4,0],[1,1]]
```

De donde deducimos que el hessiano tiene por valores propios -4 y 0, por tanto es **semidefinido negativa** y concluímos que la CSO **no decide**.

- Punto $(-1, \frac{1}{2},)$: Calculamos el Hessiano y sus valores propios:

```
(%i) Hf(-1,1/2);
(%o)
```

$$\begin{pmatrix} -3 & -6 \\ -6 & -10 \end{pmatrix}$$

```
(%i) eigenvalues(Hf(-1,1/2));
(%o)
```

$$[[-\frac{\sqrt{193}+13}{2}, \frac{\sqrt{193}-13}{2}], [1,1]]$$

puesto que no queda claro el signo de los valores propios, podemos calcular su expresión decimal

```
(%i) float( eigenvalues(Hf(-1,1/2)));
(%o)[[-13.4462219947249,0.446221994724902],[1.0,1.0]]
```

De donde deducimos que el hessiano es **indefinido** y concluímos que el punto $(-1, 1/2)$ es un **punto silla**.

- Punto $(-2, 1)$: Calculamos el Hessiano y sus valores propios:

```
(%i) Hf(-2,1);
(%o)
```

$$\begin{pmatrix} -6 & -12 \\ -12 & -28 \end{pmatrix}$$

```
(%i) float(eigenvalues(Hf(-2,1)));
(%o)[[-33.27882059609971,-0.7211794039002939],[1.0,1.0]]
```

De donde deducimos que el hessiano es **definido negativo** y concluímos que el punto $(-2, 1)$ es un **máximo local**.

8.3.3. Teorema local-global

El teorema local-global permite clasificar extremos relativos como extremos absolutos si la función es concava o convexa, en concreto:

> Sea $f : A \to \mathbb{R}$ una función real, con A abierto y convexo, y f diferenciable, entonces:
>
> - Si f es **concava**, todo punto crítico es **máximo**.
> - Si f es **convexa**, todo punto crítico es **mínimo**.

La concavidad y convexidad puede ser determinada según el carácter del Hessiano, de hecho se puede demostrar que:

- f es **cóncava** en $A \Leftrightarrow Hf$ es **semidefinida negativa** en todo $x \in A$.
- f es **convexa** en $A \Leftrightarrow Hf$ es **semidefinida positva** en todo $x \in A$.

Para ilustrar el uso de este teorema resolveremos el siguiente ejercicio:

Ejercicio. Justificar que la función $f(x,y) = x^2 + y(y^3 - 4)$ tiene un extremo global en $\mathbb{R}^2$, indicando de qué tipo es.

Si procedemos como en la sección anterior:

- Definición de la función objetivo:

```
(%i) f(x,y):= x^2 + y * ( y^3 -4);
```

- Definición y cálculo del Jacobiano:

```
(%i) define(Jf(x,y), jacobian([f(x,y)],[x,y]));
```

- Cálculo de los puntos críticos:

```
(%i) algexact:true;
(%i) algsys( [ Jf(x,y)[1,1], Jf(x,y)[1,2] ], [x,y] );
(%o)
```

$$[[x=0, y=1], [x=0, y=-\frac{\sqrt{3}\,i+1}{2}], [x=0, y=\frac{\sqrt{3}\,i-1}{2}]]$$

Hemos obtenido, 3 soluciones, de las que descartamos las dos últimas por ser algunas de sus componentes números imaginarios. El único puntos críticos es:

$$(0,1)$$

- Clasificamos el puntos crítico. Definimos el hessiano:

```
(%i) define( Hf(x,y), hessian( f(x,y), [x,y] ) );
(%o)
```

$$\begin{pmatrix} 2 & 0 \\ 0 & 12\,y^2 \end{pmatrix}$$

y a continuación analizamos cada punto crítico. Calculamos el Hessiano y sus valores propios:

```
(%i) Hf(0,1);
(%o)
```

$$\begin{pmatrix} 2 & 0 \\ 0 & 12 \end{pmatrix}$$

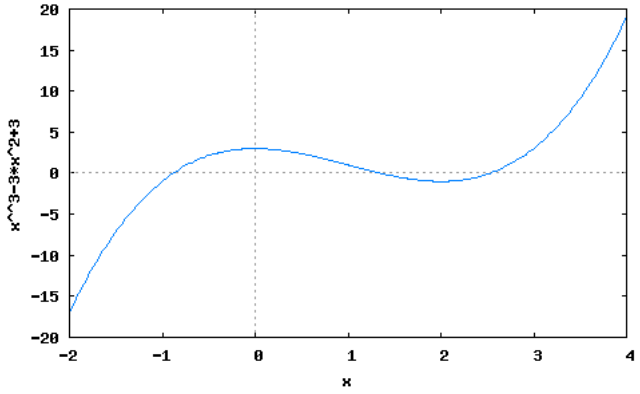
Dado que la matriz es diagonal, deducimos que los valores propios son 2 y 12 ambos positivos. Por tanto se trata de un máximo local.

Para determinar que además es global veamos si el hessiano es semidefinido positivo en todo punto $(x,y) \in \mathbb{R}^2$. Algo que es evidente, pues los valores propios para un punto (x,y) resultan ser:

$$2 \geq 0 \qquad 12y^2 \geq 0 \text{ para todo } (x,y) \in \mathbf{R}^2.$$

De aquí concluímos que la función $f(x,y)$ es convexa, y aplicando el teorema local-global podemos deducir que el punto $(0,1)$ es un máximo global, como podemos comprobar en el siguiente gráfico:

```
(%i) wxplot3d(f(x,y), [x,-1,1], [y,0,2]);
```



Nota: Todo lo que se ha descrito en esta sección es válida para problemas del tipo:

$$\begin{aligned} &\text{Optimizar } f(x_1, \ldots, x_n) \\ &\text{s.a.} \qquad \varphi(x_1, \ldots, x_n) < 0 \end{aligned}$$

La única diferencia con lo descrito es que sólo deben considerarse los puntos críticos que pertenecen al abierto, es decir, que verifcan la restricción.

8.4. Optimización con restricciones de igualdad

8.4.1. Extremos locales.

Dada una función objetivo $f(x_1, \ldots, x_n)$ y una restricción $\varphi(x_1, \ldots, x_n)$ en este caso nos planteamos el cáculo de los extremos relativos de f *condicionados* (o sujetos) a $\varphi(x_1, \ldots, x_n) = 0$, es decir:

$$\begin{aligned} &\text{Optimizar } f(x_1, \ldots, x_n) \\ &\text{s.a.} \qquad \varphi(x_1, \ldots, x_n) = 0 \end{aligned}$$

Nota: La restricción φ pude ser una función a valores vectoriales es decir, $\varphi = (\varphi_1, \ldots, \varphi_m)$. En los ejercicios que resolveremos en esta práctiva trabajaremos con 2 ó 3 variables y 1 ó 2 restricciones.

Existen dos procedimientos para resolver el problema anterior. El primero consiste en utilizar la restricción para *eliminar* (tras despejar) alguna o algunas de las variables, y de ese modo reducir el problema a uno de las características de los que fue analizado en la sección anterior. La segunda alternativa esta basada en el uso de los llamados *multiplicadores de Lagrange*.

8.4.2. Reducción a un extremo ordinario

El procedimiento es básicamente el mismo que el de la sección anterior con dos diferencias: es necesario redefinir la función objetivo de acuerdo a la restricción, y en este caso no existen puntos de ensilladura:

1. Definición de la función objetivo: F.

   Usando la restricción $\varphi = (\varphi_1, \varphi_2, \ldots, \varphi_m)$ se despejan m variables (supongamos que son las últimas) de $(x_1, \ldots x_n)$ en función de las $n - m$ restantes:

$$\left.\begin{array}{ll} \varphi_1(x_1,\ldots,x_n) & = 0 \\ \ldots & \\ \varphi_m(x_1,\ldots,x_n) & = 0 \end{array}\right\} \Rightarrow \left.\begin{array}{ll} x_{n-m+1} & = \phi_{n-m+1}(x_1,\ldots,x_{n-m}) \\ \ldots & \\ x_n & = \phi_n(x_1,\ldots,x_{n-m}) \end{array}\right\}$$

$\Rightarrow f(x_1,\ldots,x_{n-m},\phi(x_1,\ldots,x_{n-m}),\ldots,\phi(x_1,\ldots,x_{n-m}))$

$\Rightarrow F(x_1,\ldots,x_{n-m})$

Es decir, conseguimos expresar la función objetivo en función de $n-m$ variables.

Nota: Para poder aplicar este procedimiento es necesario que sea posible *despejar* $n-m$ variables. Formalmente, esto es posible si:

$$\det \begin{pmatrix} \frac{\partial \varphi_1}{\partial x_{n-m+1}} & \ldots & \frac{\partial \varphi_1}{\partial x_n} \\ \ldots & \ldots & \ldots \\ \frac{\partial \varphi_m}{\partial x_{n-m+1}} & \ldots & \frac{\partial \varphi_m}{\partial x_n} \end{pmatrix}_a \neq 0,$$

para todo a punto crítico. Si la condición anterior no se cumple es posible que durante el proceso se produzca la perdida de algún punto relevante.

2. Cálculo del Jacobiano de $F(x_1,\ldots,x_{n-m})$:

3. Cálculo de los puntos críticos mediante la condición de primer orden (CPO)

4. Clasificación de los puntos críticos.

   Se calcula el hessiano de la función F: HF. A continuación, para cada punto crítico: $\{(x_1^i, x_2^i, \ldots, x_{n-m}^i)\}$

   a) Calcular el hessiano de F en el punto crítico: $HF((x_1^i, x_2^i, \ldots, x_{n-m}^i))$.

   b) Clasificación de $HF((x_1^i, x_2^i, \ldots, x_{n-m}^i))$ como forma cuadrática

   - Si es **definido positiva** el punto $(x_1^i, \ldots, x_{n-m}^i, \ldots, x_n^i)$ es **mínimo local** (donde las últimas coordenadas se determinan en función de las funciones ϕ_i).
   - Si es **definida negativa** el punto $(x_1^i, \ldots, x_{n-m}^i, \ldots, x_n^i)$ es **máximo local** (donde las últimas coordenadas se determinan en función de las funciones ϕ_i).
   - En otro caso, **no decide**.

8.4.3. Ejemplo

Utilizaremos el esquema anterior para calcular los extremos relativos de $f(x,y,z) = x^2+y^2+z^2$ a lo largo de la curva $g(x,y,z) = 2x^2+2y-z+3=0$:

$$\begin{array}{ll} \text{Optimizar } x^2+y^2+z^2 & \\ \text{s.a.} & 2x+2y-z+3=0 \end{array}$$

1. Definición de la función objetivo: Desde la restricción, observamos que es sencillo despejar la variable z en función de las restantes:

$$z = 2x^2+2y+3$$

De modo que definimos la función f, la función ϕ y la función objetivo F:

```
(%i) f(x,y,z) := x^2 + y^2 + z^2;
(%i) phi(x,y) := 2*x+2*y+3;
(%i) F(x,y) := f(x,y,phi(x,y));
```

Podemos comprobar como tras la sustitución F sólo depende de dos variables:

```
(%i) F(x,y);
(%o)
```

$$(2\,y+2\,x+3)^2+y^2+x^2$$

2. Definición y cálculo del Jacobiano:

```
(%i) define(JF(x,y), jacobian([F(x,y)],[x,y]));
```

3. Cálculo de los puntos críticos:

```
(%i) algexact:true;
(%i) algsys( [ JF(x,y)[1,1], JF(x,y)[1,2] ], [x,y] );
(%o)
```

$$[[x=-\frac{2}{3},y=-\frac{2}{3}]]$$

Hemos obtenido un único punto crítico.

4. Clasificación de puntos críticos. Definimos el hessiano:

```
(%i) define( HF(x,y), hessian( F(x,y), [x,y] ) );
(%o)
```

$$\mathrm{HF}\,(x,y):=\begin{pmatrix}10 & 8\\ 8 & 10\end{pmatrix}$$

y a continuación analizamos el punto crítico:

```
(%i) eigenvalues(HF(x,y));
(%o)[[18,2],[1,1]]
```

resultando que el hessiano es definido positivo (tiene por valores propios el 18 y el 2). Por tanto el punto

$$(-\frac{2}{3},-\frac{2}{3},\phi(-\frac{2}{3},-\frac{2}{3}))=(-\frac{2}{3},-\frac{2}{3},\frac{1}{3})$$

es mínimo local condicionado.

8.4.4. Multiplicadores de Lagrange

En ocasiones no es posible utilizar la restricciones para eliminar algunas variables y es preciso utilizar el método de los multiplicadores de Lagrange. Para poder hacer uso de este procedimiento se requiere que la restricción cumpla la denominada *condición de regularidad*:

$$\mathrm{rang}(d_a\varphi)=\mathrm{rang}\begin{pmatrix}\frac{\partial\varphi_1}{\partial x_n} & \cdots & \frac{\partial\varphi_1}{\partial x_n}\\ \cdots & \cdots & \cdots\\ \frac{\partial\varphi_m}{\partial x_n} & \cdots & \frac{\partial\varphi_m}{\partial x_n}\end{pmatrix}_a=m,$$

para todo a punto crítico.

Expondremos ahora este procedimiento:

1. Defínción de la función objetivo: L. En este caso se denomina *lagrangiano*.
   A partir de la restricción $\varphi = (\varphi_1, \varphi_2, \ldots, \varphi_m)$ se define la función:

$$\begin{aligned} L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) =& f(x_1, \ldots, x_n) \\ & - \lambda_1 \varphi_1(x_1, \ldots, x_n) - \ldots - \lambda_m \varphi_m(x_1, \ldots, x_n) \end{aligned}$$

2. Cálculo del Jacobiano del Lagrangiano $L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m)$:

$$JL = \left(\frac{\partial L}{\partial x_1}, \ldots, \frac{\partial L}{\partial x_n}, \frac{\partial L}{\partial \lambda_1}, \ldots, \frac{\partial L}{\partial \lambda_m} \right)$$

3. Cálculo de los puntos críticos mediante la la condición de primer orden (CPO)

$$\begin{aligned} &\frac{\partial L}{\partial x_1} = 0 = JL[1,1] \\ &\ldots \\ &\frac{\partial L}{\partial x_n} = 0 = JL[1,n] \\ &\frac{\partial L}{\partial \lambda_1} = 0 = JL[1,n+1] \\ &\ldots \\ &\frac{\partial L}{\partial \lambda_m} = 0 = JL[1,n+m] \end{aligned}$$

   De donde se obtiene una colección de puntos críticos:

$$\left\{ (x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i) \right\}$$

4. Clasificación de los puntos críticos.

   Se calcula el hessiano del Lagrangiano L con respecto a las variables originales $(x_1, \ldots, x_n)$: HL. A continuación, para cada punto crítico: $(x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i)$

   a) Calcular el hessiano de L en el punto crítico: $HL(x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i)$.

   b) Clasificación de $HL(x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i)$ como forma cuadrática

   - Si es **definida positva** el punto $(x_1^i, \ldots, x_n^i)$ es **mínimo local condicionado**.
   - Si es **definida negativa** el punto $(x_1^i, \ldots, x_n^i)$ es **máximo local condicionado**.
   - En otro caso:
     i) Calcular el jacobiano de la restricción $\varphi(x_1^i, x_2^i, \ldots, x_n^i)$ en el punto crítico.
     ii) Calcular una base del núcleo del jacobiano, que la almacenamos en forma matricial y la denotamos BJ.
     iii) Restringir el hessiano $HL(x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i)$ al núcleo del jacobiano. Es decir, calcular

$$HR = BJ^t \cdot HL(x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i) \cdot BJ$$

     iv) Clasificación de HR:

- Si HR es definido positivo entonces $(x_1^i, x_2^i, \ldots, x_n^i)$ es un mínimo local condicionado
- Si HR es definido negativo entonces $(x_1^i, x_2^i, \ldots, x_n^i)$ es un máximo local condicionado.
- Si HR es indefinido $(x_1^i, x_2^i, \ldots, x_n^i)$ no es extremo local.
- En otro caso, no decide.

8.4.5. Ejemplo. Dos restricciones.

Calcular los extremos relativos de la función $f(x, y, z) = x^2+y^2+z^2$ condicionado a $2x + y + z = 2$ y $x - y - 3z = 4$:

$$\begin{aligned} &\text{Optimizar } x^2 + y^2 + z^2 \\ &\text{s.a.} \qquad 2x + y + z = 2 \\ &\qquad\quad\ \ x - y - 3z = 4 \end{aligned}$$

1. Defínción del lagrangiano. Definimos la función f, las restricciones $g1$, $g2$ (notar que todos los términos se han pasado al primer miembro). Denotaremos por L al lagrangiano y usaremos $l1$ y $l2$ como multiplicadores:

```
(%i) f(x,y,z):= x^ 2 + y^ 2 + z^ 2;
(%i) g1(x,y,z):= 2*x+ y+z - 2;
(%i) g2(x,y,z):= x-y-3*z-4;
(%i) L(x,y,z,l1,l2):= f(x,y,z) - l1*g1(x,y,z)-l2*g2(x,y,z);
```

2. Definición y cálculo del Jacobiano:

```
(%i) define(JL(x,y,z,l1,l2), jacobian([L(x,y,z,l1,l2)],[x,y,z,l1,l2]));
```

3. Cálculo de los puntos críticos:

```
(%i) algsys( [ JL(x,y,z,l1,l2)[1,1],
JL(x,y,z,l1,l2)[1,2] ,
JL(x,y,z,l1,l2)[1,3],
JL(x,y,z,l1,l2)[1,4],
JL(x,y,z,l1,l2)[1,5] ], [x,y,z,l1,l2] );
(%o)
```

$$[[x = \frac{14}{25}, y = -\frac{1}{5}, z = -\frac{27}{25}, l1 = \frac{6}{25}, l2 = \frac{16}{25}]]$$

Hemos obtenido un único punto crítico.

4. Clasificación de puntos críticos. Definimos el hessiano (notar que sólo derivamos con respecto a las variables originales):

```
(%i) define( HL(x,y,z,l1,l2), hessian( L(x,y,z,l1,l2), [x,y,z]
) );
(%o)
```

$$HL(x, y, z, l1, l2) := \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Notar que el hessiano es constante y definido positivo (tiene valor propio 2 con multiplicidad 3) y por tanto concluímos que el punto

$$\left(\frac{14}{25}, -\frac{1}{5}, -\frac{27}{25}\right)$$

es mínimo local condicionado.

8.4.6. Ejemplo. Rectricción al núcleo del jacobiano.

Calcular los extremos relativos de la función $f(x,y,z) = xy + yz + xz$ condicionado a $x + y + z = 6$:

$$\begin{aligned} &\text{Optimizar } xy + yz + xz \\ &\text{s.a.} \qquad\quad x + y + z = 6 \end{aligned}$$

1. Definición del lagrangiano. Definimos la función f, las restricciones (notar que todos los términos se han pasado al primer miembro). Denotaremos por L al lagrangiano y usaremos l como multiplicador:

```
(%i) f(x,y,z):= x*y + y*z + x*z;
(%i) g(x,y,z):= x+y+z-6;
(%i) L(x,y,z,l):= f(x,y,z) - l*g(x,y,z);
```

2. Definición y cálculo del Jacobiano:

```
(%i) define(JL(x,y,z,l), jacobian([L(x,y,z,l)],[x,y,z,l]));
```

3. Cálculo de los puntos críticos:

```
(%i) algsys( [ JL(x,y,z,l)[1,1],
JL(x,y,z,l)[1,2],
JL(x,y,z,l)[1,3],
JL(x,y,z,l)[1,4] ], [x,y,z,l] );
(%o) [[x=2,y=2,z=2,l=4]]
```

   Hemos obtenido un único puntos crítico.

4. Clasificación de puntos críticos. Definimos el hessiano (notar que sólo derivamos con respecto a las variables originales):

```
(%i) define( HL(x,y,z,l), hessian( L(x,y,z,l), [x,y,z] )
);
(%o)
```

$$HL(x,y,z,l) := \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

   Hemos obtenido un hessiano constante, que ahora tratamos de clasificar:

```
(%i) eigenvalues(HL(2,2,2,4));
(%o)[[2,-1],[1,2]]
```

   de donde resulta que el hessiano es indefinido. Por tanto, para concluir debemos restringuir al núcleo del jacobiano de la restricción φ.

   i) Calculamos el jacobiano de la restricción:

```
(%i) define( Jg(x,y,z), jacobian([g(x,y,z)], [x,y,z]));
```

   ii) Caclulamos una base del núcleo del jacobiano y lo almacenamos en forma de matriz.

```
(%i) nullspace(Jg(2,2,2))
(%o)
```

$$\text{span}\left(\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}\right)$$

```
(%i) BJ : matrix([-1,0],[1,1],[0,-1])
```

iii) Restringimos al núcleo el hessiano:

```
(%i) HR : transpose(BJ). HL(2,2,2,4) . BJ
(%o)
```

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

iv) Finalmente clasificamos HR como forma cuadrática:

```
(%i) eigenvalues(HR);
(%o)[[−3, −1], [1, 1]]
```

y deducimos que el punto $(2, 2, 2)$ es un máximo local condicionado.

8.4.7. Extremos globales.

El problema modelo en este caso sería: dada una función objetivo $f(x_1, \ldots, x_n)$ y una restricción $\varphi(x_1, \ldots, x_n)$ nos planteamos el cálculo de los extremos globales de f *condicionados* (o sujetos) a $\varphi(x_1, \ldots, x_n) = 0$.

Si la restricción φ determina un conjunto compacto, el problema anterior se simplifica gracias al teorema de Weierstrass y no requiere el uso de las condiciones de segundo orden.

Teorema de Weierstrass. Si $f : A \to \mathbb{R}$ es continua y $A \subset \mathbb{R}^n$ es un compacto (cerrado y acotado) entonces f alcanza un máximo y un mínimo en A.

Utilizando este teorema basta con una vez calculados los punto críticos, evaluar la función en dichos puntos para determinar el máximo y mínimo. La dificultad en este tipo de problemas radica en asegurar si la restricción delimita o no un compacto. Sin embargo, en `wxMaxima` y si la dimensión es 2 o 3 siempre es posible dibujar la curva o superficie asociada.

Es decir, el esquema sería el siguiente:

1. Comprobar que la restricción determina un compacto.

2. Defindión del lagrangiano.

$$\begin{aligned} L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) = & f(x_1, \ldots, x_n) \\ & - \lambda_1 \varphi_1(x_1, \ldots, x_n) - \ldots - \lambda_m \varphi_m(x_1, \ldots, x_n) \end{aligned}$$

3. Cálculo del Jacobiano del Lagrangiano $L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m)$:

4. Cálculo de los puntos críticos mediante la la condición de primer orden (CPO)

$$\left\{(x_1^i, x_2^i, \ldots, x_n^i, \lambda_1^i, \ldots, \lambda_m^i)\right\}$$

5. Para cada punto crítico se calcula:

$$f(x_1^i, x_2^i, \ldots, x_n^i)$$

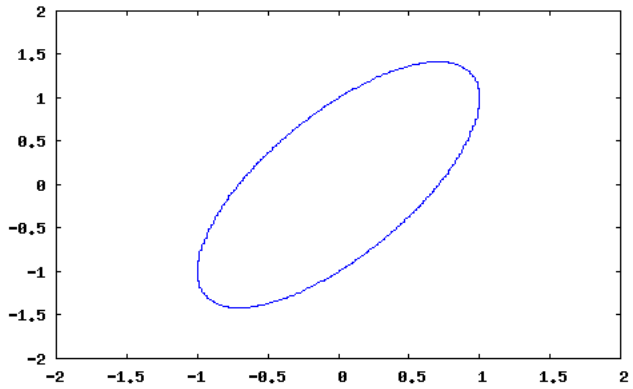
El punto (o los puntos) crítico (s) que proporcionen el mayor/menor valor será máximo/mínimo absoluto respectivamente.

8.4.8. Ejemplo

Calcular los extremos absolutos de la función $f(x,y) = x^2 + y^2 - 2xy$ condicionados a $g(x,y) = 2x^2 + y^2 - 2xy - 1 = 0$.

1. Comprobar que la restricción $g(x,y) = 0$ delimita un compacto. Para ello vamos a dibujar la curva $g(x,y) = 0$ con el paquete draw y comprobar que es un elipse:

```
(%i) g(x,y) := 2*x^2+y^2 - 2*x*y -1;
(%i) load(draw);
(%i) wxdraw2d(implicit(g(x,y)=0, x, -2,2,y,-2,2));
(%o)
```



2. Definición del lagrangiano.

```
(%i) f(x,y ) : =x^2+y^2-2*x*y;
(%i) L(x,y,l):= f(x,y) - l*g(x,y);
```

3. Cálculo del Jacobiano del Lagrangiano :

```
(%i) define(JL(x,y,l) , jacobian([L(x,y,l)],[x,y,l]));
```

4. Cálculo de los puntos críticos mediante la la condición de primer orden (CPO)

```
(%i) algsys([JL(x,y,l)[1,1],JL(x,y,l)[1,2],JL(x,y,l)[1,3]],[x,y,l]);
(%o)
```

$$[[x=-1, y=-1, l=0], [x=1, y=1, l=0], [x=0, y=1, l=1], [x=0, y=-1, l=1]]$$

Hemos obtenido cuatro puntos críticos.

5. Para cada punto crítico se cálcula:

```
(%i) f(-1,1);
(%o) 4
(%i) f(1,1);
(%o) 1
(%i) f(0,1);
(%o) 0
(%i) f(0,-1);
(%o) 1
```

de donde obtendríamos que el punto $(-1,1)$ es máximo global y $(1,1)$ es mínimo global.

8.5. Optimización con restricciones de desigualdad

8.5.1. Extremos locales

Buscamos en este caso determinar los extremos de una funcón $f(x_1, \ldots, x_n)$ sujeta a la restricción $\varphi(x_1, \ldots, x_n) \leq 0$, cuya formulación es:

$$\begin{aligned} &\text{Optimizar } f(x_1, \ldots, x_n) \\ &\text{s.a.} \qquad \varphi(x_1, \ldots, x_n) \leq 0 \end{aligned}$$

En principio la función $\varphi(x_1, \ldots, x_n)$ podría ser de naturaleza vectorial, lo que correspondería a varias restricciones. Con el objetivo de simplificar la práctica y la exposición, en lo que sigue $\varphi(x_1, \ldots, x_n)$ denotará una función real, es decir en la formulación del problema aparecerá una **única** restricción.

El problema anterior se puede descomponer en dos subproblemas:

1. Optimización sin restriccón:

$$\begin{aligned} &\text{Optimizar } f(x_1, \ldots, x_n) \\ &\text{s.a.} \qquad \varphi(x_1, \ldots, x_n) < 0 \end{aligned}$$

   que se resuelve como fue descrito en la sección 8.3, pero **sólo se adminten** los extremos que cumplan la condición $\varphi(x_1, \ldots, x_n) < 0$.

2. Optimización con restricción de igualdad.

$$\begin{aligned} &\text{Optimizar } f(x_1, \ldots, x_n) \\ &\text{s.a.} \qquad \varphi(x_1, \ldots, x_n) = 0 \end{aligned}$$

   que se resuelve como fue indicado en la sección 8.4

Nota. Si la función $\varphi(x_1, \ldots, x_n)$ es de naturaleza vectorial, es decir existen varias restricciones, el problema original se divide en dos subproblemas: uno en el interior y otro en la frontera. La resolución sobre la frontera, puede dar lugar a su vez, a diferentes subproblemas que deben ser analizados en cada caso. Como adelantabamos al inicio de esa sección, este tipo de problemas no serán objeto de este curso.

8.5.2. Ejemplo

En este ejemplo calcularemos los extremos relativos de la función $f(x, y) = x^3 + y^3 + x^2 + 2y^2$ sujeta a la restricción $(x-1)^2 + y^2 \leq 1$.

Como avanzamos en la sección anterior, dividimos el problema en dos subproblemas:

- Optimización sin restricciones:

$$\begin{aligned} &\text{Optimizar } x^3 + y^3 + x^2 + 2y^2 \\ &\text{s.a.} \qquad (x-1)^2 + y^2 - 1 < 0 \end{aligned}$$

  1. Definición de la función objetivo:

```
(%i) f(x,y):= x^3+ y^3 + x^2 + 2*y^2;
```

  2. Definición y cálculo del Jacobiano:

```
(%i) define(Jf(x,y), jacobian([f(x,y)],[x,y]));
```

3. Cálculo de los puntos críticos:

```
(%i) algsys( [ Jf(x,y)[1,1], Jf(x,y)[1,2] ], [x,y] );
```

(%o) $[[x = 0, y = 0], [x = -2/3, y = 0], [x = 0, y = -4/3], [x = -2/3, y = -4/3]]$

Hemos obtenido, 4 soluciones, veamos cuáles verifican la restricción $g(x, y) = (x-1)^2 + y^2 - 1 < 0$:

```
(%i) g(x,y) = (x-1)^2 + y^2 - 1;
(%i) g(0,0);
(%o) 0
(%i) g(-2/3,0);
(%o) 16/9
(%i) g(0,-4/3);
(%o) 16/9
(%i) g(-2/3,-4/3);
(%o) 32/9
```

Resulta que ninguna de ellas verifica la restricción, por tanto no hay que analizar ningún punto crítico.

- Optimización con restricciones de igualdad.

$$\begin{aligned} &\text{Optimizar } x^3 + y^3 + x^2 + 2y^2 \\ &\text{s.a.} \qquad\quad (x-1)^2 + y^2 - 1 = 0 \end{aligned}$$

1. Definición del lagrangiano.

```
(%i) L(x,y,l):= f(x,y) - l*g(x,y);
```

2. Definición y cálculo del Jacobiano:

```
(%i) define(JL(x,y,l), jacobian([L(x,y,l)],[x,y,l]));
```

3. Cálculo de los puntos críticos:

```
(%i) algsys( [ JL(x,y,l)[1,1],
JL(x,y,l)[1,2] ,
JL(x,y,l)[1,3] ], [x,y,l] );
(%o)
```

$$\begin{aligned} &[[x = 2, y = 0, l = 8], [x = 0, y = 0, l = 0], \\ &[x = 1{,}12 - 1{,}23\,i, y = {,}095\,i + 1{,}58, l = {,}143\,i + 4{,}38], \\ &[x = 1{,}23\,i + 1{,}12, y = 1{,}58 - {,}095\,i, l = 4{,}38 - {,}143\,i], \\ &[x = {,}21 - {,}52\,i, y = {,}44\,i - {,}92, l = {,}67\,i + {,}61], \\ &[x = {,}52\,i + {,}21, y = -{,}44\,i - {,}92, l = {,}61 - {,}67\,i]] \end{aligned}$$

De las cuales sólo las dos primeras son reales.

4. Clasificación de puntos críticos. Definimos el hessiano

```
(%i) define( HL(x,y,l), hessian( L(x,y,l), [x,y] ) );
(%o)
```

$$HL(x, y, l) := \begin{pmatrix} 6\,x - 2\,l + 2 & 0 \\ 0 & 6\,y - 2\,l + 4 \end{pmatrix}$$

Ahora clasificamos el hessiano en cada punto crítico:

```
(%i) eigenvalues(HL(2,0,8));
(%o)
```

$$[[-12,-2],[1,1]]$$

de donde resulta que el punto $(2,0)$ es un máximo local. Procedemos del mismo modo con el punto $(0,0)$:

```
(%i) eigenvalues(HL(0,0,0));
(%o)
```

$$[[2,4],[1,1]]$$

por tanto $(2,4)$ es mínimo local.

Nota Como veremos en la última sección, el cálculo realizado en el apartado 4 del ejemplo anterior es innecesario. Dado que la restricción delimita un compacto (se trata de un círculo) el teorema de Weierstrass asegura la existencia de máximo y mínimo. Puesto que sólo se han obtenido dos puntos críticos forzosamente deben ser los dos que anticipa el teorema de Weierstrass. Por tanto una simple evaluación habría determinado su naturaleza sin ser preciso el cálculo del hessiano.

Notar que si se hubieran obtenido más de dos puntos críticos es preciso utilizar las CSO para determinar el tipo de extremo.

8.5.3. Extremos absolutos

Por último estudiamos el cálculo de los extremos absolutos de una funcón $f(x_1,\ldots,x_n)$ sujeta a la restricción $\varphi(x_1,\ldots,x_n)\leq 0$. En este caso, si la restricción $\varphi(x_1,\ldots,x_n)\leq 0$ delimita un compacto se procede igual que se hizo en la sección 8.4.7. Es decir, se calculan los puntos críticos del problema en el interior:

$$\begin{array}{ll}\text{Optimizar} & f(x_1,\ldots,x_n)\\ \text{s.a.} & \varphi(x_1,\ldots,x_n)<0\end{array}$$

y los puntos críticos del problema en la frontera:

$$\begin{array}{ll}\text{Optimizar} & f(x_1,\ldots,x_n)\\ \text{s.a.} & \varphi(x_1,\ldots,x_n)=0\end{array}$$

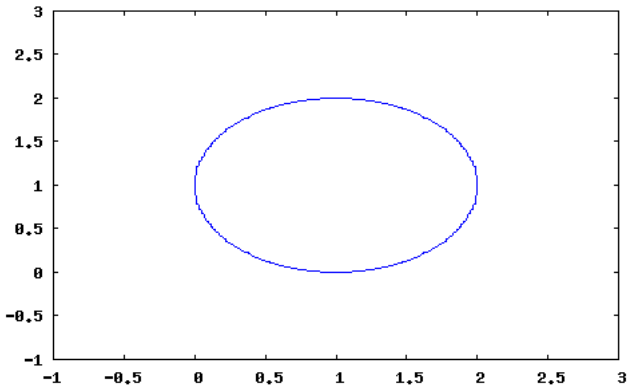
y después se evalua la función $f(x_1,\ldots,x_n)$ para decidir qué puntos crí icos son el máximo y mínimo.

8.5.4. Ejemplo.

Nos interesa determinar los extremos absolutos de la función $f(x,y)=x^3+y^2-2xy$ en el recinto $(x-1)^2+(y-1)^2\leq 1$.

Dado que la restricción delimita un compacto (es una circunferencia de centro $(1,1)$ y radio 1), usando el teorema de Weierstrass basta con calcular los puntos críticos que están en el recinto y evaluar en la función $f(x,y)$ para determinar los extremos.

```
(%i) g(x,y):= (x-1)^2+(y-1)^2-1;
(%i) load(draw);
(%i) wxdraw2d(implicit(g(x,y),x,-1,3,y,-1,3));
(%o)
```



Procedemos por tanto al cálculo de los puntos críticos:

- Interior $((x-1)^2+(y-1)^2-1<0)$

  1. Definición de la función objetivo:

     ```
     (%i) f(x,y):= x^3+ y^2 - 2*x*y;
     ```

  2. Definición y cálculo del Jacobiano:

     ```
     (%i) define(Jf(x,y), jacobian([f(x,y)],[x,y]));
     ```

  3. Cálculo de los puntos críticos:

     ```
     (%i) algsys( [ Jf(x,y)[1,1], Jf(x,y)[1,2] ], [x,y] );
     (%o)
     ```

     $$[[x=\frac{2}{3},y=\frac{2}{3}],[x=0,y=0]]$$

     Hemos obtenido, 2 soluciones, veamos cuáles verifican la restricción $g(x,y)=(x-1)^2+(y-1)^2-1<0$:

     ```
     (%i) g(0,0);
     (%o) 1
     (%i) g(2/3,2/3);
     (%o) -7/9
     ```

     Resulta que el punto $(2/3,2/3)$ está situado dentro del dominio de estudio y debe ser considerado.

- Frontera $((x-1)^2+(y-1)^2-1=0)$

  1. Definición del lagrangiano.

     ```
     (%i) L(x,y,l):= f(x,y) - l*g(x,y);
     ```

  2. Definición y cálculo del Jacobiano:

     ```
     (%i) define(JL(x,y,l), jacobian([L(x,y,l)],[x,y,l]));
     ```

  3. Cálculo de los puntos críticos:

     ```
     (%i) algsys( [ JL(x,y,l)[1,1],
     JL(x,y,l)[1,2],
     JL(x,y,l)[1,3] ], [x,y,l] );
     (%o)
     ```

     $$[[x=1{,}972939944134078,y=,7689417658108955,l=5{,}210800744878958],$$
     $$[x=0{,}25651518414818,y=1{,}668752793920429,l=2{,}111748295102294],$$
     $$[x=-1{,}45\,i-{,}21,y=1{,}06\,i-{,}66,l=,87-,95\,i],$$
     $$[x=1{,}45\,i-{,}21,y=-1{,}06\,i-{,}66,l=,95\,i+,87],$$
     $$[x=1{,}193428722840488,y=1{,}981114258734655,l=,8028479403288693],$$
     $$[x=,3385499207606973,y=,2500108061378863,l=,1180538555691554]]$$

De los cuales los dos primeros y dos últimos son reales y con ellos se completa el listado de los 5 puntos críticos.

Para concluir, basta con evaluar la función $f(x, y)$ sobre la lista de puntos:

```
(%i)float(f(2/3,2/3));
(%o)−,1481481481481481
(%i)f(1.972939944134078,0.7689417658109);
(%o)5,236772584960951
(%i)f(0.25651518414818,1.668752793920429);
(%o) 1,945493736056898
(%i)f(1.193428722840488,1.981114258734655);
(%o),8959436399289267
(%i)f(0.3385499207607,0.25001080613789);
(%o)−,06797361977380298
```

De donde deducimos que el mínimo absoluto esta en el punto $(2/3, 2/3)$ y el máximo absoluto en el punto: $(1{,}972939944134078, 0{,}7689417658109)$.

8.6. Ejercicios

1. Calcular los extremos relativos de las siguientes funciones:
   a) $x^3 + 3x^2y - 2y^2(1 + y)^2$
   b) $x^3 + y^3 + 2x^2 + 4y^2$
   c) $x^4 + x^2y + x^2 + y^2$
   d) $(x^2 + 3y^2)e^{1-x^2-y^2}$
   e) $x^2 + (1 - x^3)y^2$
   f) $x^3e^y - xy^2$
   g) $x^3 + 3xy^2 - 15x - 12y$
   h) $x^3 + xz + y^2 + xyz$
   i) $x^3 + yz + y^2 + xy + 3z$
   j) $x^3 + 3xz + y^2 + yz$
   k) $\log(y^2 + 1) - \cos(xz) - \operatorname{sen}(y^2z)$

2. Justificar que la función $f(x, y) = x^2 + y(y^3 - 4)$ tiene un extremo global en $\mathbb{R}^2$, indicando de qué tipo es.

3. Justificar que la función $f(x, y) = xy - x^2 - y^2$ tiene un extremo global en $\mathbb{R}^2$, indicando de qué tipo es.

4. Justificar que la función $f(x, y) = x - 2y - x^2 - y^2$ tiene un extremo global en $\mathbb{R}^2$, indicando de qué tipo es.

5. Justificar que la función $f(x, y) = 3x - 2y - x^3 - y^2$ tiene un extremo global en en $C = \{(x, y) : x > 0\}$, indicando de qué tipo es.

6. Justificar que la función $v(x, y, z) = \frac{x^3}{6} - x + 2y^2 + 2yz - 7y + 6z^2$ tiene un extremo global en $C = \{(x, y, z) : x > 0\}$, indicando de qué tipo es.

7. Calcular lo extremos relativos de las siguientes funciones sujetas a las correspondientes restricciones, reduciendo los problemas al cálculo de extremos ordinarios:

a) xy sujeta a la restricción $y + x^3 - x^2 = 1$.

b) $x^2 + y^2$ sujeta a la restricción $y + x^2 = 1$.

c) $\frac{1}{x} + \frac{1}{y}$ sujeta a la restricción $4x + x = 1$.

d) $x^2 + y^2 + z^2$ sujeta a la restricción $2x + 2y - z + 3 = 0$.

e) $xy + yz + xz$ sujeta a la restricción $x + y + z - 6 = 0$.

f) xyz sujeta a la restricción $x + y + z - 1 = 0$.

8. Un fabricante puede producir tres productos distintos en cantidades Q_1, Q_2, Q_3 respectivamente, y genera un benecio igual a

$$B(Q_1, Q_2, Q_3) = 2Q_1 + 8Q_2 + 24Q_3.$$

Halla los valores de Q_1, Q_2, y Q_3 que maximizan el benecio si la producción está sujeta a la restricción

$$Q_1^2 + Q_2^2 + Q_3^2 = 4{,}5 \times 10^9.$$

9. Determinar los extremos relativos de las siguientes funciones usando la técnica de los multiplicadores de Lagrange.

a) $x^2 + y^2 - xy$ sujeta a la restricción $x^2 + y^2 - 1$.

b) $x^2 + y^2 - x$ sujeta a la restricción $x^2 + y^2 - 1$.

c) $x^2 + y^2 - 2xy$ sujeta a la restricción $2x^2 + y^2 - 2xy - 1$.

d) $x^2 + y^2$ sujeta a la restricción $5(x^2 + y^2) + 2x - 6y$.

e) $6x - y^2 + xz + 60$ sujeta a la restricción $x^2 + y^2 + z^2 - 6$.

10. Calcular los extremos absolutos de los siguientes problemas.

a) $x^3 + y^3 + x^2 + 2y^2$ sujeta a la restricción $x^2 + y^2 \leq 1$.

b) $x^2 + y^2 - x$ sujeta a la restricción $x^2 + y^2 \leq 1$.

c) $x^2 + y^2 - xy$ sujeta a la restricción $2x^2 + y^2 - xy \leq 1$.

d) $x^3 + y^3 + x^2 + 2y^2$ sujeta a la restricción $(x-1)^2 + y^2 \leq 1$.

e) $x^3 + y^3 + x^2 + 2y^2$ sujeta a la restricción $x^2 + (y+1)^2 \leq 1$.

f) $x^3 + y^2 - 2xy$ sujeta a la restricción $(x-1)^2 + (y-1)^2 \leq 1$.

g) $x^2 - xy$ sujeta a la restricción $x^2 + y^2 - xy \leq 1$.

Capítulo 9

Bibliografía

- Manual de *Maxima*.
  - En formato html: http://maxima.sourceforge.net/docs/manual/es/maxima.html
  - En formato pdf: http://maxima.sourceforge.net/es/documentation.html
- J. Alaminos, C. Aparicio, J. Extremera, P. Munoz, A. Villena. Prácticas de Ordenador con *Maxima*. http://www.ugr.es/~ alaminos/page5/index.html
- Practicas de Cálculo con WxMaxima. Escuela Politécnica de Ingeniería. Gijón. Universidad de Oviedo.