PRÁCTICA 2: Funciones de una variable. Gráficas, límites y continuidad. Ecuaciones

Funciones

Para definir una función en *Maxima* se utiliza el operador `:=`. Se pueden definir funciones de una o varias variables, con valores escalares o vectoriales,

`funcion(var1,var2,..):=(expr1,expr2,...)`	definición de función
`define (func,expr)`	la función vale *expr*
`fundef(func)`	devuelve la definición de la función
`functions`	lista de funciones definidas por el usuario
`remfunction(func1,func2,...)`	borra las funciones

```
(%i1)  f(x):=sin(x);
(%o1)  f(x):=sin(x)
```

```
(%i2)  f(%pi/4);
(%o2)  1/√2
```

También se puede utilizar el comando `define` para definir una función. Por ejemplo, podemos utilizar la función g para definir una nueva función y, de hecho veremos que ésta es la manera correcta de hacerlo cuando la definición involucra funciones previamente definidas, derivadas de funciones, etc. El motivo es que la orden `define` evalúa los comandos que pongamos en la definición.

```
(%i1) define(f(x),sin(x));
(%o1) f(x):=sin(x)
```

Funciones definidas a trozos

Las funciones definidas a trozos plantean algunos problemas de difícil solución para *Maxima*. Esencialmente hay dos formas de definir y trabajar con funciones a trozos:

a) definir una función para cada trozo con lo que tendremos que ocuparnos nosotros de ir escogiendo de elegir la función adecuada, o
b) utilizar una estructura `if-then-else` para definirla.[4]

Cada uno de los métodos tiene sus ventajas e inconvenientes. El primero de ellos nos hace aumentar el número de funciones que definimos, usamos y tenemos que nombrar y recordar. Además de esto, cualquier cosa que queramos hacer, ya sea representar gráficamente o calcular una integral tenemos que plantearlo nosotros. *Maxima* no se encarga de esto. La principal limitación del segundo método es que las funciones definidas de esta manera no nos sirven para derivarlas o integrarlas, aunque sí podremos dibujar su gráfica. Por ejemplo, la función

$$f(x) = \begin{cases} x^2, & \text{si } x < 0 \\ x^3, & \text{en otro caso} \end{cases}$$

la podemos definir de la siguiente forma utilizando el segundo método

```
(%i13)   f(x):=if x< 0 then x^2 else x^3;
(%o13)   f(x):=if x< 0 then x^2 else x^3
```

Gráficos en el plano con plot2d

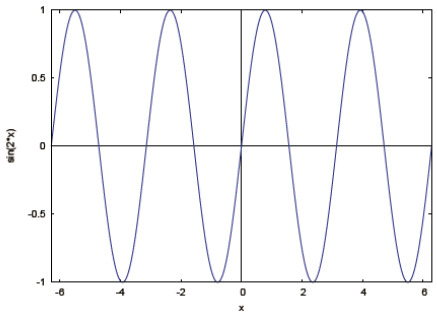
Coordenadas cartesianas

El comando que se utiliza para representar la gráfica de una función de una variable real es `plot2d` que actúa, como mínimo, con dos parámetros: la función (o lista de funciones a representar), y el intervalo de valores para la variable x. Al comando `plot2d` se puede acceder también a través del menú **Gráficos→Gráficos 2D** o, directamente, a través del botón Gráficos 2D. **plot2d**

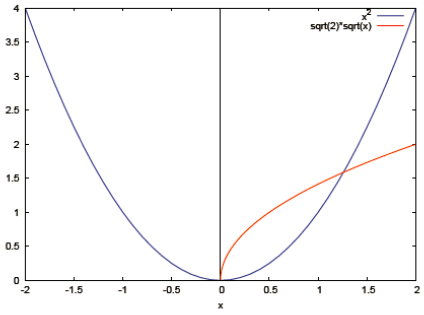
`plot2d(f(x),[x,a,b])`	gráfica de $f(x)$ en $[a,b]$
`plot2d([f1(x),f2(x),...],[x,a,b])`	gráfica de una lista de funciones en $[a,b]$

Por ejemplo:

```
(%i23)   plot2d(sin(2*x),[x,-2*%pi,2*%pi]);
(%o23)
```




```
(%i24)   plot2d([x^2,sqrt(2*x)],[x,-2,2]);
(%o24)
```



Observa en esta última salida cómo el programa asigna a cada gráfica un color distinto para diferenciarlas mejor y añade la correspondiente explicación de qué color representa a cada función.

Cuando pulsamos el botón Gráficos 2D, aparece una ventana de diálogo con varios campos que podemos completar o modificar:

a) Expresión(es). La función o funciones que queramos dibujar. Por defecto, *wxMaxima* rellena este espacio con % para referirse a la salida anterior.

b) Variable x. Aquí establecemos el intervalo de la variable x donde queramos representar la función.

Figura 2.2 Gráficos en 2D

c) Variable y. Ídem para acotar el recorrido de los valores de la imagen.

d) Graduaciones. Nos permite regular el número de puntos en los que el programa evalúa una función para su representación en polares. Veremos ejemplos en la sección siguiente.

e) Formato. *Maxima* realiza por defecto la gráfica con un programa auxiliar. Si seleccionamos `en línea`, dicho programa auxiliar es *wxMaxima* y obtendremos la gráfica en una ventana alineada con la salida correspondiente. Hay dos opciones más y ambas abren una ventana externa para dibujar la gráfica requerida: `gnuplot` es la opción por defecto que utiliza el programa *Gnuplot* para realizar la representación; también está disponible la opción `openmath` que utiliza el programa *XMaxima*. Prueba las diferentes opciones y decide cuál te gusta más.

f) Opciones. Aquí podemos seleccionar algunas opciones para que, por ejemplo, dibuje los ejes de coordenadas (`"set zeroaxis;"`); dibuje los ejes de coordenadas, de forma que cada unidad en el eje Y sea igual que el eje X (`"set size ratio 1; set zeroaxis;"`); dibuje una cuadrícula (`"set grid;"`) o dibuje una gráfica en coordenadas polares (`"set polar; set zeroaxis;"`).

g) Gráfico al archivo. Guarda el gráfico en un archivo con formato Postscript.

Evidentemente, estas no son todas las posibles opciones. La cantidad de posibilidades que tiene *Gnuplot* es inmensa.

⚠ **Observación** El prefijo "wx" añadido a `plot2d` o a cualquiera del resto de las órdenes que veremos en este capítulo (`plot3d, draw2d, draw3d`) hace que *wxMaxima* pase automáticamente a mostrar los gráficos en la misma ventana y no en una ventana separada. Es lo mismo que seleccionar `en línea`.

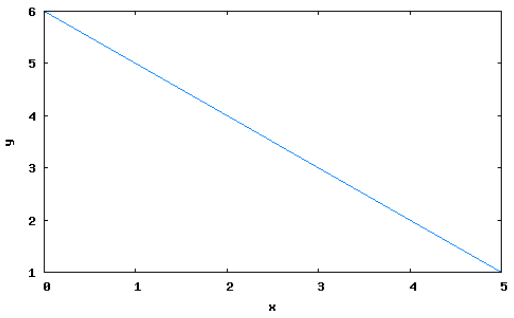
Gráficas de curvas poligonales o listas

Podemos dibujar líneas poligonales o listas de puntos haciendo uso de una opción dentro de **Especial** (concretamente, **Gráfico discreto**) en la ventana de diálogo de **Gráficos 2D**. Cuando elegimos esta opción, aparece una nueva ventana en la que tendremos que escribir las coordenadas, separadas por comas, de los puntos que van a ser los vértices de la curva poligonal que queremos dibujar.

`plot2d([discrete,[x1,x2,...],[y1,y2,...]],opc)`	poligonal que une los puntos $(x1, y1)$, $(x2, y2)$,...
`plot2d([discrete,[[x1,y1],[x2,y2],...]],opc)`	poligonal que une los puntos $(x1, y1)$, $(x2, y2)$,...

Por ejemplo, para dibujar la recta que une los puntos (0, 6) y (5,1), escribimos:

```
(%i1) wxplot2d([discrete,[[0,6],[5,1]]]);
```



Gráficos con draw

El módulo "draw" es relativamente reciente en la historia de *Maxima* y permite dibujar gráficos en 2 y 3 dimensiones con relativa comodidad. Se trata de un módulo adicional que hay que cargar previamente para poder usarlo. Comencemos por esto.

```
(%i60)    load(draw)$
```

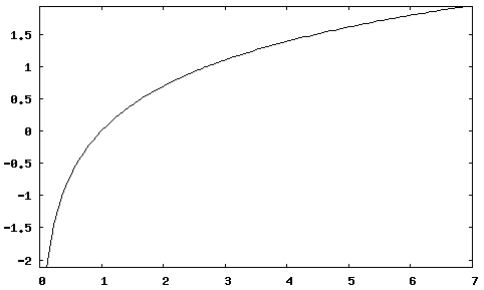
Una vez cargado el módulo draw, podemos utilizar la orden draw2d para dibujar gráficos en 2 dimensiones. Un gráfico está compuesto por varias opciones y el objeto gráfico que queremos dibujar.

`draw2d(opciones, objeto gráfico,...)` dibuja gráfico dos dimensional

Las opciones son numerosas y permiten controlar prácticamente cualquier aspecto imaginable. Aquí comentaremos algunas de ellas pero la ayuda del programa es insustituible. En segundo lugar aparece el objeto gráfico.
Estos pueden ser de varios tipos aunque los que más usaremos son quizás `explicit` y `implicit`.

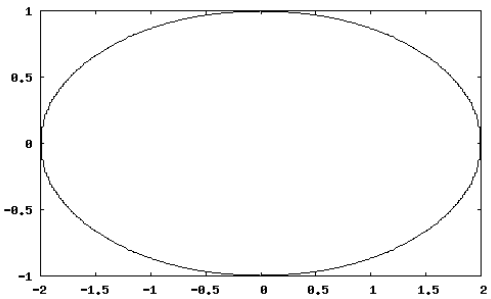
explicit: Usaremos explicit($f(x),x,a,b$) para dibujar $f(x)$ en $[a, b]$.

```
(%i13) draw2d(explicit(log(x),x,0,7) );
```



implicit: nos permite dibujar el lugar de los puntos que verifican una ecuación en el plano. Además de la ecuación debemos indicar los intervalos dónde pueden tomar valores las variables.

```
(%i9) wxdraw2d(implicit(x^2/4+y^2-1,x,-2,2,y,-1,1));
```



Observación El comando plot2d no permite dibujar curvas en implícitas y debemos usar en este caso el comando draw2d.

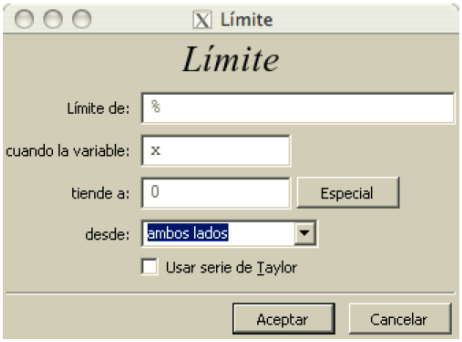
Límites y Continuidad

Uno de los primeros conceptos que se presentan en un curso de Cálculo es el de continuidad.
Este concepto está íntimamente ligado al concepto de límite.
En este capítulo veremos cómo usar *Maxima* para resolver algunos problemas relacionados con todos esto.

Límites

El cálculo de límites se realiza con la orden `limit`. Con ella podemos calcular límites de funciones o de sucesiones en un número, en $+\infty$ o en $-\infty$. También podemos usar el menú **Análisis→Calcular límite**. Ahí podemos escoger, además de a qué función le estamos calculando el límite, a qué tiende la variable incluyendo los valores "especiales" como π, e o infinito. Además de esto, también podemos marcar si queremos calcular únicamente el límite por la derecha o por la izquierda.

limit



`limit (expr,x,a)`	$\lim_{x\to a} expr$
`limit (expr,x,a,plus)`	$\lim_{x\to a^+} expr$
`limit (expr,x,a,minus)`	$\lim_{x\to a^-} expr$
`inf`	$+\infty$
`minf`	$-\infty$
`und`	indefinido
`ind`	indefinido pero acotado

El cálculo de límites con *Maxima*, como puedes ver, es sencillo. Sabe calcular límites de cocientes de polinomios en infinito

```
(%i1)   limit(n/(n+1),n,inf);
(%o1)   1
```

o en $-\infty$,

```
(%i2)   limit((x^2+3*x+1)/(2*x+3),x,minf);
(%o2)   -∞
```

aplicar las reglas de L'Hôpital,

```
(%i3)   limit(sin(x)/x,x,0);
(%o3)   1
```

Incluso es capaz de dar alguna información en el caso de que no exista el límite. Por ejemplo, sabemos que las funciones periódicas, salvo las constantes, no tienen límite en ∞. La respuesta de *Maxima* cuando calculamos el límite de la función coseno en ∞ es

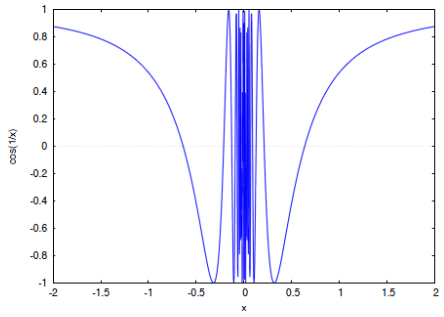
```
(%i4)   limit(cos(x),x,inf);
(%o4)   ind
```

Indeterminado. Este límite es equivalente a

```
(%i5)   limit(cos(1/x),x,0);
(%o5)   ind
```

La función $\cos\left(\frac{1}{x}\right)$ oscila cada vez más rápidamente cuando nos acercamos al origen. Observa su gráfica.

```
(%i6)   plot2d([cos(1/x)], [x,-2,2]);
(%o6)
```




```
(%i8)   limit(abs(x)/x,x,0);
(%o8)   und
```

En este último límite lo que ocurre es que tenemos que estudiar los límites laterales

```
(%i9)   limit(abs(x)/x,x,0,plus);
(%o9)   1
(%i10)   limit(abs(x)/x,x,0,minus);
(%o10)   -1
```

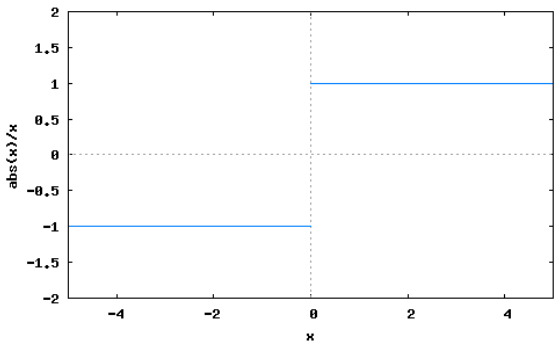
Por tanto, no existe el límite puesto que los límites laterales no coinciden. Si recuerdas la definición de función derivable, acabamos de comprobar que la función valor absoluto no es derivable en el origen.

Continuidad

El estudio de la continuidad de una función es inmediato una vez que sabemos calcular límites. Una función $f : A \subset \mathbb{R} \to \mathbb{R}$ es continua en $a \in A$ si $\lim_{x\to a} f(x) = f(a)$. Conocido el valor de la función en el punto, la única dificultad es, por tanto, saber si coincide o no con el valor del límite.

Con respecto a las funciones discontinuas, la gráfica puede darnos una idea del tipo de discontinuidad. Si la discontinuidad es evitable, es difícil apreciar un único pixel en la gráfica. Una discontinuidad de salto es fácilmente apreciable. Por ejemplo, la función signo, esto es, $\frac{|x|}{x}$, tiene un salto en el origen que *Maxima* une con una línea vertical.

```
(%i12) wxplot2d([abs(x)/x], [x,-5,5], [y,-2,2])$
```



Resolución de ecuaciones

Maxima puede resolver los tipos más comunes de ecuaciones y sistemas de ecuaciones algebraicas de forma exacta. Por ejemplo, sabe encontrar las raíces de polinomios de grado bajo (2,3 y 4). En cuanto a polinomios de grado más alto o ecuaciones más complicadas, no siempre será posible encontrar la solución exacta. En este caso, podemos intentar encontrar una solución aproximada en lugar de la solución exacta.

La orden solve

La orden `solve` nos da todas las soluciones, ya sean reales o complejas de una ecuación o sistema de ecuaciones. **solve**

```
(%i26)  solve(x^2-3*x+1=0,x);
(%o26)  [x=-(sqrt(5)-3)/2,x=(sqrt(5)+3)/2]
```

`solve(`*ecuación*`,`*incógnita*`)`	resuelve la ecuación
`solve([`*ecuaciones*`],[`*variables*`])`	resuelve el sistema
`multiplicities`	guarda la multiplicidad de las soluciones

Cuando buscamos las raíces de un polinomio hay veces que es conveniente tener en cuenta la multiplicidad de las raíces. Ésta se guarda automáticamente en la variable `multiplicities`. Por ejemplo, el polinomio $x^7 - 2x^6 + 2x^5 - 2x^4 + x^3$ tiene las raíces 0, 1, i, $-i$, **multiplicities**

```
(%i32)  solve(x^7-2*x^6+2*x^5-2*x^4+x^3);
(%o32)  [x=-%i,x=%i,x=1,x=0]
```

pero estas raíces no pueden ser simples: es un polinomio de grado 7 y sólo tenemos 4 raíces. ¿Cuál es su multiplicidad?

```
(%i33)  multiplicities;
(%o33)  [1,1,2,3]
```

O sea que i y $-i$ son raíces simples, 1 tiene multiplicidad 2 y 0 es una raíz triple.

La orden `solve` no sólo puede resolver ecuaciones algebraicas.

```
(%i36)  solve(sin(x)*cos(x)=0,x);
        'solve' is using arc-trig functions to get a solution.
        Some solutions will be lost.
(%o36)  [x=0,x=%pi/2]
```

¿Qué ocurre aquí? La expresión $\operatorname{sen}(x)\cos(x)$ vale cero cuando el seno o el coseno se anulen. Para calcular la solución de $\operatorname{sen}(x) = 0$ aplicamos la función arcoseno a ambos lados de la ecuación. La función arcoseno vale cero en cero pero la función seno se anula en muchos más puntos. Nos estamos dejando todas esas soluciones y eso es lo que nos está avisando *Maxima*.

También podemos resolver sistemas de ecuaciones. Sólo tenemos que escribir la lista de ecuaciones y de incógnitas. Por ejemplo, para resolver el sistema

$$\left.\begin{aligned} x^2+y^2 &= 1 \\ (x-2)^2+(y-1)^2 &= 4 \end{aligned}\right\}$$

escribimos

```
(%i44)  solve([x^2+y^2=1,(x-2)^2+(y-1)^2=4],[x,y]);
(%o44)  [[x=4/5,y=-3/5],[x=0,y=1]]
```

Sistemas de ecuaciones lineales

`linsolve([`*ecuaciones*`],[`*variables*`])`	resuelve el sistema

linsolve

En el caso particular de sistemas de ecuaciones lineales puede ser conveniente utilizar `linsolve` en lugar de `solve`. Ambas órdenes se utilizan de la misma forma, pero `linsolve` es más eficiente en estos casos.

```
(%i50)  eq:[x+y+z+w=1,x-y+z-w=-2,x+y-w=0]$
(%i51)  linsolve(eq,[x,y,z]);
(%o51)  [x=(4*w-3)/2,y=-(2*w-3)/2,z=1-2*w]
```

Algsys

`algsys([`*ecuaciones*`],[`*variables*`])`	resuelve la ecuación o ecuaciones
`realonly`	si vale `true`, `algsys` muestra sólo soluciones reales

algsys

La orden `algsys` resuelve ecuaciones o sistemas de ecuaciones algebraicas. La primera diferencia con la orden `solve` es pequeña: `algsys` siempre tiene como entrada listas, en otras palabras, tenemos que agrupar la ecuación o ecuaciones entre corchetes igual que las incógnitas.

La segunda diferencia es que `algsys` intenta resolver numéricamente la ecuación si no es capaz de encontrar la solución exacta.

```
(%i54)  solve(eq:x^6+x+1);
(%o54)  [0=x^6+x+1]
(%i55)  algsys([eq],[x]);
```

```
(%o55)  [[x=-1.038380754458461%i-0.15473514449684],
         [x=1.038380754458461%i-0.15473514449684],
         [x=-0.30050692030955%i-0.79066718881442],
         [x=0.30050692030955%i-0.79066718881442],
         [x=0.94540233331126-0.61183669378101%i],
         [x=0.61183669378101%i+0.94540233331126]]
```

Soluciones aproximadas

Las ecuaciones polinómicas se pueden resolver de manera aproximada. Los comandos `allroots` y `realroots` están especializados en encontrar soluciones racionales aproximadas de polinomios en una variable.

`allroots(`*polinomio*`)`	soluciones aproximadas del polinomio
`realroots(`*polinomio*`)`	soluciones aproximadas reales del polinomio

Estás órdenes nos dan todas las soluciones y las soluciones reales de un polinomio en una variable y pueden ser útiles en polinomios de grado alto.

```
(%i61)  eq:x^9+x^7-x^4+x$
(%i62)  allroots(eq);
(%o62)  [x=0.0,x=0.30190507748312%i+0.8440677798278,
        x=0.8440677798278-0.30190507748312%i,
        x=0.8923132916888%i-0.32846441923834,
        x=-0.8923132916888%i-0.32846441923834,
        x=0.51104079208431%i-0.80986929589487,
        x=-0.51104079208431%i-0.80986929589487,
        x=1.189238256723466%i+0.29426593530541,
        x=0.29426593530541-1.189238256723466%i]
```

```
(%i63)  realroots(eq);
(%o63)  [x=0]
```

El teorema de los ceros de Bolzano

Uno de los primeros resultados que aprendemos sobre funciones continuas es que si cambian de signo tienen que valer cero en algún momento.

El comando find_root encuentra una solución de una función (ecuación) continua que cambia de signo.

`find_root (f(x),x,a,b)`	solución de f en $[a,b]$

```
(%i64)  f(x):=exp(x)+log(x);
(%o64)  f(x):=exp(x)+log(x)
```

Buscamos un par de puntos donde cambie de signo

```
(%i65)  f(1)
(%o65)  %e
(%i66)  f(exp(-3)),numer;
(%o66)  -1.948952728663784
```

Vale, ya que tenemos dos puntos donde cambia de signo podemos utilizar `find_root`:

```
(%i69)  find_root(f(x),x,exp(-3),1);
(%o69)  0.26987413757345
```

⚠ **Observación** Este método encuentra *una* solución pero no nos dice cuántas soluciones hay.

EJERCICIOS PROPUESTOS

1) Definir la función $e + x + x^{10}$ y calcular f$(\frac{1}{2})$

2) Construir las funciones compuestas $f \circ g$ en los casos siguientes:
a) $f(x) = sen(x)$; $g(x) = 1 - x^2$

b) $f(u) = \frac{u-1}{u+1}$; $g(u) = \frac{u+1}{1-u}$

c) $f(x) = \frac{1}{x}$; $g(x) = tg(x)$

3) Calcular los límites siguientes:

a) $\lim_{x\to\infty} \frac{3x^4 sen^2(\frac{1}{x}) \ln(1+\frac{1}{x})}{(x+6)^{\frac{\pi x+2}{4x+1}}}$

b) $\lim_{x\to 0} \frac{\cos x + e^{-\frac{x^2}{2}} - 2}{x^2}$

c) $\lim_{x\to 0} (\cos x)^{\cot^2 x}$

d) $\lim_{x\to 0} \frac{\sqrt[n]{1+x}-1}{x}$ $x \in \mathbb{N}$

4) Estudiar la continuidad de la función $f(x) = xLn|x|$ si $x \neq 0$ y $f(0) = 0$

5) Representar en una misma gráfica las funciones seno y coseno en el intervalo $[-2\pi, 2\pi]$, de tal forma que una sea en color rojo, la otra en azul y con grosores distintos.

6) Definir y representar gráficamente la función:

$$f(x) = \begin{cases} e^{x+1} & si \quad 0 \leq x < 1 \\ \ln(1+x^2) & si \quad x \geq 1 \end{cases}$$

7) Definir y representar gráficamente la siguiente función:

$$f(x) = \begin{cases} 1 & si \quad x < 0 \\ 1 - x^2 & si \quad 0 \leq x \leq 2 \\ -3 & si \quad x > 2 \end{cases}$$

8) Hallar las raices de las siguientes ecuaciones y el orden de multiplicidad:
a) $x^3 + 3x^2 + 3x + 1 = 0$

b) $x^4 + x^2 + 3x + 1 = 0$

9) Resolver el sistema de ecuaciones: $\begin{cases} x^2 - y^2 = 1 \\ x + y + z = 0 \\ y^2 - z^2 = 0 \end{cases}$

10) Hallar las raices de la ecuación: $e^x + 1 = tg(x)$