Gráficos

2

El objetivo de este capítulo es aprender a representar gráficos en dos dimensiones. Lo haremos, tanto para gráficas en coordenadas cartesianas como para gráficas en coordenadas paramétricas y polares. *wxMaxima* permite hacer esto fácilmente aunque también veremos cómo utilizar el módulo `draw` que nos da algunas posibilidades más sin complicar excesivamente la escritura.

Aunque sólo vamos a hablar de gráficos en dos dimensiones, hay que decir que se pueden realizar representaciones en tres dimensiones de manera análoga. En la ayuda de *Maxima* puedes encontrar todos los detalles.

2.1 Funciones

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

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

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

que se pueden utilizar como cualquier otra función.

```
(%i2)  f(%pi/4);
(%o2)  1/√2
```

Si la función tiene valores vectoriales o varias variables tampoco hay problema:

```
(%i3)  g(x,y,z):=[2*x,3*cos(x+y)];
```

```
(%o3)   g(x,y,z):=[2x,3cos(x+y)]
(%i4)   g(1,%pi,0);
(%o4)   [2,-3cos(1)]
```

define 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.

```
(%i5)   define(h(x,y,z),g(x,y,z)^2);
(%o5)   h(x,y,z):=[4x^2,9cos(y+x)^2]
```

Eso sí, aunque hemos definido las funciones f, g y h, para utilizarlas debemos añadirles variables:

```
(%i6)   g;
(%o6)   g
```

Si queremos saber cuál es la definición de la función g, tenemos que preguntar

```
(%i7)   g(x,y);
        Too few arguments supplied to g(x,y,z):
        [x,y]
        -- an error. To debug this try debugmode(true);
```

pero teniendo cuidado de escribir el número correcto de variables

```
(%i8)   g(x,y,z);
(%o8)   [2x,3cos(y+x)]
```

Esto plantea varias cuestiones muy relacionadas entre sí: cuando llevamos un rato trabajando y hemos definido varias funciones, ¿cómo sabemos cuales eran? y ¿cuál era su definición?. La lista de funciones que hemos definido se guarda en la variable `functions` a la que también puedes acceder desde el menú **Maxima→Mostrar funciones** de manera similar a como accedemos a la lista de variables. En el mismo menú, **Maxima→Borrar**

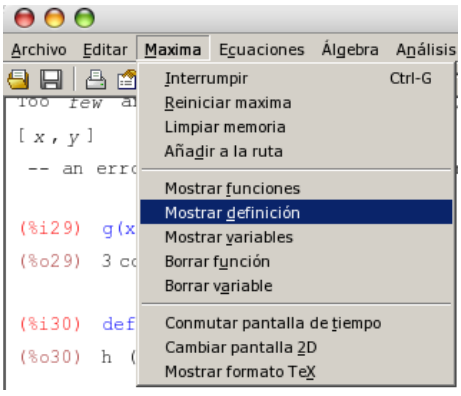


Figura 2.1 Desde el menú podemos consultar las funciones que tenemos definidas, cuál es su definición y borrar algunas o todas ellas

función tenemos la solución a cómo borrar una función (o todas). También podemos hacer esto con la orden `remfunction`. **remfunction**

```
(%i9)  functions;
(%o9)  [f(x),g(x,y,z),h(x,y,z)]
```

Ya sabemos preguntar cuál es la definición de cada una de ellas. Más cómodo es, quizás, utilizar la orden `fundef` que nos evita escribir las variables **fundef**

```
(%i10)  fundef(f);
(%o10)  f(x):=sin(x)
```

que, si nos interesa, podemos borrar

```
(%i11)  remfunction(f);
(%o11)  [f]
```

o, simplemente, borrar todas las que tengamos definidas

```
(%i12)  remfunction(all);
(%o12)  [g,h]
```

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

[4] En la sección 5.3 explicamos con más detalle este tipo de estructuras

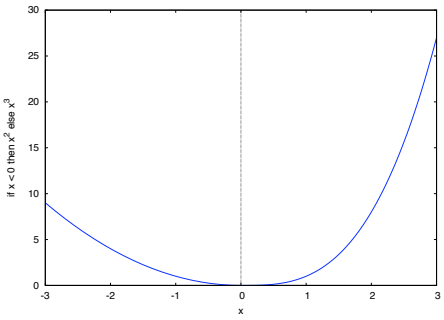
```
(%i13)  f(x):=if x< 0 then x^2 else x^3;
(%o13)  f(x):=if x< 0 then x^2 else x^3
```

y podemos evaluarla en un punto

```
(%i14)  f(-2);
(%o14)  4
(%i15)  f(2);
(%o15)  8
```

o dibujarla

```
(%i16)  plot2d(f(x),[x,-3,3]);
(%o16)
```



pero no podemos calcular $\int_{-3}^{3} f(x)\,dx$:

```
(%i17)  integrate(f(x),x,-3,3);
(%o17)  ∫_-3^3 if x<0 then x^2 else x^3 dx
```

La otra posibilidad es mucho más de andar por casa, pero muy práctica. Podemos definir las funciones

```
(%i18)  f1(x):=x^2$
(%i19)  f2(x):=x^3$
```

y decidir nosotros cuál es la que tenemos que utilizar:

```
(%i20)  integrate(f1(x),x,-3,0)+integrate(f2(x),x,0,3);
(%o20)  117/4
```

Evidentemente, si la función tiene "muchos" trozos, la definición se alarga; no cabe otra posibilidad. En este caso tenemos que anidar varias estructuras if-then-else o definir tantas funciones como trozos. Por ejemplo, la función

$$g(x) = \begin{cases} x^2, & \text{si } x \leq 1, \\ \text{sen}(x), & \text{si } 1 \leq x \leq \pi, \\ -x+1, & \text{si } x > \pi \end{cases}$$

la podemos escribir como sigue anidando dos condicionales

```
(%i21)  g(x):=if x<=1 then x^2 else
          if x <= %pi then sin(x) else -x+1$
```

Comprobamos que la definición se comporta correctamente en un valor de cada intervalo

```
(%i22)  [g(-3),g(2),g(5)];
(%o22)  [9,sin(2),-4]
```

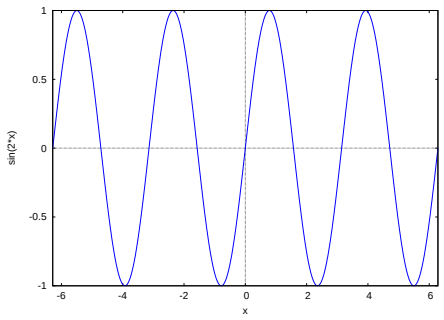
2.2 Gráficos en el plano con plot2d

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**. **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 $[a,b]$

Podemos dibujar la gráfica de una función

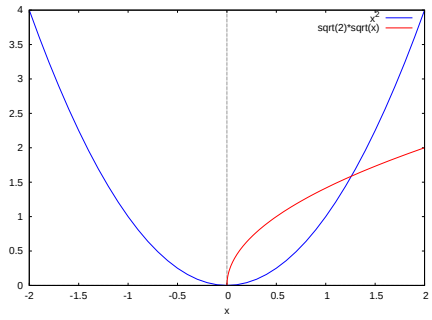
```
(%i23)  plot2d(sin(2*x),[x,-2*%pi,2*%pi]);
(%o23)
```



o de varias

```
(%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 accedemos a través del menú, 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.

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.

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.

Figura 2.2 Gráficos en 2D

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;"`). Esta última opción la comentamos más adelante.

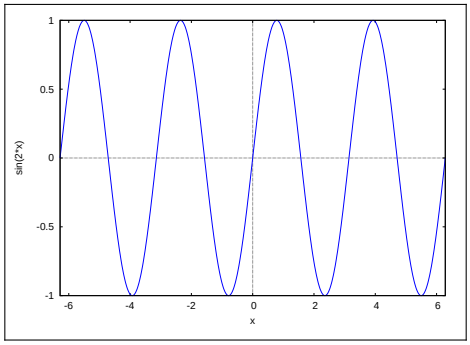
g) Gráfico al archivo. Guarda el gráfico en un archivo con formato Postscript.

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

⚠ **Observación 2.1.** El prefijo "wx" añadido a `plot2d` o a cualquiera del resto de las órdenes que veremos en este capítulo 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`. Por ejemplo,

```
(%i25)  wxplot2d(sin(2*x),[x,-2*%pi,2*%pi]);
```

(%t25)

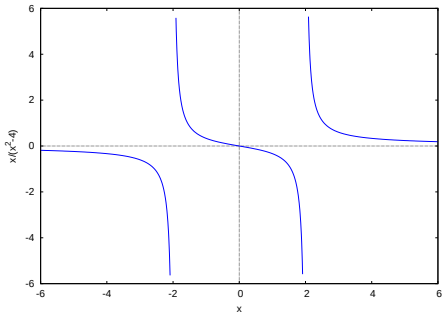


Es complicado representar una ventana separada en unas notas escritas así que, aunque no utilicemos `wxplot2d`, sí hemos representado todas las gráficas a continuación de la correspondiente orden. **wxplot2d**

Veamos algunos ejemplos de las opciones que hemos comentado. Podemos añadir ejes,

```
(%i26)  plot2d(x/(x^2-4), [x,-6,6], [y,-6,6],
          [gnuplot_preamble, "set zeroaxis;"])$
```

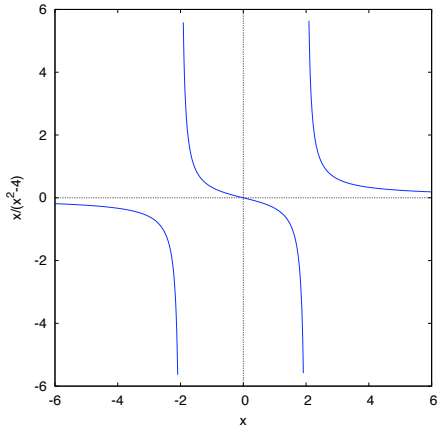
(%o26)



podemos cambiar la proporción entre ejes.

```
(%i27)  plot2d(x/(x^2-4), [x,-6,6], [y,-6,6],
          [gnuplot_preamble, "set size ratio 1; set zeroaxis;"])$
```

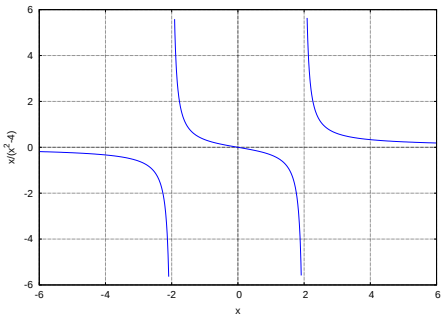
(%o27)



`set size ratio 1` dibuja ambos ejes con el mismo tamaño en pantalla, `set size ratio 2` o `set size ratio 0.5` dibuja el eje X el doble o la mitad de grande que el eje Y. O podemos añadir una malla que nos facilite la lectura de los valores de la función.

```
(%i28)  plot2d(x/(x^2-4),[x,-6,6],[y,-6,6],
          [gnuplot_preamble, "set grid;"])$
```

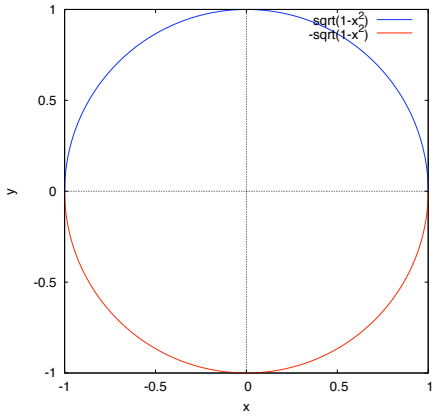
(%o28)



Con el siguiente ejemplo vamos a ver la utilidad de la opción `"set size ratio 1; set zeroaxis;"`. En primer lugar dibujamos las funciones $\sqrt{1-x^2}$ y $-\sqrt{1-x^2}$, con $x \in [-1,1]$. El resultado debería ser la circunferencia unidad. Sin embargo, aparentemente es una elipse. Lo arreglamos de la siguiente forma:

```
(%i29)  plot2d([sqrt(1-x^2,-sqrt(1-x^2)],[x,-1,1],[y,-1,1],
          [gnuplot_preamble, "set size ratio 1; set zeroaxis;"])$
```

(%o29)



También podemos dibujar gráficas de funciones a trozos. Antes, tenemos que recordar cómo se definen estas funciones. Lo hacemos con un ejemplo. Consideremos la función $f : \mathbb{R} \to \mathbb{R}$ definida como

$$f(x) = \begin{cases} \sqrt{-x} & \text{si } x < 0 \\ x^3 & \text{si } x \geq 0. \end{cases}$$

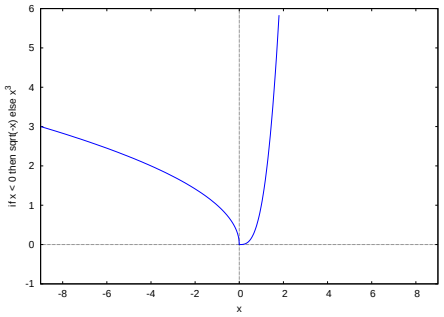
Vamos, en primer lugar, a definirla:

```
(%i30)  f(x):= if x<0 then sqrt(-x) else x^3;
```

```
(%o30)   f(x):= if x<0 then sqrt(-x) else x^3
```

y luego la representamos

```
(%i31)   plot2d(f(x),[x,-9,9],[y,-1,6],
           [gnuplot_preamble,"set zeroaxis;"])$
(%o31)
```



2.3 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. Este se hace de la siguiente forma

```
(%i32)   load(draw)$
```

`gr2d(`*opciones, objeto gráfico,...*`)`	gráfico dos dimensional
`draw(`*opciones, objeto gráfico,...*`)`	dibuja un gráfico
`draw2d(`*opciones, objeto gráfico,...*`)`	dibuja gráfico dos dimensional

El paquete `draw`, permite utilizar, entre otras, la orden `draw2d` para dibujar gráficos en dos dimensiones. Un gráfico está compuesto por varias opciones y el objeto gráfico que queremos dibujar. Por ejemplo, en dos dimensiones tendríamos algo así:

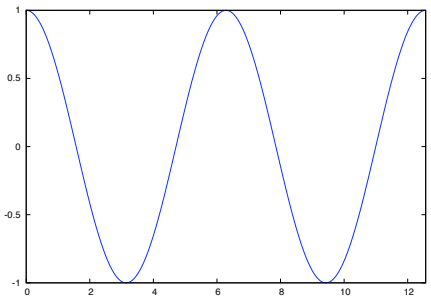
```
objeto:gr2d(
    color=blue,
    nticks=60,
    explicit(cos(t),t,0,2$*$\%pi)
    )
```

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. En este caso "`explicit(cos(t),t,0,2*%pi)`". Estos pueden ser de varios tipos aunque los que más usaremos son quizás `explicit` e `implicit`. Para dibujar un gráfico tenemos dos posibilidades

a) Si tenemos previamente definido el objeto, `draw(objeto)`, o bien, **draw**

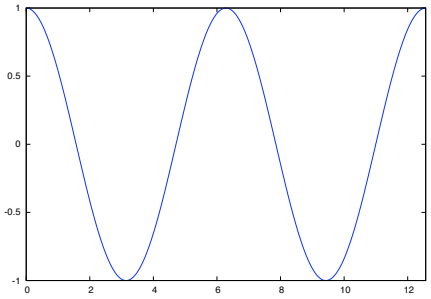
draw2d b) `draw2d(definición del objeto)` si lo definimos en ese momento para dibujarlo. Por ejemplo,

```
(%i33)  coseno:gr2d(
          color=blue,
          explicit(cos(x),x,0,4*%pi))$
(%i34)  draw(coseno);
(%o34)
```



da el mismo resultado que

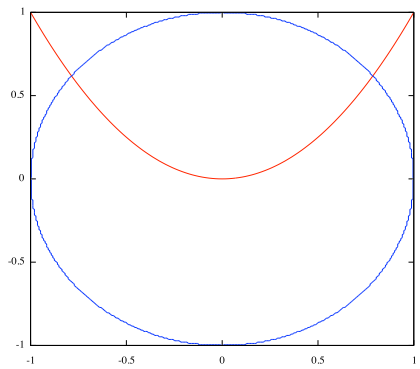
```
(%i35)  draw2d(
          color=blue,
          explicit(cos(x),x,0,4*%pi)$
(%o35)
```



También podemos representar más de un objeto en un mismo gráfico. Simplemente escribimos uno tras otro separados por comas. En el siguiente ejemplo estamos mezclando una función dada explícitamente y una curva en coordenadas paramétricas.

```
(%i36)  draw2d(
          color=red,
          explicit(x^2,x,-1,1),
          color=blue,nticks=60,
          implicit(x^2+y^2=1,x,-1,1,y,-1,1));
```

```
(%o36)
```

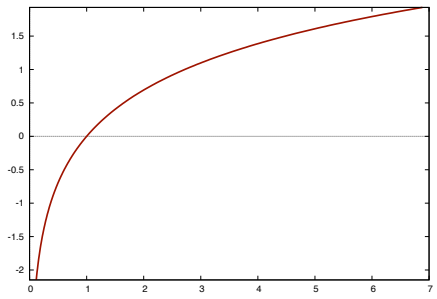


Vamos a comentar brevemente alguno de los objetos y de las opciones del módulo `draw`. Comenzamos con algunos de los objetos que podemos representar y, posteriormente, comentamos algunas opciones.

2.3.1 Objetos

`explicit`: nos permite dibujar una función de una o dos variables. Para funciones de una variable usaremos `explicit(`$f(x)$`,`x`,`a`,`b`)` para dibujar $f(x)$ en $[a,b]$. Con funciones de dos variables escribiremos `explicit(`$f(x,y)$`,`x`,`a`,`b`,`y`,`c`,`d`)`.

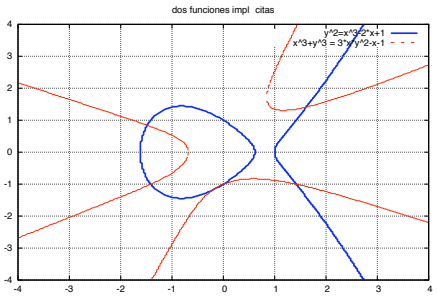
```
(%i37)  draw2d(
          color=dark-red,line_width=5,
          xaxis=true,yaxis=true,
          explicit(log(x),x,0,7));
(%o37)
```



`implicit`: nos permite dibujar el lugar de los puntos que verifican una ecuación en el plano

```
(%i38)  draw2d(
          grid=true,
          line_type=solid,
          color=blue,
          key="y^2=x^3-2x+1",
          implicit(y^2=x^3-2*x+1, x, -4,4, y, -4,4),
          line_type=dots,
          color=red,
          key="x^3+y^3 = 3xy^2-x-1",
          implicit(x^3+y^3 = 3*x*y^2-x-1, x,-4,4, y,-4,4),
          title="dos funciones implícitas");
```

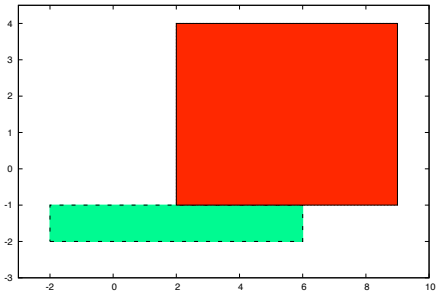
(%o38)



rectangle: para dibujar un rectángulo sólo tenemos que indicar el vértice inferior izquierdo y su opuesto.

```
(%i39)  draw2d(line_width=6,
          line_type=dots,
          transparent=false,
          fill_color=spring-green,
          rectangle([-2,-2],[6,-1]),
          transparent=false,
          fill_color=red,
          line_type=solid,
          line_width=2,
          rectangle([9,4],[2,-1]),
          xrange=[-3,10],
          yrange=[-3,4.5]);
```

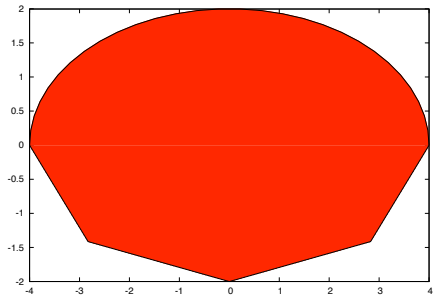
(%o39)



ellipse: la orden `ellipse` permite dibujar elipses indicando 3 pares de números: los dos primeros son las coordenadas del centro, los dos segundos indican la longitud de los semiejes y los últimos son los ángulos inicial y final.
En el dibujo siguiente puedes comprobar cómo la opción `nticks` permite mejorar, aquí empeorar, un gráfico aumentando o, como en este caso, disminuyendo el número de puntos que se utilizan para dibujarlo.

```
(%i40)  draw2d(
          ellipse(0,0,4,2,0,180),
          nticks = 5,
          ellipse(0,0,4,2,180,360));
```

(%o40)



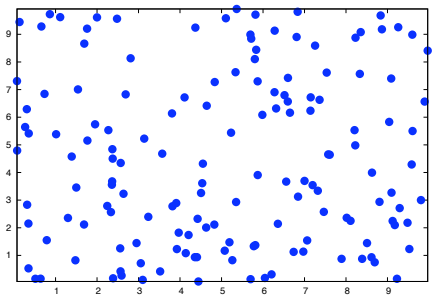
La parte superior de la elipse se ha dibujado utilizando 30 puntos y la inferior únicamente 5.

points: para representar una lista de puntos en el plano o en el espacio tenemos dos posibilidades. Podemos dar los vectores de la forma[5] `[[x1,y1],[x2,y2],...]`, como por ejemplo

```
(%i41)  draw2d(
          color=blue,
          point_type=filled_circle,
          point_size=2,
          points(makelist([random(10.0),random(10.0)],k,1,150)));
```

[5] En el ejemplo usaremos la orden `makelist` que genera una lista de acuerdo a la regla que aparece como primera entrada con tantos elementos como indique el contador que le sigue. En el próximo capítulo lo comentaremos con más detalle.

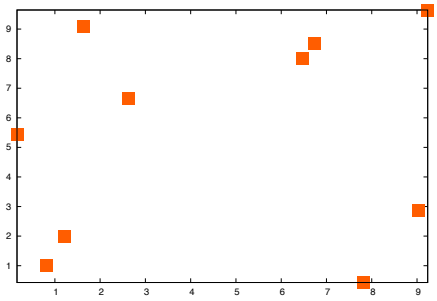
(%o41)



o podemos agrupar por coordenadas de la forma `[[x1,x2,x3,...],[y1,y2,y3,...]]` como aquí.

```
(%i42)  draw2d(
          color=orange-red,
          point_type=5,
          point_size=3,
          points(makelist(random(10.0),k,1,10),
          makelist(random(10.0),k,1,10)));
```

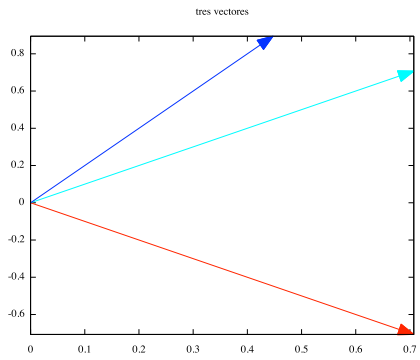
(%o42)



vector: dibuja vectores tanto en dos como en tres dimensiones. Para dar un vector hay que fijar el origen y la dirección.

```
(%i43)  draw2d(
          head_length = 0.03, head_angle=20,
          color=cyan, vector([0,0],[1,1]/sqrt(2)),
          color=red, vector([0,0],[1,-1]/sqrt(2)),
          color=blue, vector([0,0],[1,2]/sqrt(5)),
          title="tres vectores");
```

```
(%o43)
```



En la ayuda puedes encontrar varios opciones sobre el aspecto como se representan los vectores. Nosotros hemos usado `head_length` y `head_angle` para el tamaño de la punta de la fecha de los vectores.

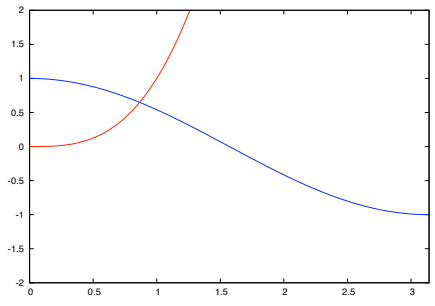
2.3.2 Opciones

Es importante destacar que hay dos tipos de opciones: locales y globales. Las locales sólo afectan al objeto que les sigue y, obligatoriamente, tienen que precederlo. En cambio las globales afectan a todos los objetos dentro de la orden `draw` y da igual su posición (aunque solemos escribirlas todas juntas al final).

Opciones globales

`xrange, yrange`: rango de las variables x e y. Por defecto se ajusta automáticamente al objeto que se esté representando pero hay ocasiones en que es preferible fijar un rango común.

```
(%i44)  draw2d(color=blue,
          explicit(cos(x),x,0,4*%pi),
          color=red,
          explicit(x^3,x,-5,5),
          xrange=[0,%pi],yrange=[-2,2])$
(%o44)
```

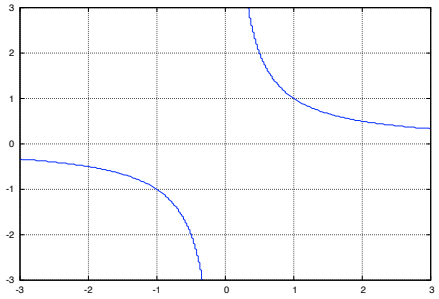


Si en el ejemplo anterior no limitamos el rango a representar, al menos en la coordenada y, es difícil poder ver a la vez la función coseno que toma valores entre 1 y -1 y la función x^3 que en 5 vale bastante más.

`grid`: dibuja una malla sobre el plano XY si vale `true`.

```
(%i45)  draw2d(
          color=blue,nticks=100,
          implicit(x*y=1,x,-3,3,y,-3,3),
          grid=true)$
```

(%o45)

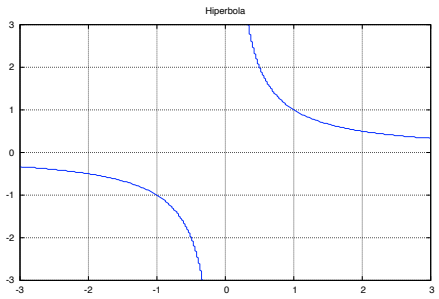


Acabamos de dibujar la hipérbola definida implícitamente por la ecuación $xy = 1$. La opción `grid` nos ayuda a hacernos una idea de los valores que estamos representando.

`title` ="título de la ventana" nos permite poner un título a la ventana donde aparece el resultado final. Es una opción global.

```
(%i46)  draw2d(
          color=blue,
          nticks=100,
          implicit(x*y=1,x,-3,3,y,-3,3),
          grid=true,
          title="Hiperbola"
        )$
```

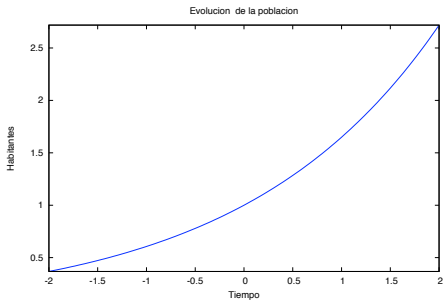
(%o46)



`xlabel, ylabel, zlabel`: indica la etiqueta de cada eje. Es una opción global.

```
(%i47)  draw2d(color=blue,
          explicit(exp(x/2),x,-2,2),
          xlabel="Tiempo",
          ylabel="Habitantes",
          title="Evolucion de la poblacion");
```

(%o47)



xaxis, yaxis: si vale `true` se dibuja el correspondiente eje. Es una opción global.

Opciones locales

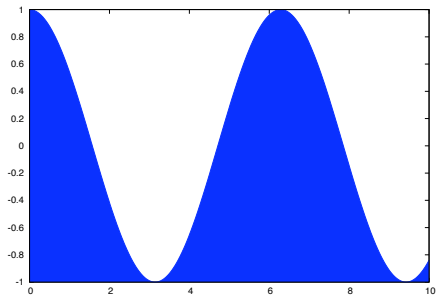
point_size: tamaño al que se dibujan los puntos. Su valor por defecto es 1. Afecta a los objetos de tipo `point`.

point_type: indica cómo se van a dibujar los puntos . El valor para esta opción puede ser un nombre o un número: none (-1), dot (0),plus (1), multiply (2), asterisk (3), square (4), filled_square (5), circle (6), filled_circle (7), up_triangle (8), filled_up_triangle (9), down_triangle (10), filled_down_triangle (11), diamant (12) y filled_diamant (13). Afecta a los objetos de tipo `point`.

filled_func: esta orden nos permite rellenar con un color la gráfica de una función. Existen dos posibilidades: si `filled_func` vale `true` se rellena la gráfica de la función hasta la parte inferior de la ventana con el color establecido en `fill_color`

```
(%i48)  draw2d(fill_color=blue,
          filled_func=true,
          explicit(cos(x),x,0,10)
        ):
```

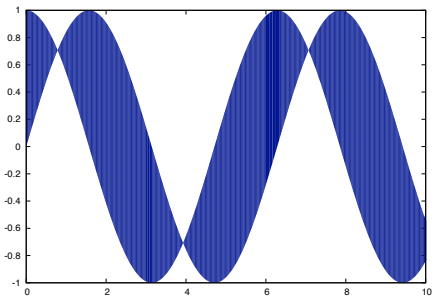
(%o48)



en cambio, si `filled_func` es una función, entonces se colorea el espacio entre dicha función y la gráfica que estemos dibujando

```
(%i49)  draw2d(
          filled_func=sin(x),
          fill_color=navy,
          explicit(cos(x),x,0,10));
```

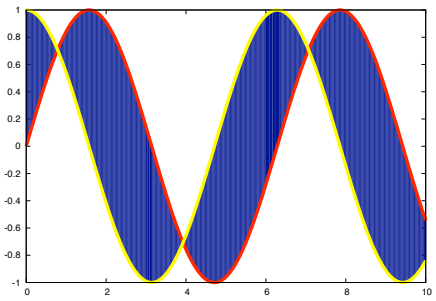
(%o49)



En este caso, tenemos sombreada el área entre las funciones seno y coseno. Podemos dibujar éstas también pero es necesario suprimir el sombreado si queremos que no tape a las funciones:

```
(%i50)  draw2d(
          filled_func=sin(x),fill_color=navy,
          explicit(cos(x),x,0,10),
          filled_func=false,color=red,line_width=10,
          explicit(sin(x),x,0,10),
          color=yellow,line_width=10,
          explicit(cos(x),x,0,10));
```

(%o50)



fill_color: ver el apartado anterior `filled_func`.

color: especifica el color en el que se dibujan líneas, puntos y bordes de polígonos. Directamente de la ayuda de *Maxima*:

```
Los nombres de colores disponibles son: ‘"white"’, ‘"black"’,
‘"gray0"’, ‘"grey0"’, ‘"gray10"’, ‘"grey10"’, ‘"gray20"’,
‘"grey20"’, ‘"gray30"’, ‘"grey30"’, ‘"gray40"’, ‘"grey40"’,
‘"gray50"’, ‘"grey50"’, ‘"gray60"’, ‘"grey60"’, ‘"gray70"’,
‘"grey70"’, ‘"gray80"’, ‘"grey80"’, ‘"gray90"’, ‘"grey90"’,
‘"gray100"’, ‘"grey100"’, ‘"gray"’, ‘"grey"’, ‘"light-gray"’,
‘"light-grey"’, ‘"dark-gray"’, ‘"dark-grey"’, ‘"red"’,
‘"light-red"’, ‘"dark-red"’, ‘"yellow"’, ‘"light-yellow"’,
‘"dark-yellow"’, ‘"green"’, ‘"light-green"’, ‘"dark-green"’,
‘"spring-green"’, ‘"forest-green"’, ‘"sea-green"’, ‘"blue"’,
‘"light-blue"’, ‘"dark-blue"’, ‘"midnight-blue"’, ‘"navy"’,
‘"medium-blue"’, ‘"royalblue"’, ‘"skyblue"’, ‘"cyan"’,
```

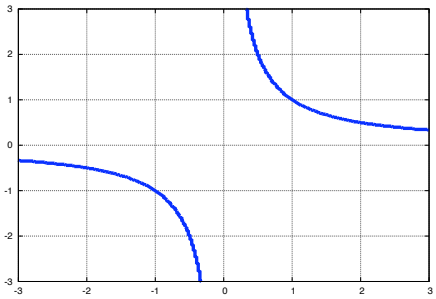
```
'"light-cyan"', '"dark-cyan"', '"magenta"', '"light-magenta"',
'"dark-magenta"', '"turquoise"', '"light-turquoise"',
'"dark-turquoise"', '"pink"', '"light-pink"', '"dark-pink"',
'"coral"', '"light-coral"', '"orange-red"', '"salmon"',
'"light-salmon"', '"dark-salmon"', '"aquamarine"', '"khaki"',
'"dark-khaki"', '"goldenrod"', '"light-goldenrod"',
'"dark-goldenrod"', '"gold"', '"beige"', '"brown"', '"orange"',
'"dark-orange"', '"violet"', '"dark-violet"', '"plum"' y
'"purple"'.
```

Ya lo hemos usado en casi todos los ejemplos anteriores.

line_width: grosor con el que se dibujan las líneas. Su valor por defecto es 1.

```
(%i51)  draw2d(
          color=blue,line_width=10,nticks=100,
          implicit(x*y=1,x,-3,3,y,-3,3),
          grid=true,
          )$
```

(%o51)

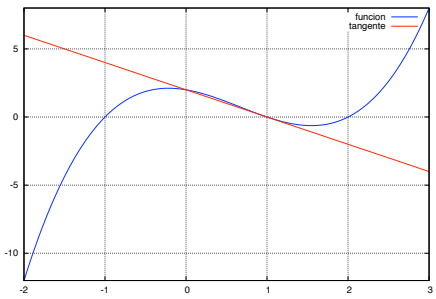


nticks: número de puntos que se utilizan para calcular los dibujos. Por defecto es 30. Un número mayor aumenta el detalle del dibujo aunque a costa de un mayor tiempo de cálculo y tamaño del fichero (si se guarda). Sólo afecta a los objetos de tipo `ellipse`, `explicit`, `parametric`, `polar` y `parametric`.

key: indica la leyenda con la que se identifica la función.

```
(%i52)  draw2d(color=blue,key="función",explicit(f(x),x,-2,3),
          color=red,key="tangente",explicit(tangente(x,1),x,-2,3),
          grid=true);
```

(%o52)



2.4 Animaciones gráficas

Con *wxMaxima* es muy fácil hacer animaciones gráficas que dependen de un parámetro. Por ejemplo, la función sen($x + n$) depende del parámetro n. Podemos representar su gráfica para distintos valores de n y con ello logramos una buena visualización de su evolución (que en este caso será una onda que se desplaza). Para que una animación tenga calidad es necesario que todos los gráficos individuales tengan el mismo tamaño y que no "den saltos" para lo que elegimos un intervalo del eje de ordenadas común.

Para ver la animación, cuando se hayan representado las gráficas, haz clic con el ratón sobre ella y desplaza la barra (slider) que tienes bajo el menú. De esta forma tú mismo puedes controlar el sentido de la animación , así como la velocidad.

`with_slider`	animación de `plot2d`
`with_slider_draw`	animación de `draw2d`

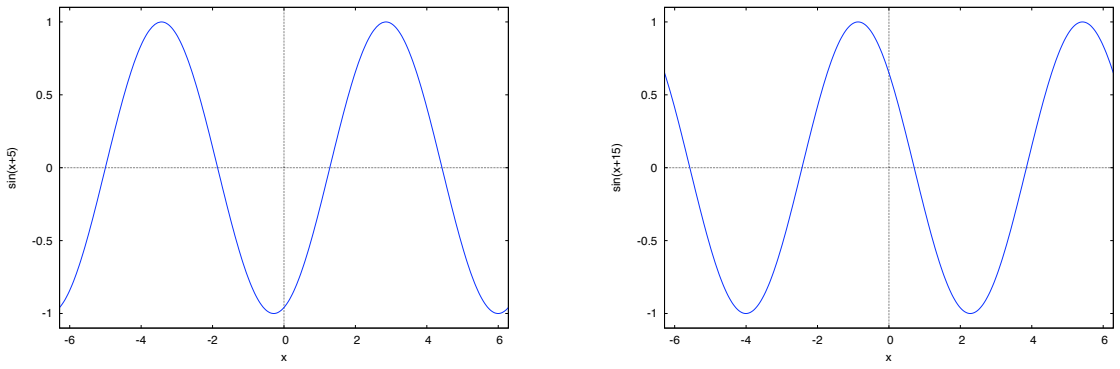
Tenemos dos posibilidades para construir animaciones dependiendo de si queremos que *Maxima* utilice `plot2d` o `draw2d`. En cualquier caso, en primer lugar siempre empezamos con el parámetro, una lista de valores del parámetro y el resto debe ser algo aceptable por el correspondiente comando con el que vayamos a dibujar.

with_slider

Por ejemplo, vamos a crear una animación con la orden `with_slider` de la función sen($x+n$), donde el parámetro n va a tomar los valores desde 1 a 20. La orden `makelist(i,i,1,20)` nos da todos los números naturales comprendidos entre 1 y 20. Ya veremos con más detalle en el Capítulo 3 cómo podemos manejar listas.

```
(%i53)  with_slider(n,
          makelist(i,i,1,20),
          sin(x+n),
          [x,-2*%pi,2*%pi],
          [y,-1.1,1.1]);
```

En la Figura 2.3 tienes la representación de algunos valores.



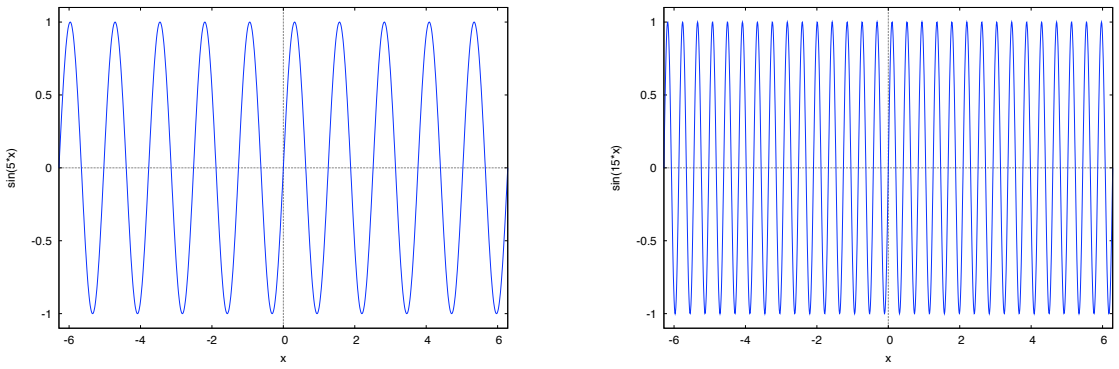
$n = 5$ $n = 15$

Figura 2.3 sen($x + n$)

Si en lugar de sumar el parámetro a la variable (que traslada la función), multiplicamos el parámetro y la variable conseguimos cambiar la frecuencia de la onda que estamos dibujando.

```
(%i54)  with_slider(n,makelist(i,i,1,20),sin(x*n),
          [x,-2*%pi,2*%pi],[y,-1.1,1.1]);
```

Puedes ver en la Figura 2.4 puedes ver cómo aumenta la frecuencia con n.



$n = 5$ $n = 15$

Figura 2.4 sen($n\,x$)

Si en lugar de `plot2d`, utilizamos el módulo `draw` para diseñar los dibujos, tenemos que usar `with_slider_draw`. De nuevo, en primer lugar va el parámetro, después, una lista que indica los valores que tomará el parámetro y el resto debe ser algo aceptable por la orden `draw`. Un detalle importante en este caso es que el parámetro no sólo puede afectar a la función sino que podemos utilizarlo en cualquier otra parte de la expresión. Por ejemplo, podemos utilizar esto para ir dibujando poco a poco una circunferencia en coordenadas paramétricas de la siguiente forma

```
(%i55)  with_slider_draw(
          t,makelist(%pi*i/10,i,1,20),
          parametric(cos(x),sin(x),x,0,t),
          xrange=[-1,1],
          yrange=[-1,1],
          user_preamble="set size ratio 1")$
```

En la Figura 2.5 tenemos representados algunos pasos intermedios

El tipo de objeto "parametric" no lo hemos comentado en los apartados anteriores. Nos permite representar la gráfica de una curva en el plano. En la ayuda de *Maxima* puedes encontrar más detalles.

En el último ejemplo podemos ver cómo se pueden combinar funciones definidas explícita e implícitamente juntos con vectores para obtener una representación de las funciones seno y coseno.

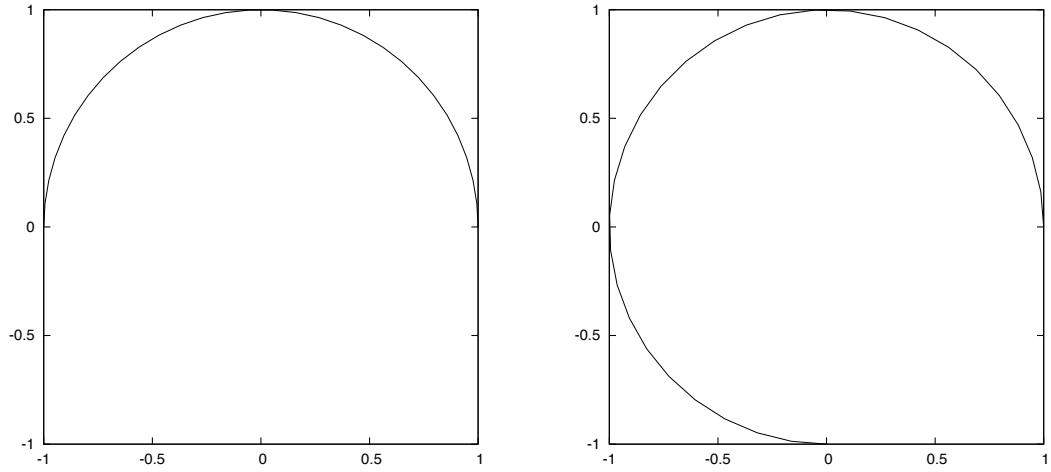


Figura 2.5 Construcción de una circunferencia en paramétricas

```
(%i56)  with_slider_draw(t,makelist(2*%pi*i/39,i,1,40),
          line_width=3, color=blue,
          parametric(cos(x),sin(x),x,0,t),
          color=light-red, key="seno",
          explicit(sin(x),x,0,t),
          color=dark-red, key="coseno",
          explicit(cos(x),x,0,t),
          line_type=dots, head_length=0.1,
          color=dark-red, key="",
          vector([0,0],[cos(t),0]),
          color=light-red, line_type=dots,
          head_length=0.1, key="",
          vector([0,0],[0,sin(t)]),
          line_type=dots, head_length=0.1, key="",
          vector([0,0],[cos(t),sin(t)]),
          xaxis=true,yaxis=true,
          title="Funciones seno y coseno",
          xrange=[-1,2*%pi],yrange=[-1,1]);
```

Para $t = 5$, el resultado lo puedes ver en la Figura 2.6

2.5 Ejercicios

Ejercicio 2.1. Representa en una misma gráfica las funciones seno y coseno en el intervalo $[-2\pi, 2\pi]$. Utiliza las opciones adecuadas para que una de las funciones se represente en azul y otra en rojo y, además, tengan grosores distintos.

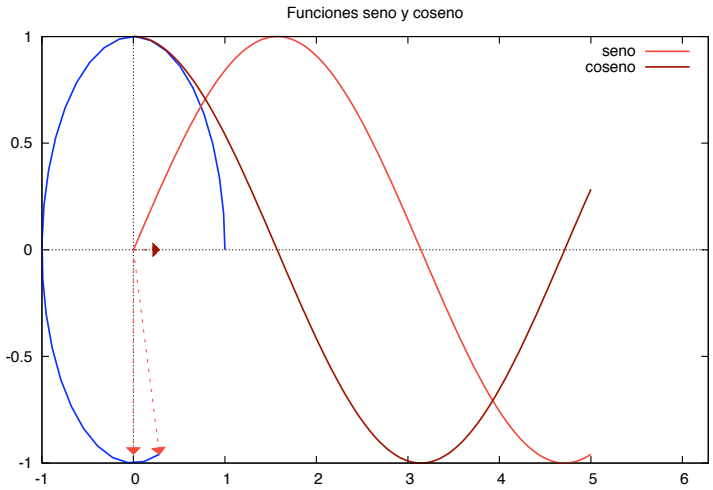


Figura 2.6 Las funciones seno y coseno

Ejercicio 2.2. Compara las gráficas de las funciones $\cos(x)$ y $\cos(-x)$. ¿A qué conclusión llegas sobre la paridad o imparidad de la función coseno? Haz lo mismo con las funciones $\operatorname{sen}(x)$ y $\operatorname{sen}(-x)$.

Ejercicio 2.3. Representa las funciones logaritmo neperiano, exponencial y $f(x) = x^2$ con colores diferentes. Compara el crecimiento de estas funciones cerca de cero y lejos de cero. ¿Qué ocurre si la base de la exponencial y del logaritmo es menor que 1?

Ejercicio 2.4. Dibuja las gráficas de las funciones coseno hiperbólico, seno hiperbólico, argumento seno hiperbólico y argumento coseno hiperbólico. ¿Alguna de ellas es par o impar? ¿Son positivas?

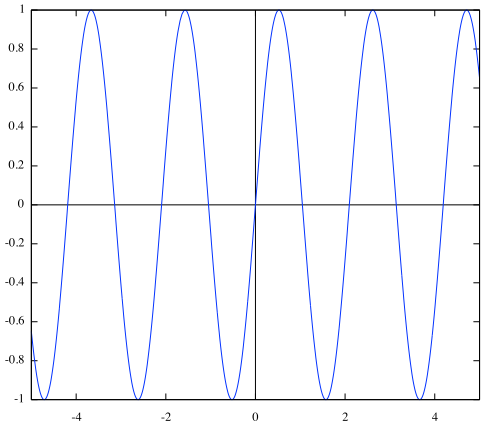
Ejercicio 2.5. Representa la curva $\cos(x)^2 - x\,\operatorname{sen}(x)^2$ en el intervalo $[-\pi, \pi]$ y sobre ella 5 puntos cuyo tamaño y color debes elegir tú. ¿Sabrías hacer lo mismo con 8 puntos elegidos aleatoriamente?[6]

Ejercicio 2.6. Representa la gráfica de la función $f : \mathbb{R}_0^+ \to \mathbb{R}$ definida como

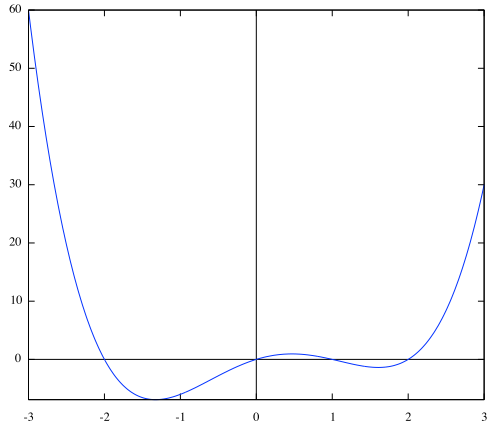
$$f(x) = \begin{cases} e^{3x+1}, & \text{si } 0 \leq x < 10, \\ \ln\left(x^2 + 1\right), & \text{si } x \geq 10. \end{cases}$$

Ejercicio 2.7. Encuentra las funciones cuyas gráficas corresponden a las siguientes curvas:

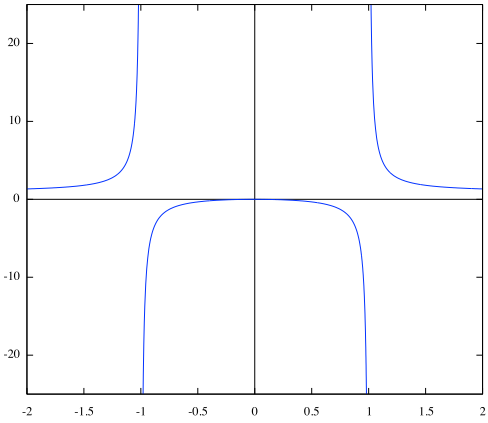
[6] En el siguiente capítulo puedes encontrar una explicación más detallada sobre como definir y operar con listas.


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-4
-2
0
2
4

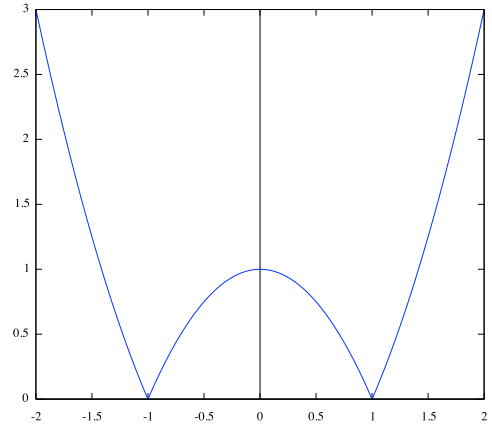
(a)


60
50
40
30
20
10
0
-3
-2
-1
0
1
2
3

(b)


20
10
0
-10
-20
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

(c)


3
2.5
2
1.5
1
0.5
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

(d)