Práctica_2

November 8, 2022

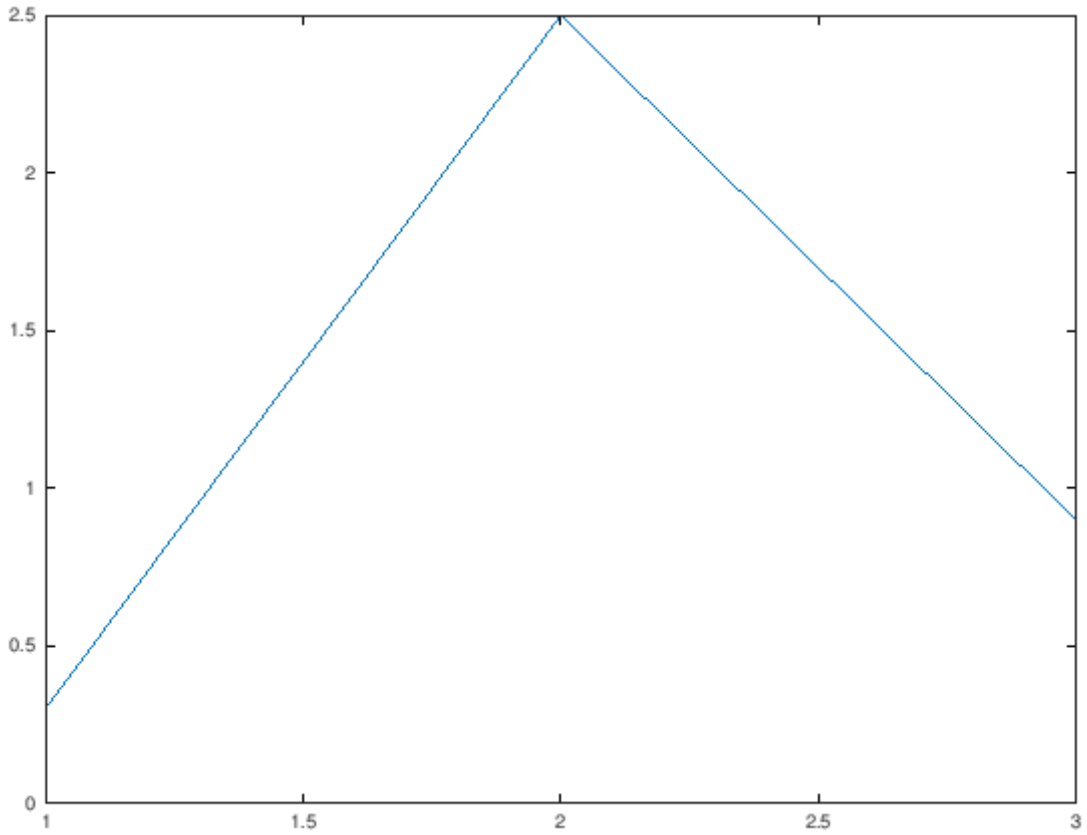
1 Práctica 2

[ ]:
```
clear all
addpath('./Biblioteca')
graphics_toolkit ("gnuplot"); %% Comando solo para jupyter notebooks
```

1.1 Los primeros gráficos

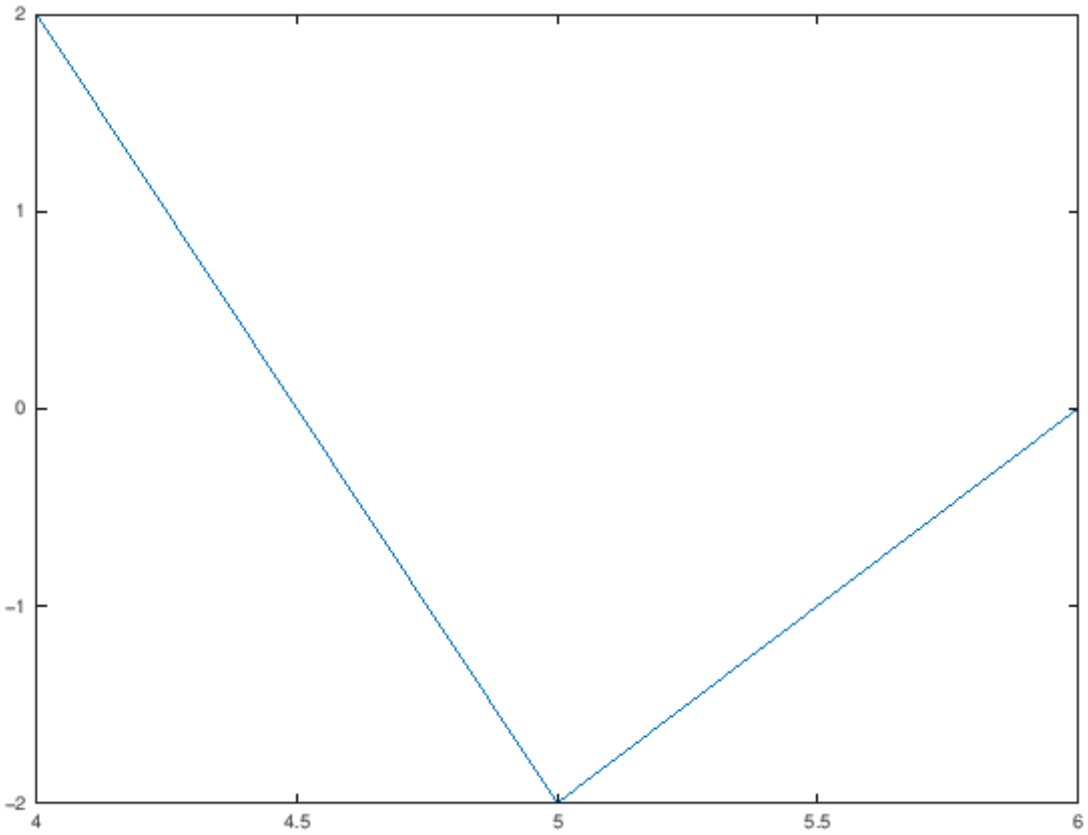
El comando básico para realizar gráficas bidimensionales es: plot(x,y), donde x será el vector de abscisas e y será el de ordenadas. Por ejemplo: teclea en la ventana de comandos

[2]:
```
x = [1, 2, 3];
y = [0.3, 2.5, 0.9];
plot(x,y)
```



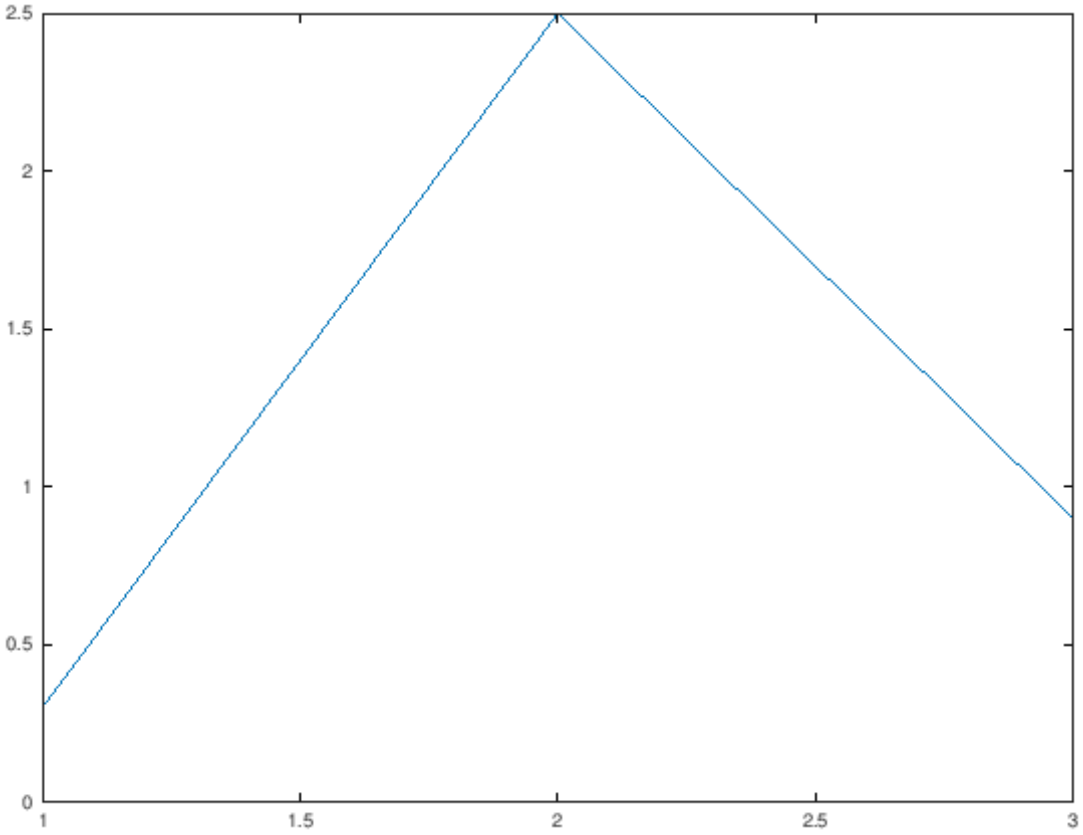
Ahora vamos a crear varias ventanas. Podemos intentar lo siguiente (teclea en la ventana de comandos):

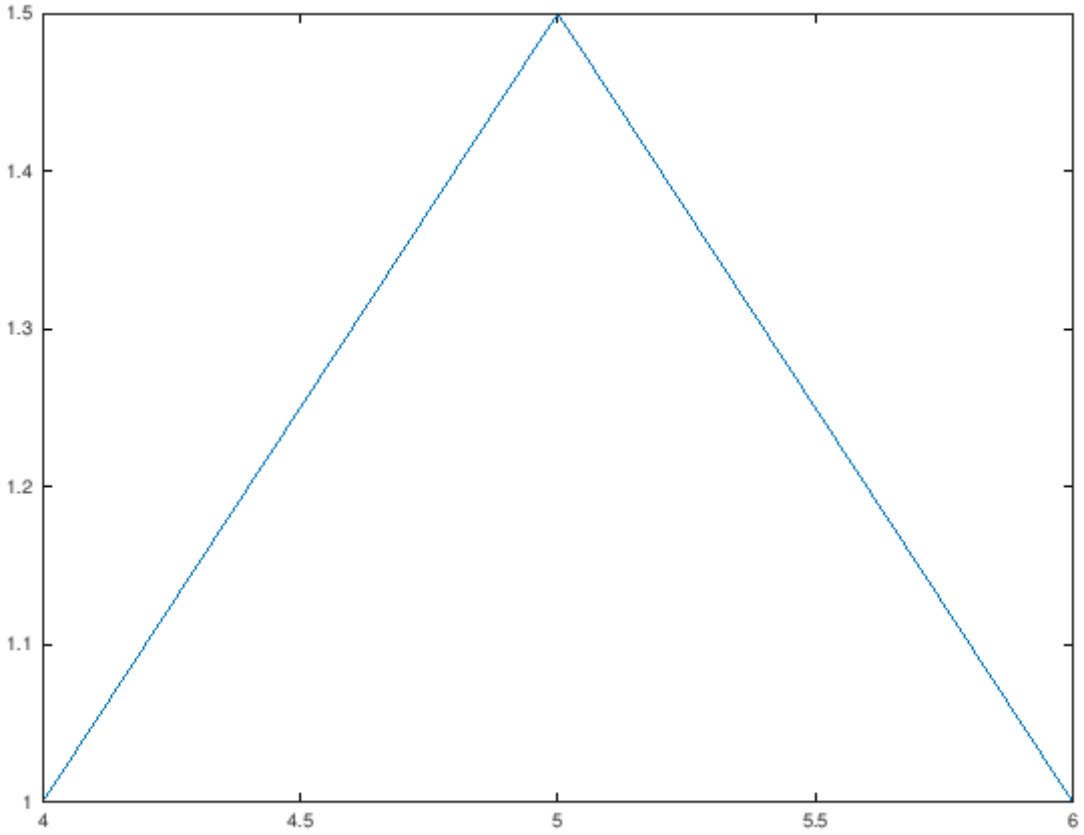
[3]:
```
plot(x,y)

plot([4,5,6], [2, -2, 0])
```



¿Qué ha ocurrido?… que el segundo gráfico ha reemplazado al primero en la ventana de la «Figure 1». Para evitar esto hacemos:

[4]:
```
plot(x,y)
figure(2)
plot([4,5,6], [1,1.5,1])
```


2.5
2
1.5
1
0.5
0
1
1.5
2
2.5
3



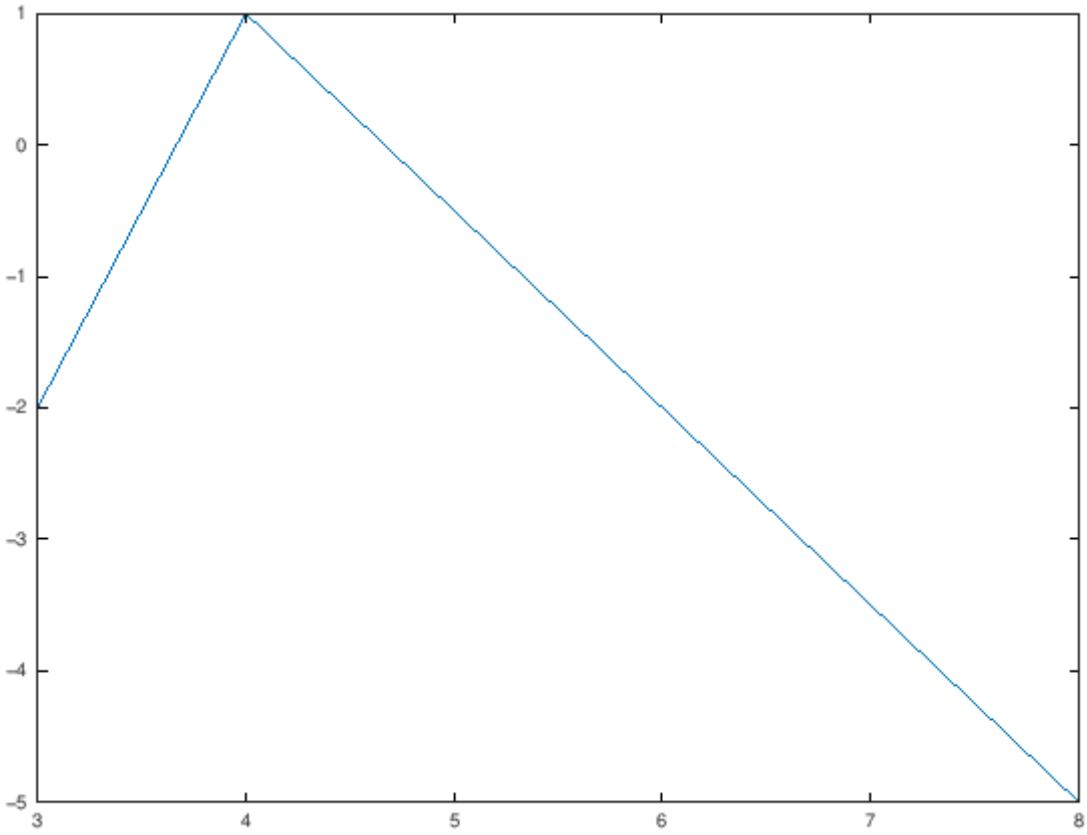
En los ejemplos que siguen repetiremos con frecuencia algunas órdenes en la ventana de comandos: si utilizas la flecha hacia arriba del teclado, en esta ventana, podrás ir repitiendo órdenes y editarlas, si quieres modificarlas.

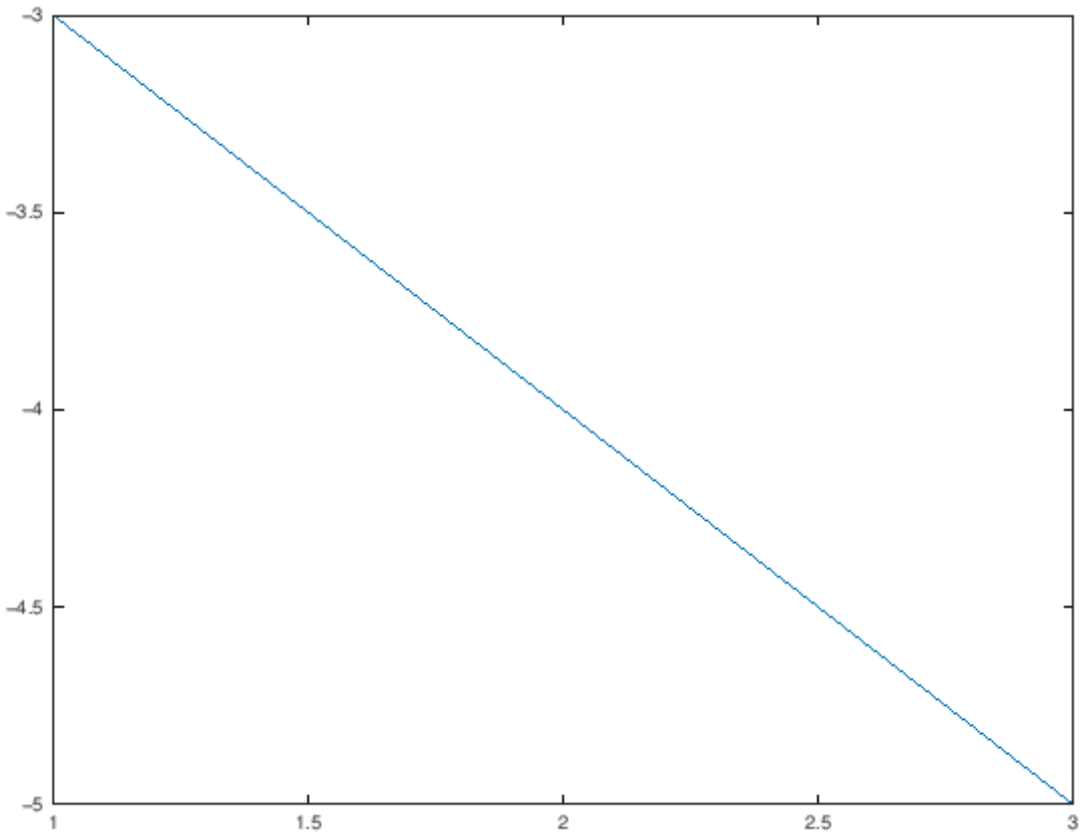
1.2 Algunas órdenes interesantes para el manejo de distintos gráficos

Cierra la ventanas gráficas creadas antes. A continuación teclea en la ventana de comandos las órdenes indicadas en lo que sigue, en el orden indicado.

- *figure(n)* Crea una ventana gráfica con nombre «Figure n». Si ya existe esta ventana entonces la declara como activa (tiene el foco): es decir, los comandos gráficos siguientes le afectarán a dicha ventana.

[5]:
```
figure(1)
plot([1,2,3],[3,4,5])
figure(2)
plot([1,2,3], [-3,-4,-5])
figure(1)
plot([3,4,8], [-2,1,-5])
```

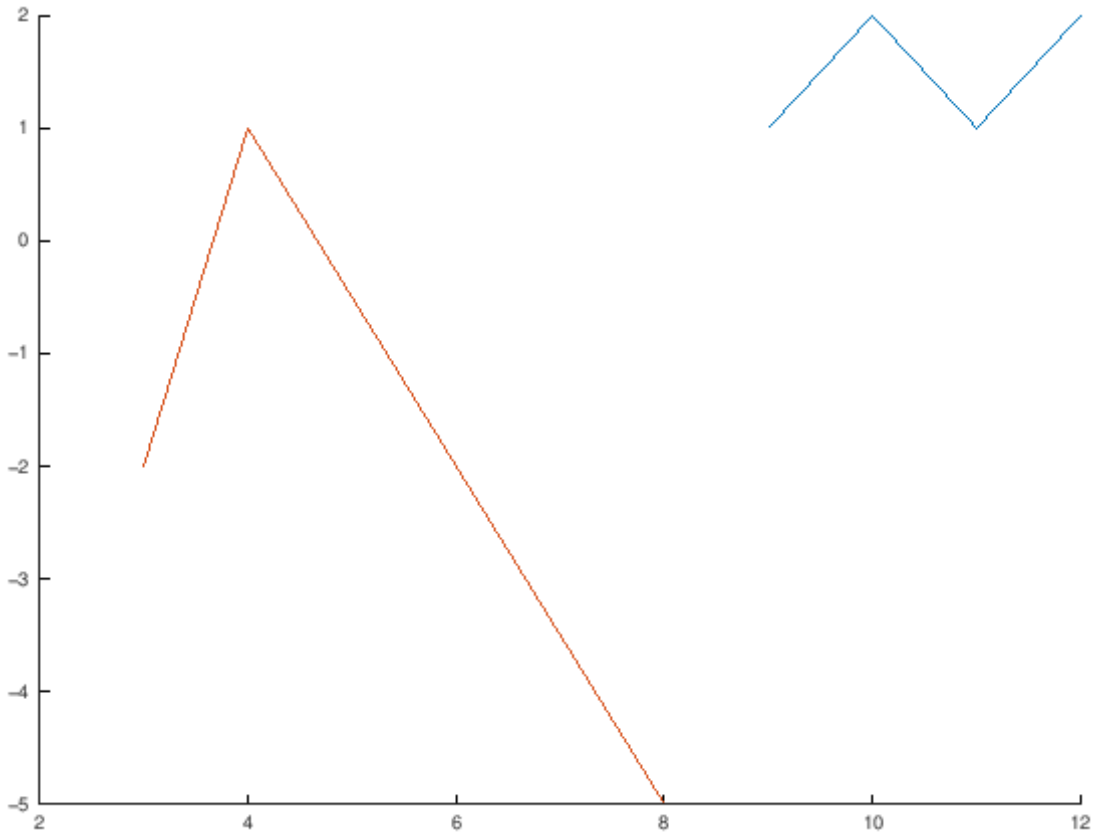

1
0
−1
−2
−3
−4
−5
3
4
5
6
7
8



¿En qué ventana se ha dibujado la última gráfica?

- *clf(n)* Vacía el contenido de la ventana «Figure n», pero la ventana sigue abierta. Qué es exactamente el contenido y qué desaparece es complicado de explicar ahora: una ventana gráfica es como un contenedor… clf elimina ciertas cosas del contenedor, pero no otras y esta orden tiene otras sintaxis más ricas, mediante las que se puede especificar más detalles de los elementos a eliminar… De momento nos contentamos con esto.
- *hold on* Permite añadir elementos gráficos a la ventana gráfica que tiene el foco, sin eliminar los que ya existen.

[6]:
```
figure(2) % %Por si no tiene el foco en estos momentos, lo ponemos en la fig. 2
hold on
plot([9,10,11,12], [1,2,1,2])
hold on
plot([3,4,8], [-2,1,-5])
```



- *close(n)* Cierra la ventana «**Figure n**»
- *close all* Cierra todas las ventanas «**Figure...**»

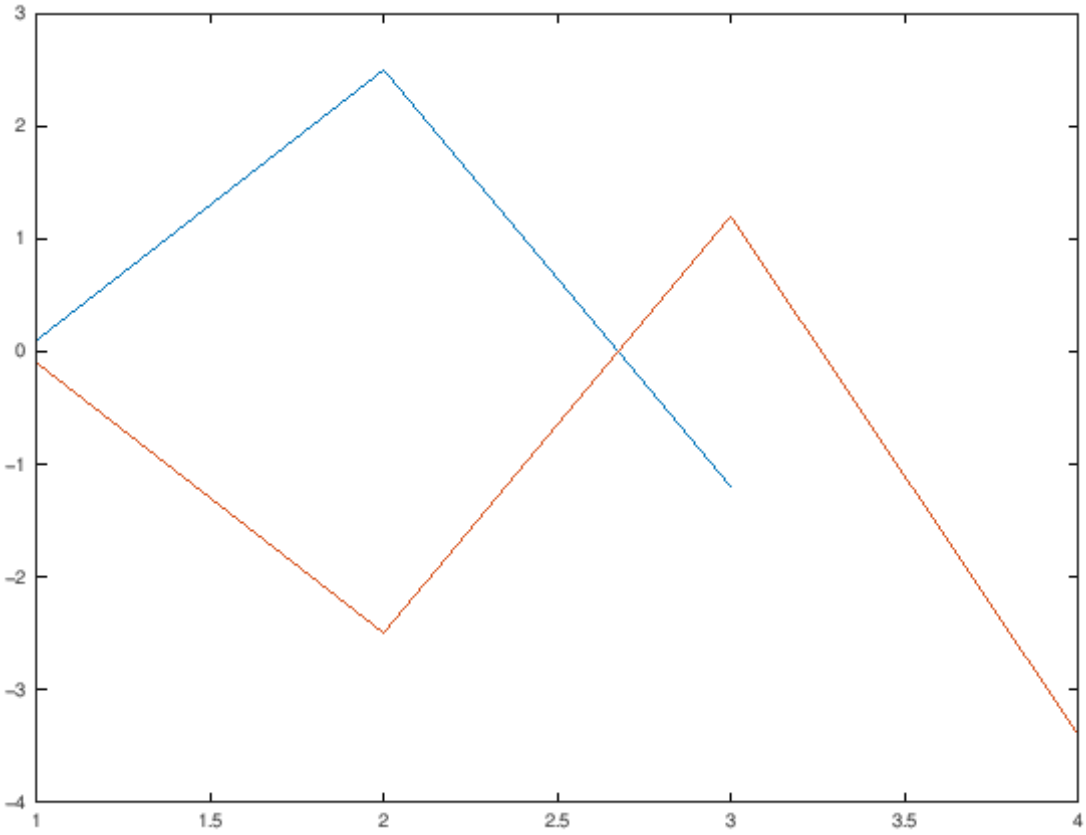
1.3 Elementos gráficos adicionales

Las posibilidades gráficas en Octave son bastante completas. Aquí vamos a describir sólo algunas de ellas.

En primer lugar, y **mmuy importante**:

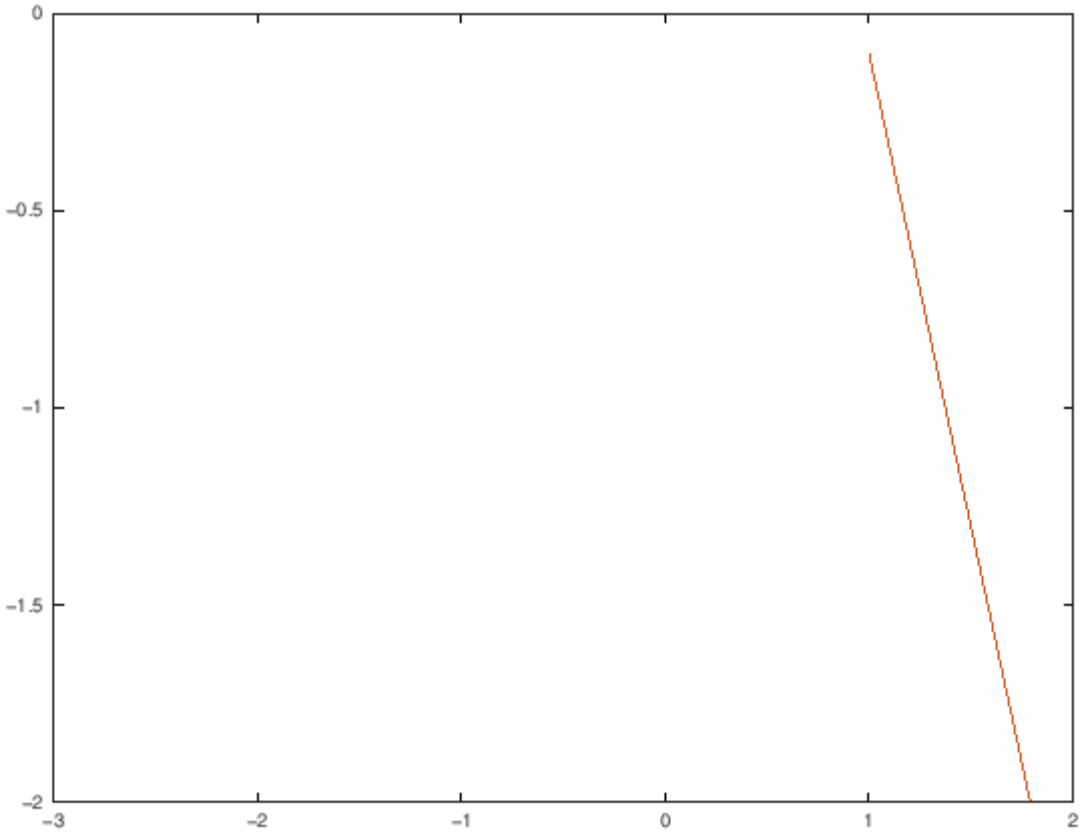
- Podemos incluir varias gráficas simultáneamente en un mismo comando plot:

[7]:
```
close all %% Para empezar de nuevo
x1 = [1,2,3];
y1 = [0.1,2.5,-1.2];
x2 = [1,2,3,4];
y2 = [-0.1, -2.5, 1.2, -3.4];
plot(x1,y1,x2,y2)
```



- Octave escala automáticamente el eje de ordenadas para que se ajuste al tamaño de la gráfica (también el de abscisas, si por ejemplo se añade un segundo gráfico a una ventana ya existente). Podemos fijar los límites de los intervalos de abscisas y ordenadas, con las órdenes *xlim([xmin,xmax])*, *ylim[ymin,ymax]*. Por ejemplo, siguiendo con las órdenes ya tecleadas en el punto anterior:

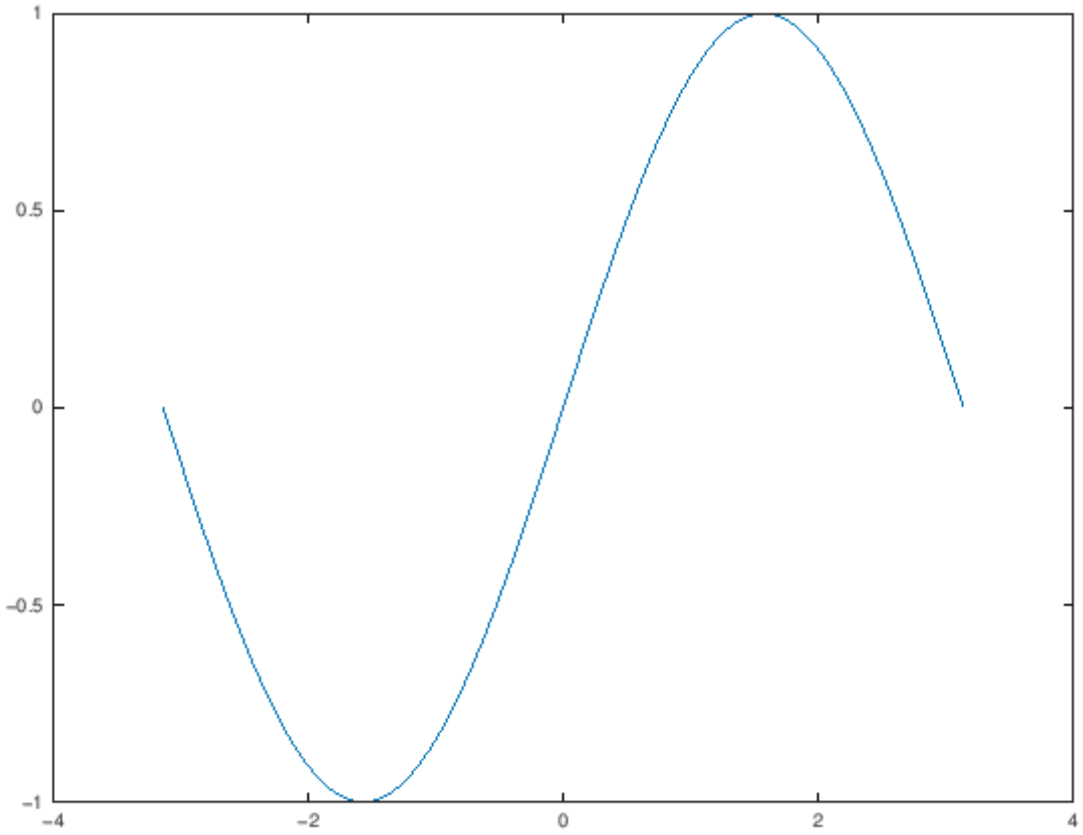
[8]:
```
plot(x1,y1,x2,y2)
xlim([-3,2])
ylim([-2,0])
```



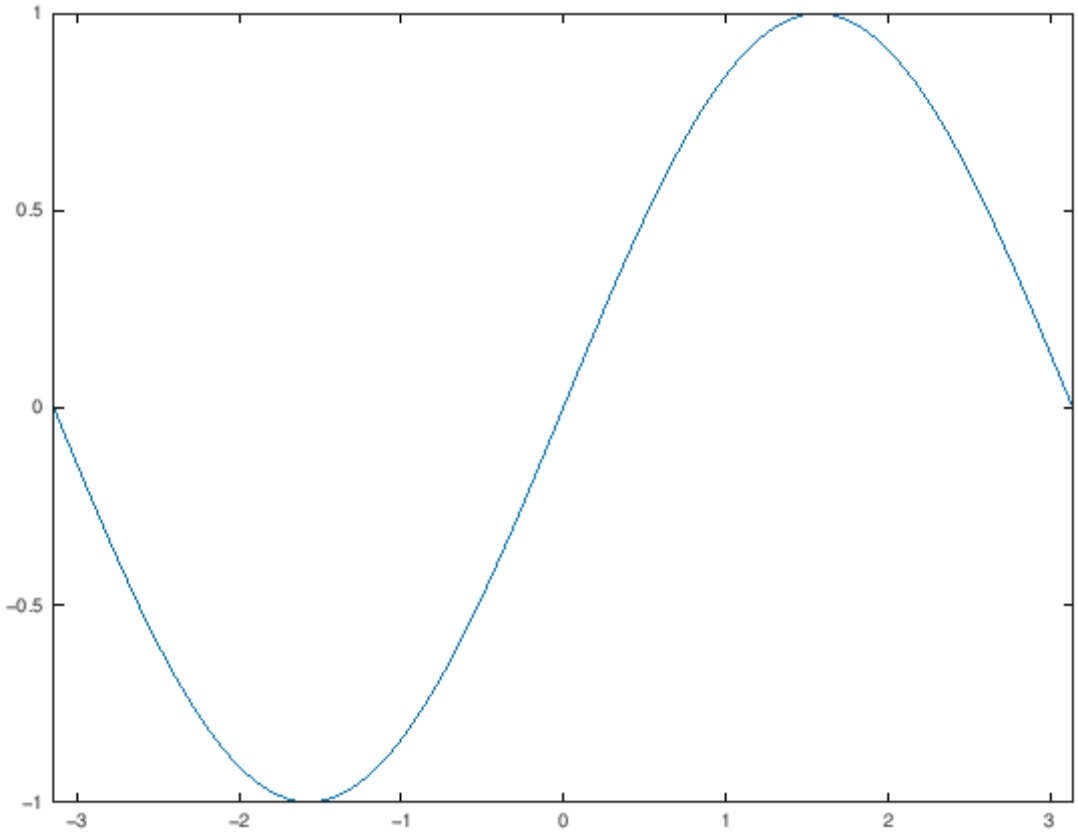
Antes de introducir otros elementos de control de gráficos vamos a dibujar algunas gráficas menos triviales que las de los ejemplos anteriores.

Por ejemplo, para dibujar la función sen(x) en una malla de puntos (201) en el intervalo $[-\pi, \pi]$:

[9]:
```
x = linspace(-pi,pi,201);
plot(x, sin(x)) %% Observa el intervalo de abcisas en el gráfico.
```

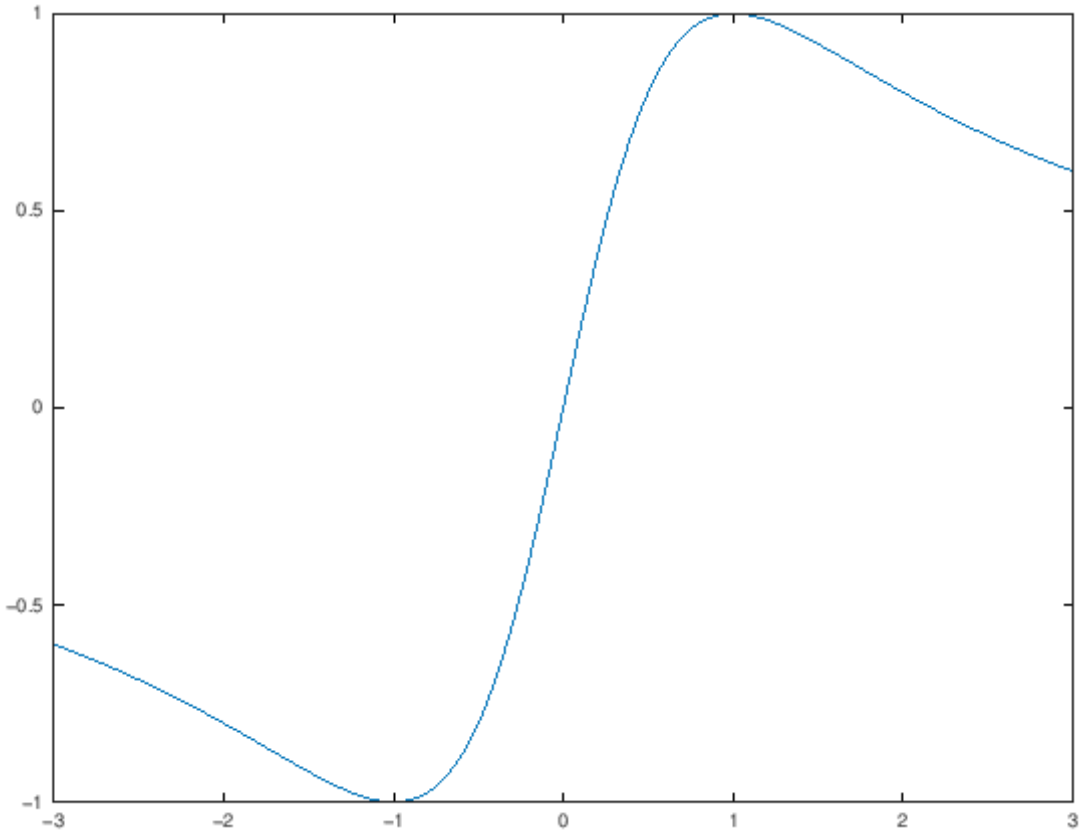


[10]:
```
plot(x, sin(x))
xlim([-pi, pi]) %% Ahora el intervalo de abcisas ha cambiado
```



Otro ejemplo, utilizando <>:

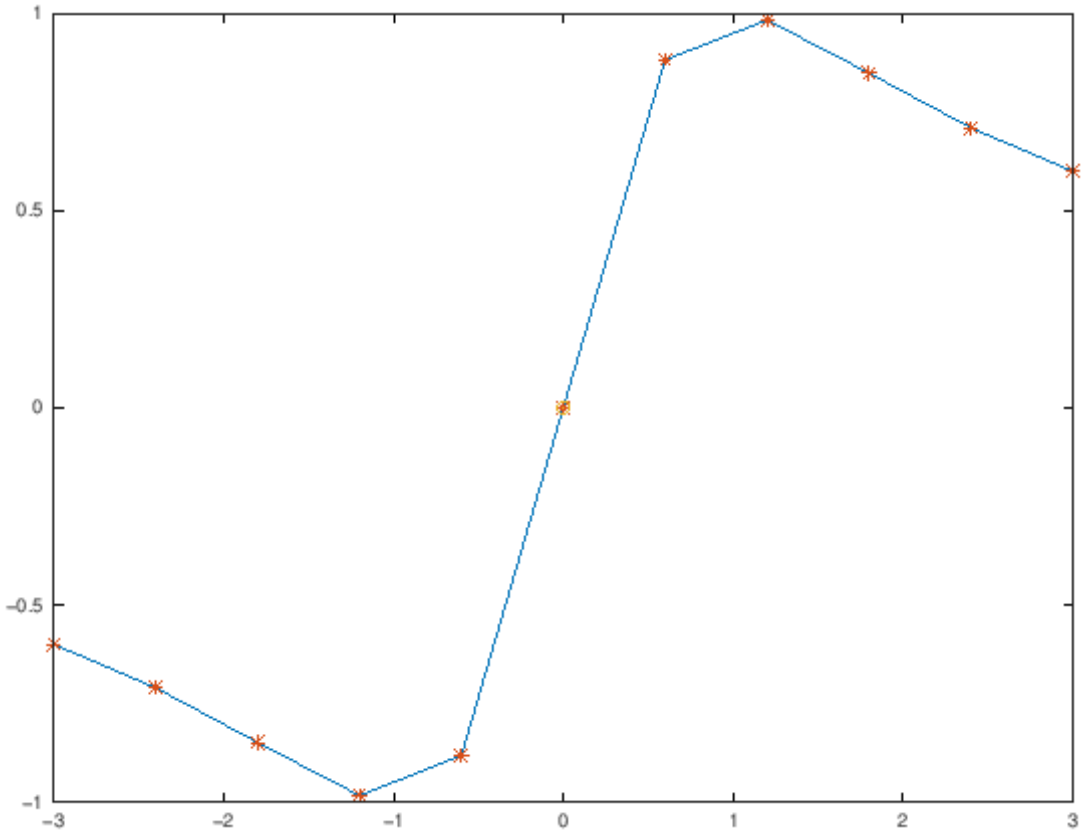
[11]:
```
mifun = @(x) 2*x ./ (1+x.^2);
x = linspace(-3,3,201);
plot(x, mifun(x))
```



1.4 Incluyendo puntos en un gráfico

Se pueden incluir puntos en un gráfico pasando a **plot** dos vectores: un vector con las abscisas de los puntos que se quieren dibujar, y un vector con sus ordenadas, e incluyendo además una indicación de cómo dibujar un punto, mediante el símbolo adecuado, por ejemplo '*', 'o':

[12]:
```
mifun = @(x) 2*x./(1+x.^2);
x = linspace(-3,3,11);
plot(x,mifun(x))
hold on
plot(x, mifun(x), '*', [0], [0], 'o')
```



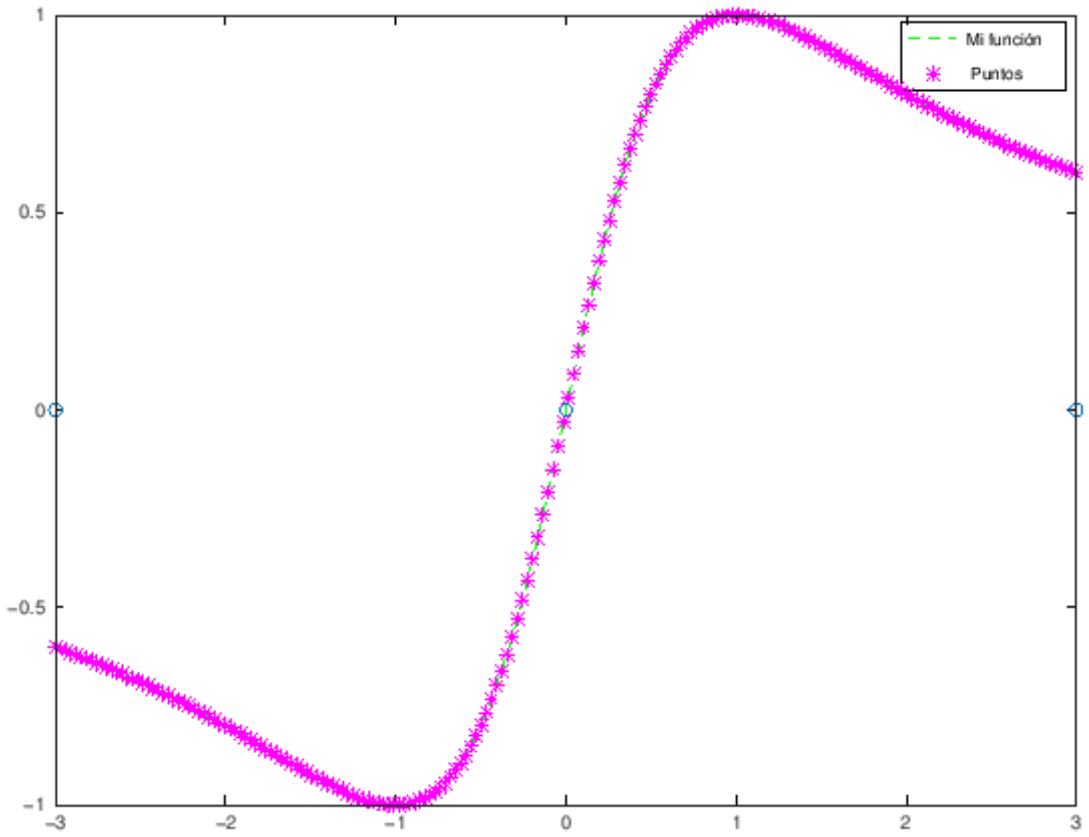
1.5 Colores y leyendas

Podemos controlar muchos otros elementos, como los colores con los que se dibujan las gráficas o los puntos, el estilo de las líneas y los puntos, así como la inclusión de leyendas, títulos de la ventana, etc. He aquí algunos ejemplos.

La orden **plot** tiene numerosas variantes en cuanto a las variables que espera. Una de ellas incluye el «formato», una cadena de caracteres que controla cómo se muestra cada elemento gráfico. El «formato» es una cadena de caracteres compuesta por cuatro partes, todas ellas optativas. Estas partes son: estilo de la línea, estilo de puntos (marcadores), color y leyenda. Atención: la leyenda, si se incluye, debe encerrarse entre signos de punto y coma, es decir, debe ser de la forma ;leyenda;.

Veamos un ejemplo rápido:

[13]:
```
close all
x = linspace(-3,3,200);
plot(x,mifun(x), 'g--;Mi función;')
hold on
plot(x, mifun(x), 'm*; Puntos;', [0,-3,3], [0,0,0], 'o')
```



En el primer caso hemos especificado en el formato el color verde (g, de «green>), un estilo de línea discontinua (–), y la leyenda (Mi función). En el segundo caso se dibujan sólo puntos (debido al formato especificado con *), en color magenta (m) y una nueva leyenda para este elemento gráfico adicional.

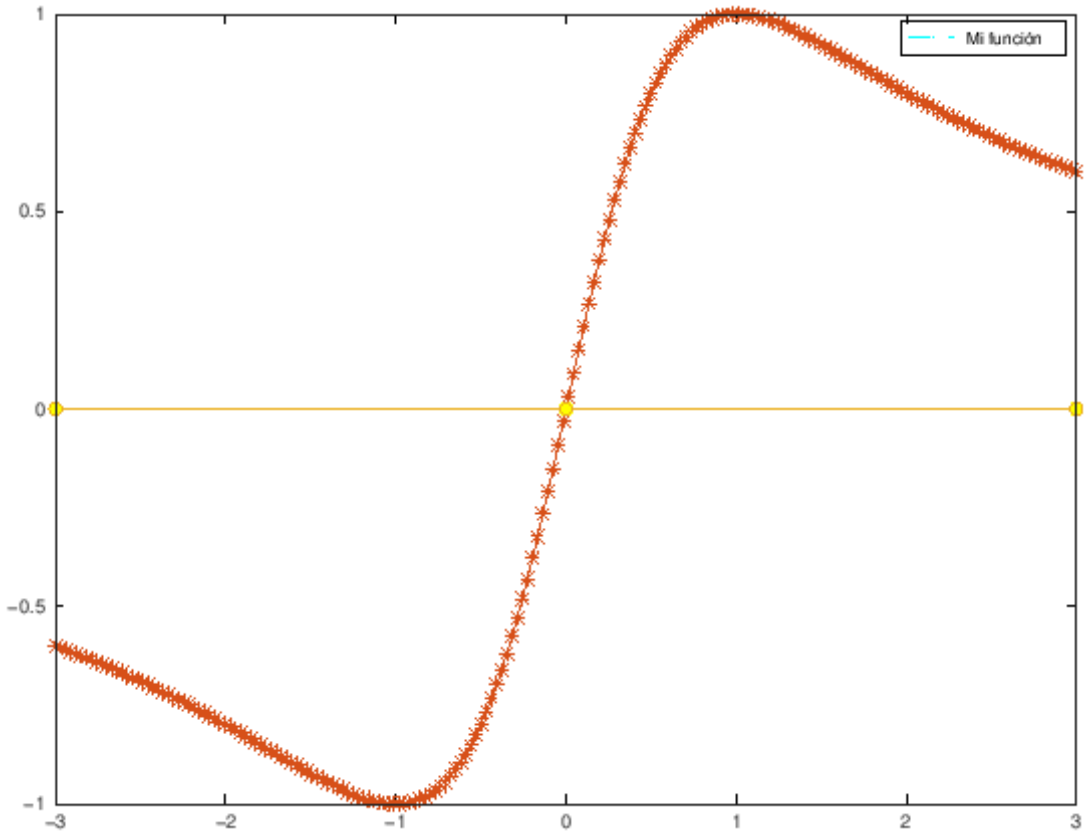
A continuación tenéis una lista de valores admitidos en el formato:

- **Estilo de línea** Posibles valores:
  - **-** Línea continua, valor por defecto.
  - – Línea discontinua.
  - **:** Línea de puntos.
  - **-.** Discontinua compuesta de raya y punto.
- **Colores** Posibles valores: *b, r, g, b, y, m, c, w.*
- **Marcas de puntos** Entre otros: _+, o, *, ., x, s, etc…_

Una sintaxis alternativa, y que ofrece más posibilidades es **plot(x,y, prop1,valor1,prop2,valor2,...)**, donde prop1, prop2 representan el nombre de distintas propiedades, seguidas cada una de su valor. Entre las propiedades: *linestyle, linewidth, color, marker, markersize, markeredgecolor,…*

El nombre de la propiedad y los valores que no sean numéricos deben escribirse entre apóstrofos o comillas (como todas las cadenas). Un ejemplo con pequeñas variaciones sobre el anterior:

[14]:
```
close all
x = linspace(-3,3,200);
plot(x, mifun(x), 'color', 'c', 'linestyle', '-.', 'linewidth', 2)
legend('Mi función')
hold on
plot(x,mifun(x),'marker','*',[0,-3,3],[0,0,0],'marker','o','markerfacecolor','y')
```



La orden **legend** es otra forma de incluir una leyenda, admite muchas variantes, pero no entraremos en ellas ahora.

1.6 Ejercicios para la práctica 1

1.6.1 Ejercicio 1:

Este primer ejercicio consiste en dibujar algunas gráficas sencillas, para familiarizarnos con los elementos gráficos. Debemos escribir el código en un archivo denominado Ejercicio1.m. Como se van añadiendo elementos apartado a apartado, utiliza close all, desde la ventana de comandos, para cerrar la ventana gráfica entre apartado y apartado, antes de volver a ejecutar el script (también puedes cerrar directamente la ventana con el ratón).

[15]:
```
clear %% Borra los valores de todas las variables usadas hasta el momento
```

1.1 Dibuja en una ventana la gráfica de la función $f(x) = \pi + \sqrt{1+x^2}$ en el intervalo [-3,3], utilizando **179** puntos. Recuerda que en un script podemos definir una función con el código:

[16]:
```
f = @(x) pi+sqrt(1+x.^2);
```

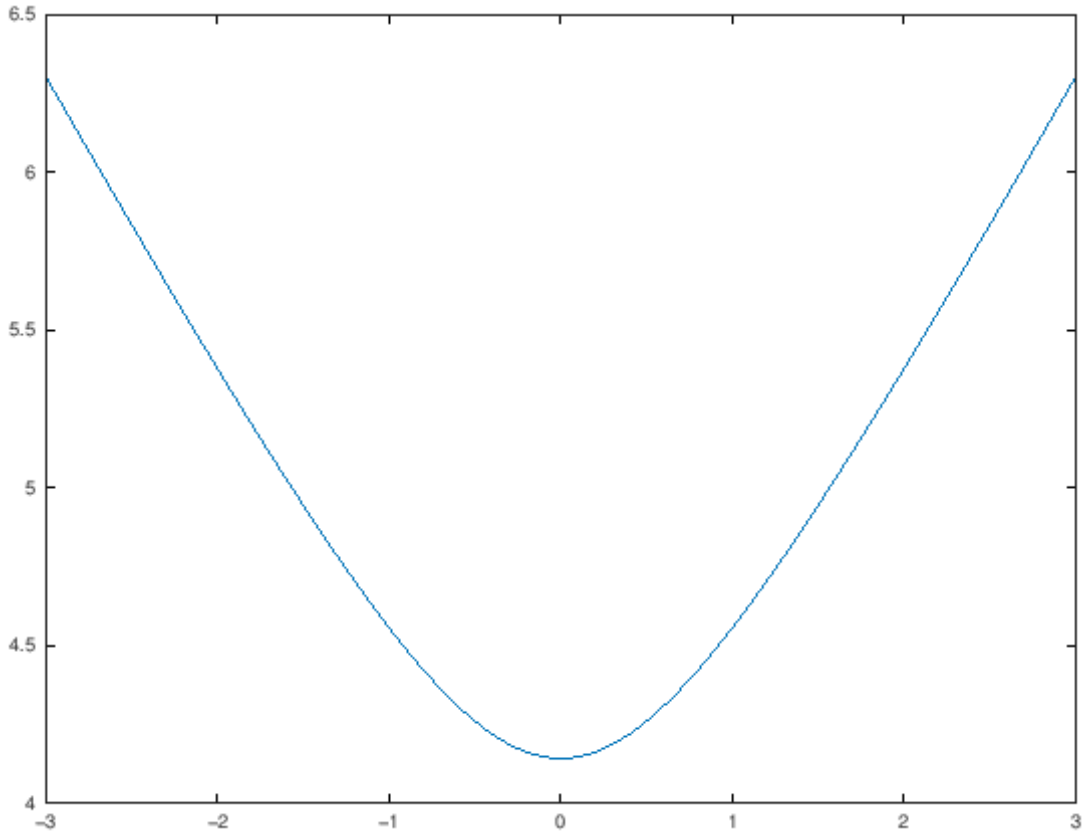
Aunque de forma alternativa podemos definirla, dentro del propio archivo script, mediante el código:

[17]:
```
function ret=f(x)
ret = pi +sqrt(1+x.^2)
endfunction
```

Aunque en este ejemplo no se ve la necesidad de la segunda opción, es conveniente recordar que la primera sintaxis sólo permite una expresión tras la parte de código @(x). En ambas formas, la definición de la función debe preceder en el código a cualquier llamada a la misma.

Solución 1.1

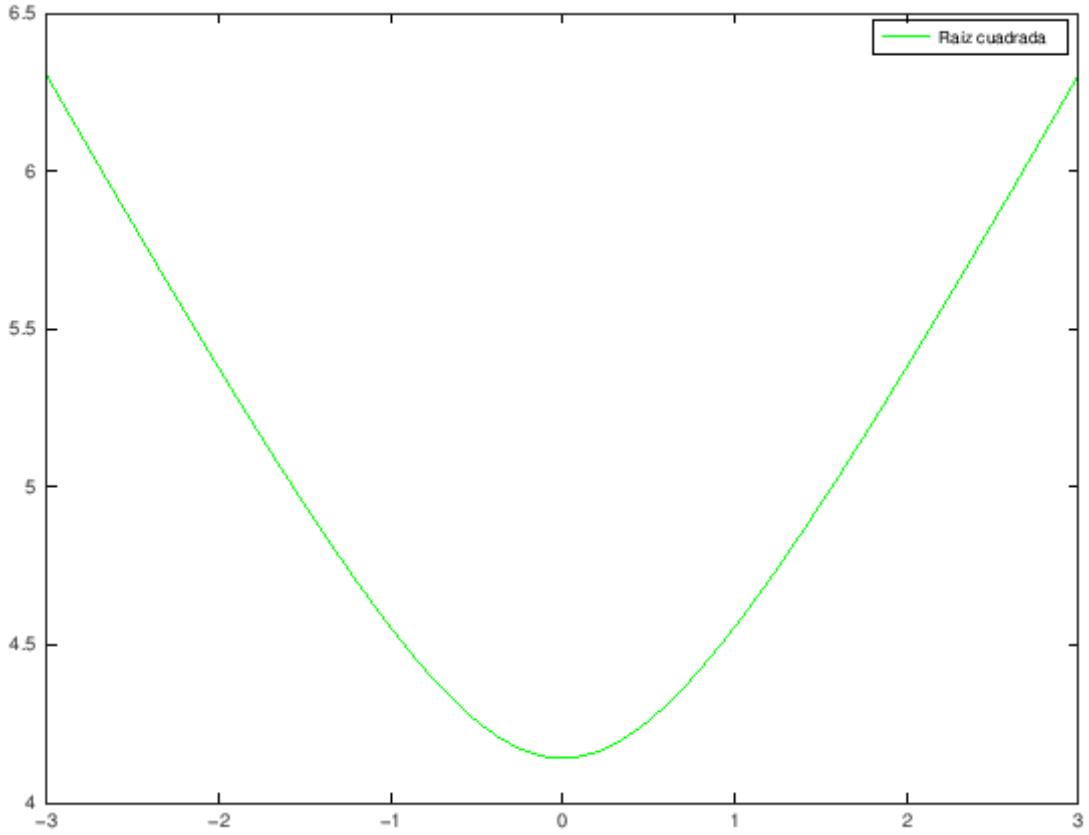
[18]:
```
raiz_cuadrada = @(x) pi+sqrt(1+x.^2);
x = linspace(-3,3,179);
plot(x, raiz_cuadrada(x))
```



1.2 Cambia el color de la gráfica y el grosor de la línea (el valor del grosor por defecto es 0.5). Añade una leyenda.

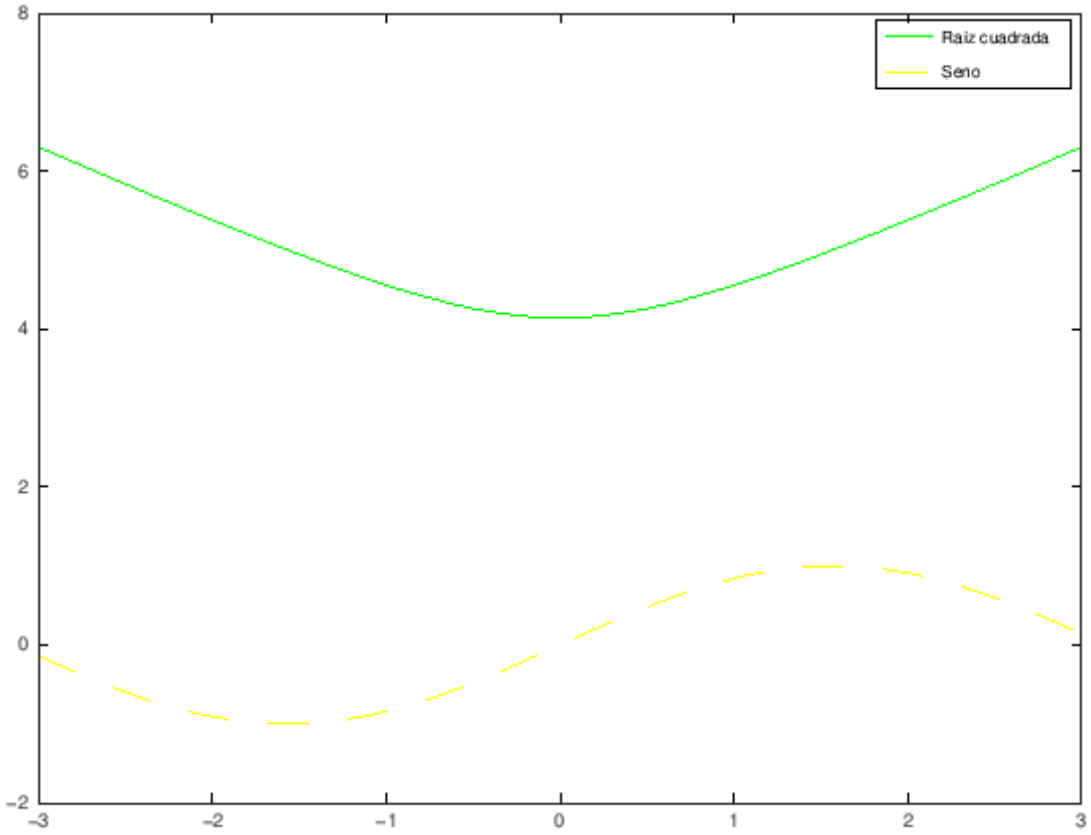
Solución 1.2

[19]:
```
plot(x, raiz_cuadrada(x), 'color', 'g', 'linewidth', 1)
legend('Raíz cuadrada')
```



1.3 Añade a la misma ventana la gráfica de la función g(x) = sen(x) sobre el mismo intervalo. Elige el color de esta nueva gráfica y fija un estilo de línea discontinuo. Incluye también una leyenda.

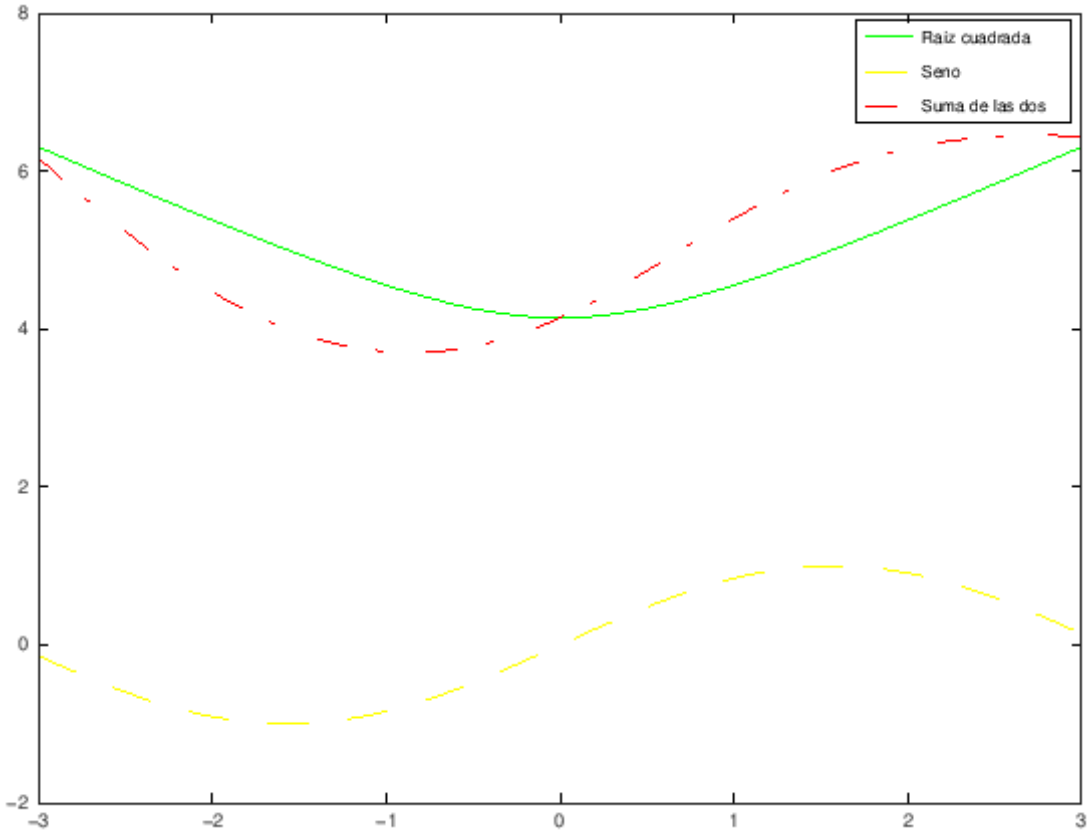
[20]:
```
close all
seno = @(x) sin(x);

%% O alternativamente podemos definir la función como:
%%
%% function ret = seno(x)
%%      ret = sin(x);
%% endfunction

plot(x, raiz_cuadrada(x), 'color', 'g', 'linewidth', 1)
hold on
plot(x, seno(x), 'color', 'y', 'linestyle', '--', 'linewidth', 2)
legend( 'Raíz cuadrada', 'Seno')
```



1.4 Finalmente, añade en la misma ventana la gráfica de $h = f + g$.

Solución 1.4

[21]:
```
h = @(x) seno(x) + raiz_cuadrada(x);

%{
    También podemos definir la función como:

    function ret = h(x)
        ret = seno(x) + raiz_cuadrada(x);
    endfunction
%}

plot(x, raiz_cuadrada(x), 'color', 'g', 'linewidth', 1)
hold on
plot(x, seno(x), 'color', 'y', 'linestyle', '--', 'linewidth', 2)
hold on
plot(x, h(x), 'color', 'r', 'linestyle', '-.', 'linewidth', 3)
legend('Raíz cuadrada', 'Seno', 'Suma de las dos')
```

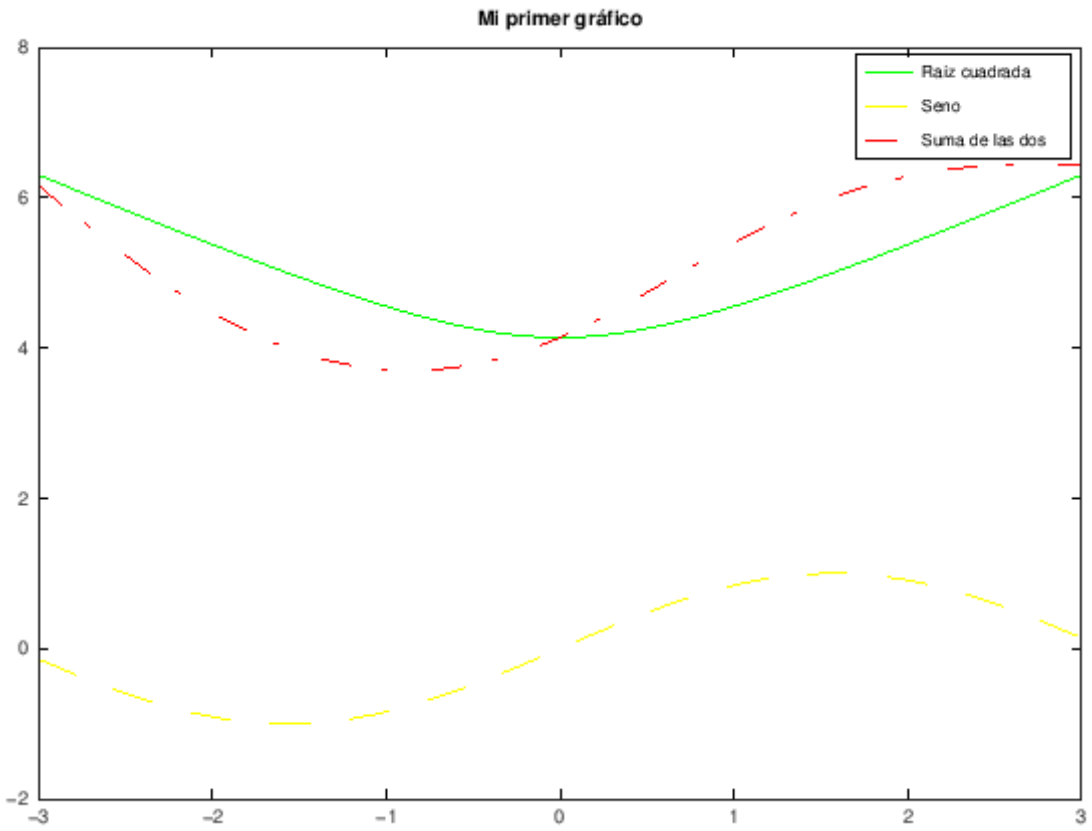


1.5 y 1.6 Añade la orden **title('Mi primer gráfico')**. ¿Qué efecto produce?

Conviene terminar el archivo .m, con el código **close all**, para que las ventanas gráficas se cierren (si es lo que deseamos), de forma que si volvemos a ejecutar el programa los gráficos y leyendas no se acumulen, ¡recuérdalo!

Solución 1.5 y 1.6

[22]:
```
plot(x, raiz_cuadrada(x), 'color', 'g', 'linewidth', 1)
hold on
plot(x, seno(x), 'color', 'y', 'linestyle', '--', 'linewidth', 2)
hold on
plot(x, h(x), 'color', 'r', 'linestyle', '-.', 'linewidth', 3)
legend('Raíz cuadrada', 'Seno', 'Suma de las dos')
title('Mi primer gráfico')
%% close all: debido al software que estoy usando no es necesario escribir
%% esta línea de código.
```



[23]:
```
clear
```

1.6.2 Ejercicio 2:

Vamos a definir una función a trozos en Octave y a representar su gráfica.

La principal dificultad para definir una función a trozos en Octave es que se espera que el argumento, *x*, de una función, *f(x)*, sea vectorial, por ejemplo para poder dibujar su gráfica con *plot(x,f(x))*, siendo x el vector de las abscisas. Puesto que para poder definir la función por trozos debemos conocer en qué intervalo o región está la variable, debemos realizar comparaciones… ¿Y cómo se comparan vectores?

2.1 Escribe los siguientes comandos. ¿Cómo explicas el resultado?

Solución 2.1

[24]:
```
x = [1, -1, 2, -2];
x<0
```

```
ans =

  0  1  0  1
```

Octave comprueba para cada uno de los componentes de la matrix x si se verifica que sea menor que cer0. Devuelve una matriz de ceros y unos dependiendo de si la condición se cumple (1) o no (0) para cada compoente.

2.2 ¿Qué ocurre si utilizamos la condición x<0 para tomar una decisión? Escribe los siguientes ejemplos e intenta llegar a una conclusión:

Solución 2.2

[25]:
```
x = [1, -1, 2, -2];
if x<0 disp('negativo') else disp('positivo') endif

x=[-1,1,-2,2];
if x<0 disp('negativo') else disp('positivo') endif

x=[1,2,3,4];
if x<0 disp('negativo') else disp('positivo') endif

x=[-1,-2,-3,-4];
if x<0 disp('negativo') else disp('positivo') endif
```

```
positivo
positivo
positivo
negativo
```

La conclusión es que si hay varios valores solo es necesario que la condición se cumpla para uno de ellos para que Octave interpréte que la condición se ha cumplido en general.

2.3 Tenemos varias formas de aproximar el problema.

2.3.1 Manejar las funciones características simuladas en Octave. Recuerda que si A es un conjunto, digamos en R, la función característica de A es:

$$\mathcal{X}_A(x) = \begin{cases} 1 \text{ si } x \in A \\ 0 \text{ si } x \notin A \end{cases}$$

Así, por ejemplo, si $A = (-\infty, 0)$ y \$B = [0, $+\infty$) \$ la función $f(x) := -x\mathcal{X}_A(x) + x\mathcal{X}_B(x)$ coincide con $|x|$

Esto es fácil de escribir en Octave: f=@(x) -x.*(x<0)+x.*(x>=0). Por ejemplo, escribe

[26]:
```
x = [1,-1,2,-2];
f=@(x) -x.*(x<0)+x.*(x>=0);
f(x)
```

```
ans =

   1   1   2   2
```

¿Obtienes el resultado buscado? Haz otras pruebas para confirmarlo…

2.3.2 La segunda opción para resolver este problema es recorrer las componentes del vector x en un bucle y realizar la comparación componente a componente. Por ejemplo, escribe:

[27]:
```
function ret=f(x)
    for i = 1:length(x)
        if (x(i)<0)
            ret(i) = -x(i);
        else
            ret(i) = x(i);
        endif
    endfor
endfunction

f([-1,2,-3,-10])
```

```
ans =

    1    2    3   10
```

¡Octave es muchísimo más rápido manejando expresiones vectoriales o matriciales que recorriendo las componentes del vector o matriz y operando directamente sobre ellas!

2.3.3 Terminamos el ejercicio escribiendo un archivo **Ejercicio2.m** donde vamos a representar en dos ventanas distintas las gráficas de las funciones:

$$f(x) = \begin{cases} -1 & \text{si } x \leq -2 \\ (x+2)^2 - 1 & \text{si } -2 < x < 2 \\ 15 & \text{si } x \geq 2 \end{cases}$$

$$g(x) = \begin{cases} xsen(\frac{1}{x}) & \text{si } x < 0 \\ 0 & \text{si } x = 0 \\ x^2 sen(\frac{1}{x}) & \text{si } x > 0 \end{cases}$$

Dibuja la gráfica de f en el intervalo $[-5,\ 5]$ y la de g en el intervalo $[-0.25,\ 0.25]$. Finalmente añade en la segunda ventana las gráficas de las funciones «moduladoras» de g:

$$m_1(x) = \begin{cases} x & \text{si } x < 0 \\ -x^2 & \text{si } x \geq 0 \end{cases}, \qquad m_2(x) = \begin{cases} -x & \text{si } x < 0 \\ x^2 & \text{si } x \geq 0 \end{cases}$$

En cada uno de los casos puedes definir las funciones de forma anónima (con *f=@(x)…*) o mediante un bucle, componente a componente. Para realizar un condicional con la conjunción de dos condiciones escribe:

if ((cond1) & (cond2))… endif

Otra opción para definir funciones a trozos es utilizar lo que se denomina «indexación booleana». Es una opción muy recomendable. Vamos a intentar aproximarnos a esta forma de introducir condiciones.

Escribe en la ventana de comandos *x=linspace(-1,1,5)*. Y después escribe *x(x>0)*, como ves se construye un nuevo vector con las componentes del vector x cuyo valor satisface la condición. Pero hemos perdido otras componentes, por lo, por ejemplo, no podemos conservar de esta forma todas las componentes de x modificando algunas de ellas. Sin embargo, el código siguiente nos permite realizar esta tarea:

[28]:
```
x=linspace(-1,1,5)
y=x;
y(x>0)=5
```

```
x =

  -1.0000  -0.5000         0   0.5000   1.0000

y =

  -1.0000  -0.5000         0   5.0000   5.0000
```

Y ahora podríamos modificar otras componentes, por ejemplo haciendo

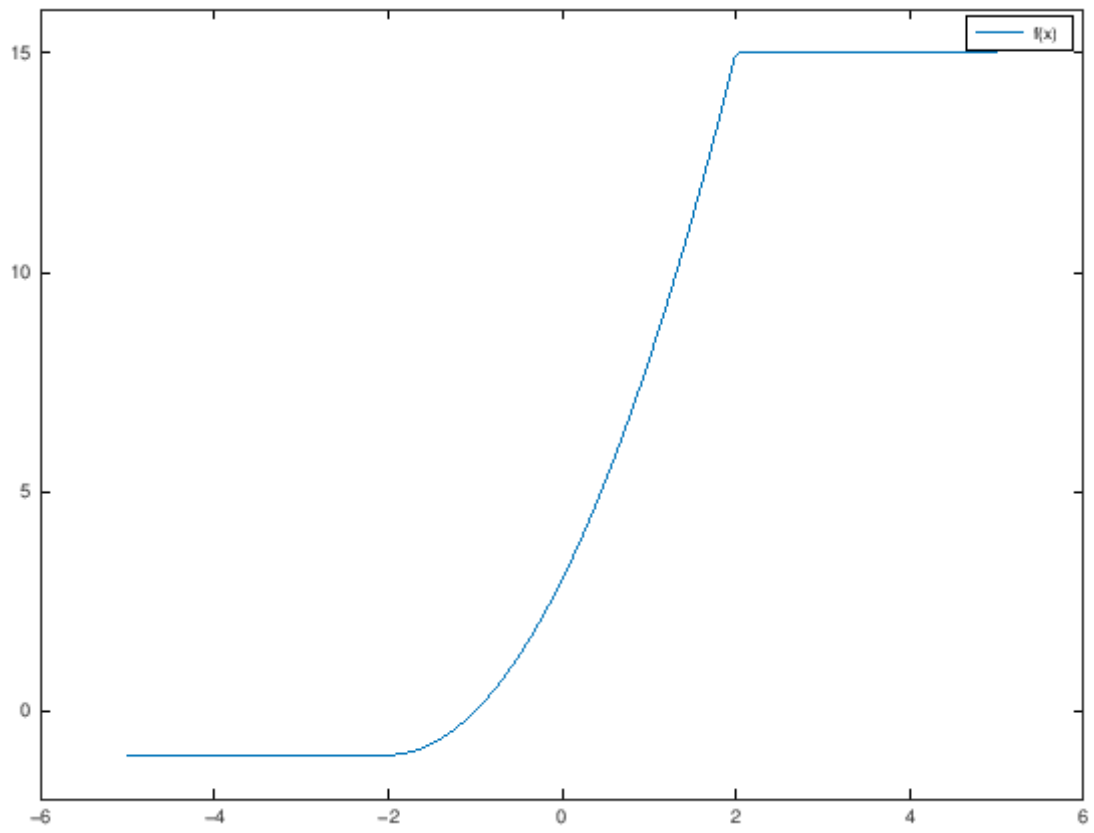
[29]:
```
y(x<0)=x(x<0).^2
```

```
y =

   1.0000   0.2500         0   5.0000   5.0000
```

Modifica alguna de las funciones a trozos en el código del ejercicio usando esta técnica.

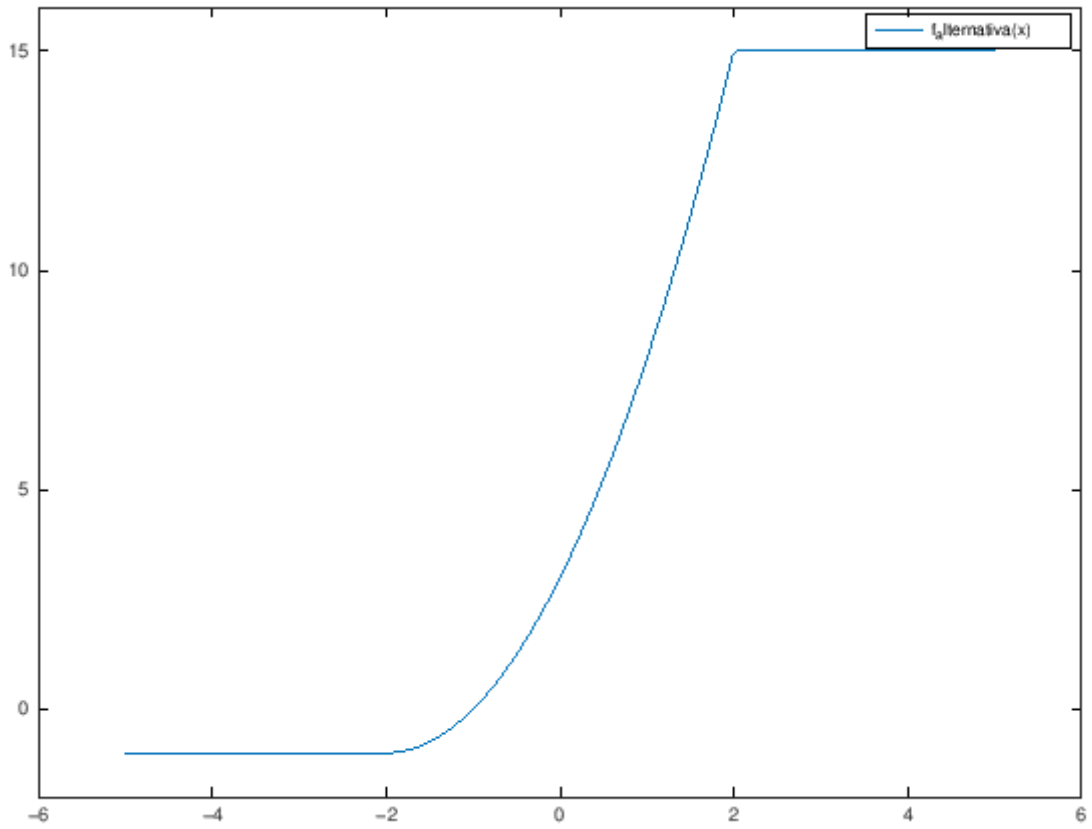
Solución 2.3.3 Empezamos dibujando la función f.

[30]:
```
x = linspace(-5,5,200);
y = x;
y(x<=-2) = -1;
y((x>-2)&(x<2)) = (x((x>-2)&(x<2))+2).^2 - 1;
y(x>=2) = 15;
plot(x, y)
ylim([-2,16])
legend('f(x)')
```



De forma alternativa podemos definir la función f(x) a trozos como:

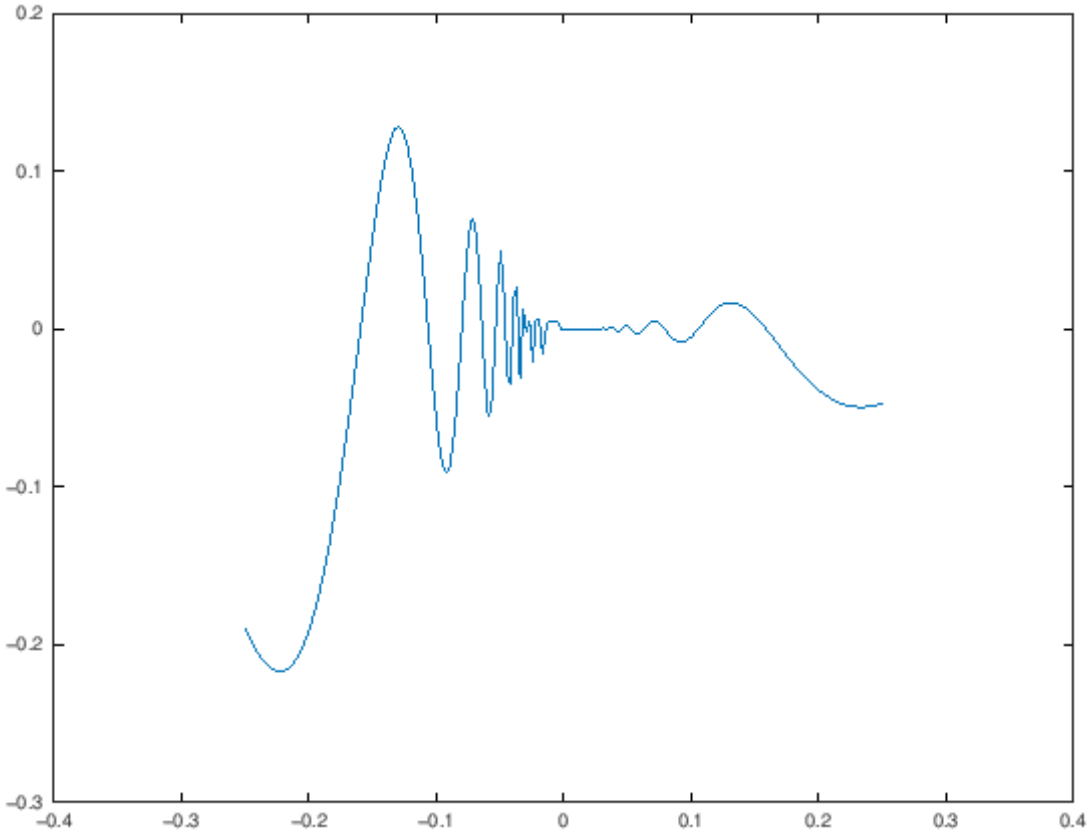
[31]:
```
function ret=f_alternativa(x)

    for i = 1:length(x)
        if (x(i)<= -2)
            ret(i) = -1;
        elseif ((x(i) > -2) & (x(i) < 2))
            ret(i) = ((x(i) + 2).^2) - 1;
        else
            ret(i) = 15;
        endif
    endfor

endfunction

plot(x, f_alternativa(x))
ylim([-2,16])
legend('f_alternativa(x)')
```



Como vemos, es indiferente la forma en que definamos f(x).

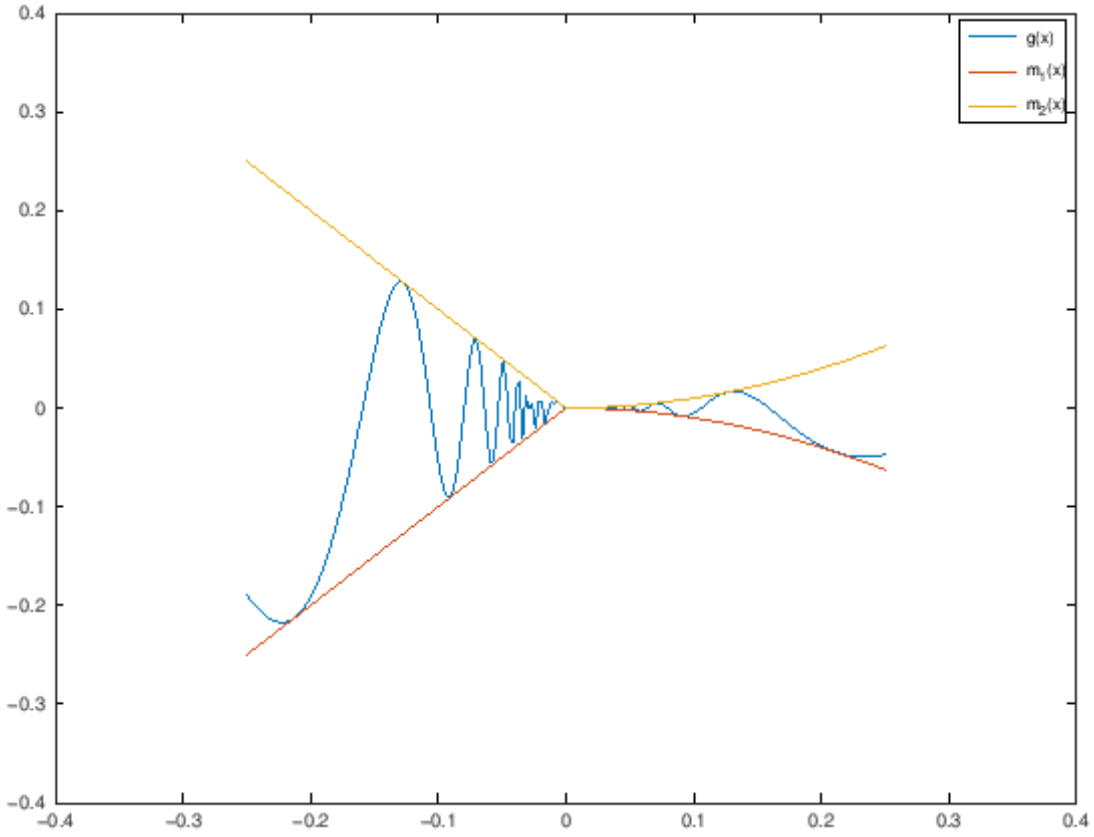
Ahora dibujamos la gráfica de g(x).

[32]:
```
x = linspace(-0.25, 0.25, 200);
y = x;
y(x<0) = x(x<0).*sin(1./(x(x<0)));
%%y(x=0) = 0;
y(x>0) = (x(x>0)).^2 .*sin(1./(x(x>0)));
plot(x, y)
```



Añadimos las funciones moduladoras de g:

[33]:
```
close all
m_1 = x;
m_1(x<0) = x(x<0);
m_1(x>=0) = -x(x>=0).^2;

m_2 = x;
m_2(x<0) = -x(x<0);
m_2(x>=0) = x(x>=0).^2;

plot(x,y)
hold on
plot(x, m_1)
hold on
plot(x, m_2)

legend('g(x)', 'm_1(x)', 'm_2(x)')
```



1.6.3 Ejercicio 3:

En este último ejercicio vamos a manipular funciones de Octave e intentar comprender cómo funcionan para evitar posibles problemas en el futuro.

Recuerda: - Un archivo de función en Octave es un archivo de extensión .m que comienza con la orden function… NombreFuncion… y en ese caso el nombre del archivo debe coincidir con el nombre de la función. El archivo puede contener otras funciones, pero solo la inicial (principal) será accesible desde otros script o funciones (pero serán accesibles desde el propio archivo de función). - Un archivo script también tiene extensión .m, pero no tiene ninguna estructura particular y puede contener una o varias funciones.

Las tareas en este ejercicio serán:

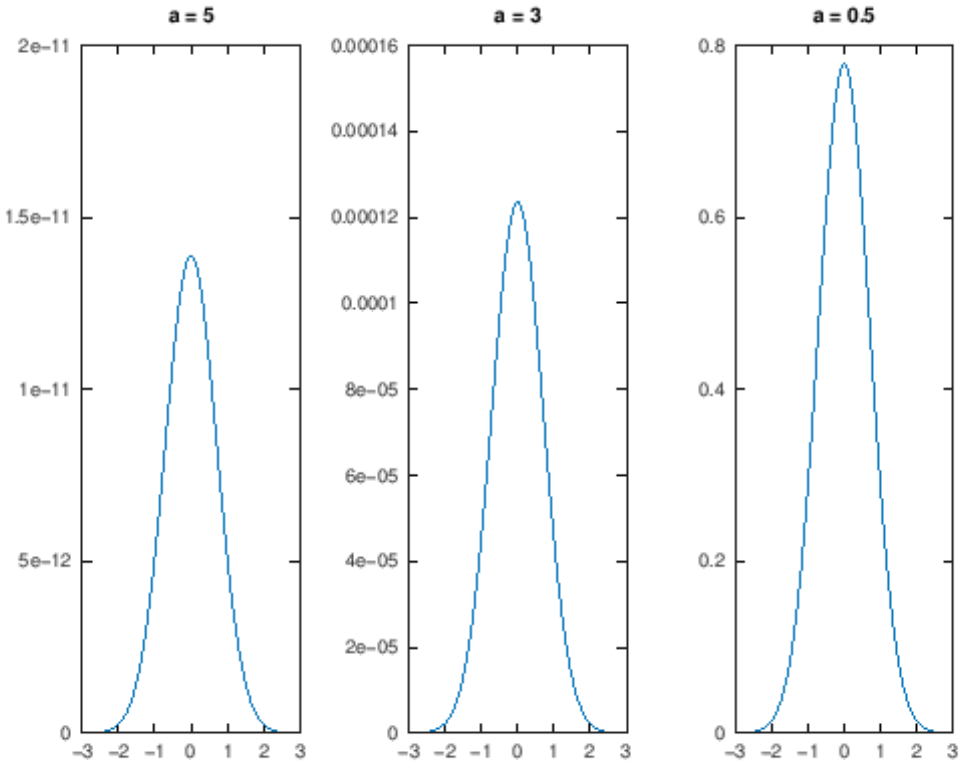
3.1 Definir en el script Ejercicio1.m la función $f(x, a) = exp(-x2) * exp(-a^2)$ mediante una sentencia *function*. En esta función (de dos variables) interpretamos la variable a como parámetro: el objetivo es dibujar la gráfica de $f_a(x) = f(x, a)$ para distintos valores de a y, además, aproximar el valor máximo de f_a en cierto intervalo.

Solución 3.1 Vamos a escribir primero f_a:

[34]:
```
function ret = f_a(x, a)
    ret = exp(-x.^2) * exp(-a.^2);
end
```

Ahora podemos porceder con el resto del ejercicio.

[35]:
```
%Escribir en otro sript de nombre Ejercicio1.
%function ret = f_a(x, a)
%    ret = exp(-x.^2) * exp(-a.^2);
%end

figure(1)
subplot(1,3,1)
x = linspace(-2.5, 2.5, 200);
a = 5;
plot(x, f_a(x, a))
title('a = 5')

subplot(1,3,2)
a = 3;
plot(x, f_a(x, a))
title('a = 3')

subplot(1,3,3)
a = 0.5;
plot(x, f_a(x, a))
title('a = 0.5')
```



3.2 y 3.3 Crearemos en la carpeta Biblioteca el archivo de función **maximum.m** que contendrá una única función: *maximum(func,par,s)*.

La función **maximum** recibirá como parámetro el nombre de una función (variable func, que será el nombre de f), un valor del parámetro a (variable par) y un vector de abscisas en el intervalo $[-5, 5]$ (variable s). La función **maximum** debe definir la función f_a, calcular el máximo de los valores de esta función en el vector de abscisas s y dibujar su gráfica con el mismo vector de abscisas.

Solución 3.2 y 3.3: Primeramente, el código de la función que tendriamos que escribir en su propio script sería:

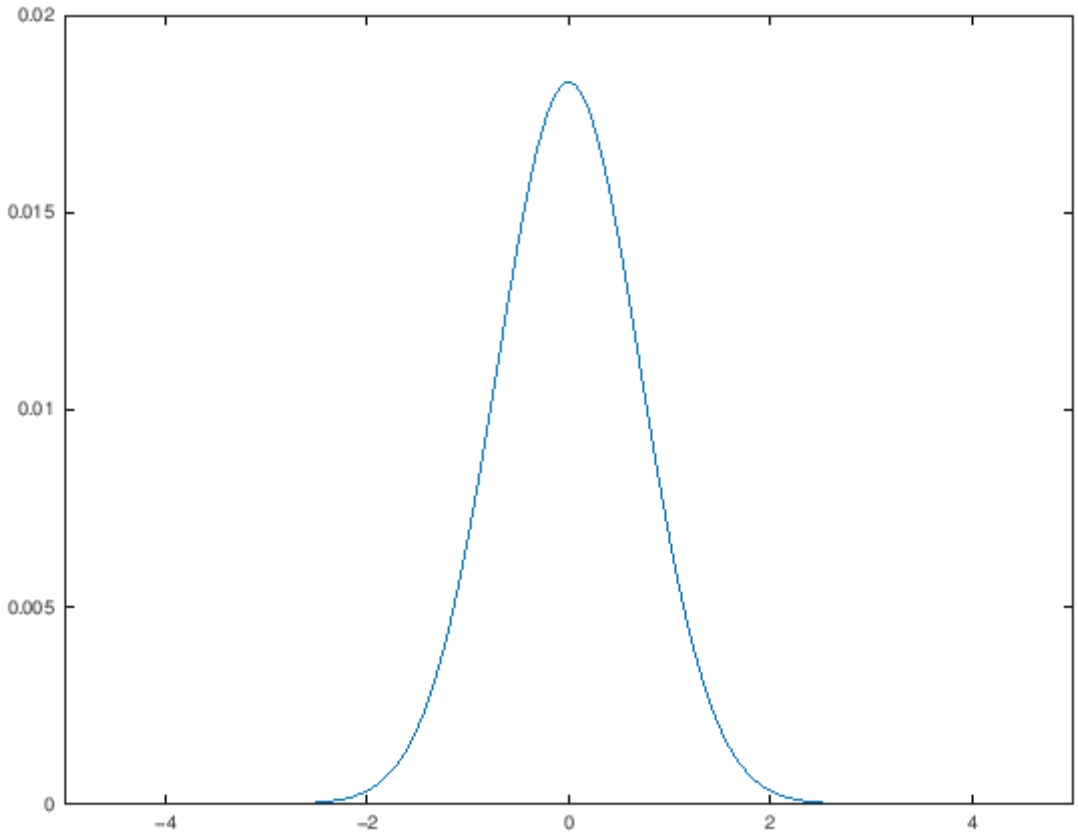
[36]:
```
function ans = maximum(func, par, s)
    a = par;
    max_s = s(1);

    plot(s, func(s, a))
    xlim([-5, 5])

    for i = 1:length(s)
        if func(s(i), a) > func(max_s, a)
        max_s = (s(i));
```

```
        endif
    endfor

    ans = func(max_s, a);
end
```

Entonces, una vez que la hayamos guardado en nuestra carpeta de biblioteca podemos entonces continuar.

[37]:
```
a = 2;
s = linspace(-5, 5, 200);
maximo = maximum(@f_a, a, s)
```

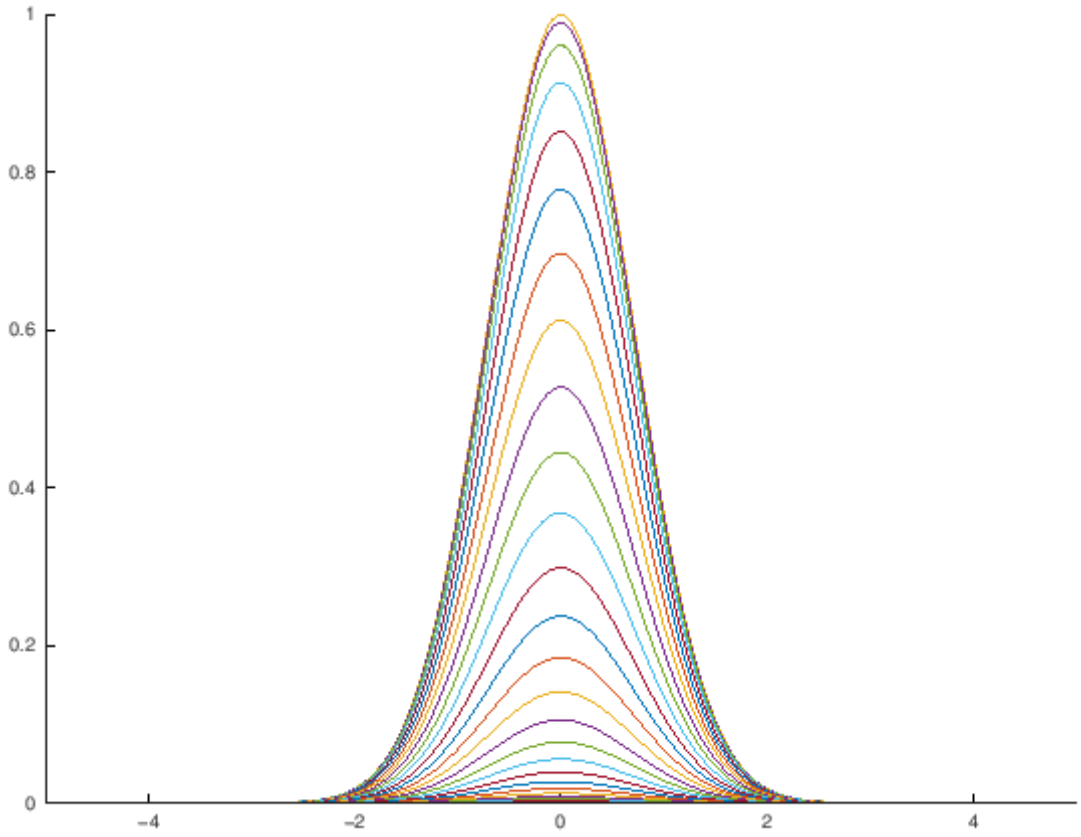
```
maximo = 0.018304
```



3.4 El script *Ejercicio3.m* incluirá un bucle que asigne a valores al parámetro a, variando desde -3 a 3 con incrementos de 0.1, y para cada uno de esos valores hará una llamada a la función *maximum*.

Solución 3.4: *Nota: Realmente e este caso habría que dibujar cada gráfica en una ventana diferente, pero por la cantidad de gráficas que son las he juntado todas en una sola.*

[38]:
```
a = [-3:0.1:3];

figure(1)
for i = 1:length(a)
    hold on
    maximo = maximum(@f_a, a(i), s);
endfor
```



3.5 Una vez conseguido que funcione todo lo anterior introduciremos algunos detalles: las gráficas se deben dibujar todas en la misma ventana, cada una reemplazando a la anterior, manteniendo una escala vertical fija (con *ylim[-0.1,1.1]*) y entre un paso del bucle y otro introduciremos la orden *pause(0.15)*, es decir, una pausa de 0.15 segundos.

Solución 3.5

[39]:
```
figure(1)
for i = 1:length(a)
    hold on
    maximo = maximum(@f_a, a(i), s);
    pause(0.15);
endfor
```

```
ylim([-0.1, 1.1])
```