Tutorial de Octave

basado en "GNU Octave Beginner's Guide"
Jesper Schmidt Hansen, Packt Publishing, 2011

Octave

- Una colección de funciones matemáticas resueltas por métodos numéricos (no simbólicos), con capacidades de dibujar gráficas y su propio lenguaje de programación
- Versión abierta de Matlab
- Disponible para descarga en: http://www.gnu.org/software/octave/
- Paquetes adicionales en: http://octave.sourceforge.net/
- Se puede extender añadiendo nuevos paquetes y funciones enlazadas dinámicamente programadas en C++. API en C++
- Se lanza desde línea de órdenes y aparece un prompt donde podemos evaluar expresiones. Se configura a través de un archivo .octaverc
- Al tratarse de un lenguaje interpretado no es la solución más eficiente al manejar grandes volúmenes de datos
- Armadillo (arma.sourceforge.net) es una biblioteca de funciones matemáticas en C++ que intencionalmente usa una nomenclatura similar a la de Matlab (y Octave), con lo que una vez que se consiga hacer funcionar en Octave, se puede pasar fácilmente a Armadillo

Tipos de datos

Variables

- No se declara el tipo de las variables, que pueden ser números reales, strings, arrays o registros
- En los nombres se distinguen mayúsculas y minúsculas. Variables ya definidas (cuidado con redefinirlas): ans, pi, i ($\sqrt{-1}$) y j ($\sqrt{-1}$)
- who: muestra los nombres de las funciones y variables que hemos definido
- whos: muestra información detallada sobre los nombres de las funciones y variables que hemos definido
- clear *nombre*: elimina del entorno la variable con ese nombre
- clear: elimina del entorno todas las variables

```
octave:> deg = pi/180
deg =  0.017453

octave:> 1.2 * sin(40*deg + log(2.4^2))
ans =  0.76618
```

Números

- Se pueden escribir números complejos: 3+4i
- El número Inf representa el resultado de dividir un número por cero
- El número NaN representa el resultado de cero dividido por cero
- format long y format short permiten cambiar la precisión con la que se muestran los números
- Los números se representan en binario en coma flotante con doble precisión (64 bits), lo que puede dar lugar a algunos problemas de precisión:

```
octave:> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2
ans =  5.5511e-17
```

Definición de vectores

- Se escriben entre corchetes, separados por espacios o comas para vectores fila y por punto y coma para vectores columna
- También se pueden definir a partir de un intervalo min:inc:max con inc=1 si no se especifica

```
octave:> a = [1 4 5]
a =
   1   4   5

octave:> d = [a 6]
d =
   1   4   5   6

octave:> e = 2:6
e =
   2   3   4   5   6

octave:> e = 2:0.3:4
e =
    2.0000    2.3000    2.6000    2.9000    3.2000    3.5000    3.8000
```

Acceso a los elementos de un vector

- Se accede escribiendo el índice entre paréntesis, empezando desde 1. También se puede acceder a subvectores

```
octave:> a = [1:2:6 -1 0]
a =
   1   3   5  -1   0

octave:> a(3)
ans =  5

octave:> a(3:5)
ans =

   5  -1   0

octave:> a(1:2:5)
ans =

   1   5   0
```

```
octave:> a(3:5) = [0 0 0]
a =
   1   3   0   0   0

octave:> a(1:5) = 1
a =
   1   1   1   1   1

octave:> a = [ 0 a 0 ]
a =
   0   1   1   1   1   1   0
```

Matrices

```
octave:> A = [1 2 3; 4 5 6]
A =
   1   2   3
   4   5   6

octave:> A(2,3)
ans = 6

octave:> A(:,2)
ans =
   2
   5

octave:> A(1,:)
ans =
   1   2   3
```

```
octave:> A(2,3) = -10.1
A =
   1.0000   2.0000    3.0000
   4.0000   5.0000  -10.1000

octave:> b = [1, 2, 3]
b =
   1   2   3

octave:> A(2,:) = b
A =
   1   2   3
   1   2   3

octave:> A(:,2) = b
error: A(I,J,...) = X:
dimension mismatch

octave:> A(:,2) = 42
A =
   1   42   3
   1   42   3
```

Funciones de creación de vectores y matrices

`zeros`(M, N)	Create a matrix where every element is zero. For a row vector of size n, set $M = 1, N = n$
`ones`(M, N)	Create a matrix where every element is one. For a row vector of size n, set $M = 1, N = n$
`linspace`$(x1, x2, N)$	Create a vector of N elements, evenly spaced between $x1$ and $x2$
`logspace`$(x1, x2, N)$	Create a vector of N elements, logarithmically spaced between 10^{x1} and 10^{x2}

```
octave:> b = linspace(-2.1, 0.5, 7)
b =
  -2.1000  -1.6667  -1.2333  -0.8000  -0.3667   0.0667   0.5000

octave:> zeros(1, 4)
ans = 0  0  0  0

octave:> eye(3)
ans =
Diagonal Matrix
  1  0  0
  0  1  0
  0  0  1
```

Matrices

```
octave:> D = [1 2 3];

octave:> D = [D ; 4 5 6];

octave:> D = [D ; 7 8 9]
D =
   1   2   3
   4   5   6
   7   8   9

octave:> C = [ 1:3; 8:-2:4 ]
C =
   1   2   3
   8   6   4
```

```
octave:> B = [ 2 0; 0 -1; 1 0 ];

octave:> A = [ 5 7 9; -1 3 -2 ];

octave:> comp = [ eye(3) B; A zeros(2) ]
comp =
   1   0   0   2   0
   0   1   0   0  -1
   0   0   1   1   0
   5   7   9   0   0
  -1   3  -2   0   0
```

Variables de texto

- Las cadenas se tratan como arrays de caracteres

```
octave:> t = "Hello"
t = Hello

octave:> t(2)
ans = e

octave:> t(2:4)
ans = ell

octave:> t = [t " World"]
t = Hello World

octave:> T= ["Hello" ; "George"]
T =
  Hello
  George

octave:> T(1,6)
ans =
```

Estructuras

- Se usan con la sintaxis nombre_estructura.nombre_campo
- Se implementan como tablas hash

```
octave:> projectile.mass = 10.1
projectile =
{
mass = 10.100
}
```

```
octave:> projectile.velocity = [1 0 0]
projectile =
{
mass = 10.100
velocity =
  1  0  0
}
```

```
octave:> projectile.type = "Cannonball"
projectile =
{
mass = 10.100
velocity =
  1  0  0
type = Cannonball
}
```

```
octave:> projectile = struct("mass", 10.1,
"velocity", [1 0 0], "type", "Cannonball");
```

Estructuras

```
octave:> projectile.velocity(2) = -0.1
projectile =
{
mass = 10.100
velocity =
  1  -0.1  0
type = Cannonball
}
```

```
octave:> s(1) = projectile;

octave:> s(2) = projectile;

octave:> s(2).type
ans = Cannonball

octave:> s(2) = setfield(s(2), "type",
"Cartridge");

octave:> getfield(s(2), "type")
ans = Cartridge
```

```
octave:> projectiles = struct("type1", s(1), "type2", s(2));

octave:> projectiles.type2.type
ans = Cartridge
```

Arrays de celdas

- Arrays que permiten almacenar datos de distinto tipo

```
octave:> projectile = {10.1, [1 0 0], "Cannonball"}
projectile =
{
   [1,1] = 10.1
   [1,2] =
1  0  0
   [1,3] = Cannonball
}

octave:> projectile{2}
ans =
  1  0  0

octave:> projectiles = {10.1, [1 0 0], "Cannonball"; 1.0, [0 0 0],
"Cartridge”}
```

Información de las variables del entorno

```
octave:48>whos

Variables in the current scope:

 Attr    Name          Size        Bytes       Class
 ====    ====          ====        =====       =====
         A             2x2         32          double
         B             2x2x2       128         double
         T             2x6         12          char
         Z             2x2         64          double
         a             1x1         8           double
         ans           1x9         9           char
         b             1x5         40          double
         c             3x1         24          double
         d             1x12        96          double
         projectile    1x3         42          cell
         projectiles   2x3         83          cell
         s             1x2         83          struct
         t             1x11        11          char
         z             1x1         16          double

Total is 81 elements using 648 bytes
```

Información sobre las variables

```
octave:> size(A)
ans =
  2  2
octave:> rows(A)
ans = 2

octave:> columns(A)
ans = 2

octave:> length(c)
ans = 3

octave:> length(T)
ans = 6
```

```
octave:> isscalar(a)
ans = 1

octave:> isvector(b)
ans = 1

octave:> ismatrix(b)
ans = 1

octave:> typeinfo(b)
ans = matrix

octave:> is<Tab><Tab>
...
```

Aritmética y operadores relacionales

Aritmética

- Los operadores +, -, *, / y ^ se aplican sobre números, vectores y matrices con la semántica habitual

```
octave:> clear

octave:> a = 2; b=[1 2 3]; c=[1; 2; 3]; A=[1 2 3; 4 5 6];

octave:> a+a
ans = 4

octave:> a+b
ans =
  3  4  5

octave:> b+b
ans =
  2  4  6

octave:> b+c
error: operator +: nonconformant arguments (op1 is 1x3, op2 is 3x1)
```

Aritmética

```
octave:> clear
octave:> a = 2; b=[1 2 3]; c=[1; 2; 3]; A=[1 2 3; 4 5 6];

octave:> a*a
ans = 4

octave:> a*b
ans =
   2   4   6

octave:> b*b
error: operator *: nonconformant arguments (op1 is 1x3, op2 is 1x3)

octave:> b*c
ans = 14

octave:> A=[1 2; 3 4]; B=[1 2 3; 4 5 6];

octave:> A*B
ans =
   9  12  15
  19  26  33

octave:> B*A
error: operator *: nonconformant arguments (op1 is 2x3, op2 is 2x2)
```

Operaciones elemento a elemento

- Se incluyen también operadores aritméticos que se aplican elemento a elemento: .+, .-, ./, .*, .^, que se aplican entre una matriz y un número o entre dos matrices de las mismas dimensiones

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} .* \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \end{pmatrix}$$

```
octave:> B=[1 2 3; 4 5 6]

octave:> B.+B
ans =
   2   4   6
   8  10  12

octave:> B.^2.1
ans =
   1.0000    4.2871   10.0451
  18.3792   29.3655   43.0643
```

División

- La división entre matrices no tiene sentido. En Octave se utiliza para resolver sistemas de ecuaciones lineales

$$\begin{aligned} 2x_1 + x_2 - 3x_3 &= 1 \\ 4x_1 + 2x_2 - 2x_3 &= 1 \\ -x_1 + x_2/2 - x_3/2 &= 1.5 \end{aligned} \qquad \mathbf{Ax} = \mathbf{y} \qquad \mathbf{A} = \begin{bmatrix} 2 & 1 & -3 \\ 4 & 2 & -2 \\ -1 & 1/2 & -1/2 \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ and } \mathbf{y} = \begin{bmatrix} 1 \\ 3 \\ 3/2 \end{bmatrix}. \qquad \mathbf{x} = \mathbf{A}^{-1}\mathbf{y}.$$

- División por la izquierda en Octave: A\y $\equiv A^{-1}y$
- División por la derecha en Octave: A/y $\equiv yA^{-1}$

```
octave:> A=[2 1 -3; 4 -2 -2; -1 0.5 -0.5]; y = [1; 3; 1.5];

octave:> A\y
ans =
  -1.6250
  -2.5000
  -2.2500
```

```
octave:> A/y
error: operator /: nonconformant arguments
(op1 is 3x3, op2 is 3x1)
octave> A/y'
ans =
   0.040816
  -0.408163
  -0.020408
```

Operadores relacionales

- Los operadores relacionales son: ==, >, <, >=, <= y != con la semántica habitual
- Cuando se aplican entre matrices o vectores siempre operan elemento a elemento y devuelven matrices o vectores de la misma dimensión

```
octave> A=[2 2 -2; 1 1 2; -1
0 0]

octave> A(2,1) == 1
ans = 1

octave> A(2,1) == 2
ans = 0

octave> A(:,1) >= [2; 1; 0]
ans =
  1
  1
  0
```

```
octave> "a"=="a"
ans = 1

octave> "hello"=="henno"
ans = 1 1 0 0 1

octave>strcmp("hello", "helloo")
ans = 0
```

Funciones predefinidas

Funciones

- Invocación de funciones

  nombre_función (input1, input2, …)

  output = nombre_función (input1, input2, …)

  [output1, output2, …] = nombre_función (input1, input2, …)

- Las funciones de Octave normalmente se pueden aplicar sobre un número o sobre una matriz de números, en cuyo caso aplican la función a cada valor y devuelven una matriz con los resultados

```
octave:> cos(pi)
ans = -1

octave:> x = [0:pi/2:2*pi]
x =
   0.00000   1.57080   3.14159   4.71239   6.28319

octave:> cos(x)
ans =
   1.0000e+00   6.1232e-17  -1.0000e+00  -1.8370e-16   1.0000e+00
```

Funciones matemáticas básicas

`cos`	Cosine of an angle (in radians)
`sin`	Sine of an angle (in radians)
`tan`	Tangent of an angle (in radians)
`exp`	Exponential function (e^x)
`log`	Natural logarithm (NB this is $\log_e$, not $\log_{10}$)
`log10`	Logarithm to base 10
`sinh`	Hyperbolic sine
`cosh`	Hyperbolic cosine
`tanh`	Hyperbolic tangent
`acos`	Inverse cosine
`acosh`	Inverse hyperbolic cosine
`asin`	Inverse sine
`asinh`	Inverse hyperbolic sine
`atan`	Inverse tangent
`atan2`	Two-argument form of inverse tangent
`atanh`	Inverse hyperbolic tangent
`abs`	Absolute value
`sign`	Sign of the number (-1 or $+1$)
`round`	Round to the nearest integer
`floor`	Round down (towards minus infinity)
`ceil`	Round up (towards plus infinity)
`fix`	Round towards zero
`rem`	Remainder after integer division

Es importante que nuestras funciones se puedan aplicar de la misma forma a escalares y matrices

```
octave:> x = 0.5;

octave:> f = exp(-5*sqrt(x))*sin(2*pi*x)
f =  3.5690e-18

octave:> x = [0:0.1:1];

octave:> f = exp(-5*sqrt(x)).*sin(2*pi*x)
f =

 Columns 1 through 7:

   0.00000   0.12093   0.10165   0.06150   0.02488   0.00000  -0.01222

 Columns 8 through 11:

  -0.01450  -0.01086  -0.00512  -0.00000
```

Utilidades

- Distintas funciones para generar (matrices de) números aleatorios, utilizando distintas distribuciones de probabilidad: rand, randn, randg, rande, randp

```
octave:> A = rand(3,5)
A =
   0.891418   0.274142   0.255747   0.613664   0.880733
   0.893423   0.502311   0.078323   0.388216   0.520998
   0.675750   0.023727   0.179975   0.068018   0.973729
```

- min y max para obtener el mínimo y el máximo de un vector. Si se aplican sobre una matriz, obtienen el resultado columna por columna

```
octave:> min(A)
ans =
   0.675750   0.023727   0.078323   0.068018   0.520998

octave:> min(ans)
ans =  0.023727

octave:> min(ans)
ans =  0.023727
```

Utilidades

sort ordena vectores y matrices, columna a columna

```
octave:> sort(A)
ans =

   0.675750   0.023727   0.078323   0.068018   0.520998
   0.891418   0.274142   0.179975   0.388216   0.880733
   0.893423   0.502311   0.255747   0.613664   0.973729
```

o fila por fila (help sort lo explica todo)

```
octave:> [S i] = sort(A, 2, "descend")
S =
   0.891418   0.880733   0.613664   0.274142   0.255747
   0.893423   0.520998   0.502311   0.388216   0.078323
   0.973729   0.675750   0.179975   0.068018   0.023727

i =
   1   5   4   2   3
   1   5   2   4   3
   5   1   3   4   2
```

Utilidades

- find, any, all: búsquedas
- floor, ceil, round, fix, abs: redondeos
- sum, prod: sumas y productos acumulados

```
octave:> find( [ 1 2 3 4 ] < [ 3 3 3 3 ] )
ans =

   1   2

octave:> length( find( [ 1 0 0 1 1 ] == 1 ) )
ans =  3

octave:> sum([1 2; 3 4])
ans =
   4   6

octave:> prod([1 2 3 4])
ans =  24
```

Algebra de matrices

`inv`	Inverse of a matrix
`det`	Determinant of a matrix
`trace`	Trace of a matrix
`eig`	Calculate the eigenvectors and eigenvalues of a matrix
`rank`	Calculate an estimate of the rank of a matrix
`null`	Calculate a basis for the null space of a matrix
`rref`	Perform Gaussian elimination on an augmented matrix
`lu`	Calculate the LU decomposition of a matrix
`qr`	Calculate the QR decomposition of a matrix
`svd`	Calculate the SVD of a matrix
`pinv`	Calculate the pseudoinverse of a matrix

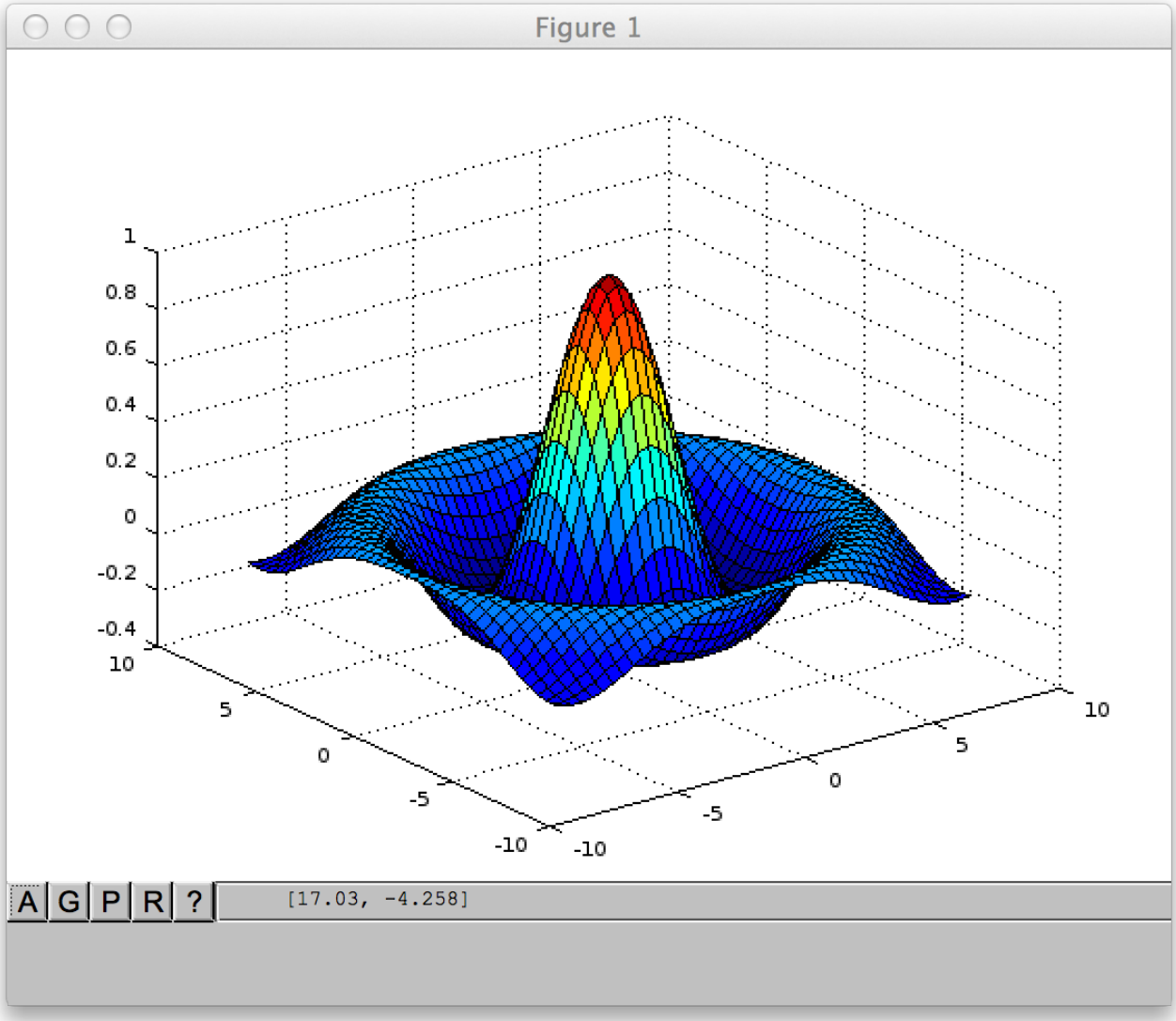
Gráficos en 2D

Gráficos en 2D

- Se incluyen dos toolkits gráficos: FLTK y GNU Plot
- En Mac, si usa GNU Plot es necesario añadir esta línea al archivo .octaverc: setenv GNUTERM x11

```
octave:> available_graphics_toolkits
ans =
{
  [1,1] = fltk
  [1,2] = gnuplot
}

octave:> graphics_toolkit("gnuplot")

octave:> sombrero()

gnuplot> set terminal aqua enhanced title "Figure 1" size 560 420  font "*,6"
                      ^
         line 0: unknown or ambiguous terminal type; type just 'set terminal'
for a list

octave:> graphics_toolkit("fltk")

octave:> sombrero()
```

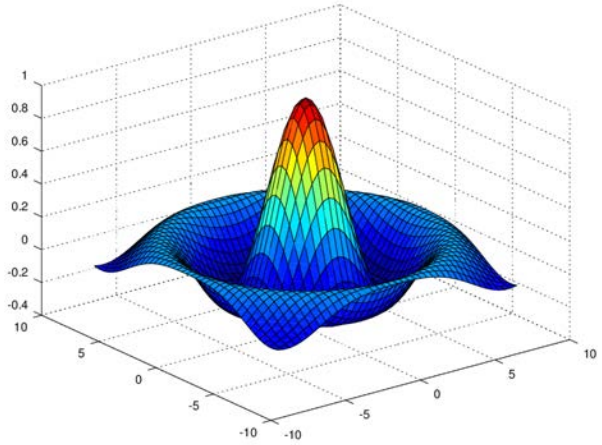
ventana que permite interactuar con la gráfica:



print permite guardar en un archivo el contenido de la ventana con la figura

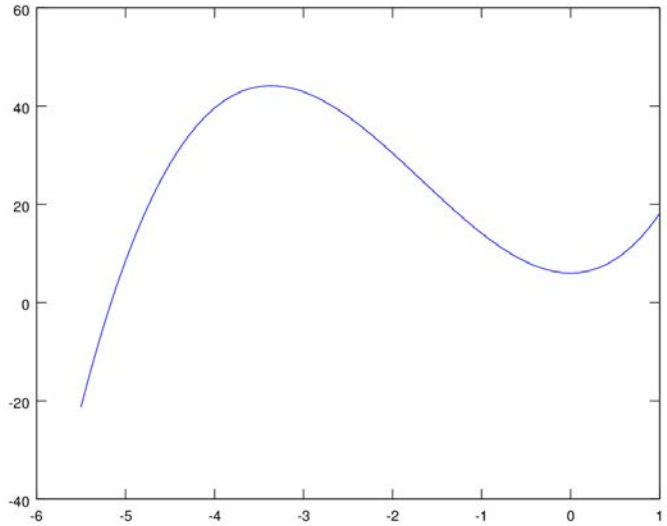
```
octave:> print("sombrero.png", "-dpng")
```

eps	Encapsulated PostScript (I recommend this format if your text program or printer supports it).
ps	PostScript.
pdf	Portable Document Format.
jpg/jpeg	Joint Photographic Experts Group image.
gif	Graphics Interchange Format image.
tex	TeX picture (to be included in a TeX document).
pslatex	LaTex picture file for labels and PostScript for the graphics. This enables you to edit the labels later.
png	Portable Network Graphics image.



plot recibe dos vectores, uno con las X y otro con las Y

```
octave:> x = [-5.5:0.1:1]; f = 2*x.^3 + 10.1*x.^2 + 6;

octave:> plot(x, f)
```

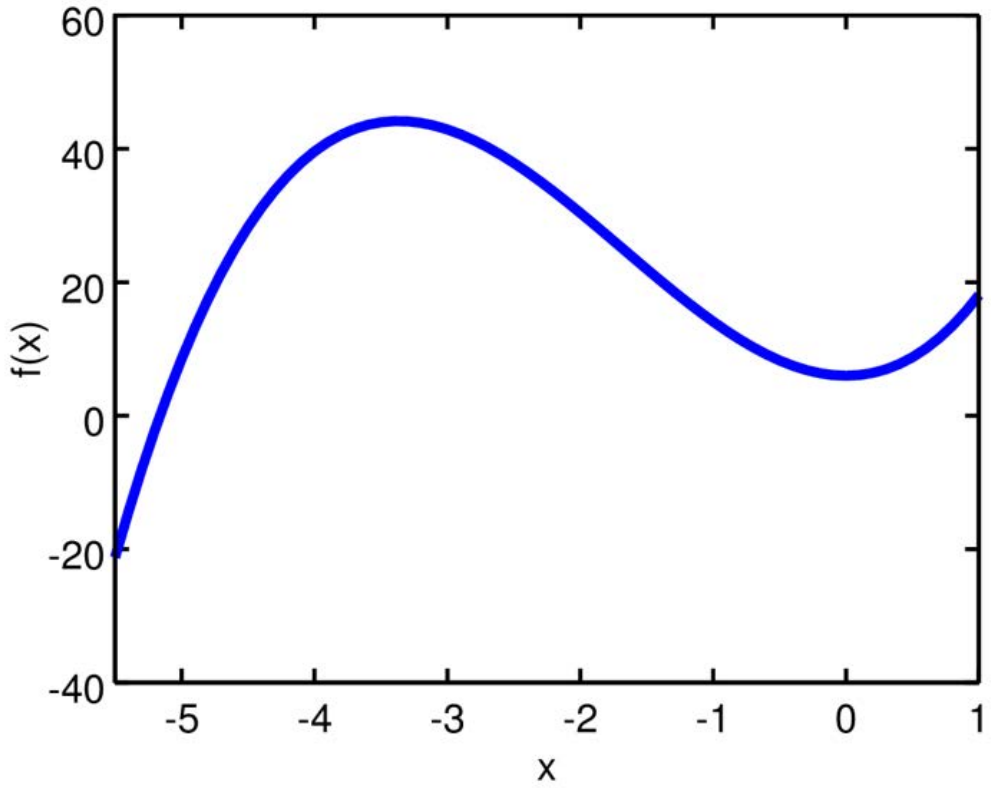


hay dos formas de modificar los atributos de una gráfica

```
plot(x, y, fmt, property, value, ...)
```

```
set(handle, property, value, ...)
```

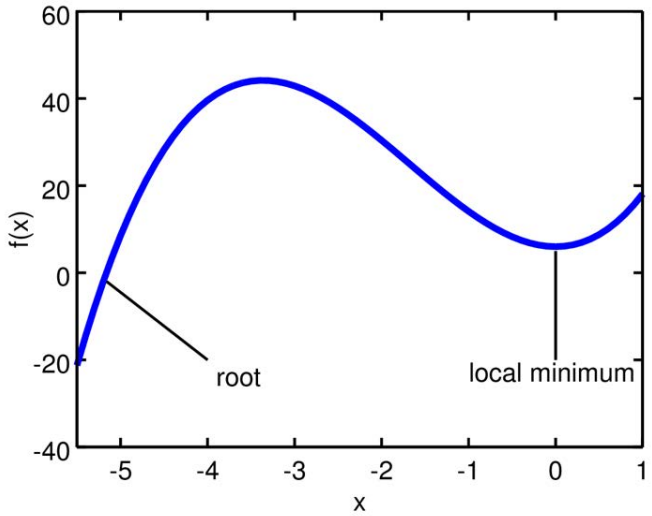
```
octave:> plot(x, f, "linewidth", 5)

octave:> set(gca, "xlim", [-5.5 1])

octave:> set(gca, "linewidth", 2)

octave:> set(gca, "fontsize", 20)

octave:> xlabel("x", "fontsize", 20)

octave:> ylabel("f(x)", "fontsize", 20)
```



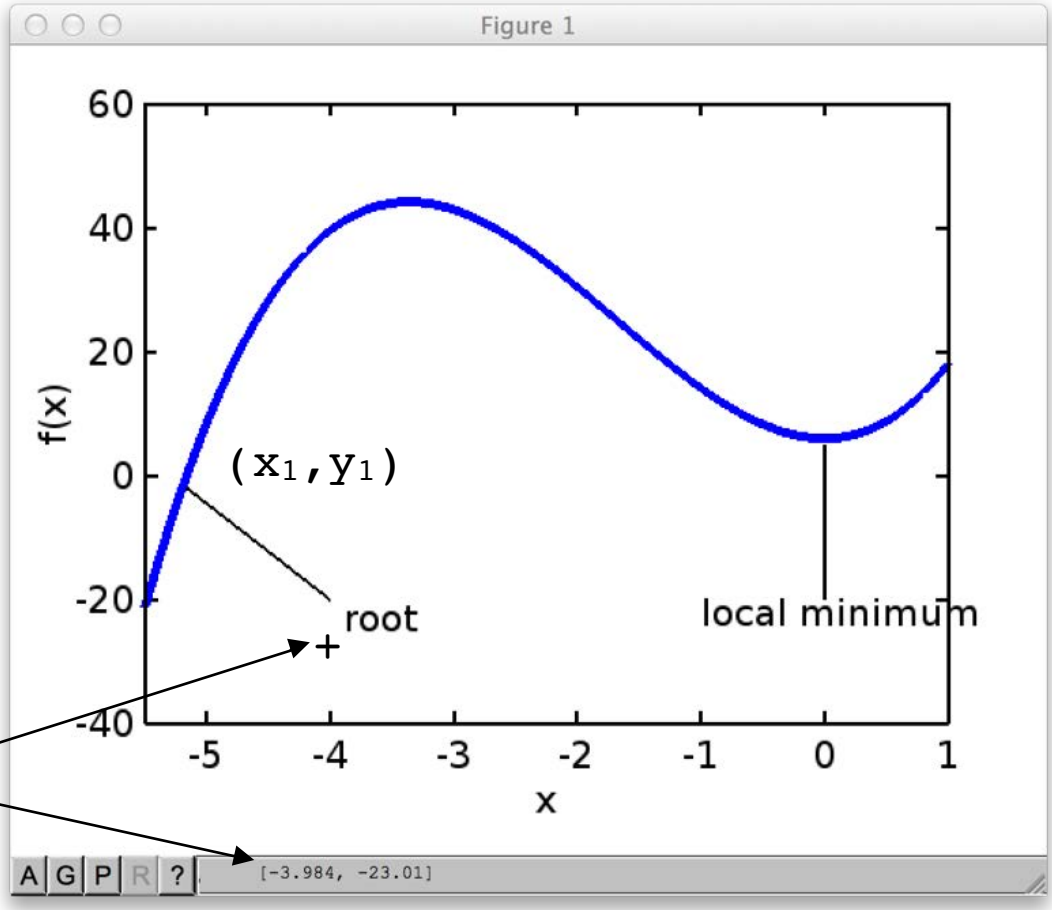

```
gca ()
     Return a handle to the current axis object. If no axis object
     exists, create one and return its handle. The handle may then be
     used to examine or set properties of the axes.
```

```
octave:> line([-5.16 -4], [-2 -20], "linewidth", 2)

octave:> text(-3.9, -23, "root", "fontsize", 20);

octave:> line([0 0], [5 -20], "linewidth", 2)

octave:> text(-1.0, -22, "local minimum", "fontsize", 20)
```

line([x_1 x_2 ...], [y_1 y_2 ...])

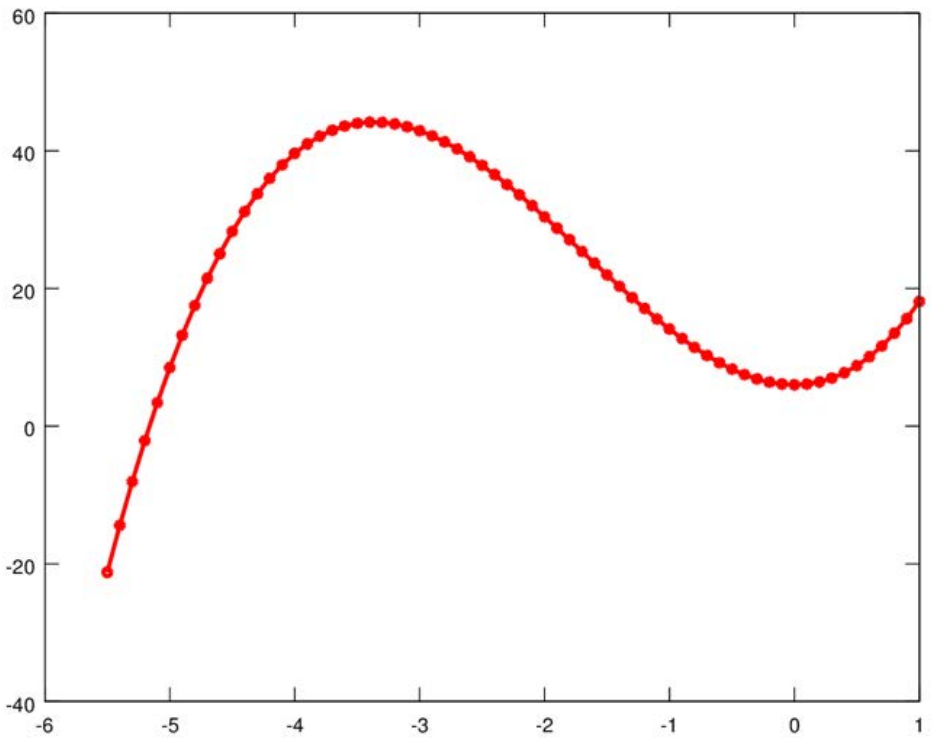


[-3.984, -23.01]



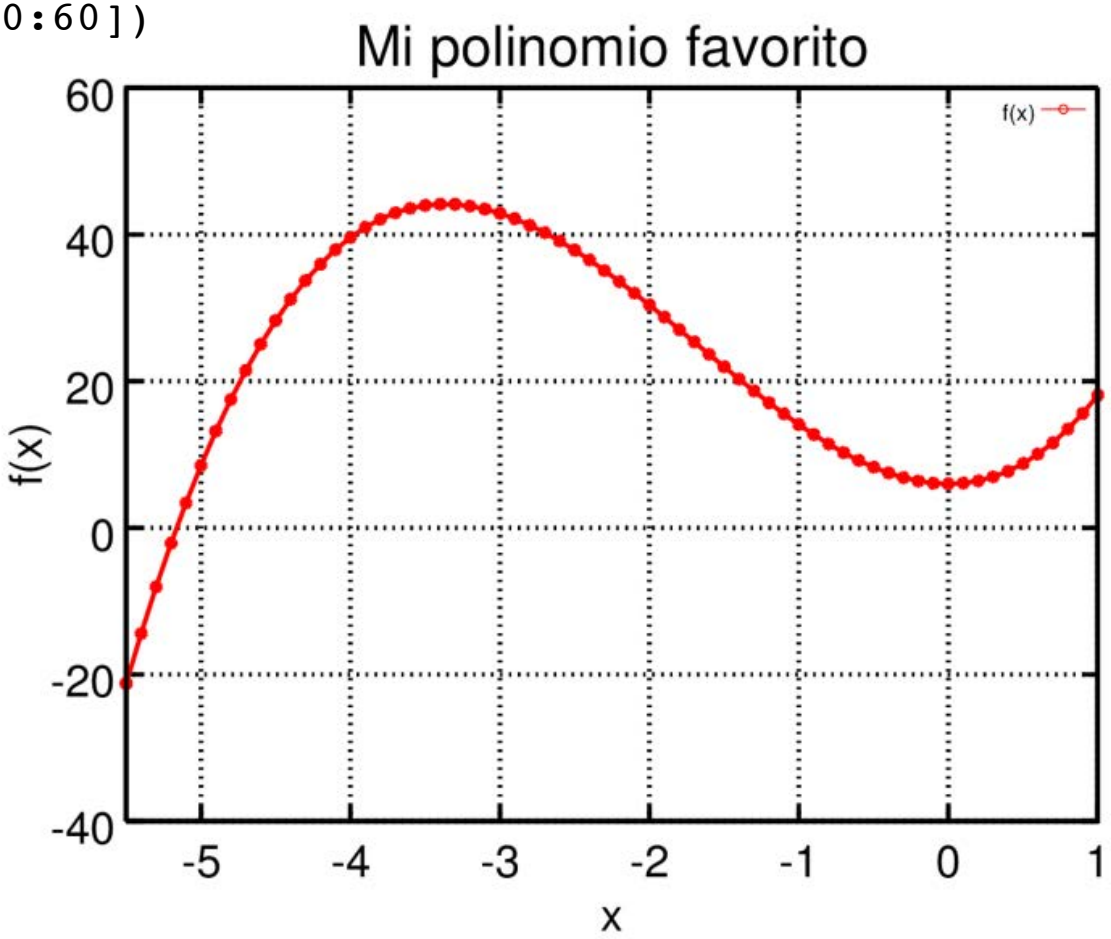
Parámetro de formato en plot

color		marcador		línea	
`w`	whitew	`.`	point	`-`	solid
`m`	magenta	`o`	circle	`:`	dotted†
`c`	cyan	`x`	x-mark	`-.`	dashdot†
`r`	red	`+`	plus	`--`	dashed†
`g`	green	`*`	star		
`b`	blue	`s`	square†		
`y`	yellow†	`d`	diamond†		
`k`	black†	`v`	triangle (down)†		
		`^`	triangle (up)†		
		`<`	triangle (left)†		
		`>`	triangle (right)†		
		`p`	pentagram†		
		`h`	hexagram†		




```
plot(x, f, "ro-", "markersize", 4, "linewidth", 2)
```

```
octave:> set(gca, "xlim", [-5.5 1], "ylim", [-40 60], "linewidth", 2,
"fontsize", 20)

octave:> xlabel( "x", "fontsize", 20)

octave:> ylabel( "f(x)", "fontsize", 20)

octave:> legend("f(x)")

octave:> title( "Mi polinomio favorito","fontsize", 25 )

octave:> set(gca, "ytick", [-40:20:60])

octave:> grid on
```

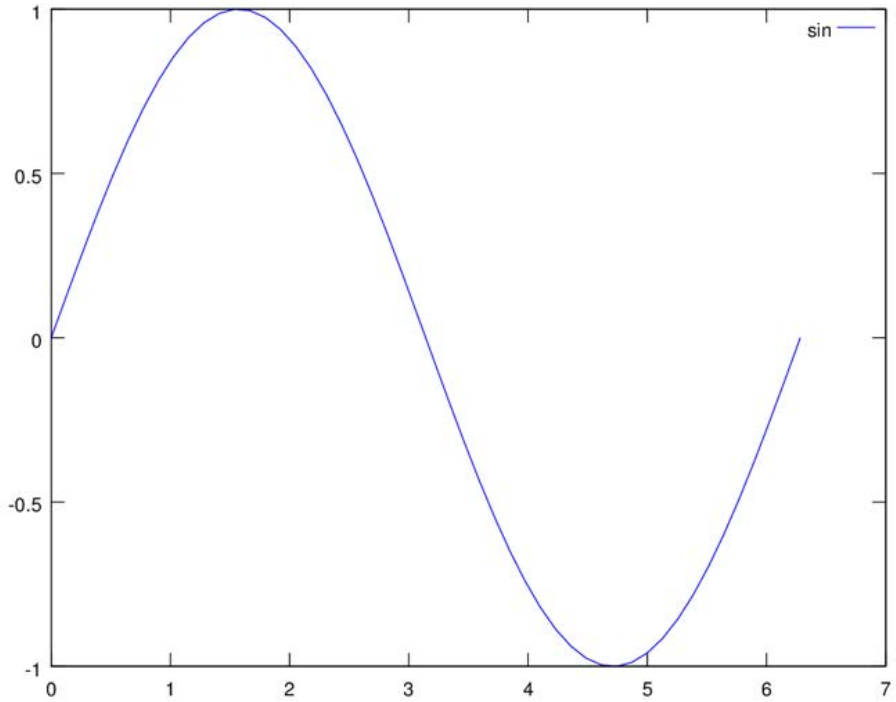


- *figure*: abre otra ventana donde se enviarán las siguientos gráficas. Para cambiar la figura activa se invoca *figure* con un número de orden: figure(1), figure(2), ... *gcf* devuelve el número que identifica a la ventana activa

- *clf*: vacía la ventana de la figura activa

- *fplot*: dibuja la gráfica de una función dada, en el intervalo especificado y con el número de puntos que se le indique

```
octave:> clf
octave:> fplot( "sin", [0 2*pi], 50 )
```

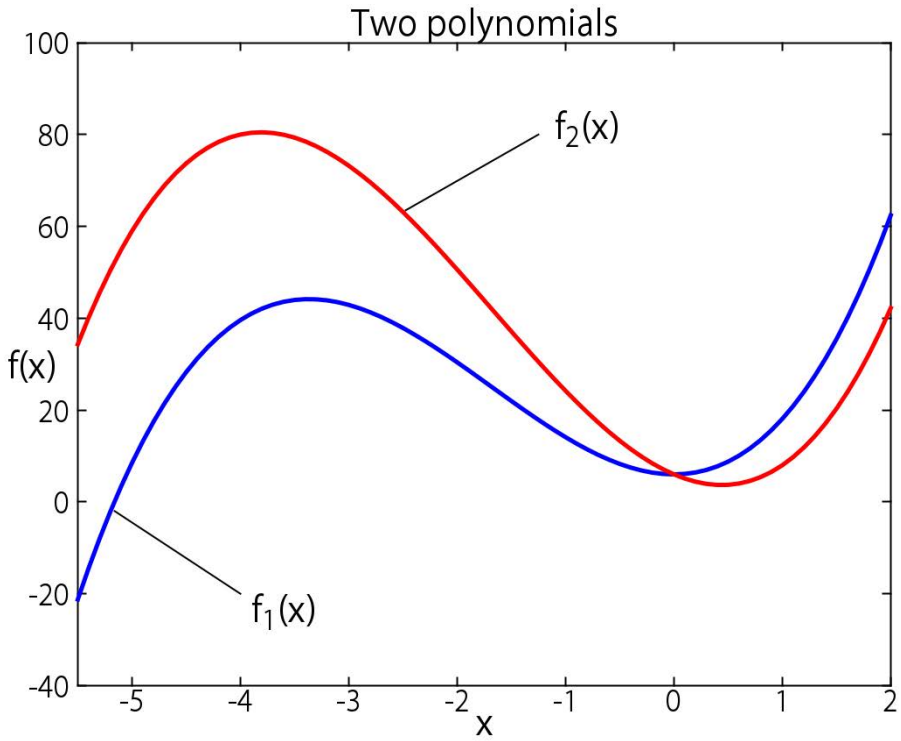


Varias gráficas en la misma figura

- Se pueden pasar varias parejas de vectores X Y a la función *plot* o invocar *hold on* entre llamadas a *plot*

$$f_1(x) = 2x^3 + 10.1x^2 + 6 \qquad f_2(x) = 2x^3 + 10.1x^2 - 10.1x + 6$$

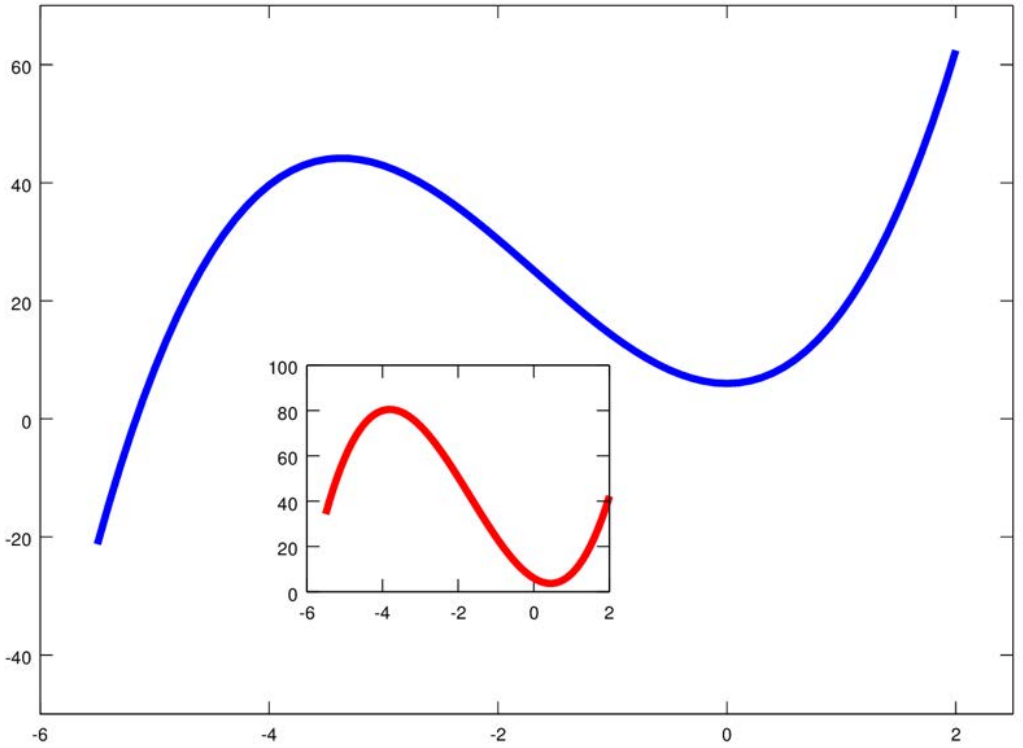
```
octave:> x = [-5.5:0.1:2]; c_1 = [2 10.1 0 6];
octave:> c_2 = [2 10.1 -10.1 6];
octave:> f_1 = polyval(c_1, x); f_2 = polyval(c_2, x);
octave:> plot(x, f_1, "linewidth", 5, x, f_2, "linewidth", 5,
"color", "red")
octave:> text(-3.9, -23, "f_1(x)")
...
```



Varias gráficas en la misma figura

- *subplot* permite dibujar varias gráficas en distintas zonas de la misma figura y *axes* permite especificar la posición y el tamaño de la gráfica contenida, como porcentaje de las dimensiones de la gráfica continente

```
octave:> subplot(1,1,1)
octave:> plot(x, f_1, "linewidth", 4)
octave:> set(gca, "xlim", [-6 2.5], "ylim", [-50 70])
octave:> axes("position", [0.3 0.2 0.3 0.3])
octave:> plot(x, f_2, "color", "red", "linewidth", 4)
```



Gráficos en 3D

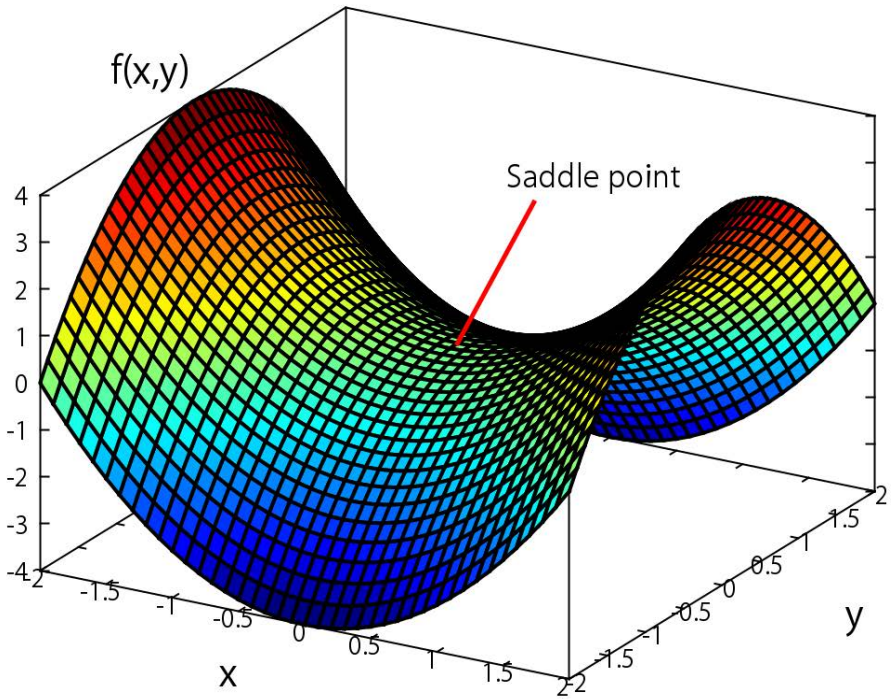
Gráficos en 3D

- Primero se genera una matriz de puntos en el plano XY y luego se calcula el valor z para cada punto x y
- Dados dos vectores *x*, *y* *meshgrid* genera sendas matrices con las coordenadas de todos los puntos que se pueden formar tomando una coordenada de *x* y otra de *y*
- *surface* genera una gráfica en tres dimensiones a partir de las matrices con las coordenadas XY Z

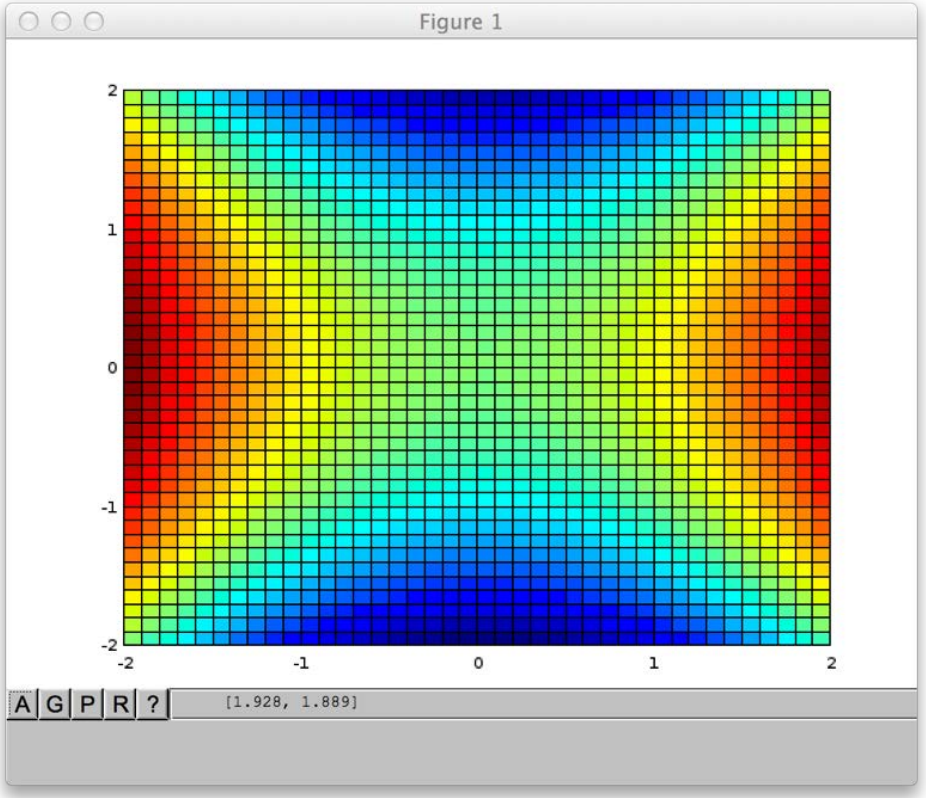
```
octave:> x = [1 2]; y = [3 4];
octave:>[X Y] = meshgrid(x,y)
X =

   1   2
   1   2

Y =

   3   3
   4   4
```

$$f(x,y) = x^2 - y^2$$

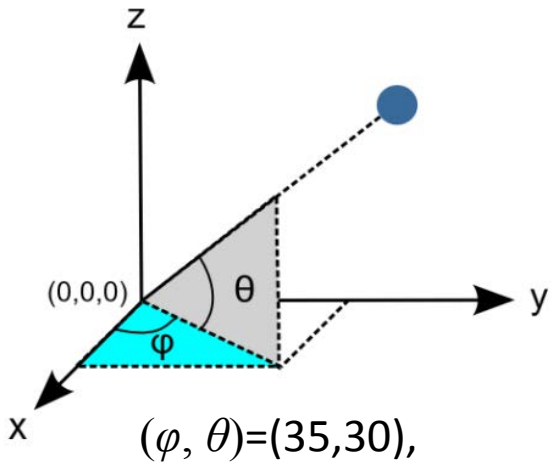
```
octave:> x = [-2:0.1:2]; y = x;
octave:> [X Y] = meshgrid(x,y);
octave:> Z = X.^2 - Y.^2;
octave:> surface(X, Y, Z)
octave:> set(gca, "linewidth", 2, "fontsize", 20, "xlim", [-2 2])
octave:> xlabel("x", "fontsize", 25)
octave:> ylabel("y", "fontsize", 25)
octave:> text(-3.2, 1, 3, "f(x,y)", "fontsize", 25)
octave:> line([0 0], [0 1], [0 2], "linewidth", 5, "color", "red")
octave:> text(-0.5, 1.5, 1.8, "Saddle point", "fontsize", 20)
```



en realidad se vería así:



es necesario rotar la vista con la función *view*

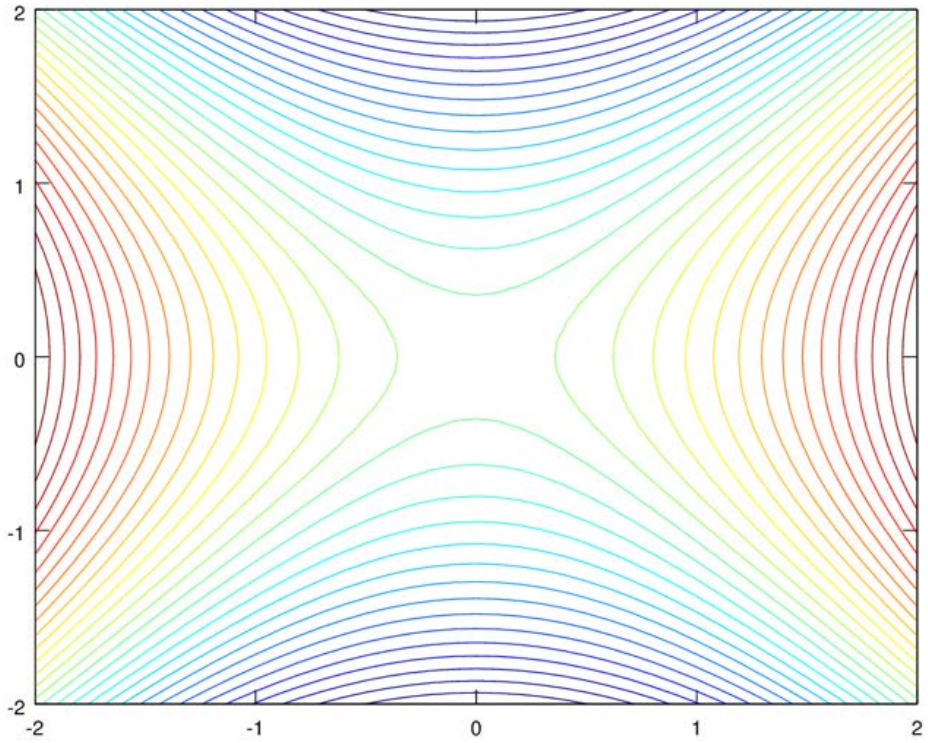


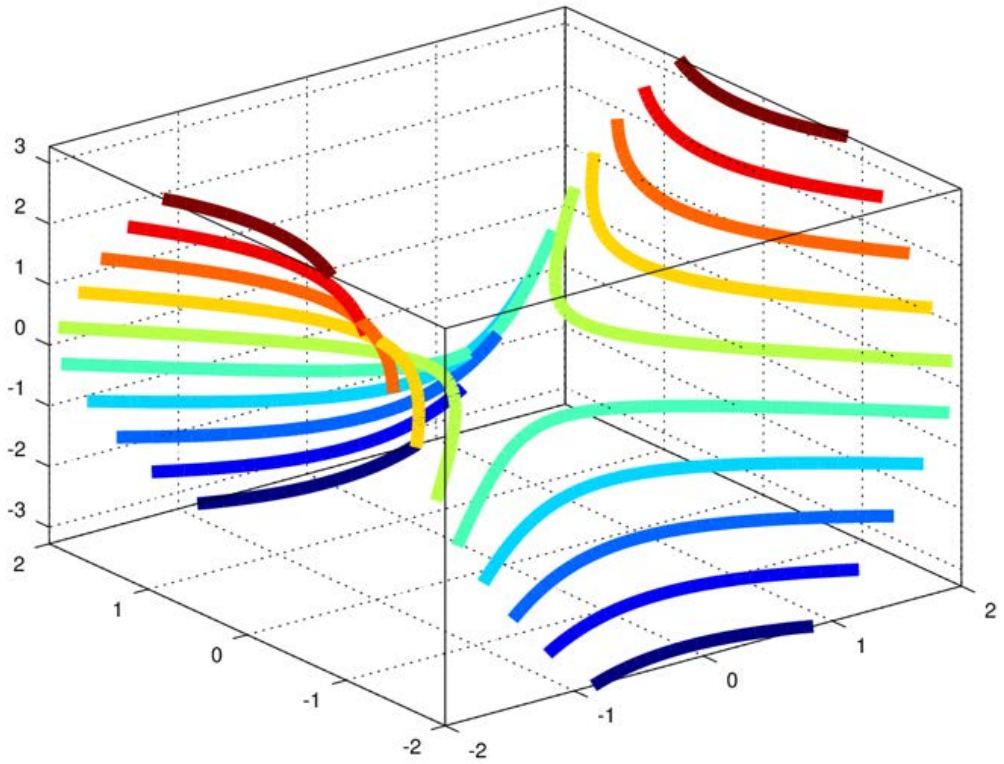
(φ, θ)=(35,30),

```
octave:> view(35,30)
```

Gráficas de contorno

- En ocasiones las gráficas en 3D se aprecian mejor con gráficas de contorno que en Octave se generan con las funciones: *contour*, *contourf*, *contour3*



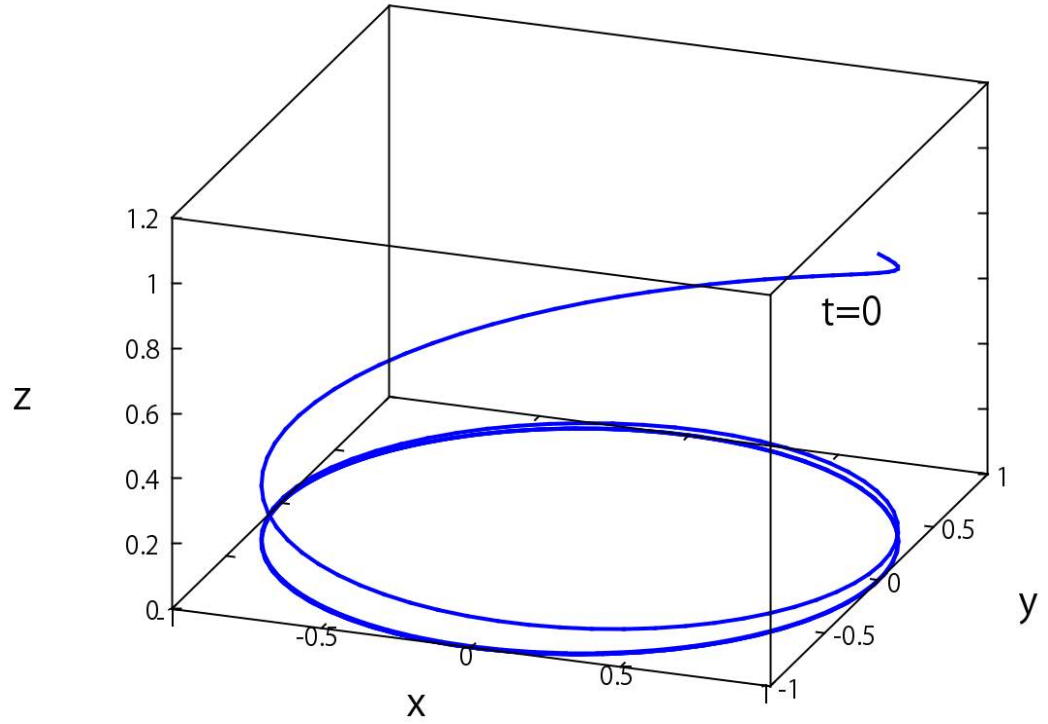


Curvas paramétricas

- Para dibujar curvas paramétricas en tres dimensiones se utiliza la función *plot3* que recibe tres vectores con las coordenadas x, y, z de cada punto a dibujar

$$f(x) = \left[cos(x), sin(x), e^{-x/2}\right]$$

```
octave:> x = linspace(0, 10*pi)';
octave:> f = [cos(x), sin(x), exp(-0.5*x)];
octave:> plot3(f(:,1), f(:,2), f(:,3), "linewidth", 4)
```



Programación de scripts

Scripts en Octave

- Un script en Octave es un archivo de texto donde se pueden utilizar las mismas funciones y sintaxis que en el intérprete (es buena idea probar primero en el intérprete que las cosas funcionan)
- Un script se guarda en un archivo con extensión .m y se carga escribiendo su nombre (sin extensión) en el intérprete
- El archivo tiene que estar en el directorio activo o en la ruta de búsqueda, que se consulta con la función *path*. Se pueden añadir carpetas a la ruta de búsqueda con la función *addpath* y es habitual hacerlo en el archivo .octaverc que se carga al invocar Octave
- Están definidas las funciones homónimas en Unix para consultar por el directorio actual, cambiar de directorio, y listar el contenido del directorio actual: pwd, cd, ls


```
octave:> prueba1
ans =  0.045235
```

Entrada/salida

- La lectura se hace con la función *input* y la escritura con *disp*
- En algunos sistemas es buena idea vaciar el búfer de salida antes de hacer una lectura

```
octave:> fflush(stdout);

octave:> a = input("Introduce un número: ");
Introduce un número: 33
octave:> a
a =  33

octave:> s = input("Introduce una cadena: ", "s");
Introduce una cadena: hola mundo
octave:> s
s = hola mundo

octave:> A = input("Introduce una matriz: ")
Introduce una matriz: [1 2; 3 4]
A =
   1   2
   3   4

octave:> disp("El valor de a es: "), a
El valor de a es:
a =  33
```

Estructuras de control

- Los operadores booleanos son &, |, ! que cuando se aplican entre matrices de la misma dimensión operan elemento a elemento
- Los operadores && y || interpretan las matrices como valores lógicos, de forma que una matriz es cierta si no tiene ningún 0, sin importar su dimensión

```
octave:> A=eye(2); B=[1 2;3 4];
octave:> A==eye(2) & B==eye(2)
ans =

   1   0
   0   0

octave:> A==eye(2) | B==eye(2)
ans =
   1   1
   1   1

octave:> !A
ans =
   0   1
   1   0
```

```
octave:> A==B && 1
ans = 0

octave:> A==B || [ 1 1 ]
ans =  1
```

Selección condicional

```
if expresión
  sentencias
elseif expresión
  sentencias
else
  sentencias
endif
```

```
x = 1;

if (x == 1)
  disp ("one");
elseif (x == 2)
  disp ("two");
else
  disp ("not one or two");
endif
```

```
switch expresión
  case x1
    sentencias
  case x1
    sentencias
  otherwise
    sentencias
endswitch
```

```
yesno = "yes"

switch yesno
  case {"Yes" "yes" "YES" "y" "Y"}
    value = 1;
  case {"No" "no" "NO" "n" "N"}
    value = 0;
  otherwise
    error ("invalid value");
endswitch
```

Iteración

```
for variable = rango
  sentencias
endfor
```

```
for i = 1:10
  i
endfor
```

```
while expresión
  sentencias
endwhile
```

```
x = 1;

while 1+x > 1
  x = x/2;
endwhile
```

```
do
  sentencias
until expresión
```

```
x = 1;

do
  x = x/2;
until 1+x > 1
```

Manejo de excepciones

- Las *sentencias* de *unwind_protect_cleanup* se ejecutan siempre, ocurra o no un error en las sentencias de *unwind_protect*
- Las sentencias de *catch* sólo se ejecutan si hay un error en las sentencias de *try*

```
try
  sentencias
catch
  sentencias
end_try_catch
```

```
unwind_protect
  sentencias
unwind_protect_cleanup
  sentencias
end_unwind_protect
```

Entrada/salida estilo C

- Octave implementa las funciones de entrada/salida tipo C: printf, fprintf, fgets, fscanf

Format specifiers		Escape sequence	
`%d`	Integer format	`\n`	Newline
`%f`	Floating point format	`\t`	Horizontal tab
`%e` or `%E`	Scientific floating point format	`\b`	Backspace
`%c`	Character format	`\r`	Carriage return
`%s`	String format		

```
octave:> for n=1:5
> printf("n is %f \t", n);
> endfor
n is 1.000000    n is 2.000000    n is 3.000000    n is 4.000000
   n is 5.000000
```

Persistencia

- *Save* y *load* permiten guardar y volver a cargar variables entre sesiones de Octave

```
save –option1 –option2 filename variable1 variable2 ...
load –option1 –option2 filename
```

Option	Description
`-text`	Saves the variables in readable text format with information about the variables (names, dimensions, and so on.) Also, Octave prints a small file header about who created the file and when. This option is set as default.
`-ascii`	Saves the variables in ASCII format. This format will not include variable information. This is not recommended if you save more than one variable and wish to load them into Octave at a later stage. This is useful when data is read by other programs.
`-binary`	Saves the variables in Octave's own binary format. This could speed up things.
`-hdf5`	Portable binary format.
`-v`x or `–V`x	Saves the variables in MATLAB format. Currently, x can have values 4, 6, or 7 and indicates the MATLAB version number.
`-zip` or `-z`	Compressed output format (for saving hard disk space). This option can be used together with any format option above.

Definición de funciones

Funciones

- Una función se guarda en un archivo de texto con el mismo nombre que la función que contiene, con extensión .m, en el directorio activo o alguno incluido en la ruta de búsqueda
- Las primeras líneas de comentarios son la ayuda de la función que se invoca con *help* en el intérprete
- Todos los parámetros, de entrada o salida, se pasan por valor, y las variables locales son locales

```
function [output1, output2, ...] = functionname(input1,input2,...)
     sentencias
endfunction
```

minmax.m

```
#
# Usage:
#   [minx, maxx] = minmax(x)
#
# Returns the minimum and maximum values of a vector
# array x
#

function [minx, maxx] = minmax(x)

# Using the Octave build-in max and min functions
maxx = max(x);
minx = min(x);

endfunction
```

Control de errores

- *warning* emite un mensaje y continua la ejecución y *error* emite el mensaje y sale de la función
- *nargin* y *nargout* obtienen, respectivamente, el número de parámetros de entrada y salida de la función

```
octave:> [mina maxa] = minmax("Hello World")
error: max: wrong type argument 'string'
error:...

octave:> [mina maxa] = minmax([1 2; 3 4])
mina =
  1  2
maxa =
  3  4

octave:> [a b c] = minmax([1 2 3 4])
mina = 1
maxa = 4
error: element number 3 undefined in return list
```

```
function [minx, maxx] = minmax(x)

  if ( nargin!=1 )
    usage("Number of inputs to minmax must be 1");
  elseif ( nargout>2 )
    usage("Number of outputs from minmax cannot exceed 2");
  endif

  [nr nc] = size(x);
  if ( nr>1 & nc>1 )
    warning("Input to minmax is a matrix array:\
    output will be vectors");
  elseif ( ischar(x) )
    error("Input to minmax cannot be a character array");
  endif
  maxx = max(x);
  minx = min(x);

endfunction
```

Funciones anónimas

- Muchas funciones en Octave reciben a otras funciones como parámetros, por ejemplo *quad* hace la integración numérica de una función dada en el intervalo especificado
- Se pueden definir funciones anónimas que sirvan de parámetros a otras funciones o asignarlas a variables

```
octave:> quad(@sin, 0, pi)
ans =  2

octave:> quad( @(x) (-2*x.^2 + 3*x), 0, 1 )
ans =  0.83333

octave:> funcion = @(x) (-2*x.^2 + 3*x)
funcion =

@(x) (-2 * x .^ 2 + 3 * x)

octave:> funcion(1)
ans =  1

octave:> quad( funcion, 0, 1)
ans =  0.83333
```

Funciones anónimas con menos parámetros

- La función *quad* espera como parámetro una función de la forma y=f(*x*) ¿podemos pasarle funciones con más de un parámetro?

recta.m

```
function y = recta(x, a, b)

  y = a + x * b;

endfunction
```

```
octave:> a0 = 0
a0 = 0
octave:> a1 = 1
a1 =  1
octave:> recta1 = @ (x) recta(x, a0, a1)
recta1 =

@(x) recta (x, a0, a1)

octave:> recta1(2)
ans =  2
```

```
octave:> quad( @(x) recta(x, a0, a1), 0, 1 )
ans =  0.50000

octave:> meta(@(x) recta(x, 0, 1), 2)
ans =  4
```

```
function y = meta(F, x)

  y = 2 * F(x);

endfunction
```

Vectorización y eficiencia

- Para medir el tiempo de ejecución se usan las funciones predefinidas *tic* y *toc*
- Siempre que sea posible se deben usar operaciones entre matrices en lugar de bucles

```
octave:> A=rand(1000,1000); B=rand(1000,1000);

octave:> tic(); C=A+B; add_time = toc()
add_time =  0.0074439

octave:> tic(), for i=1:1000, for j=1:1000, C(i,j) =
A(i,j)+B(i,j); end, end, add_time = toc()
add_time =  14.518
```

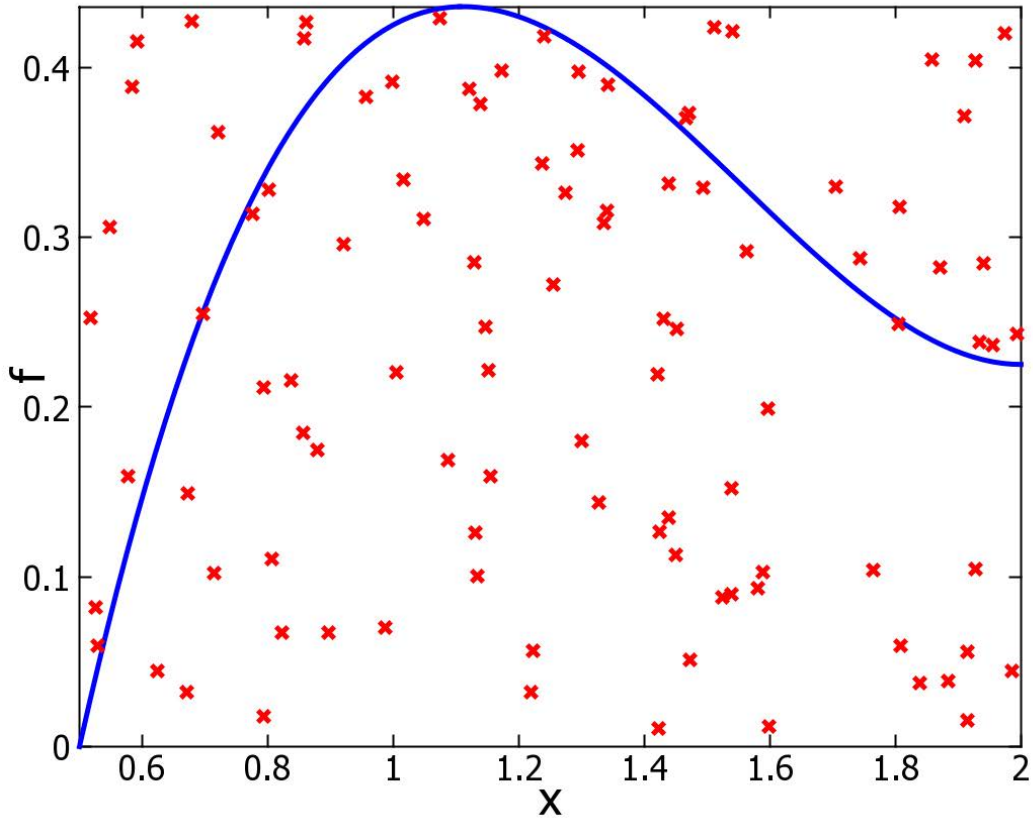
Depuración

- *dbstop*(*nombre_funcion*, *num_linea*) añade un punto de ruptura en la línea indicada de la función especificada y *dbclear*(*nombre_funcion*) los elimina. Al invocar a la función, el intérprete se detendrá en la línea especificada y entrará en modo depuración. El modo depuración es como el intérprete de alto nivel, pero con acceso a las variables locales de la función
- *dbnext* ejecuta la siguiente instrucción y *dbnext*(num) las siguientes *num* instrucciones
- *dbquit* saca del modo depuración
- También útiles: *dbstatus*, *debug_on_error*, *debug_on_warning*, *debug_on_interrupt*

```
octave:>dbstop( "recta", 1 )
ans =  3

octave:>recta(1, 0, 1)
stopped in /Users/pedro/local/aa/octave_utils/recta.m at line 3
3:   y = a + x * b;

debug> y
error: `y' undefined near line 37 column 1

debug> x
x =  1

debug> dbnext
stopped in /Users/pedro/local/aa/octave_utils/recta.m at line 5
5: endfunction

debug> dbnext
ans =  1

octave:>
```

Práctica 0

Cáculo de la integral por el método de Monte Carlo



$$I = \int_a^b f(x)dx = F(b) - F(a) \approx \frac{N_{debajo}}{N_{total}}(b-a)M$$