



## Curso rápido de Matlab

**Rafael Palacios (dic/2004)**

# Temario

1. Introducción a Matlab.
  2. Estructuras básicas de datos.
  3. Programación en Matlab.
  4. Estructuras avanzadas de datos.
  5. Optimización de código.
  6. Representaciones gráficas.
  7. Desarrollo de aplicaciones con Matlab.
- 
- 29/nov
- 13/dic
- 20/dic

# Temario (1)

## 1. Introducción a Matlab.

- Versiones de Matlab,
- Mejoras de la versión 7
- Entorno de desarrollo,
- Ayudas y documentación,
- Toolboxes.

2. Estructuras básicas de datos.

3. Programación en Matlab.

4. Estructuras avanzadas de datos.

5. Optimización de código.

6. Representaciones gráficas.

7. Desarrollo de aplicaciones con Matlab.

# Introducción a Matlab

- ¿Qué es Matlab?
  - Matlab = **Mat**rix **Lab**oratory.
  - Programa interactivo para realizar cálculos numéricos y visualizaciones en el ordenador.
  - Programa comercial de The Mathworks Inc (Natick, MA). <http://www.mathworks.com>
  - Creado en California por Jack Little and Cleve Moler en 1984, para realizar cálculo matricial en ordenadores sin necesidad de conocimientos de programación.

# Introducción a Matlab

Entorno interactivo

+

Lenguaje de programación  
(con interfaces externos Fortran, C...)

+

Entorno para desarrollo de aplicaciones

# Versiones de Matlab

- Plataformas donde corre Matlab
  - Sistema Operativo
    - Unix: Linux, solaris, HP-UX
    - MacOS
    - MS-Windows
  - Arquitectura
    - RISC: Sparc, HP-PA
    - PowerMac (G4, G5)
    - Intel Pentium(III, IV, Xeon, M), AMD (Athlon, Opteron)

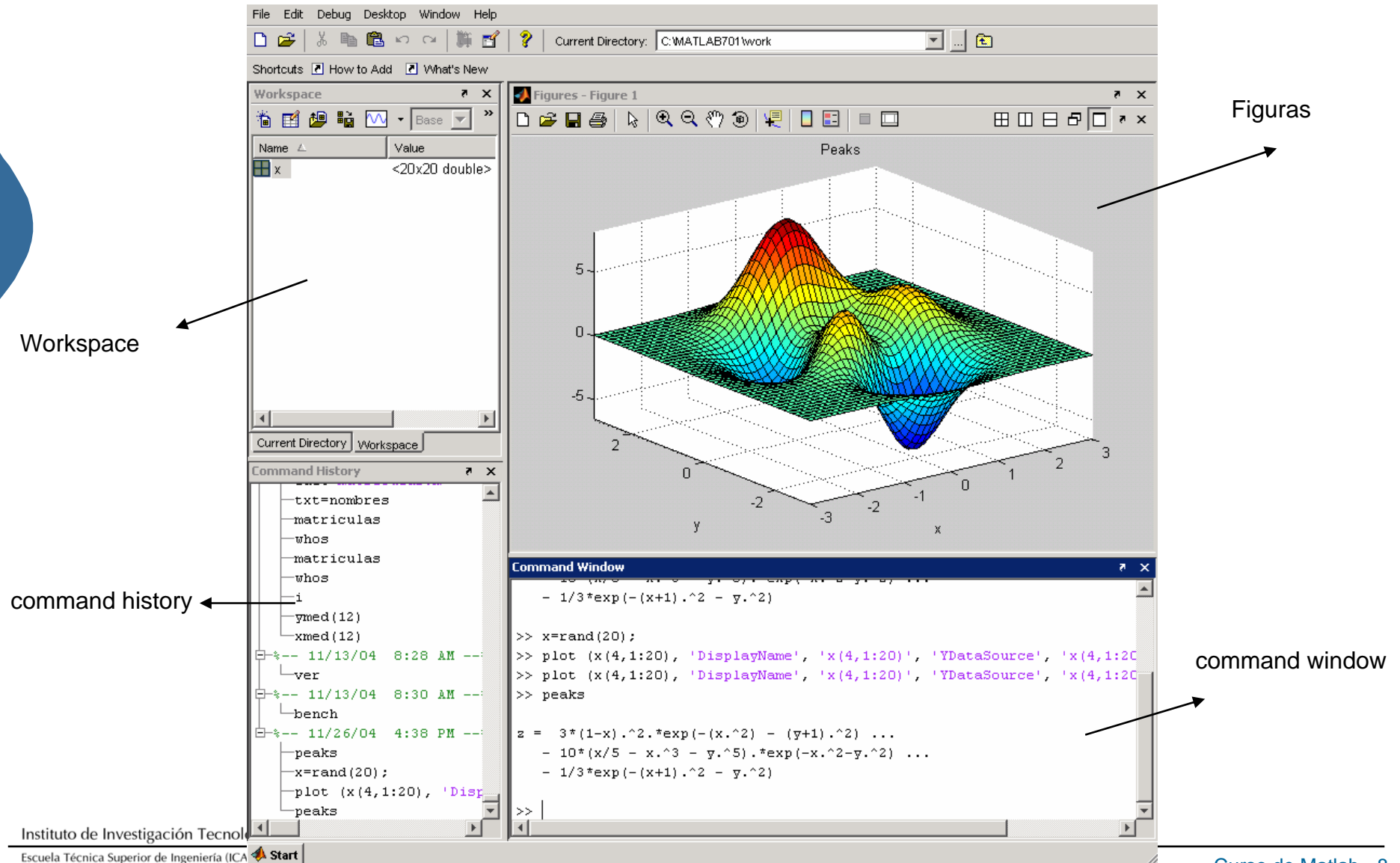
# Versiones de Matlab

---

- Matlab 5
  - Gráficos de calidad (2D, 3D)
  - PC: Corre bajo windows utilizando toda la memoria disponible
- Matlab 6
  - Entorno de desarrollo con interfaz Java.
  - Matrices 3D, estructuras, cell arrays
- Matlab 7
  - Mejoras en el interfaz y mejora de Simulink
  - Matlab compiler admite objetos
  - Cálculo con enteros

# Algunas mejoras de Matlab 7

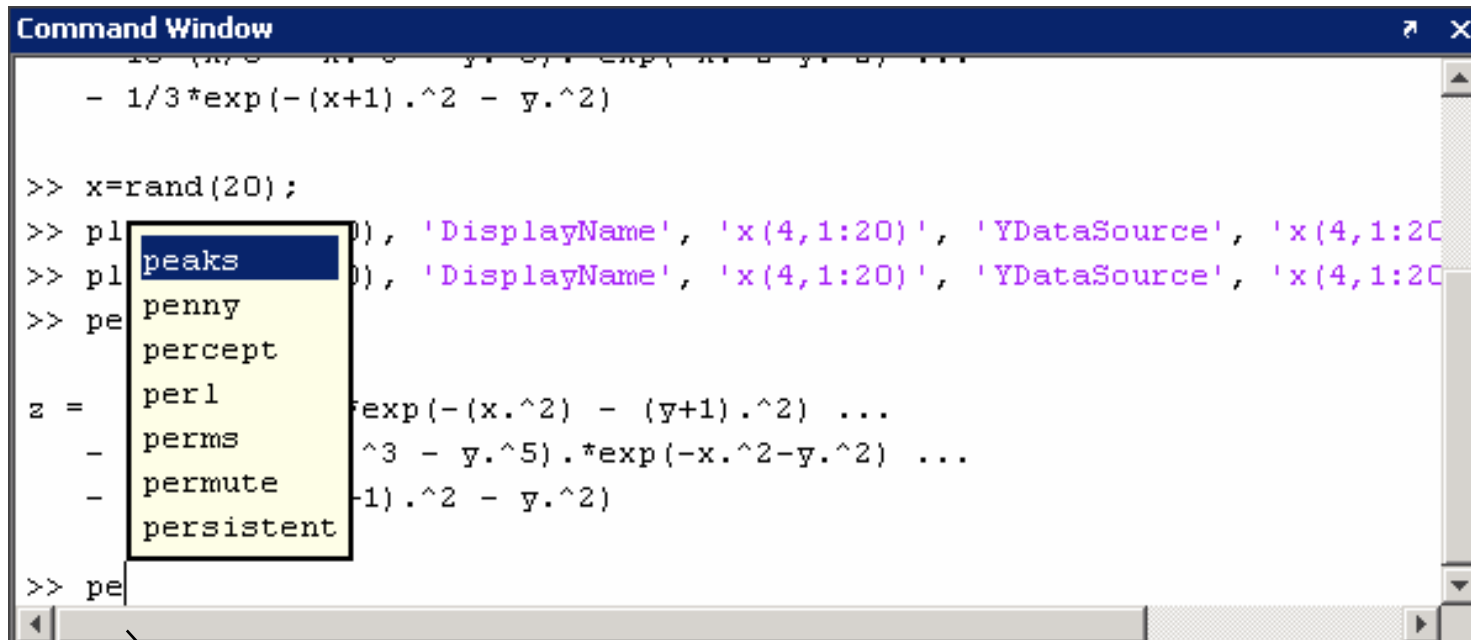
- Se puede poner todas las ventanas dentro del entorno de desarrollo





# Algunas mejoras de Matlab 7

- Acceso a comandos anteriores



The screenshot shows the MATLAB Command Window with a list of previous commands. A dropdown menu is open over the command history, displaying a list of commands starting with 'pe'. The command 'peaks' is highlighted at the top of the list. The command history includes:

```
- 1/3*exp(-(x+1).^2 - y.^2)
>> x=rand(20);
>> pl = plot(x(4,1:20), 'DisplayName', 'x(4,1:20)', 'YDataSource', 'x(4,1:20)');
>> pl = plot(x(4,1:20), 'DisplayName', 'x(4,1:20)', 'YDataSource', 'x(4,1:20)');
>> pe
z =
    exp(-(x.^2) - (y+1).^2) ...
    ^3 - y.^5).*exp(-x.^2-y.^2) ...
    -1).^2 - y.^2)
>> pe
```

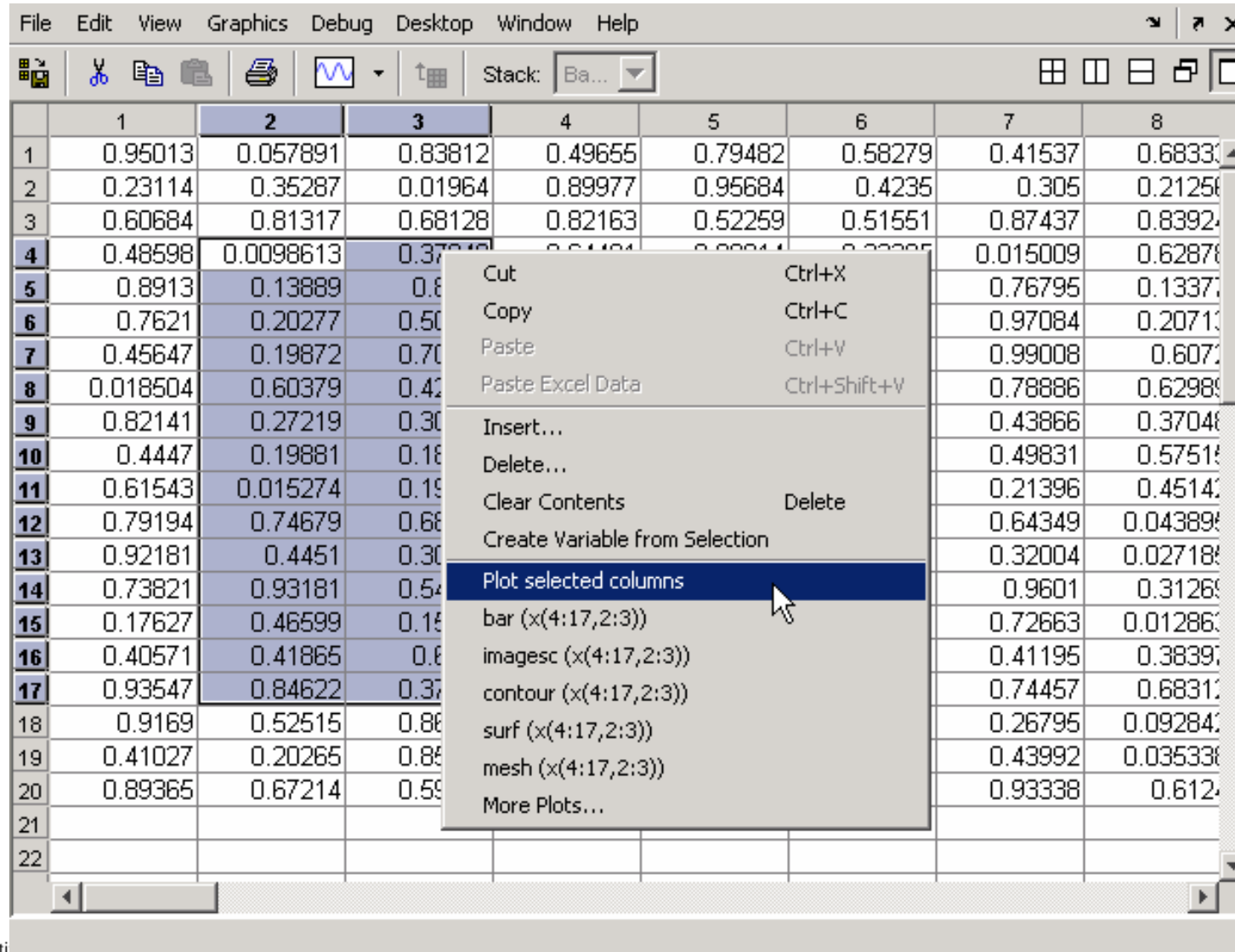
The dropdown menu lists the following commands:

- peaks
- penny
- percept
- perl
- perms
- permute
- persistent

Teclas: PE TAB

# Algunas mejoras de Matlab 7

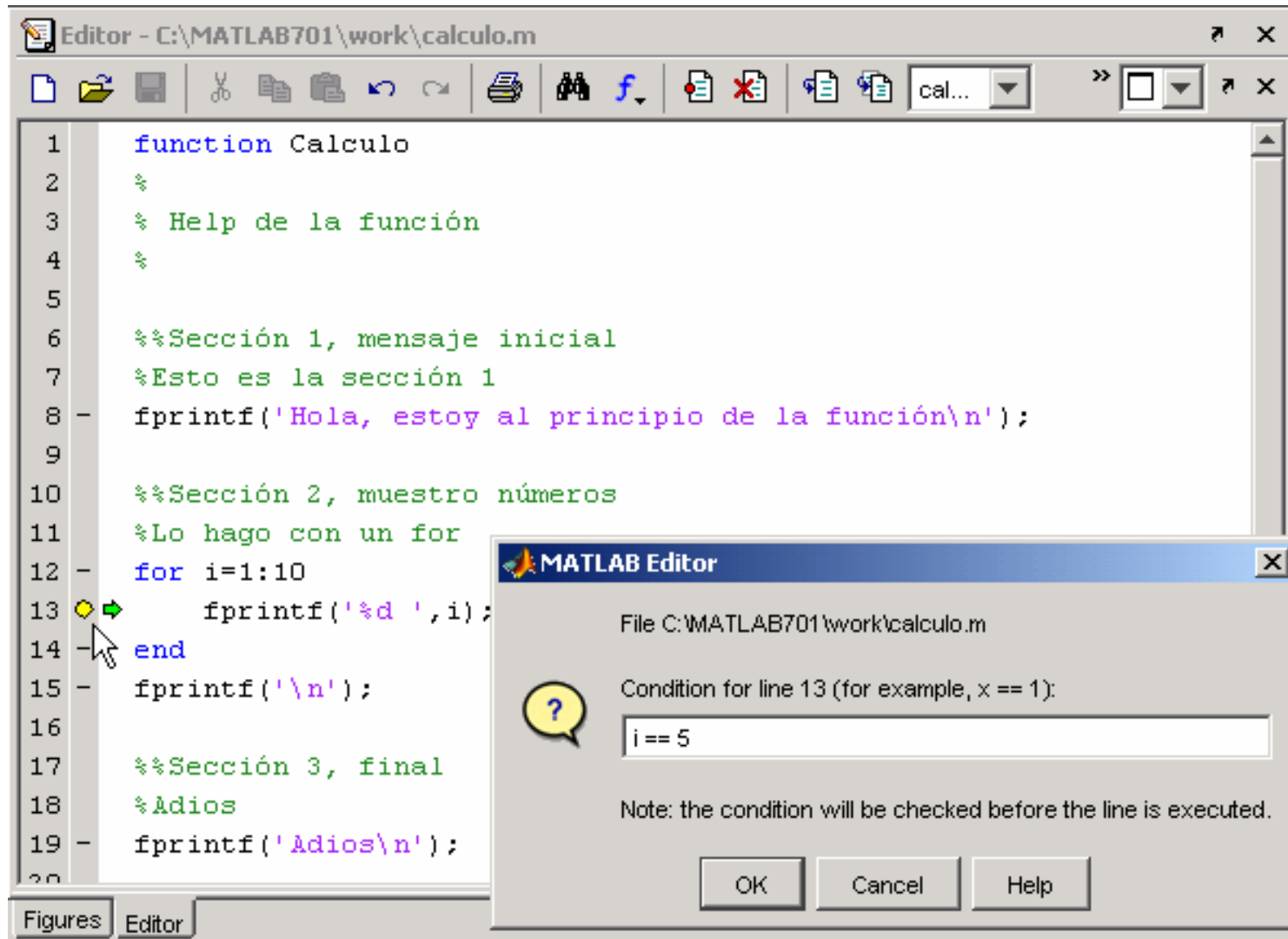
- Dibujar gráficos directamente desde el editor de matrices



Tamaño máximo  
de la matriz:  
524288 elementos

# Algunas mejoras de Matlab 7

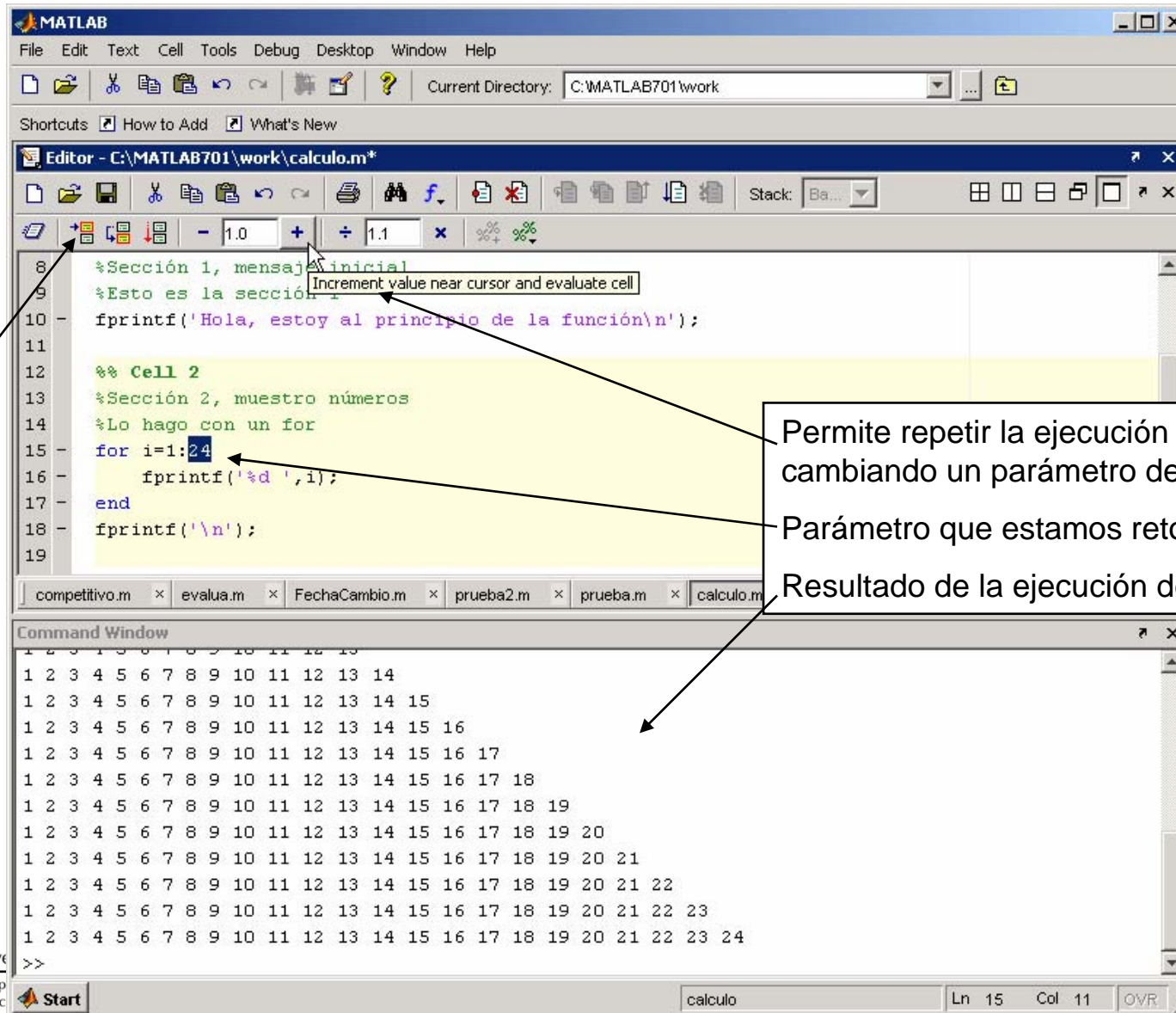
- Breakpoints con expresiones condicionales



# Algunas mejoras de Matlab 7

- Ejecución del código por secciones (cell→enable cell mode)

Ejecución  
sección por  
sección



Permite repetir la ejecución de una sección cambiando un parámetro de la misma.

Parámetro que estamos retocando

Resultado de la ejecución de la sección

# Otras mejoras de Matlab 7

- Ejecución/Debugger por secciones de código (Cell Mode)
- Generación automática de documentación
- Optimización de código con M-Lint y profiler (ver cap6)
- Mejoras en la generación automática de código desde Simulink
- Interactive plot tool (ver cap 6)
- Función textscan para leer archivos
- Cálculo con variables enteras.
  - Matrices más pequeñas, código más rápido. → imágenes
- Acentos y caracteres internacionales en el editor.

# Entorno de desarrollo

- Arraque de Matlab

- Windows

- Inicio/Programas/Matlab 7.0.1/Matlab 7.0.1

- icono de Matlab en el escritorio



- comando: `matlab`

- comando: `matlab -r programa`

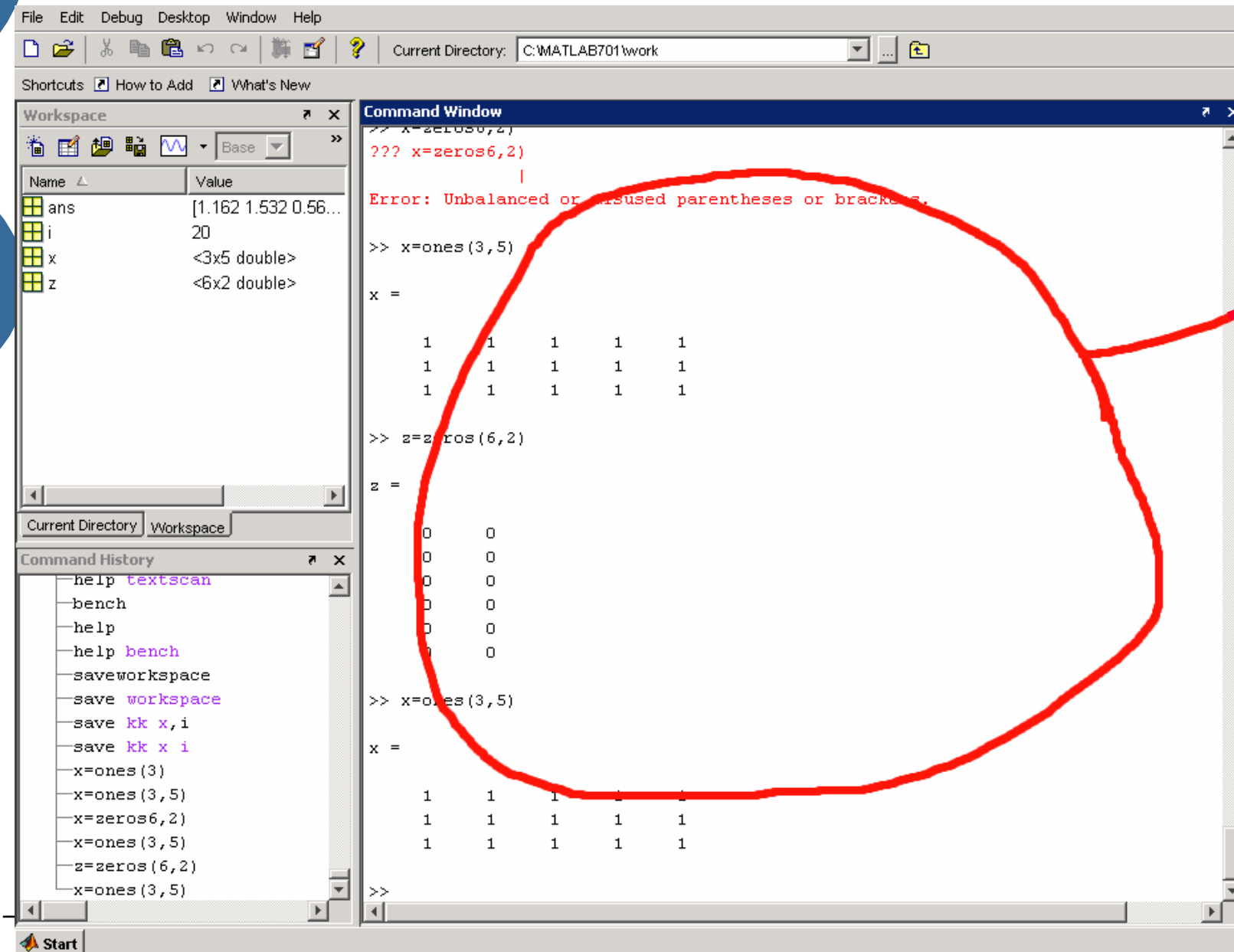
## Unix

- comando: `matlab`

- En IIT comandos: `matlab`, `matlab5`, `matlab61`,  
`matlab65`, `matlab7`

- Ejemplo útil: `matlab65 -nodesplay` (Modo Consola sin gráficos)

# Entorno de desarrollo



Ventana de comandos



# Comandos básicos

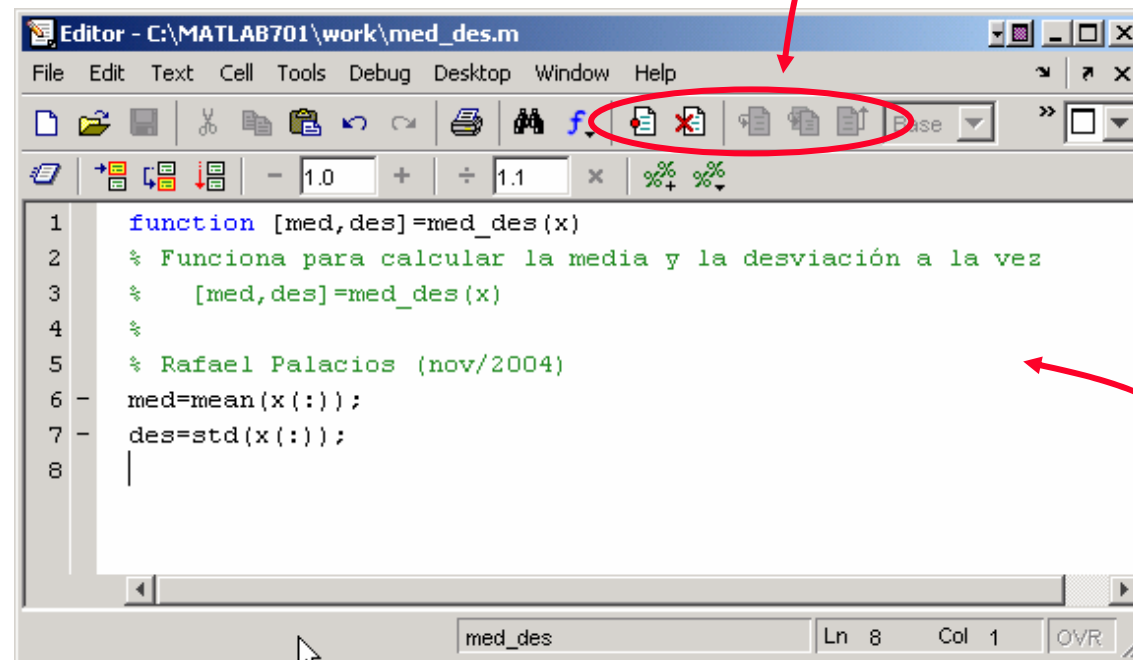
- `ver` → muestra la versión, el código de licencia y las toolboxes disponibles
  - Licencia 46431: profesores (para investigación y proyectos)
  - Licencia 205966: aulas (sólo para docencia)
- `whos` → lista todas las variables disponibles
- `save archivo` → guarda todas las variables
- `save archivo a b` → guarda las variables a y b
- `load archivo` → carga variables
- `quit` → salir



# Editor

- Matlab incorpora un editor que interacciona con el resto del entorno.

Ejecución por secciones  
en cell mode

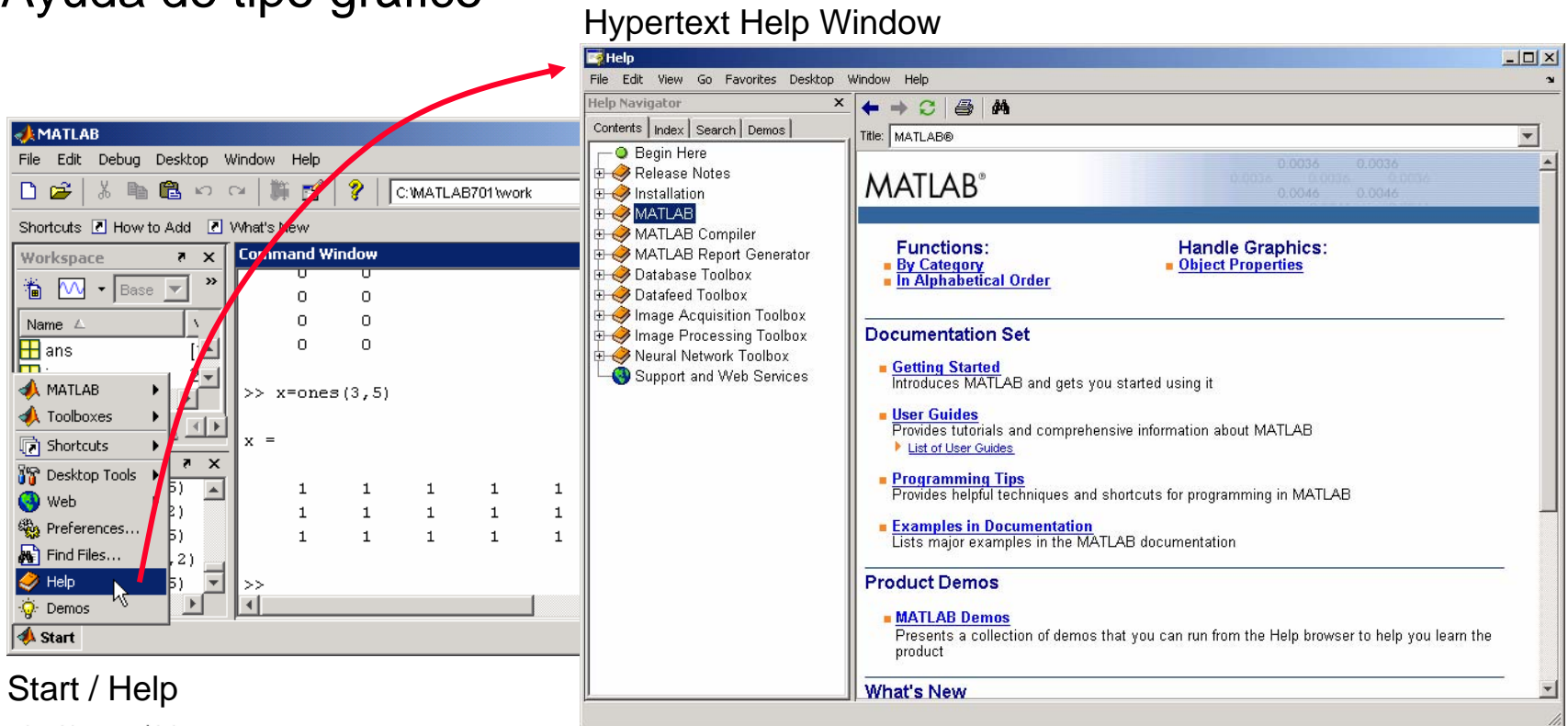


Controles del debugger

sintaxis

# Ayudas y documentación

- Ayuda en modo texto mediante comandos
  - `help función` → muestra la ayuda de una función
  - `help` vale tanto para las funciones del sistema como para desarrollos propios
- Ayuda de tipo gráfico



# Ejemplo de consulta

## Secciones de la ayuda

- Syntax
- Description
- Arguments
- Examples
- Algorithm
- Limitations
- See Also
- References

developed by scholars



$$f(x) = 100(x_2 - x_1^2)^2 + (a - x_1)^2$$

This changes the location of the minimum to the point  $[a, a^2]$ . To minimize this function for a specific value of  $a$ , for example  $a = \text{sqrt}(2)$ , create a one-argument anonymous function that captures the value of  $a$ .

```
a = sqrt(2);  
banana = @(x) 100*(x(2)-x(1)^2)^2+(a-x(1))^2;
```

Then the statement

```
[x,fval] = fminsearch(banana, [-1.2, 1], ...  
    optimset('TolX',1e-8));
```

seeks the minimum  $[\text{sqrt}(2), 2]$  to an accuracy higher than the default on  $x$ .

### Algorithm

`fminsearch` uses the simplex search method of [1]. This is a direct search method that does not use numerical or analytic gradients.

If  $n$  is the length of  $x$ , a simplex in  $n$ -dimensional space is characterized by the  $n+1$  distinct vectors that are its vertices. In two-space, a simplex is a triangle; in three-space, it is a pyramid. At each step of the search, a new point in or near the current simplex is generated. The function value at the new point is compared with the function's values at the vertices of the simplex and, usually, one of the vertices is replaced by the new point, giving a new simplex. This step is repeated until the diameter of the simplex is less than the specified tolerance.

### Limitations

`fminsearch` can often handle discontinuity, particularly if it does not occur near the solution. `fminsearch` may only give local solutions.

`fminsearch` only minimizes over the real numbers, that is,  $x$  must only consist of real numbers and  $f(x)$  must only return real numbers. When  $x$  has complex variables, they must be split into real and imaginary parts.

### See Also

[fminbnd](#), [optimset](#), [function\\_handle](#) (@), [anonymous functions](#)

### References

[1] Lagarias, J.C., J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," *SIAM Journal of Optimization*, Vol. 9 Number 1, pp. 112-147, 1998.

# Ayuda on-line

- **Página oficial de soporte**

[http: //www. mathworks. com/support/](http://www.mathworks.com/support/)

- Documentación
- Soluciones a problemas ordenadas por categorías
- Ejemplos de código
- Noticias
- Actualizaciones

- **Matlab Central**

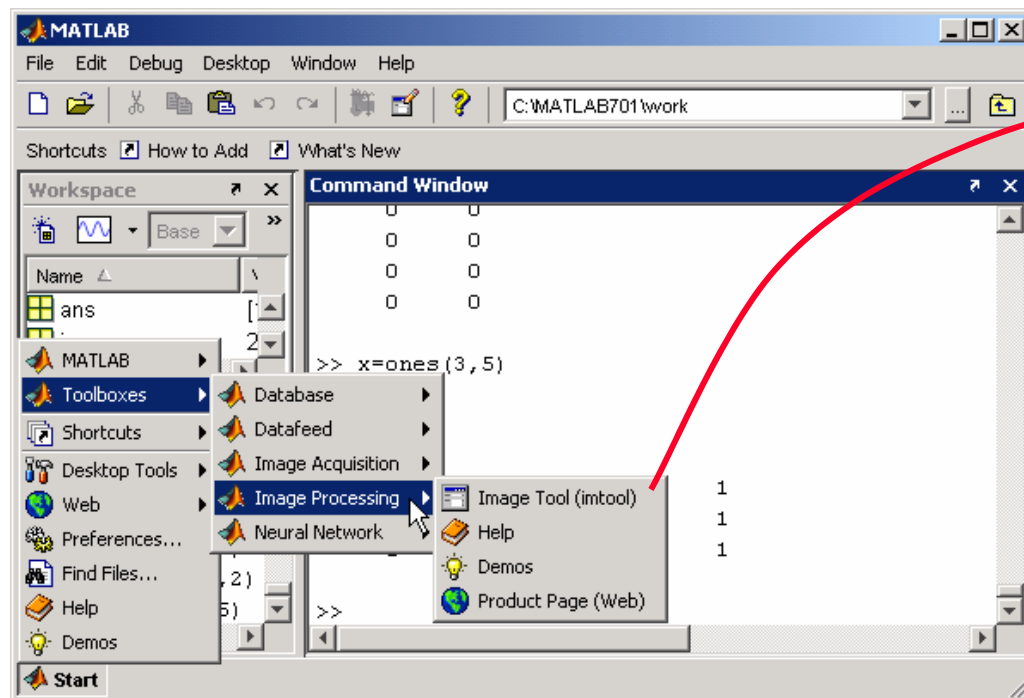
- Newsgroups
- File Exchange
- Link Exchange

- **Soporte técnico personal por correo electrónico**

- Utilizar un código de licencia válido
- Describir la plataforma
- Acotar el problema

# Toolboxes

- Librerías especializadas en materias concretas. Incluyen:
  - Manuales tipo tutorial (User's Guide) [HTML, PDF]
  - Referencia de las funciones (Reference Guide) [HTML, PDF]
  - Programas de demo
  - Aplicaciones completas listas para utilizar



# Ejemplos de Toolboxes

```
>> ver
```

```
-----  
MATLAB Versi on 6.5.0.180913a (R13)
```

```
MATLAB License Number: 46431
```

```
Operating System: SunOS 5.8 Generi c_108528-29 sun4u
```

```
Java VM Version: Java 1.3.1_02 with Sun Microsystems Inc. Java HotSpot(TM) Server VM
```

```
-----  
MATLAB                               Versi on 6.5           (R13)  
Simulink                             Versi on 5.0           (R13)  
Control System Tool box              Versi on 5.2           (R13)  
Fuzzy Logi c Tool box                Versi on 2.1.2         (R13)  
Image Processi ng Tool box            Versi on 3.2           (R13)  
MATLAB Compi ler                     Versi on 3.0           (R13)  
MATLAB Web Server                    Versi on 1.2.2         (R13)  
Mu-Analysi s and Synthesi s Tool box Versi on 3.0.7         (R13)  
Neural Network Tool box              Versi on 4.0.2         (R13)  
Nonl i near Control Design Blockset  Versi on 1.1.6         (R13)  
Optimizati on Tool box               Versi on 2.2           (R13)  
Real -Time Workshop                  Versi on 5.0           (R13)  
Robust Control Tool box              Versi on 2.0.9         (R13)  
SB2SL (converts SystemBuild to Simu... Versi on 2.5           (R13)  
Signal Processing Tool box           Versi on 6.0           (R13)  
System I denti fi cati on Tool box    Versi on 5.0.2         (R13)
```

```
>> date
```

```
ans =
```

```
27-Nov-2004
```

## Application Areas

- **Technical Computing**
  - Mathematical computation, analysis, visualization, and algorithm development
- **Control Design**
  - Model-Based Design for control systems, including simulation, rapid prototyping, and code generation for embedded systems
- **Signal Processing and Communications**
  - Model-Based Design for signal processing and communication systems, including simulation, code generation, and verification
- **Image Processing**
  - Image acquisition, analysis, visualization, and algorithm development
- **Test & Measurement**
  - Hardware connectivity and data analysis for test and measurement applications
- **Financial Modeling and Analysis**
  - Financial modeling, analysis, and application deployment



# Temario (2)

---

1. Introducción a Matlab.
- 2. Estructuras básicas de datos.**
  - Variables
  - Vectores y matrices
  - Ejemplos de operaciones
  - Tipos de datos
3. Programación en Matlab.
4. Estructuras avanzadas de datos.
5. Optimización de código.
6. Representaciones gráficas.
7. Desarrollo de aplicaciones con Matlab.



# Variables

- Matlab no requiere declarar ni dimensionar variables
  - Las variables se auto-declaran al inicializarlas
  - La memoria se reasigna dinámicamente

```
>> x=5;
>> y=20;
>> z=x*y

Z =

    100

>> datos=load('datos.txt');
>> cadena='hol a' ;
```

poniendo ';' se realiza la asignación pero no se muestra el resultado

sin poner ';' se muestra el resultado final

# Vectores y Matrices

- Matlab considera que todas las variables son matrices (vectores y escalares son casos particulares).

Ejemplos de inicialización de **vectores fila**

>> x=[1, 2, 3, 5, 7, 11, 13];	→	[ 1   2   3   5   7   11   13 ]
>> x=[1 2 3 5 7 11 13];	→	[ 1   2   3   5   7   11   13 ]
>> y=1: 5;	→	[ 1   2   3   4   5 ]
>> pares=2: 2: 10;	→	[ 2   4   6   8   10 ]
>> i mp_down=9: -2: 1;	→	[ 9   7   5   3   1 ]
>>a(5)=7;	→	[ 0   0   0   0   7 ]

# Vectores y Matrices

Ejemplos de inicialización de **vectores columna**

```
>> x=[1; 2; 3; 5; 7; 11; 13]
```

X =

1  
2  
3  
5  
7  
11  
13

```
>> x=[1, 2, 3, 5, 7, 11, 13]';
```

Vector fila

traspuesta

# Vectores y Matrices

Ejemplos de inicialización de **Matrices**

```
>> M = [1 2 3; 4 5 6; 7 8 9];
```

1	2	3
4	5	6
7	8	9

```
>> ceros=zeros(2, 5);
```

0	0	0	0	0
0	0	0	0	0

```
>> unos=ones(3, 4);
```

1	1	1	1
1	1	1	1
1	1	1	1

```
>> M2=[ 20, 21, 22; M];
```

```
>> M2=[[20, 21, 22]; M];
```

20	21	22
1	2	3
4	5	6
7	8	9

```
>> M3=[ [15; 16; 17], M];
```

```
>> aleatorio=rand(20, 30);
```

```
>> normal=randn(20, 30);
```

15	1	2	3
16	4	5	6
17	7	8	9

# Acceso a los elementos de una matriz

- Matlab utiliza los paréntesis para acceder a elementos de la matriz
- Los subíndices empiezan en 1, por lo tanto el primer elemento es  $a(1, 1)$
- Ejemplo:  $a(3, 5) = 56.8$ ;

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	0.7264
0.7769	0.1482	0.4851	0.0232	0.6947



0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	<b>56.8000</b>
0.7769	0.1482	0.4851	0.0232	0.6947

# Acceso a los elementos de una matriz

- Se pueden utilizar vectores para definir índices
- Ejemplo 1: `a(2:3, 1:4)=zeros(2, 4);`  
o bien: `a(2:3, 1:4)=0;`

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

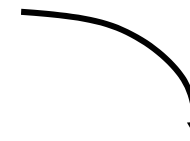


0.1737	0.3421	0.6391	0.1632	0.2313
0	0	0	0	0.8453
0	0	0	0	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

# Acceso a los elementos de una matriz

- Se pueden utilizar vectores para definir índices
- Ejemplo 2: `a([2, 3], [2, 4])=ones(2, 2);`  
o bien: `a([2, 3], [2, 4])=0;`

0.1737	0.3421	0.6391	0.1632	0.2313
0	0	0	0	0.8453
0	0	0	0	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947



0.1737	0.3421	0.6391	0.1632	0.2313
0	1.0000	0	1.0000	0.8453
0	1.0000	0	1.0000	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

# Acceso a los elementos de una matriz

- El operador ' : ' se utiliza para indicar "todos los elementos"

0.1737	0.3421	0.6391	0.1632	0.2313
0.7858	0.7742	0.0934	0.2763	0.8453
0.3656	0.1478	0.9288	0.1310	56.8000
0.7769	0.1482	0.4851	0.0232	0.6947

→ `a(3, :)`  
`size(a(3, :)) → [1 5]`

↓  
`a(:, 2)`  
`size(a(:, 2)) → [4 1]`

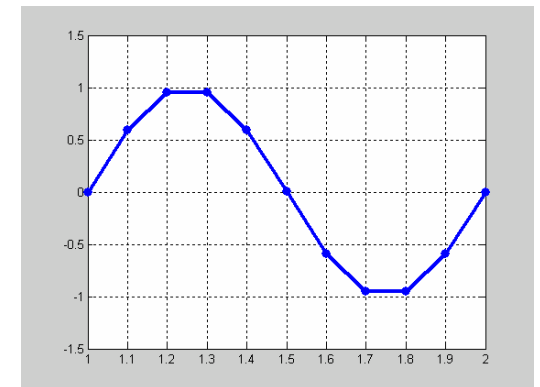
↘  
`a(:)` → todos los elementos  
`size(a(:)) → [20 1]`  
todo en un vector columna



# Acceso a los elementos de una matriz

- El operador 'end' significa "último elemento"
  - Ejemplo: Vector de diferencias

```
>> t=1:0.1:2;  
>> y=sin(2*pi*t);  
>> diferencias=[NaN; y(2:end)-y(1:end-1)];
```



t =	1.00	1.10	1.20	1.30	1.40	1.50	1.60	1.70	1.80	1.90	2.00
-----	------	------	------	------	------	------	------	------	------	------	------

y =	-0.00	0.59	0.95	0.95	0.59	0.00	-0.59	-0.95	-0.95	-0.59	-0.00
-----	-------	------	------	------	------	------	-------	-------	-------	-------	-------

diferencias =	NaN	0.59	0.36	0.00	-0.36	-0.59	-0.59	-0.36	-0.00	0.36	0.59
---------------	-----	------	------	------	-------	-------	-------	-------	-------	------	------

# Operaciones básicas

- Operaciones aritméticas: + - \* / ^
  - Matlab trabaja con matrices, a diferencia de otros lenguajes que sólo trabajan con escalares

```
>> a=rand(2, 5);  
>> b=rand(5, 2);  
>> c=a*b;           % matriz de 2x2  
>> d=b*a;           % matriz de 5x5
```

- Operaciones elemento a elemento: + - .\* ./ .^

mn = magic(4);

16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

mn\*mn  
mn^2

345	257	281	273
257	313	305	281
281	305	313	257
273	281	257	345

mn.\*mn  
mn.^2

256	4	9	169
25	121	100	64
81	49	36	144
16	196	225	1

# Otras operaciones matriciales

- Suma: `sum`

```
>> b=sum(A);    % como a es matriz, se suma por columnas y b es un vector fila
>> c=sum(b);    % como b es vector, se suman todos sus elementos

>> c=sum(sum(a)); % suma todos los elementos de la matriz a
>> c=sum(a(:));  % suma todos los elementos de la matriz a
```

- Media y desviación: `m=mean(A); sigma=std(A);`
- Elementos de la diagonal: `v=diag(A);`
- Left division: `x=A\B;` La solución por mínimos cuadrados de  $Ax = b$  se obtiene mediante  $x = A \setminus b$ ;
- Determinante: `c=det(A);`
- Inversa: `B=inv(A);`
- Autovalores: `v=eig(A);`
- Valor absoluto ó módulo de complejos: `B=abs(A);`

# Otras operaciones

- Trig: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `cosh`, `tanh`, `asinh`, `acosh`, `atanh`
- Rounding: `floor`, `ceil`, `round`, `fix`
- Modular: `rem`, `mod`
- Exponential: `exp`, `log`, `log2`, `log10`, `sqrt`
- Primes: `factor`, `primes`

# Tipos de datos

- Matlab opera normalmente en formato double según el estándar IEEE
- Maneja correctamente los valores Inf (infinito) y NaN (not-a-number)
- Complejos automáticos

```
>> a=123/0
Warni ng: Di vi de by zero.
a =
    Inf

>> b=0/0
Warni ng: Di vi de by zero.
b =
    NaN

>> Inf-Inf
ans =
    NaN

>> c=15+sqrt(-1)
c =
    15.0000 + 1.0000i
```

# Tipos de datos

- Matrices reales
  - double      real min  $\rightarrow 2.2251e-308$ , real max  $\rightarrow 1.7977e+308$ , eps  $\rightarrow 2.2204e-016$
  - single      real min  $\rightarrow 1.1755e-038$ , real max  $\rightarrow 3.4028e+038$ , eps  $\rightarrow 1.1921e-007$
- Matrices enteras
  - int8, uint8
  - int16, uint16
  - int32, uint32
  - int64, uint64
- Otros
  - char
  - logical
  - cell
  - struct

# Tipos de datos: Matrices dispersas

- Las matrices dispersas ahorran memoria y los calculos son más eficientes

```
s = sparse(1000, 1000);  
s(2, 11) = 2;  
s(992, 875) = 3;  
s(875, 992) = 4.7;
```

- Todas las operaciones de matrices funcionan con matrices dispersas
- Si se vuelve ineficiente, se convierte automáticamente a matriz normal

```
s=s+3;    % s dej a de ser di spersa
```

# Temario (3)

---

1. Introducción a Matlab.
2. Estructuras básicas de datos.
- 3. Programación en Matlab.**
  - Scripts
  - Funciones
  - Expresiones lógicas
  - Control de flujo
4. Estructuras avanzadas de datos.
5. Optimización de código.
6. Representaciones gráficas.
7. Desarrollo de aplicaciones con Matlab.



# Scripts

- Un script es una secuencia de instrucciones de Matlab guardada en un archivo con extensión .m

```
%Script de ejemplo

%% Inicio
a=magic(4);
fprintf('Inicio cálculos\n');

%% Traza
traza=sum(diag(a));

%% Resultado
fprintf('La traza vale: %f\n',traza)
```

ejem\_script.m

- Se ejecuta escribiendo su nombre:

```
>> ejem_script
```

# Funciones (llamada)

- Las funciones puede recibir varios valores y devolver varios resultados

```
[m, d]=med_des(x);
```

- Puede haber argumentos opcionales

```
mi t=i mread(' cameraman. ti f' , ' T I F F' );
```

```
mi t=i mread(' cameraman. ti f' );
```

- No es necesario asignar todos los valores retornados

```
[mi t, map]=i mread(' i mageman. gi f' );
```

```
mi t=i mread(' i mageman. gi f' );
```

# Funciones (definición)

- Las funciones se escriben en archivos .m que deben encontrarse en el directorio actual (o en un directorio definido en el path)

```
function [med,des]=med_des(x)
% Funciona para calcular la media y la desviación a la vez
% [med,des]=med_des(x)
%
% Rafael Palacios (nov/2004)
med=mean(x(:));
des=std(x(:));
```

Med\_des.m

Información que aparece al hacer help med\_des

# Funciones

- La variable `nargin` (local de la función) es el número de argumentos recibidos.
- La variable `nargout` (local de la función) es el número de argumentos que se recogerán en la llamada. Puede ahorrarnos unos cálculos.
- Todos los argumentos llegan por valor, no es posible hacer paso por referencia.

• Los scripts comparten las variables del workspace, mientras que las funciones utilizan variables en local

# Expresiones lógicas

- Operadores relacionales: `~=` `==` `>` `<` `>=` `<=`
- Operadores lógicos:
  - `&&` Short-circuit AND
  - `||` Short-circuit OR
  - `&` AND
  - `|` OR
- Hay una función `xor`, pero no es un operador

# Control de Flujo: if

- bloque i f

```
i f a > b  
    tmp=a;  
    a=b;  
    b=tmp;  
end
```

```
i f rem(n, 2) ~= 0  
    M = odd_magi c(n)  
el sei f rem(n, 4) ~= 0  
    M = si ngl e_even_magi c(n)  
el se  
    M = doubl e_even_magi c(n)  
end
```

A diferencia de C, en Matlab no es necesario utilizar paréntesis en la expresión lógica

# Control de Flujo: for

- bucle for

```
for n = 3:32  
    r(n) = rank(magic(n));  
end
```

```
a=[];  
for n = [ 1 2 3 5 7 11 ]  
    a = [a, isprime(n)];  
end
```

# Control de Flujo: while

- bucle while

```
while ~isprime(x)
    x = x + 1;
end
```



# Control de Flujo: switch

- switch-case

```
switch (rem(n, 4)==0)+(rem(n, 2)==0)
    case 0
        M = odd_magic(n)
    case 1
        M = single_even_magic(n)
    case 2
        M = double_even_magic(n)
    otherwise
        error('This is impossible')
end
```

A diferencia de C, en Matlab no hace falta utilizar break.

# Control de Flujo: try

- try-catch

```
try
    statement
    ...
    statement
catch
    statement
    ...
    statement
end
```

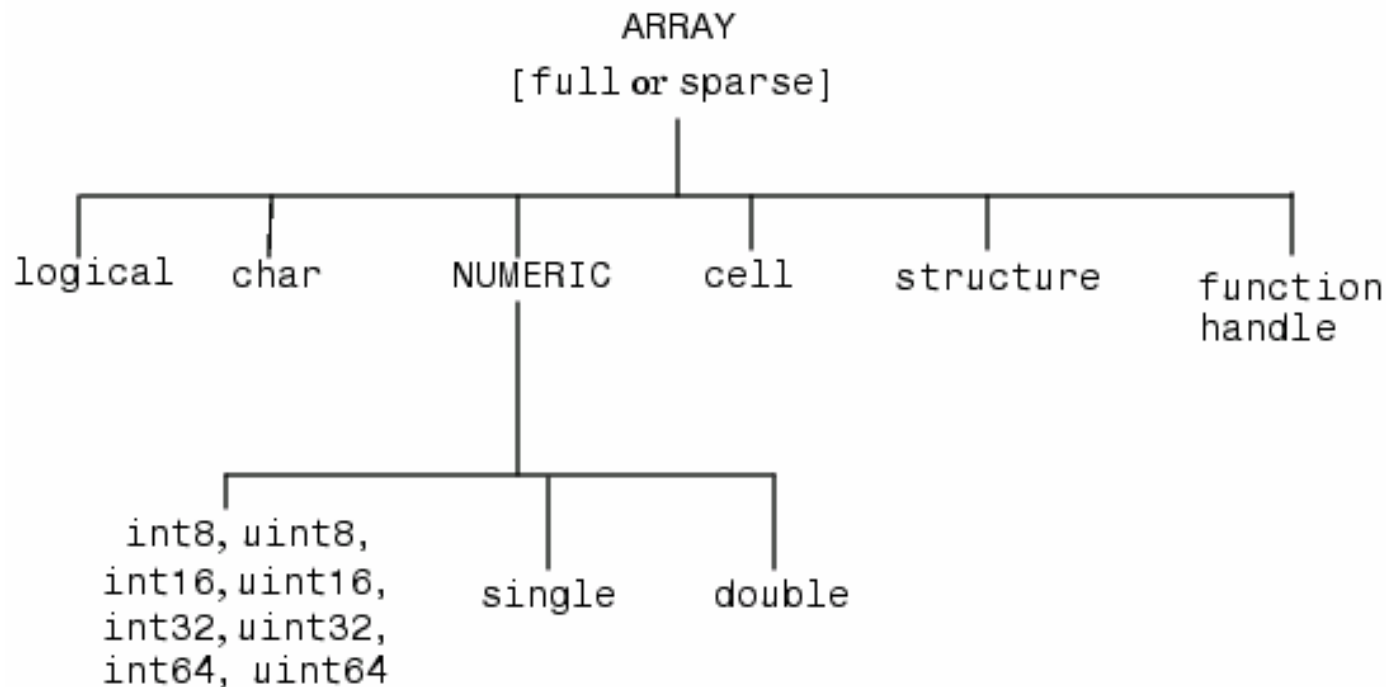
Las instrucciones comprendidas entre catch y end sólo se ejecutan si se produce un error en las primeras. Utilizar `lasterr` para ver el último error.

# Temario (4)

1. Introducción a Matlab.
2. Estructuras básicas de datos.
3. Programación en Matlab.
- 4. Estructuras avanzadas de datos.**
  - Todos los tipos de datos
  - Cadenas de caracteres
  - Estructuras y Cells arrays
  - Matrices de N dimensiones (arrays)
  - Date and time
5. Optimización de código.
6. Representaciones gráficas.
7. Desarrollo de aplicaciones con Matlab.

# Todos los tipos de datos

- Matlab tiene en total 15 tipos de datos que se utilizan para formar matrices o arrays



Adicionalmente existen tipos de datos definibles por el usuario para programación orientada a objetos: *user classes*, y *Java classes*

# Identificación del tipo de dato

- Descripción del tipo de dato

```
>> tipo=class(x)
tipo =
double
>>
```

- Identificación lógica

```
isinteger(x)
isfloat(x)
ischar(x)
islogical(x)
iscell(x)
isstruct(fecha)
```

```
int8, uint8
int16, uint16
int32, uint32
int64, uint64
```

```
single
double
```

```
isempty([])
isinf(Inf)
isnan(NaN)
```

# Conversión de tipos numéricos

- La conversión se realiza utilizando el nombre del tipo como si fuese una función

```
>> a=randn(5, 7)
```

```
a =
```

-0.4326	1.1909	-0.1867	0.1139	0.2944	0.8580	-0.3999
-1.6656	1.1892	0.7258	1.0668	-1.3362	1.2540	0.6900
0.1253	-0.0376	-0.5883	0.0593	0.7143	-1.5937	0.8156
0.2877	0.3273	2.1832	-0.0956	1.6236	-1.4410	0.7119
-1.1465	0.1746	-0.1364	-0.8323	-0.6918	0.5711	1.2902

```
>> b=int8(a)
```

```
b =
```

0	1	0	0	0	1	0
-2	1	1	1	-1	1	1
0	0	-1	0	1	-2	1
0	0	2	0	2	-1	1
-1	0	0	-1	-1	1	1

*Matlab aplica redondeo en la conversión a enteros*

# Conversión de tipos numéricos

- Funciones de redondeo
  - round: redondea al entero más próximo
  - floor: redondea hacia -Inf
  - ceil: redondea hacia +Inf
  - fix: redondea hacia cero

```
>> class(round(3.5))  
ans =  
double
```

No cambia el tipo de dato

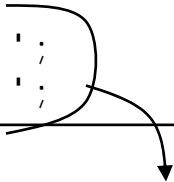
```
>> b(4.7)  
??? Subscript indices must either be real positive integers or logicals.  
  
>> b(round(4.7))  
ans =  
-1
```

Es útil para calcular índices

# Cadenas de caracteres

- En Matlab las cadenas de caracteres son vectores de tipo char (igual que en C)

```
>> str='Hello World';  
>> whos  
    Name      Size      Bytes  Class  
    str       1x11       22    char array  
Grand total is 11 elements using 22 bytes  
  
>> str(7)  
ans =  
W  
  
>> str=['H' , 'o' , 'l' , 'a'];  
  
>> nombres(1,:)='Rafael';  
>> nombres(2,:)='Ana';
```



Para utilizar variables que contengan más de una cadena de caracteres es necesario que todas las cadenas tengan la misma longitud. Esto se facilita con la función de conversión char:

```
>> nombres=char('Rafael' , 'Ana');
```

Utilizando **cell arrays** se puede almacenar cadenas de distintas longitudes



# Cadenas de caracteres

- Comparación de cadenas de caracteres
  - El operador `==` trata las cadenas como vectores

```
A = 'fate';  
B = 'cake';  
  
A == B  
ans =  
     0     1     0     1
```

- **strcmp**: compara cadenas y retorna 1 para cierto y 0 para falso. (OJO: distinto que en C).
- Otras funciones de comparación: **strncmp**, **strcmpi**, **strncmpi**

# Cadenas de caracteres

- Otras funciones de cadenas de caracteres
  - **strrep**: típico find-and-replace  
`cadena=strrep(cadena, ' busca' , ' susti tuye' );`
  - **findstr**: busca una cadena dentro de otra  
`posi ci on = fi ndstr(' busca' , cadena);`
  - **strcat**: concatena 2 o más cadenas  
`texto = strcat(cadena1, cadena2, cadena3);`
  - **sprintf**: construye una cadena a partir de variables. Equivale a `printf` de C  
`cadena=sprintf(' Tengo %6. 2f EUR' , mi _di nero);`

# Estructuras y cell arrays

- Las estructuras permite almacenar valores de diferente naturaleza bajo un nombre de variable

```
>> punto.x=123;  
>> punto.y=34;  
>> punto.col or=' red' ;  
>> punto  
punto =  
      x: 123  
      y: 34  
  col or: ' red'
```

- No requiere definición previa
- se accede a los campos igual que en C
- Vectores de estructuras se obtienen dinámicamente añadiendo elementos

```
>> punto(2).x=435;
```

# Estructuras y cell arrays

- Se puede acceder a un campo concreto poniendo el nombre del campo en una cadena de caracteres:

```
>> punto.x=123;
>> punto.y=34;
>> punto.col or=' red' ;
>> punto
punto =
      x: 123
      y: 34
  col or: ' red'

>> campo=' col or' ;
>> punto.(campo)
ans =
red
```

# Estructuras y cell arrays

- Un cell array permite construir vectores en las que cada elemento es de un tipo diferente:

```
>> c={12, 'Red', magic(4)};
>> c
C =
    [12]    'Red'    [4x4 double]

>> b{1}=12;
>> b{2}='Red';
>> b{3}=magic(4);
>> b
b =
    [12]    'Red'    [4x4 double]
```

- Se utilizan llaves { } en lugar de corchetes [ ] o paréntesis ( )
- La diferencia con las estructuras es que se accede a los valores utilizando un índice en lugar del nombre del campo
- Trabajar con estructuras es muy ineficiente

# Estructuras y cell arrays

- Un cell array permite contruir matrices en las que cada fila es diferente:

```
>> a{1,1} = 12;
>> a{1,2} = 'Red' ;
>> a{1,3} = magic(4);
>> a{2,1}=ones(3);
>> a{2,2}=43;
>> a{2,3}='texto' ;
>> a
a =
      [          12]      'Red'      [4x4 double]
      [3x3 double]      [ 43]      'texto'
>>
```

¿útil para algo?

# Estructuras y cell arrays

- Utilizando ( ) accedo a un elemento, que es tipo cell
- Utilizando { } accedo al valor

```
>> a
a =
    [          12]    ' Red'    [4x4 double]
    [3x3 double]    [ 43]    ' texto'

>> class(a)
ans =
cell

>> class(a(1,1))
ans =
cell

>> class(a{1,1})
ans =
double
```

# Estructuras y cell arrays

- Ejemplo de acceso a base de datos (database toolbox)

```
conn=database('base_de_datos_ODBC','usuario','password');
query='SELECT to_number(PROD),HORA,EST FROM TB_CENT WHERE CENTRAL=''ROBLA''';
curs=exec(conn,query);
curs=fetch(curs);

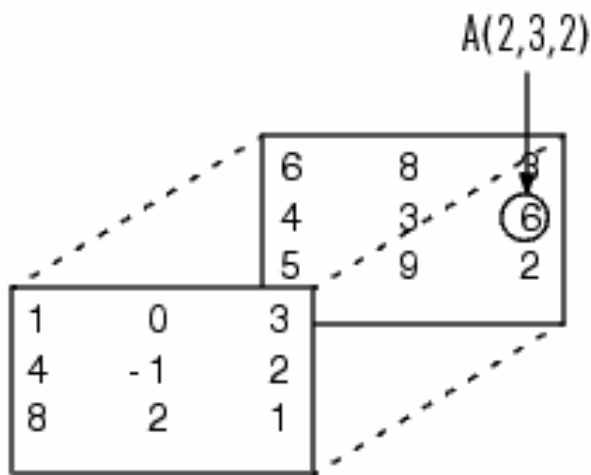
for i=1:size(curs.Data,1)
    producciones(i)=curs.Data{i,1};
    horas(i)=curs.Data{i,2};
    estados(i)=curs.Data{i,3};
end

close(curs);
close(conn);
```



# Matrices de N dimensiones

- Las matrices de más de 2 dimensiones se llaman **Multidimensional Arrays**
- Matlab soporta todas las operaciones matemáticas en matrices de N dimensiones



$A(:, :, 1) =$

1	0	3
4	-1	2
8	2	1

$A(:, :, 2) =$

6	8	3
4	3	6
5	9	2

# Matrices de N dimensiones

```
>> c=imread('autumn.tif');
```

```
>> whos c
```

Name	Size	Bytes	Class
c	206x345x3	213210	uint8 array

Grand total is 213210 elements using 213210 bytes

```
>> imshow(c)
```

```
>> max(c(:))
```

ans =

248

todos los elementos

```
>> gris=(c(:,:,1)+c(:,:,2)+c(:,:,3))/3;
```

```
>> imshow(gris)
```



# Date and Time

- Matlab representa fechas y horas en tres formatos:
  - cadena de caracteres
  - valor numérico (número de días desde 1/ene/0000)
  - vector numérico [año, mes, día, hora, min, sec]

Date Format	Example	
Date string	02-Oct-1996	date
Serial date number	729300	now
Date vector	1996 10 2 0 0 0	clock

- Tiene en cuenta años bisiestos
- No tiene en cuenta hora UTC/hora local ni cambios de hora invierno/verano

# Date and Time

- Funciones de conversión

Function	Description
<a href="#">datetime</a>	Convert a date string to a serial date number.
<a href="#">datestr</a>	Convert a serial date number to a date string.
<a href="#">datevec</a>	Split a date number or date string into individual date elements.

← también fprintf

- Ejemplo de manejo de fechas

```
function fecha_corregida=FechaCambio(fecha_calculo,dias)
%
%Obtiene una nueva estructura de fecha adelantando o retrasando dias
%function fecha_corregida=FechaCambio(fecha_calculo,dias)
%  fecha_corregida y fecha_calculo son estructuras con los campos dia, mes, aNo.
%
%Ejemplo: function fecha_corregida=FechaCambio(fecha_calculo,-1); %dia anterior
%
%Rafael Palacios Nov/2004
%
fecha_num=datetime(fecha_calculo.aNo,fecha_calculo.mes,fecha_calculo.dia);
fecha_num=fecha_num+dias;
fecha_vec=datevec(fecha_num);
fecha_corregida.aNo=fecha_vec(1);
fecha_corregida.mes=fecha_vec(2);
fecha_corregida.dia=fecha_vec(3);
```

# Temario (5)

1. Introducción a Matlab.
2. Estructuras básicas de datos.
3. Programación en Matlab.
4. Estructuras avanzadas de datos.
- 5. Optimización de código.**
  - Medida de tiempos: tic, toc, cputime
  - Análisis del código: profiler, M-Lint
  - Orden de los bucles
  - Predeclaración de variables
  - Find en lugar de for
  - Variables globales
  - Compilador
6. Representaciones gráficas.
7. Desarrollo de aplicaciones con Matlab.

# Medida de tiempos

- Funciones básicas para medir tiempos
  - `tic` y `toc` miden el tiempo en segundos

```
>> tic; inv(inv(inv(randn(1000)))); toc  
Elapsed time is 10.015000 seconds.
```

```
tic  
    for k = 1:100  
        -- programa rápido --  
    end  
toc
```

- `cputime` indica el tiempo de CPU en segundos

```
>> t=cputime; inv(inv(inv(randn(1000)))); e=cputime-t  
e =  
    9.5137
```

# Análisis del código

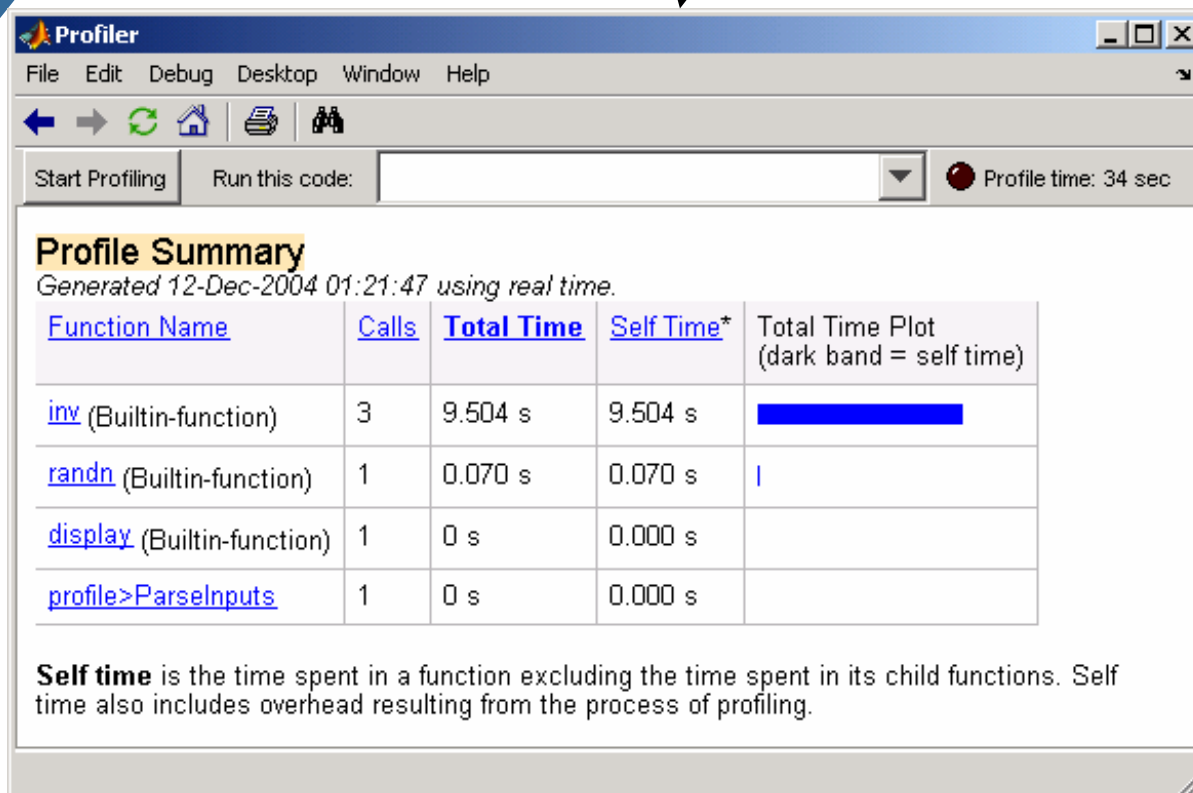
- profiler genera un informe del rendimiento de un programa

Modo gráfico:

```
>> profile viewer
```

Modo comandos:

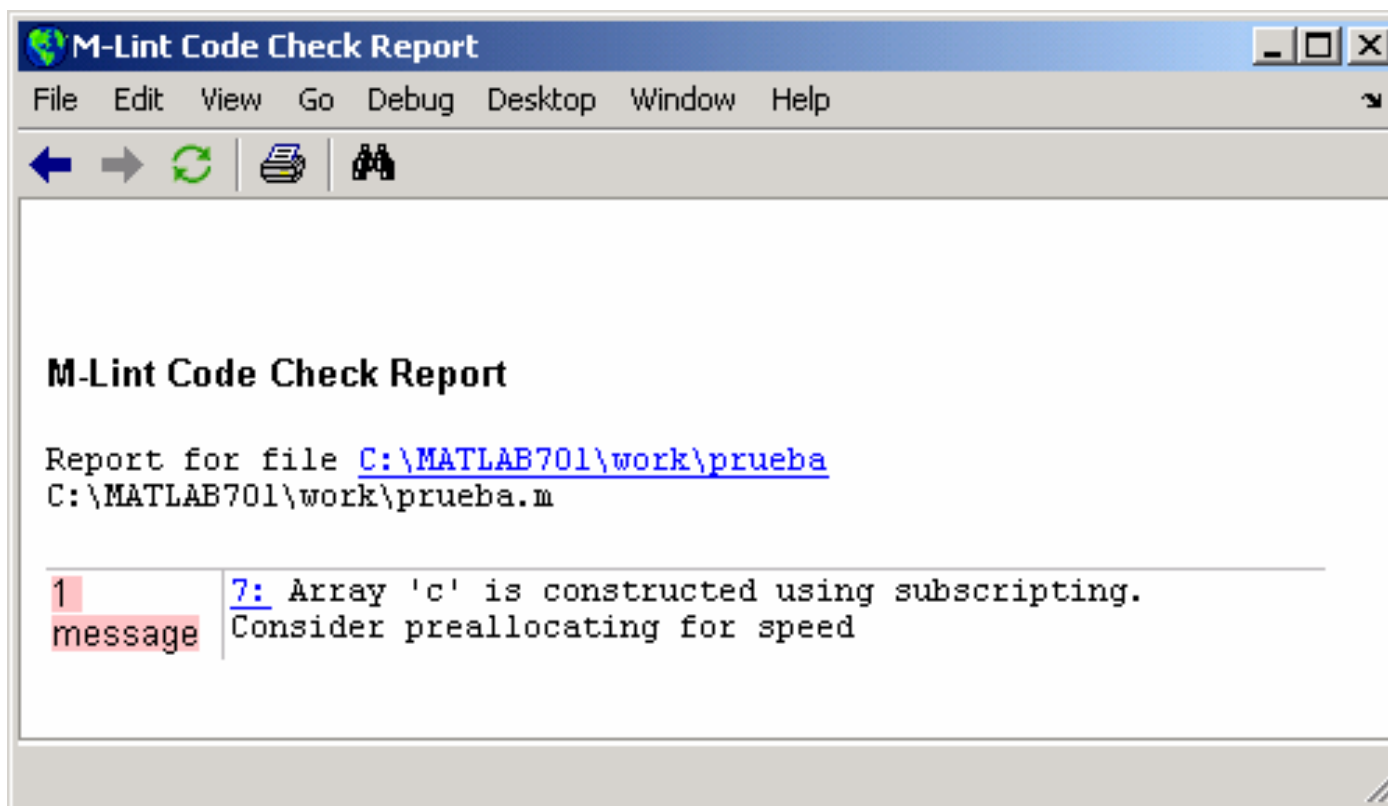
```
>> profile on  
>> inv(inv(inv(randn(1000))));  
>> profile off  
>> profile report
```



Profiler nos indica qué función merece la pena optimizar.

# Análisis del código

- M-Lint analiza el código e identifica posibles problemas y posibles puntos de mejora de rendimiento.





# Optimización de bucles

- Al ser un lenguaje interpretado, los bucles son muy lentos

```
>> tic, for t=1:100, prueba, end, toc  
Elapsed time is 3.856000 seconds.  
  
>> tic, for t=1:100, prueba2, end, toc  
Elapsed time is 2.554000 seconds.  
  
>> tic, for t=1:100, suma=sum(z(:)); end, toc  
Elapsed time is 1.893000 seconds.
```

```
%prueba2  
suma=0;  
for i=1:1000  
    for j=1:1000  
        suma=suma+z(j,i);  
    end  
end
```

```
%prueba  
suma=0;  
for i=1:1000  
    for j=1:1000  
        suma=suma+z(i,j);  
    end  
end
```

# Predeclaración de variables

- La predeclaración evita que Matlab reasigne memoria dinámicamente

```
>> tic, prueba, toc  
Elapsed time is 54.589000 seconds.
```

```
>> tic, for t=1:100, prueba2, end, toc  
Elapsed time is 10.846000 seconds.
```

500 veces más rápido!!!!

```
%prueba2  
z2=ones(size(z));  
for i=1:1000  
    for j=1:1000  
        z2(j,i)=z(j,i);  
    end  
end
```

```
%prueba  
for i=1:1000  
    for j=1:1000  
        z2(j,i)=z(j,i);  
    end  
end
```

Nota: Estos tiempos no mejoran utilizando el compilador porque son retrasos del sistema operativo, no de Matlab. En Unix el código malo es 50 veces más lento, en lugar de 500 veces más lento.

# Find en lugar de for

- En muchas ocasiones se puede utilizar `find` en lugar de realizar un bucle
  - `find` devuelve un vector con los índices de un vector que corresponden a valores "true"

Ejemplo: Busco los pixels con nivel mayor de 200

```
>> tic, for t=1:100, prueba, end, toc  
Elapsed time is 2.293000 seconds.
```

```
>> tic, for t=1:100, length(find(c>200)); end, toc  
Elapsed time is 1.062000 seconds.
```

```
%prueba imagen  
num=0;  
for i=1:size(c,1)  
    for j=1:size(c,2)  
        for k=1:size(c,3)  
            if c(i,j,k)>200  
                num=num+1;  
            end  
        end  
    end  
end
```

Nota: este ejemplo no es muy significativo

# Find en lugar de for

Ejemplo2: Pongo a cero los pixels con nivel mayor de 200

```
>> c=imread('autumn.tif');  
>> tic, for t=1:100, prueba, end, toc  
Elapsed time is 2.293000 seconds.  
  
>> c=imread('autumn.tif');  
>> tic, for t=1:100, c(find(c>200))=0; end, toc  
Elapsed time is 0.611000 seconds.
```

`c>200` genera una matriz 3D de ceros y unos  
`find(c>200)` genera un vector con los índices que valen 1  
`c(find(c>200))` equivale a decir `c([23, 267, ...])`

```
%prueba imagen  
for i=1:size(c,1)  
    for j=1:size(c,2)  
        for k=1:size(c,3)  
            if c(i,j,k)>200  
                c(i,j,k)=0;  
            end  
        end  
    end  
end
```

# Find en lugar de for

- Otras funciones útiles al estilo de find son:
  - all : determina si todos los elementos son nonzero
    - if all (A>0.5)
  - any: determina si algún elemento es nonzero
    - if any(A>0.5)
  - reshape: reorganiza los elementos de una matriz para adaptarse a otras dimensiones
  - sort: ordena elementos y obtiene una tabla de índices de ordenación

```
function x=aleat(rango)
%% function x=aleat(rango)
% Genera una lista de números aleatorios no repetidos de tamaño rango
%
z=rand(1,rango);
[s,x]=sort(z);
```

# Variables globales

- En las llamadas a función el paso de variables es por valor
  - Hay muchas llamadas del tipo:  
`mi_fecha=Di aSi gui ente(mi_fecha);`
  - En general las funciones que transforman matrices muy grandes son ineficiente debido al uso de la memoria

Nota: Aunque desde el punto de vista del programador las variables van por valor, Matlab es "listo" no siempre copia la variable en el stack. Matlab generalmente hace el paso por referencia y sólo copia la variable si la función la modifica (para proteger los valores originales)

# Variables globales

- Utilizar variables globales no es muy estructurado, pero en caso de matrices grandes evita asignaciones de memoria

```
global GRAVITY  
GRAVITY = 32;  
y = falling((0:1:5)');
```

```
function h = falling(t)  
global GRAVITY  
h = 1/2*GRAVITY*t.^2;
```

# Compilador

- Convierte código Matlab a C y permite crear un ejecutable independiente
  - El programa resultante no requiere licencia
  - Ejecuta más deprisa por estar compilado
  - Compila funciones, no scripts

Instalación: La primera vez hay que seleccionar el compilador por defecto

```
mbuild -setup
```

Comando general para compilar

```
mcc -m prueba.m
```



# Compilador

- El compilador permite crear los siguientes elementos:
  - Aplicación independiente: `mcc -m file1.m`
  - Librerías de funciones: `mcc -l file1.m`
  - COM object (component object model)
  - Excel Add-in

# Compilador

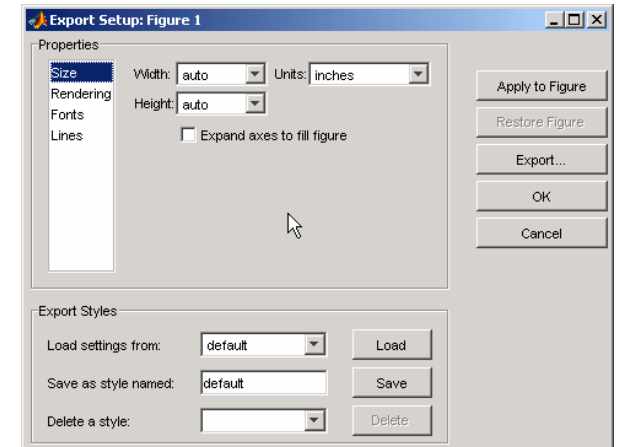
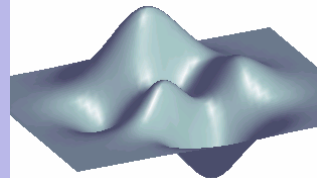
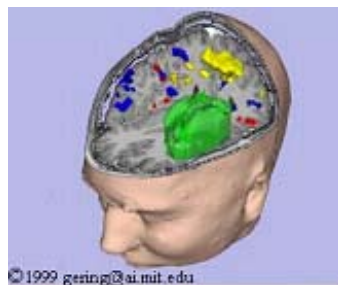
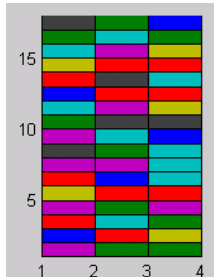
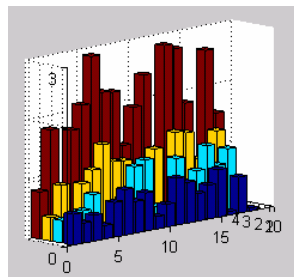
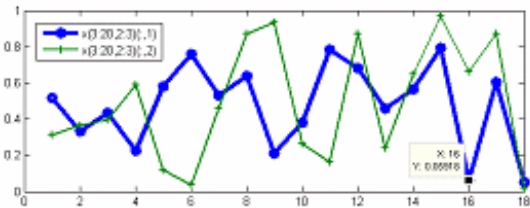
- Para instalar una aplicación en un ordenador que no tenga Matlab:
  - Crear un CD en el ordenador de desarrollo con:  
prueba.exe  
prueba.ctf  
<matlabroot>\toolbox\compiler\deploy\win32\MCRInstaller.exe
  - En el ordenador de destino:
    - Instalar MCRInstaller en C:\MCR (por ejemplo)
    - Asegurarse de que c:\MCR\runtime\win32 está en el PATH
    - Copiar prueba.exe y prueba.ctf al directorio de la aplicación.

# Temario (6)

1. Introducción a Matlab.
2. Estructuras básicas de datos.
3. Programación en Matlab.
4. Estructuras avanzadas de datos.
5. Optimización de código.
- 6. Representaciones gráficas.**
  - Tipos de gráficos
  - Crear gráficos con plot y surf
  - Retoque de gráficos desde menú
  - Guardar gráficos: hgsave, hgload, print
  - Creación de animaciones
7. Desarrollo de aplicaciones con Matlab.

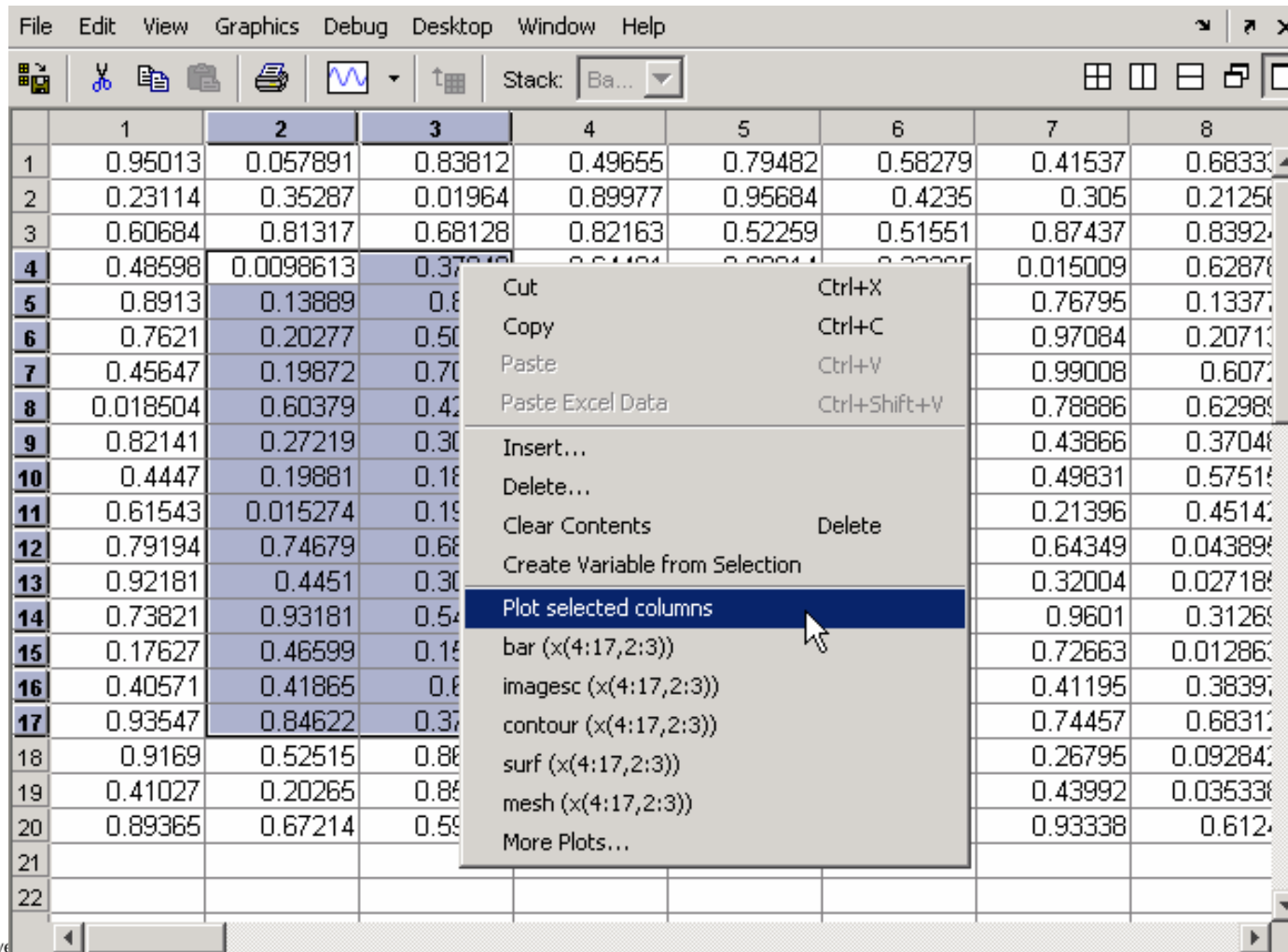
# Crear gráficos

- Matlab permite crear gráficos de varios tipos, que se utilizan para:
  - visualizar el contenido de las variables
  - crear imágenes/películas/VR/GIS
  - generar interfaces de usuario (ver capítulo 7)

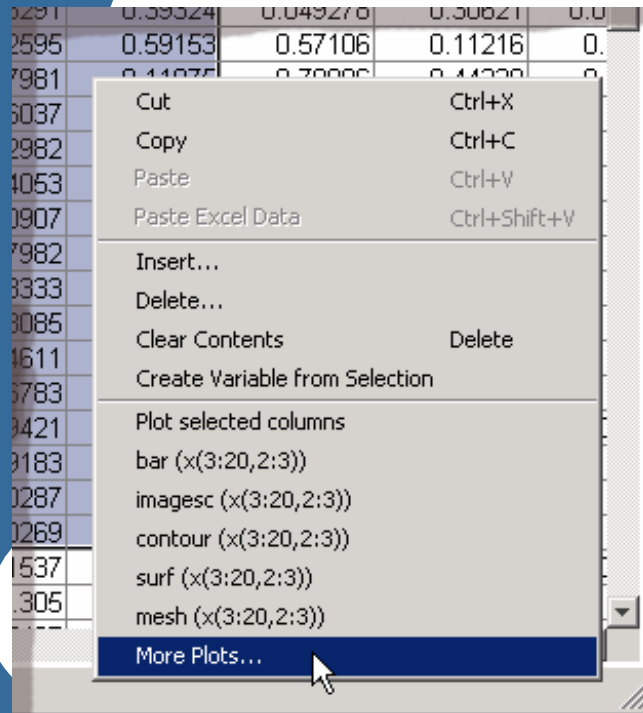


# Crear/ajustar gráficos

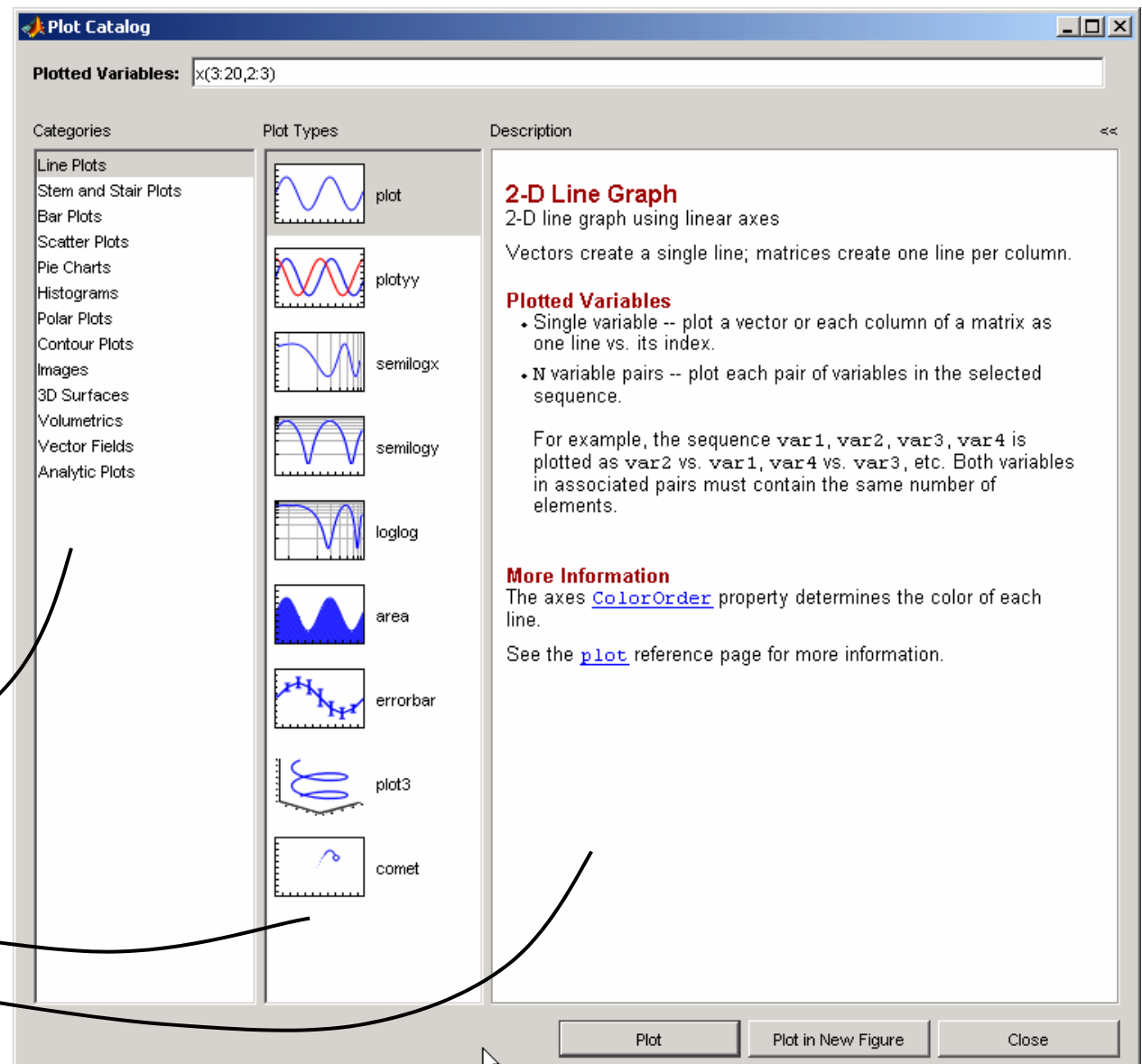
- Crear gráficos desde el editor de matrices (Permite representar filas, columnas o regiones mediante varios tipos de gráficos)



# Selección del tipo de gráfico



Matrix editor



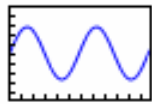
Categories

Plot types

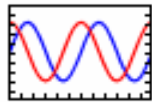
Description and function references

# Tipos de gráficos (1D, 2D)

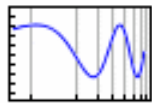
## Line



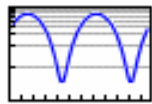
plot



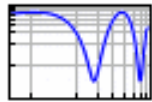
plotyy



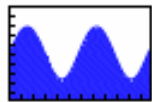
semilogx



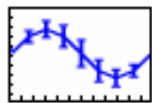
semilogy



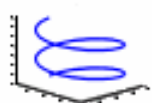
loglog



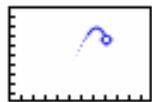
area



errorbar

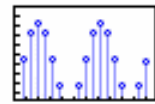


plot3

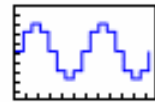


comet

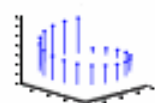
## Stem & stair



stem

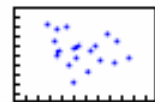


stairs

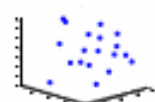


stem3

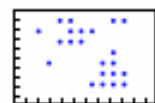
## Scatter



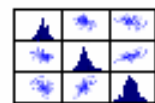
scatter



scatter3

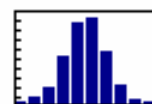


spy



plotmatrix

## Histogram



hist



rose

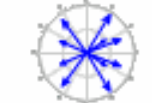
## Polar



polar

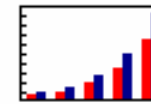


rose

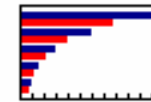


compass

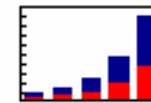
## Bar



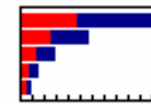
bar



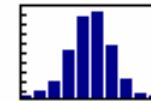
barh



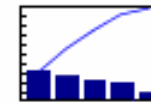
bar



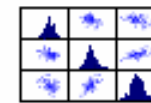
barh



hist



pareto



plotmatrix

## Pie



pie

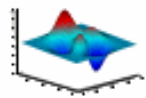


pie3

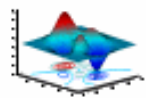


# Tipos de gráficos ( $\geq 3D$ )

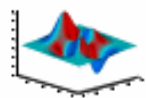
## 3D surfaces



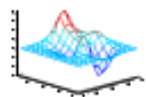
surf



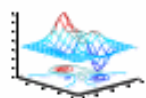
surfc



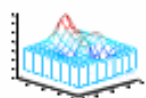
surfz



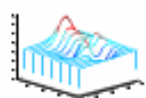
mesh



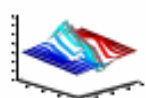
meshc



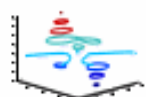
meshz



waterfall

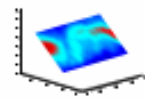


ribbon

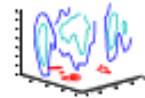


contour3

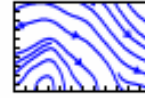
## Volumetrics



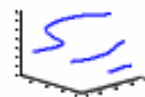
slice



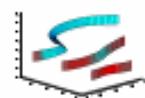
contourslice



streamslice



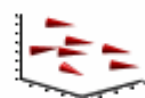
streamline



streamribbon

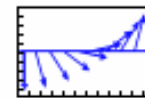


streamtube



coneplot

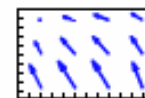
## Vector Fields



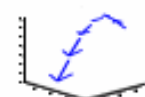
feather



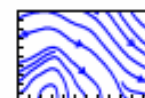
compass



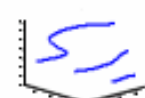
quiver



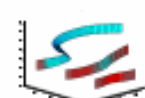
quiver3



streamslice



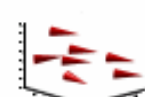
streamline



streamribbon

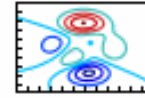


streamtube

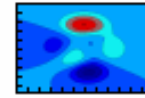


coneplot

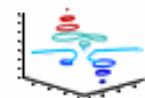
## Contour



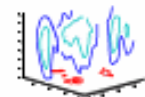
contour



contourf

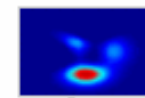


contour3

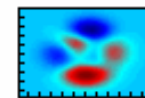


contourslice

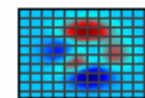
## Images



image

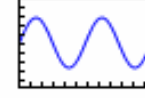


imagesc

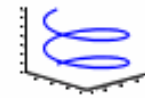


pcolor

## Analytic



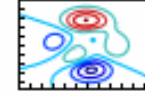
ezplot



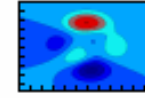
ezplot3



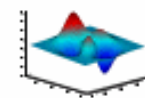
ezpolar



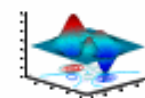
ezcontour



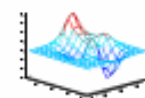
ezcontourf



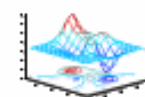
ezsurf



ezsurfc



ezmesh



ezmeshc



# Crear gráficos con plot

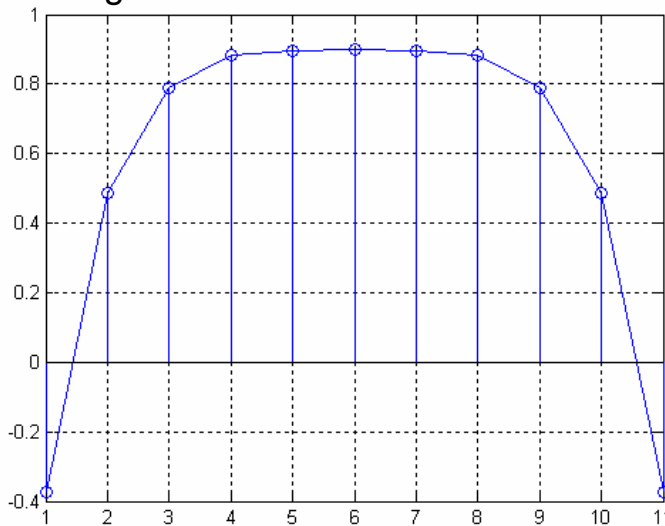
`plot(Y)`

`plot(X1, Y1, ...)`

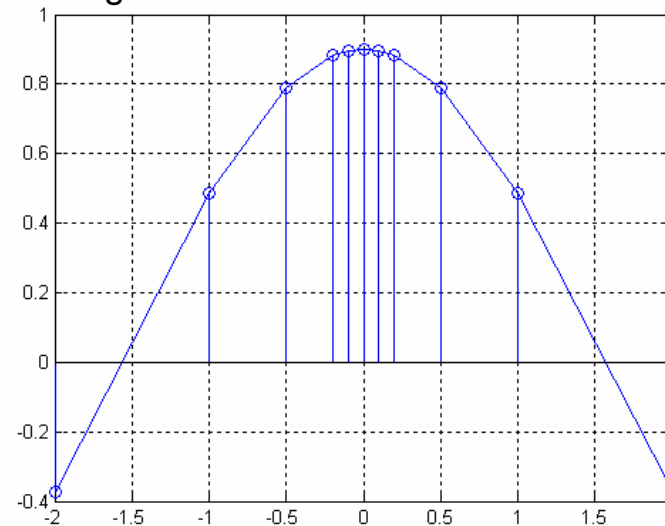
`plot(X1, Y1, LineSpec, ...)`

`plot(..., 'PropertyName', PropertyValue, ...)`

```
>> plot(yy,'o-')  
>> hold on; stem(yy); hold off  
>> grid on
```

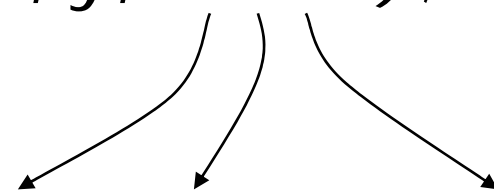


```
>> plot(xx,yy,'o-')  
>> hold on; stem(xx,yy); hold off  
>> grid on
```



# Crear gráficos con plot

```
plot(x, y, 'rx-');
```

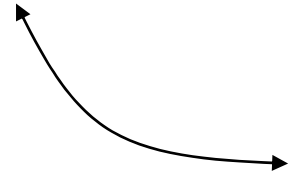


y	yellow	.	punto	-	línea continua
m	magenta	o		:	línea de puntos
c	cyan	x		-.	punto y raya
r	red	+		--	línea discontinua
g	green	*			
b	blue				
w	white				
k	black				

```
plot(x, y1, 'rx-', x, y2, 'g--');
```

Truco: Para pintar una línea:

```
hold on  
plot([x1, x2], [y1, y1], 'k');  
hold off
```



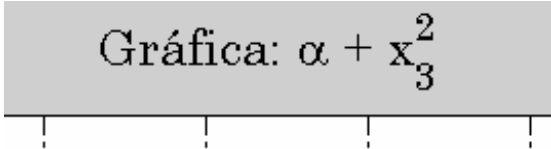
```
plot(x, y1, 'rx-');  
hold on;  
plot(x, y2, 'g--');  
hold off;
```

# Texto en gráficos

```
xlabel ('Eje X');  
ylabel ('Eje y');  
zlabel ('Eje z');  
title ('Título de la gráfica');  
text(x, y, 'Texto en (x, y)');
```

Truco: Los texto admiten expresiones tipo Latex

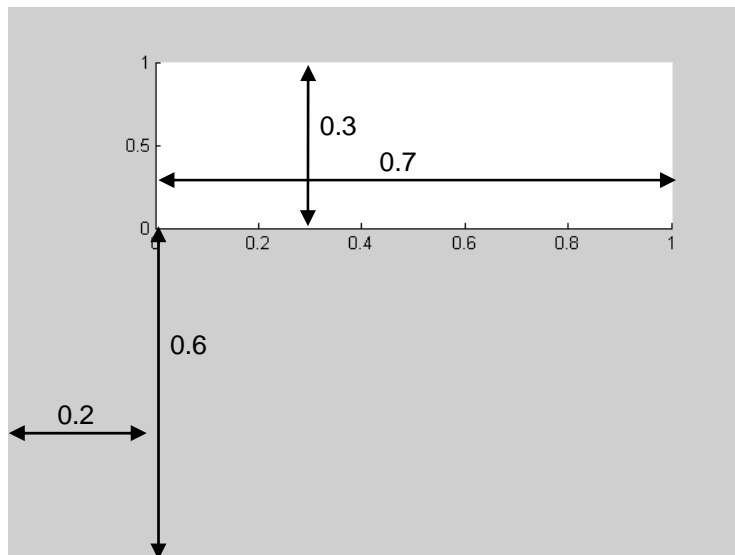
```
title('Gráfica: \alpha + x_3^2')
```



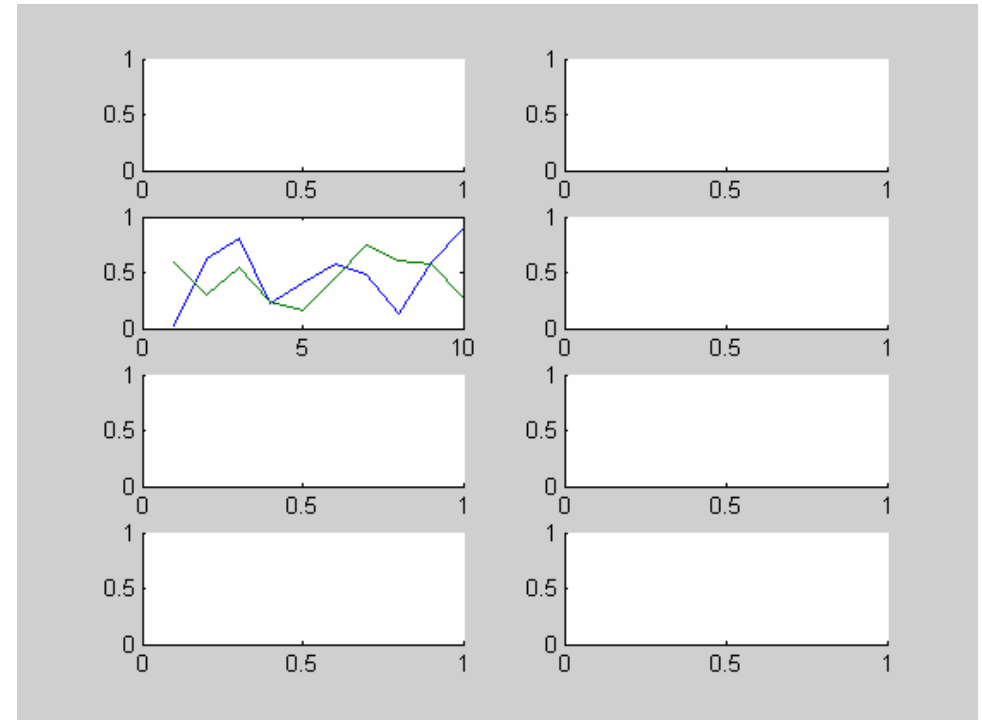
Gráfica:  $\alpha + x_3^2$

# Subplots: varios gráficos por figura

```
subplot(4, 2, 3)  
plot(rand(10, 2))
```



```
subplot('position', [0.2, 0.6, 0.7, 0.3])
```



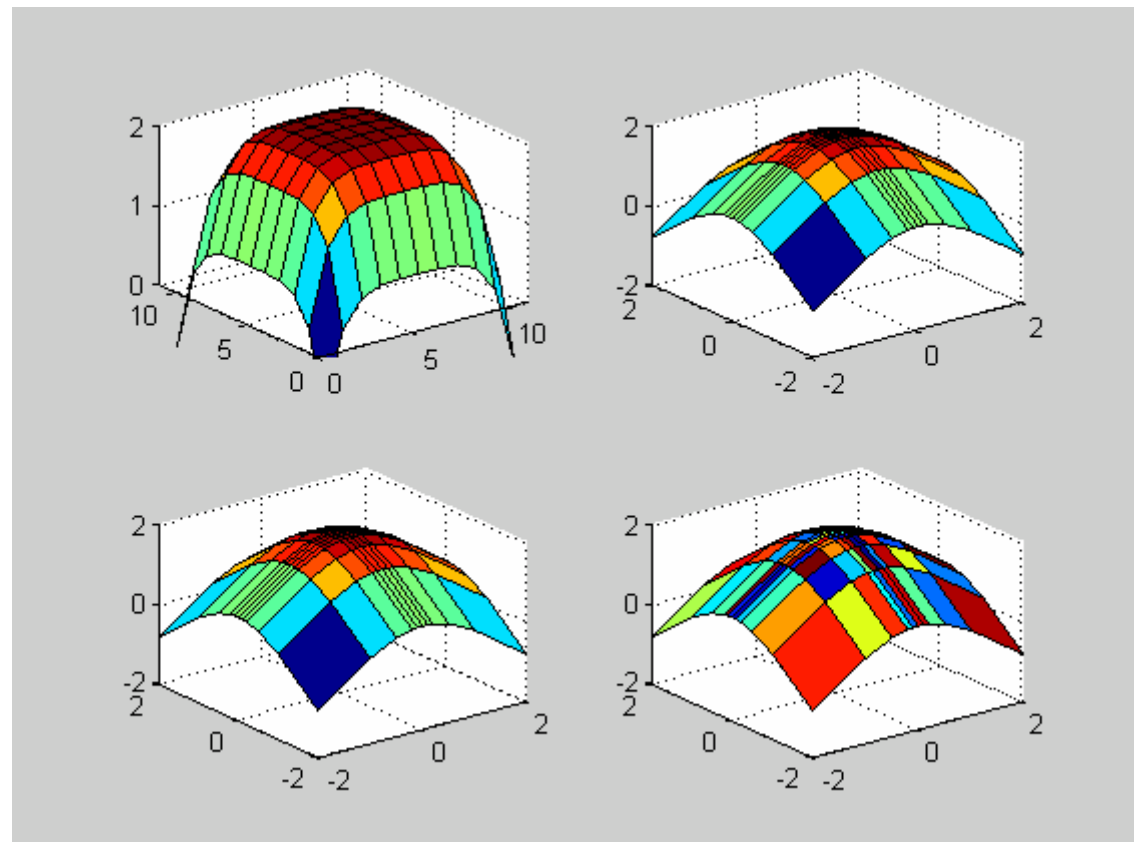
# Gráficos de superficies

surf(Z)

surf(X, Y, Z)

surf(X, Y, Z, C) colores

```
x=[-20,-10,-5,-2,-1,0,1,2,5,10,20]/10;  
y=x;  
  
[X,Y]=meshgrid(x,y);  
Z=cos(X)+cos(Y);  
  
subplot(2,2,1)  
surf(Z);  
axis([0 11 0 11 0 2])  
  
subplot(2,2,2)  
surf(x,y,Z);  
  
subplot(2,2,3)  
surf(X,Y,Z);  
  
subplot(2,2,4)  
C=rand(size(Z));  
surf(X,Y,Z,C);
```



# Superficies especiales

```
function [x,y,z]=torus()
% Dibuja un toro
%
r=0.5; %radio lateral
n=30; %número de elementos
a=1; %radio central

%Calculo ángulos en función de la resolución
theta=pi*(0:2:2*n)/n;
phi=2*pi*(0:2:n)'/n;

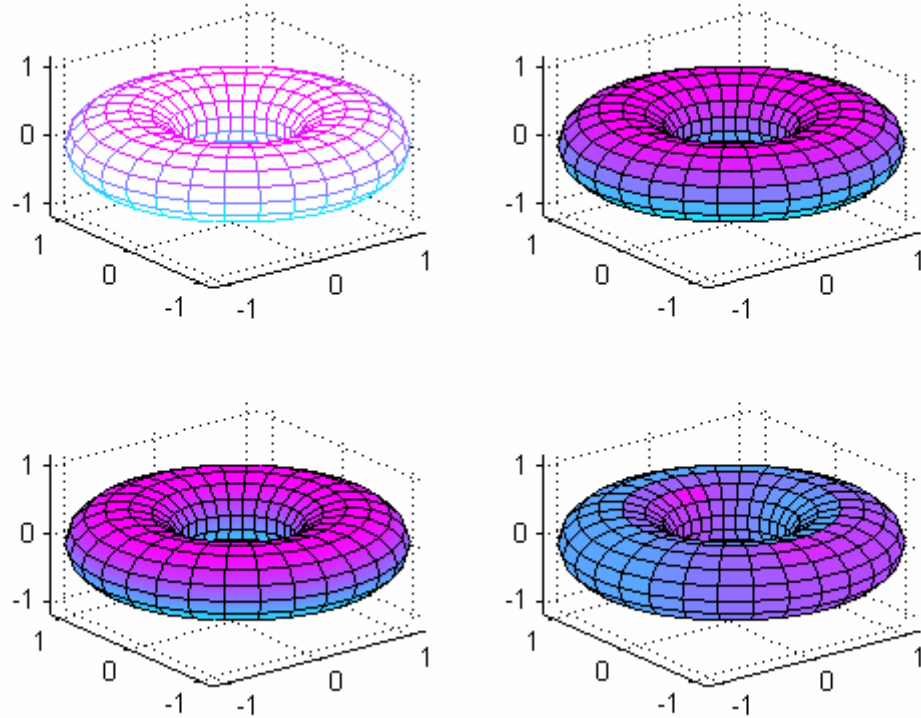
%Calculo y proyecto en x,y,z.
xx=(a + r * cos(phi))*cos(theta);
yy=(a + r * cos(phi))*sin(theta);
zz=r * sin(phi)*ones(size(theta));

%Dibujo la figura
ar=(a+r)/sqrt(2)*1.1;
colormap('cool')
subplot(2,2,1); mesh(xx,yy,zz);
axis([-ar,ar,-ar,ar,-ar,ar]);

subplot(2,2,2); surf(xx,yy,zz);
axis([-ar,ar,-ar,ar,-ar,ar]);

subplot(2,2,3); p=surf(xx,yy,zz);
shading interp
set(p,'EdgeColor','k');
axis([-ar,ar,-ar,ar,-ar,ar]);

subplot(2,2,4); surf1(xx,yy,zz);
axis([-ar,ar,-ar,ar,-ar,ar]);
```

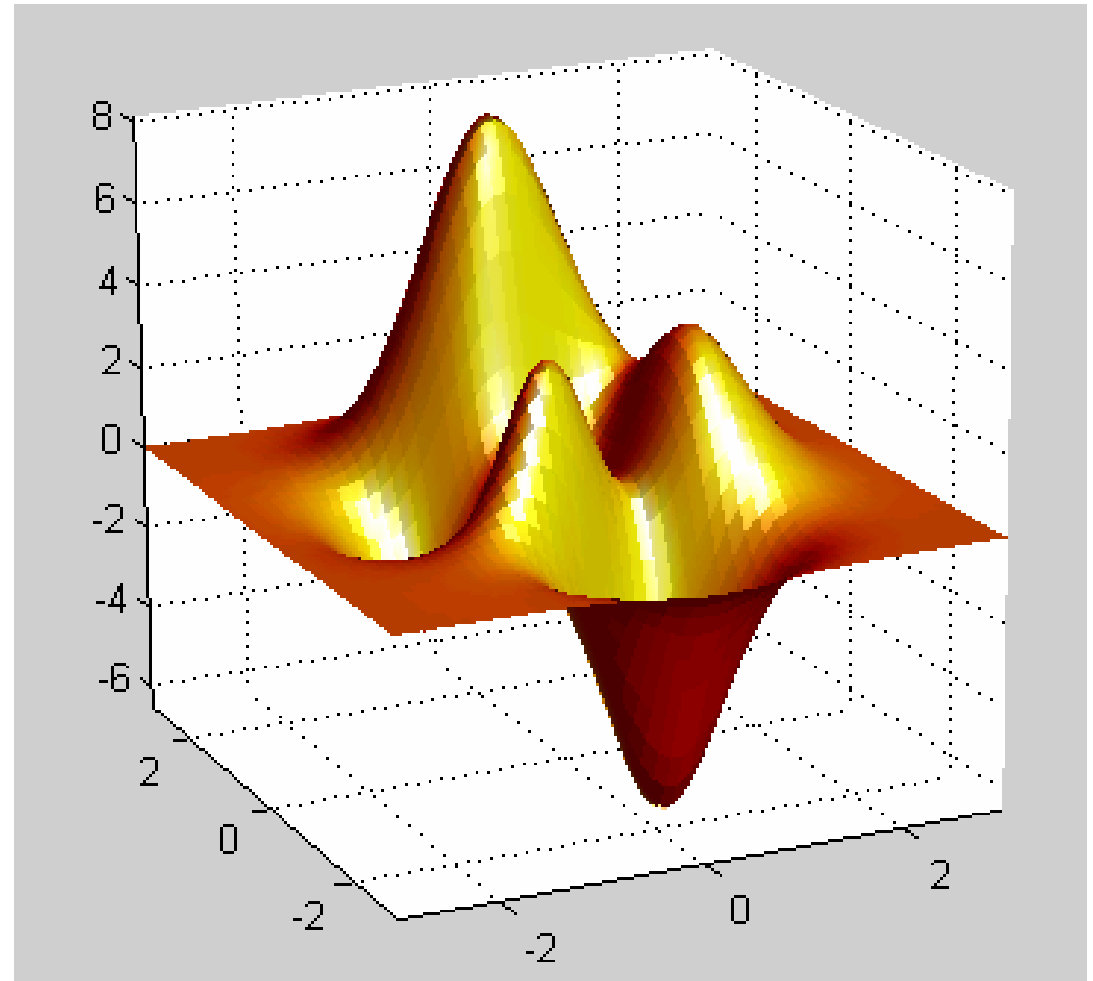


# Iluminación y Punto de vista

```
[x, y, z]=peaks;  
surfl (x, y, z);  
shading interp
```

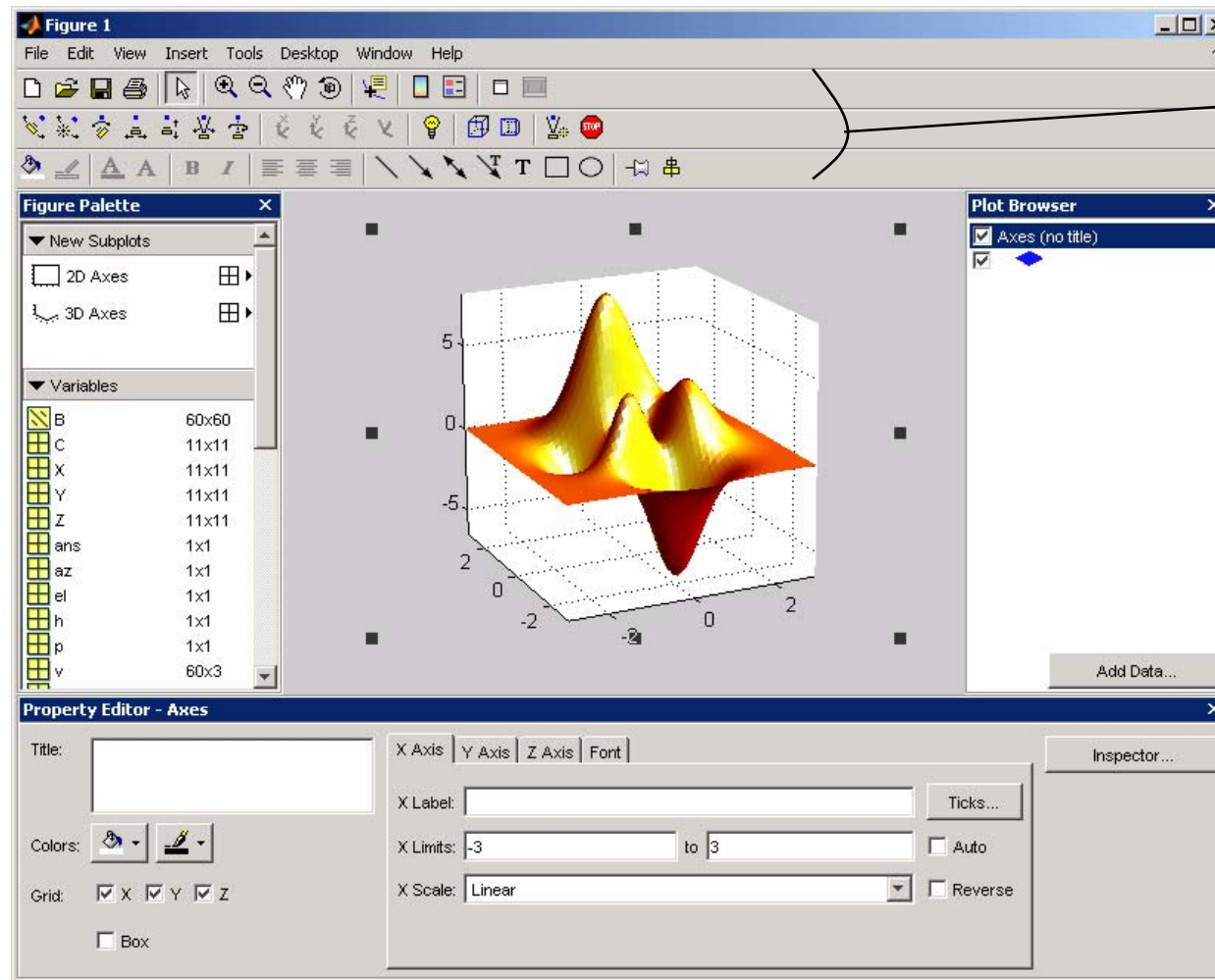
```
Punto de vista  
view(azimuth, elevation)  
view(-37.5, 30)
```

```
Iluminación  
lightangle(az, el )  
lightangle(90, 21.8)
```



# Retoque de gráficos desde menú

Ventana de la figura con todas las opciones activadas



Toolbars

Selector de objetos de la figura

Para crear nuevos subplot

Variables del workspace

Properties:

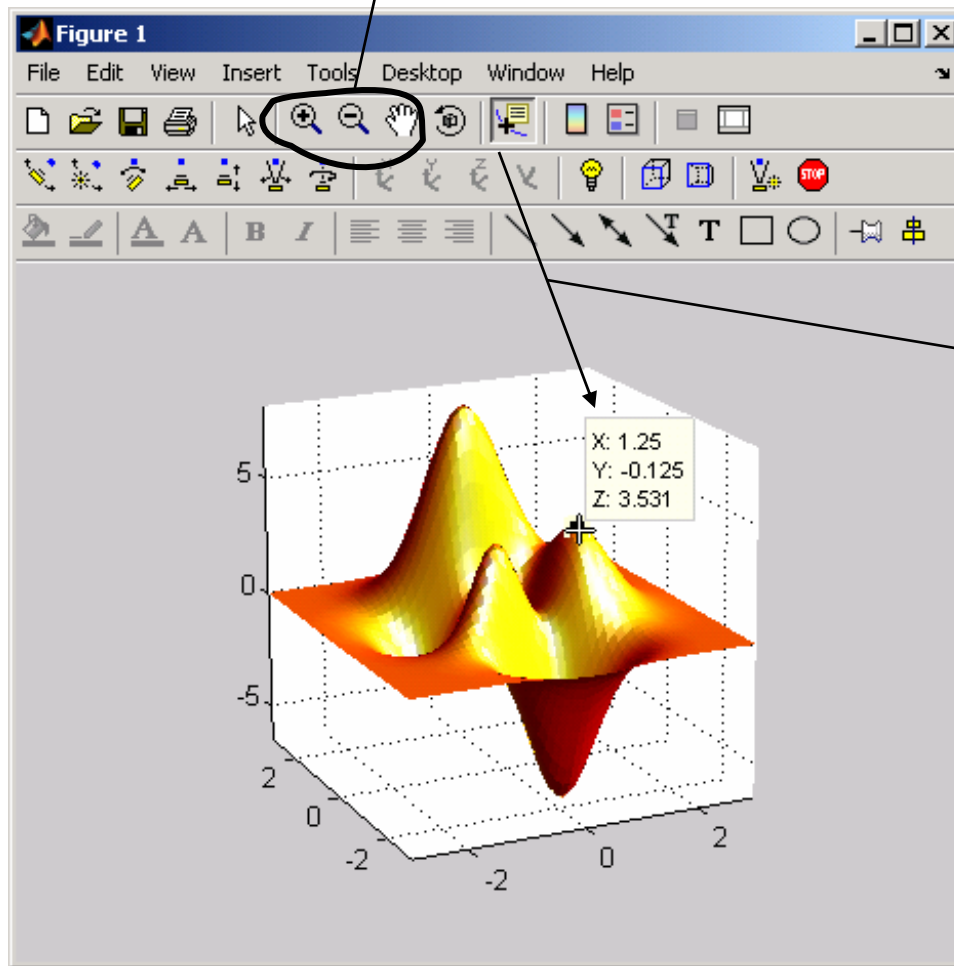
- Figure
- Axes
- Current Object



# Retoque de gráficos desde menú

## Figure Toolbar

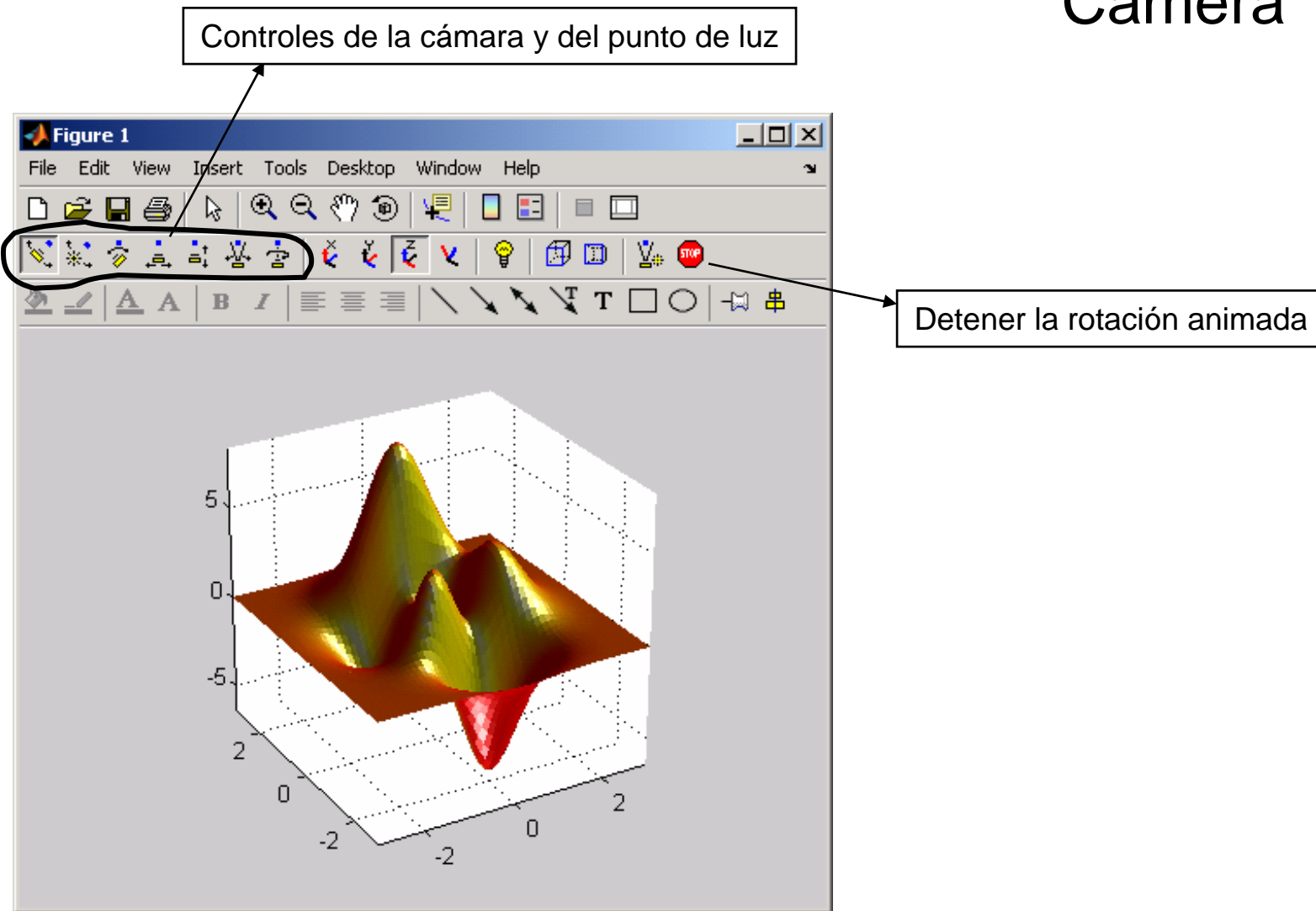
Zoom y desplazamiento de ejes



Explora valores pinchando en la gráfica

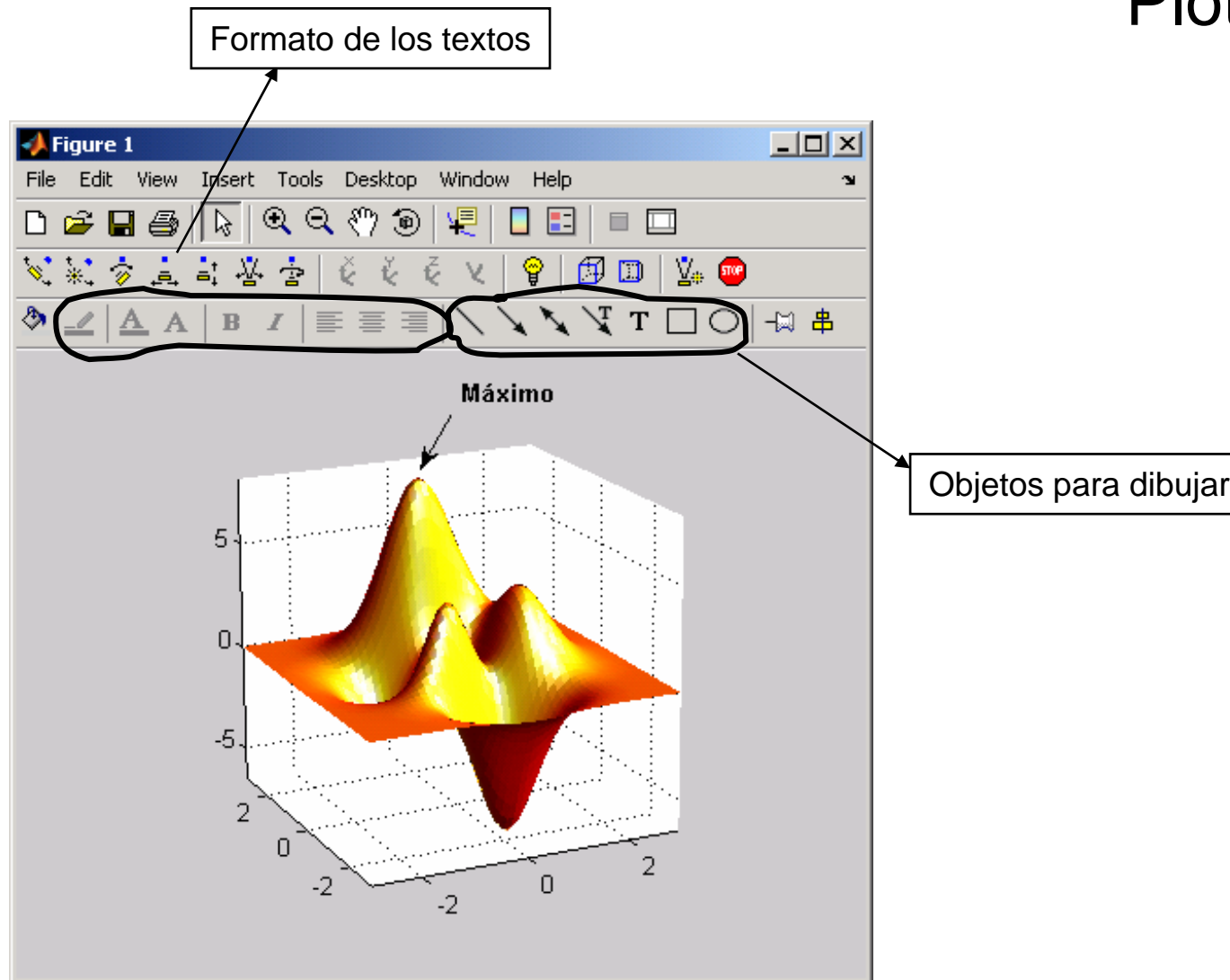
# Retoque de gráficos desde menú

## Camera Toolbar



# Retoque de gráficos desde menú

## Plot edit Toolbar



# Retoque de gráficos por propiedades

- Todas las propiedades de cada objeto del gráfico (figura, ejes, elemento gráfico,...) están guardadas en "handles"
- `gcf` → current figure, `gca` → current axis
- `get(handle)` muestra todas las propiedades que se pueden cambiar
- `set(handle, 'PropertyName', 'Value', ...)` cambia propiedades

- Ejemplo:

```
set(gca, 'Xtick', [1 2 3 4 5 6]);
```

```
set(gca, 'XtickLabel', ['ene'; 'feb'; 'mar'; 'abr'; 'may'; 'jun']);
```

# Guardar gráficos

- Desde menu de figure

- File/Save As → .fig, .eps, .png, .jpeg, .bmp, .pcx, .tiff
- File/Generate M file

Esta opción nos permite ver qué comandos se utilizan para crear las modificaciones que hemos realizado por menú

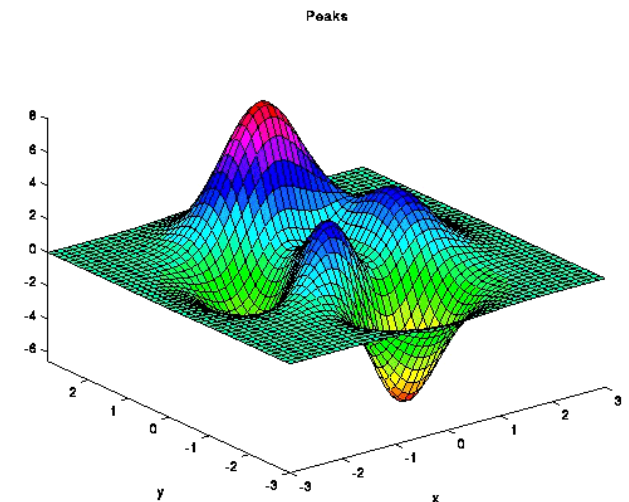
- Por comandos (útil para sesiones de consola)

- hgsave pepito → pepito.fig Se puede cargar con hgload para retocar
- print guarda la figura como imagen
  - `print -depsc -tiff -r300 archivo`
  - `print -dpng -r150 archivo`

# Creación de animaciones

- Hay dos maneras de crear animaciones:
  - Offline: generar una película para verla luego
  - On-Line: ir repintando la gráfica desde Matlab
- Las películas pueden generarse de dos maneras:
  - Guardar "fotogramas" en el disco (normalmente utilizando print) y luego utilizar un programa externo para crear la película.
  - getframe, movie

```
for k = 1:16  
    plot(fft(eye(k+16)))  
    axis equal  
    M(k) = getframe;  
end  
movie(M, 1); %play the movie  
movie2avi(M, 'mi_peli', 'fps', 1);
```



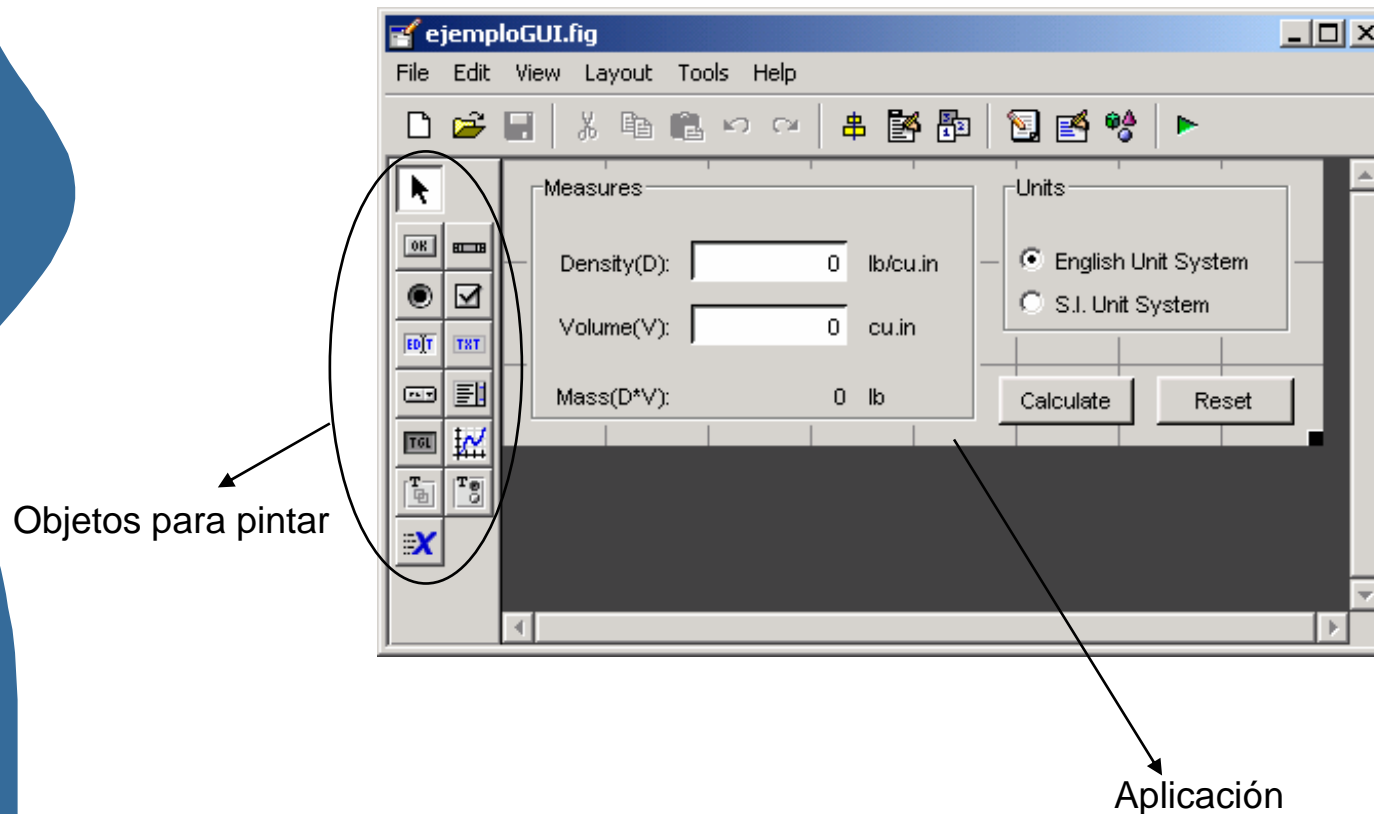
# Temario (7)

---

1. Introducción a Matlab.
2. Estructuras básicas de datos.
3. Programación en Matlab.
4. Estructuras avanzadas de datos.
5. Optimización de código.
6. Representaciones gráficas.
- 7. Desarrollo de aplicaciones con Matlab.**
  - Creación de interfaces gráficas
  - Métodos de comunicación externa
  - Generación de documentación

# Creación de interfaces gráficas

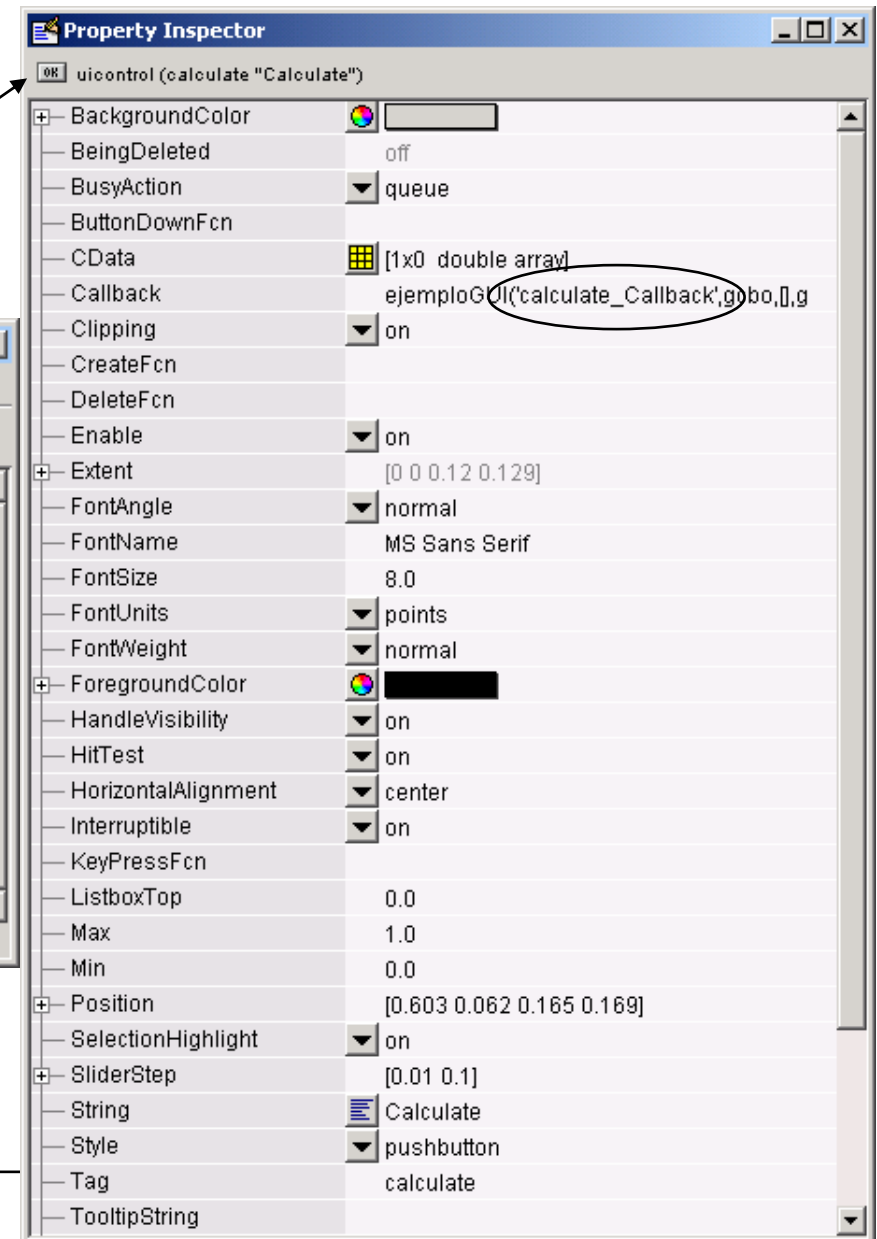
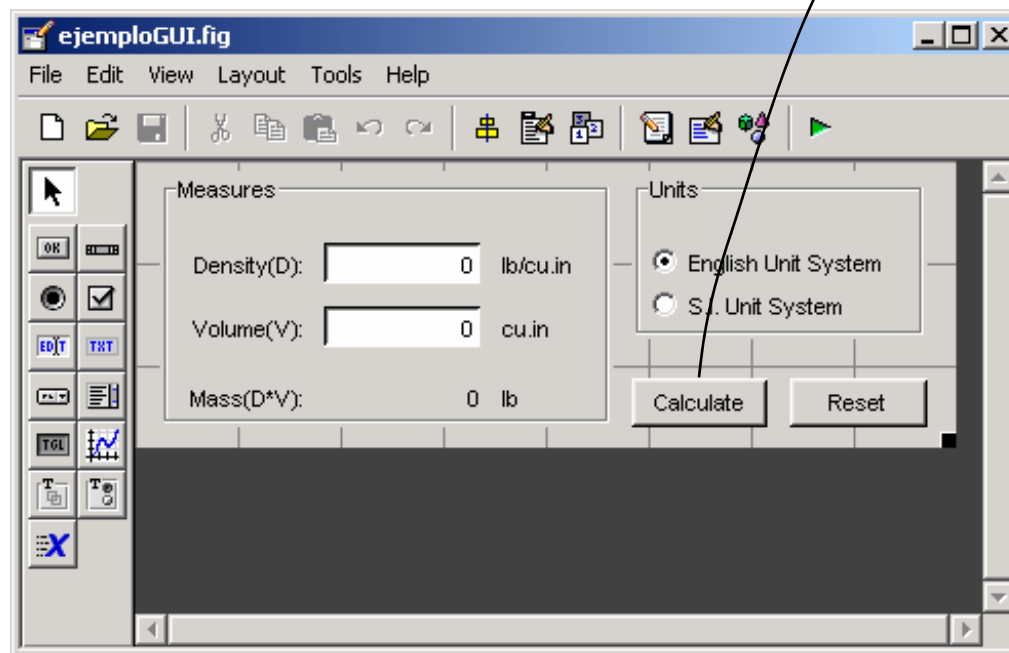
- Ejecutar la aplicación gui de desde Matlab





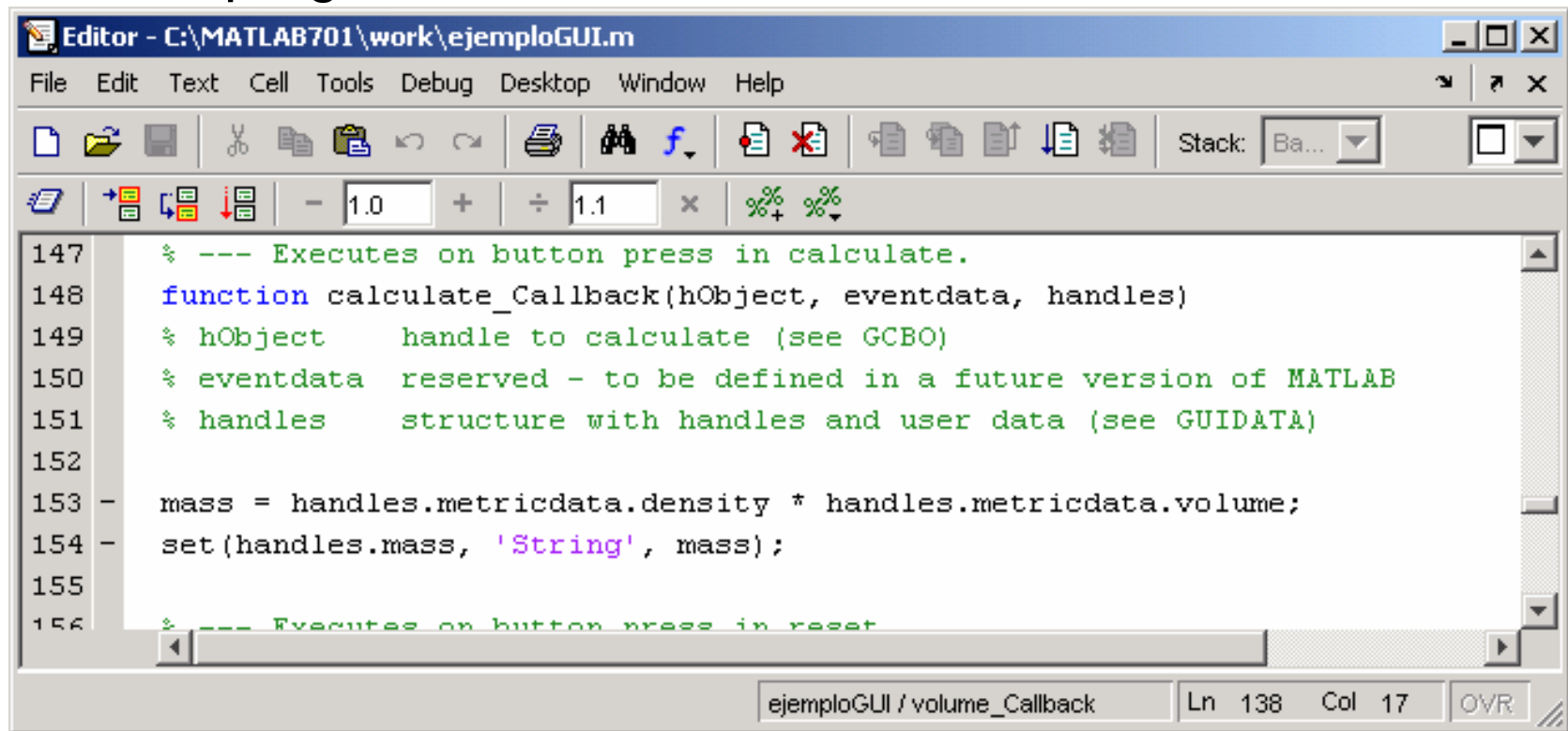
# Creación de interfaces gráficas

- Cada objeto tiene sus atributos y una función callback



# Creación de interfaces gráficas

- Guide genera un archivo .m para escribir el código de nuestro programa



```
147 % --- Executes on button press in calculate.
148 function calculate_Callback(hObject, eventdata, handles)
149 % hObject      handle to calculate (see GCBO)
150 % eventdata    reserved - to be defined in a future version of MATLAB
151 % handles      structure with handles and user data (see GUIDATA)
152
153 - mass = handles.metricdata.density * handles.metricdata.volume;
154 - set(handles.mass, 'String', mass);
155
156 % --- Executes on button press in reset
```

Como en todos los lenguajes, es aconsejable mantener el código de cálculo aislado del código del interface

# Comunicaciones externas

- Adquisición directa de datos
  - Database toolbox
  - Data Acquisition toolbox
  - Image Acquisition toolbox
- Adquisición de datos desde archivos
  - xlsread, load, textscan (números)
  - auread, wavread (sonido)
  - imread (imagen)
  - aviread (película)

# Comunicaciones externas

- Programas externos
  - Matlab llama a otros programas
    - system, dos, uni x
  - Matlab llama a funciones de otros lenguajes
    - crear un fichero MEX (`#include "mex.h"`)
    - Matlab llama a la función como si fuese un .m
  - Excel llama a Matlab
    - Excel Link toolbox
  - Cualquier programa llama a Matlab
    - `matlab -nodesplay -m programa`
    - `matlab -nodesktop -m programa`

# Generación de documentación

- Consejos iniciales
  - Documentar todas la funciones:
    - Descripción
    - Argumentos de entrada
    - Valores retornados
    - Ejemplo de utilización
    - Advertencias de utilización
  - Mantener el código de cálculo independiente del interface gráfico. Facilita la depuración, la mejora de rendimiento, la actualización del interface.
  - Crear secciones con comentarios del tipo %%

# Generación de documentación

- Matlab 7 incorpora una opción de generación de documentación.
  - De momento sólo funciona para scripts
  - Genera documentación en HTML, XML, LaTeX, Word y Power Point.
  - Se basa en los comentarios de las secciones
- Procedimiento:
  - Activar "cell mode" en el editor con Cell/Enable Cell Mode
  - Seleccionar File/Publish to HTML
  - Matlab ejecuta el script y genera un HTML con los comentarios, el código y gráficas de los resultados.

# Ejemplo: script torus.m

```
Editor - C:\MATLAB701\work\torus.m
File Edit Text Cell Tools Debug Desktop Window Help
Base

1 %convierto a script para generar documentación
2 %function [x,y,z]=torus()
3 % Dibuja un toro
4 %
5 r=0.5; %radio lateral
6 n=30; %número de elementos
7 a=1; %radio central
8
9 %% Calculo ángulos en función de la resolución
10 theta=pi*(0:2:2*pi)/n;
11 phi=2*pi*(0:2:n)/n;
12
13 %% Calculo y proyecto en x,y,z.
14 xx=(a+r*cos(phi))*cos(theta);
15 yy=(a+r*cos(phi))*sin(theta);
16 zz=r*sin(phi)*ones(size(theta));
17
18 %% Dibujo mesh
19 ar=(a+r)/sqrt(2)*1.1;
20 colormap('cool')
21 subplot(2,2,1); mesh(xx,yy,zz); axis([-ar,ar,-ar,ar]);
22
23 %% Dibujo Superficie
24 subplot(2,2,2); surf(xx,yy,zz); axis([-ar,ar,-ar,ar]);
25
26 %% Dibujo con interpolación de colores
27 subplot(2,2,3); p=surf(xx,yy,zz);
28 shading interp
29 set(p,'EdgeColor','k');
30 axis([-ar,ar,-ar,ar]);
31
32 %% Dibujo con iluminación
33 subplot(2,2,4); surf1(xx,yy,zz); axis([-ar,ar,-ar,ar]);
```

