Introducción a la Neurociencia Cognitiva y Computacional

Diego Fernández Slezak (dfslezak@dc.uba.ar)

Facultades de Ciencias Exactas y Naturales, UBA

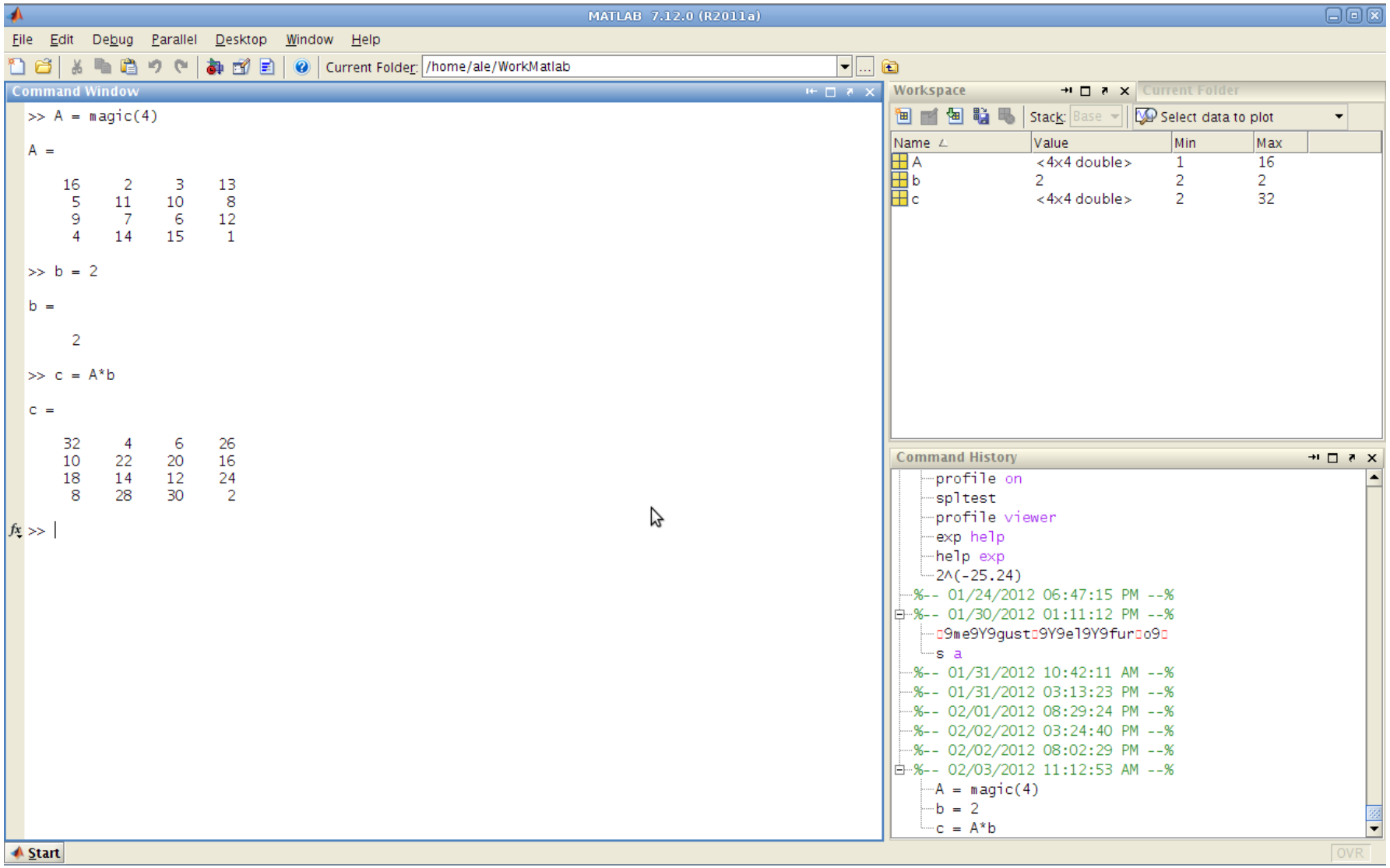
CONICET

Basado en material de A. Otero.

Matlab – ¿Qué y Por qué?

- Software de cálculo numérico
  - ejecución interactiva: consola de comandos
  - ejecución de programas:
    - funciones
    - scripts
- Librería de funciones matemáticas
  - Herramientas de Álgebra lineal
  - . . . de Cálculo
  - . . . Interpolación
  - . . . Optimización
  - . . .
  - Toolboxes de expansión
- Lenguaje de programación interpretado:
  - Matlab
  - Octave
  - SciLab
- Interfaces a otras librerías (por ej. solvers de sistemas de ecuaciones de alta performance)

Matlab – Entorno gráfico de desarrollo



Matlab – Entorno gráfico de desarrollo

- Consola de comandos
- Ventana de variables en memoría (Workspace)
- Navegador de directorios
- Historial de comandos
- Editor
- Ventana de figuras
- Ayuda

Operaciones básicas por línea de comandos:

```
>> 2 + 2

ans =
     4

>> 1/(1 - 2*2)

ans =
   -0.3333

>> ans

ans =
   -0.3333

>> a = 2/3*pi

a =
    2.0944
```

No es necesaria una definición del tipo de datos previa a la asignación:
Matlab asume que es el tipo de variable más general posible un arreglo de números complejos

```
>> b = sin(a)

b =
    0.8660

>> c = b^2 + cos(a)^2

c =
    1.0000
```

- ¿Qué pasa si cada sentencia se finaliza con un punto y coma (;)?
- Probar `format long, short, shortE, rat, hex`

Aritmética finita y errores numéricos

Representación en aritmética de punto flotante de doble precisión

s : bit de signo

$e = [\ e_1 \quad e_2 \quad e_3 \quad \dots \quad e_{11}\ ]$: exponente 11 dígitos binarios.
$1 \leq e \leq 2046$ con 0 y 2047 como valores especiales

$f = [\ f_1 \quad f_2 \quad f_3 \quad \dots \quad f_{52}\ ]$: mantisa 52 dígitos binarios.
$f < 2^{52}$

Los números representables tienen la forma:

$$(-1)^s 2^{(e-1023)}(1+f)$$

Consecuencias

- Vacíos entre números representables. ¿Distancia entre números? `eps`
- Números reales no representables $\Rightarrow$ errores

Algunos desastres causados por errores numéricos:

`http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html`

Aritmética finita y errores numéricos

Ejemplo 1

```
>> (.1 + .1 +.1) - 0.3

>> (.1 + .1 +.1) == 0.3

>> eps(.1)
```

Ejemplo 2

```
>> a = 2^100
>> c =  2^47
>> b = a + c
>> format long, a, b
>> a == b
>> eps(a)
```

¿Dónde prestar atención?

- Comparación de números en punto flotante (cuidado con el ==)
- Las operaciones ya no son asociativas
- Resta de números muy parecidos
- Condiciones de corte e índice de control en ciclos
- Orden de las operaciones:

```
>> format long e
>> eps/2 + 1 - eps/2
>> eps/2 - eps/2 + 1
```

Tipos de datos

- Punto flotante, simple y doble precisión.
- Enteros
- Números complejos
- Arreglos multidimensionales
- Cadenas de texto: `string`
- Estructuras y `cells`: arreglos de arreglos

Definición y asignación de variables

- No es necesaria una definición previa a la asignación
- Todas las variables se interpretan como arreglos (matrices y/o vectores) de complejos
- Los elementos de una fila se separan por espacio o coma (,)
- Los elementos de una columna se separan por punto y coma (;)
- Arreglos de arreglos

```
>> b = [1 2 3 4 5]
```

```
>> c = [1;2;3]
```

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

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

```
>> e = [c d]
```

Definición y asignación de variables

- No es necesaria una definición previa a la asignación
- Todas las variables se interpretan como arreglos (matrices y/o vectores) de complejos
- Los elementos de una fila se separan por espacio o coma (,)
- Los elementos de una columna se separan por punto y coma (;)
- Arreglos de arreglos

```
>> b = [1 2 3 4 5]

b =
    1     2     3     4     5

>> c = [1;2;3]

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

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

>> e = [c d]
```

Definición y asignación de variables

- No es necesaria una definición previa a la asignación
- Todas las variables se interpretan como arreglos (matrices y/o vectores) de complejos
- Los elementos de una fila se separan por espacio o coma (,)
- Los elementos de una columna se separan por punto y coma (;)
- Arreglos de arreglos

```
>> b = [1 2 3 4 5]

b =
    1     2     3     4     5

>> c = [1;2;3]

c =
    1
    2
    3

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

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

>> e = [c d]
```

Definición y asignación de variables

- No es necesaria una definición previa a la asignación
- Todas las variables se interpretan como arreglos (matrices y/o vectores) de complejos
- Los elementos de una fila se separan por espacio o coma (,)
- Los elementos de una columna se separan por punto y coma (;)
- Arreglos de arreglos

```
>> b = [1 2 3 4 5]

b =
    1     2     3     4     5

>> c = [1;2;3]

c =
    1
    2
    3

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

d =
    1     2     3
    4     5     6
    7     8     9
```

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

>> e = [c d]
```

Definición y asignación de variables

- No es necesaria una definición previa a la asignación
- Todas las variables se interpretan como arreglos (matrices y/o vectores) de complejos
- Los elementos de una fila se separan por espacio o coma (,)
- Los elementos de una columna se separan por punto y coma (;)
- Arreglos de arreglos

```
>> b = [1 2 3 4 5]

b =
    1     2     3     4     5

>> c = [1;2;3]

c =
    1
    2
    3

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

d =
    1     2     3
    4     5     6
    7     8     9
```

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

d =

    1     2     3
    4     5     6
    7     8     9

>> e = [c d]
```

Definición y asignación de variables

- No es necesaria una definición previa a la asignación
- Todas las variables se interpretan como arreglos (matrices y/o vectores) de complejos
- Los elementos de una fila se separan por espacio o coma (,)
- Los elementos de una columna se separan por punto y coma (;)
- Arreglos de arreglos

```
>> b = [1 2 3 4 5]

b =
    1     2     3     4     5

>> c = [1;2;3]

c =
    1
    2
    3

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

d =
    1     2     3
    4     5     6
    7     8     9
```

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

d =

    1     2     3
    4     5     6
    7     8     9

>> e = [c d]

e =
     1     1     2     3
     2     4     5     6
     3     7     8     9
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)
```

```
>> a(4)
```

```
>> a([1 3],2)
```

```
>> a(end,end)
```

Operador :

Definición de una secuencia

```
>> a = 1:5
```

```
>> b = 0:0.5:1.5
```

```
>> c = 5:-1:1
```

```
>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)


>> a([1 3],2)


>> a(end,end)
```

Operador :

Definición de una secuencia

```
>> a = 1:5


>> b = 0:0.5:1.5


>> c = 5:-1:1


>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)



>> a(end,end)
```

Operador :

Definición de una secuencia

```
>> a = 1:5

>> b = 0:0.5:1.5

>> c = 5:-1:1

>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)

ans =
     2
     8

>> a(end,end)
```

Operador :

Definición de una secuencia

```
>> a = 1:5

>> b = 0:0.5:1.5

>> c = 5:-1:1

>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)

ans =
     2
     8

>> a(end,end)

ans =
     9
```

Operador :

Definición de una secuencia

```
>> a = 1:5
```

```
>> b = 0:0.5:1.5
```

```
>> c = 5:-1:1
```

```
>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)

ans =
     2
     8

>> a(end,end)

ans =
     9
```

Operador :

Definición de una secuencia

```
>> a = 1:5

a =
     1     2     3     4     5

>> b = 0:0.5:1.5
```

```
>> c = 5:-1:1
```

```
>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)

ans =
     2
     8

>> a(end,end)

ans =
     9
```

Operador :

Definición de una secuencia

```
>> a = 1:5

a =
     1     2     3     4     5

>> b = 0:0.5:1.5

b =
         0    0.5000    1.0000    1.5000

>> c = 5:-1:1



>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)

ans =
     2
     8

>> a(end,end)

ans =
     9
```

Operador :

Definición de una secuencia

```
>> a = 1:5

a =
     1     2     3     4     5

>> b = 0:0.5:1.5

b =
         0    0.5000    1.0000    1.5000

>> c = 5:-1:1

c =
     5     4     3     2     1

>> d = 0:0.2:0.7
```

Definición y asignación de variables (2)

Indexación

```
>> a(2,3)

ans =
     6

>> a(4)

ans =
     2

>> a([1 3],2)

ans =
     2
     8

>> a(end,end)

ans =
     9
```

Operador :

Definición de una secuencia

```
>> a = 1:5

a =
     1     2     3     4     5

>> b = 0:0.5:1.5

b =
         0    0.5000    1.0000    1.5000

>> c = 5:-1:1

c =
     5     4     3     2     1

>> d = 0:0.2:0.7

d =
         0    0.2000    0.4000    0.6000
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

>> a(1:2,2:3)

>> a(2,:)

>> a(:)

>> a(2,1:2:3)

>> a(2,2:3)
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

ans =
     1
     4
     7

>> a(2,:)

>> a(2,1:2:3)

>> a(2,2:3)
```

```
>> a(1:2,2:3)

>> a(:)
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

ans =
     1
     4
     7

>> a(2,:)

ans =
     4     5     6

>> a(2,1:2:3)



>> a(2,2:3)
```

```
>> a(1:2,2:3)
```

```
>> a(:)
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

ans =
     1
     4
     7

>> a(2,:)

ans =
     4     5     6

>> a(2,1:2:3)

ans =
     4     6

>> a(2,2:3)
```

```
>> a(1:2,2:3)
```

```
>> a(:)
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

ans =
     1
     4
     7

>> a(2,:)

ans =
     4     5     6

>> a(2,1:2:3)

ans =
     4     6

>> a(2,2:3)

ans =
     5     6
```

```
>> a(1:2,2:3)
```

```
>> a(:)
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

ans =
     1
     4
     7

>> a(2,:)

ans =
     4     5     6

>> a(2,1:2:3)

ans =
     4     6

>> a(2,2:3)

ans =
     5     6

>> a(1:2,2:3)

ans =
     2     3
     5     6

>> a(:)
```

Definición y asignación de variables (3)

Operador :

Indexación de elementos en las matrices

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

>> a(:,1)

ans =
     1
     4
     7

>> a(2,:)

ans =
     4     5     6

>> a(2,1:2:3)

ans =
     4     6

>> a(2,2:3)

ans =
     5     6
```

```
>> a(1:2,2:3)

ans =
     2     3
     5     6

>> a(:)

ans =
     1
     4
     7
     2
     5
     8
     3
     6
     9
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]
```

```
>> a(:)
```

```
>> reshape(a,3,2)
```

```
>> a'
```

```
>> a([1 2 1 2 2],:)
```

```
>> fliplr(a), flipud(a)
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]

a =
     1     2     3
     4     5     6

>> a(:)
```

```
>> reshape(a,3,2)
```

```
>> a'
```

```
>> a([1 2 1 2 2],:)
```

```
>> fliplr(a), flipud(a)
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]

a =
     1     2     3
     4     5     6

>> a(:)

ans =
     1
     4
     2
     5
     3
     6

>> reshape(a,3,2)
```

```
>> a'
```

```
>> a([1 2 1 2 2],:)
```

```
>> fliplr(a), flipud(a)
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]

a =
     1     2     3
     4     5     6

>> a(:)

ans =
     1
     4
     2
     5
     3
     6

>> reshape(a,3,2)

ans =
     1     5
     4     3
     2     6
```

```
>> a'
```

```
>> a([1 2 1 2 2],:)
```

```
>> fliplr(a), flipud(a)
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]

a =
     1     2     3
     4     5     6

>> a(:)

ans =
     1
     4
     2
     5
     3
     6

>> reshape(a,3,2)

ans =
     1     5
     4     3
     2     6
```

```
>> a'

ans =
     1     4
     2     5
     3     6

>> a([1 2 1 2 2],:)
```

```
>> fliplr(a), flipud(a)
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]

a =
     1     2     3
     4     5     6

>> a(:)

ans =
     1
     4
     2
     5
     3
     6

>> reshape(a,3,2)

ans =
     1     5
     4     3
     2     6
```

```
>> a'

ans =
     1     4
     2     5
     3     6

>> a([1 2 1 2 2],:)

ans =
     1     2     3
     4     5     6
     1     2     3
     4     5     6
     4     5     6

>> fliplr(a), flipud(a)
```

Modificación de las dimensiones de las matrices

```
>> a = [1 2 3;4 5 6]

a =
     1     2     3
     4     5     6

>> a(:)

ans =
     1
     4
     2
     5
     3
     6

>> reshape(a,3,2)

ans =
     1     5
     4     3
     2     6
```

```
>> a'

ans =
     1     4
     2     5
     3     6

>> a([1 2 1 2 2],:)

ans =
     1     2     3
     4     5     6
     1     2     3
     4     5     6
     4     5     6

>> fliplr(a), flipud(a)

ans =
     3     2     1
     6     5     4
ans =
     4     5     6
     1     2     3
```

Arreglos especiales

`eye`: matriz identidad

```
>> I1 = eye(3)
>> I2 = eye(3,5)
```

`zeros`: matriz de ceros

```
>> Ceros = zeros(4)
```

`ones`: matriz de unos

```
>> Unos = ones(2)
```

`rand`: matriz de números aleatorios

```
>> A = rand(2)
>> V = rand(1,4)
```

`diag`: diagonal de una matriz y matrices diagonales y banda

```
>> a = [1 2 3;4 5 6;7 8 9]
>> diag(a)

>> diag(1:5)
```

`linspace`: vector equiespaciado

```
>> c = linspace(0,1,12)
```

`linspace`: vector equiespaciado logarítmicamente

```
>> c = logspace(0,1,12)
```

Operadores

Operadores aritméticos

+	Suma
-	Resta
.*	Multiplicación
./	División
.^	Potencia
*	Mult. matrices
/	Div. matrices
^	Potencia matrices

Operadores relacionales

<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
==	Igual
~=	Distinto

Operadores lógicos

&	y
—	o
~	no
any	alguno
all	todos

Funciones

Funciones sobre escalares

- `abs`
- `cos, cosd`
- `sin, sind`
- `exp`
- `log`
- `log10`
- `tan, tand`
- `sqrt`
- `sign`
- `floor`
- `round`
- `ceil`

Funciones sobre vectores

`max`: elemento máximo de un vector

`min`: elemento mínimo de un vector

`sort`: ordena un vector en forma ascendente o descendente

`sum`: suma los elementos de un vector

`prod`: producto de los elementos de un vector

`mean`: promedio de los elementos de un vector

Tamaño de los arreglos

`length`: tamaño de la mayor dimensión

`ndims`: número de dimensiones

`numel`: número de elementos

`size`: tamaño de cada dimensión

Funciones sobre matrices y herramientas de álgebra lineal

`sum`: suma los elementos a lo largo de una de las dimensiones

`triu`: parte triangular superior de una matriz

`tril`: parte triangular inferior de una matriz

`det`: determinante

`eig`: autovalores y autovectores

`svd`: descomposición en valores singulares

`inv`: inversa de una matriz

`lu`: factorización LU

`chol`: factorización de Cholesky

`qr`: factorización QR

`cond`: número de condición en la norma 2

`norm`: norma 1, norma 2, norma de Frobenius, norma ∞

`poly`: polinomio característico

`rank`: rango

Resolución de sistemas de ecuaciones

Si $\mathbf{A}$ es una matriz inversible y $\mathbf{b}$ es un vector columna de dimensión adecuada, entonces:

```
x = A\b = inv(A) * b
```

es la solución de $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ y

```
x = b/A = b * inv(A)
```

es la solución de $\mathbf{x}\,\mathbf{A} = \mathbf{b}$

Arreglos multidimensionales

- Matlab extiende la sintáxis naturalmente a más de 2 dimensiones
- Las funciones y operaciones elemento a elemento son válidas
- `eye`, `zeros`, `ones`, `rand` y otras con n argumentos permiten crear arreglos de n dimensiones donde cada argumento indica el cardinal de la dimensión correspondiente

  $>>$ `I2 = eye(3,5,2,4)`
- El operador (**:**) funciona análogamente
- Para concatenar arreglos de dimensiones mayores a 2 se puede usar la función `cat` que funciona análogamente a la coma (**,**) y al punto y coma (**;**) utilizados en 2 dimensiones

```
>> A = cat(3, [1 2 3; 9 8 7; 4 6 5], [0 3 2; 8 8 4; 5 3 5],...
[6 4 7; 6 8 5; 5 4 3])

>> A(1,:,:)

>> A(:)
```

Arreglos multidimensionales

- Matlab extiende la sintáxis naturalmente a más de 2 dimensiones
- Las funciones y operaciones elemento a elemento son válidas
- `eye`, `zeros`, `ones`, `rand` y otras con n argumentos permiten crear arreglos de n dimensiones donde cada argumento indica el cardinal de la dimensión correspondiente

  $>>$ `I2 = eye(3,5,2,4)`
- El operador (**:**) funciona análogamente
- Para concatenar arreglos de dimensiones mayores a 2 se puede usar la función `cat` que funciona análogamente a la coma (**,**) y al punto y coma (**;**) utilizados en 2 dimensiones

```
>> A = cat(3, [1 2 3; 9 8 7; 4 6 5], [0 3 2; 8 8 4; 5 3 5],...
[6 4 7; 6 8 5; 5 4 3])

>> A(1,:,:)

ans(:,:,1) =
     1     2     3

ans(:,:,2) =
     0     3     2

ans(:,:,3) =
     6     4     7

>> A(:)
```

Matrices ralas (`sparse`)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

Matrices ralas (sparse)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

- Definición directa de los elementos `S = sparse(i,j,s,m,n)`

```
>> S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

Matrices ralas (sparse)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

- Definición directa de los elementos `S = sparse(i,j,s,m,n)`

```
>> S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

- Definición a partir de las diagonales de la matriz `S = spdiags(B,d,m,n)`

Matrices ralas (`sparse`)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

- Definición directa de los elementos `S = sparse(i,j,s,m,n)`

```
>> S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

- Definición a partir de las diagonales de la matriz `S = spdiags(B,d,m,n)`
- Alocamiento de espacio para matrices ralas: `S = spalloc(m,n,nzmax)`

Matrices ralas (`sparse`)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

- Definición directa de los elementos `S = sparse(i,j,s,m,n)`

```
>> S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

- Definición a partir de las diagonales de la matriz `S = spdiags(B,d,m,n)`
- Alocamiento de espacio para matrices ralas: `S = spalloc(m,n,nzmax)`
- Propiedades de matrices ralas: `nnz`, `nonzeros`, `nzmax`

Matrices ralas (`sparse`)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

- Definición directa de los elementos `S = sparse(i,j,s,m,n)`

```
>> S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

- Definición a partir de las diagonales de la matriz `S = spdiags(B,d,m,n)`
- Alocamiento de espacio para matrices ralas: `S = spalloc(m,n,nzmax)`
- Propiedades de matrices ralas: `nnz`, `nonzeros`, `nzmax`
- Matrices especiales: `speye`, `sprand`, `sprandsym`

Matrices ralas (sparse)

Soportadas nativamente en Matlab

- Conversión de `full` a `sparse`

```
>> A = [ 0   0   0   5; 0   2   0   0; 1   3   0   0; 0   0   4   0]
>> S = sparse(A)
>> B = full(S)
>> whos
```

- Definición directa de los elementos `S = sparse(i,j,s,m,n)`

```
>> S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

- Definición a partir de las diagonales de la matriz `S = spdiags(B,d,m,n)`
- Alocamiento de espacio para matrices ralas: `S = spalloc(m,n,nzmax)`
- Propiedades de matrices ralas: `nnz`, `nonzeros`, `nzmax`
- Matrices especiales: `speye`, `sprand`, `sprandsym`
- Visualización de matrices ralas: `spy(S)`

```
S = sprand(100,100,0.1)
spy(S)
```

Entrada / Salida

- `save FILENAME <variables>`
- `load FILENAME`
- `importdata(FILENAME)`

Gráficos 2D

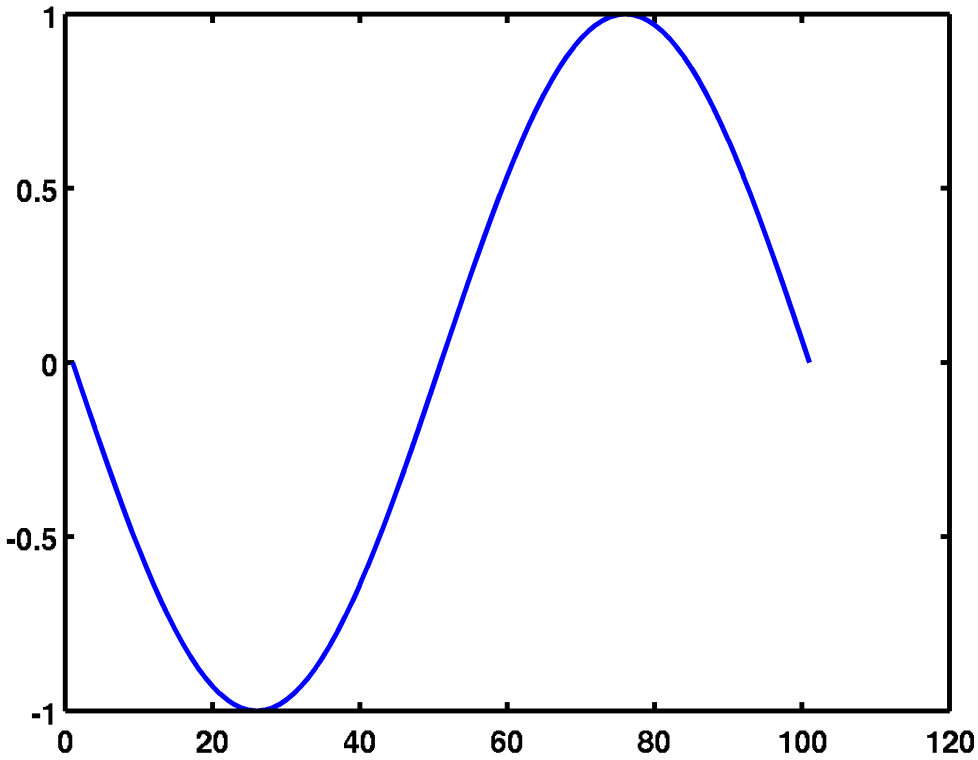
Graficar un vector

```
>> x = linspace(-pi,pi,101);
>> y = sin(x);
>> plot(y)
```

Gráficos 2D

Graficar un vector

```
>> x = linspace(-pi,pi,101);
>> y = sin(x);
>> plot(y)
```



Gráficos 2D

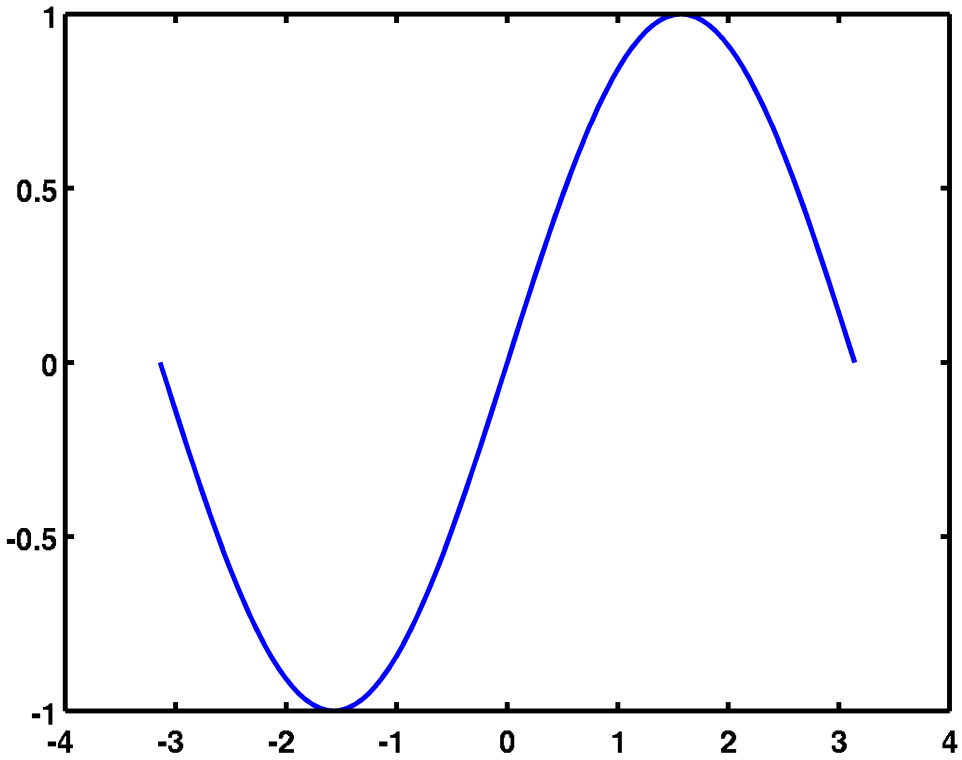
Graficar un vector

```
>> x = linspace(-pi,pi,101);
>> y = sin(x);
>> plot(x,y)
```

Gráficos 2D

Graficar un vector

```
>> x = linspace(-pi,pi,101);
>> y = sin(x);
>> plot(x,y)
```



Gráficos 2D

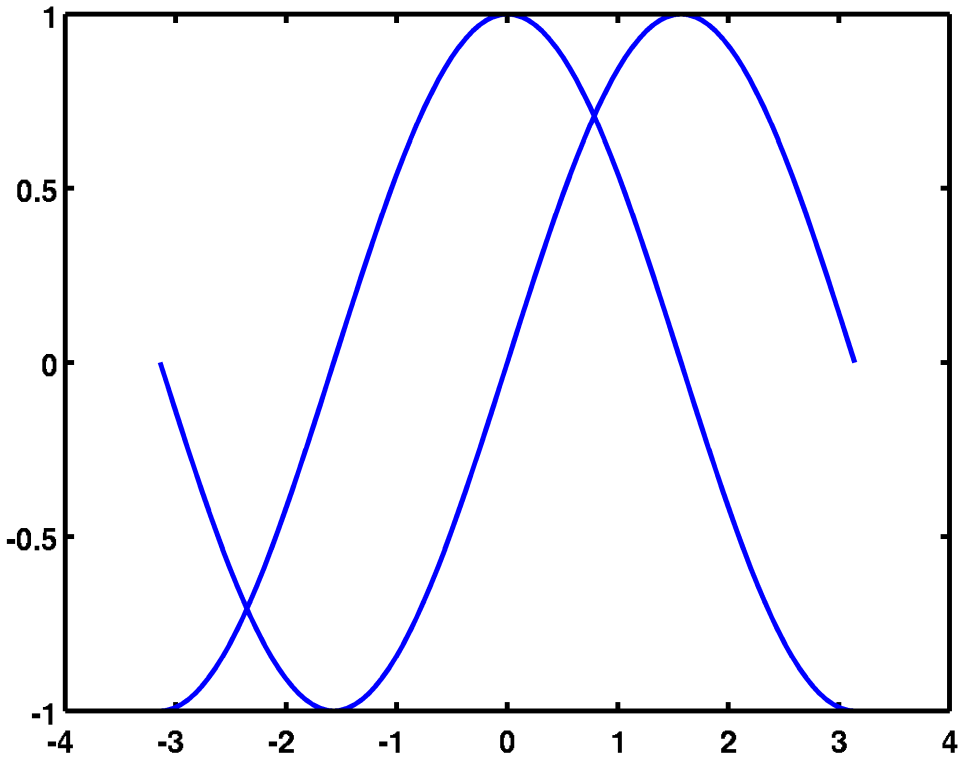
Graficar más de un vector

```
>> x = linspace(-pi,pi,101);
>> y = sin(x);
>> plot(x,y)

>> hold on
>> z = cos(x);
>> plot(x,z)
>> hold off
```

Gráficos 2D

Graficar más de un vector

```
>> x = linspace(-pi,pi,101);
>> y = sin(x);
>> plot(x,y)

>> hold on
>> z = cos(x);
>> plot(x,z)
>> hold off
```



Gráficos 2D

Graficar más de un vector:

Solución fácil

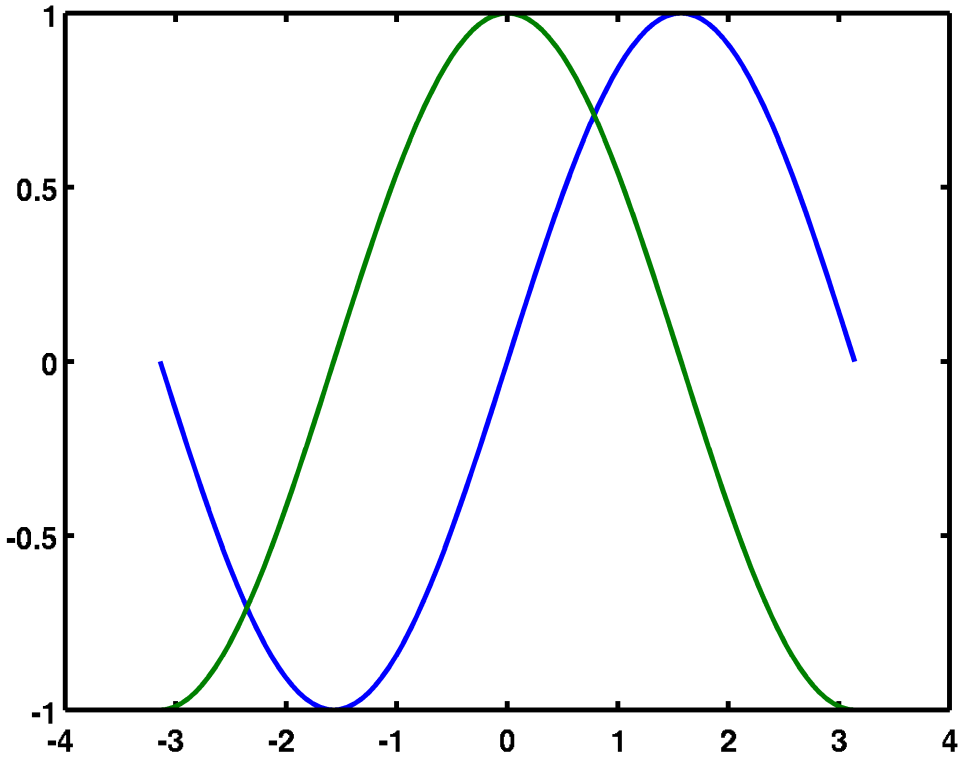
```
>> plot(x,y,x,z)
```

Gráficos 2D

Graficar más de un vector:

Solución fácil

```
>> plot(x,y,x,z)
```



Graficar más de un vector:

Solución más compleja

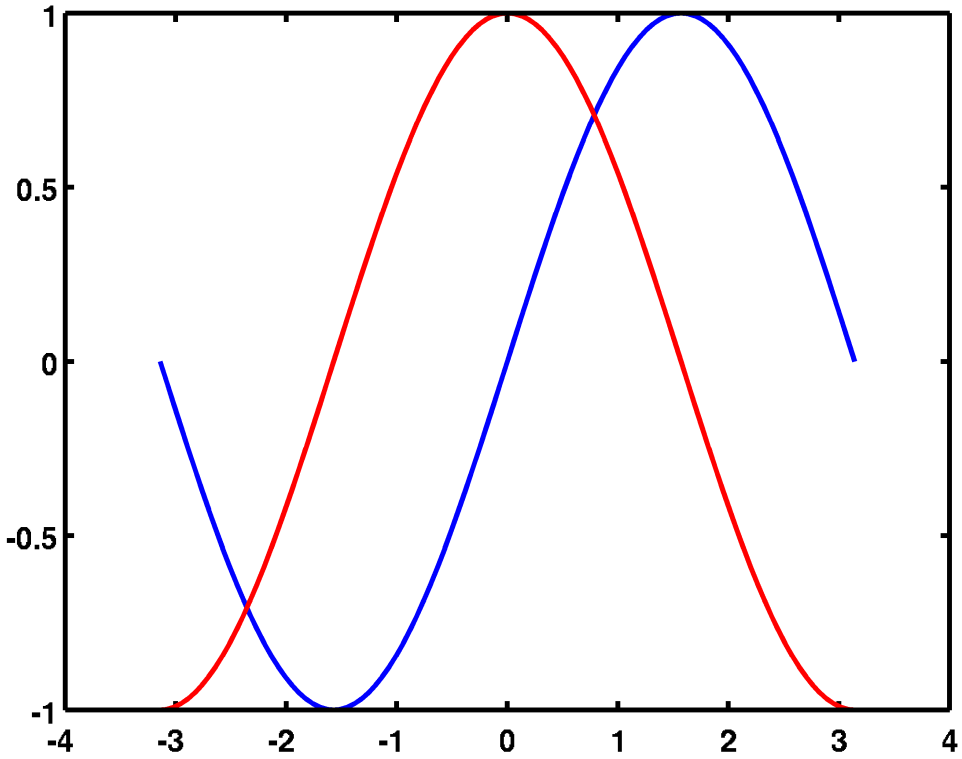
```
>> plot(x,y,'b',x,z,'r')
```

Gráficos 2D

Graficar más de un vector:

Solución más compleja

```
>> plot(x,y,'b',x,z,'r')
```

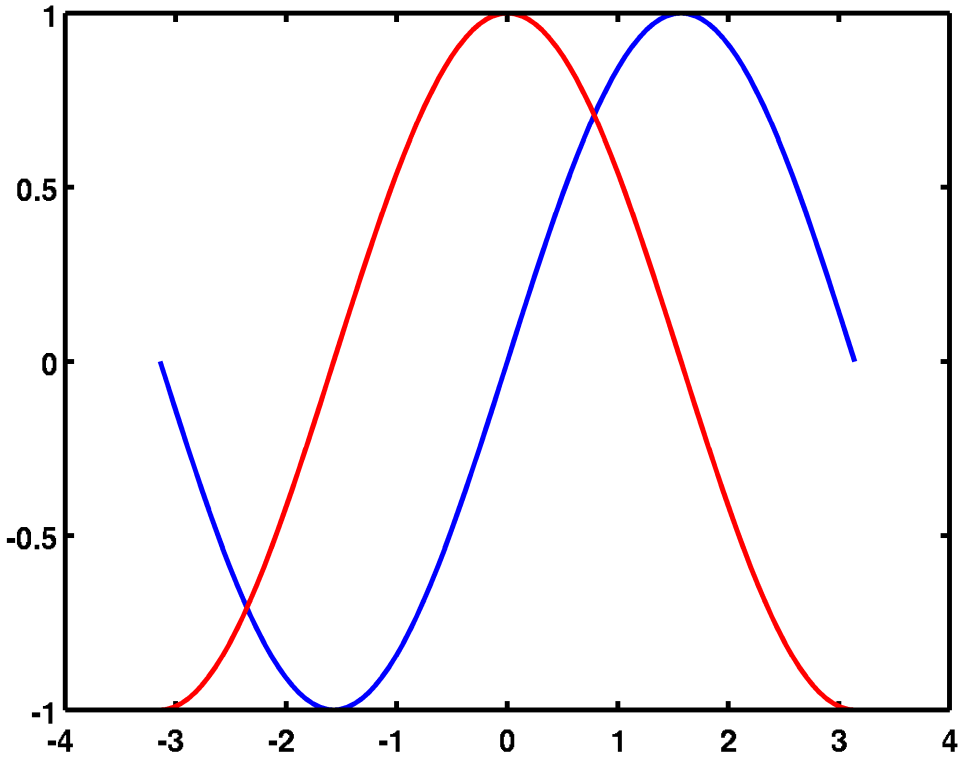


Gráficos 2D

Graficar más de un vector:

Solución más compleja

```
>> plot(x,y,'b')
>> hold on
>> plot(x,z,'r')
>> hold off
```



Gráficos 2D

Modificadores de las propiedades de línea

Tipo de línea

- `-` : Línea contínua
- `--` : Línea de trazos
- `:` : Línea de puntos
- `-.` : Línea de trazos cortos y largos

Color

- `y` : amarillo
- `m` : magenta
- `c` : cian
- `r` : rojo
- `g` : verde
- `b` : azul
- `w` : blanco
- `k` : negro

Markers

- `+` : signo +
- `o` : círculo
- `*` : asterisco
- `.` : punto
- `x` : cruz
- `s` : cuadrado
- `d` : diamante
- `^,v,<,>` : triángulos con distinta orientación

Gráficos 2D

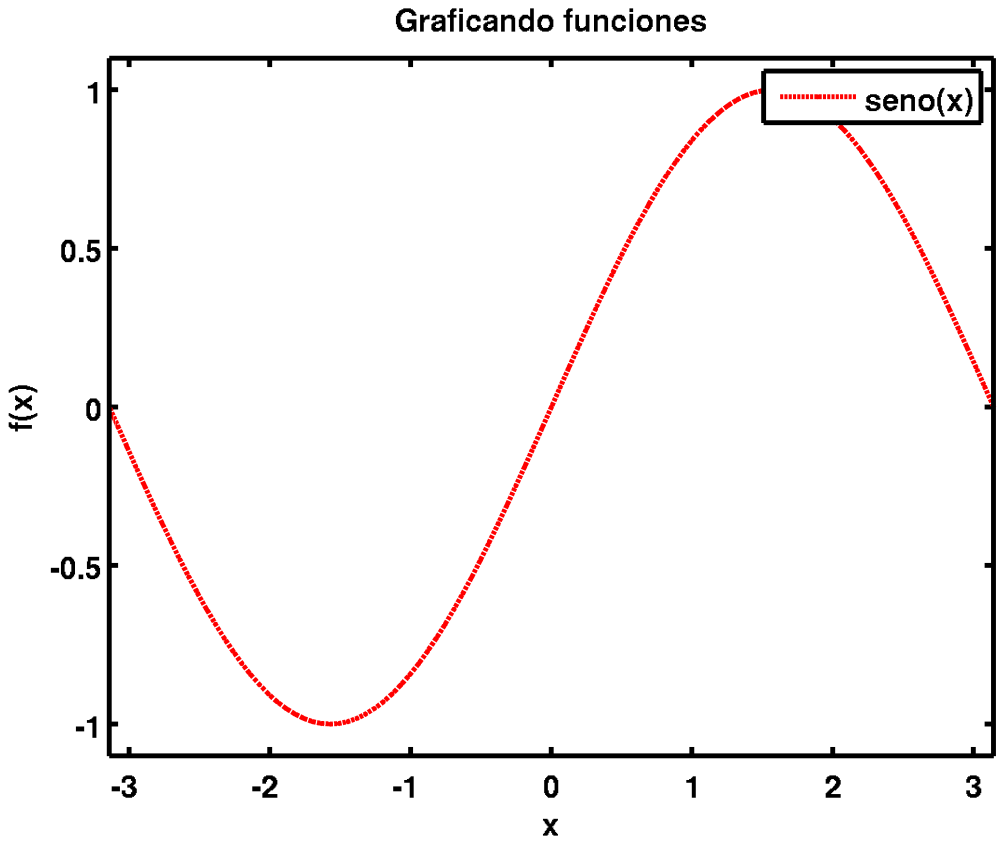
Modificadores del gráfico y la figura

```
>> plot(x,y,'DisplayName','seno(x)')
>> legend('show')
>> xlabel('x')
>> ylabel('f(x)')
>> title('Graficando funciones')
>> xlim([-pi pi])
>> ylim([-1.1 1.1])
```

Gráficos 2D

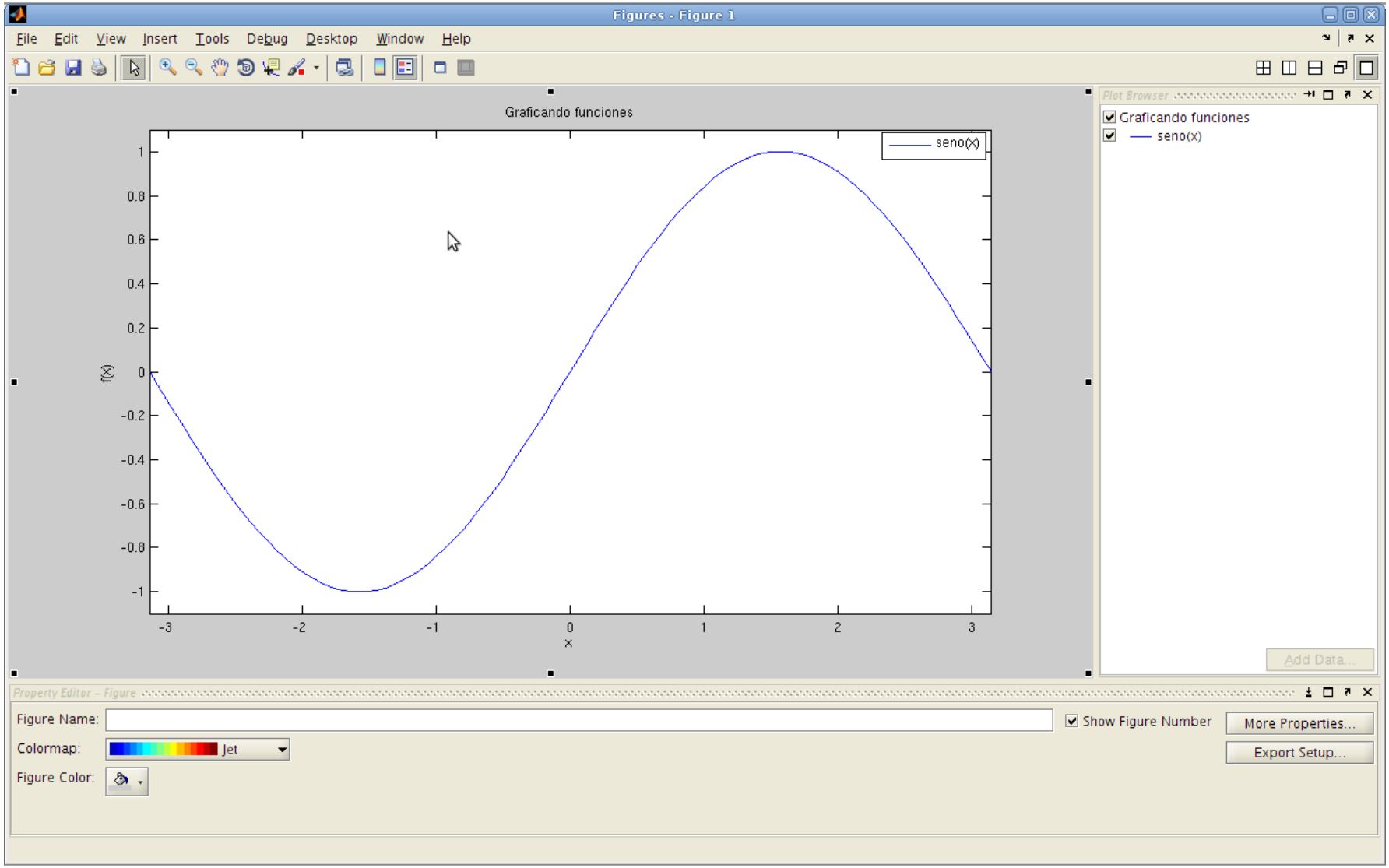
Modificadores del gráfico y la figura

```
>> plot(x,y,'DisplayName','seno(x)
>> legend('show')
>> xlabel('x')
>> ylabel('f(x)')
>> title('Graficando funciones')
>> xlim([-pi pi])
>> ylim([-1.1 1.1])
```



Gráficos 2D

Modificadores del gráfico y la figura: Ventana interactiva



plotyy : Dos series de datos con diferentes ejes `y` y mismo eje `x`

semilogx : Logarítmico en el eje `x`

semilogy : Logarítmico en el eje `y`

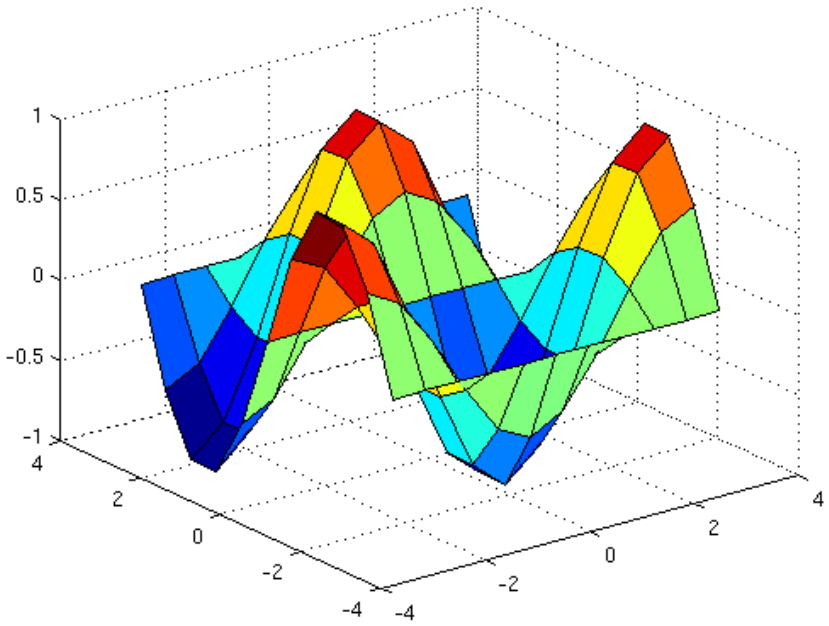
loglog : Logarítmico en ambos ejes

Gráficos 3D

```
>> x = linspace(-pi,pi,11)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
```

Gráficos 3D

```
>> x = linspace(-pi,pi,11)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
```

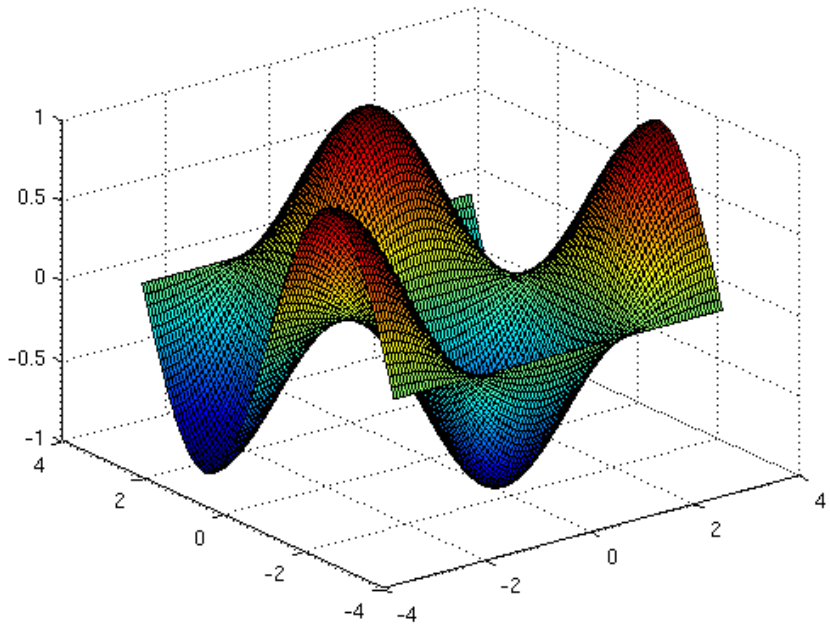


Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
```

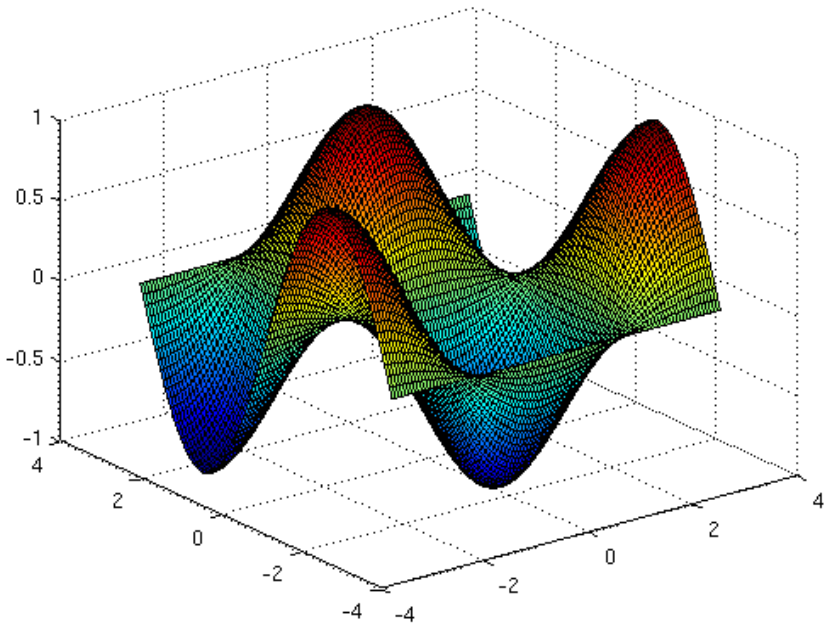
Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
```



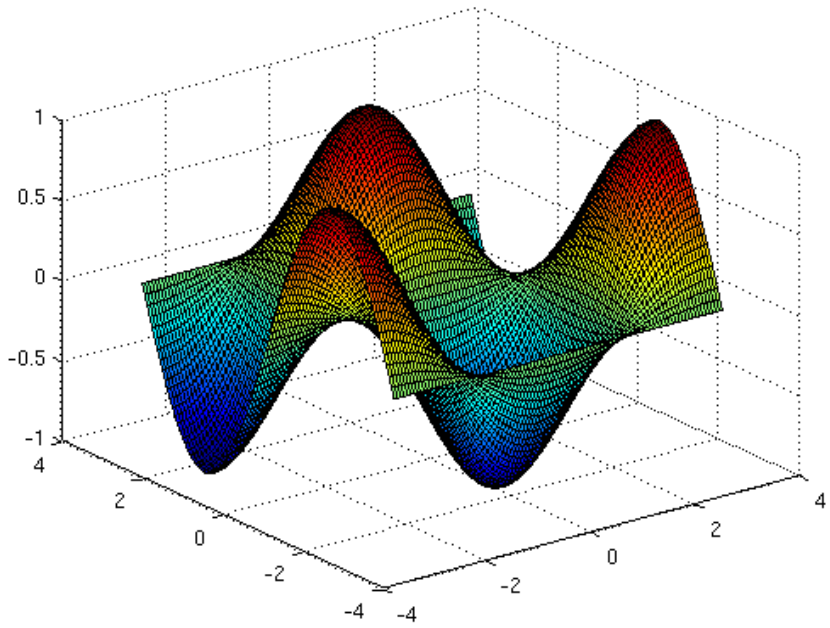
Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
>> shading interp
```



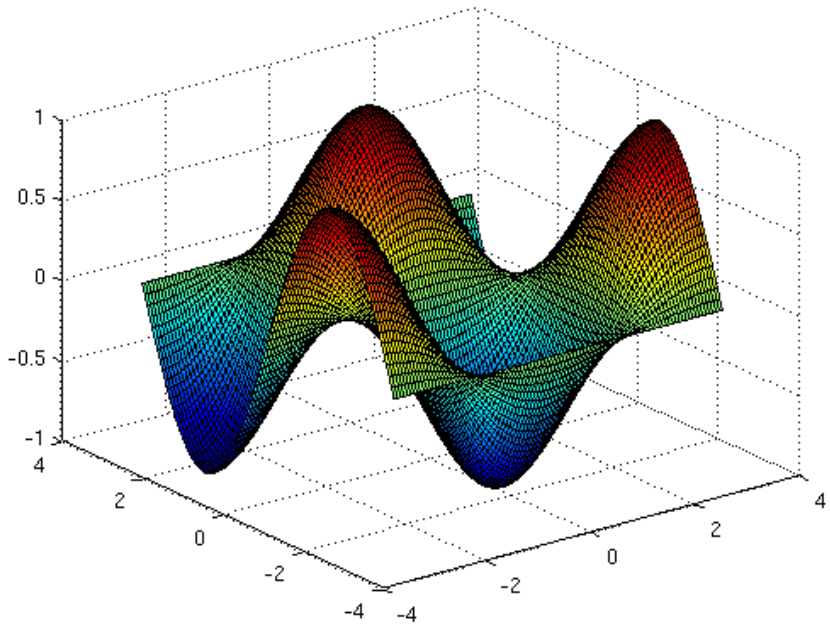
Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
>> shading interp
>> view(2)
```



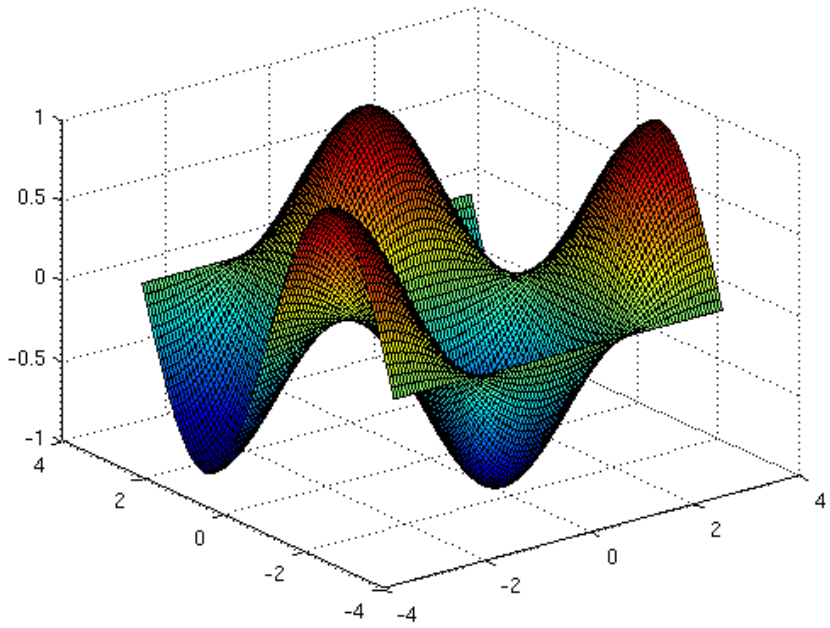
Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
>> shading interp
>> view(2)
>> colorbar
```



Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> whos
>> surf(X,Y,Z)
>> shading interp
>> view(2)
>> colorbar
>> caxis([0 1])
>> colorbar
```

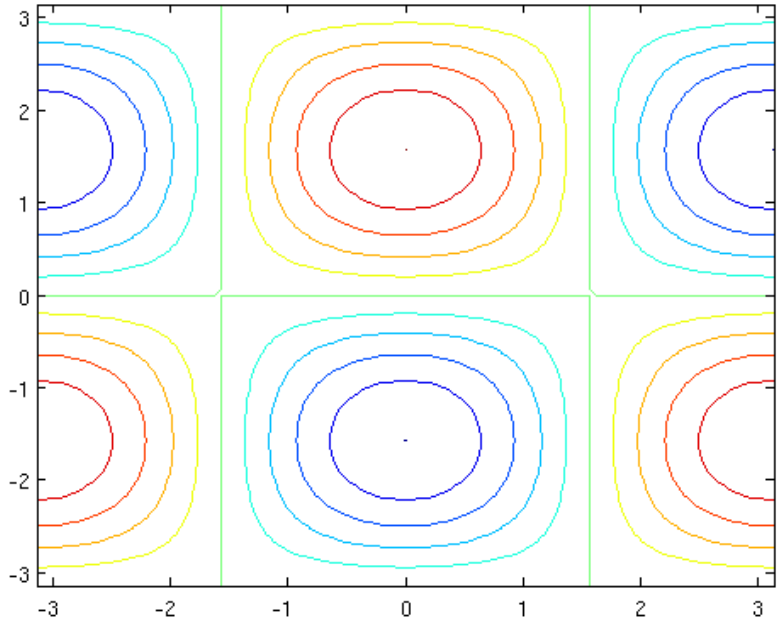


Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> contour(X,Y,Z)
```

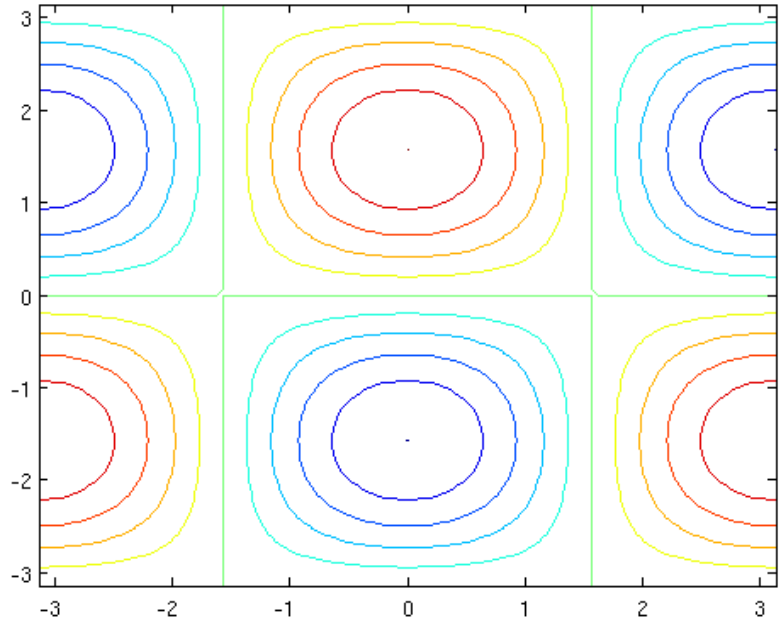
Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> contour(X,Y,Z)
```



Gráficos 3D

```
>> x = linspace(-pi,pi,101)
>> y = x;
>> [X,Y] = meshgrid(x,y)
>> Z = cos(X).*sin(Y)
>> contour(X,Y,Z)
```



Ver la ayuda de Matlab

Scripts

```
% Script graficar_seno
%   Toma el vector 'x' calcula el seno y lo grafica

Seno = sin(x);

plot(x,Seno,'-r*','DisplayName','sin(x)')
xlabel('x')
ylabel('f(x)')
title('Script graficar seno')
xlim([min(x) max(x)])
ylim([min(Seno) max(Seno)])
axis image
```

Ejecutar

```
>> x = linspace(-pi,pi,21);
>> graficar_seno

>> x = linspace(-pi,pi,101);
>> graficar_seno

>> help graficar_seno
```

Scripts

Ejercicio 1

Escribir un `script` que dados 2 vectores de coordenadas `x` e `y` construya una grilla y evalue la función

$$f = xy^3 - yx^3$$

en cada punto de la misma, mostrando un gráfico 3D con las anotaciones necesarias y devolviendo en variables `sumrow` y `sumcol` las sumas de los elementos de cada fila y cada columna respectivamente

Funciones

- Cada función tiene su propio espacio de trabajo donde ”viven´´ sus variables
- Los nombres de las variables son propios del espacio de trabajo de la función
- El espacio de trabajo desaparece cuando la función termina
- El nombre de la función debe ser idéntico al del archivo que la contiene

```
function [ output_args ] = function_name( input_args )
%UNTITLED2 Summary of this function goes here
%Detailed explanation goes here
:
end
```

Ejercicio 2

Escribir una función que haga lo mismo que el `script` de Ejercicio 1 pero recibiendo `x` e `y` como variables de entrada y devolviendo `sumrow` y `sumcol` en la salida

Argumentos de Entrada / Salida

- `nargin`
- `nargout`
- `varargin`
- `varargout`

Sentencia `if`

```
if expression
   statements
elseif expression
   statements
else
   statements
end
```

Ejercicio 3

Escribir una función reciba 2 números `a` y `b` y los devuelva ordenados de menor a mayor en un vector

Sentencia `switch`

```
switch switch_expression
   case case_expression
      statements
   case case_expression
      statements
    :
   otherwise
      statements
end
```

Sentencia `for`

```
for index = values
   program statements
           :
end
```

Ejercicio 4

Escribir una función que realice lo mismo que la del Ejercicio 2 pero con la función

$$f = \begin{cases} xy^3 - yx^3 & xy > 0 \\ 0 & xy <= 0 \end{cases}$$

Sentencia `while`

```
while expression
   statements
end
```

Sentencias `break`, `return` y `continue`

`break` : interrumpe la ejecución de un ciclo `for` o `while`, saliendo fuera del mismo

`return` : interrumpe la ejecución de la función actual y pasa el control a la instancia que la invoca

`continue` : interrumpe temporariamente la ejecución del código dentro de un ciclo y avanza a la iteración siguiente

Ejercicio 5

Escribir una función que haga lo mismo que la del ejercicio 4 pero utilizando comandos `while` en vez de `for`

Funciones `inline`

El comando inline permite crear un objeto función en el espacio de trabajo de Matlab. La expresión de la función se escribe como `string`.

Ejemplos:

1. `func = inline('3*sin(2*x.^2)')`

2. `otrafunc = inline('sin(x)*cos(x)','x')`

3. `func2arg = inline('sin(alpha*x)','x','alpha')`

`Function handles` o la función por el mango (?!)

Matlab permite pasar funciones (nativas o programadas por uno mismo) como parámetros de otras funciones. Esto reemplaza a las funciones `inline` y es la forma recomendada.

Dada una función determinada, creamos un `handle` a la misma:

1. ```
   handle_sin = @sin
   >> handle_sin(1)
   ```
2. ```
   handle_mifun = @mifun
   ```

También se puede definir un `handle` de una función que no existe, llamadas *funciones anónimas* de la siguiente manera:

1. `sqr = @(x) x.^2`
2. Una función de 2 variables: `fh = @(x,y)(y*sin(x)+x*cos(y))`
3. Una función vectorial: `fh2 = @(x)[2*sin(x+3),cos(x)]`

Para qué sirven los `handles` de funciones?

Para pasar la función como argumento a otras funciones, como por ejemplo:

1. Para graficar una función: `fplot(@humps,[0,2]);`

Para qué sirven los `handles` de funciones?

Para pasar la función como argumento a otras funciones, como por ejemplo:

1. Para graficar una función: `fplot(@humps,[0,2]);`
2. Para encontrar el cero de una función (hay que pasarle un valor para iniciar la busqueda)

```
z = fzero(@humps,1);
fplot(@humps,[0,2]);
hold on; plot(z,0,'r*'); hold off
```

Para qué sirven los `handles` de funciones?

Para pasar la función como argumento a otras funciones, como por ejemplo:

1. Para graficar una función: `fplot(@humps,[0,2]);`
2. Para encontrar el cero de una función (hay que pasarle un valor para iniciar la busqueda)

```
z = fzero(@humps,1);
fplot(@humps,[0,2]);
hold on; plot(z,0,'r*'); hold off
```

3. Para encontrar el mínimo de una función (hay que pasarle un intervalo)

```
m = fminbnd(@humps,0.25,1);
fplot(@humps,[0 2]);
hold on; plot(m,humps(m),'r*'); hold off
```

Para qué sirven los `handles` de funciones?

Para pasar la función como argumento a otras funciones, como por ejemplo:

1. Para graficar una función: `fplot(@humps,[0,2]);`
2. Para encontrar el cero de una función (hay que pasarle un valor para iniciar la busqueda)

```
z = fzero(@humps,1);
fplot(@humps,[0,2]);
hold on; plot(z,0,'r*'); hold off
```

3. Para encontrar el mínimo de una función (hay que pasarle un intervalo)

```
m = fminbnd(@humps,0.25,1);
fplot(@humps,[0 2]);
hold on; plot(m,humps(m),'r*'); hold off
```

4. Para integrar una función en un intervalo

```
q = quad(@humps,0.5,1);
fplot(@humps,[0,2]);
title(['Area = ',num2str(q)]);
```

Cómo utilizar funciones pasadas como parámetro?

```
[y1, y2, ...] = feval(fhandle, x1, ..., xn)
```

Ejercicio 6

Escribir una función que reciba como parámetro una función de $\mathbb{R} \rightarrow \mathbb{R}$ vectorizada y un vector con un muestreo cualquiera sobre los reales y calcule el máximo, mínimo y la media de dicha función en los puntos de muestreo.