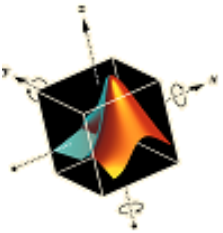
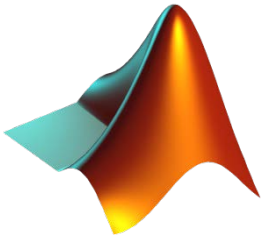
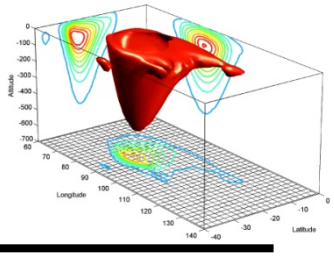




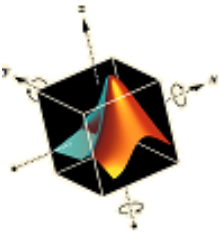
Introducción a MATLAB

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

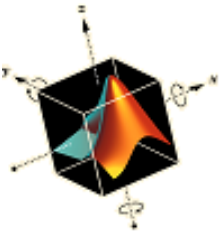
Universidad de Cantabria

corcuerp@unican.es



Objetivos

- Capacidad de generar programas MATLAB legibles, compactos y correctamente verificables que obtengan soluciones numéricas a un amplio rango de modelos y mostrar los resultados con gráficos autodescriptivos



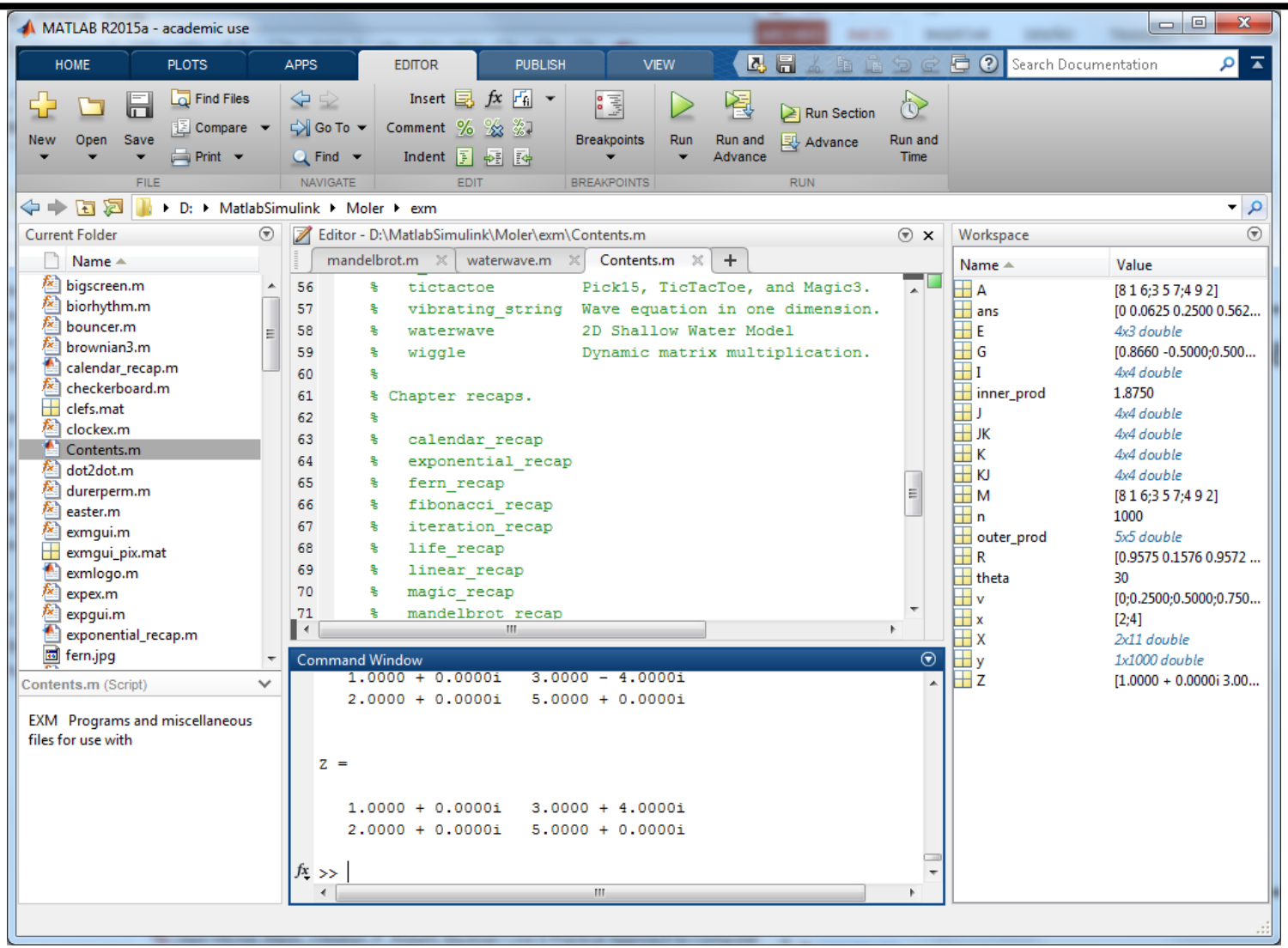
Indice

- Creación de clases propias
- Introducción a funciones
- Creación de funciones

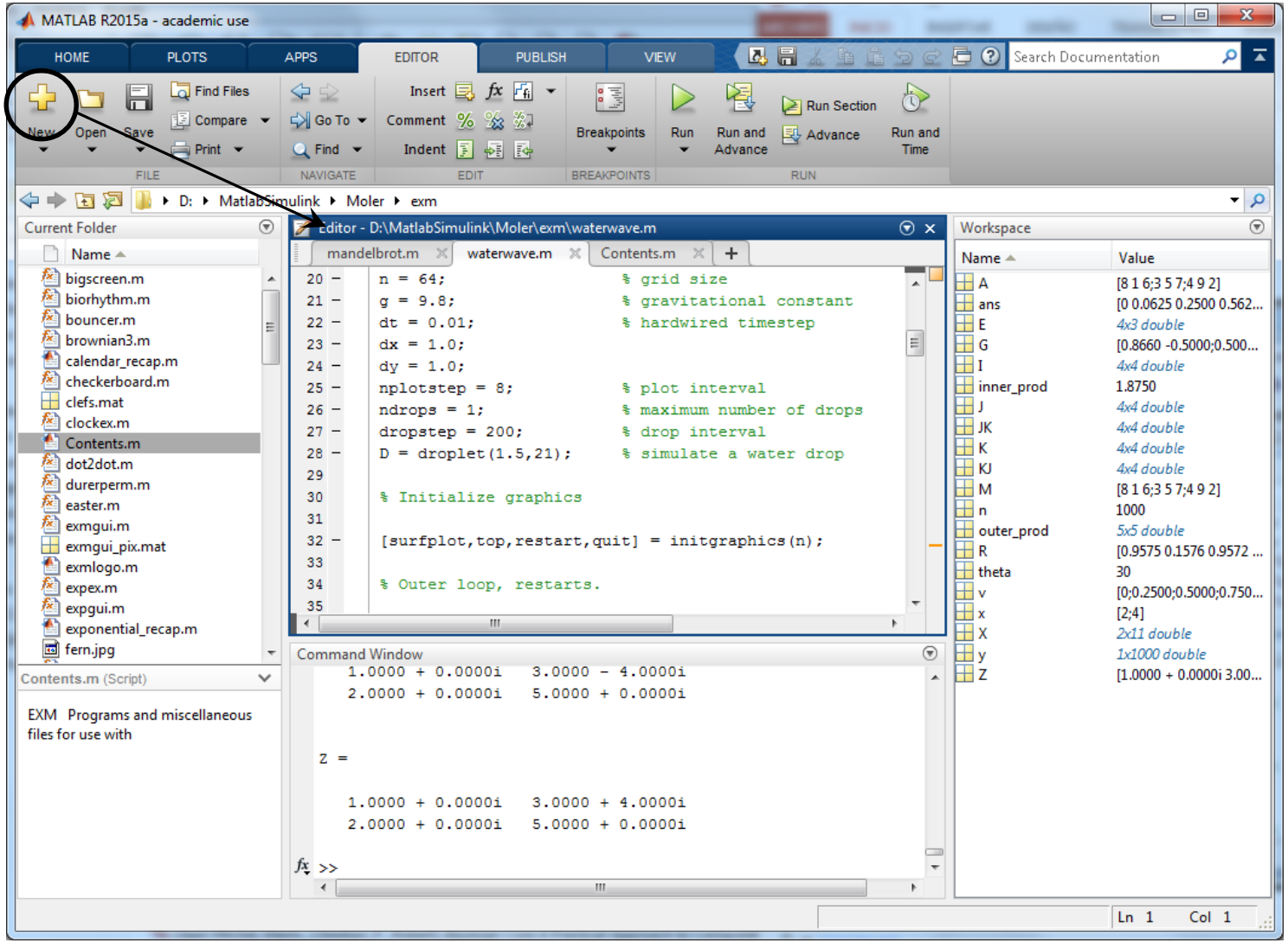
Historia

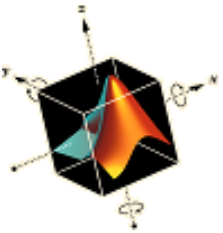
- MATLAB fue fundado en 1984 por Jack Little, Steve Bangert, and Cleve Moler
- El nombre deriva de *MAT*rix *LAB*oratory
- El equivalente (con limitaciones) en software libre es: Octave
- Dispone de un amplio conjunto de toolboxes

Entorno


MATLAB R2015a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
D: ▸ MatlabSimulink ▸ Moler ▸ exm
Current Folder
Name
bigscreen.m
biorhythm.m
bouncer.m
brownian3.m
calendar_recap.m
checkerboard.m
clefs.mat
clockex.m
Contents.m
dot2dot.m
durerperm.m
easter.m
exmgui.m
exmgui_pix.mat
exmlogo.m
expex.m
expgui.m
exponential_recap.m
fern.jpg
Contents.m (Script)
EXM Programs and miscellaneous files for use with
Editor - D:\MatlabSimulink\Moler\exm\Contents.m
mandelbrot.m
waterwave.m
Contents.m
% tictactoe Pick15, TicTacToe, and Magic3.
% vibrating_string Wave equation in one dimension.
% waterwave 2D Shallow Water Model
% wiggle Dynamic matrix multiplication.
%
% Chapter recaps.
%
% calendar_recap
% exponential_recap
% fern_recap
% fibonacci_recap
% iteration_recap
% life_recap
% linear_recap
% magic_recap
% mandelbrot_recap
Command Window
1.0000 + 0.0000i 3.0000 - 4.0000i
2.0000 + 0.0000i 5.0000 + 0.0000i
Z =
1.0000 + 0.0000i 3.0000 + 4.0000i
2.0000 + 0.0000i 5.0000 + 0.0000i
>>
Workspace
Name
Value
A [8 1 6;3 5 7;4 9 2]
ans [0 0.0625 0.2500 0.562...
E 4x3 double
G [0.8660 -0.5000;0.500...
I 4x4 double
inner_prod 1.8750
J 4x4 double
JK 4x4 double
K 4x4 double
KJ 4x4 double
M [8 1 6;3 5 7;4 9 2]
n 1000
outer_prod 5x5 double
R [0.9575 0.1576 0.9572 ...
theta 30
v [0;0.2500;0.5000;0.750...
x [2;4]
X 2x11 double
y 1x1000 double
Z [1.0000 + 0.0000i 3.00...

Editor

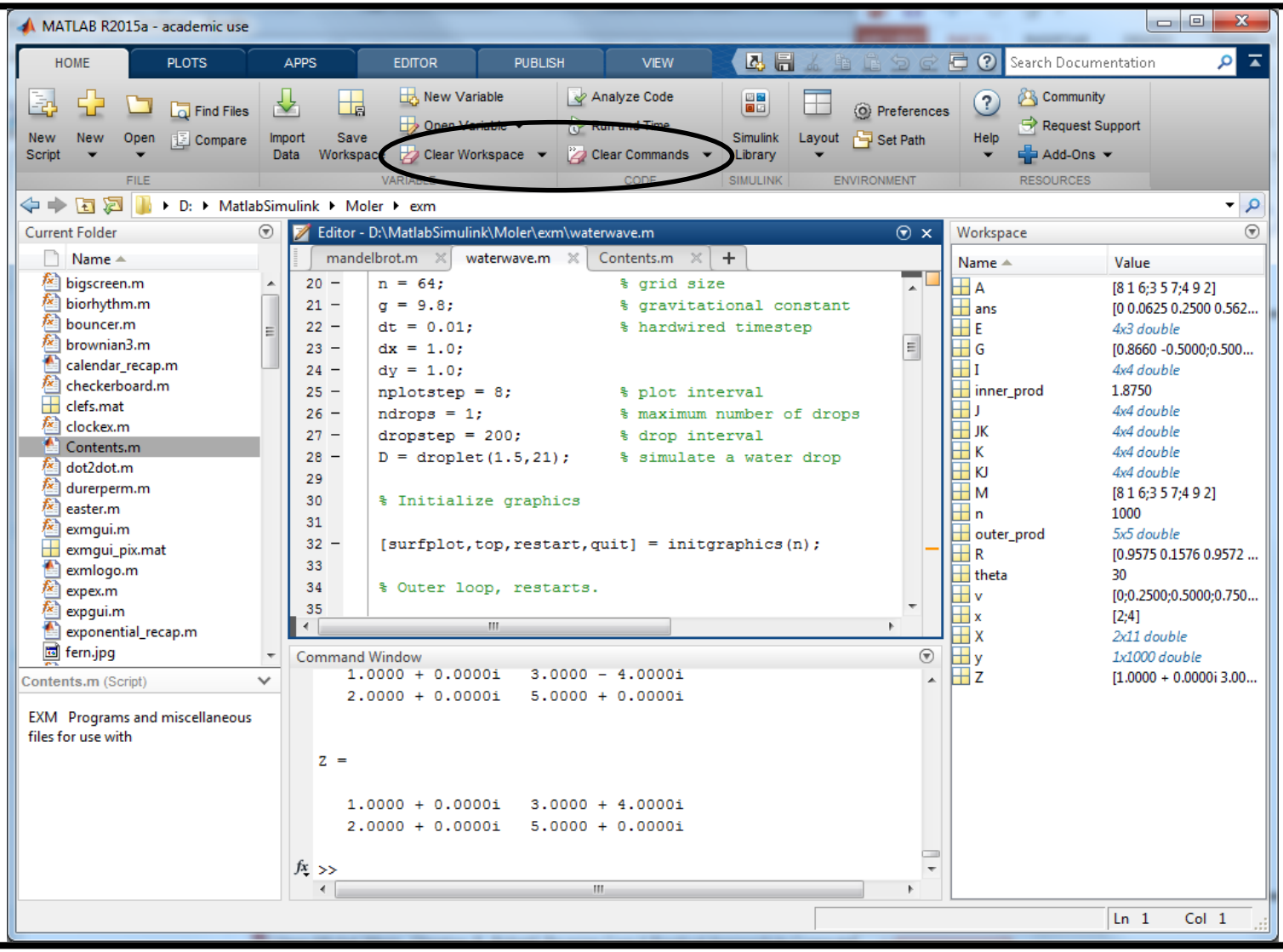

MATLAB R2015a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
D: ▸ MatlabSimulink ▸ Moler ▸ exm
Current Folder
Name
bigscreen.m
biorhythm.m
bouncer.m
brownian3.m
calendar_recap.m
checkerboard.m
clefs.mat
clockex.m
Contents.m
dot2dot.m
durerperm.m
easter.m
exmgui.m
exmgui_pix.mat
exmlogo.m
expex.m
expgui.m
exponential_recap.m
fern.jpg
Contents.m (Script)
EXM Programs and miscellaneous files for use with
Editor - D:\MatlabSimulink\Moler\exm\waterwave.m
mandelbrot.m
waterwave.m
Contents.m
20 n = 64; % grid size
21 g = 9.8; % gravitational constant
22 dt = 0.01; % hardwired timestep
23 dx = 1.0;
24 dy = 1.0;
25 nplotstep = 8; % plot interval
26 ndrops = 1; % maximum number of drops
27 dropstep = 200; % drop interval
28 D = droplet(1.5,21); % simulate a water drop
29
30 % Initialize graphics
31
32 [surfplot,top,restart,quit] = initgraphics(n);
33
34 % Outer loop, restarts.
35
Command Window
1.0000 + 0.0000i 3.0000 - 4.0000i
2.0000 + 0.0000i 5.0000 + 0.0000i
Z =
1.0000 + 0.0000i 3.0000 + 4.0000i
2.0000 + 0.0000i 5.0000 + 0.0000i
>>
Workspace
Name
Value
A [8 1 6;3 5 7;4 9 2]
ans [0 0.0625 0.2500 0.562...
E 4x3 double
G [0.8660 -0.5000;0.500...
I 4x4 double
inner_prod 1.8750
J 4x4 double
JK 4x4 double
K 4x4 double
KJ 4x4 double
M [8 1 6;3 5 7;4 9 2]
n 1000
outer_prod 5x5 double
R [0.9575 0.1576 0.9572 ...
theta 30
v [0;0.2500;0.5000;0.750...
x [2;4]
X 2x11 double
y 1x1000 double
Z [1.0000 + 0.0000i 3.00...
Ln 1 Col 1



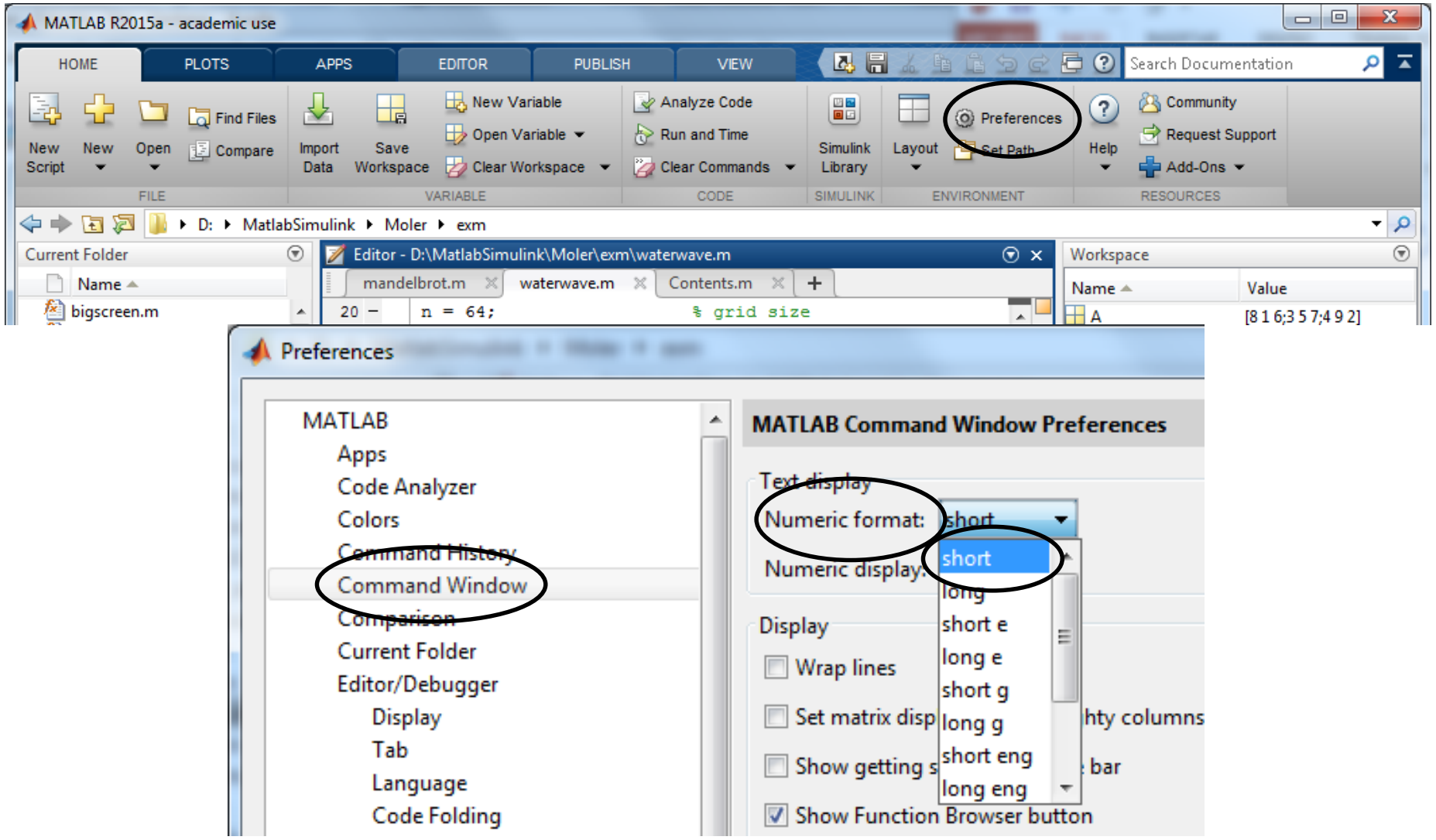
Comandos de limpieza en la ventana de comandos y espacio de trabajo

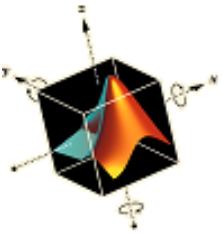
Función MATLAB	Descripción
`clc`	Clear the command window
`clear`	Removes variables from the workspace (computer memory)
`close all`	Closes (deletes) all graphic windows
`format`	Formats the display of numerical output to the command window
`format compact`	Removes empty (blank) lines

Comandos de limpieza en la ventana de comandos y espacio de trabajo



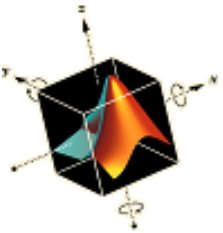
Configuración de formato numérico





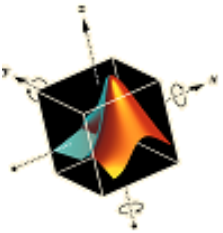
Resultados de diferentes formatos

Opción	Display (número > 0)	Display (número < 0)
short	444.4444	0.0044
long	4.44444444444445e+002	0.00444444444444
short e	4.4444e+002	4.4444e-003
long e	4.44444444444445e+002	4.44444444444444e-003
short g	444.44	0.0044444
long g	444.444444444444	0.00444444444444444
short eng	444.4444e+000	4.4444e-003
long eng	444.444444444444e+000	4.44444444444444e-003
rational	4000/9	1/225
hex	407bc71c71c71c72	3f723456789abcdf
bank	444.44	0.00



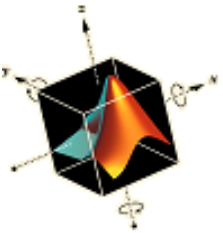
Nombres de Variables

- 63 caracteres alfanuméricos
- Empiezan con una letra (mayúscula o minúscula) seguida por una combinación de letras, números y el caracter _
- Diferencia entre mayúsculas y minúsculas
- La longitud es un compromiso entre identificadores fácilmente reconocibles y legibilidad de las expresiones
- No coincida con una palabra reservada



Palabras reservadas en el lenguaje de programación

`break`	`global`
`case`	`if`
`catch`	`otherwise`
`continue`	`persistent`
`else`	`return`
`elseif`	`switch`
`end`	`try`
`for`	`while`
`function`	



Interacción en la ventana de comandos

```
» p = 7.1
p =
    7.1000
» x = 4.92
x =
    4.9200
» k = -1.7
k =
   -1.7000
» p = 7.1;
» x = 4.92;
» k = -1.7;
» p = 7.1, x = 4.92, k = 1.7
```

← Usuario escribe y pulsa *Enter*

← Respuesta del sistema

← Usuario escribe y pulsa *Enter*

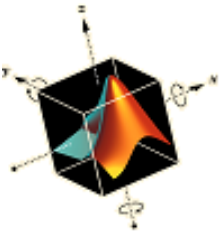
← Respuesta del sistema

← Usuario escribe y pulsa *Enter*

← Respuesta del sistema

← Se usa punto y coma al final para suprimir la respuesta del sistema

← Varias expresiones en una misma línea



Operadores aritméticos

	Prioridad
( ) Paréntesis	1
' Prime (Apóstrofe)	1
^ Exponenciación	2
* Multiplicación	3
/ División	3
\ División izquierda	3
+ Adición	4
– Subtracción	4

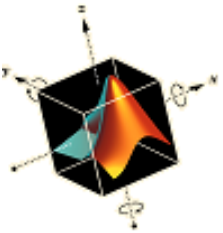
Ejemplos

Expresión Matemática	Expresión MATLAB
$1-dc^{x+2}$	`1-d*c^(x+2)`
$dc^{x}+2$	`d*c^x+2  o  2+d*c^x`
$(2/d)c^{x+2}$	`(2/d)*c^(x+2) o 2/d*c^(x+2) o 2*c^(x+2)/d`
$(dc^{x}+2)/g^{2.7}$	`(d*c^x+2)/g^2.7`
$\sqrt{dc^{x}+2}$	`sqrt(d*c^x+2)  o  (d*c^x+2)^0.5`

Números Complejos: $z = a + 1jb$ o $z = a + 1ib$ $(i = j = \sqrt{-1})$

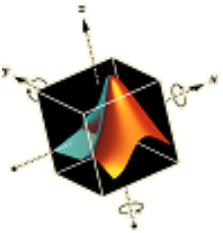
Ejemplo:

```
a = 2;  b = 3;
z = (a+1j*b)*(4-7j);
z = 4 + sqrt(-4);
z = 1i^1i;
```



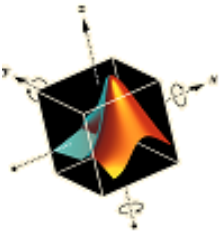
Funciones Elementales MATLAB

Función Matemática	Expresión MATLAB
e^x	`exp(x)`
$e^x - 1$ $\quad x << 1$	`expm1(x)`
$\sqrt{x}$	`sqrt(x)`
$\ln(x)$ o $\log_e(x)$	`log(x)`
$\log_{10}(x)$	`log10(x)`
$\lvert x \rvert$	`abs(x)`
signum(x)	`sign(x)`
$\log_e(1+x)$ $\quad x << 1$	`log1p(x)`
$n!$	`factorial(n)`
Todos los primos $\leq n$	`primes(n)`



Algunas constantes y cantidades especiales MATLAB

Cantidad u operación Matemática	Expresión MATLAB	Comentarios
π	pi	3.141592653589793
$\sqrt{-1}$	i o j	Operador complejo
Precisión relative en punto flotante	eps	$\approx 2.22\times10^{-16}$
∞	inf	Infinito
0/0, 0×∞, ∞/∞	NaN	Operación matemática indefinida
Mayor número en punto flotante antes overflow	realmax	≈ 1.7977e+308
Menor número en punto flotante antes underflow	realmin	≈ 2.2251e-308



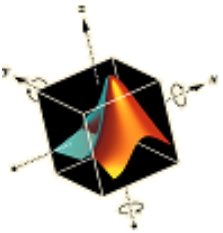
Funciones trigonométricas e hiperbólicas MATLAB

	Trigonométrica			Hyperbólica	
Función	**(radianes)**	**(grados)**	**Inversa**		**Inversa**
Seno	`sin(x)`	`sind(x)`	`asin(x)`	`sinh(x)`	`asinh(x)`
coseno	`cos(x)`	`cosd(x)`	`acos(x)`	`cosh(x)`	`acosh(x)`
tangente	`tan(x)`	`tand(x)`	`atan(x)`†	`tanh(x)`	`atanh(x)`
secante	`sec(x)`	`secd(x)`	`asec(x)`	`sech(x)`	`asech(x)`
cosecante	`csc(x)`	`cscd(x)`	`acsc(x)`	`csch(x)`	`acsch(x)`
cotangente	`cot(x)`	`cotd(x)`	`acot(x)`	`coth(x)`	`acoth(x)`

† **`atan2`(y, x) es la version del cuarto cuadrante.**

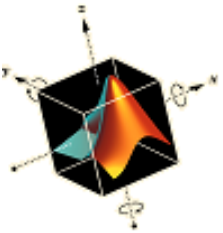
Algunas funciones matemáticas especializadas

Función Matemática	Expresión MATLAB	Descripción
Ai(x)	`airy`(0,x)	Airy function
Bi(x)	`airy`(2,x)	Airy function
$I_\nu(x)$	`besseli`(nu, x)	Modified Bessel function of first kind
$J_\nu(x)$	`besselj`(nu, x)	Bessel function of first kind
$K_\nu(x)$	`besselk`(nu, x)	Modified Bessel function of second kind
$Y_\nu(x)$	`bessely`(nu, x)	Bessel function of second kind
$B(x,w)$	`beta`(x, w)	Beta function
$K(m)$, $E(m)$	`ellipke`(m)	Complete elliptic integrals of 1st & 2nd kind
erf(x), erfc(x)	`erf`(x), `erfc`(x)	Error and complementary error function
$E_1(z)$	`expint`(x)	Exponential integral
Γ(a)	`gamma`(a)	Gamma function
$P_n^m(x)$	`legendre`(n, x)	Associated Legendre function



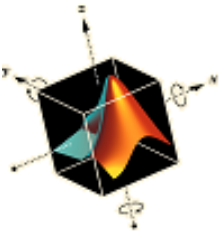
Algunas funciones estadísticas especializadas

Función Matemática	Expresión MATLAB	Descripción
maximum value of x	`max(x)`	Largest element(s) in an array
μ	`mean(x)`	Average or mean value of an array
median	`median(x)`	Median value of an array
minimum value of x	`min(x)`	Smallest element(s) in an array
mode	`mode(x)`	Most frequent values in an array
σ or s	`std(x)`	Standard deviation of an array
σ^2 or s^2	`var(x)`	Variance of an array of values



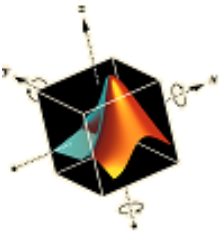
Operadores relacionales MATLAB

Condicional	Símbolo matemático	Símbolo MATLAB
igual	=	==
no igual (diferente)	≠	~=
menor que	<	<
mayor que	>	>
menor que o igual	≤	<=
mayor que o igual	≥	>=



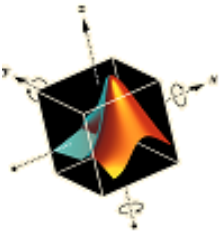
Funciones de conversión decimal a entero

Función MATLAB	*x*	*y*(Resultado)	Descripción
y = `fix`(x)	2.7 –1.9 2.49-2.51`j`	2.0000 –1.0000 2.0000 – 2.0000`i`	Redondeado a cero
y = `round`(x)	2.7 –1.9 2.49–2.51`j`	3.0000 –2.0000 2.0000 – 3.0000`i`	Redondeado al entero más próximo
y = `ceil`(x)	2.7 –1.9 2.49 – 2.51`j`	3.0000 –1.0000 3.0000 -– 2.0000`i`	Redondeado hacia infinito
y = `floor`(x)	2.7 –1.9 2.49 – 2.51`j`	2.0000 –2.0000 2.0000 – 3.0000`i`	Redondeado hacia menos infinito



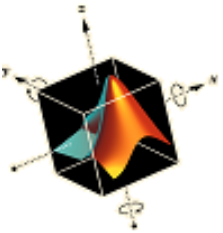
Funciones de manipulación de números complejos

Función MATLAB	*Z*	*y*	Descripción
z = `complex`(a, b)	a + b*`j`	-	Forma número complejo; *a* y *b* real
y = `abs`(z)	3 + 4`j`	5	Valor absoluto: $\sqrt{a^2+b^2}$
y = `conj`(z)	3 + 4`j`	3 – 4`j`	Conjugado Complejo
y = `real`(z)	3 + 4`j`	3	Parte Real
y = `imag`(z)	3 + 4`j`	4	Parte Imaginaria
y = `angle`(z)	a +b*`j`	`atan2`(b, a)	Angulo de fase en radianes: $-\pi \le y \le \pi$



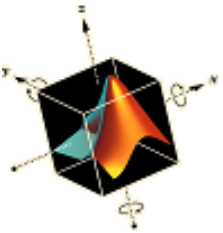
Caracteres especiales y resumen de su uso

Caracter	Nombre	Uso
.	Period	(a) Decimal point. (b) Part of arithmetic operators to indicate a special type of vector or matrix operation, called the dot operation, such as *c* = *a*.**b*.
,	Comma	(a) Separator within parentheses of matrix elements such as *b*(2,7) and functions such as `besselj`(1, x) or brackets creating vectors such as v = [1, x] or the output of function arguments such as [x, s] = `max`(a). (b) Placed at the end of an expression when several expressions appear on one line.
;	Semicolon	(a) Suppresses display of the results when placed at end of an expression. (b) Indicates the end of a row in matrix creation statement such as *m* = [*x y z*; *a b c*].
:	Colon	(a) Separator in the vector creation expression x = a:b:c. (b) For a matrix, *z* it indicates “all rows” when written as *z*(:,*k*) or “all columns” when written as *z*(*k*,:).



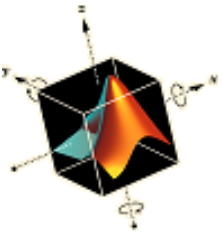
Caracteres especiales y resumen de su uso

( )	**Parentheses**	**(a) Denote subscript of an element of matrix *z*, where *z*(*j*, *k*) is the element in row *j* and column *k.*** **(b) Delimiters in mathematical expressions such as *a*^(*b*+*c*).** **(c) Delimiters for the arguments of functions, such as `sin`(*x*).**
[ ]	**Brackets**	**Creates an array of numbers, either a vector or a matrix, or a string (literal).**
{ }	**Braces**	**Creates a cell matrix or structure.**
%	**Percentage**	**Comment delimiter; used to indicate the beginning of a comment wherein the MATLAB compiler ignores everything to its right. The exception is when it is used inside a pair of quotes to define a string such as *a* = 'p1 = 14 % of the total'.**
%%	**Percentage**	**Used to delimit the start and end of a cell in the MATLAB editor, which is a portion of program code.**
%{ %}	**Percentage and brace**	**Used to enclose a block of contiguous comment lines. Comments placed between these delimiters do not have to be preceded by a %. However, no text can be placed on the line containing these delimiters.**



Caracteres especiales y resumen de su uso

Character	Name	Usage
'	Quote or Apostrophe	(a) '*Expression*' indicates that *Expression* is a string (literal) (b) Indicates the transpose of a vector or matrix.
…	Ellipsis	Continuation of a MATLAB expression to the next line. Used to create code that is more readable.
	Blank	Context dependent: either ignored, indicates a delimiter in a data creation statement such as c = [a b], or is a character in a string statement.
@	At sign	Construct a function handle by placing @ before a function name, such as @FunctionName.
\	Backslash	A mathematical operator to perform certain matrix operations.

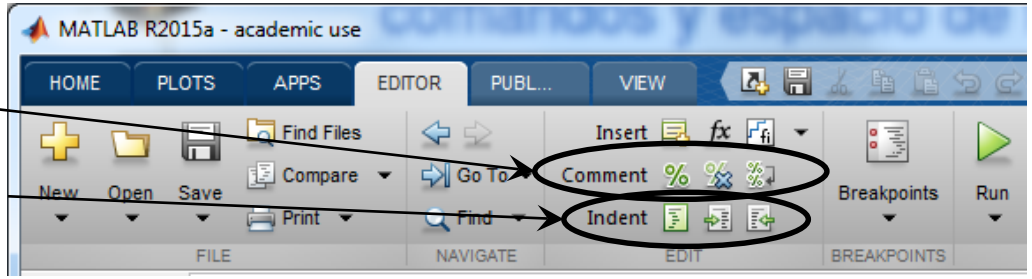


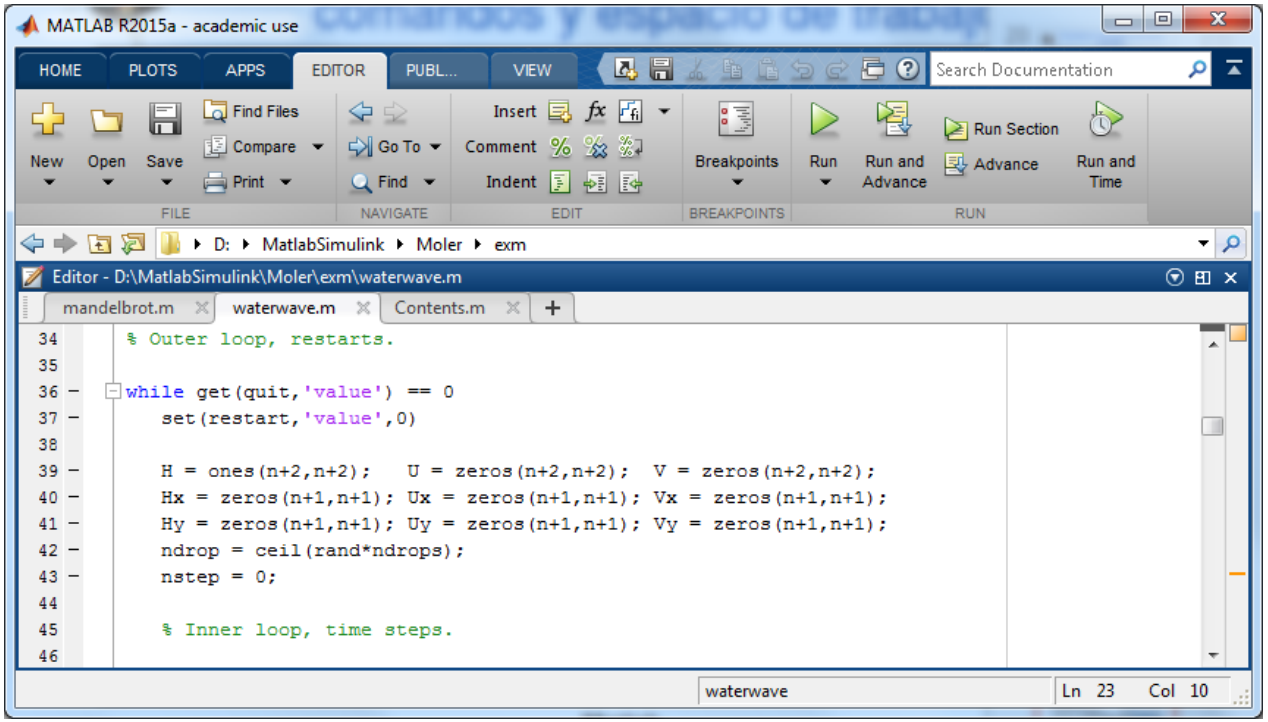
Creación de programas – Uso del editor

- Si el programa contiene varias líneas de código
- Si el programa se usará otra vez
- Si se desea un registro permanente
- Se espera que una actualización se requiera
- Se requiere herramientas de depuración
- Se requiere transferir el código a otra persona u organización
- El editor contiene una serie de características que facilitan la programación

Creación de programas – Uso del editor

- Comentarios
- Indentación inteligente
- Comprobación de paréntesis

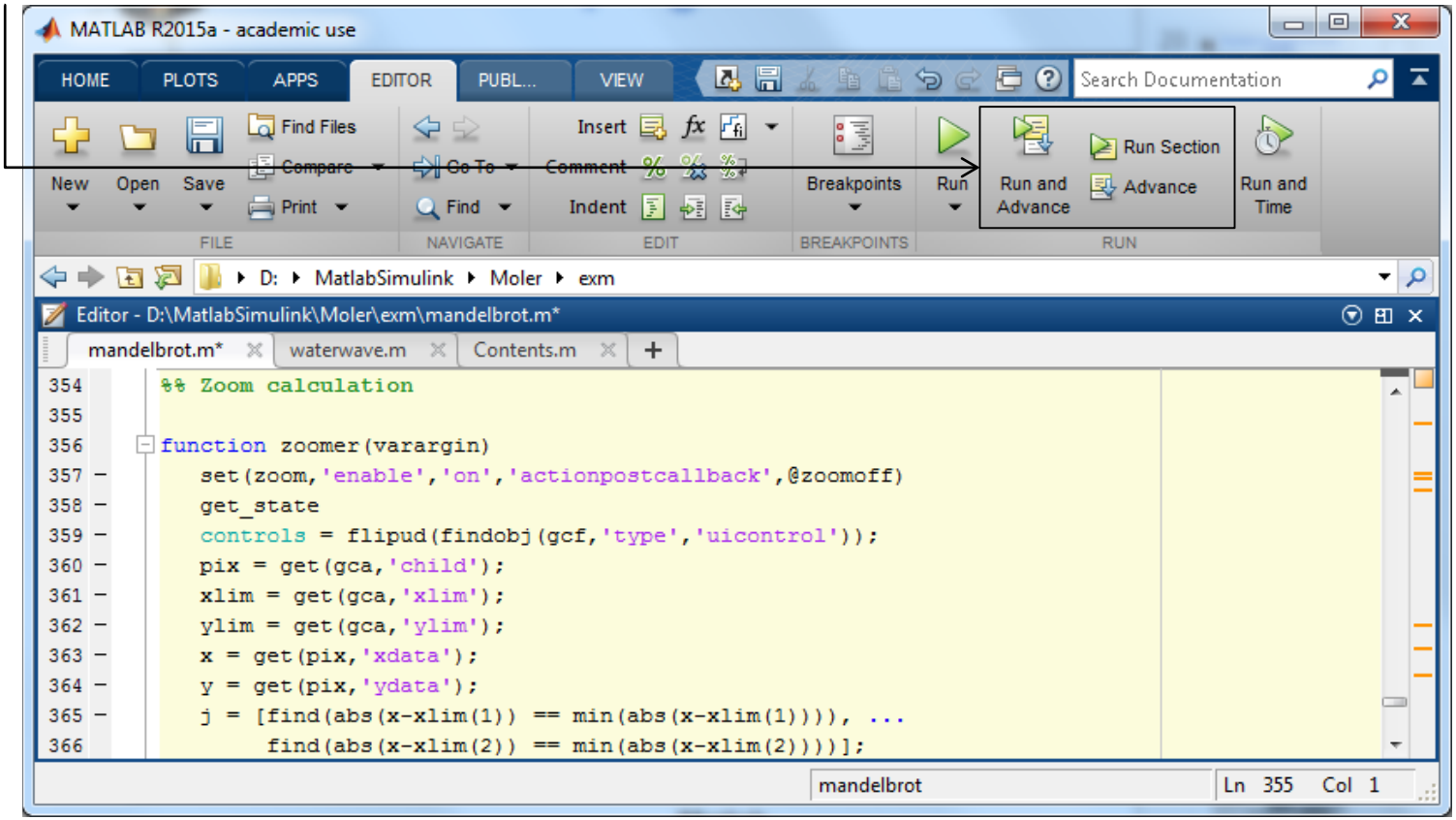


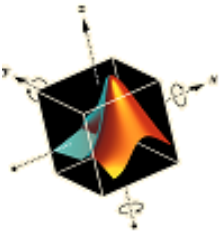


- Comentarios en verde
- Palabras clave en azul
- Cadenas en violeta
- Indentación inteligente

Creación de programas – Uso del editor

- Cell es una sección de código (definida por %%)
- Permite la codificación/ejecución rápida iterativa





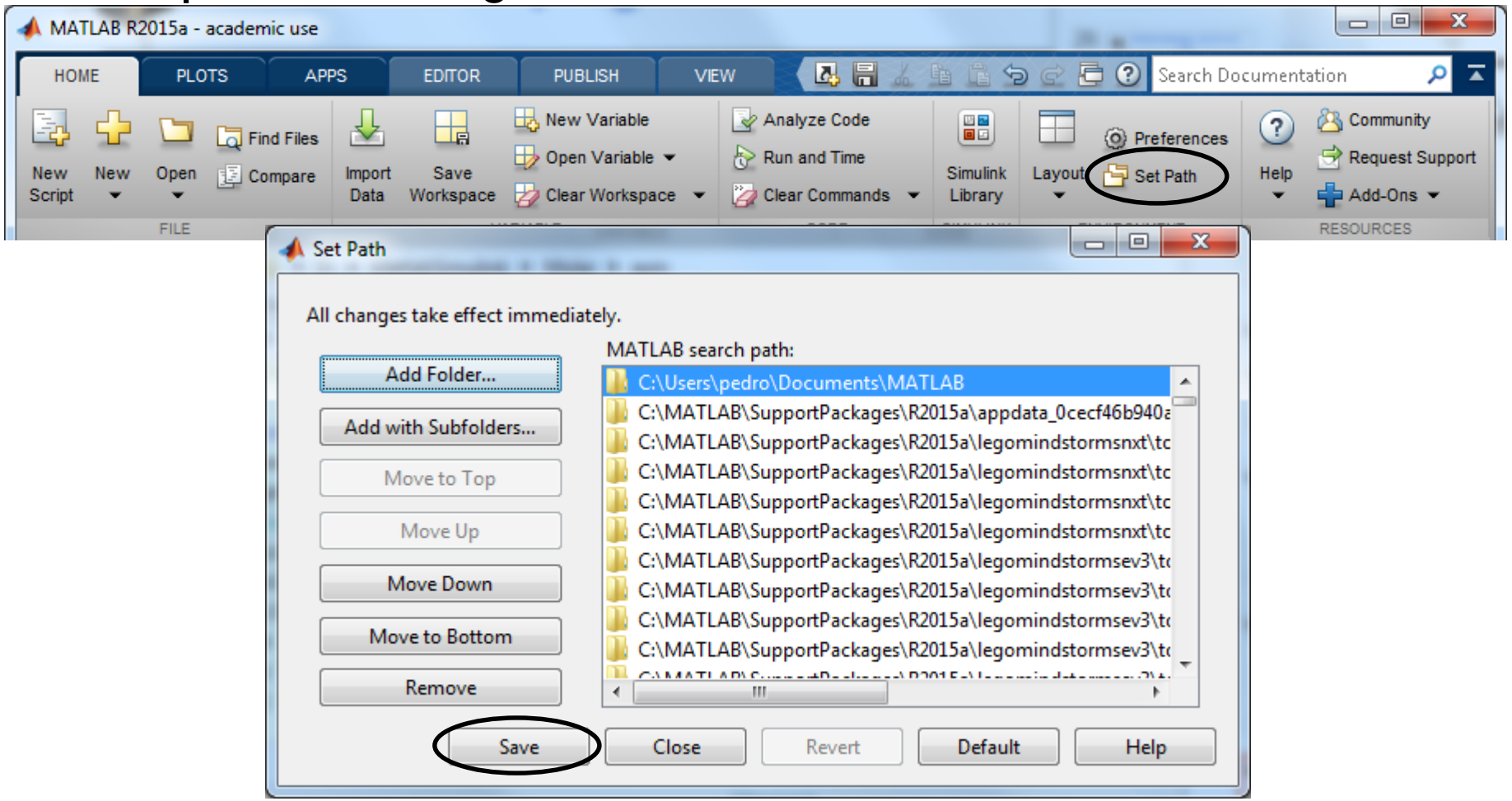
Creación de programas – Atributos de Scripts o Funciones

- Documentación
  - Propósito y operaciones realizadas, nombre del programador, fecha de creación, fecha(s) de revisión, descripción de las entradas, descripción de las salidas
- Input
- Inicialización
- Computación
- Output

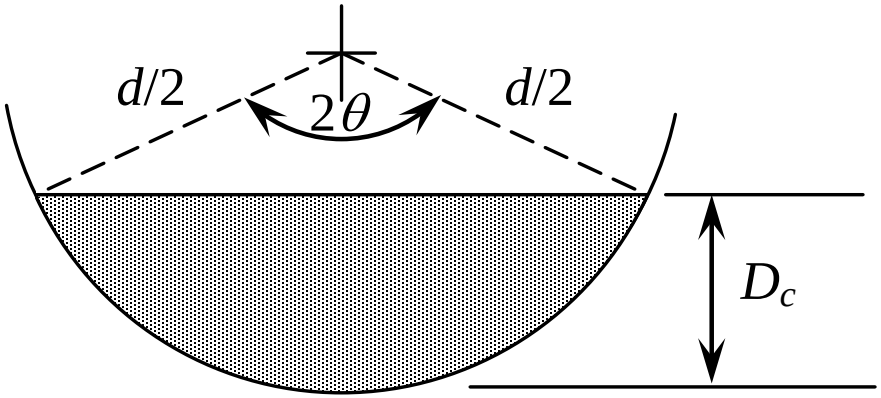
- El nombre del fichero debe coincidir con el de la función y debe tener como sufijo .m (ficheros M)

Ejecución de ficheros M

- Se puede configurar el Path o el directorio en curso



Ejemplo – Flujo en un canal circular



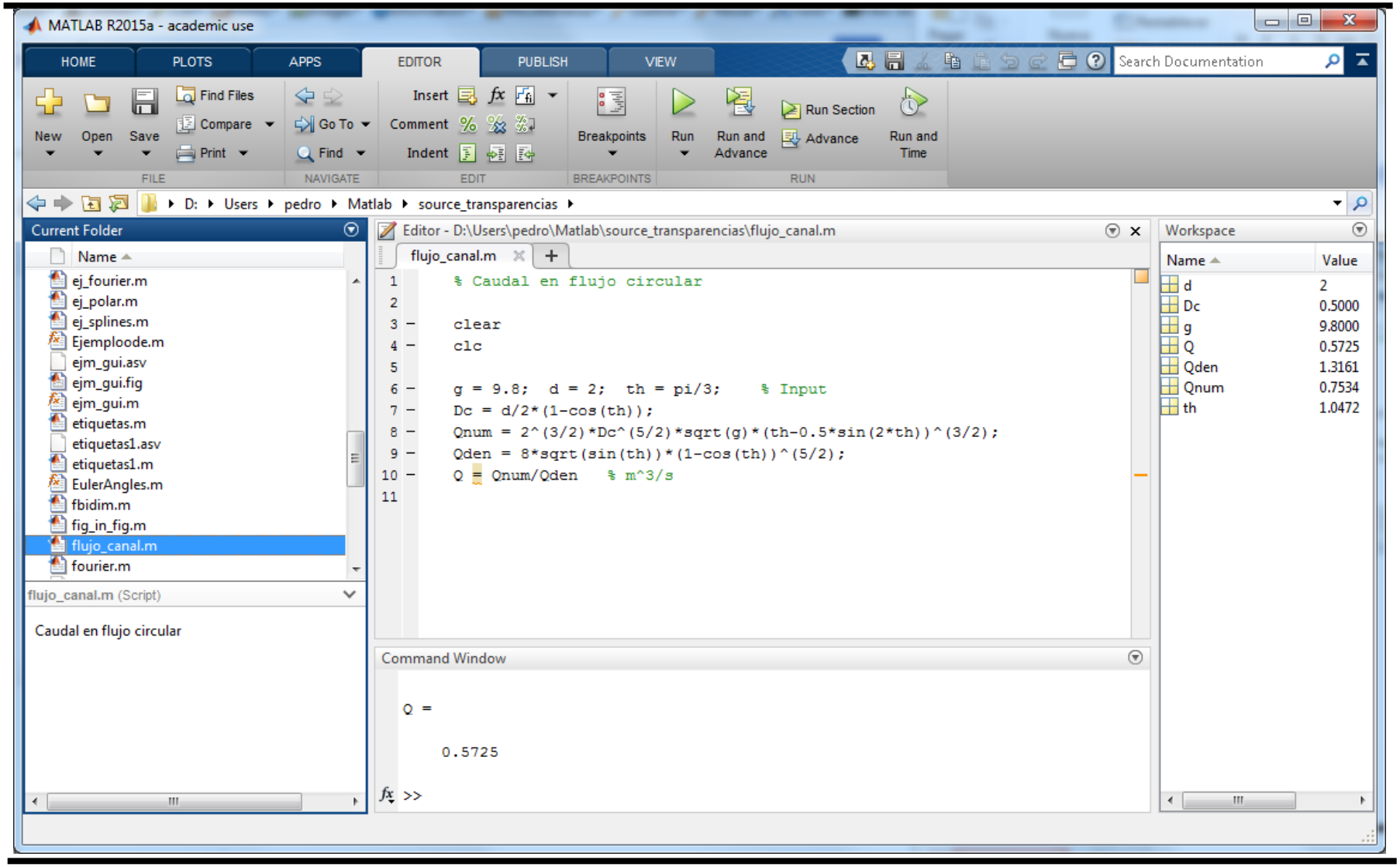
$$Q = \frac{2^{3/2} D_c^{5/2} \sqrt{g} \left(\theta - 0.5\sin(2\theta)\right)^{3/2}}{8\sqrt{\sin\theta}\left(1-\cos\theta\right)^{5/2}}$$

$$D_c = \frac{d}{2}(1-\cos\theta)$$

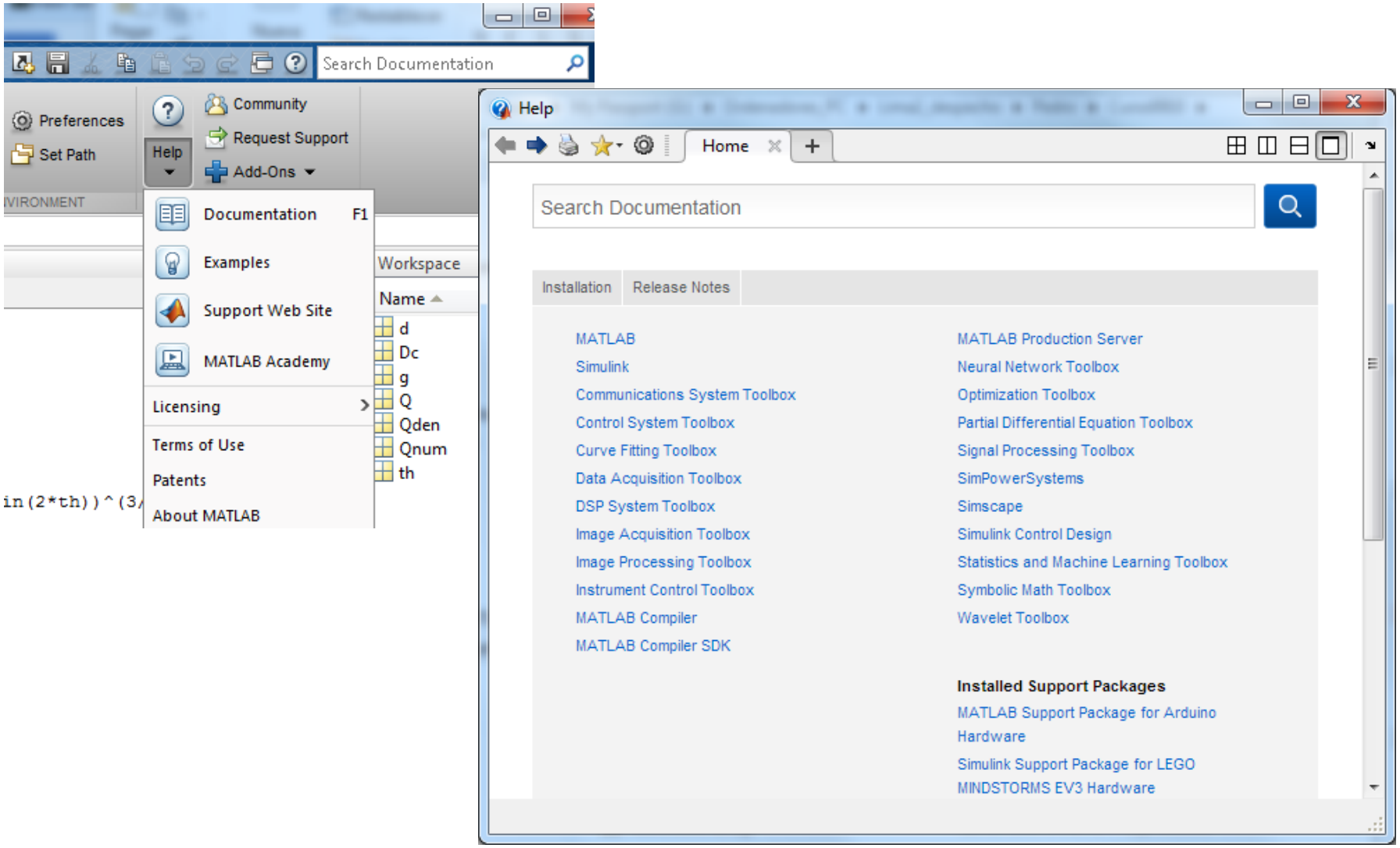
- Si d = 2 m, g = 9.8 m/s^2, y θ = 60° = π/3

```
g = 9.8;  d = 2;  th = pi/3;     % Input
Dc = d/2*(1-cos(th));
Qnum = 2^(3/2)*Dc^(5/2)*sqrt(g)*(th-0.5*sin(2*th))^(3/2);
Qden = 8*sqrt(sin(th))*(1-cos(th))^(5/2);
Q = Qnum/Qden   % m^3/s
```

Ejemplo – Flujo en un canal circular

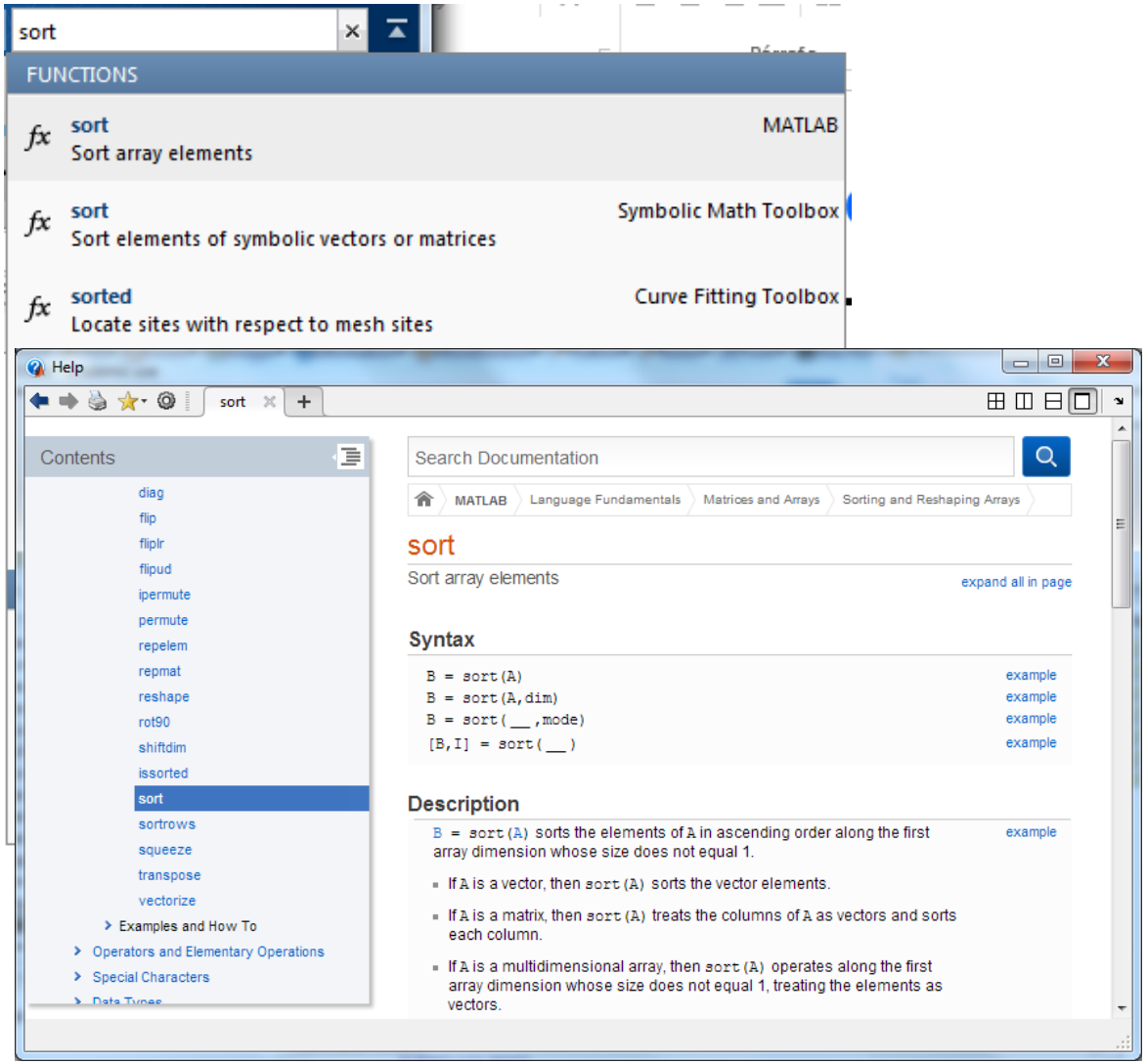


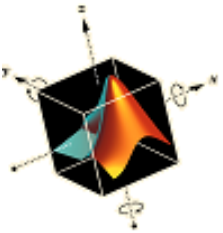
Ayuda disponible



Ayuda disponible: funciones

- Si se conoce el nombre de la función
  - Comand windows: help sort
- Si no se conoce el nombre de la función buscar en documentación





Vectores

- Creación de vectores

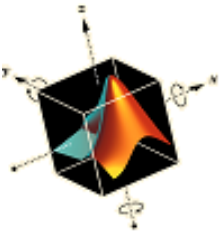
  Por defecto Matlab crea vectores fila

```
f = [a b c]     % 1xn
f = [a, b, c]
f = [a; b; c]  % vector columna nx1
x = s:d:f
x = (s:d:f)
x = [x:s:d]
```

  donde s = valor inicial

  d = incremento o decremento

  f = valor final

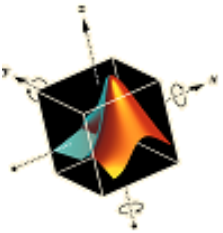


Operaciones con Vectores

- Longitud del vector: `length(x)`
- Transposición: `x=(0.2:0.1:1)'`
- Vector con valores equiespaciados: `x=linspace(s,f,n)`
- Vector con valores equiespaciados (escala logarítmica) : `x=logspace(s,f,n)`
- Acceso a elementos: `b=x(2)  c=x(2:5)`
- Ordena un vector y:

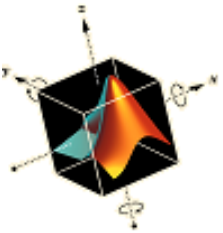
[ynew, indx] = `sort`(y, mode) % mode = 'ascend‘ or 'descend'

donde *ynew* es el vector ordenado y *indx* es un vector conteniendo la ubicación original de los elementos



Operaciones con Vectores

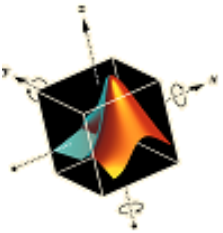
- `find`: Determina los índices de todos los elementos que satisfacen una expresión Expr:
  `indxx = find(Expr)`
- Mínimo *xmin* y su ubicación *locmin*:
  `[xmin, locmin] = min(x)`
- Máximo *xmax* y su ubicación *locmax*:
  `[xmax, locmax] = max(x)`
- Promedio: `x=mean(x)`



Matrices

- Creación de matrices

```
A = [a11 a12 a13; a21 a22 a23; a31 a32 a33]
A = [a11 a12 a13;...
     a21 a22 a23;...
     a31 a32 a33]
A = [a11 a12 a13 %<Enter>
     a21 a22 a23 %<Enter>
     a31 a32 a33]
v1 = [a11 a12 a13];
v2 = [a21 a22 a23];
v3 = [a31 a32 a33];
A = [v1; v2; v3]
```



Operaciones con Matrices

- Orden de una matriz: `[r, c] = size(A)`
  `indxx = find(Expr)`
- Matrices especiales:

  Matriz de 1s: `ones(r,c)`

  Matriz nula 0s: `zeros(r,c)`

  Matriz diagonal (a vector): `diag(a)`

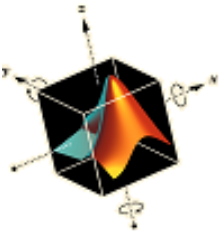
  Vector diagonal (A matriz): `diag(a)`

  Matriz identidad (rango n): `eye(n)`

  Matriz mágica (rango n): `magic(n)`

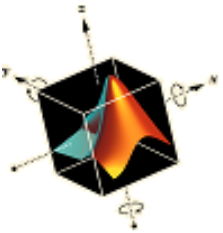
  Intercambio de columnas: `fliplr(A)`

  Intercambio de filas: `flipud(A)`



Operaciones con Matrices

- `find`: Determina los índices de todos los elementos que satisfacen una expresión Expr:
  `[r, c] = find(Expr)`
- Mínimo y máximo opera sobre columnas:
  `minM = min(M)`
  `maxM = max(M)`
  Acceso a elementos: `B=A(1,3)`
  `B=A(1:3,3:5)`
  `B=A(:,2)`
  `B=A(2,:)`



Manipulación de Matrices

- Replicación de un escalar, un vector (fila o columna) o matriz (x) en filas (f) o columnas (c):

```
[r, c] = repmat(x, r, c)
```

- `meshgrid:` Para dos vectores s y t:

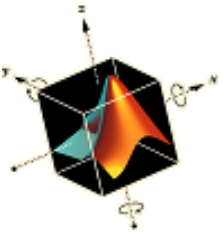
```
[U , V] = meshgrid(s, t)
```

produce el mismo resultado que:

```
U = repmat(s, length(t), 1)
V = repmat(t', 1, length(s))
```

En ambos casos, U y V son matrices de orden:

```
(length(t)xlength(s))
```

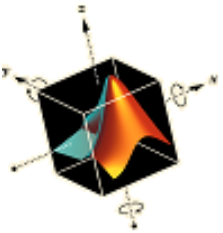


Operaciones con Matrices

- Suma o resta: `C = A ± B`
- Aumento de columnas: `C = [A, B]`
- Aumento de filas: `C = [A; B]`

  `fliplr(A)`
- Operaciones punto: operaciones aritméticas en matrices del mismo orden realizadas elemento a elemento

```
Z = X.*M    % Multiplicación
Z = X./M    % División
Z = X.^M    % Exponenciación
```



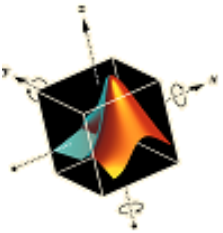
Función Sumatorio

- sum: si v es un vector

$$s = \text{sum}(\text{v}) = \sum_{k=1}^{length(\text{v})} \text{v}_k \qquad s\ escalar$$

- Si Z es una matriz nxm

$$S = \text{sum}(\text{Z}) = \sum_{i=1}^{\text{n}} \text{z}_{i,\text{m}} \qquad S\ vector$$



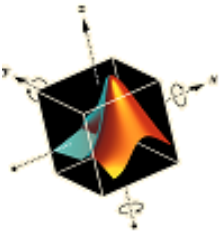
Función Suma acumulada

- cumsum: si v es un vector de n elementos

$$y = \text{cumsum}(\text{v}) = \left[\sum_{k=1}^{1} \text{v}_k, \sum_{k=1}^{2} \text{v}_k, \sum_{k=1}^{3} \text{v}_k \cdots \sum_{k=1}^{\text{n}} \text{v}_k \right]$$

- Si W es una matriz mxn

$$Y = \text{cumsum}(\text{W}) = \begin{bmatrix} \sum_{k=1}^{1} \text{w}_{k1} & \sum_{k=1}^{1} \text{w}_{k2} & \sum_{k=1}^{1} \text{w}_{k3} \cdots & \sum_{k=1}^{1} \text{w}_{kn} \\ \sum_{k=1}^{2} \text{w}_{k1} & \sum_{k=1}^{2} \text{w}_{k2} & \sum_{k=1}^{2} \text{w}_{k3} \cdots & \sum_{k=1}^{2} \text{w}_{kn} \\ \cdots & & \cdots & \\ \sum_{k=1}^{\text{m}} \text{w}_{k1} & \sum_{k=1}^{\text{m}} \text{w}_{k2} & \sum_{k=1}^{\text{m}} \text{w}_{k3} \cdots & \sum_{k=1}^{\text{m}} \text{w}_{kn} \end{bmatrix}$$

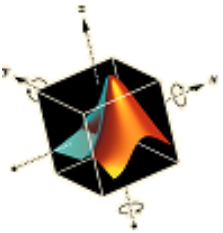


Operaciones con Matrices

- Multiplicación: `C = A * B`
- Matriz transpuesta: `C = A'`
- Aumento de filas: *C* = [*A*; *B*]

  `fliplr(A)`
- Operaciones punto: operaciones aritméticas en matrices del mismo orden realizadas elemento a elemento

```
Z = X.*M    % Multiplicación
Z = X./M    % División
Z = X.^M    % Exponenciación
```



Ejemplo – Suma de Series de Fourier

- La representación en serie de Fourier de un pulso rectangular de duración *d* y periodo *T* es:

$$f(\tau) = \frac{d}{T}\left[1 + 2\sum_{k=1}^{K\to\infty} \frac{\sin(k\pi d/T)}{(k\pi d/T)}\cos(2\pi k\tau)\right]$$

donde $\tau = t/T$. Se pide la suma de 150 términos de $f(\tau)$ ($K = 150$) y gráfica en el rango $-1/2 \le \tau \le 1/2$ cuando $d/T = 0.25$. Además:

$$r(x_i) \to f(\tau_i)$$

$$p_k \to \frac{\sin(k\pi d/T)}{(k\pi d/T)}$$

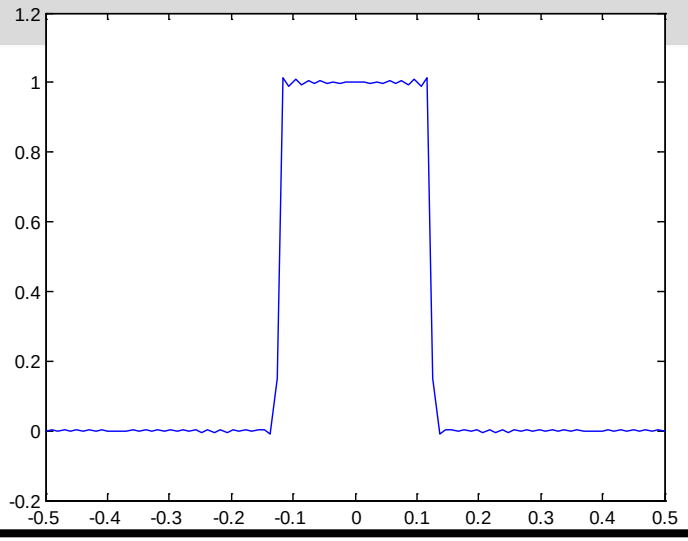
$$h_k(x_i) \to h_k(\tau_i) \to \cos(2\pi k\tau_i)$$

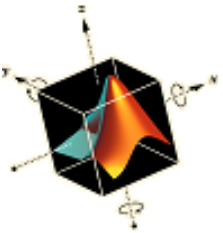
$$r(x_i) = \sum_{k=1}^{m} p_k h_k(x_i) \quad i = 1, 2, ..., n$$

Ejemplo – Suma de Series de Fourier

- El programa en Matlab es:

```
k = 1:150;                                  % (1x150)
tau = linspace(-0.5, 0.5, 100);  % (1x100)
sk = sin(pi*k/4)./(pi*k/4);          % (1x150)
cntau = cos(2*pi*k'*tau );           % (150x100)
f = 0.25*(1+2*sk*cntau);   % (1x150) (150x100) > (1x100)
plot(tau, f )
```





Ejemplo – Evaluación de función bidimensional

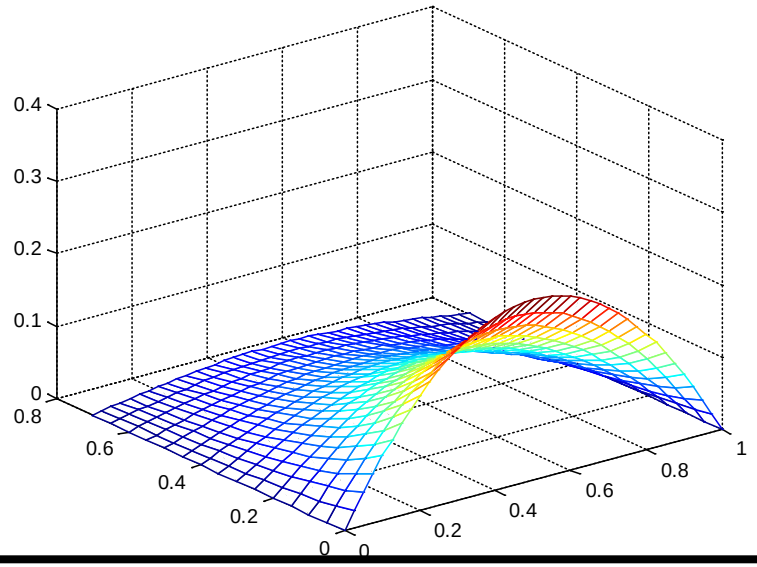
- Se desea calcular la función bidimensional:

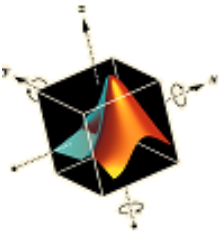
$$u(\xi,\eta) = 4\sum_{n=1}^{N\to\infty} \frac{1-\cos(n\pi)}{(n\pi)^3} e^{-n\pi\xi} \sin(n\pi\eta)$$

considerando que la longitud de ξ es N_x y que la longitud η es N_e

Ejemplo – Evaluación de función bidimensional

```
n = (1:25)*pi;                                          % (1x25)
eta = 0:0.025:1;                                        % (1x41)
xi = 0:0.05:0.7;                                        % (1x15)
[X1, temp1] = meshgrid(xi, (1-cos(n))./n.^3);           % (25x15)
temp2 = exp(-n'*xi);                                    % (25x15)
Rnx = temp1.*temp2;                                     % (25x15)
temp3 = sin(n'*eta);                                    % (25x41)
u = 4*Rnx'*temp3;                                       % (15x41)
mesh(eta, xi, u)
```



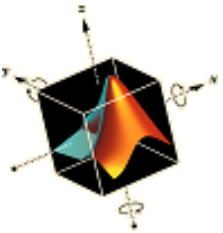


Operaciones con Matrices

- Determinante: `det(A)`
- Problemas autovalor: $|A - \lambda B| = 0$

  `lambda = eig(A,B)`
- Matriz inversa: $A^{-1}A = AA^{-1} = I$

  `inv(A)  o A^-1`



Solución de Sistema de Ecuaciones Lineales

- Un sistema de n ecuaciones:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$
$$\ldots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

- Se puede reescribir como: `Ax = b`

- La expresión preferida para resolver el problema en Matlab es: `x = A\b`
- Forma equivalente: `x = linsolve(A,b)`

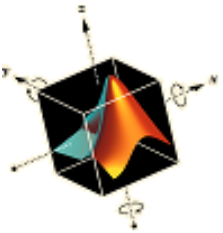
Ejemplo: Solución de Ecuaciones Lineales

```
>> A = [8, 1, 6; 3, 5, 7; 4, 9, 2];
>> b = [7.5, 4, 12]';
>> x = A\b
>> z = A*x
```

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7.5 \\ 4 \\ 12 \end{bmatrix}$$

Cuyo resultado es:

```
x =
    1.2931
    0.8972
   -0.6236
z =
    7.5000
    4.0000
   12.0000
```



El Problema de “mínimos cuadrados”

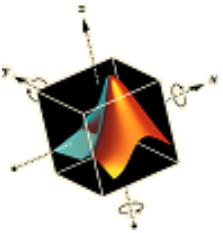
- Si A es una matriz nxm, y b es un vector nx1, sea c* el menor valor posible (para todas las posibles de vectores x de mx1)
- Expresado matemáticamante:

$$c^* = \min_{x,\, mx1} \left\| Ax - b \right\|$$

- Donde la *norma* de v (vector de mx1) es:

$$\left\| v \right\| = \sqrt{\sum_{k=1}^{m} \left| v_k \right|^2}$$

- La solución Matlab es: `x = A\b`

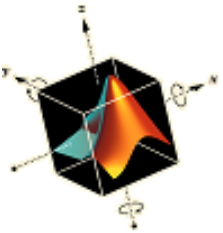


Casos del problema de “mínimos cuadrados”

- Cuatro casos de `x=A\b` como solución de

$$c^* = \min_{x, m_x 1} \|Ax - b\|$$

- No hay diferencia:
  - c*=0, y sólo un vector x cumple este mínimo.
  - c*=0, y varios vectores x diferentes cumplen el mínimo. De todos estos, selecciona el más pequeño (norma)
- Hay diferencia
  - c*>0, sólo un vector x cumple este mínimo.
  - c*>0, y varios vectores x diferentes cumplen el mínimo. De todos estos, selecciona el de menor norma



Guardar el espacio de trabajo (Workspace)

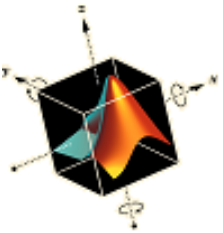
- Cuando se sale de Matlab, las variables del Workspace (WS) desaparecen
- Se pueden guardar todas las variables o algunas de ellas utilizando el comando `save`

  ```
  >> save
  ```

  guarda todas las variables en el WS en el fichero matlab.mat en el directorio en curso

  ```
  >> save myWorkS
  ```

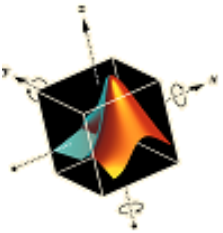
  guarda las variables en el WS en el fichero myWorkS.mat



Guardar el espacio de trabajo (Workspace)

```
>> save mainVar A B C D*
```

guarda las variables A, B, C y cualquier variable que empieza por D del WS en el fichero mainVar.mat



Cargar el WS desde un fichero .mat

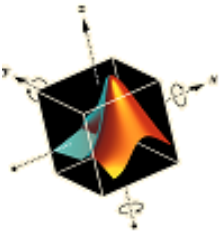
- Para leer un fichero .mat poniendo todas las variables contenidas en el WS se usa `load`, que hace lo opuesto que `save`.

  `>> load`

  carga todas las variables del fichero matlab.mat en el directorio en curso

  `>> load myWorkS`

  carga todas las variables del fichero myWorkS.mat



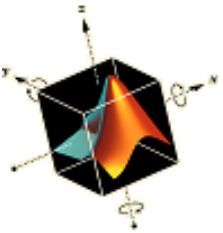
Cadenas (Strings)

- Colecciones de combinaciones de letras, números, y caracteres especiales
- Se tratan como vectores y se definen entre comillas simples '...'

```
>> st = ['ts11', 'ts12'; 'ts13', 'ts14']
```

- Operaciones comunes:

`findstr(s1, s2)` – buscar un substring en un string
`char(string)` – rellena con blancos al final
`deblank` – elimina blancos finales
`strtrim` – eliminar los blancos delante y final
`strcmp (s1, s2)` – compara los strings s1 y s2



Conversión de valores numéricos a Strings

- Para convertir un valor numérico a un string se usa

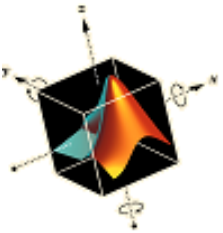
```
>> z = num2str(num)      o
>> z = num2str(num, N)
```

  donde z es un string, num es un número, array de números o expresión resultante en un número o array de números. N es el número de dígitos a considerar sin considerar el punto decimal (si se omite N=5)

- Para convertir un entero a un string se usa

```
>> z = int2str(num)
```

  donde num es un entero. Si num no es un entero se redeondea a uno



Impresión de mensajes

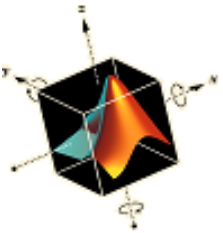
- Para imprimir un mensaje se usa `disp`

```
>> num = 12.567;
>> disp(['Peso = ' num2str(num) ' kg'])
```

debe haber al menos un blanco antes y después de num2str

- Para crear un mensaje a cada valor de un array se usa `repmat`

```
>> num = [12.567, 3458, 9.111]
>> n = length(num);
>> disp([repmat('long. objeto = ',n,1)
   num2str(num') repmat(' m', n,1)])
```



Impresión con formato

- Una forma alternativa de mostrar mensajes con datos formateados es con

  `fprintf(1,'%...',variables)`

  donde: 1 indica que se imprima en la ventana de comando,

  '%...' es la especificación de formato de las variables

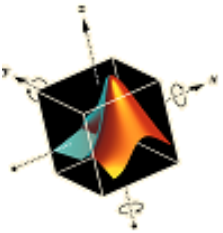
  % precede cada especificación de formato con la forma: x.yq

  q especifica el formato

  x especifica el número mínimo de dígitos a imprimir

  y es el número de dígitos decimales

```
>> num = [12, -14, 3098.458, 0.11167];
>> fprintf(1,'%5.2f',num)
```



Entrada de datos con input

- Entrada de valor numérico:

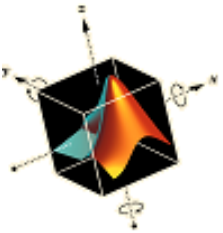
```
inputn = input('Temp. en grados C: ');
```

si es un vector o matriz hay que escribir los corchetes y seguir la notación de Matlab

```
inpn=input('Temp. grados C: ')*pi/180;
```

- Entrada de string:

```
inps=input('Nombre fichero: ','s');
```



Entrada/Salida de datos desde fichero

- Para leer desde fichero se usa:

```
load('nombrefichero.ext');
```

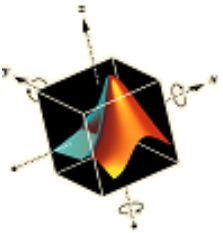
la lectura se realiza por fila. El número de columnas debe ser la misma para cada fila. Los datos se almacenan en una variable de nombre igual a nombrefichero

- Ejemplos: fichero tipo texto con nombre datosexp1.txt

```
load('datosexp1.txt')
```

- Lectura de nombre de fichero

```
f=input('Fichero (con sufijo): ','s');
load(f); m = findstr(f,'.');
data = eval(f(1:m-1)); y = data.^2
```



Entrada/Salida de datos desde fichero

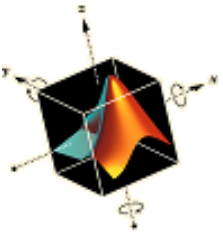
- Para almacenar datos en un fichero se usa:

```
save('nomfich','var1','var2',...,'-ascii')
```

nomfich es un string con el nombre del fichero.
var1, var2,.. Son strings con los nombres de las n variables que se guardan en el orden que aparecen
-ascii es una cadena que indica que los datos se guardan en formato ascii

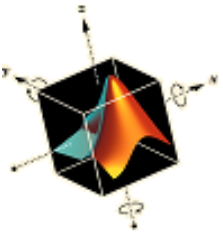
- Ejemplo: lectura, cálculo y excritura de datos

```
load('datosexp2.txt')
y = datosexp2.^2;
save('savedatosexp2.txt','y','-ascii')
```



Cell Arrays

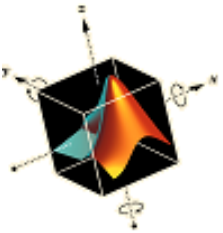
- Los cell arrays son clases especiales de arrays cuyos elementos consisten en celdas que contienen a su vez arrays
- Proporcionan una forma jerárquica de almacenar varios tipos de datos
- El acceso a una celda se realiza mediante índices, de forma similar al acceso de vectores y matrices
- La notación de celda difiere de la notación standard matricial porque utiliza llaves ({ }) en lugar de corchetes



Ejemplo de cell arrays

- Para almacenar datos en un fichero se usa:

```
A = ones(3,2);
B = magic(3);
C = char('presion','temperatura','nivel');
D = [6+7j, 15];
cell = {A, B; C, D};
celldisp(cell)   % para mostrar contenido
cell_12 = cell{1, 2} % para acceder el 1,2
```



Entrada de datos desde ficheros Excel

- Para leer desde fichero en formato Excel se usa:

```
[X, Y] = xlsread('nombfich.xls');
```

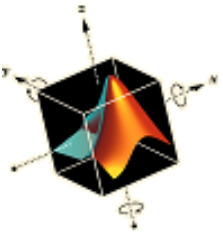
donde X será un array que contiene las filas y columnas de datos e Y será un cell array conteniendo los textos de los encabezados que acompañan los datos

- Ejemplos: para un fichero Excel datosexp2.xls

```
[X, Y] = xlsread('datosexp2.xls')
```

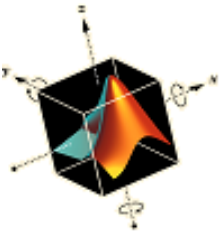
	A	B
1	Respuesta Transductor	
2	Fuerza	Desplazamiento
3	(kPa)	(mm)
4	100	0,1
5	110	0,2
6	135	0,33
7	150	0,4
8	175	0,55

```
X =
  100.0000    0.1000
  110.0000    0.2000
  135.0000    0.3300
  150.0000    0.4000
  175.0000    0.5500

Y =

    [1x21 char]                 ''
    'Fuerza'        'Desplazamiento'
    '(kPa)'         '(mm)'
```



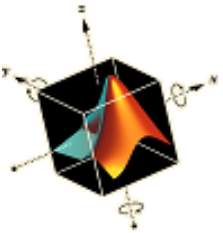
Control del flujo

- El control del flujo de un programa se realiza mediante cuatro estructuras de control `while`, `if`, `for`, y `switch`
- Cada vez que se usa una estructura de control, debe colocarse al final del bloque de instrucciones que la componen, una sentencia `end`
- Las estructuras de control se pueden *anidar*
- Es frecuente usar operadores lógicos y relacionales para determinar la condición de las estructuras de control



Operadores relacionales y lógicos

Condicional	Símbolo matemático	Símbolo MATLAB
Operadores relacionales		
igual	=	==
no igual	≠	~=
menor que	<	<
mayor que	>	>
menor que o igual	≤	<=
mayor que o igual	≥	>=
Operadores lógicos		
y	AND	& o &&
o	OR	| o ||
no	NOT	~



Operador lógico

- Ejemplo: Crear una función g(x) sujeta a:

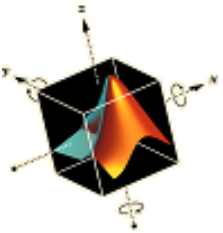
  $g(x) = f(x)$ $\quad a \leq x < b$

  $= 0$ $\quad x < a$ y $b \geq x$

- El resultado de una expresión lógica es 1 si se cumple y 0 si es falsa
- Si a = -1, b = 2 y $f(x) = e^{x/2}$, x = [-4,-1,1,4]

```
a = -1;  b = 2;
x = [-4, -1, 1, 4];
r = (a <= x)
p = (x < b)
logi = (r & p)
gfx = exp(x/2).*logi
```

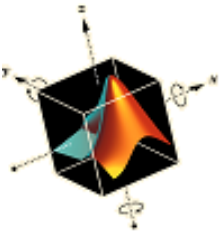
```
a = -1;  b = 2;
x = [-4, -1, 1, 4];
gfx = exp(x/2).*((a <= x) & (x < b))
```



Control del flujo: sentencia if

- La sentencia if permite la ramificación a diferentes partes de su estructura dependiendo de la satisfacción de expresiones condicionales
- La sintaxis general es:

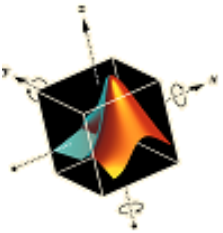
```
if condicion 1
   sentencias 1
elseif condicion 2  % opcional
   sentencias 2
else                % opcional
   sentencias 3
end
```



Control del flujo: sentencia switch

- Equivale al uso de estructuras if-elseif-else-end. La expresión condicional del switch debe ser integral
- La sintaxis general es:

```
switch expresion_switch
   case expresion_case 1
     sentencias 1
   case expresion_case n
     sentencias n
   otherwise
     sentencias n+1
 end
```



Control del flujo: ciclo for

- El ciclo for repite una serie de instrucciones un número específico de veces
- La sintaxis general es:

```
for variable = expresion
   sentencias
end
```

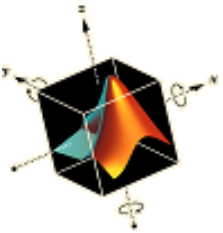
```
N = input('Ingresa un numero positivo < 15: ');
Matr = zeros(N, N);
for r = 1:N
    Matr(r,1:N) = ((r-1)*N+1):r*N;
end
disp(Matr)
```

$a_{11} = 1$ y $a_{1n} = N$

$a_{21} = N+1$ y $a_{2n} = 2N$

…

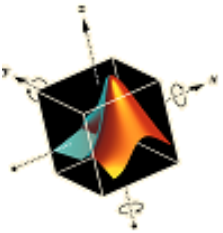
$a_{N1} = (N-1)\mathrm{N} + 1$ y $a_{NN} = N^2$



Control del flujo: ciclo while

- El ciclo while repite una o varias instrucciones un número indefinido de veces hasta que se cumpla una condición
- La expresión que define la condición usualmente está compuesta de una o más variables evaluadas en las sentencias
- La sintaxis general es:

```
while condicion
   sentencias
 end
```

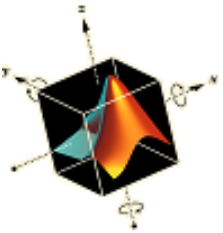


Control del flujo: sentencia break

- break se usa para terminar los ciclos for o while
- Si el break se produce en un for o while anidado, entonces continúa en la siguiente instrucción que sigue al end del ciclo donde se encuentra
- Ejemplo:

```
for j = 1:15
   ...
   b = 1
   while b < 25
   ...
      if n < 0
         break
      end
      ...
   end
   ...
end
```

Cuando *n* < 0 el ciclo while se termina y el script continúa en la siguiente instrucción después del end



Control del flujo: try/catch

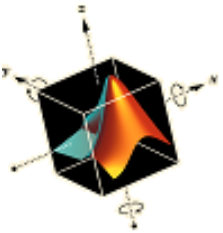
- try/catch se usa para ejecutar sentencias dependiendo de la existencia de errores en la sección try
- La sintaxis es:

```
try
    sentencias1
catch
    sentencias2
end
```

Se intenta ejecutar estos comandos

Si ocurre un error en tiempo de ejecución mientras se ejecutan las sentencias1 se pasa a ejecutar estos comandos

Si no hay error en la ejecución de sentencias1, salta hasta end, sin ejecutar sentencias2



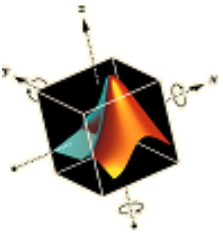
Ejemplo – Convergencia de una serie

- Determinar y mostrar el número de términos que toma a la serie:

$$S_N = \sum_{n=1}^{N} \frac{1}{n^2}$$

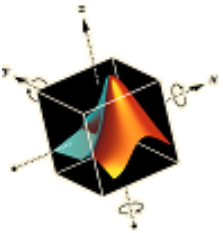
para converger a su valor exacto $S_\infty = \pi^2/6$ con una precisión de 0.01%

```
serie = 1;  k = 2;  exacto = pi^2/6;
while abs((serie - exacto)/exacto) >= 1e-4
    serie = serie+1/k^2;
    k = k+1;
end
disp(['Numero de terminos = ' num2str(k-1)])
```



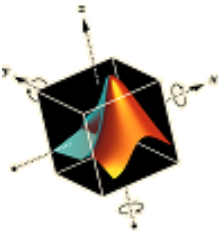
Ficheros Función

- Los ficheros función son ficheros con programas (script) que crean su propio espacio de trabajo e independiente dentro de Matlab
- Las variables definidas de una función son locales a esa función con excepción de las variables `global`
- El nombre de una función debe reflejar su propósito
- Los ficheros función residen en un fichero cuyo nombre debe ser igual al nombre de la función, salvo que tenga extensión .m.
- La primera línea tiene un formato requerido por Matlab



Funciones

- Las funciones son programas (script) que crean su propio espacio de trabajo e independiente dentro de Matlab
- Las variables definidas de una función son locales a esa función con excepción de las variables `global`
- El nombre de una función debe reflejar su propósito
- Los ficheros función residen en un fichero cuyo nombre debe ser igual al nombre de la función, salvo que tenga extensión ".m"
- La primera línea tiene un formato requerido por Matlab



Funciones

- El formato general de la interfaz de una función es:

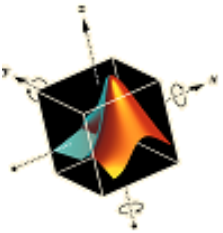
```
function [OutputVariables] = NombreFunc(InputVariables)
% Comentarios
expresion(es)
```

donde

OutputVariables es una lista de nombres separados por comas de las variables de salida

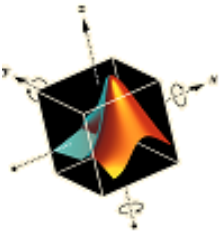
InputVariables es una lista de nombres separados por comas de las variables de entrada

- El nombre del fichero que contiene la función debe ser *NombreFunc.m*



Formas de crear funciones

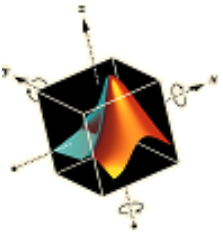
- Fichero función: creado con `function`, almacenado en un fichero y accedido por scripts y funciones o desde la ventana de comandos
- Sub función: son las funciones creadas después de la primer `function` en un fichero función, accesibles por la función principal y las otras subfunciones
- Función anónima: forma de crear funciones sin crear ficheros-m o subfunciones
- Función `inline`: creada con el comando inline y limitada a una expresión Matlab



Ficheros Función: Caso especial 1

- Las funciones se pueden usar para crear una figura, mostrar datos en la ventana de comandos, o escribir datos en ficheros.
- En estos casos no hay valores retornados a la función de llamada, que es un script u otra función que usa esta función en una o más de sus expresiones.
- En este caso, la interfaz de la función es:

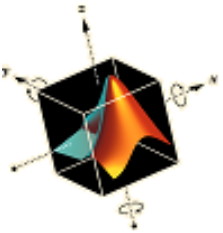
  `function` NombreFunc(InputVariables)



Ficheros Función: Caso especial 2

- Cuando una función se usa sólo para almacenar datos de una manera prescrita, la función no requiere ningún argumento de entrada
- En este caso, la línea que define la interfaz de la función es:

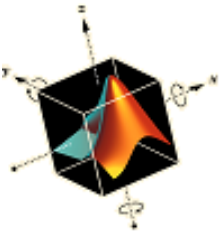
```
function [OutputVariables] = NombreFunc
```



Ficheros Función: Caso especial 3

- Cuando una función convierte un script en una función principal para reducir el número de ficheros m
- La línea que define la interfaz de la función es:

  `function` [OutputVariables] = NombreFunc

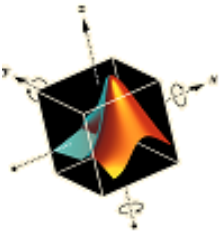


Recursividad

- Técnica donde una función, para cumplir con una tarea, se llama a sí misma
- Cada solución recursiva involucra dos partes.
  - Caso base: el problema es suficientemente simple para resolverse directamente
  - Llamada recursiva: divide el problema en problemas más simples (pequeños) e invoca la solución de cada problema y los combina
- El ejemplo típico es el cálculo factorial:

fact(n) = n*fact(n-1) n > 0

fact(n) = 1 n = 0



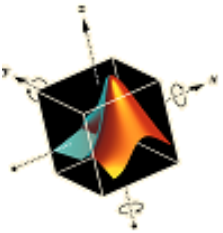
Ejemplos de recursividad

- Factorial

```
function f = fact(n)
if n == 0
  f = 1;
else
  f = n * fact(n-1);
end
```

- Torres de Hanoi

```
function hanoi(n, i, a, f)
if n > 0
  hanoi(n-1, i, f, a);
  fprintf('mover disco %d de %c a %c\n',n,i,f);
  hanoi(n-1, a, i, f);
end
```



Ejemplo de creación de una función

- Se requiere calcular las siguientes ecuaciones en una función

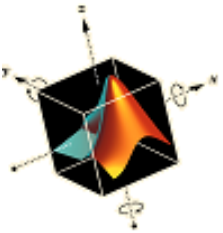
$$x = \cos(at) + b$$
$$y = |x| + c$$

- Valores retornados por la función: x e y la función.
  Nombre de la función: CalcularXY (fich. CalcularXY.m)

```
function [x, y] = CalcularXY(t, a, b, c)
% Calculo de:
%     x = cos(at)+b
%     y = |x|+c
% Escalares: a, b, c
% Vectores: t, x, y
x = cos(a*t)+b;
y = abs(x)+c;
```

>> help CalcularXY

```
Calculo de:
    x = cos(at)+b
    y = |x|+c
Escalares: a, b, c
Vectores: t, x, y
```



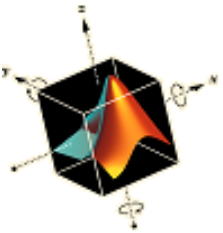
Ejemplo de creación de una función

- Si en la ventana de comandos se escribe:

```
[u,v] = CalcularXY(0:pi/4:pi, 1.4, 2, 0.75)
```

- Se obtiene

```
u =
    3.0000    2.4540    1.4122    1.0123    1.6910
v =
    3.7500    3.2040    2.1622    1.7623    2.4410
```



Variables globales

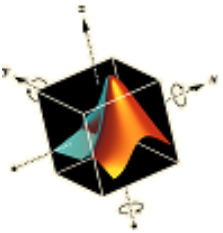
- En ocasiones el número de variables que son transferidas a una función puede ser grande
- Por ello se puede compartir la memoria global del script o función o crear crear acceso a variables globales para uso de varios funciones con `global`

```
function [x, y] = CalcularXY(t)
global A B C
x = cos(A*t)+B;
y = abs(x)+C;
```

Invocación:

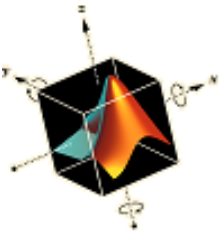
```
global A B C
A = 1.4; B = 2; C = 0.75;
[u,v] = CalcularXY(0:pi/4:pi)
```

Se debe usar espacios en blanco en lugar de comas entre los nombres de las variables globales



Subfunciones

- Si se usa `function` más de una vez en un fichero función, todas las funciones creadas después de la primera función se llaman subfunciones
- La función principal son las expresiones que están después del primer uso de `function` y la única que es accesible desde la ventana de comandos, scripts y otros ficheros función
- Las subfunciones son accesibles sólo por la función principal y otras subfunciones dentro del fichero de la función principal



Ejemplo de subfunciones

- Cálculo del promedio y desviación estandard de un vector de valores numéricos

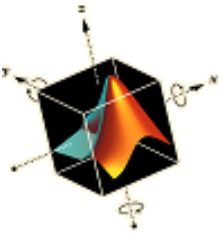
```
function [m, s] = MeanStdDev(dat)    % Main function
n = length(dat);  m = meen(dat, n);  s = stdev(dat, n);

function m = meen(v, n)              %  Sub funcion
m = sum(v)/n;

function sd = stdev(v, n)            %  Sub funcion
m = meen(v, n);                      %  llamada a una subfuncion
sd = sqrt((sum(v.^2) - n*m^2)/(n-1));
```

$$m = \frac{1}{n}\sum_{k=1}^{n} x_k$$

$$s = \left[\frac{1}{n-1}\left(\sum_{k=1}^{n} x_k^2 - nm^2\right)\right]^{1/2}$$

Un script que usa la función es

```
v = [1, 2 , 3, 4];
[m, s] = MeanStdDev(v)
```



Función Anónima

- La forma general de una `anonymous function` es:

  functionhandle = @(argumentos)(expresion)

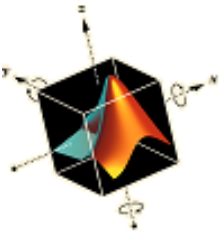
  donde:

  functionhandle es el manejador de la función (referencia) usado para invocarla y un medio de pasar la función como argumento en funciones, que se evalúa con `feval`

  argumentos es una lista de variables separada por comas

  expresion es una expresión Matlab válida

- Cualquier parámetro que aparece en *expresion* y no aparece en *argumentos* debe asignarse un valor numérico `antes` de esta sentencia



Ejemplo de función anónima

- Creación de función anónima para evaluar:

  $c_x = \left|\cos(\beta x)\right|$ con $\beta = \pi/3$ y $x = 4.1$

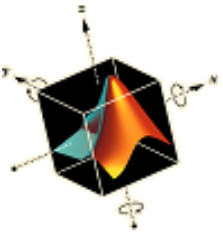
```
bet = pi/3;  % los parametros deben asignarse antes de la función
cx = @(x) (abs(cos(bet*x)));
disp(cx(4.1))
```

Versión con dos argumentos:

```
cx = @(x, bet) (abs(cos(bet*x)));
disp(cx(4.1, pi/3))
```

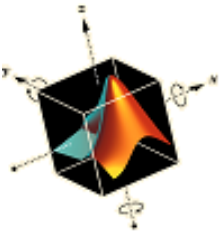
Uso de la versión con dos argumentos en otra func. anónima:

```
cx = @(x, bet) (abs(cos(bet*x)));
crt = @(x) (x^(1/3)); % calcula la raiz cubica del argumento
disp(crt(cx(4.1, pi/3)))
```



inline

- Puede ser creado en la ventana de comandos, un script o una función
- No se guarda en un fichero separado
- Limitaciones
  - No puede llamar a otra función `inline`, pero puede usar un fichero función
  - Se compone sólo de una expresión
  - Sólo puede retornar una variable
  - Cualquier función que require operaciones lógicas o múltiples para producir el resultado no puede emplear inline



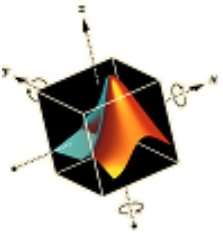
inline

- La forma general de `inline` es:

  NombreFunc = inline('expresion', 'p1', 'p2',…)

  donde:

  expresion es cualquier expresión Matlab válida convertida a cadena y p1, p2,… son los nombres de todas las variables que aparecen en expresion. Las comillas simples se requieren



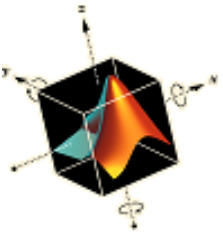
Ejemplo de inline

- Creación de función FofX para evaluar:
  $f(x) = x^2\cos(ax) - b$ donde a y b son escalares y x es un vector
  Si se escribe en la ventana de comandos

```
>> FofX = inline('x.^2.*cos(a*x)-b', 'x', 'a', 'b')
```

  Después se usar en la ventana de comandos, p.e.:

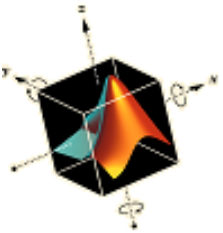
```
>> g = FofX([pi/3, pi/3.5], 4, 1)
```



Comparación de uso de Subfunciones, funciones anónimas y funciones inline

Sub functions	Anonymous functions	inline

`function` Example
dat = 1:3:52;
n = `length`(dat);
m = meen(dat, n)
s = stdev(dat, n)
`function` m = meen(v, n)
m = `sum`(v)/n;
`function` sd = stdev(v, n)
m = meen(v, n);
sd = `sqrt`((`sum`(v.^2)-
n*m^2)/(n-1));
	`function` Example
	dat = 1:3:52;
	n = `length`(dat);
	meen = @(dat, n)
	(`sum`(dat)/n);
	stdev = @(dat, n, m)
	(`sqrt`((`sum`(dat.^2)
	-n*m^2)/(n-1)));
	m = meen(dat, n)
	s = stdev(dat, n, m)
		`function` Example
		dat = 1:3:52;
		n = `length`(dat);
		meen = `inline`
		('`sum`(dat)/n ', 'dat', 'n');
		stdev = `inline`
		('`sqrt`((`sum`(dat.^2)-n*m^2)/(n-1))',
		'dat', 'n', 'm');
		m = meen(dat, n)
		s = stdev(dat, n, m)



Funciones definidas por el usuario, Function Handles y feval

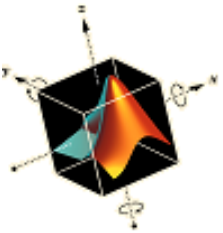
- Muchas funciones Matlab requieren que el usuario cree funciones en un formato especificado por la función Matlab
- Esas funciones usan

  `feval`(FunctionHandle, p1, p2,…,pn)

  donde

  FunctionHandle es una forma de referenciar una función para que se pueda pasar como argumentos en funciones. Un function handle se construye colocando un '@' delante del nombre de la función

  p1,p2,… son parámetros que se pasan a la función

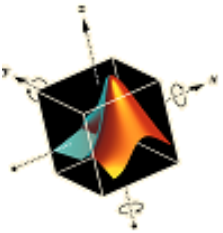


Ejemplo de feval

- Programa para calcular n raíces de una función

$$f(x) = \cos(\beta x) - \alpha \qquad \alpha < 1$$

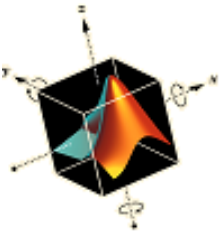
```
function d = CosBeta(x, w)    % funcion en fichero CosBeta.m
% beta = w(1);  alpha = w(2)
d = cos(x*w(1))-w(2);
function nRoots = ManyZeros(zname, n, xs, toler, dxx, [b, a])
x = xs; dx = dxx;
for m = 1: n
   s1 = sign(feval(zname, x, w));
   while dx/x >toler
      if s1 ~= sign(feval(zname, x+dx, w))
         dx = dx/2;
      else
         x = x+dx;
      end
   end
   nRoots(m) = x; dx = dxx; x = 1.05*x;
end
```



Ejemplo de feval

- Programa que usa las funciones

```
NoRoots = 5;   xStart = 0.2;
tolerance = 1e-6;   increment = 0.3;
beta = pi/3;   a = 0.5;
c = ManyZeros(@CosBeta, NoRoots, xStart, …
        tolerance, increment, beta, a)
```

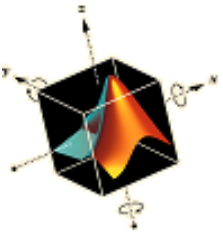


Ejemplo de feval

```
function ExampleFeval
NoRoots = 5;   xStart = 0.2;   tolerance = 1e-6;
increment = 0.3;   beta = pi/3;   a = 0.5;
c = ManyZeros(@CosBeta, NoRoots, xStart, tolerance, increment, [beta, a])
```

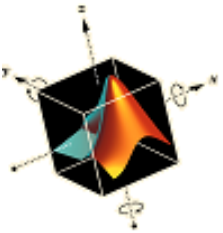
```
function nRoots = ManyZeros(zname, n, xs, toler, dxx, w)
x = xs;
dx = dxx;
...
for m = 1:n
   s1 = …
   while dx/x >toler
      if s1 ~= sign(feval(zname, x+dx, w))
      …
      end
   …
   end
   …
end
```

```
function f = CosBeta(x, w)
f = cos(w(1)*x)-w(2)
```



Algunas funciones Matlab que operan con arrays de datos

`polyfit`	Ajusta un polinomio a un array de valores
`polyval`	Evalúa un polinomio en un array de valores
`spline`	Aplica una interpolación spline cúbica a arrays de valores de coordenadas
`interp1`	Interpola entre pares de valores de coordenadas
`trapz`	Aproxima una integral a partir de un array de valores de amplitud
`polyarea`	Determina el área de un polígono
`fft/ifft`	Determina la transformada de Fourier su inversa en datos de muestra



Ajuste de datos con polinomios – polyfit

- Sea el polinomio:

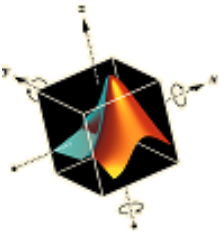
$$y(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_n x + c_{n+1}$$

los coeficientes c_k se determinan por

c = `polyfit`(x, y, n)

donde

- n es el orden del polinomio
- $c = [c_1\ c_2\ \ldots\ c_n\ c_{n+1}]$ es un vector de longitud $n+1$ representando los coeficientes del polinomio
- x e y son vectores de longitud $m \geq n + 1$



Evaluación de polinomios – polyval

- Para evaluar un polinomio a partir del vector de coeficientes c_k se usa

  y = `polyval`(c, xnew)

  donde
  - *c* es un vector de longitud *n*+1 representando los coeficientes del polinomio (determinado por polyfit)
  - *xnew* es un escalar o un vector de puntos para los cuales se evaluará el polinomio
- Se puede combinar con polyval si no interesa c_k

  y = `polyval`(`polyfit`(x,y,n), xnew)

Ejemplo de polyfit/polyval

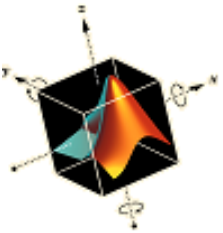
- Ajuste y evaluación de datos experimentales

```
function nd = ExpData
nd = [0.34, 0.66; 0.48, 0.46; 0.62, 0.36; 0.76, 0.29; …
      0.90, 0.23; 1.03, 0.19; 1.17, 0.14; 1.31, 0.10; …
      1.45, 0.075; 1.59, 0.050; 1.72, 0.036];
```

Script :

```
ncs = ExpData;
c = polyfit(ncs(:,1), ncs(:,2), 4);
r = input('Ingresa parametro r ');
Su = input('Ingresa valor para evaluar ');
q = 1/(1+polyval(c, Su)/sqrt(r));
disp(['Valor resultante = ' num2str(q)])
```

$$q = \left(1 + \frac{\sqrt{a}}{\sqrt{r}}\right)^{-1}$$

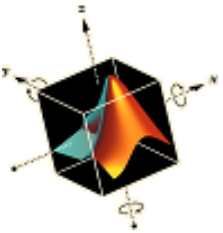


Ajuste de datos con spline

- Para generar curvas suaves que pasan (interpolan) a través de un conjunto de puntos se usa splines

  `Y = spline(x, y, X)`

  donde
  - *x* e *y* son vectores de la misma longitud que representan la relación funcional y(x)
  - *X* es un escalar o un vector de valores para los cuales se evaluará el spline Y = y(X) (en general x ≠ X)



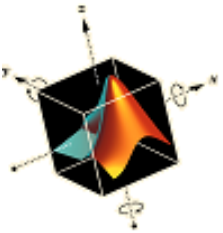
Ejemplo de spline

- Ajuste de una serie de datos, generados por una función (*DampedSineWaves*), con splines

$$f(\tau,\xi)=\frac{e^{-\xi\tau}}{\sqrt{1-\xi^2}}\sin\left(\tau\sqrt{1-\xi^2}+\varphi\right) \quad \text{donde} \quad \varphi=\tan^{-1}\frac{\sqrt{1-\xi^2}}{\xi}$$

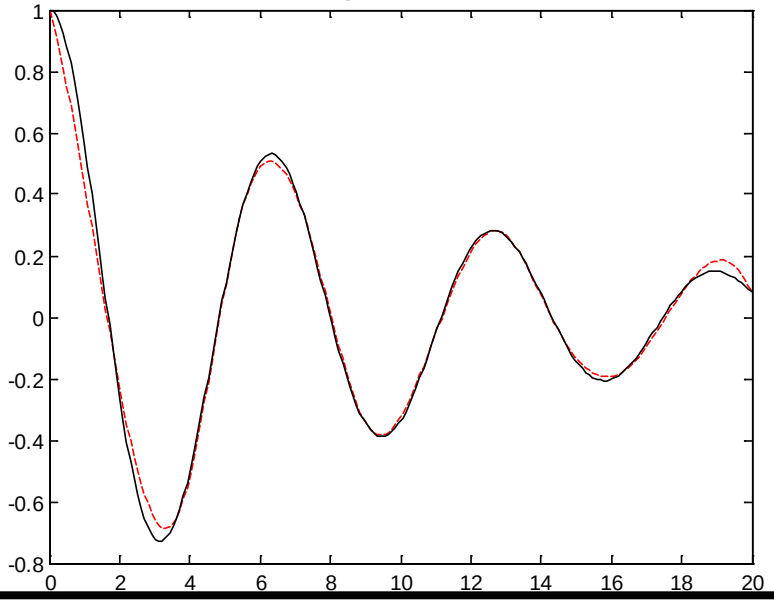
```
function f = DampedSineWave(tau, xi)
r = sqrt(1-xi^2);
phi = atan(r/xi);
f = exp(-xi*tau).*sin(tau*r+phi)/r;
```

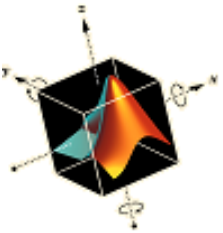
Se muestrea 12 puntos equiespaciados de $f(\tau,\xi)$ en el rango $0 \le \tau \le 20$ y se grafica el spline resultante para 200 valores de τ y $\xi = 0.1$. También se grafica la función original



Ejemplo de spline

```
n = 12;  xi = 0.1;
tau = linspace(0, 20, n);
data = DampedSineWave(tau, xi);
newtau = linspace(0, 20, 200);
yspline = spline(tau, data, newtau);
yexact = DampedSineWave(newtau, xi);
plot(newtau, yspline, 'r--', newtau, yexact, 'k-')
```



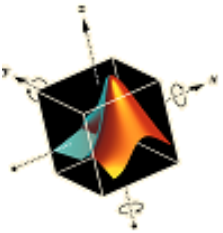


Interpolación de datos – interp1

- Para pasar un polinomio (lineal) a través de un conjunto de puntos, se usa la interpolación. La función es:

  V = `interp1`(u, v, U)

  donde el array V tendrá la misma longitud que U
  - *v* es *v(u)*
  - *u* y *v* son vectores de la misma longitud
  - *U* es un escalar o un vector de valores de *u* para los cuales se evaluará V (en general x ≠ X)

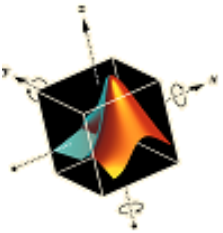


Ejemplo de interp1

- Se genera 15 puntos de $f(\tau,\xi)$ en el rango $0 \leq \tau \leq 4.5$ y $\xi = 0.1$ y se interpola para aproximar el valor de τ para $f(\tau,\xi) \approx 0$

$$f(\tau,\xi) = \frac{e^{-\xi\tau}}{\sqrt{1-\xi^2}} \sin\left(\tau\sqrt{1-\xi^2} + \varphi\right) \quad \text{donde} \quad \varphi = \tan^{-1}\frac{\sqrt{1-\xi^2}}{\xi}$$

```
xi = 0.1;
tau = linspace(0, 4.5, 15);
data = DampedSineWave(tau, xi);
TauZero = interp1(data, tau, 0)
```

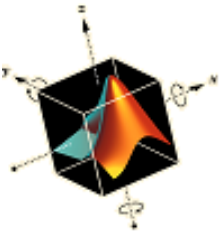


Integración numérica – trapz

- Se puede obtener una aproximación a la integral mediante trapz, cuyo formato es

  Area = `trapz`(x, y)

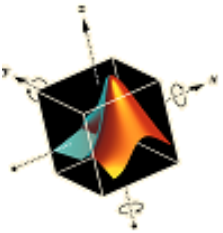
  donde
  - *x* y los valores correspondientes *y* son arrays



Ejemplo de trapz

- Area neta de $f(\tau,\xi)$ en el rango $0 \leq \tau \leq 20$ y $\xi = 0.1$ y para 200 puntos

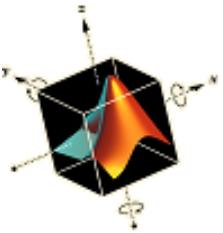
```
xi = 0.1;  N = 200;
tau = linspace(0, 20, N);
ftau = DampedSineWave(tau, xi);
Area = trapz(tau, ftau)
```



Ejemplo de trapz

- Para determinar el area de las porciones positivas y negativas de $f(\tau,\xi)$ en el rango $0 \leq \tau \leq 20$ y $\xi = 0.1$ y para 200 puntos

```
xi = 0.1;
tau = linspace(0, 20, 200);
ftau = DampedSineWave(tau, xi);
Logicp = (ftau >= 0);
PosArea = trapz(tau, ftau.*Logicp);
Logicn = (ftau < 0);
NegArea = trapz(tau, ftau.*Logicn);
disp([' Area positiva = ' num2str(PosArea)])
disp([' Area negativa = ' num2str(NegArea)])
disp([[' Area neta = ' num2str(PosArea+NegArea)]])
```



Diferencias y derivadas aproximadas – diff

- Para calcular diferencias entre elementos sucesivos de un vector se usa diff, así para un vector

$$x = [x_1\ x_2\ \ldots\ x_n]$$

diff crea un vector *q* con *n-1* elementos de la forma

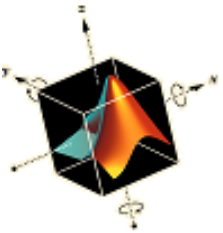
$$q = [x_2 - x_1,\ x_3 - x_2,\ \ldots\ ,\ x_n - x_{n-1}]$$

para un vector x, diff equivale a:

```
q = x(2:end)-x(1:end-1);
```

La derivada aproximada de una función y = *f*(*x*) es:

```
der = diff(y)./diff(x)
```



Ejemplo de diff

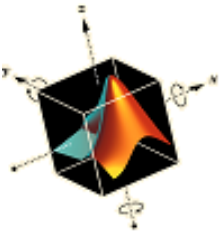
- La longitud de una línea en el espacio se puede aproximar por

$$L = \int_a^b \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2}\, dt \approx \sum_{i=1}^{N} \sqrt{(\Delta_i x)^2 + (\Delta_i y)^2 + (\Delta_i z)^2}$$

donde:

$$\Delta_i x = x(t_{i+1}) - x(t_i)$$
$$\Delta_i y = y(t_{i+1}) - y(t_i)$$
$$\Delta_i z = z(t_{i+1}) - z(t_i)$$

y $t_1 = a$ y $t_{N+1} = b$



Ejemplo de diff

- Si $x = 2t$ para $1 \leq t \leq 2$ y se asume que $N = 25$.

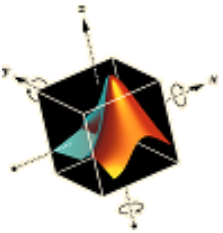
  $y = t^2$

  $z = \ln t$

  las cantidades $\Delta_i x$, $\Delta_i y$, y $\Delta_i z$, se puede evaluar con `diff`

  El script que calcula la longitud es:

```
t = linspace(1, 2, 25);
L = sum(sqrt(diff(2*t).^2+diff(t.^2).^2+diff(log(t)).^2))
```

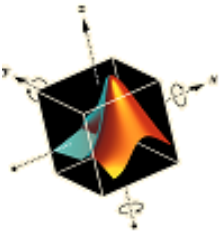


Area de un polígono – polyarea

- Se puede obtener el area de un polígono de n lados, donde cada lado del polígono es representado por sus vértices, con el uso de

  `polyarea`(x, y)

  donde
  - *x e y* son vectores de la misma longitud que contienen las coordenadas (x, y) de los vértices del polígono
  - cada par de coordenadas (x, y) es el punto final de dos lados adyacentes del polígono; por tanto, para un plígono de n lados, es necesario n +1 pares de vértices



Ejemplo de polyarea

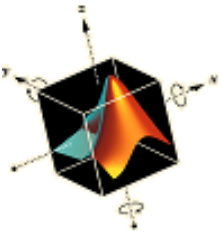
- Para calcular el área de un polígono cuyos vértices caen en una elipse especificada por las ecuaciones paramétricas

$$x = a\cos\theta$$

$$y = b\sin\theta$$

Si a = 2, b = 5 y se considera 10 lados, el script es:

```
a = 2;   b = 5;
t = linspace(0, 2*pi, 11);
x = a*cos(t);
y = b*sin(t);
disp(['Area = 'num2str(polyarea(x, y))])
```



Procesado digital de señal – fft y ifft

- La transformada de Fourier de una función real $g(t)$ que es muestreada cada Δt en el intervalo $0 \le t \le T$ se puede aproximar por su transformada discreta de Fourier:

$$G_n = G(n\Delta f) = \Delta t \sum_{k=0}^{N-1} g_k e^{-j2\pi nk/N} \qquad n = 0,1,\ldots, N-1$$

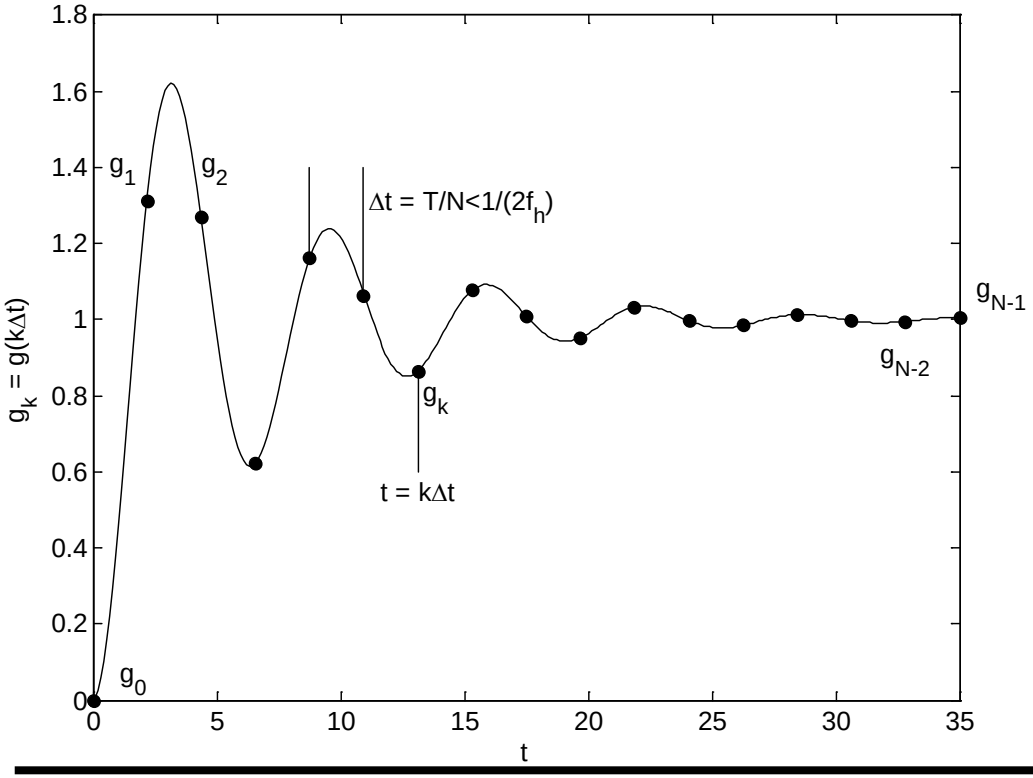
- Para estimar la magnitud de la amplitud A_n correspondiente a cada G_n y su correspondiente $n\Delta f$, se multiplica G_n por Δf: $A_n = \Delta f G_n$. Por tanto:

$$A_n = \frac{1}{N} \sum_{k=0}^{N-1} g_k e^{-j2\pi nk/N} \qquad n = 0,1,\ldots, N-1$$

Procesado digital de señal – fft y ifft

- Es usual graficar $|A_n|$ como una función de $n\Delta f$ para obtener el gráfico de la amplitud espectral:

$$\left|A_n\right|_s = 2\left|A_n\right| \qquad n = 0,1,\ldots,N/2-1$$



$g_k = g(k\Delta t)$

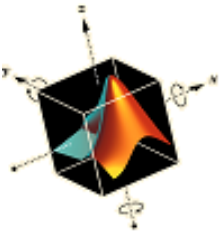
$\Delta f = 1/T$

$N = 2^m$ (m = entero positivo)

$T = N\Delta t$

$\Delta t = 1/(\alpha f_h)$ ($\alpha > 2$)

f_h = mayor frecuencia en $g(t)$



Procesado digital de señal – fft y ifft

- La transformada inversa de Fourier se aproxima por:

$$g_k = \Delta f \sum_{n=0}^{N-1} G_n e^{j2\pi nk/N} \qquad k = 0,1,\ldots,N-1$$

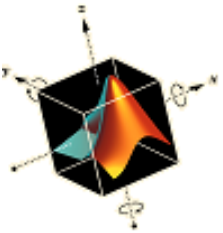
- Toda la formulación anterior se calcula mejor con la transformada rápida de Fourier (FFT) que es más efectiva cuando $N = 2^m$, donde m es un entero positivo
- En Matlab el algoritmo FFT se implementa con

  `G = fft(g, N)`

  y la inversa con

  `g = ifft(G, N)`

  donde $G = G_n/\Delta t$ y $g = g_k/\Delta f$



Ejemplo de transformada de Fourier

- Se requiere calcular la gráfica de la amplitud espectral de la sgte. onda senoidal muestreda de duración *T*:

$$g(t) = B_0 \sin(2\pi f_0 t) + B_0 / 2 \sin(2\pi 2 f_0 t)$$

$$0 \le t \le T = 2^K / f_0 = 2^K / f_K \quad K = 0,1,2,\ldots$$

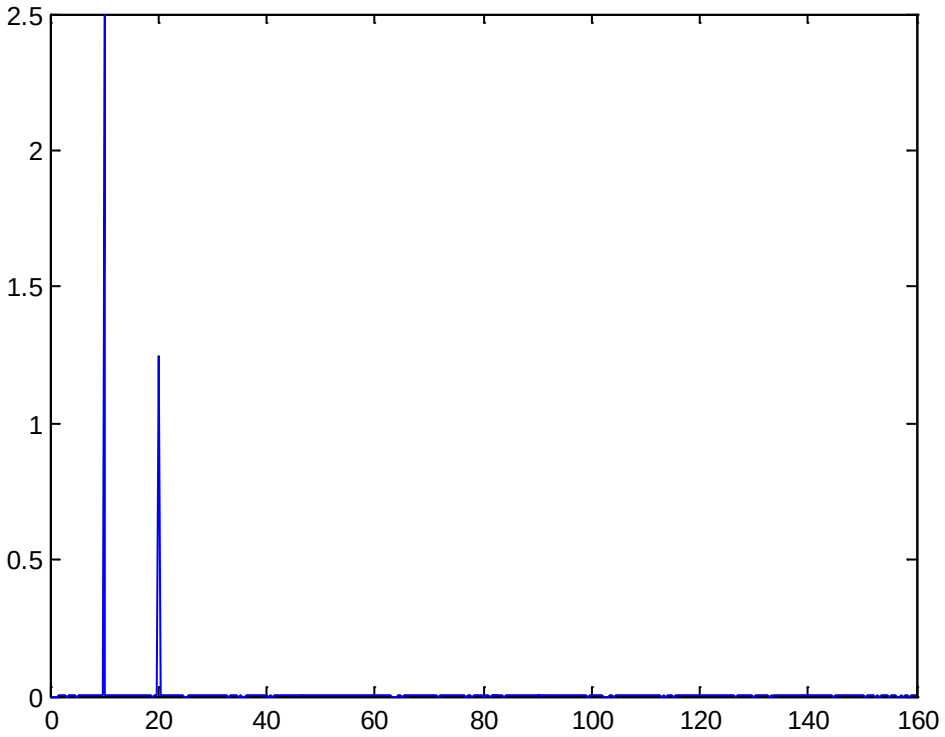
$$\Delta t = 2^{-m} T \qquad y \quad m - K > 1$$

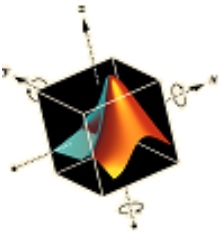
donde:

B_o = 2.5, f_o = 10 Hz, K = 5, and m = 10 (N = 1024).

Ejemplo de transformada de Fourier

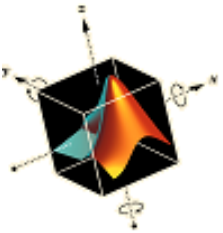
```
k = 5;   m = 10;   fo = 10;   Bo = 2.5; N = 2^m;   T = 2^k/fo;
ts = (0:N-1)*T/N; df = (0:N/2-1)/T;
SampledSignal = Bo*sin(2*pi*fo*ts)+Bo/2*sin(2*pi*fo*2*ts);
An = abs(fft(SampledSignal, N))/N;
plot(df, 2*An(1:N/2))
```





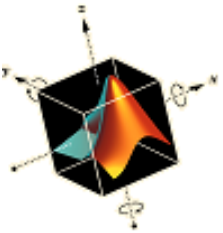
Algunas funciones Matlab que requieren funciones creadas por el usuario

`fzero`	Halla una raíz de $f(x) = 0$.
`roots`	Calcula las raíces de un polinomio
`quadl`	Integra numéricamente $f(x)$ en un intervalo
`dblquad`	Integra numéricamente $f(x,y)$ es una región específica
`ode45`	Resuelve un sistema de ec. diferenciales ordinarias con condiciones iniciales determinadas
`bvp4c`	Resuelve un sistema de ec. diferenciales ordinarias con condiciones de contorno determinadas
`dde23`	Resuelve ec. diferenciales con retardo con desfases constantes y condiciones iniciales determinadas
`pdepe`	Solución numérica de ecuaciones en derivada parcial 1d parabólica-elíptica
`fminbnd`	Halla el mínimo local de $f(x)$ en un intervalo específico
`fsolve`	Resuelve numéricamente un sistema de ecuaciones nolineales (Optimization toolbox).



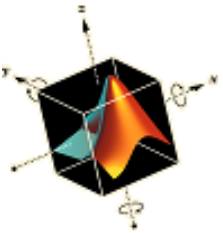
Raíz de Funciones – fzero

- Hay ecuaciones en las que no se puede establecer una solución explícita algebraica de $f(x) = 0$
- Para ello se usa `fzero` para hallar numéricamente una solución de la función real $f(x) = 0$ dentro de una tolerancia t_o en el entorno de x_o o dentro del rango de $[x_1, x_2]$
- La función puede transferir también parámetros p_j, $j = 1, 2, \ldots$, a la función que define $f(x)$



Raíz de Funciones – fzero

- La expresión general es:
  z = `fzero`(@FunctionName, x0, options, p1, p2,…)
  donde:
  z es el valor de x para el cual $f(x) \approx 0$
  FunctionName es el nombre del fichero función sin la extensión ".m" o el nombre de la subfunción
  x0 = x_o o $x_0 = [x_1\ x_2]$
  p1, p2,…, son los parámetros p_j requeridos por *FunctionName*
  options se configuran con la función `optimset` (función para el ajuste de parámetros)



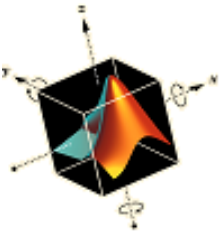
Raíz de Funciones – fzero

- La interfaz de la función requerida por `fzero` tiene la forma:

```
function z = FunctionName(x, p1, p2,...)
expresion(es)
z = ...
```

donde:

x es la variable independiente que `fzero` modifica para encontrar un valor tal que $f(x) \approx 0$. La variable independiente siempre debe aparecer en la primera posición



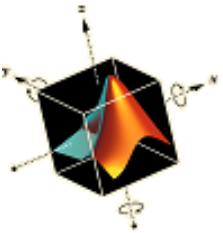
Raíz de Funciones – fzero

- La función `fzero` también se puede usar con función `inline` y función anónima:

```
InlineFunctName = inline('Expresion', 'x', 'p1','p2', ...);
z = fzero(InlineFunctName, x0, options, p1, p2,...)

fhandle = @(x, p1, p2, ...) (Expresion);
z = fzero(fhandle, x0, options, p1, p2,...)
```

hay funciones Matlab que no se pueden usar directamente, p.e. la función Bessel `besselj`(n, x) (la variable independiente no está en el primer lugar)

```
besseljx = inline('besselj(n, x)', 'x', 'n');
a = fzero(besseljx, 3, [ ], 1)  % J1(x) = 0 en 3
```



Raíces de Funciones – roots/poly

- Si $f(x)$ es un polinomio de la forma

$$f(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_n x + c_{n+1}$$

las raíces se pueden obtener facilmente usando

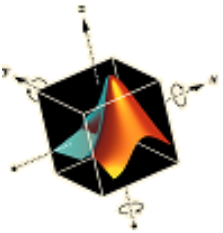
```
r = roots(c)
```

donde $c = [c_1, c_2, \ldots, c_{n+1}]$

- La inversa de roots es

```
c = poly(r)
```

que devuelve *c*, los coeficientes del polinomio y *r* es el vector de raíces, que en general, *r* es un vector de números reales y/o complejos



Raíces de Funciones – roots/poly

- Se pide calcular las raíces del polinomio:

$$f(x) = x^4 - 35x^2 + 50x + 24$$

script:

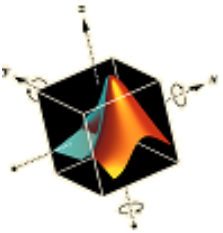
```
r = roots([1, 0, -35, 50, 24])
```

para obtener las raíces ordenadas

```
r = sort(roots([1, 0, -35, 50, 24]))
```

Para determinar los coeficientes de un polinomio con esas raíces, el script es:

```
r = roots([1, 0, -35, 50, 24]);
c = poly(r)
```



Integración numérica – quadl

- La función `quadl` integra numéricamente una función *f*(*x*) entre un límite inferior *a* hasta un límite superior *b* dentro de una tolerancia t_o.
- La expresión general de `quadl` es

  A = `quadl`(@FunctionName, a, b, t0, tc, p1, p2,…)

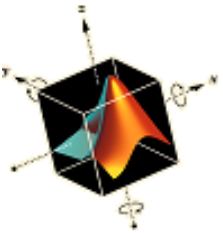
  donde

  FunctionName es el nombre de una función o subfunción

  a = *a* límite inferior b = *b* límite superior

  t0 = t_o (si se omite,se usa valor por defecto)

  p1, p2, ..., son parámetros p_j, si tc ≠ [ ] salida intermedia



Integración numérica – quadl

- La interfaz de la función tiene la forma

  `function` z = FunctionName(x, p1, p2, ...)

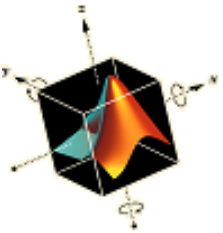
  expresion(es)

  donde *x* es la var. independiente de integración
- La expresión general de `quadl` cuando se usa función `inline` o función anónima es

  A = `quadl`(IorAFunctionName, a, b, t0, tc, p1, p2,…)
- La interfaz `inline` es

  IorAFunctionName = `inline`('Expresion', 'x', 'p1', ...)
- La interfaz para una función anónima es

  IorAFunctionName = @(x, p1, p2, …) (Expresion)

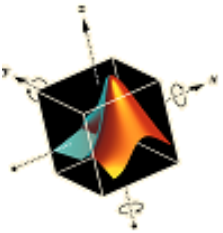


Ejemplo de integración numérica – quadl

- Se requiere obtener la integral de función *DampedSineWaves* con $\xi = 0.1$ y $0 \leq \tau \leq 20$

$$f(\tau,\xi) = \frac{e^{-\xi\tau}}{\sqrt{1-\xi^2}} \sin\left(\tau\sqrt{1-\xi^2} + \varphi\right) \quad \text{donde} \quad \varphi = \tan^{-1}\frac{\sqrt{1-\xi^2}}{\xi}$$

```
xi = 0.1;
Area = quadl(@DampedSineWave, 0, 20, [], [], xi)
```



Integración numérica – dblquad

- La función `dblquad` integra numéricamente una función $f(x,y)$ entre un los límites inferior y superior x_l, x_u en la dirección x y los límites y_l, y_u en la dirección y dentro de una tolerancia t_o.
- La expresión general de `dblquad` es

`dblquad`(@FunctionName, xl, xu, yl, yu, t0, meth, p1, p2,…)

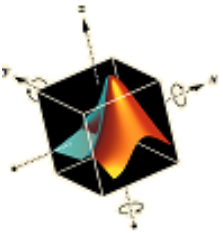
donde:

FunctionName es el nombre de una función o subfunción

xl = x_l xu = x_u yl = y_l yu = y_u

t0 = t_o (si se omite,se usa valor por defecto)

p1, p2, ..., son parámetros p_j, si meth = [ ] se usa quadl



Integración numérica – dblquad

- La interfaz de la función tiene la forma

  `function` z = FunctionName(x, p1, p2, ...)

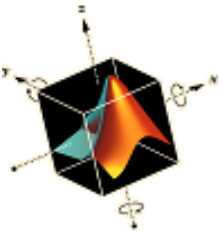
  expresion(es)

  donde *x* es la var. independiente de integración
- La expresión general de `dblquadl` cuando se usa función `inline` o función anónima es

  A = `quadl`(InoAnFN, xl, xu, yl, yu,t0,meth, p1, p2,…)
- La interfaz `inline` es

  InoAnFN = `inline`('Expresion', 'x', 'p1', ...)
- La interfaz para una función anónima es

  InoAnFN = @(x, p1, p2, …) (Expresion)

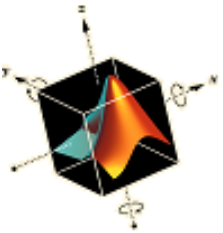


Ejemplo de integración numérica – dblquad

- Se requiere obtener la integral de la función de variables random con distribución normal, considerando $r = 0.5$

$$V = \frac{1}{2\pi\sqrt{1-r^2}} \int_{-2}^{2} \int_{-3}^{3} e^{-(x^2-2rxy+y^2)/2} dxdy$$

```
r = 0.5;
Arg = @(x, y) (exp(-(x.^2-2*r*x.*y+y.^2)));
V = dblquad(Arg, -3, 3, -2, 2, [], [], r)/2/pi/sqrt(1-r^2)
```

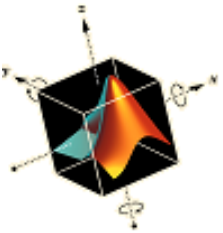


Solución numérica de ecuaciones diferenciales ordinarias - Preliminares

- A partir del análisis de la dinámica de sistemas debemos ser capaces de establecer un sistema de ecuaciones diferenciales como función de variables dependientes w_k , $k = 1, \ldots, K$
- La conversión de éstas a un conjunto de ecuaciones diferenciales de primer orden de la forma

$$\frac{dy_j(\eta)}{d\eta} = g_j\left(y_j, \eta\right) \qquad j = 1, 2, \ldots N$$

y convertir las condiciones iniciales o condiciones de contorno a funciones de y_j



Solución numérica de ecuaciones diferenciales ordinarias - Preliminares

- Ejemplo: $$\frac{d^3 f}{d\eta^3} + 3f\frac{d^2 f}{d\eta^2} - 2\left(\frac{df}{d\eta}\right)^2 + T^* = 0$$

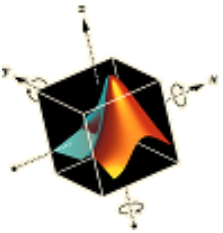
$$\frac{d^{2*}T^*}{d\eta^2} + 3\mathrm{Pr}\, f\frac{dT^*}{d\eta} = 0$$

las condiciones de contorno son:

$$\eta = 0: \qquad f = 0, \qquad \frac{df}{d\eta} = 0, \quad \text{y} \quad T^* = 1$$

$$\eta = 8: \quad \frac{df}{d\eta} = 0 \quad \text{y} \quad T^* = 0$$

  - El número de ecuaciones de primer orden y el número de condiciones de iniciales que se obtendrá es igual a la suma de cada variable dependiente (3 + 2 = 5 para el ejemplo)



Solución numérica de ecuaciones diferenciales ordinarias - Preliminares

- Si hacemos

$$y_1 = f \qquad y_4 = T^*$$

$$y_2 = \frac{df}{d\eta} \qquad y_5 = \frac{dT^*}{d\eta}$$

$$y_3 = \frac{d^2 f}{d\eta^2}$$

entonces

$$\frac{dy_1}{d\eta} = y_2 \qquad \frac{dy_4}{d\eta} = y_5$$

$$\frac{dy_2}{d\eta} = y_3 \qquad \frac{dy_5}{d\eta} = -3\,\mathrm{Pr}\, y_1 y_5$$

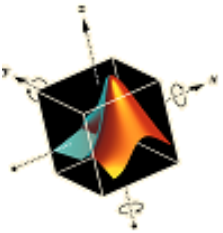
$$\frac{dy_3}{d\eta} = 2y_2^2 - 3y_1 y_3 - y_4$$

obtenido de:

$$\frac{d^3 f}{d\eta^3} + 3f\frac{d^2 f}{d\eta^2} - 2\left(\frac{df}{d\eta}\right)^2 + T^* = 0 \qquad \frac{d^2 T^*}{d\eta^2} + 3\mathrm{Pr}\, f\frac{dT^*}{d\eta} = 0$$

$$\frac{d}{d\eta}\left(\frac{d^2 f}{d\eta^2}\right) + +3f\frac{d^2 f}{d\eta^2} - 2\left(\frac{df}{d\eta}\right)^2 + T^* = 0 \qquad \frac{d}{d\eta}\left(\frac{dT^*}{d\eta}\right) + 3\mathrm{Pr}\, f\frac{dT^*}{d\eta} = 0$$

$$\frac{dy_3}{d\eta} + 3y_1 y_3 - 2y_2^2 + y_4 = 0 \qquad \frac{dy_5}{d\eta} + 3\mathrm{Pr}\, y_1 y_5 = 0$$



Solución numérica de ecuaciones diferenciales ordinarias - Preliminares

- Las condiciones de contorno son

para $\eta = 0$

$$f = 0 \rightarrow y_1 = 0$$

$$\frac{df}{d\eta} = 0 \rightarrow y_2 = 0$$

$$T^* = 1 \rightarrow y_4 = 1$$

para $\eta = 8$:

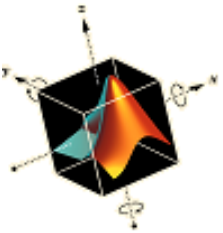
$$\frac{df}{d\eta} = 0 \rightarrow y_2 = 0$$

$$T^* = 0 \rightarrow y_4 = 0$$

$$y_1 = f \qquad y_4 = T^*$$

$$y_2 = \frac{df}{d\eta} \qquad y_5 = \frac{dT^*}{d\eta}$$

$$y_3 = \frac{d^2 f}{d\eta^2}$$

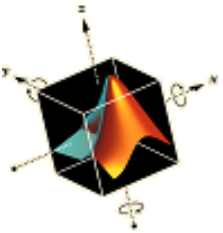


Solución numérica de ecuaciones diferenciales ordinarias – ode45

- La función `ode45` devuelve la solución numérica a un sistema de *n* ecuaciones diferenciales de primer orden

$$\frac{dy_j}{dt} = f_j(t, y_1, y_2, \ldots, y_n) \qquad j = 1, 2, \ldots, n$$

sobre el intervalo $t_0 \leq t \leq t_f$ sujeto a las condiciones iniciales $y_j(t_0) = a_j$, $j = 1, 2, \ldots, n$, donde a_j son constantes



Solución numérica de ecuaciones diferenciales ordinarias – ode45

- Los argumentos y resultados de `ode45` son

[t, y] = `ode45`(@FunctionName, [t0, tf],
[a1, a2, …, an], options, p1, p2, …)

donde:

t es un vector columna de valores $t_0 \leq t \leq t_f$ que son determinadas por ode45

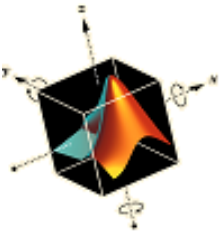
y es la matriz de soluciones tal que las filas corresponden a los valores *t* y las columnas corresponden a las soluciones

$y(:, 1) = y_1(t)$

$y(:, 2) = y_2(t)$

…

$y(:, n) = y_n(t)$



Solución numérica de ecuaciones diferenciales ordinarias – ode45

- El primer argumento de `ode45` es @FunctionName, que es un handle a la función o subfunción *FunctionName.* Su forma debe ser

  `function` yprime = FunctionName(t, y, p1, p2,…)

  donde:

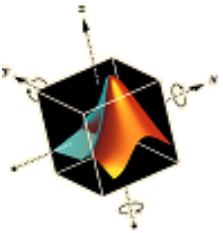
  t es la variable independiente

  y es un vector columna cuyos elementos corresponden a y_i

  *p*1, *p*2, … son parámetros pasados a *FunctionName*

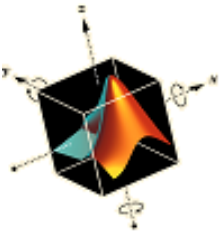
  yprime es un vector columna de longitud *n* cuyos elementos son

$$f_j(t, y_1, y_2, ..., y_n) \qquad j = 1, 2, ..., n$$



Solución numérica de ecuaciones diferenciales ordinarias – ode45

- El segundo argumento de `ode45` es un vector de dos elementos que el valor inicial y final sobre los cuales se quiere obtener la solución numérica
- El tercer argumento es un vector de condiciones iniciales $y_j(t_o) = a_j$
- El cuarto argumento,*options*,normalmente es nulo ([ ]). Si se tiene que cambiar alguna tolerancia del método de solución se usa `odeset`
- Los argumentos restantes son los que se pasan a *FunctionName*



Ejemplo de solución numérica de EDO

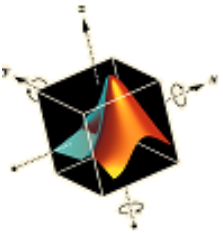
- Se tiene la ec. diferencial ordinaria de segundo orden

$$\frac{d^2y}{dt^2} + 2\xi\frac{dy}{dt} + y = h(t)$$

sujeta a las cond. iniciales $y(0) = a$ y $dy(0)/dt = b$
Reescribiendo la ecuación como un sistema de dos ecuaciones de primer orden mediante sustitución

$$y_1 = y$$

$$y_2 = \frac{dy}{dt}$$



Ejemplo de solución numérica de EDO

El sistema de ecuaciones de primer orden son

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -2\xi y_2 - y_1 + h$$

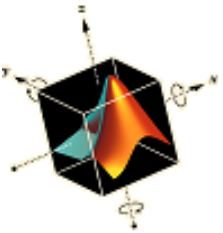
sujeta a las cond. iniciales $y_1(0) = a$ y $y_2(0) = b$

Considerando el caso donde $\xi = 0.15$, $y(0) = 1$, $dy(0)/dt = 0$

$$h(t) = \sin(\pi t/5) \qquad 0 \le t \le 5$$

$$= 0 \qquad t > 5$$

y la región de interés de la solución $0 \le t \le 35$



Ejemplo de solución numérica de EDO

Entonces, $y1(0) = 1$, $y2(0) = 0$, $t0 = 0$, y $tf = 35$

Para resolver el sistema se crea una función llamada *Ejemploode* y una subfunción llamada *HalfSine*

```
function Ejemploode
[t, yy] = ode45(@HalfSine, [0 35], [1 0], [ ], 0.15);
plot(t, yy(:,1))
```

$y_j(t_0) = a_j$

[t, y] = ode45(@FunctionName, [t0, tf], [a1, a2, …, an], options, p1, p2, …)

```
function y = HalfSine(t, y, z)
h = sin(pi*t/5).*(t<=5);
y = [y(2); -2*z*y(2)-y(1)+h];
```

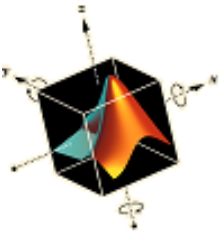
$$h(t) = \sin(\pi t/5) \quad 0 \le t \le 5$$
$$= 0 \quad t > 5$$

donde: $yy(:,1) = y_1(t) = y(t)$

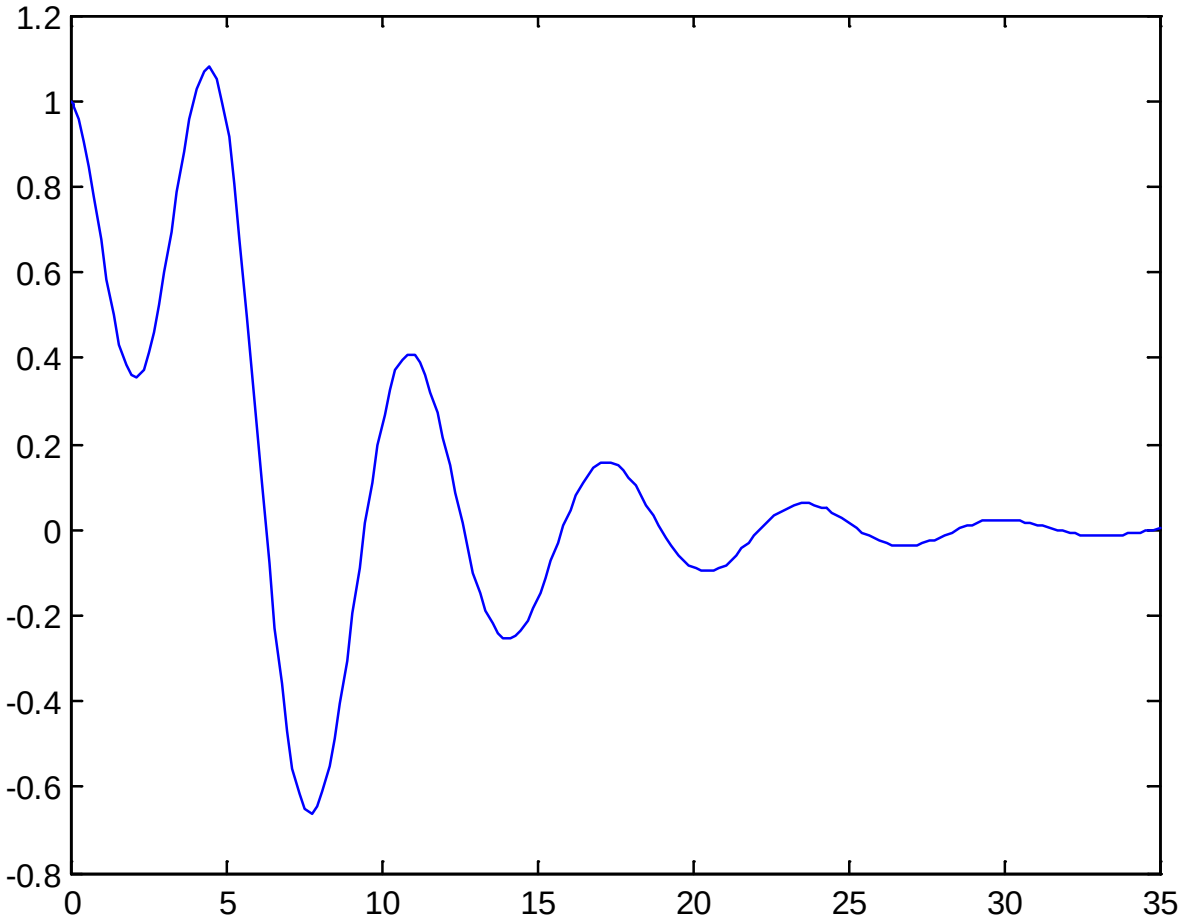
$yy(:,2) = y_2(t) = dy/dt$

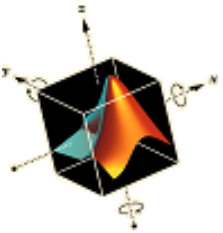
$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -2\xi y_2 - y_1 + h$$



Ejemplo de solución numérica de EDO





Solución numérica de ecuaciones diferenciales ordinarias – bvp4c

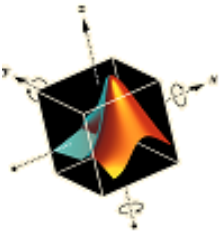
- La función `bvp4c` obtiene la solución numérica a un sistema de *n* ecuaciones diferenciales ordinarias

$$\frac{dy_j}{dx} = f_j(x, y_1, y_2, ..., y_n, q) \qquad j = 1, 2, ..., n$$

en el intervalo $a \leq x \leq b$ sujeto a las condiciones de contorno $y_j(a) = a_j$ and $y_j(b) = b_j$ j = 1, 2, …, $n/2$, donde a_j y b_j son constantes y q es un vector de parámetros desconocidos

`bvp4c` requiere el uso de varias funciones Matlab:

- `bvpinit` – función de inicialización
- `deval` – función de suavizado para gráfica



Solución numérica de ecuaciones diferenciales ordinarias – bvp4c

- Los argumentos y resultados de `bvp4c` son

  sol = `bvp4c`(@FunctionName, @BCFunction, …
  solinit, options, p1, p2...)

  donde sol es una estructura con los valores:

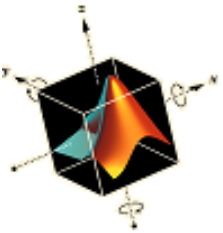
  sol.x Malla seleccioanda por `bvp4c`

  sol.y Aproximación a *y*(*x*) en la malla de puntos de sol.x

  sol.yp Aproximación a *dy*(*x*)/*dx* en la malla de puntos de sol.x

  sol.parametersValues Retornado por `bvp4c` para los parámetros desconocidos, si hay

  sol.solver 'bvp4c'



Solución numérica de ecuaciones diferenciales ordinarias – bvp4c

- La interfaz de BC*Function* contiene las condiciones de contorno $y_j(a) = a_j$ y $y_j(b) = b_j$ $j = 1, 2, \ldots$ y requiere la interfaz:

```
function Res = BCFunction (ya, yb, p1, p2,…)
```

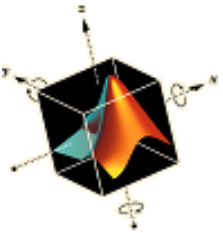
donde:

ya es un vector columna de $y_j(a)$

yb es un vector columna de $y_j(b)$

*p*1, *p*2, … son parámetros que deben aparecer aunque las cond. de contorno no los requieran

Res es un vector columna de resultado



Solución numérica de ecuaciones diferenciales ordinarias – bvp4c

- La variable *solinit* es una estructura obtenida por la función `bvpinit`

  solinit = `bvpinit`(x, y)

  donde:

  x es un vector de valores iniciales para los puntos de la malla

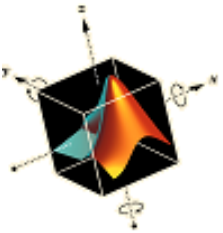
  y es un vector de valores iniciales para cada y_j

  la longitud de *x* e *y* son idependientes

  La estructura de *solinit* es:

  x nodos ordenados de la malla inicial

  y valor inicial para la solución con solinit.y(:,i) como valor de la solución en el nodo solinit.x(i)



Solución numérica de ecuaciones diferenciales ordinarias – bvp4c

- La salida de `bvp4c`, *sol*, es una estructura con la solución en un número puntos. Para obtener una curva suave en puntos intermedios se usa:

  sxint = `deval`(sol, xint)

  donde:

  xint es un vector de puntos donde se evalua la solución

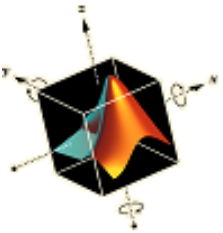
  sol es la salida (estructura) de `bvp4c`

  La salida *sxinit* es un array que contiene los valores de y_j para los valores *xint* :

  $y_1(xint) = sxint(1,:)$

  $y_2(xint) = sxint(2,:) \ldots y_n(xint) = sxint(n,:)$

sxint = `deval`(sol, xint)



Ejemplo de solución numérica de EDO con bvp4c

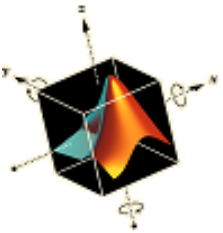
- Se tiene la ec. diferencial ordinaria de segundo orden

$$\frac{d^2y}{dx^2} + kxy = 0 \qquad 0 \le x \le 1$$

sujeta a las cond. iniciales $y(0) = 0.1$ y $y(1) = 0.05$
Transformando la ecuación en un par de ecuaciones diferenciales de primer orden mediante sustitución

$$y_1 = y$$

$$y_2 = \frac{dy}{dt}$$



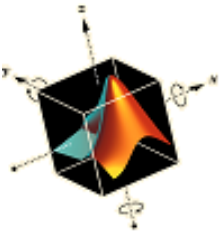
Ejemplo de solución numérica de EDO con bvp4c

El sistema de ecuaciones de primer orden es

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = kxy_1$$

con las cond. de contorno $y_1(0) = 0.1$ y $y_2(1) = 0.05$
Se seleccionan cinco puntos entre 0 y 1 y se asume que la solución para y_1 tiene una magnitud constante de –0.05 y que para y_2 tiene una magnitud constante de 0.1. Adicionalmente se asume que $k = 100$



Ejemplo de solución numérica de EDO con bvp4c

sol = bvp4c(@FunctionName, @BCFunction, solinit, options, p1, p2...)

```
function bvpEjemplo
k = 100;
solinit =  bvpinit(linspace(0, 1, 5), [-0.05, 0.1]);
exmpsol = bvp4c(@OdeBvp, @OdeBC, solinit, [], k);
x =  linspace(0, 1, 50);
y = deval(exmpsol, x);
plot(x, y(1,:))
```

solinit = bvpinit(x, y)

sxint = deval(sol, xint)

$$\frac{dy_1}{dx} = y_2$$

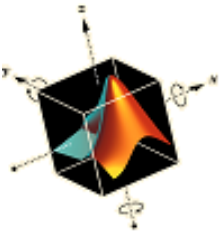
$$\frac{dy_2}{dx} = kxy_1$$

```
function dydx = FunctionName(x, y, p1, p2, ...)
function dydx = OdeBvp(x, y, k)
dydx = [y(2); k*x*y(1)];
```

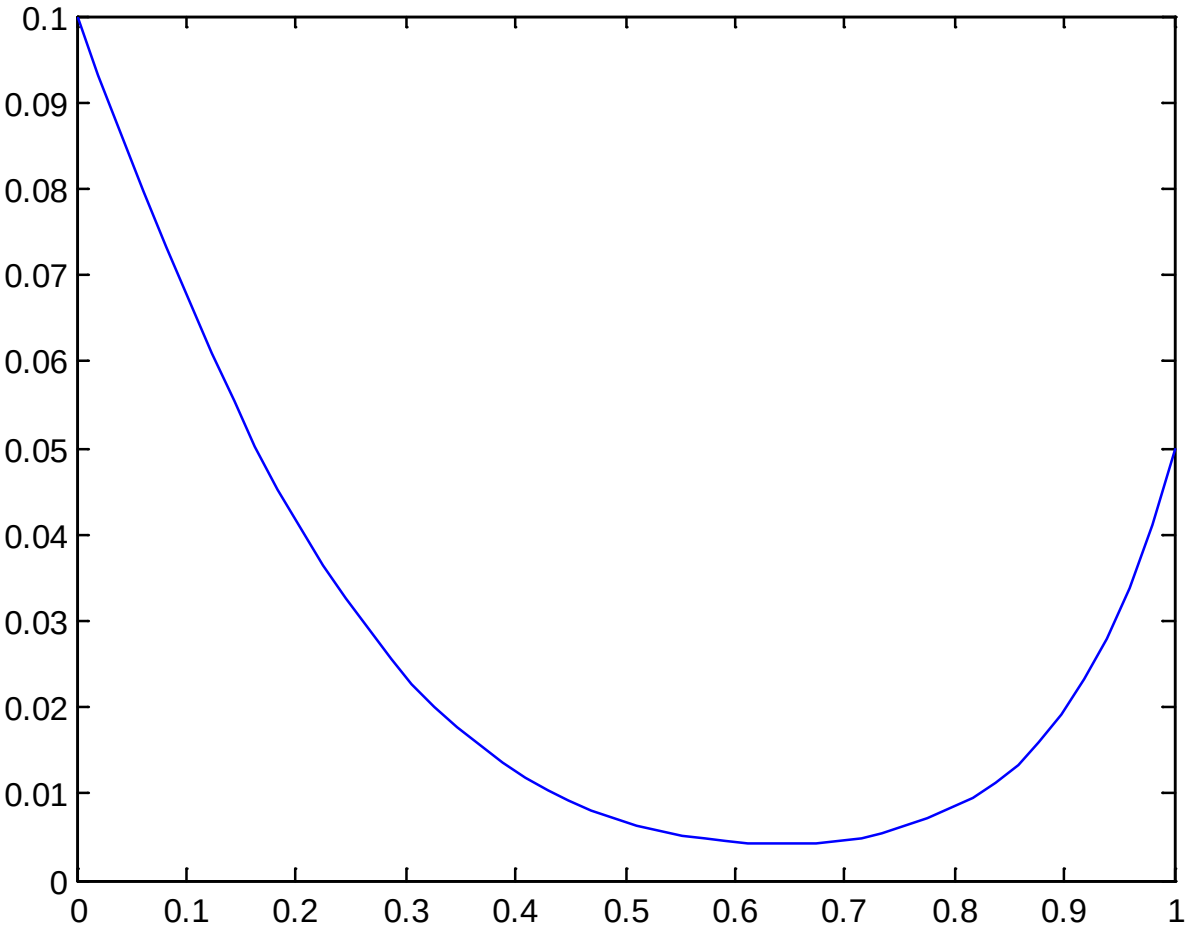
```
function Res = BCFunction(ya, yb , p1, p2, ...)
function res = OdeBC(ya, yb, k)
res = [ya(1)-0.1; yb(1)-0.05];
```

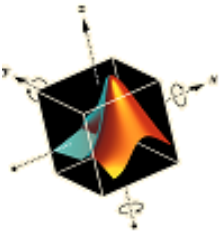
$$y_1(0) = 0.1$$

$$y_1(1) = 0.05$$



Ejemplo de solución numérica de EDO





Solución numérica de ED en derivadas parciales 1d parabólica-elíptica - pdepe

- La función `pdepe` obtiene la solución numérica de una ecuación en derivada parciales

$$c\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\right) + s$$

donde $u = u(x,t)$, $a \le x \le b$, $t_o \le t \le t_{end}$

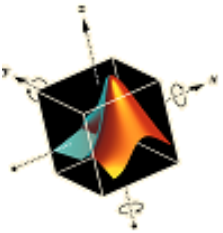
$$c = c(x,t,u,\partial u/\partial x), \quad f = f(x,t,u,\partial u/\partial x),$$

$$s = s(x,t,u,\partial u/\partial x)$$

m = 0 indica sistema Cartesiano, m = 1 sistema cilíndrico, y m = 2 un sistema esférico

Referencia en Matlab:

http://www.mathworks.com/help/techdoc/ref/pdepe.html

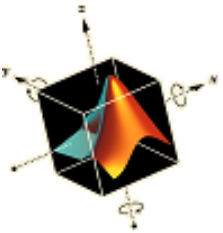


Mínimo local de una función – fminbnd

- La función `fminbnd` encuentra un mínimo local de una función real $f(x)$ en el intervalo $a \leq x \leq b$ dentro de una tolerancia t_o.
- La expresión general de `fminbnd` es

  [xmin fmin] = `fminbnd`(@FunctionName, a, b, …
  options, p1, p2...)

  donde *FunctionName* es es el nombre de la función o subfunción. a = *a*, b = *b*. *options* es un vector opcional cuyos parámetros se configuran con `optimset`. p1, etc., son los parámetros p_j



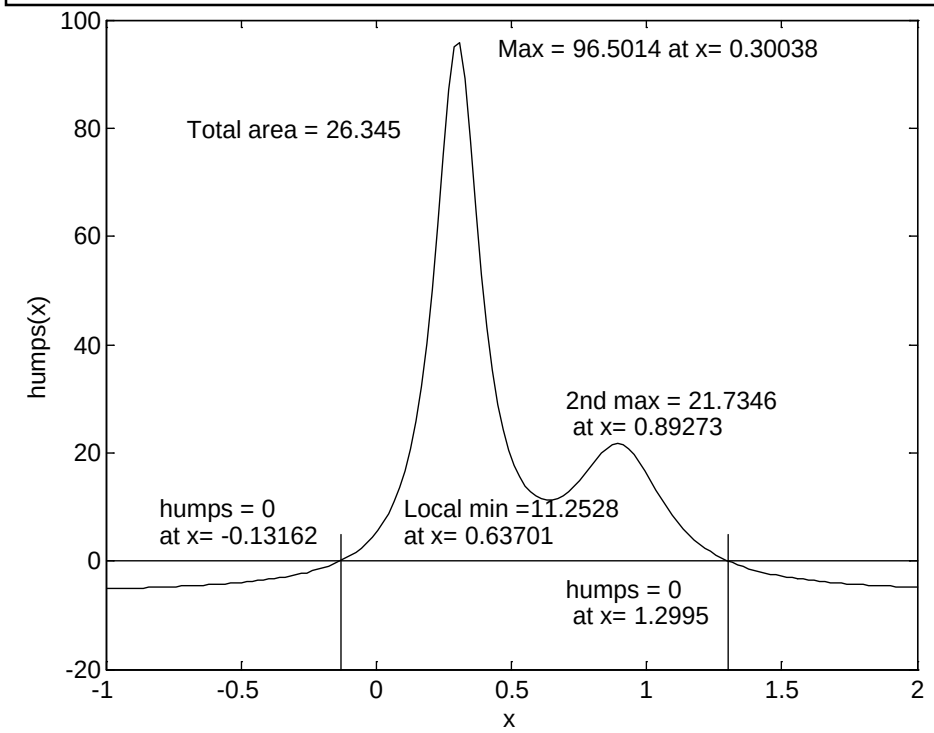
Mínimo local de una función – fminbnd

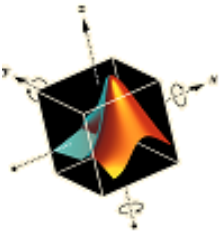
- *xmin* es el valor de x que minimiza la función especificada por FunctionName y *fmin* es el valor de FunctionName en *xmin*
- La interfaz *FunctionName* tiene la forma:
  `function` z = FunctionName(x, p1, p2,...)
  donde *x* es la variable independiente que `fminbnd` varía para minimizar *f*(*x*)

Ejemplo de mínimo local de una función – fminbnd

- Se requiere calcular el mínimo de la función humps en el intervalo $0.5 \leq x \leq 0.8$

$$\mathbf{humps}(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$$





Ejemplo de mínimo local de una función – fminbnd

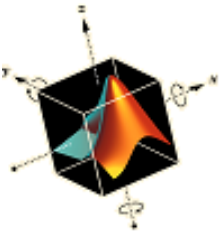
- Se obtiene con

  [xmin, fmin] = `fminbnd`(@humps, 0.5, 0.8)

  Para obtener el valor *máximo* de humps en el intervalo $0 \leq x \leq 0.5$ y dónde ocurre se opera sobre el negativo o el recíproco de la función `humps`

  El script, usando inline, es:

```
[xmax, fmax] = fminbnd(inline('-humps(x)', 'x '), 0, 0.5);
disp([' Valor maximo de humps en el intervalo 0 <= x
                    <= 0.5 is ' num2str(-fmax)])
disp([' que ocurre en x = ' num2str(xmax)])
```



Referencias

- An Engineers Guide to MATLAB, Edward B. Magrab, Shapour Azarm, Balakumar Balachandran, James Duncan, Keith Herold, Gregory Walsh – Pearson
- Engineering Computation using MATLAB, David M. Smith – Pearson
- Solving applied mathematical problems with MATLAB, Dingyu Xue, YangQuan Chen – CRC Press