

ENCUENTRO

cuarto

CIENCIAS BÁSICAS



ENCUENTRO cuarto
CIENCIAS BÁSICAS

Herramientas Básicas para Trabajo con Matlab

Carlos Toro

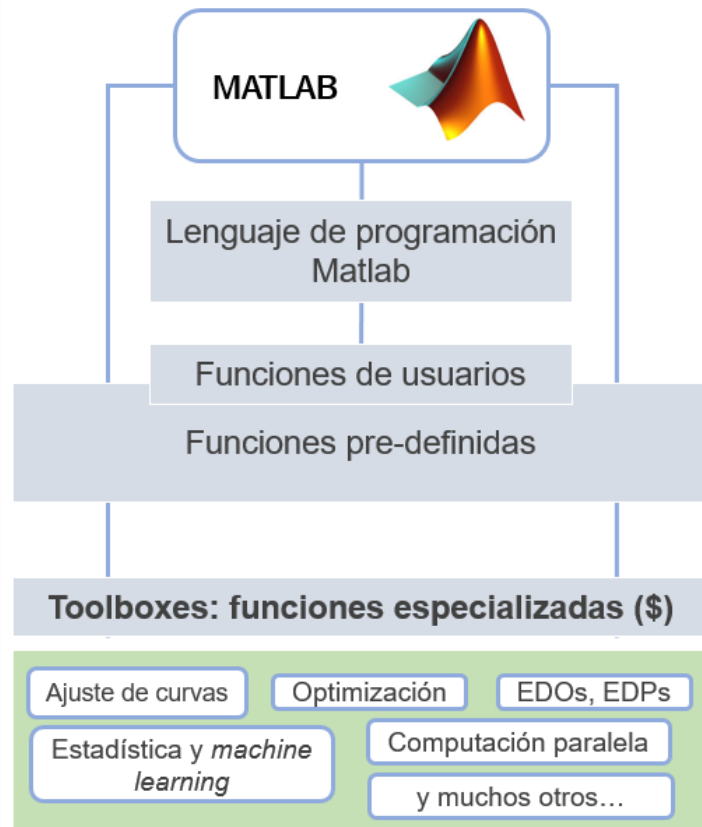
carlos.toro.ing@gmail.com

Introducción

Matlab™: Matrix Laboratory

- ❑ Lenguaje de programación de alto nivel
- ❑ Principalmente usado para:
 - Computación numérica y optimización
 - Análisis de datos y visualización
 - Desarrollo de algoritmos y programación
 - Desarrollo de aplicaciones
 - Modelación,...
- ❑ Aplicaciones: Sistemas de control; procesamiento de señales; visión por computador; mediciones; simulaciones; desarrollo de interfaces gráficas de usuario, ...

Documentación: [mathworks.com](https://www.mathworks.com)



Introducción

Página de descarga

Mi software

Licencia	Etiqueta	Opción	Uso
40904954	MATLAB (Individual)	Total Headcount	Academic



R2022b

Obtener productos de MATLAB y Simulink

Descargar para Windows
(228 MB)

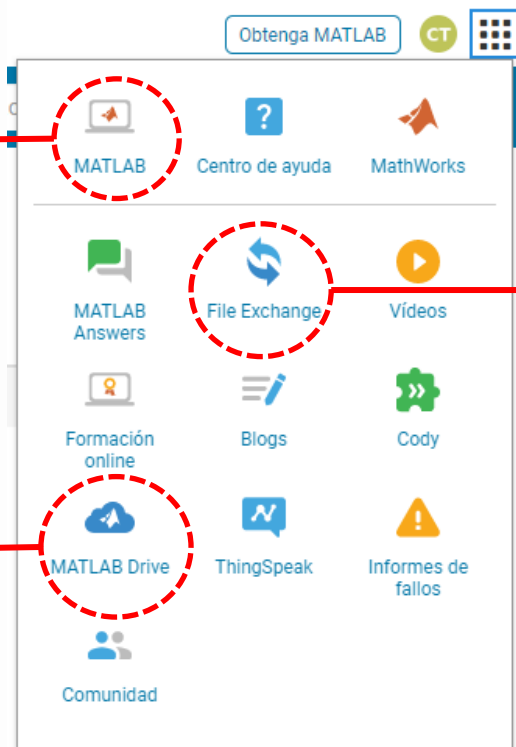
Incluye R2022b Update 3
Publicada: 22 Dic. 2022

Descarga del instalador

Introducción

Perfil Online

Para acceder a Matlab versión online

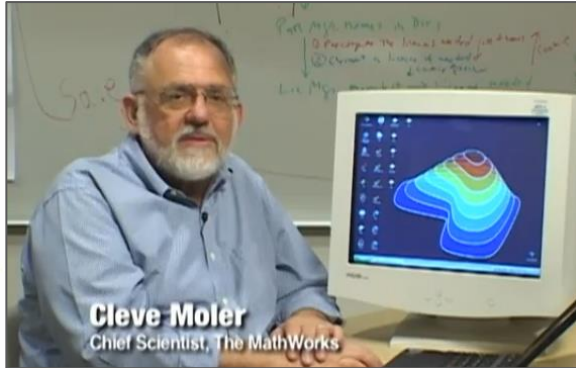


Intercambio de archivos con ejemplos de código y de aplicaciones propuestos por otros usuarios.

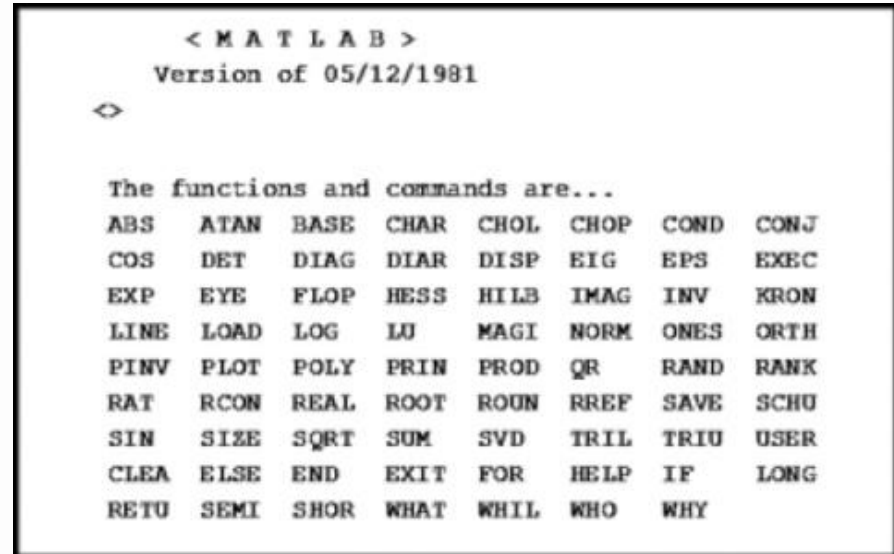
Espacio en la nube para almacenar sus archivos, código, datos, etc.

Introducción

Un poco de historia

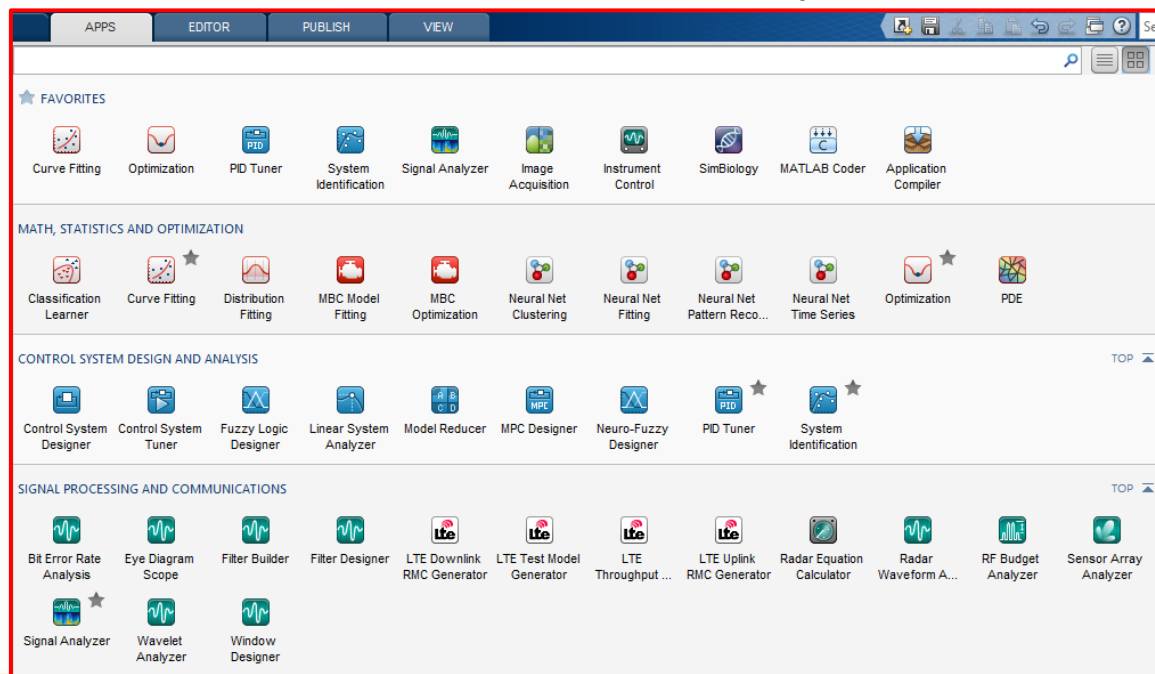


Cleve Moler es un matemático y programador de EEUU, especialista en análisis numérico.
(1939-)



Herramientas/Aplicaciones extra

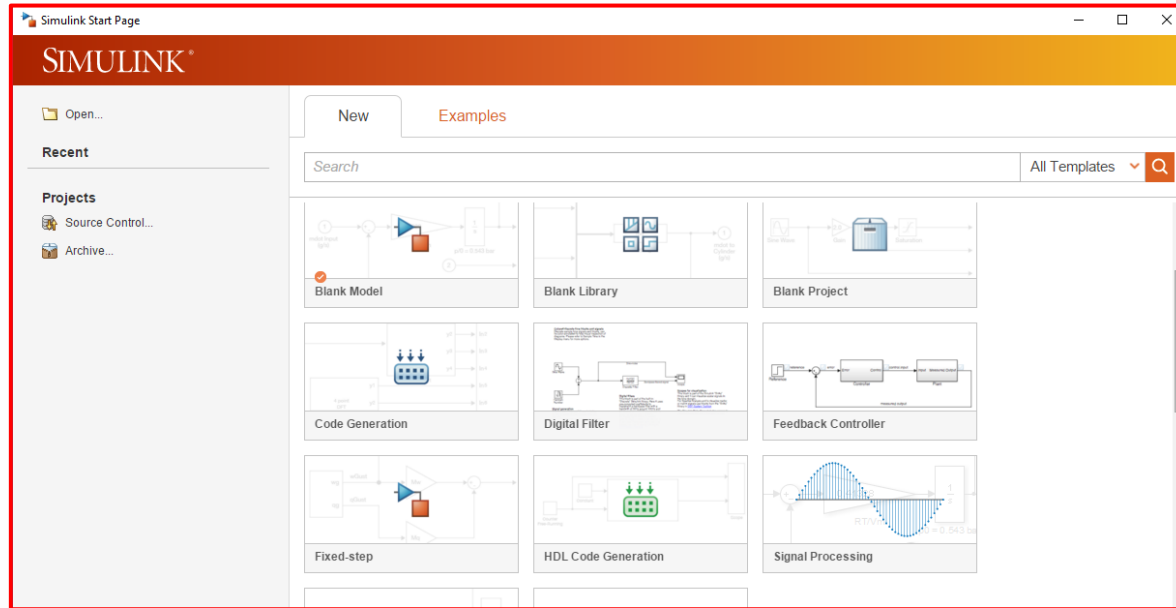
Menú con otras herramientas de análisis, diseño y modelación:



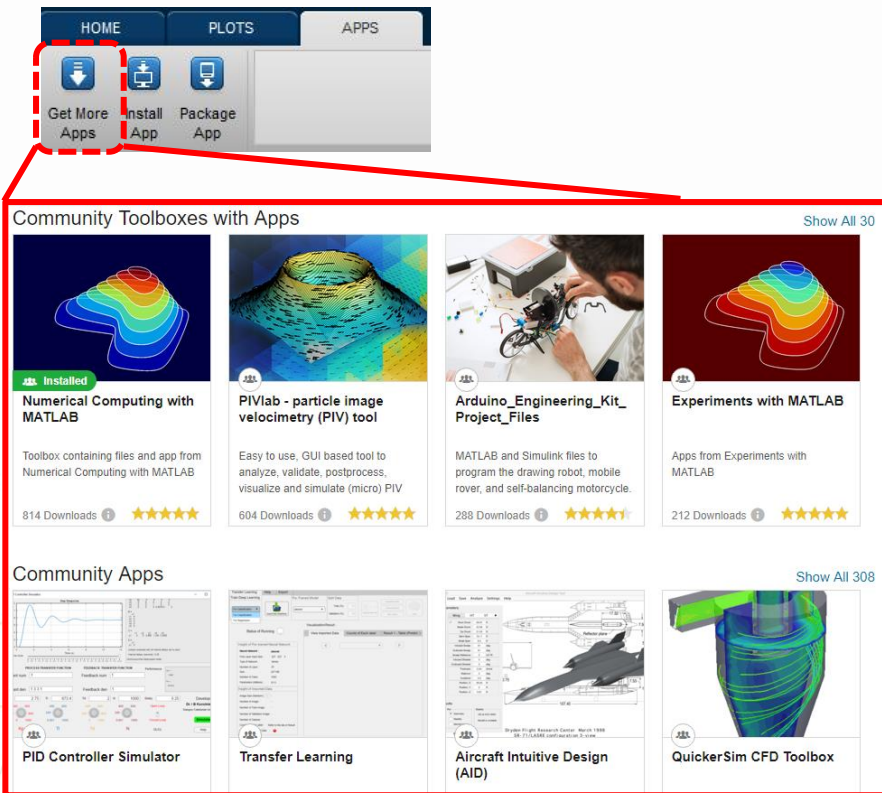
Se pueden instalar las que necesiten usando su licencia.

Herramientas/Aplicaciones extra

Simulink: Interfaz de programación gráfica, útil para modelar, simular y analizar sistemas.


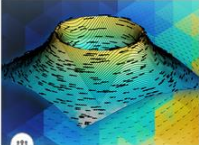

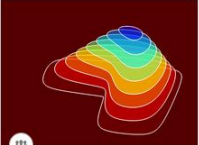


Complementos

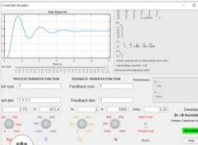

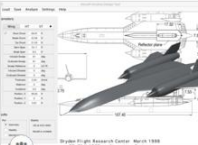
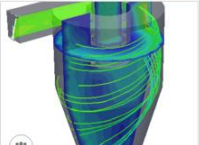


The screenshot shows the MATLAB App Designer interface. At the top, there are tabs for 'HOME', 'PLOTS', and 'APPS'. Below these, there are three buttons: 'Get More Apps' (highlighted with a red dashed box), 'Install App', and 'Package App'. A red line connects the 'Get More Apps' button to a larger view of the 'Community Toolboxes with Apps' section. This section displays a grid of available toolboxes and apps, each with a thumbnail image, a title, a description, and download statistics.

Community Toolboxes with Apps [Show All 30](#)

Thumbnail	Toolbox/App Name	Description	Downloads	Rating
	Numerical Computing with MATLAB	Toolbox containing files and app from Numerical Computing with MATLAB	814 Downloads	★★★★★
	PIVlab - particle image velocimetry (PIV) tool	Easy to use, GUI based tool to analyze, validate, postprocess, visualize and simulate (micro) PIV	604 Downloads	★★★★★
	Arduino_Engineering_Kit_Project_Files	MATLAB and Simulink files to program the drawing robot, mobile rover, and self-balancing motorcycle.	288 Downloads	★★★★★
	Experiments with MATLAB	Apps from Experiments with MATLAB	212 Downloads	★★★★★

Community Apps [Show All 308](#)

Thumbnail	App Name
	PID Controller Simulator
	Transfer Learning
	Aircraft Intuitive Design (AID)
	QuickerSim CFD Toolbox

Desde este menú pueden Instalar aplicaciones o ejemplos diseñados por otros usuarios o herramientas oficiales de Mathworks, para ello necesitarán estar vinculados con su usuario registrado.

Interfaz de Matlab (online)

Menús con diversas herramientas

Workspace con variables activas en memoria

Carpeta actual donde se está ejecutando el código

Editor de scripts y funciones

Visualizaciones

Ventana de comandos

The screenshot displays the MATLAB online environment. At the top, a navigation bar includes 'HOME', 'PLOTS', 'APPS', 'FIGURE', and 'TOOLS'. Below this is a toolbar with icons for saving, showing code, line color, and various plot formatting options like title, axis labels, legend, colorbar, grid, and remove grid. A search bar is also present.

On the left, the 'Current Folder' pane shows a file tree with folders like 'Crack_gage', 'Ejemplos Lógica L', 'MerchData', 'Pegas Freelance', 'Published (my site)', 'Taller de Machine', and 'Taller_Matlab_Tel'. A blue arrow points from the text 'Carpeta actual donde se está ejecutando el código' to this pane.

The central area is the 'Editor de scripts y funciones', showing a MATLAB script named 'SimulacionEDO.m'. The script defines a function for a differential equation and solves it using 'ode45'. A blue box highlights this editor area.

To the right of the editor is the 'Figure' window, titled 'Figure 1', which displays a plot of 'Respuesta y(t)' versus 't(s)'. The plot shows two signals: 'y(t)' (solid red line) and 'u(t)' (dashed blue line). A blue box highlights this figure area.

At the bottom right is the 'Workspace' pane, which lists variables in memory: 'CI', 'Dx', 'ft', 'k', 'ts', 'tspan', 'ut', 'wn', 'X', and 'xi'. A blue arrow points from the text 'Workspace con variables activas en memoria' to this pane.

At the bottom center is the 'Command Window', which shows the command 'SimulacionEDO' entered. A blue box highlights this window area.

SINTAXIS BÁSICA

Símbolos comúnmente usados en Matlab

- **Corchetes []**: Para crear arreglos, vectores, matrices.
- **Paréntesis ()**: Para seleccionar elementos específicos dentro de un arreglo, vector o matriz.
- **Coma ,**: Para separar elementos o argumentos de funciones.
- **Punto y coma ;**: Para terminar una fila, o suprimir la salida de una línea de código.
- **Porcentaje %**: Para comentar líneas de código o agregar comentarios.
- **Transpuesto '**: Para transponer los elementos de vectores o matrices.
- **Operador dos puntos :**: Útil para crear arreglos o indexar elementos dentro de estos.
- **Tres puntos ...**: Le dice a Matlab que el código de una línea continuará en la siguiente.

Para una revisión completa de operadores en Matlab y caracteres especiales ir a la referencia [1].

Crear variables

- ❑ **Escalares:** toman solo un valor, e.g. `a = 5; myResult = 14;`
- ❑ **Vectores:** e.g. un vector fila `y = [1 5 9]` los elementos son separados con comas o espacios.
- ❑ **Matrices:** `x = [x11 x12 ... x1m; x21 x22 ... x2m; ... ; xn1 xn2 ... xnm];`
e.g. `x = [1 2 3; 3 5 7; 5 8 9]`

`x =`

1	2	3
3	5	7
5	8	9

Las matrices equivalen a una colección de vectores filas separados con el operador punto y coma.

- ❑ **Arreglos multidimensionales:** e.g. Arreglos de matrices, **cuidado con las dimensiones y asignaciones!**. E.g. Una imagen a color puede ser vista como la combinación de tres matrices, una para el canal rojo, una para el canal verde y otra para el canal azul.

Extrayendo valores (indexación)

Ejemplos

Scalar of 1x1

Índice: 1 2 3
Row vector of 1x3

Column vector of 2x1

One dimensional matrix/array of 2x3

Two dimensional matrix/array of 2x3

N dimensional matrix/array of 2x3

□ E.g. Sea un vector v con n elementos, una matriz A de $m \times n$ elementos y un arreglo multidimensional M de $m \times n \times p$, luego:

- a) $v(1:end)$ indexa?...
- b) $A(:,n)$ indexa? ...
- c) $A(1:m,:)$ indexa?...
- d) $A(a:b,:)$ indexa?...
- e) $A(a:b,c:d)$ indexa?...
- f) $M(:,:,3)$ indexa?...

* En matrices, primero indexar filas y luego columnas

Extrayendo valores (indexación)

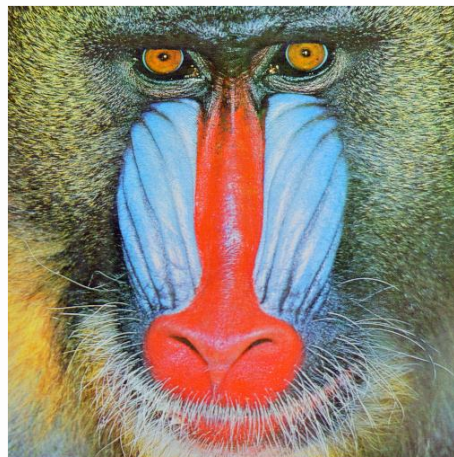
Ejemplo, una imagen a color en Matlab se puede interpretar como un arreglo de 3 matrices, una para cada canal de color (Rojo, Verde o Azul):

```
Img = imread('mandrill.png');  
figure;  
imshow(Img);  
dim = size(Img)% tamaño de la  
           % imagen
```

```
dim =  
  
    512    512     3
```

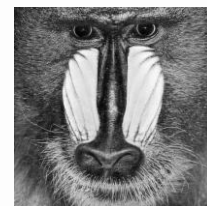
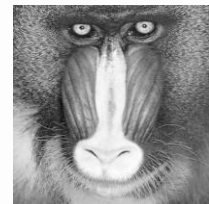
Alternativas a imshow

```
imagesc(Img)  
imtool(Img)  
image(Img)
```



```
imshow(Img(:,:,1));%R  
imshow(Img(:,:,2));%G  
imshow(Img(:,:,3));%B
```

Plano Rojo (R) **Plano Verde (G)**



Plano Azul (B)

*Imagen mandrill.png disponible internamente en Matlab

Extrayendo valores: Indexación lineal

$$\mathbf{A} = \begin{matrix} & \begin{matrix} 1 & 2 & \dots & q & \dots & n \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ \vdots \\ p \\ \vdots \\ m \end{matrix} & \begin{bmatrix} \underbrace{a_{11}}_1 & \underbrace{a_{12}}_{m+1} & \dots & \underbrace{a_{1q}}_{(q-1)m+1} & \dots & \underbrace{a_{1n}}_{(n-1)m+1} \\ \underbrace{a_{21}}_2 & \underbrace{a_{22}}_{m+2} & \dots & \underbrace{a_{2q}}_{(q-1)m+2} & \dots & \underbrace{a_{2n}}_{(n-1)m+2} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ \underbrace{a_{p1}}_p & \underbrace{a_{p2}}_{m+p} & \dots & \underbrace{a_{pq}}_{(q-1)m+p} & \dots & \underbrace{a_{pn}}_{(n-1)m+p} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ \underbrace{a_{m1}}_m & \underbrace{a_{m2}}_{2m} & \dots & \underbrace{a_{mq}}_{(q-1)m+m} & \dots & \underbrace{a_{mn}}_{(n-1)m+m} \end{bmatrix} \end{matrix}$$

Diagram illustrating linear indexing in a matrix \mathbf{A} . The matrix is shown with columns indexed 1, 2, ..., q , ..., n and rows indexed 1, 2, ..., p , ..., m . The element a_{pq} is highlighted with a blue box and labeled k . A solid blue arrow points from the element a_{11} (labeled 1) to a_{pq} (labeled k). A dashed blue arrow points from the element a_{12} (labeled $m+1$) to a_{pq} (labeled k).

Al escribir en Matlab $\mathbf{A}(1 : k)$, extraemos k elementos, donde k indexa al elemento (p, q)

Utilidades para definir variables

- **rand**: Útil para generar números aleatorios uniformemente distribuidos en el rango [0,1]. e.g. Para crear un vector r con N valores aleatorios uniformemente distribuidos en el rango $[a,b]$:

```

$$r = a + (b-a) * \text{rand}(N,1) ;$$

```

- **randn**: Útil para generar valores aleatorios normalmente distribuidos. e.g. para crear un vector r con N números aleatorios distribuidos normalmente con media = 1 desviación estándar = 2 escribiremos:

```

$$r = 1 + 2 * \text{randn}(N,1) ;$$

```

- **eye**: Para crear una matriz identidad (o matriz con 1s en su diagonal principal).
- **zeros**: Para crear un vector o matriz llena de 0s.
- **diag**: Usada para crear una matriz diagonal con los elementos dados de un vector o para extraer la diagonal principal de una matriz.

Utilidades para definir variables (cont.)

- **ones**: Para crear un vector o matriz con valores de 1s.
- **randperm(N)**: Para crear un vector con una permutación aleatoria de valores tomados de un rango de enteros de 1 a N, e.g. `randperm(5)` entrega `[5 3 2 4 1]`. ¿Qué generará `randperm(10,4)`?
- **linspace**: Para crear un vector fila con un número fijo de elementos dentro de un rango definido. e.g.: `linspace(1,3,5)` genera `[1 1.5 2 2.5 3]`. Si no se especifica el número de elementos, por defecto es 100.
- **Operador dos puntos ':'**: Equivalente a `linspace`, en este caso se especifica la separación entre los valores del vector. e.g. `2:3:8` produce `[2 5 8]`.

Variables reservadas en Matlab

- **ans**: Contiene el resultado de la última sentencia cuando la operación no es asignada a ninguna variable.
- **pi**: número π .
- **inf**: representa al infinito.
- **i**: unidad imaginaria, definido como la raíz cuadrada de -1.
- **j**: equivalente a **i**.
- **NaN**: *Not a Number* (valor no numérico). Matlab lo usa cuando no puede determinar un valor válido de una operación, e.g. resultado de 0/0.

Operaciones

Operaciones matemáticas básicas

- Operaciones escalares y vectoriales: $+$ $-$ $*$ $/$ $^$, usar las reglas del álgebra lineal, e.g. siempre se pueden multiplicar dos matrices rectangulares?
- Operaciones sobre arreglos: $+$ $-$ $.*$ $./$ $.^$
- Ejemplos:

Si $\mathbf{v1} = [1 \ 2 \ 3]$ y $\mathbf{v2} = [2 \ 7 \ 3]$ luego:

$$\begin{aligned}\mathbf{v1}.^2 &= [1 \ 4 \ 9], \\ \mathbf{v1}.*\mathbf{v2} &= [2 \ 14 \ 9] \text{ y} \\ \mathbf{v1}*\mathbf{v2}' &= 25.\end{aligned}$$

Operaciones (cont.)

Operadores lógicos y comparaciones

- Las operaciones lógicas son útiles para comparar expresiones o extraer valores específicos dentro de un arreglo. Los mas usados son: or: `||`; and: `&&`; `>`; `<`; `>=`; `<=`; not: `~`, not equal to: `~=`.
- Ejemplos:** Usando los vectores `v1 = [1 2 3]` y `v2 = [2 7 3]`, las siguientes operaciones entregan:
 - `v1 == v2` genera `[0 0 1]`,
 - `v2 > v1` genera `[1 1 0]` y
 - `v1(v1 == v2)` devolverá 3 (qué hace `find()`?)

Ejemplo: Indexación lógica

En el caso matricial tenemos:

$$A = \begin{bmatrix} 42 & 44 & 0 & 34 \\ 1 & 9 & 98 & 78 \\ 15 & 90 & 88 & 0 \\ 8 & 2 & 32 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

logical values

>> E=A (B)

E =

42
15
90
0
3

La matriz B se puede construir con operadores lógicos y/o operadores de comparación.

Orden de operaciones (simplificado)

Orden de operaciones: **1)** Paréntesis $()$, **2)** Exponenciación $^$, **3)** Multiplicación y División $*$ / \backslash y **4)** Suma y Resta $+$ - y de izquierda a derecha si son iguales. En [2] se puede encontrar la lista completa.

Ej: Predecir las siguientes expresiones Matlab y verificar escribiéndolas en la ventana de comandos:

- $6/6+5$
- $2*6^2$
- $4*3/2*8$
- 2^3^4
- $2^ (3^4)$

Recomendación: Usar en lo posible paréntesis para llevar un orden de las operaciones y asegurar la ejecución correcta.

Funciones típicas de Matlab incluidas

❑ Funciones matemáticas elementales

Para una variable x (escalar, vector o matriz):

- **`sqrt(x)`** : Raíz cuadrada de x .
- **`exp(x)`** : Exponencial de x .
- **`abs(x)`** : Valor absoluto de x .
- **`log(x)`** : Logaritmo natural de x .
- **`log10(x)`** : Logaritmo en base 10 de x .

❑ Funciones trigonométricas

Para un ángulo x :

➤ En radianes

- **`sin(x)`**
- **`cos(x)`**
- **`tan(x)`**

➤ En grados

- **`sind(x)`**
- **`cosd(x)`**
- **`tand(x)`**

❑ Inversas, hiperbólicas, etc.

Funciones típicas de Matlab incluidas

Funciones de redondeo

Para una variable x e y (escalar, vector o matriz):

- **round(x)** : Redondeo de x al entero más próximo.
- **fix(x)** : Redondeo de x hacia cero.
- **ceil(x)** : Redondeo de x hacia infinito.
- **floor(x)** : Redondeo de x hacia menos infinito.
- **rem(x,y)** : Retorna el resto de la división entre x e y .
- **sign(x)** : Función de signo. Devuelve 1 si $x > 0$, -1 si $x < 0$ y 0 si $x = 0$.

Ejemplos:

```
>> a = 1.65;  
>> round(a)  
ans =  
      2  
  
>> fix(a)  
ans =  
      1  
  
>> b = 0.1;  
>> ceil(b)  
ans =  
      1  
  
>> rem(2,3)  
ans =  
      2
```

Funciones típicas de Matlab incluidas

❑ Otras

Para una variable x (escalar, vector o matriz):

- **size(x)**: Entrega las dimensiones de un arreglo, en el caso de una matriz, entrega el número de filas y columnas.
- **length(x)**: Entrega la longitud de x (dimensión mayor).
- **numel(x)**: Entrega el número de elementos en x .

❑ **Ejemplo:** Para la siguiente matriz se tiene,

```
>> A = rand(3,2)
```

A =

0.8147	0.9134
0.9058	0.6324
0.1270	0.0975

```
>> size(A)
```

```
ans =  
      3      2
```

```
>> numel(A)
```

```
ans =  
      6
```

Funciones típicas de Matlab incluidas

❑ Otras útiles para analizar datos

Para una variable x (escalar, vector o matriz):

- **`min(x)` y `max(x)`**: Retorna el valor mínimo y máximo del vector x respectivamente, en matrices usar $x(:)$ para obtener el mínimo entre todos los elementos.
- **`mean(x)`**: Promedio de elementos de x (a lo largo de la primera dimensión del arreglo cuyo tamaño no es igual a 1).
- **`std(x)`**: Desviación estándar de los elementos de x .
- **`histogram(x,nbins)`**: Para visualizar el histograma de un conjunto de datos agrupados en `nbins` (barras).

Ejercicio

Transferencia de calor

Un objeto con temperatura inicial T_0 se introduce en el instante $t = 0$ dentro de una cámara que tiene una temperatura constante T_s . Entonces, el objeto experimenta un cambio de temperatura que se corresponde con la ecuación:

$$T = T_s + (T_0 - T_s) \cdot e^{-k \cdot t}$$

Donde T es la temperatura del objeto en el instante t (en horas), y k es una constante. Luego, si una botella, con una temperatura de 15°C , se introduce en un refrigerador que tiene en su interior una temperatura de 3°C .

Calcular, redondeando el resultado al grado más próximo, la temperatura de la botella después de dos horas.

Considerar $k = 0.45$. Declarar primero todas las variables y luego calcular la temperatura utilizando un solo comando.

Repetir considerando un vector de tiempo $t = 0:0.1:2$.

Ejercicio

Transferencia de calor

```
Ts = 3;%°C
T0 = 15;%°C
k = 0.45;%°C
t = 0:0.1:2;
T = round(Ts + (T0-Ts)*exp(-k*t))
T =
```

Columns 1 through 9

15	14	14	13	13	13	12	12	11
----	----	----	----	----	----	----	----	----

Columns 10 through 18

11	11	10	10	10	9	9	9	9
----	----	----	----	----	---	---	---	---

Columns 19 through 21

8	8	8
---	---	---

Otras utilidades

- **help**: Para obtener ayuda acerca de una función particular, ej: ejecutar el comando **help hist**.
- **clc**: Limpia la línea de comandos
- **clear**: Borra una variable específica del espacio de trabajo, ej. **clear x**, al usarla con el comando **clear all**, borra todas las variables del espacio de trabajo.
- **close**: Cierra la figura actual desplegada, **close all** cierra todas las ventanas de figuras abiertas.
- **who** y **whos**: Variables actuales guardadas en memoria.

GRÁFICOS Y VISUALIZACIÓN DE DATOS

Visualizando datos

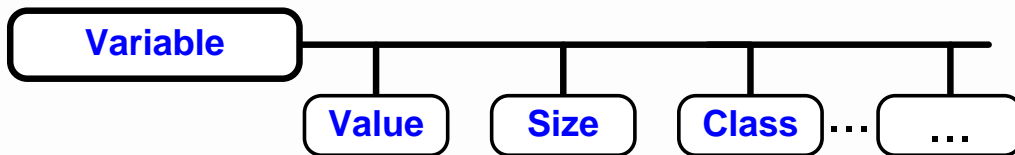
☐ Qué son los datos?:






- en inglés datum (singular) y data (plural), se puede definir como:
- **Def. 1** Hechos o información, especialmente cuando son examinados y usados para descubrir cosas o tomar decisiones. (*Oxford dictionary*).
- **Def. 2** Información que es almacenada por un computador.

Concepto 'Trendy' a tener en cuenta: DATA SCIENCE (Ciencia de los datos): involucra aplicaciones de la estadística, ciencia de la computación, ingeniería de software y otros campos relevantes a la aplicación (ej. metalurgia) para dar sentido a los datos.

Tipos de datos en Matlab

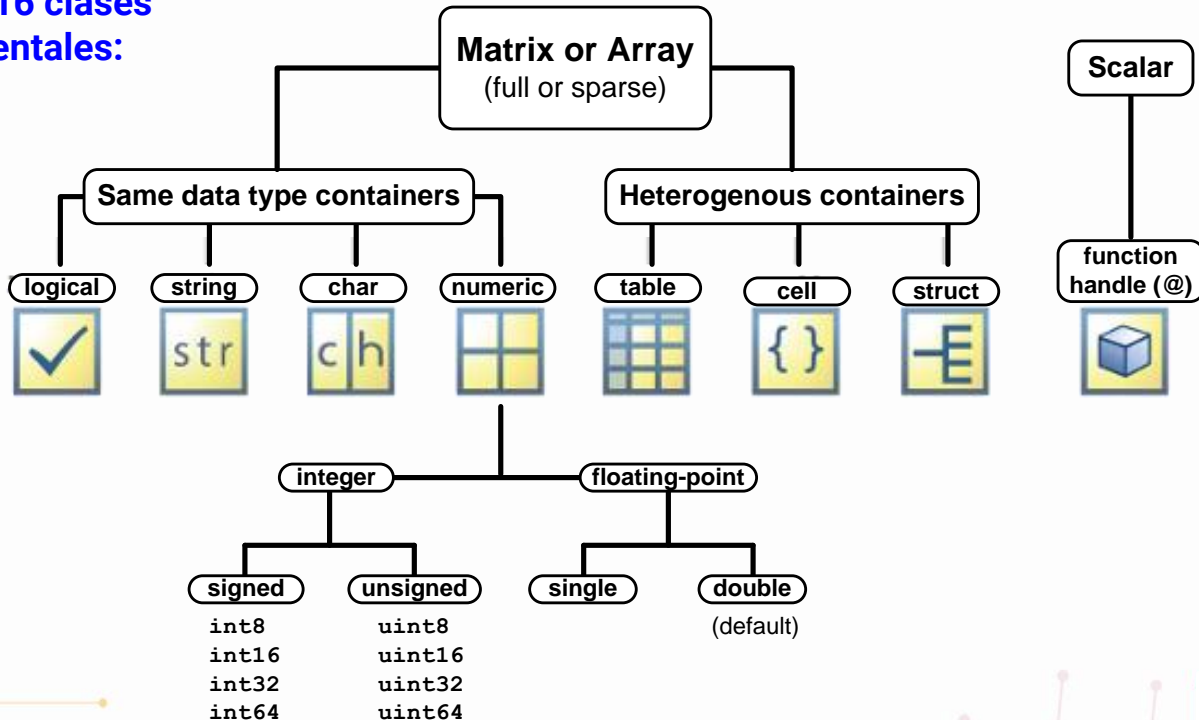
En Matlab podemos trabajar con distintos tipos de dato, en el workspace se pueden observar los valores y las clases de las variables.



Workspace			
Name ▲	Value	Size	Class
 c	'char is char'	1x12	char
 f	@(a,x)a*sin(x)	1x1	function_handle
 g	1x2 cell	1x2	cell
 x	1x100 double	1x100	double
 y	0	1x1	uint8

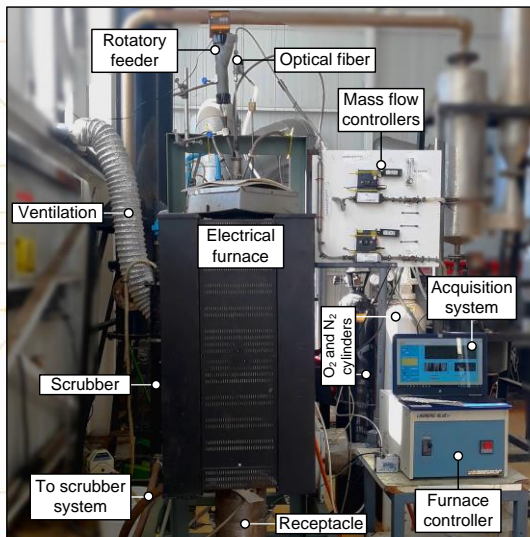
Tipos de datos en Matlab (cont.)

Existen 16 clases fundamentales:



Fuentes de datos

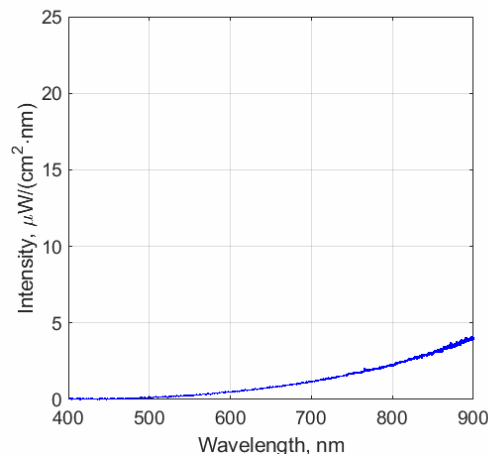
En especialidades de ingeniería o ciencias, una de las principales fuentes de información son las **SEÑALES** medidas desde procesos físicos o químicos por medio de sensores/instrumentos especiales, e.g.: Las siguientes imágenes muestran señales de radiación espectral capturadas desde un proceso de combustión de minerales de cobre y hierro en un montaje de laboratorio.



Montaje para experimentos de combustión



Llama generada



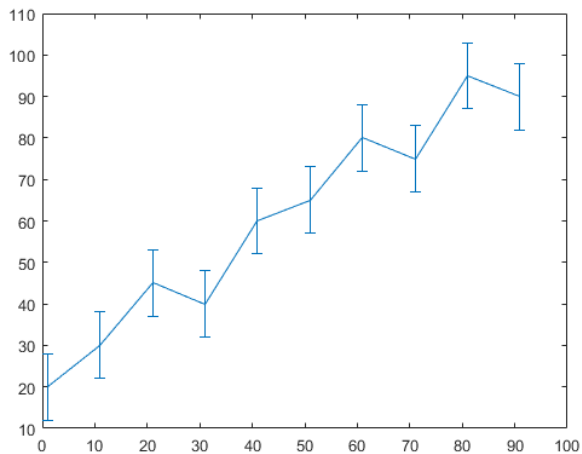
Señales capturadas

Visualizando datos

- ❑ **Funciones más usadas:**
 - Para gráficos 2D: `plot`, `errorbar`, `area`, `semilogx`, `semilogy`, `loglog`, `contour`.
 - Para gráficos 3D: `surf`, `mesh`, `plot3`. Utilidades: `colormap`, `colorbar`.
 - Para mapas de intensidad: `imagesc(datos_ordenados_en_matriz, escala)`.
- ❑ Varias gráficas en la misma figura (diferentes ejes): usar `subplot`
- ❑ Usar el comando `figure(n° de figura)` para crear y llevar un orden de cada figura, si no se sobrescribirán.
- ❑ Más de un gráfico en la misma figura (mismos ejes): usar `hold on`

Algunos tipos de gráficos

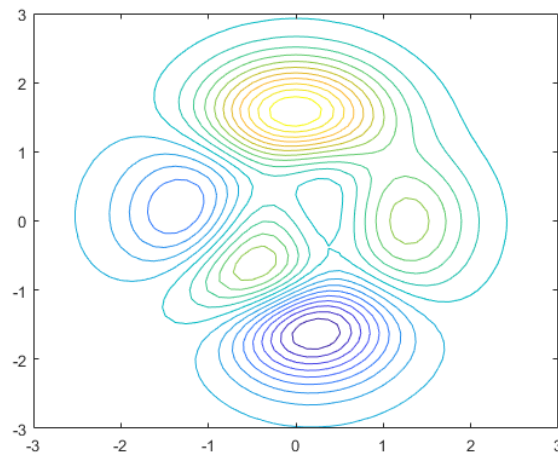
errorbar



```

x = 1:10:100;
y = [20 30 45 40 60 65 80 75 95 90];
err = 8*ones(size(y));
errorbar(x,y,err)
  
```

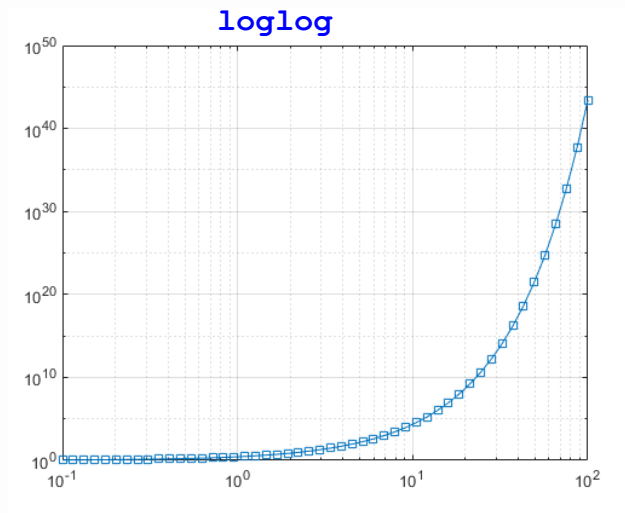
contour



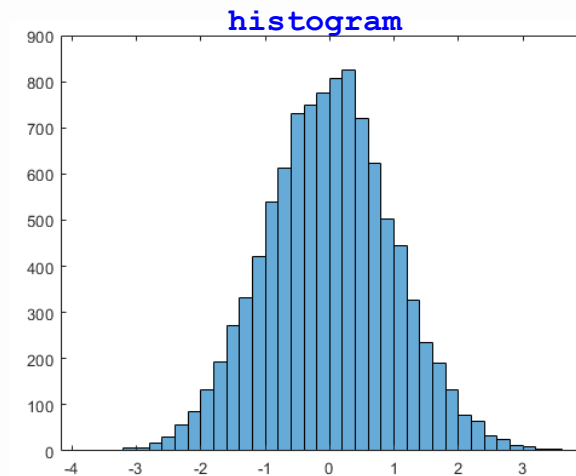
```

[X,Y,Z] = peaks;
figure
contour(X,Y,Z,20)
  
```

Algunos tipos de gráficos

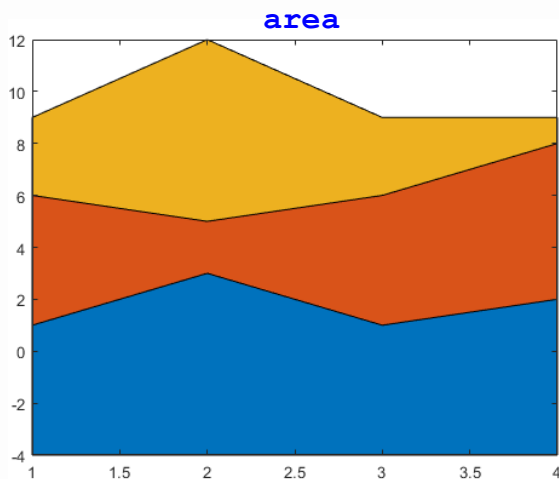


```
x = logspace(-1,2);  
y = exp(x);  
loglog(x,y,'-s') grid on
```

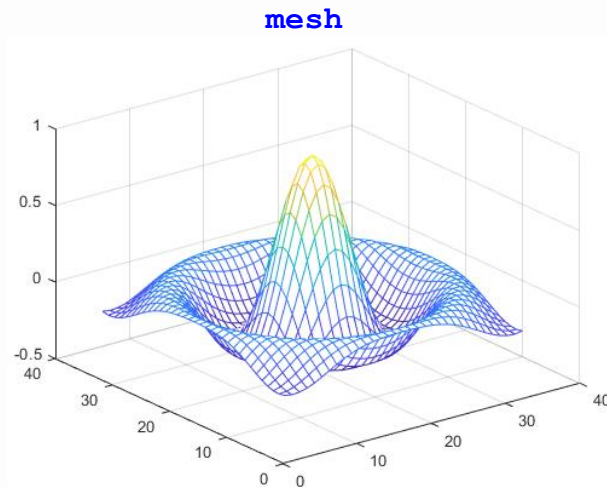


```
x = randn(10000,1);  
histogram(x)
```

Algunos tipos de gráficos



```
Y = [1, 5, 3; 3, 2, 7;  
1, 5, 3; 2, 6, 1];  
Figure;  
basevalue = -4;  
area(Y,basevalue)
```



```
[X,Y] = meshgrid(-8:.5:8);  
R = sqrt(X.^2 + Y.^2) + eps;  
Z = sin(R)./R; figure; mesh(Z)
```

Algunos tipos de gráficos

❑ Quiver o gráfico de campos vectoriales:

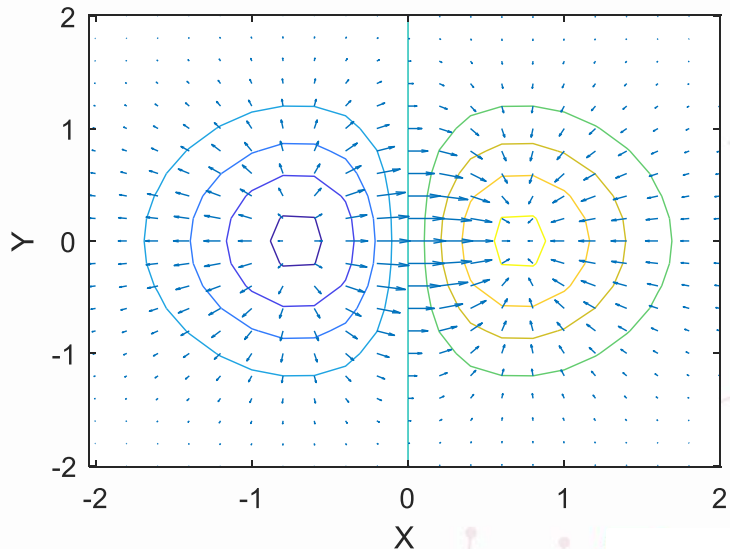
Toma los dos primeros argumentos (x,y) como el origen de cada vector, y usa los dos argumentos siguientes (u,v) para obtener la dirección y magnitud de cada flecha. Ejemplo, graficar el gradiente de la función $z = xe^{-x^2-y^2}$.

```
[X,Y] = meshgrid(-2:.2:2);
Z = X.*exp(-X.^2 - Y.^2);
[DX,DY] = gradient(Z,.2,.2);
figure
contour(X,Y,Z)
hold on
quiver(X,Y,DX,DY)
```

La función `[X,Y] = meshgrid(x,y)` devuelve coordenadas de una cuadrícula 2-D basadas en las coordenadas contenidas en los vectores `x` e `y`.

`x` es una matriz donde cada fila es una copia de `x`, e `Y` es una matriz en la que cada columna es una copia de `y`.

Útil para evaluar funciones de dos o más variables.

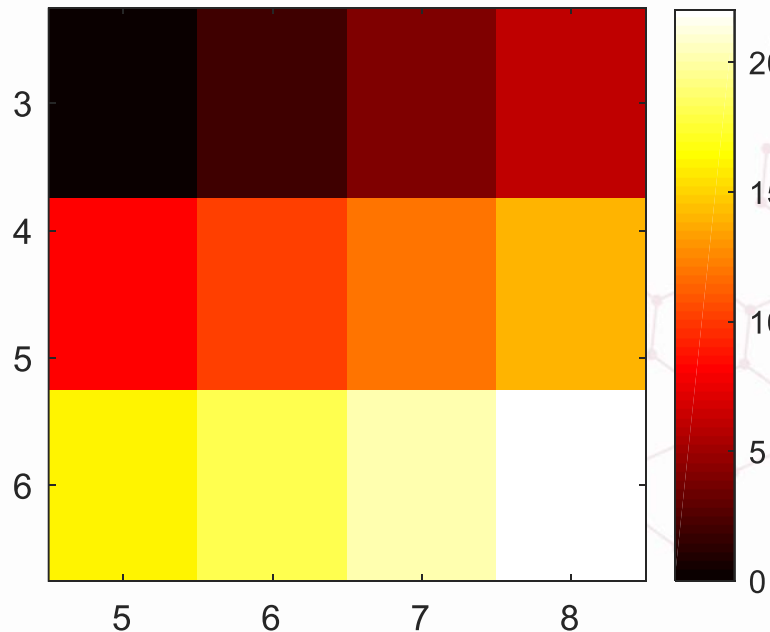


Algunos tipos de gráficos

Visualización de matrices:

Para visualizar matrices o arreglos de datos en forma de imagen podemos usar las funciones `image()`, `imshow()`, o `imagesc()`, con esta función podemos mapear los valores de cada elemento de una matriz con una escala de intensidades o pseudocolor, como se muestra en la siguiente figura.

```
C = [0 2 4 6; 8 10 12 14; 16 18 20 22];  
x = [5 8]:  
y = [3 6];  
imagesc(x,y,C)  
colorbar;  
colormap('hot');
```


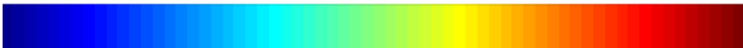













Caso práctico, cada valor de la matriz podría ser una medición, por ejemplo, de una propiedad química del suelo, luego, con esta herramienta se puede visualizar rápidamente su distribución espacial.

Algunos tipos de gráficos

❑ Visualización de matrices:

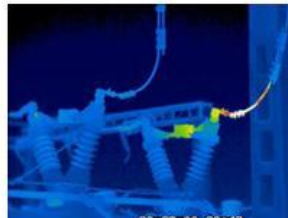
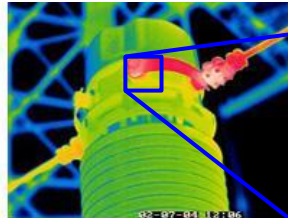
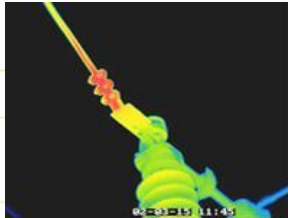
Otras opciones para mapear los valores de intensidad y asociarles un color con la función `colormap()`.

Nombre del mapa de colores	Escala de color
parula	
jet	
hsv	
hot	
cool	
spring	
summer	
autumn	
winter	
gray	
bone	
copper	
pink	

Algunos tipos de gráficos

❑ Visualización de matrices:

Un caso particular son las imágenes provenientes de cámaras, estas se interpretan en Matlab como arreglos de datos de: 1 matriz (caso monocromático), 3 matrices (caso color, una matriz para cada canal de color RGB), o más matrices (caso hiperespectral). Cada pixel de la imagen representa un valor de intensidad en una posición específica. Estas imágenes se pueden leer con la función `imread()`.



106	107	102	97	92	95
98	100	97	91	95	102
87	86	83	85	92	106
78	71	70	71	85	100
84	73	69	65	79	92
87	80	75	75	78	89
80	80	76	79	82	90
77	76	76	79	84	97
73	77	77	84	89	105
70	80	87	94	102	114
68	82	91	97	104	116

Termografías IR
(imágenes en pseudocolor)

Figura típicas: `plot()`

❑ Parámetros de ajuste de la función `plot()`:

- Color de cada línea: `b`, `g`, `r`, `c`, `m`, `y`, `k`.
- Grosor de las líneas: `..., 'LineWidth', 1.5)`
- Tipo de línea: `-`, `:`, `-.`, `--`.
- Leyendas: `legend('line1', 'line2', ...)`
- Etiquetas y nombres de los ejes y título:

```
xlabel('nombre_ejex'),  
ylabel('nombre_ejey'),  
title('texto_titulo')
```

- Rangos de los ejes: `axis([xmin xmax ymin ymax]), axis tight`.
- Símbolos para puntos: `.`, `o`, `x`, `+`, `s`, `d`, `<`, `>`, `p`, `h`, `^`, `*`.
- Tamaño, relleno y bordes de algunos tipos de puntos:

```
..., 'MarkerFaceColor', 'w', 'MarkerEdge', 'r', ... 'MarkerSize', 10);
```

Figura típicas: `plot()`

❑ Parámetros de ajuste de la función `plot()`:

- Color de cada línea: `b`, `g`, `r`, `c`, `m`, `y`, `k`.
- Grosor de las líneas: `..., 'LineWidth', 1.5)`
- Tipo de línea: `-`, `:`, `-.`, `--`.
- Leyendas: `legend('line1', 'line2', ...)`
- Etiquetas y nombres de los ejes y título:

```
xlabel('nombre_ejex'),  
ylabel('nombre_ejey'),  
title('texto_titulo')
```

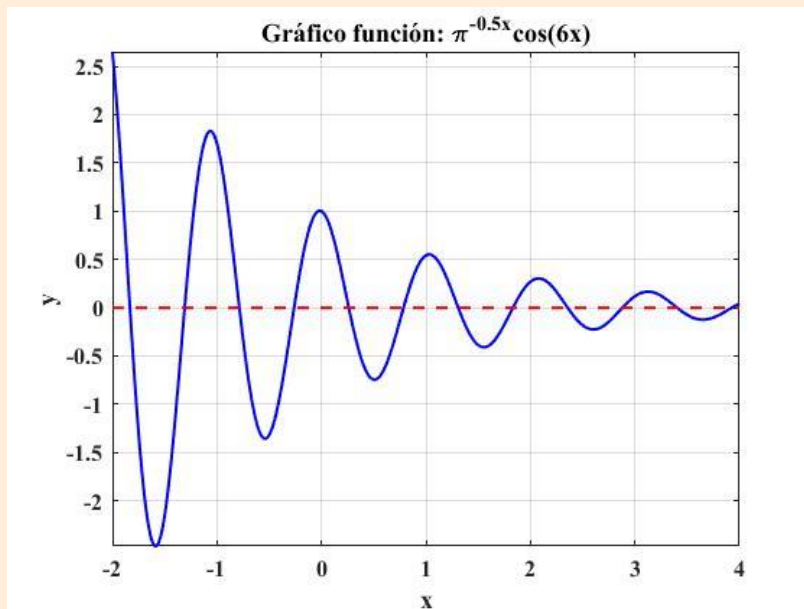
- Rangos de los ejes: `axis([xmin xmax ymin ymax]), axis tight`.
- Símbolos para puntos: `.`, `o`, `x`, `+`, `s`, `d`, `<`, `>`, `p`, `h`, `^`, `*`.
- Tamaño, relleno y bordes de algunos tipos de puntos:

```
..., 'MarkerFaceColor', 'w', 'MarkerEdge', 'r', ... 'MarkerSize', 10);
```

Ejercicio

Reproducir la siguiente figura basado en la siguiente función:

$$y = \pi^{-0.5x} \cos(6x), \forall x \in [-2, 4]$$



Ejercicio

Gráfico de la función:

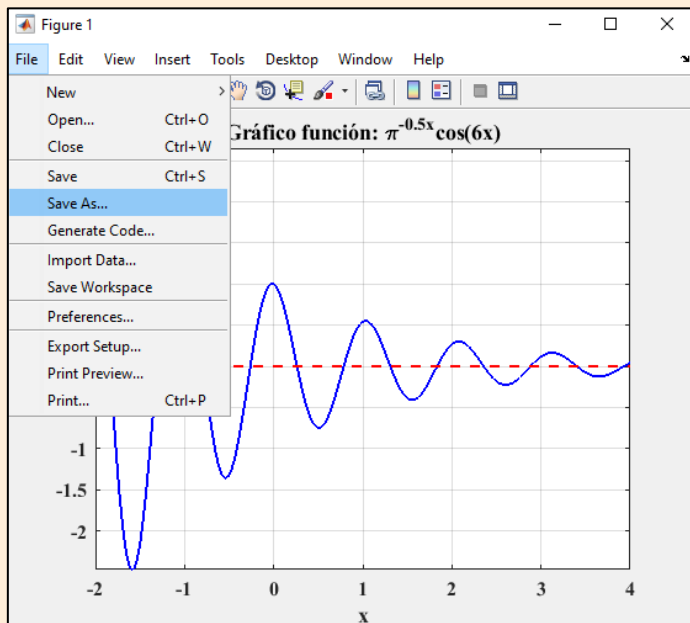
$$y = \pi^{-0.5x} \cos(6x), \forall x \in [-2, 4]$$

```
x = -2:0.001:4;% independent variable, axis x
y = pi.^(-0.5*x).*cos(6*x);% dependent variable, axis y

plot(x,y,'-b',x,0*y,'--r','LineWidth',1.5);
xlabel('x');
ylabel('y');
set(gca,'FontSize',12,'fontname','Times','FontWeight','bold');
axis tight; grid on;
title('Gráfico función: \pi^{-0.5x}\cos(6x)');
```

Ejercicio

Guardando o exportando la imagen: Se puede guardar diferentes formatos, unos con mejor calidad que otros.



MATLAB Figure (*.fig)

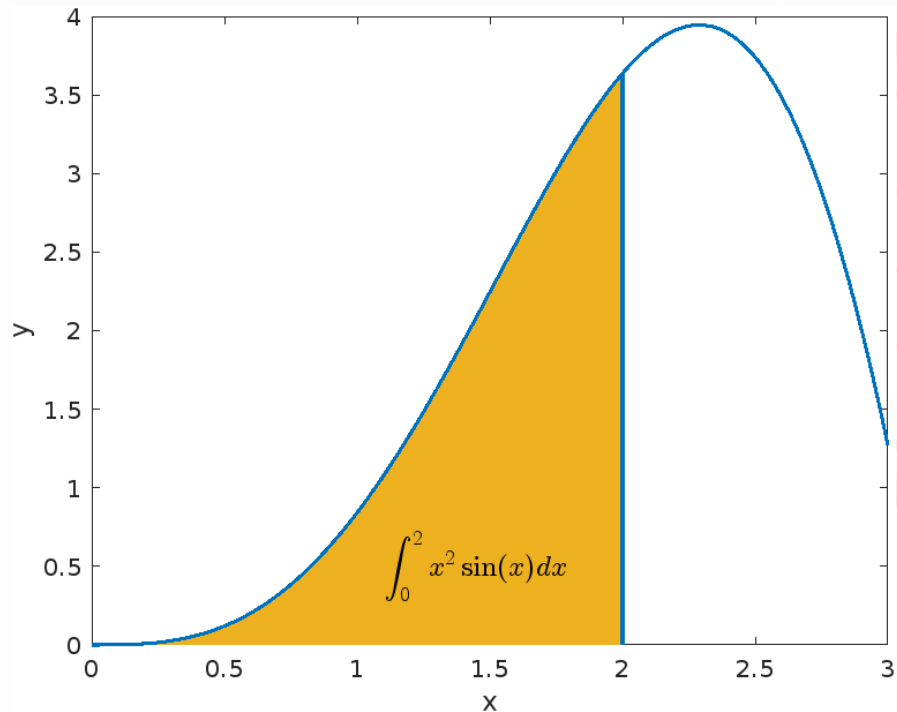
Bitmap file (*.bmp)
EPS file (*.eps)
Enhanced metafile (*.emf)
JPEG image (*.jpg)
Paintbrush 24-bit file (*.pcx)
Portable Bitmap file (*.pbm)
Portable Document Format (*.pdf)
Portable Graymap file (*.pgm)
Portable Network Graphics file (*.png)
Portable Pixmap file (*.ppm)
Scalable Vector Graphics file (*.svg)
TIFF image (*.tif)
TIFF no compression image (*.tif)

MATLAB Figure (*.fig)

Otra opción es ir a: Edit → Copy Figure o al submenú Export Setup para opciones avanzadas como calidad de impresión.

Usando Latex en los gráficos

- ❑ Como vimos, podemos añadir simbología matemática en los gráficos usando comandos Latex:
- Una referencia completa de esto la podemos encontrar en la documentación oficial de Matlab [aquí](#).
- Otro ejemplo, es el siguiente, el código se encuentra en archivo `Uso_de_Latex.m`.

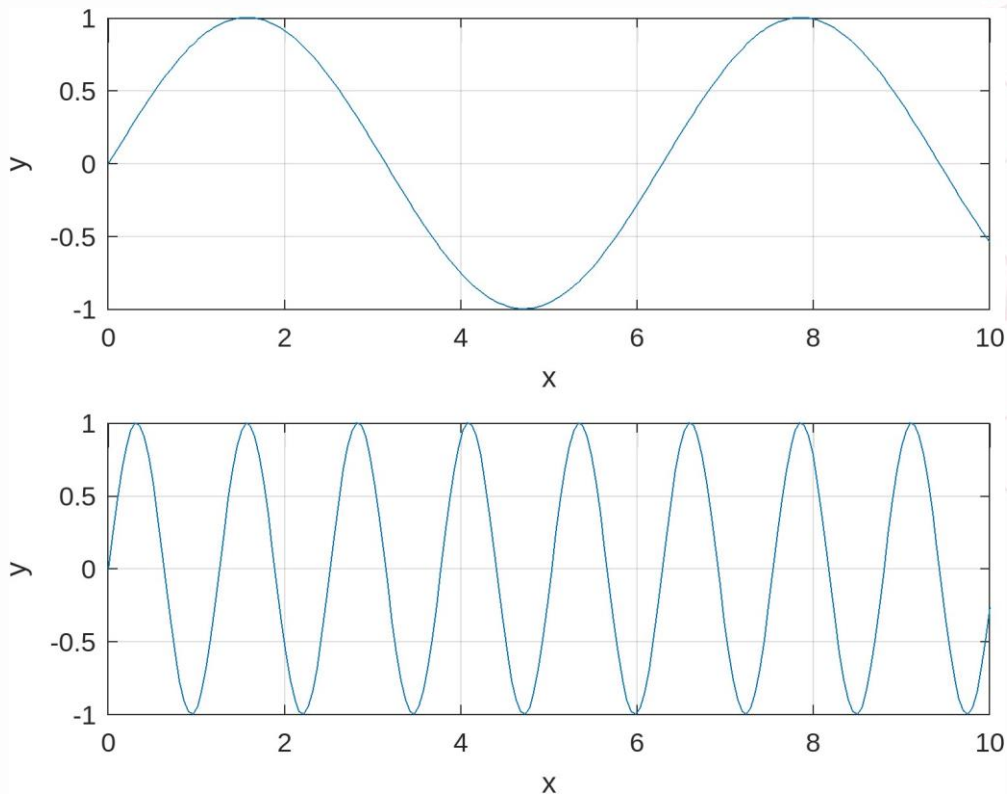


Mas de un gráfico en la misma figura

Para esto usaremos la función `subplot`.

La sintaxis es `subplot(m,n,p)`, esto separa la figura en una matriz de $m \times n$ y p identifica la porción de la ventana donde se dibujará la siguiente gráfica, con el código

`Uso_de_subplots.m` se genera la siguiente figura.



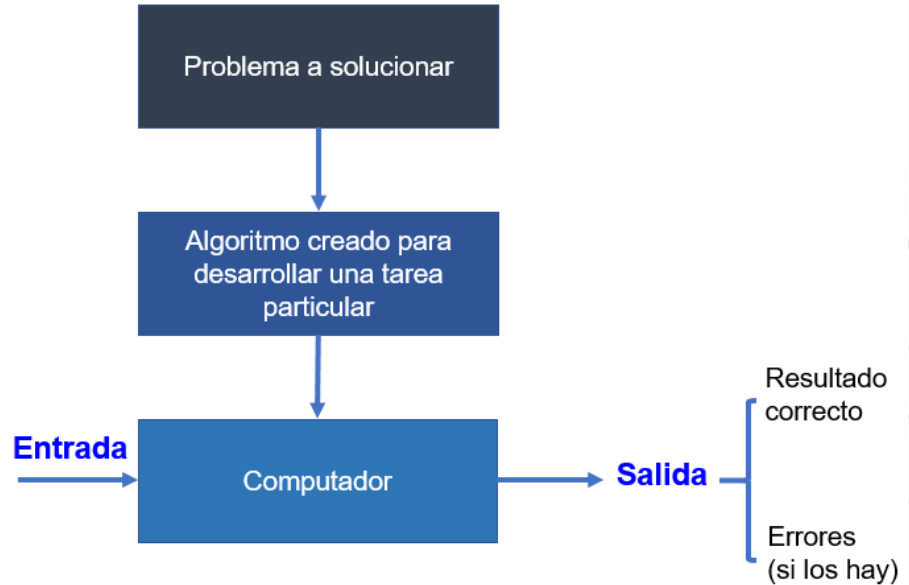
ELEMENTOS DE PROGRAMACIÓN EN MATLAB

Concepto de algoritmo

Algunas definiciones:

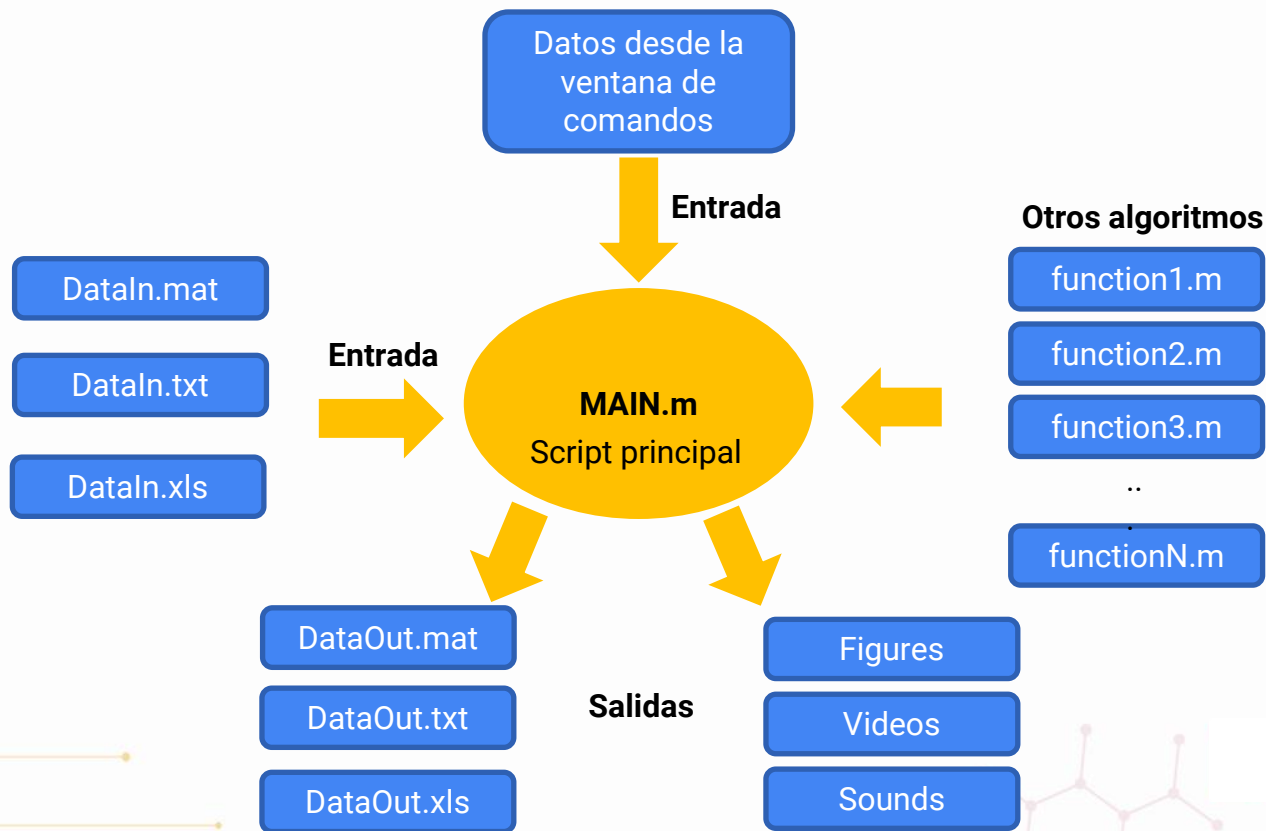
Def. 1: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema. (RAE)

Def. 2: Conjunto de *instrucciones* no ambiguas que ocurren en una *secuencia* específica y que produce una *salida* dado un conjunto de *entradas* en una cantidad de *tiempo finito* (A.A. Putanmbekar, 2010)



Nota histórica: La palabra fue nombrada por el matemático persa *al-Khowarizmi* en el siglo noveno, él uno de los padres del álgebra, aritmética, astronomía y geografía.

Programando en Matlab



Programando en Matlab

Tanto en *scripts* como en funciones los algoritmos siguen una estructura general:

Encabezado del algoritmo

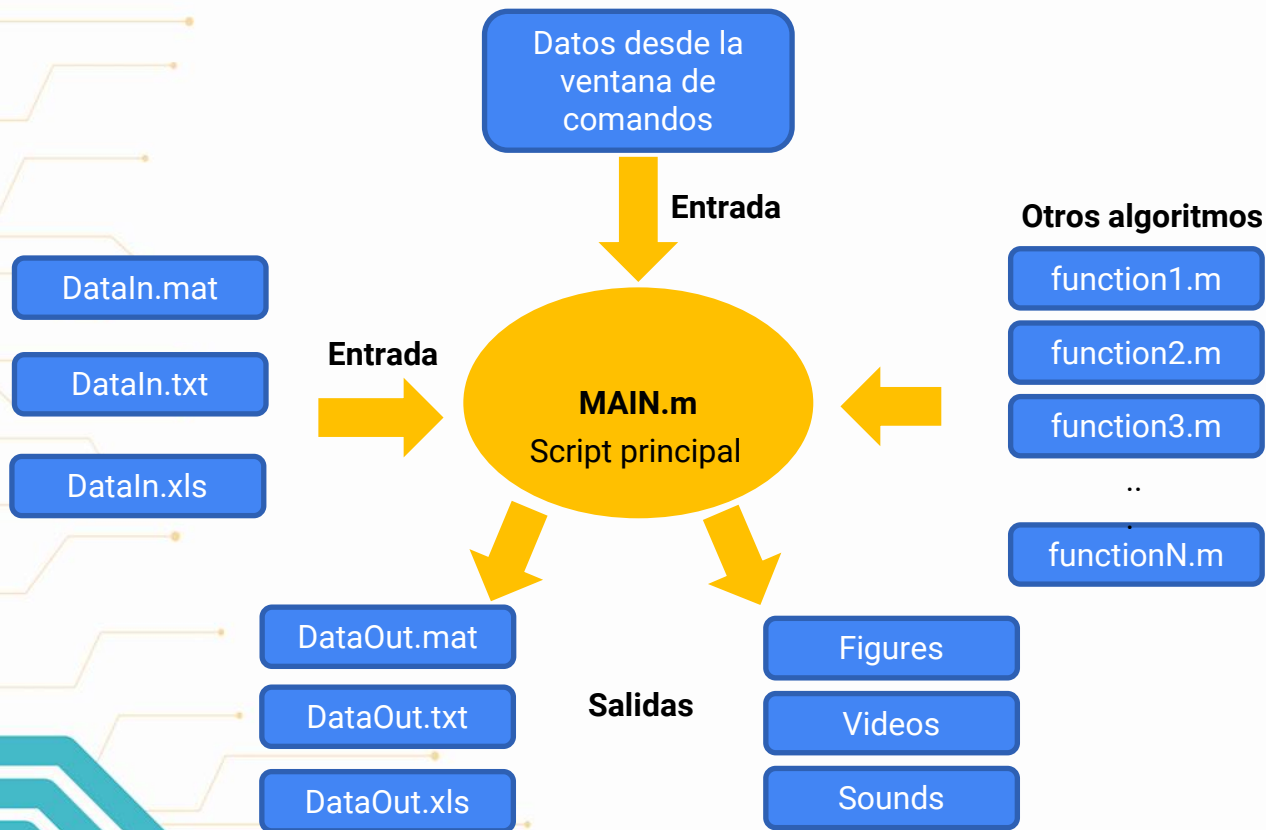
Consiste del nombre del algoritmo, la descripción del problema, de entradas y salidas

Cuerpo del algoritmo

Consiste del cuerpo lógico del algoritmo construido con sentencias de programación y asignaciones.

Nota: los archivos de código en Matlab tienen extensión “.m” y los archivos de datos tienen extensión .mat.

Programando en Matlab



Tanto en *scripts* como en funciones los algoritmos siguen una estructura general:

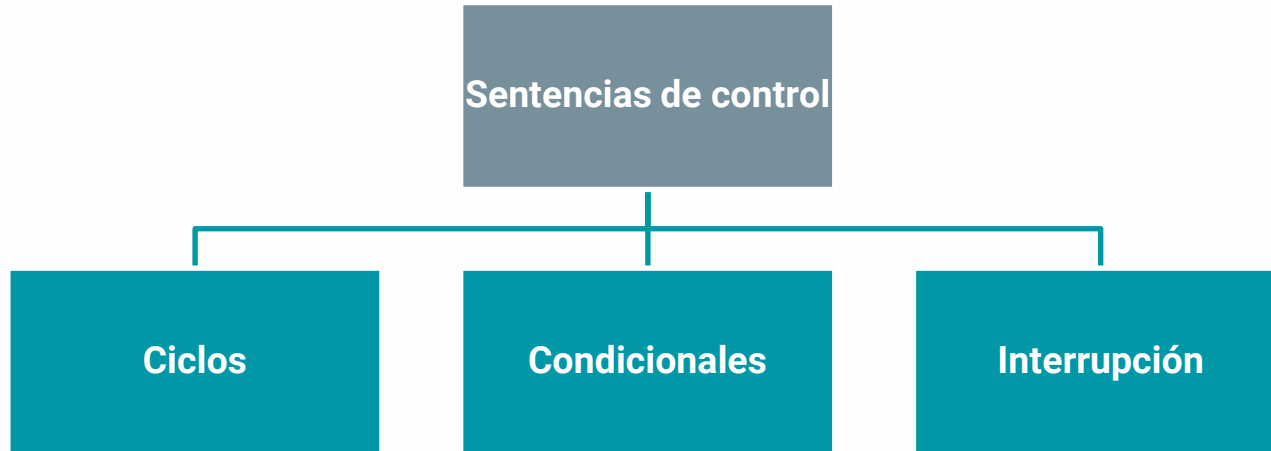
Encabezado del algoritmo

Consiste del nombre del algoritmo, la descripción del problema, de entradas y salidas

Cuerpo del algoritmo

Consiste del cuerpo lógico del algoritmo construido con sentencias de programación y asignaciones.

Programando en Matlab: Sentencias más usadas



Programando en Matlab: ciclos

❑ Ciclo for

Valor inicial de i

Incremento

Valor final de i (última iteración)

Variable índice del
ciclo

```
for i = ini:step:end  
    % operaciones que deseas repetir  
end
```

Ejemplo:

```
for i = 1:5  
    x = i*i;  
end  
x = ?
```

Cuántas iteraciones? se debiera conocer previamente

Programando en Matlab: ciclos

- ❑ **Ciclo while:** Este ciclo es útil cuando se necesita un proceso iterativo, pero se desconoce el número de iteraciones a realizar. Así, las iteraciones continúan mientras se satisface una condición.

```
while (expresion_condicional)

    % operaciones que deseas repetir

end
```

Ejemplo:

```
x = 1
while x <= 10
    x = 3*x
end
```

x = ?

Programando en Matlab: condicionales

- ❑ **Condicional `if ... else`:** Es la sentencia de control del flujo de un código más básica y transversal a los lenguajes de programación. Algunas formas de implementarlo:

```
... código ...
```

```
if (condición 1)
```

```
... código ...
```

```
end
```

```
... código ...
```

```
... código ...
```

```
if (condición 1)
```

```
... código ...
```

```
else
```

```
... código ...
```

```
end
```

```
... código ...
```

```
... código ...
```

```
if (condición 1)
```

```
... código ...
```

```
elseif (condición 2)
```

```
... código ...
```

```
else
```

```
... código ...
```

```
end
```

```
... código ...
```

Programando en Matlab: condicionales

❑ **Condicional if ... else:** Ejemplo basado en la ecuación cuadrática:

```
discr = b*b - 4*a*c;  
if discr < 0  
  
disp('raíces complejas');  
  
end
```

```
discr = b*b - 4*a*c;  
if discr < 0  
  
disp('raíces complejas');  
  
else  
  
disp('raíces reales');  
  
end
```

```
discr = b*b - 4*a*c;  
if discr < 0  
  
disp('raíces complejas');  
  
elseif discr == 0  
  
disp('raíces reales iguales');  
  
else  
  
disp('raíces reales');  
  
end
```

Programando en Matlab: interrupciones

❑ Comandos `break` y `continue`:

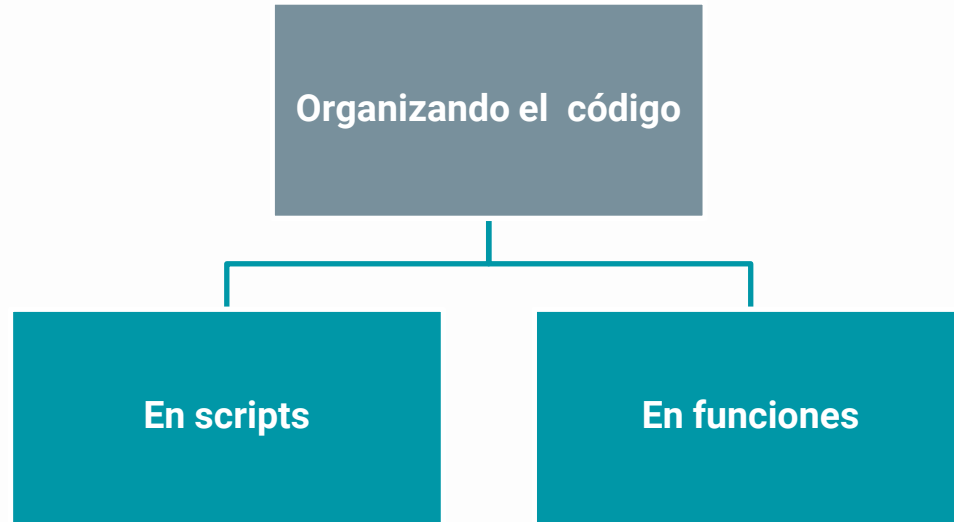
- **`break`**: al aparecer dentro del ciclo (`for` o `while`), este termina la ejecución de dicho ciclo por completo.
- **`continue`**: al aparecer dentro del ciclo (`for` o `while`), este termina la iteración actual y pasa a la siguiente.
- Estos comandos se utilizan generalmente dentro de expresiones condicionales para controlar la ejecución de los ciclos.

Programando en Matlab: interrupciones

❑ Comandos `break` y `continue`. Ejemplo:

```
for i = 1:N
    ... operaciones sobre x ...
    if isnan(x)
        error('Se ejecuto una operación indebida');% desplegamos un mensaje de error
        break %terminamos la ejecución del ciclo for
    end
end
```

Programando en Matlab: Organización



Scripts

❑ *Scripts*

- Archivos .m consistentes de un conjunto de comandos que se ejecutan de forma secuencial.
- Útiles para retener una serie de comandos y operaciones que se quieren repetir más de una vez.
- Se pueden ejecutar escribiendo el nombre del script en la línea de comandos o con el botón Run del menú Editor.
- Siempre escribir sus algoritmos en *Scripts*!
- Los valores que contienen las variables dentro del script se pueden asignar:
 - Directamente dentro del script, por ej. leyendo datos desde alguna fuente.
 - Definiéndolas en la línea de comandos antes de ejecutar el script (no recomendado).
 - Solicitando al usuario ingresar el valor al ejecutar el script, por ejemplo, usando la función `input()`.

Funciones

❑ **Funciones (en archivos apartes al script principal)**

- Archivos .m que comienzan con la palabra reservada **function**.
- Pueden aceptar argumentos de entrada y retornar argumentos de salida.
- La ventaja de una función es que se puede llamar varias veces con diferentes valores.
- Las variables y sus nombres dentro de las funciones solo viven dentro de la función, no aparecerán reflejadas en el espacio de trabajo al ejecutarse.
- **El nombre del archivo debe ser estrictamente igual al de la función.**
- Evitar usar nombres de funciones existentes en Matlab (usar el comando `help`).

Funciones

■ Funciones (en archivos apartes al script principal)

Forma general de una función guardada en un archivo `name.m`

```
function [outarg1, outarg2,...]= name(inarg1, inarg2, ...)  
% comentarios para desplegar con help  
Operaciones ...  
outarg1 = ...;  
outarg2 = ...;  
...
```

- **Ojo**, también se pueden declarar dentro del script principal, útil para no generar más de un archivo. Sólo serán visibles en dicho *script* en este caso.

Ejercicio

Factorial de un número

Escribir una función en Matlab para calcular el factorial de un número entero positivo n , recordar que $n! = 1 \times 2 \times 3 \times \dots \times n, \forall n > 0$. La función debe llamarse `miFactorial` y debe recibir como argumento el entero positivo n . La función debe retornar el valor del factorial de n o `NaN` si el número ingresado no es un entero positivo (recordar que $0! = 1$).

Cuál es el valor de $50!$?

Comparar su resultado con la función de Matlab `factorial()`

Ejercicio

Factorial de un número

```
>> miFactorial(50)
```

```
ans =
```

```
3.0414e+64
```

```
function nf = miFactorial(n)
% nf: factorial del número entero positivo n
% la función comprueba si n > 0 y entero,
% retorna NaN para los casos donde n sea decimal
% ó <0.

if n>1 && ( rem(n,1) == 0 )
    nf = 1;

    for i = 2:n
        nf = i*nf;
    end

elseif n == 0

    nf = 1;

else
    nf = NaN;
    error('debe ingresar un número entero > 0')
end
```

Funciones

❑ Funciones anónimas

- Cuando tenemos expresiones simples, puede ser conveniente definir una función de forma anónima, sin crear un archivo .m. Su forma genérica es:

```
Mi_funcion = @(argumentos) (expresion)
```

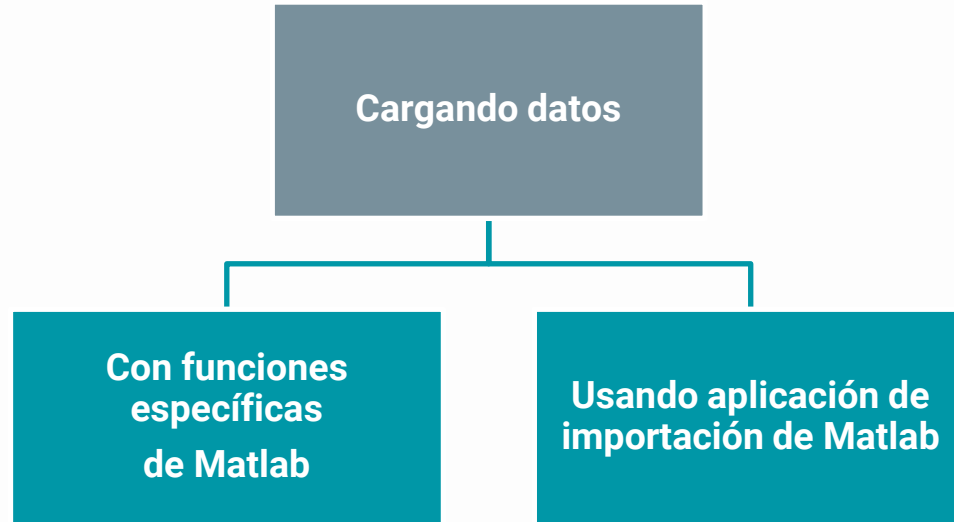
- Ej. crear una función anónima para evaluar $R = \sqrt{1 + e^{-bx/2}}$, con $b = 1$ y $x = 2$.

```
R = @(b,x) (sqrt(1+exp(-b*x/2)));
```

esto crea $R(b, x)$, para evaluarla en los valores de interés:

```
R(1,2)  
ans =  
1.1696
```

Programando en Matlab: Cargando datos



Cargando datos con funciones de Matlab

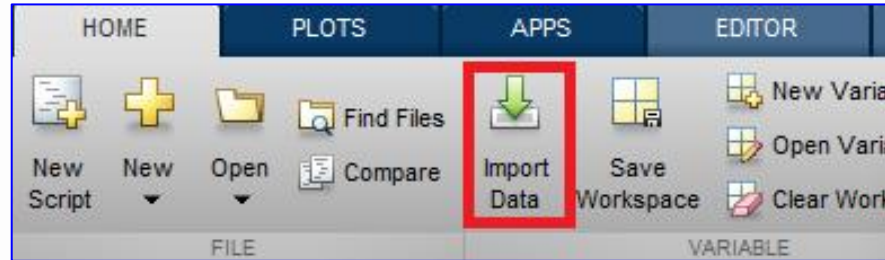
- ❑ Formatos de datos más usados: .mat, .xls, .csv, .txt, .jpg.
- ❑ Formato .mat: Para leer o guardar datos creados en Matlab usar funciones **load** y **save** respectivamente.

```
save NombreArchivo.mat datos1 datos2 ...  
load NombreArchivo.mat
```

- ❑ Formato .xls y .xlsx, .csv: Podemos usar la función **readtable('nombre_archivo')**.

Cargando datos con App. de Matlab

- ❑ Para importar datos también se puede usar la herramienta **Import Data** del menú Home:



- Para abrirla desde la línea de comandos escribir `uiimport`.
- Haciendo doble clic en el nombre del archivo en la ventana del directorio actual, se abre el [Asistente de Importación](#).

OTROS RECURSOS

Formación Online



Formación
online



Excelente recurso para
profundizar en temas
particulares, muchos
de acceso gratuito

Otros Recursos

Examinar los cursos a su ritmo online

Introducción (15)

MATLAB (5)

Simulink (7)

IA, machine learning y deep
learning (5)

Matemáticas y optimización (6)

Procesamiento de imágenes y
señales (5)

Explore más de 50 cursos en clase
presenciales y virtuales

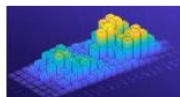
MATLAB



MATLAB Onramp

15 módulos | 2 horas | Idiomas

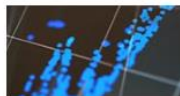
Comience rápidamente con las nociones básicas de MATLAB.



MATLAB Fundamentals

18 módulos | 16.5 horas | Idiomas

Aprenda funcionalidad básica de MATLAB para el análisis de datos, modelado y programación.



MATLAB for Data Processing and Visualization

10 módulos | 8 horas | Idiomas

Cree visualizaciones a la medida y automatice sus tareas de análisis de datos.



MATLAB Programming Techniques

10 módulos | 16 horas | Idiomas

Mejore la solidez, flexibilidad y eficiencia de su código de MATLAB.



Object-Oriented Programming Onramp

4 módulos | 2 horas | Idiomas

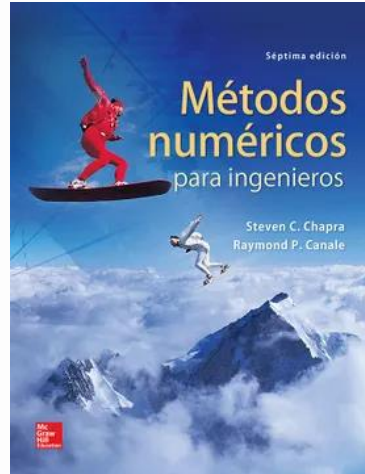
Aprenda los conceptos básicos sobre el uso de la programación orientada a objetos para modelar objetos reales y gestionar la complejidad del software

Otros Recursos

Libros

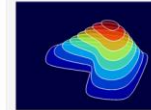
MATLAB® para ingenieros

<http://dea.unsj.edu.ar/control2/matlab%20para%20ingenieros.pdf>



Libros de texto de Cleve Moler

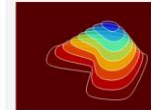
Cleve Moler es el presidente y el científico jefe de The MathWorks. El Sr. Moler fue profesor de matemáticas e informática durante casi 20 años en University of Michigan, Stanford University y University of New Mexico. Además de ser el autor de la primera versión de MATLAB, el Sr. Moler es uno de los autores de las bibliotecas de subrutinas científicas LINPACK y EISPACK. También es coautor de tres libros de texto sobre métodos numéricos.



Numerical Computing with MATLAB

Este dinámico libro de texto escrito por Cleve Moler se ha diseñado para servir como curso introductorio sobre métodos numéricos, MATLAB y el cálculo técnico.

- » Ver el libro de texto
- » Obtener materiales y herramientas curriculares



Experiments with MATLAB

Con un desarrollo activo por parte de Cleve Moler, este libro electrónico consta de capítulos que complementan los cursos de enseñanza secundaria y los primeros cursos universitarios sobre matemáticas y cálculo técnico, lo que incluye el cálculo y la teoría de matrices.

- » Ver el libro electrónico

<https://la.mathworks.com/moler.html>

Otros Recursos

Editor de código en vivo (para ejecución interactiva)

Homework

Use live scripts as the basis for assignments. Give students the live script used in them complete exercises that test their understanding of the material.

Use the techniques described above to complete the following exercises:

Exercise 1: Write MATLAB code to calculate the 3 cube roots of i .

% Put your code here

Exercise 2: Write MATLAB code to calculate the 5 fifth roots of -1 .

% Put your code here

Exercise 3: Describe the mathematical approach you would use to calculate the complex number. Include the equations you used in your approach.

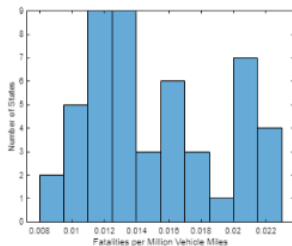
(Describe your approach here)

Distribution of Fatalities

You can include visualizations in your program. Like output, plots and figures appear together with the code that produced them.

We can use a bar chart to see the distribution of fatality rates among the states. There are 11 states that have a fatality rate greater than 0.02 per million vehicle miles.

```
histogram(rate,10)
xlabel('Fatalities per Million Vehicle Miles')
ylabel('Number of States')
```



Panel Radiation and Power Generation For a Single Day

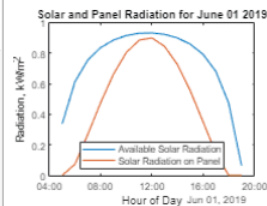
Modify parameters using interactive controls. Display plots together with the code that produced them.

Panel Radiation

For a given day of the year, calculate the total solar radiation and the radiation on the panel. To simplify the analysis, use the panelRadiation function. Try different dates to see how the solar and panel radiation change depending on the time of year.

```
selectedMonth = 'June';
selectedDay = 1;
selectedDate = datetime(2019,selectedMonth,selectedDay);
[times,solarRad,panelRad] = panelRadiation(selectedDate,lambda,phi,UTCoff,tau,beta);

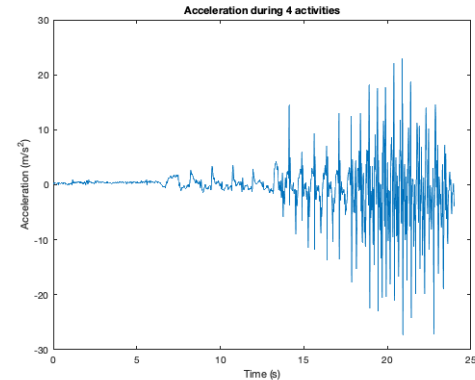
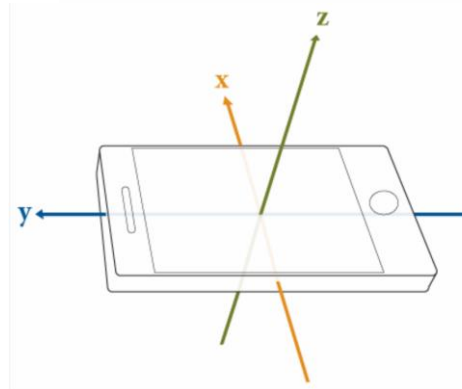
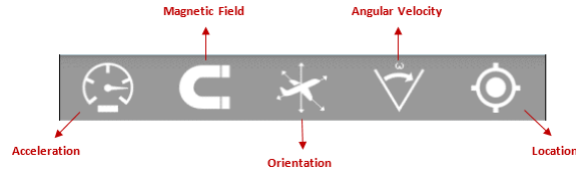
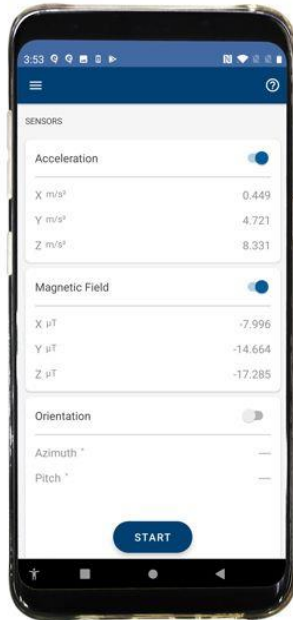
plot(times,solarRad,times,panelRad)
title(['Solar and Panel Radiation for ' datestr(selectedDate,'mmmm dd yyyy')])
xlabel('Hour of Day');
ylabel('Radiation, kW/m^2');
legend('Available Solar Radiation','Solar Radiation on Panel','Location','South')
```



https://la.mathworks.com/help/matlab/live-scripts-and-functions.html?s_tid=CRUX_lftnav

Otros Recursos

Uso de App *Matlab mobile* para capturar información de sensores del celular



Ejemplo aceleración eje x durante 4 actividades: i) estático, ii) caminar, iii) trotar, iv) correr. Ref. [aquí](#).

BIBLIOGRAFÍA

Referencias

- [1] Special characters in Matlab, reference page [Online]. Available: [link](#). [Accessed Apr. 16, 2020].
- [2] Operations precedence reference in Matlab [Online]. Available: [link](#). [Accessed Apr. 16, 2020].
- [3] Users Matlab code exchange center [Online]. Available: [link](#). [Accessed Apr. 16, 2020].
- [4] Stackoverflow, QA resources for developers [Online]. Available: [link](#). [Accessed Apr. 16, 2020].
- [5] Steven Chapra, Applied Numerical Methods with MATLAB for Engineers and Scientists, 11th ed. McGraw-Hill Education , 2015.
- [6] MOOC de Matlab y Octave en Español para profundizar: [link](#).

MUCHAS
GRACIAS