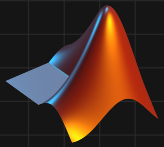


Introducción a MATLAB

Javier Paredes

Universidad de Concepción

Concepción January 19, 2026

índice

¿Qué es MATLAB?

Es un potente software para cálculo numérico y simulación, diseñado específicamente para ingenieros y científicos. Su nombre viene de MATrix LABoratory (Laboratorio de Matrices), ya que todo en MATLAB se trata como una matriz.

¿Qué nos permite hacer?

- Manipular datos y matrices de forma avanzada.
- Visualizar datos y funciones a través de gráficos.
- Implementar algoritmos complejos.
- Crear interfaces de usuario (GUI) interactivas.

¿Comó la usaremos?

Será nuestra herramienta principal para simular procesos, resolver problemas numéricos y graficar datos en cursos como Modelación de Procesos, Sistemas Lineales y Sistemas de Control.

Ventajas sobre otros lenguajes de programación

Ahora bien, MATLAB no es el único programa ni lenguaje de programación que facilita o permite hacer cálculos numéricos; otra opción bien conocida es usar Python con paquetes externos.

Puntos diferenciadores

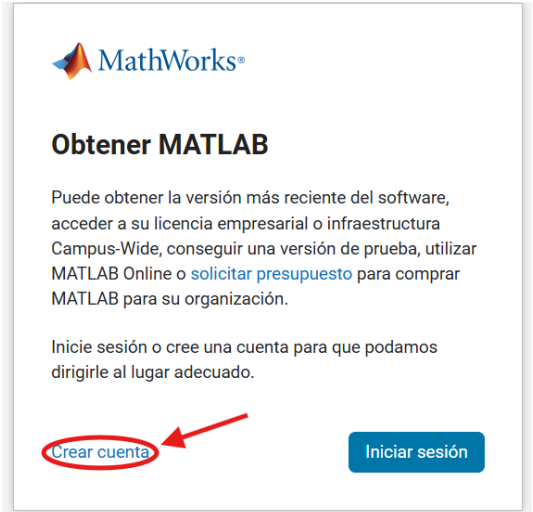
- **Toolboxes Especializados y Validados:** Esta es la mayor ventaja. MATLAB ofrece "cajas de herramientas" (Toolboxes) como la de Sistemas de Control, Procesamiento de Señales o Simscape, que son colecciones de funciones altamente especializadas, probadas y documentadas para resolver problemas de ingeniería específicos.
- **Entorno de Desarrollo Integrado (IDE):** MATLAB ofrece un paquete "todo en uno": editor de código, línea de comandos, visualizador de variables y herramientas para graficar que funcionan juntos a la perfección desde el primer momento.

Instalación MATLAB

La instalación de MATLAB se realiza a través del portal de MathWorks y requiere una conexión a internet. Sigue estos pasos para hacerlo correctamente.

Paso 1: Crear una Cuenta en MathWorks

- Ve al sitio web oficial de MathWorks.
- Crea una cuenta nueva. Es fundamental que uses tu correo electrónico institucional (ej: javparedes2022@udec.cl). Esto es lo que te permitirá acceder a la licencia gratuita de la universidad.


MathWorks®
Obtener MATLAB
Puede obtener la versión más reciente del software, acceder a su licencia empresarial o infraestructura Campus-Wide, conseguir una versión de prueba, utilizar MATLAB Online o solicitar presupuesto para comprar MATLAB para su organización.
Inicie sesión o cree una cuenta para que podamos dirigirle al lugar adecuado.
Crear cuenta
Iniciar sesión

Paso 2: Acceso al Portal MathWorks

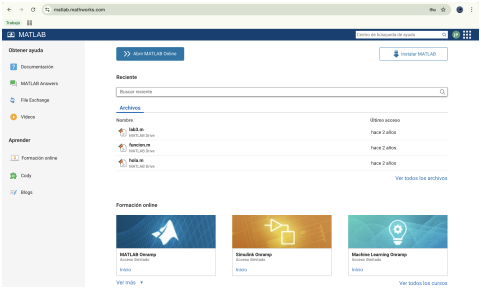
Ingreso Institucional

Para activar la licencia académica (Campus-Wide License):

1. Ir a `matlab.mathworks.com`
2. Iniciar sesión con el correo UdeC (`@udec.cl` o `@alumnos.udec.cl`).
3. Presionar el botón **Instalar MATLAB** (arriba a la derecha).

Nota Importante

No es necesario usar la VPN si se ingresa con las credenciales universitarias correctas.



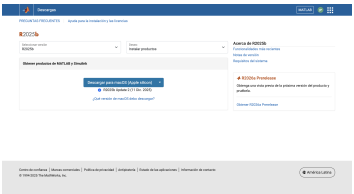
Interfaz de usuario MathWorks (2026)

Paso 3: Descarga y Versión

Versión Actual: R2025b

Se recomienda instalar la última versión estable para asegurar compatibilidad con los laboratorios del curso.

Sistema	Observación Técnica
Windows	Versión estándar de 64-bits.
macOS (Intel)	Macs antiguos (antes de 2020).
macOS (Apple Silicon)	**Importante:** Para chips M1, M2, M3, M4. Es nativo y mucho más rápido.



Paso 4: Instalación de Toolboxes

¡Atención!

El instalador base de MATLAB pesa poco, pero descargará gigabytes según lo que seleccionen. **No instalen todo.**

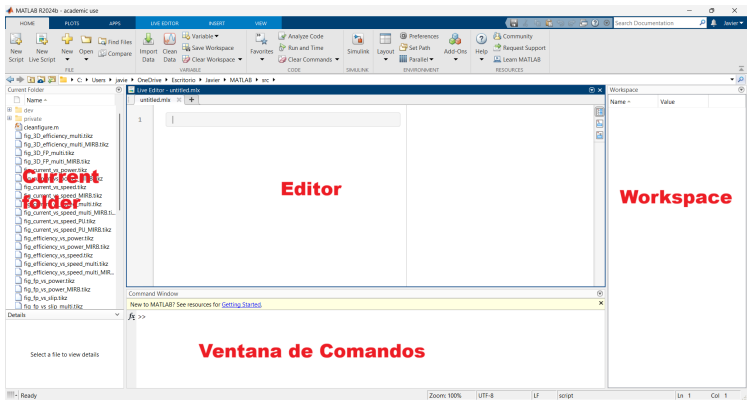
Módulos obligatorios para Ingeniería Civil Eléctrica:

- **MATLAB** (Base)
- **Simulink** (Simulación dinámica)
- **Control System Toolbox** (Bode, Nyquist)
- **Symbolic Math Toolbox** (Cálculo analítico)
- **Simscape Electrical** (Sistemas de Potencia y Máquinas)
- **Signal Processing Toolbox** (Filtros, FFT)
- **Optimization Toolbox** (Optimización)

Nota: Siempre pueden agregar más toolboxes después ejecutando el instalador nuevamente.

Partes clave de la interfaz

Al abrir MATLAB, nos encontramos con su interfaz predeterminada, que se divide en cuatro áreas principales: la Ventana de Comandos, el Workspace, el Current Folder y el Editor. En la siguiente imagen se identifica la ubicación de cada una.



Ahora ya conociendo cada una de las partes del editor, veamos la funcionalidad de cada una de ellas y la mejor configuración que se puede tener en general.

- **Ventana de comandos:** Ejecutar instrucciones de una en una, de forma inmediata.

  Sirve principalmente para hacer cálculos rápidos, probar una función, ver el valor de una variable o ejecutar un script que ya tienes guardado. La reconocerás por el símbolo >> que te indica que está lista para recibir una instrucción.
- **Workspace:** Es la memoria de tu sesión actual en MATLAB. Te permite verificar qué variables tienes disponibles, inspeccionar sus valores sin tener que escribirlos en la Command Window y entender cómo están cambiando tus datos a medida que tu programa se ejecuta.
- **Editor:** Es el lugar donde escribes tus programas. A diferencia de la Command Window, el Editor te permite escribir y guardar secuencias largas de comandos en un archivo (.m o .mlx).

- **Current folder:** Es un explorador de archivos integrado dentro de MATLAB. ESte permite ver donde están los archivos que se van guardando y si en tu código generas un archivo se guarda en esta carpeta.

Podemos personalizar la interfaz de MATLAB para adaptarla a nuestro flujo de trabajo. Por ejemplo, a menudo es útil ocultar la ventana Current Folder para dar más espacio al Editor y a la Ventana de Comandos.

Para ajustar el diseño, sigue estos pasos:

1. En la cinta de opciones, ve a la pestaña HOME.
2. Busca la sección ENVIRONMENT y haz clic en Layout.
3. Aquí puedes desmarcar Current Folder para ocultarla o elegir la configuración que más te acomode.

Survival Skills 1: Documentación

La Regla de Oro

MATLAB tiene la mejor documentación técnica del mundo. **Úsala.**
No necesitas memorizar todo, solo saber buscarlo.

`doc` (El navegador)

Abre la ventana de ayuda completa con ejemplos, teoría y sintaxis.

```
% Abre la biblia de 'plot'
doc plot

% Busca funciones sobre integrales
doc integral
```

`help` (Resumen rápido)

Muestra un resumen de texto rápido en la misma Command Window.

```
% ¿'Qué inputs pide el seno?
help sin

% ¿'Qué hace 'clc'?
help clc
```

Tip: Si no sabes el nombre de la función, busca en Google: "matlab how to..."

Live Scripts (.mlx): Informes Interactivos

¿Qué es un Live Script (.mlx)?

A diferencia del script tradicional (`.m`) que solo contiene código, el **Live Script** combina en un solo documento:

- **Código Ejecutable:** Bloques de MATLAB.
- **Resultados Visuales:** Gráficos y tablas incrustados junto al código.
- **Texto con Formato:** Títulos, ecuaciones ($\LaTeX$), imágenes y explicaciones.

Ventaja Académica

Es el formato estándar moderno (similar a *Jupyter Notebooks*).

Ideal para entregar **laboratorios y tareas** porque permite exportar todo (código + gráfico + análisis) directamente a **PDF** o **Word**.

Flujo de Trabajo

1. Creas un "New Live Script".
2. Escribes tu análisis teórico.
3. Insertas el bloque de código.
4. Ejecutas y ves el gráfico ahí mismo.
5. Guardas o Exportas.

índice

Variables

Asignar variables

En MATLAB, a diferencia de otros lenguajes como C, normalmente no necesitas declarar el tipo de dato de antemano. El sistema según lo que pongas en la variable detecta el tipo de dato, aun así para casos específicos podemos nosotros elegir qué tipo de variable queremos.

Aca un ejemplo de variables:

```
corriente = 5;   % Dato int
voltaje = 5.23;  % Dato double

voltaje_1 = int16(5.23); % Dato double--> entregado como entero
```

Además de los tipos de datos estándar como `int`, `double` o `char`, una de las grandes ventajas de MATLAB para ingeniería es su manejo nativo de números complejos. Podemos definir un número complejo de forma tan intuitiva como `Z = 200 + 6j`, y MATLAB lo entenderá sin problemas.

```
Z = 220 + 6j;   % Dato complex
```

Esto simplifica enormemente las operaciones, ya que podemos manipular conceptos como fasores o impedancias directamente. Con funciones integradas como `abs()` para obtener la magnitud y `angle()`[1] para la fase, nos abstraemos de la matemática subyacente y nos enfocamos en la solución del problema.

Más adelante se adentra en especial en problemas con este tipo de datos y las funciones utiles.

[1] **Importante:** La función `angle()` devuelve el resultado en radianes. Para convertirlo a grados, simplemente usamos `rad2deg(angle())`.

Reglas para nombrar variables

1. **Deben comenzar con una letra:** Válido: `voltaje`, `Resistencia1`, `mi_variable`
2. **Pueden contener letras, números y el guion bajo (_) y no se permiten espacios ni otros caracteres especiales como -, +, *, ñ o tildes:** Válido: `potencia_total_2`, `vector_x` ,por otro lado, Inválido: `potencia-total` (usa guion medio), `año_produccion` (contiene la letra 'ñ').
3. **MATLAB distingue entre mayúsculas y minúsculas:** Esto significa que `Voltaje`, `voltaje` y `VOLTAJE` son tres variables completamente diferentes para MATLAB.
4. **No puedes usar palabras clave reservadas de MATLAB:** Nombres como `if`, `for`, `while`, `end`, o nombres de funciones como `plot`, `sin`, `fft` no se pueden usar como variables.

Clear y clc

Estos son dos comandos no obligatorios pero recomendables al momento de crear un script. Si bien no afectan los resultados, facilitan la interpretación de la salida en MATLAB y previenen errores.

clc: Función que limpia el contenido de la ventana de comandos, permitiendo una visualización más clara de los resultados durante la ejecución.

clear: Función que elimina las variables almacenadas en memoria. Es especialmente útil durante el desarrollo del código, ya que evita que datos de pruebas anteriores afecten los resultados finales. **Recomiendo poner siempre al inicio de un código**

```
clc
clear

corriente = 5;   % Dato int
voltaje = 5.23;  % Dato double
```

Vectores y matrices

Ahora que conocemos los tipos de datos disponibles y cómo almacenarlos en variables, existen situaciones donde necesitamos agrupar múltiples datos del mismo tipo. Para estos casos utilizaremos vectores y matrices.

Vectores

Los vectores son arreglos unidimensionales que pueden ser filas o columnas. Se definen utilizando corchetes `[]`, donde los elementos se separan según su disposición:

- Espacios o comas (,) para elementos en una misma fila
- Punto y coma (;) para separar filas (elementos en columnas)

```
%Vectores

Vector_fila = [1 2 3 4 5]; %Basta con poner un espacio, pero igual sirve
    ↪ ','
Vector_columna = [9; 6; 3; 0]; %Aca es necesario realizarlo con ';'
```

Matrices

Las matrices son arreglos bidimensionales con filas y columnas. Su sintaxis sigue las mismas reglas de separación que los vectores:

```
Matriz_A = [2 3; 4 5];
```

Se mantiene la misma convención de separadores:

- Espacios o comas (,) entre elementos de una misma fila
- Punto y coma (;) para separar diferentes filas

Un detalle **importante** es que dentro de los vectores y matrices podemos poner las variables creadas. **(hay que tener cuidado con las dimenciones)**

```
Matriz = [corriente voltaje ; 45 3];
```

Acceso a elementos (Indexación)

La indexación de vectores y matrices sigue un principio similar. Para acceder a elementos, se utiliza el nombre de la variable seguido de paréntesis que contienen las posiciones o rangos deseados.

- **Elemento individual:**
  - Vector: posición única (`x`)
  - Matriz: coordenadas (`fila,columna`)
- **Fila o columna completa:**
  - Fila: (`x,:`)
  - Columna: (`:,y`)
- **Submatriz:** rango de filas y columnas (`x1:x2,y1:y2`)

El siguiente ejemplo ilustra estos conceptos:

```
M = [ 1,  2,  3,  4;
      5,  6,  7,  8;
      9, 10, 11, 12;
     13, 14, 15, 16 ];

%M(2,3) ----> 7 (fila 2 columna 3)
%M(3,:) ----> [9, 10, 11, 12] (fila 3)
%M(3:4,1:2) ->  [9, 10;
%               13, 14;]  (fila 3 a 4, columna 1 a 2)
```

Suma y resta

Requisito: Las matrices deben tener las mismas dimensiones
Sintaxis:

- Suma: `A + B` y Resta: `A - B`
- **Operación elemento a elemento:**

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

Suma o resta cada uno los elementos con el de su misma posición.

Multiplicación de matrices vs. elemento a elemento

Multiplicación matricial (*)

- Producto punto (álgebra lineal)
- **Requiere:** # columnas A = # filas B
- **Sintaxis:** `A * B`
- **Ejemplo:**
  `[1 2; 3 4] * [1; 2] = [1*1 + 2*2; 3*1 + 4*2] = [5; 11]`

Multiplicación elemento a elemento (.*)

- Producto Hadamard
- **Requiere:** mismas dimensiones
- **Sintaxis:** `A .* B`
- **Ejemplo:**
  `[1 2; 3 4] .* [2 2; 2 2] = [2 4; 6 8]`

Funciones útiles para matrices

size()

- Devuelve las dimensiones de una matriz
- Sintaxis: `[filas, columnas] = size(A)`
- Ejemplo:

```
>> size([1 2 3; 4 5 6])
ans =
    2    3
```

length()

- Devuelve la dimensión más larga (filas o columnas)
- Ejemplo:

```
>> length([1 2 3; 4 5 6])
ans =
    3
```

Transposición (' o transpose())

- Intercambia filas por columnas
- Dos sintaxis equivalentes:
  - `A'` (operador apóstrofe)
  - `transpose(A)`
- Ejemplo:

```
>> [1 2; 3 4]'
ans =
     1     3
     2     4
```

¡Recuerda!

La diferencia clave entre:

- `*` vs `.*`
- `^` vs `.^` (potencia)
- `/` (**opción como inversa de matriz**) vs `./` (división)

índice

Condicional if

En este caso, vamos a comenzar con la estructura `if`. Esta instrucción sirve para tomar decisiones entre diferentes casos. Al igual que en la mayoría de los lenguajes de programación, permite utilizar un `else` para manejar alternativas.

Sintaxis general

```
% if condicion1
    % % Bloque de codigo 1
% elseif condicion2
    % % Bloque de codigo 2
% else
    % % Bloque de codigo 3
% end
```

Al momento de utilizar estructuras `if`, es necesario definir condiciones que determinen su ejecución. Para esto contamos con dos tipos de operadores:

- **Operadores de comparación:** Permiten evaluar relaciones entre valores
- **Operadores lógicos:** Permiten combinar múltiples condiciones

Los primeros activan el bloque de código cuando se cumple una relación específica con un valor, mientras que los segundos permiten integrar y tratar varias condiciones simultáneamente.

Comparación

Operador	Descripción
==	Igual a
≅	No igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Lógicos

Operador	Descripción
&	Y lógico (and)
|	O lógico (or)
~	No lógico (not)

Bucle for

El bucle `for` es una estructura fundamental presente en la mayoría de lenguajes de programación. Su función principal es repetir un bloque de código un número específico de veces, determinado por un índice. La implementación en MATLAB sigue la misma lógica que otros lenguajes, variando principalmente en su sintaxis.

Sintaxis general

```
% t = [1,2,3];
%
% for i=1:length(t)
    % %proceso
% end
```

En este caso el `%proceso` va ser repetido 3 veces por el largo de `t`.

Ejemplos de uso de if y for

Crear señales

En sistemas de control y procesamiento de señales, es común trabajar con señales que se activan bajo ciertas condiciones. El uso combinado de bucles `for` y estructuras `if` es fundamental para modelar correctamente este tipo de señales. A continuación, analizaremos un ejemplo práctico utilizando una señal escalón.

```
t = linspace(0,2,20); %crea un vector con limites [0 2] y con 20 puntos
    ↪ dentro de este rango
u = zeros(size(t)); %creo un vector del mismo tamaño 't' en este caso
    ↪ todos los valores van a ser 0
```

```
for i=1:length(t)
    if i<=10
        u(i) = 1; % RECORDAR: u(i), 'i' va a indicar el elemento que se
    ↪ esta cambiando
    end
end
```

```
>> u
ans =
    0   0   0   0   0   0   0   0   0   1   1   1   1   1   1   1   1   1   1   1
```

Ecuaciones diferenciales en sistemas

Para modelar la evolución temporal de variables en procesos y sistemas, se utilizan ecuaciones diferenciales. Estas ecuaciones permiten describir cómo cambian las variables de estado en el tiempo.

Consideremos un sistema masa-resorte-amortiguador:

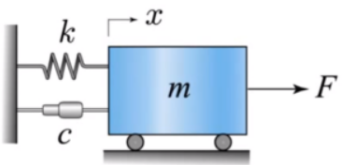


Figure: Sistema mecánico masa-resorte-amortiguador

Representación matemática y solución

Ecuaciones de estado

El sistema se representa mediante la siguiente ecuación diferencial de segundo orden:

$$m\,\ddot{x} + c\,\dot{x} + k\,x = F$$

Que puede transformarse en un sistema de ecuaciones de primer orden (forma de espacio de estado):

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\dfrac{k}{m}x_1 - \dfrac{c}{m}x_2 + \dfrac{1}{m}F \end{bmatrix}$$

donde $x_1 = x$ (posición) y $x_2 = \dot{x}$ (velocidad).

Solución numérica

Para resolver estas ecuaciones, utilizaremos métodos numérico (euler) implementados con bucles `for` en MATLAB. La representación matricial facilita la implementación computacional mediante integración numérica.

```
% Parámetros del sistema
m = 5;      % Masa (kg)
c = 1;      % Coeficiente de amortiguamiento (N*s/m)
k = 2;      % Constante del resorte (N/m)
F = u;      % Fuerza externa constante (N)

% Condiciones iniciales
x1_0 = 0;  % Posición inicial (m)
x2_0 = 0;  % Velocidad inicial (m/s)

t = linspace(0,2,20);
dt = 2/20;

% Pre-asignación de memoria
x1 = zeros(size(t));
x2 = zeros(size(t));
```

```
% Establecer condiciones iniciales
x1(1) = x1_0;
x2(1) = x2_0;

% Método de Euler para resolver el sistema
for i = 1:length(t)-1
    % Ecuaciones de estado:
    dx1 = x2(i);  % dx1/dt = x2
    dx2 = (1/m)*(F(i) - k*x1(i) - c*x2(i));  % dx2/dt = (F - kx1 - cx2)/m

    % Actualización Euler
    x1(i+1) = x1(i) + dt * dx1;
    x2(i+1) = x2(i) + dt * dx2;
end
```

Función básica: plot(x,y)

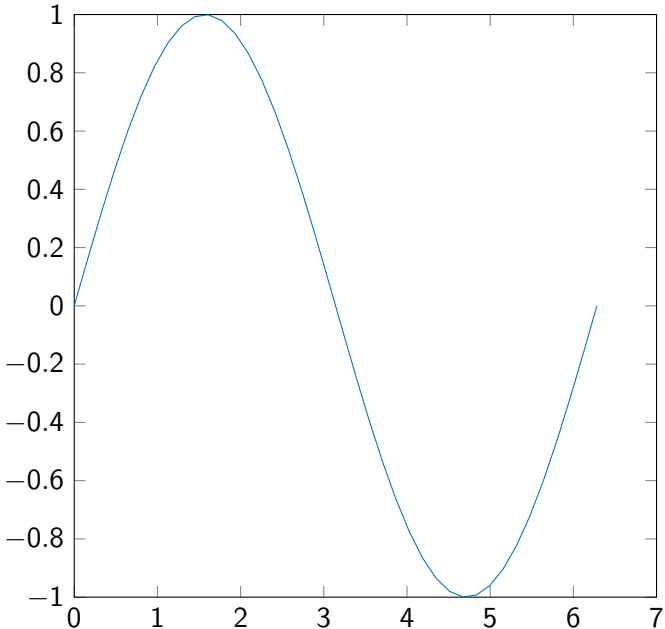
Este va a ser el gráfico más recurrente de usar, ya que permite hacer cualquier gráfico típicamente usado.

- Crea un gráfico 2D con vectores `X` e `Y`
- Requiere dos vectores de igual longitud
- Conecta los puntos con líneas rectas

Veamos un ejemplo:

```
x = linspace(0,2*pi,40);
y = sin(x);

plot(x,y);
```


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7

Personalizacion - Elementos básicos

- `title('Título')`: Añade título
- `xlabel('Etiqueta X')`: Etiqueta eje X
- `ylabel('Etiqueta Y')`: Etiqueta eje Y
- `legend('Serie 1', 'Serie 2')`: Crea leyenda

Estilos de línea

Sintaxis: `plot(x,y,'opciones')`

- Color: 'r' (rojo), 'b' (azul), 'g' (verde)
- Estilo: '-' (sólida), '–' (discontinua), ':' (punteada)
- Marcador: 'o' (círculos), '*' (estrellas), 's' (cuadrados)
- Ejemplo: `plot(x,y,'r--o')`

Veamos como queda el mismo gráfico aplicando estas personalizaciones

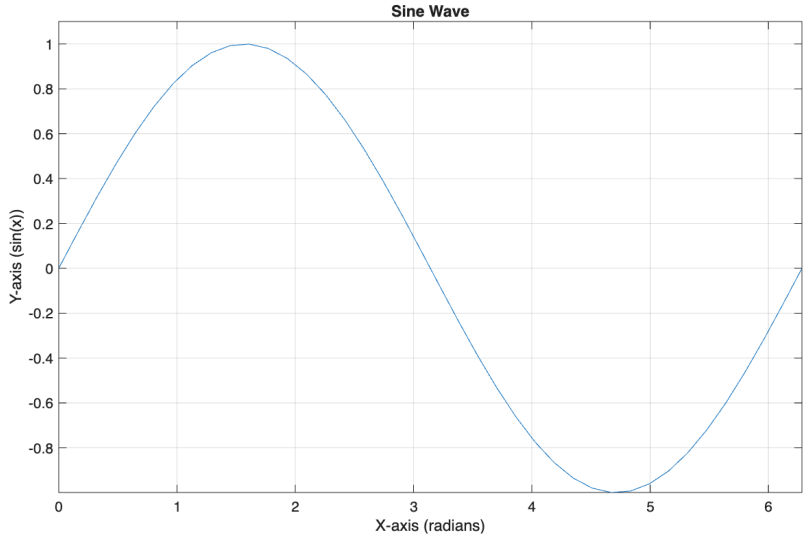


Figure: Gráfico con configuraciones de personalización

Consejos profesionales

- Siempre etiquetar ejes con unidades
- Usar títulos descriptivos pero concisos
- Limitar leyendas a 4-5 elementos máximo
- Elegir colores con buen contraste
- Usar tamaño de fuente legible (12pt mínimo)

Comandos útiles adicionales

- `figure`: Crear nueva ventana de figura
- `subplot`: Crear múltiples gráficos en una figura
- `grid on`: Activar cuadrícula
- `axis equal`: Escalar ejes igualmente
- `xlim([min max])`: Limitar rango del eje X
- `ylim([min max])`: Limitar rango del eje Y

Gráficos Especializados en MATLAB

Comandos por Dominio

stem **T. Discreto:**
Ideal para secuencias $x[n]$.

semilogx **Frecuencia:**
Eje x log (Vital para Bode).

polarplot **Fasorial:**
Magnitud y fase $(\rho, \angle\theta)$.

Sintaxis

```
n = 0:10;
x = 0.9.^n;

% Plot
stem(n,x,'filled');
grid on;
xlabel('n');
ylabel('Amp');
```

Aplicaciones en Ingeniería

- **Control Digital:** Visualización en el plano Z (Estabilidad).
- **Sistemas de Control:** Respuesta en frecuencia (Bode).
- **Redes Eléctricas:** Diagramas fasoriales (V, I).

Importación de Datos Externos

Comandos Modernos (Recomendados)

Desde la versión R2019b, se estandarizó el uso de:

- `readmatrix('archivo')`: Para datos numéricos homogéneos.
- `readtable('archivo')`: Para datos mixtos (texto y números) con encabezados.

Ejemplo 1: Excel (.xlsx)

```
% Importar datos de sensores
% Hoja 1, Rango A2:C100
data = readmatrix('mediciones.xlsx',...
       'Range','A2:C100');

tiempo = data(:,1);
voltaje = data(:,2);
```

Ejemplo 2: CSV

```
% Importar datos de osciloscopio
M = readmatrix('scope.csv');

% Eliminar filas con NaN
M = rmmissing(M);
plot(M(:,1), M(:,2));
```

Exportación y Almacenamiento de Datos

Exportar a Excel/CSV

Ideal para informes o compartir datos.

```
t = 0:0.1:10;
y = sin(t)'; % Columna

% Crear matriz unificada
datos_finales = [t', y];

% Exportar a Excel
writematrix(datos_finales,...
    'resultados.xlsx',...
    'Sheet','Simulacion');
```

Archivos Binarios (.mat)

La forma más eficiente de guardar variables de MATLAB.

```
% Guardar TODO el workspace
save('proyecto_control.mat');

% Guardar variables especificas
Kp = 1.5; Ki = 0.5;
save('params.mat', 'Kp', 'Ki');

% Cargar posteriormente
load('params.mat');
```

Comando	Formato	Uso Principal
`writematrix`	.csv, .txt, .xlsx	Intercambio con otros software
`save/load`	.mat (Binario)	Guardar estado de simulación

Manejo de Imágenes (Señales 2D)

La Imagen como Matriz

MATLAB interpreta las imágenes como matrices:

- **RGB:** 3D ($M \times N \times 3$). R, G, B.
- **Grayscale:** 2D ($M \times N$). 0 a 255.

Tip

Las imágenes cargan como `uint8`. Para operar matemáticamente, convierte a `double`.

Código: Carga y Procesamiento

```
% 1. Lectura y Conversión
img_rgb = imread('placa.jpg');      % Carga imagen RGB (3D)
img_gray = rgb2gray(img_rgb);       % Convierte a Grises (2D)

% 2. Visualización comparativa (Subplot)
figure;
subplot(1,2,1); imshow(img_rgb);  title('Original RGB');
subplot(1,2,2); imshow(img_gray); title('Escala de Grises');
```

Importación de Audio (Señales 1D)

Lectura y Teorema de Muestreo

El comando `audioread` devuelve dos variables críticas:

- **y**: El vector de datos (amplitud de la señal).
- **F_s**: La frecuencia de muestreo (ej. 44100 Hz).

Reproducción

```
[y, Fs] = audioread('voz.wav');

% Reproducir sonido
sound(y, Fs);

% Información básica
info = audioinfo('voz.wav');
disp(info.Duration); % Segundos
```

Visualización en Tiempo

Crucial: Construir el vector de tiempo real.

```
% N = número total de muestras
N = length(y);

% Vector de tiempo (0 a T final)
t = (0 : N-1) / Fs;

plot(t, y);
xlabel('Tiempo [s]');
ylabel('Amplitud');
```

índice

Extensiones Esenciales de MATLAB

Aunque MATLAB es potente por sí mismo, ciertas extensiones (toolboxes) simplifican significativamente tareas específicas en ingeniería:

- **Control System Toolbox**:
  - Funcionalidad para análisis y diseño de sistemas de control
  - Funciones clave: `tf` (función de transferencia), `bode`, `rlocus`
  - Beneficios: Análisis rápido de estabilidad, respuesta en frecuencia, lugar de las raíces
- **Signal Processing Toolbox**:
  - Herramientas especializadas para procesamiento de señales
  - Optimiza operaciones como FFT, filtrado, análisis espectral
  - Funciones destacadas: `fft`, `spectrogram`, `designfilt`
- **Symbolic Math Toolbox**:
  - Cálculo simbólico para problemas matemáticos complejos
  - Resuelve ecuaciones lineales y no lineales analíticamente
  - Funcionalidad única: Manipulación algebraica, derivación simbólica, cálculo integral

Fin