Introducción MATLAB

Modelos de la computación

Francisco Fernández Navarro

Departamento de Lenguajes y Ciencias de la Computación
Área: Ciencias de la Computación e Inteligencia Artificial

Índice de contenidos

¿Qué es MATLAB?

- MATLAB es un lenguaje de programación de alto nivel.
- Desarrollado por MathWorks, es ampliamente utilizado en ingeniería, ciencias y matemáticas.
- Proporciona un entorno interactivo para el cálculo numérico, el análisis de datos y la visualización.
- Ofrece una amplia gama de herramientas y funciones predefinidas para tareas comunes.
- Se caracteriza por su facilidad de uso y su capacidad para resolver problemas complejos de manera eficiente.
- Es multiplataforma y compatible con sistemas Windows, macOS y Linux.

Ventajas de MATLAB

- Entorno Interactivo: MATLAB proporciona un entorno interactivo que facilita la experimentación y el aprendizaje.
- Amplia Biblioteca de Funciones: Cuenta con una amplia variedad de funciones predefinidas para álgebra lineal, análisis numérico, procesamiento de señales, estadísticas y más.
- Visualización Avanzada: Permite crear gráficos de alta calidad y visualizar datos de manera efectiva.
- Comunidad Activa: Existe una comunidad global de usuarios y abundante documentación en línea.
- Integración con Otros Lenguajes: Es posible integrar MATLAB con otros lenguajes de programación, como C/C++ y Python.

Índice de contenidos

Variables y Asignación en MATLAB

Variables

En MATLAB, las variables se utilizan para almacenar datos. Algunas consideraciones importantes:

- Los nombres de las variables son sensibles a mayúsculas y minúsculas.
- MATLAB es dinámico en cuanto a tipos de datos, lo que significa que no es necesario declarar el tipo de variable de antemano.
- Para asignar un valor a una variable, utiliza el operador de asignación `=`.

```
x = 10;
nombre = 'Juan';
altura = 1.75;
```

Operadores Aritméticos en MATLAB

Operadores Aritméticos Básicos

MATLAB admite los siguientes operadores aritméticos básicos:

- Suma: `+`
- Resta: `-`
- Multiplicación: `*`
- División: `/`
- Potencia: `^`

```
a = 5;
b = 2;
suma = a + b;
resta = a - b;
producto = a * b;
division = a / b;
potencia = a^b;
```

Funciones Incorporadas en MATLAB

Funciones Incorporadas

MATLAB ofrece una amplia gama de funciones incorporadas para realizar diversas operaciones. Algunos ejemplos comunes incluyen:

- `sin(x)`: Calcula el seno de un número y `cos(x)`: Calcula el coseno de un número.
- `sqrt(x)`: Calcula la raíz cuadrada de un número.
- `exp(x)`: Calcula la función exponencial de un número y `log(x)`: Calcula el logaritmo natural de un número.

```
x = 2.0;
senox = sin(x);
cosenox = cos(x);
raizx = sqrt(x);
exponencialx = exp(x);
logaritmox = log(x);
```

Estructuras de Control en MATLAB

```
x = 3; n = 1;
% Estructura if
if x > 0
  disp('x es positivo');
else
  disp('x no es positivo');
end

% Estructura for
for i = 1:5
  disp(['It: ', num2str(i)]);
end

% Estructura while
while n < 10
  n = n + 1;
end
```

Comentarios y Buenas Prácticas en MATLAB

Comentarios

Los comentarios son una parte importante de la programación en MATLAB. Se utilizan para:

- Explicar el propósito de tu código.
- Hacer que tu código sea más legible.

```
% Este es un comentario de una sola linea

%{
  Este es un comentario de multiples lineas.
  Es util para explicar secciones largas de codigo.
%}
```

Índice de contenidos

Vectores y Matrices en MATLAB

Vectores y matrices

- Los vectores son conjuntos de elementos organizados en una sola dimensión. Pueden ser: (i) Vectores fila: una sola fila de elementos o (ii) Vectores columna: una sola columna de elementos.
- Las matrices son conjuntos de elementos organizados en dos dimensiones (filas y columnas). Se pueden utilizar para representar datos tabulares, sistemas de ecuaciones y más.

```
% Vector fila
v_fila = [1, 2, 3, 4, 5];
% Vector columna
v_columna = [1; 2; 3; 4; 5];
% Matriz
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
```

Indexación y Slicing en MATLAB

Indexación

En MATLAB, puedes acceder a elementos específicos de un vector o matriz utilizando la indexación. La indexación comienza en 1.

Slicing

El slicing te permite acceder a una parte específica de un vector o matriz. Puedes seleccionar un rango de elementos.

```
% Vectores
vector = [1, 2, 3, 4, 5];
% Acceder a un elemento especifico
elemento = vector(3); % elemento es igual a 3
% Slicing
subvector = vector(2:4); % subvector es igual a [2, 3, 4]
% Matrices
matriz = [1, 2, 3; 4, 5, 6; 7, 8, 9];
% Acceder a un elemento especifico
elementomatriz = matriz(2, 3); % elementoMatriz es igual a 6
% Slicing en filas y columnas
submatriz = matriz(1:2, 2:3); % submatriz es igual a [2, 3; 5, 6]
```

Funciones de Manipulación de Datos en MATLAB

- ‘length‘: Devuelve la longitud de un vector.
- ‘size‘: Devuelve las dimensiones de una matriz.
- ‘max‘ y ‘min‘: Encuentran el valor máximo y mínimo en un conjunto de datos.
- ‘sum‘ y ‘mean‘: Calculan la suma y el promedio de elementos.

```
% Vectores
vector = [1, 2, 3, 4, 5];
% Longitud de un vector
longitud = length(vector); % longitud es igual a 5
% Matrices
matriz = [1, 2, 3; 4, 5, 6; 7, 8, 9];
% Dimensiones de una matriz
dimensiones = size(matriz); % dimensiones es igual a [3, 3]
% Valor maximo y minimo
maximo_valor = max(vector); % maximo_valor es igual a 5
minimo_valor = min(vector); % minimo_valor es igual a 1
% Suma y promedio
suma_total = sum(vector); % suma_total es igual a 15
promedio = mean(vector); % promedio es igual a 3
```

Funciones de Manipulación de Datos en MATLAB

Importante

Las funciones aplicadas a matrices devolverán el resultado por columnas

```
% Matrices
matriz = [1, 2, 3; 4, 5, 6; 7, 8, 9];
% Dimensiones de una matriz
dimensiones = size(matriz); % dimensiones es igual a [3, 3]
% Valor maximo y minimo
maximo_valor = max(matriz); % maximo_valor es igual a [7, 8, 9]
minimo_valor = min(matriz); % minimo_valor es igual a [1, 2, 3]
% Suma y promedio
suma_total = sum(matriz); % suma_total es igual a [12, 15, 18]
promedio = mean(matriz); % promedio es igual a [4,5,6]
```

Gráficos Básicos en MATLAB (plot)

Gráficos con la función ‘plot‘

En MATLAB, la función **plot** se utiliza para crear gráficos. Puedes visualizar datos de una manera sencilla y personalizada.

```
% Datos de ejemplo
x = linspace(0, 2*pi, 100); % Crear un vector de valores x
y = sin(x); % Calcular los valores de y usando la funcion seno
% Crear un grafico
plot(x, y, 'r--', 'LineWidth', 2); % 'r--' es para linea roja
    discontinua
% Etiquetas y titulo
xlabel('Eje X'); ylabel('Eje Y');
title('Plot de la Func Seno');
% Mostrar la cuadricula
grid on;
% Annadir leyenda
legend('sin(x)');
```

Ejemplo de Gráficos en MATLAB

Gráficos 2D

MATLAB ofrece una amplia gama de herramientas para crear gráficos 2D. Puedes personalizar la apariencia de tus gráficos según tus necesidades.

```
% Datos de ejemplo
x = linspace(0, 2*pi, 100); % Crear un vector de valores x
y1 = sin(x); y2 = cos(x);
% Crear un grafico
figure; % Crear una nueva figura
plot(x, y1, 'r--', 'LineWidth', 2); % Plot de sin(x)
hold on; % Mantener el plot actual
plot(x, y2, 'b-', 'LineWidth', 2); % Plot de cos(x)
% Etiquetas y titulo
xlabel('Eje X');
ylabel('Eje Y');
title('Plot de las Funciones Seno y Coseno');
% Mostrar la cuadricula
grid on;
% Annadir leyenda
legend('sin(x)', 'cos(x)');
```

Índice de contenidos

Funciones en MATLAB

Funciones

En MATLAB, puedes definir tus propias funciones para reutilizar código y realizar tareas específicas.

```
% Def. de una funcion
function resultado = cuadrado(x)
    resultado = x^2;
end

% Uso de la funcion
valor = 5;
resultado = cuadrado(valor);
% Visualizar el resultado
fprintf('El cuadrado de %d es %d\n', valor, resultado);
```

Pasaje de Argumentos en MATLAB

Pasaje de Argumentos

En MATLAB, puedes pasar argumentos a funciones personalizadas que hayas definido. Esto te permite realizar cálculos o acciones específicas en función de los valores proporcionados.

```
% Def de una func que toma argumentos
function resultado = suma(a, b)
    resultado = a + b;
end

% Uso de la func con argumentos
n1 = 5;
n2 = 3;
resultado_suma = suma(n1, n2);
% Mostrar el resultado
fprintf('La suma de %d y %d es %d\n', n1, n2, resultado_suma);
```

Retorno de Múltiples Valores en MATLAB

Retorno de Múltiples Valores

En MATLAB, las funciones pueden devolver no solo un valor, sino también múltiples valores calculados o procesados. Esto es útil cuando necesitas obtener varios resultados de una función.

```
% Def. de una func. que retorna multiples valores
function [suma, resta, producto] = operaciones_basicas(a, b)
    suma = a + b;
    resta = a - b;
    producto = a * b;
end

numero1 = 8;
numero2 = 3;
[resultado_suma, resultado_resta, resultado_producto] =
    operaciones_basicas(numero1, numero2);
% Mostrar los resultados
fprintf('Suma: %d\n', resultado_suma);
fprintf('Resta: %d\n', resultado_resta);
fprintf('Producto: %d\n', resultado_producto);
```

Manejo de Errores en MATLAB

Manejo de Errores

En MATLAB, puedes utilizar el manejo de errores para lidiar con situaciones inesperadas o excepcionales que puedan ocurrir durante la ejecución de tu código.

```
try
    % Intenta abrir un archivo que no existe
    archivo = fopen('archivo_inexistente.txt', 'r');
    % Operaciones con el archivo (esto generara un error)
    fprintf(archivo, 'Hola, mundo');
    fclose(archivo);
catch ME
    % Captura el objeto de excepcion ME y muestra un mensaje de
        error
    fprintf('Error: %s\n', ME.message);
end

% Continua con el resto del codigo
fprintf('El programa sigue ejecutandose...\n');
```

Estructuras de Datos (Structs) en MATLAB

Estructuras de Datos (Structs)

En MATLAB, una estructura de datos, comúnmente conocida como "struct", te permite agrupar diferentes tipos de datos en una sola entidad. Cada campo de la estructura puede contener datos de diferentes tipos.

```
% Def. de una estructura (struct)
persona.nombre = 'Juan';
persona.edad = 30;
persona.ciudad = 'Ciudad de Mexico';

% Acceso a los campos de la estructura
nombre_persona = persona.nombre;
edad_persona = persona.edad;
ciudad_persona = persona.ciudad;

fprintf('Nombre: %s\n', nombre_persona);
fprintf('Edad: %d\n', edad_persona);
fprintf('Ciudad: %s\n', ciudad_persona);
```

Archivos y lectura/escritura

Archivos de Datos

MATLAB te permite trabajar con archivos para leer y escribir datos. Esto es útil para procesar información desde y hacia archivos externos.

```
% Archivo de lectura
archivo_lectura = fopen('datos_entrada.txt', 'r');
% Leer datos desde el archivo
datos = fscanf(archivo_lectura, '%f');
fclose(archivo_lectura);

% Archivo de escritura
archivo_escritura = fopen('datos_salida.txt', 'w');
% Realizar operaciones y escribir resultados
resultado = sum(datos);
fprintf(archivo_escritura, 'El resultado es: %.2f\n', resultado);
fclose(archivo_escritura);
```

Ejemplos de Aplicaciones en MATLAB

Ejemplo 1: Procesamiento de Imágenes

```
% Cargar una imagen
imagen = imread('imagen.jpg');

% Aplicar un filtro de suavizado
imagen_suavizada = imgaussfilt(imagen, 2);

% Mostrar la imagen original y la imagen suavizada
figure;
subplot(1,2,1);
imshow(imagen);
title('Imagen Original');

subplot(1,2,2);
imshow(imagen_suavizada);
title('Imagen Suavizada');
```

Ejemplos de Aplicaciones en MATLAB

Ejemplo 2: Simulación de Sistemas

```
% Definir un modelo de sistema
s = tf('s');
modelo = 1 / (s^2 + 2*s + 1);

% Simular la respuesta a una entrada
t = 0:0.01:5;
entrada = sin(t);
respuesta = lsim(modelo, entrada, t);
% Graficar la respuesta del sistema
figure;
plot(t, respuesta); xlabel('Tiempo'); ylabel('Respuesta');
title('Respuesta del Sistema a una Entrada Senoidal');
```

Índice de contenidos

Ejercicio 1

Calcular la suma de los números pares del 1 al 100.

Ejercicio 1

Calcular la suma de los números pares del 1 al 100.

```
% Inicializar la variable de suma
suma = 0;
% Recorrer los numeros del 1 al 100
for num = 1:100
    % Verificar si el num es par
    if rem(num, 2) == 0
        % Agregar el num par a la suma
        suma = suma + num;
    end
end

% Mostrar el resultado
fprintf('La suma de los num pares del 1 al 100 es: %d\n', suma);
```

Ejercicio 2

Calcular la suma de los cuadrados de los primeros 10 números naturales.

Ejercicio 2

Calcular la suma de los cuadrados de los primeros 10 números naturales.

```
% Inicializar la variable de suma
suma_cuadrados = 0;

% Calcular la suma de los cuadrados
for num = 1:10
    cuadrado = num^2;
    suma_cuadrados = suma_cuadrados + cuadrado;
end

% Mostrar el resultado
fprintf('La suma de los cuadrados es: %d\n', suma_cuadrados);
```

Ejercicio 3

Dado un arreglo de números, encuentra la suma de los números primos en el arreglo.

Ejercicio 3

Dado un arreglo de números, encuentra la suma de los números primos en el arreglo.

```
% Definir un arreglo de num
arreglo = [2, 7, 15, 8, 11, 20, 3];

% Inicializar la variable de suma
suma_primos = 0;

% Func para verificar si un num es primo
es_primo = @(n) all(mod(n, 2:n-1) ~= 0);

% Calcular la suma de num primos en el arreglo
for num = arreglo
    if es_primo(num)
        suma_primos = suma_primos + num;
    end
end

% Mostrar el resultado
fprintf('La suma de los num primos es: %d\n', suma_primos);
```

Ejercicio 4

Dado un número entero positivo, verifica si es un número perfecto. Un número es considerado perfecto si la suma de sus divisores positivos (excluyendo el propio número) es igual a ese número.

Ejercicio 4

Dado un número entero positivo, verifica si es un número perfecto. Un número es considerado perfecto si la suma de sus divisores positivos (excluyendo el propio número) es igual a ese número.

```
function es_perfecto = verificar_perfecto(numero)
    es_perfecto = false;
    % Inicializar la variable para la suma de los divisores
    suma_divisores = 0;
    % Encontrar los divisores y sumarlos
    for divisor = 1:numero-1
        if rem(numero, divisor) == 0
            suma_divisores = suma_divisores + divisor;
        end
    end
    % Verificar si la suma de los divisores es igual al num
    if suma_divisores == numero
        es_perfecto = true;
    end
end
```

¡Gracias por vuestra atención!