Working with Matrices and Arrays in MATLABTable of Contents

Introduction to Matrices and Arrays

When I started working with MATLAB, I quickly learned that its core strength lies in its ability to handle matrices and arrays efficiently. Whether it's a scalar, vector, or multi-dimensional array, understanding how to create and manipulate these structures is foundational. I began by exploring basic arrays and gradually worked my way to higher-dimensional data structures.

Creating Matrices and ArraysMATLAB Code: Scalars and Vectors

```
% I start by creating the simplest data type in MATLAB, a scalar.
% A scalar is just a single numeric value.
x = 1; % I assign the value 1 to the scalar x.

% Next, I create a row vector, which is a one-dimensional array with elements arranged horizontally.
% I separate the elements with either commas or spaces.
v = [1, 2, 3, 4]; % This is a row vector using commas.
disp(['Row vector (comma-separated): ', mat2str(v)]); % Displaying the row vector in color.
```

```
Row vector (comma-separated): [1 2 3 4]
```

```
v = [1 2 3 4]; % This is a row vector using spaces.
disp(['Row vector (space-separated): ', mat2str(v)]); % Displaying the row vector in color.
```

```
Row vector (space-separated): [1 2 3 4]
```

```
% Now, I create a column vector, where the elements are arranged vertically.
% I use semicolons to separate the elements.
v_col = [1; 2; 3; 4]; % This is a column vector.
disp(['Column vector: ', newline, mat2str(v_col)]); % Displaying the column vector vertically.
```

```
Column vector:
[1;2;3;4]
```

Matrices with Uniform Dimensions

```
% I create a 2x4 matrix by separating rows with semicolons.
% A matrix is essentially a 2D array.
M = [1 2 3 4; 5 6 7 8]; % This matrix has 2 rows and 4 columns.
disp(['2x4 Matrix:', newline, mat2str(M)]); % Displaying the matrix in color.
```

```
2x4 Matrix:
[1 2 3 4;5 6 7 8]
```

```
% I use the continuation operator (...) to split the code into multiple lines for readability.
% This is helpful for creating larger matrices.
M = [1 2; ...  % The ellipses indicate that the row definition continues on the next line.
     4 5; ...
     6 7; ...
     8 9]; % This creates a 4x2 matrix.
disp(['4x2 Matrix:', newline, mat2str(M)]); % Displaying the 4x2 matrix in color.
```

```
4x2 Matrix:
[1 2;4 5;6 7;8 9]
```

Invalid Matrices: Unequal Row or Column Sizes

```
% MATLAB enforces that all rows and columns in a matrix must have the same length.
% I know the following examples will throw errors, so I leave them as comments.

% Example of an invalid matrix due to unequal row sizes:
% M = [1 2 3; 4 5 6 7]; % MATLAB would throw an error here.

% Example of an invalid matrix due to unequal column sizes:
% M = [1 2 3; ...
%      4 5; ...
%      6 7 8; ...
%      9 10]; % MATLAB would also throw an error here.
```

Transposing Matrices and Vectors

```
% I create a row vector and transpose it into a column vector.
v_row = [1 2 3 4]; % A simple row vector.
v_col = v_row.'; % Using .' to transpose the row vector into a column vector.
disp(['Row vector transposed into column vector:', newline, mat2str(v_col)]);
```

```
Row vector transposed into column vector:
[1;2;3;4]
```

```
% I create a 2x4 matrix and transpose it into a 4x2 matrix.
M = [1 2 3 4; 5 6 7 8]; % A 2x4 matrix.
M_transposed = M.'; % Using .' to flip rows and columns.
disp(['2x4 Matrix transposed into 4x2:', newline, mat2str(M_transposed)]);
```

```
2x4 Matrix transposed into 4x2:
[1 5;2 6;3 7;4 8]
```

```
% I store a 2x2 matrix in a variable and transpose it into another variable.
A = [1 2; 3 4]; % A square 2x2 matrix.
B = A.'; % Transposing A into B.
disp(['Original matrix:', newline, mat2str(A)]);
```

```
Original matrix:
[1 2;3 4]
```

```
disp(['Transposed matrix:', newline, mat2str(B)]);
```

```
Transposed matrix:
[1 3;2 4]
```

Multi-Dimensional Arrays

Constructing Higher-Dimensional Arrays

```
% I create a 5x2x4x3 array filled with ones using the ones function.
arr = ones(5, 2, 4, 3); % This initializes a 4D array where all elements are 1.
disp('5x2x4x3 Array filled with ones:'); % Displaying the dimensions for context.
```

```
5x2x4x3 Array filled with ones:
```

```
disp(size(arr)); % Showing the size of the array instead of printing all elements.
```

```
     5     2     4     3
```

```
% I construct a 3D array by concatenating two 2D matrices along the third dimension.
arr3D = cat(3, [1 2 3; 4 5 6], [7 8 9; 0 1 2]); % Two matrices stacked along the third dimension.
disp('2x3x2 Array created using cat:'); % Showing the dimensions for context.
```

```
2x3x2 Array created using cat:
```

```
disp(size(arr3D)); % Displaying the size instead of printing all elements.
```

```
     2     3     2
```

```
% I reshape a range of numbers into a 4D array with specified dimensions.
arr4D = reshape(1:120, [5 4 3 2]); % Reshaping numbers 1 through 120 into a 4D array.
disp('Reshaped 1:120 into a 5x4x3x2 array:'); % Showing the dimensions.
```

```
Reshaped 1:120 into a 5x4x3x2 array:
```

```
disp(size(arr4D)); % Displaying the size instead of printing all elements.
```

```
     5     4     3     2
```