# MATLAB: Vectors and Matrices

Connect with me: Youtube | LinkedIn | WhatsApp Channel | Web | Facebook | Twitter

- Download PDF

- To access the updated handouts, please click on the following link:
  https://yasirbhutta.github.io/matlab/docs/vectors-matrices.html

## Vector Products

**Important:** To perform matrix multiplication, the first matrix must have the same number of columns as the second matrix has rows. The number of rows of the resulting matrix equals the number of rows of the first matrix, and the number of columns of the resulting matrix equals the number of columns of the second matrix. [1]

**Important:** In MATLAB, the concept of conjugate refers to the complex conjugate of a number or element within a matrix. The complex conjugate of a complex number is a new number with the same real part but the imaginary part negated.

## Vectors in MATLAB

Vectors are one-dimensional arrays used to store a collection of numbers. You can create them in different ways:

1. **Using square brackets [ ]:**

   ```
   myVector = [1 2 3 4 5];
   ```

2. **Using the colon operator ::**

   ```
   rangeVector = 1:5;  % Creates a vector from 1 to 5 (inclusive)
   ```

## Accessing Elements

There are two main ways to access individual elements within a vector:

1. **Using Indices:**

   - MATLAB uses **one-based indexing**, meaning the first element has an index of 1, the second has an index of 2, and so on.
   - To access a specific element, enter the vector name followed by the index in parentheses:

     ```
     myVector(2)  % Accesses the second element (value 2)
     rangeVector(4)  % Accesses the fourth element (value 4)
     ```

2. **Using Colon Operator** `::`

   - The colon operator allows you to select a range of elements:
     - `vector(start:end)`: Accesses elements from `start` (inclusive) to `end` (inclusive).
     - `vector(start:step:end)`: Accesses elements with a specific `step` between them (similar to Python slicing).
     - `vector(:)`: Accesses all elements of the vector.

```
myVector(2:4)  % Accesses elements from index 2 (value 2) to index 4
(value 4)
rangeVector(1:2:5)  % Accesses elements 1, 3, and 5 (with a step of 2)
myVector(:)  % Accesses all elements (same as `myVector`)
```

3. **Creating and Accessing a Vector:**

```
fruits = ["apple" "banana" "orange"];  % Create a vector of strings
secondFruit = fruits(2);  % Access the second element ("banana")
```

4. **Extracting Sub-vectors:**

```
temperatures = [20 32 25 18];
firstTwo = temperatures(1:2);  % Extract the first two elements ([20 32])
evenIndices = temperatures(2:2:end);  % Extract elements at even indices
([32 18])
```

**Tips**

- Remember one-based indexing.
- Use `end` as an index to refer to the last element: `myVector(end)`.
- The colon operator is versatile for selecting elements or creating new vectors.

**See also:**

- [Matrix Indexing in MATLAB - MathWorks Docs](#)

# Generating Row Vectors with Even Spacing in MATLAB

## a. linspace

Linspace in MATLAB is a function used to generate a row vector containing **evenly spaced values** between two specified endpoints.

**Syntax:**

- `y = linspace(x1, x2)`: This creates a row vector with 100 (default) points between `x1` and `x2`.

- `y = linspace(x1, x2, n)`: This allows you to specify the number of points (`n`) in the vector.

**Key Points:**

- The spacing between values is calculated as: `(x2 - x1) / (n - 1)`.
- It includes both the starting (`x1`) and ending (`x2`) points in the output vector.
- Linspace is similar to the colon operator (`:`) but offers more control over the number of points.

**Use Cases:**

- Creating data points for plotting functions.
- Setting up evenly spaced grid points for numerical calculations.
- Controlling the sampling rate for simulations.

**See also:**

- **MATLAB Documentation:** https://www.mathworks.com/help/matlab/ref/linspace.html

### b. logspace

In MATLAB, the `logspace` function is used to generate a row vector containing **logarithmically spaced values** between two specified points. It's the logarithmic counterpart of the `linspace` function you saw earlier.

**Syntax:**

- `y = logspace(a, b)`: This creates a row vector with 50 points (default) between decades of 10^a and 10^b.
- `y = logspace(a, b, n)`: This allows you to specify the number of points (`n`) in the vector.
- `y = logspace(a, pi)`: This is useful for creating logarithmically spaced frequencies (especially in digital signal processing) in the interval [10^a, pi].

**Key Points:**

- The spacing is based on logarithms, not linear values. Points are closer together at lower values and farther apart at higher values.
- It includes an approximation of the starting point (10^a) and the ending point (might not be exactly pi if used) in the output vector.

- **Use Cases:**
    - Generating frequencies for simulations or filter design.
    - Sampling data across a wide range of values.
    - Creating logarithmic axes for plotting data that exhibits exponential growth or decay.

**See also**

- **MATLAB Documentation:** https://www.mathworks.com/help/matlab/ref/logspace.html

# Vector Functions

## 1. sum(x)

```matlab
v = [1, 2, 3, 4, 5];
total_sum = sum(v);
disp(total_sum)  % Output: 15
```matlab

### 2. mean(x)

```matlab
v = [4, 6, 7, 3, 5];
average_value = mean(v);
disp(average_value)  % Output: 5
```

## 3. length(x)

```matlab
v = [1, 2, 3, 4, 5];
vector_length = length(v);
disp(vector_length)  % Output: 5
```

## 4. max(x)

**Example #:** Finding the maximum element

```matlab
v = [1, 5, 2, 8, 3];
max_element = max(v);
disp(max_element)  % Output: 8
```

## 5. min

**Example #:** Finding the minimum element

```matlab
v = [5, 2, 8, 1, 3];
minimum_value = min(v);
disp(minimum_value)  % Output: 1
```

## 6. prod(x)

**Example #:** Product of all elements

```matlab
v = [2, 3, 4, 1];
total_product = prod(v);
disp(total_product)  % Output: 24
```

## 7. sign(x)

- The sign function in MATLAB is useful for determining the sign (positive, negative, or zero) of elements in vectors and arrays.

**Example #:** Finding the sign of each element:

```
v = [1, -2, 0, 3.5, -4];
element_signs = sign(v);
disp(element_signs)  % Output:  1  -1   0   1  -1
```

This code creates a vector v and uses sign(v) to determine the sign of each element. The results are stored in element_signs. The output shows 1 for positive values, -1 for negative values, and 0 for zero.

## 8. find(x)

- The `find` function in MATLAB is a versatile tool for finding specific elements or their indices within a vector.

**1. Finding Non-Zero Elements:**

- The simplest usage is `k = find(v)`. This returns a vector `k` containing the linear indices of all non-zero elements in vector `v`.

**Example:**

```
v = [1, 0, 3, 0, 5];
k = find(v);
disp(k)  % Output: 1 3 5
```

**2. Finding a Specific Number of Elements:**

- You can specify the number of elements to find:
    - `k = find(v, n)` returns the first `n` indices of non-zero elements.
    - `k = find(v, n, 'last')` returns the last `n` indices of non-zero elements.

**Example:**

```
v = [2, 0, -1, 4, 0];
first_two = find(v, 2);  % Find the first two non-zero elements
last_two = find(v, 2, 'last');  % Find the last two non-zero elements
disp(first_two)  % Output: 1 3
disp(last_two)  % Output: 3 4
```

**3. Finding Elements Based on Conditions:**

- Use relational operators (<, >, <=, >=, ==, ~=) within `find` to locate elements meeting specific criteria.

**Example:**

```
v = [8, 2, 6, 1, 9];
greater_than_5 = find(v > 5);  % Find indices of elements greater than 5
equal_to_2 = find(v == 2);  % Find indices of elements equal to 2
disp(greater_than_5)  % Output: 1 4 5
disp(equal_to_2)  % Output: 2
```

Remember, the `find` function is a powerful tool for manipulating vectors in MATLAB. Explore the documentation https://www.mathworks.com/help/matlab/ref/find.html for more details and advanced usage scenarios.

## 9. fix(x)

**Example #:** Fixing all elements in a vector

```
v = [-2.5 1.8 4.3 -7.1];
fixed_v = fix(v);
disp(fixed_v)  % Output: -2  1  4  -7
```

This code creates a vector v with decimal values. fix(v) applies the fix function to each element, rounding them towards zero and storing the result in fixed_v.

## 10. floor

The floor function in MATLAB is useful for rounding down elements in vectors (and arrays) to the nearest integer less than or equal to that element.

**Important:** floor always rounds down towards negative infinity, regardless of the sign of the input value.

**Example #:** Rounding down elements in a vector

```
v = [1.5, 2.8, 3.1, 4.9, 5.2];
floored_vector = floor(v);
disp(floored_vector)  % Output: 1  2  3  4  5
```

**Example #:** Rounding down negative decimals

```
v = [-1.5, 2.8, -3.1, 4.9, -5.2];
floored_vector = floor(v);
disp(floored_vector)  % Output: -2  2  -4  4  -6
```

## 11. ceil

**Important point:** ceil always rounds up towards positive infinity.

**Example #:** Rounding towards positive infinity

```
v = [-1.5, 2.8, -3.1, 4.9, -5.2];
ceiling_vector = ceil(v);
disp(ceiling_vector)  % Output: -1  3  -3  5  -5
```

In this example, ceil(v) rounds each element in v towards positive infinity. So:

- Negative decimals are rounded up to the nearest negative integer (closer to zero).
- Positive decimals are rounded up to the nearest positive integer.

## 12. round

**Example #:** Rounding to nearest integers

```
v = [1.5, 2.8, -3.7, 4.9, -5.2];
rounded_vector = round(v);
disp(rounded_vector)  % Output: 2  3  -4  5  -5
```

## 13. sort

**Example #:** Sorting in ascending order (default)

```
v = [5, 1, 4, 2, 3];
sorted_vector = sort(v);
disp(sorted_vector)  % Output: 1  2  3  4  5
```

This code creates a vector v and uses sort(v) to sort its elements in ascending order. The result is stored in sorted_vector.

**Example #:** Sorting in descending order You can specify 'descend' as the second argument to sort in descending order:

```
sorted_descending = sort(v, 'descend');
disp(sorted_descending)  % Output: 5  4  3  2  1
```

## 14. mod

The mod function in MATLAB is useful for finding the remainder after element-wise division of vectors. Here are some examples of how to use it with vectors:

**1. Modulo with a Scalar:**

- This calculates the remainder for each element of the vector divided by the scalar value.

**Example:**

```
v = [10, 5, 18, 3, 12];
mod_result = mod(v, 4);  % Find remainder after dividing each element by 4
disp(mod_result)  % Output: 2 1 2 3 0
```

**2. Modulo Between Vectors:**

- You can perform modulo between two vectors of the same size. The operation is performed element-wise.

**Example:**

```
v1 = [7, 11, 15];
v2 = [3, 4, 5];
mod_result = mod(v1, v2);
disp(mod_result)  % Output: 1 3 0 (Remainder of v1(i) / v2(i))
```

**Key points:**

- The mod function follows the convention that `mod(a,0) returns a`. In other words, the remainder when dividing by zero is the original dividend itself.
- For floating-point input arguments, the output data type of mod is the same as the inputs.

By understanding these examples, you can effectively use the mod function to perform modulo operations on vectors in MATLAB.

## 15. rem

The rem function in MATLAB behaves very similarly to mod for vectors, but with a subtle difference. Here's how to use it with vectors:

**1. Rem with a Scalar:**

- This calculates the remainder for each element of the vector divided by the scalar value.

**Example:**

```
v = [10, 5, 18, 3, 12];
rem_result = rem(v, 4);  % Find remainder after dividing each element by 4
disp(rem_result)  % Output: 2 1 2 3 0
```

**2. Rem Between Vectors:**

- You can perform element-wise modulo between two vectors of the same size.

**Example:**

```
v1 = [7, 11, 15];
v2 = [3, 4, 5];
rem_result = rem(v1, v2);
disp(rem_result)  % Output: 1 3 0 (Remainder of v1(i) / v2(i))
```

**3. Key Difference from mod:**

- The key difference between rem and mod lies in the handling of negative dividends.
  - mod always returns a non-negative remainder between 0 and the divisor minus one.
  - rem however, signs the remainder according to the sign of the dividend.

**Example:**

```
v = [-10, 5, -18, 3, -12];
rem_result = rem(v, 4);  % Remainder considering sign of dividend
disp(rem_result)  % Output: -2 1 -2 3 -0
```

# Matrices

## Creating Matrices: Entering elements directly

**Example #:** Creating a 2x2 Matrix

```
A = [1 2; 3 4];
disp(A);
```

This creates a 2x2 matrix A with elements:

```
1   2
3   4
```

**Example #:** Creating a 3x3 Matrix

```
% Create a 3x3 matrix
matrix = [1 2 3; 4 5 6; 7 8 9]
```

**Example #:** Create a 2x3 matrix

```
C = [1 2 3; 4 5 6]
```

**Example #:** Specifying elements with spaces or commas

Elements within a row can be separated by either commas ( , ) or spaces. Both notations achieve the same result.

```matlab
% Using commas
matrix_comma = [1, 4, 7; 2, 5, 8; 3, 6, 9]

% Using spaces
matrix_space = [1 4 7; 2 5 8; 3 6 9]
```

**Example #:** Creating a 4x2 Matrix

```matlab
C = [2, 4; 6, 8; 10, 12; 14, 16];
```

This creates a 4x2 matrix C with elements:

```
 2   4
 6   8
10  12
14  16
```

**Example #:** Creating a 3x4 Matrix

```matlab
D = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12];
```

This creates a 3x4 matrix D with elements:

```
1   2   3   4
5   6   7   8
9  10  11  12
```

**Example #:** Creating a 1x5 Row Vector

```matlab
E = [1, 2, 3, 4, 5];
```

This creates a 1x5 row vector E with elements:

```
  1    2    3    4    5
```

**Example #:** Creating a 5x1 Column Vector:

```
F = [6; 7; 8; 9; 10];
```

This creates a 5x1 column vector F with elements:

```
6
7
8
9
10
```

**See also:**

- Creating Matrices and Arrays - MathWorks Help Center

## Matrix Indexing in MATLAB

In MATLAB, indexing into matrices allows you to access and manipulate specific elements, rows, columns, or subsets of a matrix. Here are some examples of matrix indexing in MATLAB:

**Example #:** Accessing single elements:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
element = A(2, 3); % Accesses the element in the second row and third column
disp(element); % Displays: 6
```

**Example #:** Accessing entire rows or columns:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
row = A(2, :); % Accesses the entire second row
disp(row); % Displays: 4 5 6

column = A(:, 3); % Accesses the entire third column
disp(column); % Displays: 3 6 9
```

**Example #:** Assigning values to specific elements:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
A(1, 2) = 10; % Assigns the value 10 to the element in the first row and second
```

```
    column
    disp(A); % Displays the updated matrix

    % Resulting matrix:
    % 1   10   3
    % 4    5   6
    % 7    8   9
```

**Example #:** Indexing with linear indices:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
linear_index = [1, 4, 7]; % Linear indices of elements to access
elements = A(linear_index); % Accesses elements using linear indices
disp(elements); % Displays elements corresponding to linear indices
```

The Output of the code is:

```
   1      2      3
```

**Example #:** Indexing with logical arrays:

```
A = [1, 2, 3; 10, 5, 1; 4, 8, 9];
logical_index = A > 5; % Creates a logical array indicating elements greater than
5
disp(logical_index); % Displays the logical array

% Resulting logical array:
% 0    0    0
% 1    0    0
% 0    1    1

% Using logical array to access elements
elements_gt_5 = A(logical_index);
disp(elements_gt_5); % Displays elements greater than 5

% Output
% 10
% 8
% 9
```

# Concatenation of Matrices

**Using square brackets ([]):**

- This is a simpler approach for basic concatenation.

- Use commas (,) to concatenate matrices horizontally (side-by-side). The matrices must have the same number of rows.
- Use semicolons (😉 to concatenate matrices vertically (on top of each other). The matrices must have the same number of columns.

**Example #:** Horizontal concatenation:

```matlab
A = [1 2; 3 4];
B = [5 6; 7 8];

% Using square brackets
C = [A, B];

disp(C);
```

**Example #:** Vertical concatenation

```matlab
A = [1 2; 3 4];
B = [5 6; 7 8; 9 10];

% Using square brackets
C = [A; B];

disp(C);
```

**Example #**

```matlab
% Creating two matrices
A = [1 2 3; 4 5 6];
B = [7 8 9; 10 11 12];

% Vertical concatenation
C = [A; B];
disp(C);
```

**Output:**

```
1      2      3
4      5      6
7      8      9
10     11     12
```

**Example #:** Mixed Concatenation (Combination of Vertical and Horizontal)

```matlab
% Creating matrices
A = [1 2; 3 4];
B = [5 6; 7 8];
C = [9 10 11 12];

% Horizontal concatenation of A and B, followed by vertical concatenation with C
D = [A B; C];
disp(D);
```

**Output:**

```
1     2     5     6
3     4     7     8
9    10    11    12
```

**Class Activity**

**Creating Matrices with Different Sizes:**

- The size of a matrix is determined by its rows and columns. Let's create matrices of different sizes:
    - Create a 3x4 matrix named D with elements 1 to 12.
    - Create a 1x5 row vector named E.
    - Create a 5x1 column vector named F.
    - Use `disp(D)`, `disp(E)`, and `disp(F)` to view their outputs.

**Accessing Single Elements:**

- We can access specific elements within a matrix using row and column indices.

    - Create a matrix A like before (e.g., A = [1 2; 3 4]).
    - How can you access the element in the second row and third column of A?
    - Write the code to access that element and display its value using disp().
    - Accessing Rows and Columns:

- We can access entire rows or columns at once.

- Write the code to access the second row and third column of A and display them using disp().

- Use matrix indexing to perform the following tasks:

    - Access the element in the second row and third column of matrix M1.
    - Extract the entire second row of matrix M2.
    - Change the element in the third row and second column of matrix M3 to 5.

**Modifying Elements:**

- We can change specific values within the matrix.
    - Write the code to change the element in the first row and second column of A to the value 10.
    - Use disp(A) again to see the updated matrix.

**Horizontal Concatenation:**

- Concatenation allows us to combine matrices.
    - Create two matrices A and B with some example values (e.g., 2x2 matrices).
    - How can you join them horizontally (side-by-side)?
    - Write the code and use disp() to see the result.

**Vertical Concatenation:**

- We can also join matrices vertically (on top of each other).
- Create new matrices A and B with different dimensions (e.g., A is 3x2, B is 2x2).
- Write the code to concatenate them vertically and use disp() to see the outcome.

**Mixed Concatenation:**

- Create matrices X = [1 2; 3 4] and Y = [5 6; 7 8].

- How can we combine these matrices horizontally (side-by-side)? (Hint: Use ,)

- Now, let's combine them vertically (on top of each other). (Hint: Use 😉)

- We can combine horizontal and vertical concatenation.

- Create three matrices A, B, and C with different sizes.

- Write the code to join A and B horizontally, then concatenate the result vertically with C.

- Use disp() to see the final matrix.

# Special matrices

1. zeros

2. ones

3. eye

4. rand

5. rands

6. vender

7. diag

# Functions related to matrices

1. det

2. rank

**Example #:**

```matlab
% Create a 3x3 matrix A
A = [1, 2, 3; 2, 4, 6; 0, 0, 0];

% Calculate the rank of matrix A
% Rank represents the maximum number of linearly independent rows/columns
rank_of_A = rank(A);

% Display the rank of matrix A
disp(['The rank of matrix A is: ', num2str(rank_of_A)]);
```

## 3. trace

The most common use of trace in MATLAB is to calculate the sum of the elements on the main diagonal of a square matrix.

**Example #:**

```matlab
% Define a square matrix
A = [1 2 3; 4 5 6; 7 8 9];

% Calculate the trace of A (sum of diagonal elements)
trace_of_A = trace(A);

% Display the result
disp(['The trace of A is: ', num2str(trace_of_A)]);
```

## 4. inv

## 5. norm

The norm function in MATLAB calculates the norm of a vector or matrix. The norm represents the **magnitude or size** of the mathematical object. There are different types of norms depending on the value of a second argument (p) you provide to the function.

Here's a breakdown of how norm works in MATLAB:

**Syntax:**

```matlab
norm(X)
norm(X, p)
```

- X is the vector or matrix for which you want to find the norm.

- p (optional) specifies the type of norm:

  - If omitted (p is empty), norm calculates the **2-norm** (Euclidean norm) for vectors and the **maximum singular value** (operator norm) for matrices (which approximates the 2-norm for

matrices).

- n = norm(X,p) returns the p-norm of matrix X, where p is 1, 2, or Inf:

    - If p = 1, then n is the maximum absolute column sum of the matrix.
    - If p = 2, then n is approximately max(svd(X)). This value is equivalent to norm(X).
    - If p = Inf, then n is the maximum absolute row sum of the matrix.

**Examples:**

1. **2-norm (Euclidean norm) of a vector:**

**Important:** The Euclidean norm of a square matrix is the square root of the sum of all the squares of the elements.

```
v = [1 2 3];
vector_norm = norm(v);
disp(['The 2-norm (Euclidean norm) of v is: ', num2str(vector_norm)])
```

2. **1-norm of a matrix:**

**Important:** The 1-norm of a square matrix is the maximum of the absolute column sums. [2]

```
A = [1 2; 3 4];
matrix_norm_1 = norm(A, 1);
disp(['The 1-norm of A is: ', num2str(matrix_norm_1)])
```

3. **Infinity norm of a matrix:**

**Important:** The infinity-norm of a square matrix is the maximum of the absolute row sums.

```
B = [5 0; -1 2];
matrix_norm_inf = norm(B, Inf);
disp(['The infinity norm of B is: ', num2str(matrix_norm_inf)])
```

**In summary:**

- norm calculates the norm of a vector or matrix.
- The type of norm depends on the second argument (p).
- By default (p omitted), it calculates the 2-norm for vectors and the maximum singular value for matrices.

**See also:**

- [Vector and matrix norms - MATLAB norm](#)
- https://www.sciencedirect.com/topics/mathematics/euclidean-norm
- https://bathmash.github.io/HELM/30_4_mtrx_norms-web/30_4_mtrx_norms-webse1.html

## 6. transpose

In MATLAB, you can transpose a matrix using two main methods:

1. **Single Quote ('):**

   This is the most common and concise way to find the transpose. The single quote symbol (') is appended to the matrix name. The transpose operation swaps the rows and columns of the original matrix.

   Here's the syntax:

   ```matlab
   B = A'  % ' denotes transpose
   ```

   - A is the original matrix.
   - B is the resulting transposed matrix.

2. **transpose function:**

   This function offers an alternative way to achieve the transpose. It's functionally equivalent to the single quote method.

   Here's the syntax:

   ```matlab
   B = transpose(A)
   ```

**Important Notes:**

- Both methods create a **new** matrix (B) containing the transposed elements. They don't modify the original matrix (A).
- This operation works for matrices of any dimension (not just square matrices).

**Example:**

```matlab
% Define a matrix
A = [1 2 3; 4 5 6];

% Find the transpose using single quote
B = A';

% Find the transpose using transpose function
C = transpose(A);

% Display the original and transposed matrices
disp('Original matrix A:');
disp(A);

disp('Transposed matrix B (using single quote):');
```

```
   disp(B);

   disp('Transposed matrix C (using transpose function):');
   disp(C);
```

**This code will output:**

```
Original matrix A:
   1  2  3
   4  5  6

Transposed matrix B (using single quote):
   1  4
   2  5
   3  6

Transposed matrix C (using transpose function):
   1  4
   2  5
   3  6
```

As you can see, both methods produce the same transposed matrix.

## 7. x = A\b

## 8. poly

**Example #:** Finding the Characteristic Polynomial of a Matrix:

If you provide a square matrix (A), the poly function will return the coefficients of the characteristic polynomial of that matrix. The characteristic polynomial is a polynomial whose roots are the eigenvalues of the matrix.

```
A = [1 2; 3 4];
p = poly(A);
disp(p);  % This will display the coefficients of the characteristic polynomial
```

## 9. eig

The `eig` function in MATLAB is a powerful tool for working with eigenvalues and eigenvectors. Here's a detailed explanation of its functionalities:

**What it does:**

- The `eig` function computes the eigenvalues and eigenvectors of a square matrix.

**Inputs:**

- The primary input to `eig` is a square matrix (A).

**Outputs:**

- `eig` can return two or three outputs depending on how you use it:
  - `e = eig(A)`: This returns a column vector containing the eigenvalues of `A`.
  - `[V, D] = eig(A)`: This returns two matrices:
    - `V`: A matrix whose columns are the eigenvectors of `A`. Each column corresponds to an eigenvalue in the diagonal matrix `D`.
    - `D`: A diagonal matrix containing the eigenvalues of `A` on its main diagonal. (The order of eigenvalues in `D` corresponds to the columns in `V`).

**Understanding Eigenvalues and Eigenvectors:**

Before diving deeper into `eig`, it's crucial to understand eigenvalues (λ) and eigenvectors (v) of a matrix:

- **Eigenvalue (λ):** A scalar value associated with a square matrix. It represents the scaling factor when a particular eigenvector is multiplied by the matrix.
- **Eigenvector (v):** A non-zero vector that, when multiplied by the matrix, gets scaled by the eigenvalue but doesn't change direction.

**Eigenvalue Equation:**

Mathematically, this relationship is expressed by the equation Av = λv, where:

- A represents the matrix that defines the linear transformation.
- v represents the eigenvector.
- λ represents the eigenvalue.

**`eig` Examples:**

Here are some examples of how to use `eig` in MATLAB:

1. **Finding Eigenvalues:**

```
A = [2 1; 1 3];
e = eig(A);
disp(e);  % This will display the eigenvalues of A
```

2. **Finding Eigenvalues and Eigenvectors:**

```
A = [4 -1; 2 1];
[V, D] = eig(A);
disp(V);  % This will display the eigenvectors as columns of V
disp(D);  % This will display the eigenvalues on the diagonal of D
```

**See also:**

- For more advanced usage and detailed information, refer to the official MATLAB documentation:
  https://www.mathworks.com/help/matlab/ref/eig.html

10. eigs

11. orth

## Key Terms

## True/False (Mark T for True and F for False)

**Answer Key (True/False):**

## Multiple Choice (Select the best answer)

How can you access the second element (value 2) of a vector named myVector?

a) myVector(2) Correct b) myVector[2] c) myVector.get(2)

What is the function in MATLAB used to generate a row vector with evenly spaced values between two specified endpoints?

a) linspace Correct b) arange c) logspace

How can you create a 2x3 matrix in MATLAB?

a) matrix = [[1, 2, 3]; [4, 5, 6]] Correct b) matrix = (1, 2, 3; 4, 5, 6) c) matrix = [1 2 3; 4:6]

How can you create a row vector containing 5 evenly spaced values between 2 and 10 in MATLAB?

a) vector = [2:10] b) vector = linspace(2, 10) ✔ c) vector = ones(1, 5) * 2 + 8 d) vector = zeros(1, 5) + 2

What is the output of the following code?

Matlab v = [1 -2 3.5]; element_signs = sign(v); disp(element_signs) Use code with caution. content_copy a) [1 1 1] b) [1 -1 3.5] c) [1 -2 3.5] d) [1 -1 3]

How can you concatenate two matrices A and B horizontally (side-by-side) in MATLAB?

a) C = A + B b) C = A * B c) C = [A, B] ✔ d) C = vertcat(A, B)

## Fill in the Blanks

**Answer Key (Fill in the Blanks):**

## Practice Exercises

1. Create a vector of your favorite numbers and access the third element.
2. Create a vector of temperatures and extract the elements for Monday and Wednesday (assuming even indices represent Wednesdays).
3. Experiment with the colon operator to create sub-vectors with different intervals.

## Review Questions

## Review Questions for MATLAB: Vectors and Matrices

**Concepts:**

1. What are vectors in MATLAB? How are they different from matrices?
2. Explain one-based indexing in MATLAB.
3. How can you create vectors using square brackets and the colon operator?
4. Describe the functionalities of the following functions related to vectors:
   - sum
   - mean
   - length
   - max
   - min
   - prod
   - sign
   - find
   - fix
   - floor
   - ceil
   - round
   - sort
   - mod
   - rem

**Matrices:**

5. How can you create matrices in MATLAB by entering elements directly?
6. Explain different ways to specify elements within a row (commas vs. spaces).
7. How do you access single elements, entire rows or columns, and specific elements using linear indices in matrices?
8. Describe matrix concatenation using square brackets ([]).
9. What are some special matrices you can create in MATLAB (zeros, ones, eye, etc.)?
10. What are the different methods for creating matrices in MATLAB?
11. How do you access specific elements in a matrix using indexing?
12. When is horizontal concatenation preferable over vertical concatenation?
13. Can you explain the concept of mixed concatenation with an example?

**Matrix Functions:**

10. Explain the purpose of the following functions related to matrices:
    - det
    - rank
    - trace
    - inv
    - norm
    - transpose

**Additional Questions:**

11. What are some resources available to learn more about MATLAB and its functionalities (documentation, tutorials, etc.)?

12. Describe a practical scenario where you might use vectors and matrices in MATLAB.

**Challenge Questions:**

13. Write a MATLAB script that takes two vectors as input and calculates their inner product (dot product).
14. Write a MATLAB script that creates a random 5x5 matrix and finds the eigenvalues and eigenvectors of the matrix.

## References and Bibliography

[1] "M.2 Matrix Arithmetic | STAT ONLINE," PennState: Statistics Online Courses.
https://online.stat.psu.edu/statprogram/reviews/matrix-algebra/arithmetic [2] "Matrix norms,"
bathmash.github.io. https://bathmash.github.io/HELM/30_4_mtrx_norms-web/30_4_mtrx_norms-webse1.html
(accessed Apr. 16, 2024).