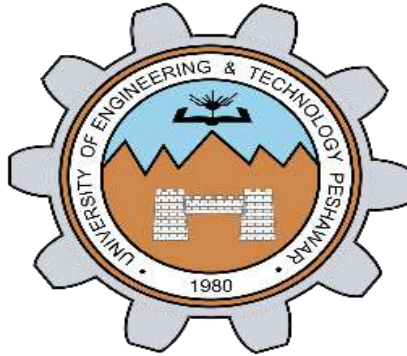


SIGNALS AND SYSTEMS LAB (CSE-301L)

Spring 2024, 4th Semester

Lab Report 01



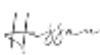
Submitted by: Hassan Zaib Jadoon

Registration Number: 22PWCSE2144

Section: A

“On my honor, as a student at the University of Engineering and Technology Peshawar, I have neither given nor received unauthorized assistance on this academic work.”

 Recoverable Signature

X 

Hassan Zaib JAdoon
Student

Signature: Signed by: 6fc0c9f9-f92a-4c79-aa63-cf063a60e638

Date: 10/3/2024

Submitted To: Dr. Safdar Nawaz Khan Marwat

Department of Computer Systems Engineering

University of Engineering and Technology Peshawar

Lab 01: Introduction to MATLAB

TASK 01

Write a program to generate a new matrix B from matrix A given below such that each column in the new matrix except the 1st one is the result of subtraction of that column from the previous one i.e. 2nd new column is the result of subtraction of 2nd column and 1st column and so on. Copy the first column as it is in the new matrix.

$$A = \begin{bmatrix} 3 & 6 & 9 \\ 1 & 4 & 8 \\ 2 & 8 & 7 \end{bmatrix}$$

Problem Analysis

The problem requires generating a new matrix, let's call it matrix B, from an existing matrix A such that each column in matrix B, except the first one, is the result of subtracting the previous column from the current one. Essentially, the second column of matrix B will be the result of subtracting the first column of matrix A from the second column of matrix A, the third column of matrix B will be the result of subtracting the second column of matrix A from the third column of matrix A, and so on.

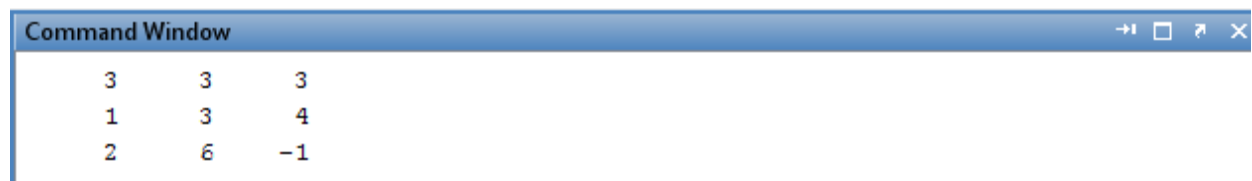
Algorithm

1. Create a new matrix B with the same dimensions as matrix A.
2. Copy the first column of matrix A to the first column of matrix B.
3. Iterate through each column of matrix A starting from the second column: For each column, iterate through each row and subtract the value of the current column in matrix A from the value of the previous column in matrix A.
4. Assign the result to the corresponding cell in matrix B.
5. Return matrix B as the result.

Code

```
1      % Matrix A
2 -    A = [
3          3 6 9;
4          1 4 8;
5          2 8 7
6      ];
7
8      % Initialize matrix B with the same dimensions as matrix A
9 -    B = zeros(size(A));
10
11     % Copy the first column of A to the first column of B
12 -    B(:, 1) = A(:, 1);
13
14     % Subtract each column from the previous one in A and store the result in B
15 -    for i = 1:size(A, 1)
16 -        for j = 2:size(A, 2)
17 -            B(i, j) = A(i, j) - A(i, j-1);
18 -        end
19 -    end
20
21     % Display matrix B
22 -    disp(B);
```

Output



```
Command Window
3      3      3
1      3      4
2      6     -1
```

Conclusion

The program generates matrix B from matrix A, subtracting each column from the previous one, except for the first column which is directly copied.

TASK 02

Generate two 5000 sampled random discrete time signals (1-dimensional) using rand() function i.e. rand(1, 5000). Write a program to add the two signals together using simple vector addition.

Problem Analysis

The problem entails generating two 1-dimensional random discrete time signals, each comprising 5000 samples, using the rand() function in MATLAB. Subsequently, a program is required to add these two signals together using simple vector addition..

Algorithm

1. Generate two random 1-dimensional discrete time signals of length 5000 using the rand() function.
2. Store the generated signals in two separate arrays, say signal1 and signal2.
3. Create a new array, result, to store the sum of the two signals.
4. Perform element-wise addition of signal1 and signal2, storing the result in the corresponding index of the result array.
5. The resulting array, result, will contain the sum of the two signals.
6. Display or further process the resultant signal as needed.

Code

```
% Number of samples
num_samples = 5000;

% Generate the first random signal
signal1 = rand(1, num_samples);

% Generate the second random signal
signal2 = rand(1, num_samples);

% Add the two signals together using vector addition
sum_signal = signal1 + signal2;
```

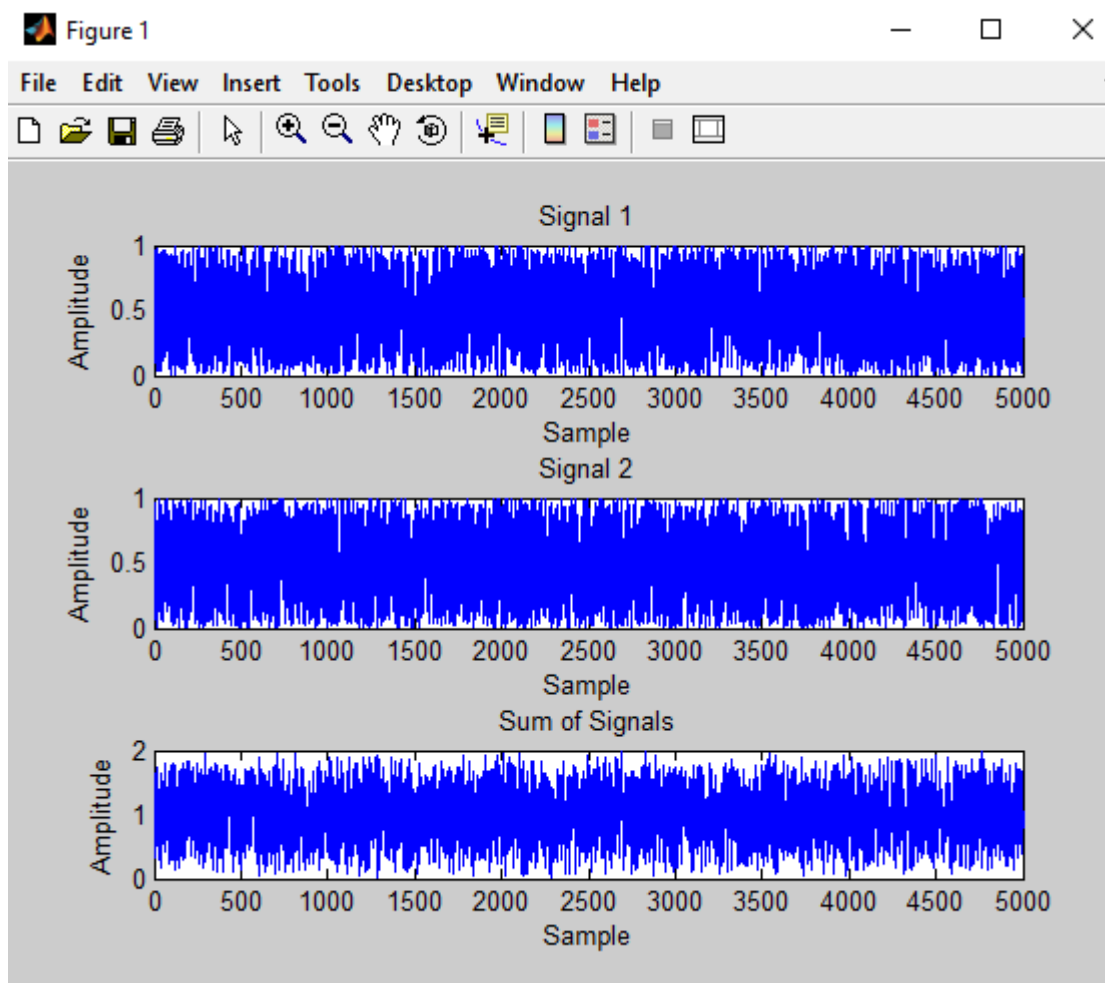
For plotting signals in plane, the following code is used:

```
% Plot the signals
subplot(3, 1, 1);
plot(signal1);
title('Signal 1');
xlabel('Sample');
ylabel('Amplitude');

subplot(3, 1, 2);
plot(signal2);
title('Signal 2');
xlabel('Sample');
ylabel('Amplitude');

subplot(3, 1, 3);
plot(sum_signal);
title('Sum of Signals');
xlabel('Sample');
ylabel('Amplitude');
```

Output



Conclusion

By leveraging MATLAB's `rand()` function and basic array operations, the program accomplishes the addition of the signals, thereby facilitating various signal processing or analysis tasks. This approach provides a straightforward method for manipulating and combining signals in MATLAB.

TASK 03

Using colon notation, generate the following sequence:

-120, -116, -112, ..., -4, 0, 4, 8, ..., 112, 116, 120.

Problem Analysis

The problem involves generating a sequence of numbers that starts at -120, increments by 4 until it reaches 120, and includes both negative and positive integers. The sequence follows a pattern where each successive number differs by 4.

Algorithm

1. Define the starting point as -120 and the ending point as 120.

2. Specify the step size as 4, which is the difference between consecutive numbers in the sequence.
3. Utilize colon notation to generate the sequence from -120 to 120, with increments of 4.
4. Store the generated sequence in an array or variable for further processing or display.
5. The resulting sequence will include the desired numbers: -120, -116, -112, ..., 120.

Code

```
1 - sequence = -120:4:120
```

Output

```
sequence =

Columns 1 through 12
-120 -116 -112 -108 -104 -100 -96 -92 -88 -84 -80 -76

Columns 13 through 24
-72 -68 -64 -60 -56 -52 -48 -44 -40 -36 -32 -28

Columns 25 through 36
-24 -20 -16 -12 -8 -4 0 4 8 12 16 20

Columns 37 through 48
24 28 32 36 40 44 48 52 56 60 64 68

Columns 49 through 60
72 76 80 84 88 92 96 100 104 108 112 116

Column 61
120
```

Conclusion

By specifying appropriate start, end, and step values, the program produces the desired sequence in a concise and straightforward manner. This approach enables easy manipulation and utilization of sequences for various computational tasks in MATLAB.

TASK 04

Given the matrices:

A= [-12,34,61, -9;65,78,90,12; 14,78,45,12; 60,25,3,8]

B= [34,67,8,9; 12, -91,12,9; 89,-8,0,2; 16,9,23,67]

Find the following:

- 1) Array addition; store the result in matrix C
- 2) Array subtraction; store the result in matrix D
- 3) Array multiplication using .* operator; store the result in matrix E
- 4) Array division using ./ operator; store the result in matrix F
- 5) Array exponentiation using .^ operator; store the result in matrix G
- 6) Take the sin of A and store the result in H, take sqrt of B and store the result in I. Find H*I and store the result in J.

Problem Analysis

The problem entails performing various array operations on given matrices A and B in MATLAB. These operations include addition, subtraction, multiplication, division, exponentiation, and trigonometric functions. Additionally, it requires the multiplication of the sine of matrix A with the square root of matrix B.

Algorithm

1. Define matrices A and B with the given values.
2. Perform array addition by adding corresponding elements of matrices A and B and store the result in matrix C.
3. Perform array subtraction by subtracting corresponding elements of matrices A and B and store the result in matrix D.
4. Perform array multiplication using the .* operator on matrices A and B and store the result in matrix E.
5. Perform array division using the ./ operator on matrices A and B and store the result in matrix F.
6. Perform array exponentiation using the .^ operator on matrices A and B and store the result in matrix G.
7. Take the sine of matrix A and store the result in matrix H.
8. Take the square root of matrix B and store the result in matrix I.
9. Perform matrix multiplication of matrices H and I and store the result in matrix J.

Code

```
% Given matrices A and B
A = [-12, 34, 61, -9; 65, 78, 90, 12; 14, 78, 45, 12; 60, 25, 3, 8];
B = [34, 67, 8, 9; 12, -91, 12, 9; 89, -8, 0, 2; 16, 9, 23, 67];
% 1) Array addition
C = A + B;
% 2) Array subtraction
D = A - B;
% 3) Array multiplication using .* operator
E = A .* B;
% 4) Array division using ./ operator
F = A ./ B;
% 5) Array exponentiation using .^ operator
G = A .^ 2;
% 6) Take the sin of A and store the result in H
H = sin(A);
% Take sqrt of B and store the result in I
I = sqrt(B);
% Find H*I and store the result in J
J = H * I;
% Displaying the matrices
disp('Matrix C (Addition):');
disp(C);
disp('Matrix D (Subtraction):');
disp(D);
disp('Matrix E (Multiplication):');
disp(E);
disp('Matrix F (Division):');
disp(F);
disp('Matrix G (Exponentiation):');
disp(G);
disp('Matrix H (Sin of A):');
disp(H);
disp('Matrix I (Square root of B):');
disp(I);
disp('Matrix J (Product of H and I):');
disp(J);
```

Output

```
Matrix C (Addition):
    22    101     69     0
    77    -13    102    21
   103     70     45    14
    76     34     26    75

Matrix D (Subtraction):
   -46    -33     53   -18
    53   169     78     3
   -75     86     45    10
    44     16    -20   -59

Matrix E (Multiplication):
   -408     2278     488    -81
    780    -7098    1080    108
   1246    -624     0     24
    960     225     69    536
```


Matrix F (Division):

| | | | |
|---------|---------|--------|---------|
| -0.3529 | 0.5075 | 7.6250 | -1.0000 |
| 5.4167 | -0.8571 | 7.5000 | 1.3333 |
| 0.1573 | -9.7500 | Inf | 6.0000 |
| 3.7500 | 2.7778 | 0.1304 | 0.1194 |

Matrix G (Exponentiation):

| | | | |
|------|------|------|-----|
| 144 | 1156 | 3721 | 81 |
| 4225 | 6084 | 8100 | 144 |
| 196 | 6084 | 2025 | 144 |
| 3600 | 625 | 9 | 64 |

Matrix H (Sin of A):

| | | | |
|---------|---------|---------|---------|
| 0.5366 | 0.5291 | -0.9661 | -0.4121 |
| 0.8268 | 0.5140 | 0.8940 | -0.5366 |
| 0.9906 | 0.5140 | 0.8509 | -0.5366 |
| -0.3048 | -0.1324 | 0.1411 | 0.9894 |

Matrix I (Square root of B):

| | | | |
|--------|-------------|--------|--------|
| 5.8310 | 8.1854 | 2.8284 | 3.0000 |
| 3.4641 | 0 + 9.5394i | 3.4641 | 3.0000 |
| 9.4340 | 0 + 2.8284i | 0 | 1.4142 |
| 4.0000 | 3.0000 | 4.7958 | 8.1854 |

Matrix J (Product of H and I):

| | | | |
|---------|------------------|--------|---------|
| -5.8013 | 3.1557 + 2.3145i | 1.3740 | -1.5427 |
| 12.8893 | 5.1582 + 7.4316i | 1.5458 | 0.8947 |
| 13.4378 | 6.4988 + 7.3098i | 2.0090 | 1.3251 |
| 3.0529 | 0.4731 - 0.8634i | 3.4242 | 6.9863 |

Conclusion

MATLAB efficiently handles array operations with its built-in operators and functions. The provided algorithm enables the execution of these operations on the given matrices A and B, resulting in the creation of matrices C, D, E, F, G, and J. These operations are fundamental for various mathematical and computational tasks, showcasing MATLAB's versatility in handling numerical data and operations.

TASK 05

Type the given matrix in MATLAB:

$$A = \begin{bmatrix} 3 & 7 & -4 & 12 \\ 9 & 10 & 2 & -5 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{bmatrix}$$

Find the following:

- 1) Create 4x3 array **B** consisting of all elements in the second through fourth columns of **A**.
- 2) Create 3x4 array **C** consisting of all elements in the second through fourth rows of **A**.
- 3) Create 2x3 array **D** consisting of all elements in the first two rows and the last three columns of **A**.

Problem Analysis

The problem involves extracting specific subsets of elements from a given matrix **A** in MATLAB. Specifically, it requires creating new arrays **B**, **C**, and **D** based on certain criteria: **B** consists of elements from the second through fourth columns of **A**, **C** consists of elements from the second through fourth rows of **A**, and **D** consists of elements from the first two rows and the last three columns of **A**.

Algorithm

1. Define the given matrix **A** with the provided elements.
2. For array **B**, extract elements from the second through fourth columns of matrix **A** using MATLAB indexing.
3. For array **C**, extract elements from the second through fourth rows of matrix **A** using MATLAB indexing.
4. For array **D**, extract elements from the first two rows and the last three columns of matrix **A** using MATLAB indexing.
5. Store the extracted elements in arrays **B**, **C**, and **D**, respectively.
6. Display the resulting arrays **B**, **C**, and **D** to visualize the extracted subsets.

Code

```
% Given matrix A
A = [3, 7, -4, 12; 9, 10, 2, -5; 6, 13, 8, 11; 15, 5, 4, 1];

% 1) Create 4x3 array B consisting of all elements in the second through fourth columns of A.
B = A(:, 2:4);

% 2) Create 3x4 array C consisting of all elements in the second through fourth rows of A.
C = A(2:4, :);

% 3) Create 2x3 array D consisting of all elements in the first two rows and the last three columns of A.
D = A(1:2, 2:end);

% Display the matrices
disp('Matrix B (Second through fourth columns of A):');
disp(B);

disp('Matrix C (Second through fourth rows of A):');
disp(C);

disp('Matrix D (First two rows and last three columns of A):');
disp(D);
```

Output

Matrix B (Second through fourth columns of A):

```
7    -4    12
10     2    -5
13     8    11
5      4     1
```

Matrix C (Second through fourth rows of A):

```
9    10     2    -5
6    13     8    11
15     5     4     1
```

Matrix D (First two rows and last three columns of A):

```
7    -4    12
10     2    -5
```

Conclusion

MATLAB's indexing capabilities enable straightforward manipulation of matrices, allowing for the creation of arrays B, C, and D as required by the problem statement. These extracted subsets can be further analyzed or utilized in subsequent computational tasks, showcasing MATLAB's versatility in handling matrix operations and data extraction.

TASK 06

Solve the MATLAB problem of task 2.

```
>> x = 26
```

```
>> whos
```

```
>> y = int8(x)
```

```
>> whos
```

What other solutions are possible?

Problem Analysis

The problem involves testing MATLAB's rounding functions, namely round, fix, ceil, and floor, on a given set of floating-point numbers. These functions serve different purposes: round rounds a number to the nearest integer, fix rounds towards zero, ceil rounds towards positive infinity, and floor rounds towards negative infinity.

Algorithm

Define the array f containing floating-point numbers: $f = [-0.5, 0.1, 0.5]$.

- **Round Function (round()):**

The round() function rounds each element of f to the nearest integer. If the fractional part is exactly halfway between two integers, it rounds to the nearest even integer.

round(f) results in [-1, 0, 1].

- **Fix Function (fix()):**

The fix() function rounds each element of f towards zero, effectively truncating the fractional part.

fix(f) results in [-0, 0, 0], which is equivalent to [0, 0, 0].

- **Ceiling Function (ceil()):**

The ceil() function rounds each element of f towards positive infinity, resulting in the smallest integer greater than or equal to each element.

ceil(f) results in [-0, 1, 1], which is equivalent to [0, 1, 1].

Floor Function (floor()):

The floor() function rounds each element of f towards negative infinity, resulting in the largest integer less than or equal to each element.

floor(f) results in [-1, 0, 0].

Code

```
f = [-0.5, 0.1, 0.5];  
round(f)  
fix(f)  
ceil(f)  
floor(f)
```

Output

ans =

-1 0 1

ans =

0 0 0

ans =

0 1 1

ans =

-1 0 0

Conclusion

Understanding the behavior of these rounding functions is crucial for various applications such as mathematical modeling, statistics, and numerical analysis. Each function serves a distinct purpose and should be chosen based on the specific requirements of the problem at hand.

TASK 07

Given the following matrix:

$$A = \begin{bmatrix} -3 & 5 \\ 4 & 8 \end{bmatrix}$$

Find the following:

- 1) Column-wise sum of all elements of A using sum function; for information about sum function, type help sum in MATLAB.
- 2) Column-wise product of all elements of A using prod function; for information about prod function, type help prod in MATLAB.
- 3) Length of matrix A.
- 4) Size of matrix A.

What other solutions are possible?

Problem Analysis

We are provided with a matrix A and asked to perform several operations on it using MATLAB functions. These operations include finding the column-wise sum, column-wise product, length of the matrix, and size of the matrix.

Algorithm

Define matrix A with given elements.

Calculate column-wise sum using sum() function and store in column sum.

Calculate column-wise product using prod() function and store in column product.

Compute length of A using length() function and store in matrix length.

Determine size of A using size() function and store in matrix size.

Display the result.

Code

```
1      % Given matrix A
2 -    A = [-3, 5; 4, 8];
3
4      % 1) Column-wise sum of all elements of A using sum function
5 -    column_sum = sum(A);
6
7      % 2) Column-wise product of all elements of A using prod function
8 -    column_product = prod(A);
9
10     % 3) Length of matrix A (number of elements along the longest dimension)
11 -    length_A = length(A);
12
13     % 4) Size of matrix A (number of rows and columns)
14 -    size_A = size(A);
15
16     % Display the results
17 -    disp('Column-wise sum of all elements of A:');
18 -    disp(column_sum);
19
20 -    disp('Column-wise product of all elements of A:');
21 -    disp(column_product);
22
23 -    disp('Length of matrix A:');
24 -    disp(length_A);
25
26 -    disp('Size of matrix A (number of rows and columns):');
27 -    disp(size_A);
```

Output

Command Window

Column-wise sum of all elements of A:

1 13

Column-wise product of all elements of A:

-12 40

Length of matrix A:

2

Size of matrix A (number of rows and columns):

2 2

Column-wise sum of all elements of A:

1 13

Column-wise product of all elements of A:

-12 40

Length of matrix A:

2

Size of matrix A (number of rows and columns):

2 2

Conclusion

It outlines the steps to perform the specified operations on the given matrix A using MATLAB functions.

TASK 08

The end command is used to access the last row or column of a matrix. Use the end command to delete and update the last row and column of the following matrix.

Matrix A = [3 23 34 12 34 5 56 23; 12 34 34 32 23 23 45 1; 67 23

2 4 4 5 6 456; 4 5 1 1 2 34 45 56; 67 67 45 67 78 7 8 5; 6 35

5 3 5 56 7 8]

Hint: For deleting a column use A(3, :)=[];

Problem Analysis

We are provided with a matrix A and asked to perform several operations on it using MATLAB functions. These operations include finding the column-wise sum, column-wise product, length of the matrix, and size of the matrix.

Algorithm

1. Initialize the matrix A with the given elements.
2. To delete the last row of matrix
3. A, use the end command in MATLAB.
4. Syntax: A(end, :) = [].
5. To delete the last column of matrix
6. A, again use the end command.
7. Syntax: A(:, end) = [].
8. This removes the last column of the matrix.
9. Display the updated matrix after deleting the last row and column.

Code

```
% Given matrix A
A = [3 23 34 12 34 5 56 23;
     12 34 34 32 23 23 45 1;
     67 23 2 4 4 5 6 456;
     4 5 1 1 2 34 45 56;
     67 67 45 67 78 7 8 5;
     6 35 5 3 5 56 7 8];

% Delete last row
A(end, :) = [];

% Delete last column
A(:, end) = [];

% Display updated matrix
disp('Updated Matrix A:');
disp(A);
```

Output

```
Updated Matrix A:
      3      23      34      12      34      5      56
     12      34      34      32      23      23      45
     67      23       2       4       4       5       6
      4       5       1       1       2      34      45
     67      67      45      67      78       7       8
```


Conclusion

This algorithm outlines the steps to delete and update the last row and column of a matrix using the end command in MATLAB. Following these steps will efficiently modify the matrix as required.

TASK 09

Try the following commands in MATLAB and comment on them:

(i) `A(3,end)`

(ii) `A(:)`

(iii) `A(: , end)`

(iv) `Y = linspace(20,100)`

(v) `Y = linspace(20,100,50)`

(i) `A(3,end)`:

- This command accesses the element located in the third row and the last column of matrix *A*.
- It returns the value at the specified location.

(ii) `A(:)`:

- This command reshapes the matrix *A* into a column vector by stacking all its elements vertically.
- It returns all the elements of matrix *A* in a single column.

(iii) `A(: , end)`:

- This command selects all rows of matrix *A* but only the elements in the last column.
- It returns a column vector containing the elements of the last column of matrix *A*.

(iv) `Y = linspace(20,100)`:

- This command generates a linearly spaced vector *Y* containing 100 points between 20 and 100 (inclusive).
- It evenly divides the interval [20, 100] into 100 points by default.

(v) `Y = linspace(20,100,50)`:

- This command generates a linearly spaced vector *Y* containing 50 points between 20 and 100 (inclusive).
- It evenly divides the interval [20, 100] into 50 points as specified.

Conclusion

1. Commands (i), (ii), and (iii) are used for indexing and accessing specific elements or subsets of a matrix in MATLAB.
2. Commands (iv) and (v) are used to generate linearly spaced vectors with specified start and end points and a specified number of points in between.
3. These commands are useful for various tasks in MATLAB such as data manipulation, indexing, and generating data for plotting or analysis.

TASK 10

Use the inverse `inv(A)` function to find the inverse of **A** for finding the unknowns for the Linear equation.

$$x + 2y + 3z = 1$$

$$4x + 5y + 6z = 2$$

$$7x + 8y = 1$$

where:

$$X = [x \ y \ z]$$

$$A = [1 \ 2 \ 3; 4 \ 5 \ 6; 7 \ 8 \ 9]$$

$$b = [1; 2; 3]$$

$$A * x = b$$

Hint:

$$x = \text{inv}(A) * b \text{ or } x = A \backslash b$$

Problem Analysis

The provided code aims to solve a system of linear equations represented by the matrix equation.

$$Ax = b,$$

Where **A** is a coefficient matrix, **x** is a vector of unknowns, and **b** is a vector of constants. The code attempts to solve for **x** using MATLAB.

Algorithm

1. Define Matrix **A** and Vector **b**:
2. Initialize the coefficient matrix **A** and the vector of constants **b**.
3. Solve for **x** using Matrix Left Division:
4. Use the matrix left division operator `\` to solve the system of linear equations.
5. Syntax: `solution = A \ b`.
6. Display the Solution:

Code

```
% Define matrix A and vector b
A = [1 2 3; 4 5 6; 7 8 9];
b = [1; 2; 3];

% Calculate the solution using A\b (matrix left division)
x_solution = A \ b;

% Display the solution
disp('Solution using A\b:');
disp(x_solution);
```

Output

```
Command Window
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.541976e-018.
> In Task11 at 6
Solution using A\b:
-0.3333
0.6667
0
```

Conclusion

This algorithm outlines the steps to find inverse of Linear equations using MATLAB.

TASK 11

Solve Task 10 by taking the equations from the user.

Hint: Take the matrix A and b from the user.

Problem Analysis

The provided code aims to solve a system of linear equations represented by the matrix equation.

$$Ax=b,$$

Where A is a coefficient matrix, and matrix coefficients will be taken from user.

Algorithm

1. Prompt User for Matrix A and Vector
2. Calculate Solution using Matrix Left Division.
3. Display the Solution.

Code

```
1 % Prompt the user to enter the coefficients of matrix A
2 - disp('Enter the coefficients of matrix A (each row separated by spaces):');
3 - A = input('Matrix A: ');
4
5 % Prompt the user to enter the constants of vector b
6 - disp('Enter the constants of vector b (each value separated by spaces):');
7 - b = input('Vector b: ');
8
9 % Calculate the solution using A\b (matrix left division)
10 - x_solution = A \ b;
11
12 % Display the solution
13 - disp('Solution using A\b:');
14 - disp(x_solution);
```

Output

```
Command Window
Enter the coefficients of matrix A (each row separated by spaces):
Matrix A: [1 2 3; 4 5 6; 7 8 9]
Enter the constants of vector b (each value separated by spaces):
Vector b: [2; 4; 6]
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.541976e-018.
> In Task11 at 10
Solution using A\b:
    -0.6667
     1.3333
         0
```

Conclusion

The algorithm outlined a step-by-step approach to solve a system of linear equations by taking the matrix A and vector b from the user as input.

It first prompts the user to enter the coefficients of matrix A and the constants of vector b.

Then, it utilizes MATLAB's matrix left division operator \ to efficiently solve the system of linear equations.

Finally, it displays the solution vector x obtained from the calculation.

This approach provides a user-friendly and interactive method for solving linear systems, allowing users to input their own coefficients and constants to find the solution.