LA MATLAB Programs with Sample MCQs

1. Gaussian Elimination

Gaussian elimination transforms a system *Ax=b* into an upper triangular system using row operations, then solves it via back substitution.

The steps are:

1.Form the augmented matrix: Combine *A* and *b*.

2.Forward elimination: Zero out elements below the pivot in each column.

3.Back substitution: Solve for the unknowns from the upper triangular system.

Example System

Consider the system:

$$\begin{cases} 2x_1 + x_2 - x_3 = 8 \\ -3x_1 - x_2 + 2x_3 = -11 \\ -2x_1 + x_2 + 2x_3 = -3 \end{cases}$$

Matrix form:

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

```
% Define the matrix A and vector b
A = [2 1 -1; -3 -1 2; -2 1 2];
b = [8; -11; -3];
n = length(b);
% Form the augmented matrix
aug = [A b];
disp('Augmented matrix [A|b]:');
disp(aug);
```

```
% Forward Elimination
for k = 1:n-1 % Loop over pivot columns
% Check for zero pivot
if aug(k,k) == 0
error('Zero pivot encountered. Consider pivoting.');
end
for i = k+1:n % Loop over rows below pivot
% Compute multiplier to zero out aug(i,k)
multiplier = aug(i,k) / aug(k,k);
% Update row i: subtract multiplier * row k
aug(i,k:n+1) = aug(i,k:n+1) - multiplier * aug(k,k:n+1);
end
disp(['After eliminating column ', num2str(k), ':']);
disp(aug);
end

% Back Substitution
x = zeros(n,1); % Initialize solution vector
x(n) = aug(n,end) / aug(n,n); % Solve last variable
for i = n-1:-1:1 % Work upwards
% Solve for x(i) using known x(i+1), ..., x(n)
x(i) = (aug(i,end) - aug(i,i+1:n) * x(i+1:n)) / aug(i,i);
end

disp('Solution vector x:');
disp(x);

% Verify solution
disp('Verification: A*x - b =');
disp(A*x - b);
```

Explanation of Each Step

1. **Setup**:

   - Define A and b as MATLAB arrays.
   - n is the system size (number of equations).
   - Create the augmented matrix $\text{aug} = [A \mid b]$, which is A with b as an extra column.
   - **Output**: Displays the initial augmented matrix, e.g.:

$$\begin{bmatrix} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{bmatrix}$$

2. **Forward Elimination**:

   - **Outer loop (k)**: Iterates over pivot columns (1 to $n-1$).
   - **Pivot check**: Ensures the pivot $\text{aug}(k,k) \neq 0$. (Note: This code assumes non-zero pivots for simplicity; in practice, partial pivoting may be needed.)
   - **Inner loop (i)**: For each row i below the pivot row k:
     - Compute the multiplier: $\text{multiplier} = \frac{\text{aug}(i,k)}{\text{aug}(k,k)}$.
     - Update row i: Subtract $\text{multiplier} \times \text{row } k$ from row i to zero out $\text{aug}(i,k)$.
     - Update columns k to $n+1$ (including b).

- **After column 1**: Zero out below $\text{aug}(1,1) = 2$.
  - Row 2: Multiplier = $\frac{-3}{2} = -1.5$. New row 2 = row 2 - (-1.5) * row 1.
  - Row 3: Multiplier = $\frac{-2}{2} = -1$. New row 3 = row 3 - (-1) * row 1.
  - Result:

$$\begin{bmatrix} 2 & 1 & -1 & 8 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 2 & 1 & 5 \end{bmatrix}$$

- **After column 2**: Zero out below $\text{aug}(2,2) = 0.5$.
  - Row 3: Multiplier = $\frac{2}{0.5} = 4$. New row 3 = row 3 - 4 * row 2.
  - Result:

$$\begin{bmatrix} 2 & 1 & -1 & 8 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

4. **Verification**:
   - Compute $Ax - b$ to check if the solution is correct. Ideally, this is zero (or near zero due to numerical precision).
   - For our solution:

$$Ax = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix} = \begin{bmatrix} 4+3+1 \\ -6-3+(-2) \\ -4+3+(-2) \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix} = b$$

   Thus, $Ax - b = [0, 0, 0]$.

2. Gauss - Jordan Method

Overview of Gauss-Jordan for Matrix Inversion

To find the inverse of a square matrix A, we:

1. Form an augmented matrix [A ∣ I], where I is the identity matrix of the same size.
2. Apply Gauss-Jordan elimination to transform the left side (A) into the identity matrix.
3. If successful, the right side becomes A^{-1}, resulting in [I ∣ A^{-1}] .

4. The matrix is invertible if it can be reduced to the identity matrix (i.e., it's non-singular).

Example Matrix

We'll use the matrix A from your previous question:

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}$$

We aim to find A^{-1}, such that $AA^{-1} = I$, where:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
% Define the matrix A
A = [2 1 -1; -3 -1 2; -2 1 2];
n = size(A, 1); % Size of the matrix

% Form the augmented matrix [A | I]
I = eye(n); % Identity matrix
aug = [A I];
disp('Initial augmented matrix [A | I]:');
disp(aug);

% Gauss-Jordan Elimination
for k = 1:n % Loop over pivot columns
    % Check for zero pivot
    if aug(k,k) == 0
        error('Zero pivot encountered. Matrix may be singular.');
    end

    % Normalize pivot row: Make pivot = 1
    aug(k,:) = aug(k,:) / aug(k,k);
    disp(['After normalizing row ', num2str(k), ':']);
    disp(aug);

    % Eliminate above and below pivot in column k
    for i = 1:n
        if i ~= k % Skip the pivot row
            multiplier = aug(i,k);
            aug(i,:) = aug(i,:) - multiplier * aug(k,:);
```

```
        end
    end
    disp(['After eliminating column ', num2str(k), ':']);
    disp(aug);
end

% Extract the inverse
A_inv = aug(:,n+1:end);
disp('Inverse matrix A^(-1):');
disp(A_inv);

% Verify the inverse
disp('Verification: A * A^(-1) (should be identity):');
disp(A * A_inv);
disp('Verification: A^(-1) * A (should be identity):');
disp(A_inv * A);
```

Explanation of Each Step

1. **Setup**:

- Define A, a 3x3 matrix.
- Compute n, the matrix size.
- Create the identity matrix $I = \mathrm{eye}(n)$.
- Form the augmented matrix $\mathrm{aug} = [A \mid I]$, which is 3x6 for a 3x3 matrix.
- **Output**: Initial augmented matrix:

$$\begin{bmatrix} 2 & 1 & -1 & | & 1 & 0 & 0 \\ -3 & -1 & 2 & | & 0 & 1 & 0 \\ -2 & 1 & 2 & | & 0 & 0 & 1 \end{bmatrix}$$

2. **Gauss-Jordan Elimination**:

- **Loop over pivot columns ($k = 1$ to n)**:
  - **Pivot check**: Ensure $\mathrm{aug}(k, k) \neq 0$. A zero pivot suggests the matrix may be singular (non-invertible).
  - **Normalize pivot row**: Divide row k by $\mathrm{aug}(k, k)$ to make the pivot 1.
  - **Eliminate above and below**: For all rows $i \neq k$, subtract $\mathrm{multiplier} \times \mathrm{row}\ k$ from row i, where $\mathrm{multiplier} = \mathrm{aug}(i, k)$, to zero out column k.

- **Step 1: Pivot column 1 ($k = 1$)**:
  - **Normalize row 1**: Pivot is $\text{aug}(1, 1) = 2$. Divide row 1 by 2:

$$\text{Row } 1 = \frac{[2, 1, -1, 1, 0, 0]}{2} = [1, 0.5, -0.5, 0.5, 0, 0]$$

    Matrix:

$$\begin{bmatrix} 1 & 0.5 & -0.5 & 0.5 & 0 & 0 \\ -3 & -1 & 2 & 0 & 1 & 0 \\ -2 & 1 & 2 & 0 & 0 & 1 \end{bmatrix}$$

- **Eliminate column 1**:
  - Row 2: Multiplier = -3. Row 2 = row 2 - (-3) * row 1:

$$[-3, -1, 2, 0, 1, 0] + 3 \cdot [1, 0.5, -0.5, 0.5, 0, 0] = [0, 0.5, 0.5, 1.5, 1, 0]$$

  - Row 3: Multiplier = -2. Row 3 = row 3 - (-2) * row 1:

$$[-2, 1, 2, 0, 0, 1] + 2 \cdot [1, 0.5, -0.5, 0.5, 0, 0] = [0, 2, 1, 1, 0, 1]$$

  - Result:

$$\begin{bmatrix} 1 & 0.5 & -0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 1.5 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 & 1 \end{bmatrix}$$

- **Step 2: Pivot column 2 ($k = 2$)**:
  - **Normalize row 2**: Pivot is $\text{aug}(2, 2) = 0.5$. Divide row 2 by 0.5:

$$\text{Row 2} = \frac{[0, 0.5, 0.5, 1.5, 1, 0]}{0.5} = [0, 1, 1, 3, 2, 0]$$

    Matrix:

$$\begin{bmatrix} 1 & 0.5 & -0.5 & 0.5 & 0 & 0 \\ 0 & 1 & 1 & 3 & 2 & 0 \\ 0 & 2 & 1 & 1 & 0 & 1 \end{bmatrix}$$

- **Eliminate column 2**:
  - Row 1: Multiplier = 0.5. Row 1 = row 1 - 0.5 * row 2:

$$[1, 0.5, -0.5, 0.5, 0, 0] - 0.5 \cdot [0, 1, 1, 3, 2, 0] = [1, 0, -1, -1, -1, 0]$$

  - Row 3: Multiplier = 2. Row 3 = row 3 - 2 * row 2:

$$[0, 2, 1, 1, 0, 1] - 2 \cdot [0, 1, 1, 3, 2, 0] = [0, 0, -1, -5, -4, 1]$$

  - Result:

$$\begin{bmatrix} 1 & 0 & -1 & -1 & -1 & 0 \\ 0 & 1 & 1 & 3 & 2 & 0 \\ 0 & 0 & -1 & -5 & -4 & 1 \end{bmatrix}$$

- **Step 3: Pivot column 3 ($k = 3$)**:
  - **Normalize row 3**: Pivot is $\text{aug}(3, 3) = -1$. Divide row 3 by -1:

$$\text{Row 3} = \frac{[0, 0, -1, -5, -4, 1]}{-1} = [0, 0, 1, 5, 4, -1]$$

    Matrix:

$$\begin{bmatrix} 1 & 0 & -1 & -1 & -1 & 0 \\ 0 & 1 & 1 & 3 & 2 & 0 \\ 0 & 0 & 1 & 5 & 4 & -1 \end{bmatrix}$$

- **Eliminate column 3**:
  - Row 1: Multiplier = -1. Row 1 = row 1 - (-1) * row 3:

$$[1, 0, -1, -1, -1, 0] + 1 \cdot [0, 0, 1, 5, 4, -1] = [1, 0, 0, 4, 3, -1]$$

  - Row 2: Multiplier = 1. Row 2 = row 2 - 1 * row 3:

$$[0, 1, 1, 3, 2, 0] - 1 \cdot [0, 0, 1, 5, 4, -1] = [0, 1, 0, -2, -2, 1]$$

  - Result:

$$\begin{bmatrix} 1 & 0 & 0 & 4 & 3 & -1 \\ 0 & 1 & 0 & -2 & -2 & 1 \\ 0 & 0 & 1 & 5 & 4 & -1 \end{bmatrix}$$

3. **Extract the Inverse**:
   - The augmented matrix is now $[I \mid A^{-1}]$.
   - The right half (columns 4 to 6) is the inverse:

$$A^{-1} = \begin{bmatrix} 4 & 3 & -1 \\ -2 & -2 & 1 \\ 5 & 4 & -1 \end{bmatrix}$$

4. **Verification:**

- Compute AA^{-1}:

$$AA^{-1} = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} 4 & 3 & -1 \\ -2 & -2 & 1 \\ 5 & 4 & -1 \end{bmatrix} = \begin{bmatrix} 8-2-5 & 6-2-4 & -2+1+1 \\ -12+2+10 & -9+2+8 & 3-1-2 \\ -8-2+10 & -6-2+8 & 2+1-2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

- Compute $A^{-1}A$, which should also equal I:

$$A^{-1}A = \begin{bmatrix} 4 & 3 & -1 \\ -2 & -2 & 1 \\ 5 & 4 & -1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 8-9+2 & 4-3-1 & -4+6-2 \\ -4+6-2 & -2+2+1 & 2-4+2 \\ 10-12+2 & 5-4-1 & -5+8-2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

- Both products confirm the inverse is correct.

3. LU Decomposition

Overview of LU Decomposition

For a square matrix A, LU decomposition finds:

- L: Lower triangular matrix with 1s on the diagonal and non-zero entries only below the diagonal.
- U: Upper triangular matrix with non-zero entries on or above the diagonal.
- The goal is A=LU, where the process mirrors Gaussian elimination but stores the row operation multipliers in L.

Example Matrix

We'll use the same 3x3 matrix A from your prior questions for continuity:

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}$$

Our goal is to find L and U such that $A = LU$.

```
% Define the matrix A
A = [2 1 -1; -3 -1 2; -2 1 2];
n = size(A, 1); % Size of the matrix
```

```
% Initialize L (lower triangular, 1s on diagonal) and U (upper triangular)
L = eye(n); % L starts as identity
U = zeros(n); % U starts as zero matrix

% Copy A to U for modification
U = A;

% LU Decomposition (without pivoting)
for k = 1:n-1 % Loop over pivot columns
    % Check for zero pivot
    if U(k,k) == 0
        error('Zero pivot encountered. Decomposition requires pivoting.');
    end
    for i = k+1:n % Loop over rows below pivot
        % Compute multiplier and store in L
        L(i,k) = U(i,k) / U(k,k);
        % Update row i of U: subtract multiplier * row k
        U(i,k:n) = U(i,k:n) - L(i,k) * U(k,k:n);
    end
    disp(['After step ', num2str(k), ' - L:']);
    disp(L);
    disp(['After step ', num2str(k), ' - U:']);
    disp(U);
end

% Display final L and U
disp('Final L matrix:');
disp(L);
disp('Final U matrix:');
disp(U);

% Verify decomposition
disp('Verification: L*U - A (should be zero):');
disp(L*U - A);

% MATLAB's built-in LU decomposition (with pivoting)
[L_builtin, U_builtin, P] = lu(A);
disp('MATLAB built-in L (with pivoting):');
disp(L_builtin);
disp('MATLAB built-in U (with pivoting):');
disp(U_builtin);
disp('MATLAB built-in permutation matrix P:');
disp(P);
disp('Verification: P*A - L_builtin*U_builtin (should be zero):');
disp(P*A - L_builtin*U_builtin);
```

Explanation of Each Step

1. **Setup**:

   - Define the matrix A.
   - Initialize n as the matrix size (3 for a 3x3 matrix).
   - Initialize L as the identity matrix ($\text{eye}(n)$), since L has 1s on the diagonal.
   - Initialize U as a copy of A, which we'll modify to become upper triangular.
   - **Output**: L starts as:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   U starts as A:

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}$$

2. **LU Decomposition (Forward Elimination)**:

   - The process is similar to Gaussian elimination (from your first question), but instead of forming an augmented matrix, we:
     - Store the multipliers used to zero out sub-diagonal elements in L.
     - Transform U into an upper triangular matrix.
   - **Loop over pivot columns ($k = 1$ to $n - 1$)**:
     - **Pivot check**: Ensure $U(k, k) \neq 0$. A zero pivot halts the process unless pivoting is used.
     - **Inner loop ($i = k + 1$ to n)**: For each row below the pivot:
       - Compute the multiplier: $L(i, k) = U(i, k)/U(k, k)$.
       - Update row i of U: Subtract $L(i, k) \times \text{row } k$ from row i to zero out $U(i, k)$.
       - Columns k to n of U are updated.

- **Step 1: Pivot column 1 ($k = 1$):**
  - Pivot is $U(1,1) = 2$.
  - Row 2 ($i = 2$): Multiplier = $U(2,1)/U(1,1) = -3/2 = -1.5$. Store in $L(2,1)$.
    - Update row 2 of U: Row 2 = row 2 - (-1.5) * row 1:

$$[-3, -1, 2] - (-1.5) \cdot [2, 1, -1] = [-3 + 3, -1 + 1.5, 2 - 1.5] = [0, 0.5, 0.5]$$

  - Row 3 ($i = 3$): Multiplier = $U(3,1)/U(1,1) = -2/2 = -1$. Store in $L(3,1)$.
    - Update row 3 of U: Row 3 = row 3 - (-1) * row 1:

$$[-2, 1, 2] - (-1) \cdot [2, 1, -1] = [-2 + 2, 1 - 1, 2 + 1] = [0, 2, 3]$$

- **After Step 1**:
  - L:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

  - U:

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 2 & 3 \end{bmatrix}$$

- **Step 2: Pivot column 2 ($k = 2$):**
  - Pivot is $U(2,2) = 0.5$.
  - Row 3 ($i = 3$): Multiplier = $U(3,2)/U(2,2) = 2/0.5 = 4$. Store in $L(3,2)$.
    - Update row 3 of U: Row 3 = row 3 - 4 * row 2:

$$[0, 2, 3] - 4 \cdot [0, 0.5, 0.5] = [0, 2 - 2, 3 - 2] = [0, 0, 1]$$

- **After Step 2**:
  - L:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix}$$

  - U:

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

3. **Final Matrices**:
   - After the loop, U is upper triangular, and L is lower triangular with 1s on the diagonal.
   - The multipliers from each elimination step are stored in L, and U is the result of Gaussian elimination (like the upper triangular matrix in your first question).
   - **Output**:
     - L:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix}$$

     - U:

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

4. **Verification**:

- Compute LU:

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -1 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} = A$$

- Check $LU - A$, which should be the zero matrix (or near-zero due to numerical precision).
- **Output**: $LU - A \approx [0, 0, 0; 0, 0, 0; 0, 0, 0]$.

```
% Define the matrix A
A = [2 1 -1; -3 -1 2; -2 1 2];

% Perform LU decomposition using built-in lu function
[L, U, P] = lu(A);

% Display results
disp('Matrix A:');
disp(A);
disp('Lower triangular matrix L:');
disp(L);
disp('Upper triangular matrix U:');
disp(U);
disp('Permutation matrix P:');
disp(P);
```

```
Matrix A:

     2     1    -1
    -3    -1     2
    -2     1     2

Lower triangular matrix L:

    1.0000         0         0
    0.6667    1.0000         0
   -0.6667    0.2000    1.0000

Upper triangular matrix U:

   -3.0000   -1.0000    2.0000
         0    1.6667    0.6667
         0         0    0.2000

Permutation matrix P:

     0     1     0
     0     0     1
     1     0     0
```

4. Four Fundamental Subspaces

```
clc;
clear all;
close all;
% Bases of four fundamental vector spaces of matrix A.
A=[1,2,3;2,-1,1];
% Row Reduced Echelon Form
[R, pivot] = rref(A)
% Rank
rank = length(pivot)
% basis of the column space of A
columnsp = A(:,pivot)
% basis of the nullspace of A
nullsp = null(A,'r')
% basis of the row space of A
rowsp = R(1:rank,:)'
% basis of the left nullspace of A
leftnullsp = null(A','r')
```

```
R = 2×3
       1     0     1
       0     1     1

pivot = 1×2
       1     2

rank = 2
columnsp = 2×2
       1     2
       2    -1

nullsp = 3×1
      -1
      -1
       1

rowsp = 3×2
       1     0
       0     1
       1     1

leftnullsp =

  2×0 empty double matrix
```

5. Gram-Schmidt Orthogonalization

Overview of Gram-Schmidt Orthogonalization

Given a set of linearly independent vectors {v1,v2,…,vn}, Gram-Schmidt produces an orthogonal set {u1,u2,…,un} (or orthonormal set {q1,q2,…,qn}) spanning the same subspace. The process:

1. Takes each vector and subtracts its projections from the previously computed orthogonal vectors to make it orthogonal to them.
2. Optionally normalizes the vectors to make them orthonormal (unit length).

Example Matrix

We'll use the same 3x3 matrix A from your previous questions:

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}$$

The columns of A are:

$$a_1 = \begin{bmatrix} 2 \\ -3 \\ -2 \end{bmatrix}, \quad a_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \quad a_3 = \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}$$

Since A is invertible (as shown in your LU decomposition and inverse questions), its columns are linearly independent, and we expect a full basis for $C(A) = \mathbb{R}^3$.

```
% Define the matrix A
A = [2 1 -1; -3 -1 2; -2 1 2];
[m, n] = size(A);

% Initialize matrices for orthogonal and orthonormal vectors
U = zeros(m, n); % Orthogonal basis
Q = zeros(m, n); % Orthonormal basis

% Gram-Schmidt Orthogonalization
for k = 1:n
    % Start with the k-th column of A
    U(:, k) = A(:, k);

    % Subtract projections onto previous orthogonal vectors
    for j = 1:k-1
        U(:, k) = U(:, k) - (dot(U(:, j), A(:, k)) / dot(U(:, j), U(:, j))) * U(:, j);
    end

    % Normalize for orthonormal basis
    Q(:, k) = U(:, k) / norm(U(:, k));

end

% Display final results
disp('Orthogonal basis U (columns are u_1, u_2, u_3):');
disp(U);
```

```
disp('Orthonormal basis Q (columns are q_1, q_2, q_3):');
disp(Q);

% Verification
% Check orthogonality of U
disp('Verification: Dot products of orthogonal vectors (should be zero for i != j):');
disp('u_1^T * u_2:'); disp(dot(U(:, 1), U(:, 2)));
disp('u_1^T * u_3:'); disp(dot(U(:, 1), U(:, 3)));
disp('u_2^T * u_3:'); disp(dot(U(:, 2), U(:, 3)));

% Compare with MATLAB's orth function
orth_basis = orth(A);
disp('MATLAB orth(A) basis for C(A):');
disp(orth_basis);
```

```
Orthogonal basis U (columns are u_1, u_2, u_3):

    2.0000    0.6471    0.1190
   -3.0000   -0.4706    0.0952
   -2.0000    1.3529   -0.0238

Orthonormal basis Q (columns are q_1, q_2, q_3):

    0.4851    0.4117    0.7715
   -0.7276   -0.2994    0.6172
   -0.4851    0.8608   -0.1543

Verification: Dot products of orthogonal vectors (should be zer

u_1^T * u_2:
  -4.4409e-16

u_1^T * u_3:
   6.6613e-16

u_2^T * u_3:
  -2.9143e-16


MATLAB orth(A) basis for C(A):

   -0.4546   -0.4550   -0.7657
    0.7256    0.3093   -0.6147
    0.5165   -0.8350    0.1895
```

Detailed Explanation of Each Step

1. **Setup**:

   - Define A, a 3x3 matrix.
   - Get dimensions: $m = 3$ (rows), $n = 3$ (columns).
   - Initialize:

     - U: Matrix to store orthogonal vectors (u_1, u_2, u_3).
     - Q: Matrix to store orthonormal vectors (q_1, q_2, q_3).

   - **Goal**: Apply Gram-Schmidt to A's columns to produce U (orthogonal) and Q (orthonormal) bases for $C(A)$.

2. **Gram-Schmidt Process**:

   - **Loop over columns ($k = 1$ to n)**:

     - Start with $u_k = a_k$, the k-th column of A.
     - Subtract projections of a_k onto all previous orthogonal vectors $u_1, \ldots, u_{k-1}$.
     - Formula for orthogonalization:

$$u_k = a_k - \sum_{j=1}^{k-1} \text{proj}_{u_j}(a_k), \quad \text{where} \quad \text{proj}_{u_j}(a_k) = \frac{u_j^T a_k}{u_j^T u_j} u_j$$

     - Normalize to get orthonormal vector:

$$q_k = \frac{u_k}{\|u_k\|}$$

- **Step 1: Process a_1 ($k = 1$)**:
  - Set $u_1 = a_1 = [2, -3, -2]^T$.
  - No previous vectors, so no projections to subtract.
  - Normalize:

$$\|u_1\| = \sqrt{2^2 + (-3)^2 + (-2)^2} = \sqrt{4 + 9 + 4} = \sqrt{17} \approx 4.1231$$

$$q_1 = \frac{u_1}{\|u_1\|} = \frac{[2, -3, -2]}{\sqrt{17}} \approx [0.4851, -0.7276, -0.4851]^T$$

  - **Output**:
    - u_1: $[2, -3, -2]$
    - q_1: $[0.4851, -0.7276, -0.4851]$

- **Step 2: Process a_2 ($k = 2$)**:
  - Set $u_2 = a_2 = [1, -1, 1]^T$.
  - Subtract projection onto u_1:

$$u_1^T a_2 = [2, -3, -2] \cdot [1, -1, 1] = 2 + 3 - 2 = 3$$

$$u_1^T u_1 = [2, -3, -2] \cdot [2, -3, -2] = 4 + 9 + 4 = 17$$

$$\text{proj}_{u_1}(a_2) = \frac{3}{17} u_1 = \frac{3}{17}[2, -3, -2] \approx [0.3529, -0.5294, -0.3529]$$

$$u_2 = a_2 - \text{proj}_{u_1}(a_2) = [1, -1, 1] - [0.3529, -0.5294, -0.3529] \approx [0.6471, -0.4706, 1.3529]$$

  - Normalize:

$$\|u_2\| = \sqrt{(0.6471)^2 + (-0.4706)^2 + (1.3529)^2} \approx \sqrt{0.4188 + 0.2215 + 1.8299} \approx \sqrt{2.4702} \approx 1.5717$$

$$q_2 = \frac{u_2}{\|u_2\|} \approx \frac{[0.6471, -0.4706, 1.3529]}{1.5717} \approx [0.4118, -0.2994, 0.8607]^T$$

- **Output**:
  - u_2: $[0.6471, -0.4706, 1.3529]$
  - q_2: $[0.4118, -0.2994, 0.8607]$

- **Step 3: Process a_3 ($k = 3$)**:
  - Set $u_3 = a_3 = [-1, 2, 2]^T$.
  - Subtract projections onto u_1 and u_2:
    - Projection onto u_1:

$$u_1^T a_3 = [2, -3, -2] \cdot [-1, 2, 2] = -2 - 6 - 4 = -12$$

$$\text{proj}_{u_1}(a_3) = \frac{-12}{17} u_1 \approx -0.7059 \cdot [2, -3, -2] \approx [-1.4118, 2.1176, 1.4118]$$

    - Projection onto u_2:

$$u_2^T a_3 = [0.6471, -0.4706, 1.3529] \cdot [-1, 2, 2] \approx -0.6471 - 0.9412 + 2.7058 \approx 1.1176$$

$$u_2^T u_2 = [0.6471, -0.4706, 1.3529] \cdot [0.6471, -0.4706, 1.3529] \approx 0.4188 + 0.2215 + 1.8299 \approx 2.4702$$

$$\text{proj}_{u_2}(a_3) = \frac{1.1176}{2.4702} u_2 \approx 0.4526 \cdot [0.6471, -0.4706, 1.3529] \approx [0.2929, -0.2129, 0.6122]$$

  - Compute:

$$u_3 = a_3 - \text{proj}_{u_1}(a_3) - \text{proj}_{u_2}(a_3)$$

$$\approx [-1, 2, 2] - [-1.4118, 2.1176, 1.4118] - [0.2929, -0.2129, 0.6122]$$

$$\approx [-1 + 1.4118 - 0.2929, 2 - 2.1176 + 0.2129, 2 - 1.4118 - 0.6122] \approx [0.1179, 0.0953, -0.0240]$$

- Normalize:

$$\|u_3\| = \sqrt{(0.1179)^2 + (0.0953)^2 + (-0.0240)^2} \approx \sqrt{0.0139 + 0.0091 + 0.0006} \approx \sqrt{0.0236} \approx 0.1536$$

$$q_3 = \frac{u_3}{\|u_3\|} \approx \frac{[0.1179, 0.0953, -0.0240]}{0.1536} \approx [0.7674, 0.6204, -0.1562]^T$$

- **Output**:
  - u_3: $[0.1179, 0.0953, -0.0240]$
  - q_3: $[0.7674, 0.6204, -0.1562]$

3. **Final Bases:**

- **Orthogonal basis U:**

$$U = \begin{bmatrix} 2 & 0.6471 & 0.1179 \\ -3 & -0.4706 & 0.0953 \\ -2 & 1.3529 & -0.0240 \end{bmatrix}$$

  Columns u_1, u_2, u_3 are orthogonal ($u_i^T u_j = 0$ for $i \neq j$).

- **Orthonormal basis Q:**

$$Q = \begin{bmatrix} 0.4851 & 0.4118 & 0.7674 \\ -0.7276 & -0.2994 & 0.6204 \\ -0.4851 & 0.8607 & -0.1562 \end{bmatrix}$$

  Columns q_1, q_2, q_3 are orthonormal ($q_i^T q_j = \delta_{ij}$, i.e., 1 if $i = j$, 0 otherwise).

- **Output:**
  - Displays U and Q, confirming the orthogonal and orthonormal sets.

6. QR Decomposition

QR decomposition is used in numerical linear algebra for solving linear systems, least squares problems, eigenvalue computations, and more. Given an m×n matrix A:

- Q is an m×m m \times m m×m orthogonal matrix (i.e., $Q^TQ=I$, where I is the identity matrix).
- R is an m×n upper triangular matrix (all entries below the main diagonal are zero).
- The product QR reconstructs the original matrix A.

```
% Define the matrix
A = [1 2 3; 4 5 6; 7 8 9; 10 11 12];
disp('Original matrix A:');
disp(A);

% Full QR decomposition
[Q, R] = qr(A);
disp('Orthogonal matrix Q (full):');
disp(Q);
disp('Upper triangular matrix R (full):');
disp(R);
```

```
% Verify full QR
A_reconstructed = Q * R;
disp('Reconstructed A (Q * R):');
disp(A_reconstructed);

% Check orthogonality
disp('Orthogonality check for Q (full): Q''*Q');
disp(Q' * Q);
```

```
Original matrix A:

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

Orthogonal matrix Q (full):

   -0.0776   -0.8331    0.5336    0.1236
   -0.3105   -0.4512   -0.8036    0.2329
   -0.5433   -0.0694    0.0065   -0.8366
   -0.7762    0.3124    0.2636    0.4801

Upper triangular matrix R (full):

  -12.8841  -14.5916  -16.2992
         0   -1.0413   -2.0826
         0         0   -0.0000
         0         0         0

Reconstructed A (Q * R):

    1.0000    2.0000    3.0000
    4.0000    5.0000    6.0000
    7.0000    8.0000    9.0000
   10.0000   11.0000   12.0000

Orthogonality check for Q (full): Q'*Q
    1.0000    0.0000   -0.0000         0
    0.0000    1.0000   -0.0000   -0.0000
   -0.0000   -0.0000    1.0000    0.0000
         0   -0.0000    0.0000    1.0000
```

Projection matrices and least squares

Find the projection for the matrix $A=\begin{pmatrix}1&0\\0&1\\1&1\end{pmatrix}$ **;** $x=\begin{pmatrix}u\\v\end{pmatrix}$ **and** $b=\begin{pmatrix}1\\3\\4\end{pmatrix}$.

Code:

```
A=[1,0;0,1;1,1]
b=[1;3;4]
x = lsqr(A,b)
```

Output:

```
A =
        1   0
        0   1
        1   1
b =
        1
        3
        4
lsqr converged at iteration 2 to a solution with relative residual 6.7e-17.
x =
        1.0000
        3.0000
```

Find the point on a plane x+y-z=0 that is closest to (2,1,0)

Code:

```
syms   c
P=[2,1,0]+c*[1,1,-1]
s=1*(c+2)+1*(c+1)-1(-c)==0
s1=solve(s,c)
p=[2,1,0]+s1*[1,1,-1]
```

Output

```
P =
        [3*c + 1, 4*c, c + 1]

s =
        26*c + 4 == 1

s1 =
        -3/26

p =
        [17/26, -6/13, 23/26]
```

Let $u = \begin{bmatrix} 1 \\ 7 \end{bmatrix}$ onto $v = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$ and find P, the matrix that will project any matrix onto the vector v. Use the result to find $proj_v$ u.

Code:

```
u=[1;7]
u =
      1
      7
v=[-4;2]
v =   -4
      2
P=(v*transpose(v))/(transpose(v)*v)

P =
        0.8000      -0.4000
        -0.4000      0.2000

P*u

ans =
        -2
         1
```

The Pseudo-inverse A^+ of a mxn matrix A is an extension of the inverse of a square matrix to non-square matrices and to singular(non-invertible) square matrices.

The Pseudo-inverse matrix A^+ is a nxm matrix with the following properties:

1) If $m \geq n$, then A^+A is invertible and $A^+=(A^TA)^{-1}A^T$ and so $A^+A=I$, A^+ is left inverse of A.
2) If $m \leq n$, then AA^+ is invertible and $A^+=A^T(AA^T)^{-1}$ and so $AA^+=I$, A^+ is right inverse of A.

In other words,

Case 1: If $\rho(A) = n$ (n is the number of columns), Ax = b has at most one solution x for every b if and only if the columns are linearly independent. Then A will have left inverse of order n x m such that $B_{nxm}A_{mxn}=I_{nxn}$
Thus, $B=(A^TA)^{-1}A^T$

Case 2: If $\rho(A) = m$ (m is the number of rows), Ax = b has at least one solution x for every b if and only if the columns span R^m. Then A will have right inverse of order m x n such that $A_{mxn}C_{nxm}=I_{mxm}$
Thus, $C=A^T(AA^T)^{-1}$

Example 1:

$$A=\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \end{bmatrix}_{2x3}$$

Rank r=2=m (m<n)

$$C=A^T(AA^T)^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & 4 \\ 0 & 0 \end{bmatrix}_{3x2} \begin{bmatrix} 1/4 & 0 \\ 0 & 1/16 \end{bmatrix}_{2x2} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/4 \\ 0 & 0 \end{bmatrix}_{3x2}$$

Example 2:

$$A=\begin{bmatrix}1 & 0\\1 & 1\\0 & 1\end{bmatrix}_{3x2}$$

Rank r=2=n (n<m)

$$B=(A^TA)^{-1}A^T=\begin{bmatrix}2/3 & -1/3\\-1/3 & 2/3\end{bmatrix}_{2x2}\begin{bmatrix}1 & 1 & 1\\0 & 1 & 1\end{bmatrix}_{2x3}=\begin{bmatrix}2/3 & 1/3 & -1/3\\-1/3 & 1/3 & 2/3\end{bmatrix}_{2x3}$$

Matlab code using in-built function:

```
% Define a matrix
A = [1, 2, 3; 4, 5, 6];
% Compute the pseudo-inverse of A
A_pseudo_inv = pinv(A);
% Display the pseudo-inverse
disp('Pseudo-inverse of A:')
disp(A_pseudo_inv)
```

Singular Value Decomposition (SVD)

Use in-built function in Matlab:

```
A=[2 3 4; 4 5 6];
[U, S, V] = svd(A)
```

Output:

```
U =
 -0.5221 -0.8529
 -0.8529  0.5221
```

S =

10.2846	0	0
0	0.4763	0

V =

-0.4332	0.8035	0.4082
-0.5669	0.1092	-0.8165
-0.7006	-0.5852	0.4082

Sample MCQs

1	**1. Gaussian Elimination (Theory)** What is the primary purpose of the forward elimination step in Gaussian elimination? A) To solve for the unknowns directly B) To transform the matrix into a lower triangular form C) To zero out elements below the pivot in each column D) To normalize the pivot elements to 1 **Answer**: C) To zero out elements below the pivot in each column

2	**2. Gaussian Elimination (MATLAB)** In the MATLAB code for Gaussian elimination, what does the line `aug = [A b];` do? A) Initializes the solution vector B) Forms the augmented matrix [A∣b] C) Checks for zero pivots D) Performs back substitution **Answer:** B) Forms the augmented matrix [A∣b]
3	**3. Gaussian Elimination (Theory)** What happens if a zero pivot is encountered during forward elimination without pivoting? A) The algorithm continues without issues B) The algorithm stops with an error C) The solution is automatically zero D) The matrix is inverted instead **Answer:** B) The algorithm stops with an error
4	**4. Gauss-Jordan Method (Theory)** What is the key difference between Gaussian elimination and the Gauss-Jordan method for matrix inversion? A) Gauss-Jordan transforms [A∣I] into [I∣A^{-1}] B) Gauss-Jordan only works for singular matrices C) Gaussian elimination normalizes pivot rows D) Gauss-Jordan skips back substitution **Answer:** A) Gauss-Jordan transforms [A∣I] into [I∣A^{-1}]
5	**5. Gauss-Jordan Method (MATLAB)** In the Gauss-Jordan MATLAB code, what does `aug(k,:) = aug(k,:)/aug(k,k);` accomplish? A) Zeros out elements below the pivot B) Normalizes the pivot row to make the pivot equal to 1 C) Computes the inverse matrix directly D) Subtracts the pivot row from other rows **Answer:** B) Normalizes the pivot row to make the pivot equal to 1

6	**6. LU Decomposition (Theory)** In LU decomposition, what is the role of the matrix L? A) Stores the upper triangular part of A B) Stores the multipliers used in Gaussian elimination C) Represents the inverse of A D) Contains the pivots of A **Answer:** B) Stores the multipliers used in Gaussian elimination
7	**7. LU Decomposition (MATLAB)** What does the MATLAB line `L(i,k) = U(i,k)/U(k,k);` do in LU decomposition? A) Normalizes the pivot row B) Computes and stores the multiplier in L C) Updates the upper triangular matrix U D) Checks for a zero pivot **Answer:** B) Computes and stores the multiplier in L
8	**8. Four Fundamental Subspaces (Theory)** Which subspace is represented by the basis of the null space of A? A) Column space B) Row space C) Null space D) Left null space **Answer:** C) Null space
9	**9. Four Fundamental Subspaces (MATLAB)** What does the MATLAB command `columnsp = A(:,pivot)` compute for a matrix A? A) Basis for the row space B) Basis for the column space C) Basis for the null space D) Basis for the left null space **Answer:** B) Basis for the column space

10	**10. Gram-Schmidt Orthogonalization (Theory)** What is the purpose of normalizing vectors in the Gram-Schmidt process? A) To make vectors orthogonal B) To create an orthonormal basis C) To reduce computational errors D) To ensure linear independence **Answer**: B) To create an orthonormal basis
11	**11. Gram-Schmidt Orthogonalization (MATLAB)** In the Gram-Schmidt MATLAB code, what does `Q(:,k) = U(:,k)/norm(U(:,k));` do? A) Computes the projection of a vector B) Normalizes the orthogonal vector to create an orthonormal vector C) Subtracts projections from previous vectors D) Verifies orthogonality **Answer**: B) Normalizes the orthogonal vector to create an orthonormal vector
12	**12. QR Decomposition (Theory)** In QR decomposition, what property does the matrix Q have? A) It is upper triangular B) It is orthogonal ($Q^TQ = I$) C) It is lower triangular D) It is singular **Answer**: B) It is orthogonal ($Q^TQ = I$)
13	**13. QR Decomposition (MATLAB)** What does the MATLAB command `[Q,R] = qr(A)` return for a matrix A? A) Q as a lower triangular matrix, R as an upper triangular matrix B) Q as an orthogonal matrix, R as an upper triangular matrix C) Q as an identity matrix, R as the inverse of A D) Q as a singular matrix, R as a diagonal matrix **Answer**: B) Q as an orthogonal matrix, R as an upper triangular matrix

14	**14. Projection Matrices (Theory)** What is the formula for the projection matrix P onto a vector v? A) $P = vv^T$ B) $P = (vv^T)/(v^Tv)$ C) $P = v^Tv$ D) $P = (v^Tv)/(vv^T)$ **Answer**: B) $P = (vv^T)/(v^Tv)$
15	**15. Projection Matrices (MATLAB)** In the MATLAB code for projection, what does `P*u` compute? A) The orthogonal basis of u B) The projection of u onto v C) The inverse of the projection matrix D) The norm of u **Answer**: B) The projection of u onto v
16	**16. Singular Value Decomposition (Theory)** In SVD, what does the matrix S represent? A) Orthogonal matrix of left singular vectors B) Diagonal matrix of singular values C) Orthogonal matrix of right singular vectors D) Inverse of the original matrix **Answer**: B) Diagonal matrix of singular values
17	**17. Singular Value Decomposition (MATLAB)** What does the MATLAB command `[U,S,V] = svd(A)` return? A) U and V as diagonal matrices, S as an orthogonal matrix B) U and V as orthogonal matrices, S as a diagonal matrix C) U as the inverse of A, S as the rank, V as the null space D) U and V as upper triangular matrices, S as a lower triangular matrix **Answer:** B) U and V as orthogonal matrices, S as a diagonal matrix

18	**18. Least Squares (Theory)** What does the MATLAB function `lsqr(A,b)` solve for? A) The exact solution to Ax = b B) The least squares solution to Ax ≈ b C) The inverse of A D) The eigenvalues of A **Answer:** B) The least squares solution to Ax ≈ b
19	**19. Point on a Plane (Theory)** In the problem of finding the closest point on the plane x + y - z = 0 to (2,1,0), what does the vector [1,1,-1] represent? A) The normal vector to the plane B) The point on the plane C) The solution vector D) The projection matrix **Answer:** A) The normal vector to the plane
20	**20. Verification (MATLAB)** In the Gaussian elimination MATLAB code, what should `A*x - b` equal for a correct solution? A) The identity matrix B) The zero vector C) The inverse of A D) The rank of A **Answer:** B) The zero vector