Mathematics for AI

Course Code: MEAD-603

Lab File

Submitted to: **Mr. Rahul Mishra**

Submitted by

**Shantanu Shukla**

**Enrollment No. - 01211805424**

in partial fulfillment for the award of the degree

of

*Master of Technology - AI/DS*



Center for Development of Advanced Computing, Noida

*Affiliated to Guru Gobind Singh Indraprastha University*

**Index**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1. | Write a program to take 2 matrices as input and find their product | 3 |
| 2. | Write a program to take a matrix as input and transpose it within the matrix | 6 |
| 3. | Write a program to solve a system of equations using Gauss Elimination method | 8 |
| 4. | Write a program to solve a system of equations using Gauss Jordan method | 10 |
| 5. | Write a program to find the inverse of a matrix using Identity Matrix | 12 |
| 6. | Write a program to find the Inverse of a matrix using Adjoint Matrix | 15 |
| 7. | Write a program to determine the rank of a matrix | 18 |
| 8. | Write a program to solve a system of equations using Cramer’s rule | 20 |

* **EXERCISE-1**

**Problem Statement:** Write a program to take 2 matrices as input and find their product.

**Code:**

1. # Function to take matrix input from user

2. def input\_matrix(name):

3. rows = int(input(f"Enter the number of rows for {name}: "))

4. cols = int(input(f"Enter the number of columns for {name}: "))

5.

6. matrix = []

7. print(f"Enter the elements for {name} row-wise:")

8. for i in range(rows):

9. row = list(map(int, input(f"Row {i + 1}: ").split()))

10. matrix.append(row)

11.

12. return matrix

13.

14.

15. # Function to multiply two matrices

16. def multiply\_matrices(matrix1, matrix2):

17. # Get dimensions

18. rows\_matrix1 = len(matrix1)

19. cols\_matrix1 = len(matrix1[0])

20. rows\_matrix2 = len(matrix2)

21. cols\_matrix2 = len(matrix2[0])

22.

23. # Ensure multiplication is valid

24. if cols\_matrix1 != rows\_matrix2:

25. raise ValueError(

26. "Number of columns in Matrix 1 must equal number of rows in Matrix 2"

27. )

28.

29. # Initialize the result matrix with zeros

30. result = [[0 for \_ in range(cols\_matrix2)] for \_ in range(rows\_matrix1)]

31.

32. # Perform matrix multiplication

33. for i in range(rows\_matrix1):

34. for j in range(cols\_matrix2):

35. for k in range(cols\_matrix1):

36. result[i][j] += matrix1[i][k] \* matrix2[k][j]

37.

38. return result

39.

40.

41. # Function to display matrix

42. def display\_matrix(matrix):

43. for row in matrix:

44. print(" ".join(map(str, row)))

45.

46.

47. # Main program

48. if \_\_name\_\_ == "\_\_main\_\_":

49. print("Matrix 1")

50. matrix1 = input\_matrix("Matrix 1")

51. print("\nMatrix 2")

52. matrix2 = input\_matrix("Matrix 2")

53.

54. try:

55. result = multiply\_matrices(matrix1, matrix2)

56. print("\nResultant Matrix after multiplication:")

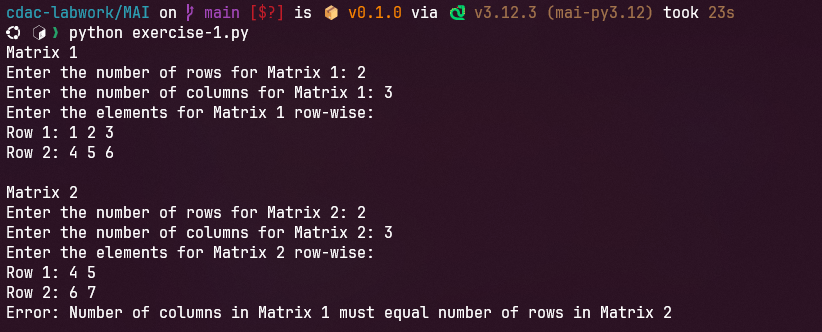
57. display\_matrix(result)

58. except ValueError as e:

59. print(f"Error: {e}")

60.

**Sample Output-1:**

****

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-3:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-2**

**Problem Statement:** Write a program to take a matrix as input and transpose it within the matrix.

**Code:**

1. # For taking input from user for m\*n matrix

2. n = int(input("Enter the number of rows and columns in nxn matrix: "))

3.

4. # For taking the matrix input by the user

5. matrix = []

6.

7. for i in range(n):

8. print(f"Enter the {i+1} row of the matrix:")

9. a = []

10. for j in range(n):

11. a.append(int(input()))

12. matrix.append(a)

13.

14.

15. # Printing the original matrix

16. print("The matrix is: ")

17. for i in range(n):

18. for j in range(n):

19. print(matrix[i][j], end=" ")

20. print()

21.

22.

23. # Here is the fuction which transpose the matrix within the same matrix

24. for i in range(n):

25. for j in range(i + 1):

26. temp = matrix[i][j]

27. matrix[i][j] = matrix[j][i]

28. matrix[j][i] = temp

29.

30.

31. # Printing the transpose of the original matrix

32. print("The matrix after being transposed is: ")

33.

34. for i in range(n):

35. for j in range(n):

36. print(matrix[i][j], end=" ")

37. print()

38.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-3:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-3**

**Problem Statement:** Write a program to solve a system of equations using Gauss Elimination method.

**Code:**

1. def gauss\_elimination():

2. n = int(input("Enter the number of unknowns: "))

3. a = []

4.

5. # Reading the Augmented Matrix

6. print("Enter the Augmented Matrix row-wise:")

7. for i in range(n):

8. b = []

9. print(f"Enter the Row {i+1}:")

10. for j in range(n + 1):

11. b.append(int(input()))

12. a.append(b)

13.

14. print("\n")

15.

16. # Printing the matrix

17. print("The matrix is: ")

18. for i in range(n):

19. for j in range(n + 1):

20. print(a[i][j], end=" ")

21. print()

22.

23. # Applying Gauss Elimination

24. for i in range(n - 1):

25. if a[i][i] == 0.0:

26. print("Mathematical error!")

27. return

28.

29. for j in range(i + 1, n):

30. c = a[j][i] / a[i][i]

31. for k in range(n + 1):

32. a[j][k] = a[j][k] - c \* a[i][k]

33.

34. # Back Substitution

35. x = [0 for \_ in range(n)]

36. x[n - 1] = a[n - 1][n] / a[n - 1][n - 1]

37.

38. for i in range(n - 2, -1, -1):

39. x[i] = a[i][n]

40. for j in range(i + 1, n):

41. x[i] = x[i] - a[i][j] \* x[j]

42. x[i] = x[i] / a[i][i]

43.

44. # Displaying the Solutions

45. print("\nSolutions:")

46. for i in range(n):

47. print(f"x[{i+1}] = {x[i]:.3f}")

48.

49.

50. # Here we use the Gauss Elemination Function

51. if \_\_name\_\_ == "\_\_main\_\_":

52. gauss\_elimination()

53.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-4**

**Problem Statement:** Write a program to solve a system of equations using Gauss Jordan method.

**Code**:

1. def gauss\_jordan():

2. n = int(input("Enter the number of unknowns: "))

3. a = []

4.

5. # Reading the Augmented Matrix

6. print("Enter the Augmented Matrix row-wise:")

7. for i in range(n):

8. b = []

9. print(f"Enter the Row {i+1}:")

10. for j in range(n + 1):

11. b.append(int(input()))

12. a.append(b)

13.

14. print("\n")

15.

16. # Printing the matrix

17. print("The matrix is: ")

18. for i in range(n):

19. for j in range(n + 1):

20. print(a[i][j], end=" ")

21. print()

22.

23. # Applying Gauss Jordan

24. for i in range(n):

25. if a[i][i] == 0.0:

26. print("Mathematical error!")

27. return

28.

29. for j in range(n):

30. if i != j:

31. ratio = a[j][i] / a[i][i]

32.

33. for k in range(n + 1):

34. a[j][k] = a[j][k] - ratio \* a[i][k]

35.

36. # Printing the matrix

37. print("The matrix after applying Gauss Jordan elimination is: ")

38. for i in range(n):

39. for j in range(n + 1):

40. print(a[i][j], end=" ")

41. print()

42.

43. # Printing the solution

44. print("The solution is: ")

45. for i in range(n):

46. print(f"x{i+1} = {a[i][n]/a[i][i]}")

47.

48.

49. # Here we use the Gauss Jordan Function

50. if \_\_name\_\_ == "\_\_main\_\_":

51. gauss\_jordan()

52.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-5**

**Problem Statement:** Write a program to find the inverse of a matrix using Identity Matrix.

**Code**:

1. # Get the elements of the matrix from user

2. def get\_matrix\_from\_user(size):

3. print(f"Enter the elements of the {size}x{size} matrix row-wise:")

4. matrix = []

5. for i in range(size):

6. row = list(map(float, input(f"Row {i + 1}: ").strip().split()))

7. if len(row) != size:

8. raise ValueError("Number of columns must match the matrix size.")

9. matrix.append(row)

10. return matrix

11.

12.

13. # Print the matrix

14. def print\_matrix(matrix):

15. for row in matrix:

16. print(" ".join(f"{elem:.2f}" for elem in row))

17.

18.

19. # Get the identity matrix for the given matrix

20. def get\_identity\_matrix(size):

21. return [[1 if i == j else 0 for j in range(size)] for i in range(size)]

22.

23.

24. # Calculate the inverse of the given matrix

25. def calculate\_inverse(matrix):

26. size = len(matrix)

27. augmented\_matrix = [

28. row[:] + identity\_row[:]

29. for row, identity\_row in zip(matrix, get\_identity\_matrix(size))

30. ]

31.

32. for i in range(size):

33. # Make the diagonal contain all 1s

34. diag = augmented\_matrix[i][i]

35. if diag == 0:

36. return None # Matrix is singular

37.

38. for j in range(2 \* size):

39. augmented\_matrix[i][j] /= diag

40.

41. # Make the other rows contain 0s in the current column

42. for k in range(size):

43. if k != i:

44. factor = augmented\_matrix[k][i]

45. for j in range(2 \* size):

46. augmented\_matrix[k][j] -= factor \* augmented\_matrix[i][j]

47.

48. # Extract the inverse matrix

49. inverse\_matrix = [row[size:] for row in augmented\_matrix]

50. return inverse\_matrix

51.

52.

53. # Main function to perform matrix inversion

54. def main():

55. # Check user input is positive for size of matrix

56. try:

57. size = int(input("Enter the size of the square matrix (e.g., 2 for 2x2): "))

58. if size <= 0:

59. raise ValueError("Size must be a positive integer.")

60.

61. matrix = get\_matrix\_from\_user(size)

62.

63. print("\nInput Matrix:")

64. print\_matrix(matrix)

65.

66. inverse = calculate\_inverse(matrix)

67.

68. if inverse:

69. print("\nInverse Matrix:")

70. print\_matrix(inverse)

71. else:

72. print("The matrix is singular and does not have an inverse.")

73.

74. except ValueError as e:

75. print(f"Error: {e}")

76.

77.

78. if \_\_name\_\_ == "\_\_main\_\_":

79. main()

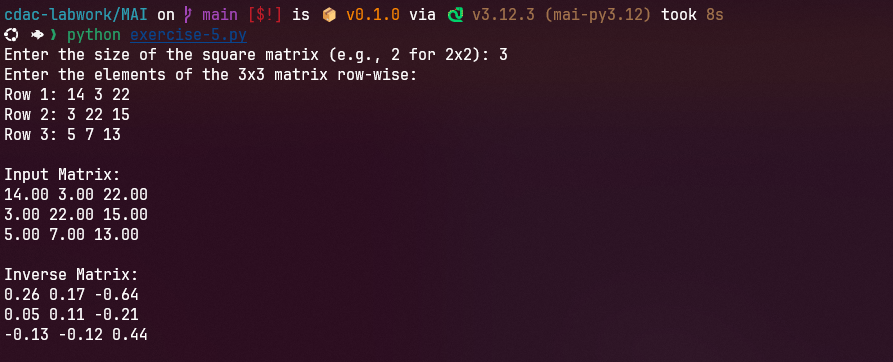
80.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

****

**Sample Output-3:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-6**

**Problem Statement:** Write a program to find the Inverse of a matrix using Adjoint Matrix.

**Code:**

1. # Function to calculate the determinant of a matrix

2. def determinant(matrix):

3. if len(matrix) == 2: # Base case for 2x2 matrix

4. return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

5.

6. det = 0

7. for c in range(len(matrix)):

8. det += ((-1) \*\* c) \* matrix[0][c] \* determinant(minor(matrix, 0, c))

9. return det

10.

11.

12. # Function to calculate the minor of a matrix (removes i-th row and j-th column)

13. def minor(matrix, i, j):

14. return [row[:j] + row[j + 1 :] for row in (matrix[:i] + matrix[i + 1 :])]

15.

16.

17. # Function to calculate the cofactor matrix

18. def cofactor\_matrix(matrix):

19. cofactors = []

20. for r in range(len(matrix)):

21. cofactor\_row = []

22. for c in range(len(matrix)):

23. minor\_det = determinant(minor(matrix, r, c))

24. cofactor\_row.append(((-1) \*\* (r + c)) \* minor\_det)

25. cofactors.append(cofactor\_row)

26. return cofactors

27.

28.

29. # Function to transpose a matrix (to get the adjugate matrix)

30. def transpose(matrix):

31. return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix))]

32.

33.

34. # Function to find the inverse of a matrix using the cofactor adjoint method

35. def inverse(matrix):

36. det = determinant(matrix)

37. if det == 0:

38. raise ValueError("The matrix is singular and cannot have an inverse.")

39.

40. # Find the cofactor matrix

41. cofactors = cofactor\_matrix(matrix)

42.

43. # Find the adjugate matrix (transpose of the cofactor matrix)

44. adjugate = transpose(cofactors)

45.

46. # Divide adjugate matrix by the determinant to get the inverse matrix

47. inverse\_matrix = [

48. [adjugate[r][c] / det for c in range(len(adjugate))]

49. for r in range(len(adjugate))

50. ]

51.

52. return inverse\_matrix

53.

54.

55. # Function to take matrix input from the user

56. def input\_matrix(name):

57. n = int(input(f"Enter the size of the {name} matrix (n x n): "))

58. matrix = []

59. print(f"Enter the elements of {name} matrix row-wise (separated by spaces):")

60. for i in range(n):

61. row = list(map(float, input(f"Row {i + 1}: ").split()))

62. matrix.append(row)

63. return matrix

64.

65.

66. # Function to display a matrix

67. def display\_matrix(matrix):

68. for row in matrix:

69. print(" ".join(map(lambda x: f"{x:.6f}", row)))

70.

71.

72. # Main program

73. if \_\_name\_\_ == "\_\_main\_\_":

74. matrix = input\_matrix("input")

75.

76. try:

77. inv\_matrix = inverse(matrix)

78. print("\nInverse of the given matrix:")

79. display\_matrix(inv\_matrix)

80. except ValueError as e:

81. print(f"Error: {e}")

82.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-3:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-7**

**Problem Statement:** Write a program to determine the rank of a matrix.

**Code:**

1. # Function to determine rank of the given matrix

2. def matrix\_rank(matrix):

3. def swap\_rows(m, row1, row2):

4. m[row1], m[row2] = m[row2], m[row1]

5.

6. def scale\_row(m, row, scalar):

7. m[row] = [element \* scalar for element in m[row]]

8.

9. def add\_scaled\_row(m, target\_row, source\_row, scalar):

10. m[target\_row] = [

11. target\_element + scalar \* source\_element

12. for target\_element, source\_element in zip(m[target\_row], m[source\_row])

13. ]

14.

15. # Convert all rows to floats for consistent arithmetic

16. for row in matrix:

17. row[:] = [float(x) for x in row]

18.

19. rows = len(matrix)

20. cols = len(matrix[0]) if rows > 0 else 0

21. rank = 0

22.

23. for col in range(cols):

24. # Find a pivot row

25. pivot\_row = None

26. for row in range(rank, rows):

27. if matrix[row][col] != 0:

28. pivot\_row = row

29. break

30.

31. if pivot\_row is None:

32. continue # No pivot in this column, move to the next column

33.

34. # Swap the pivot row to the current rank position

35. swap\_rows(matrix, rank, pivot\_row)

36.

37. # Normalize the pivot row

38. scale\_row(matrix, rank, 1.0 / matrix[rank][col])

39.

40. # Eliminate the current column in all other rows

41. for row in range(rows):

42. if row != rank:

43. add\_scaled\_row(matrix, row, rank, -matrix[row][col])

44.

45. rank += 1

46.

47. return rank

48.

49. # Function to take matrix input from user

50. def get\_matrix\_input():

51. rows = int(input("Enter the number of rows in the matrix: "))

52. cols = int(input("Enter the number of columns in the matrix: "))

53. matrix = []

54. print("Enter the matrix row by row (space-separated values):")

55. for i in range(rows):

56. row = list(map(float, input(f"Row {i + 1}: ").split()))

57. while len(row) != cols:

58. print(f"Row {i + 1} must have exactly {cols} values. Try again.")

59. row = list(map(float, input(f"Row {i + 1}: ").split()))

60. matrix.append(row)

61. return matrix

62.

63.

64. # Main program

65. print("Matrix Rank Calculator")

66. user\_matrix = get\_matrix\_input()

67. print("Rank of the matrix:", matrix\_rank(user\_matrix))

68.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**

* **EXERCISE-8**

**Problem Statement:** Write a program to solve a system of equations using Cramer’s rule.

**Code:**

1. # Function to find the determinant of given matrix

2. def determinant(matrix):

3. """Calculate the determinant of a square matrix."""

4. n = len(matrix)

5. if n == 1:

6. return matrix[0][0]

7. if n == 2:

8. return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

9.

10. det = 0

11. for col in range(n):

12. sub\_matrix = [row[:col] + row[col + 1 :] for row in matrix[1:]]

13. det += ((-1) \*\* col) \* matrix[0][col] \* determinant(sub\_matrix)

14. return det

15.

16.

17. # Function to apply Cramer's Rule

18. def cramer\_rule(coeff\_matrix, constants):

19. """Solve a system of linear equations using Cramer's Rule."""

20. n = len(coeff\_matrix)

21. det\_main = determinant(coeff\_matrix)

22. if det\_main == 0:

23. raise ValueError("The system has no unique solution (determinant is 0).")

24.

25. solutions = []

26. for i in range(n):

27. temp\_matrix = [

28. row[:i] + [constants[row\_idx]] + row[i + 1 :]

29. for row\_idx, row in enumerate(coeff\_matrix)

30. ]

31. det\_temp = determinant(temp\_matrix)

32. solutions.append(det\_temp / det\_main)

33. return solutions

34.

35.

36. # Function to get matrix from user

37. def get\_matrix\_input():

38. """Get coefficient matrix and constants from the user."""

39. n = int(input("Enter the number of variables (n): "))

40. print("Enter the coefficient matrix row by row (space-separated values):")

41. coeff\_matrix = []

42. for i in range(n):

43. row = list(map(float, input(f"Row {i + 1}: ").split()))

44. while len(row) != n:

45. print(f"Row {i + 1} must have exactly {n} values. Try again.")

46. row = list(map(float, input(f"Row {i + 1}: ").split()))

47. coeff\_matrix.append(row)

48.

49. print("Enter the constants vector (space-separated values):")

50. constants = list(map(float, input().split()))

51. while len(constants) != n:

52. print(f"The constants vector must have exactly {n} values. Try again.")

53. constants = list(map(float, input().split()))

54.

55. return coeff\_matrix, constants

56.

57.

58. # Main Program

59. print("Solve a System of Linear Equations using Cramer's Rule")

60. try:

61. coeff\_matrix, constants = get\_matrix\_input()

62. solutions = cramer\_rule(coeff\_matrix, constants)

63. print("The solution is:")

64. for i, sol in enumerate(solutions):

65. print(f"x{i + 1} = {sol}")

66. except ValueError as e:

67. print("Error:", e)

68.

**Sample Output-1:**

**A screenshot of a computer

Description automatically generated**

**Sample Output-2:**

**A screenshot of a computer

Description automatically generated**