Batch: A1 **Roll No.: 16010123012**

Experiment No. 2

Title: Linear Algebra - Solving System of Linear Equations using R

Aim: To explore methods for solving systems of linear equations using R, including visualization, and understanding their application in data science.

Course Outcome: CO**2**

Books/ Journals/ Websites referred:

1. The Comprehensive R Archive Network
2. Posit

Resources used:
https://www.rdocumentation.org/
https://www.w3schools.com/r/
https://www.geeksforgeeks.org/r-programming-language-introduction/

Theory:
Linear algebra is fundamental in data science for operations like feature transformations, dimensionality reduction (e.g., PCA), solving optimization problems (e.g., regression), and working with graph structures.
Procedure and Implementation in R:
A system of linear equations can be represented in matrix form as:
AX = B
Where:

- **A** is the coefficient matrix.
- **X** is the column vector of variables.
- **B** is the column vector of constants.

The solution to this system involves finding **X** such that the equation holds true.
In R, we can solve such systems using the following methods:

1. **Solving Using Gauss-Jordan Elimination:** Perform row operations manually in R to transform A into its reduced row-echelon form.

2. **Direct Inversion:** Compute $X = A^{-1}B$, where A^{-1} is the inverse of matrix A.
3. **Built-in Functions:** R provides the solve() function to solve **AX = B** directly.

Part 1: A system of two linear equations

1. Define the System of Linear Equations:

Solve:

1. $2x + y = 5$
2. $x - y = -1$

2. Represent in Matrix Form:

Define A as the coefficient matrix and B as the constant matrix:

$$A = [2\ 1\ 1\ \ -1\ ]$$

$$B = [5\ \ -1\ ]$$

```
> A <- matrix(c(2, 1, 1, -1), nrow = 2, byrow = TRUE)
> B <- c(5, -1)

> print(A)
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> print(B)
[1]  5 -1
```

Create augmented matrix:

$$[2\ 1\ 5\ 1\ \ -1\ \ -1\ ]$$

```
> augmented_matrix <- cbind(A, B)

> print(augmented_matrix)
           B
[1,] 2  1  5
[2,] 1 -1 -1
```

3. Check whether there is a unique solution

```
> determinant_A <- det(A)
> if (determinant_A == 0) {
+     cat("The system does not have a unique solution.\n")
+ } else {
+     cat("The system has a unique solution.\n")
+ }
The system has a unique solution.
```

4. Solve using Gauss Jordan elimination

```
> # Row operations for Gauss-Jordan elimination
  augmented_matrix[1, ] <- augmented_matrix[1, ] / augmented_matrix[1, 1]  # Make pivot 1
  augmented_matrix[2, ] <- augmented_matrix[2, ] - augmented_matrix[2, 1] * augmented_matrix[1, ]
  augmented_matrix[2, ] <- augmented_matrix[2, ] / augmented_matrix[2, 2]  # Make second pivot 1
  augmented_matrix[1, ] <- augmented_matrix[1, ] - augmented_matrix[1, 2] * augmented_matrix[2, ]
  # Solution
  solution_gauss <- augmented_matrix[, 3]
  print(solution_gauss)

[1] 1.333333 2.333333
```

5. Solve using inbuilt solve() method

```
> solution_solve <- solve(A, B)
> print(solution_solve)
[1] 1.333333 2.333333
```

6. Inversion $X = A^{-1}B$

```
> A_inverse <- solve(A)
> solution_alt <- A_inverse %*% B
> print(solution_alt)
         [,1]
[1,] 1.333333
[2,] 2.333333
```

7. Visualization
    a. Define the system of equations

    2x + y = 5; y = 5 - 2x

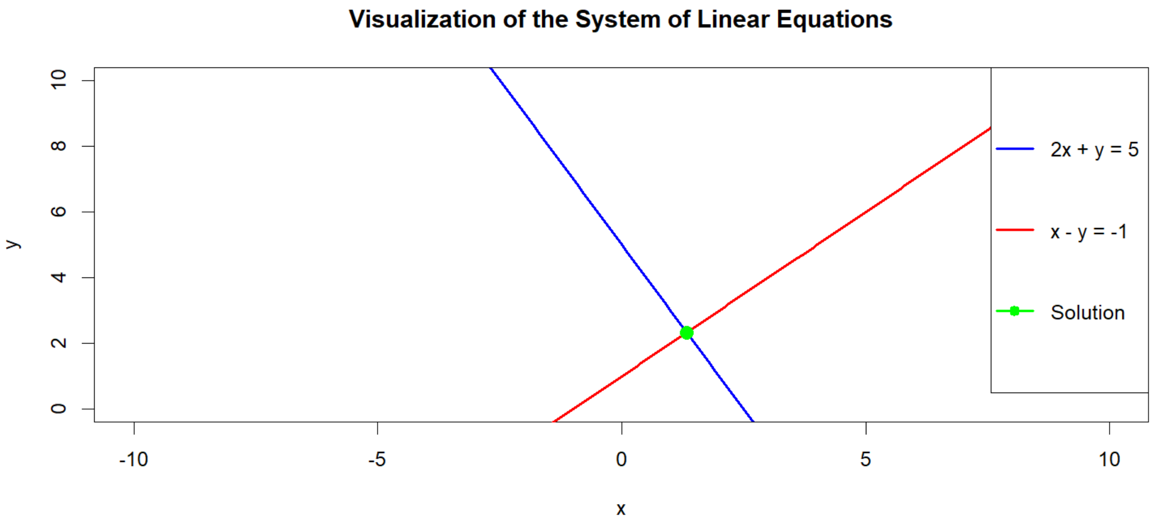
    x - y = -1; y = x + 1

    b. Generate x values

```
> # Define the functions for y
> f1 <- function(x) { 5 - 2 * x }
> f2 <- function(x) { x + 1 }
> # Generate x values for plotting
> x_vals <- seq(-10, 10, by = 0.1)
```

    c. Plot the equations and solution

```
> # Plot the equations
  plot(x_vals, f1(x_vals), type = "l", col = "blue", lwd = 2, ylim = c(0, 10),
       xlab = "x", ylab = "y", main = "Visualization of the System of Linear Equations")
  lines(x_vals, f2(x_vals), col = "red", lwd = 2)

  # Add the solution point
  points(solution_solve[1], solution_solve[2], col = "green", pch = 19, cex = 1.5)

  # Add legend
  legend("topright", legend = c("2x + y = 5", "x - y = -1", "Solution"),
         col = c("blue", "red", "green"), lwd = 2, pch = c(NA, NA, 19))
```



Part 2: A system of three linear equations

1. Define the System of Linear Equations:

Solve:

$$
\begin{aligned}
x + y + z &= 6 \\
2x - y + z &= 3 \\
x - 2y + 3z &= 14
\end{aligned}
$$

2. Represent in Matrix Form:

Define A as the coefficient matrix and B as the constant matrix:

$$A = [1\ 1\ 1\ 2\ -1\ 1\ 1\ -2\ 3] \quad B = [6\ 3\ 14]$$

```
> A <- matrix(c(1, 1, 1, 2, -1, 1, 1, -2, 3), nrow = 3, byrow = TRUE)
> B <- c(6, 3, 14)
```

```
> print(A)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2   -1    1
[3,]    1   -2    3
> print(B)
[1]  6  3 14
```

Create augmented matrix:

$$[1\ 1\ 1\ 6\ 2\ -1\ 1\ 3\ 1\ -2\ 3\ 14\ ]$$

```
> augmented_matrix <- cbind(A, B)
> print(augmented_matrix)
           B
[1,] 1  1 1  6
[2,] 2 -1 1  3
[3,] 1 -2 3 14
```

3. Check whether there is a unique solution

```
> determinant_A <- det(A)
> if (determinant_A == 0) {
+     cat("The system does not have a unique solution.\n")
+ } else {
+     cat("The system has a unique solution.\n")
+ }
The system has a unique solution.
```

4. Solve using Gauss Jordan elimination

```
> # Function for Gauss-Jordan Elimination
  gauss_jordan_3d <- function(A, B) {
      # Combine the coefficient matrix A and the constant matrix B to form the augmented matrix
      augmented_matrix <- cbind(A, B)

      # Number of rows
      n <- nrow(A)

      # Apply Gauss-Jordan elimination
      for (i in 1:n) {
          # Make the pivot element 1 by dividing the row by the pivot value
          augmented_matrix[i, ] <- augmented_matrix[i, ] / augmented_matrix[i, i]

          # Eliminate the variable from all rows except the pivot row
          for (j in 1:n) {
              if (j != i) {
                  augmented_matrix[j, ] <- augmented_matrix[j, ] - augmented_matrix[j, i] * augmented_matrix[i, ]
              }
          }
      }

      # Extract the solution from the last column of the augmented matrix
      solution <- augmented_matrix[, n+1]
      return(solution)
  }
```

```
> solution_gauss_3 <- gauss_jordan_3d(A, B)
> solution_gauss_3
[1] -0.7777778  1.1111111  5.6666667
```

5. Solve using inbuilt solve() method

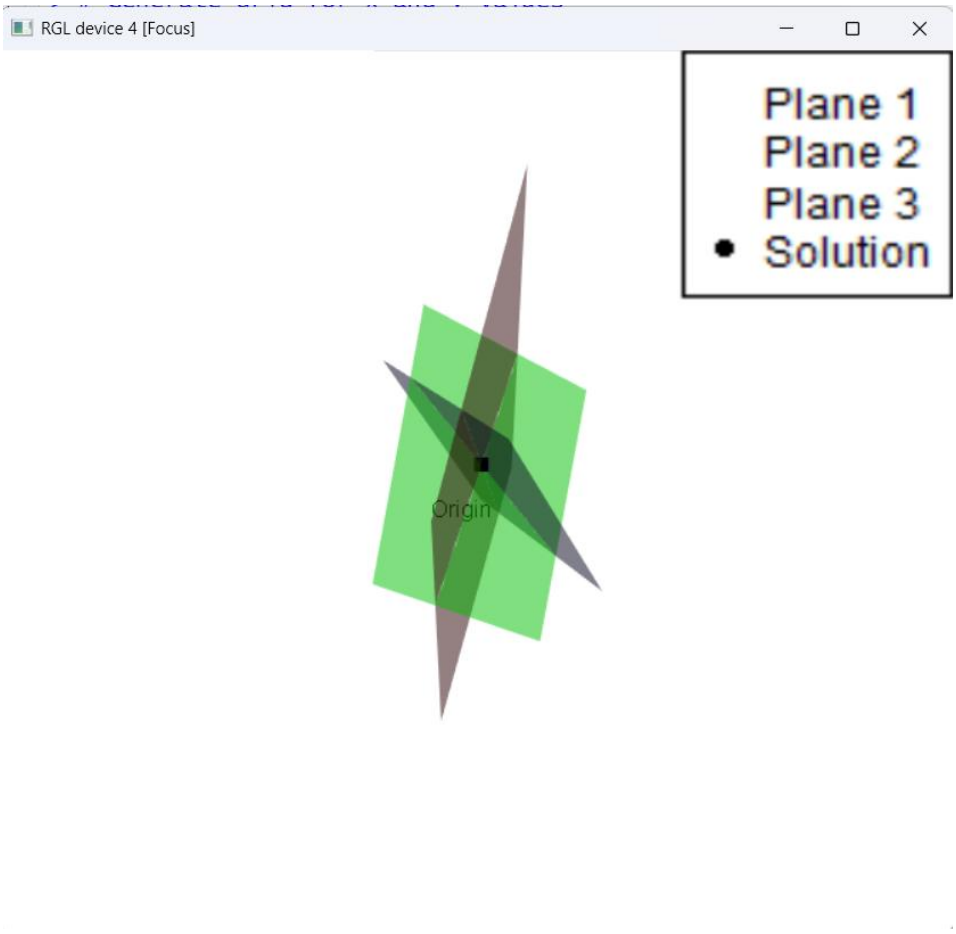
```
> solution_solve_3 = solve(A,B)
> solution_solve_3
[1] -0.7777778  1.1111111  5.6666667
```

6. Inversion

```
> A_inverse <- solve(A)
> solution_alt <- A_inverse %*% B
> print(solution_alt)
           [,1]
[1,] -0.7777778
[2,]  1.1111111
[3,]  5.6666667
```

7. Visualization

```
> library(rgl)

  # Define planes
  plane1 <- function(x, y) 6 - x - y
  plane2 <- function(x, y) 3 - 2 * x + y
  plane3 <- function(x, y) (14 - x + 2 * y) / 3

  # Generate grid for x and y values
  x_vals <- seq(-10, 10, length.out = 30)
  y_vals <- seq(-10, 10, length.out = 30)

  # Compute z values for the planes
  z1 <- outer(x_vals, y_vals, plane1)
  z2 <- outer(x_vals, y_vals, plane2)
  z3 <- outer(x_vals, y_vals, plane3)

  # Open a 3D plot
  open3d()

  # Plot planes
  surface3d(x_vals, y_vals, z1, color = "blue", alpha = 0.5)
  surface3d(x_vals, y_vals, z2, color = "red", alpha = 0.5)
  surface3d(x_vals, y_vals, z3, color = "green", alpha = 0.5)

  # Add the intersection point (solution)
  solution <- solve(A, B) # Calculate solution using R
  points3d(solution[1], solution[2], solution[3], col = "black", size = 10)

  # Labels and legend
  rgl.texts(x = 0, y = 0, z = 0, text = "Origin", col = "black")
  legend3d("topright", legend = c("Plane 1", "Plane 2", "Plane 3", "Solution"),
           col = c("blue", "red", "green", "black"), pch = c(NA, NA, NA, 19))
```



Students have to generate a system of 2 linear equations and a system of 3 linear equations with random coefficients and then perform the above steps on them.

```
> # Generate coefficients and constants for the first equation
  a1 <- sample(-10:10, 1)
  b1 <- sample(-10:10, 1)
  c1 <- sample(-10:10, 1)

  # Generate coefficients and constants for the second equation
  a2 <- sample(-10:10, 1)
  b2 <- sample(-10:10, 1)
  c2 <- sample(-10:10, 1)

  # Form the coefficient matrix and constant vector
  A <- matrix(c(a1, b1, a2, b2), nrow = 2, byrow = TRUE)
  B <- c(c1, c2)
```

```
> # Generate coefficients and constants for the first equation
  a1 <- sample(-10:10, 1)
  b1 <- sample(-10:10, 1)
  c1 <- sample(-10:10, 1)
  d1 <- sample(-10:10, 1)

  # Generate coefficients and constants for the second equation
  a2 <- sample(-10:10, 1)
  b2 <- sample(-10:10, 1)
  c2 <- sample(-10:10, 1)
  d2 <- sample(-10:10, 1)

  # Generate coefficients and constants for the third equation
  a3 <- sample(-10:10, 1)
  b3 <- sample(-10:10, 1)
  c3 <- sample(-10:10, 1)
  d3 <- sample(-10:10, 1)

  # Form the coefficient matrix and constant vector
  A <- matrix(c(a1, b1, c1, a2, b2, c2, a3, b3, c3), nrow = 3, byrow = TRUE)
  B <- c(d1, d2, d3)
```

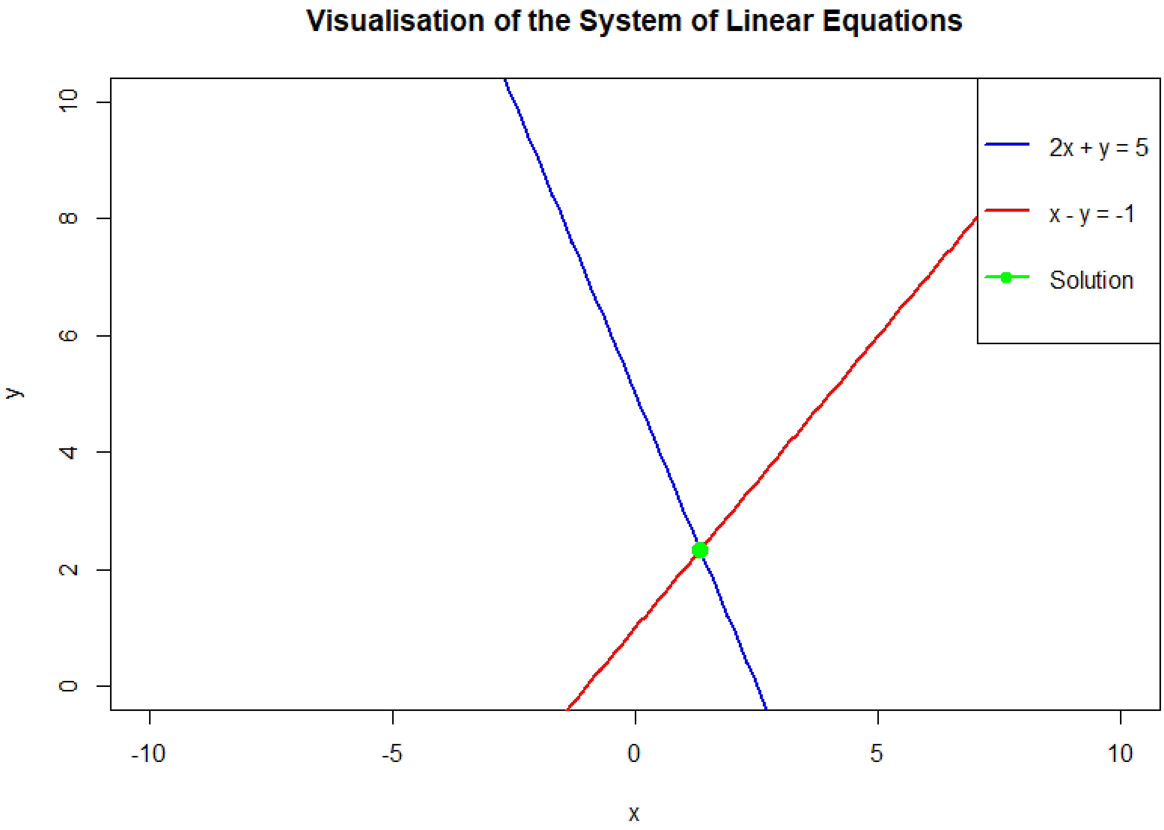
TWO LINEAR EQUATION:

```
> A <- matrix(c(2, 1, 1, -1), nrow =  2, byrow = TRUE)
> B <- c(5, -1)
> print(A)
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> print(B)
[1]  5 -1
> augmented_matrix <- cbind(A, B)
> print(augmented_matrix)
           B
[1,] 2  1  5
[2,] 1 -1 -1
> determinant_A <- det(A)
> if(determinant_A == 0){
+     cat("No unique solution\n")
+ }else{
+     cat("Unique solution")
+ }
Unique solution
> augmented_matrix[1, ] <- augmented_matrix[1, ] / augmented_matrix[1, 1]
> augmented_matrix[2, ] <- augmented_matrix[2, ] - augmented_matrix[2, 1] * augmented_matrix[1, ]
> augmented_matrix[2, ] <- augmented_matrix[2, ] / augmented_matrix[2, 2]
> augmented_matrix[1, ] <- augmented_matrix[1, ] - augmented_matrix[1, 2] * augmented_matrix[2, ]
> solution_gauss <- augmented_matrix[, 3]
> print(solution_gauss)
[1] 1.333333 2.333333
> sol_solve <- solve(A, B)
> print(sol_solve)
[1] 1.333333 2.333333
> A_inverse <- solve(A)
> solution_alt <- A_inverse %*% B
> print(solution_alt)
         [,1]
[1,] 1.333333
[2,] 2.333333
> 
```

R ▾ Global Environment ▾

Data	
A	num [1:2, 1:2] 2 1 1 -1
A_inverse	num [1:2, 1:2] 0.333 0.333 0.333 -0.667
augmented_matrix	num [1:2, 1:3] 1 0 0 1 1.33 ...
solution_alt	num [1:2, 1] 1.33 2.33
Values	
B	num [1:2] 5 -1
determinant_A	-3
sol_solve	num [1:2] 1.33 2.33
solution_gauss	num [1:2] 1.33 2.33
x_vals	num [1:201] -10 -9.9 -9.8 -9.7 -9.6 -9.5 -9.4 -9.3 -9.2 -9.1 ...
Functions	
f1	function (x)
f2	function (x)

```
> A <- matrix(c(1, 1, 1, 2, -1, 1, 1, -2, 3), nrow =  3, byrow = TRUE)
> B <- c(6, 3, 14)
>
> print(A)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2   -1    1
[3,]    1   -2    3
> print(B)
[1]  6  3 14
>
> augmented_matrix <- cbind(A, B)
> print(augmented_matrix)
             B
[1,] 1  1 1  6
[2,] 2 -1 1  3
[3,] 1 -2 3 14
>
> determinant_A <- det(A)
> if(determinant_A == 0){
+     cat("No unique solution\n")
+ }else{
+     cat("Unique solution")
+ }
Unique solution>
> gauss_jordan <- function(A, B){
+     augmented_matrix <- cbind(A, B)
+     n <- nrow(A)
+     for(i in 1:n){
+         augmented_matrix[i, ] <- augmented_matrix[i, ] / augmented_matrix[i, i]
+         for(j in 1:n){
+             if(j != i){
+                 augmented_matrix[j, ] <- augmented_matrix[j, ] - augmented_matrix[j, i] * augmented_matrix[i, ]
+             }
+         }
+     }
+     solution <- augmented_matrix[, n + 1]
+     return(solution)
+ }
>
> solution_gauss <- gauss_jordan(A, B)
> print(solution_gauss)
[1] -0.7777778  1.1111111  5.6666667
>
> sol_solve <- solve(A, B)
> print(sol_solve)
[1] -0.7777778  1.1111111  5.6666667
>
> A_inverse <- solve(A)
> solution_alt <- A_inverse %*% B
> print(solution_alt)
           [,1]
[1,] -0.7777778
[2,]  1.1111111
```


Visualisation of the System of Linear Equations
2x + y = 5
x - y = -1
Solution
y
x
-10
-5
0
5
10
0
2
4
6
8
10

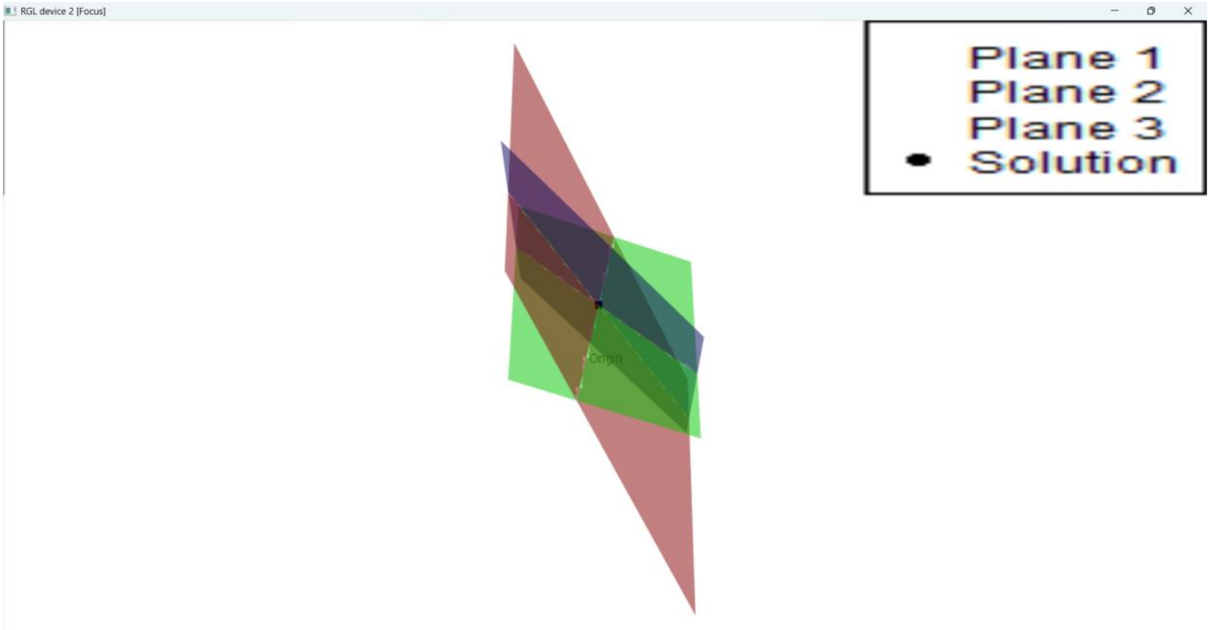
THREE LINEAR EQUATION:

```
> A <- matrix(c(1, 1, 1, 2, -1, 1, 1, -2, 3), nrow =  3, byrow = TRUE)
> B <- c(6, 3, 14)
>
> print(A)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2   -1    1
[3,]    1   -2    3
> print(B)
[1]  6  3 14
>
> augmented_matrix <- cbind(A, B)
> print(augmented_matrix)
             B
[1,] 1  1 1  6
[2,] 2 -1 1  3
[3,] 1 -2 3 14
>
> determinant_A <- det(A)
> if(determinant_A == 0){
+     cat("No unique solution\n")
+ }else{
+     cat("Unique solution")
+ }
Unique solution>
> gauss_jordan <- function(A, B){
+     augmented_matrix <- cbind(A, B)
+     n <- nrow(A)
+     for(i in 1:n){
+         augmented_matrix[i, ] <- augmented_matrix[i, ] / augmented_matrix[i, i]
+         for(j in 1:n){
+             if(j != i){
+                 augmented_matrix[j, ] <- augmented_matrix[j, ] - augmented_matrix[j, i] * augment
ed_matrix[i, ]
+             }
+         }
+     }
+     solution <- augmented_matrix[, n + 1]
+     return(solution)
```

```
+     return(solution)
+ }
>
> solution_gauss <- gauss_jordan(A, B)
> print(solution_gauss)
[1] -0.7777778  1.1111111  5.6666667
>
> sol_solve <- solve(A, B)
> print(sol_solve)
[1] -0.7777778  1.1111111  5.6666667
>
> A_inverse <- solve(A)
> solution_alt <- A_inverse %*% B
> print(solution_alt)
           [,1]
[1,] -0.7777778
[2,]  1.1111111
[3,]  5.6666667
>
> plane1 <- function(x, y) 6 - x - y
> plane2 <- function(x, y) 3 - 2 * x + y
> plane3 <- function(x, y) (14 - x + 2 * y) / 3
>
> x_vals <- seq(-10, 10, length.out = 30)
> y_vals <- seq(-10, 10, length.out = 30)
>
> z1 <- outer(x_vals, y_vals, plane1)
> z2 <- outer(x_vals, y_vals, plane2)
> z3 <- outer(x_vals, y_vals, plane3)
>
> open3d()
wgl
  2
>
> surface3d(x_vals, y_vals, z1, color = "blue", alpha = 0.5)
> surface3d(x_vals, y_vals, z2, color = "red", alpha = 0.5)
> surface3d(x_vals, y_vals, z3, color = "green", alpha = 0.5)
>
> solution <- solve(A, B)
> points3d(solution[1], solution[2], solution[3], col = "black", size = 10)
>
> texts3d(x = 0, y = 0, z = 0, text = "Origin", col = "black")
> legend3d("topright", legend = c("Plane 1", "Plane 2", "Plane 3", "Solution"),
+          col = c("blue", "red", "green", "black"), pch = c(NA, NA, NA, 19))
>
```

R ▾ Global Environment ▾

Data	
A	num [1:3, 1:3] 1 2 1 1 -1 -2 1 1 3
A_inverse	num [1:3, 1:3] 0.111 0.556 0.333 0.556 -0.222 ...
augmented_matrix	num [1:3, 1:4] 1 2 1 1 -1 -2 1 1 3 6 ...
solution_alt	num [1:3, 1] -0.778 1.111 5.667
z1	num [1:30, 1:30] 26 25.3 24.6 23.9 23.2 ...
z2	num [1:30, 1:30] 13 11.62 10.24 8.86 7.48 ...
z3	num [1:30, 1:30] 1.333 1.103 0.874 0.644 0.414 ...
Values	
B	num [1:3] 6 3 14
determinant_A	-9
sol_solve	num [1:3] -0.778 1.111 5.667
solution	num [1:3] -0.778 1.111 5.667
solution_gauss	num [1:3] -0.778 1.111 5.667
x_vals	num [1:30] -10 -9.31 -8.62 -7.93 -7.24 ...
y_vals	num [1:30] -10 -9.31 -8.62 -7.93 -7.24 ...
Functions	
gauss_jordan	function (A, B)
plane1	function (x, y)
plane2	function (x, y)
plane3	function (x, y)



Conclusion:
I have successfully completed this experiment and learnt how to solve system of linear equations using R programming language.

Post-lab questions:

1. Why might certain systems of equations have no solution, a unique solution, or infinitely many solutions?
2. Describe atleast three real-world data science problems in detail where solving systems of linear equations is crucial.
3. Investigate what happens when you attempt to invert a singular matrix using solve(). How does R handle this scenario?
4. What are eigen values and eigen vectors? Describe atleast three real-world data science problems in detail where eigen values and eigen vectors can be applied.

Aaryan Sharma
16010123012

Date
Page

IDS Exp 2

1) The number of solutions to a system of equations depends on relationship between the eqn.
No solution: No intersection, they are inconsistent.
Unique solution: Intersected at exactly one point
Infinitely many solution: Eqns are dependent, they overlap.

2) (i) Linear Regression: One predicts a dependent variable based on multiple independent variable. The parameters are estimated by linear eqns.
(ii) Principal Component Analysis (PCA): It reduces the dimensionality of large datasets by finding principal components. This involve solving a system of linear eqn related to covariance matrix to identify directions of max variance.
(iii) Hidden Markov Models (HMM): HMMs are used in speech recognition. Model involves hidden states. Solving lin eqns gives the most probable sequence of hidden states.

3) It will fail because a singular matrix does not have an inverse. R handles this by throwing an error indicating matrix is not invertible.

4) Eigenvalues and eigenvectors are properties of square matrices. They help un~~der~~cover key pattern in data.
Real World Applications:
(i) Principal Component Analysis: Used for dimensionality reduction, eigenvectors represent principal component and eigenvalue their importance
(ii) Spectral Clustering: Eigenvectors of a ~~similarity~~ graph laplacian matrix are used to group data points.
(iii) Recommendation Systems: Eigenvalues and Eigenvectors are used in matrix factorisation to identify latent features.