Batch: E2 **Roll No.:** 16010123325

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:

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

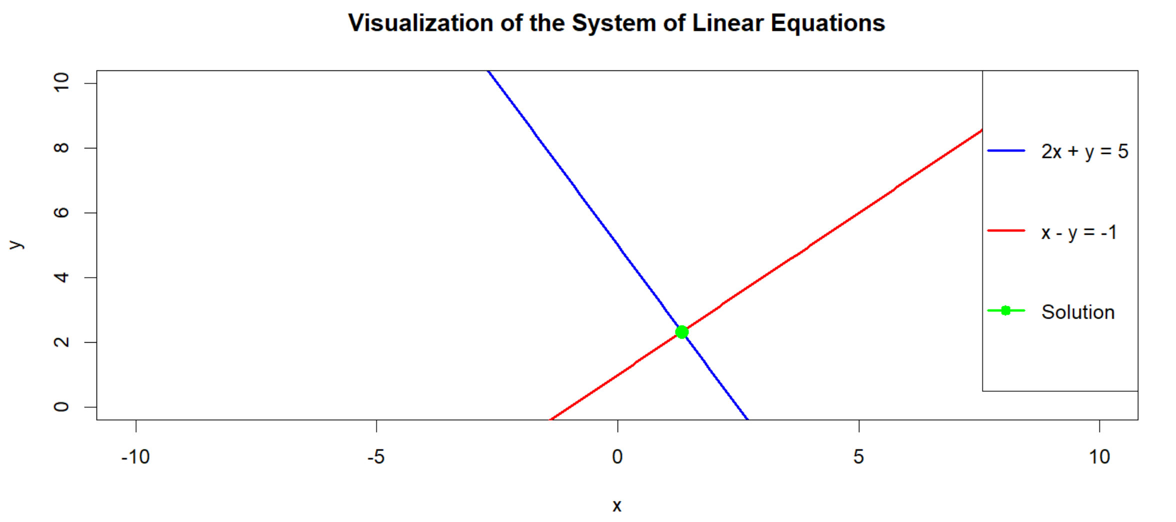
x - y = -1; y = x + 1

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:

$$x + y + z = 6$$
$$2x - y + z = 3$$
$$x - 2y + 3z = 14$$

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

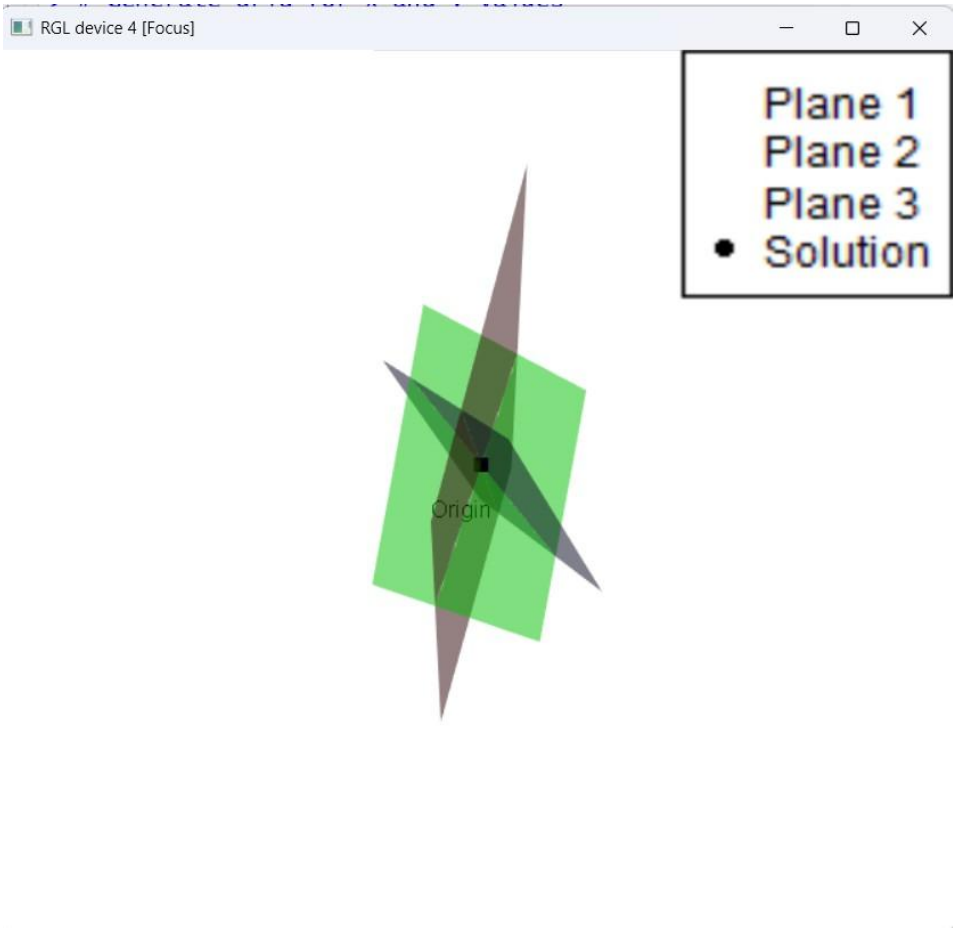
```
> solution_solve_3 = solve(A,B)
> solution_solve_3
[1] -0.7777778  1.1111111  5.6666667
```

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)
```

Performance in Lab:

Part 1: A system of two linear equations

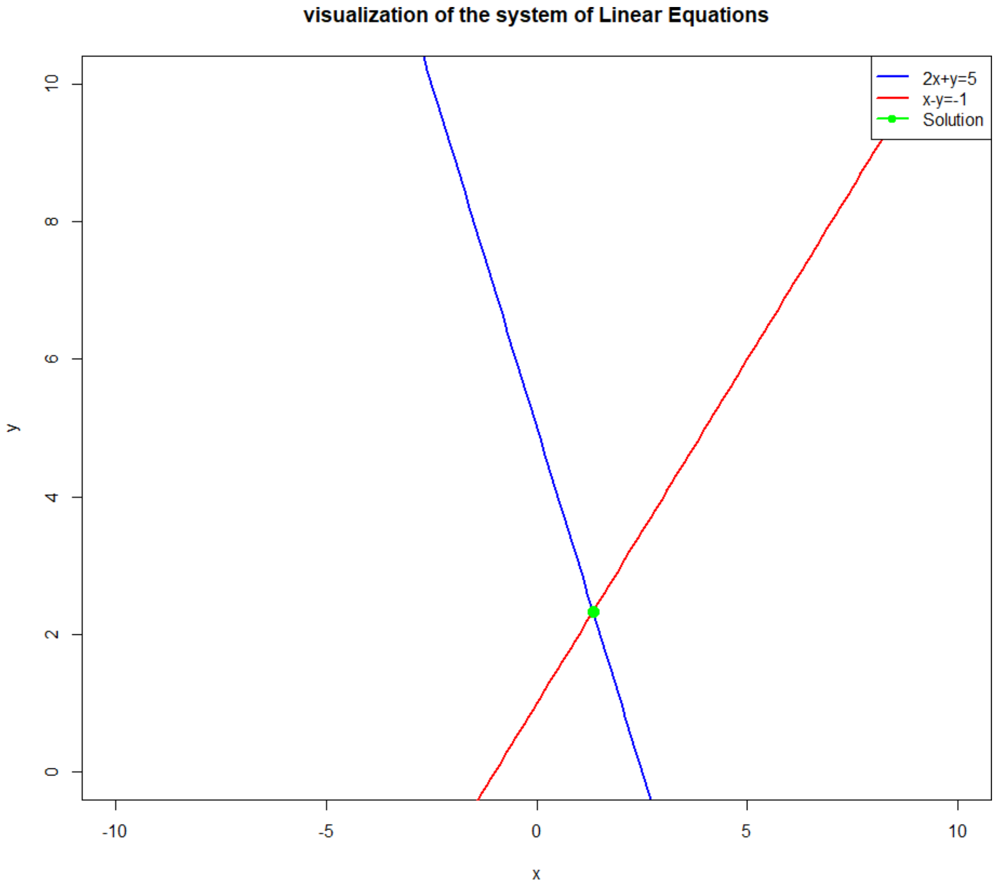
```
> A<-matrix(c(2,1,1,-1), nrow=2,byrow=TRUE)
> B<-c(5,-1)
> print(A)
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> print(B)
[1]  5 -1
> augmented_matrix<-cbind(A,B)
> print(augmented_matrix)
           B
[1,] 2  1  5
[2,] 1 -1 -1
>
```

```
           B
[1,] 2  1  5
[2,] 1 -1 -1
> det_a<-det(A)
> print(det_a)
[1] -3
> if(det_a==0)
+ {}
> if(det_a!=0)
+ {cat("System has unique solution.\n")}
System has unique solution.
> 
```

```
> augmented_matrix<-cbind(A,B)
> augmented_matrix<-augmented_matrix[1,]/augmented_matrix[1,1]
> augmented_matrix[1,]<-augmented_matrix[1,]/augmented_matrix[1,1]
Error in augmented_matrix[1, ] : incorrect number of dimensions
> rm(augmented_matrix)
> augmented_matrix<-cbind(A,B)
> 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,]
> augmented_matrix
                B
[1,] 1 0 1.333333
[2,] 0 1 2.333333
> solution<-solve(A,B)
> print(solution)
[1] 1.333333 2.333333
```

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

```
> 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[1],solution[2],col="green",pch=19,cex=1.5)
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  display list redraw incomplete
2: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
3: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
4: In doTryCatch(return(expr), name, parentenv, handler) :
  display list redraw incomplete
5: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
6: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
> print(solution)
[1] 1.333333 2.333333
> 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

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

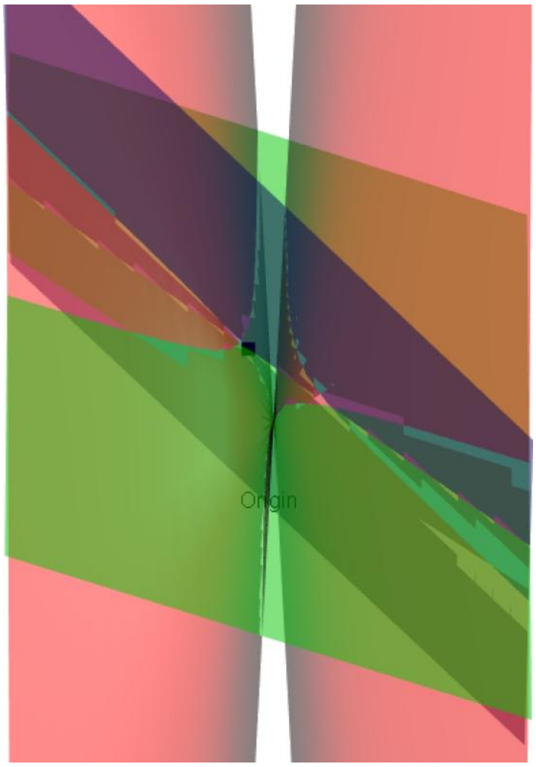
```
> detA<-det(A)
> if(detA==0){
+     cat("The system does not have a unique solution.\n")
+ } else{}
NULL
> if(detA==0){
+ cat("The system does not have a unique solution.\n")
+ }else{
+ cat("The system has a unique solution.\n")
+ }
The system has a unique solution.
```

```
> #Function for Gauss-Jordan Elmination
> gauss_jordan_3d<-function(A,B){
+ augmatrix<-cbind(A,B)
+ n<-nrow(A)
+ #Applying Gauss Jordan Elimination
+ for(i in 1:n){
+ augmatrix[i, ]<-augmatrix[i, ]/augmatrix[i,i]
+ for(j in 1:n){
+ if(j!=i){
+ augmatrix[j, ]<-augmatrix[j, ]-augmatrix[j,i]*augmatrix[i, ]
+ }
+ }
+ }
+ solution<-augmatrix[,n+1]
+ return(solution)
+ }
> solution_gauss_3<-gauss_jordan_3d(A,B)
> solution_gauss_3
[1] -0.7777778  1.1111111  5.6666667
```

```
> solution_solve_3=solve(A,B)
> solution_solve_3
[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
>
```

```
> 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
> #Generating 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()
wgl
  1
> #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)
>
> #Adding the intersection point
> solution<-solve(A,B)
> 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")
Warning message:
In rgl.texts(x = 0, y = 0, z = 0, text = "Origin", col = "black") :
  'rgl.texts' is deprecated.
Use 'text3d' instead.
See help("Deprecated")
> legend3d("topright",legend=c("Plane 1","Plane 2","Plane 3","Solution")
+         col=c("blue","red","green","black"),pch=c(NA,NA,NA,19))
```



Conclusion:

Solving matrix equations in R and visualizing the solutions through plotting provides a clear understanding of linear relationships and solution spaces.

Post-lab questions:

1. Why might certain systems of equations have no solution, a unique solution, or infinitely many solutions?
Ans:

No Solution Scenarios:

- Inconsistent equations
- Parallel lines with different y-intercepts
- Contradictory constraints
- Mathematically represented by 0 = non-zero constant
- Example: $2x + 3y = 10$ and $2x + 3y = 15$

Unique Solution Scenarios:

- Lines intersect at exactly one point
- Independent equations with distinct constraints
- Mathematically represented by unique (x,y) coordinates
- Precise balance of coefficients
- Example: x + y = 5 and 2x - y = 3

Infinitely Many Solutions:

- Completely overlapping lines
- Identical equation representations
- Mathematically represented by 0 = 0
- Equivalent equations with same slope/intercept
- Example: 2x + y = 6 and 4x + 2y = 12

Determining Factors:

- Coefficient relationships
- Rank of augmented matrix
- Linear independence
- Geometric interpretation of equations
- Computational method of solving (elimination, substitution)

Mathematical Conditions:

- Consistent systems: At least one solution
- Dependent equations: Infinite solutions
- Independent equations: Unique solution

Visualization:

- Graphical representation shows intersection points
- Linear algebra techniques reveal solution characteristics

2. Describe at least three real-world data science problems in detail where solving systems of linear equations is crucial.
Ans:
Three Detailed Data Science Problems Using Linear Equation Systems:

1. Economic Forecasting Model

- Objective: Predict economic indicators
- Linear Equations Role:
    - Multiple regression analysis
    - Modeling relationships between variables
    - Estimating coefficients

- Variables:
  - GDP growth
  - Inflation rates
  - Unemployment
- Solving Technique:
  - Least squares regression
  - Matrix-based coefficient estimation
- Practical Impact:
  - Policy decision making
  - Investment strategy development

2. Supply Chain Optimization

- Objective: Resource allocation optimization
- Linear Equations Applications:
  - Constraint modeling
  - Production capacity planning
  - Minimizing transportation costs
- Key Variables:
  - Production volumes
  - Shipping routes
  - Inventory levels
- Solving Techniques:
  - Linear programming
  - Simplex method
  - Matrix transformations
- Business Outcomes:
  - Cost reduction
  - Efficiency improvement
  - Inventory management

3. Machine Learning Feature Selection

- Objective: Identifying most significant predictors
- Linear Equations Approach:
  - Regression coefficient analysis
  - Multivariate feature weighting
  - Dimensionality reduction
- Techniques:
  - Ordinary least squares
  - Ridge regression
  - Lasso regression
- Implementation:
  - Solving high-dimensional equation systems
  - Identifying feature importance
- Applications:

- Predictive modeling
- Customer behavior prediction
- Risk assessment

3. Investigate what happens when you attempt to invert a singular matrix using solve(). How does R handle this scenario?

Ans:

Matrix Inversion with Singular Matrices in R:

Singular Matrix Characteristics:

- Determinant = 0
- Non-invertible
- Linear dependencies exist
- Rank < full matrix dimension

R's solve() Function Behavior:

- Throws error when matrix is singular
- Generates warning/exception
- Prevents computational breakdown

Error Handling Mechanisms:

- Generates "system is computationally singular" message
- Stops matrix inversion process
- Prevents invalid mathematical operations

Alternative Approaches:

1. Pseudoinverse Method
- Uses MASS package
- pinv() function
- Approximates inverse
- Handles near-singular matrices
2. Regularization Techniques
- Add small constant to diagonal
- Improves matrix conditioning
- Reduces numerical instability
3. Decomposition Methods
- Singular Value Decomposition (SVD)
- Eigenvalue decomposition
- Identifies matrix structural issues

4. What are eigen values and eigen vectors? Describe at least three real-world data science problems in detail where eigen values and eigen vectors can be applied.

Ans:
Definition:
- Eigenvalues: Scalar values representing matrix's scaling factor
- Eigenvectors: Corresponding vectors unchanged in direction
- Represent fundamental linear transformation characteristics

Mathematical Representation:
- $Av = \lambda v$
- A: Original matrix
- v: Eigenvector
- λ: Corresponding eigenvalue

Three Real-World Applications:
1. Principal Component Analysis (PCA)
- Dimensionality Reduction
- Feature Extraction
- Key Steps:
    - Identify principal components
    - Capture maximum variance
    - Compress high-dimensional data
- Applications:
    - Image recognition
    - Face detection
    - Medical imaging analysis

2. Recommender Systems
- User Preference Modeling
- Latent Feature Identification
- Techniques:
    - Matrix factorization
    - Collaborative filtering
    - Eigenvalue decomposition
- Use Cases:
    - Netflix recommendation engine
    - E-commerce product suggestions
    - Personalized content platforms

3. Network Analysis
- Social Network Characterization
- Centrality Measurement
- Applications:
    - Identifying influential nodes

    - Understanding network structure
    - Analyzing communication patterns
- Computational Techniques:
    - PageRank algorithm
    - Graph spectral analysis
    - Community detection

Computational Characteristics:

- Capture system's fundamental dynamics
- Reveal underlying structural properties
- Provide dimensionality insights

Mathematical Properties:

- Invariant under linear transformations
- Represent system's core characteristics
- Enable complex data interpretation

Visualization Techniques:

- Spectral decomposition
- Geometric interpretation
- Dimensional mapping