Test Report: Test Plan for the Library of Linear Algebraic Equation Solver

Devi Prasad Reddy Guttapati

December 18, 2017

Contents

Table 1: **Revision History**

Date	**Version**	**Notes**
December 18, 2017	1.0	Initial Draft

List of Tables

List of Figures

1 Introduction

This document serves as the detailed Test Report for the Library of Linear Algebraic Equation Solver. It shows the detail results of the execution of Library of Linear Algebraic Equation Solver Test Plan. The requirements are described in Commonality Analysis of Library of Linear Algebraic Equation Solver. The aim of the Test Report is to show that Library of Linear Algebraic Equation Solver produces accurate and valid results.

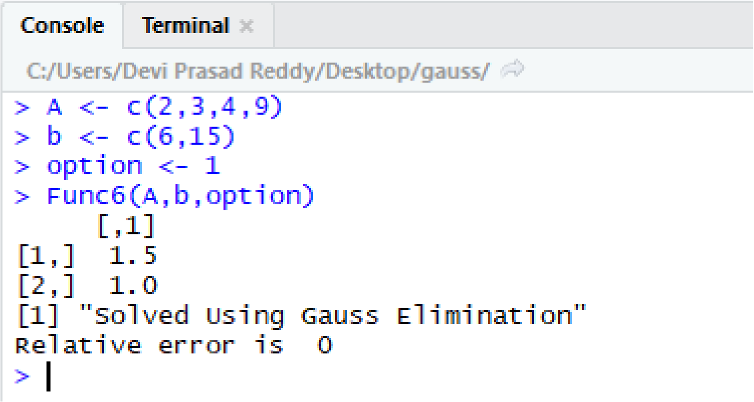
2 Functional Requirements Evaluation

2.1 Calculation Tests

2.1.1 Using Gaussian Elimination: T1, T2, T3, T4

Test Case: T1

Input: A = (2, 3, 4, 9) b = (6, 15)
Expected Output: x = (1.5, 1). This test passes.

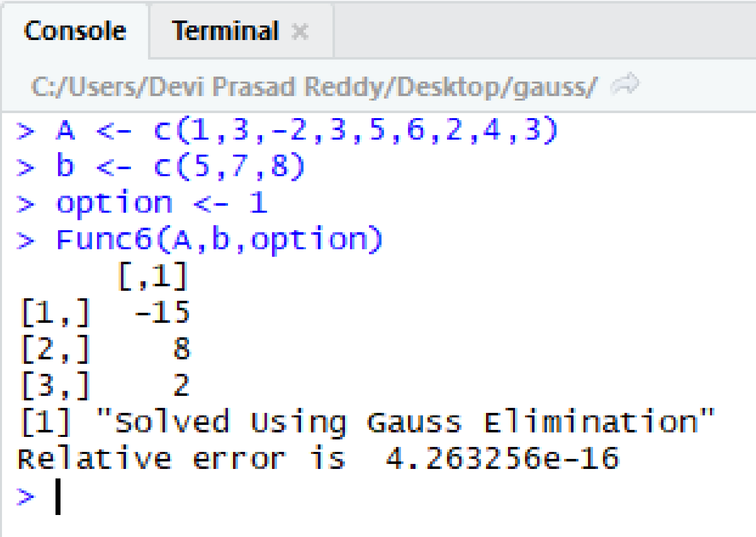


```
> A <- c(2,3,4,9)
> b <- c(6,15)
> option <- 1
> Func6(A,b,option)
     [,1]
[1,]  1.5
[2,]  1.0
[1] "Solved Using Gauss Elimination"
Relative error is  0
> 
```

Figure 1: T1 result.

Test Case: T2

Input: A = (1, 3, -2, 3, 5, 6, 2, 4, 3) b = (5, 7, 8)
Expected Output: x = (-15, 8, 2). This test passes.



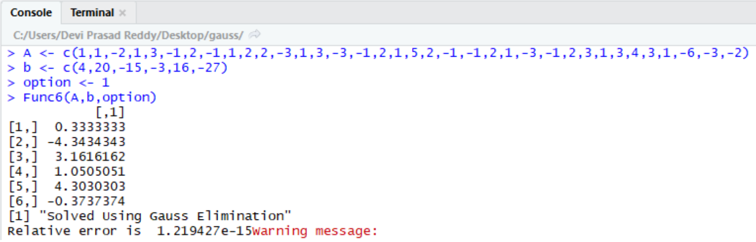
```
C:/Users/Devi Prasad Reddy/Desktop/gauss/
> A <- c(1,3,-2,3,5,6,2,4,3)
> b <- c(5,7,8)
> option <- 1
> Func6(A,b,option)
     [,1]
[1,]  -15
[2,]    8
[3,]    2
[1] "Solved Using Gauss Elimination"
Relative error is  4.263256e-16
> 
```

Figure 2: T2 result.

Test Case: T3

Input: A = (1, 1, -2, 1, 3, -1, 2, -1, 1, 2, 2, -3, 1, 3, -3, -1, 2, 1, 5, 2, -1, -1, 2, 1, -3, -1, 2, 3, 1, 3, 4, 3, 1, -6, -3, -2) b = (4, 20, -15, -3, 16, -27)

Expected Output: x = (1/3, -430/99, 313/99, 104/99, 142/33, -37/99). This test passes.



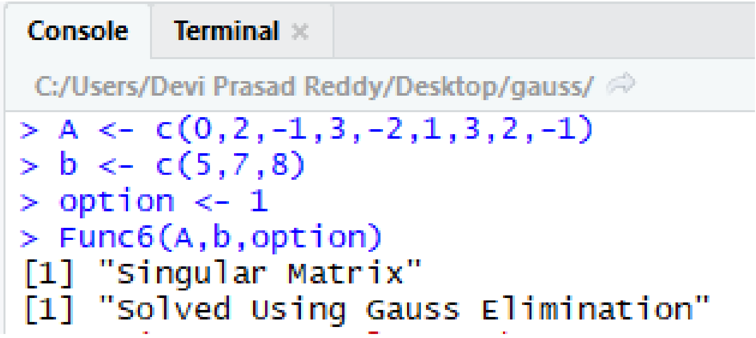
```
C:/Users/Devi Prasad Reddy/Desktop/gauss/
> A <- c(1,1,-2,1,3,-1,2,-1,1,2,2,-3,1,3,-3,-1,2,1,5,2,-1,-1,2,1,-3,-1,2,3,1,3,4,3,1,-6,-3,-2)
> b <- c(4,20,-15,-3,16,-27)
> option <- 1
> Func6(A,b,option)
          [,1]
[1,]  0.3333333
[2,] -4.3434343
[3,]  3.1616162
[4,]  1.0505051
[5,]  4.3030303
[6,] -0.3737374
[1] "Solved Using Gauss Elimination"
Relative error is  1.219427e-15Warning message:
```

Figure 3: T3 result.

Test Case: T4

Input: A = (0, 2, -1, 3, -2, 1, 3, 2, -1) b = (5, 7, 8)

Expected Output: x = Singular Matrix. This test passes.



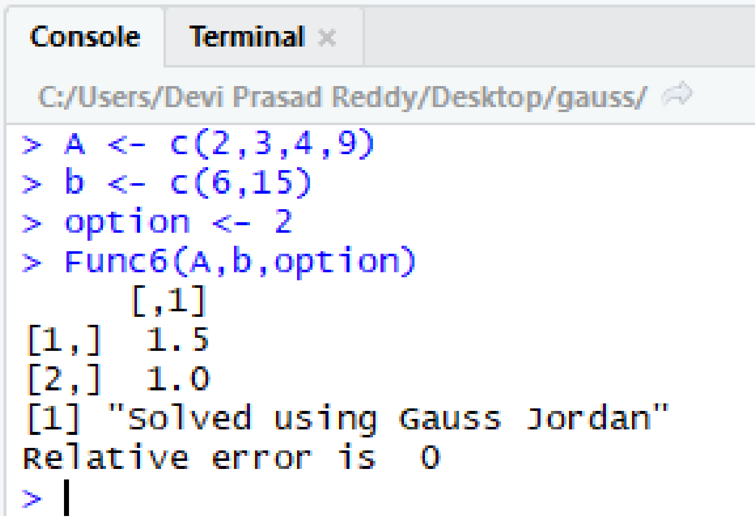
```
> A <- c(0,2,-1,3,-2,1,3,2,-1)
> b <- c(5,7,8)
> option <- 1
> Func6(A,b,option)
[1] "Singular Matrix"
[1] "Solved using Gauss Elimination"
```

Figure 4: T4 result.

2.1.2 Using Gauss-Jordan Elimination: T5, T6, T7, T8

Test Case: T5

Input: A = (2, 3, 4, 9) b = (6, 15)
Expected Output: x = (1.5, 1). This test passes.



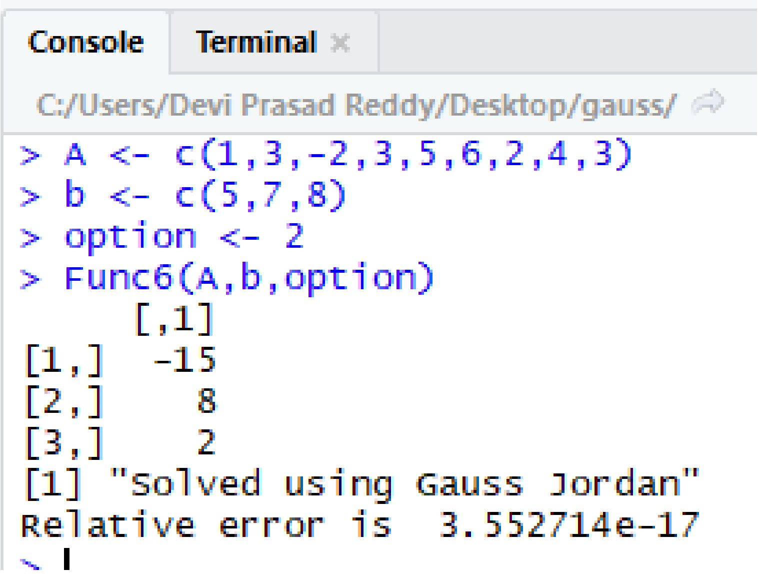
```
> A <- c(2,3,4,9)
> b <- c(6,15)
> option <- 2
> Func6(A,b,option)
     [,1]
[1,]  1.5
[2,]  1.0
[1] "Solved using Gauss Jordan"
Relative error is  0
> 
```

Figure 5: T5 result.

Test Case: T6

Input: A = (1, 3, -2, 3, 5, 6, 2, 4, 3) b = (5, 7, 8)

Expected Output: x = (-15, 8, 2). This test passes.



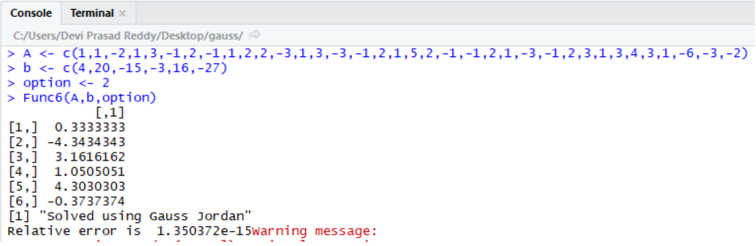
```
C:/Users/Devi Prasad Reddy/Desktop/gauss/
> A <- c(1,3,-2,3,5,6,2,4,3)
> b <- c(5,7,8)
> option <- 2
> Func6(A,b,option)
     [,1]
[1,]  -15
[2,]    8
[3,]    2
[1] "Solved using Gauss Jordan"
Relative error is  3.552714e-17
```

Figure 6: T6 result.

Test Case: T7

Input: A = (1, 1, -2, 1, 3, -1, 2, -1, 1, 2, 2, -3, 1, 3, -3, -1, 2, 1, 5, 2, -1, -1, 2, 1, -3, -1, 2, 3, 1, 3, 4, 3, 1, -6, -3, -2) b = (4, 20, -15, -3, 16, -27)

Expected Output: x = (1/3, -430/99, 313/99, 104/99, 142/33, -37/99). This test passes.

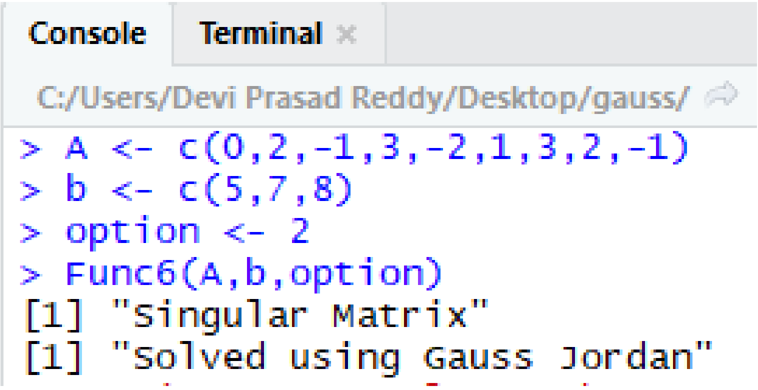


```
C:/Users/Devi Prasad Reddy/Desktop/gauss/
> A <- c(1,1,-2,1,3,-1,2,-1,1,2,2,-3,1,3,-3,-1,2,1,5,2,-1,-1,2,1,-3,-1,2,3,1,3,4,3,1,-6,-3,-2)
> b <- c(4,20,-15,-3,16,-27)
> option <- 2
> Func6(A,b,option)
           [,1]
[1,]  0.3333333
[2,] -4.3434343
[3,]  3.1616162
[4,]  1.0505051
[5,]  4.3030303
[6,] -0.3737374
[1] "Solved using Gauss Jordan"
Relative error is  1.350372e-15Warning message:
```

Figure 7: T7 result.

Test Case: T8

Input: A = (0, 2, -1, 3, -2, 1, 3, 2, -1) b = (5, 7, 8)
Expected Output: x = Singular Matrix. This test passes.



```
> A <- c(0,2,-1,3,-2,1,3,2,-1)
> b <- c(5,7,8)
> option <- 2
> Func6(A,b,option)
[1] "Singular Matrix"
[1] "Solved using Gauss Jordan"
```

Figure 8: T8 result.

3 Nonfunctional Requirements Evaluation

3.1 Accuracy Test

Test Case: T9

$\epsilon_{Relative}$ will be calculated by comparing the result obtained by Library of Linear Algebraic Equation Solver and the RStudio's optR library functional programs by the following equation as norm

$$\epsilon_{\text{rel}} = \text{norm} = \frac{||x_{\text{R}} - x_{\text{LAES}}||}{||x_{\text{R}}||}$$

Test	$\epsilon_{Relative}$
T1	0
T2	4.26×10^{-16}
T3	1.21×10^{-15}
T5	0
T6	4.26×10^{-16}
T7	1.21×10^{-15}

4 Comparison to Existing Implementation

In this section the comparison is done between the Library of Linear Algebraic Solver and Rstudio's optR library by unit testing and by using Functional Requirement Tests.

Test Case: T10

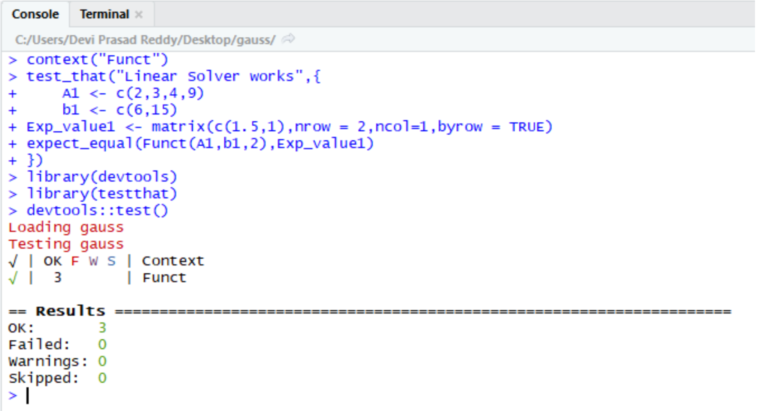
The following input parameters are tested as above in Functional Requirement Tests.

- Test 1 (4.1): $A = (2, 3, 4, 9), b = (6, 15)$
- Test 2 (4.2): $A = (1, 3, -2, 3, 5, 6, 2, 4, 3), b = (5, 7, 8)$
- Test 3 (4.3): $A = (1, 1, -2, 1, 3, -1, 2, -1, 1, 2, 2, -3, 1, 3, -3, -1, 2, 1, 5, 2, -1, -1, 2, 1, -3, -1,$ $2, 3, 1, 3, 4, 3, 1, -6, -3, -2)$

  $b = (4, 20, -15, -3, 16, -27)$

4.1 Test 1

$A = (2, 3, 4, 9), b = (6, 15)$

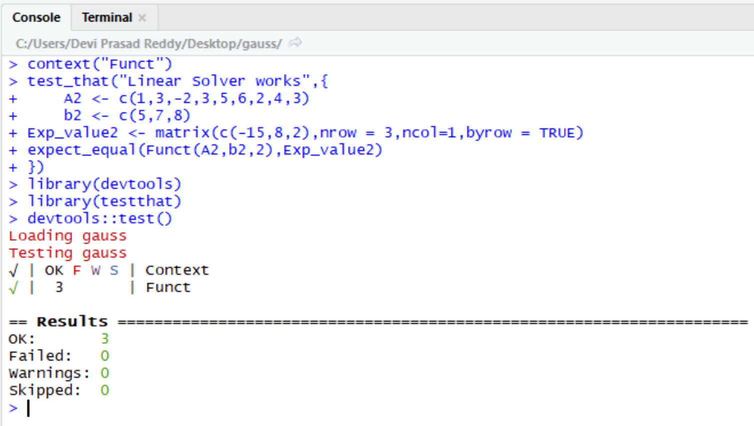


```
> context("Funct")
> test_that("Linear Solver works",{
+     A1 <- c(2,3,4,9)
+     b1 <- c(6,15)
+ Exp_value1 <- matrix(c(1.5,1),nrow = 2,ncol=1,byrow = TRUE)
+ expect_equal(Funct(A1,b1,2),Exp_value1)
+ })
> library(devtools)
> library(testthat)
> devtools::test()
Loading gauss
Testing gauss
√ | OK F W S | Context
√ |  3       | Funct

== Results =====================================================================
OK:       3
Failed:   0
Warnings: 0
Skipped:  0
> |
```

Figure 9: T10 Test1 result.

4.2 Test 2

$A = (1, 3, -2, 3, 5, 6, 2, 4, 3), b = (5, 7, 8)$

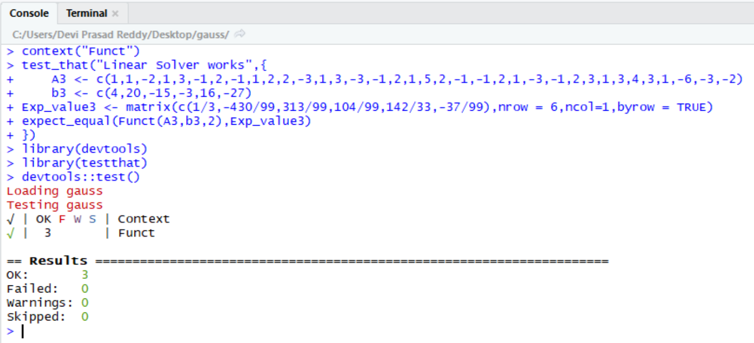


```
> context("Funct")
> test_that("Linear Solver works",{
+     A2 <- c(1,3,-2,3,5,6,2,4,3)
+     b2 <- c(5,7,8)
+ Exp_value2 <- matrix(c(-15,8,2),nrow = 3,ncol=1,byrow = TRUE)
+ expect_equal(Funct(A2,b2,2),Exp_value2)
+ })
> library(devtools)
> library(testthat)
> devtools::test()
Loading gauss
Testing gauss
√ | OK F W S | Context
√ |  3       | Funct

== Results =====================================================================
OK:       3
Failed:   0
Warnings: 0
Skipped:  0
> |
```

Figure 10: T10 Test2 result.

4.3 Test 3

$$A = (1, 1, -2, 1, 3, -1, 2, -1, 1, 2, 2, -3, 1, 3, -3, -1, 2, 1, 5, 2, -1, -1, 2, 1, -3, -1, 2, 3, 1, 3, 4, 3, 1, -6, -3, -2),\ b = (4, 20, -15, -3, 16, -27)$$



```
> context("Funct")
> test_that("Linear Solver works",{
+     A3 <- c(1,1,-2,1,3,-1,2,-1,1,2,2,-3,1,3,-3,-1,2,1,5,2,-1,-1,2,1,-3,-1,2,3,1,3,4,3,1,-6,-3,-2)
+     b3 <- c(4,20,-15,-3,16,-27)
+ Exp_value3 <- matrix(c(1/3,-430/99,313/99,104/99,142/33,-37/99),nrow = 6,ncol=1,byrow = TRUE)
+ expect_equal(Funct(A3,b3,2),Exp_value3)
+ })
> library(devtools)
> library(testthat)
> devtools::test()
Loading gauss
Testing gauss
√ | OK F W S | Context
√ |  3       | Funct

== Results =====================================================================
OK:       3
Failed:   0
Warnings: 0
Skipped:  0
> |
```

Figure 11: T10 Test3 result.

5 Unit Testing

Unit testing was performed, the results and executions are similar to the Functional Requirements Evaluation which is section 2 and Comparison to Existing Implementation which is section 4.

Unit testing is implemented by Rstudio's Unit Testing packages such as "devtools" and "testhat".

6 Changes Due to Testing

None

7 Automated Testing

Automated Testing is performed in section 3 and section 4.

8 Trace to Requirements

The following table shows the traceability mapping for the test cases to the requirements described in the Commonality Analysis.

Table 2: Requirements Traceability Matrix

Test Number	CA Requirements
T1	IM1, C1
T2	IM1, C1
T3	IM1, C1
T4	IM1, C1
T5	IM2, C1
T6	IM2, C1
T7	IM2, C1
T8	IM2, C1
T9	IM1, IM2
T10	IM1, Im2

9 Trace to Modules

The following table shows the traceability mapping for the test cases to the modules in the Module Guide (MG).

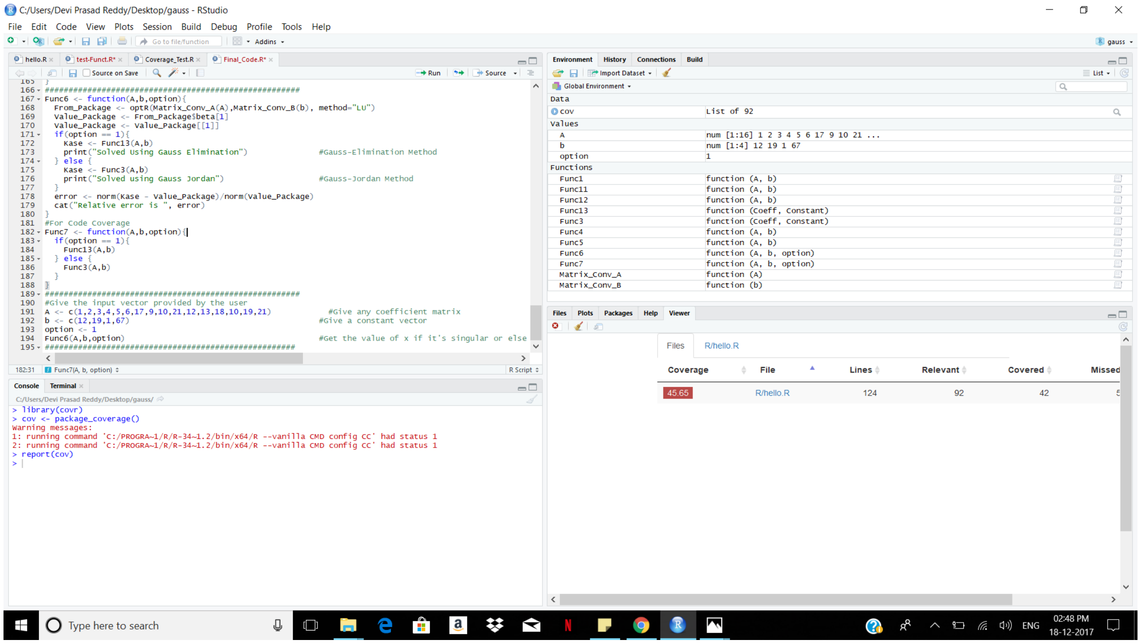
Table 3: Design Traceability Matrix

Test Number	**MG Modules**
T1	M1, M2, M3, M4, M5, M6
T2	M1, M2, M3, M4, M5, M6
T3	M1, M2, M3, M4, M5, M6
T4	M1, M2, M3, M4, M5, M6
T5	M1, M2, M3, M4, M6, M7
T6	M1, M2, M3, M4, M5, M7
T7	M1, M2, M3, M4, M5, M7
T8	M1, M2, M3, M4, M5, M7
T9	M1, M2, M3, M4, M5, M6, M7
T10	M1, M2, M3, M4, M5, M6, M7

10 Code Coverage Metrics

The following Code Coverage figure was obtained from Rstudio's Code Coverage Library “Covr”. Running the test program with in-boundary conditions, the code coverage obtained is:

10.1 Using Gaussian Elimination



```r
}
############################################################
Func6 <- function(A,b,option){
  From_Package <- optR(Matrix_Conv_A(A),Matrix_Conv_B(b), method="LU")
  Value_Package <- From_Package$beta[1]
  Value_Package <- Value_Package[[1]]
  if(option == 1){
    Kase <- Func13(A,b)
    print("Solved Using Gauss Elimination")                #Gauss-Elimination Method
  } else {
    Kase <- Func3(A,b)
    print("Solved using Gauss Jordan")                     #Gauss-Jordan Method
  }
  error <- norm(Kase - Value_Package)/norm(Value_Package)
  cat("Relative error is ", error)
}
#For Code Coverage
Func7 <- function(A,b,option){
  if(option == 1){
    Func13(A,b)
  } else {
    Func3(A,b)
  }
}
############################################################
#Give the input vector provided by the user
A <- c(1,2,3,4,5,6,17,9,10,21,12,13,18,10,19,21)           #Give any coefficient matrix
b <- c(12,19,1,67)                                         #Give a constant vector
option <- 1
Func6(A,b,option)                                          #Get the value of x if it's singular or else
############################################################
```

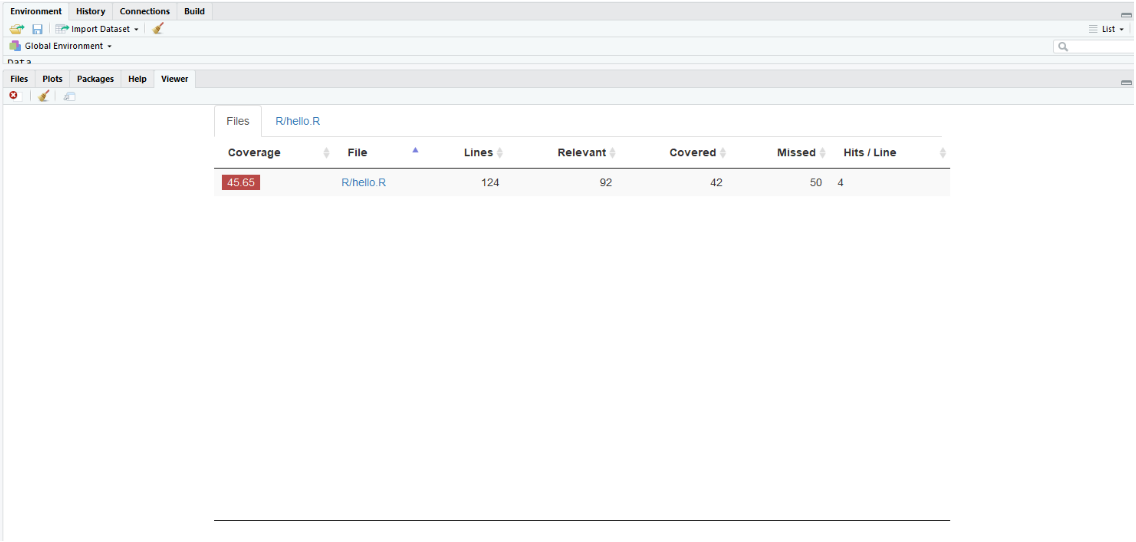
```
C:/Users/Devi Prasad Reddy/Desktop/gauss/
> library(covr)
> cov <- package_coverage()
Warning messages:
1: running command 'C:/PROGRA~1/R/R-34~1.2/bin/x64/R --vanilla CMD config CC' had status 1
2: running command 'C:/PROGRA~1/R/R-34~1.2/bin/x64/R --vanilla CMD config CC' had status 1
> report(cov)
> 
```

Data	
cov	List of 92
Values	
A	num [1:16] 1 2 3 4 5 6 17 9 10 21 ...
b	num [1:4] 12 19 1 67
option	1
Functions	
Func1	function (A, b)
Func11	function (A, b)
Func12	function (A, b)
Func13	function (Coeff, Constant)
Func3	function (Coeff, Constant)
Func4	function (A, b)
Func5	function (A, b)
Func6	function (A, b, option)
Func7	function (A, b, option)
Matrix_Conv_A	function (A)
Matrix_Conv_B	function (b)

Files R/hello.R

Coverage	File	Lines	Relevant	Covered	Missed
45.65	R/hello.R	124	92	42	5

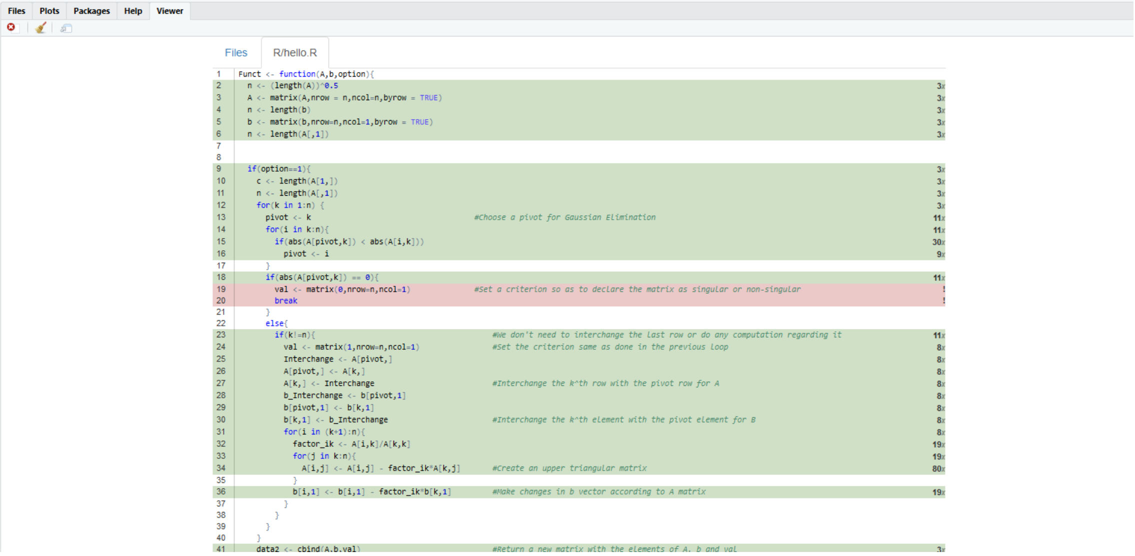
Figure 12: Code Coverage 1



Files R/hello.R

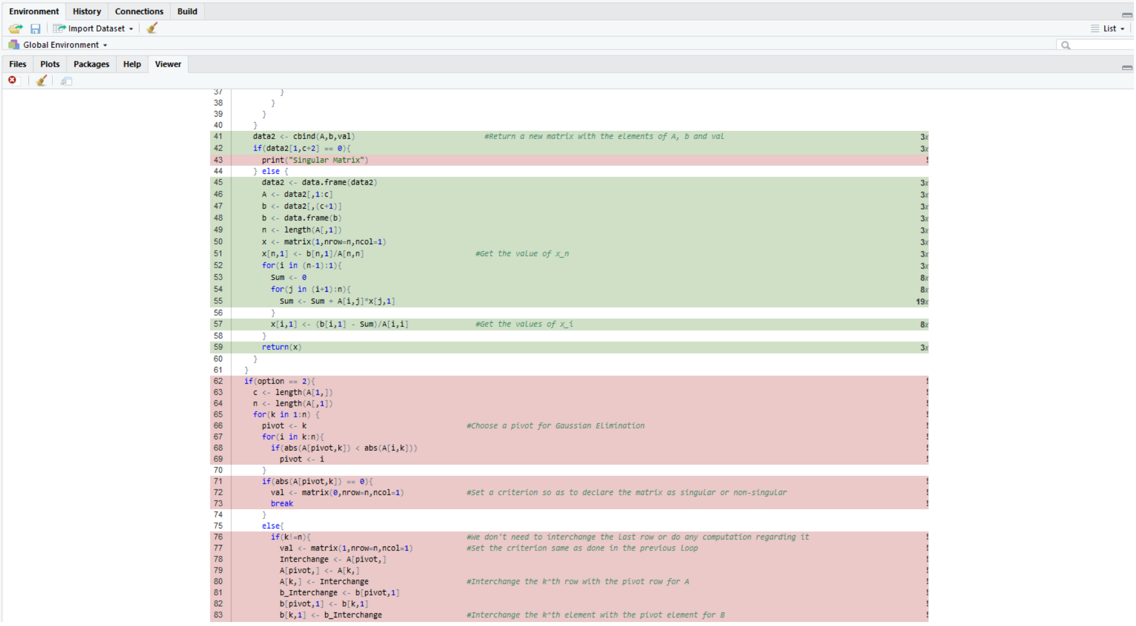
Coverage	File	Lines	Relevant	Covered	Missed	Hits / Line
45.65	R/hello.R	124	92	42	50	4

Figure 13: Code Coverage 2



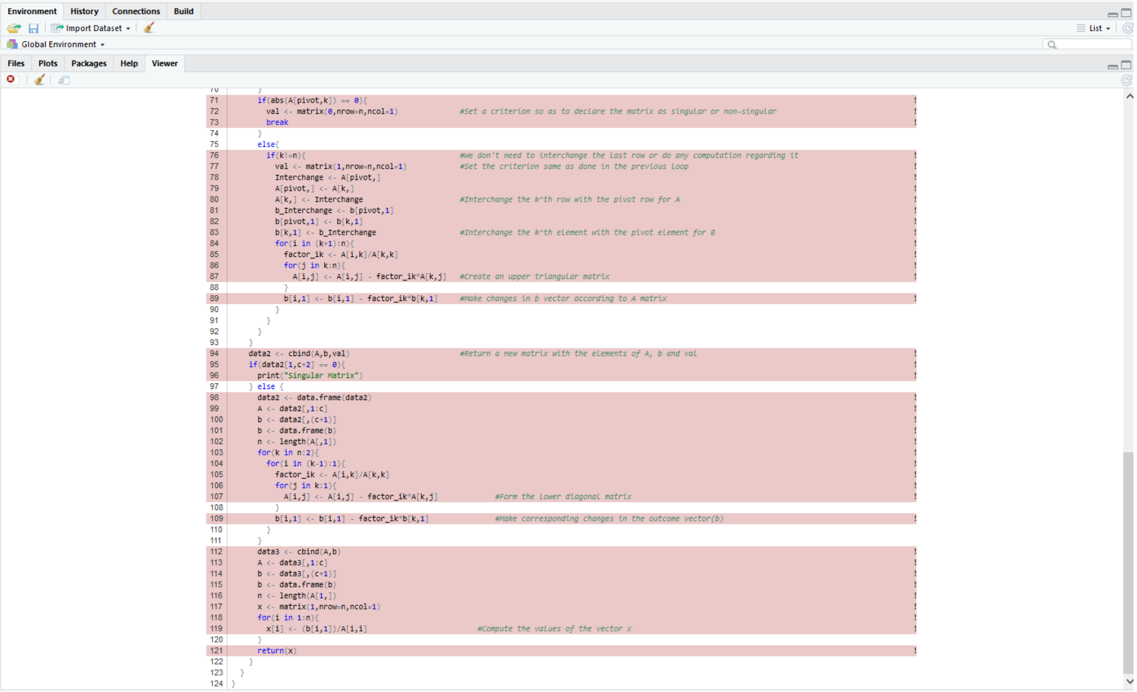
```r
Funct <- function(A,b,option){
  n <- (length(A))^0.5
  A <- matrix(A,nrow = n,ncol=n,byrow = TRUE)
  n <- length(b)
  b <- matrix(b,nrow=n,ncol=1,byrow = TRUE)
  n <- length(A[,1])


  if(option==1){
    c <- length(A[1,])
    n <- length(A[,1])
    for(k in 1:n) {
      pivot <- k                                 #Choose a pivot for Gaussian Elimination
      for(i in k:n){
        if(abs(A[pivot,k]) < abs(A[i,k]))
          pivot <- i
      }
      if(abs(A[pivot,k]) == 0){
        val <- matrix(0,nrow=n,ncol=1)           #Set a criterion so as to declare the matrix as singular or non-singular
        break
      }
      else{
        if(k!=n){                                  #We don't need to interchange the last row or do any computation regarding it
          val <- matrix(1,nrow=n,ncol=1)           #Set the criterion same as done in the previous loop
          Interchange <- A[pivot,]
          A[pivot,] <- A[k,]
          A[k,] <- Interchange                     #Interchange the k^th row with the pivot row for A
          b_Interchange <- b[pivot,1]
          b[pivot,1] <- b[k,1]
          b[k,1] <- b_Interchange                  #Interchange the k^th element with the pivot element for B
          for(i in (k+1):n){
            factor_ik <- A[i,k]/A[k,k]
            for(j in k:n){
              A[i,j] <- A[i,j] - factor_ik*A[k,j]  #Create an upper triangular matrix
            }
            b[i,1] <- b[i,1] - factor_ik*b[k,1]    #Make changes in b vector according to A matrix
          }
        }
      }
    }
    data2 <- cbind(A,b,val)                        #Return a new matrix with the elements of A, b and val
```

Figure 14: Code Coverage 3



```r
          }
        }
      }
    }
    data2 <- cbind(A,b,val)                        #Return a new matrix with the elements of A, b and val
    if(data2[1,c+2] == 0){
      print("Singular Matrix")
    } else {
      data2 <- data.frame(data2)
      A <- data2[,1:c]
      b <- data2[,(c+1)]
      b <- data.frame(b)
      n <- length(A[,1])
      x <- matrix(1,nrow=n,ncol=1)
      x[n,1] <- b[n,1]/A[n,n]                      #Get the value of x_n
      for(i in (n-1):1){
        Sum <- 0
        for(j in (i+1):n){
          Sum <- Sum + A[i,j]*x[j,1]
        }
        x[i,1] <- (b[i,1] - Sum)/A[i,i]            #Get the values of x_i
      }
      return(x)
    }
  }
  if(option == 2){
    c <- length(A[1,])
    n <- length(A[,1])
    for(k in 1:n) {
      pivot <- k                                   #Choose a pivot for Gaussian Elimination
      for(i in k:n){
        if(abs(A[pivot,k]) < abs(A[i,k]))
          pivot <- i
      }
      if(abs(A[pivot,k]) == 0){
        val <- matrix(0,nrow=n,ncol=1)             #Set a criterion so as to declare the matrix as singular or non-singular
        break
      }
      else{
        if(k!=n){                                  #We don't need to interchange the last row or do any computation regarding it
          val <- matrix(1,nrow=n,ncol=1)           #Set the criterion same as done in the previous loop
          Interchange <- A[pivot,]
          A[pivot,] <- A[k,]
          A[k,] <- Interchange                     #Interchange the k^th row with the pivot row for A
          b_Interchange <- b[pivot,1]
          b[pivot,1] <- b[k,1]
          b[k,1] <- b_Interchange                  #Interchange the k^th element with the pivot element for B
```

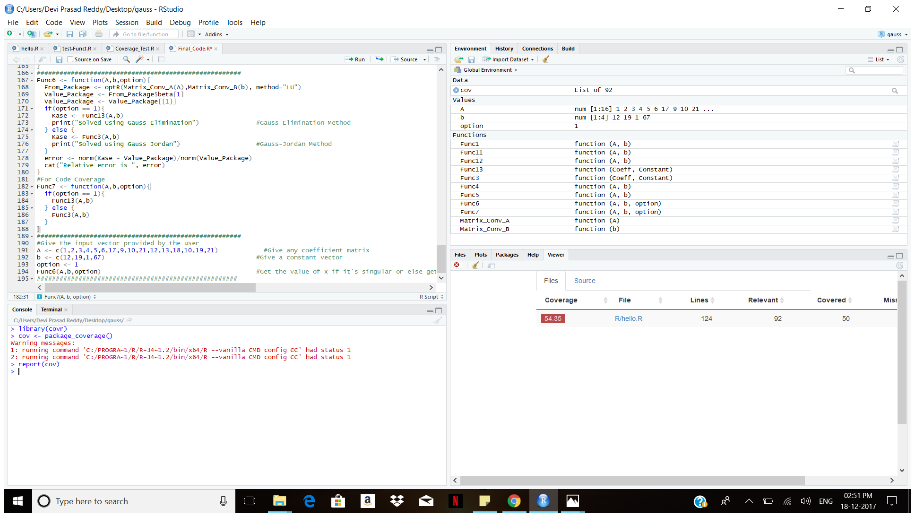
Figure 15: Code Coverage 4



```
      }
      if(abs(A[pivot,k]) == 0){
        val <- matrix(0,nrow=n,ncol=1)             #Set a criterion so as to declare the matrix as singular or non-singular
        break
      }
      else{
        if(k!=n){                                  #We don't need to interchange the last row or do any computation regarding it
          val <- matrix(1,nrow=n,ncol=1)           #Set the criterion same as done in the previous loop
          Interchange <- A[pivot,]
          A[pivot,] <- A[k,]
          A[k,] <- Interchange                     #Interchange the k^th row with the pivot row for A
          b_Interchange <- b[pivot,1]
          b[pivot,1] <- b[k,1]
          b[k,1] <- b_Interchange                  #Interchange the k^th element with the pivot element for B
          for(i in (k+1):n){
            factor_ik <- A[i,k]/A[k,k]
            for(j in k:n){
              A[i,j] <- A[i,j] - factor_ik*A[k,j]  #Create an upper triangular matrix
            }
            b[i,1] <- b[i,1] - factor_ik*b[k,1]    #Make changes in b vector according to A matrix
          }
        }
      }
    }
    data2 <- cbind(A,b,val)                        #Return a new matrix with the elements of A, b and val
    if(data2[1,c+2] == 0){
      print("Singular Matrix")
    } else {
      data2 <- data.frame(data2)
      A <- data2[,1:c]
      b <- data2[,(c+1)]
      b <- data.frame(b)
      n <- length(A[,1])
      for(k in n:2){
        for(i in (k-1):1){
          factor_ik <- A[i,k]/A[k,k]
          for(j in k:1){
            A[i,j] <- A[i,j] - factor_ik*A[k,j]            #Form the lower diagonal matrix
          }
          b[i,1] <- b[i,1] - factor_ik*b[k,1]              #Make corresponding changes in the outcome vector(b)
        }
      }
      data3 <- cbind(A,b)
      A <- data3[,1:c]
      b <- data3[,(c+1)]
      b <- data.frame(b)
      n <- length(A[1,])
      x <- matrix(1,nrow=n,ncol=1)
      for(i in 1:n){
        x[i] <- (b[i,1])/A[i,i]                        #Compute the values of the vector x
      }
      return(x)
    }
  }
}
```

Figure 16: Code Coverage 5

10.2 Using Gauss-Jordan Elimination



```r
}
########################################################
Func6 <- function(A,b,option){
  From_Package <- optR(Matrix_Conv_A(A),Matrix_Conv_B(b), method="LU")
  Value_Package <- From_Package$beta[1]
  Value_Package <- Value_Package[[1]]
  if(option == 1){
    Kase <- Func13(A,b)
    print("Solved Using Gauss Elimination")            #Gauss-Elimination Method
  } else {
    Kase <- Func3(A,b)
    print("Solved using Gauss Jordan")                 #Gauss-Jordan Method
  }
  error <- norm(Kase - Value_Package)/norm(Value_Package)
  cat("Relative error is ", error)
}
#For Code Coverage
Func7 <- function(A,b,option){
  if(option == 1){
    Func13(A,b)
  } else {
    Func3(A,b)
  }
}
########################################################
#Give the input vector provided by the user
A <- c(1,2,3,4,5,6,17,9,10,21,12,13,18,10,19,21)        #Give any coefficient matrix
b <- c(12,19,1,67)                                     #Give a constant vector
option <- 1
Func6(A,b,option)                                      #Get the value of x if it's singular or else get
#######################################################
```

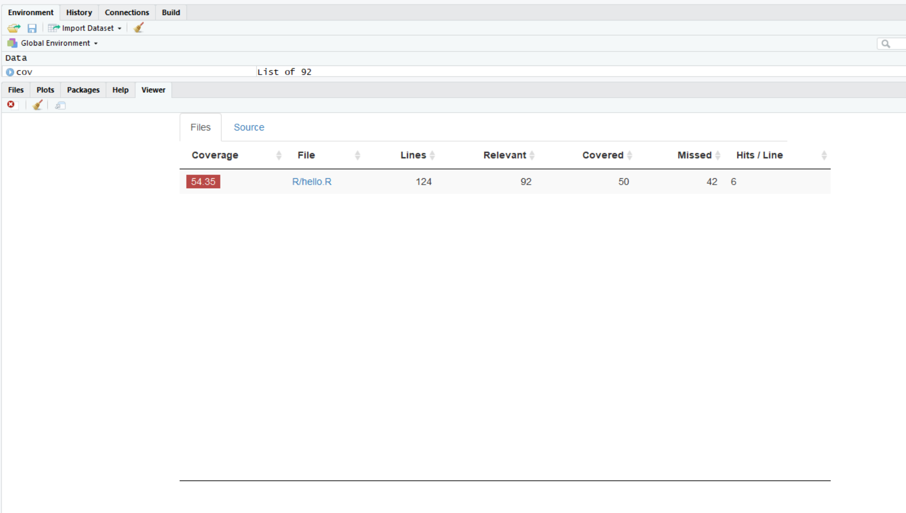
```
> library(covr)
> cov <- package_coverage()
Warning messages:
1: running command 'C:/PROGRA~1/R/R-34~1.2/bin/x64/R --vanilla CMD config CC' had status 1
2: running command 'C:/PROGRA~1/R/R-34~1.2/bin/x64/R --vanilla CMD config CC' had status 1
> report(cov)
> 
```

```
Data
cov             List of 92
Values
  A             num [1:16] 1 2 3 4 5 6 17 9 10 21 ...
  b             num [1:4] 12 19 1 67
  option        1
Functions
  Func1         function (A, b)
  Func11        function (A, b)
  Func12        function (A, b)
  Func13        function (Coeff, Constant)
  Func3         function (Coeff, Constant)
  Func4         function (A, b)
  Func5         function (A, b)
  Func6         function (A, b, option)
  Func7         function (A, b, option)
  Matrix_Conv_A function (A)
  Matrix_Conv_B function (b)
```

Files | Source

Coverage	File	Lines	Relevant	Covered	Miss
54.35	R/hello.R	124	92	50	

Figure 17: Code Coverage 6

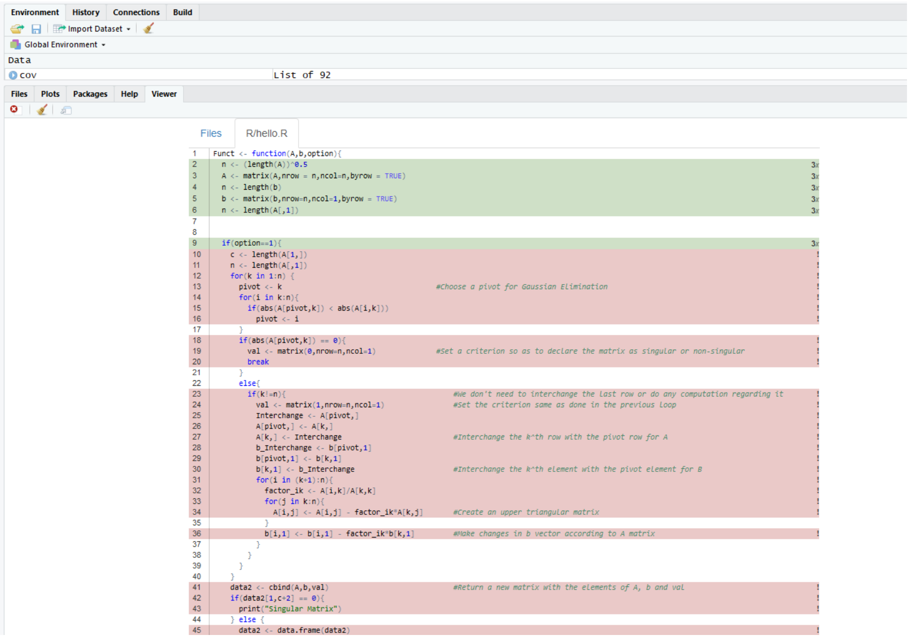


```
Data
cov             List of 92
```

Files | Source

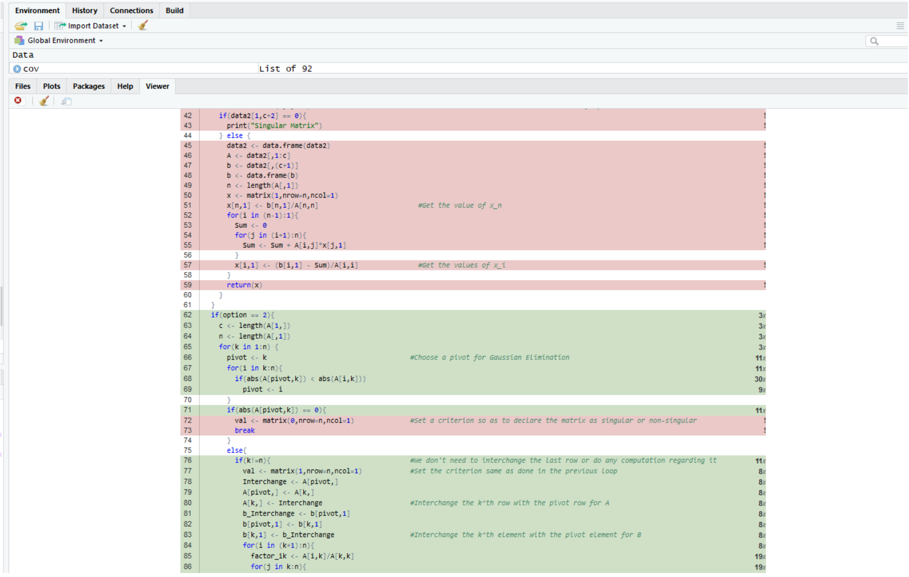
Coverage	File	Lines	Relevant	Covered	Missed	Hits / Line
54.35	R/hello.R	124	92	50	42	6

Figure 18: Code Coverage 7



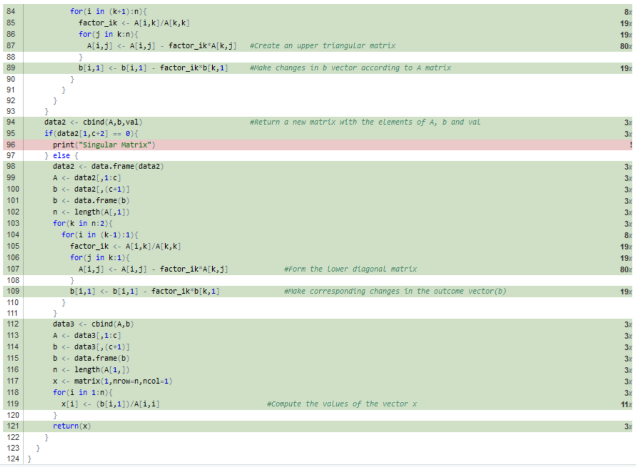
```
Funct <- function(A,b,option){
  n <- (length(A))^0.5
  A <- matrix(A,nrow = n,ncol=n,byrow = TRUE)
  n <- length(b)
  b <- matrix(b,nrow=n,ncol=1,byrow = TRUE)
  n <- length(A[,1])


  if(option==1){
    c <- length(A[1,])
    n <- length(A[,1])
    for(k in 1:n) {
      pivot <- k                                    #Choose a pivot for Gaussian Elimination
      for(i in k:n){
        if(abs(A[pivot,k]) < abs(A[i,k]))
          pivot <- i
      }
      if(abs(A[pivot,k]) == 0){
        val <- matrix(0,nrow=n,ncol=1)              #Set a criterion so as to declare the matrix as singular or non-singular
        break
      }
      else{
        if(k!=n){                                       #We don't need to interchange the last row or do any computation regarding it
          val <- matrix(1,nrow=n,ncol=1)                #Set the criterion same as done in the previous Loop
          Interchange <- A[pivot,]
          A[pivot,] <- A[k,]
          A[k,] <- Interchange                          #Interchange the k^th row with the pivot row for A
          b_Interchange <- b[pivot,1]
          b[pivot,1] <- b[k,1]
          b[k,1] <- b_Interchange                       #Interchange the k^th element with the pivot element for B
          for(i in (k+1):n){
            factor_ik <- A[i,k]/A[k,k]
            for(j in k:n){
              A[i,j] <- A[i,j] - factor_ik*A[k,j]       #Create an upper triangular matrix
            }
            b[i,1] <- b[i,1] - factor_ik*b[k,1]         #Make changes in b vector according to A matrix
          }
        }
      }
    }
    data2 <- cbind(A,b,val)                             #Return a new matrix with the elements of A, b and val
    if(data2[1,c+2] == 0){
      print("Singular Matrix")
    } else {
      data2 <- data.frame(data2)
```

Figure 19: Code Coverage 8



```
    if(data2[1,c+2] == 0){
      print("Singular Matrix")
    } else {
      data2 <- data.frame(data2)
      A <- data2[,1:c]
      b <- data2[,(c+1)]
      b <- data.frame(b)
      n <- length(A[,1])
      x <- matrix(1,nrow=n,ncol=1)
      x[n,1] <- b[n,1]/A[n,n]                           #Get the value of x_n
      for(i in (n-1):1){
        Sum <- 0
        for(j in (i+1):n){
          Sum <- Sum + A[i,j]*x[j,1]
        }
        x[i,1] <- (b[i,1] - Sum)/A[i,i]                 #Get the values of x_i
      }
      return(x)
    }
  }
  if(option == 2){
    c <- length(A[1,])
    n <- length(A[,1])
    for(k in 1:n) {
      pivot <- k                                    #Choose a pivot for Gaussian Elimination
      for(i in k:n){
        if(abs(A[pivot,k]) < abs(A[i,k]))
          pivot <- i
      }
      if(abs(A[pivot,k]) == 0){
        val <- matrix(0,nrow=n,ncol=1)              #Set a criterion so as to declare the matrix as singular or non-singular
        break
      }
      else{
        if(k!=n){                                   #We don't need to interchange the last row or do any computation regarding it
          val <- matrix(1,nrow=n,ncol=1)            #Set the criterion same as done in the previous Loop
          Interchange <- A[pivot,]
          A[pivot,] <- A[k,]
          A[k,] <- Interchange                      #Interchange the k^th row with the pivot row for A
          b_Interchange <- b[pivot,1]
          b[pivot,1] <- b[k,1]
          b[k,1] <- b_Interchange                   #Interchange the k^th element with the pivot element for B
          for(i in (k+1):n){
            factor_ik <- A[i,k]/A[k,k]
            for(j in k:n){
```

Figure 20: Code Coverage 9



```
          for(i in (k+1):n){                                                                                          8x
            factor_ik <- A[i,k]/A[k,k]                                                                                  19x
            for(j in k:n){                                                                                              19x
              A[i,j] <- A[i,j] - factor_ik*A[k,j]   #Create an upper triangular matrix                                  80x
            }
            b[i,1] <- b[i,1] - factor_ik*b[k,1]     #Make changes in b vector according to A matrix                     19x
          }
        }
      }
    }
    data2 <- cbind(A,b,val)                         #Return a new matrix with the elements of A, b and val              3x
    if(data2[1,c+2] == 0){                                                                                              3x
      print("Singular Matrix")                                                                                          !
    } else {
      data2 <- data.frame(data2)                                                                                        3x
      A <- data2[,1:c]                                                                                                  3x
      b <- data2[,(c+1)]                                                                                                3x
      b <- data.frame(b)                                                                                                3x
      n <- length(A[,1])                                                                                                3x
      for(k in n:2){                                                                                                    3x
        for(i in (k-1):1){                                                                                              8x
          factor_ik <- A[i,k]/A[k,k]                                                                                    19x
          for(j in k:1){                                                                                                19x
            A[i,j] <- A[i,j] - factor_ik*A[k,j]             #Form the lower diagonal matrix                             80x
          }
          b[i,1] <- b[i,1] - factor_ik*b[k,1]               #Make corresponding changes in the outcome vector(b)        19x
        }
      }
      data3 <- cbind(A,b)                                                                                               3x
      A <- data3[,1:c]                                                                                                  3x
      b <- data3[,(c+1)]                                                                                                3x
      b <- data.frame(b)                                                                                                3x
      n <- length(A[1,])                                                                                                3x
      x <- matrix(1,nrow=n,ncol=1)                                                                                      3x
      for(i in 1:n){                                                                                                    3x
        x[i] <- (b[i,1])/A[i,i]                        #Compute the values of the vector x                              11x
      }
      return(x)                                                                                                         3x
    }
  }
}
```

Figure 21: Code Coverage 10