R Programming LAB

Contents

Data Management

1. Working with vectors and Matrices

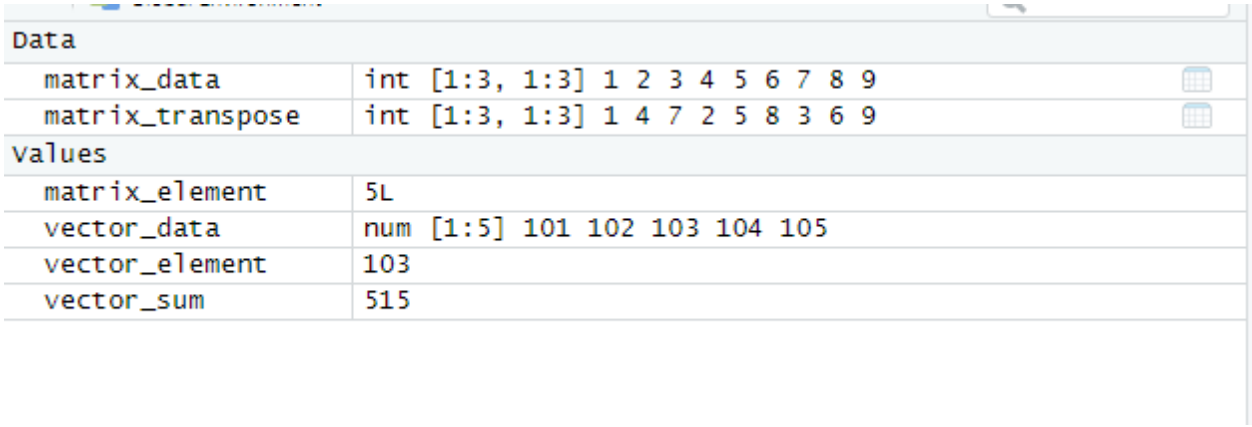
Aim:

To write R program working with vectors and Matrices

Program:

```
# creating a vector and a matrix
vector_data <- c(101,102,103,104,105)
matrix_data <- matrix(1:9, nrow = 3)

# accessing elements in a vector and matrix
vector_element <- vector_data[3]
matrix_element <- matrix_data[2, 2]

# Performing operations on vectors and matrices
vector_sum <- sum(vector_data)
matrix_transpose <- t(matrix_data)
```

Output:



Data	
matrix_data	int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
matrix_transpose	int [1:3, 1:3] 1 4 7 2 5 8 3 6 9
Values	
matrix_element	5L
vector_data	num [1:5] 101 102 103 104 105
vector_element	103
vector_sum	515

Result:

Thus the required output is obtained.

2. Sorting, Merging and Aggregating Data sets

Aim:

To write R program Sorting, Merging and Aggregating Data sets

Program:

```
vector_data <- c(106,109,133,89104,23105)

sorted_vector <- sort(vector_data)

# merging two vectors
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
merged_vector <- c(vector1, vector2)

# aggregating data with mean and sum
data <- c(12, 15, 18, 24, 9)
mean_data <- mean(data)
sum_data <- sum(data)
```

Output:

values

data	num [1:5] 12 15 18 24 9
mean_data	15.6
merged_vector	num [1:6] 1 2 3 4 5 6
sorted_vector	num [1:5] 106 109 133 23105 89104
sum_data	78
vector_data	num [1:5] 106 109 133 89104 23105
vector1	num [1:3] 1 2 3
vector2	num [1:3] 4 5 6

Result:

Thus the required output is obtained.

Testing Statistical Hypothesis using R

3. Test for Single, difference of mean and paired mean

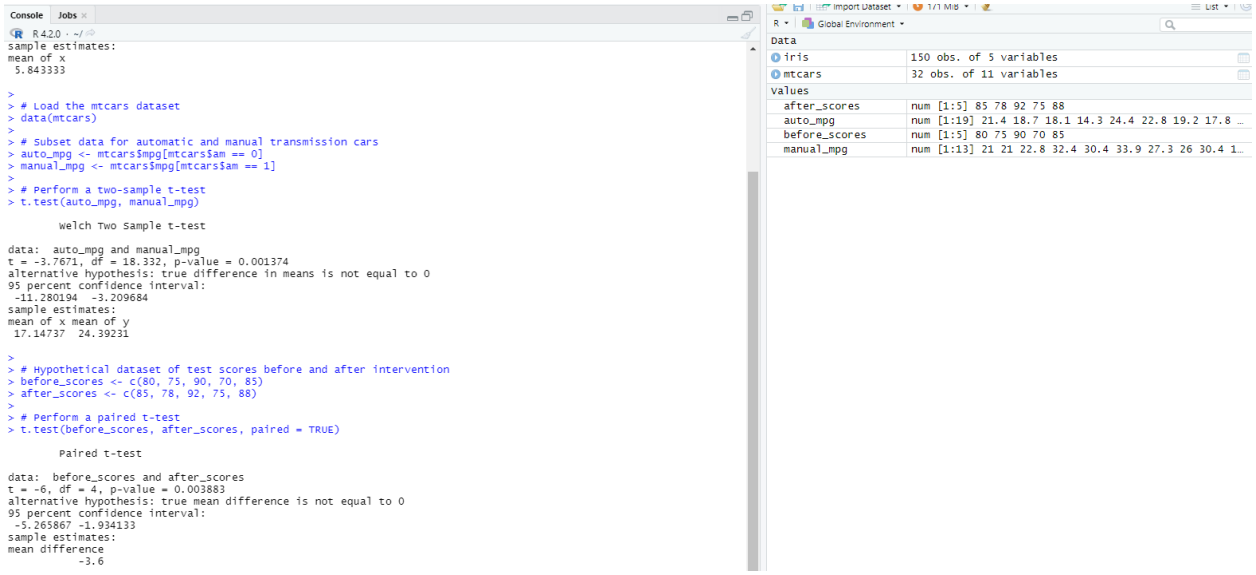
Aim:

To write R program Test for Single, difference of mean and paired mean

Program:

```
# Load the iris dataset
data(iris)

# Perform a one-sample t-test
t.test(iris$Sepal.Length, mu = 5.0)

# Load the mtcars dataset
data(mtcars)

# Subset data for automatic and manual transmission cars
auto_mpg <- mtcars$mpg[mtcars$am == 0]
manual_mpg <- mtcars$mpg[mtcars$am == 1]

# Perform a two-sample t-test
t.test(auto_mpg, manual_mpg)

# Hypothetical dataset of test scores before and after intervention
before_scores <- c(80, 75, 90, 70, 85)
after_scores <- c(85, 78, 92, 75, 88)

# Perform a paired t-test
t.test(before_scores, after_scores, paired = TRUE)
```

Output:



Result:

Thus the required output is obtained.

4. Test for equality of variance

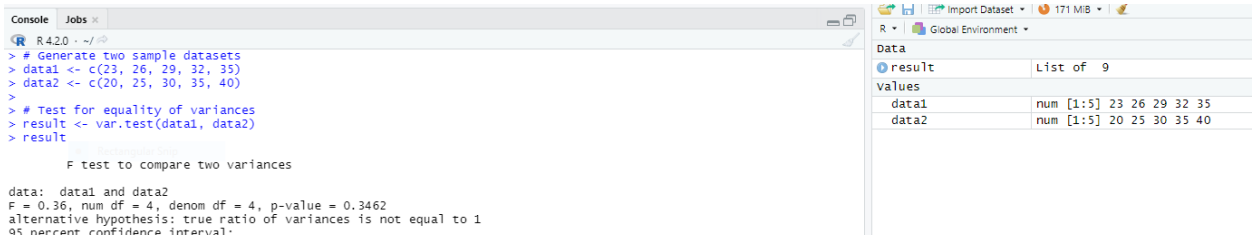
Aim:

To write R program Test for equality of variance.

Program:

```
# Generate two sample datasets
data1 <- c(23, 26, 29, 32, 35)
data2 <- c(20, 25, 30, 35, 40)

# Test for equality of variances
result <- var.test(data1, data2)
result
```

Output:



Result:

Thus the required output is obtained.

5. Applications: Chi-Square test for Goodness of fit and independence of Attributes

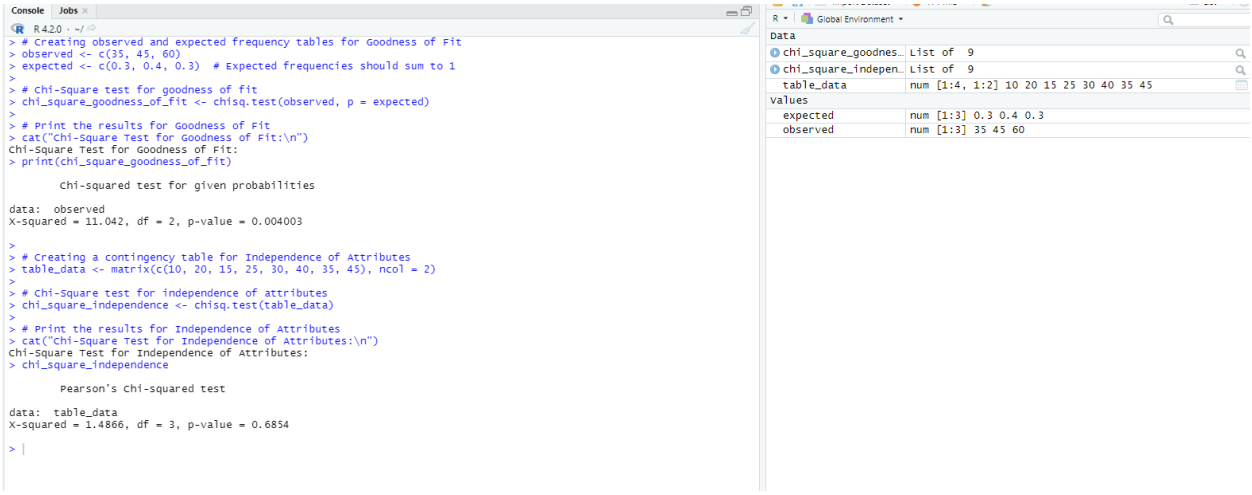
Aim:

To write R program Chi-Square test for Goodness of fit and independence of Attributes.

Program:

```
# Creating observed and expected frequency tables for Goodness of Fit
observed <- c(35, 45, 60)
expected <- c(0.3, 0.4, 0.3)  # Expected frequencies should sum to 1

# Chi-Square test for goodness of fit
chi_square_goodness_of_fit <- chisq.test(observed, p = expected)

# Print the results for Goodness of Fit
cat("Chi-Square Test for Goodness of Fit:\n")
print(chi_square_goodness_of_fit)

# Creating a contingency table for Independence of Attributes
table_data <- matrix(c(10, 20, 15, 25, 30, 40, 35, 45), ncol = 2)

# Chi-Square test for independence of attributes
chi_square_independence <- chisq.test(table_data)

# Print the results for Independence of Attributes
cat("Chi-Square Test for Independence of Attributes:\n")
chi_square_independence
```

Output:



Result:

Thus the required output is obtained.

6. Applications: One way ANOVA and two way ANOVA

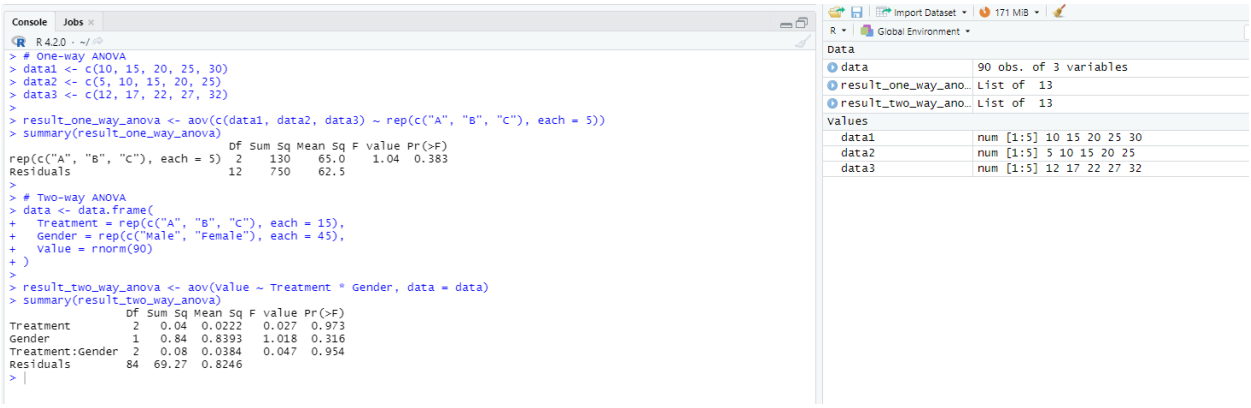
Aim:

To write R program one way ANOVA and two way ANOVA.

Program:

```
# One-way ANOVA
data1 <- c(10, 15, 20, 25, 30)
data2 <- c(5, 10, 15, 20, 25)
data3 <- c(12, 17, 22, 27, 32)

result_one_way_anova <- aov(c(data1, data2, data3) ~ rep(c("A", "B", "C"), each = 5))
summary(result_one_way_anova)

# Two-way ANOVA
data <- data.frame(
  Treatment = rep(c("A", "B", "C"), each = 15),
  Gender = rep(c("Male", "Female"), each = 45),
  Value = rnorm(90)
)

result_two_way_anova <- aov(Value ~ Treatment * Gender, data = data)
summary(result_two_way_anova)
```

Output:



Result:

Thus the required output is obtained.

7. Applications: Latin Square Design

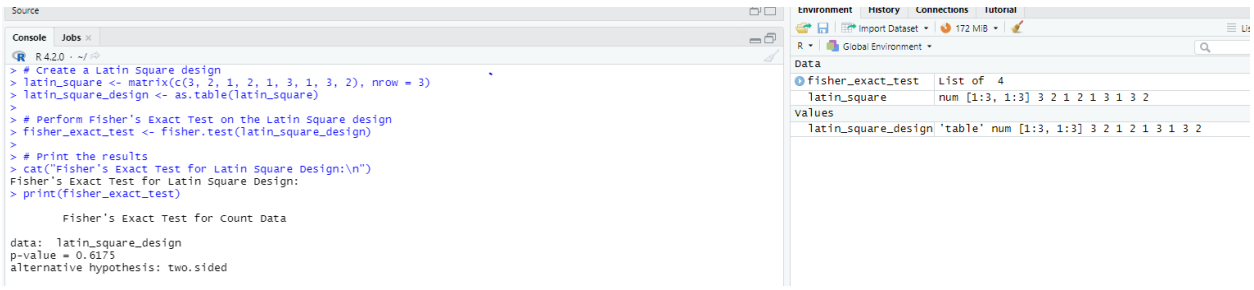
Aim:

To write R program Latin Square Design

Program:

```
# Create a Latin Square design
latin_square <- matrix(c(3, 2, 1, 2, 1, 3, 1, 3, 2), nrow = 3)
latin_square_design <- as.table(latin_square)

# Perform Fisher's Exact Test on the Latin Square design
fisher_exact_test <- fisher.test(latin_square_design)

# Print the results
cat("Fisher's Exact Test for Latin Square Design:\n")
print(fisher_exact_test)
```

Output:



Result:

Thus the required output is obtained.

Numerical Solution of Equations using R

8. Newton-Raphson method

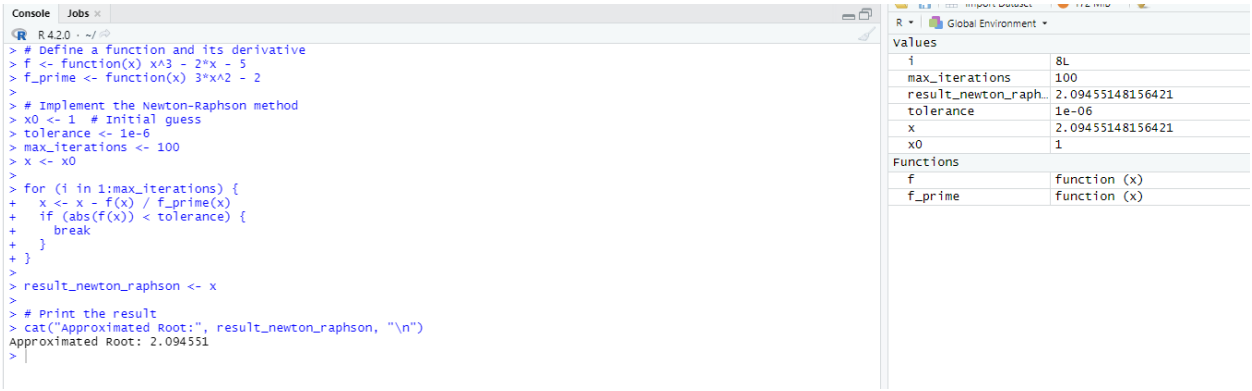
Aim:

To write R program Newton-Raphson method

Program:

```
# Define a function and its derivative
f <- function(x) x^3 - 2*x - 5
f_prime <- function(x) 3*x^2 - 2

# Implement the Newton-Raphson method
x0 <- 1  # Initial guess
tolerance <- 1e-6
max_iterations <- 100
x <- x0

for (i in 1:max_iterations) {
  x <- x - f(x) / f_prime(x)
  if (abs(f(x)) < tolerance) {
    break
  }
}

result_newton_raphson <- x

# Print the result
cat("Approximated Root:", result_newton_raphson, "\n")
```

Output:



Result:

Thus the required output is obtained.

9. Solving system of Linear Equations (Gauss elimination, Gauss Jacobi and Gauss-Seidel)

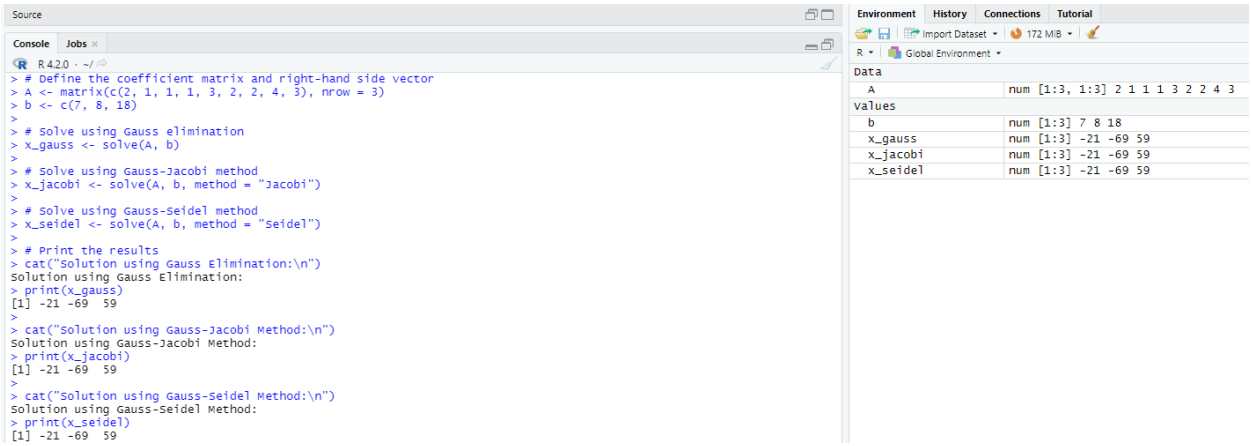
Aim:

To write R program solving system of Linear Equations (Gauss elimination, Gauss Jacobi and Gauss-Seidel)

Program:

```
# Define the coefficient matrix and right-hand side vector
A <- matrix(c(2, 1, 1, 1, 3, 2, 2, 4, 3), nrow = 3)
b <- c(7, 8, 18)

# Solve using Gauss elimination
x_gauss <- solve(A, b)

# Solve using Gauss-Jacobi method
x_jacobi <- solve(A, b, method = "Jacobi")

# Solve using Gauss-Seidel method
x_seidel <- solve(A, b, method = "Seidel")

# Print the results
cat("Solution using Gauss Elimination:\n")
print(x_gauss)

cat("Solution using Gauss-Jacobi Method:\n")
print(x_jacobi)

cat("Solution using Gauss-Seidel Method:\n")
print(x_seidel)
```

Output:



Result:

Thus the required output is obtained.

10. Power method to approximate dominant Eigen value and Eigen vector

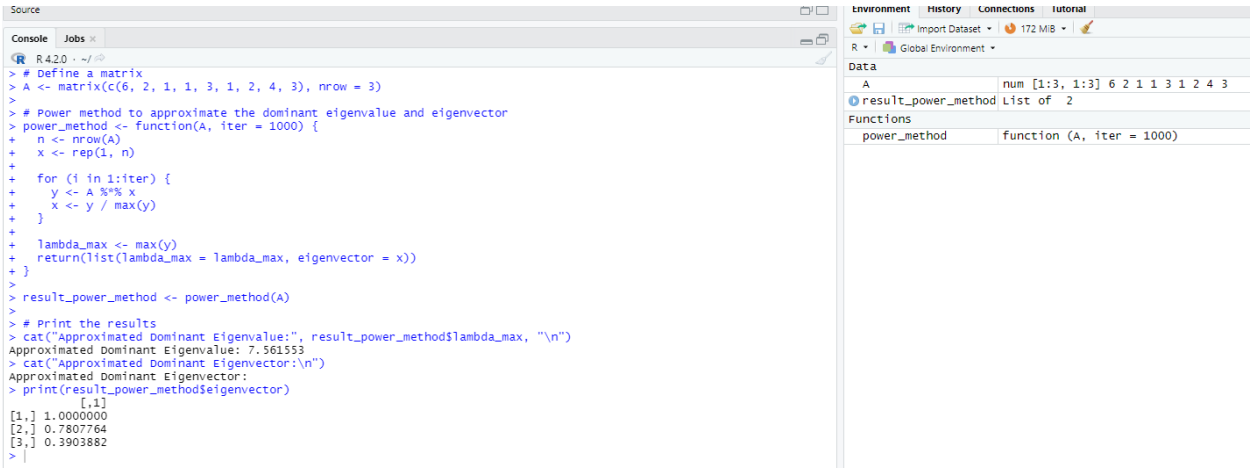
Aim:

To write R program Power method to approximate dominant Eigen value and Eigen vector

Program:

```
# Define a matrix
A <- matrix(c(6, 2, 1, 1, 3, 1, 2, 4, 3), nrow = 3)

# Power method to approximate the dominant eigenvalue and eigenvector
power_method <- function(A, iter = 1000) {
  n <- nrow(A)
  x <- rep(1, n)

  for (i in 1:iter) {
    y <- A %*% x
    x <- y / max(y)
  }

  lambda_max <- max(y)
  return(list(lambda_max = lambda_max, eigenvector = x))
}

result_power_method <- power_method(A)

# Print the results
cat("Approximated Dominant Eigenvalue:", result_power_method$lambda_max, "\n")
cat("Approximated Dominant Eigenvector:\n")
print(result_power_method$eigenvector)
```

Output:



Result:

Thus the required output is obtained.

Numerical Interpolations Using R

11. Lagrange Interpolation

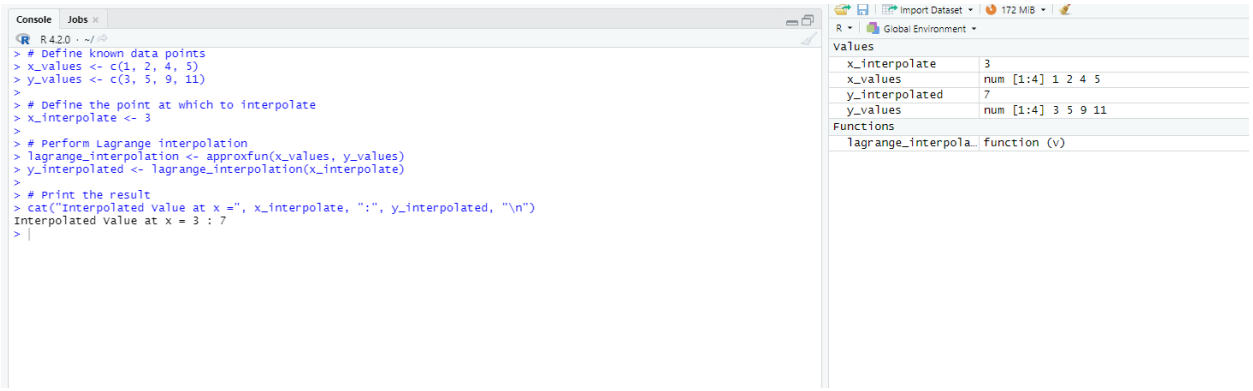
Aim:

To write R program Lagrange Interpolation

Program:

```
# Define known data points
x_values <- c(1, 2, 4, 5)
y_values <- c(3, 5, 9, 11)

# Define the point at which to interpolate
x_interpolate <- 3

# Perform Lagrange interpolation
lagrange_interpolation <- approxfun(x_values, y_values)
y_interpolated <- lagrange_interpolation(x_interpolate)

# Print the result
cat("Interpolated Value at x =", x_interpolate, ":", y_interpolated, "\n")
```

Output:



Result:

Thus the required output is obtained.

12. Newton's forward and Backward Interpolation

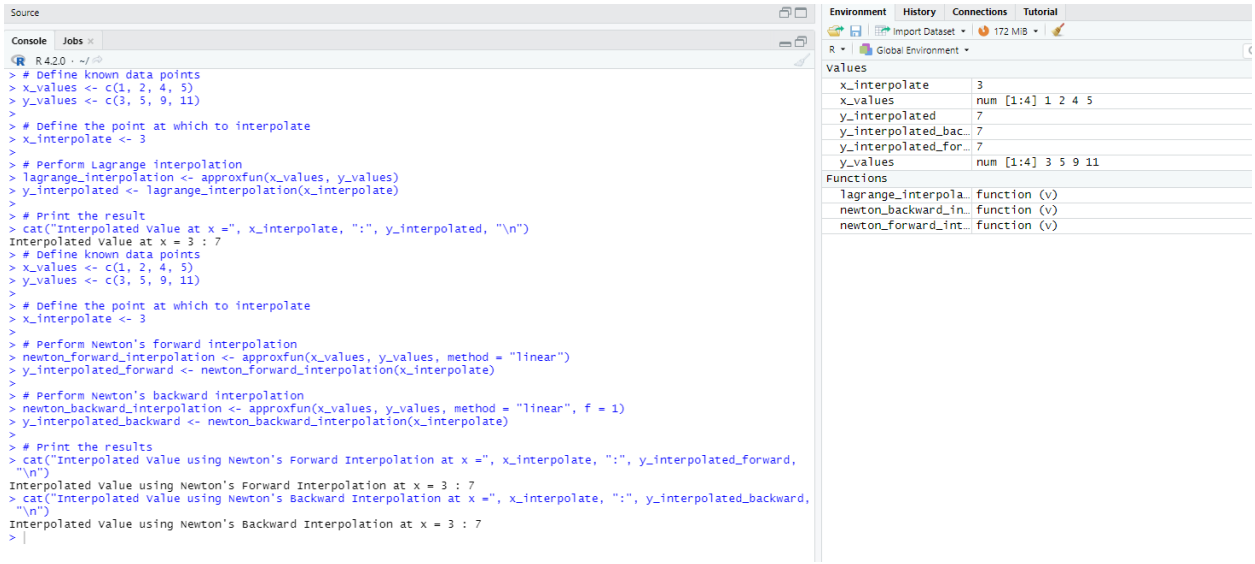
Aim:

To write R program Newton's forward and Backward Interpolation

Program:

```
# Define known data points
x_values <- c(1, 2, 4, 5)
y_values <- c(3, 5, 9, 11)

# Define the point at which to interpolate
x_interpolate <- 3

# Perform Newton's forward interpolation
newton_forward_interpolation <- approxfun(x_values, y_values, method = "linear")
y_interpolated_forward <- newton_forward_interpolation(x_interpolate)

# Perform Newton's backward interpolation
newton_backward_interpolation <- approxfun(x_values, y_values, method = "linear", f = 1)
y_interpolated_backward <- newton_backward_interpolation(x_interpolate)

# Print the results
cat("Interpolated Value using Newton's Forward Interpolation at x =", x_interpolate, ":",
y_interpolated_forward, "\n")
cat("Interpolated Value using Newton's Backward Interpolation at x =", x_interpolate, ":",
y_interpolated_backward, "\n")
```

Output:



Result:

Thus the required output is obtained.

Numerical integration using R

13. Numerical integration using Trapezoidal and Simpson's 1/3rd and 3/8th rules

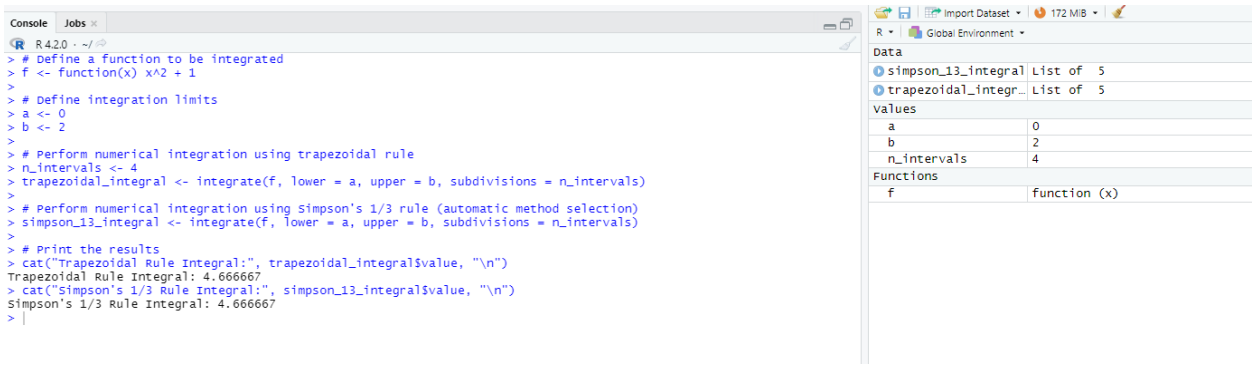
Aim:

To write R program Numerical integration using Trapezoidal and Simpson's 1/3rd and 3/8th rules

Program:

```
# Define a function to be integrated
f <- function(x) x^2 + 1

# Define integration limits
a <- 0
b <- 2

# Perform numerical integration using trapezoidal rule
n_intervals <- 4
trapezoidal_integral <- integrate(f, lower = a, upper = b, subdivisions = n_intervals)

# Perform numerical integration using Simpson's 1/3 rule (automatic method selection)
simpson_13_integral <- integrate(f, lower = a, upper = b, subdivisions = n_intervals)

# Print the results
cat("Trapezoidal Rule Integral:", trapezoidal_integral$value, "\n")
cat("Simpson's 1/3 Rule Integral:", simpson_13_integral$value, "\n")
```

Output:



Result:

Thus the required output is obtained.

Solution of Ordinary differential equations using R

14. Euler's method, Euler's modified method, Runge-Kutta methods

Aim:

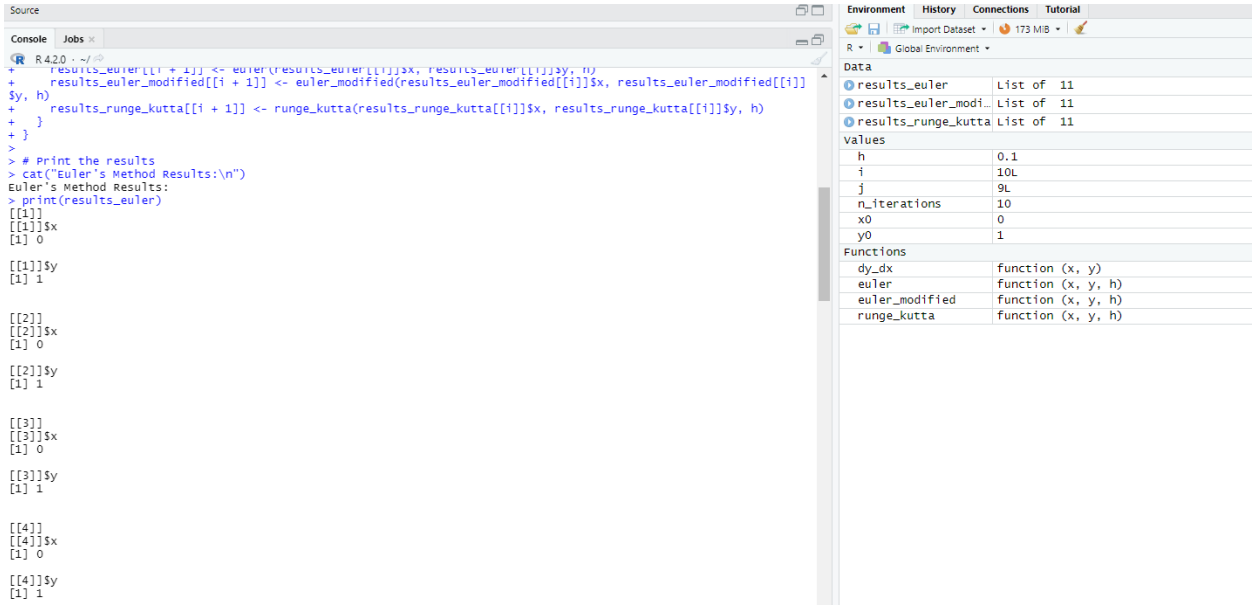
To write R program Euler's method, Euler's modified method, Runge-Kutta methods

Program:

```
# Define a differential equation
dy_dx <- function(x, y) -2 * x * y

# Define initial values
x0 <- 0
y0 <- 1
h <- 0.1  # Step size

# Euler's method
euler <- function(x, y, h) {
  y_new <- y + h * dy_dx(x, y)
  return(list(x_new = x + h, y_new = y_new))
}

# Euler's modified method
euler_modified <- function(x, y, h) {
  y_prime <- y + h * dy_dx(x, y)
  y_new <- y + 0.5 * h * (dy_dx(x, y) + dy_dx(x + h, y_prime))
  return(list(x_new = x + h, y_new = y_new))
}

# Runge-Kutta method (4th order)
runge_kutta <- function(x, y, h) {
  k1 <- h * dy_dx(x, y)
  k2 <- h * dy_dx(x + 0.5 * h, y + 0.5 * k1)
  k3 <- h * dy_dx(x + 0.5 * h, y + 0.5 * k2)
  k4 <- h * dy_dx(x + h, y + k3)
  y_new <- y + (1/6) * (k1 + 2 * k2 + 2 * k3 + k4)
  return(list(x_new = x + h, y_new = y_new))
}

# Perform iterations using each method
n_iterations <- 10
results_euler <- list()
results_euler_modified <- list()
results_runge_kutta <- list()
```

```
for (i in 1:n_iterations) {
  results_euler[[i]] <- list(x = x0, y = y0)
  results_euler_modified[[i]] <- list(x = x0, y = y0)
  results_runge_kutta[[i]] <- list(x = x0, y = y0)

  for (j in 1:(n_iterations - 1)) {
    results_euler[[i + 1]] <- euler(results_euler[[i]]$x, results_euler[[i]]$y, h)
    results_euler_modified[[i + 1]] <- euler_modified(results_euler_modified[[i]]$x,
results_euler_modified[[i]]$y, h)
    results_runge_kutta[[i + 1]] <- runge_kutta(results_runge_kutta[[i]]$x, results_runge_kutta[[i]]$y, h)
  }
}

# Print the results
cat("Euler's Method Results:\n")
print(results_euler)

cat("Euler's Modified Method Results:\n")
print(results_euler_modified)

cat("Runge-Kutta Method Results:\n")
print(results_runge_kutta)
```

Output:



Result:

Thus the required output is obtained.