R Programming LAB

Contents

[Data Management 2](#_Toc149990171)

[1. Working with vectors and Matrices 2](#_Toc149990172)

[2. Sorting, Merging and Aggregating Data sets 3](#_Toc149990173)

[Testing Statistical Hypothesis using R 4](#_Toc149990174)

[Test for Single, difference of mean and paired mean 4](#_Toc149990175)

[Test for equality of variance 4](#_Toc149990176)

[Applications: Chi-Square test for Goodness of fit and independence of Attributes 4](#_Toc149990177)

[Applications: One way ANOVA and two way ANOVA 5](#_Toc149990178)

[Applications: Latin Square Design 5](#_Toc149990179)

[Numerical Solution of Equations using R 6](#_Toc149990180)

[Newton-Raphson method 6](#_Toc149990181)

[Solving system of Linear Equations (Gauss elimination, Gauss Jacobi and Gauss-Seidel) 6](#_Toc149990182)

[Power method to approximate dominant Eigen value and Eigen vector 6](#_Toc149990183)

[Numerical Interpolations Using R 7](#_Toc149990184)

[Lagrange Interpolation 7](#_Toc149990185)

[Newton's forward and Backward Interpolation 7](#_Toc149990186)

[Numerical integration using R 8](#_Toc149990187)

[Numerical integration using Trapezoidal and Simpson's 1/3rd and 3/8th rules 8](#_Toc149990188)

[Solution of Ordinary differential equations using R 8](#_Toc149990189)

[Euler's method, Euler's modified method, Runge-Kutta methods 8](#_Toc149990190)

# Data Management

## 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]

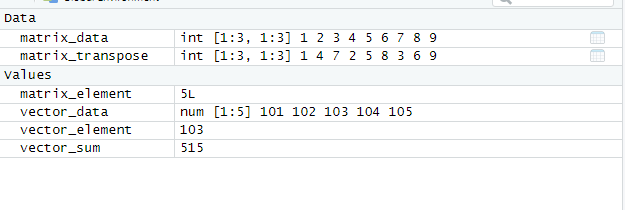
matrix\_element <- matrix\_data[2, 2]

# Performing operations on vectors and matrices

vector\_sum <- sum(vector\_data)

matrix\_transpose <- t(matrix\_data)

**Output:**

****

**Result:**

Thus the required output is obtained.

## 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)

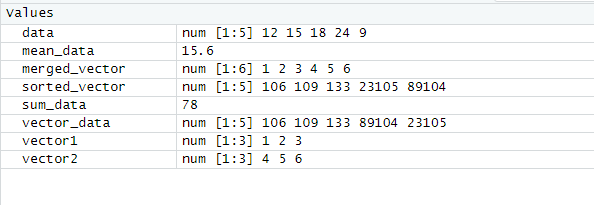
# aggregating data with mean and sum

data <- c(12, 15, 18, 24, 9)

mean\_data <- mean(data)

sum\_data <- sum(data)

**Output:**

****

**Result:**

Thus the required output is obtained.

# Testing Statistical Hypothesis using R

## 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

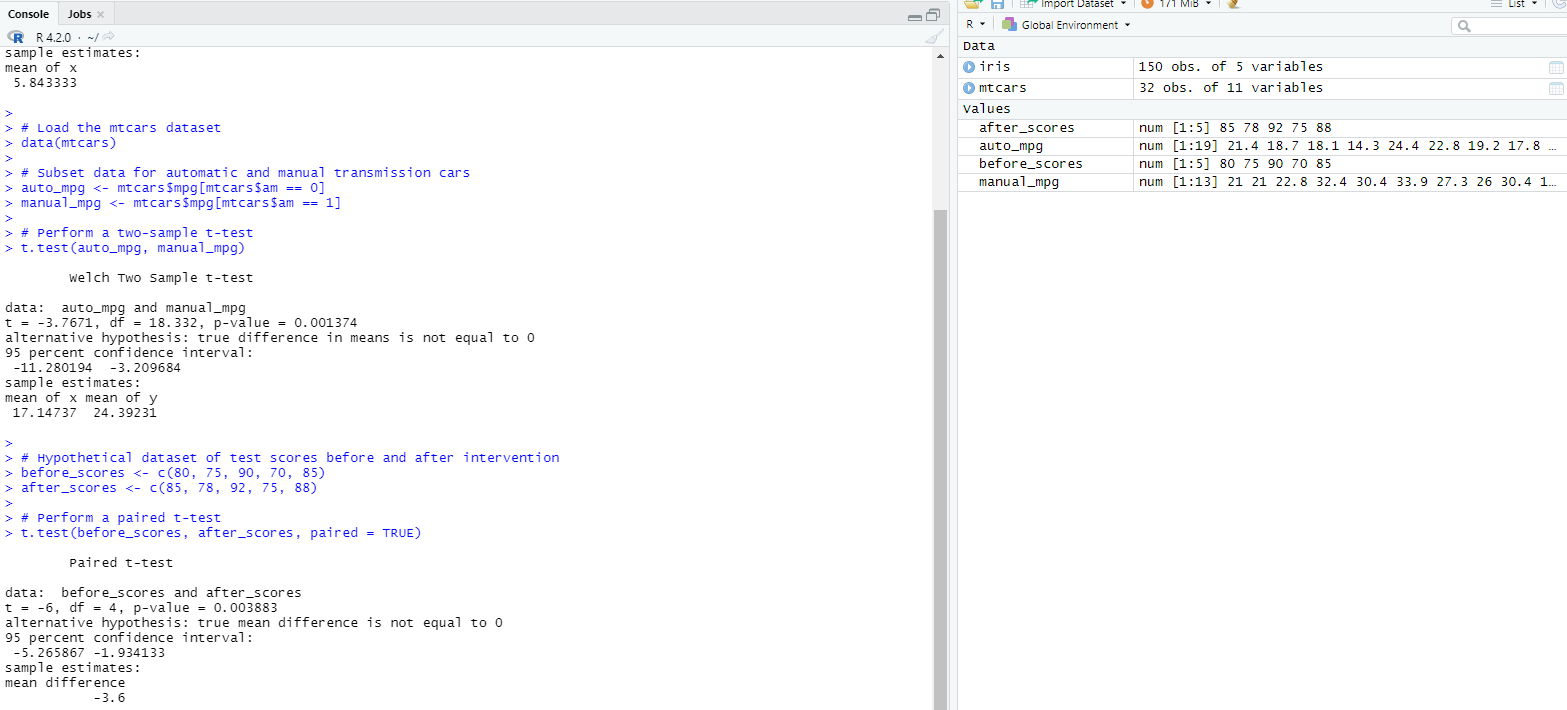
before\_scores <- c(80, 75, 90, 70, 85)

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.

## 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)

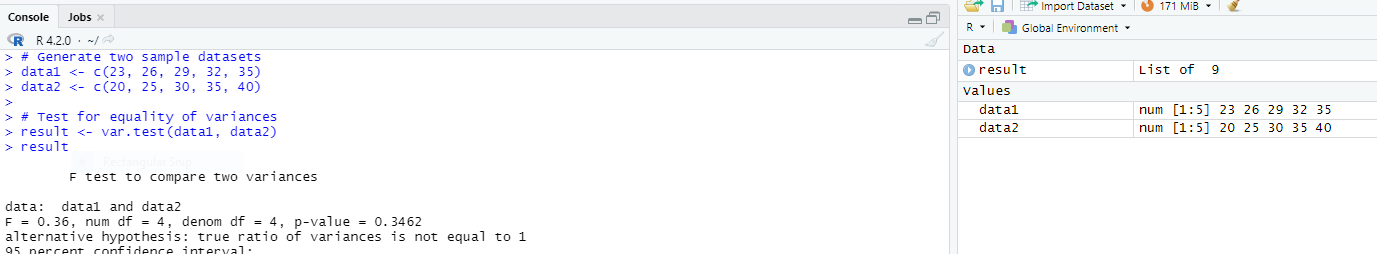
data2 <- c(20, 25, 30, 35, 40)

# Test for equality of variances

result <- var.test(data1, data2)

result

**Output:**

****

**Result:**

Thus the required output is obtained.

## 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

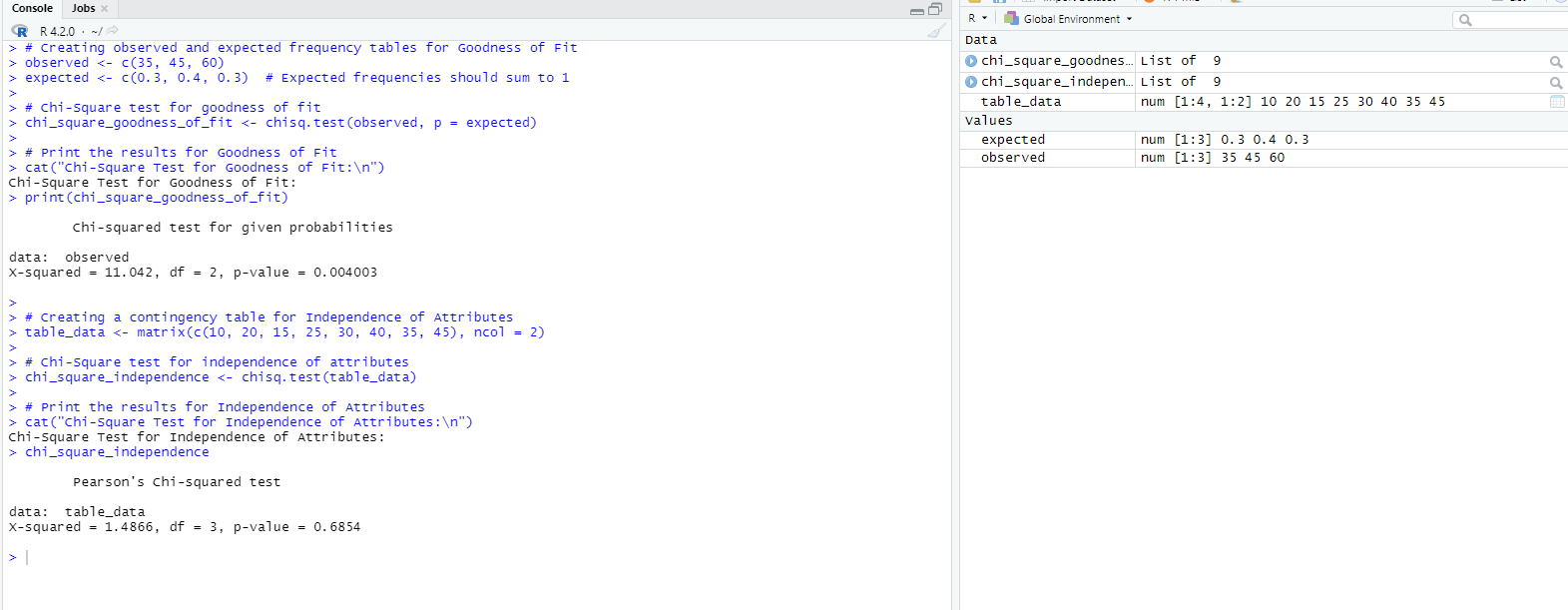
chi\_square\_independence <- chisq.test(table\_data)

# 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.

## 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),

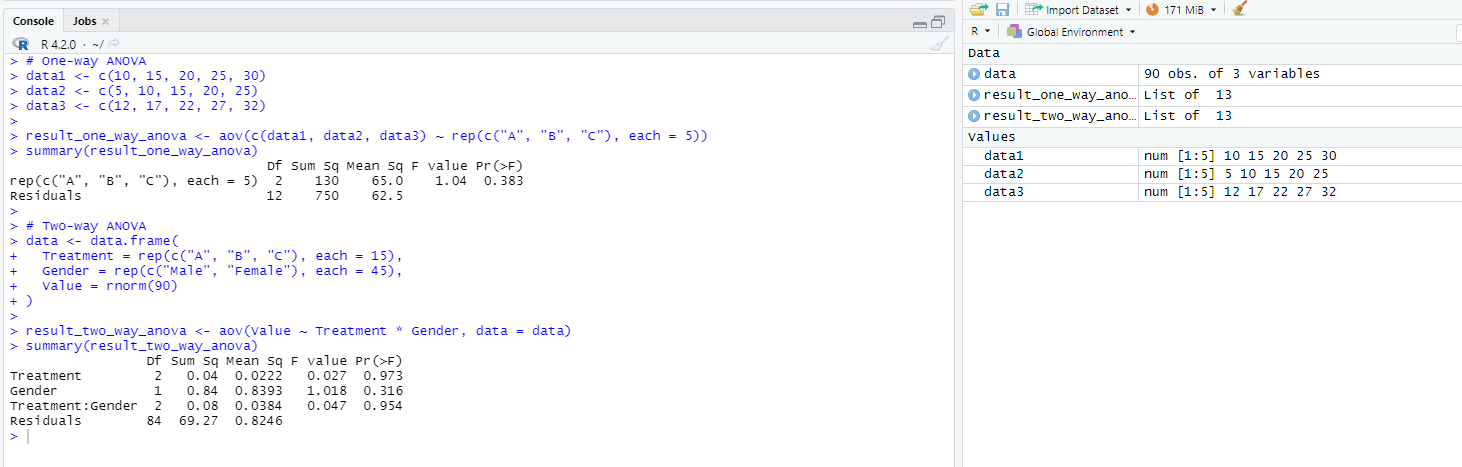
Value = rnorm(90)

)

result\_two\_way\_anova <- aov(Value ~ Treatment \* Gender, data = data)

summary(result\_two\_way\_anova)

**Output:**

****

**Result:**

Thus the required output is obtained.

## 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

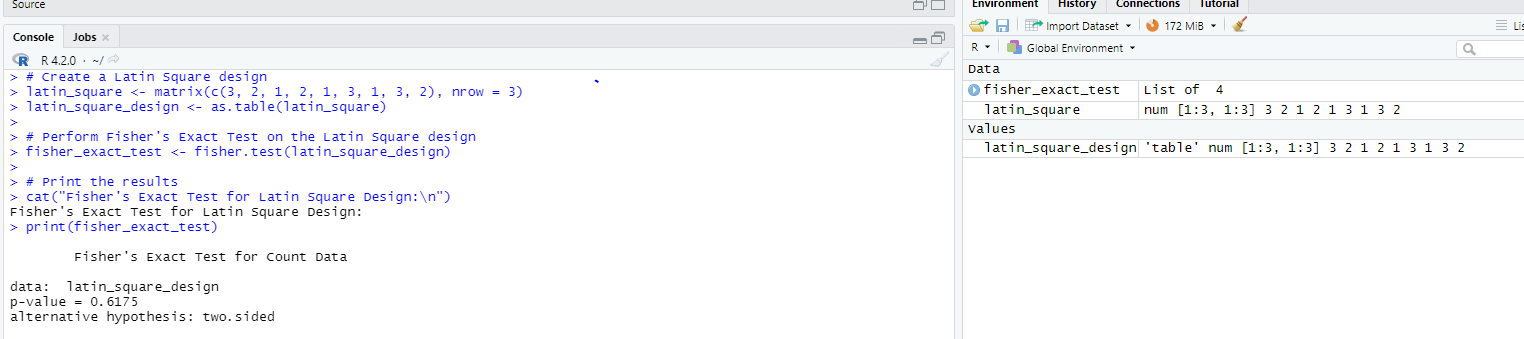
fisher\_exact\_test <- fisher.test(latin\_square\_design)

# 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

## 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

}

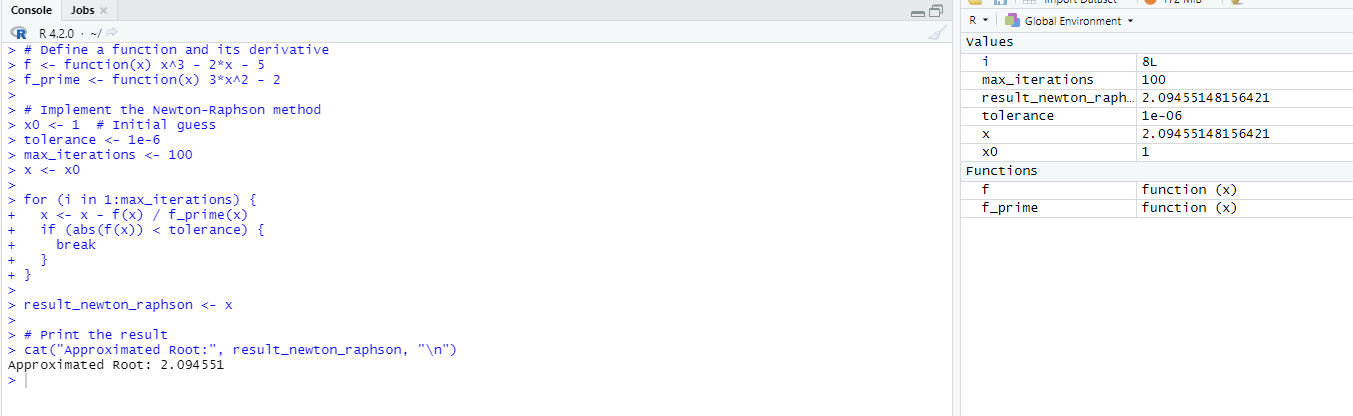
}

result\_newton\_raphson <- x

# Print the result

cat("Approximated Root:", result\_newton\_raphson, "\n")

**Output:**

****

**Result:**

Thus the required output is obtained.

## 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)

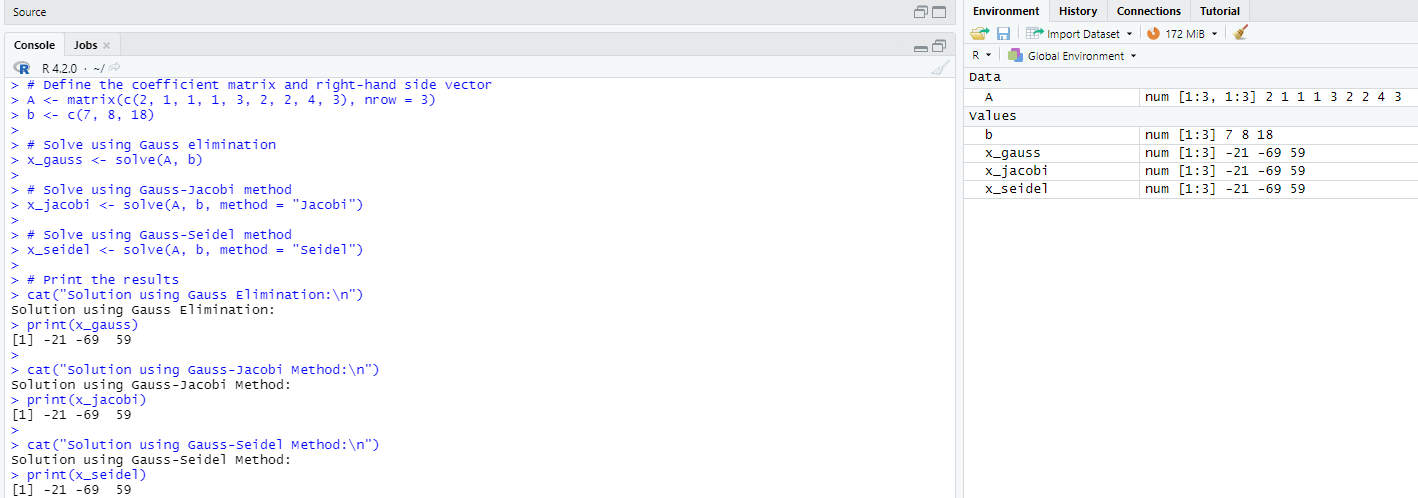
cat("Solution using Gauss-Jacobi Method:\n")

print(x\_jacobi)

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

print(x\_seidel)

**Output:**

****

**Result:**

Thus the required output is obtained.

## 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)

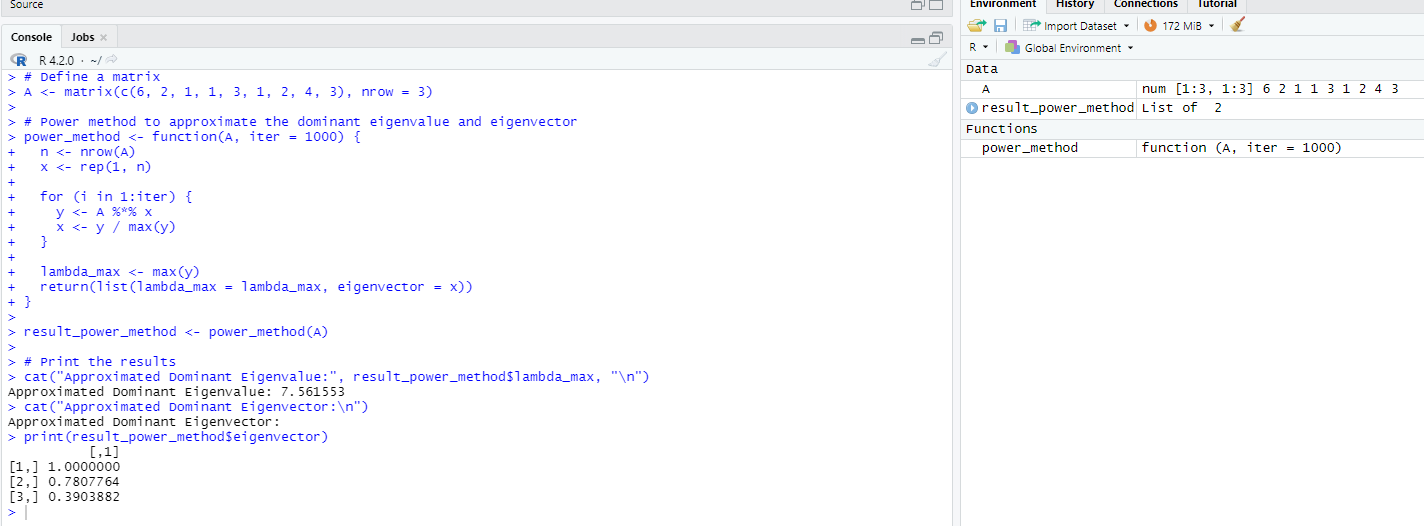
# Print the results

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

## 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

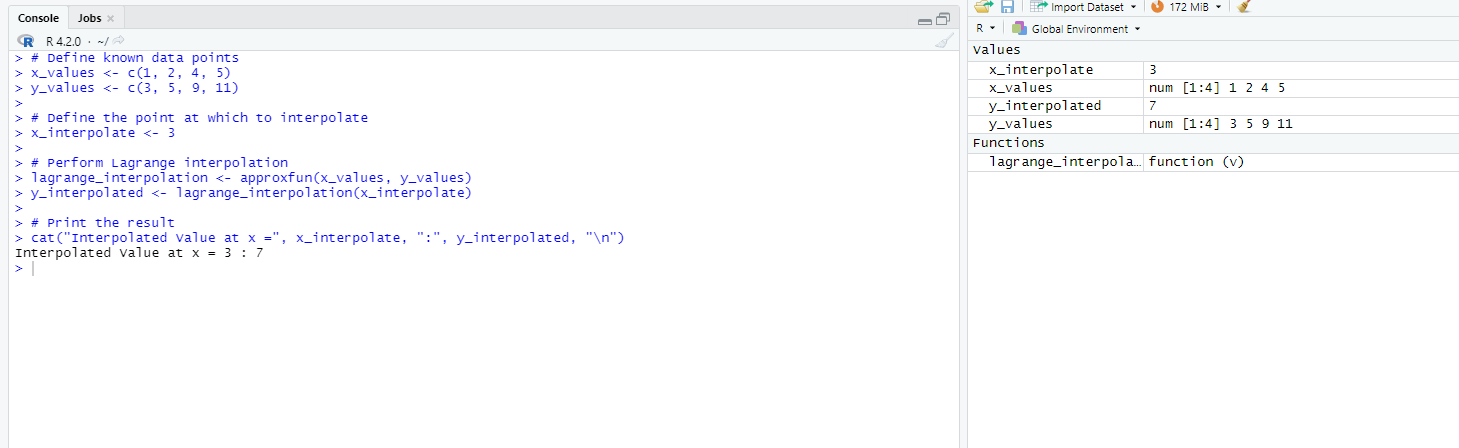
lagrange\_interpolation <- approxfun(x\_values, y\_values)

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.

## 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)

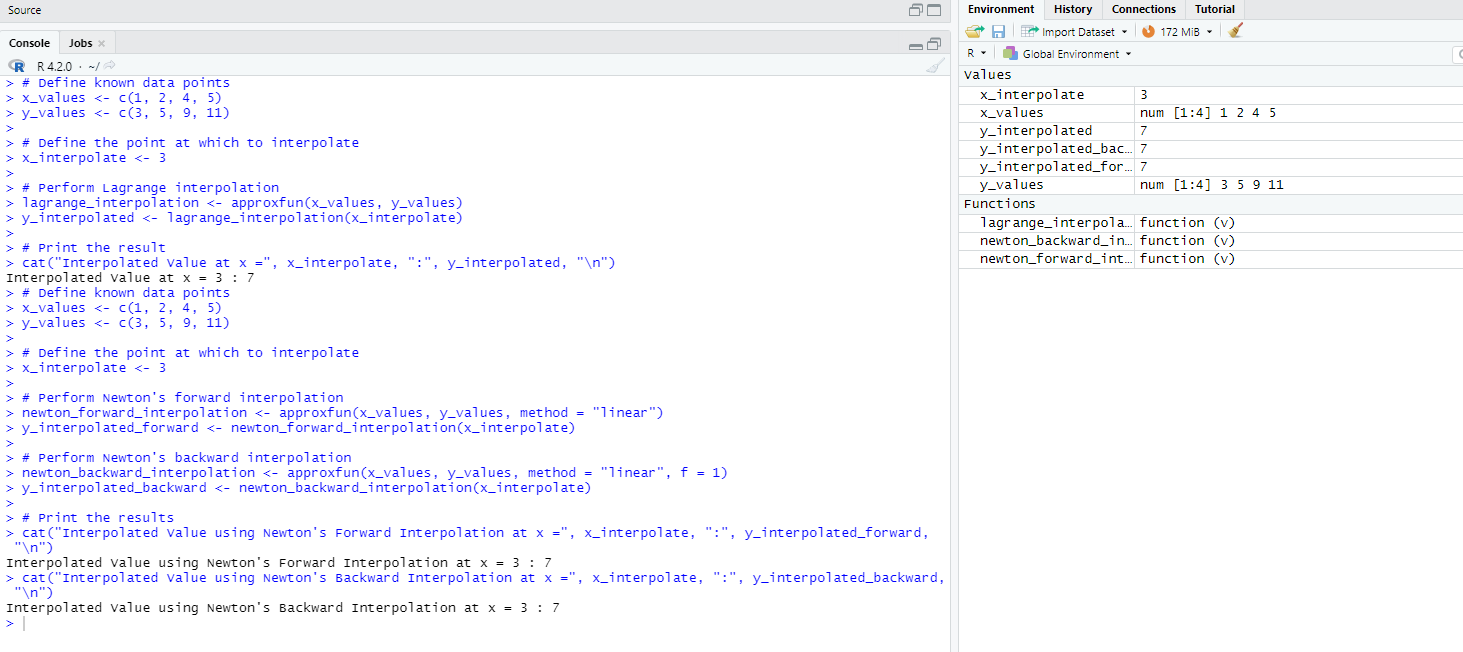
y\_interpolated\_backward <- newton\_backward\_interpolation(x\_interpolate)

# 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

## 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)

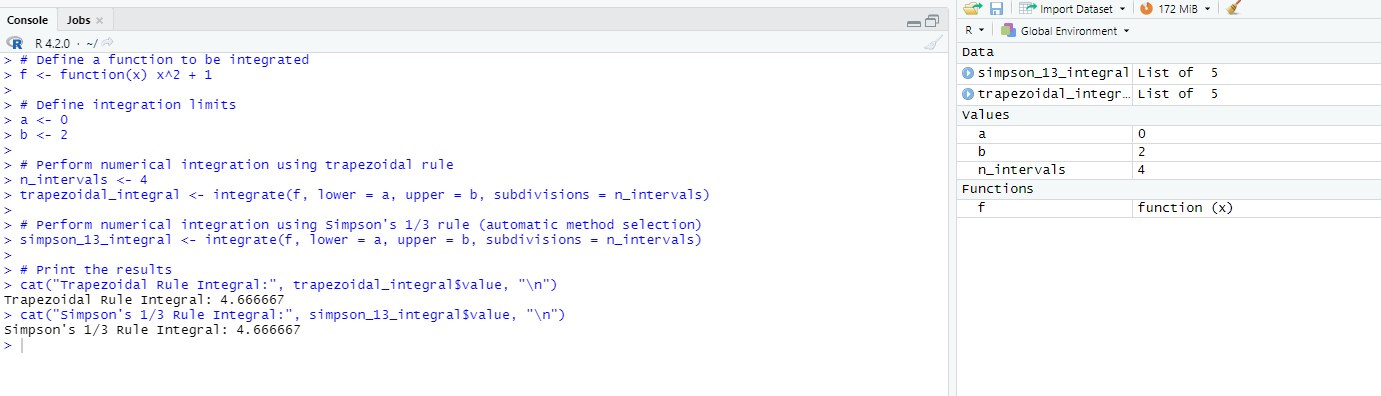
simpson\_13\_integral <- integrate(f, lower = a, upper = b, subdivisions = n\_intervals)

# 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

## 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)

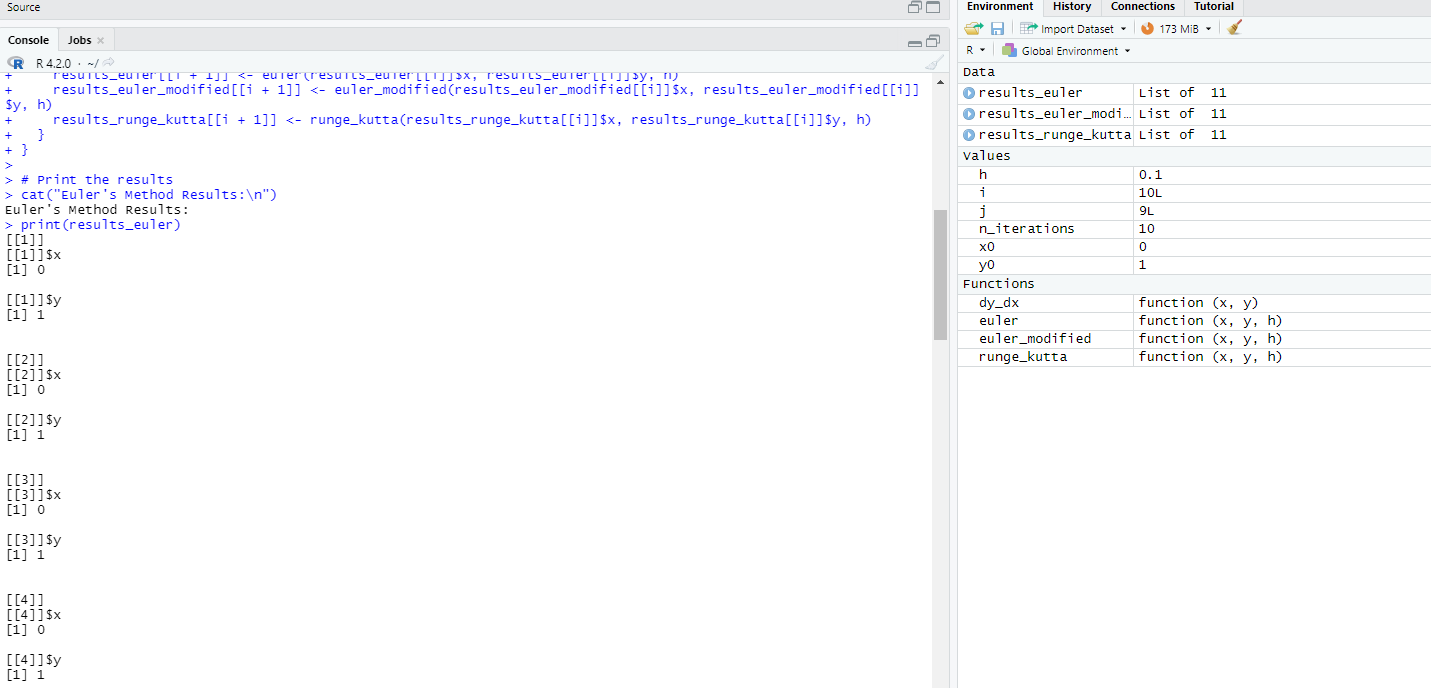
cat("Euler's Modified Method Results:\n")

print(results\_euler\_modified)

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

print(results\_runge\_kutta)

**Output:**

****

**Result:**

Thus the required output is obtained.