

R Programming LAB

Contents

Data Management	2
1. Working with vectors and Matrices.....	2
2. Sorting, Merging and Aggregating Data sets	3
Testing Statistical Hypothesis using R.....	4
3. Test for Single, difference of mean and paired mean	4
4. Test for equality of variance	5
5. Applications: Chi-Square test for Goodness of fit and independence of Attributes	7
6. Applications: One way ANOVA and two way ANOVA.....	8
7. Applications: Latin Square Design.....	10
Numerical Solution of Equations using R.....	11
8. Newton-Raphson method.....	11
9. Solving system of Linear Equations (Gauss elimination, Gauss Jacobi and Gauss-Seidel)	12
10. Power method to approximate dominant Eigen value and Eigen vector.....	13
Numerical Interpolations Using R	14
11. LagRange Interpolation.....	14
12. Newton's forward and Backward Interpolation	15
Numerical integration using R	16
13. Numerical integration using Trapezoidal and Simpson's 1/3rd and 3/8th rules.....	16
Solution of Ordinary differential equations using R	17
14. Euler's method, Euler's modified method, Runge-Kutta methods.....	17

Data Management

1. Working with vectors and Matrices

Aim:

To write R program working with vectors and Matrices

Program:

```
v_data <- c(1, 2, 3, 4, 5)
m_data <- matrix(1:9, nrow = 3)
print(m_data)
v_element <- v_data[3]
cat("Vector Element: ", v_element)
m_element <- m_data[2, 2]
cat("Matrix Element", m_element)
v_sum <- sum(v_data)
cat("Vector Sum: ", v_sum)
m_transpose <- t(m_data)
print(m_transpose)
```

Output:

Data	
m_data	int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
m_transpose	int [1:3, 1:3] 1 4 7 2 5 8 3 6 9
Values	
m_element	5L
v_data	num [1:5] 1 2 3 4 5
v_element	3
v_sum	15

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(8, 2, 6, 4, 5)
sorted_vector <- sort(vector_data)
cat("Sorted Vector: ", sorted_vector)
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
merged_vector <- c(vector1, vector2)
cat("Merged Vector: ", merged_vector)
data <- c(12, 15, 18, 24, 9)
mean_data <- mean(data)
sum_data <- sum(data)
cat("Mean Data: ", mean_data)
cat("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] 2 4 5 6 8
sum_data	78
vector_data	num [1:5] 8 2 6 4 5
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

Program:

```
data1 <- c(23, 26, 29, 32, 35)
data2 <- c(20, 25, 30, 35, 40)
result <- var.test(data1, data2)
print(result)
```

Output:

```
      F test to compare two variances

data:  data1 and data2
F = 0.36, num df = 4, denom df = 4, p-value = 0.3462
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.03748231 3.45763076
sample estimates:
ratio of variances
              0.36
```

Result:

Thus the required output is obtained.

4. Test for equality of variance

Aim:

To write R program

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:

One Way Test –

```
one sample t-test

data: iris$Sepal.Length
t = 12.473, df = 149, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
 5.709732 5.976934
sample estimates:
mean of x
 5.843333
```

Two Way Test –

```
welch Two Sample t-test

data: auto_mpg and manual_mpg
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-11.280194 -3.209684
sample estimates:
mean of x mean of y
 17.14737  24.39231
```

Paired Test –

```
Paired t-test

data: before_scores and after_scores
t = -6, df = 4, p-value = 0.003883
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
-5.265867 -1.934133
sample estimates:
mean difference
      -3.6
```

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

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)
cat("Chi-Square Test for Goodness of Fit:\n")
print(chi_square_goodness_of_fit)
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")
print(chi_square_independence)
```

Output:

```
Chi-Square Test for Goodness of Fit:
> print(chi_square_goodness_of_fit)

      Chi-squared test for given probabilities

data:  observed
X-squared = 11.042, df = 2, p-value = 0.004003

      Pearson's Chi-squared test

data:  table_data
X-squared = 1.4866, df = 3, p-value = 0.6854
```

Result:

Thus the required output is obtained.

6. Applications: One way ANOVA and two way ANOVA

Aim:

To write R program

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:

```
rep(c("A", "B", "C"), each = 5)    Df Sum Sq Mean Sq F value Pr(>F)
Residuals                          12   750    62.5
>
> # 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)
```

	Df	Sum Sq	Mean Sq	F	value	Pr(>F)
Treatment	2	2.64	1.3192	1.226	0.299	
Gender	1	0.40	0.4029	0.374	0.542	
Treatment:Gender	2	1.88	0.9379	0.872	0.422	
Residuals	84	90.37	1.0758			

```
> |
```



```

rep(c("A", "B", "C"), each = 5)  Df Sum Sq Mean Sq F value Pr(>F)
Residuals                        12   750    62.5
>
> # 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)

```

	Df	Sum Sq	Mean Sq	F	value	Pr(>F)
Treatment	2	2.64	1.3192	1.226	0.299	
Gender	1	0.40	0.4029	0.374	0.542	
Treatment:Gender	2	1.88	0.9379	0.872	0.422	
Residuals	84	90.37	1.0758			

```

> |

```

Result:

Thus the required output is obtained.

7. Applications: Latin Square Design

Aim:

To write R program

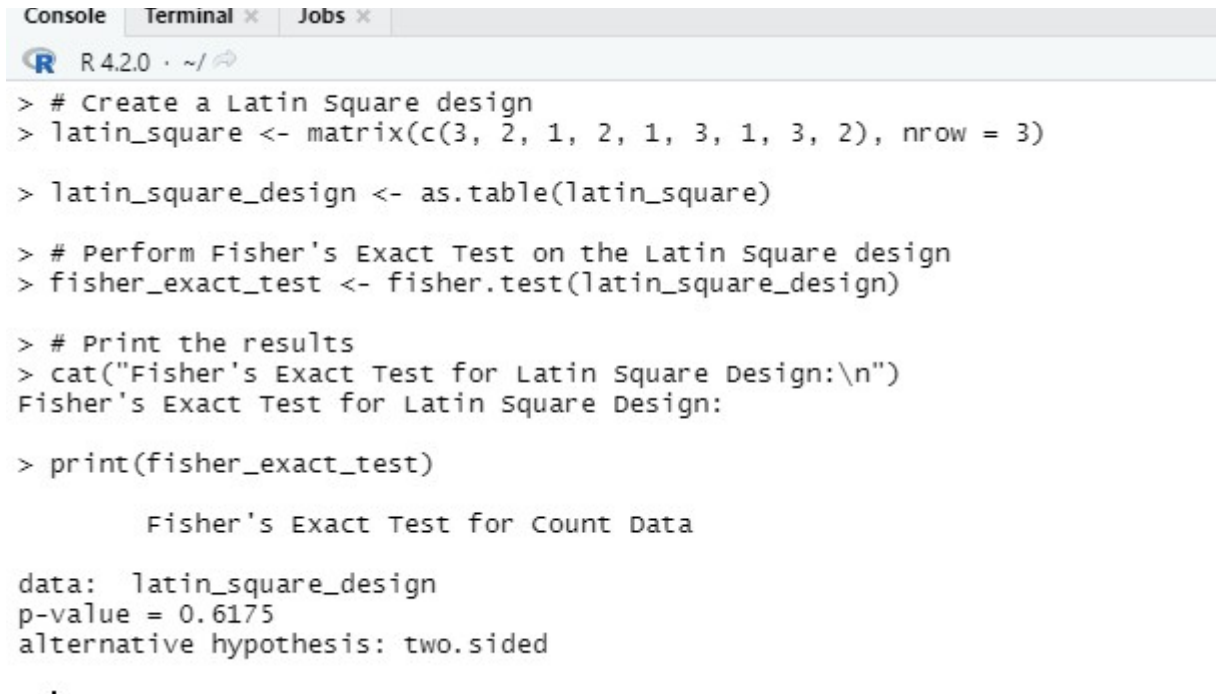
Program:

```
# 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:



The screenshot shows an R console window with the following output:

```
R 4.2.0 ~/  
> # 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")  
Fisher's Exact Test for Latin Square Design:  
  
> print(fisher_exact_test)  
  
      Fisher's Exact Test for Count Data  
  
data:  latin_square_design  
p-value = 0.6175  
alternative hypothesis: two.sided  
.  
.
```

Result:

Thus the required output is obtained.

Numerical Solution of Equations using R

8. Newton-Raphson method

Aim:

To write R program

Program:

```
f <- function(x) x^3 - 2*x - 5
f_prime <- function(x) 3*x^2 - 2
x0 <- 1
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
cat("Approximated Root:", result_newton_raphson, "\n")
```

Output:

R Global Environment	
Values	
i	8L
max_iterations	100
result_newton_rap...	2.09455148156421
tolerance	1e-06
x	2.09455148156421
x0	1
Functions	
f	function (x)
f_prime	function (x)

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

Program:

```
A <- matrix(c(2, 1, 1, 1, 3, 2, 2, 4, 3), nrow = 3)
b <- c(7, 8, 18)
x_gauss <- solve(A, b)
x_jacobi <- solve(A, b, method = "Jacobi")
x_seidel <- solve(A, b, method = "Seidel")
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:

Data	
A	num [1:3, 1:3] 2 1 1 1 3 2 2 4 3
values	
b	num [1:3] 7 8 18
x_gauss	num [1:3] -21 -69 59
x_jacobi	num [1:3] -21 -69 59
x_seidel	num [1:3] -21 -69 59

Result:

Thus the required output is obtained.

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

Aim:

To write R program

Program:

```
A <- matrix(c(6, 2, 1, 1, 3, 1, 2, 4, 3), nrow = 3)
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)
cat("Approximated Dominant Eigenvalue:", result_power_method$lambda_max, "\n")
cat("Approximated Dominant Eigenvector:\n")
print(result_power_method$eigenvector)
```

Output:

Data	
A	num [1:3, 1:3] 6 2 1 1 3 1 2 4 3
▶ result_power_meth...	List of 2
Functions	
power_method	function (A, iter = 1000)

Result:

Thus the required output is obtained.

Numerical Interpolations Using R

11. LagRange Interpolation

Aim:

To write R program

Program:

```
x_values <- c(1, 2, 4, 5)
y_values <- c(3, 5, 9, 11)
x_interpolate <- 3
lagrange_interpolation <- approxfun(x_values, y_values)
y_interpolated <- lagrange_interpolation(x_interpolate)
cat("Interpolated Value at x =", x_interpolate, ":", y_interpolated, "\n")
```

Output:

```
> # 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")
Interpolated value at x = 3 : 7
> |
```

Result:

Thus the required output is obtained.

12. Newton's forward and Backward Interpolation

Aim:

To write R program

Program:

```
x_values <- c(1, 2, 4, 5)
y_values <- c(3, 5, 9, 11)

x_interpolate <- 3

newton_forward_interpolation <- approxfun(x_values, y_values, method = "linear")
y_interpolated_forward <- newton_forward_interpolation(x_interpolate)

newton_backward_interpolation <- approxfun(x_values, y_values, method = "linear",
f = 1)
y_interpolated_backward <- newton_backward_interpolation(x_interpolate)

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:

values	
x_interpolate	3
x_values	num [1:4] 1 2 4 5
y_interpolated_ba...	7
y_interpolated_fo...	7
y_values	num [1:4] 3 5 9 11
Functions	
newton_backward_i...	function (v)
newton_forward_in...	function (v)

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

Program:

```
f <- function(x) x^2 + 1
a <- 0
b <- 2
n_intervals <- 4
trapezoidal_integral <- integrate(f, lower = a, upper = b, subdivisions =
n_intervals)
simpson_13_integral <- integrate(f, lower = a, upper = b, subdivisions =
n_intervals)
cat("Trapezoidal Rule Integral:", trapezoidal_integral$value, "\n")
cat("Simpson's 1/3 Rule Integral:", simpson_13_integral$value, "\n")
```

Output:

Data	
▶ simpson_13_integr...	List of 5
▶ trapezoidal_integ...	List of 5
values	
a	0
b	2
n_intervals	4
Functions	
f	function (x)

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

Program:

```
dy_dx <- function(x, y) -2 * x * y
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)
}
}

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:

Data	
▶ results_euler	List of 11
▶ results_euler_mod...	List of 11
▶ results_runge_kut...	List of 11
values	
h	0.1
i	10L
j	9L
n_iterations	10
x0	0
y0	1
Functions	
dy_dx	function (x, y)
euler	function (x, y, h)
euler_modified	function (x, y, h)
runge_kutta	function (x, y, h)

Result:

Thus the required output is obtained.