R

Probability Distributions

These functions are used to compute density (`d`), distribution (`p`), quantile (`q`), and to generate random numbers (`r`) for various probability distributions.

Normal Distribution

- `dnorm(x, mean = 0, sd = 1)`
  - Computes the **density** (height of the probability density function) at point `x`.
- `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE)`
  - Computes the **cumulative distribution function** up to point `q`.
- `qnorm(p, mean = 0, sd = 1, lower.tail = TRUE)`
  - Computes the **quantile function** (inverse of `pnorm`) for probability `p`.
- `rnorm(n, mean = 0, sd = 1)`
  - Generates `n` **random numbers** from the normal distribution.

Binomial Distribution

- `dbinom(x, size, prob)`
  - Computes the **probability mass function** at point `x`.
- `pbinom(q, size, prob, lower.tail = TRUE)`
  - Computes the **cumulative distribution function** up to point `q`.
- `qbinom(p, size, prob, lower.tail = TRUE)`
  - Computes the **quantile function** for probability `p`.
- `rbinom(n, size, prob)`
  - Generates `n` **random numbers** from the binomial distribution.

Poisson Distribution

- `dpois(x, lambda)`
  - Computes the **probability mass function** at point `x`.
- `ppois(q, lambda, lower.tail = TRUE)`
  - Computes the **cumulative distribution function** up to point `q`.
- `qpois(p, lambda, lower.tail = TRUE)`
  - Computes the **quantile function** for probability `p`.
- `rpois(n, lambda)`
  - Generates `n` **random numbers** from the Poisson distribution.

Chi-Square Distribution

- `dchisq(x, df)`
  - Computes the **density** at point `x`.
- `pchisq(q, df, lower.tail = TRUE)`
  - Computes the **cumulative distribution function** up to point `q`.

- `qchisq(p, df, lower.tail = TRUE)`
  - Computes the **quantile function** for probability `p`.
- `rchisq(n, df)`
  - Generates `n` **random numbers** from the chi-square distribution.

Student's t-Distribution

- `dt(x, df)`
  - Computes the **density** at point `x`.
- `pt(q, df, lower.tail = TRUE)`
  - Computes the **cumulative distribution function** up to point `q`.
- `qt(p, df, lower.tail = TRUE)`
  - Computes the **quantile function** for probability `p`.
- `rt(n, df)`
  - Generates `n` **random numbers** from the t-distribution.

Data Manipulation and Computation

Sequence Generation

- `seq(from, to, by)`
  - Generates a sequence of numbers from `from` to `to` in steps of `by`.
- `seq(from, to, length.out)`
  - Generates a sequence from `from` to `to` with a specified number of elements.

Example:

```
seq(0, 1, by = 0.2)          # 0.0 0.2 0.4 0.6 0.8 1.0
seq(0, 1, length.out = 5)    # 0.0 0.25 0.5 0.75 1.0
```

Repetition

- `rep(x, times)`
  - Repeats the value `x`, `times` number of times.
- `rep(x, each)`
  - Repeats each element of `x`, `each` number of times.

Example:

```
rep(1:3, times = 2)          # 1 2 3 1 2 3
rep(1:3, each = 2)           # 1 1 2 2 3 3
```

Apply Functions

- `apply(X, MARGIN, FUN, ...)`
  - Applies a function `FUN` over the margins of an array `X`.
  - `MARGIN` = 1 applies over rows, `MARGIN` = 2 applies over columns.

Example:

```
mat <- matrix(1:9, nrow = 3)
apply(mat, 1, sum)           # Sum over rows
apply(mat, 2, mean)          # Mean over columns
```

- `lapply(X, FUN, ...)`

- Applies function `FUN` to each element of a list `X`.

- `sapply(X, FUN, ...)`
  - Same as `lapply` but tries to simplify the result.

Example:

```
lst <- list(a = 1:5, b = 6:10)
lapply(lst, sum)            # Returns a list
sapply(lst, sum)            # Returns a vector
```

Combining Functions

- `c(...)`
  - Combines values into a vector.

Example:

```
c(1, 2, 3, 4)               # 1 2 3 4
```

- `outer(X, Y, FUN)`
  - Computes the outer product of two vectors `X` and `Y` using function `FUN`.

Example:

```
x <- 1:3
y <- 4:6
outer(x, y, "*")            # Multiplies each element of x with each element of
y
```

Plotting Functions

Basic Plotting

- `plot(x, y, ...)`
  - Creates a scatter plot of `y` versus `x`.

Example:

```
x <- 1:10
y <- x^2
plot(x, y, main = "Scatter Plot", xlab = "X-axis", ylab = "Y-axis")
```

Histogram

- `hist(x, breaks, probability, ...)`
  - Plots a histogram of `x`.
  - `breaks`: Number or vector of breakpoints.
  - `probability = TRUE`: Plots density instead of frequency.

Example:

```
data <- rnorm(100)
hist(data, breaks = 10, probability = TRUE, main = "Histogram")
```

Curve Plotting

- `curve(expr, from, to, add, ...)`
  - Plots the curve of an expression `expr` over the interval `from` to `to`.

- `add = TRUE`: Adds the curve to an existing plot.

Example:

```
curve(sin, from = 0, to = 2*pi)
```

3D Surface Plot

- `persp(x, y, z, ...)`
  - Creates a 3D perspective plot of the surface defined by `z`.

Example:

```
x <- seq(-10, 10, length = 30)
y <- seq(-10, 10, length = 30)
z <- outer(x, y, function(x, y) sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2))
persp(x, y, z, theta = 30, phi = 30, expand = 0.5)
```

Quantile-Quantile Plot

- `qqnorm(y, ...)`
  - Produces a normal QQ plot of the values in `y`.
- `qqline(y, ...)`
  - Adds a line to a normal QQ plot.

Example:

```
data <- rnorm(100)
qqnorm(data)
qqline(data, col = "red")
```

Adding Reference Lines

- `abline(a, b, h, v, ...)`
  - Adds a straight line to a plot.
  - `a` and `b`: Intercept and slope.
  - `h`: Adds horizontal lines at specified y-values.
  - `v`: Adds vertical lines at specified x-values.

Example:

```
plot(1:10, 1:10)
abline(a = 0, b = 1, col = "blue")  # Line with intercept 0 and slope 1
abline(h = 5, col = "red")          # Horizontal line at y = 5
abline(v = 5, col = "green")        # Vertical line at x = 5
```

Segments and Arrows

- `segments(x0, y0, x1, y1, ...)`
  - Draws line segments between points `(x0, y0)` and `(x1, y1)`.

Example:

```
plot(1:10, 1:10)
segments(2, 2, 8, 8, col = "red", lwd = 2)
```

Statistical Tests and Confidence Intervals

▼ Reading Data

- `read.table(file, header, sep, ...)`
  - Reads a table from a file.
  - `header = TRUE`: The first line contains variable names.
  - `sep`: Specifies the field separator character.

Example:

```
data <- read.table("data.txt", header = TRUE, sep = "\\t")
```

▼ t-Test

- `t.test(x, y, alternative, mu, paired, var.equal, conf.level)`
  - Performs a t-test.
  - `x`, `y`: Vectors of data values.
  - `alternative`: Specifies the alternative hypothesis (`"two.sided"`, `"less"`, `"greater"`).
  - `mu`: True value of the mean (one-sample test) or difference in means (two-sample test).
  - `paired = TRUE`: Performs a paired t-test.
  - `var.equal = TRUE`: Assumes equal variances in two-sample test.
  - `conf.level`: Confidence level for the interval.

Example:

```
# One-sample t-test
t.test(x, mu = 0)

# Two-sample t-test
t.test(x, y, var.equal = TRUE)

# Paired t-test
t.test(x, y, paired = TRUE)
```

▼ Proportion Test

- `prop.test(x, n, p, alternative, correct, conf.level)`
  - Tests for a proportion or difference in proportions.
  - `x`: Number of successes.
  - `n`: Number of trials.
  - `p`: Expected proportion under null hypothesis.
  - `correct = FALSE`: Disables Yates' continuity correction.

Example:

```
# Single proportion test
prop.test(x = 50, n = 100, p = 0.5)

# Two-sample proportion test
prop.test(x = c(50, 60), n = c(100, 120))
```

▼ Chi-Square Test

- `chisq.test(x, y, correct)`

- Performs chi-square test for independence or goodness-of-fit.
- `x`: For goodness-of-fit, a vector of observed counts.
- `y`: For independence, a second vector or matrix.
- `correct = FALSE`: Disables Yates' continuity correction.

Example:

```
# Goodness-of-fit test
observed <- c(50, 30, 20)
expected <- c(40, 40, 20)
chisq.test(x = observed, p = expected / sum(expected))

# Test for independence
matrix <- matrix(c(10, 20, 30, 40), nrow = 2)
chisq.test(matrix)
```

▼ Confidence Intervals

- `confint(object, parm, level)`
  - Computes confidence intervals for model parameters.
  - Often used after fitting a model (e.g., `lm`, `glm`).

Example:

```
model <- lm(y ~ x)
confint(model, level = 0.95)
```

▼ Probability Distribution Functions for Testing

- `pt(q, df, lower.tail = TRUE)`
  - Cumulative distribution function of the t-distribution.
- `qt(p, df, lower.tail = TRUE)`
  - Quantile function of the t-distribution.

Example:

```
# Compute p-value for t-test statistic
t_stat <- 2.5
df <- 20
p_value <- 2 * pt(-abs(t_stat), df)
```

▼ Additional Useful Functions

- `mean(x, trim = 0, na.rm = FALSE)`
  - Computes the mean of `x`.
  - `trim`: Trims a fraction of observations from each end.
  - `na.rm = TRUE`: Removes `NA` values.
- `median(x, na.rm = FALSE)`
  - Computes the median of `x`.
- `var(x, y = NULL, na.rm = FALSE)`
  - Computes the variance of `x`.
- `sd(x, na.rm = FALSE)`

    - Computes the standard deviation of `x`.
- `sum(x, na.rm = FALSE)`
    - Sums up the elements of `x`.
- `length(x)`
    - Returns the number of elements in `x`.
- `dim(x)`
    - Retrieves the dimensions of an object `x`.

▼ Example Workflow

Here's an example that brings together several of these functions:

Task: Perform a two-sample t-test to compare the means of two groups, plot the data, and visualize the distribution of the test statistic.

```
# Generate sample data
set.seed(123)
group1 <- rnorm(30, mean = 5, sd = 1)
group2 <- rnorm(30, mean = 6, sd = 1)

# Perform t-test
t_result <- t.test(group1, group2, var.equal = TRUE)

# View results
print(t_result)

# Extract p-value and confidence interval
p_value <- t_result$p.value
conf_int <- t_result$conf.int

# Plot data
boxplot(group1, group2, names = c("Group 1", "Group 2"),
        main = "Comparison of Two Groups", ylab = "Value")

# Visualize t-distribution
t_stat <- t_result$statistic
df <- t_result$parameter
curve(dt(x, df = df), from = -4, to = 4, col = "blue", lwd = 2,
      ylab = "Density", main = "t-Distribution")
abline(v = t_stat, col = "red", lwd = 2)
```