Statistical Computing

Lab Notes

Table of Contents

Statistical Computing Lab Notes **3**
R Fundamentals and Data Handling . . . 3
Probability Theory and Logic . . . 3
Random Variables and Probability Distributions . . . 10
Z-Scores and Normal Probabilities . . . 12
Sampling Distributions and The Central Limit Theorem . . . 12
Confidence Intervals . . . 13
Hypothesis Testing Framework: Step-by-Step Guide . . . 13
Enumerative Data Analysis . . . 14
Maximum Likelihood Estimation (MLE) . . . 15
Numerical Optimization and Calculus in R . . . 16

Statistical Computing Lab Notes

R Fundamentals and Data Handling

R is a language for expressing statistical algorithms, likelihood-based inference, and simulation studies rather than solely a data analysis tool.

Reading and Creating Data

- **Import CSV:** `data <- read.csv("file.csv")` — loads a file into a data frame.
- **Create a vector:** `x <- c(1, 2, 3, 4, 5)`
- **Create a data frame:** `df <- data.frame(x = c(1,2,3), y = c(4,5,6))`
- **View structure:** `str(df)`, `head(df)`, `summary(df)`
- **Access a column:** `df$column_name` or `df[["column_name"]]`
- **Filter rows:** `df[df$x > 2, ]`

Core R Functions

- **Reproducibility:** Control random number generation to ensure simulations can be accurately duplicated by using `set.seed()`.
- **Vectorized Operations:** Most R functions operate on vectors element-wise. Logical indexing is essential for subsetting data based on conditions (e.g., `vec[vec > 0]`).
- **Missing Data:** Real datasets frequently contain `NA` values. Aggregate functions require explicit handling using the `na.rm = TRUE` argument to avoid errors.
- **Apply Family:** Avoid explicit loops for iterative operations. Apply functions across subsets or groups using functions like `sapply()`, `tapply()`, and `aggregate()`.
- **Numerical Stability:** How quantities are computed matters numerically; an iteratively stable algorithm prevents precision loss when calculating statistics for datasets with extreme values.
- **Numerically Stable Mean:**

$$\text{mean} = \frac{\sum(x_i - x_1) + n \cdot x_1}{n}$$

Probability Theory and Logic

Probability quantifies uncertainty and forms the backbone of statistical simulations and models.

- **Sample Space:** The complete set of all possible outcomes, mathematically denoted as Ω.
- **Mutually Exclusive Events:** Events that cannot occur at the same time, meaning $P(A \cap B) = 0$.
- **Independent Events:** Two events are independent if the occurrence of one does not affect the probability of the other.
- **Independence Verification:**

$$P(A \cap B) = P(A) \cdot P(B)$$

- **General Addition Rule (Union):**

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

- **Conditional Probability:**

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

- **Law of Total Probability:**

$$P(T) = P(T \mid D) \cdot P(D) + P(T \mid D^c) \cdot P(D^c)$$

- **Bayes Theorem:**

$$P(D \mid T) = \frac{P(T \mid D) \cdot P(D)}{P(T)}$$

Probability Calculations in R

In R, probabilities are derived by defining a **sample space** and **events as subsets of that space**, then counting: $P(A) = \frac{|A|}{|\Omega|}$. R has a full suite of set operations for this.

Step 1 — Define the Sample Space and Events as Vectors

```r
# Sample space: all possible outcomes
omega <- 1:6   # rolling a fair die

# Define events as subsets (conditions on omega)
A <- omega[omega %% 2 == 0]   # even numbers: {2, 4, 6}
B <- omega[omega > 4]          # greater than 4: {5, 6}

# Or define explicitly
A <- c(2, 4, 6)
B <- c(5, 6)
```

Step 2 — R Set Operations

Operation	Math	R function	Result (A, B above)
Intersection	$A \cap B$	`intersect(A, B)`	`{6}`
Union	$A \cup B$	`union(A, B)`	`{2, 4, 5, 6}`
Complement	A^c	`setdiff(omega, A)`	`{1, 3, 5}`
Difference	$A \setminus B$	`setdiff(A, B)`	`{2, 4}`
Membership	$x \in A$	`x %in% A`	`TRUE/FALSE`

Step 3 — Calculate Probabilities from Sets

```
omega <- 1:6
A <- c(2, 4, 6)   # even
B <- c(5, 6)       # > 4

# Basic probabilities
P <- function(event, space = omega) length(event) / length(space)

P(A)              # P(A) = 3/6 = 0.5
P(B)              # P(B) = 2/6 = 0.333

# Complement: P(A^c)
P(setdiff(omega, A))          # 0.5

# Intersection (AND): P(A & B)
P(intersect(A, B))            # P({6}) = 1/6 = 0.167

# Union (OR): P(A | B)
P(union(A, B))                # P({2,4,5,6}) = 4/6 = 0.667

# Conditional: P(B | A) = P(A & B) / P(A)
P(intersect(A, B)) / P(A)     # 0.333

# Independence check: P(A & B) == P(A) * P(B)?
isTRUE(all.equal(P(intersect(A, B)), P(A) * P(B)))   # FALSE -> dependent
```

Working from a Data Frame (Real Dataset) When your sample space is rows in a data frame, use logical indexing and `nrow()`.

```
# Example: 200 students, columns: passed (TRUE/FALSE), studied (TRUE/FALSE)
df <- data.frame(
  passed  = c(rep(TRUE, 120), rep(FALSE, 80)),
  studied = c(rep(TRUE, 100), rep(FALSE, 20), rep(TRUE, 30), rep(FALSE, 50))
)

n <- nrow(df)   # size of sample space

# Define events as logical vectors (one TRUE/FALSE per row)
A <- df$passed          # event A: student passed
B <- df$studied         # event B: student studied

# Probabilities
P_A <- mean(A)          # P(passed)        - mean() of logical = proportion
P_B <- mean(B)          # P(studied)

# Intersection: P(A & B) -- passed AND studied
```

```r
P_AB <- mean(A & B)

# Union: P(A | B) -- passed OR studied (or both)
P_AuB <- mean(A | B)

# Complement: P(A^c) - did NOT pass
P_Ac <- mean(!A)

# Conditional: P(A | B) - P(passed | studied)
P_A_given_B <- mean(A[B])   # subset A to rows where B is TRUE, then average

# Independence check
isTRUE(all.equal(P_AB, P_A * P_B))
```

Key insight: for logical vectors, `mean()` gives proportions, `&` is intersection, `|` is union, `!` is complement, and `[B]` subsets to condition on B.

Working from a Frequency Table

```r
# Contingency table - rows: Defective/OK, cols: Machine A / Machine B
tbl <- matrix(c(10, 40, 5, 45), nrow = 2,
              dimnames = list(c("Defective", "OK"), c("MachineA", "MachineB")))

n <- sum(tbl)

# Marginal probabilities
P_defective  <- sum(tbl["Defective", ]) / n   # P(Defective)
P_machineA   <- sum(tbl[, "MachineA"]) / n    # P(Machine A)

# Joint probability: P(Defective & Machine A)
P_def_and_A  <- tbl["Defective", "MachineA"] / n

# Conditional: P(Defective | Machine A)
P_def_given_A <- tbl["Defective", "MachineA"] / sum(tbl[, "MachineA"])

# Or equivalently:
P_def_given_A <- P_def_and_A / P_machineA

# Independence check
isTRUE(all.equal(P_def_and_A, P_defective * P_machineA))
```

Complement: $P(A^c) = 1 - P(A)$ "The probability that A does NOT happen."

```r
omega <- 1:6
A <- c(2, 4, 6)
P_not_A <- length(setdiff(omega, A)) / length(omega)   # 0.5
```

```r
# or equivalently
P_not_A <- 1 - length(A) / length(omega)
```

Use when: you want "at least one", "none", or "not A" style questions.

Union (OR): $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ "The probability that A or B (or both) happen."

```r
omega <- 1:6
A <- c(2, 4, 6)   # even
B <- c(5, 6)       # > 4
P <- function(event) length(event) / length(omega)

P(union(A, B))                          # direct: 4/6 = 0.667
P(A) + P(B) - P(intersect(A, B))        # formula: same result
```

Special case — **mutually exclusive** events ($A \cap B = \emptyset$):

```r
A <- c(1, 3)   # odd < 4
B <- c(5)      # odd > 4
# No overlap, so:
length(intersect(A, B)) == 0   # TRUE - mutually exclusive
P(union(A, B))                  # = P(A) + P(B) = 0.5
```

Intersection (AND): $P(A \cap B)$ "The probability that both A and B happen."

```r
omega <- 1:6
A <- c(2, 4, 6)   # even
B <- c(5, 6)       # > 4
P <- function(event) length(event) / length(omega)

# Direct from sets:
P(intersect(A, B))   # P({6}) = 1/6

# If A and B are INDEPENDENT, multiply (verify first):
P(A) * P(B)          # only valid when independence holds

# General multiplication rule:
# P(A & B) = P(B|A) * P(A)
P_B_given_A <- length(intersect(A, B)) / length(A)
P_B_given_A * P(A)   # = 1/6
```

Conditional Probability: $P(B \mid A) = \frac{P(A \cap B)}{P(A)}$ "The probability of B, given that A has already occurred."

```r
omega <- 1:6
A <- c(2, 4, 6)   # even
B <- c(5, 6)       # > 4
P <- function(event) length(event) / length(omega)

# Formula approach:
P(intersect(A, B)) / P(A)   # 0.333

# Set approach - restrict sample space to A, then ask what fraction is in B:
restricted <- A   # new sample space is A
mean(restricted %in% B)      # 0.333 - same result
```

Use when: "given that …", "knowing that …", "if A occurred …"

Independence Check: $P(A \cap B) = P(A) \cdot P(B)$ "Are A and B independent?"

```r
omega <- 1:6
A <- c(2, 4, 6)
B <- c(5, 6)
P <- function(event) length(event) / length(omega)

P(intersect(A, B))          # 1/6 = 0.1667
P(A) * P(B)                 # 0.5 * 0.333 = 0.1667

isTRUE(all.equal(P(intersect(A, B)), P(A) * P(B)))   # TRUE -> independent
```

Law of Total Probability "The overall probability of B, broken down across a partition of the sample space."

$$P(B) = \sum_i P(B \mid A_i) \cdot P(A_i)$$

```r
# Example: 600 items from 3 machines
# Each machine produces a different share, with different defect rates
items <- data.frame(
  machine  = c(rep("A", 200), rep("B", 250), rep("C", 150)),
  defective = c(rep(c(TRUE, FALSE), c(20, 180)),    # Machine A: 10% defective
                rep(c(TRUE, FALSE), c(12, 238)),    # Machine B: ~5% defective
                rep(c(TRUE, FALSE), c(30, 120)))    # Machine C: 20% defective
)

# Partition: the three machines
partition <- split(items, items$machine)

# P(B | A_i): defect rate within each machine group
P_B_given_Ai <- sapply(partition, function(g) mean(g$defective))
```

```r
# P(A_i): proportion of items from each machine
P_Ai <- sapply(partition, nrow) / nrow(items)

# Law of Total Probability
P_B <- sum(P_B_given_Ai * P_Ai)

# Verify directly:
mean(items$defective)    # should match P_B
```

Use when: problem has multiple groups/scenarios and you need the overall probability.

Bayes' Theorem: $P(A \mid B) = \frac{P(B|A) \cdot P(A)}{P(B)}$ "Reverse a conditional probability — update your belief after observing B."

```r
# Same items data frame from above
# Question: given an item is defective, what is P(it came from Machine C)?

B  <- items$defective                    # event B: defective
Ac <- items$machine == "C"               # event A: from Machine C

# All probabilities derived from the data:
P_B       <- mean(B)                     # P(defective)
P_Ac      <- mean(Ac)                    # P(Machine C)
P_B_Ac    <- mean(B & Ac)                # P(defective AND Machine C)

# Bayes:
P_Ac_given_B <- P_B_Ac / P_B             # P(Machine C | defective)

# Verify directly (restrict to defective rows):
mean(items$machine[items$defective] == "C")    # same answer
```

Quick Reference: Which Formula to Use?

Question type	Math	R (from sets)
Probability of A	$\|A\|/\|\Omega\|$	`length(A) / length(omega)`
A does NOT happen	$1 - P(A)$	`length(setdiff(omega, A)) / length(omega)`
A or B	$P(A) + P(B) - P(A \cap B)$	`P(union(A, B))`
A or B, mutually exclusive	$P(A) + P(B)$	`P(union(A, B))` — intersection is empty
A and B	$P(A \cap B)$	`P(intersect(A, B))`
A and B, independent	$P(A) \cdot P(B)$	`P(A) * P(B)` — only after confirming independence

Question type	Math	R (from sets)
B given A	$P(A \cap B)/P(A)$	`P(intersect(A,B)) / P(A)` or `mean(B_lgl[A_lgl])`
Are A and B independent?	$P(A \cap B) \stackrel{?}{=} P(A)P(B)$	`isTRUE(all.equal(P(intersect(A,B)), P(A)*P(B)))`
Overall P across groups	$\sum P(B \mid A_i)P(A_i)$	`sum(P_B_giv * P_Ai)` or `mean(B_col)` directly
Reverse conditional	Bayes' Theorem	`P_AB / P_B` or `mean(A_col[B_col])`

Random Variables and Probability Distributions

A random variable is a numerical mapping of outcomes from a sample space. In R, distributions are accessed using specific prefix letters: `d` (probability density/mass function), `p` (cumulative distribution function), `q` (quantile function), and `r` (random deviate generation).

- **Expected Value:** The long-run average of a distribution over many trials, denoted as $E(X)$ or μ.
- **Variance:** The measure of spread or dispersion around the expected value, denoted as $\text{Var}(X)$ or σ^2.

Discrete Distributions

- **Binomial Distribution:** Models the number of successes in n independent trials with a constant probability of success p. Use `dbinom(x, size, prob)`.
- **Poisson Distribution:** Models the count of rare events occurring within a fixed interval, where the mean rate is λ. The variance is equal to the expected value. Use `dpois(x, lambda)`.
- **Hypergeometric Distribution:** Models drawing from a finite population *without* replacement, making the events dependent. Use `dhyper(x, m, n, k)`.
- **Negative Binomial Distribution:** Models the number of failures before a specified number of successes occurs. Use `dnbinom(x, size, prob)`.
- **Expected Value (Binomial):**

$$E(X) = n \cdot p$$

- **Variance (Binomial):**

$$\text{Var}(X) = n \cdot p \cdot (1 - p)$$

- **Poisson Approximation to Binomial:** If a Binomial distribution has a large number of trials ($n \geq 20$) and a small probability of success ($p \leq 0.05$), the Poisson distribution acts as an excellent approximation using:

$$\lambda = n \cdot p$$

Calculating Probabilities in R

Every distribution in R follows a four-function naming convention: prefix + distribution name.

Prefix	Function	Returns
`d`	Density / PMF	$P(X = x)$ — exact probability (discrete) or density (continuous)
`p`	CDF	$P(X \leq x)$ — cumulative probability up to x
`q`	Quantile	The value x such that $P(X \leq x) = p$
`r`	Random	Generates random samples from the distribution

Binomial Examples

```r
# P(X = 3) where X ~ Bin(10, 0.4)
dbinom(3, size = 10, prob = 0.4)

# P(X <= 3)
pbinom(3, size = 10, prob = 0.4)

# P(X >= 4) = 1 - P(X <= 3)
1 - pbinom(3, size = 10, prob = 0.4)

# P(2 <= X <= 5)
pbinom(5, 10, 0.4) - pbinom(1, 10, 0.4)
```

Poisson Examples

```r
# P(X = 2) where X ~ Pois(lambda = 3)
dpois(2, lambda = 3)

# P(X <= 2)
ppois(2, lambda = 3)

# P(X > 2)
1 - ppois(2, lambda = 3)
```

Normal Examples

```r
# P(X < 70) where X ~ N(65, 9)  [sd = 3]
pnorm(70, mean = 65, sd = 3)

# P(X > 70)
pnorm(70, mean = 65, sd = 3, lower.tail = FALSE)
```

```r
# P(62 < X < 68)
pnorm(68, 65, 3) - pnorm(62, 65, 3)

# Find x such that P(X <= x) = 0.95
qnorm(0.95, mean = 65, sd = 3)
```

Continuous Distributions

- **Uniform Distribution:** Models outcomes that are equally likely across a continuous interval. Use `dunif(x, min, max)`.
- **Normal Distribution:** A symmetric, bell-shaped distribution completely defined by its mean μ and standard deviation σ. Use `dnorm(x, mean, sd)`.
- **Gamma Distribution:** Models positive, right-skewed data such as waiting times or insurance claims. It is defined by a shape parameter α and rate parameter β. Use `dgamma(x, shape, rate)`.
- **Exponential Distribution:** A specific case of the Gamma distribution modelling waiting times between events in a Poisson process. Use `dexp(x, rate)`.
- **Chi-Squared Distribution:** A right-skewed distribution dependent on degrees of freedom, often used in goodness-of-fit and likelihood ratio tests. Use `dchisq(x, df)`.

Z-Scores and Normal Probabilities

A Z-score standardizes a data point to represent how many standard deviations it falls away from the mean, allowing for the use of standard normal tables or probability functions to find the area under the curve.

- **Z-Score Formula:**

$$Z = \frac{X - \mu}{\sigma}$$

To calculate specific continuous probabilities in R using the Normal Distribution:

1. **Probability Less Than ($P(X < x)$):** Use `pnorm(x, mean, sd)`.
2. **Probability Greater Than ($P(X > x)$):** Use `pnorm(x, mean, sd, lower.tail = FALSE)` or `1 - pnorm(x, mean, sd)`.
3. **Probability Between ($P(a < X < b)$):** Calculate the cumulative area up to b and subtract the cumulative area up to a using `pnorm(b, mean, sd) - pnorm(a, mean, sd)`.

Sampling Distributions and The Central Limit Theorem

The behavior of sample statistics over repeated sampling forms the basis of frequentist inference.

- **Central Limit Theorem:** Regardless of the underlying population distribution, the sampling distribution of the sample mean $\bar{X}$ approaches a Normal distribution as the sample size n increases.
- **Standard Error:** The standard deviation of the sampling distribution, quantifying the precision of the sample mean.
- **Standard Error Formula:**

$$SE = \frac{\sigma}{\sqrt{n}}$$

Confidence Intervals

A confidence interval gives a range of plausible values for a population parameter based on sample data.

- **Interpretation:** If we repeated the sampling procedure many times, $(1-\alpha)\%$ of the constructed intervals would contain the true parameter.

CI for a Mean (Known σ or Large n, use Z)

$$\bar{x} \pm z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}}$$

```
# Manual Z-based CI
z <- qnorm(0.975)           # 1.96 for 95% CI
lower <- xbar - z * (sigma / sqrt(n))
upper <- xbar + z * (sigma / sqrt(n))
```

CI for a Mean (Unknown σ, use t)

$$\bar{x} \pm t_{\alpha/2,\, n-1} \cdot \frac{s}{\sqrt{n}}$$

```
# R computes this automatically
t.test(x, conf.level = 0.95)$conf.int
```

CI for a Proportion

```
prop.test(x, n, conf.level = 0.95)$conf.int
# or for exact binomial CI:
binom.test(x, n, conf.level = 0.95)$conf.int
```

Key Values

Confidence Level	$z_{\alpha/2}$
90%	1.645
95%	1.960
99%	2.576

- **Decision rule:** If μ_0 (the hypothesised value) lies **outside** the CI, reject H_0.

Hypothesis Testing Framework: Step-by-Step Guide

Hypothesis testing is a structured way to evaluate competing claims about population parameters based on sample data.

1. **Define Hypotheses:** Establish the Null Hypothesis (H_0), representing the status quo or "no difference", and the Alternative Hypothesis (H_1), representing the specific effect you are testing for (e.g., $H_1 : \mu_1 \neq \mu_2$ for a two-sided test or $H_1 : \mu_1 < \mu_2$ for a one-sided test). Define your significance level α (typically 0.05).
2. **Check for Normality:** Run the Shapiro-Wilk test via `shapiro.test(x)` on your data.
   - If $p > 0.05$: The data does not significantly deviate from a Normal distribution. Proceed with Parametric tests.
   - If $p < 0.05$: The data is not normal. Proceed with Non-Parametric tests.
3. **Check for Equal Variance (Parametric Only):** If comparing two independent normally distributed groups, use Bartlett's test via `bartlett.test(list(group1, group2))`.
   - If $p > 0.05$: Assume equal variances.
   - If $p < 0.05$: Variances are unequal.
4. **Select and Execute the Correct Test:**
   - *Normal + Equal Variance:* Standard Two-Sample T-Test `t.test(x, y, alternative="two.sided")`.
   - *Normal + Unequal Variance:* Welch's T-Test (R handles this automatically in `t.test` by default).
   - *Non-Normal (Independent):* Wilcoxon Rank Sum Test `wilcox.test(x, y)`.
   - *Paired Data (Before/After):* Add the `paired = TRUE` argument to `t.test` or `wilcox.test`.
   - *Categorical/Proportions:* Use `binom.test()` for exact binomial tests on count data or `prop.test()` for proportions.
5. **Interpret the Results:** Look at the p-value in the R output.
   - If $p < \alpha$: Reject the Null Hypothesis (H_0) and accept the Alternative Hypothesis (H_1).
   - If $p > \alpha$: Fail to reject the Null Hypothesis (H_0).

Enumerative Data Analysis

Analysis of qualitative, categorical data relies on comparing observed frequencies against expected frequencies.

- **Goodness-of-Fit Test:** Evaluates if a single categorical variable matches a claimed theoretical distribution. Use `chisq.test(x, p = expected_probs)`.
- **Test of Independence:** Evaluates if two categorical variables are associated or independent. Use `chisq.test(table_data)`.
- **Rule of 5 Assumption:** Chi-squared tests require at least 80% of expected counts to be ≥ 5, and no expected count to be < 1.
- **Fisher's Exact Test:** The non-parametric alternative for independence testing when the Rule of 5 assumption is violated in small samples. Use `fisher.test(table_data)`.
- **Effect Size:** Quantifies the magnitude of the difference (e.g., Cohen's d for means, Phi coefficient for categorical associations) independent of sample size.
- **Expected Counts Formula:**

$$E_{ij} = \frac{(\text{Row Total})(\text{Column Total})}{\text{Grand Total}}$$

- **Chi-Squared Test Statistic:**

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

- **Phi Coefficient Formula:**

$$\phi = \sqrt{\frac{\chi^2}{n}}$$

- **Cohen's d Formula:**

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s}$$

Creating Contingency Tables in R

```r
# From raw data
tbl <- table(df$var1, df$var2)

# From counts directly
tbl <- matrix(c(10, 20, 30, 40), nrow = 2, byrow = TRUE)

# Run chi-squared test
chisq.test(tbl)

# Fisher's exact test (small samples)
fisher.test(tbl)
```

Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation is a method for estimating the parameters of a statistical model by selecting the values that make the observed data most probable.

- **Log-Likelihood:** Because multiplying many small probabilities causes numerical instability (underflow), it is standard practice to take the natural logarithm. R density functions natively support this via the `log = TRUE` argument.
- **The MLE Concept:** The Maximum Likelihood Estimate (MLE) is the specific parameter value $\hat{\theta}$ that maximizes the log-likelihood function.
- **Analytical Solutions:** Some models have closed-form MLEs. For a Poisson distribution, $\hat{\lambda}$ is the sample mean $\bar{x}$. For a Binomial distribution, $\hat{p}$ is the sample success rate $\bar{y}/n$.
- **Likelihood Surfaces:** When a model contains two unknown parameters (e.g., Normal distribution with unknown μ and σ), the log-likelihood becomes a 3D surface over a parameter space.
- **Likelihood Function:**

$$L(\theta) = \prod_{i=1}^{n} f(x_i \mid \theta)$$

- **Log-Likelihood Function:**

$$\ell(\theta) = \sum_{i=1}^{n} \log f(x_i \mid \theta)$$

- **Likelihood Ratio Test Statistic:**

$$\Lambda = -2\left[\ell(\hat{\theta}_0) - \ell(\hat{\theta})\right]$$

Numerical Optimization and Calculus in R

When analytical differentiation yields no closed-form solution (such as the shape parameter α in a Gamma distribution), numerical optimization algorithms must be used to find the MLE.

- **Minimization via `mle()`:** R's `mle()` function (from the `stats4` package) estimates parameters by *minimizing* a provided negative log-likelihood function. It requires the negative log-likelihood function, a `start` list of initial guesses, and `nobs` (number of observations).
- **Checking Convergence:** An optimization model that hasn't converged can look perfectly reasonable but be completely wrong. Always check `fit@details$convergence`. A code of 0 indicates success, while non-zero values (1, 52, 54) indicate failures or numerical faults.
- **Symbolic Differentiation:** R can compute analytical derivatives using the `D(expression, variable)` function.

Optimization Methods

Under the hood, `mle()` utilizes the general-purpose `optim()` function, which relies on different algorithms to navigate the parameter space.

- **Nelder-Mead:** A derivative-free method that navigates using a simplex (geometric shape). It is robust and handles irregular surfaces well, but can be slow to converge. It is the default method in `optim()`.
- **BFGS:** A quasi-Newton method that approximates the Hessian matrix (second derivatives) from successive gradient evaluations. It is much faster than Nelder-Mead on smooth log-likelihoods and is the default algorithm in `mle()`.
- **Gradient Ascent:** An iterative algorithm that steps in the direction of the gradient by a fixed step size γ (the learning rate). It is sensitive to γ and requires manual implementation using analytical derivatives.
- **L-BFGS-B and Brent:** Specialized algorithms employed when the parameter space must be constrained by lower and upper bounds (e.g., standard deviation and probabilities cannot be negative). `Brent` is strictly for single-parameter bounded optimization.
- **Gradient Ascent Update Rule:**

$$\theta^{(t+1)} = \theta^{(t)} + \gamma \cdot \left.\frac{\partial \ell}{\partial \theta}\right|_{\theta^{(t)}}$$