Exercise 3

Monte Carlo

Olesia Galynskaia 12321492

2025

Fix the random seed for reproducibility

```
set.seed(12321492)
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE, fig.width = 6, fig.height = 4, dpi
```

Preparation

Monte Carlo helpers

Monte Carlo performs basic Monte Carlo integration over an interval using uniform sampling.
It draws random points, evaluates the function at those points, averages the values, and multiplies by the interval length.
It also provides an estimated variance and standard error of the Monte Carlo estimator.

Importance sampling is used when uniform sampling is inefficient or unsuitable for the integral.
It draws samples from a chosen proposal distribution and corrects them using importance weights.
This approach allows the estimator to reduce variance and handle integrals with more difficult shapes or supports.

```
# Uniform Monte Carlo integration over [a, b]
mc_integrate_unif <- function(h, a, b, m) {
  x  <- runif(m, min = a, max = b)
  hx <- h(x)
  estimate <- mean(hx) * (b - a)
  var_hat  <- var(hx * (b - a)) / m
  se_hat   <- sqrt(var_hat)
  list(estimate = estimate, var = var_hat, se = se_hat)
}

# Importance sampling
mc_importance <- function(h, rgen, dgen, m) {
  x <- rgen(m)
  w <- h(x) / dgen(x)
  estimate <- mean(w)
  var_hat  <- var(w) / m
  se_hat   <- sqrt(var_hat)
  list(estimate = estimate, var = var_hat, se = se_hat)
}
```

MSE helper

This block estimates the mean squared error of a given estimator.
It repeatedly generates samples, computes the estimator for each sample, and compares each estimate with the true parameter value.
The final result shows how far the estimator deviates from the truth on average.

```
mc_mse <- function(theta_true, theta_hat_fun, rgen_sample_fun, m) {
  thetas <- replicate(m, {
    x <- rgen_sample_fun()
    theta_hat_fun(x)
  })
  mse_hat <- mean((thetas - theta_true)^2)
  list(estimates = thetas, mse = mse_hat)
}
```

Confidence Interval coverage helper

This helper evaluates how often a confidence interval contains the true parameter value.
It runs many simulated samples, constructs a confidence interval for each, and checks whether the true value lies inside.
The average proportion of successful cases is the estimated coverage probability.

```
mc_coverage <- function(rgen_sample_fun, ci_fun, theta_true, m) {
  inside <- replicate(m, {
    x <- rgen_sample_fun()
    bounds <- ci_fun(x)
    (bounds[1] <= theta_true) && (theta_true <= bounds[2])
  })
  mean(inside)
}
```

Power helper

This helper estimates the power curve of a statistical test.
It simulates samples under different effect sizes, applies the test to each sample, and records how often the null hypothesis is rejected.
The resulting power values describe how sensitive the test is to varying magnitudes of deviation from the null.

```
mc_power <- function(rgen_sample_fun, test_fun, al, deltas, m) {
  power_vals <- sapply(deltas, function(d) {
    pvals <- replicate(m, {
      x <- rgen_sample_fun(d)
      test_fun(x)
    })
    mean(pvals < al)
  })
  names(power_vals) <- paste0("Delta_", deltas)
  power_vals
}
```

Task 5 — Exponential via Inverse Transform

1. Uniform Monte Carlo on [1,6]

n this first step we approximate the integral $\int_1^6 e^{-x^3}\,dx$ using Monte Carlo integration with uniformly distributed random variables on [1,6]. We then compare the Monte Carlo estimate with the numerical result from the function integrate.

```
# integrand
h <- function(x) exp(-x^3)

# number of Monte Carlo samples
m <- 100000

# Monte Carlo integration with uniform sampling on [1, 6]
res_mc_5_1 <- mc_integrate_unif(h = h, a = 1, b = 6, m = m)
res_mc_5_1
```

```
## $estimate
## [1] 0.08512251
##
## $var
## [1] 8.205e-07
##
## $se
## [1] 0.0009058146
```

```
int_exact_5_1 <- integrate(function(x) exp(-x^3), lower = 1, upper = 6)
int_exact_5_1
```

```
## 0.08546833 with absolute error < 3.2e-07
```

The Monte Carlo estimate for the integral on $[1, 6]$ is 0.0851225 with an estimated standard error of 9.06×10^{-4}.
The numerical routine `integrate` returns 0.0854683.

The Monte Carlo result is very close to the value from integrate().
The small gap between them is normal, because Monte Carlo always has some random noise, and the standard error we got explains this difference.
So overall the method worked fine for this integral.

2. Monte Carlo for $b = \infty$

In the second step we consider the integral

$$I = \int_1^\infty e^{-x^3}\,dx.$$

For this case it is more convenient to use Monte Carlo integration with importance sampling. We choose a shifted exponential distribution as proposal: let $Y \sim \text{Exp}(1)$ and $X = 1 + Y$, so the proposal density is

$$f_X(x) = e^{-(x-1)}, \quad x \geq 1.$$

This density has the same support as the integral, puts most mass near $x = 1$ where the integrand is non-negligible, and decays in the tail, which helps to keep the weights stable. We then compare the Monte Carlo estimate with the result of the function `integrate()`.

```
h <- function(x) exp(-x^3)

rgen_shifted_exp <- function(m) rexp(m, rate = 1) + 1
dgen_shifted_exp <- function(x) dexp(x - 1, rate = 1)

m <- 100000

res_mc_5_2 <- mc_importance(
  h    = h,
  rgen = rgen_shifted_exp,
  dgen = dgen_shifted_exp,
  m    = m
)
res_mc_5_2
```

```
## $estimate
## [1] 0.08506672
##
## $var
## [1] 1.296265e-07
##
## $se
## [1] 0.0003600368
```

```
int_exact_5_2 <- integrate(function(x) exp(-x^3), lower = 1, upper = Inf)
int_exact_5_2
```

```
## 0.08546833 with absolute error < 6.2e-06
```

The Monte Carlo estimate for the integral on $[1, \infty)$ is 0.0850667 with an estimated standard error of 3.6×10^{-4}.
The function `integrate()` returns 0.0854683.
The Monte Carlo estimate for the integral on $[1, \infty]$ is almost identical to the value returned by integrate(). The small difference between them is fully explained by the Monte Carlo standard error, which is very low in this setup.
Using the shifted exponential distribution works well here, because most of the probability mass falls in the region where the integrand is non-negligible, so the estimator stays stable and accurate.

3.

In part 1 we used a uniform distribution on [1,6]. The problem is that the function e^{-x^3} becomes extremely small as x grows.
Most of the random points from the uniform distribution fall in places where the function is almost zero. Because of that, the Monte Carlo estimates jump around more, and the result is not as stable.

In part 2 we used a better distribution (a shifted exponential) that puts most random points near x=1, where the integrand actually has non-negligible values.

Since the sampling happens in the "useful" region, the Monte Carlo weights are much more stable, and the estimate becomes very accurate.

Shortly, Monte Carlo works better in part 2 because the sampling distribution matches the shape of the integrand.
In part 1 the uniform distribution wastes points where the function is basically zero, which increases variability.

Task 6 — Exponential via Inverse Transform

We observe an i.i.d. sample $x_1, \ldots, x_n$ from a distribution symmetric around μ. We want to test

$$H_0 : \mu = \mu_0 \quad \text{vs.} \quad H_1 : \mu > \mu_0.$$

The sign test uses the statistic

$$t_n = \sum_{i=1}^{n} \psi(x_i - \mu_0), \qquad \psi(t) = \begin{cases} 1, & t > 0, \\ 0, & t \leq 0. \end{cases}$$

Under H_0 we have $t_n \sim \text{Bin}(n, 0.5)$, and for large n the distribution of t_n is approximately normal with

$$t_n \approx N\big(n \cdot 0.5,\ n \cdot 0.5(1 - 0.5)\big) = N\big(n/2,\ n/4\big).$$

Large values of t_n support H_1.

1. Function for the sign test (asymptotic)

First we implement the sign test using the asymptotic normal distribution of t_n.
The function takes a sample `x` and the value μ_0 and returns the test statistic and the one-sided p-value for the alternative $\mu > \mu_0$.

The asymptotic version of the sign test treats t_n as approximately normal with mean $n/2$ and variance $n/4$ under H_0.
We standardise t_n to a z-statistic and use the upper tail of the normal distribution to obtain the one-sided p-value.

```r
sign_test_asymp <- function(x, mu0) {
  n <- length(x)
  # number of positive signs relative to mu0
  t_n <- sum(x > mu0)

  # mean and variance of Bin(n, 0.5) under H0
  mean0 <- n * 0.5
  var0  <- n * 0.5 * 0.5

  # asymptotic z-statistic
  z <- (t_n - mean0) / sqrt(var0)

  # one-sided p-value: H1: mu > mu0 (more positive signs than expected)
  pval <- 1 - pnorm(z)

  list(statistic = t_n, z = z, p.value = pval)
}
```

2. Exact sign test using binom.test

Next we implement the exact version of the sign test.
Here the p-value is computed from the exact $\text{Bin}(n, 0.5)$ distribution using `binom.test()`.

The exact sign test uses the binomial distribution of t_n under H_0.
The one-sided p-value is $P\big(T_n \geq t_n^{\text{obs}}\big)$ for
$T_n \sim \text{Bin}(n, 0.5)$, which is obtained via `binom.test()`.

```
sign_test_exact <- function(x, mu0) {
  n <- length(x)
  t_n <- sum(x > mu0)

  bt <- binom.test(
    x = t_n,
    n = n,
    p = 0.5,
    alternative = "greater"  # H1: mu > mu0
  )

  list(statistic = t_n, p.value = bt$p.value)
}
```

3. Simulation study: Cauchy, sign tests vs t-test

We now compare the asymptotic and exact sign tests with the usual one-sample t-test by simulation.
We consider Cauchy data, which are symmetric but have heavy tails and no finite variance. We set the significance level to $\alpha = 0.1$, use sample size $n = 30$ and perform $m = 500$ repetitions with seed 12321492 (numeric part of the student number).

We look at two cases:

1. Case 1: $\mu_0 = 0$ and $X_i \sim \text{Cauchy}(0, 1)$
   (the null hypothesis is true; the rejection rate estimates the type I error).
2. Case 2: $\mu_0 = -0.5$ and $X_i \sim \text{Cauchy}(0, 1)$
   (the true location is larger than μ_0; the rejection rate estimates the power).

```
set.seed(12321492)

n     <- 30
m     <- 500
alpha <- 0.1

# helper to run all three tests for one data set
run_all_tests <- function(x, mu0, alpha) {
  p_asymp <- sign_test_asymp(x, mu0)$p.value
  p_exact <- sign_test_exact(x, mu0)$p.value
  p_ttest <- t.test(x, mu = mu0, alternative = "greater")$p.value

  c(
    asymptotic_sign = (p_asymp < alpha),
    exact_sign      = (p_exact < alpha),
    t_test          = (p_ttest < alpha)
```

```
  )
}

simulate_case <- function(mu_true, mu0) {
  rejections <- replicate(m, {
    x <- rcauchy(n, location = mu_true, scale = 1)
    run_all_tests(x, mu0, alpha)
  })

  # each row corresponds to one test; take row means
  colMeans(t(rejections))
}

# Case 1: mu0 = 0, data from Cauchy(0, 1)  -> type I error
res_case1 <- simulate_case(mu_true = 0, mu0 = 0)

# Case 2: mu0 = -0.5, data still from Cauchy(0, 1) -> power
res_case2 <- simulate_case(mu_true = 0, mu0 = -0.5)

res_case1
```

```
## asymptotic_sign      exact_sign          t_test
##           0.096           0.056           0.074
```

```
res_case2
```

```
## asymptotic_sign      exact_sign          t_test
##           0.626           0.500           0.200
```

```
res_table <- rbind(
  "Case 1: mu0 = 0, Cauchy(0,1)"   = res_case1,
  "Case 2: mu0 = -0.5, Cauchy(0,1)" = res_case2
)
res_table
```

```
##                                asymptotic_sign exact_sign t_test
## Case 1: mu0 = 0, Cauchy(0,1)             0.096      0.056  0.074
## Case 2: mu0 = -0.5, Cauchy(0,1)          0.626      0.500  0.200
```

The empirical rejection probabilities are summarised in the table below.

```
##                                asymptotic_sign exact_sign t_test
## Case 1: mu0 = 0, Cauchy(0,1)             0.096      0.056  0.074
## Case 2: mu0 = -0.5, Cauchy(0,1)          0.626      0.500  0.200
```

In Case 1 (null hypothesis true), the asymptotic sign test is very close to the nominal level $\alpha = 0.1$, and the exact sign test is slightly conservative, which is typical for exact binomial tests. The t-test also rejects less often than α, but its behaviour is less reliable because the Cauchy distribution has heavy tails and produces unstable sample variances.

In Case 2 (alternative true), both sign tests show clear power: the asymptotic version rejects in about 63% of repetitions, and the exact version in about 50%. The t-test performs much worse (about 20% rejections),

since the heavy tails of the Cauchy distribution strongly affect the test statistic and reduce its ability to detect a location shift.

Overall, the sign tests are considerably more robust than the t-test in this setting.