FCT NOVA

Project Report

Random Variable Generation and Monte Carlo Methods

Leonard Storcks

Course by Dr. Regina Bispo

January 12, 2024

Abstract

In this first project sampling methods (like the inverse transform method) and Monte Carlo methods (like Monte Carlo Integration with Importance Sampling) are introduced and applied.

Contents

1 Sampling Methods

1.1 What is sampling? - an intuitive introduction

Probability distributions are ubiquitous. Consider the air molecules around you - what we feel as a temperature stems from the microscopic movement of those molecules, some moving slower, some quicker, some in one, some in another direction. Actually (under ideal conditions), if I was to measure those velocities of many molecules only along one axis and do a normalized histogram of my measurements, I would see a Gaussian emerge.

Now we want to do the opposite: Given a probability distribution $f(x)$, we want to generate measurements x_i such that in the limit of lots or measurements their normalized histogram resembles the probability distribution $f(x)$, e.g. to simulate my previous velocity measurement. This is called sampling and has numerous applications, e.g. in smartly approximating integrals (Monte Carlo integration).

Base of all sampling methods: Sampling from the uniform distribution is *easy*

The standard continuous uniform distribution is given by the probability density function (PDF)

$$f(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

We would now like to generate pseudo-random numbers $U \sim \mathcal{U}(0,1)$. Let us present RANDU, an infamous, simple linear congruential generator (LCG), given by

$$v_{n+1} = (65539 \cdot v_n) \mod 2^{31}, \quad \text{seed } v_0, \quad \text{pseudo-random number } u_n = \frac{v_n}{2^{31}} \tag{2}$$

where based on an integer sequence uniformly distributed numbers u_n are generated. Now RANDU is not infamous because it is especially good, but rather because it is especially bad in the sense that generated numbers are closely related. A more in-depth discussion about the problems of RANDU and linear congruential generators in general can be found in Press et al., 2007, chapter 7, where more advanced methods are also discussed.

In R for instance the default number generator is the Marsenne-Twister (Matsumoto and Nishimura, 1998) according to the Comprehensive R Archive Network.

1.2 Inverse Transform Method | sampling from distributions with algebraically invertible cumulative distribution functions (CDFs)

Consider we want to sample from a distribution with PDF $f(x)$ and cumulative distribution function (CDF) $F(x) = \int_{-\infty}^{x} f(x')dx'$. Given a uniformly distributed variable $U \sim \mathcal{U}(0,1)$, $F^{-1}(U)$ is distributed according to f, as

$$P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x), \quad x \in \mathbb{R} \tag{3}$$

where we used that as of $f(x) \geq 0$, $F(x)$ is monotonically increasing, so its application keeps the inequality intact.

This is the foundation of the inverse transform method which is implemented in R in code-snippet 1 and illustrated at the hand of sampling from a Gaussian in figure 1. Note that the inverse transform method would actually not be used on a Gaussian as the inverse of its CDF has no closed-form expression and e.g. the Box-Muller method (Box and Muller, 1958) would be used.

```
1    inverse_transform_method <- function(n, F_inv) {
2        # Draw n samples from a distribution with CDF F
3        # given its inverse F_inv.
4        u <- runif(n) # draw n samples from U(0,1)
5        return(F_inv(u)) # apply the inverse transform
6    }
```

Code-Snippet 1: Inverse Transform Method in R

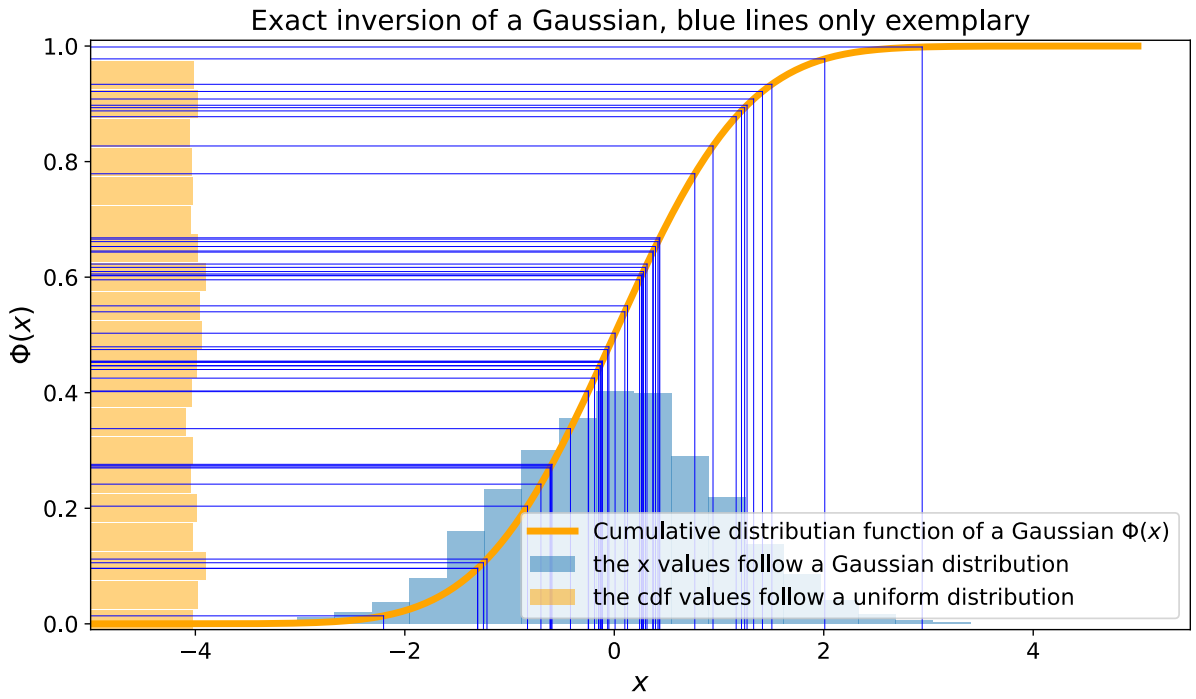

Figure 1: Illustration of the Inverse Transform Method

1.2.1 Inverse Transform Method applied to the standard Laplace distribution (task 1)

The standard Laplace distribution is given by the PDF

$$f(x) = \frac{1}{2}e^{-|x|}, \quad x \in \mathbb{R} \tag{4}$$

with the CDF following from $F(x) = \int_{-\infty}^{x} f(x')dx'$ to

$$F(x) = \begin{cases} \frac{1}{2}e^{x} & \text{if } x \leq 0 \\ 1 - \frac{1}{2}e^{-x} & \text{if } x > 0 \end{cases} \tag{5}$$

with the inverse CDF

$$F^{-1}(u) = \begin{cases} \ln(2u) & \text{if } u \leq \frac{1}{2} \\ -\ln(2(1-u)) & \text{if } u > \frac{1}{2} \end{cases} \tag{6}$$

where $\frac{1}{2}$ as the transitioning point between the pieces follows intuitively from the symmetry of the PDF around $x = 0$.

The PDF and inverse CDF are implemented in R in code-snippet 2 and 3 respectively. The results are illustrated in figure 2 based on the plotting code in code-snippet 4.

```r
f_laplace <- function(x) {
    # PDF of the Laplace distribution
    return(1/2 * exp(-abs(x)))
}
```

Code-Snippet 2: PDF of the standard Laplace distribution in R

```r
F_inv_laplace <- function(u) {
    # Inverse CDF of the standard Laplace distribution
    # in a vectorized implementation
    res <- numeric(length(u)) # initialize output vector of correct length
    res[u <= 1/2] <- log(2*u[u <= 1/2]) # inverse for u <= 1/2
    res[u > 1/2] <- -log(2*(1-u[u > 1/2])) # inverse for u > 1/2
    return(res)
}
```

Code-Snippet 3: Inverse CDF of the standard Laplace distribution in R

```r
plot_hist_and_dist <- function(f, x, title, hist_label, dist_label, filename) {
    # This function plots a histogram of the data in x and overlays
    # a distribution function f with the given labels and title
    # applied. The plot is then saved.
    data <- data.frame(x)
    colors <- eval(parse(text = paste0('c("', hist_label, '" = "cornflowerblue", "', dist_label , '" = "coral1")')))
    p <- ggplot(data, aes(x)) +
        geom_histogram(aes(y = ..density.., color = hist_label), bins = 60, fill = "cornflowerblue", key_glyph = "rect") +
        stat_function(fun = f_laplace, aes(color = dist_label), n = 1000, key_glyph = "rect", size = 2, alpha = .8) +
        labs(title = title, x = "x", y = "Density", color = "Legend") +
        scale_color_manual(values = colors) +
        theme_bw() +
        theme(legend.position=c(.75,.8), legend.box.background = element_rect(colour = "black"), text = element_text(size = 20))
    ggsave(file = paste0("figures/", filename), plot = p, width = 10, height = 8)
}
```

Code-Snippet 4: Plotting code for histograms with PDFs

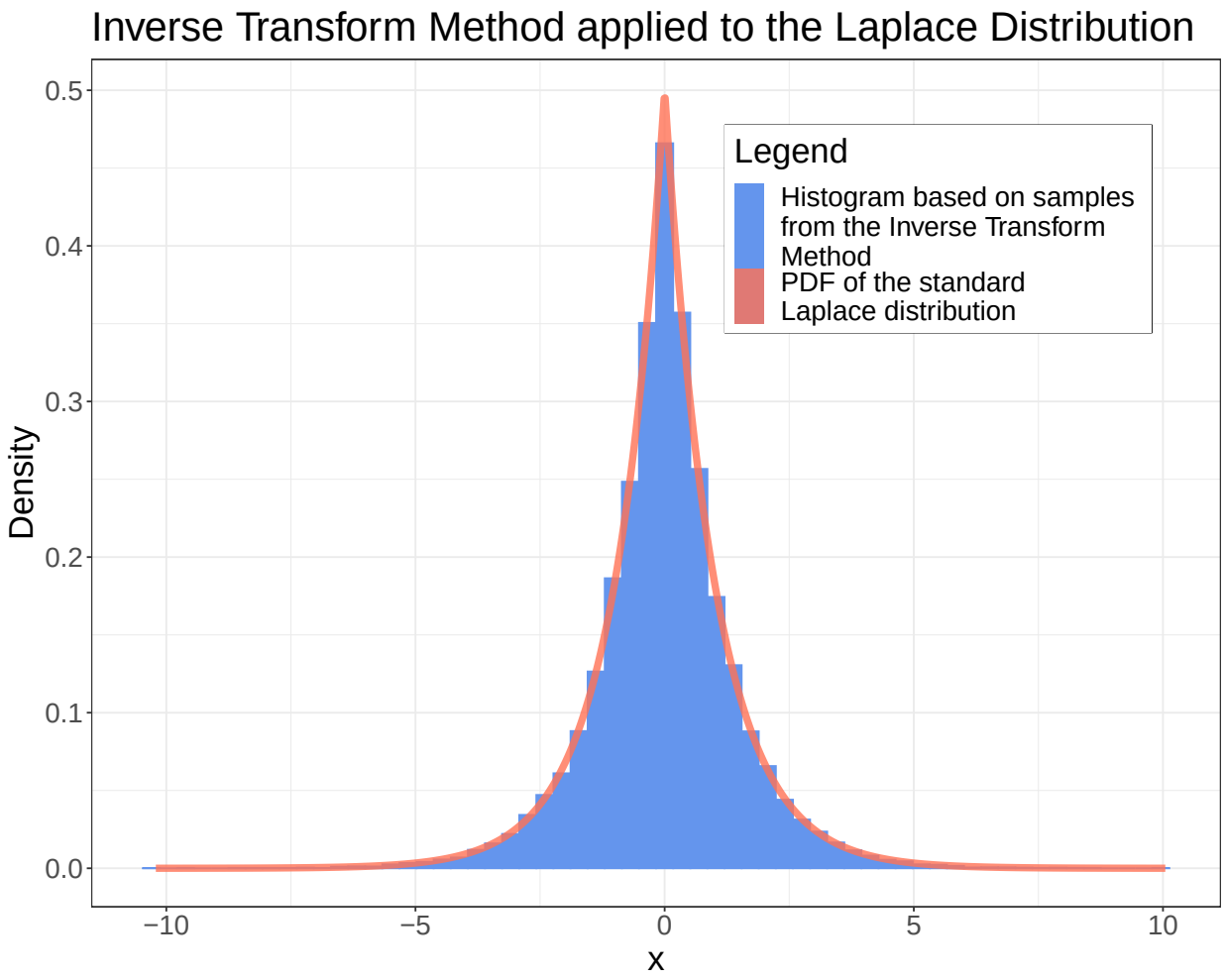

Figure 2: Illustration of the Inverse Transform Method applied to the standard Laplace distribution

1.3 Sampling from a discrete distribution (task 2)

Consider a random variable X with the following probability mass function

$$P(X = x) = \begin{cases} 0.1 & \text{if } x = 0 \\ 0.15 & \text{if } x = 1 \\ 0.25 & \text{if } x = 2 \\ 0.3 & \text{if } x = 3 \\ 0.2 & \text{if } x = 4 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Intuitively sampling according to such a discrete distribution is simple: For uniform draws between 0 and 1, each section $a \leq x < b$ within has a probability $b - a$. Therefore, we only

have to associate each x with a section of length $P(X = x)$.

The sections we are interested in are directly given by the cumulative distribution function, as illustrated in figure 3. An implementation of the discrete inverse transform is given in code-snippet 5, an application to the problem at hand is given in code-snippet 6 and the result is illustrated in figure 3, where e.g. at a sample size of 10000 in code-snippet 6 and 1000 samples for figure 3 close resemblance to the true distribution is achieved, the better, the larger the sample size.

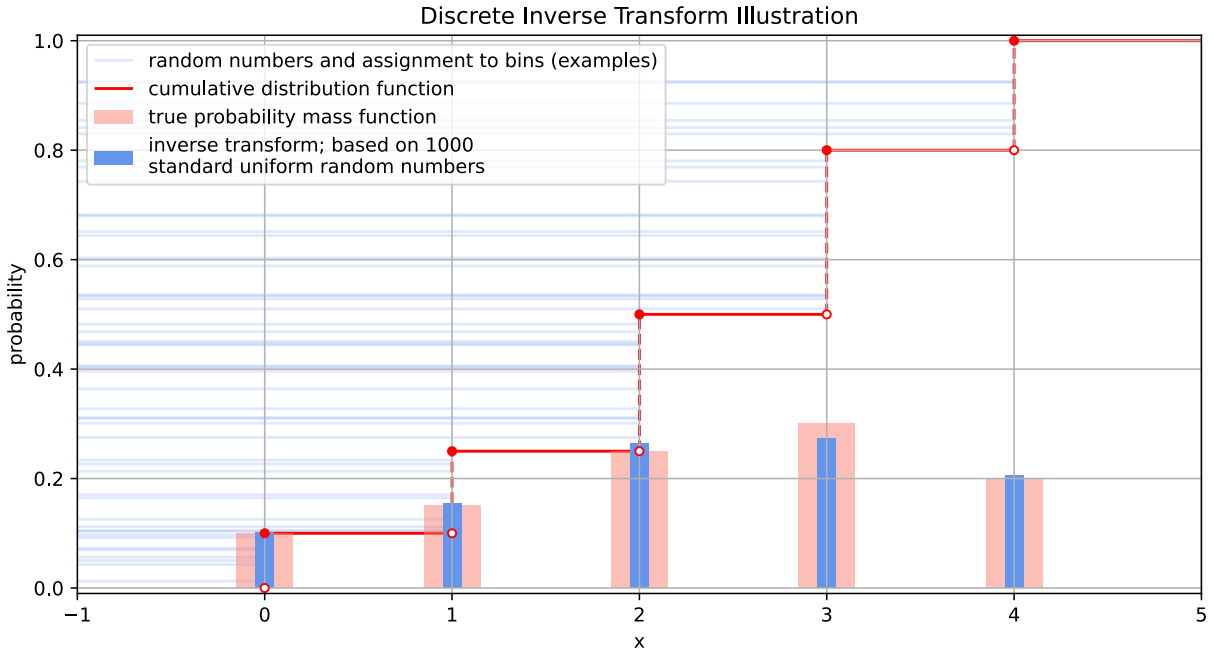

Figure 3: Illustration of the Inverse Transform Method applied to the standard Laplace distribution

```r
    # give the PDF as [[x1, x2, ...], [P1, P2, ...]]
    # where P1, P2 are the probabilities
    # and x1, x2 are the values of the random variable
    discrete_inverse_transform <- function(n, x_and_p) {
        # get the cumulative probabilities
        cum_probs <- cumsum(x_and_p[2, ])
        # generate n random numbers
        u <- runif(n)
        # find the intervals in which the random numbers fall
        unqs <- findInterval(u, cum_probs, rightmost.closed = TRUE)
        ips <- x_and_p[1, unqs + 1] # x-values with #occurences based
        #                             on hits in the defined bins
        return(ips) # return the sample
    }
```

Code-Snippet 5: Discrete Inverse Transform Method in R

```r
    x <- c(0, 1, 2, 3, 4)
    p <- c(0.1, 0.15, 0.25, 0.3, 0.2)
    x_and_p <- rbind(x, p)
    disc_sample <- discrete_inverse_transform(10000, x_and_p)
    print(proportions(table(disc_sample)))
    # result (seed: 1234), sample size 10000:
    #      0      1      2      3      4
    # 0.1032 0.1586 0.2438 0.3006 0.1938
    # for sample size 100000:
    # 0.10051 0.15127 0.24971 0.29966 0.19885
```

Code-Snippet 6: Discrete Inverse Transform for sampling from the distribution specified in equation 7.

1.4 Acceptance-rejection method

Let's say we want to sample from a complex distribution $p(x)$ and we can sample from a simple distribution $f(x)$ with $p(x) \leq Cf(x)$, some constant C.

We can then draw samples from $p(x)$ using the Acceptance-Rejection method, consisting of the following steps:

1. Generate a trial value x from $f(x)$ (e. g. using exact inversion)
2. Generate a value y from a uniform distribution $0 \leq y < Cf(x)$
3. If $y \leq p(x)$ return x as the sample value
4. Else, reject the trial value x and draw again

An illustration is provided in figure 4.

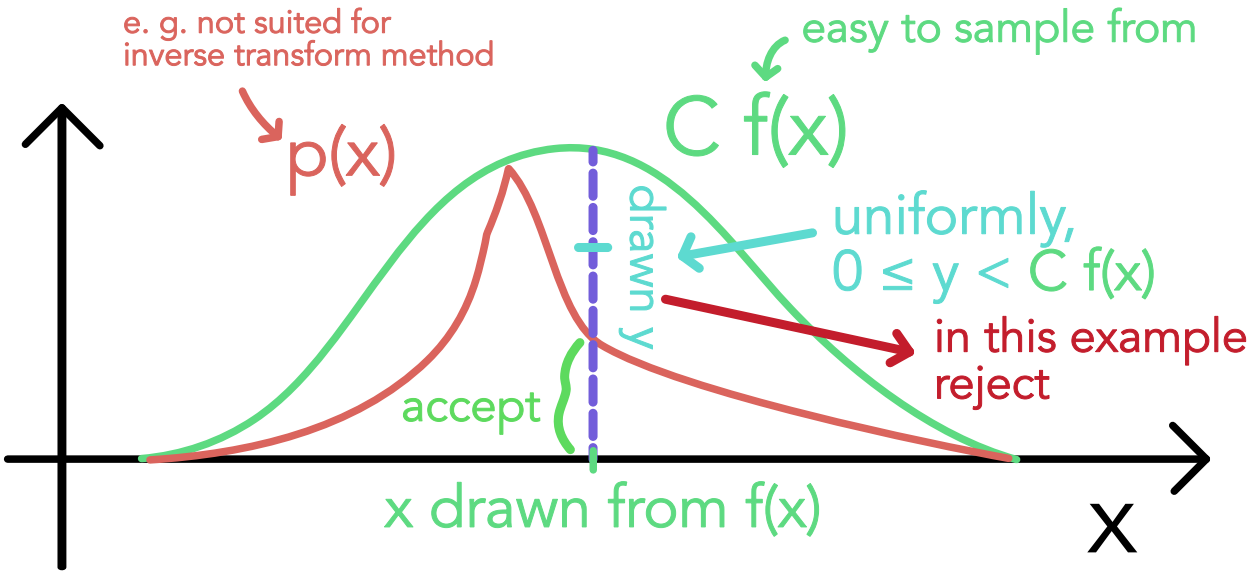

Figure 4: Illustration of the Acceptance-Rejection Method

Intuition: The logic behind this is very clear if we just choose $f(x) = const.$ i. e. we uniformly sample x. This, however, would be very inefficient, so we use $Cf(x)$ that lies on top $p(x)$ as closely as possible.

Proof that we get the correct distribution: The probability dq to get (accept) a certain x within dx is

$$dq = \underbrace{f(x)dx}_{\substack{\text{from} \\ \text{step 1}}} \underbrace{\frac{p(x)}{Cf(x)}}_{\substack{\text{step} \\ \text{2 to 4}}} = \frac{1}{C}p(x)dx \propto p(x)dx \tag{8}$$

Note that this is very inefficient if the rejection area is large and efficiency rapidly decreases in higher dimensions (then one might try to construct samples through a stochastic process).

1.4.1 Sampling from a conditioned Gamma distribution (task 3)

We would like to sample from $Gamma(2, 1)$ conditional on the random variable being greater than 5, i.e.

$$p(x) = \begin{cases} \frac{x\exp(-x)}{6\exp(-5)} & \text{if } x \geq 5 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

(normalized). We use the acceptance-rejection method with $f(x)$ being an exponential distribution with $\lambda = 0.5$ conditional on the random variable being greater than 5, i.e.

$$f(x) = \begin{cases} \frac{1}{2\exp\left(-\frac{5}{2}\right)} \exp(-\frac{x}{2}) & \text{if } x \geq 5 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

which we have normalized. We can draw from $f(x)$ using the inverse transform method, where we can e.g. efficiently take care of the conditioning by drawing uniformly between $\int_0^5 \frac{1}{2}\exp(-\frac{x}{2})dx = 1 - \exp\left(-\frac{5}{2}\right)$ and 1 in the inverse transform method.

The result is illustrated in figure 5 and the code is given in code-snippet 7.

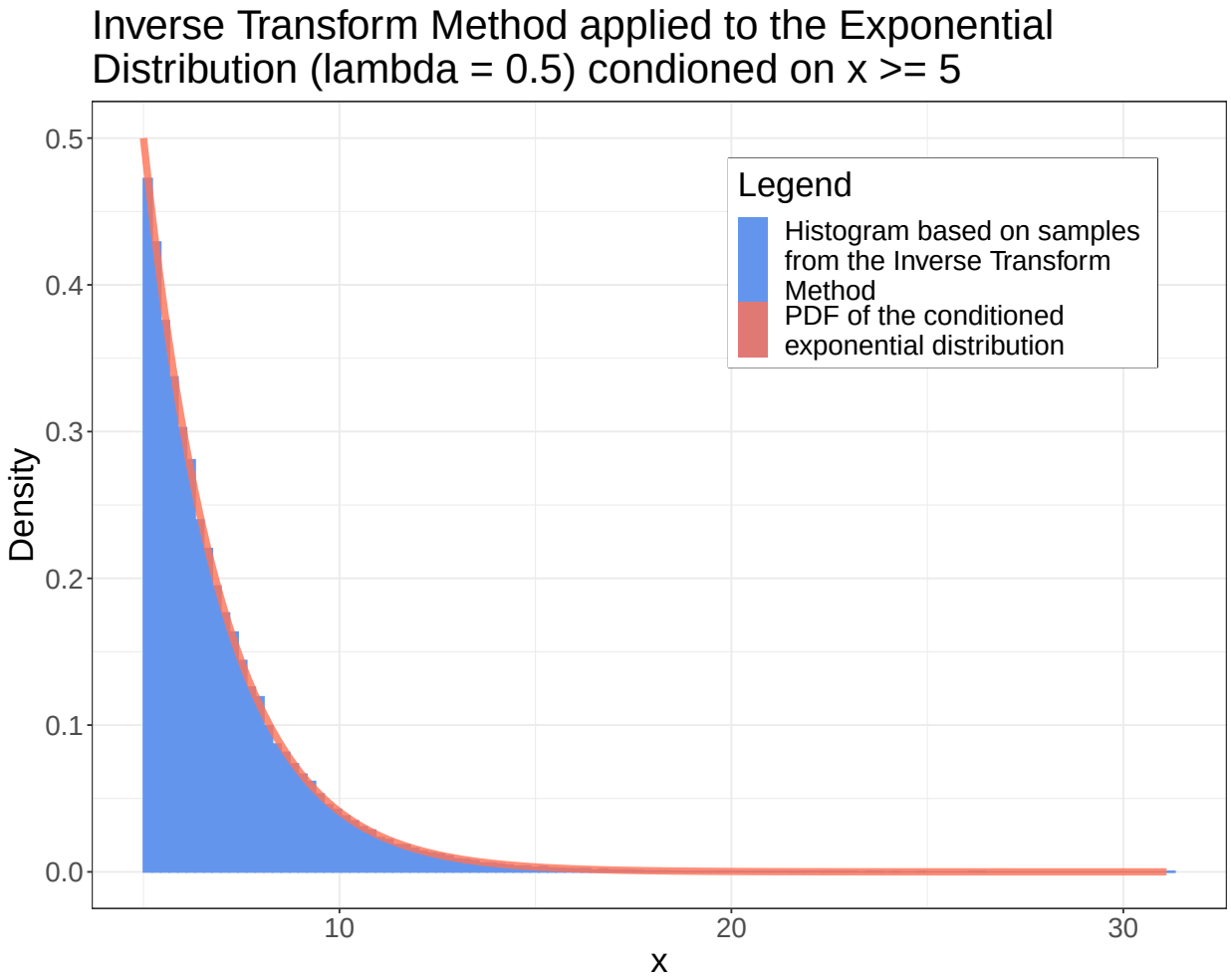

Figure 5: Application of the Inverse Transform Method to sample from a conditioned Exponential distribution

As for both the exponential and the gamma distribution of consideration for $x \geq 5$ the PDFs

```r
inverse_transform_method <- function(n, F_inv, F_min = 0, F_max = 1) {
    # Draw n samples from a distribution with CDF F
    # given its inverse F_inv.
    # Using F_min and F_max one can condition one sampling
    # only between F_inv(F_min) and F_inv(F_max).
    u <- runif(n, F_min, F_max) # draw n samples from U(0,1)
    return(F_inv(u)) # apply the inverse transform
}

F_inv_exp <- function(x, lamb = 0.5) {
    # Inverse CDF of the exponential distribution
    return(-log(1-x)/lamb)
}

f_exp_c <- function(x, lamb = 0.5, x_min = 5) {
    # PDF of the exponential distribution
    # conditioned on x >= x_min
    return(lamb * exp(-lamb * x) / exp(-lamb * x_min))
}

conditioned_exp_sample <- inverse_transform_method(100000, F_inv_exp, 1 - exp(-5/2), 1)
plot_hist_and_dist(f_exp_c, conditioned_exp_sample, "Inverse Transform Method applied to the Exponential \nDistribution (lambda = 0.5) condioned on x >= 5", "Histogram based on samples \nfrom the Inverse Transform \nMethod", "PDF of the conditioned \nexponential distribution", "exp_conditional.pdf")

```

Code-Snippet 7: Sampling from a conditioned exponential distribution

are monotonically decreasing, for C in the acceptance-rejection method, we can use

$$C = \frac{p(5)}{f(5)} = \frac{5}{3} \tag{11}$$

The results from applying the acceptance-rejection method are illustrated in figure 6 and the code is given in code-snippet 8.

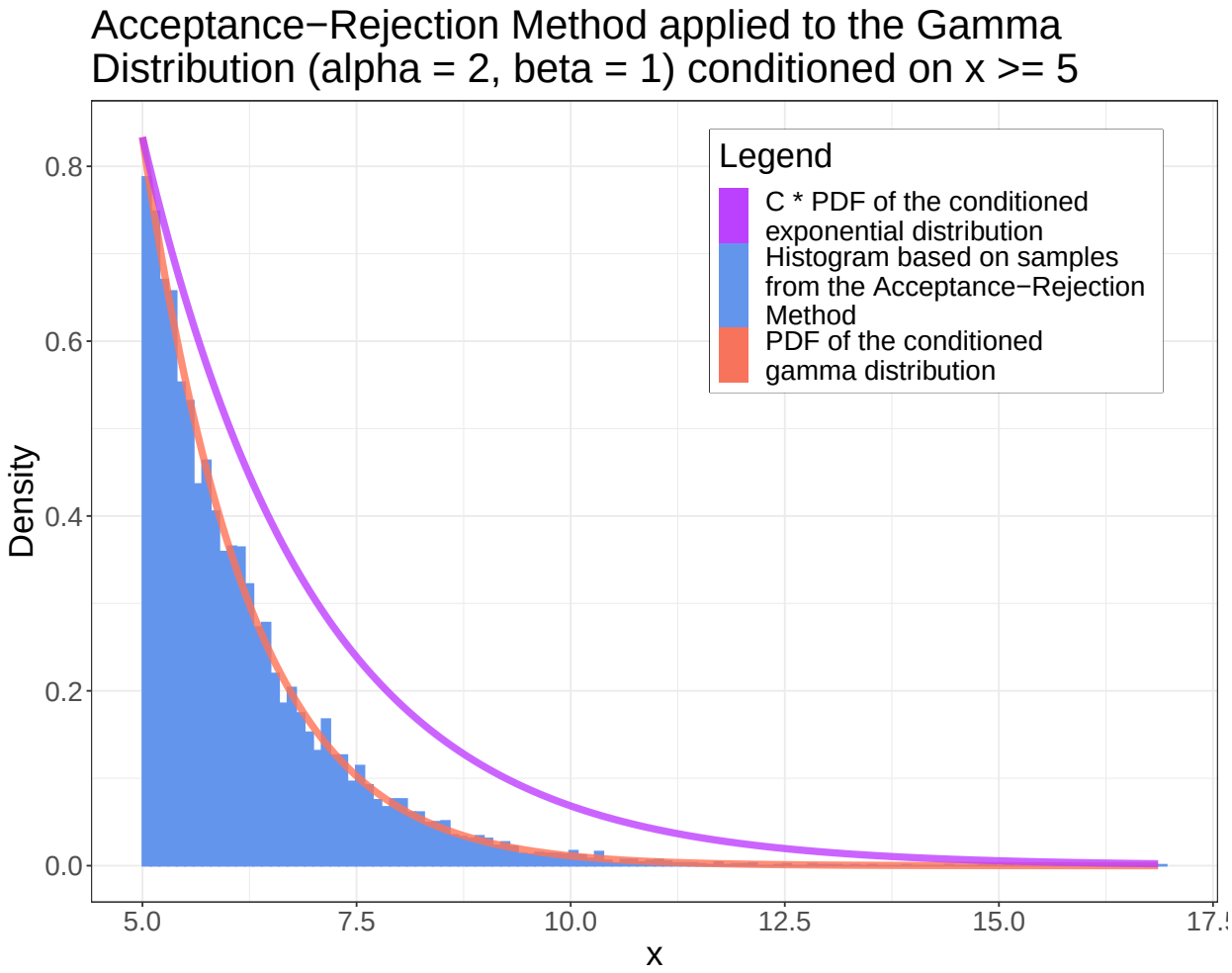

Figure 6: Application of the Acceptance-Rejection Method to sample from a conditioned Gamma distribution

```r
# acceptance-rejection method for sampling n values from a distribution
# with PDF f_to_sample using a proposal distribution with PDF bound_fun
acc_rej_meth <- function(n, f_to_sample, bound_fun, bound_sampler, c) {
    x <- numeric(n) # initialize output vector
    i <- 1
    it_count <- 0
    while (i <= n) {
        it_count <- it_count + 1
        # generate a canditate from the proposal distribution
        prop <- bound_sampler(1)
        # generate a uniform random number
        u <- runif(1) * c * bound_fun(prop)
        if (u <= f_to_sample(prop)) {
            x[i] <- prop
            i <- i + 1
        }
    }
    return(list(x = x, it_count = it_count))
}

# sampler for the conditioned exponential distribution (x>=5, lambda = 0.5)
conditioned_exp_sampler <- function(n) return(inverse_transform_method(n, F_inv_exp, 1 - exp(-5/2), 1))
    # gamma function conditioned on x >= 5
    f_gamma_c <- function(x) {
    return(x * exp(-x) / (6 * exp(-5)))
}

# sample from the conditioned gamma distribution (x>=5, alpha = 2, beta = 1)
C <- 5 / 3 # C = p(5) / f(5) as both PDFs are monotonically decreasing
sample_size <- 10000
cond_gamma_sampl_res <- acc_rej_meth(sample_size, f_gamma_c, f_exp_c, conditioned_exp_sampler, C)
cond_gamma_sampl <- cond_gamma_sampl_res$x # this is the sample, we are interested in
cond_gamma_it_count <- cond_gamma_sampl_res$it_count
print(paste0("Number of iterations: ", cond_gamma_it_count)) # e. g. 16658 for sample_size = 10000
```

Code-Snippet 8: Sampling from a conditioned gamma distribution

2 Monte Carlo Integration

2.1 A basic intuition for estimation

We generally draw information about reality from samples. Consider for instance we measure the energy deposited by energetic muons in a thin silicon layer. We might measure (for different muons of the same energy) $X = [0.5\,\text{MeV}, 0.6\,\text{MeV}, 1.3\,\text{MeV}]$ and we might say that in the mean the energy deposit might in general be 0.8 MeV (without knowing the true distribution, from which we sampled; here quite intricate, see Particle-Data-Group et al., 2020, figure 34.7).

We have just declared our sample estimator to be at least an approximation for the population parameter - which might not be a good idea, especially for such a small sample size.

2.2 Monte Carlo Estimation

Monte Carlo Estimation is concerned with estimating statistics by approximating their expectations. So for a random variable with probability distribution $f(x)$ (continuous or discrete) the expectation of a function g, so

$$\theta := E[g(X)] = \begin{cases} \sum_{k=1}^{\infty} g(x_k) \cdot f(x_k) & \text{if } X \text{ is discrete} \\ \int_{-\infty}^{\infty} g(x) \cdot f(x) dx & \text{if } X \text{ is continuous} \end{cases} \tag{12}$$

is approximated by a sample mean over $(X_1, \ldots, X_m)$ (m independent realizations from the distribution of X)

$$\hat{\theta} := \frac{1}{m} \sum_{i=1}^{m} g(X_i) \tag{13}$$

which is an unbiased estimator for θ if the expectation exists and converges almost surely to θ in the limit of a large sample size (by the strong law of large numbers).

Distribution and Error of the Monte Carlo Estimator

Assume that $\text{Var}[g(X)] = \sigma^2 < \infty$. Then by the basic properties of the variance, we can write

$$\mathrm{Var}[\hat{\theta}_m] = \frac{1}{m^2}\sum_{i=1}^{m}\mathrm{Var}[g(X_i)] = \frac{\sigma^2}{m} \tag{14}$$

Notice that for the distribution of our sample mean (an addition of m i.i.d random variables) the central limit theorem applies, so

$$\frac{\hat{\theta}_m - \theta}{\sigma/\sqrt{m}} \xrightarrow[m\to\infty]{d} Z \sim \mathcal{N}(0,1) \tag{15}$$

We would like to use the above to specify confidence intervals for our estimator $\hat{\theta}_m$. Note, however, that σ is unknown to us. Luckily, the above still holds in the limit of a large sample size when using the estimator

$$S_n^2 := \frac{1}{m-1}\sum_{i=1}^{m}(g(X_i) - \hat{\theta}_m)^2 \tag{16}$$

or rather the square root S_n. Notice that while S_n is, different from S_n^2, not an unbiased estimator, it converges in probability to σ which is sufficient to apply Slutsky's theorem yielding

$$\frac{\hat{\theta}_m - \theta}{S_n/\sqrt{m}} \xrightarrow[m\to\infty]{d} Z \sim \mathcal{N}(0,1) \tag{17}$$

2.3 Monte Carlo Integration

Classical numerical methods for integration (e.g. Simpson's rule) become infeasible for high-dimensional integrals, essentially, as the errors scale with the number of function-evaluation-points per dimension (so in high dimensions, we need to perform lots of function evaluations).

Intuition for Monte Carlo Integration: Consider we want to calculate the integral

$$I := \int_V g(\underline{x})d^d\underline{x} \tag{18}$$

where V is a d-dimensional volume and g is a function $g : V \to \mathbb{R}$. Then it makes intuitive sense to approximate this integral by the mean of the function g on the domain of interest, as found as the mean of g evaluated at N random points $\underline{x}_i$ in V, multiplied by the volume of V (also denoted by V)

$$\hat{I}_N := \frac{V}{N} \sum_{i=1}^{N} g(\underline{x}_i) \xrightarrow[N\to\infty]{a.s.} I \tag{19}$$

Connection to the Monte Carlo Estimator: We can also obtain the result from above from the Monte Carlo Estimator (eq. 12) by using the uniform distribution

$$f(\underline{x}) = \begin{cases} \frac{1}{V} & \text{if } \underline{x} \in V \\ 0 & \text{else} \end{cases} \tag{20}$$

and rewriting the integral as

$$I = \int_V g(\underline{x}) d^d\underline{x} = V \int_V g(\underline{x}) \cdot \frac{1}{V} d^d\underline{x} = V \int_V g(\underline{x}) \cdot f(\underline{x}) d^d\underline{x} \tag{21}$$

Error in Monte Carlo Integration - scaling with $\frac{1}{\sqrt{N}}$: Based on the connection to the Monte Carlo Estimator, the variance of our estimator for the integral is given by

$$\mathrm{Var}[\hat{I}_N] = \frac{V^2}{N} \mathrm{Var}[g(x)], \quad \text{estimate } \mathrm{Var}[g(x)] \text{ by } S^2_{g;N} := \frac{1}{N-1} \sum_{i=1}^{N} (g(x_i) - \hat{I}_N)^2 \tag{22}$$

so the standard deviation scales with $1/\sqrt{N}$ (independent of dimensionality) and is thus favorable over classical numerical methods in high dimensions.

Illustrative Example: Consider the integral

$$I = \int_0^1 g(x) dx, \quad g(x) = \exp\left(-\frac{x^2}{2}\right) \tag{23}$$

Our Monte Carlo estimator for this integral is given by

$$\hat{I}_N = \frac{1}{N} \sum_{i=1}^{N} g(x_i), \quad x_i \sim \mathcal{U}(0,1) \tag{24}$$

which is illustrated in figure 7. The higher the number of sampled points N the lower the variance of our estimator; the lower the variance of g along the y axis, the lower the variance of our estimator.

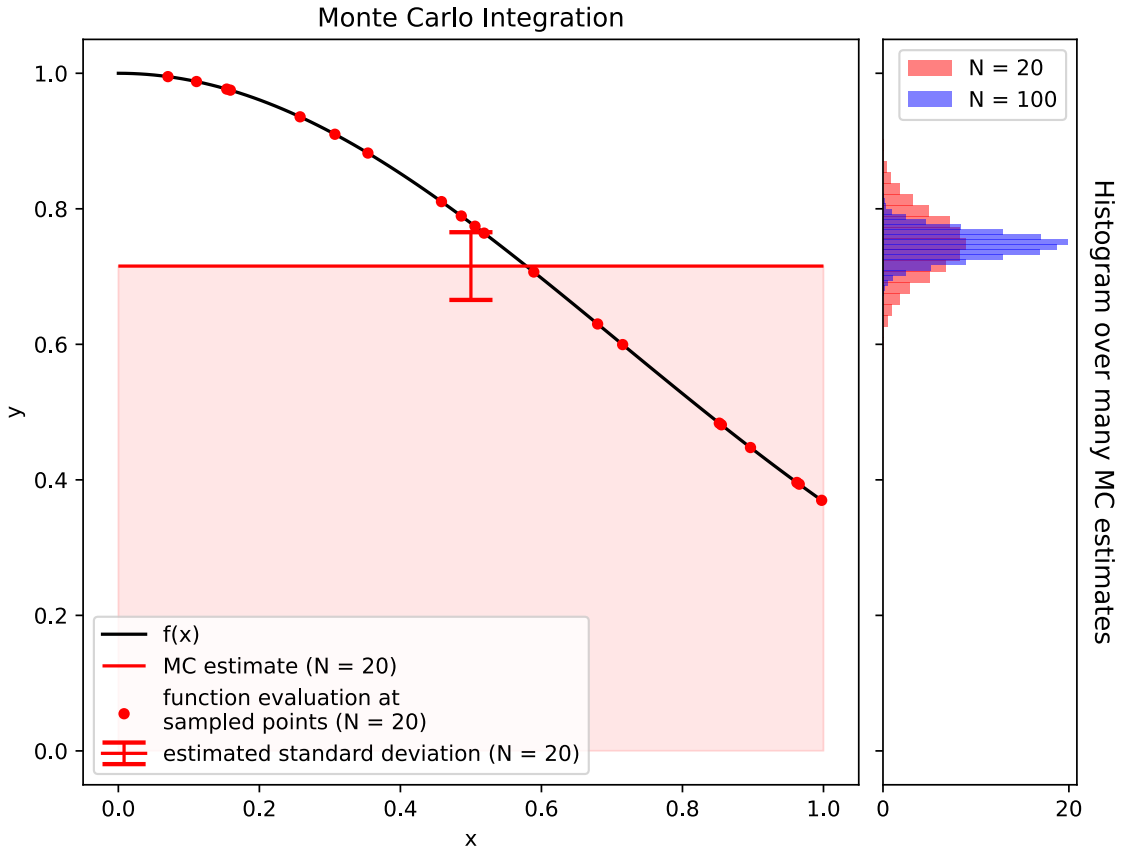

Figure 7: Illustration of Monte Carlo Integration for $I = \int_0^1 \exp\left(-\frac{x^2}{2}\right) dx$

2.4 Reducing the variance of the Monte Carlo Estimator

2.4.1 Antithetic Estimators (task 4)

Two random variables with the same distribution on the same probability space are called *antithetic*, if their covariance is negative. We can use such antithetic variables to reduce the variance of a Monte Carlo estimator.

This motivates the following ansatz

$$\begin{aligned} I &:= \int_0^1 g(x)dx, \quad \hat{I}_{anti} = \frac{\hat{I}_{N/2} + \tilde{I}_{N/2}}{2} \\ \hat{I}_{N/2} &= \frac{1}{N/2}\sum_{i=1}^{N/2} g(u_i), \quad \tilde{I}_{N/2} = \frac{1}{N/2}\sum_{i=1}^{N/2} g(1-u_i), \quad u_i \sim \mathcal{U}(0,1) \end{aligned} \tag{25}$$

with

$$\mathrm{Var}[\hat{I}_{anti}] = \frac{1}{4}\left(\mathrm{Var}[\hat{I}_{N/2}] + \mathrm{Var}[\tilde{I}_{N/2}] + 2\,\mathrm{Cov}[\hat{I}_{N/2}, \tilde{I}_{N/2}]\right) \underbrace{=}_{\text{see eq. 22}} \mathrm{Var}[\hat{I}_N] - \frac{1}{2}\,\mathrm{Cov}[\hat{I}_{N/2}, \tilde{I}_{N/2}] \tag{26}$$

where for g continuous and monotonic, $\mathrm{Cov}[\hat{I}_{N/2}, \tilde{I}_{N/2}]$ is negative, as U and $1-U$ with $U \sim \mathcal{U}(0,1)$ are negatively correlated. So we can reduce the variance compared to an estimator $\hat{I}_N$ using the same number of function evaluation and only half the number of random numbers.

Let us give an intuition why using antithetic evaluation points makes sense for estimating an integral of a monotonic function from 0 to 1. Using uniform antithetic variables we create pairs of points $(x_i, 1-x_i)$, which are symmetric around $x = 0.5$ and as of the monotony of the function g we want to integrate, we obtain a better balance between sampling points where g is large and where g is small, reducing the variance of the estimate.

Example of using Antithetic Estimators (Task 4)

Consider the integral

$$I = \int_0^2 \frac{1}{1+x^2} dx \underbrace{=}_{\text{exact solution}} \arctan(2) \approx 1.107149 \tag{27}$$

Estimates based on the standard Monte Carlo estimator and the antithetic estimator are given in table 1 and illustrated in figure 8. A relative reduction in standard deviation of roughly 80% is achieved using the antithetic estimator. The code for this task is given in code-snippet 10 and 9.

Estimator	Estimate	Standard Deviation	95% Confidence Interval
Standard Monte Carlo	1.096	0.017	[1.0627 1.128]
Antithetic Monte Carlo	1.106	0.0033	[1.099 1.112]

Table 1: Rounded estimates for the integral $\int_0^2 \frac{1}{1+x^2} dx$ using the standard Monte Carlo estimator and the antithetic Monte Carlo estimator for $N = 100$ samples with seed 1234

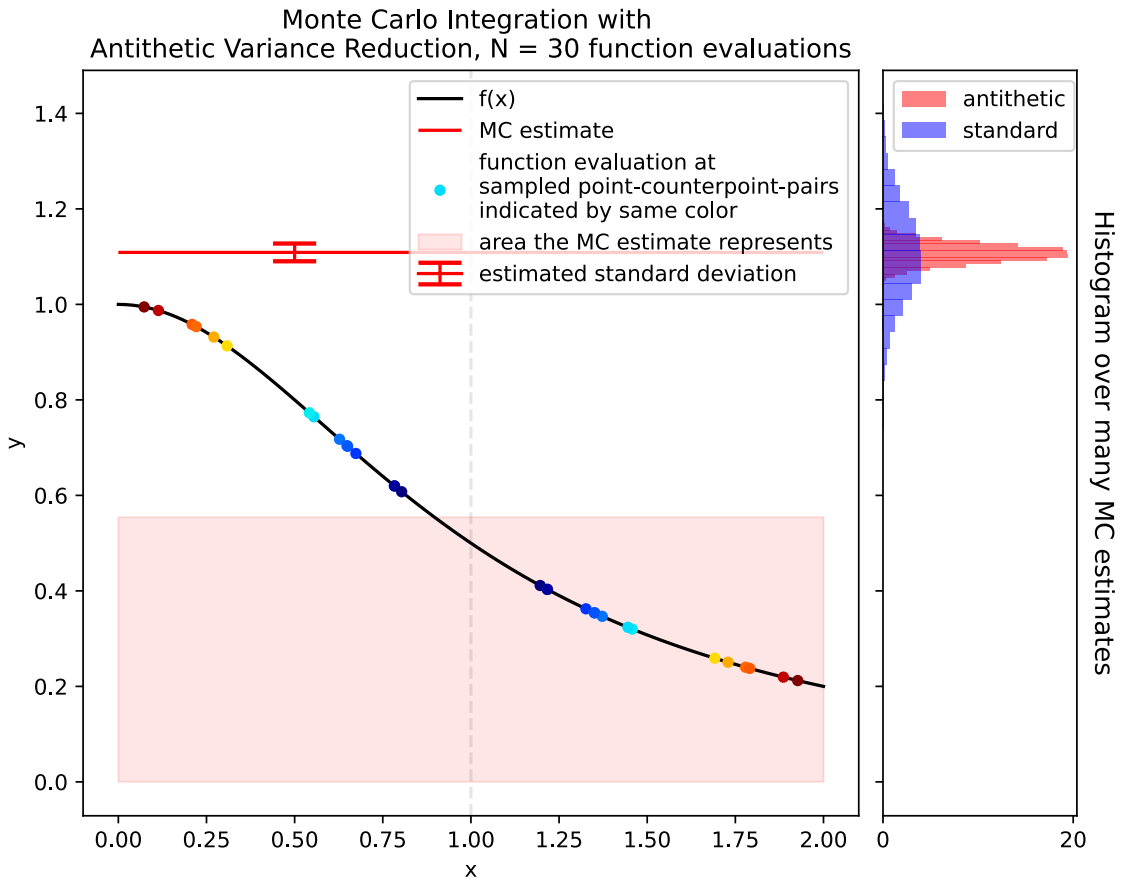

Figure 8: Illustration of Monte Carlo Integration with Antithetic Variables. Notice the even distribution of points around $\frac{a+b}{2} = 1$ by construction yielding an advantage over the standard Monte Carlo estimator.

```r
# Simple Monte Carlo Integration
mc <- function(f, a, b, n, alpha = 0.05) {
    # Draw n samples from a uniform distribution on [a,b]
    x <- runif(n, a, b)
    # Do the estimate
    est <- mean(f(x)) * (b - a)
    # Calculate the estimate of the standard deviation
    standard_error <- sd(f(x)) * (b - a) / sqrt(n)
    # Confidence interval estimate based on the normal approximation
    z <- qnorm(1 - alpha / 2)
return(c(est, standard_error, est - z * standard_error, est + z *
 ↪  standard_error))
}

# Monte Carlo Integration with Antithetic Variables
mc_anti <- function(f, a, b, n, alpha = 0.05) {
    # Draw n samples from a uniform distribution
    x <- runif(n / 2, 0, 1)
    # compute necessary evaluations; generalized to arbitrary a, b
    evals <- 0.5 * (f(a + x * (b - a)) + f(b - x * (b - a))) * (b -
     ↪  a)
    # Do the estimate
    est <- mean(evals)
    # Calculate the estimate of the standard deviation
    standard_error <- sd(evals) / sqrt(n / 2)
    # Confidence interval estimate based on the normal approximation
    z <- qnorm(1 - alpha / 2)
    return(c(est, standard_error, est - z * standard_error, est + z
     ↪  * standard_error))
}
```

Code-Snippet 9: Functions written for task 4

```r
# ######## Task 4 ##########
# Function to integrate
f <- function(x) 1 / (1 + x^2)
# Number of samples used
N <- 1000

# a) Standard Monte Carlo
res_stand <- mc(f, 0, 2, N)
print(res_stand)
# roughly:
# estimate:            1.096
# standard deviation:  0.017
# confidence interval: [1.06, 1.13]

# b) Anti-thetic Variables
res_anti <- mc_anti(f, 0, 2, N)
print(res_anti) # 1.105492851 0.003340705 1.098945189 1.112040513
# roughly:
# estimate:            1.105
# standard deviation:  0.003
# confidence interval: [1.099, 1.112]

# Relative reduction in standard deviation
red <- (res_stand[2] - res_anti[2]) / res_stand[2]
print(red) # roughly 80% reduction
```

Code-Snippet 10: Code for performing Task 4; relative reduction in standard deviation is roughly 80%

2.4.2 Importance Sampling (task 5)

The higher the variance of the output of the function $g(x)$ over the domain of integration, the higher the variance in our Monte Carlo Estimate for our integral.

$$I := \int_V g(x)dx = \int_V \frac{g(x)}{f(x)} f(x)dx, \quad I_N = \frac{1}{N}\sum_{i=1}^{N} \frac{g(x_i)}{f(x_i)} \tag{28}$$
$$\text{sample } \{x_i\}_{i=1,\dots,N} \text{ drawn from } f(x) \text{ limited to } V$$

Let us as an example tackle the integral

$$I = \int_0^1 \frac{4}{1+x^2}dx \underbrace{=}_{\text{exact solution}} \pi \tag{29}$$

using the importance sampling function $f(x) = \frac{4-2x}{3}$ on $0 \leq x \leq 1$. We can sample from $f(x)$ using the inverse transform method using the CDF $F(x) = \frac{4x-x^2}{3}$ (with $F(1) = 1$ as we want) so $F^{-1}(u) = 2 - \sqrt{4-3u}$. We can then simply use equation 28 to obtain an estimate.

The importance sampling approach compared to the standard Monte Carlo approach is illustrated in figure 9 and 10. There it is easy to see how effectively integrating a flatter function is to our advantage. An implementation in R is given in code-snippet 11 with task 5 carried out in code-snippet 12, where using importance sampling led to a relative reduction in standard deviation of roughly 88%.

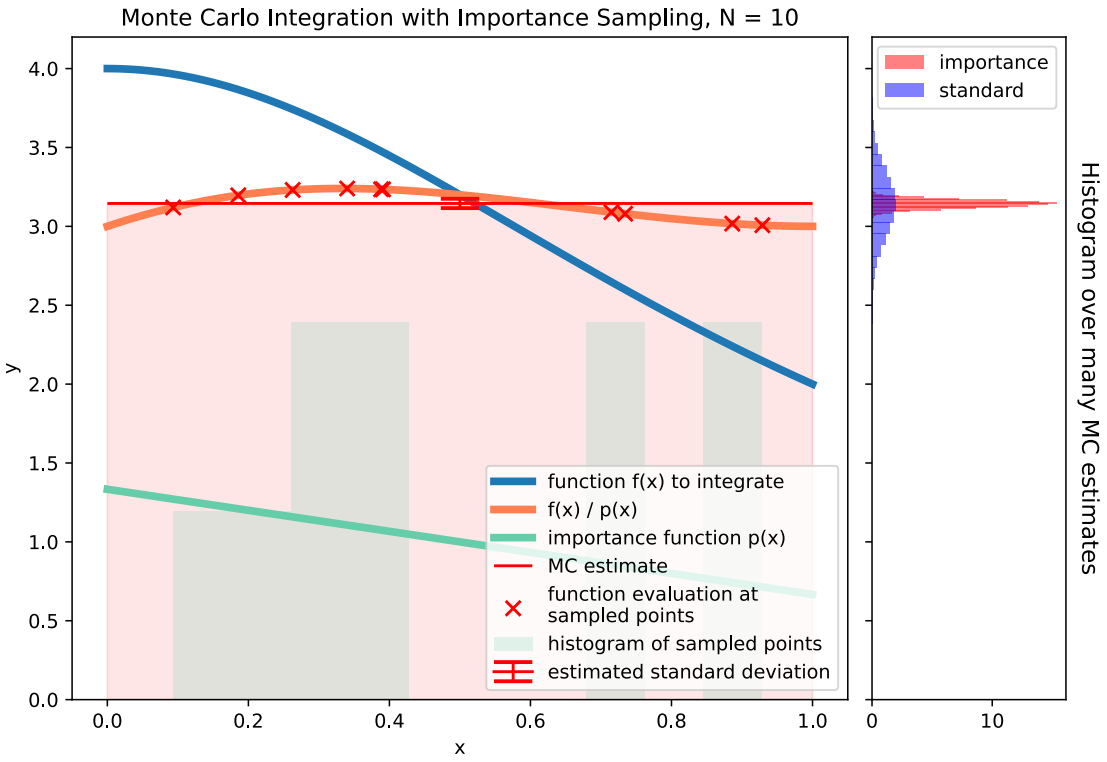

Figure 9: Illustration of Monte Carlo Integration with Importance Sampling for $N = 10$ samples.

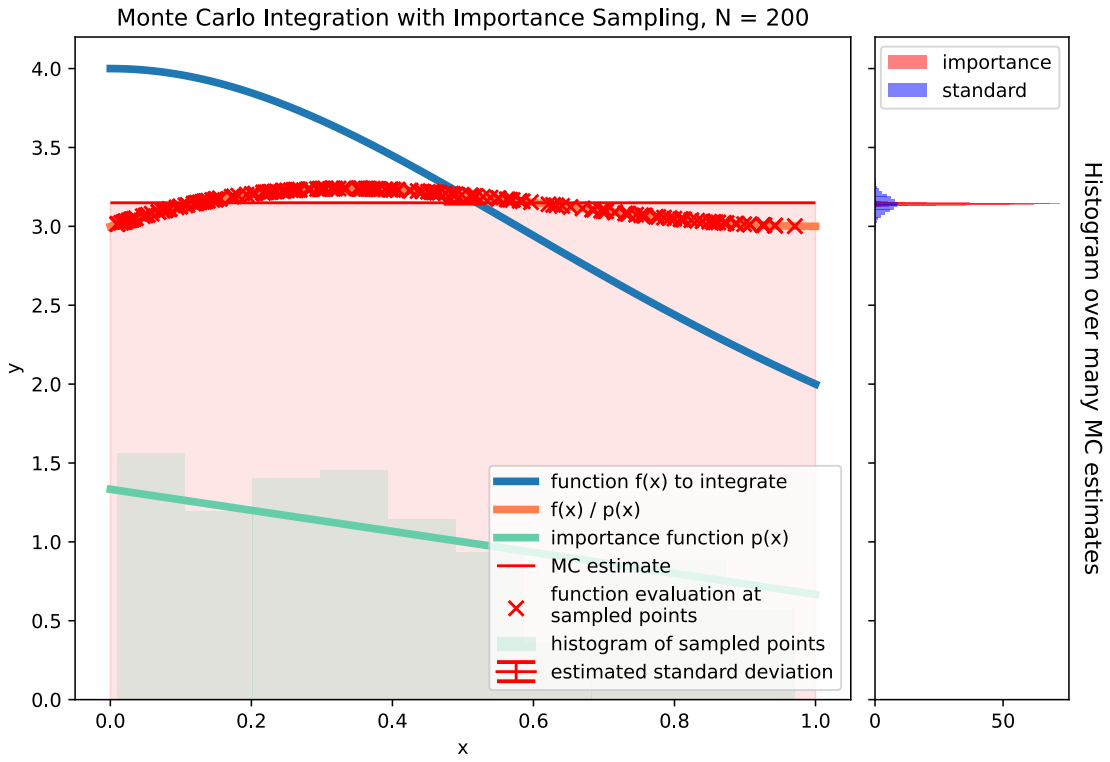

Figure 10: Illustration of Monte Carlo Integration with Importance Sampling for $N = 200$ samples.

```r
set.seed(1234)

# General Monte Carlo Integration
mc_gen_conf <- function(f, p, gen_from_p, n, alpha = 0.05) {
    # get samples from a distribution p
    x <- gen_from_p(n)
    # Evaluations for our importance sampling estimate
    vals <- f(x) / p(x)
    # Do the estimate
    est <- mean(vals)
    # Calculate the estimate of the standard deviation
    sigman <- sd(vals) / sqrt(n)
    # Confidence interval estimate
    z <- qnorm(1 - alpha / 2)
    return(c(est, sigman, est - z * sigman, est + z * sigman))
}

# Simple Monte Carlo Integration
mc <- function(f, a, b, n) {
    x <- runif(n, a, b)
    est <- mean(f(x)) * (b - a)
    standard_error <- sd(f(x)) * (b - a) / sqrt(n)
    return(c(est, standard_error))
}

inverse_transform_method <- function(n, F_inv, F_min = 0, F_max = 1)
 ↪  {
    # Draw n samples from a distribution with CDF F
    # given its inverse F_inv.
    # Using F_min and F_max one can condition one sampling
    # only between F_inv(F_min) and F_inv(F_max).
    u <- runif(n, F_min, F_max) # draw n samples from U(0,1)
    return(F_inv(u)) # apply the inverse transform
}
```

Code-Snippet 11: Code for performing Monte Carlo Integration with Importance Sampling

```r
########## Task 5 ##########
# Function to integrate
f <- function(x) 4 / (1 + x^2)
# Number of samples used
N <- 1000

# a) Standard Monte Carlo
res_stand <- mc(f, 0, 1, N)
print(res_stand[1]) # 3.125621
sigma_stand <- res_stand[2]
print(sigma_stand) # 0.02054352

# b) Importance Sampling
# Importance Sampling Function
p <- function(x) (4-2*x) / 3
# Inverse of the CDF of the importance function
F_inv <- function(u) 2 - sqrt(4 - 3 * u)
# Sampler based on the inverse transform method
inverse_sampler <- function(n) inverse_transform_method(n, F_inv)
# Get importance sampling result
res_imp <- mc_gen_conf(f, p, inverse_sampler, N)
print(res_imp[1]) # 3.146345
sigma_imp <- res_imp[2]
print(sigma_imp) # 0.002463229

# Relative reduction in standard deviation
red <- (sigma_stand - sigma_imp) / sigma_stand
print(red) # roughly 88% reduction
```

Code-Snippet 12: Code for performing Task 5; relative reduction in standard deviation is roughly 88%

3 Testing Skewness (task 6)

Consider a random variable X with mean $E[x] = \mu$. The skewness is defined as

$$\gamma_1 = \frac{E[(X-\mu)^3]}{(\mathrm{Var}[X])^3} \tag{30}$$

where for $\gamma_1 = 0$ the distribution is said to be symmetric. For γ_1 we use the estimator

$$\hat{b}_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)^{3/2}}, \qquad \text{sample } \{X_i\}_{i=1,\dots,n} \tag{31}$$

We test the null hypothesis $H_0 : \gamma_1 = 0$ against the alternative $H_1 : \gamma_1 \neq 0$.

For $X \sim \mathcal{N}(\mu, \sigma^2)$ we make two approximations for the distribution of $\hat{b}_1$:

(a) For n large, $\hat{b}_1 \sim \mathcal{N}(0, 6/n)$.

(b) For n large, $\hat{b}_1 \sim \mathcal{N}(0, \frac{6(n-2)}{(n+1)(n+3)})$.

and we use sample sizes $n \in \{10, 20, 30, 50, 100, 500\}$.

Let us start by comparing those distributions to empirical distributions of the test statistic for some of the given sample sizes, see figure 11. The (a) approximation has a higher variance than the empirical distribution, so the rejection regions are farther out, which will lead to underestimating the rejection rate. As the sample size increases, both approximations get closer to the empirical distribution, so we expect that the rejection rates will (in the mean) get closer to the nominal level $\alpha = 0.05$.

The results for the rejection rates are shown in figure 12 and 13. As expected, in figure 12 we see that the rejection rates are underestimated for low sample sizes, but get closer to the nominal level as the sample size increases and the approximated distribution becomes a closer fit to the empirical one. In figure 13 we see that the (b) approximation is a better fit to the empirical distribution, so the rejection rates are closer to the nominal level for smaller sample sizes.

Note that each check for rejection is just a Bernoulli trial, so the rejection rate has the standard error

$$\hat{\mathrm{SE}} = \sqrt{\frac{\hat{\alpha}(1-\hat{\alpha})}{m}}, \quad m = \#\text{ independent hypothesis tests carried out}, \quad \hat{\alpha} = \frac{\#\text{ rejections}}{m} \tag{32}$$

where this standard error is monotonically increasing for $\hat{\alpha} \in [0, 1/2]$. This explains why in figure 12 the (wrongly) low estimates at smaller sample sizes stray less around their mean.

```r
type1rate <- function(statistic, sampler, statistic_quantile_func,
    sample_size, simulation_size, alpha) {
    # Calculate the type 1 error rate of a two-sided test
    number_of_rejections <- 0 # count the number of rejections
    reject_at_abs_higher_than <- statistic_quantile_func(1 - alpha /
        2, sample_size) # reject test if statistic is higher than
        this value
    # do the simulations
    for (i in 1:simulation_size) {
        # sample from the given distribution
        sample <- sampler(sample_size)
        # evaluate the test statistic
        statistic_value <- statistic(sample)
        # check for rejection
        if (abs(statistic_value) > reject_at_abs_higher_than) {
            number_of_rejections <- number_of_rejections + 1
        }
    }
    return(number_of_rejections / simulation_size)
}
```

Code-Snippet 13: Estimation of type I error rates

The code for general estimation of type I error rates is given in code-snippet 13, the functions for the skewness test are given in code-snippet 14, the specific code to reproduce our skewness tests is given in code-snippet 15 and finally the code for the plots is given in code-snippet 16.

```r
skewness_stat <- function(sample) {
    # Calculate the skewness of a sample
    n <- length(sample)
    mean <- mean(sample)
    sd <- sd(sample) * ((n - 1) / n) ^ 0.5 # use biased estimator
    ↪ for sd
    return((1 / n) * sum((sample - mean) ^ 3) / sd ^ 3)
}

skewness_stat_quantA <- function(eval_point, sample_size) {
    # Calculate the quantile of the skewness statistic
    # using the normal approximation
    return(qnorm(eval_point, mean = 0, sd = (6 / sample_size) ^
    ↪ 0.5))
}

skewness_stat_quantB <- function(eval_point, sample_size) {
    # Calculate the quantile of the skewness statistic
    # using the normal approximation
    n <- sample_size # for brevity
    return(qnorm(eval_point, mean = 0, sd = (6 * (n - 2) / ((n + 1)
    ↪ * (n + 3)))^0.5))
}
```

Code-Snippet 14: Functions used in the skewness test

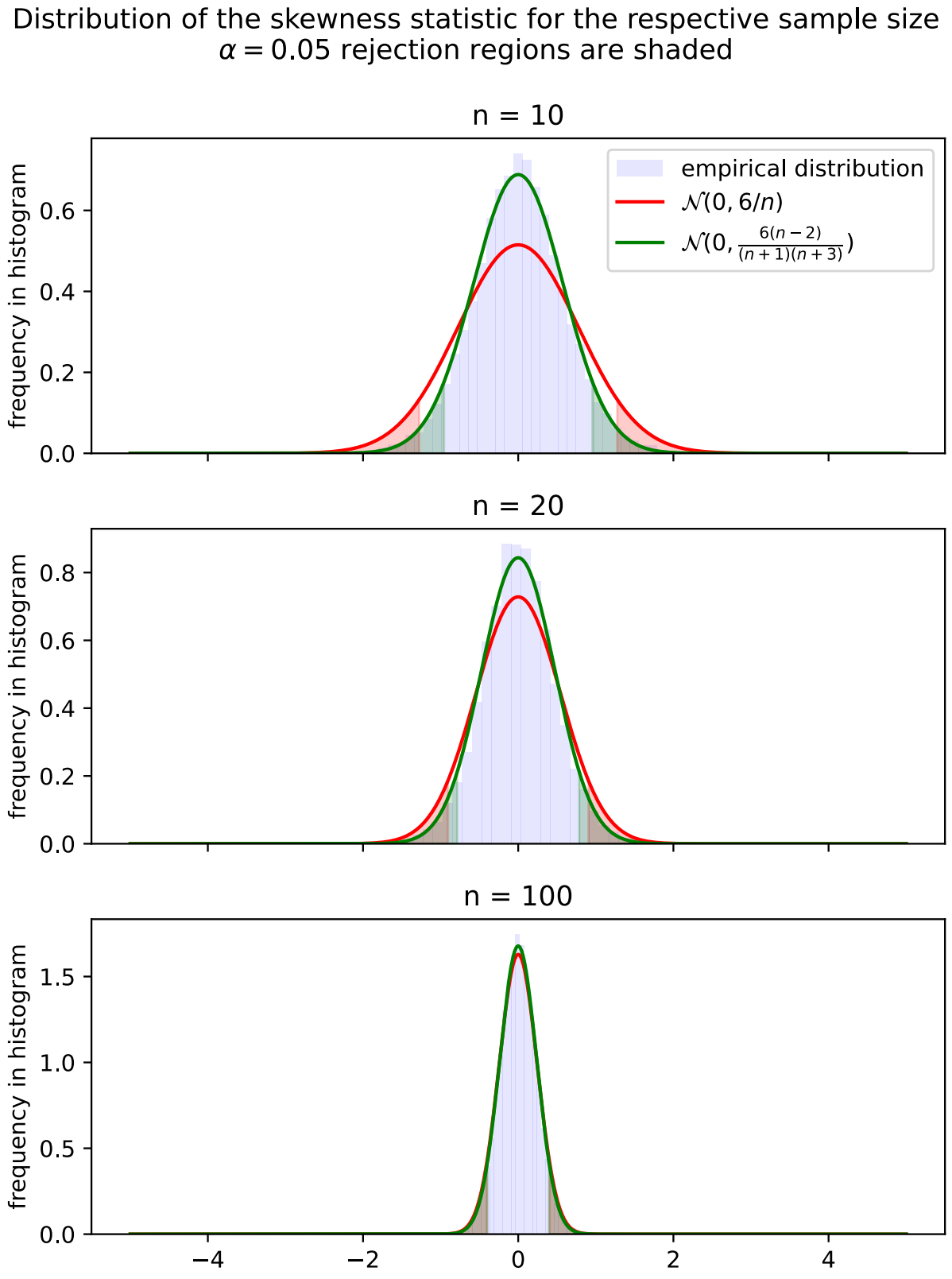

Figure 11: Empirical distribution of the test statistic $\hat{b}_1$ for different sample sizes n and the two approximations.

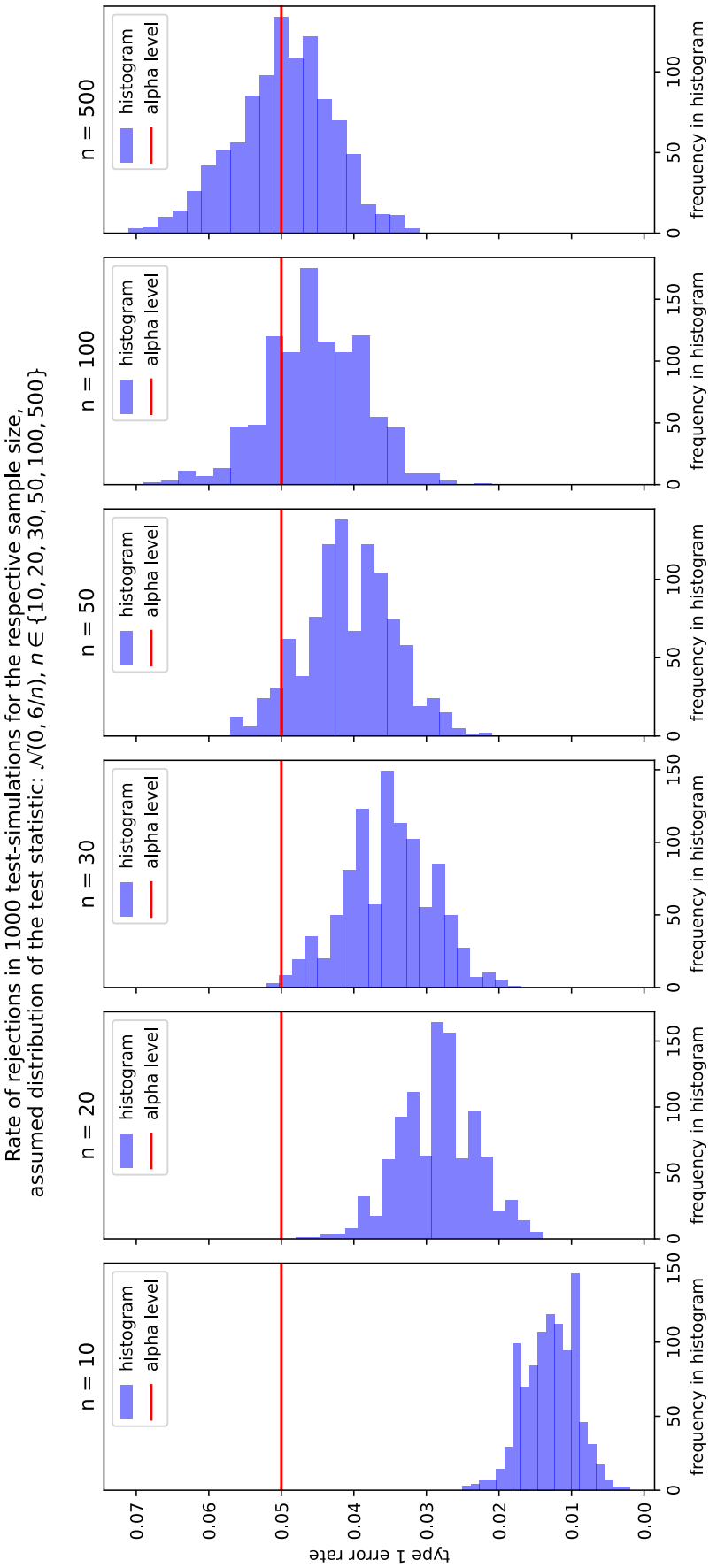

Figure 12: Rejection rates for approximation (a) for different sample sizes n.

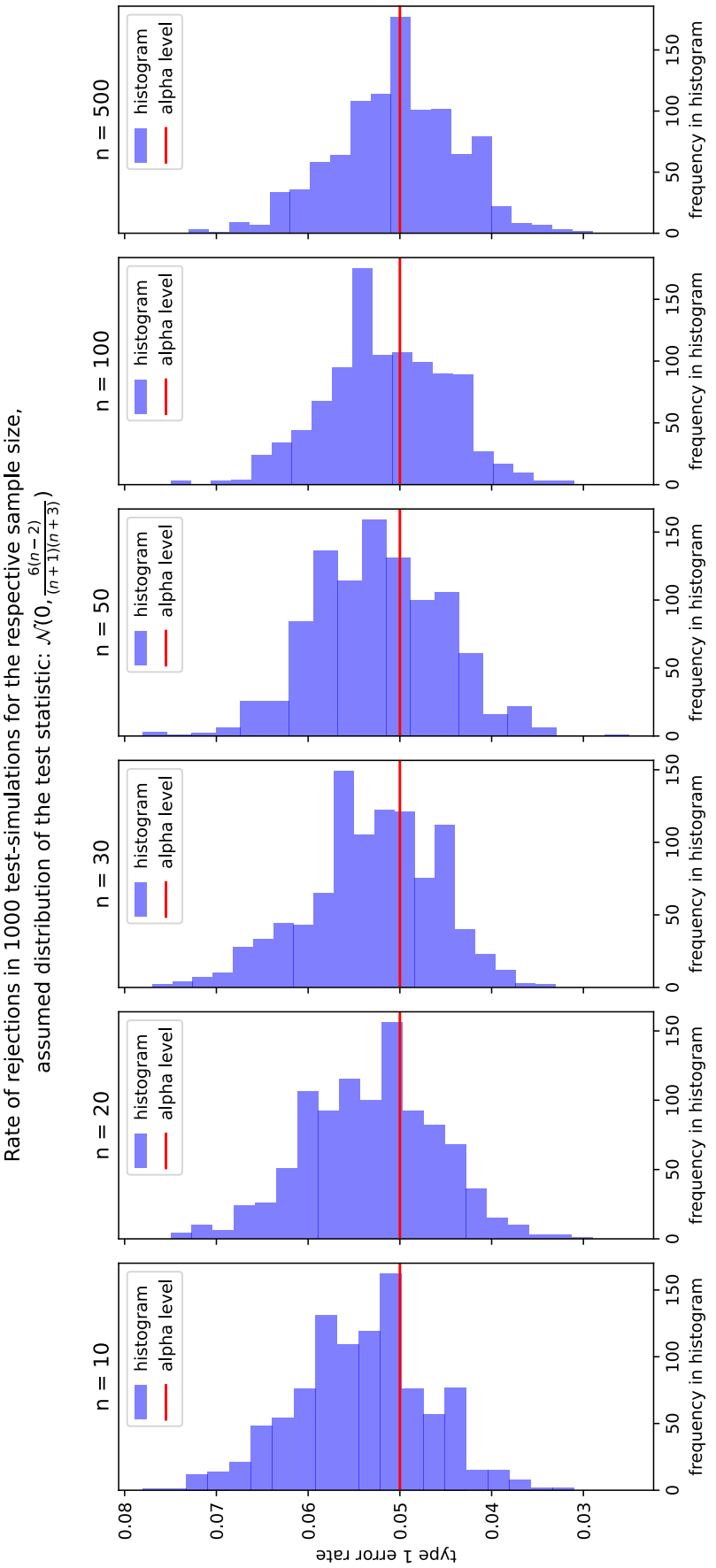

Figure 13: Rejection rates for approximation (b) for different sample sizes n.

```
    # a) Type I error rate for the skewness test
    # for different sample sizes
    sample_sizes <- c(10, 20, 30, 50, 100, 500)
    simulation_size <- 1000
    alpha <- 0.05
    # we sample from a standard normal distribution
    sampler <- function(sample_size) {rnorm(sample_size)}
    stat_quant <- skewness_stat_quantA
    # get multiple estimates of the rejection rate
    # for each sample size
    num_simulations <- 1000
    error_rate_sets <- list()
    for (i in seq_along(sample_sizes)) {
        error_rates <- replicate(num_simulations,
        ↪  type1rate(skewness_stat, sampler, stat_quant,
        ↪  sample_sizes[i], simulation_size, alpha))
        error_rate_sets[[i]] <- error_rates
    }
    # use reticulate to save the error rates to a numpy array
    # for further plotting
    np$save("error_ratesA.npy", error_rate_sets)

    # b) Type I error rate for the skewness test
    # better approximation of the test statistic distribution
    stat_quant <- skewness_stat_quantB
    error_rate_sets <- list()
    for (i in seq_along(sample_sizes)) {
        error_rates <- replicate(num_simulations,
        ↪  type1rate(skewness_stat, sampler, stat_quant,
        ↪  sample_sizes[i], simulation_size, alpha))
        error_rate_sets[[i]] <- error_rates
    }
    np$save("error_ratesB.npy", error_rate_sets)
```

Code-Snippet 15: Skewness test of the task

```python
    def plot_vert_hist(error_rate_sets, sample_sizes, simulation_size,
    ↪  alpha, assumed_distribution, fname):
        # create a figure with a grid of subplots
        fig, axes = plt.subplots(nrows=1, ncols=len(sample_sizes),
        ↪  sharey=True, figsize=(13, 6))
        fig.suptitle("Rate of rejections in " + str(simulation_size) + "
        ↪  test-simulations for the respective sample size,\n assumed
        ↪  distribution of the test statistic: " +
        ↪  assumed_distribution)

        # iterate over the axes and the error_rate_sets
        for i, ax in enumerate(axes):
            ax.hist(error_rate_sets[i], bins=20,
            ↪  orientation='horizontal', color='b', alpha=0.5, label =
            ↪  "histogram")
            ax.set_title("n = " + str(sample_sizes[i]))
            ax.set_xlabel("frequency in histogram")
            if i == 0:
                ax.set_ylabel("type 1 error rate")
            ax.axhline(y=alpha, color='r', linestyle='-', label = "alpha
            ↪  level")
            ax.legend()

        # show the figure
        plt.tight_layout()
        plt.savefig(fname)
```

Code-Snippet 16: Core code for the plots; doing this in R was tried with considerable effort (ggplot2, plotly, ...) but satisfactory results could not be obtained.

References

Box, G. E. P. and Mervin E. Muller (1958). »A Note on the Generation of Random Normal Deviates«. In: *The Annals of Mathematical Statistics* 29.2, pages 610–611. DOI: 10.1214/aoms/1177706645. URL: https://doi.org/10.1214/aoms/1177706645.

Matsumoto, Makoto and Takuji Nishimura (1998). »Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator«. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1, pages 3–30.

Particle-Data-Group et al. (Aug. 2020). »Review of Particle Physics«. In: *Progress of Theoretical and Experimental Physics* 2020.8, page 083C01. DOI: 10.1093/ptep/ptaa104. URL: https://doi.org/10.1093/ptep/ptaa104.

Press, William H. et al. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd edition. USA: Cambridge University Press.