```
using Statistics
```

```
using CairoMakie
```

```
using Random
```

```
using SpecialFunctions
```

```
using Distributions
```

Note: This is a Pluto.jl notebook. The original notebook is available on my website at thomasmarks.space/notebooks/variance_reduction.jl. A PDF version is also available.

Variance reduction for Monte Carlo integration

One of the most common uses of Monte Carlo methods is computing integrals, especially high-dimensional ones. Let I be integral of a function $f(x)$ over some domain $\Omega \subseteq \mathbb{R}^d$, where d is the dimension:

$$I = \int_\Omega f(x) dx.$$

We can approximate this integral by drawing uniform samples from the domain Ω and averaging the value of $f(x)$, evaluated at those samples and multiplying by the volume of the domain $V(\Omega)$.

$$\hat{I} = \frac{V(\Omega)}{N} \sum_{i=1}^{N} f(x_i) \quad x_i \sim \mathcal{U}(\Omega)$$

The advantage of this technique that the error in this approximation only depends on the number of samples N (the error scales as $1/\sqrt{N}$) and not the dimension of the input space. This makes it very handy for certain problems. However, the convergence rate can be quite slow for practical problems, leading to quite noisy estimates. To improve the convergence rate, we can apply a number of different techniques, which I investigate in this notebook.

1. Quasi-Monte Carlo methods

These are methods in which we employ *low-discrepancy sequences* as samples from uniform distributions, instead of psuedo-random or truly-random sequences. This can significantly improve the convergence rate of a Monte Carlo estimator. In this section, I implement a Halton sequence generator. This is a very popular and easy-to-implement low-discrepancy sequence. I then demonstrate its performance sampling from uniform and normal distributions, as well as from a 2-D

uniform distribution on a circle.

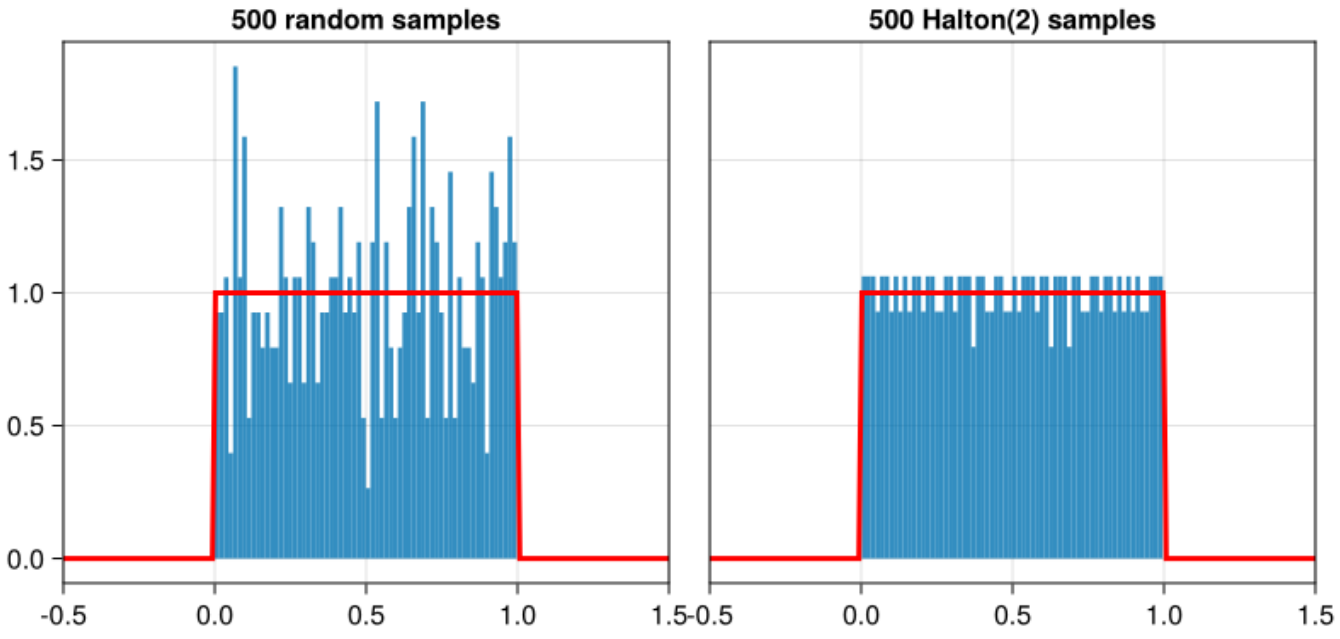
```
begin

"""
Halton sequences are deterministic sequences of numbers, parameterized by a base.
For good results, these bases should be prime. The classic 'Halton Sequence' is the
Halton (2) sequence. To generate random numbers in multiple dimensions, you create a
Halton sequence for each dimension, each with its own distinct prime base.
The 'state' of the generator is stored in a numerator `n` and a denominator `d`.
"""
mutable struct HaltonGenerator
    base::Int
    n::Int
    d::Int
end

HaltonGenerator(base) = HaltonGenerator(base, 0, 1)

"""
We overload `Base.rand` for a generating the next element in the Halton sequence.
The implementation derives from [Wikipedia](https://en.wikipedia.org/wiki/
Halton_sequence).
"""
function Base.rand(seq::HaltonGenerator)
    (;base, n, d) = seq
    x = d - n
    if x == 1
        n = 1
        d *= base
    else
        y = d ÷ base
        while x <= y
            y ÷= base
        end
        n = (base + 1) * y - x
    end
    seq.n = n
    seq.d = d
    return n / d
end

"""
Generate the next `N` numbers in the given Halton sequence
"""
function Base.rand(seq::HaltonGenerator, N::Integer)
    vals = zeros(N)
    for i in eachindex(vals)
        vals[i] = rand(seq)
    end
    return vals
end

nothing

end
```

We can check how the Halton sequence performs on sampling numbers from a uniform distribution

on [0, 1]. Here, we draw 500 samples from both a standard pseudorandom generator as well as from a Halton sequence.

```
sample_uniform_halton (generic function with 1 method)
```

```
function sample_uniform_halton(N; b = 2)
    random_samples = rand(N)
    halton_samples = rand(HaltonGenerator(b), N)

    xs = LinRange(-0.5, 1.5, 200)
    p(x) = 0 ≤ x ≤ 1 ? 1.0 : 0.0
    ps = p.(xs)
    bins = round(Int, sqrt(N)) * 3

    # Plotting
    linewidth = 3
    color = :red

    f = Figure(; resolution = (800, 400))
    ax1 = Axis(f[1,1], title = "$(N) random samples")
    hist!(ax1, random_samples; bins, normalization = :pdf)
    lines!(ax1, xs, ps; linewidth, color)

    ax2 = Axis(f[1,2], title = "$(N) Halton($b) samples")
    xlims!(ax2, (-0.5, 1.5))
    hist!(ax2, halton_samples; bins, normalization = :pdf)
    lines!(ax2, xs, ps; linewidth, color)
    hideydecorations!(ax2, grid=false)

    linkyaxes!(ax1, ax2)
    linkxaxes!(ax1, ax2)

    f
end
```




```
sample_uniform_halton(500)
```

We can see that the samples generated from the Halton sequence are significantly more uniform

than the classic pseudo-random numbers. We can do the same thing with a Gaussian (Normal) distribution)

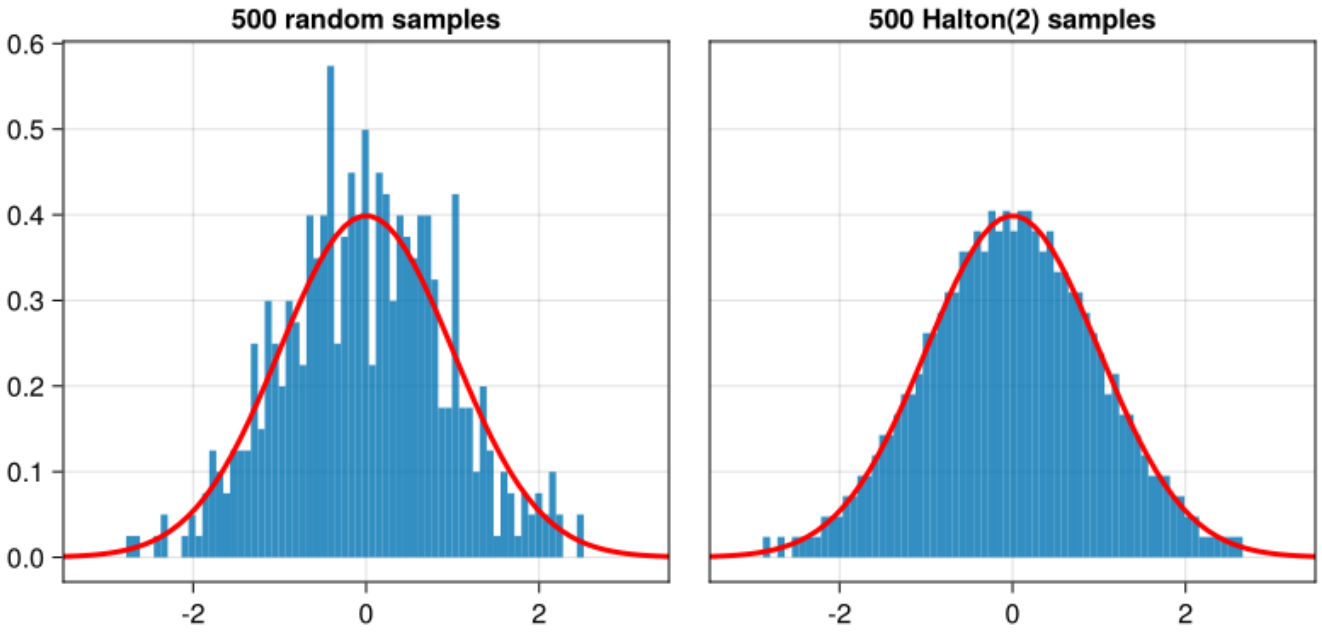
sample_normal

Given a generator which can draw from a uniform distribution, generate samples from a normal distribution with mean 0 and std 1.

```
"""
Given a generator which can draw from a uniform distribution, generate samples from a normal distribution with mean 0 and std 1.
"""
function sample_normal(rng, N)
    uniform_pts = rand(rng, N)
    normal_points = quantile(Normal(), uniform_pts)
    return normal_points
end
```

```
sample_normal_halton (generic function with 1 method)
```

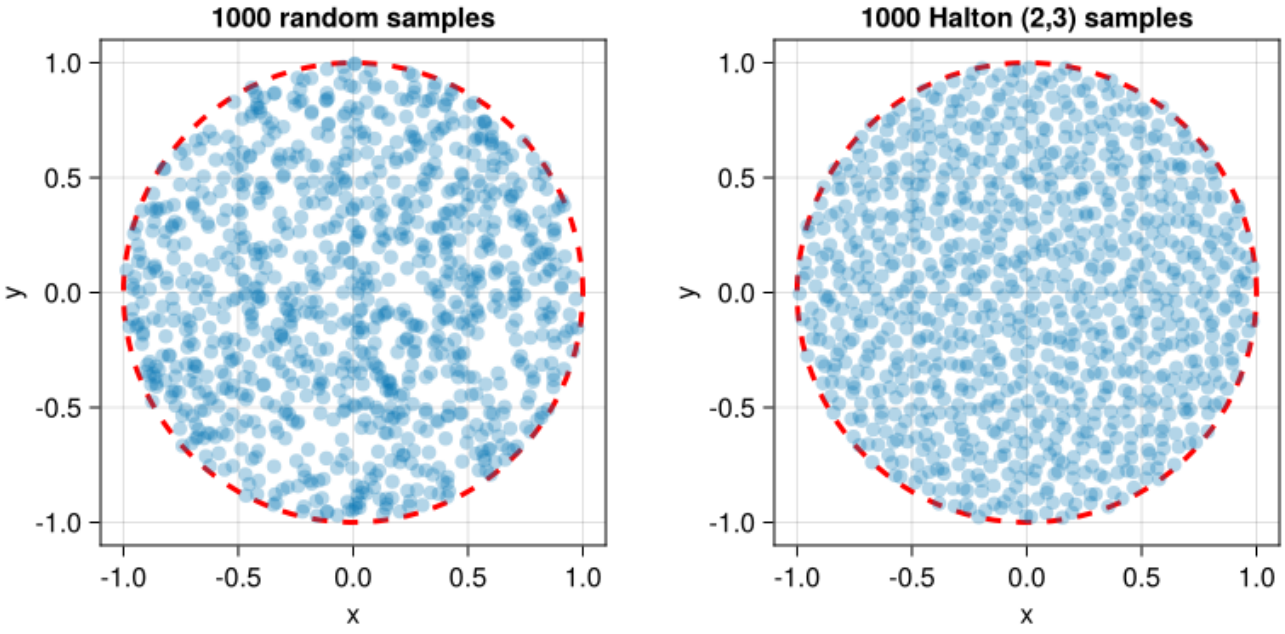
```
function sample_normal_halton(N; b = 2)
    random_samples = randn(N)
    halton_samples = sample_normal(HaltonGenerator(b), N)

    # plotting
    bins = round(Int, sqrt(N)) * 3

    linewidth = 3
    color = :red

    xs = LinRange(-10.0, 10.0, 200)
    ps = @. exp(-xs^2/2) / √(2π)

    f = Figure(; resolution = (800, 400))
    ax1 = Axis(f[1,1], title = "$(N) random samples")
    hist!(ax1, random_samples; bins, normalization = :pdf)
    lines!(ax1, xs, ps; linewidth, color)

    ax2 = Axis(f[1,2], title = "$(N) Halton($b) samples")
    xlims!(ax2, (-3.5, 3.5))
    hist!(ax2, halton_samples; bins, normalization = :pdf)
    lines!(ax2, xs, ps; linewidth, color)
    hideydecorations!(ax2, grid=false)

    linkyaxes!(ax1, ax2)
    linkxaxes!(ax1, ax2)

    f
end
```




```
sample_normal_halton(500)
```

Just as in the uniform case, these samples more quickly converge to our target distribution than the pseudorandom samples. Lastly, we can try a 2D example by combining a Halton(2) and a Halton(3) sequence to sample points on a circle.

```
sample_in_circle (generic function with 3 methods)
```



1.1: Quasi-Monte Carlo integration

Using low-discrepancy sequences, we can increase the convergence rate from $\mathcal{O}(1/\sqrt{N})$ to $\mathcal{O}(1/N)$ — a quadratic improvement!

To demonstrate this in practice, we will use our Halton sequence in order to estimate the following one-dimensional integral:

$$I = \int_0^1 \frac{1}{1+x} dx.$$

The exact solution is $\log(2) \approx 0.693147\ldots$

Just as above, we can write our Monte Carlo estimator of this integral as

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1+x_i} \quad x_i \sim \mathcal{U}(0,1).$$

To use the quasi-Monte carlo method, we simply swap our the built in `rand()` function for our Halton sequence generator when generating samples from the uniform distribution, just as we did above.

```
f (generic function with 1 method)
```

```
f(x) = 1 / (1 + x)
```

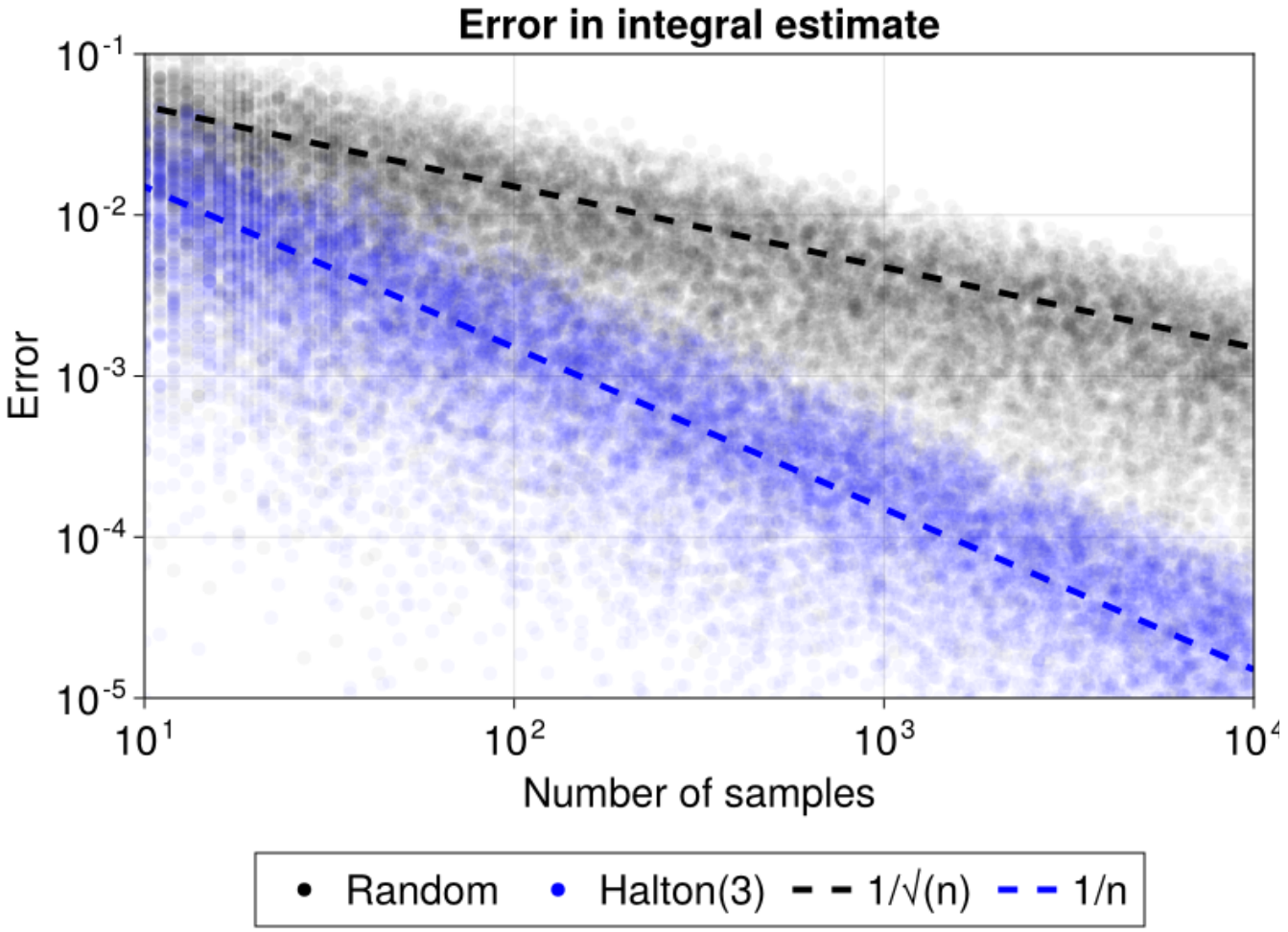
```
estimate_integral (generic function with 2 methods)
```

```
function estimate_integral(N, rng = Random.default_rng())
    samples = rand(rng, N)
    return mean(f.(samples))
end
```

Below, we plot independent estimates of our integral with different numbers of samples between 10 and 10,000. Note the log-scales on both axes.

We can see that, as expected, using a low-discrepancy sequence like the Halton sequence reduces the variance in our estimate dramatically, and furthermore increases the convergence rate from the rather dismal base level of $1/\sqrt{N}$ to a more respectable $1/N$.

To put this in practical terms, to get to an approximation error of 1%, we need (on average) between 100 and 1000 samples of the standard random number generator, but only 10 to 50 samples from the Halton sequence. This difference grows the lower the approximation error we desire. To achieve an error of 0.1%, we need upwards of 10,000 samples from the pseudo-random generator, but still only $\mathcal{O}(100)$ samples from the Halton sequence.


Error in integral estimate
Error
Number of samples
Random
Halton(3)
1/√(n)
1/n

2. Control variates

Quasi-Monte carlo methods are not the only ways to reduce the variance on Monte Carlo estimates. A large and widely-used class of methods are known as *control variates*. In these estimators, we exploit some knowledge of our function to reduce the variance in our estimate of its integral. The general principle works as follows.

Suppose we have some quantity of interest, μ, that can be expressed as the mean of some random variable. In our case, μ might be the integral of a function $f(x)$, which can be expressed as the mean value of $f(x)V(\Omega)$ when evaluated on samples $X \sim \mathcal{U}(\Omega)$. Evaluations of $f(x)V(\Omega)$ on the random variable X constitute a new random variable Y.

Now suppose we have some other random variable Z which has a known mean value ζ. We can write down a new variable Y_1 whose mean is also μ in terms of Y and Z:

$$Y_1 = Y - c(Z - \zeta),$$

where c is a free parameter that can take any value. This follows straightforwardly from the fact means are linear, i.e. the mean of $A + B$ is equal to the mean of A plus the mean of B. Thus, the mean of $Z - \zeta$ is zero and the mean of $Y - c(0)$ is just the mean of Y.

This all seems rather pointless so far, but the magic happens when we try to compute the variance in our estimation of μ. After some math, we ultimately find that there is an optimal value of c that minimizes the variance. The optimal value of c is

$$c^* = \frac{\mathrm{Cov}[Y, Z]}{\mathrm{Var}[Z]}.$$

Using c^*, we can write the variance as

$$\mathrm{Var}[Y_1] = (1 - \mathrm{Corr}(Y, Z)^2)\mathrm{Var}[Y_1]$$

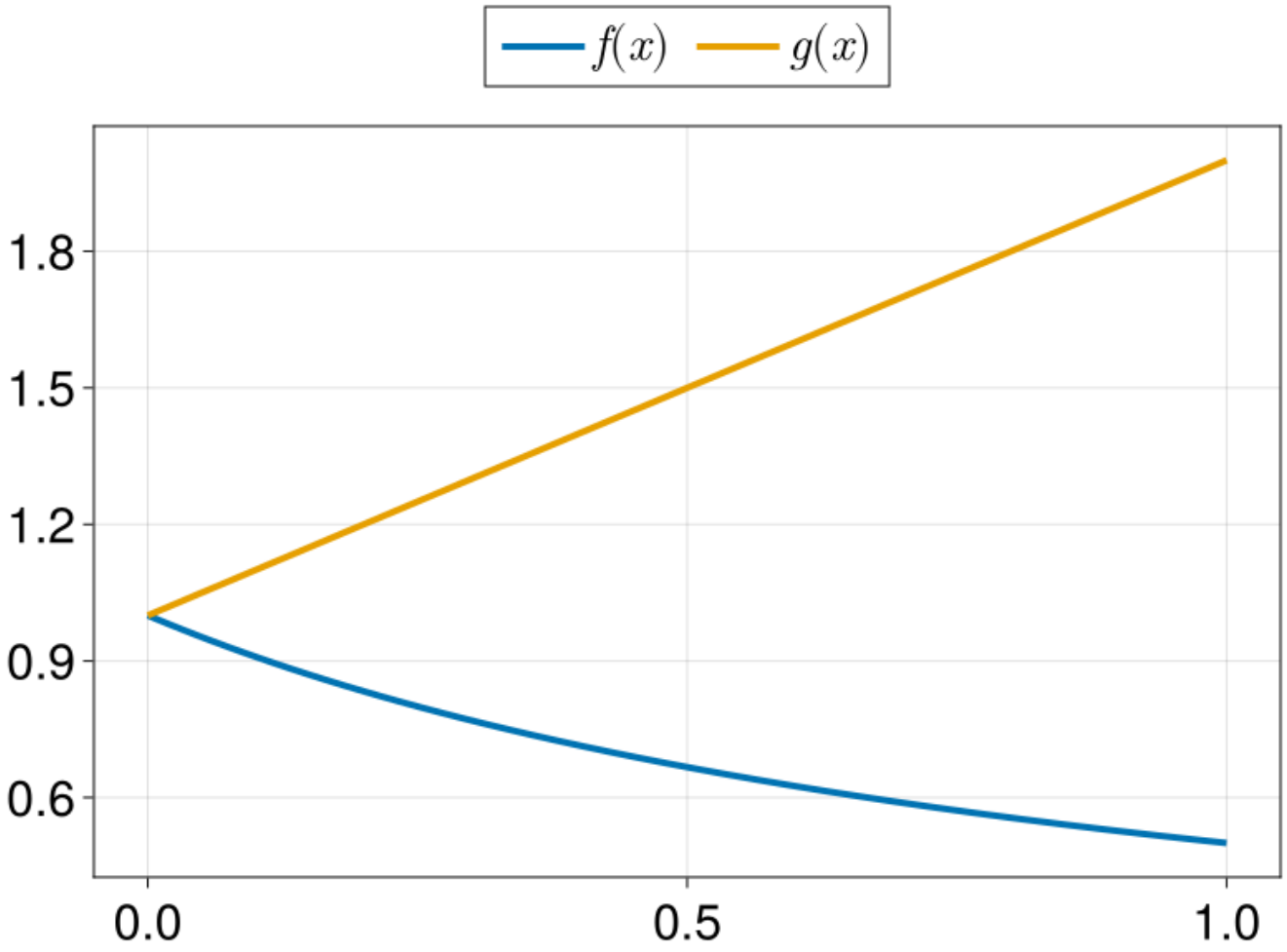
Here, $\mathrm{Corr}(Y, Z)$ is the Pearson correlation coefficient between Y and Z. Thus, using Y_1, the variance in our estimate of μ is *always as good or better* than if we had used Y. The improvement we get increases the more correlated (or anti-correlated) Y and Z are.

What does this mean for our integral? Well, if we can find some function $g(x)$ which is a good approximation of $f(x)$ over a large part of its domain, then $Y = f(X)V(\Omega)$ and $z = g(X)V(\Omega)$ will be highly correlated. Given some value of c (which can also be estimated on-line as we sample both functions), we can replace our original integral approximation with the following, provided we already know the integral of $g(x)$ over our domain.

$$\hat{I} = \frac{V(\Omega)}{N} \sum_{i=1}^{N} \left[f(x_i) + c \left(g(x_i) - \int_\Omega g(\xi) d\xi \right) \right]$$

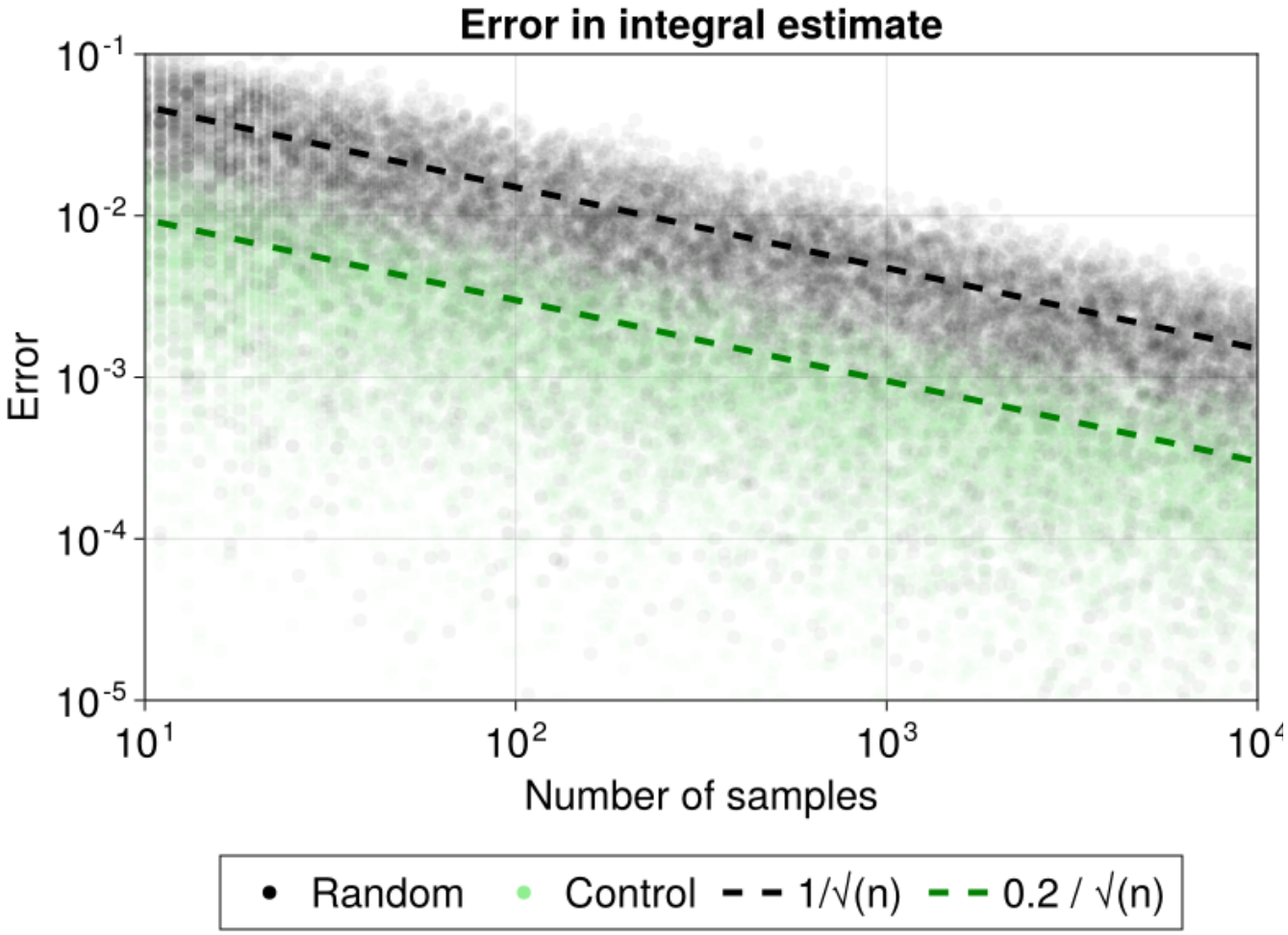
Here, we demonstrate control variates on our test problem by using $g(x) = 1 + x$. As shown below,

this is (anti-) correlated with $f(x) = 1/(1+x)$ and has an easy-to-compute integral on our domain (namely, 3/2).




```
estimate_integral_control_variates (generic function with 2 methods)
```

```julia
function estimate_integral_control_variates(N, rng = Random.default_rng())
    samples = rand(rng, N)
    gs = @. 1 + samples
    fs = @. f(samples)
    mean_f = mean(fs)
    mean_g = mean(gs)

    # Compute optimal c
    var_g = cov(gs)
    cov_f_g = cov(fs, gs)
    c = -cov_f_g / var_g
    if !isfinite(c)
        c = 0.5
    end
    return mean_f + c * (mean_g - 3/2)
end
```


Error in integral estimate
Error
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
10^{3}
10^{4}
Number of samples
Random
Control
1/√(n)
0.2 / √(n)

3. Antithetical variates

Applying control variates in practie can be complicated and usually requires some domain knowlege. One simple variant of this technique is the method of antithetical (or antithetic) variates. This method works by finding a function g(x) which is anti-correlated to our desired function, but which has the same integral. This can often be done by rescaling or flipping our function to induce an anticorrelation.

Consider two random variables, Y_1 and Y_2 with the same mean, μ. An unbiased estimator of μ can be written as

$$\mu \approx \frac{Y_1 + Y_2}{2}$$

The variance of this estimate is

$$\sigma^2 \approx \frac{\mathrm{Var}[Y_1] + \mathrm{Var}[Y_2] + 2\,\mathrm{Cov}[Y_1, Y_2]}{4}$$

If Y_1 and Y_2 are anti-correlated, the covariance term is negative and the variance is reduced.

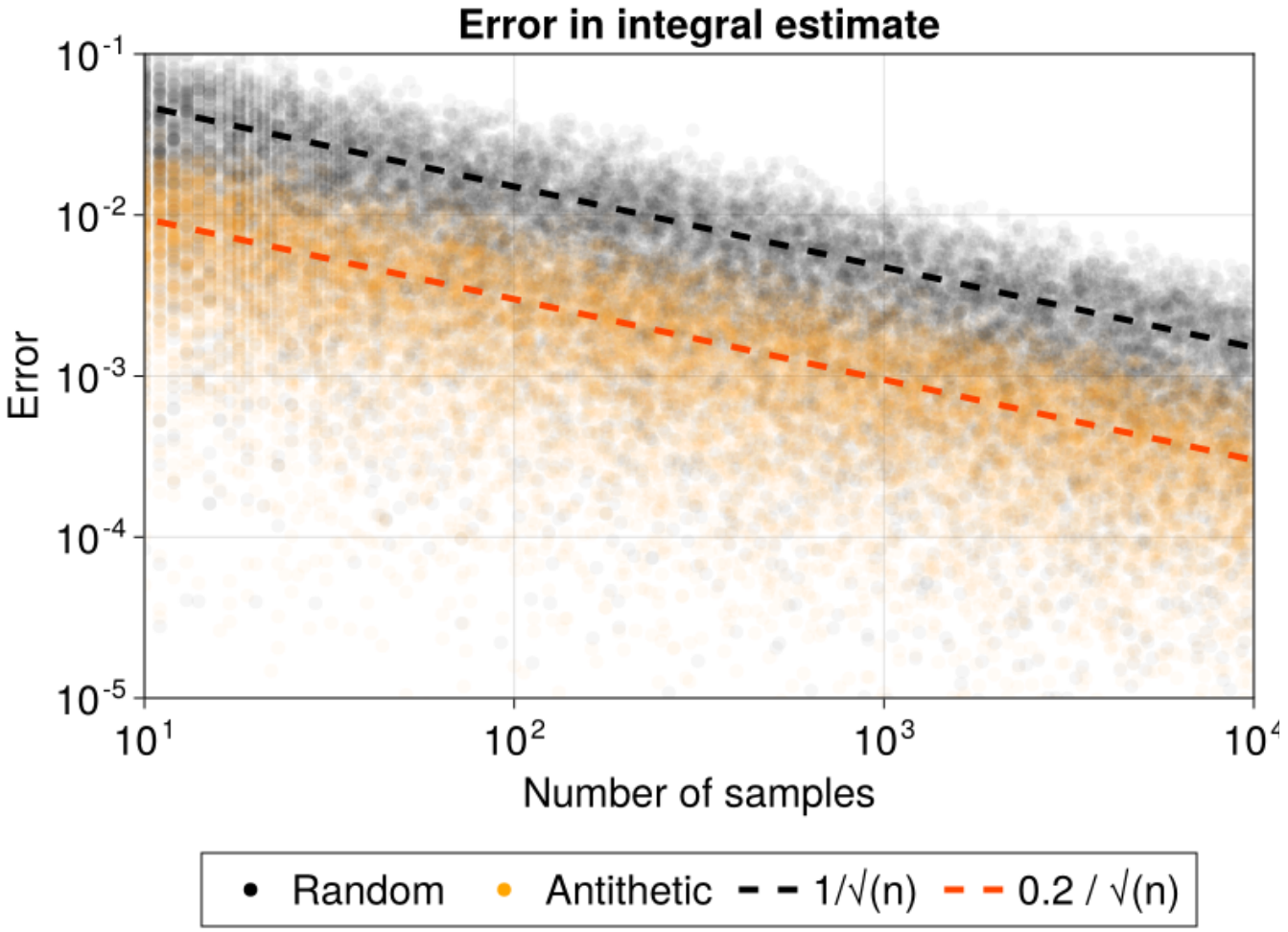
To apply this to our integral estimation problem, we need to find a variable which is anticorrelated with x_i but has the same mean. Since $x_i \sim \mathcal{U}(0, 1)$, we can pick $y_i \sim 1 - x_i$. So, for each x_i, we construct y_i and evaluate our function on both samples. We can then add the average of these estimates to our sum. If we were instead sampling x_i from a Gaussian, we could instead pick $y_i = -x_i$.

Below, we implement antithetical variates for our chosen integral and compare it to baseline Monte Carlo.

```
estimate_integral_anti (generic function with 2 methods)
```

```
function estimate_integral_anti(N, rng = Random.default_rng())
    samples = rand(rng, N÷2)
    f_x1 = f.(samples)
    f_x2 = f.(1 .- samples)
    return mean(f_x1 .+ f_x2) / 2
end
```

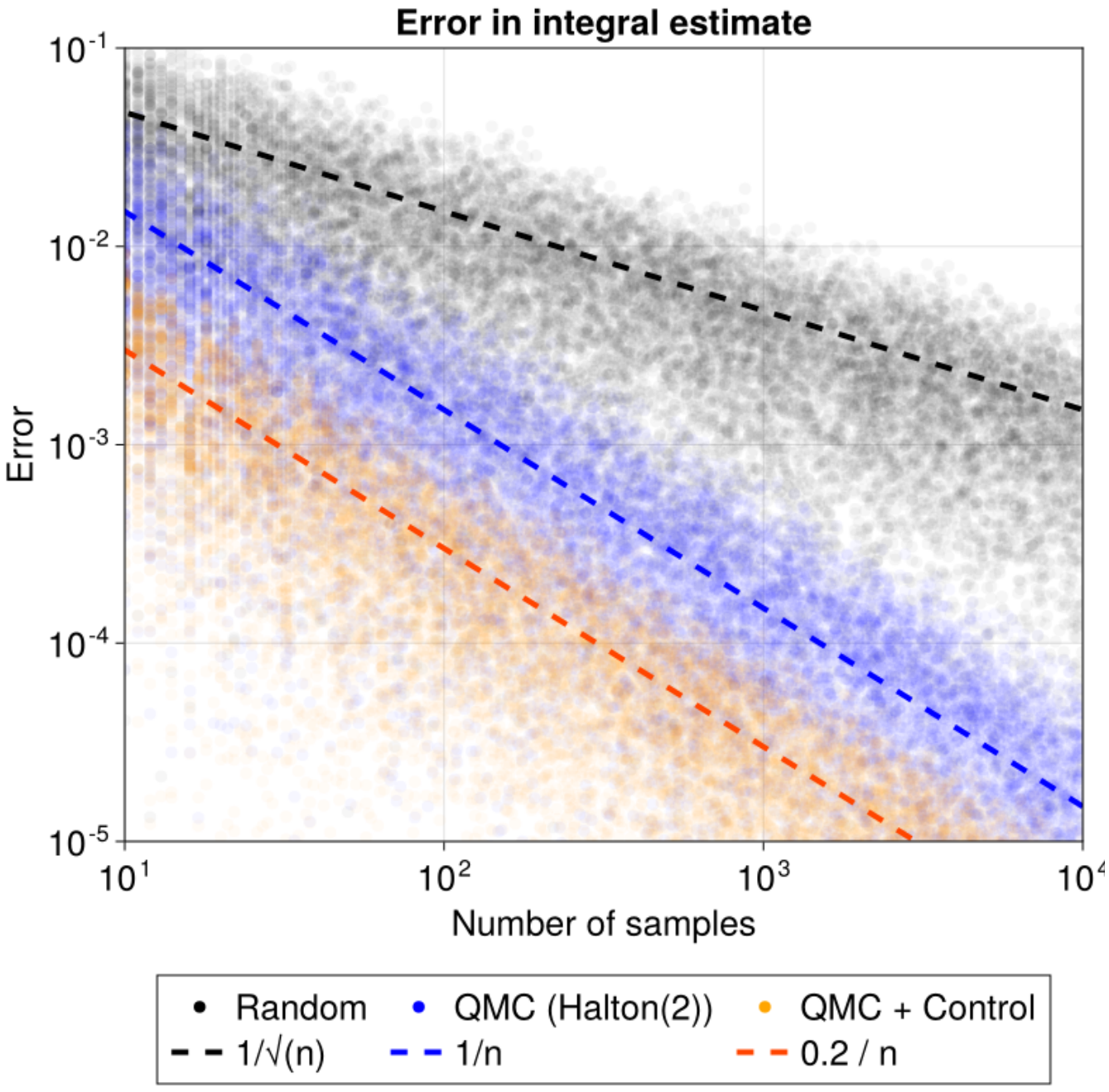
We can see that the variance is reduced by a factor of around 5, although the convergence rate is not improved.


Error in integral estimate
Error
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
10^{3}
10^{4}
Number of samples
Random
Antithetic
1/√(n)
0.2 / √(n)

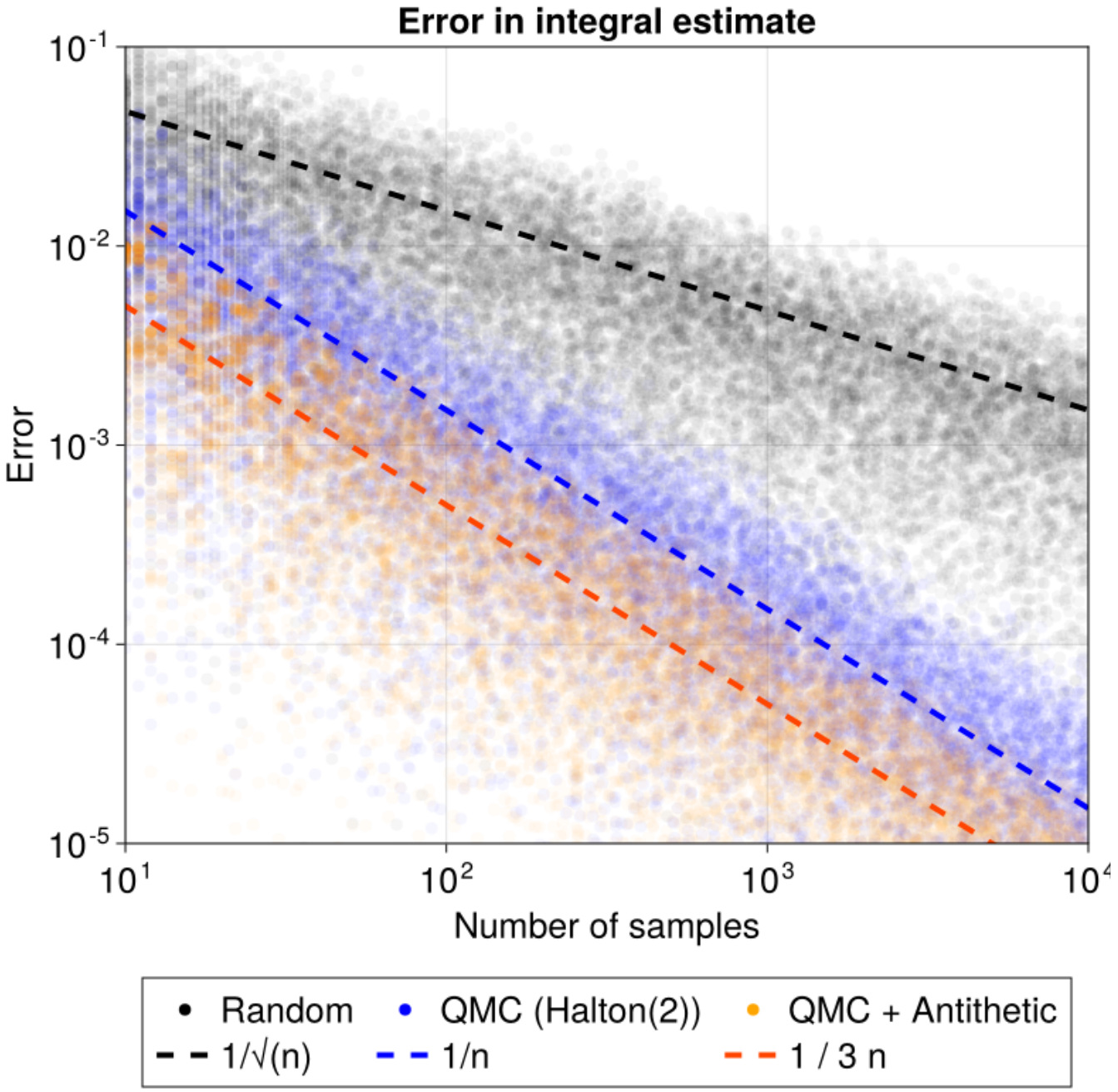
4. Combining methods

It is possible to achieve even larger variance reduction by combining the techniques above. Here we have combined control variates with the Halton sequence to achieve a 5-fold reduction in error over the base quasi-Monte Carlo estimator while still maintaining the desirable $1/N$ convergence rate of that method. To do this,we apply the control variates technique just as above, but replace the random number generator with our Halton sequence generator.

With just 10 samples of our hybrid estimator, we have the same error as we would get with 10,000 samples of the baseline MC estimator.



We can likewise combine the quasi-Monte carlo estimator with the antithetic variate method to achieve a similar improvement in our esimator. This time, the error is about threefold lower than the baseline estimator.


Error in integral estimate
Error
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
10^{3}
10^{4}
Number of samples
Random
QMC (Halton(2))
QMC + Antithetic
1/√(n)
1/n
1 / 3 n

5. Conclusion

Variance reduction techniques are at the core of the effective use of Monte Carlo methods. In this notebook, we have shown that they can dramatically reduce the error in the estimate of a simple integral. In higher-dimensional situations, however, finding suitable control variates becomes more complicated and usually requires the user to exploit some knowledge of the problem at hand. Quasi-monte carlo sequences (especially the Halton sequence) are also know the begin exhibiting some issues at larger dimensions. Care must be taken to reduce correlations between dimensions as much as possible, or else integrals evaluated using QMC techniques may be biased. This can be achieved via randomization or other techniques.

Effective variance reduction is an active area of research, but the techniques described here are some of the most common and widely-used of Monte Carlo methods, and form the basis of those being researched to this day.