Exercise – Understanding Monte Carlo integration using the identity function

1. Aim of the exercise

Monte Carlo integration (MC) is a technique for estimating integrals. In this exercise we demonstrate this technique by estimating the identity function over the unit interval.

2. Theory

We do not provide a thorough treatment of MC theory here; this is covered in the dedicated exercise on MC integration theory. Instead, we illustrate the basic idea through an example.

One of the simplest integrals is that of the identity function, $\phi(x) = x$, over the unit interval $[0,1]$:

$$\int_0^1 x\,dx = \left[\frac{1}{2}x^2\right]_0^1 = \frac{1}{2}.$$

This integral has an analytic solution, but we will approximate it using MC integration.

To apply Monte Carlo integration, we must choose a sampling distribution from which to draw random sample points, that is, values for x, that we use to estimate the integral. Since we are working over the unit interval, we choose $X \sim \text{Unif}(0,1)$. The corresponding PDF $f_X : [0,1] \to \mathbb{R}$ is given by:

$$f_X(x) = \frac{1}{1-0} = 1.$$

We then define the target function

$$\phi(x) = x$$

and write it as

$$\phi(x) = f_X(x) \cdot g(x),$$

where

$$g(x) = \frac{\phi(x)}{f_X(x)} = x$$

The integral becomes

$$\int_0^1 \phi(x)\,dx = \int_0^1 f_X(x) \cdot g(x)\,dx = \int_0^1 1 \cdot x\,dx = \mathbb{E}[X].$$

Note that to compute the expected value of $g(X)$, we use a rule from probability theory: if we have a function of a random variable, we can compute its expectation by integrating that function times the PDF of the original variable. This is called the law of the unconscious statistician. This law allows us to avoid computing the distribution of $g(X)$ directly.

Averaging $\phi(x) = x$ over random values in $[0,1]$ gives an estimate of the integral. To approximate this average, we draw a number of independent samples from X, evaluate $g(x)$ at each sampled point, and compute the estimate:

$$\frac{1}{N}\sum_{i=1}^{N} g(x_i).$$

This sample mean approximates the expected value $\mathbb{E}[X]$, which equals the value of the integral.

3. Define the number of samples

Clear the memory. Set the sample size. In the context of MC integration, the term sample refers to a single random selection from a broader set of possibilities, in this case, the interval from 0 to 1. We can think of that interval as a population: a continuous range of values we are interested in exploring. Each time we randomly pick a number from that interval, we are drawing one observation from that population. Calling it a sample emphasises that we are not just picking numbers at random, but we are collecting data points to estimate something larger, like the average behaviour of a function over the interval. Just as researchers sample individuals to learn about a population, we sample values from a distribution to learn how a function behaves across its domain. By averaging many such samples, we estimate the function's typical value over the interval. The more samples we take, the more reliable our estimate becomes.

```
%% 3. Define the number of samples

% 3.1. Clear workspace and memory
clear;

% 3.2. Number of random samples to estimate the integral
N_samples = 1000;
```

4. Monte Carlo integration: Estimating integrals via sampling

Here we perform the actual MC estimation of the integral. First, we generate a set of random values between 0 and 1. These values are drawn independently from a uniform distribution, meaning each number in the interval has an equal chance of being selected. We store these values in a column vector called `uniform_samples`. The number of values generated is determined by the sample size, which was defined earlier.

Next, we compute the average of these sampled values using the `mean` function. Since the function we are integrating is simply the identity, each sample represents both the input and the output; the average of the samples directly estimates the value of the integral. This is the essence of MC integration: using random sampling to approximate an average, which in turn approximates the integral.

```
%% 4. Monte Carlo integration: Estimating integrals via sampling

% 4.1. Generate random samples from Uniform[0,1]
uniform_samples = random('Uniform',0,1,[N_samples 1]);

% 4.2. MC estimate of the integral of f(x) = x over [0,1]
integral_estimate = mean(uniform_samples);
```

5. Set the true value of the integral of the identity function

This section sets the known, exact value of the integral for reference, which is calculated in the Theory section. This allows us to later compare the MC estimate with the exact result, which is useful for assessing the accuracy of our approximation. It also reinforces the idea that

MC methods are often used when analytic solutions are unavailable, but in this case, we use a simple example where the true value is known.

```
%% 5. Set the true value of the integral of the identity function

% 5.1. Set the true value of the integral
integral_true_value = 0.5;
```

6. Geometric interpretation of Monte Carlo integration

We estimate the area under $\phi(x) = x$ on the interval $[0, 1]$ using a geometric Monte Carlo method. We sample random points (x, y) uniformly in the unit square $[0, 1] \times [0, 1]$. Each x value is treated as a fixed input to the function $\phi(x)$, while each y value is drawn independently from $[0, 1]$. Rather than numerically integrating $\phi(x)$, we estimate the area by checking whether $y < \phi(x)$ for each sampled point. Points that satisfy this condition lie below the curve and contribute to the area estimate. The fraction of such points approximates the integral of $\phi(x)$ over $[0, 1]$.

Figure 1 visualizes this approach. It shows sampled points in the unit square, with those below the diagonal line $y = x$, that is, where $y < \phi(x)$. The diagonal represents the function $\phi(x) = x$, and the shaded region beneath it corresponds to the estimated area. The proportion of highlighted points serves as a Monte Carlo approximation of the integral.

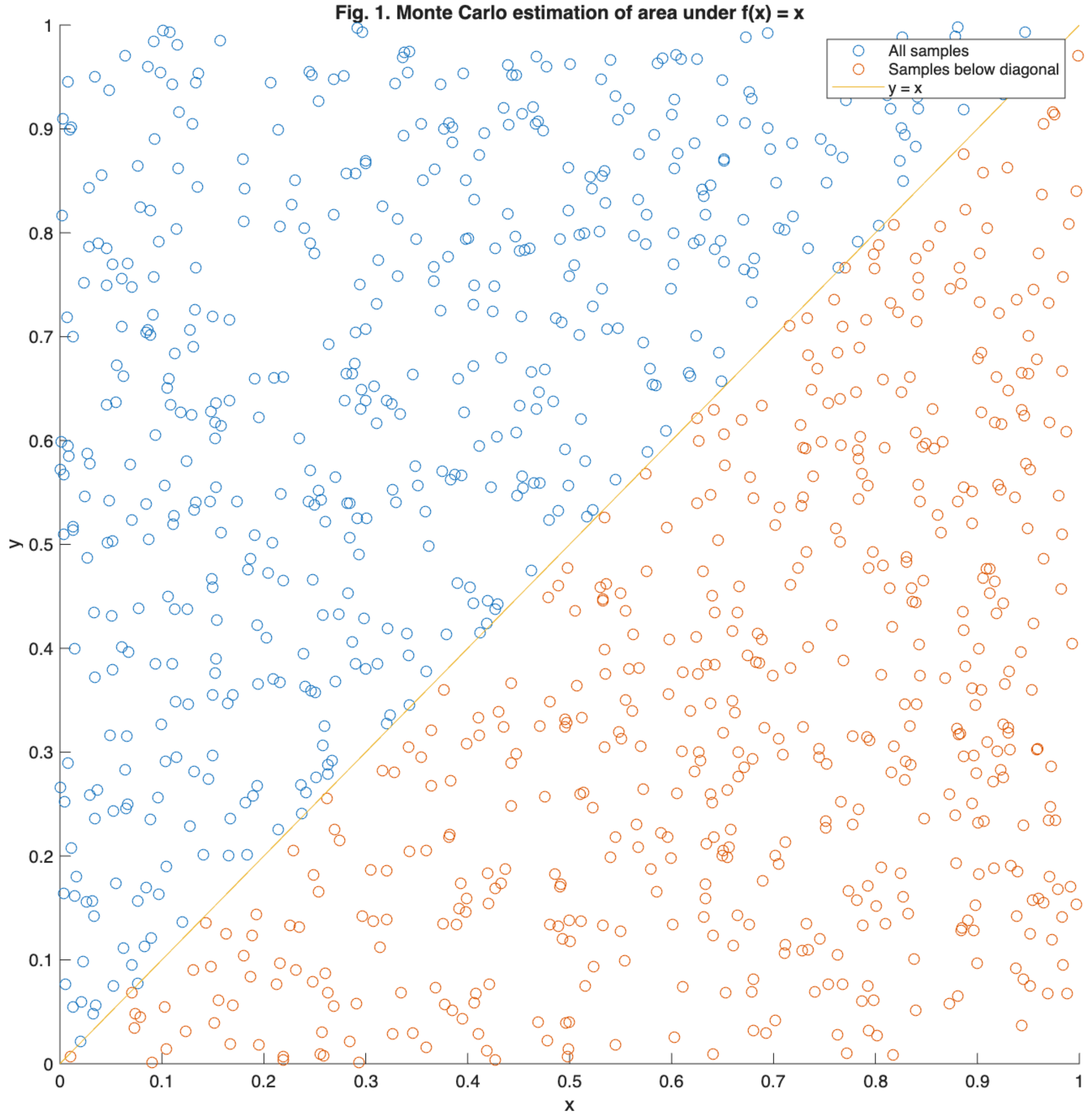
```
%% 6. Geometric interpretation of Monte Carlo integration

% 6.1 Create (x, y) points uniformly in the unit square
x_samples = uniform_samples; % x-coordinates sampled uniformly in [0,1]
y_samples = random('Uniform',0,1, ...
    [N_samples 1]); % y-coordinates sampled independently in [0,1]

% 6.2. Identify points below the diagonal y < x
below_diagonal = y_samples < x_samples;

% 6.3. Visualizing area estimation via Monte Carlo sampling
figure
hold on
scatter(x_samples,y_samples, ...
    'DisplayName','All samples') % All samples
scatter(x_samples(below_diagonal),y_samples(below_diagonal), ...
    'DisplayName','Samples below diagonal'); % Samples below diagonal
plot([0 1],[0 1], ...
    'DisplayName','y = x') % Diagonal line y = x
title('Fig. 1. Monte Carlo estimation of area under f(x) = x')
xlabel('x')
ylabel('y')
legend('show')
hold off
```

Fig. 1. Monte Carlo estimation of area under f(x) = x



7. Convergence behavior of the Monte Carlo integral estimate

We examine how the Monte Carlo integration estimate evolves as the sample size increases, focusing on its convergence toward the true value of the integral. To prepare for visualization, we compute the running estimate and its mean squared error (MSE) across a range of sample sizes. These quantities will allow us to assess how quickly the estimate stabilizes and how the error decays as more samples are added. For a deeper understanding of the convergence behavior discussed here, we recommend completing the exercise titled "Understanding convergence speed through Big O notation".

The first line uses `cumsum` to incrementally accumulate the sampled values from the uniform distribution. Dividing the cumulative total by the number of samples seen so far yields the running mean, which reflects how the Monte Carlo estimate evolves as more samples are added. This process mimics sequential sampling and shows how the estimate stabilizes over time. By the law of large numbers, the running mean should eventually settle near the true value of 0.5.

The second line computes the squared error of each sample relative to the true integral using `(uniform_samples-integral_true_value).^2`. Applying `cumsum` accumulates these squared errors over time. Dividing the cumulative total by the number of samples seen so far yields the running mean squared error. This sequence tracks how the average error of the Monte Carlo estimate decreases as more samples are added. Ideally, the MSE should decay toward zero with increasing sample size, reflecting improved accuracy.

MC error typically decreases at a rate proportional to $\mathcal{O}(1/n^{\frac{1}{2}})$ as the number of samples n increases. The third line prepares a reference curve that illustrates this expected decay in error with sample size. The underlying reason is variance reduction: each sample has some variance, and averaging n independent samples reduces the variance of the mean by a factor of $\frac{1}{n}$. Since the standard deviation, the typical magnitude of error, is the square root of the variance, it decays as $\mathcal{O}(1/n^{\frac{1}{2}})$. This behavior follows from the Central Limit Theorem, which states that the distribution of the sample mean approaches a normal distribution with variance shrinking at the $\mathcal{O}(1/n)$ rate.

```
%% 7. Convergence behavior of the Monte Carlo integral estimate

% 7.1. Track how the estimate evolves with more samples
convergence_integral_estimate = cumsum(uniform_samples)./(1:N_samples)';

% 7.2. Track how the mean squared error (MSE) evolves with more samples
convergence_MSE = cumsum((uniform_samples-integral_true_value).^2) ...
    ./(1:N_samples)';

% 7.3. Theoretical benchmark
theoretical_error_decay = 1./sqrt(1:N_samples);
```

8. Visualize convergence of the Monte Carlo estimate and its error

To visualize how the Monte Carlo estimate behaves as more samples are added, we generate two plots: one showing the estimate itself, and another showing how its error decays. The first figure plots the running estimate of the integral against the number of samples. We overlay this with a horizontal line at the true value of the integral, $\frac{1}{2}$, to highlight convergence. As the sample size increases, the estimate stabilizes near this target, illustrating the law of large numbers in action.

The second figure shows how the MSE decreases as more samples are added. Because the MSE values become very small, we use a log-log scale to better capture their behavior across several orders of magnitude. This transformation makes the decay pattern more interpretable and allows us to compare it directly to the theoretical benchmark, which decays at a rate of $\mathcal{O}(1/n)$. The plot confirms that the empirical error closely follows the expected trend.

```
%% 8. Visualize convergence of the Monte Carlo estimate and its error

% 8.1. Plot how the Monte Carlo estimate converges to the true value
figure
hold on
plot(1:N_samples,convergence_integral_estimate, ...
    'DisplayName','MC integral estimate');
yline(integral_true_value, ...
    'DisplayName','True integral value');
title('Fig. 2. Convergence of Monte Carlo estimate');
xlabel('Number of samples (draws)');
ylabel('Integral estimate');
legend('show');
hold off

% 8.2. Plot how the MSE decreases with more samples (log-log scale)
figure
hold on
loglog(1:N_samples,convergence_MSE, ...
    'DisplayName','MSE of MC estimate');
loglog(1:N_samples,theoretical_error_decay.^2, ...
    'DisplayName','Theoretical MSE decay');
title('Fig. 3. Log-log convergence of MC estimation error')
xlabel('Number of samples (log scale)')
ylabel('Mean Squared Error (log scale)')
legend('show')
hold off
```

Fig. 2. Convergence of Monte Carlo estimate

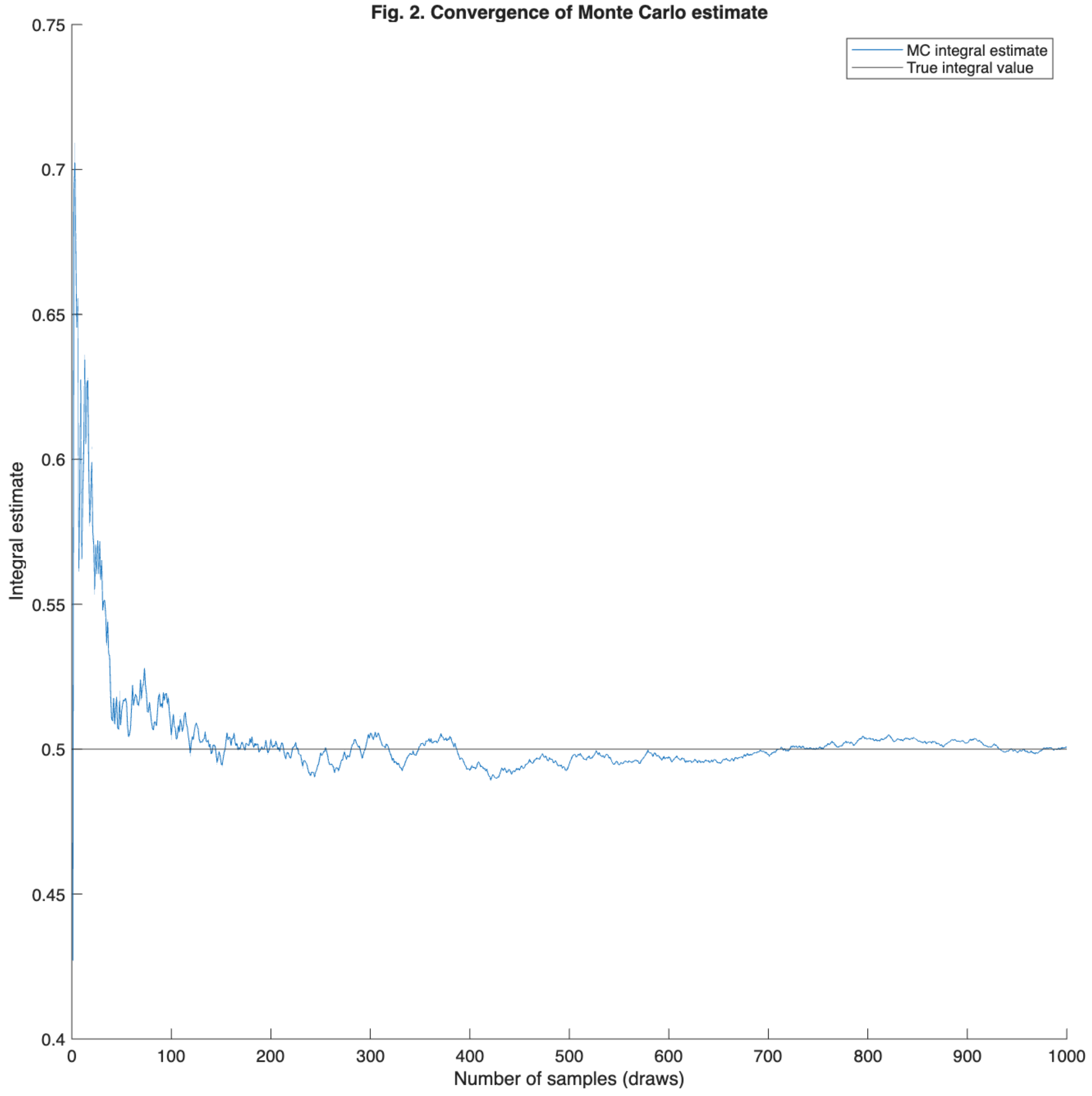
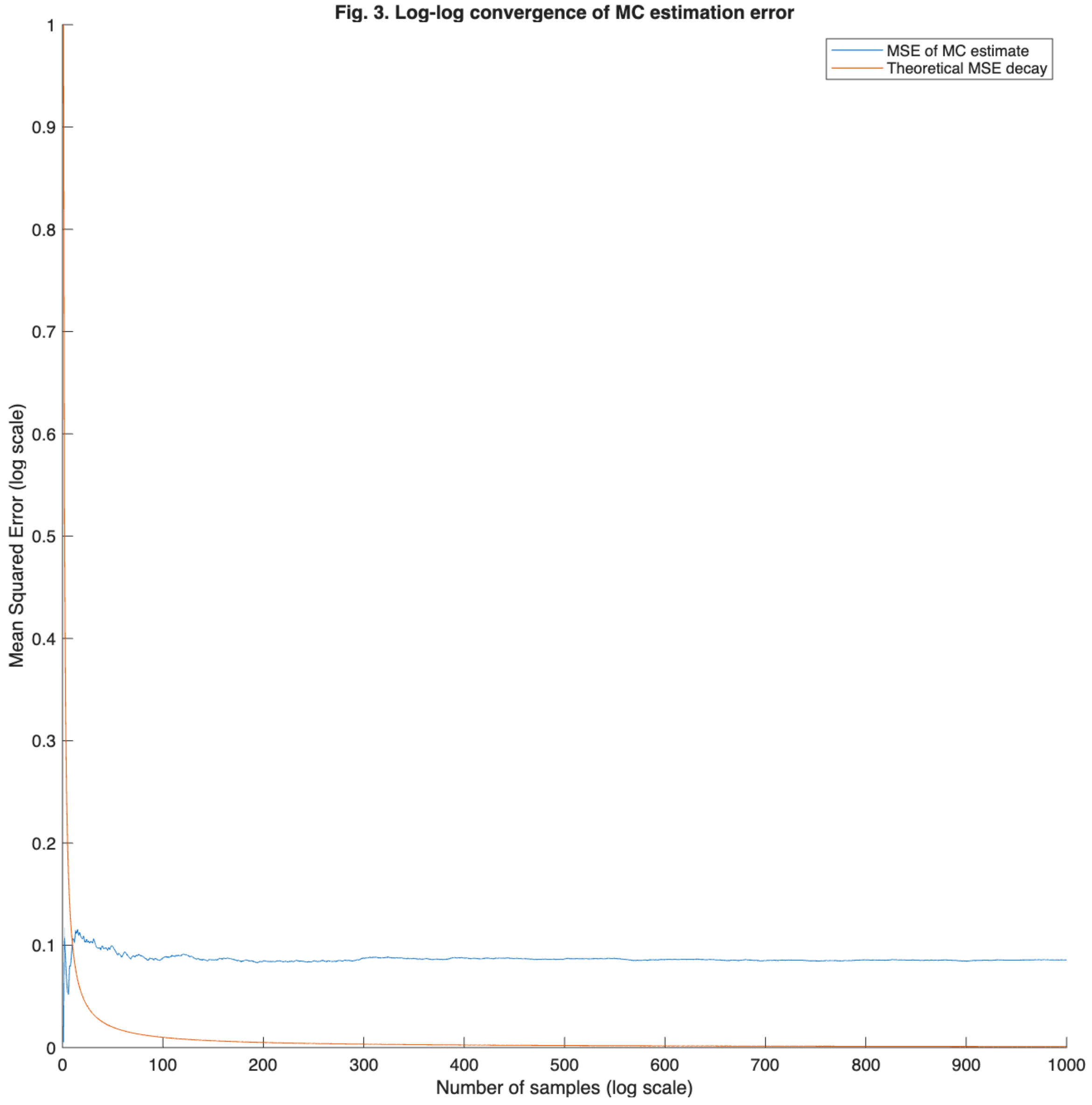


Fig. 3. Log-log convergence of MC estimation error



15. Final notes