Use simulation to calculate integral

1. Summary

Feature	Standard Monte Carlo	Importance Sampling	Halton Draw (Quasi-MC)
Randomness	**Pseudo-Random**. Each draw is random and independent of the others. This can lead to clustering and gaps.	**Biased Pseudo-Random**. Also uses pseudo-random numbers, but draws them from a different distribution to focus on important regions.	**Quasi-Random (Deterministic)**. Not random at all. The sequence is deterministic and designed to fill the space evenly.
Key idea	The Law of Large Numbers. A simple average of random samples converges to the true value.	Reduce variance by sampling more from high-impact regions and then re-weighting the results to correct for the bias.	Reduce error by ensuring samples are spread out more evenly than by pure chance, avoiding clusters and gaps.
Formula	$\frac{1}{N}\sum f(x_i)$	$\frac{1}{N}\sum f(x_i)\frac{p(x_i)}{q(x_i)}$	$\frac{1}{N}\sum f(x_i)$
Error rate	The estimation error decreases proportionally to $\frac{1}{\sqrt{N}}$. To halve the error, you need 4x the samples.	Also decreases at $\frac{1}{\sqrt{N}}$, but with a smaller constant factor (lower variance), leading to faster convergence in practice.	The estimation error decreases closer to $\frac{1}{N}$. This is **a much faster rate of convergence**, especially in low dimensions.
Best for	General-purpose, high-dimensional problems. Simple to implement.	Rare event simulation or when the function's value is concentrated in a small region.	Low-to-moderate dimensional numerical integration. It excels at efficiently exploring the parameter space.

2. Problem Setting

Use simulation methods to approximate the integral. Simulation is usually used to calculate the choice probability in demand estimation or broadly speaking, empirical IO.

Consider a random variable $X \sim N(\mu, \sigma^2)$ with $\mu = 2, \sigma^2 = 2$, and we need to calculate $E(X^2)$.

1. Calculate the value of $E(X^2)$ analytically.
2. Calculate this value by a Monte Carlo simulation. Hint: we approximate the integral by
   $E(X^2) = \int x^2 f(x) df \simeq \frac{1}{R} \sum_{r=1}^{R} (x_r)^2$
   where x_r is drawn from the distribution $N(\mu, \sigma^2)$. R is the number of random draws, which we can choose. Report the value and estimation errors. How does it depend on the choice of R? Try alternative simulation methods like importance sampling and Halton draw, respectively.

solution for 1

$E(X^2) = Var(X) + [E(x)]^2 = 2 + 4 = 6$

3. Monte Carlo Simulation

```
%% Monte Carlo Simulation to Calculate E[X^2]
% This script calculates the expected value of X^2 where X ~ N(mu, sigma^2)
% both analytically and through Monte Carlo simulation.

%% 0. Setup
clear;
clc;
rng(42);        % Set the seed for reproducibility

%% 1. Analytical Calculation
% For a random variable X, the variance is Var(X) = E[X^2] - (E[X])^2.
% Rearranging this gives E[X^2] = Var(X) + (E[X])^2.
```

```
% Given X ~ N(mu, sigma^2), we know E[X] = mu and Var(X) = sigma^2.
% Therefore, the analytical value is sigma^2 + mu^2.

mu = 2;
sigma_sq = 2; % The problem gives sigma^2 = 2
analytical_E_X2 = sigma_sq + mu^2;

fprintf('Part 1: Analytical Calculation\n');
fprintf('The analytical value of E[X^2] is: %.4f\n\n', analytical_E_X2);

%% 2. Monte Carlo Simulation
% We will approximate the integral E[X^2] by taking the average of many
% squared random draws from the distribution.

fprintf('Part 2: Monte Carlo Simulation\n');

% --- Simulation Parameters ---
R = 100000; % Number of random draws (simulations)
sigma = sqrt(sigma_sq);

% --- Generate Draws and Calculate X^2 (Vectorized) ---
% We first draw R random numbers from N(0,1)
v = randn(R, 1); % 'randn' draws from the standard normal distribution

% Then, we transform these draws to be from N(mu, sigma^2)
x = mu + sigma * v;

% Finally, we square the results
x_sq = x.^2;

% --- Calculate the Monte Carlo Estimate ---
% The estimate is the sample mean of the squared draws
monte_carlo_E_X2 = mean(x_sq);

fprintf('Using R = %d draws...\n', R);
fprintf('The Monte Carlo estimate of E[X^2] is: %.4f\n', monte_carlo_E_X2);
```

```
% --- Discussion: Dependence on R ---
% The accuracy of the Monte Carlo simulation depends on R. By the Law of
% Large Numbers, as R -> infinity, the sample mean converges to the true
% expected value.

R_values = [100, 1000, 10000, 100000, 1000000]; % Different values of R to
test
errors = zeros(length(R_values), 1);          % Pre-allocate error vector

fprintf('\n--- Analyzing the effect of R ---\n');
for i = 1:length(R_values)
    current_R = R_values(i);

    % Re-run simulation for the current R
    x_sim = mu + sigma * randn(current_R, 1);
    mc_estimate = mean(x_sim.^2);

    % Calculate absolute error
    errors(i) = abs(mc_estimate - analytical_E_X2);

    fprintf('R = %-7d | Estimate = %.4f | Error = %.4f\n', ...
            current_R, mc_estimate, errors(i));
end

% Plot the results to visualize the convergence
figure;
loglog(R_values, errors, '-o', 'LineWidth', 2);
title('Monte Carlo Error vs. Number of Draws (R)');
xlabel('Number of Draws (R)');
ylabel('Absolute Error |Simulated - Analytical|');
grid on;


%% Results are
Part 1: Analytical Calculation
```

```
The analytical value of E[X^2] is: 6.0000

Part 2: Monte Carlo Simulation
Using R = 100000 draws...
The Monte Carlo estimate of E[X^2] is: 5.9935

--- Analyzing the effect of R ---
R = 100     | Estimate = 6.4290 | Error = 0.4290
R = 1000    | Estimate = 5.9963 | Error = 0.0037
R = 10000   | Estimate = 6.1061 | Error = 0.1061
R = 100000  | Estimate = 5.9960 | Error = 0.0040
R = 1000000 | Estimate = 5.9983 | Error = 0.0017
```

3. Importance Sampling Simulation

3.1 Some Explaination

In a nutshell, **importance sampling is not a method for drawing random numbers, but a technique for using them more efficiently to calculate an expectation or an integral.** We still use standard random number generators.

The core idea is to concentrate our samples in the "important" regions of a function, dramatically speeding up calculation, especially when dealing with rare events.

3.2 Why we need it? A example

Imagine you want to estimate the probability that a standard normal random variable $X \sim N(0,1)$ is greater than 5. This is an extremely rare event. (The analytical answer is about 2.87×10^-7)

If you use a standard Monte Carlo approach, you would:

- Draw millions of samples from N(0,1).
- Count how many are greater than 5.
- Divide by the total number of samples.

You would likely need to draw billions of samples just to get a few "hits." This is incredibly inefficient because *you're wasting almost all your computational effort sampling in regions (like -2 to 2)* where the event of interest never occurs. This is where importance sampling comes in.

3.3 How to do it?

Instead of sampling from the original distribution $p(x)$, we introduce a new, clever proposal distribution, $q(x)$, that is intentionally chosen to generate more samples in the regions that matter most.

1. Our original goal is to calculate an expectation:
   $E_p[f(X)] = \int f(x)p(x)dx$ where $p(x)$ is the PDF of $N(0,1)$ and $f(x)$ is an indicator function that is 1 if $x > 5$ and 0 otherwise.
2. **The Mathematical Trick**
   We can rewrite the integral by multiplying and dividing by our new proposal distribution, $q(x)$:
   $E_p[f(X)] = \int f(x)\frac{p(x)}{q(x)}q(x)dx$ This new expression can be interpreted as the expectation of a different function, $f(x)\frac{p(x)}{q(x)}$, with respect to a new distribution, $q(x)$.
3. **The importance weight**
   The term $\frac{p(x)}{q(x)}$ is called the importance weight or likelihood ratio. It's a correction factor. It accounts for the fact that we sampled from the "wrong" distribution (q) instead of the target distribution (p).

- If we sample from a region where $q(x)$ is more likely than $p(x)$, the weight is small, down-weighting that sample.
- If we sample from a region where $q(x)$ is less likely than $p(x)$, the weight is large, up-weighting that sample to its "proper" level of importance.

4. **The Algorithm**

- Choose a proposal distribution $q(x)$. For our example, instead of sampling from $N(0,1)$, let's sample from a distribution centered in our area of interest, like $q(x) \sim N(6,1)$. This will produce lots of samples greater than 5.
- Draw N samples $(x_1, x_2, \ldots, x_N)$ from your proposal distribution $q(x)$.
- Calculate the importance weight $w(i) = p(x_i)/q(x_i)$ for each sample.
- Estimate the expectation using the weighted average:
  $E_p(X) = \frac{1}{N}\sum_i^N f(x_i)w_i$ With this method, almost every sample we draw

from $q(x) \sim N(6,1)$ will be in the "important" region. We then use the weights to correct for this biased sampling, giving us an accurate estimate with far fewer draws.

3.3 Use the importance sampling to solve the problem.

To adapt the code to use importance sampling, the key change is to draw random numbers from a different distribution (the "proposal") and then re-weight the results to get the correct estimate for the original ("target") distribution. A good proposal distribution should emphasize the "important" regions.Since we are calculating $E[X^2]$, the regions where $\mid x \mid$ is large contribute most to the final value. Therefore, a good proposal distribution, $q(x)$, should have "fatter tails" (a larger variance) than our target. Let's choose $q(x) \sim N(2,8)$ We keep the mean the same but increase the variance to sample the tails more often.

Here is the modified MATLAB code.

```
%% Importance Sampling to Calculate E[X^2]
% This script estimates E[X^2] where X is drawn from a target distribution
% p(x) ~ N(2, 2), using samples from a different proposal distribution
q(x).

%% 0. Setup
clear;          % Clear workspace
clc;            % Clear command window
rng(42);        % Set the seed for reproducibility

%% 1. Define Distributions and Analytical Value
% --- Target Distribution: p(x) ---
mu_p = 2;
sigma_sq_p = 2;
sigma_p = sqrt(sigma_sq_p);

% --- Proposal Distribution: q(x) ---
% We choose a distribution with a larger variance to better sample the
tails,
% which are important for the function f(x) = x^2.
mu_q = 2;
sigma_sq_q = 8; % Higher variance (fatter tails)
sigma_q = sqrt(sigma_sq_q);

% --- Analytical Value (for comparison) ---
```

```
% E[X^2] = Var(X) + E[X]^2 for the target distribution p(x).
analytical_E_X2 = sigma_sq_p + mu_p^2;

fprintf('The analytical value of E[X^2] is: %.4f\n', analytical_E_X2);

%% 2. Importance Sampling Simulation
% --- Simulation Parameters ---
R = 100000; % Number of random draws

% --- Step 1: Draw samples from the PROPOSAL distribution q(x) ---
x_q = mu_q + sigma_q * randn(R, 1);

% --- Step 2: Calculate the function value f(x) = x^2 for each sample ---
f_x = x_q.^2;

% --- Step 3: Calculate the importance weights w(x) = p(x) / q(x) ---
% The weight corrects for the fact we sampled from q(x) instead of p(x).
pdf_p = normpdf(x_q, mu_p, sigma_p); % PDF of target at each sample
pdf_q = normpdf(x_q, mu_q, sigma_q); % PDF of proposal at each sample
weights = pdf_p ./ pdf_q;

% --- Step 4: Calculate the final estimate ---
% The estimate is the mean of the function values multiplied by their
weights.
is_estimate = mean(f_x .* weights);

fprintf('Using R = %d draws...\n', R);
fprintf('The Importance Sampling estimate of E[X^2] is: %.4f\n',
is_estimate);
fprintf('The error is: %.4f\n', abs(is_estimate - analytical_E_X2));

%% Results are
The analytical value of E[X^2] is: 6.0000
Using R = 100000 draws...
The Importance Sampling estimate of E[X^2] is: 6.0110
The error is: 0.0110
```

4. Halton Draws

4.1 Some explanation

Halton draws are used to generate **quasi-random** numbers, which are not truly random but are designed to cover a space more evenly than pseudo-random numbers. **You don't "draw" them in the probabilistic sense; you generate a deterministic sequence that fills the space with low discrepancy.**

The key difference is that **Halton sequences are deterministic and uniform**, whereas standard Monte Carlo uses pseudo-random numbers and Importance Sampling uses biased pseudo-random numbers that are re-weighted.

4.2 How to use it?

A Halton sequence is generated using prime numbers. To create a sequence of points in a k-dimensional space, you need k different prime numbers. For a simple one-dimensional sequence, you only need one prime, typically 2.

The process, known as the radical inverse, involves representing integers in a prime base and then reflecting the digits across the decimal point.

Example: Generating a 1D Halton Sequence with prime base 2

1. Integer (n): 1, 2, 3, 4, 5, 6...
2. Binary (Base 2): 1, 10, 11, 100, 101, 110...
3. Reverse the digits: .1, .01, .11, .001, .101, .011...
4. Convert to Decimal: 0.5, 0.25, 0.75, 0.125, 0.625, 0.375...

This sequence [0.5, 0.25, 0.75, 0.125, 0.625, 0.375, ...] starts filling the interval (0, 1) in a very structured way, ensuring no large gaps are left.

Fortunately, you don't need to do this by hand. Most scientific programming software, including MATLAB, has built-in functions to generate these sequences.

4.4 Use Quasi-Monte Carlo using Halton Draws to Calculate $E[X^2]$

In MATLAB, you create a haltonset object and then use it to generate points. The following code modifies the Monte Carlo simulation to use Halton draws instead of pseudo-random numbers.

```
1
2 %% Quasi-Monte Carlo using Halton Draws to Calculate E[X^2]
```

```
% This script estimates E[X^2] for X ~ N(2, 2) using a Halton sequence.

%% 0. Setup
clear; clc;

%% 1. Define Target Distribution and Analytical Value
mu = 2;
sigma_sq = 2;
sigma = sqrt(sigma_sq);
analytical_E_X2 = sigma_sq + mu^2;

fprintf('The analytical value of E[X^2] is: %.4f\n', analytical_E_X2);

%% 2. Quasi-Monte Carlo Simulation with Halton Draws
% --- Simulation Parameters ---
R = 100000; % Number of points
num_dims = 1; % We only need one dimension for this problem

% --- Step 1: Create a Halton sequence ---
% A Halton sequence generates quasi-random numbers in the interval (0, 1).
p = haltonset(num_dims, 'Skip', 1); % Create the set, skip first point
u = net(p, R); % Generate R points from the sequence

% --- Step 2: Transform the uniform draws to be from N(mu, sigma^2) ---
% We use the inverse CDF (icdf) of the normal distribution. This is a
% standard way to turn uniform draws into draws from another distribution.
x = norminv(u, mu, sigma);

% --- Step 3: Calculate the function f(x) = x^2 and the estimate ---
f_x = x.^2;
halton_estimate = mean(f_x);

fprintf('Using R = %d Halton points...\n', R);
fprintf('The Halton draw estimate of E[X^2] is: %.4f\n', halton_estimate);
fprintf('The error is: %.4f\n', abs(halton_estimate - analytical_E_X2));

%% Results
The analytical value of E[X^2] is: 6.0000
Using R = 100000 Halton points...
The Halton draw estimate of E[X^2] is: 5.9989
The error is: 0.0011

```

Explanation of Changes:

- Instead of randn, we first create a haltonset.
- We generate R points from this set, which are uniformly distributed in (0, 1).
- To convert these uniform points into normally distributed points, we use the inverse transform sampling method via the norminv (Normal Inverse CDF) function.