Final Exam

CMU Monte Carlo Methods and Applications / Fall 2024

due: December 13, 2024 at 5:00pm EST

Instructions.

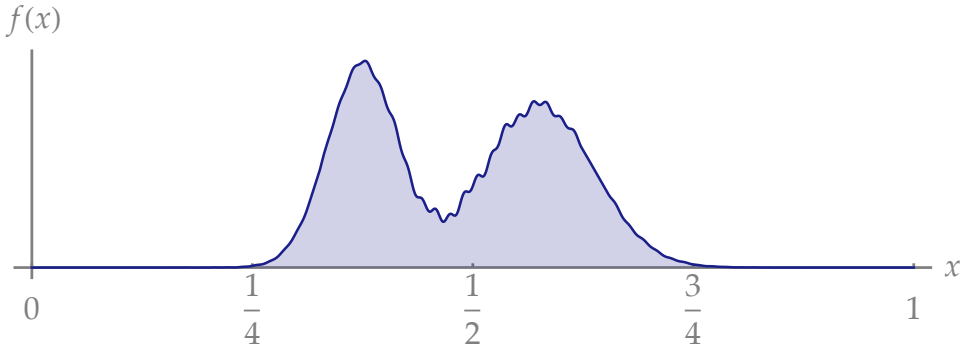
The final is not collaborative. You must solve it on your own, without help from anyone else (in person, or online). You may use whatever resources you can find online or in books; however, you may not post on discussion boards. You are not permitted to use AI (e.g., ChatGPT or other LLMs) for this exam, even though it was allowed for the midterm.

Instructors and TAs will be available on Discord to answer clarifying questions, but otherwise will not provide any help. There will be no office hours during finals week.

There is no late day for the exam; it must be turned in before the due date. There are four questions; each question is worth 25%. (Note that this distribution does *not* necessarily imply anything about the relative difficulty of the questions.)

Please submit your written and coding solutions through Gradescope, exactly as you submit ordinary homework. **You must comment your code in order to receive full credit.** Including comments also helps us assign partial credit when the code has bugs, but the intent is correct.

1 Variance Reduction



Problem 1.1 Suppose we want to estimate the integral of the function $f(x)$ plotted above over the domain $[0,1]$, using Monte Carlo integration. Assuming the cost of evaluating $f(x)$ is negligible, rank the following variance reduction strategies in terms of "least efficient" to "most efficient," and give a rationale for your ordering. Your ranking does not need to be a strict ordering, e.g., you can include ties if you think two methods will exhibit very similar performance. (Note: your justifications are more important than the specific ordering you pick—so please spell them out clearly!)

To ensure an apples-to-apples comparison, you should assume that all estimators use the same total number of function evaluations N. For example, an antithetic sampler will have half as many terms as a basic Monte Carlo estimator, since each term involves two function evaluations.

- **Antithetic sampling.** Generate samples in pairs $X_i, 1 - X_i$, where the X_i are i.i.d. samples of the uniform distribution on $[0,1]$. Then use these samples to form a standard antithetic estimate.
- **Importance sampling.** Prior to sampling, approximate $p(x)$ as an average of two normal distributions $g_1(x), g_2(x)$. (You do not need to account for the cost of precomputing this approximation.) To generate each sample X_i, flip a fair coin to decide which of the two normal distributions to sample, and draw a sample from the chosen distribution using a standard algorithm like Box-Muller. Then use these samples X_i in an importance sampled Monte Carlo estimator $\sum_{i=1}^{N} f(X_i)/(\frac{1}{2}g_1(X_i) + \frac{1}{2}g_2(X_i))$.
- **Control variates.** Letting g_1, g_2 be the same two Gaussians as before, integrate the function $g_1(x) + g_2(x)$ in closed form (easily done by hand), and subtract this value from a (basic) Monte Carlo estimate of the integral of the function $f(x) - (\frac{1}{2}g_1(x) + \frac{1}{2}g_2(x))$ over $[0,1]$. In short, apply a standard control variate estimator, using the sum of the two Gaussians as the approximating function.
- **Stratified Sampling.** Break the domain into four strata of equal size, and apply a standard stratified sampling estimator (i.e., draw one sample uniformly from within each stratum).

Note: you do not need to implement these methods! You just have to give a written argument for why you chose your given ordering. Of course, if you *want* to implement them, to increase your confidence in your answer, you're more than welcome to. But you don't need to turn in any code, and won't earn any additional credit for implementation on this problem.

2 MCMC Sampling Meets MC Integration

We briefly mentioned in class that there is a *strong law of large numbers for Markov chains*

$$\lim_{N\to\infty} \underbrace{\frac{1}{N}\sum_{k=0}^{N} f(X_k)}_{\widehat{I}_N^{\mathrm{MCMC}}} = \int_\Omega f(x)\pi(x)\,dx, \tag{1}$$

where $\pi(x)$ is the stationary distribution of our Markov chain $X_1, X_2, \ldots$, and f is a function on Ω with finite mean with respect to π. Hence, we can use samples generated via an MCMC sampler (Metropolis-Hastings, Langevin Monte Carlo, etc.) to estimate the integral on the right-hand side of Equation 1, or equivalently, to estimate the expected value $E_\pi[f]$ of f with respect to π.

Problem 2.1 When we first studied Monte Carlo integration, using i.i.d. samples, we said we could estimate any integral of the form

$$I := \int_\Omega g(x)\,dx \tag{2}$$

using, e.g., the basic estimator $\widehat{I}_N$, or the importance-sampled estimator $\widehat{I}_N^p$. But our MCMC-based estimator $\widehat{I}_N^{\mathrm{MCMC}}$ seems to require that the integral have a special form, namely, $\int_\Omega f(x)\pi(x)\,dx$. Is this just a superficial difference in the way we write the estimators? Or is it a fundamental limitation of MCMC-based integration? More precisely, can you use the samples $X_1, X_2, \ldots$ generated by an MCMC sampler to formulate an efficient estimator for a general integral I? If so, do it. If not, explain why it can't be done.

Problem 2.2 Suppose we let $f(x) = g(x)$, and a target (unnormalized) distribution $\pi(x) = 1$ for MCMC, so that $f(x)\pi(x) = g(x)$. Under what condition on the problem data do we get a valid estimator for I? If this condition is violated, what additional information would we need to formulate a valid estimator?

Problem 2.3 Now suppose we let $f(x) = 1$ and use a target (unnormalized) distribution $\pi(x) = g(x)$ for MCMC, so that again $f(x)\pi(x) = g(x)$. (This time you may also assume that g is nonnegative.) Under what condition on the problem data do we get a valid estimator for I? If this condition is violated, what additional information would we need to formulate a valid estimator?

3 Rejection Sampling vs. Uniform Proposals

Let $\pi(x)$ be a probability density on the domain $\Omega := [0,1]^d$ (in some dimension d), and assume we know the maximum value $\pi_{\max}$ of π. Consider two possible strategies for generating samples of π:

- **Rejection Sampling.** Sample a candidate point $\tilde{x}$ uniformly from Ω, and sample a value u uniformly from $[0,1]$. If $u < \pi(\tilde{x})/\pi_{\max}$, accept the sample $\tilde{x}$; otherwise, try again.
- **Metropolis-Hastings with Uniform Proposals.** Starting at any point $x \in \Omega$, sample a candidate point $\tilde{x}$ uniformly from Ω, and sample a value u uniformly from $[0,1]$. If $u < \pi(\tilde{x})/\pi(x)$, accept the sample $\tilde{x}$; otherwise, remain at x.

Problem 3.1 You may have noticed that, when we use uniform proposals, Metropolis-Hastings looks extremely similar to ordinary rejection sampling. Are they in fact the same algorithm? If so, explain why. If not, describe how their behavior differs, giving some pros and cons.

Now let's check if our analysis matches what we observe in practice, by implementing the two samplers. In particular:

Problem 3.2 Implement a method `sampleRejection(pi)` that returns a single sample drawn from the distribution proportional to a given function[1] $\pi : [0,1]^2 \to \mathbb{R}_{\geq 0}$, using rejection sampling.

Problem 3.3 Implement a method `sampleMHUniform(pi,x)` that returns a single sample drawn from the distribution proportional to a given function $\pi : [0,1]^2 \to \mathbb{R}_{\geq 0}$ using Metropolis-Hastings with uniform proposals. The argument `x` specifies the previous point in the chain.

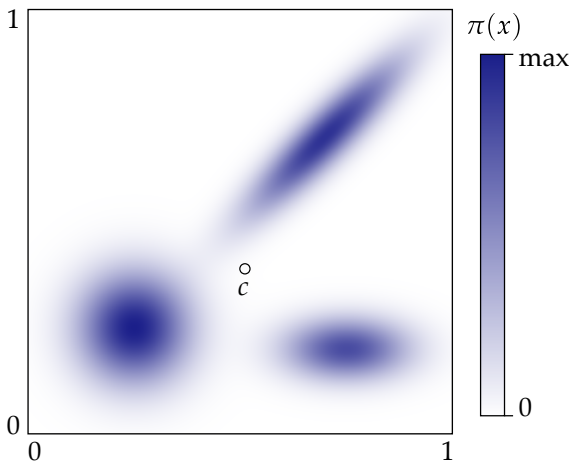
Problem 3.4 Use your two methods to generate 1024 samples of the specific function `pi(x)` given below, and plot the resulting samples in the unit square (making one plot for each method). The maximum of this function is found at the center of the first Gaussian component, with a value of $\pi(\mu_1) \approx 19.9094$.

Problem 3.5 Use your two methods to estimate the center of mass c of the given distribution, i.e., the point

$$\mu := E_\pi[x] = \int_0^1 \int_0^1 (x,y)\pi(x,y)\, dxdy.$$

The exact location of this point is given by the weighted average of the Gaussian means, namely, $\mu = \sum_{k=1}^3 w_i\mu_i / \sum_{k=1}^3 w_i \approx (0.511, 0.388)$. Plot the exact location of c within the unit square, as well as $m = 16$ estimates of c using each of your two methods. Each estimate should use 1024 samples. Make it clear which dots correspond to which quantities (e.g., via colors and/or styling). Do your results match your preconceptions about which of the two methods would work better?

[1] Note that in Python you can call a function passed into another function just as you would any other function. In this case, for instance, you can call `pi(x)`, where `x=[x1,x2]` is any point.




```python
import numpy as np
from scipy.stats import multivariate_normal

# means
mu1 = np.array([0.25, 0.25])
mu2 = np.array([0.7, 0.7])
mu3 = np.array([0.75, 0.2])

# covariance matrices
Sigma1 = 0.008 * np.array([[1, 0], [0, 1]])
Sigma2 = 0.015 * np.array([[1, 0.9], [0.9, 1]])
Sigma3 = 0.01 * np.array([[1, 0], [0, 0.25]])

# weights
w1 = 1.0
w2 = 0.75
w3 = 0.5

# target density
def pi(x):
    g1 = multivariate_normal.pdf(x, mean=mu1, cov=Sigma1)
    g2 = multivariate_normal.pdf(x, mean=mu2, cov=Sigma2)
    g3 = multivariate_normal.pdf(x, mean=mu3, cov=Sigma3)
    return w1*g1 + w2*g2 + w3*g3
```

4 Options Pricing

In our overview of Monte Carlo integration, we briefly mentioned the use of Monte Carlo methods in computational finance to predict the behavior of stock options. Here we're actually going to write some code to make some basic predictions.

4.1 Background: Stocks and Stock Options

Stocks. When you buy a *stock*, you are essentially buying a small slice of a company: if the company does well, the value of your stock increases; if the company does poorly, the stock value decreases. The game here is to "buy low, sell high," so that in the long term you make a profit[2]. For instance, if you bought 4000 shares of Apple's stock in 1997 for a price of \$0.13/share, and sold it in 2001 for a price of \$0.39/share, you would make a profit of $(4000 \text{ shares} \times \$0.39/\text{share}) - (4000 \text{ shares} \times \$0.13/\text{share}) = \$1040$, almost enough to buy a new Mac. If instead you waited until today, it would be worth about \$109 million USD.[3] Throughout we will use S_t to denote the price of a stock S at time t.

Stock Options. A *stock option* is different from a stock: you are not purchasing the stock itself. Instead, you are reserving the right to buy the stock at a later date, but with a pricing scheme agreed upon ahead of time. The simplest variant are "European stock options," where you pay a *premium* (usually a small fraction of the stock price) for the option to purchase a given number of shares for an agreed-upon *strike price*, at a fixed *maturity date*. For instance, you might pay a premium of \$10 for the option to purchase 100 shares of a stock at a strike price of \$5/share on the maturity date of December 31, 2025. If the stock actually has a market price of, say, \$6/share on that date, then you make a profit by exercising the option: you can purchase an asset worth $100 \text{ shares} \times \$6/\text{share} = \$600$ for only \$500, which means your *net profit* is $(\$600 - \$500) - \$10 = \$90$, i.e., the amount you make by being able to purchase stock "at a discount," minus the premium you originally paid for the option. If on the other hand the stock is worth less than \$5 on that date, you've effectively lost money: you paid a premium for the option to buy something at a loss (which you don't want to do!).

Asian Stock Options. An "Asian stock option," or more specifically, an *Asian call option*, is almost exactly like a European stock option, except that your profit is the difference between the strike price K agreed upon at the purchase date $t = 0$, and the *average* price $\overline{S}$ over the interval $[0, T]$ between the purchase date and the maturity date $t = T$ (rather than the market price S_T on the maturity date). Averaging the price over time helps to smooth out volatility, making Asian options depend less critically on the exact maturity date. More explicitly, if

$$\overline{S} := \frac{1}{T} \int_0^T S_t \, dt$$

is the average price of stock S over the interval $0 \leq t \leq T$, the net profit made on an Asian call option is

$$\max(0, \overline{S} - K) - C,$$

where C is the premium paid for the option. (Taking a max with zero models the fact that you don't want to exercise an option that would *lose* money!)

[2]The reason companies sell stock in the first place, and don't just keep all the profit for themselves, is that it's a way for them to raise money for things they need to do—not much different from taking a loan from a bank, except they're effectively getting a loan from a large collection of investors.

[3]These dates and prices are, sadly, based on a true story.

4.2 Simulating the Stock Market

So far we know "the rules of the trading game," but don't yet have a model that helps us predict how stocks might actually behave. On fairly common model is to assume that the stock price S_t exhibits *geometric Brownian motion (GBM)*, given by the stochastic differential equation

$$dS_t = \mu S_t\, dt + \sigma S_t dW_t, \tag{3}$$

where the drift μ models the *expected rate of return* (e.g., $\mu = 0.03$ represents a 3% growth rate), and the volatility σ models the *standard deviation of returns* (e.g., $\sigma = 0.15$ represents a volatility of 15%, relative to the current stock price). Notice that the deterministic part of this equation, $dS_t = \mu S_t\, dt$ is just our usual linear ODE for exponential growth (or decay). The additional term $\sigma S_t dW_t$ models *proportional volatility*: empirically, more expensive stocks experience bigger jumps in price; hence, we model volatility as a *percentage*, rather than an absolute amount.

Problem 4.1 Implement a method `simulateGBM( S0, mu, sigma, T, n )` that simulates the geometric Brownian motion in Equation 3 using the given initial value S_0 and parameters μ, σ over the interval $[0, T]$ using n time steps. This method should return prices as a list of values `[S1,...,Sn]`. You can assume that the stock prices S are in US dollars, μ is the annualized rate of return, σ is the annualized volatility, and that the time T is in years. Likewise, the values you return should be in US dollars. For simplicity, assume that the calendar year (CY) is evenly divided into 250 trading days; do not worry about the fact that trading does not occur on weekends, etc.

Problem 4.2 To get some sense of whether our model provides meaningful predictions, we can compare it to historical data for the S&P 500[4]. Run your code for a starting price of $S_0 = \$3824.14$, annualized rate of return[5] of $\mu = 0.221$, annualized volatility of $\sigma = 0.130$, and a duration of $T = 1$ year. Plot the result of several runs of this code against the provided dataset `GSPC.csv`, which gives the daily price of S&P 500 for each trading day of 2023. Do you obtain a plausible-looking simulation of what *could* have happened that year (even if the specific market fluctuations/prices don't match the historical data)?

4.3 Establishing a Fair Price

Now that we can simulate stock prices over time, we can try to decide a fair price for an option we want to sell (in particular, a fair premium C).

[4]The S&P 500 is an index of the biggest 500 stocks in the US, and is the most common such index used for long-term investments. E.g., if you are saving for retirement, there's a good chance you've already invested in the S&P 500!

[5]Note that we "cheated" here and matched μ and σ to the historical data from 2023, to better focus our experiment on the validity of the GBM model (rather than our ability to predict return rates!).

Problem 4.3 Implement a method `estimatePremium(S0, mu, sigma, T, n, m, K)` that uses `m` Monte Carlo samples to establish a fair premium C for an Asian call option with the same parameters as `simulateGBM()`, plus a fixed strike price of `K`. In particular, you can establish a fair price by setting the premium C equal to the expected payout $E[\max(0, \bar{S} - K)]$, since the net profit is then zero (i.e., the buyer pays as much as they expect to gain). For each run of GBM, you can compute the average price over time $\bar{S}$ as a simple mean of the simulated prices, $\frac{1}{n}\sum_{k=1}^{n} S_k$. This method should return both the estimated premium C, and an estimate of the standard deviation of a fair premium.

Problem 4.4 Estimate a premium for an Asian call option on the S&P 500, where the initial price is $S_0 = \$4500$, the strike price is fixed at $K = \$4600$, and the time period is $T = 1$ year. The historical annualized rate of return and volatility for the S&P 500 are about $\mu = 0.08$ (i.e., people earn about 8% on their investment each year), and $\sigma = 0.15$ (i.e., despite generally positive returns, the stock market can go up and down quite a bit in any given year—about 15% in a typical year). As before, assume that the calendar year is divided evenly into 250 trading days. Print the estimated premium C, and the estimated standard deviation. Do you trust that your simulation is accurate enough to make a reasonable prediction? Why or why not? Would you literally bet money on it? (You're at least betting your grade on it!)