IEOR 4404 Simulation **Fall 2024**

Homework 10

Instructor: Henry Lam

Problem 1 Suppose we use a control variate estimator to estimate $E[Y]$, where each replication outputs $Y - \beta^*(Z - \mu)$ with control variate Z whose mean $E[Z] = \mu$ is known exactly. Suppose that β^* is selected optimally, and we know its value. Furthermore, assume that it takes 1 unit of computer time to obtain each Y, but K units of computer time to obtain and compute each $Y - \beta^*(Z - \mu)$. What is the range of values of ρ, the correlation between Y and Z, as a function of K, for us to justify the use of this control variate method? Show your reasoning and derivation.

Problem 2 To estimate $E[Y]$, we use both Z_1 and Z_2 as control variates. Each replication of the control variate estimator is

$$W = Y - \beta_1(Z_1 - \mu_1) - \beta_2(Z_2 - \mu_2)$$

where $\mu_1 = E[Z_1]$ and $\mu_2 = E[Z_2]$ are known.

(a) Suppose that both the pair Y and Z_1, and the pair Y and Z_2, are positively correlated. Provide an example where $\beta_1 > 0$ and $\beta_2 < 0$, but $Var(W) < Var(Y)$ (note how this case is different from the case where only one control variate is used).

(b) Now suppose that Z_1 and Z_2 are independent. Can you still find an example as in part (a)? Provide your reasoning and calculations.

Problem 3 Consider a single-server queue, where arrivals follow a Poisson process with rate 2 per minute and service times are exponentially distributed with mean 1 minute. Let T_i denote the amount of time that customer i spends in the system. We are interested in using simulation to estimate $E[T_1 + \dots + T_{10}]$.

- Run a naive Monte Carlo simulation with 100 replications. Give a point estimate of the target quantity and an estimate of the variance per replication.
- Run a simulation with 100 replications using the control variate $\sum_{i=1}^{10} S_i$, where S_i is the ith service time. Give a point estimate of the target quantity and an estimate of the variance per replication, and describe the estimation improvement over the naive estimate in part (a).
- Run a simulation with 100 replications using the control variate $\sum_{i=1}^{10} S_i - \sum_{i=1}^{10} X_i$, where X_i is the interarrival time between the ith and $(i+1)$th arrivals. Give a point estimate of the target quantity and an estimate of the variance per replication, and describe the estimation quality compared to the naive estimate in part (a) and the control variate estimate in part (b).

Problem 4 A bank has a portfolio of $N = 100$ loans to N companies and want to evaluate its credit risk. Given that company n defaults, the loss for the bank X_n follows $N(3, 1)$. Defaults are described by indicator variables $D_1, D_2, \dots, D_N$, with a random background default probability P so that given $P = p$, D_n are i.i.d. Bernoulli random variables with parameter p. P itself follows a $Beta(1, 19)$ distribution with density function given by $f(p) = 19(1 - p)^{18}, 0 < p < 1$. Estimate $P(L > x)$, where $L = \sum_{n=1}^{N} D_n X_n$ is the total loss, and $x = 3E[L] = 3NE[P]E[X_n] = 3 \cdot 100 \cdot 0.05 \cdot 3 = 45$. Run naive Monte Carlo simulation and conditional Monte Carlo. In each approach, use 100 replications, and give a point estimate of the target quantity and an estimate of the variance per replication. Comment briefly on their performance comparisons.

Alexandre
Duhamel
afd2153
MSOR

HW10: Simulation

Problem 1:

- Estimator: $\hat{Y} = Y - \beta(Z - \mu)$

  Best β to reduce variance: $\beta^* = \frac{cov(Y,Z)}{Var(Z)}$

  So $Var(\hat{Y}) = Var(Y)(1-\rho^2)$

  ρ: corr(Y,Z)

- Naive estimator: - time per replication: $t_Y = 1$

  - total replications to T: $m_Y = T$

  So $Var_{naive} = \frac{Var(Y)}{m_Y} = \frac{Var(Y)}{T}$

- Control variate estimator: - time per replication: $t_{cv} = K$

  - total replications to T: $m_{cv} = \frac{T}{K}$

  So $Var_{cv} = \frac{Var(\hat{Y})}{m_{cv}} = \frac{Var(Y)}{T}(1-\rho^2)K$

We want:

$Var_{cv} < Var_{naive}$

$\Rightarrow \frac{Var(Y)}{T}(1-\rho^2)K < \frac{Var(Y)}{T}$

$\Rightarrow \rho^2 > 1 - \frac{1}{K}$

- To reduce variance, we must have $\rho > \sqrt{1-\frac{1}{K}}$: if
  - $K \to +\infty$ we have $\rho > 1$, control variate doesn't work
  - $K = 1$ we have $\rho > 0$, cv works for positively correlated vars
  - $K < 1$, doesn't make sense as naive quicker than cv

Problem 2:

a) The optimal coefficients minimizing $Var(W)$ are: $\beta = \begin{pmatrix}\beta_1\\ \beta_2\end{pmatrix} = \Gamma^{-1} cov(Y,Z)$ (cov matrix of Z)

Let's take $\begin{cases} Var(Y) = 1 \\ Var(Z_1) = 1 \\ Var(Z_2) = 1 \end{cases}$ and $\begin{cases} cov(Y,Z_1) = 0.9 > 0 \\ cov(Y,Z_2) = 0.8 > 0 \\ cov(Z_1,Z_2) = 0.9 \text{ high corr} \end{cases}$ so that $\Gamma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}$

We get $\beta = \begin{pmatrix} 0.9474 \\ -0.053 \end{pmatrix} \begin{matrix} > 0 \\ < 0 \end{matrix}$

And:

$Var(W) = Var(Y - \beta_1(Z_1 - \mu_1) - \beta_2(Z_2 - \mu_2))$

$= Var(Y) + \beta_1^2 Var(Z_1) + \beta_2^2 Var(Z_2) + 2\beta_1\beta_2 cov(Z_1,Z_2) - 2\beta_1 cov(Y,Z_1) - 2\beta_2 cov(Y,Z_2)$

$= 0.1895 < Var(Y) = 1$

b) Suppose $Z_1 \perp Z_2 \rightarrow \text{cov}(Z_1, Z_2) = 0$

We keep $\begin{cases} \text{cov}(Y, Z_1) > 0 \\ \text{cov}(Y, Z_2) > 0 \end{cases}$ so $\Gamma = \begin{pmatrix} Var(Z_1) & 0 \\ 0 & Var(Z_2) \end{pmatrix}$

so $\beta = \begin{pmatrix} \frac{\text{cov}(Y, Z_1)}{Var(Z_1)} \\ \frac{\text{cov}(Y, Z_2)}{Var(Z_2)} \end{pmatrix}$, since $Var(Z_i) > 0$: $\beta_i = \frac{\text{positive}}{\text{positive}} = \text{positive} > 0$

$= E[\underbrace{(Z_i - \mu)^2}_{\geq 0}]$

and $\text{cov}(Y, Z_i) > 0$

So β_1 and β_2 are positive (we can't get $\beta_1 > 0$ and $\beta_2 < 0$ when $Z_1 \perp Z_2$ and $\text{cov}(Y, Z_i) > 0$)

Problem 3:

M | M | 1 with $\lambda = 2$ and $\mu = 1$

λ μ

a) Estimate $E[\sum_{i=1}^{10} T_i]$ for 100 repetitions

Pseudo-code:
1. Empty system at $t = 0$
2. For each repetition: $\begin{cases} \text{Generate inter-arrival times } A_i \sim \exp(\lambda) \text{ and } t_i = t_{i-1} + A_{i-1} \\ \text{Generate service times } S_i \sim \exp(\mu) \\ \text{Customer completion time } \begin{cases} C_i = \max(t_i, C_{i-1}) + S_i \\ T_i = C_i - t_i \end{cases} \\ \text{Total time in system: } T_{total} = \sum_{i=1}^{10} T_i \end{cases}$
3. Return $\begin{cases} \hat{E}[T_{total}] = \frac{1}{100} \sum_{k=1}^{100} T_{total}^{(k)} \\ \hat{Var}(T_{total}) = \frac{1}{99} \sum_{k=1}^{100} (T_{total}^{(k)} - \hat{E}[T_{total}])^2 \end{cases}$

```
total_time_in_system_list = []

for replication in range(num_replications):
    arrival_times = np.cumsum(np.random.exponential(scale=1/lambda_arrival, size=num_customers))
    service_times = np.random.exponential(scale=1/mu_service, size=num_customers)
    start_service_times = np.zeros(num_customers)
    completion_times = np.zeros(num_customers)
    time_in_system = np.zeros(num_customers)

    start_service_times[0] = arrival_times[0]
    completion_times[0] = start_service_times[0] + service_times[0]
    time_in_system[0] = completion_times[0] - arrival_times[0]

    for i in range(1, num_customers):
        start_service_times[i] = max(arrival_times[i], completion_times[i-1])
        completion_times[i] = start_service_times[i] + service_times[i]
        time_in_system[i] = completion_times[i] - arrival_times[i]

    total_time_in_system = np.sum(time_in_system)
    total_time_in_system_list.append(total_time_in_system)

point_estimate_naive = np.mean(total_time_in_system_list)
variance_per_replication_naive = np.var(total_time_in_system_list, ddof=1)

print("Part (a): Naive Monte Carlo Simulation")
print(f"Point Estimate: {point_estimate_naive:.4f} minutes")
print(f"Variance per Replication: {variance_per_replication_naive:.4f}")
```

```
Part (a): Naive Monte Carlo Simulation
Point Estimate: 38.2929 minutes
Variance per Replication: 457.9150
```

b) Estimate $E[\sum_{i=1}^{10} T_i]$ for 100 repetitions

Pseudo-code:
1. Empty system at $t=0$
2. For each repetition: $\begin{cases} \text{Generate inter-arrival times } A_i \sim \exp(\lambda) \text{ and } t_i = t_{i-1} + A_{i-1} \\ \text{Generate service times } S_i \sim \exp(\mu) \\ \text{Customer completion time } \begin{cases} C_i = \max(t_i, C_{i-1}) + S_i \\ T_i = C_i - t_i \end{cases} \\ \text{Total time in system: } T_{total} = \sum_{i=1}^{10} T_i \\ S_{total} += S_i \end{cases}$
3. Compute $\hat{\beta} = cov(T_{total}, S_{total}) / \hat{Var}(S_{total})$
4. Return $\begin{cases} \hat{E}[T_{total}] = \frac{1}{100} \sum_{k=1}^{100} \left(T_{total}^{(k)} - \hat{\beta}\left(S_{total}^{(k)} - E[S_{total}]\right)\right) \\ \hat{Var}(T_{total}) = \frac{1}{99} \sum_{k=1}^{100} \left(T_{total}^{(k)} - \hat{\beta}\left(S_{total}^{(k)} - E[S_{total}]\right) - \hat{E}[T_{total}]\right)^2 \end{cases}$

```python
expected_total_service_time = num_customers * (1 / mu_service)

total_time_in_system_list = []
total_service_time_list = []

for replication in range(num_replications):
    arrival_times = np.cumsum(np.random.exponential(scale=1/lambda_arrival, size=num_customers))
    service_times = np.random.exponential(scale=1/mu_service, size=num_customers)
    start_service_times = np.zeros(num_customers)
    completion_times = np.zeros(num_customers)
    time_in_system = np.zeros(num_customers)

    start_service_times[0] = arrival_times[0]
    completion_times[0] = start_service_times[0] + service_times[0]
    time_in_system[0] = completion_times[0] - arrival_times[0]

    for i in range(1, num_customers):
        start_service_times[i] = max(arrival_times[i], completion_times[i-1])
        completion_times[i] = start_service_times[i] + service_times[i]
        time_in_system[i] = completion_times[i] - arrival_times[i]

    total_time_in_system = np.sum(time_in_system)
    total_service_time = np.sum(service_times)

    total_time_in_system_list.append(total_time_in_system)
    total_service_time_list.append(total_service_time)

total_time_in_system_array = np.array(total_time_in_system_list)
total_service_time_array = np.array(total_service_time_list)

covariance = np.cov(total_time_in_system_array, total_service_time_array, ddof=1)[0, 1]
variance_service_time = np.var(total_service_time_array, ddof=1)
beta_optimal = covariance / variance_service_time

adjusted_estimates = total_time_in_system_array - beta_optimal * (total_service_time_array - expe

point_estimate_cv = np.mean(adjusted_estimates)
variance_per_replication_cv = np.var(adjusted_estimates, ddof=1)

print("\nPart (b): Control Variate with Total Service Time")
print(f"Optimal Beta: {beta_optimal:.4f}")
print(f"Point Estimate: {point_estimate_cv:.4f} minutes")
print(f"Variance per Replication: {variance_per_replication_cv:.4f}")

variance_reduction = ((variance_per_replication_naive - variance_per_replication_cv) / variance_p
print(f"Variance Reduction: {variance_reduction:.2f}%")
```

```
Part (b): Control Variate with Total Service Time
Optimal Beta: 5.2783
Point Estimate: 36.2446 minutes
Variance per Replication: 91.9840
Variance Reduction: 79.91%
```

c) Estimate $E[\sum_{i=1}^{10} T_i]$ for 100 repetitions

Pseudo-code:

1. Empty system at $t=0$
2. For each repetition: $\begin{cases} \text{Generate inter-arrival times } A_i \sim \exp(\lambda) \text{ and } t_i = t_{i-1} + A_{i-1} \\ \text{Generate service times } S_i \sim \exp(\mu) \\ \text{Customer completion time } \begin{cases} C_i = \max(t_i, C_{i-1}) + S_i \\ T_i = C_i - t_i \end{cases} \\ \text{Total time in system: } T_{total} = \sum_{i=1}^{10} T_i \\ S_{total} += S_i,\ A_{total} += A_i,\ V^{(k)} = S_{total}^{(k)} - A_{total}^{(k)} \end{cases}$
3. Compute $\hat{\beta} = cov(T_{total}, V) / \hat{Var}(V)$
4. Return $\begin{cases} \hat{E}[T_{total}] = \frac{1}{100} \sum_{k=1}^{100} \left(T_{total}^{(k)} - \hat{\beta}\left(V^{(k)} - E[V]\right)\right) \\ \hat{Var}(T_{total}) = \frac{1}{99} \sum_{k=1}^{100} \left(T_{total}^{(k)} - \hat{\beta}\left(V^{(k)} - E[V]\right) - \hat{E}[T_{total}]\right)^2 \end{cases}$

```python
expected_total_service_time = num_customers * (1 / mu_service)
expected_total_interarrival_time = num_customers * (1 / lambda_arrival)
expected_control_variate = expected_total_service_time - expected_total_interarrival_time

total_time_in_system_list = []
control_variate_list = []

for replication in range(num_replications):
    interarrival_times = np.random.exponential(scale=1/lambda_arrival, size=num_customers)
    arrival_times = np.cumsum(interarrival_times)
    service_times = np.random.exponential(scale=1/mu_service, size=num_customers)
    start_service_times = np.zeros(num_customers)
    completion_times = np.zeros(num_customers)
    time_in_system = np.zeros(num_customers)

    start_service_times[0] = arrival_times[0]
    completion_times[0] = start_service_times[0] + service_times[0]
    time_in_system[0] = completion_times[0] - arrival_times[0]

    for i in range(1, num_customers):
        start_service_times[i] = max(arrival_times[i], completion_times[i-1])
        completion_times[i] = start_service_times[i] + service_times[i]
        time_in_system[i] = completion_times[i] - arrival_times[i]

    total_time_in_system = np.sum(time_in_system)
    total_service_time = np.sum(service_times)
    total_interarrival_time = np.sum(interarrival_times)
    control_variate = total_service_time - total_interarrival_time

    total_time_in_system_list.append(total_time_in_system)
    control_variate_list.append(control_variate)

total_time_in_system_array = np.array(total_time_in_system_list)
control_variate_array = np.array(control_variate_list)

covariance = np.cov(total_time_in_system_array, control_variate_array, ddof=1)[0, 1]
variance_control_variate = np.var(control_variate_array, ddof=1)
beta_optimal = covariance / variance_control_variate

adjusted_estimates = total_time_in_system_array - beta_optimal * (control_variate_array - expec

point_estimate_cv = np.mean(adjusted_estimates)
variance_per_replication_cv = np.var(adjusted_estimates, ddof=1)

print("\nPart (c): Control Variate with Total Service Time minus Total Interarrival Time")
print(f"Optimal Beta: {beta_optimal:.4f}")
print(f"Point Estimate: {point_estimate_cv:.4f} minutes")
print(f"Variance per Replication: {variance_per_replication_cv:.4f}")

variance_reduction = ((variance_per_replication_naive - variance_per_replication_cv) / variance
print(f"Variance Reduction compared to Naive Simulation: {variance_reduction:.2f}%")

variance_reduction_b = ((variance_per_replication_cv - variance_per_replication_cv) / variance_
print(f"Variance Reduction compared to Part (b): {variance_reduction_b:.2f}%")
```

```
Part (c): Control Variate with Total Service Time minus Total Interarrival Time
Optimal Beta: 4.4266
Point Estimate: 34.7598 minutes
Variance per Replication: 81.8711
Variance Reduction compared to Naive Simulation: 82.12%
Variance Reduction compared to Part (b): 0.00%
```

Problem 4:

```python
# Naive Monte Carlo Simulation

indicator_values = []

for replication in range(num_replications):
    P = np.random.beta(alpha, beta_param)

    D_n = np.random.binomial(1, P, size=N)

    num_defaults = np.sum(D_n)
    X_defaults = np.random.normal(loc=3, scale=1, size=num_defaults)

    X_n = np.zeros(N)
    X_n[D_n == 1] = X_defaults

    L = np.sum(X_n)

    indicator = 1 if L > x else 0
    indicator_values.append(indicator)

point_estimate_naive = np.mean(indicator_values)

variance_naive = point_estimate_naive * (1 - point_estimate_naive)

print("Naive Monte Carlo Simulation")
print(f"Point Estimate of P(L > {x}): {point_estimate_naive:.4f}")
print(f"Variance per Replication: {variance_naive:.6f}")
```

```
Naive Monte Carlo Simulation
Point Estimate of P(L > 45): 0.0300
Variance per Replication: 0.029100
```

```python
# Conditional Monte Carlo Simulation

PL_given_P_values = []

for replication in range(num_replications):
    P = np.random.beta(alpha, beta_param)

    mu_L_given_P = 3 * N * P

    var_L_given_P = N * P * (1 + 9 * (1 - P))

    sigma_L_given_P = np.sqrt(var_L_given_P)

    if sigma_L_given_P > 0:
        z = (x - mu_L_given_P) / sigma_L_given_P
        PL_given_P = 1 - norm.cdf(z)
    else:
        PL_given_P = 1.0 if mu_L_given_P > x else 0.0

    PL_given_P_values.append(PL_given_P)

point_estimate_conditional = np.mean(PL_given_P_values)

variance_conditional = np.var(PL_given_P_values, ddof=1)

print("\nConditional Monte Carlo Simulation")
print(f"Point Estimate of P(L > {x}): {point_estimate_conditional:.4f}")
print(f"Variance per Replication: {variance_conditional:.6f}")
```

```
Conditional Monte Carlo Simulation
Point Estimate of P(L > 45): 0.0436
Variance per Replication: 0.016077
```

```python
# Variance Reduction
variance_reduction = ((variance_naive - variance_conditional) / variance_naive) * 100

print("\nPerformance Comparison")
print(f"Variance Reduction: {variance_reduction:.2f}%")
```

```
Performance Comparison
Variance Reduction: 44.75%
```

- The conditional version of MC algorithm outperforms the naive version by achieving a variance reduction of 44.75%, resulting in more consistant and reliable estimates.
The results demonstrate the efficiency of leveraging analytical conditioning, making the conditional method preferable for estimating rare-event probabilities.