SDS 5531 Homework 2

Gifty Osei

2024-09-20

Remark: If you would like to insert images for your handwritten part into this file, please refer to this article.

Problem 1. Multivariate Normal

Suppose that $\mathbf{Z}$ follows a multivariate normal distribution $N(\mu, \Sigma)$. And a partition of $\mathbf{Z}$ is given as

$$\mathbf{Z} = \begin{pmatrix} \mathbf{X} \\ \mathbf{Y} \end{pmatrix}$$

with corresponding partitions of the mean vector and covariance matrix given as

$$\mu = \begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix} \text{ and } \Sigma = \begin{pmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{XY}^T & \Sigma_{YY} \end{pmatrix}.$$

The multivariate normal theory states that the conditional distribution of $\mathbf{X}|\mathbf{Y} = \mathbf{y}$ is a multivariate normal with mean

$$\mu_{X|Y} = \mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(\mathbf{y} - \mu_Y)$$

and covariance matrix

$$\Sigma_{X|Y} = \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^T.$$

1. Write an R function to simulate from the multivariate normal distribution $N(\mu_{X|Y}, \Sigma_{X|Y})$, i.e. the conditional distribution of $\mathbf{X}|\mathbf{Y} = \mathbf{y}$, using the standard approach discussed in class using Cholesky decomposition. That is, generate $\mathbf{X} = \mu_{X|Y} + \Sigma_{X|Y}^{1/2} N(0, I)$, where $\Sigma_{X|Y}^{1/2}$ is from the Cholesky decomposition of $\Sigma_{X|Y}$. The input of your function should include the sample size, $\mathbf{y}$, μ and Σ.

Solution:

```r
# simulate from multivariate normal using Cholesky decomposition

mvn_conditional <- function(n, y, mu, Sigma) {
    # Determine dimensions
    p <- length(mu)  # Total dimension
    q <- length(y)  # Dimension of Y
    k <- p - q  # Dimension of X

    # Partition mu
    mu_X <- mu[1:k]
    mu_Y <- mu[(k + 1):p]

    # Partition Sigma
    Sigma_XX <- Sigma[1:k, 1:k]
    Sigma_XY <- Sigma[1:k, (k + 1):p]
    Sigma_YX <- Sigma[(k + 1):p, 1:k]
    Sigma_YY <- Sigma[(k + 1):p, (k + 1):p]

    # Compute the inverse of Sigma_YY
    Sigma_YY_inv <- solve(Sigma_YY)

    # Compute the conditional mean
    mu_cond <- mu_X + Sigma_XY %*% Sigma_YY_inv %*% (y - mu_Y)

    # Compute the conditional covariance matrix
    Sigma_cond <- Sigma_XX - Sigma_XY %*% Sigma_YY_inv %*% Sigma_YX

    # Ensure the covariance matrix is symmetric
    Sigma_cond <- (Sigma_cond + t(Sigma_cond))/2

    # Perform Cholesky decomposition
    L <- chol(Sigma_cond)

    # Generate standard normal random variables
    Z <- matrix(rnorm(n * k), nrow = n, ncol = k)

    # Generate samples from the conditional distribution
    x_samples <- Z %*% t(L) + matrix(rep(mu_cond, each = n), nrow = n, byrow = TRUE)

    return(x_samples)
}
```

Simulated Example under Cholesky:

```r
# Define parameters
n <- 1000
mu <- c(1, 2, 3, 4)

# Generate a positive definite covariance matrix
C <- matrix(rnorm(16), nrow = 4)
Sigma <- crossprod(C)  # Sigma = C^T * C, which is positive definite

# Partition the mean vector and specify y
y <- c(3.5, 4.5)  # Assuming the last two variables are Y

# Simulate samples
x_samples <- mvn_conditional(n, y, mu, Sigma)

# View the first few samples
```

```r
kable(head(x_samples), caption = "Cholesky Samples", latex_options = c("hold_position",
    "striped"))
```

Table 1: Cholesky Samples

0.5141765	0.6317567
0.9142202	0.7069134
-0.0090426	0.2816815
2.0902876	0.7455789
4.4798499	0.7061998
4.9754389	0.2079093

2. An alternative approach proposed by Hoffman and Ribak (1991) runs differently. It first generates

$$\mathbf{Z} = \begin{pmatrix} \mathbf{X} \\ \mathbf{Y} \end{pmatrix} \sim N(\mu, \Sigma)$$

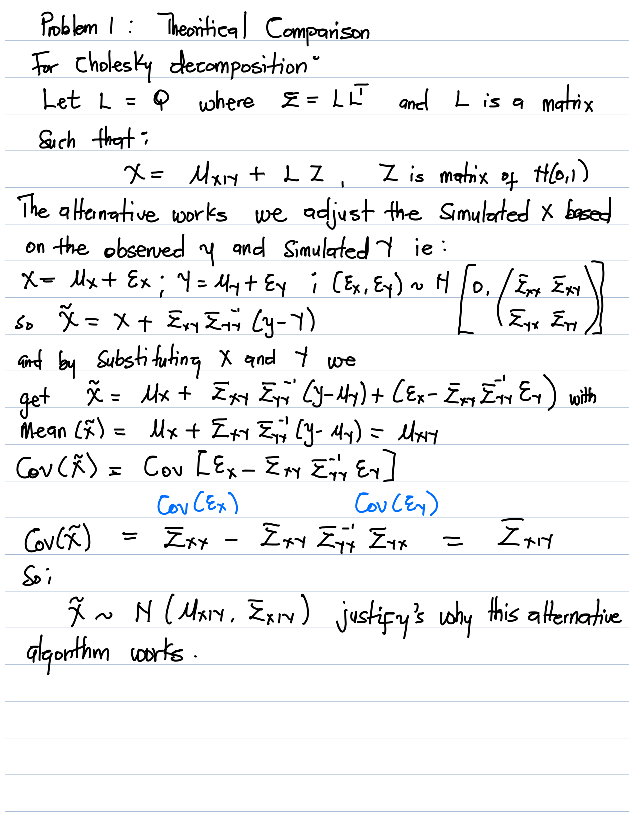
and then get

$$\tilde{\mathbf{X}} = \mathbf{X} + \Sigma_{XY}\Sigma_{YY}^{-1}(\mathbf{y} - \mathbf{Y}).$$

Then $\tilde{\mathbf{X}}$ is from $N(\mu_{X|Y}, \Sigma_{X|Y})$. Justify the validity of this algorithm and give an R implementation. Next comment on its computational cost by comparing with the standard approach above. You can compare from both a theoretical and empirical perspective. For the empirical perspective, you may time your implementations of the two methods for a specific simulation with a relatively large sample size.

Solution:

```
knitr::include_graphics("D:/WashU/First Year/Sem1/SDS5531_StatsComputing/Homework/New folder/Prob1a.pdf
```



Problem 1 : Theoritical Comparison

For Cholesky decomposition:

Let $L = Q$ where $\Sigma = LL^T$ and L is a matrix

Such that:

$X = \mu_{X|Y} + LZ$, Z is matrix of $N(0,1)$

The alternative works we adjust the Simulated X based on the observed y and Simulated Y ie:

$X = \mu_X + \varepsilon_X$; $Y = \mu_Y + \varepsilon_Y$; $(\varepsilon_X, \varepsilon_Y) \sim N\left[0, \begin{pmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{pmatrix}\right]$

So $\tilde{X} = X + \Sigma_{XY}\Sigma_{YY}^{-1}(y - Y)$

and by Substituting X and Y we get $\tilde{X} = \mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(y - \mu_Y) + (\varepsilon_X - \Sigma_{XY}\Sigma_{YY}^{-1}\varepsilon_Y)$ with

Mean $(\tilde{X}) = \mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(y - \mu_Y) = \mu_{X|Y}$

$\mathrm{Cov}(\tilde{X}) = \mathrm{Cov}[\underbrace{\varepsilon_X}_{\mathrm{Cov}(\varepsilon_X)} - \Sigma_{XY}\Sigma_{YY}^{-1}\underbrace{\varepsilon_Y}_{\mathrm{Cov}(\varepsilon_Y)}]$

$\mathrm{Cov}(\tilde{X}) = \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX} = \Sigma_{X|Y}$

So:

$\tilde{X} \sim N(\mu_{X|Y}, \Sigma_{X|Y})$ justify's why this alternative algorithm works.

Figure 1: Problem 1, Question 2

The Alternative approach invloves simulating from both X and Y and for smaller n's, so the approach performs poorly but for the Cholesky approach, we only have to simulate from a single variable X. So in general, the Cholesky decomposition will performs better for smaller n's.

Simulate from multivariate normal using the Hoffman and Ribak method:

```r
# simulate from multivariate normal using the Hoffman and Ribak method

MVN_conditional_alternative <- function(n, y, mu, Sigma) {
    # Determine dimensions
    p <- length(mu)  # Total dimension
    q <- length(y)  # Dimension of Y
    k <- p - q  # Dimension of X

    # Partition mu
    mu_X <- mu[1:k]
    mu_Y <- mu[(k + 1):p]

    # Partition Sigma
    Sigma_XX <- Sigma[1:k, 1:k]
    Sigma_XY <- Sigma[1:k, (k + 1):p]
    Sigma_YY <- Sigma[(k + 1):p, (k + 1):p]

    # Compute the inverse of Sigma_YY
    Sigma_YY_inv <- solve(Sigma_YY)

    # Compute K = Sigma_XY * Sigma_YY_inv
    K <- Sigma_XY %*% Sigma_YY_inv

    # Simulate Z = (X; Y) ~ N(mu, Sigma)
    library(MASS)
    Z <- mvrnorm(n, mu, Sigma)
    X <- Z[, 1:k, drop = FALSE]
    Y <- Z[, (k + 1):p, drop = FALSE]

    # Adjust X
    delta <- (matrix(y, nrow = n, ncol = q, byrow = TRUE) - Y) %*% t(K)
    X_tilde <- X + delta

    return(X_tilde)
}
```

Alternative Example:

```r
# Define parameters
n <- 1000
mu <- c(1, 2, 3, 4)
# Generate a positive definite covariance matrix
L <- matrix(rnorm(16), nrow = 4)
Sigma <- crossprod(L)  # Sigma = L^T * L, which is positive definite

y <- c(3.5, 4.5)

# Simulate samples
samples_alternative <- MVN_conditional_alternative(n, y, mu, Sigma)
```

```r
# View the first few samples


kable(head(samples_alternative), caption = "Alternative Samples", latex_options = c("hold_position",
    "striped"))
```

Table 2: Alternative Samples

0.9014796	0.9770038
-1.0992492	2.3964634
2.3381629	2.2173343
1.2695642	2.9236243
1.7584016	3.7434883
1.2282920	2.5551529

Time Comparison:

```r
# comparison of the two methods

# Define parameters
n <- 1e+05
mu <- rep(0, 30)
Sigma <- diag(30)
y <- rep(0, 15)

# Adjust dimensions
k <- 15  # Dimension of Y
p <- length(mu)  # Total dimension

# Generate a positive definite covariance matrix
set.seed(132)
A <- matrix(rnorm(p^2), nrow = p)
Sigma <- crossprod(A)

# Timing the two methods
time_comp <- microbenchmark(cholesky = mvn_conditional(n, y, mu, Sigma), alternative = MVN_conditional_a
    y, mu, Sigma), times = 10)

## Summarize
kable(summary(time_comp), caption = "Time to Compute", latex_options = c("hold_position",
    "striped"))
```

Table 3: Time to Compute

expr	min	lq	mean	median	uq	max	neval
cholesky	92.6696	94.6665	104.1333	101.0474	105.8698	133.7535	10
alternative	260.1241	306.3156	333.7477	335.7081	376.3763	394.7736	10

```
# N <- 200 n <- 100000 mu <- rep(0, 50) Sigma <- diag(50) y <- rep(0, 25)
# system.time(for (i in 1:N) mvn_conditional(n, y, mu, Sigma)) #
# system.time(for (i in 1:N) MVN_conditional_alternative(n, y, mu, Sigma))
```

Table 3: From the descriptive summary, we can see that Cholesky decomposition is much faster to compute and simulate the multivariate normal random variables than the alternative approach proposed by Hoffman and Ribak (1991).The Alternative method directly computes the conditional distribution $X \mid Y = y$, avoiding the need to simulate the full joint distribution. Then, this results in potentially lower computational cost when the dimension of Y is much smaller than the total dimension "n". Cholesky decomposition first samples from the full multivariate distribution and then extracts the conditional distribution. However, the method may involve more computation when dealing with high-dimensional data. For large datasets, the Hoffman and Ribak method should be faster as it bypasses the need to generate and then discard the variables in Y, especially when X is much smaller than Y.

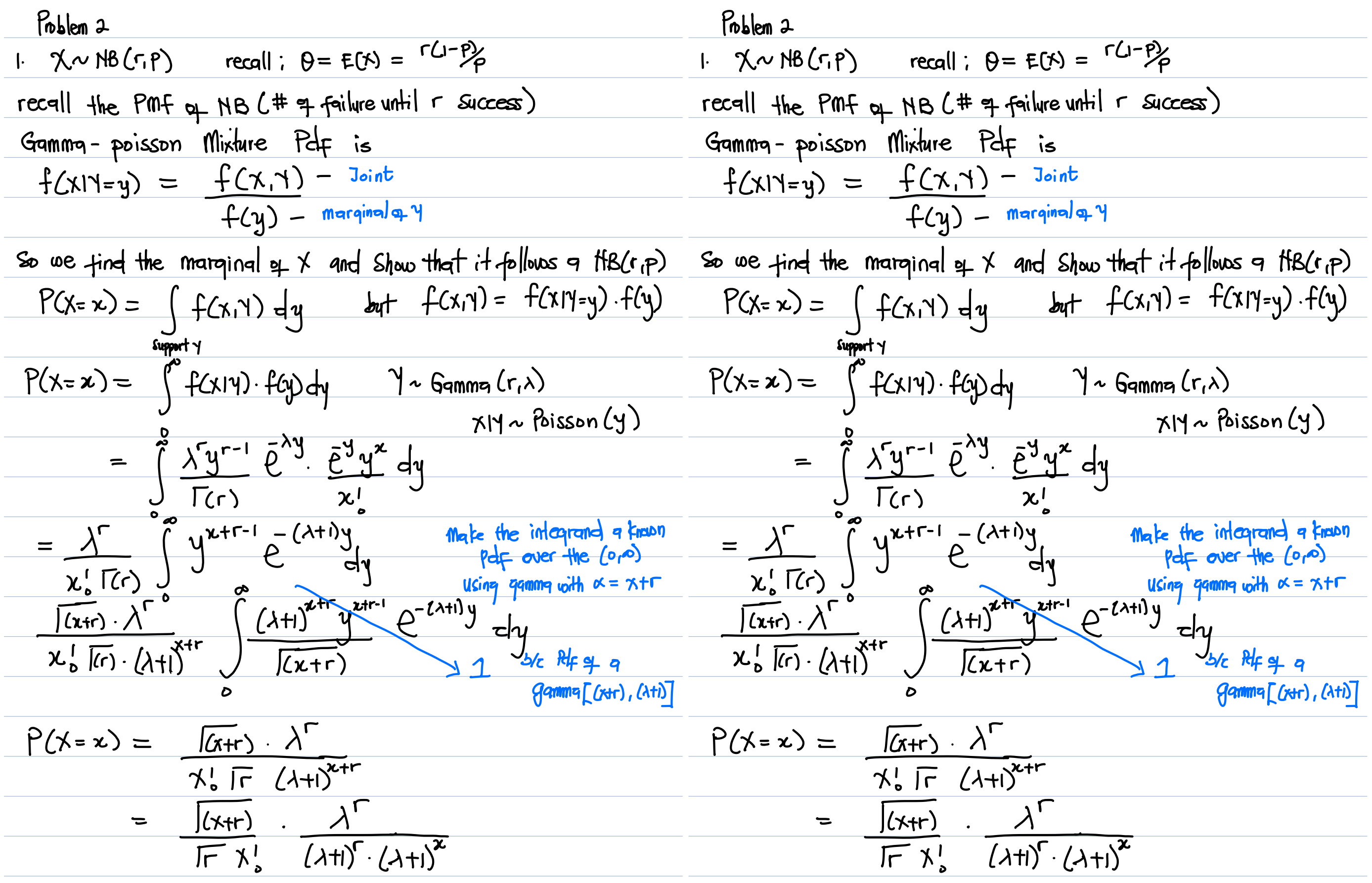
Problem 2. Rao-Blackwellization

Consider Monte Carlo approximation to the mean of a negative binomial random variable $X \sim NB(r, p)$. (Note that the true value of the expectation is $\theta = E(X) = r(1-p)/p$.)

1. Prove that a mixture representation of $NB(r, p)$ is $X|Y = y \sim Poisson(y)$ and $Y \sim gamma(r, \frac{p}{1-p})$.

Solution:

```
knitr::include_graphics(c("D:/WashU/First Year/Sem1/SDS5531_StatsComputing/Homework/New folder/Prob21a.p
    "D:/WashU/First Year/Sem1/SDS5531_StatsComputing/Homework/New folder/Prob21a.pdf"))
```



Problem 2

1. $X \sim NB(r,p)$ recall; $\theta = E(X) = r(1-p)/p$

recall the PMF of NB (# of failure until r success)

Gamma - poisson Mixture Pdf is

$$f(x|Y=y) = \frac{f(x,Y) - \text{Joint}}{f(y) - \text{marginal of } Y}$$

So we find the marginal of X and show that it follows a NB(r,p)

$$P(X=x) = \int_{\text{support } Y} f(x,Y)\,dy \qquad \text{but } f(x,Y) = f(x|Y=y)\cdot f(y)$$

$$P(X=x) = \int_0^\infty f(x|Y)\cdot f(y)\,dy \qquad Y \sim \text{Gamma}(r,\lambda) \quad x|Y \sim \text{Poisson}(y)$$

$$= \int_0^\infty \frac{\lambda^r y^{r-1}}{\Gamma(r)} e^{-\lambda y} \cdot \frac{e^{-y} y^x}{x!}\,dy$$

$$= \frac{\lambda^r}{x!\,\Gamma(r)} \int_0^\infty y^{x+r-1} e^{-(\lambda+1)y}\,dy$$

make the integrand a known pdf over the $(0,\infty)$ using gamma with $\alpha = x+r$

$$\frac{\Gamma(x+r)\cdot\lambda^r}{x!\,\Gamma(r)\cdot(\lambda+1)^{x+r}} \int_0^\infty \frac{(\lambda+1)^{x+r} y^{x+r-1} e^{-(\lambda+1)y}}{\Gamma(x+r)}\,dy \rightarrow 1$$

b/c Pdf of a gamma$[(x+r),(\lambda+1)]$

$$P(X=x) = \frac{\Gamma(x+r)\cdot\lambda^r}{x!\,\Gamma r\,(\lambda+1)^{x+r}}$$

$$= \frac{\Gamma(x+r)}{\Gamma r\; x!} \cdot \frac{\lambda^r}{(\lambda+1)^r\cdot(\lambda+1)^x}$$

Problem 2

1. $X \sim NB(r,p)$ recall; $\theta = E(X) = r(1-p)/p$

recall the PMF of NB (# of failure until r success)

Gamma - poisson Mixture Pdf is

$$f(x|Y=y) = \frac{f(x,Y) - \text{Joint}}{f(y) - \text{marginal of } Y}$$

So we find the marginal of X and show that it follows a NB(r,p)

$$P(X=x) = \int_{\text{support } Y} f(x,Y)\,dy \qquad \text{but } f(x,Y) = f(x|Y=y)\cdot f(y)$$

$$P(X=x) = \int_0^\infty f(x|Y)\cdot f(y)\,dy \qquad Y \sim \text{Gamma}(r,\lambda) \quad x|Y \sim \text{Poisson}(y)$$

$$= \int_0^\infty \frac{\lambda^r y^{r-1}}{\Gamma(r)} e^{-\lambda y} \cdot \frac{e^{-y} y^x}{x!}\,dy$$

$$= \frac{\lambda^r}{x!\,\Gamma(r)} \int_0^\infty y^{x+r-1} e^{-(\lambda+1)y}\,dy$$

make the integrand a known pdf over the $(0,\infty)$ using gamma with $\alpha = x+r$

$$\frac{\Gamma(x+r)\cdot\lambda^r}{x!\,\Gamma(r)\cdot(\lambda+1)^{x+r}} \int_0^\infty \frac{(\lambda+1)^{x+r} y^{x+r-1} e^{-(\lambda+1)y}}{\Gamma(x+r)}\,dy \rightarrow 1$$

b/c Pdf of a gamma$[(x+r),(\lambda+1)]$

$$P(X=x) = \frac{\Gamma(x+r)\cdot\lambda^r}{x!\,\Gamma r\,(\lambda+1)^{x+r}}$$

$$= \frac{\Gamma(x+r)}{\Gamma r\; x!} \cdot \frac{\lambda^r}{(\lambda+1)^r\cdot(\lambda+1)^x}$$

Figure 2: Problem 2, Question 1

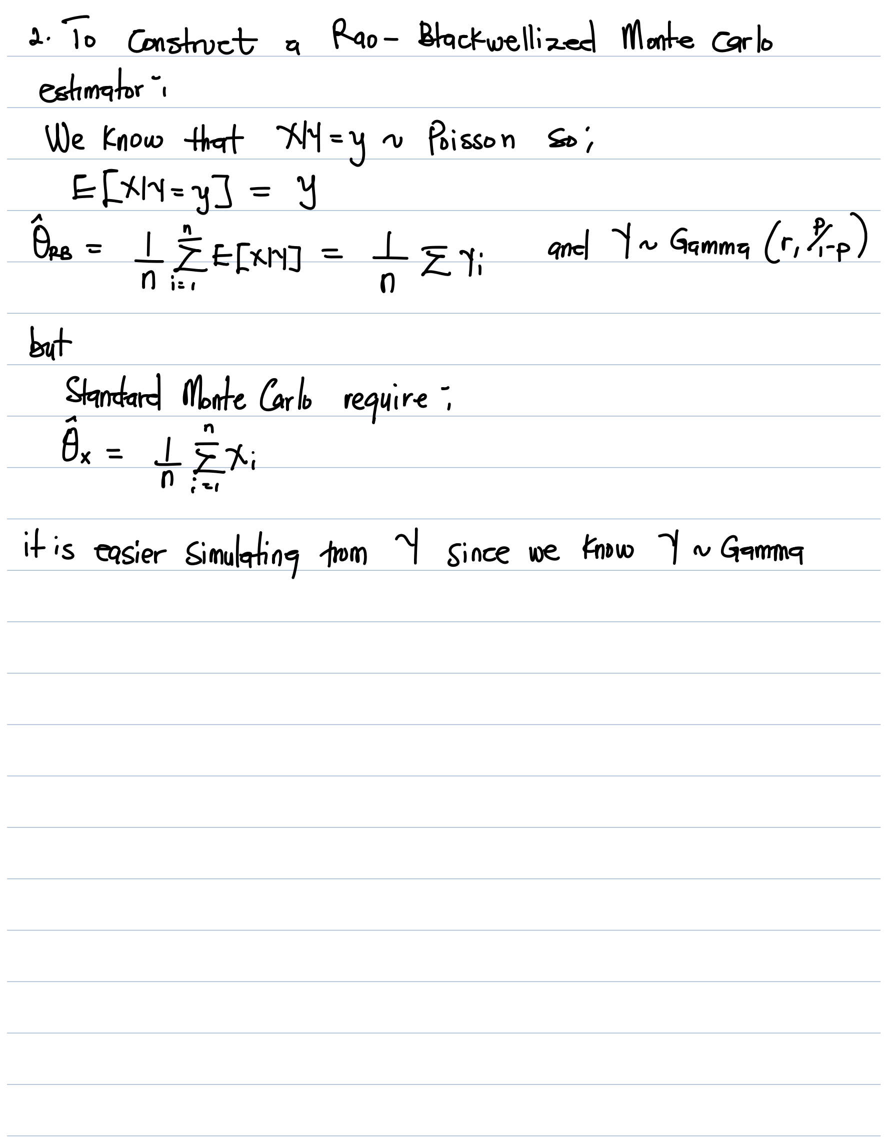
2

. Construct a Rao-Blackwellized Monte Carlo estimator of $\theta = E(X)$ using the auxiliary variable Y. Then check the empirical coverage of the 95% confidence interval of θ based on simulated values of X and Y as the size of the simulated sample increases. In the simulation, use $r = 2$ and $p = 2/3$.

Solution:

First derive the form of the Rao-Blackwellized Monte Carlo estimator:

```
knitr::include_graphics("D:/WashU/First Year/Sem1/SDS5531_StatsComputing/Homework/New folder/Prob22.pdf
```



2. To Construct a Rao- Blackwellized Monte Carlo estimator :

We know that $X|Y=y \sim$ Poisson so;

$$E[X|Y=y] = y$$

$$\hat{\theta}_{RB} = \frac{1}{n}\sum_{i=1}^{n} E[X|Y] = \frac{1}{n}\sum Y_i \quad \text{and } Y \sim \text{Gamma}\left(r, \tfrac{p}{1-p}\right)$$

but

Standard Monte Carlo require :

$$\hat{\theta}_x = \frac{1}{n}\sum_{i=1}^{n} x_i$$

it is easier Simulating from Y since we know $Y \sim$ Gamma

Figure 3: Problem 2, Question 2

Next, write an R program to implement the simulation:

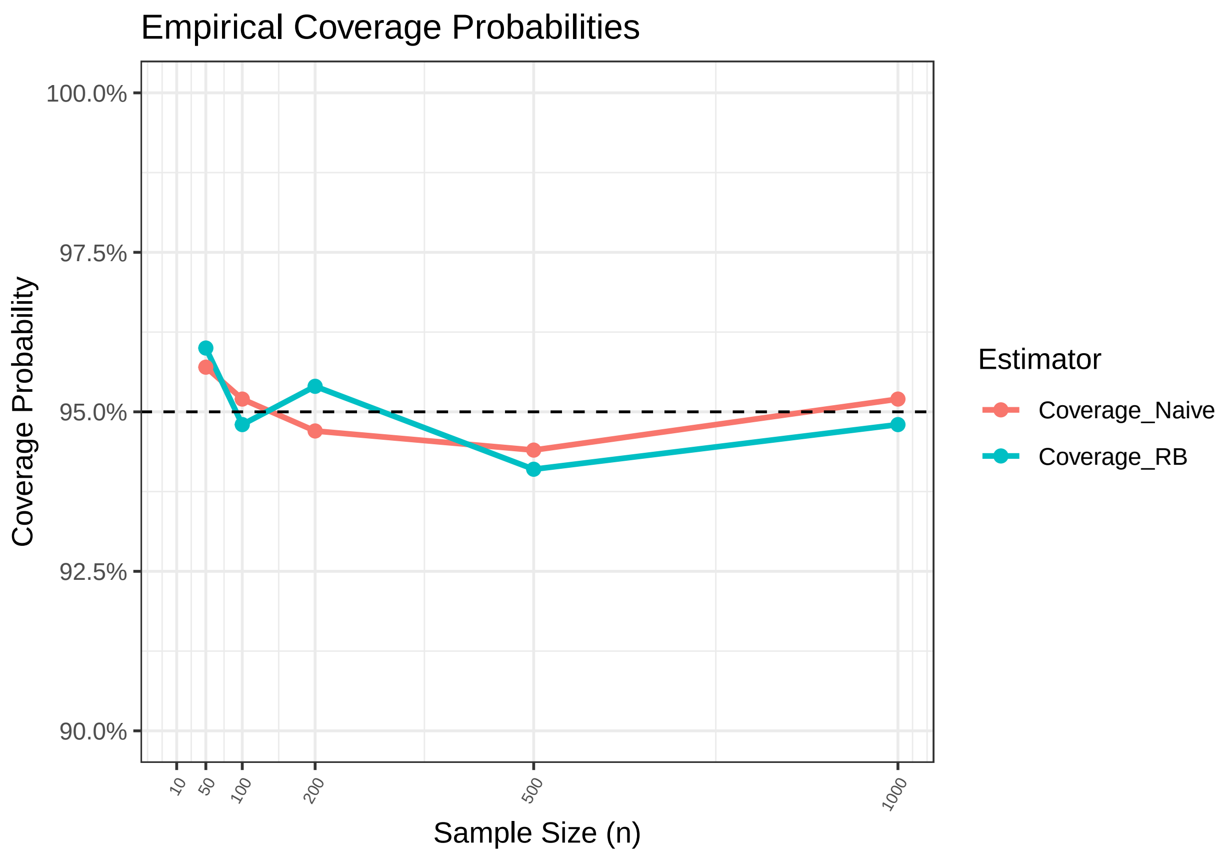
```r
# Next, write an R program to implement the simulation

# Parameters
r <- 2
p <- 2/3
lambda <- p/(1 - p)
theta <- r * (1 - p)/p
z_alpha <- qnorm(0.975)

# Sample sizes and number of simulations
n_values <- c(10, 50, 100, 200, 500, 1000)
N <- 1000  # Number of simulations for each sample size

# Function to perform simulations for a given sample size
simulate_coverage <- function(n) {
    # Simulate Y_i: N x n matrix
    Y_matrix <- matrix(rgamma(N * n, shape = r, rate = lambda), nrow = N, ncol = n)

    # Simulate X_i given Y_i: ~ Poisson(Y_i)
    X_vector <- rpois(N * n, lambda = as.vector(Y_matrix))
    X_matrix <- matrix(X_vector, nrow = N, ncol = n)

    # Compute estimators
    theta_hat_X <- rowMeans(X_matrix)
    theta_hat_RB <- rowMeans(Y_matrix)

    # Compute sample variances
    sigma2_X <- apply(X_matrix, 1, var)
    sigma2_RB <- apply(Y_matrix, 1, var)

    # Compute standard errors
    SE_X <- sqrt(sigma2_X/n)
    SE_RB <- sqrt(sigma2_RB/n)

    # Compute confidence intervals Standard Monte Carlo
    CI_lower_X <- theta_hat_X - z_alpha * SE_X
    CI_upper_X <- theta_hat_X + z_alpha * SE_X

    ## Rao Blackwellization
    CI_lower_RB <- theta_hat_RB - z_alpha * SE_RB
    CI_upper_RB <- theta_hat_RB + z_alpha * SE_RB

    # Condition for coverage
    cover_X <- (CI_lower_X <= theta) & (theta <= CI_upper_X)
    cover_RB <- (CI_lower_RB <= theta) & (theta <= CI_upper_RB)

    # coverage probabilities
    Coverage_Naive <- mean(cover_X)
    coverage_RB <- mean(cover_RB)

    return(c(Coverage_Naive = Coverage_Naive, Coverage_RB = coverage_RB))
}

# Perform simulations for each sample sizes
coverage_results <- sapply(n_values, simulate_coverage)

coverage_results <- t(coverage_results)

rownames(coverage_results) <- n_values

## Make into data
coverage_df <- as.data.frame(coverage_results)

coverage_df$SampleSize <- as.numeric(rownames(coverage_results))


# data frame to long format for plot
coverage_long <- pivot_longer(coverage_df, cols = c("Coverage_Naive", "Coverage_RB"),
    names_to = "Estimator", values_to = "Coverage")

# Display the results
kable(coverage_df, caption = "Empirical Coverage Probabilities:", latex_options = c("hold_position",
    "striped"))
```

Table 4: Empirical Coverage Probabilities:

	Coverage_Naive	Coverage_RB	SampleSize
10	0.889	0.883	10
50	0.957	0.960	50
100	0.952	0.948	100
200	0.947	0.954	200
500	0.944	0.941	500
1000	0.952	0.948	1000

Plot:

```r
ggplot(coverage_long, aes(x = SampleSize, y = Coverage, color = Estimator)) + geom_line(size = 1) +
    geom_point(size = 2) + geom_hline(yintercept = 0.95, linetype = "dashed", color = "black") +
    scale_x_continuous(breaks = coverage_df$SampleSize) + scale_y_continuous(limits = c(0.9,
    1), labels = scales::percent) + labs(title = "Empirical Coverage Probabilities",
    x = "Sample Size (n)", y = "Coverage Probability", color = "Estimator") + theme_bw() +
    theme(axis.text.x = element_text(angle = 60, hjust = 1, size = 6))
```



```
## Plotting the coverage probabilities plot(n_values, coverage_results[,
## 'Coverage_Naive'], type = 'o', col = 'blue', ylim = c(0.9, 1), xlab =
## 'Sample Size (n)', ylab = 'Coverage Probability', main = 'Empirical Coverage
## Probabilities') lines(n_values, coverage_results[, 'Coverage_RB'], type =
## 'o', col = 'red') abline(h = 0.95, lty = 2) legend('bottomright', legend =
## c('Standard Estimator', 'Rao-Blackwellized Estimator'), col = c('blue',
## 'red'), lty = 1, pch = 1)
```

Problem 3. Control variate

Consider Monte Carlo evaluation of the tail probability $E[1(X > a)] = P(X > a) = \int_a^\infty f(x)dx$, where $f(x)$ is the pdf of X. Sometimes, we know the value of $P(X > \mu)$ for some μ. For example, if f is symmetric around μ, then $P(X > \mu) = 1/2$. Then we may use the indicator $1(X > \mu)$ as a control variate and construct the following Monte Carlo estimator of $P(X > a)$,

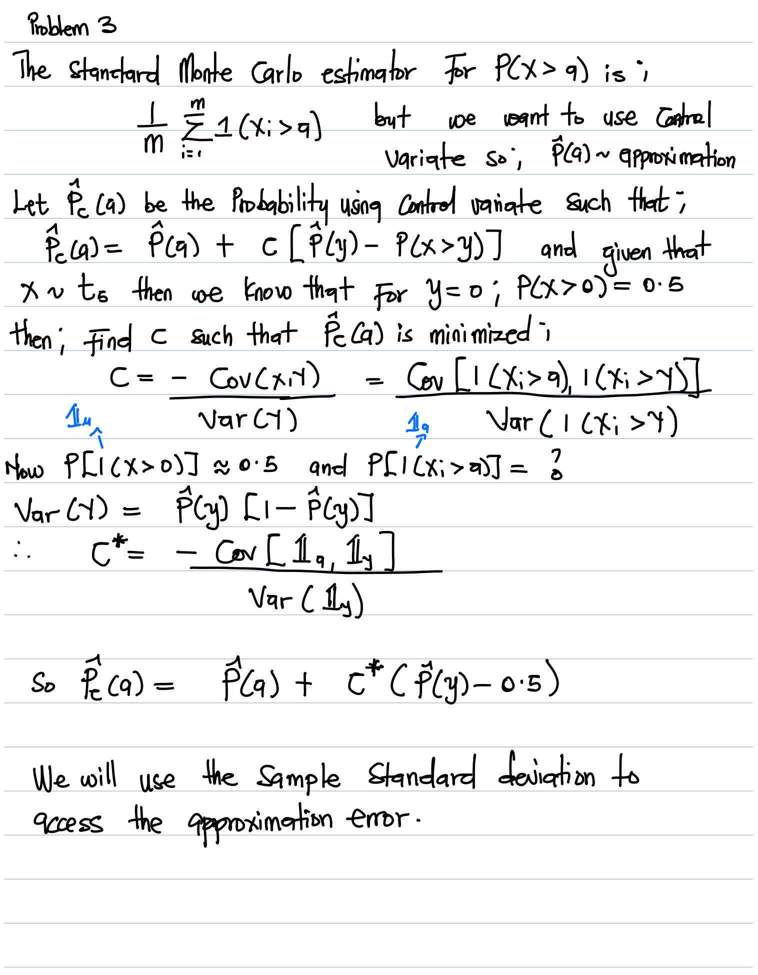
$$\frac{1}{m}\sum_{i=1}^{m} 1(X_i > a) + c\left(\frac{1}{m}\sum_{i=1}^{m} 1(X_i > \mu) - P(X > \mu)\right).$$

Use this approach to approximate the quantile a such that $P(X > a) = 0.01$, where X has the t_5 distribution.

Solution :

First we show the control variate approach as;

```r
knitr::include_graphics("D:/WashU/First Year/Sem1/SDS5531_StatsComputing/Homework/New folder/Prob3.pdf"
```



Problem 3

The standard Monte Carlo estimator for $P(X > a)$ is ;

$\frac{1}{m}\sum_{i=1}^{m} 1(X_i > a)$ but we want to use Control Variate so ; $\hat{P}(a) \sim$ approximation

Let $\hat{P}_c(a)$ be the Probability using Control variate such that ;

$\hat{P}_c(a) = \hat{P}(a) + c\,[\hat{P}(y) - P(x > y)]$ and given that

$x \sim t_5$ then we know that for $y = 0$; $P(X > 0) = 0.5$

then; find c such that $\hat{P}_c(a)$ is minimized ;

$$c = -\frac{Cov(x, Y)}{\underset{\overset{\uparrow}{\mathbb{1}_u}}{}Var(Y)} = \frac{Cov[1(X_i > a), 1(X_i > Y)]}{\underset{\overset{\nearrow}{\mathbb{1}_a}}{}Var(1(X_i > Y))}$$

Now $P[1(X > 0)] \approx 0.5$ and $P[1(X_i > a)] = \, ?$

$Var(Y) = \hat{P}(y)\,[1 - \hat{P}(y)]$

$$\therefore \quad c^* = -\frac{Cov[\mathbb{1}_a, \mathbb{1}_y]}{Var(\mathbb{1}_y)}$$

So $\hat{P}_c(a) = \hat{P}(a) + c^*(\hat{P}(y) - 0.5)$

We will use the Sample Standard deviation to access the approximation error.

Figure 4: Problem 3

Implementation:

```r
# implement the simulation and evaluate the Monte Carlo approximation error Set
# seed for reproducibility
set.seed(233)

# Parameters
m <- 1e+06  # Sample size
df <- 5  # Degrees of freedom for t-distribution
target_prob <- 0.01  # Target tail probability
```

```r
# Generate samples from t_5 distribution
X <- rt(m, df = df)

# Compute indicator for control variate (I_y)
I_y <- as.numeric(X > 0)
P_y_hat <- mean(I_y)  # Should be close to 0.5
Var_I_y <- P_y_hat * (1 - P_y_hat)  # Variance of I_mu

# Define function to compute adjusted estimator and find quantile 'a'
find_quantile <- function(target_prob, X, I_y, P_y_hat, Var_I_y) {
    # Define the function f(a)
    f <- function(a) {
        # Compute indicator I_a
        I_a <- as.numeric(X > a)

        # Estimate P_hat(a)
        P_a_hat <- mean(I_a)

        # Compute covariance between I_a and I_mu
        Cov_Ia_Iy <- mean((I_a - P_a_hat) * (I_y - P_y_hat))

        # Compute optimal c
        c_star <- -Cov_Ia_Iy/Var_I_y

        # Compute adjusted estimator
        P_c_hat <- P_a_hat + c_star * (P_y_hat - 0.5)

        # Return the difference from target probability
        return(P_c_hat - target_prob)
    }

    # Use uniroot to find the quantile 'a'
    result <- uniroot(f, lower = 2, upper = 10)

    return(result$root)


}

# Find the quantile 'a' such that P(X > a) = 0.01
a_estimate <- find_quantile(target_prob, X, I_y, P_y_hat, Var_I_y)

# Display the estimated quantile cat('Estimated quantile a such that P(X > a)
# =', target_prob, 'is', a_estimate, '\n')

# Compare with the true quantile using qt function
true_quantile <- qt(1 - target_prob, df = df)
# cat('True quantile from qt(1 -', target_prob, ', df =', df, ') is',
# true_quantile, '\n')

## Approximation error
SE <- abs((true_quantile - a_estimate)/sqrt(m))
```

True quantile from the t-distribution (df = 5) is 3.36493 and the estimated quantile using the control variate approach such that $P(X > a) = 0.01$ is 3.3710322. We can see that the approximation is not far off given that the standard error is 6.1×10^{-6}.

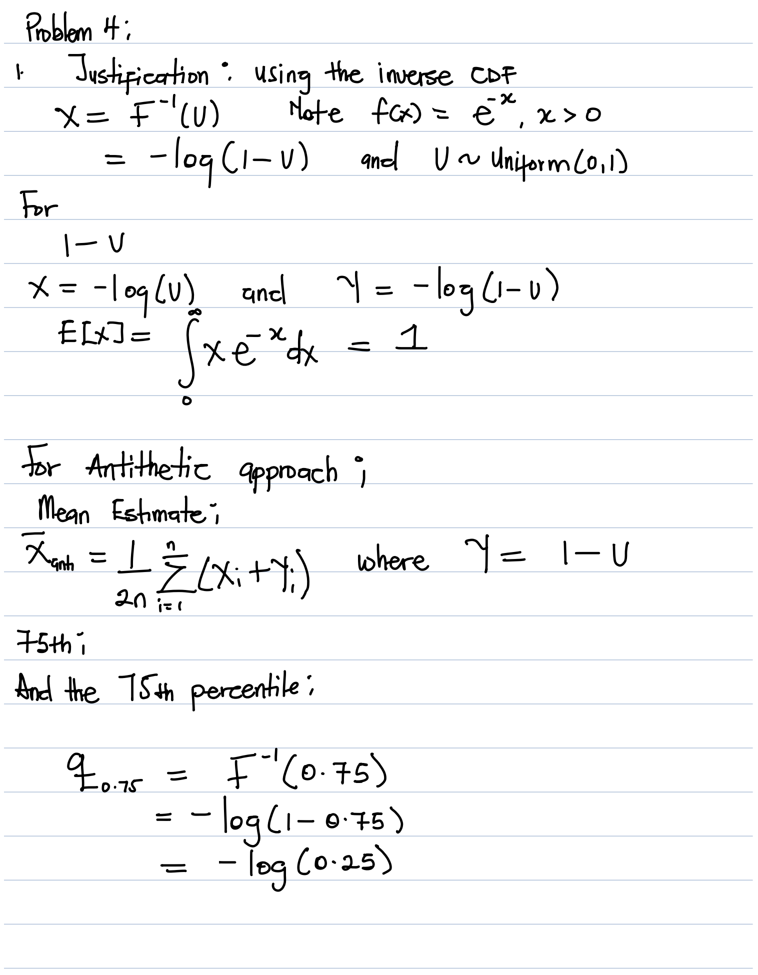
Problem 4. Antithetic variable

Apply the antithetic variable approach to obtain Monte Carlo estimates of the mean and 75th percentile of the exponential distribution $Exp(1)$ with pdf $f(x) = e^{-x}, x > 0$.

Solution:

First justify the validity of using the antichetic variable approach:

```
knitr::include_graphics("D:/WashU/First Year/Sem1/SDS5531_StatsComputing/Homework/New folder/Prob4.pdf"
```



Problem 4;

1. Justification: using the inverse CDF

$X = F^{-1}(U)$ Note $f(x) = e^{-x}, x > 0$

$= -\log(1-U)$ and $U \sim \text{Uniform}(0,1)$

For

$1-U$

$X = -\log(U)$ and $Y = -\log(1-U)$

$E[X] = \int_0^\infty x e^{-x} dx = 1$

For Antithetic approach;

Mean Estimate;

$\bar{X}_{anth} = \frac{1}{2n} \sum_{i=1}^{n} (X_i + Y_i)$ where $Y = 1-U$

75th;

And the 75th percentile;

$q_{0.75} = F^{-1}(0.75)$

$= -\log(1-0.75)$

$= -\log(0.25)$

Figure 5: Problem 4

Next implement the simulation:

```r
# Set seed for reproducibility
set.seed(52)

# Number of simulations
n <- 1e+06  # Adjust as needed for desired accuracy

# Generate uniform random variables
U <- runif(n)

# Generate antithetic variables
U_antithetic <- 1 - U

# Transform to Exponential(1) variables
X <- -log(U)
X_antithetic <- -log(U_antithetic)

# Combine the samples
X_combined <- c(X, X_antithetic)

# Estimate the mean using antithetic variables
mean_antithetic <- mean(c(X, X_antithetic))

# True mean of Exponential(1)
true_mean <- 1

# Estimate the 75th percentile
percentile_75_index <- ceiling(0.75 * length(X_combined))
X_sorted <- sort(X_combined)
percentile_75_estimate <- X_sorted[percentile_75_index]

# True 75th percentile
true_percentile_75 <- -log(1 - 0.75)  # Approximately 1.386294

# Display the results cat('Estimated mean using antithetic variables:',
# mean_antithetic, '\n') cat('True mean:', true_mean, '\n\n')

# cat('Estimated 75th percentile using antithetic variables:',
# percentile_75_estimate, '\n') cat('True 75th percentile:',
# true_percentile_75, '\n')

# Next implement the simulation
```

True Mean is 1 while the estimated mean using the antithetic variable is 0.9999805. The estimated 75% percentile under antithetic is 1.3855737 and the true 75% percentile is 1.3862944.