Assignment5: Inplementation of Different Algorithms for Calculating Stochastic Simulation Integrals

yifan

2025-03-19

Introduction

In class, we studied various algorithms for calculating stochastic simulation integrals using the Monte Carlo method, such as the random point method, the average value method, importance sampling, standardized importance sampling, and appropriately weighted sampling, among others. Some of these methods are more intuitive, some have smaller asymptotic variances, and some require fewer computational resources, each with its own advantages and disadvantages. In this assignment, I will implement Examples 3.2.4 and 3.2.5 from the textbook using the R language and compare the strengths and weaknesses of different algorithms.

Programming implementation of Example 3.2.4

In this example, we want to calculate:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy \tag{1}$$

Here,

$$f(x,y) = exp\{-45(x+0.4)^2 - 60(y-0.5)^2\} + 0.5exp\{-90(x-0.5)^2 - 45(y+0.1)^4\} \tag{2}$$

First, we try to calculate this integral using the average value method:

$$\hat{I_2} = \prod_{j=1}^{2}(b_j - a_j)\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{U_i}) \tag{3}$$

Where $a_j, b_j, j = 1, 2$ represent the lower and upper limits of the integral, respectively, N represents the number of simulations, and $\mathbf{U_i}, i = 1, 2, ..., N$ are i.i.d. two-dimensional uniformly distributed random variables on $[a_1, b_1] \times [a_2, b_2]$. We first define $f(x, y)$ as an R function:

```r
f <- function(x, y) {
  term1 <- exp(-45 * (x + 0.4)^2 - 60 * (y - 0.5)^2)
  term2 <- 0.5 * exp(-90 * (x - 0.5)^2 - 45 * (y + 0.1)^4)
  return(term1 + term2)
}
```

Then, we implement the average value method using R language:

```r
# Define the function for Monte Carlo integration using the average method
monte_carlo_integral <- function(a1, b1, a2, b2, N, f, seed = 123) {
  # Set the random seed for reproducibility
  set.seed(seed)
  # Generate N random points uniformly distributed in the 2D space (U_i1, U_i2)
  U1 <- runif(N, a1, b1)
  U2 <- runif(N, a2, b2)

  # Calculate the values of f(U_i1, U_i2)
  f_values <- sapply(1:N, function(i) f(U1[i], U2[i]))

  # Estimate the integral using the average method
  integral_estimate <- (b1 - a1) * (b2 - a2) * mean(f_values)

  return(integral_estimate)
}
```

In this case, $a_1 = a_2 = -1, b_1 = b_2 = 1$. We choose $N = 10,000$:

```r
a1 <- -1
b1 <- 1
a2 <- -1
b2 <- 1
N <- 10000

result <- monte_carlo_integral(a1, b1, a2, b2, N, f)
print(result)
```

```
## [1] 0.1305121
```

And the result is $\hat{I}_2 = 0.1305$. Now, we want to analyze the asymptotic variance of this method. Based on equation (3), it is easy to see that (We denote asymptotic variance as AV)

$$AV = \frac{(\prod_{j=1}^{2}(b_j - a_j))^2 Var(f(\mathbf{U}))}{N} \tag{4}$$

We can estimate $Var(f(\mathbf{U}))$ by

$$Var(f(\mathbf{U})) \approx \frac{\sum_{i=1}^{N}(f_i(\mathbf{U_i}) - \bar{f})^2}{N-1} \tag{5}$$

Here, $\bar{f} = \sum_{i=1}^{N} f(\mathbf{U_i})$.

```r
set.seed(123)
U1 <- runif(N, a1, b1)
U2 <- runif(N, a2, b2)
f_values <- sapply(1:N, function(i) f(U1[i], U2[i]))

# Compute the sample mean of f_values
f_mean <- mean(f_values)

# Compute the sample variance of f_values
f_var <- sum((f_values - f_mean)^2) / (N - 1)

# Compute the AV estimate using the given formula
prod_diff <- (b1 - a1) * (b2 - a2)  # Product of the differences (b_j - a_j)
(prod_diff^2 * f_var) / N
```

```
## [1] 1.949975e-05
```

Hence, we can estimate the asymptotic variance of the average value method to be approximately 1.95×10^{-5}.

Moreover, We can repeatedly invoke the function for the average method $B = 100$ times and calculate its mean and sample variance:

```
integral_estimates_1 <- numeric(100)
for (seed in 1:100) {
    integral_estimates_1[seed] <- monte_carlo_integral(a1, b1, a2, b2, N, f, seed)
  }

  # Calculate the mean and sample variance of the 100 estimates
mean(integral_estimates_1)
```

```
## [1] 0.1263618
```

```
var(integral_estimates_1)
```

```
## [1] 2.040548e-05
```

And we attain $\bar{I_2} = 0.1264, \bar{AV} = 2.04 \times 10^{-5}$.

We could also use importance sampling. Observing the form of $f(x, y)$, we can discern that $f(x, y)$ possesses two peaks centered at $(-0.4, 0.5)$ and $(0.5, -0.1)$ respectively, resembling the superposition of two normal distribution densities in its form. Therefore, we can choose the trial density as:

$$g(x, y) \propto \tilde{g}(x, y) = exp\{-45(x + 0.4)^2 - 60(y - 0.5)^2\} + 0.5exp\{-90(x - 0.5)^2 - 10(y + 0.1)^2\} \tag{6}$$

Since we have

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{g}(x, y)dxdy = \sqrt{\frac{2\pi}{90}}\sqrt{\frac{2\pi}{120}} + 0.5\sqrt{\frac{2\pi}{180}}\sqrt{\frac{2\pi}{20}} \approx 0.11282 \tag{7}$$

We can obtain

$$g(x, y) \approx \tilde{g}(x, y)/0.11282 \approx 0.5359f(x; -0.4, 90^{-1})f(y; 0.5, 120^{-1}) + 0.4641f(x; 0.5, 180^{-1})f(y; -0.1, 20^{-1}) \tag{8}$$

Here, we denote the probability density function of $N(\mu, \sigma^2)$ as $f(x; \mu, \sigma^2)$.

First, we define $g(x, y)$ as an R function:

```
# Define the function g(x, y)
g <- function(x, y) {
  # Define the two distributions f(x) and f(y) based on the normal distribution
  f_x1 <- dnorm(x, mean = -0.4, sd = sqrt(90^-1))
  f_y1 <- dnorm(y, mean = 0.5, sd = sqrt(120^-1))

  f_x2 <- dnorm(x, mean = 0.5, sd = sqrt(180^-1))
  f_y2 <- dnorm(y, mean = -0.1, sd = sqrt(20^-1))

  # Compute the value of g(x, y)
  result <- 0.5359 * f_x1 * f_y1 + 0.4641 * f_x2 * f_y2
  return(result)
}
```

We can employ the composition sampling method to perform random sampling from $g(x, y)$:

```
# Define the function to sample from the mixture distribution
sample_from_mixture <- function(N, seed = 123) {
  set.seed(seed)  # Set the seed

  # Define the two distributions and their corresponding weights
  weight1 <- 0.5359
```

```r
  weight2 <- 0.4641

  # Define the parameters of the two normal distributions
  params1_x <- c(-0.4, 90^-1)
  params1_y <- c(0.5, 120^-1)

  params2_x <- c(0.5, 180^-1)
  params2_y <- c(-0.1, 20^-1)

  # Generate N samples from the mixture distribution
  samples <- matrix(NA, nrow = N, ncol = 2)  # Matrix to store x and y samples

  for (i in 1:N) {
    # Choose which distribution to sample from based on the weights
    mix_choice <- sample(1:2, 1, prob = c(weight1, weight2))

    if (mix_choice == 1) {
      # Sample from the first distribution
      x <- rnorm(1, mean = params1_x[1], sd = sqrt(params1_x[2]))
      y <- rnorm(1, mean = params1_y[1], sd = sqrt(params1_y[2]))
    } else {
      # Sample from the second distribution
      x <- rnorm(1, mean = params2_x[1], sd = sqrt(params2_x[2]))
      y <- rnorm(1, mean = params2_y[1], sd = sqrt(params2_y[2]))
    }

    # Store the sampled values
    samples[i, ] <- c(x, y)
  }

  return(samples)
}
```

Then, we implement the importance sampling algorithm:

$$\hat{I_3} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{X_i})}{g(\mathbf{X_i})} \tag{9}$$

Using R language:

```r
# Importance Sampling Estimation Function
importance_sampling <- function(N, f, g, seed = 123) {
  # Generate samples
  samples <- sample_from_mixture(N, seed)

  # Calculate the estimate \hat{I_3}
  estimate <- 0
  for (i in 1:N) {
    x_i <- samples[i, 1]  # Get the first component x_i
    y_i <- samples[i, 2]  # Get the second component y_i

    # Calculate f(X_i) and g(X_i)
    f_value <- f(x_i, y_i)  # Value of the target distribution
    g_value <- g(x_i, y_i)  # Value of the proposal distribution
    # Accumulate f(X_i) / g(X_i)
    estimate <- estimate + f_value / g_value
  }

  # Calculate the final estimate of \hat{I_3}
  estimate <- estimate / N
  return(estimate)
}
```

And now we can obtain the result:

```
importance_sampling(10000,f,g)
```

```
## [1] 0.1253415
```

Which is $\hat{I_3} = 0.1253$. Now, we want to estimate the asymptotic variance as well. Based on equation (9), it is easy to see that

$$Var(\hat{I_3}) = Var(\frac{f(\mathbf{X})}{g(\mathbf{X})})\frac{1}{N} \tag{10}$$

Again, we can estimate $Var(\frac{f(\mathbf{X})}{g(\mathbf{X})})$ by

$$Var(\frac{f(\mathbf{X})}{g(\mathbf{X})}) \approx \frac{1}{N-1}\sum_{i=1}^{N}(\frac{f(\mathbf{X_i})}{g(\mathbf{X_i})} - \sum_{i=1}^{N}\frac{f(\mathbf{X_i})}{g(\mathbf{X_i})})^2 \tag{11}$$

Implement this using R language:

```
N <- 10000
samples <- sample_from_mixture(N, 123)

values <- numeric(N)

    for (i in 1:N) {
    x_i <- samples[i, 1]  # Get the first component x_i
    y_i <- samples[i, 2]  # Get the second component y_i

    # Calculate f(X_i) and g(X_i)
    f_value <- f(x_i, y_i)  # Value of the target distribution
    g_value <- g(x_i, y_i)  # Value of the proposal distribution

    values[i] = f_value / g_value
    }
# Compute the sample mean of values
v_mean <- mean(values)

# Compute the sample variance of values
v_var <- sum((values - v_mean)^2) / (N - 1)

v_var / N
```

```
## [1] 6.973395e-08
```

In this way, we can obtain the estimation of the asymptotic variance of importance sampling method to be 6.98×10^{-8}.

Again, we can repeatedly invoke the function for the importance sampling method $B = 100$ times and calculate its mean and sample variance:

```
integral_estimates_2 <- numeric(100)
for (seed in 1:100) {
    integral_estimates_2[seed] <- importance_sampling(N,f,g,seed)
  }

  # Calculate the mean and sample variance of the 100 estimates
mean(integral_estimates_2)
```

```
## [1] 0.1258733
```

```
var(integral_estimates_2)
```

```
## [1] 6.607244e-08
```

We can obtain the mean: 0.1259 and the sample variance 6.61×10^{-8}. The asymptotic variance for the average method is approximately 309 times greater than the asymptotic variance for the importance sampling method, making the importance sampling method a more accurate way for estimating the integral.

Programming implementation of Example 3.2.5

In this example, we assume that i.i.d samples Y_j follow the beta-binomial distribution:

$$f(y_j|K,\eta) = P(Y_j = y_j) = \binom{n}{y_j}\frac{B(k\eta + y_j, k(1-\eta)+n-y_j)}{B(k\eta, k(1-\eta))} \tag{12}$$

Here, $y_j = 0, 1, ..., n, B(\cdot,\cdot)$ is the Beta function, n is a given integer, and $K > 0, 0 < \eta < 1$ are parameters. We treat K and η as random variables as well, with prior density:

$$\pi(K,\eta) \propto \frac{1}{(1+k)^2}\frac{1}{\eta(1-\eta)} \tag{13}$$

then (K,η) has the posterior distribution:

$$\tilde{p}(K,\eta|\mathbf{Y}) \propto \pi(K,\eta)\prod_{j=1}^{n} f(y_j|K,\eta) \propto \frac{1}{(1+k)^2}\frac{1}{\eta(1-\eta)}\prod_{j=1}^{n}\frac{B(k\eta + y_j, K(1-\eta)+n-y_j)}{B(k\eta, K(1-\eta))} \tag{14}$$

and we want to calculate

$$E(lnK|\mathbf{Y}) = \int_0^1\int_0^\infty lnK\tilde{p}(k,\eta)d\eta dK \tag{15}$$

Since it is hard to sample from the posterior density (14) and we do not know the normalization constant that makes the integral of equation (14) equal to 1, we can use the normalized importance sampling method.

To remove the constraints on the values of (K,η), we perform the transformation:

$$\begin{cases}\alpha = \ln K, \\ \beta = \ln\dfrac{\eta}{1-\eta}\end{cases} \tag{16}$$

And we can obtain:

$$p(\alpha,\beta|\mathbf{Y}) \propto \frac{e^\alpha}{(1+e^\alpha)^2}\prod_{j=1}^{n}\frac{B(\frac{e^\alpha}{1+e^{-\beta}} + y_j, \frac{e^\alpha}{1+e^{-\beta}} + n - y_j)}{B(\frac{e^\alpha}{1+e^{-\beta}}, \frac{e^\alpha}{1+e^{\beta}})} \tag{17}$$

In this example, we generate the sample of (α, β) using independent $t(4)$ distributions: $(\alpha_i, \beta_i), i = 1, 2, \ldots, N$, and estimate $E(\ln K|\mathbf{Y})$ as:

$$\hat{\alpha} = \frac{\sum_{i=1}^{N} \alpha_i \frac{p(\alpha_i,\beta_i|\mathbf{Y})}{g(\alpha_i)g(\beta_i)}}{\sum_{i=1}^{N} \frac{p(\alpha_i,\beta_i|\mathbf{Y})}{g(\alpha_i)g(\beta_i)}} \tag{18}$$

Here, we assume the probability density function of $t(4)$ to be $g(\cdot)$. In equation (18), we can calculate $p(\alpha_i, \beta_i|\mathbf{Y})$using the right hand side of equation (17), because the normalization constants in the numerator and denominator can be canceled out.

Now, we implement the aforementioned algorithm using R language. However, during my implementation, I noticed that due to numerical overflow issues in computers, $p(\alpha, \beta|\mathbf{Y})$might return NaN or Inf. Therefore, we need to avoid this problem in the actual implementation. Before outputting the result, I added a check to determine whether result is NaN or Inf. If it is, the function returns 0, indicating that this particular estimation is invalid. Since the final number of samples N is very large, and the proportion of cases returning NaN or Inf is very small, this approach is feasible.

```r
p <- function(n, alpha, beta, Y) {
  # Calculate the constant term
  constant_term <- exp(alpha) / (1 + exp(alpha))^2

  # Calculate the Beta function part in the denominator
  beta_term <- beta(exp(alpha) / (1 + exp(-beta)), exp(alpha) / (1 + exp(beta)))

  # Calculate the product term
  product_term <- 1
  for (j in 1:n) {
    y_j <- Y[j]
    numerator <- beta(exp(alpha) / (1 + exp(-beta)) + y_j,
                      exp(alpha) / (1 + exp(-beta)) + n - y_j)
    denominator <- beta_term
    product_term <- product_term * (numerator / denominator)
  }

  # Final result
  result <- constant_term * product_term

  # Check if the result is less than 1e-16
  if (is.nan(result) || is.na(result) || is.infinite(result)) {
    return(0)  # Return 0 if result is NaN or NA
  } else {
    return(result)
  }
}
```

```r
alpha_hat <- function(n, N, alpha, beta, p, g, Y) {

  # Initialize the numerator and denominator
  numerator <- 0
  denominator <- 0

  # Loop through each i from 1 to N to calculate the sums
  for (i in 1:N) {
    # Calculate p(alpha_i, beta_i | Y) (Note: replace with the actual form of p)
    p_value <- p(n, alpha[i], beta[i], Y)

    # Calculate g(alpha_i) and g(beta_i) (Note: replace with the actual form of g)
    g_alpha <- g(alpha[i])
    g_beta <- g(beta[i])

    # Update the numerator and denominator
    numerator <- numerator + alpha[i] * p_value / (g_alpha * g_beta)
    denominator <- denominator + p_value / (g_alpha * g_beta)
  }

  # Compute the final result
  result <- numerator / denominator

  return(result)
}
```

Now, we can apply these functions. First, we have to set the value of n and $\mathbf{Y}$. In this case, I choose $n = 5$ and $\mathbf{Y} = [1, 2, 4, 5, 3]^T$.

```r
n <- 5
N <- 10000
Y <- c(1,2,4,5,3)

g <- function(x) {
  return(dt(x, df = 4))
}
# Set the random seed to ensure reproducibility
set.seed(123)

# Generate 10,000 independent random variables from the t(4) distribution
N <- 10000
alpha <- rt(N, df = 4)
beta <- rt(N, df = 4)
alpha_hat(n, N, alpha, beta, p, g, Y)
```

```
## [1] 6.508684
```

We obtain the estimation for $E(lnK|\mathbf{Y}) \approx 6.509$. Again, we can repeatedly invoke the function $B = 100$ times and calculate its mean and sample variance:

```r
integral_estimates_3 <- numeric(100)
for (seed in 1:100) {
  set.seed(seed)
  alpha <- rt(N, df = 4)
  beta <- rt(N, df = 4)
    integral_estimates_3[seed] <- alpha_hat(n, N, alpha, beta, p, g, Y)
  }

  # Calculate the mean and sample variance of the 100 estimates
mean(integral_estimates_3)
```

```
## [1] 6.352004
```

```r
var(integral_estimates_3)
```

```
## [1] 0.1055294
```

The mean is approximately 6.352, and the sample variance is actually quite large: approximately 0.1055. This is probably because the expectation $E(lnk|\mathbf{Y})$ is intrinsically difficult to calculate.