Financial Engineering - HA 2

Burzler R., Krejci T., Wittmann M., Zaborskikh E.

March 17, 2019

Exercise 1

Part A - Geometric Brownian Motion

Suppose we have process X following geometric Brownian motion dynamics with drift $\mu = 0.1$, $\sigma = 0.1$. By using the random walk approximation as well as the Cholesky decomposition to generate Brownian motion, simulate 1000 paths for the time interval $[0, 1]$. In your computations use $n = 250$ equidistant time discritization points with $t_0 = 0$ and $t_n = T = 1$. Compare the computation times corresponding to the two method and plot the simulated paths.

```
> ## Function Random Walk Approximation
> GBM <- function(S0, mu, sigma){
+   set.seed(1)
+   t <- seq(0, 1, 1/250)
+   St_vec.rw <- matrix(S0, 251, 1000)
+   for(i in 1:1000){
+     Z1 <- rnorm(250, 0, 1)
+     for(j in 1:250){
+       St_vec.rw[j+1, i] <- St_vec.rw[j,i] * exp((mu-1/2*sigma^2)*(t[j+1]-t[j]) +
+                                               sigma * sqrt(t[j+1]-t[j]) * Z1[j])
+       }}
+   return(St_vec.rw)
+ }
> ## Function Cholesky Approach
> GBM_COL <- function(S0, mu, sigma){
+   set.seed(1)
+   t <- seq(0, 1, 1/250)
+   A <- matrix(0, 250, 250)
+   index <- 0
+   for(j in 1:250){
+     index <- index + 1
+     for(i in index:250){
+       A[i,j] <- sqrt(t[i+1]-t[i])
+     }}
+   St_vec.col <- matrix(S0, 250, 1000)
+   for(i in 1:1000){
+     Z2 <- rnorm(250, 0, 1)
+     W <- A %*% Z2
+     St_vec.col[,i] <- S0 * exp((mu-1/2*sigma^2)*(t[j+1]-t[j]) + sigma * W)
+   }
+   return(St_vec.col)
+ }

> # S0 = 50, mu=0.1, sigma=0.1
```

```
> St_vec.rw <- GBM(50, 0.1, 0.1) # rw
> St_vec.col <- GBM_COL(50, 0.1, 0.1) # col
> # plotting
> par(mfrow=c(2,1))
> plot(St_vec.rw[,1], type="l", ylim=c(0, 150), main = "Random Walk Approximation")
> for(i in 2:1000){lines(St_vec.rw[,i], type="l")}
> plot(St_vec.col[,1], type="l", ylim=c(0, 150), main = "Cholesky Approach")
> for(i in 2:1000){lines(St_vec.col[,i], type="l")}
```

Repeat this exercise for $\mu = 0.4$, $\sigma = 0.4$. (keep the seed fixed in order to see the impact of different parameters). We can see that obviously increasing the standard deviation from 0.1 to 0.4 increases the volatility of the plotted Geometric Brownian Motion.

```
> # S0 = 50, mu=0.4, sigma=0.4
> St_vec.rw <- GBM(50, 0.4, 0.4) # rw
> St_vec.col <- GBM_COL(50, 0.4, 0.4) # col
> # plotting
> par(mfrow=c(2,1))
> plot(St_vec.rw[,1], type="l", ylim=c(0, 250), main = "Random Walk Approximation")
> for(i in 2:1000){lines(St_vec.rw[,i], type="l")}
> plot(St_vec.col[,1], type="l", ylim=c(0, 250), main = "Cholesky Approach")
> for(i in 2:1000){lines(St_vec.col[,i], type="l")}
```

Finally, we compare the computation times:

```
> # S0 = 50, mu=0.1, sigma=0.1
> system.time(GBM(50, 0.1, 0.1)) # rw

   user  system elapsed
   0.14    0.00    0.14

> system.time(GBM_COL(50, 0.1, 0.1)) # col

   user  system elapsed
   0.14    0.00    0.15

> # S0 = 50, mu=0.4, sigma=0.4
> system.time(GBM(50, 0.4, 0.4)) # rw

   user  system elapsed
   0.12    0.00    0.12

> system.time(GBM_COL(50, 0.4, 0.4)) # col

   user  system elapsed
   0.14    0.00    0.14
```

Part B - Poisson Process

Simulate 50 paths for a Poisson process with parameter $\lambda = 2$. Now keep the seed fixed and take $\lambda = 0.5$. For both cases provide a plot with the simulated paths.
We need to compute the successive arrivals τ_i for $i = 1, 2, \ldots$ as cumulative sums of independent exponential interarrivals.

```
> sim.Poiss <- function(lambda, n, T.stop, X =list()) {
+   for (i in 1:n) {
+     t <- 0
+     I <- 0
+     S <- c()
+     u <- runif(1, 0, 1)
+     t <- t - log(u)/lambda
+
+     while (t < T.stop) {
+       I <- I + 1
+       S[I] <- t
+       u <- runif(1, 0, 1)
+       t <- t-log(u)/lambda
+     }
+     X[[i]] <- c(0, S)
+     X[[i]] <- X[[i]][1:150]
+   }
+   Poiss <- do.call("cbind", X)
+ }
> # inputs
> T.stop <- 500
> n <- 50
> lambda.1 <- 2
> lambda.2 <- 0.5
> # simulations
> set.seed(2019)
> draws.1 <- sim.Poiss(lambda.1, n, T.stop)
> set.seed(2019)
> draws.2 <- sim.Poiss(lambda.2, n, T.stop)
> # plotting
> par(mfrow=c(2,1))
> matplot(x = draws.1, y = 0:149, type = "s", col = "darkcyan",
+         xlim = c(0, 50), xlab = "time", ylab = "N(t)",
+         main = "Poisson Process paths (lambda = 2)")
> matplot(x = draws.2, y = 0:149, type = "s", col = "darkcyan",
+         xlim = c(0, 50), xlab = "time", ylab = "N(t)",
+         main = "Poisson Process paths (lambda = 0.5)")
```

As the parameter τ represents the intensity, the higher one chooses τ the more frequent the arrivals (i.e. the more ”explosive” the process gets).

Exercise 2

Suppose we want to price a European call option written on a stock with initial value $S_0 = 80, \sigma = 0.2, \mu = 0.2$. The maturity of the option is in $T = 1$ year and the strike price is $K = 100$. The risk-free interest rate is $r = 2\%$.

First, we price the option analytically by using the closed-form price Black-Scholes formula. Second, we price it numerically by using naive Monte Carlo (with $n = 10000$ paths) and additionally compute the corresponding Monte Carlo standard error and confidence interval for $\alpha = 0.05$.

```
> set.seed(1)
> # given (\mu not needed)
> S0 <- 80
> K <- 100
> vol <- 0.2
> T_years <- 1
> r <- 0.02
> ## Analytically (by use of closed-form price BS-formula)
> d.1 <- (log(S0/K) + (r+vol^2/2)*T_years) / vol*sqrt(T_years)
> d.2 <- d.1 - vol*sqrt(T_years)
> Call.price <- S0*pnorm(d.1) - exp(-r*T_years)*K*pnorm(d.2)
> Call.price

[1] 1.427365

> ## naive MC Call Option Pricing
> Call.naive.mc <- function(S0, K, vol, T_years, r, n, alpha) {
+   Z <- rnorm(n)
+   ST.est <- S0*exp((r-vol^2/2)*T_years + vol*sqrt(T_years)*Z) # Simulate ST values (GBM)
+   payoff <- exp(-r*T_years)*pmax(ST.est-K, 0) # payoffs
+
+   Price <- mean(payoff)  # naive MC estimate
+   se <- sd(payoff)/sqrt(n)
+   z.score <- qnorm(1-alpha/2, mean = 0, sd = 1)
+   low.b <- Price - z.score*se
+   up.b <- Price + z.score*se
+   width <- up.b - low.b
+   return(c(MC.naive=Price, s.e.=se, Lower=low.b, Upper=up.b, ci.width = width))
+ }
> # given
> n <- 10000
> alpha <- 0.05
> Call.naive.mc(S0, K, vol, T_years, r, n, alpha)

  MC.naive       s.e.      Lower      Upper   ci.width
1.45657787 0.05092886 1.35675914 1.55639660 0.19963746
```

Exercise 3

Suppose we want to price a European call option written on a stock with initial value $S0 = 80$, $\sigma = 0.2$. The maturity of the option is in $T = 1$ year and the strike price is $K = 80$. Assume that the risk-free interest rate is $r = 2\%$. Price the option with antithetic variates (simulate n = 10000 paths). Estimate the reduction in the variance relative to naive Monte Carlo.

```
> # given
> S0 <- 80
> K <- 80
> vol <- 0.2
> T_years <- 1
> r <- 0.02
> #Antithetic Variates
> Call.anti.var <- function(S0, K, vol, T_years, r, n, alpha){
+   Z1 <- rnorm(n/2)
+   Z2 <- -Z1
+   ST.est1 <- S0*exp((r-vol^2/2)*T_years + vol*sqrt(T_years)*(Z1)) # Simulate ST values (GBM) 1
+   ST.est2 <- S0*exp((r-vol^2/2)*T_years + vol*sqrt(T_years)*(Z2)) # Simulate ST values (GBM) 2
+   payoff1 <- exp(-r*T_years)*pmax(ST.est1-K, 0) # payoff 1
+   payoff2 <- exp(-r*T_years)*pmax(ST.est2-K, 0) # payoff 2
+   payoff <- (payoff1 + payoff2)/2
+   Price <- mean(payoff)
+   se <- sd(payoff)/sqrt(n/2)
+   z.score <- qnorm(1-alpha/2, mean = 0, sd = 1)
+   low.b <- Price - z.score*se
+   up.b <- Price + z.score*se
+   width <- up.b - low.b
+   return(c(Anti.Var=Price, s.e.=se, Lower=low.b, Upper=up.b, ci.width = width))
+ }
> # given
> n <- 10000
> alpha <- 0.05
> #Analytical
> d.1 <- (log(S0/K) + (r+vol^2/2)*T_years) / vol*sqrt(T_years)
> d.2 <- d.1 - vol*sqrt(T_years)
> Call.price <- S0*pnorm(d.1) - exp(-r*T_years)*K*pnorm(d.2)
> Call.price

[1] 7.13283

> #Simulation
> set.seed(1)
> Call.naive.mc(S0, K, vol, T_years, r, n, alpha)

 MC.naive      s.e.     Lower     Upper  ci.width
7.1559634 0.1109028 6.9385979 7.3733289 0.4347310

> set.seed(1)
> Call.anti.var(S0, K, vol, T_years, r, n, alpha)

  Anti.Var       s.e.      Lower      Upper   ci.width
7.33876751 0.08702008 7.16821130 7.50932373 0.34111243
```

The reduction in variance is quite small as our european call option is at the money (K=S=80). This means that we apply the antithetic variates method to the non-increasing part of a call-payoff-function.

Now take the strike value K = 40 (option is deep-in-the-money) and redo your computations (keep the seed fixed). How is the reduction in variance affected?

```
> K <- 40
> n <- 10000
> alpha <- 0.05
> #Analytical
> d.1 <- (log(S0/K) + (r+vol^2/2)*T_years) / vol*sqrt(T_years)
> d.2 <- d.1 - vol*sqrt(T_years)
> Call.price <- S0*pnorm(d.1) - exp(-r*T_years)*K*pnorm(d.2)
> Call.price

[1] 40.79255

> #Simulation
> set.seed(1)
> Call.naive.mc(S0, K, vol, T_years, r, n, alpha)

  MC.naive       s.e.      Lower      Upper   ci.width
40.7245106  0.1630251 40.4049873 41.0440339  0.6390467

> set.seed(1)
> Call.anti.var(S0, K, vol, T_years, r, n, alpha)

   Anti.Var        s.e.       Lower       Upper    ci.width
40.87870140  0.03346012 40.81312076 40.94428203  0.13116127
```

As we are now in the money ($S > K$), the variance decreases is way higher now, since we apply the method to the increasing and monotone part of the call-payoff-function.

Exercise 4

Suppose we want to estimate $\theta = E((1-X^2)^{1/2})$, $X \sim U(0,1)$.

a) Computing naive Monte Carlo estimator of θ as well as the confidence interval for $\alpha = 0.05$ by taking $n = 10000$.

```
> set.seed(2019)
> # Naive MC-algorithm
> mc <- function(f, ndraws, alpha) {
+   draws <- runif(n = ndraws, 0, 1)
+   theta <- mean(f(draws))
+   se <- sd(f(draws))/sqrt(ndraws)
+   z.score <- qnorm(1-alpha/2, mean = 0, sd = 1)
+   low.b <- theta - z.score*se
+   up.b <- theta + z.score*se
+   width <- up.b - low.b
+   return(c(MC.naive=theta, s.e.=se, Lower=low.b, Upper=up.b, ci.width = width))
+ }
> f <- function(x) (sqrt(1-x^2)) # given function
> ndraws <- 10000 # number of draws
> alpha <- 0.05 # alpha
> MC.naive <- mc(f, ndraws, alpha); MC.naive
```

```
   MC.naive        s.e.       Lower       Upper    ci.width
0.789296626 0.002214547 0.784956194 0.793637059 0.008680866
```

b)

The ratio of the variance of the optimally controlled estimator to that of the uncontrolled estimator is given by

$$\frac{Var(X + c^*(Y - E(Y)))}{Var(X)} = \frac{Var(X) - \frac{Cov(X,Y)^2}{Var(Y)}}{Var(X)} = 1 - \rho(X,Y)^2$$

Thus, the effectiveness of a control variate, as measured by the variance reduction ratio is determined by the strength of the correlation between the quantity of interest X and the control Y. The sign of the correlation is irrelevant because it is absorbed in the optimal coefficient c^*. Therefore, the higher the correlation the better the control variant (ceteris paribus).

```
> set.seed(2019)
> u <- runif(n = ndraws, 0, 1)
> X <- sqrt(1-u^2)
> Y.1 <- u
> Y.2 <- u^2
> cor(X, Y.1)

[1] -0.9198958

> cor(X, Y.2)

[1] -0.9832095
```

As we can see $|\rho((1-X^2)^{1/2}, X^2)| > |\rho((1-X^2)^{1/2}, X)|$, thus it is better to use X^2 instead of X as a control variate.

c)

Now, using X^2 as a control variate we stimate θ and obtain the corresponding confidence intervals. For the pilot simulation we used $n = 1000$ to come up with optimal c (in the code called a). For the main part of simulation we used $n = 10000$ samples.

```
> set.seed(2019)
> mc.cv <- function(f, k, n, alpha) {
+   # pilot simulation
+   u1 <- runif(k)
+   X1 <- f(u1)
+   Y1 <- u1^2
+   a <- -cov(X1, Y1) / var(Y1) # estimate of c*
+
+   # main simulation
+   u <- runif(n)
+   X <- f(u)
+   Y <- u^2
+
+   Z <- X + a * (Y - mean(Y))
+   theta <- mean(Z) # MC-CV-estimate
+   se <- sd(Z)/sqrt(n)
+   z.score <- qnorm(1-alpha/2, mean = 0, sd = 1)
+   low.b <- theta - z.score*se
+   up.b <- theta + z.score*se
+   width <- up.b - low.b
+   return(c(MC.cv=theta, s.e.=se, Lower=low.b, Upper=up.b, ci.width = width))
+ }
> f <- function(x) (sqrt(1-x^2)) # given function
> k <- 1000 # draws pilot
> n <- 10000 # draws main
> alpha <- 0.05 # alpha
> MC.CV <- mc.cv(f, k, n, alpha); MC.CV

       MC.cv         s.e.        Lower        Upper     ci.width
0.7882717198 0.0004101219 0.7874678956 0.7890755441 0.0016076485
```

d)

Comparing these results shows that the method using the control variate performs better, as the standard error (s.e.), as well as the width of the confidence interval is smaller.

```
> MC.naive

   MC.naive        s.e.       Lower       Upper    ci.width
0.789296626 0.002214547 0.784956194 0.793637059 0.008680866

> MC.CV

       MC.cv         s.e.        Lower        Upper     ci.width
0.7882717198 0.0004101219 0.7874678956 0.7890755441 0.0016076485
```

Exercise 5

Recall we want to price a European call option written on a stock with initial value $S_0 = 80, \sigma = 0.2, \mu = 0.2$. The maturity of the option is in $T = 1$ year and the strike price is $K = 100$. The risk-free interest rate is $r = 2\%$.

We price the option by using the underlying stock as a control variate (with $n = 10000$) and compare the results to the analytical price. Furthermore, we compare it to the naive Monte Carlo, for which we also calculate the amount of the reduction in the variance.

```
> # given
> S0 <- 80
> K <- 100
> vol <- 0.2
> T_years <- 1
> r <- 0.02
> ## analytically (see Ex 2)
> 1.427365
```

```
[1] 1.427365
```

```
> ## naive MC (see Ex 2)
> n <- 10000
> alpha <- 0.05
> set.seed(1)
> Call.naive.mc(S0, K, vol, T_years, r, n, alpha)
```

```
  MC.naive       s.e.      Lower      Upper   ci.width
1.45657787 0.05092886 1.35675914 1.55639660 0.19963746
```

```
> ## MC with control variates
> Call.mc.cv <- function(S0, K, vol, T_years, r, k, n, alpha) {
+   # pilot simulation
+   Z1 <- rnorm(k)
+   ST.est <- S0*exp((r-vol^2/2)*T_years + vol*sqrt(T_years)*Z1)  # Simulate ST values (GBM)
+   payoff <- exp(-r*T_years)*pmax(ST.est-K, 0) # payoffs
+   a <- -cov(ST.est,payoff)/var(ST.est) # estimate of c*
+
+   # main simulation
+   Z2 <- rnorm(n)
+   ST.est <- S0*exp((r-vol^2/2)*T_years + vol*sqrt(T_years)*Z2)  # Simulate ST values (GBM)
+   payoff <- exp(-r*T_years)*pmax(ST.est-K, 0) # payoffs
+
+   payoff_cv <- payoff + a*(ST.est - S0*exp(r*T_years))
+   Price <- mean(payoff_cv) # MC-CV-estimate
+   se <- sd(payoff_cv)/sqrt(n)
+   z.score <- qnorm(1-alpha/2, mean = 0, sd = 1)
+   low.b <- Price - z.score*se
+   up.b <- Price + z.score*se
+   width <- up.b - low.b
+   return(c(MC.cv=Price, s.e.=se, Lower=low.b, Upper=up.b, ci.width = width))
+ }
> # given
> k <- 10000
> n <- 10000
> alpha <- 0.05
> set.seed(1)
> Call.mc.cv(S0, K, vol, T_years, r, k, n, alpha)
```

```
     MC.cv       s.e.      Lower      Upper   ci.width
1.46006096 0.03954383 1.38255647 1.53756545 0.15500898
```

```
> ## Amount of reduction in the variance
> # Naive
> set.seed(1)
> var.naive <- (Call.naive.mc(S0, K, vol, T_years, r, n, alpha)["s.e."])^2
> names(var.naive) <- "Var"
> # Control variate
> set.seed(1)
> var.cv <- (Call.mc.cv(S0, K, vol, T_years, r, k, n, alpha)["s.e."])^2
> names(var.cv) <- "Var"
> # reduction
> reduc.abs <- var.naive - var.cv
> reduc.rel <- (var.naive - var.cv) / var.naive*100
> reduc <- c(reduc.abs, reduc.rel)
> reduc  <- round(reduc, digits = 6)
> names(reduc) <- c("Absolute Var reduction", "Relative Var reduction %")
> reduc
```

```
  Absolute Var reduction Relative Var reduction %
                 0.00103                 39.71217
```

As we can see, pricing the option by using the underlying stock as a control variate comes closer to the analytical solution (not argued by the point estimate, but rather by judging the s.e. and confidence interval), compared to using naive MC. This fact is also indicated by the amount of the reduction in the variance, i.e. we were able to reduce the variance of the estimator by 39.71%.

Finally, we repeat this exercise for a strike price of $K = 20$ (keeping the seed fixed).

```
> ## K=20
> K <- 20
> # Analytically (by use of closed-form price BS-formula)
> d.1 <- (log(S0/K) + (r+vol^2/2)*T_years) / vol*sqrt(T_years)
> d.2 <- d.1 - vol*sqrt(T_years)
> Call.price2 <- S0*pnorm(d.1) - exp(-r*T_years)*K*pnorm(d.2)
> Call.price2
```

```
[1] 60.39603
```

```
> # naive MC
> set.seed(1)
> Call.naive.mc(S0, K, vol, T_years, r, n, alpha)
```

```
  MC.naive       s.e.      Lower      Upper   ci.width
60.3282687  0.1630306 60.0087346 60.6478027  0.6390681
```

```
> var.naive2 <- (Call.naive.mc(S0, K, vol, T_years, r, n, alpha)["s.e."])^2
> names(var.naive2) <- "Var"
> # Control variates
> set.seed(1)
> Call.mc.cv(S0, K, vol, T_years, r, k, n, alpha)
```

```
       MC.cv         s.e.        Lower        Upper     ci.width
6.039603e+01 4.440972e-17 6.039603e+01 6.039603e+01 0.000000e+00
```

```
> var.cv2 <- (Call.mc.cv(S0, K, vol, T_years, r, k, n, alpha)["s.e."])^2
> names(var.cv2) <- "Var"
> # Amount of reduction in the variance
> reduc.abs2 <- var.naive2 - var.cv2
> reduc.rel2 <- (var.naive2 - var.cv2) / var.naive2*100
> reduc2 <- c(reduc.abs2, reduc.rel2)
> reduc2  <- round(reduc2, digits = 6)
> names(reduc2) <- c("Absolute Var reduction", "Relative Var reduction %")
> reduc2
```

```
 Absolute Var reduction Relative Var reduction %
               0.025912               100.000000
```

We can observe that in our case pricing the option by using the underlying stock as a control variate gives nearly the exact same price as the analytical solution (not argued by the point estimate, but rather by judging the s.e. and confidence interval). The s.e. as well as the width of the confidence interval is extremely small. Compared to the naive MC we were able to reduce the variance of the estimator by $\sim 100\%$. This is due to the fact that the call option with $K = 20$ is always in the money, and therefore our control variate is perfectly correlated with the option payoff.

Exercise 6
a)

As a consequence of $X \sim exp(1)$, generating such samples that would yield non zero sets for $X > 20$ is very computationally burdensome. In case of the naive Monte Carlo estimation, we sample 10,000 times from the exponential distribution (*i.e.* $m = 10000$) with $\lambda = 1$ and obtain an estimate as follows:

$$\hat{\Theta}_m = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{n}\sum_{j=1}^{n} h(x_{i,j})\right) \tag{1}$$

Where $h(x_{i,j}) = \mathbf{I}_{x_{i,j}>20}$ and the number of draws in each iteration is $n = 50000$
In this case, the estimator variance is

$$\hat{\Theta}_{MC,n,m} = 4 \cdot 10^{-14}$$

.

```
> library(tictoc)
> tic()
> n <- 50000
> reps <- 10000
> prob20 <- rep(NA,reps)
> for(i in 1:reps) {
+   set.seed(i)
+   X <- rexp(n = n, 1)
+   prob20[i] <- sum(X > 20)/n
+ }
> Theta_n <- mean(prob20)
> Theta_n

[1] 2e-09

> v_naive <- var(prob20)
> v_naive

[1] 4e-14

> toc()

27.31 sec elapsed
```

b)

Now we use importance sampling to estimate θ where we sample from an exponential density with parameter λ. In the importance sampling application, we resample $X \sim g = exp(\lambda)$, i.e. g is taken to be from the same family of distributions as f. It remains to chose λ appropriately (note that if we could choose g such that it is similar to $h()f()$, then we might reasonably expect to obtain a large variance reduction).
Therefore, we choose a λ relatively small, e.g. the inverse $\lambda = \frac{1}{20}$ to approximate the product of $h(X)f(x)$.

c)

Now, we estimate θ using 10000 samples and the importance sampling density from b). Furthermore, we estimate the variance of our estimator and compare it to your answers in a). To evaluate the reduction in variance of our estimator, the above approach is applied to 10,000 samples of $\mathbf{X} \sim exp(1/20)$. The results for an identical size of generated samples (*i.e.* $n = 50000$) are summarized in the Table 1, showing that estimator variance may indeed be reduced by multiple orders in this case. Moreover, we include results for $n = 500$ to show the precision price of bringing down the computational time by a factor of roughly 100.

```
> n <- 50000
> n2 <- 500
> reps <- 10000
> lamb_g <- 1/20
> prob20_IS <- rep(NA,reps)
> prob20_IS2 <- rep(NA,reps)
> h <- function(x) {
+   if(x > 20) {
+     a <- 1
+   } else {
+       a <- 0
+       }
+   return(a)
+ }
> f <- function(x) {
+   return(exp(-x))
+ }
> g <- function(x) {
+   return(lamb_g*exp(-lamb_g*x))
+ }
> for(i in 1:reps) {
+   set.seed(i)
+   X_g <- rexp(n, lamb_g)
+   X_g2 <- rexp(n2, lamb_g)
+   prob20_IS[i] <- sum(sapply(X_g,h)*f(X_g)/g(X_g))/n
+   prob20_IS2[i] <- sum(sapply(X_g2,h)*f(X_g2)/g(X_g2))/n2
+ }
> Theta_IS_n <- mean(prob20_IS)
> Theta_IS_n2 <- mean(prob20_IS2)
> v_IS <- var(prob20_IS)
> v_IS2 <- var(prob20_IS2)
> rbind(c('Theta_IS_50K','VAR_IS_50K'), c(Theta_IS_n,v_IS))

     [,1]                  [,2]
[1,] "Theta_IS_50K"        "VAR_IS_50K"
[2,] "2.06055411150663e-09" "2.31128264247423e-21"
```

Table 1: Variance Comparison

MC Estimator	Estimate	Variance
Naive 50K	2e-9	4e-14
IS 50K	2e-9	2.31e-21
IS .5K	2e-9	2.34e-19

```
> rbind(c('Theta_IS_500','VAR_IS_500'), c(Theta_IS_n2,v_IS2))

     [,1]                  [,2]
[1,] "Theta_IS_500"        "VAR_IS_500"
[2,] "2.07147352955439e-09" "2.34435374884801e-19"
```