金融计量学期末PJ报告

23210980044　加兴华

注：本PJ基于R语言实现

Problem 1 (Example 3.14)

Implement a Monte Carlo method for single-asset European options, based on the Black-Scholes model. Perform experiments with various values of N and a random number generator of your choice. Compare results obtained by using the analytic solution formula for S_t with results obtained by using Euler's scheme. The payoff functions are

(a) vanilla put: $\Psi(S) = (K - S)^+, S_0 = 5, K = 10, r = 0.06, \sigma = 0.3, T = 1$.
(b) binary call: $\Psi(S) = I(S > K), S_0 = K = \sigma = T = 0.5, r = 0.1$.
(c) up-and-out barrier: call and $S_0 = 5, K = 6, r = 0.05, \sigma = 0.3, T = 1, B = 8$. ($B$ is the barrier such that the option expires worthless when $S_t \geq B$ for some t.)

(a)

$$\text{vanilla put: } \Psi(S) = (K - S)^+, S_0 = 5, K = 10, r = 0.06, \sigma = 0.3, T = 1.$$

欧式vanilla期权的Call和Put定价公式如下：

$$
\begin{aligned}
C &= S\Phi\left(d_1\right) - K\mathrm{e}^{-r(T-t)}\Phi\left(d_2\right) \\
P &= K\mathrm{e}^{-r(T-t)}\Phi\left(-d_2\right) - S\Phi\left(-d_1\right)
\end{aligned}
$$

其中

$$
\begin{aligned}
d_1 &= \frac{\ln(S/K)+\left(r+\frac{\sigma^2}{2}\right)(T-t)}{\sqrt{T-t}\sigma} \\
d_2 &= \frac{\ln(S/K)+\left(r-\frac{\sigma^2}{2}\right)(T-t)}{\sqrt{T-t}\sigma}
\end{aligned}
$$

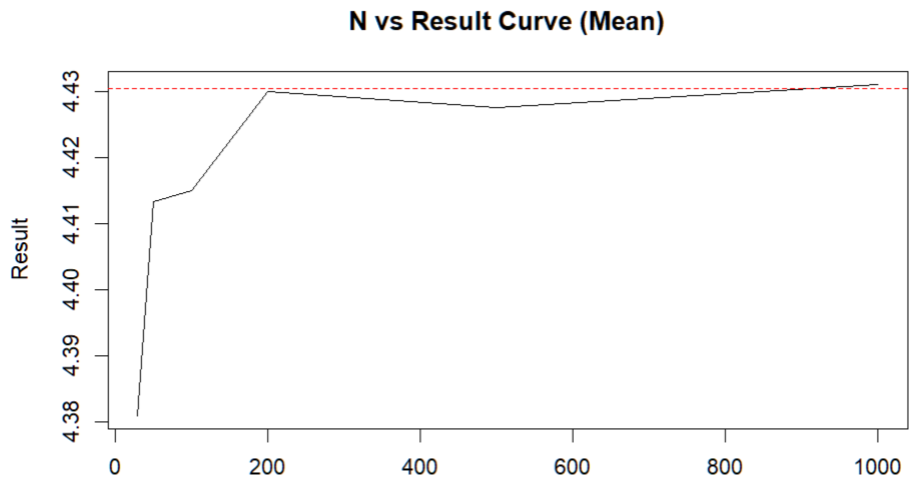
```
# 定义Black-Scholes模型的期权定价公式
analytic_v_put <- function(S, K, r, sigma, T){
  d1 <- (log(S/K) + (r + 0.5 * sigma^2) * T) / (sigma * sqrt(T))
  d2 <- d1 - sigma * sqrt(T)
  put_price <- K * exp(-r * T) * pnorm(-d2) - S * pnorm(-d1)
  return(put_price)
}

phi_X <- function(X,K) pmax(K - X, 0)

# 定义Euler方法
euler_discretization <- function(S0, r, sigma, T,num_steps , N){
  dt <- T / num_steps
  paths <- matrix(0, num_steps+1, N)
  paths[1,] <- S0
  for (i in 1:N){
    for (j in 1:num_steps){
      paths[j+1,i] <- paths[j,i] + r*paths[j,i]*dt +
sigma*paths[j,i]*sqrt(dt)*rnorm(1)
    }
  }
  return(paths)
}
```

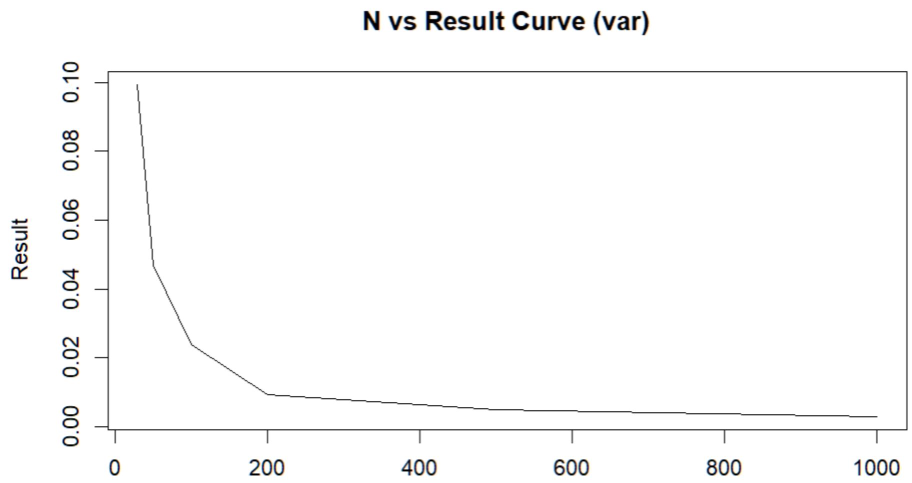
```r
# 执行蒙特卡罗模拟
num_steps <- 100 # 模拟时间步数
N <- 1000 # 模拟次数
S0 <- 5
K <- 10
r <- 0.06
sigma <- 0.3
T <- 1

#--------------------------------测试部分--------------------------------
# 使用解析解公式计算期权价格
analytic_put_price <- analytic_v_put(S0, K, r, sigma, T)

# 使用Euler方法计算期权价格
paths <- euler_discretization(S0, r, sigma, T, num_steps, N)
v_put_price <- mean(phi_X( paths[num_steps+1,],K)) * exp(-r * T)

# 输出结果
# 输出参数设置
cat("Parameter Settings:\n")
cat("S0:", S0, "\n","K:", K, "\n","r:", r, "\n","sigma:", sigma, "\n","T:", T,
"\n")
cat("Analytic Vanilla Put Price:", analytic_put_price, "\n")
cat("MC Vanilla Put Price:", v_put_price, "\n")

#--------------------------------实验部分--------------------------------
# 定义函数来执行实验并计算结果
run_experiment <- function(N, num_repeats) {
  results <- matrix(0, length(N), 2)  # 创建一个空的矩阵来存储结果
  for (i in 1:length(N)) {
    temp_results <- numeric(num_repeats)  # 创建一个临时向量来存储每个N值的重复实验结果
    for (j in 1:num_repeats) {
      paths <- euler_discretization(S0, r, sigma, T, num_steps, N[i])
      temp_results[j] <- mean(phi_X(paths[num_steps+1,], K)) * exp(-r * T)
    }
    results[i,] <- c(mean(temp_results),var(temp_results))  # 计算每个N值的平均结果和
方差
  }
  return(results)
}

N_values <- c(30, 50, 100, 200, 500, 1000)
# 运行实验
experiment_results <- run_experiment(N_values, 100)
# 绘制N值与结果的曲线
plot(N_values, experiment_results[,1], type = "l", xlab = "N", ylab = "Result",
main = "N vs Result Curve (Mean)")
# 添加水平红虚线
abline(h = analytic_put_price, col = "red", lty = 2)
# 绘制N值与结果的曲线
plot(N_values, experiment_results[,2], type = "l", xlab = "N", ylab = "Result",
main = "N vs Result Curve (var)")
```

测试结果如下：

```
Parameter Settings:
S0: 5
 K: 10
 r: 0.06
 sigma: 0.3
 T: 1
Analytic Vanilla Put Price: 4.430465
MC Vanilla Put Price: 4.507242
```

实验结果如下：





结论：

欧式vanilla期权Put，MC方法的数值解会快速向解析解逼近，同时估计值的稳定性也随模拟次数增加而提升

(b)

binary call: $\Psi(S) = I(S > K), S_0 = K = \sigma = T = 0.5, r = 0.1.$

二元期权的定价公式如下：

$$C = \exp(-r(T-t)) * \Phi\left(d_2\right)$$

其中

$$d_2 = \frac{\ln(S/K) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sqrt{T-t}\sigma}$$

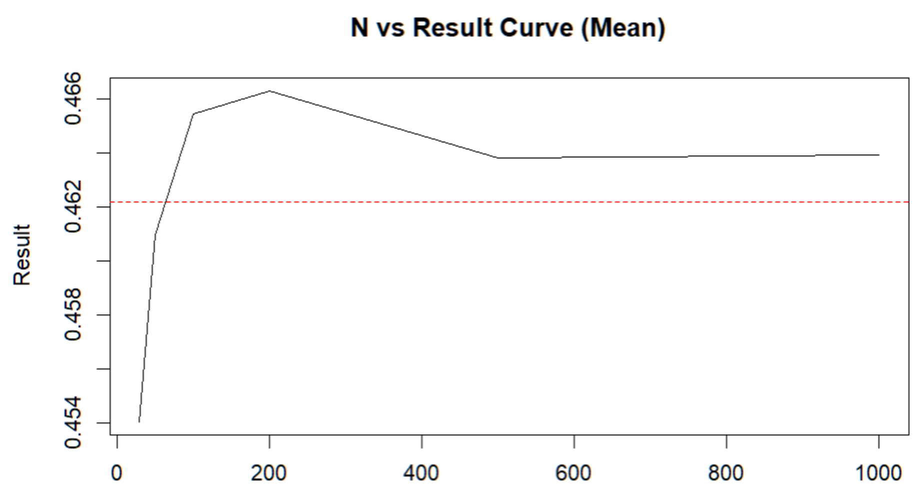
```
# 定义Black-Scholes模型的期权定价公式
analytic_b_call <- function(S, K, r, sigma, T){
  d2 <- (log(S/K) + (r -0.5 * sigma^2) * T) / (sigma * sqrt(T))
  res <- exp(-r *T) * pnorm(d2)
  return(res)
}

phi_X <- function(X,K) ifelse(X > K, 1, 0)

# 定义Euler方法
euler_discretization <- function(S0, r, sigma, T,num_steps , N){
  dt <- T / num_steps
  paths <- matrix(0, num_steps+1, N)
  paths[1,] <- S0
  for (i in 1:N){
    for (j in 1:num_steps){
      paths[j+1,i] <- paths[j,i] + r*paths[j,i]*dt +
sigma*paths[j,i]*sqrt(dt)*rnorm(1)
    }
  }
  return(paths)
}

num_steps <- 100 # 模拟时间步数
N <- 1000 # 模拟次数
S0 <- 0.5
K <- 0.5
r <- 0.1
sigma <- 0.5
T <- 0.5

#------------------------------测试部分------------------------------
# 使用解析解公式计算期权价格
analytic_call <- analytic_b_call(S0, K, r, sigma, T)

# 使用Euler方法计算期权价格
paths <- euler_discretization(S0, r, sigma, T, num_steps, N)
MC_b_call <- mean(phi_X(paths[num_steps+1,],K))* exp(-r * T)

# 输出结果
# 输出参数设置
cat("Parameter Settings:\n")
cat("S0:", S0, "\n","K:", K, "\n","r:", r, "\n","sigma:", sigma, "\n","T:", T,
"\n")
cat("Analytic Binary Call Price:", analytic_call, "\n")
cat("MC Binary call Price:", MC_b_call, "\n")

#------------------------------实验部分------------------------------
# 定义函数来执行实验并计算结果
run_experiment <- function(N, num_repeats) {
  results <- matrix(0, length(N), 2)  # 创建一个空的矩阵来存储结果
  for (i in 1:length(N)) {
```

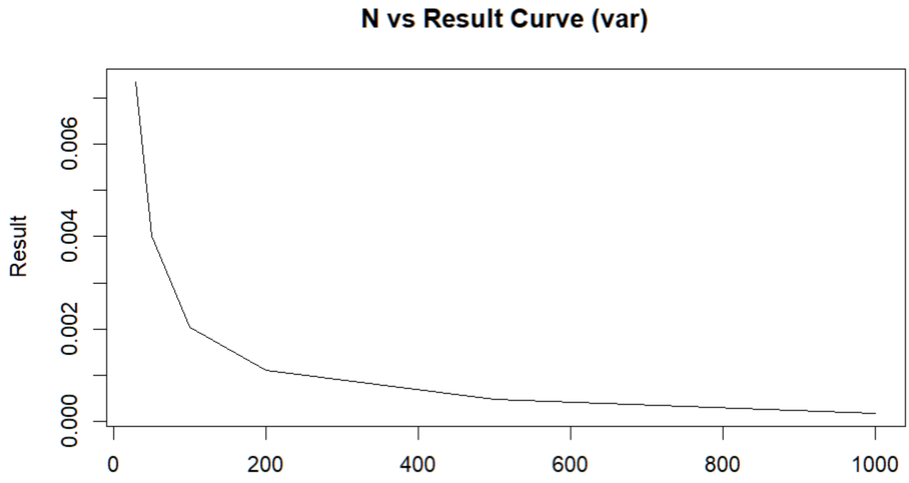
```
    temp_results <- numeric(num_repeats)  # 创建一个临时向量来存储每个N值的重复实验结果
    for (j in 1:num_repeats) {
      paths <- euler_discretization(S0, r, sigma, T, num_steps, N[i])
      temp_results[j] <-  mean(phi_X(paths[num_steps+1,],K))* exp(-r * T)
    }
    results[i,] <- c(mean(temp_results),var(temp_results))  # 计算每个N值的平均结果和方差
  }
  return(results)
}

N_values <- c(30, 50, 100, 200, 500, 1000)
# 运行实验
experiment_results <- run_experiment(N_values, 100)
# 绘制N值与结果的曲线
plot(N_values, experiment_results[,1], type = "l", xlab = "N", ylab = "Result",
main = "N vs Result Curve (Mean)")
# 添加水平红虚线
abline(h = analytic_call, col = "red", lty = 2)
# 绘制N值与结果的曲线
plot(N_values, experiment_results[,2], type = "l", xlab = "N", ylab = "Result",
main = "N vs Result Curve (var)")
```

测试结果如下:

```
Parameter Settings:
S0: 0.5
 K: 0.5
 r: 0.1
 sigma: 0.5
 T: 0.5
Analytic Binary Call Price: 0.4622007
MC Binary call Price: 0.4680049
```

实验结果如下:





结论：

二元期权Call，MC方法的数值解会快速向解析解逼近，同时估计值的稳定性也随模拟次数增加而提升

(c)

up-and-out barrier: call and $S_0 = 5, K = 6, r = 0.05, \sigma = 0.3, T = 1, B = 8$. ($B$ is the barrier such that the option expires worthless when $S_t \geq B$ for some t.)

根据Merton（1973）和Reiner和Rubinstein（1991a）提出了定价标准障碍期权的公式（Rich (1994).），up and out barrier call(Cuo)的解析解公式如下：

$$
\begin{aligned}
&Cuo = A - B + C - D \\
&A = S\Phi\left(\phi x_1\right) - Xe^{-rT}\Phi\left(x_1 - \sigma\sqrt{T}\right) \\
&B = S\Phi\left(\phi x_2\right) - Xe^{-rT}\Phi\left(x_2 - \sigma\sqrt{T}\right) \\
&C = S(H/S)^{2(\mu+1)}\Phi\left(y_1\right) - Xe^{-rT}(H/S)^{2\mu}\Phi\left(y_1 - \sigma\sqrt{T}\right) \\
&D = S(H/S)^{2(\mu+1)}\Phi\left(y_2\right) - Xe^{-rT}(H/S)^{2\mu}\Phi\left(y_2 - \sigma\sqrt{T}\right)
\end{aligned}
$$

其中

$$
\begin{array}{ll}
x_1 = \frac{\ln(S/X)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} & x_2 = \frac{\ln(S/H)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \\
y_1 = \frac{\ln\left(H^2/(SX)\right)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} & y_2 = \frac{\ln(H/S)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T}
\end{array}
$$

```
analytic_uob_call <- function(S,K,H, r, sigma, T) {
  lambda <- (r+sigma^2/2)/sigma^2
  y1 <- log(H^2/(S*K))/(sigma*sqrt(T))+lambda*sigma*sqrt(T)
  y2 <- log(H/S)/ (sigma * sqrt(T)) +lambda*sigma*sqrt(T)
  x1 <- log(S/K)/ (sigma * sqrt(T)) +lambda*sigma*sqrt(T)
  x2 <- log(S/H)/ (sigma * sqrt(T)) +lambda*sigma*sqrt(T)
  A <- S*pnorm(x1)-K*exp(-r*T)*pnorm(x1-sigma*sqrt(T))
  B <- S*pnorm(x2)-K*exp(-r*T)*pnorm(x2-sigma*sqrt(T))
  C <- S*(H/S)^(2*lambda)*pnorm(-y1)-K*exp(-r*T)*(H/S)^(2*lambda-2)*pnorm(-
y1+sigma*sqrt(T))
  D <- S*(H/S)^(2*lambda)*pnorm(-y2)-K*exp(-r*T)*(H/S)^(2*lambda-2)*pnorm(-
y2+sigma*sqrt(T))
  cuo <-  A-B+C-D


  if(K>H){
```

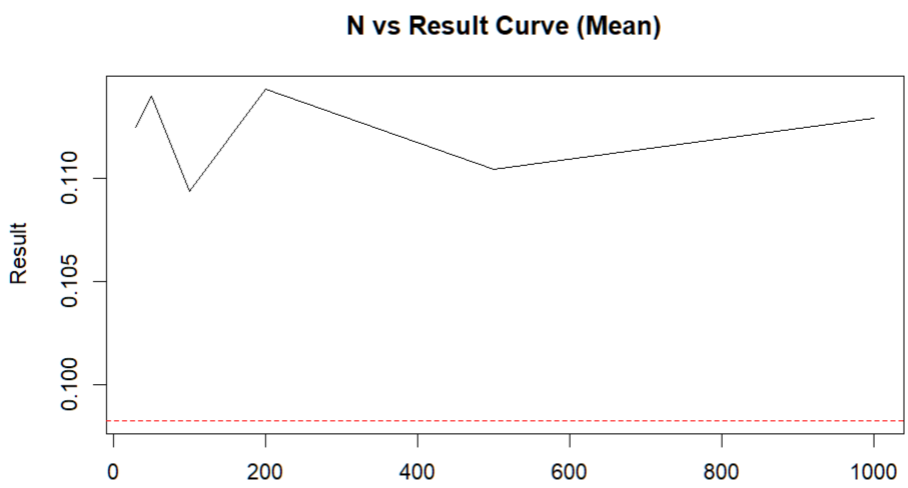
```
    return (0)
  } else if (K<=H) {
    return (cuo)
  }
}

phi_X <- function(X,K,B){
  barrier_hit <- apply(X, 2, function(x) max(x) >= B)  # 检查每次模拟是否达到barrier
  payoff <- pmax(X[nrow(X), ] - K, 0)  # 计算期权到期时的支付
  payoff[barrier_hit] <- 0  # 如果价格超过barrier，期权价值为0
  return (payoff)
}

# 定义Euler方法
euler_discretization <- function(S0, r, sigma, T,num_steps , N){
  dt <- T / num_steps
  paths <- matrix(0, num_steps+1, N)
  paths[1,] <- S0
  for (i in 1:N){
    for (j in 1:num_steps){
      paths[j+1,i] <- paths[j,i] + r*paths[j,i]*dt +
sigma*paths[j,i]*sqrt(dt)*rnorm(1)
    }
  }
  return(paths)
}

num_steps <- 100 # 模拟时间步数
N <- 1000 # 模拟次数
S0 <- 5
K <- 6
r <- 0.05
sigma <- 0.3
T <- 1
B <- 8

#------------------------------测试部分------------------------------
# 使用解析解公式计算期权价格
analytic_call <- analytic_uob_call(S0, K, B, r, sigma, T)

# 使用Euler方法计算期权价格
paths <- euler_discretization(S0, r, sigma, T, num_steps, N)
MC_uob_call <- mean(phi_X(paths,K,B)) * exp(-r * T)

# 输出结果
# 输出参数设置
cat("Parameter Settings:\n")
cat("S0:", S0, "\n","K:", K, "\n","r:", r, "\n","sigma:", sigma, "\n","T:", T,
"\n")
cat("Analytic Up-and-Out Barrier Put Price:", analytic_call, "\n")
cat("MC Up-and-Out Barrier Price:", MC_uob_call, "\n")

#------------------------------实验部分------------------------------
# 定义函数来执行实验并计算结果
run_experiment <- function(N, num_repeats) {
  results <- matrix(0, length(N), 2)  # 创建一个空的矩阵来存储结果
```

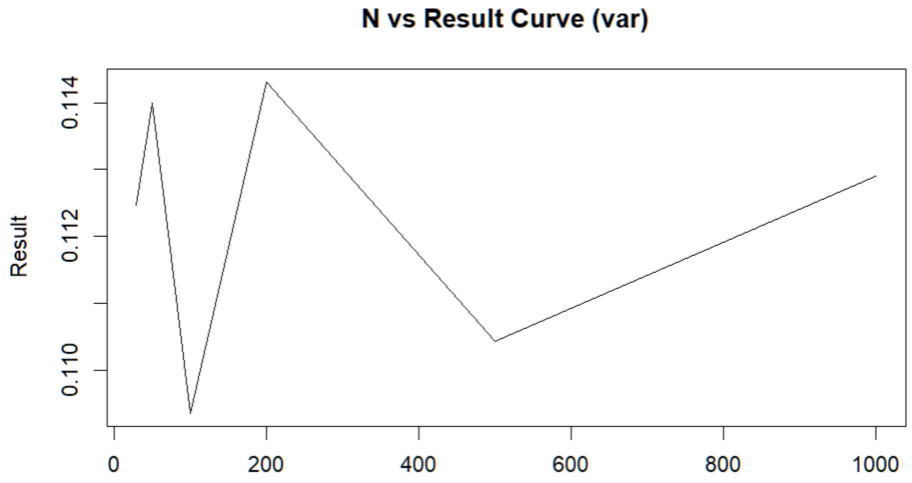
```r
  for (i in 1:length(N)) {
    temp_results <- numeric(num_repeats)  # 创建一个临时向量来存储每个N值的重复实验结果
    for (j in 1:num_repeats) {
      paths <- euler_discretization(S0, r, sigma, T, num_steps, N[i])
      temp_results[j] <- mean(phi_X(paths,K,B)) * exp(-r * T)
    }
    results[i,] <- c(mean(temp_results))  # 计算每个N值的平均结果和方差
  }
  return(results)
}

N_values <- c(30, 50, 100, 200, 500, 1000)
# 运行实验
experiment_results <- run_experiment(N_values, 100)
# 绘制N值与结果的曲线
plot(N_values, experiment_results[,1], type = "l", xlab = "", ylab = "Result",
                                            ylim =
range(c(experiment_results[,1], analytic_call)),
                                            main = "N vs Result Curve (Mean)")
# 添加水平红虚线
abline(h = analytic_call, col = "red", lty = 2)
# 绘制N值与结果的曲线
plot(N_values, experiment_results[,2], type = "l", xlab = "", ylab = "Result",
main = "N vs Result Curve (var)")
```

测试结果如下:

```
Parameter Settings:
S0: 5
 K: 6
 r: 0.05
 sigma: 0.3
 T: 1
Analytic Up-and-Out Barrier Put Price: 0.09827914
MC Up-and-Out Barrier Price: 0.1233865
```

实验结果如下:





结论：

up-and-out障碍期权Call，MC方法的数值解在模拟次数较小的情况的无法逼近到解析解，同时估计值的稳定性也没有明显提升

Problem 2 (Example 3.16)

An Asian option has the following payoff at maturity T:

$$\max\left(\frac{1}{m}\sum_{i=1}^{m} S_{t_i} - K, 0\right),$$

where K is the strike price and S_{t_i} is the level of the underlying asset at i-th monitoring time t. We assume m total monitoring times $0 < t_1 < t_2, < \ldots < t_m = T$. Set $S_0 = 100, K = 90, \sigma = 0.3, r = 0.05, T = 2$ and $m = 20$ with equidistant times. Price the Asian option with antithetic variates and control variates methods.

对立变量（AV）：由于每条价格轨迹靠若干次 $Z_i \sim N(0,1)$ 变量生成,我以基于（$-Z_i$)生成的轨迹作为其对立变量，在计算数字特征时先对每组对立变量取平均，以防止样本量翻倍导致的样本方差变大被错误纳入比较分析；

控制变量（CV）：出于节约成本考量，我直接选取生成的价格轨迹的最终价格S_T 作为控制变量，根据Euler迭代公式以及每次采样的独立性，易见$ES_T = S_0 * (1 + r * T/m)^m$，最优系数 $c = -\frac{cov(S_T,h(S_T))}{var(S_T)}$，并通过MC方法对c进行估计。

```
phi_X <- function(X,K) pmax( colMeans(X)- K, 0)

# 定义Euler方法
euler_discretization <- function(S0, r, sigma, T,num_steps , N ){
  dt <- T / num_steps
  # 申明变量
  paths <- matrix(0, num_steps+1, N)
  paths[1,] <- S0
  antithetic_paths <- matrix(0, num_steps+1, N)
  antithetic_paths[1,] <- S0
  # 采样标准正态
  Z <- matrix(rnorm(num_steps*N), nrow=num_steps)
  for (i in 1:N){
    for (j in 1:num_steps){
      paths[j+1,i] <- paths[j,i]* (1 + r*dt + sigma*sqrt(dt)*Z[j,i] )
      antithetic_paths[j+1,i] <- antithetic_paths[j,i]* (1 + r*dt -
sigma*sqrt(dt)*Z[j,i] )
    }
  }
  return(list(paths, antithetic_paths))
```

```r
}

# 建模参数
S0 <- 100
K <- 90
r <- 0.05
sigma <- 0.3
T <- 2
m <- 20
N <- 50

# 使用Euler方法计算期权价格
result <- euler_discretization(S0, r, sigma, T, num_steps=m, N)
paths <- result[[1]]
anti_paths <- result[[2]]
hST_samples <- phi_X(paths[-1,],K)
# 对立变量方法
hST_AV_samples <- (hST_samples+phi_X(anti_paths[-1,],K))/2
# 控制变量方法, 直接用轨迹最后时刻作为随机变量Xi
c <- - cov(paths[(m+1),],hST_samples)/var(paths[(m+1),])
Eh0 <- S0*(1+r*T/m)^m
hST_CV_samples <- hST_samples + c*(paths[(m+1),]- Eh0)

cat("h(ST) samples     mean: ",mean(hST_samples),"var: " ,var(hST_samples) ,"\n")
cat("h(ST)_AV samples  mean: ",mean(hST_AV_samples),"var: " ,var(hST_AV_samples)
,"\n")
cat("h(ST)_CV samples  mean: ",mean(hST_CV_samples),"var: " ,var(hST_CV_samples)
,"\n")

cat("h(S0) est: ",mean(hST_samples)* exp(-r * T),"\n")
cat("h(S0)_AV est: ",mean(hST_AV_samples)* exp(-r * T),"\n")
cat("h(S0)_CV est: ",mean(hST_CV_samples)* exp(-r * T),"\n")
```

测试结果如下:

```
h(ST) samples     mean:  17.10514 var:  381.6694
h(ST)_AV samples  mean:  18.54946 var:  81.12317
h(ST)_CV samples  mean:  19.88204 var:  107.8389
h(S0) est:  15.47737
h(S0)_AV est:  16.78424
h(S0)_CV est:  17.99001
```

结论：在模拟次数较少（N=50）的情况下，原始MC方法生成的样本 $h(S_T)$ 与在增加对立变量和控制变量方法后获得的样本 $h(S_T)^{AV}$ 和 $h(S_T)^{CV}$，它们在均值估计结果接近的同时后两者呈现出更小的波动性，符合理论中两种方法能够缩小样本方差的结论。