Assignment 3 - Napat L

Problem 1: Comparing different Monte Carlo schemes

(a) and (b)

From the comparison table below, the most accurate method is MC with Delta-based Control Variate with the least stand error equal to 0.015849. On the other hand, the fastest method with 36.42 spending time is MC Euler.

```
##                           MC         MC.av         MC.d       MC.av.d
## call.price        20.28401561 20.378851477 2.036121e+01 2.035484e+01
## call.se            0.03938683  0.023921973 5.077676e-02 1.584900e-02
## call.time.elapsed 36.42000000 52.420000000 1.452490e+03 3.025220e+03
## put.price         17.48380652 17.477857803 1.748291e+01 1.748510e+01
## put.se             0.02025526  0.007486986 1.591632e-02 9.281464e-03
## put.time.elapsed  36.60000000 52.080000000 1.464170e+03 3.041890e+03
```

Code:

```
MonteCarloEuler <- function(isCall, S0, K, T, sig,div,r,N,M){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    lns <- log(S0)

    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){
        w <- rnorm(N)
        lnSt <- lns

        for(i in 1:N){
            lnSt <- lnSt + nudt + sigsdt*w[i]
        }
        St <- exp(lnSt)
        OT <- ifelse(isCall, max(0,St-K), max(0,K-St))
        sum_OT <- sum_OT+OT
        sum_OT2 <- sum_OT2+OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)
    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}

MC.call <- MonteCarloEuler(TRUE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 1000000)
MC.put <- MonteCarloEuler(FALSE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 1000000)


#1.b
MonteCarloAntithetic <- function(isCall, S0, K, T, sig,div,r,N,M){
    ## Precompute constants
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
```

```
    lnS <- log(S0)

    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){

        lnSt1 <- lnS
        lnSt2 <- lnS

        w <- rnorm(N)

        lnSt <- lnS

        for(i in 1:N){
            lnSt1 <- lnSt1 + nudt + sigsdt*w[i]
            lnSt2 <- lnSt2 + nudt + sigsdt*(-1*w[i])
        }
        St1 <- exp(lnSt1)
        St2 <- exp(lnSt2)

        OT <- ifelse(isCall, 0.5*(max(0,St1-K)+max(0,St2-K)), 0.5*(max(0,K-St1)+max(0,K-St2)))
        sum_OT <- sum_OT+OT
        sum_OT2 <- sum_OT2+OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)
    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}

MC.av.call <- MonteCarloAntithetic(TRUE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 1000000)
MC.av.put <- MonteCarloAntithetic(FALSE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 1000000)
####
black_scholes_delta <- function(isCall, S0, t, K, T, r, sig, div){
    d1 <- (log(S0/K)+(r-div+sig^2/2)*(T-t))/(sig*sqrt(T-t))
    d2 <- d1-sig*sqrt(T-t)
    result <- ifelse(isCall, exp(-div*(T-t))*pnorm(d1), exp(-div*(T-t))*(pnorm(d1)-1))
    return(result)
}

MonteCarloEulerDelta <- function(isCall, S0, K, T, sig,div,r,N,M, beta){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    erddt <- exp((r-div)*dt)

    beta1 <- beta
    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){
        w <- rnorm(N)
        St <- S0
        cv <- 0
        for(i in 1:N){

            t <- (i-1)*dt
            delta <- black_scholes_delta(isCall, St, t, K, T, sig, r, div)
            Stn <- St*exp(nudt + sigsdt*w[i])
            cv <- cv + delta*(Stn-St*erddt)*exp(T-(t+dt))
            St = Stn
        }

        OT <- ifelse(isCall, max(0,St-K) + beta1*cv, max(0,K-St) + beta1*cv)
```

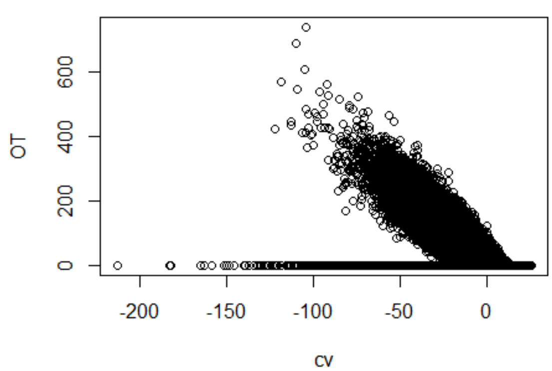
```
        sum_OT <- sum_OT + OT
        sum_OT2 <- sum_OT2 + OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)

    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}

MC.d.call <- MonteCarloEulerDelta(TRUE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 1000000, -1)
MC.d.put <- MonteCarloEulerDelta(FALSE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 1000000, -1)

MonteCarloAntitheticDelta <- function(isCall, S0, K, T, sig, div, r, N, M, beta){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    erddt <- exp((r-div)*dt)
    beta1 <- beta
    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){
        St1 <- S0
        St2 <- S0
        cv1 <- 0
        cv2 <- 0
        w <- rnorm(N)
        for(i in 1:N){
            t <- (i-1)*dt
            delta1 <- black_scholes_delta(isCall, St1, t, K, T, sig, r, div)
            delta2 <- black_scholes_delta(isCall, St2, t, K, T, sig, r, div)
            Stn1 <- St1*exp(nudt + sigsdt*w[i])
            Stn2 <- St2*exp(nudt + sigsdt*-1*w[i])
            cv1 <- cv1 + delta1*(Stn1-St1*erddt)*exp(T-(t+dt))
            cv2 <- cv2 + delta2*(Stn2-St2*erddt)*exp(T-(t+dt))
            St1 = Stn1
            St2 = Stn2
        }
        OT <- ifelse(isCall, 0.5*(max(0,St1-K) + beta1*cv1 + max(0,St2-K)+ beta1*cv2)
                     , 0.5*(max(0,K-St1) + beta1*cv1 + max(0,K-St2) + beta1*cv2))
        sum_OT <- sum_OT+OT
        sum_OT2 <- sum_OT2+OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)

    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}

MC.av.d.call <- MonteCarloAntitheticDelta(TRUE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 100000
0, -1)
MC.av.d.put <- MonteCarloAntitheticDelta(FALSE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 300, 100000
0, -1)

df.compare <- data.frame(MC = c(call.price = MC.call$opt_value, call.se = MC.call$SE, call.time = MC.call$
time, put.price = MC.put$opt_value, put.se = MC.put$SE, put.time = MC.put$time),
                         MC.av = c(call.price = MC.av.call$opt_value, call.se = MC.av.call$SE, call.time =
MC.av.call$time, put.price = MC.av.put$opt_value, put.se = MC.av.put$SE, put.time = MC.av.put$time),
                         MC.d = c(call.price = MC.d.call$opt_value, call.se = MC.d.call$SE, call.time = MC
.d.call$time, put.price = MC.d.put$opt_value, put.se = MC.d.put$SE, put.time = MC.d.put$time),
                         MC.av.d = c(call.price = MC.av.d.call$opt_value, call.se = MC.av.d.call$SE, call.
time = MC.av.d.call$time, put.price = MC.av.d.put$opt_value, put.se = MC.av.d.put$SE, put.time = MC.av.d.p
```

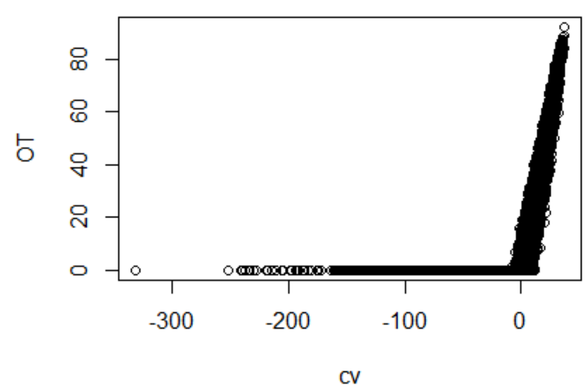
```
ut$time))
df.compare
```

(C)

We run the regression between payoff and control variate to get the beta. We have that the beta for our barrier option equal to 1.052 for call option and -0.6807 for put option.

```
##
## Call:
## lm(formula = OT ~ cv)
##
## Coefficients:
## (Intercept)           cv
##      11.241       -1.052

##
## Call:
## lm(formula = OT ~ cv)
##
## Coefficients:
## (Intercept)           cv
##      18.5676       0.6807
```





For barrier option, the most accurate method is still MC with Delta-based Control Variate with the least stand error equal to 0.1871046. On the other hand, the fastest method with 80 spending time is also MC Euler.

```
##                         MC.id      MC.a.di       MC.d.di      MC.ad.di
## call.price         10.0654951 10.07859424    11.5602859    11.3662762
## call.se             0.2589899  0.17118391     0.3088872     0.1871046
## call.time.elapsed  80.0000000  143.000000 232405.00000 396276.00000
## put.price          17.7994309 17.53071688    16.4371210    16.7810565
## put.se              0.2049184  0.09157694     0.4673072     0.1208783
## put.time.elapsed   82.00000   144.000000 450534.00000 878356.00000
```

Code:

```
MonteCarloEulerDownIn <- function(isCall, S0, K, T, sig,div,r,N,M, H){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){
        w <- rnorm(N)
        St <- S0
        BARRIER_CROSSED <- FALSE
        for(i in 1:N){
            St <- St*exp(nudt + sigsdt*w[i])
            if(St <= H){
                BARRIER_CROSSED <- TRUE
            }
        }
        if(BARRIER_CROSSED){
            OT <- ifelse(isCall, max(0,St-K), max(0,K-St))
        }else{
            OT <- 0
        }
        sum_OT <- sum_OT + OT
        sum_OT2 <- sum_OT2 + OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)

    end.time <- proc.time()

    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}

MC.di.call <- MonteCarloEulerDownIn(TRUE, S0=100, K=100, T=1, sig = 0.5, div=0.03, r=0.06, 300, 1000000, 9
0)

MC.di.put <- MonteCarloEulerDownIn(FALSE, S0=100, K=100, T=1, sig = 0.5, div=0.03, r=0.06, 300, 1000000, 9
0)

MonteCarloAntitheticDownIn <- function(isCall, S0, K, T, sig,div,r,N,M,H){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){
        w <- rnorm(N)
        St1 <- S0
        St2 <- S0
        BARRIER_CROSSED1 <- FALSE
        BARRIER_CROSSED2 <- FALSE
        for(i in 1:N){
            St1 <- St1*exp(nudt + sigsdt*w[i])
            St2 <- St2*exp(nudt + sigsdt*(-1)*w[i])
            if(St1 <= H){
                BARRIER_CROSSED1 <- TRUE
            }
            if(St2 <= H){
                BARRIER_CROSSED2 <- TRUE
            }
        }
```

```
        if(BARRIER_CROSSED1 & BARRIER_CROSSED2){
            OT <- ifelse(isCall, 0.5*(max(0,St1-K)+max(0,St2-K)), 0.5*(max(0,K-St1)+max(0,K-St2)))
        }else{
            OT <- 0
        }
        sum_OT <- sum_OT + OT
        sum_OT2 <- sum_OT2 + OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)

    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}

MC.a.di.call <- MonteCarloAntitheticDownIn (TRUE, S0=100, K=100, T=1, sig = 0.5, div=0.03, r=0.06, 300, 10
00000, 90)

MC.a.di.put <- MonteCarloAntitheticDownIn (FALSE, S0=100, K=100, T=1, sig = 0.5, div=0.03, r=0.06, 300, 10
00000, 90)


library(derivmkts)

di.delta <- function(S0,K,T,t,r,sig,H,d, isCall = TRUE, h = 10e-10){

    delta <- (calldownin(s=S0+h, k=K, v=sig, r=r, tt=T-t, d=d, H=H) -
              calldownin(s=S0, k=K, v=sig, r=r, tt=T-t, d=d, H=H))/(h)
    return(delta)
}

findBeta <- function(S0, K, r, div, sig,T, H, N, M){

    S <- matrix(0, N+1, M+1)
    S[1,] <- S0
    OT <- 0
    dt <- T/N
    for(i in 1:(M+1)){
        for(j in 1:(N)){
            S[j+1,i] <- S[j,i]+(r-div)*S[j,i]*dt+sig*S[j,i]*sqrt(dt)*rnorm(1)
        }
        OT[i] <- ifelse((min(S[,i]) <= H), 1, 0)*max(S[N+1,i]-K,0)
    }
    cv <- 0
    for(i in 1:(N)){
        t <- (i-1)*dt
        cv <- cv+di.delta(S0,K,T=T, t=t,r=r,sig=sig,H=H,d=div, isCall = TRUE)*(S[i+1,]-(S[i,]*exp((r-div)*
dt)))
    }
    plot(cv, OT)
    model <- lm(OT ~ cv)
    return(model)
}
findBeta(S0=100, K=100, r=0.06, div=0.03, sig=0.5,T=1, H=90, N=1000, M=1000000)

MonteCarloEulerDeltaDownIn <- function(isCall, S0, K, T, sig,div,r,N,M, beta, H){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    erddt <- exp((r-div)*dt)
    beta1 <- beta
    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()
    for (j in 1:M){
```

```
        w <- rnorm(N)
        St <- S0
        cv <- 0
        BARRIER_CROSSED <- FALSE
        for(i in 1:N){
            t <- (i-1)*dt
            delta <- di.delta(S0=St,K=K,T=T, t=t,r=r,sig=sig,H=H,d=div, isCall = isCall)
            Stn <- St*exp(nudt + sigsdt*w[i])
            cv <- cv + delta*(Stn-St*erddt)*exp(T-(t+dt))
            St = Stn
            if(St <= H){
                BARRIER_CROSSED <- TRUE
            }
        }
        if(BARRIER_CROSSED){
            OT <- ifelse(isCall, max(0,St-K) + beta1*cv, max(0,K-St) + beta1*cv)
        }else{
            OT <- 0
        }

        sum_OT <- sum_OT + OT
        sum_OT2 <- sum_OT2 + OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)
    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}
MC.d.di.call <- MonteCarloEulerDeltaDownIn(TRUE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 100, 10000
, 1.052, 90)
MC.d.di.put <- MonteCarloEulerDeltaDownIn(FALSE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 100, 10000
, -0.6807, 90)


MonteCarloAntitheticDeltaDownIn <- function(isCall, S0, K, T, sig, div, r, N, M, beta, H){
    dt <- T/N
    nudt <- (r-div-0.5*sig^2)*dt
    sigsdt <- sig*sqrt(dt)
    erddt <- exp((r-div)*dt)
    beta1 <- beta
    sum_OT <- 0
    sum_OT2 <- 0
    start.time <- proc.time()

    for (j in 1:M){
        St1 <- S0
        St2 <- S0
        cv1 <- 0
        cv2 <- 0
        BARRIER_CROSSED1 <- FALSE
        BARRIER_CROSSED2 <- FALSE
        w <- rnorm(N)
        for(i in 1:N){
            t <- (i-1)*dt
            delta1 <- di.delta(S0=St1,K=K,T=T, t=t,r=r,sig=sig,H=H,d=div, isCall = isCall)
            delta2 <- di.delta(S0=St2,K=K,T=T, t=t,r=r,sig=sig,H=H,d=div, isCall = isCall)
            Stn1 <- St1*exp(nudt + sigsdt*w[i])
            Stn2 <- St2*exp(nudt + sigsdt*-1*w[i])
            cv1 <- cv1 + delta1*(Stn1-St1*erddt)*exp(T-(t+dt))
            cv2 <- cv2 + delta2*(Stn2-St2*erddt)*exp(T-(t+dt))
            St1 = Stn1
            St2 = Stn2
            if(St1 <= H){
                BARRIER_CROSSED1 <- TRUE
            }
            if(St2 <= H){
```

```
                BARRIER_CROSSED2 <- TRUE
            }
        }
        if(BARRIER_CROSSED1 & BARRIER_CROSSED2){
            OT <- ifelse(isCall, 0.5*(max(0,St1-K) + beta1*cv1 + max(0,St2-K)+ beta1*cv2), 0.5*(max(0,K-St
1) + beta1*cv1 + max(0,K-St2) + beta1*cv2))
        }else{
            OT <- 0
        }
        sum_OT <- sum_OT+OT
        sum_OT2 <- sum_OT2+OT*OT
    }
    opt_value <- sum_OT/M*exp(-r*T)
    SD <- sqrt((sum_OT2 - sum_OT*sum_OT/M)*exp(-2*r*T)/(M-1))
    SE <- SD/sqrt(M)
    end.time <- proc.time()
    timetaken <- end.time - start.time
    list(opt_value = opt_value, SE = SE, time = timetaken[3])
}
MC.ad.di.call <- MonteCarloAntitheticDeltaDownIn(TRUE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 100,
10000, 1.052, 90)

MC.ad.di.put <- MonteCarloAntitheticDeltaDownIn(FALSE, S0=100, K=100, T=1, sig=0.5, div=0.03, r=0.06, 100,
10000, -0.6807, 90)

df.di.compare <- data.frame(MC.id = c(call.price = MC.di.call$opt_value, call.se =
                                       MC.di.call$SE, call.time = MC.di.call$time,
                                    put.price = MC.di.put$opt_value, put.se =
                                       MC.di.put$SE, put.time = MC.di.put$time),
                            MC.a.di = c(call.price = MC.a.di.call$opt_value, call.se
                                      = MC.a.di.call$SE, call.time = MC.a.di.call$time,
                                      put.price = MC.a.di.put$opt_value, put.se =
                                          MC.a.di.put$SE, put.time = MC.a.di.put$time),
                            MC.d.di = c(call.price = MC.d.di.call$opt_value, call.se =
                                            MC.d.di.call$SE, call.time = MC.d.di.call$time,
                                     put.price = MC.d.di.put$opt_value, put.se =
                                         MC.d.di.put$SE, put.time = MC.d.di.put$time),
                            MC.ad.di = c(call.price = MC.ad.di.call$opt_value,
                                        call.se = MC.ad.di.call$SE, call.time = MC.ad.di.call$time,
                                        put.price = MC.ad.di.put$opt_value,
                                        put.se = MC.ad.di.put$SE, put.time = MC.ad.di.put$time))
df.di.compare
```

Problem 2: Simulating the Heston model

For Euler scheme, the best scheme for Heston volatility is full truncation with the least RMSE, bias and time spending. However, for Euler-Milstein, since the Euler-Milstein can make less zero for volatility, it makes stable to the model and the best scheme is reflection with the least bias.

```
heston.Euler

##              method    price time.taken.elapsed   rmse   bias
## 1    full.truncation 6.808328            1082.62 0.0074 0.0022
## 2 partial.truncation 6.811183            2204.86 0.0074 0.0051
## 3         higham.mao 6.833203            3319.87 0.0075 0.0271
## 4         reflection 6.958183            4498.59 0.0078 0.1521
## 5         absorption 6.868755            5728.12 0.0076 0.0627

heston.Milstein

##              method    price time.taken.elapsed   rmse    bias
## 1    full.truncation 6.780408            1120.92 0.0074 -0.0257
## 2 partial.truncation 6.778836            2265.14 0.0074 -0.0273
## 3         higham.mao 6.779307            3355.53 0.0074 -0.0268
## 4         reflection 6.794432            4523.43 0.0074 -0.0117
## 5         absorption 6.780879            5719.31 0.0074 -0.0252
```

Code:

```
MCHestonEuler <- function(NS, NT)
{
    table <- NULL
    method <- c("full.truncation" ,"partial.truncation" ,"higham.mao" ,"reflection" ,"absorption")
    start.time <- proc.time()

    for(k in method){
        S0 <- 100.0
        K <- 100.0
        r <- 0.0319
        v0 <- 0.010201
        T <- 1.00
        dt <- T/NT
        rho <- -0.7
        kappa <- 6.21
        theta <- 0.019
        sigma <- 0.61

        rmse <- 0
        payoff.sum <- 0
        bias.sum <- 0
        for( j in 1:NS) {
            Zv <- 0
            Zs <- 0

            v.current <- v0
            s.path <- S0
            v.past <- 0
            for( i in 1:NT){

                if(k == "full.truncation"){
                    v.past <- max(v.current,0)
                    v.current <- v.current+kappa*dt*(theta-v.past)+sigma*sqrt(v.past*dt)*Zv
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "partial.truncation"){
                    v.past <- max(v.current,0)
```

```
                    v.current <- v.current+kappa*dt*(theta-v.current)+sigma*sqrt(v.past*dt)*Zv
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "higham.mao"){
                    v.past <- abs(v.current)
                    v.current <- v.current+kappa*dt*(theta-v.current)+sigma*sqrt(v.past*dt)*Zv
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "reflection"){
                    v.past <- abs(v.current)
                    v.current <- abs(v.current)+kappa*dt*(theta-v.past)+sigma*sqrt(v.past*dt)*Zv
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "absorption"){
                    v.past <- max(v.current,0)
                    v.current <- v.past+kappa*dt*(theta-v.past)+sigma*sqrt(v.past*dt)*Zv
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)
                }
                Zv <- rnorm(1,0,1)
                Zs <- rho*Zv+sqrt(1-rho^2)*rnorm(1,0,1)
            }
            payoff <- max(s.path-K,0)
            payoff.sum <- payoff.sum+payoff
            rmse <- rmse+(payoff*exp(-r*T)-6.8061)^2
            bias.sum <- bias.sum+(max(s.path-K,0)*exp(-r*T)-6.8061)
        }
        option.price <- (payoff.sum/NS)*exp(-r*T)
        end.time <- proc.time()
        timetaken <- end.time - start.time
        rmse <- sqrt(rmse)/NS
        bias.sum <- bias.sum/NS
        table <- rbind(table, c(method = k,price=round(option.price, 6),time.taken=round(timetaken[3],4) ,
rmse = round(rmse,4),bias= round(bias.sum,4)))
    }
    return(data.frame(table))
}
heston.Euler <- MCHestonEuler(1000000, 300)


MCHestonMilstein <- function(NS, NT)
{
    table <- NULL
    method <- c("full.truncation" ,"partial.truncation" ,"higham.mao" ,"reflection" ,"absorption")
    start.time <- proc.time()

    for(k in method){

        S0 <- 100.0
        K <- 100.0
        r <- 0.0319
        v0 <- 0.010201
        T <- 1.00
        dt <- T/NT
        rho <- -0.7
        kappa <- 6.21
        theta <- 0.019
        sigma <- 0.61
        rmse <- 0
        payoff.sum <- 0
        bias.sum <- 0

        for( j in 1:NS) {
            Zv <- 0
            Zs <- 0
            v.current <- v0
            s.path <- S0
            v.past <- 0
            for( i in 1:NT){
```

```
                if(k == "full.truncation"){
                    v.past <- max(v.current,0)
                    v.current <- v.current+kappa*dt*(theta-v.past)+sigma*sqrt(v.past*dt)*Zv + 0.25*sigma*s
igma*dt*(Zv*Zv-1.0)
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "partial.truncation"){
                    v.past <- max(v.current,0)
                    v.current <- v.current+kappa*dt*(theta-v.current)+sigma*sqrt(v.past*dt)*Zv + 0.25*sigm
a*sigma*dt*(Zv*Zv-1.0)
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "higham.mao"){
                    v.past <- abs(v.current)
                    v.current <- v.current+kappa*dt*(theta-v.current)+sigma*sqrt(v.past*dt)*Zv + 0.25*sigm
a*sigma*dt*(Zv*Zv-1.0)
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "reflection"){
                    v.past <- abs(v.current)
                    v.current <- abs(v.current)+kappa*dt*(theta-v.past)+sigma*sqrt(v.past*dt)*Zv + 0.25*si
gma*sigma*dt*(Zv*Zv-1.0)
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)

                }else if(k == "absorption"){
                    v.past <- max(v.current,0)
                    v.current <- v.past+kappa*dt*(theta-v.past)+sigma*sqrt(v.past*dt)*Zv + 0.25*sigma*sigm
a*dt*(Zv*Zv-1.0)
                    s.path <- s.path* exp((r - 0.5*v.past)*dt +sqrt(v.past*dt)*Zs)
                }
                Zv <- rnorm(1,0,1)
                Zs <- rho*Zv+sqrt(1-rho^2)*rnorm(1,0,1)
            }
            payoff <- max(s.path-K,0)
            payoff.sum <- payoff.sum+payoff
            rmse <- rmse+(payoff*exp(-r*T)-6.8061)^2
            bias.sum <- bias.sum+(max(s.path-K,0)*exp(-r*T)-6.8061)
        }
        option.price <- (payoff.sum/NS)*exp(-r*T)
        end.time <- proc.time()
        timetaken <- end.time - start.time
        rmse <- sqrt(rmse)/NS
        bias.sum <- bias.sum/NS

        table <- rbind(table, c(method = k,price=round(option.price, 6),time.taken=round(timetaken[3],4) ,
rmse = round(rmse,4),bias= round(bias.sum,4)))
    }
    return(data.frame(table))
}
heston.Milstein <- MCHestonMilstein(1000000, 300)
```

Problem 3: Multiple Monte Carlo Processes

(1)

Since we can invest per unit, when we round down, we have that the actual amount of IBM is 50,000, 10-year Treasury Bill is 33 and Chinese Yaun is 91,803. As the result, the weights of portfolio get changed to 0.4 for IBM, 0.297 for 10-year Treasury Bill and 0.29999983 for Chinese Yaun.

Asset	Weight	Money	Start price	Amount	Round down
IBM stock	0.40	$4,000,000.00	$80.00	50,000.00	50,000.00
10-year Treasury Bill	0.30	$3,000,000.00	$90,000.00	33.33	33.00
Chinese Yaun	0.30	$3,000,000.00	$6.10	91,803.28	91,803.00

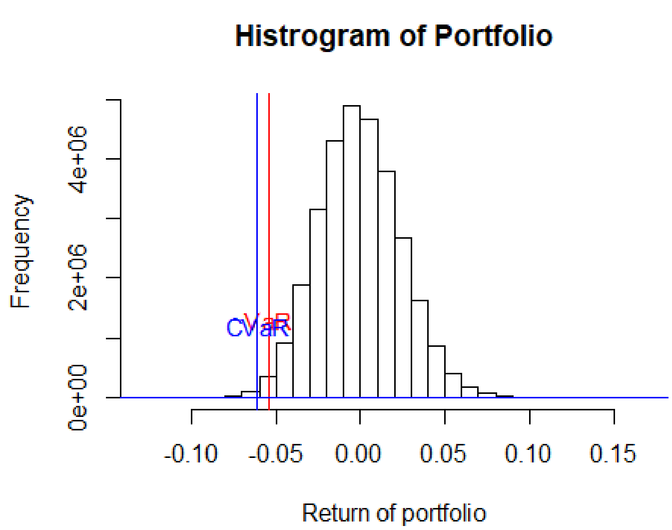
Actual Money	Actual weight
4,000,000.00	0.40000000
2,970,000.00	0.29700000
2,999,998.30	0.29999983

Total money invested	$9,969,998.30
Cash flow	$30,001.70

(2) and (3)

The VaR is 5.405913% of total money invested ($9,969,998.30) which is $538,969.4342 and CVaR is 6.110031% which is $609,169.9868.

```
## $VaR
##             [,1]
## VaR -0.05405913
##
## $CVaR
##            [,1]
## ES -0.06110031
```



Code:

```
library(PerformanceAnalytics)

MonteCarloPort <- function(X0, Y0, Z0, d, M, dt = 0.001, alpha = 0.01){
    ## Precompute constants

    T <- d/252
    N <- ceiling(T/dt)
    XT.mat <- matrix(0, nrow = M, ncol = 1)
    YT.mat <- matrix(0, nrow = M, ncol = 1)
    ZT.mat <- matrix(0, nrow = M, ncol = 1)
    port <- vector("numeric", length = M)
    port.ret <- vector("numeric", length = M)
    start.time <- proc.time()

    for (j in 1:M){

        w <- rnorm(N)
        Xt <- X0
        Yt <- Y0
        Zt <- Z0
        for(i in 1:N){
            t <- (i-1)*dt
            dXt <- 0.01*Xt*dt + 0.3*Xt*sqrt(dt)*w[i]
            dYt <- 100*(90000+1000*t-Yt)*dt+sqrt(Yt)*sqrt(dt)*w[i]
            dZt <- 5*(6 - Zt)*dt+0.01*sqrt(Zt)*sqrt(dt)*w[i]
            Xt <- max(0, Xt+dXt)
            Yt <- max(0, Yt+dYt)
            Zt <- max(0, Zt+dZt)
        }
        port[j] <- (Xt/X0-1)*0.4  + (Yt/Y0-1)*0.297  + (Zt/Z0-1)*0.29999983
    }

    end.time <- proc.time()
    timetaken <- end.time - start.time
    hist(sort(port), main = "Histrogram of Portfolio", xlab = "Return of portfolio")
    sort.port <- sort(port)
    VaR <- VaR(sort(sort.port), p = 1-alpha, method = "historical")
    CVaR <- ETL(sort.port, p = 1-alpha, method = "historical")
    abline(h=0, v=VaR,col="red")
    text(VaR, 9e5,"VaR", srt = 0.2, pos = 3, col = "red")
    abline(h=0, v=CVaR,col="blue")
    text(CVaR, 8e5,"CVaR", srt = 0.2, pos = 3, col = "blue")

    list(VaR = VaR, CVaR = CVaR)
}
risk <- MonteCarloPort(X0 = 80, Y0 = 90000, Z0 = 6.1,d = 10, M = 30000000)
```

Problem 4: (Bonus) SABR parameter estimation

(1), (2), (3) and (4)

From the table below, when we increase the beta for 2 year swaption, rho and alpha get decreased. However, the volatility fluctuates up and down. The best model comes to 0.4 beta model with the least squared error equal to 0.01790219.

```
##   beta        rho volatility    alpha     error2
## 1  0.5 -0.6922816   3.405967 3.277694 0.01989369
## 2  0.7 -0.5904847   3.368234 2.955362 0.01853600
## 3  0.4 -0.7317131   2.585623 6.436908 0.01790219
```

Code:

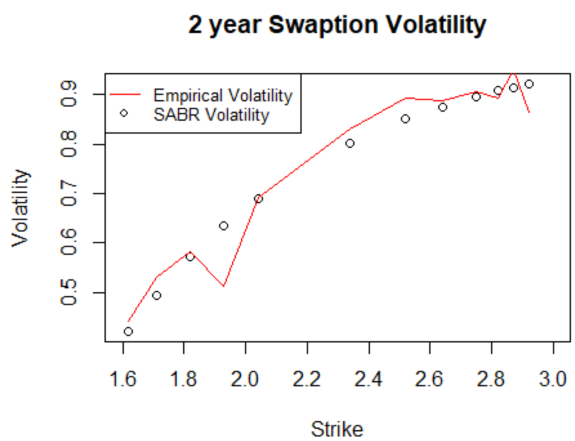
```
setwd("C://Users//nackz//Desktop//Stevens Institute//Subjects//FE621 - Computational Methods in Finance//A
ssignments//Assignment 3")
library(pracma)

Bvol <- function(alpha, beta, rho, vol, f, K, Tm){
    if(f == K) {
        term1 <- (1-beta)^2/24*alpha^2/(f^(2-2*beta))
        term2 <- 1/4*rho*beta*vol*alpha/f^(1-beta)
        term3 <- (2-3*rho^2)/24*vol^2
        sigma <- alpha*(1+(term1+term2+term3)*Tm)/(f^(1-beta))
        return(sigma)
    }
    else {
        z <- vol/alpha*(f*K)^((1-beta)/2)*log(f/K)
        Xz <- log((sqrt(1-2*rho*z+z^2)+z-rho)/(1-rho))
        term1 <- (1-beta)^2/24*alpha^2/(f*K)^(1-beta)
        term2 <- 1/4*rho*beta*vol*alpha/(f*K)^((1-beta)/2)
        term3 <- (2-3*rho^2)/24*vol^2
        term4 <- (f*K)^((1-beta)/2)
        term5 <- (1+(1-beta)^2/24*(log(f/K))^2+(1-beta)^4/1920*(log(f/K))^4)
        sigma <- alpha*(1+(term1+term2+term3)*Tm)/(term4*term5)*z/Xz
        return(sigma)
    }
}

EstimateSABR <- function(mpvol, K, beta,Tm){
    EstimateRhoVol <- function(x, beta, mkt.vol, f, K, Tm) {
        rho <- x[1]
        vol <- x[2]
        alpha <- x[3]
        if (-1 < rho && rho < 1 && vol > 0) {
            estimate.vol <- 0
            error <- 0
            for ( i in 1:length(mkt.vol)) {
                estimate.vol[i] <- Bvol(alpha, beta, rho, vol, f[i], K[i], Tm)
                error[i] <- (mkt.vol[i] - estimate.vol[i])^2
            }
            return (sum(error))
        } else {
            return(1000)
        }
    }

    opt <- nlm(EstimateRhoVol,c(-0.5,3,2),beta,mpvol/100,K, K,Tm,hessian = TRUE,ndigit=8)

    return(opt)
}
data <- read.csv("2017_2_15_mid.csv")
```

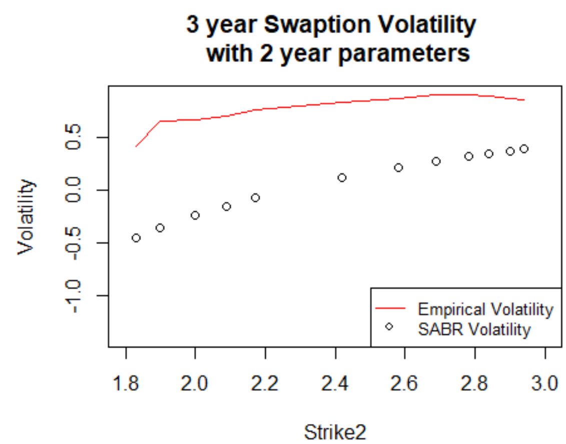
```
data1 <- data$X2Yr
mkt.vol <- data1[seq(1,length(data1),2)][1:12]
Strike <- data1[seq(2,length(data1), 2)][1:12]
result1 <- EstimateSABR(mkt.vol,Strike,0.5,2)
result2 <- EstimateSABR(mkt.vol,Strike,0.7,2)
result3 <- EstimateSABR(mkt.vol,Strike,0.4,2)

result.compare1 <- data.frame(beta = c(0.5,0.7,0.4),
                             rho = c(result1$estimate[1], result2$estimate[1], result3$estimate[1]),
                             volatility = c(result1$estimate[2], result2$estimate[2], result3$estimate[2])
,
                             alpha = c(result1$estimate[3], result2$estimate[3], result3$estimate[3]),
                             error2 = c(result1$minimum, result2$minimum, result3$minimum))
result.compare1
```

(5)

Since the 0.4 beta model is the best model from previous question, we use the same parameters to implement 3-year swaption as benchmark. From the results below, the volatility from calibrated parameters is very different to the market data. This means that we cannot use the calibrated parameters with different maturity.

```
##       Strike2      b.vol2 mkt.vol2
##  [1,]    1.83 -1.3903156   0.4198
##  [2,]    1.90 -1.2855456   0.6536
##  [3,]    2.00 -1.1454620   0.6708
##  [4,]    2.09 -1.0284590   0.7057
##  [5,]    2.17 -0.9311824   0.7617
##  [6,]    2.42 -0.6634993   0.8447
##  [7,]    2.58 -0.5172119   0.8745
##  [8,]    2.69 -0.4263298   0.9149
##  [9,]    2.78 -0.3572445   0.9050
## [10,]    2.84 -0.3136282   0.8956
## [11,]    2.90 -0.2718472   0.8802
```






```
b.vol <- 0
for ( i in 1:length(Strike)) {
    b.vol[i]= Bvol(alpha=result3$estimate[3],beta=0.4,rho=result3$estimate[1], vol=result3$est
imate[2],Strike[i],Strike[i],2)
}
b.vol
```

```
## [1] 0.4463131 0.4994205 0.5608374 0.6179361 0.6705806 0.7925975 0.8522900
## [8] 0.8872281 0.9162008 0.9332394 0.9447859

plot(Strike,b.vol,col="black",type="p",main="2 year Swaption Volatility",ylab="Volatility")
lines(Strike,mkt.vol/100,col="red",type="l")
legend("topleft", legend=c("Empirical Volatility", "SABR Volatility"), col=c("red", "black"),p
ch=c(NA,1), lty=c(1,NA),cex=0.8)

data2 <- data$X3Yr
b.vol2 <- 0
mkt.vol2 <- data2[seq(1,length(data1),2)][1:12]
mkt.vol2 <- mkt.vol2/100
Strike2 <- data2[seq(2,length(data1), 2)][1:12]

for ( i in 1:length(Strike)) {
    b.vol2[i] <- Bvol(alpha=result3$estimate[3],beta=0.4,rho=result3$estimate[1], vol=result3$estimate[2],
Strike2[i],Strike2[i],3)
}
cbind(Strike2, b.vol2, mkt.vol2)

plot(Strike2,b.vol2,col="black",type="p",main="3 year Swaption Volatility \n with 2 year parameters",ylab=
"Volatility", ylim = c(-1.4, 0.9))
lines(Strike2,mkt.vol2,col="red",type="l")
legend("bottomright", legend=c("Empirical Volatility", "SABR Volatility"), col=c("red", "black"),pch=c(NA,
1), lty=c(1,NA),cex=0.8)
```