```
###########################################################################################
#       Project
#       Created By
#       Inna Williams
#       Jonathan McGreal
#       Eleanor Marshall
###########################################################################################
Sample_GBM <- function(S_0,MT,r,sigma,d,n,X){
  delta <- MT/d
  BM<-sqrt(delta)*t(apply(X,1,cumsum))
  grid<-seq(delta,MT,length.out=d)
  S<-S_0*exp(sweep(sigma*BM,MARGIN=2,(r-sigma^2/2)*grid,'+'))
  return (S)
}
Jackpot <- setRefClass(Class = "Jackpot",
      fields = list(
        name = "character",
        d = "numeric",
        S_0 = 'numeric',
        K = 'numeric',
        T = 'numeric',
        r0 = 'numeric',
        sigma ='numeric',
        betta = 'numeric',
        delta = 'numeric',
        alpha = 'numeric',
        error_tolerance = "numeric",
        r_t = 'matrix',
        S_T = 'matrix',
        S_T_SIGMA_RATE_CONST = 'matrix',
        S_Maturity = 'numeric',
        rate_at_maturity = 'numeric',
        Discounted_payoffs = 'numeric',
        n ='numeric',
        Jackpot_Call_Option_Price = 'numeric',
        const_1='numeric',
        const_2='numeric',
        const_3= 'numeric',
        const_4='numeric',
        const_5= 'numeric',
        Z='matrix',
        Z_T='matrix',
        YX='matrix',
        X_SIGMA_RATE_CONST='matrix',
        X='matrix',
```

```
    hat_sigma="numeric",
    sample_size="numeric",
    save_n = "numeric",
    estimated_error = "numeric",
    Estimated_Fair_Price="numeric",

    GeoCallPriceConVar='numeric',
    ExactGeoCallPriceConVar='numeric',
    hat_beta='numeric',
    JackpotCallMCVPrice="numeric",
    error_sm='numeric',
    error_mcv='numeric'
  ),

  methods = list(
    GeometricMeanAsianControlVariate=function(crate)
    {
      S<-Sample_GBM(.self$S_0,.self$T,.self$r0,.self$sigma,.self$d,.self$n,.self$X)
      GeoMean<-apply(S^(1/.self$d),1,prod)

      GeoCallPayoff<-pmax(GeoMean-.self$K,0)*exp(-crate*.self$T)

      .self$GeoCallPriceConVar<-mean(GeoCallPayoff)
      Bar_T<-.self$T*(1+1/.self$d)/2
      Bar_sigma<-sqrt(.self$sigma^2*(2+1/.self$d)/3)
      Bar_r<-crate+(Bar_sigma^2-.self$sigma^2)/2
      .self$ExactGeoCallPriceConVar<-(.self$S_0*pnorm((log(.self$S_0/.self$K)+
                      (Bar_r+Bar_sigma^2/2)*Bar_T)/(Bar_sigma*sqrt(Bar_T)))-
                      .self$K*exp(-Bar_r*Bar_T)*pnorm((log(.self$S_0/.self$K)+
                      (Bar_r-Bar_sigma^2/2)*Bar_T)/(Bar_sigma*sqrt(Bar_T))))*
                      exp(Bar_r*Bar_T-crate*.self$T)

      .self$hat_beta<-cov(.self$Discounted_payoffs,GeoCallPayoff)/var(GeoCallPayoff)

      .self$JackpotCallMCVPrice<-.self$Jackpot_Call_Option_Price +
        .self$hat_beta*(.self$ExactGeoCallPriceConVar-.self$GeoCallPriceConVar)
      .self$error_sm<-abs(.self$Estimated_Fair_Price-.self$Jackpot_Call_Option_Price)
      .self$error_mcv<-abs(.self$Estimated_Fair_Price-.self$JackpotCallMCVPrice)

    },
    Calculate_Estimated_Euro_Call_Fair_Price=function(crate)
    {
        .self$Estimated_Fair_Price<-.self$S_0*pnorm((log(.self$S_0/.self$K)+
        (crate+.self$sigma^2/2)*.self$T)/(.self$sigma*sqrt(.self$T)))-
        .self$K*exp(-crate*.self$T)*pnorm((log(.self$S_0/.self$K)+
```

```
    (crate-.self$sigma^2/2)*.self$T)/(.self$sigma*sqrt(.self$T)))
},
#How do these prices compare to the price of the European call
#option with a constant interest
#rate of r = 0.07 (and constant volatility of 13% for the jackpot model)?
s_t_rate_sigma_const=function()
{
  .self$S_T_SIGMA_RATE_CONST=matrix(0, ncol=.self$d,nrow=.self$n, byrow = TRUE)
  .self$S_T_SIGMA_RATE_CONST[,1] =  exp(log(.self$S_0) + .self$X_SIGMA_RATE_CONST[,1] )
  for(i in 2:.self$d)
  {
    .self$S_T_SIGMA_RATE_CONST[,i] = exp(log(.self$S_T_SIGMA_RATE_CONST[,i-1])
                                 + .self$X_SIGMA_RATE_CONST[,i])
  }
},
r = function() {
  # const_1 = 1 - 0.18 *delta
  # intialize matrix with value = const1= 1 - 0.18 *delta
  .self$r_t=matrix(  .self$const_1, ncol=.self$d,nrow=.self$n, byrow = TRUE)

  # multiply first column of r_t on value r0 and add  value of the first
  #column of  Z_T to first column of t_t
  .self$r_t[,1]= .self$r_t[,1] * .self$r0 + .self$Z_T[,1]
  # do the same for each column starting with column to
  # but multiply instead of r0 by r[i-1]
  for(i in 2:.self$d)
  {
    .self$r_t[,i]=.self$r_t[,i]*.self$r_t[,i-1] + .self$Z_T[,i]
  }
},
s = function() {
  .self$S_T=matrix(0, ncol=.self$d,nrow=.self$n, byrow = TRUE)
  # set column 1 of the Matrix S_T= exp(log($S_0) + r0 *delta  + YX[,1])
  # where YX initialized in function init to :
  # YX = sigma*(y^(1/2))*x + const_4 where
  # const_4 = (delta/0.15)*log(  1-0.15*sigma^2/2 )
  # and calculated in initialize
  .self$S_T[,1] =  exp(log(.self$S_0)
                     + .self$r0 *.self$delta
                     + .self$YX[,1])
  for(i in 2:.self$d)
  {
    # Set column i start from 2 to the same appoch as S_T[,1]
   .self$S_T[,i] =  exp(log(.self$S_T[,i-1])
                      + (.self$r_t[,i-1]) * .self$delta
```

```
                        + .self$YX[,i])
  }
},
Run = function() {
  .self$init()
  # calculate rate r(tj)
  .self$r()
  # calculate S(tj)
  .self$s()
  # get Vector of prices at Maturiry
  .self$S_Maturity = .self$S_T[,.self$d]
  # get rate at maturity
  .self$rate_at_maturity = .self$r_t[,.self$d]
  # Calculate Discounted Payoffs = MAX(S_Maturity-K,0)*
  # exp((-1)*rate_at_maturity)
  .self$Discounted_payoffs=pmax(.self$S_Maturity-.self$K,0)*
                          exp((-1)*.self$rate_at_maturity)
  # Calculate Option Price = mean(Discounted_payoffs)
  .self$Jackpot_Call_Option_Price = mean(.self$Discounted_payoffs)
  # calculate standard deviation of the discounted payoffs
  .self$hat_sigma = sd(.self$Discounted_payoffs)
  # calculate the requared sample with error with an
  # error tolerance of $0.05.
  .self$sample_size<-
    ceiling((2.58*1.1*(.self$hat_sigma)/(.self$error_tolerance))^2)
  # calculate extimated error
  .self$estimated_error=2.58*(.self$hat_sigma)/sqrt(.self$n)

  .self$Calculate_Estimated_Euro_Call_Fair_Price( .self$r0)
  .self$GeometricMeanAsianControlVariate(.self$r0)
},
Run_For_Const_RATE_SIGMA = function() {
  .self$init()
  # calculate S(tj)
  .self$s_t_rate_sigma_const()

  # get Vector of prices at Maturiry
  .self$S_Maturity = .self$S_T_SIGMA_RATE_CONST[,.self$d]
  # Calculate Discounted Payoffs = MAX(S_Maturity-K,0)*
  # exp((-1)*rate_at_maturity)
  .self$Discounted_payoffs=pmax(.self$S_Maturity-.self$K,0)*exp((-1)*.self$r0)
  # Calculate Option Price = mean(Discounted_payoffs)
  .self$Jackpot_Call_Option_Price = mean(.self$Discounted_payoffs)
  # calculate standard deviation of the discounted payoffs
  .self$hat_sigma = sd(.self$Discounted_payoffs)
```

```
    # calculate the requared sample with error with an error tolerance of $0.05.
    .self$sample_size<-
          ceiling((2.58*1.1*(.self$hat_sigma)/
          (.self$error_tolerance))^2)
    # calculate extimated error
    .self$estimated_error=
          2.58*(.self$hat_sigma)/sqrt(.self$n)
    .self$Calculate_Estimated_Euro_Call_Fair_Price( .self$r0)
  },
  SetSampleSize=function(sample_size=.self$sample_size)
  {
    .self$n=sample_size
    .self$name=paste(.self$name , ", " , .self$n)
  },

  DisplayOutput=function(rate=.self$rate_at_maturity)
  {
    cat("##########     Start Output     ##########\n")
    cat( .self$name,"\n")
    cat("Current Sample Size = ",.self$n ,"\n")
    cat("Current Step Size = ",.self$d ,"\n")
    if(n<=50)
    {
      cat("r_t = \n")
      print(.self$r_t)
      cat("\n")
      cat("S_T = \n")
      print(.self$S_T)
      cat("\n")
      cat("S_t at Maturity = ",.self$S_Maturity,"\n")
      cat("Rate at Maturity =",rate,"\n")
      cat("Discounted Payoffs =", .self$Discounted_payoffs,"\n")
    }
    cat("Jackpot Call Option Price = ", .self$Jackpot_Call_Option_Price,"\n")
    cat("Estimated Fair Price", .self$Estimated_Fair_Price,"\n")
    cat("Sample Standard Deviation =", .self$hat_sigma, "\n")
    cat("Calculated Required Sample Size =", .self$sample_size,"\n")
    cat("Estimated Error = ", .self$estimated_error, "\n")
    cat("##########      End Output     ##########\n")
  },
  DisplayControlVarOutput=function()
  {
    cat("##########     Start Control Variate Output     ##########\n")
    cat("Asian Geometric mean call option as control variate. \n")
    cat("Asian Geometric mean call option price = ",
```

```
                          .self$GeoCallPriceConVar,"\n")
  cat("Exact Asian Geometric mean call option price = ",
                          .self$ExactGeoCallPriceConVar,"\n")
  cat("hat_beta = ", .self$hat_beta,"\n")
  cat("Jackpot Call Option Price = ", .self$Jackpot_Call_Option_Price,"\n")
  cat("Jackpot Call option MCV Price = ",
                          .self$JackpotCallMCVPrice,"\n")
  cat("Estimated Fair Price", .self$Estimated_Fair_Price,"\n")
  cat("error_sm = ", .self$error_sm,"\n")
  cat("error_mcv = ", .self$error_mcv,"\n")
  cat("##########      End Control Variate Output     ##########\n")
},
init=function()
{
  .self$Z = matrix(rnorm(.self$n*.self$d), ncol=.self$d,nrow=.self$n, byrow = TRUE)
  # const_3 = 0.02*((delta)^(1/2))
  # const_2 = 0.18*0.086*delta
  # Z_t = const_2 + const_3 * Z = 0.18*0.086*(1/12) + 0.02*(1/12)^(1/2)*(Z)
  .self$Z_T = .self$const_2 + .self$const_3*.self$Z
  # Generate Gamma
  y=(matrix(rgamma(.self$n*.self$d, shape=.self$alpha,
                  scale=.self$betta), ncol=.self$d,nrow=.self$n, byrow = TRUE))
  # Generate Normal
  .self$X=matrix(rnorm(.self$n*.self$d), ncol=.self$d,nrow=.self$n, byrow = TRUE)
  # Generate X_SIGMA_CONST for case with const volatility const_5 =
  #(r0+(log(  1-0.15*.sigma^2/2 ))/0.15)*.delta
  .self$X_SIGMA_RATE_CONST = .self$sigma*(.self$delta^(1/2))*.self$X + .self$const_5
  # add const_4 = (delta/0.15)*log(  1-0.15*sigma^2/2 )
  .self$YX = .self$sigma*(y^(1/2))*.self$X + .self$const_4
  # set zero vectors for S_Maturity, rate_at_maturity,Discounted_payoffs
  .self$S_Maturity=numeric(.self$n)
  .self$rate_at_maturity=numeric(.self$n)
  .self$Discounted_payoffs=numeric(.self$n)
},
initialize=function(...) {
  callSuper(...)

  .self
  .self$delta = .self$T/.self$d
  .self$alpha=.self$delta/.self$betta

  .self$const_1 = 1 - 0.18 *.self$delta
  .self$const_2 = 0.18*0.086*.self$delta
  .self$const_3 = 0.02*((.self$delta)^(1/2))
  .self$const_4 = (.self$delta/0.15)*log(  1-0.15*.self$sigma^2/2 )
```

```
            # this const for const rate and constant volatility
            .self$const_5 = (.self$r0+(log(  1-0.15*.self$sigma^2/2 ))/0.15)*.self$delta
        }
    )
)


            # Script to run for a given step = d and given start size = n
            Run_Jackpot=function(step,size,tname,number_of_tries)
            {
              s <- Jackpot$new(name = tname, d=step, S_0=50, K=50,T=1,
                               r0=0.07,sigma =0.13, betta=0.15,error_tolerance=0.05,n=size)
              s$Run()
              s$DisplayOutput()
              sample_size = 0
              for(i in 1:number_of_tries)
              {
                s$SetSampleSize()
                sample_size = sample_size+s$sample_size
                s$Run()
              }
              s$DisplayOutput()
              sample_size = sample_size/number_of_tries
              cat("Sample size = ", sample_size," for d = ",s$d,"\n")
              return(s)
            }
            Run_Jackpot_WithDetermined_Size=function(step,size,tname,
                                                rate_type="variable",con_var=FALSE)
            {
              s <- Jackpot$new(name = tname, d=step, S_0=50, K=50,T=1,
                               r0=0.07,sigma =0.13, betta=0.15,error_tolerance=0.05,n=size)
              if(rate_type=='const')
              {
                s$Run_For_Const_RATE_SIGMA()
                s$DisplayOutput(s$r0)
              }
              else
              {
                 s$Run()
                 if(con_var==TRUE)
                   s$DisplayControlVarOutput()
                 else
                   s$DisplayOutput()
              }
            }
```

```
    # this is the try run to check the data set d=12 and n=10
    Run_Jackpot_WithDetermined_Size(12,10,"Test for d=12 n=10")

    # Run case with d=12, size = 10^4
    o=Run_Jackpot(12,10000,"Test for d=12 n=10000",10)
    # Run case with d=52, size = 1000
    o=Run_Jackpot(52,1000, "Test for d=52 n=1000",10)

####################################################################
############    Questions      #####################################
# What is the sample size required?
# Sample size requared is ~ 88000

# Run for d=12 after size was determined = 88000
Run_Jackpot_WithDetermined_Size(12,88000,"Test for d=12 n=88000")
# Run for d=52  after size was determined = 88000
Run_Jackpot_WithDetermined_Size(52,88000,"Test for d=52 n=88000")

# Which estimated price is higher? Time step of 1 month or time step of 1 week?
# Run the code for d = 4,12,26,52,104,365 and compare the prices. How the output
# depends on different d
Run_Jackpot_WithDetermined_Size(4,88000,"Test for d=4 different d size n=88000")
Run_Jackpot_WithDetermined_Size(12,88000,"Test for d=12 different d size n=88000")
Run_Jackpot_WithDetermined_Size(26,88000,"Test for d=26 different d size n=88000")
Run_Jackpot_WithDetermined_Size(52,88000,"Test for d=52 different d size n=88000")
Run_Jackpot_WithDetermined_Size(104,88000,"Test for d=104 different d size n=88000")
Run_Jackpot_WithDetermined_Size(365,88000,"Test for d=365 different d sizen=88000")

# Run for constant rate and constant volatility for d=12,52
Run_Jackpot_WithDetermined_Size(12,88000,
                        "Test for d=12 n=88000 const rate const volatility",'const')
Run_Jackpot_WithDetermined_Size(52,88000,
                        "Test for d=52 n=88000 const rate const volatility",'const')
# Use GeoMean Control Variate
Run_Jackpot_WithDetermined_Size(12,88000,"Test for d=12 n=88000",'variable',TRUE)
Run_Jackpot_WithDetermined_Size(52,88000,"Test for d=52 n=88000",'variable',TRUE)
```