Module 8.4: SRM ABM-Based Option Models
R Commentary

See *SRM ABM CT OVM Test.R*.

SRM ABM CT OVM Test.R

The main program for testing SRMs based on the ABMOVM.

```
# SRM ABM CT OVM Test.R
# ABMOVM, ES and AS BOVMs
# rmarkdown::render("SRM ABM CT OVM Test.R", "word_document")
rm(list = ls())  # Take out the Environment "trash"
cat("\014") # Clear Console, making error checking easier.
while (!is.null(dev.list()))  dev.off() # Clear old plots
par(family = 'Times New Roman') # Globally set fonts for graphs
# Libraries
#  beepr - functions for beeping to let you know when program is finished
Packages <- c("beepr")
if(length(setdiff(Packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(Packages, rownames(installed.packages())))
} # Make sure libraries are installed on this computer
lapply(Packages, library, character.only=TRUE) # Load and attach libraries
rm(Packages)
# Functions defined in:
#   ABMOVM w Greeks Functions.R
#   ESABMBINOVM With SRM Functions.R (European-style only)
#   ASABMBINOVM With SRM Functions.R (American-style also)
#
# Test inputs
# ABMOVM inputs
inputStockPrice = 100       # Need "input" as using variable names below
inputStrikePrice = 100
inputInterestRate = 5.0        # In percent
inputDividendYield = 5.0       # In percent
inputVolatility = 29.884768295208 # In percent
inputTimeToMaturity = 1.0
inputType = 1                  # 1 for call, -1 for put
# Additional BOVM inputs
inputEMMProbability = 50.0     # In percent
inputGreekIncrement = 1.0      # In percent of greek underlying variable
inputNumberOfSteps = as.integer(250) # Or use L: 250 or 1000L
inputPayoutType = 1L        # 1 Plain vanilla, 2 digital (digital not built)
inputStyle = 2              # 1 for terminal ES, 2 for AS *** NOT USED NOW ***???
inputDigitalPayout = 100    # In dollars
# Plot ranges
LowerBound = inputStockPrice*0.5 # Stock price
UpperBound = inputStockPrice*1.5
LowerBoundV = inputVolatility*0.5 # Volatility
UpperBoundV = inputVolatility*1.5
LowerBoundT = inputTimeToMaturity*0.5 # Time to maturity
UpperBoundT = inputTimeToMaturity*1.5
LowerBoundD = 0.0 # Dividend yield
UpperBoundD = 25.0
NumberOfObservations = 51
#
#  BINInputData - list of inputs with associated names
#
BINInputData <- list(inputStockPrice, inputStrikePrice, inputInterestRate,
  inputDividendYield, inputVolatility, inputTimeToMaturity, inputType,
  inputNumberOfSteps, inputPayoutType, inputStyle,
  inputEMMProbability, inputDigitalPayout, inputGreekIncrement)
names(BINInputData) <- c("StockPrice", "StrikePrice", "InterestRate",
  "DividendYield", "Volatility", "TimeToMaturity", "Type",
  "NumberOfSteps", "PayoutType", "Style",
```

```
  "EMMProbability", "DigitalPayout", "GreekIncrement")
#
# ABM list
#
ABMInputData <- list(inputStockPrice, inputStrikePrice, inputInterestRate,
  inputDividendYield, inputVolatility, inputTimeToMaturity, inputType)
names(ABMInputData) <- c("StockPrice", "StrikePrice", "InterestRate",
  "DividendYield", "Volatility", "TimeToMaturity", "Type")
#
# Source the appropriate functions
#
source("ABMOVM With SRM Functions.R")
source('ESABMBINOVM With SRM Functions.R')
source('ASABMBINOVM With SRM Functions.R')
# Console Rounding Setting
RDigits <- 4
source("Core Functions Tests.R")
source("ABMOVM Stock Price Plots.R")
source("ABMOVM Time To Maturity Plots.R")
source("ABMOVM Volatility Plots.R")
source("ABMOVM Dividend Yield Plots.R")
beep(sound = 3, print('Finished')) # fanfare
```

ESABMBINOVM With SRM Functions.R (Selected Excerpts and Output)

The complete function for delta is given based on European-style options.

```
#
# Delta Direct Enhanced Method -- have to do full backward recursion
#
ESBINOptionDeltaDirectEnh <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OriginalNumberOfSteps <- NumberOfSteps
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    TimeToMaturity <- TimeToMaturity + 2.0*Delta
    NumberOfSteps <- NumberOfSteps + 2
    OHigh <- OLow <- SHigh <- SLow <- 0
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Rate <- InterestRate/100.0  # Local variable, in decimal
    Sigma <- Volatility/100.0   # Local variable, in decimal
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    PeriodRate <- exp(Rate*Delta)             # Periodic rate (1+r)
    PeriodDriftRate <- exp(DriftRate*Delta)   # Periodic rate (1+r)
    AdjustedStockPrice <- StockPrice
    Prob <- EMMProbability/100.0
    A <- exp( Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob)) )
    Up <- ( PeriodDriftRate * A ) / (Prob*A + (1-Prob))
    Down <- PeriodDriftRate / (Prob*A + (1-Prob))
    if(Up < PeriodRate || Down > PeriodRate)return(-99)
    dSteps <- as.numeric(NumberOfSteps)
    TNS <- NumberOfSteps + 1
    OptionValue <- c(1:TNS)
    # OptionValue <- c(1:NumberOfSteps)
    for (TimeStep in NumberOfSteps:0){
      dTimeStep <- as.numeric(TimeStep)
# Inner loop
      for (i in 0:TimeStep){
        if(TimeStep == NumberOfSteps){  # At expiration
          if(Type == 1){    # Plain vanilla call
            Moneyness <- (Up^i) * (Down^(TimeStep-i)) * AdjustedStockPrice -
              StrikePrice
```

```
            OptionValue[i+1] <- max(0, Moneyness)
          }
          if(Type == -1){   # Plain vanilla put
            Moneyness <- StrikePrice -
              ((Up^i)*(Down^(TimeStep-i)) * AdjustedStockPrice)
            OptionValue[i+1] <- max(0, Moneyness)
          }
        } else { # Prior to expiration
          OptionValue[i+1] <- (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
        }
        if(TimeStep == 2){
          if(i == 0){
            OLow <- OptionValue[i+1]
            SLow <- (Up^i) * (Down^(TimeStep-i)) * AdjustedStockPrice
            B$StockPrice <- SLow
            B$TimeToMaturity <- OriginalTimeToMaturity
# Check lower boundary conditions
            if(PayoutType == 1){    # Plain vanilla option
              LowerBoundL <- ESBINOptionLowerBound(B)
            } else {
              LowerBoundL <- 0
            }
            LowerBoundL <- max(0, LowerBoundL)
            OLow <- max(OLow, LowerBoundL)
            B$TimeToMaturity <- OriginalTimeToMaturity + 2*Delta
            B$StockPrice <- OriginalStockPrice
          }
          if(i == 2){
            OHigh <- OptionValue[i+1]
            SHigh <- (Up^i) * (Down^(TimeStep-i)) * AdjustedStockPrice
            B$StockPrice <- SHigh
            B$TimeToMaturity <- OriginalTimeToMaturity - 2*Delta
# Check lower boundary conditions
            if(PayoutType == 1){    # Plain vanilla option
              LowerBoundH <- ESBINOptionLowerBound(B)
            } else {
              LowerBoundH <- 0
            }
            LowerBoundH <- max(0, LowerBoundH)
            OHigh <- max(OHigh, LowerBoundH)
            B$TimeToMaturity <- OriginalTimeToMaturity + 2*Delta
            B$StockPrice <- OriginalStockPrice
          }
        }
      }
    }
    Value <- (OHigh - OLow) / (SHigh - SLow)
    return(Value)
  })
}
```

ASABMBINOVM With SRM Functions.R

```
# ASABMBINOVM With SRM Functions.R
#  American-style, geometric Brownian motion, binomial OVM
# Present value function (See ESABMBINOVM With SRM Functions.R)
# Binomial probability
source('ESABMBINOVM With SRM Functions.R')

# American-style lower bound
#
# NEED TO FIX for ES version -- check dividend yield
#
```

```
ASBINOptionLowerBound <- function(B){
  with(B, {
    LowerBound <- Type * (StockPrice * PV1(TimeToMaturity, DividendYield) -
      StrikePrice * PV1(TimeToMaturity, InterestRate))
    IntrinsicValue <- Type * (StockPrice - StrikePrice)
    LowerBound <- max(0, LowerBound, IntrinsicValue)
    return( LowerBound )
  })
}
#
# American-style upper bound
#
ASBINOptionUpperBound <- function(B){
  with(B, {
    if(Type == 1){
      UpperBound <-
        StockPrice * PV1(TimeToMaturity, DividendYield)
      UpperBound <- min(UpperBound, StockPrice)
    }
    if(Type == -1){
      UpperBound <-
        StrikePrice * PV1(TimeToMaturity, InterestRate)
      UpperBound <- max(UpperBound, max(0, StrikePrice - StockPrice))
    }
    return( UpperBound )
  })
}
#
# American-style binomial option valuation function (ABM)
#
ABMASOptionValue = function(B){
  with(B,{
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
    for (TimeStep in NumberOfSteps:0){
      TTM <- Delta*(NumberOfSteps - TimeStep)
      for (StateStep in 0:TimeStep){
        if(TimeStep == NumberOfSteps){
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          CallValue[StateStep+1] <- max(0, S - StrikePrice)
          PutValue[StateStep+1] <- max(0, StrikePrice - S)
          if(S > StrikePrice){
            DigitalCallValue[StateStep+1] <- DigitalPayout
            DigitalPutValue[StateStep+1] <- 0
          } else {
            DigitalCallValue[StateStep+1] <- 0
            DigitalPutValue[StateStep+1] <- DigitalPayout
          }
        } else {
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
```

```
          phi <- (S*R - Down)/(Up - Down)
          CallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
          PutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
          DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalCallValue[StateStep+2] +
            (1 - phi)*DigitalCallValue[StateStep+1])
          DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalPutValue[StateStep+2] +
            (1 - phi)*DigitalPutValue[StateStep+1])

# Check for lower boundary violation
          IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
          IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM), StrikePrice - S )

          IVDCall <- 0
          IVDPut <- 0
          CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
          PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
          DigitalCallValue[StateStep+1] <-
            max(DigitalCallValue[StateStep+1], IVDCall)
          DigitalPutValue[StateStep+1] <-
            max(DigitalPutValue[StateStep+1], IVDPut)
        }
      }
    }
    CV <- CallValue[1]
    PV <- PutValue[1]
    DCV <- DigitalCallValue[1]
    DPV <- DigitalPutValue[1]
    ABMOptionOutput <- list(CV, PV, DCV, DPV)
    names(ABMOptionOutput) <- c("CallValue", "PutValue", "DigitalCallValue",
      "DigitalPutValue")
    return(ABMOptionOutput)
  })
}
#
# Delta Direct
#
ABMASBINOptionDeltaDirect <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OHigh <- OLow <- SHigh <- SLow <- 0
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
    for (TimeStep in NumberOfSteps:0){
```

```
  TTM <- Delta*(NumberOfSteps - TimeStep)
  for (StateStep in 0:TimeStep){
    if(TimeStep == NumberOfSteps){
      S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
      CallValue[StateStep+1] <- max(0, S - StrikePrice)
      PutValue[StateStep+1] <- max(0, StrikePrice - S)
      if(S > StrikePrice){
        DigitalCallValue[StateStep+1] <- DigitalPayout
        DigitalPutValue[StateStep+1] <- 0
      } else {
        DigitalCallValue[StateStep+1] <- 0
        DigitalPutValue[StateStep+1] <- DigitalPayout
      }
    } else {
      S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
      phi <- (S*R - Down)/(Up - Down)
      CallValue[StateStep+1] <- (1/PeriodRate) *
        (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
      PutValue[StateStep+1] <- (1/PeriodRate) *
        (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
      DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
        (phi*DigitalCallValue[StateStep+2] +
        (1 - phi)*DigitalCallValue[StateStep+1])
      DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
        (phi*DigitalPutValue[StateStep+2] +
        (1 - phi)*DigitalPutValue[StateStep+1])

      # Check for lower boundary violation
      IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
                      StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
      IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
                    S*exp(-(DividendYield/100)*TTM), StrikePrice - S )
      IVDCall <- 0
      IVDPut <- 0
      CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
      PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
      DigitalCallValue[StateStep+1] <-
        max(DigitalCallValue[StateStep+1], IVDCall)
      DigitalPutValue[StateStep+1] <-
        max(DigitalPutValue[StateStep+1], IVDPut)
    }
    if(TimeStep == 1){ # Increment time to maturity and time to PV Div
      if(StateStep == 0){
        OLowCallValue <- CallValue[StateStep+1]
        OLowPutValue <- PutValue[StateStep+1]
        OLowDigitalCallValue <- DigitalCallValue[StateStep+1]
        OLowDigitalPutValue <- DigitalPutValue[StateStep+1]
        SLow <- S
      }
      if(StateStep == 1){
        OHighCallValue <- CallValue[StateStep+1]
        OHighPutValue <- PutValue[StateStep+1]
        OHighDigitalCallValue <- DigitalCallValue[StateStep+1]
        OHighDigitalPutValue <- DigitalPutValue[StateStep+1]
        SHigh <- S
      }
    }
  }
}
CV <- (OHighCallValue - OLowCallValue)/((SHigh - SLow))
PV <- (OHighPutValue - OLowPutValue)/((SHigh - SLow))
DCV <- (OHighDigitalCallValue - OLowDigitalCallValue)/((SHigh - SLow))
DPV <- (OHighDigitalPutValue - OLowDigitalPutValue)/((SHigh - SLow))
ABMOptionOutput <- list(CV, PV, DCV, DPV)
```

```
    names(ABMOptionOutput) <- c("ASCallDelta", "ASPutDelta",
                                "ASDigitalCallDelta", "ASDigitalPutDelta")
    return(ABMOptionOutput)
  })
}
#
# Delta Direct Enhanced Method -- have to do full backward recursion
#
ABMASBINOptionDeltaDirectEnh <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OriginalNumberOfSteps <- NumberOfSteps
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    TimeToMaturity <- TimeToMaturity + 2.0*Delta
    NumberOfSteps <- NumberOfSteps + 2
    OHigh <- OLow <- SHigh <- SLow <- 0
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    # Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
    for (TimeStep in NumberOfSteps:0){
      TTM <- Delta*(NumberOfSteps - TimeStep)
      for (StateStep in 0:TimeStep){
        if(TimeStep == NumberOfSteps){
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          CallValue[StateStep+1] <- max(0, S - StrikePrice)
          PutValue[StateStep+1] <- max(0, StrikePrice - S)
          if(S > StrikePrice){
            DigitalCallValue[StateStep+1] <- DigitalPayout
            DigitalPutValue[StateStep+1] <- 0
          } else {
            DigitalCallValue[StateStep+1] <- 0
            DigitalPutValue[StateStep+1] <- DigitalPayout
          }
        } else {
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          phi <- (S*R - Down)/(Up - Down)
          CallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
          PutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
          DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalCallValue[StateStep+2] +
               (1 - phi)*DigitalCallValue[StateStep+1])
          DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalPutValue[StateStep+2] +
               (1 - phi)*DigitalPutValue[StateStep+1])

          # Check for lower boundary violation
          IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
                          StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
```

```
          IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
                        S*exp(-(DividendYield/100)*TTM), StrikePrice - S )

          IVDCall <- 0
          IVDPut <- 0
          CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
          PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
          DigitalCallValue[StateStep+1] <-
            max(DigitalCallValue[StateStep+1], IVDCall)
          DigitalPutValue[StateStep+1] <-
            max(DigitalPutValue[StateStep+1], IVDPut)
        }
        if(TimeStep == 2){ # Increment time to maturity and time to PV Div
          if(StateStep == 0){
            OLowCallValue <- CallValue[StateStep+1]
            OLowPutValue <- PutValue[StateStep+1]
            OLowDigitalCallValue <- DigitalCallValue[StateStep+1]
            OLowDigitalPutValue <- DigitalPutValue[StateStep+1]
            SLow <- S
          }
          if(StateStep == 2){
            OHighCallValue <- CallValue[StateStep+1]
            OHighPutValue <- PutValue[StateStep+1]
            OHighDigitalCallValue <- DigitalCallValue[StateStep+1]
            OHighDigitalPutValue <- DigitalPutValue[StateStep+1]
            SHigh <- S
          }
        }
      }
    }
    CV <- (OHighCallValue - OLowCallValue)/((SHigh - SLow))
    PV <- (OHighPutValue - OLowPutValue)/((SHigh - SLow))
    DCV <- (OHighDigitalCallValue - OLowDigitalCallValue)/((SHigh - SLow))
    DPV <- (OHighDigitalPutValue - OLowDigitalPutValue)/((SHigh - SLow))
    ABMOptionOutput <- list(CV, PV, DCV, DPV)
    names(ABMOptionOutput) <- c("ASCallDelta", "ASPutDelta",
                                "ASDigitalCallDelta", "ASDigitalPutDelta")
    return(ABMOptionOutput)
  })
}
#
# Gamma Direct
#
ABMASBINOptionGammaDirect <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OHigh <- OLow <- SHigh <- SLow <- 0
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
```

```
for (TimeStep in NumberOfSteps:0){
  TTM <- Delta*(NumberOfSteps - TimeStep)
  for (StateStep in 0:TimeStep){
    if(TimeStep == NumberOfSteps){
      S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
      CallValue[StateStep+1] <- max(0, S - StrikePrice)
      PutValue[StateStep+1] <- max(0, StrikePrice - S)
      if(S > StrikePrice){
        DigitalCallValue[StateStep+1] <- DigitalPayout
        DigitalPutValue[StateStep+1] <- 0
      } else {
        DigitalCallValue[StateStep+1] <- 0
        DigitalPutValue[StateStep+1] <- DigitalPayout
      }
    } else {
      S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
      phi <- (S*R - Down)/(Up - Down)
      CallValue[StateStep+1] <- (1/PeriodRate) *
        (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
      PutValue[StateStep+1] <- (1/PeriodRate) *
        (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
      DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
        (phi*DigitalCallValue[StateStep+2] +
           (1 - phi)*DigitalCallValue[StateStep+1])
      DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
        (phi*DigitalPutValue[StateStep+2] +
           (1 - phi)*DigitalPutValue[StateStep+1])

      # Check for lower boundary violation
      IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
                      StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
      IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
                    S*exp(-(DividendYield/100)*TTM), StrikePrice - S )
      IVDCall <- 0
      IVDPut <- 0
      CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
      PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
      DigitalCallValue[StateStep+1] <-
        max(DigitalCallValue[StateStep+1], IVDCall)
      DigitalPutValue[StateStep+1] <-
        max(DigitalPutValue[StateStep+1], IVDPut)
    }
    if(TimeStep == 2){ # Increment time to maturity and time to PV Div
      if(StateStep == 0){
        OLowCallValue <- CallValue[StateStep+1]
        OLowPutValue <- PutValue[StateStep+1]
        OLowDigitalCallValue <- DigitalCallValue[StateStep+1]
        OLowDigitalPutValue <- DigitalPutValue[StateStep+1]
        SLow <- S
      }
      if(StateStep == 1){
        OMidCallValue <- CallValue[StateStep+1]
        OMidPutValue <- PutValue[StateStep+1]
        OMidDigitalCallValue <- DigitalCallValue[StateStep+1]
        OMidDigitalPutValue <- DigitalPutValue[StateStep+1]
        SMid <- S
      }
      if(StateStep == 2){
        OHighCallValue <- CallValue[StateStep+1]
        OHighPutValue <- PutValue[StateStep+1]
        OHighDigitalCallValue <- DigitalCallValue[StateStep+1]
        OHighDigitalPutValue <- DigitalPutValue[StateStep+1]
        SHigh <- S
      }
```

```
        }
      }
    }
    CV <- ( (OHighCallValue - OMidCallValue)/(SHigh - SMid) -
              (OMidCallValue - OLowCallValue)/(SMid - SLow) ) / (0.5*(SHigh - SLow))
    PV <- ( (OHighPutValue - OMidPutValue)/(SHigh - SMid) -
              (OMidPutValue - OLowPutValue)/(SMid - SLow) ) / (0.5*(SHigh - SLow))
    DCV <- ( (OHighDigitalCallValue - OMidDigitalCallValue)/(SHigh - SMid) -
               (OMidDigitalCallValue - OLowDigitalCallValue)/(SMid - SLow) ) /
      (0.5*(SHigh - SLow))
    DPV <- ( (OHighDigitalPutValue - OMidDigitalPutValue)/(SHigh - SMid) -
               (OMidDigitalPutValue - OLowDigitalPutValue)/(SMid - SLow) ) /
      (0.5*(SHigh - SLow))
    ABMOptionOutput <- list(CV, PV, DCV, DPV)
    names(ABMOptionOutput) <- c("ASCallGamma", "ASPutGamma",
                                "ASDigitalCallGamma", "ASDigitalPutGamma")
    return(ABMOptionOutput)
  })
}
#
# Gamma Direct Enhanced
#
ABMASBINOptionGammaDirectEnh<- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OriginalNumberOfSteps <- NumberOfSteps
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    TimeToMaturity <- TimeToMaturity + 2.0*Delta
    NumberOfSteps <- NumberOfSteps + 2
    OHigh <- OLow <- SHigh <- SLow <- 0
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    # Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
    for (TimeStep in NumberOfSteps:0){
      TTM <- Delta*(NumberOfSteps - TimeStep)
      for (StateStep in 0:TimeStep){
        if(TimeStep == NumberOfSteps){
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          CallValue[StateStep+1] <- max(0, S - StrikePrice)
          PutValue[StateStep+1] <- max(0, StrikePrice - S)
          if(S > StrikePrice){
            DigitalCallValue[StateStep+1] <- DigitalPayout
            DigitalPutValue[StateStep+1] <- 0
          } else {
            DigitalCallValue[StateStep+1] <- 0
            DigitalPutValue[StateStep+1] <- DigitalPayout
          }
        } else {
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          phi <- (S*R - Down)/(Up - Down)
```

```
      CallValue[StateStep+1] <- (1/PeriodRate) *
        (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
      PutValue[StateStep+1] <- (1/PeriodRate) *
        (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
      DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
        (phi*DigitalCallValue[StateStep+2] +
           (1 - phi)*DigitalCallValue[StateStep+1])
      DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
        (phi*DigitalPutValue[StateStep+2] +
           (1 - phi)*DigitalPutValue[StateStep+1])

      # Check for lower boundary violation
      IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
                      StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
      IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
                    S*exp(-(DividendYield/100)*TTM), StrikePrice - S )
      IVDCall <- 0
      IVDPut <- 0
      CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
      PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
      DigitalCallValue[StateStep+1] <-
        max(DigitalCallValue[StateStep+1], IVDCall)
      DigitalPutValue[StateStep+1] <-
        max(DigitalPutValue[StateStep+1], IVDPut)
    }
    if(TimeStep == 2){ # Increment time to maturity and time to PV Div
      if(StateStep == 0){
        OLowCallValue <- CallValue[StateStep+1]
        OLowPutValue <- PutValue[StateStep+1]
        OLowDigitalCallValue <- DigitalCallValue[StateStep+1]
        OLowDigitalPutValue <- DigitalPutValue[StateStep+1]
        SLow <- S
      }
      if(StateStep == 1){
        OMidCallValue <- CallValue[StateStep+1]
        OMidPutValue <- PutValue[StateStep+1]
        OMidDigitalCallValue <- DigitalCallValue[StateStep+1]
        OMidDigitalPutValue <- DigitalPutValue[StateStep+1]
        SMid <- S
      }
      if(StateStep == 2){
        OHighCallValue <- CallValue[StateStep+1]
        OHighPutValue <- PutValue[StateStep+1]
        OHighDigitalCallValue <- DigitalCallValue[StateStep+1]
        OHighDigitalPutValue <- DigitalPutValue[StateStep+1]
        SHigh <- S
      }
    }
  }
}
CV <- ( (OHighCallValue - OMidCallValue)/(SHigh - SMid) -
          (OMidCallValue - OLowCallValue)/(SMid - SLow) ) / (0.5*(SHigh - SLow))
PV <- ( (OHighPutValue - OMidPutValue)/(SHigh - SMid) -
          (OMidPutValue - OLowPutValue)/(SMid - SLow) ) / (0.5*(SHigh - SLow))
DCV <- ( (OHighDigitalCallValue - OMidDigitalCallValue)/(SHigh - SMid) -
           (OMidDigitalCallValue - OLowDigitalCallValue)/(SMid - SLow) ) /
  (0.5*(SHigh - SLow))
DPV <- ( (OHighDigitalPutValue - OMidDigitalPutValue)/(SHigh - SMid) -
           (OMidDigitalPutValue - OLowDigitalPutValue)/(SMid - SLow) ) /
  (0.5*(SHigh - SLow))
ABMOptionOutput <- list(CV, PV, DCV, DPV)
names(ABMOptionOutput) <- c("ASCallGamma", "ASPutGamma",
                            "ASDigitalCallGamma", "ASDigitalPutGamma")
return(ABMOptionOutput)
```

```
  })
}
#
# Theta Direct
#
ABMASBINOptionThetaDirect <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OHigh <- OLow <- SHigh <- SLow <- 0
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
    for (TimeStep in NumberOfSteps:0){
      TTM <- Delta*(NumberOfSteps - TimeStep)
      for (StateStep in 0:TimeStep){
        if(TimeStep == NumberOfSteps){
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          CallValue[StateStep+1] <- max(0, S - StrikePrice)
          PutValue[StateStep+1] <- max(0, StrikePrice - S)
          if(S > StrikePrice){
            DigitalCallValue[StateStep+1] <- DigitalPayout
            DigitalPutValue[StateStep+1] <- 0
          } else {
            DigitalCallValue[StateStep+1] <- 0
            DigitalPutValue[StateStep+1] <- DigitalPayout
          }
        } else {
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          phi <- (S*R - Down)/(Up - Down)
          CallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
          PutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
          DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalCallValue[StateStep+2] +
               (1 - phi)*DigitalCallValue[StateStep+1])
          DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalPutValue[StateStep+2] +
               (1 - phi)*DigitalPutValue[StateStep+1])

          # Check for lower boundary violation
          IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
                            StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
          IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
                        S*exp(-(DividendYield/100)*TTM), StrikePrice - S )

          IVDCall <- 0
          IVDPut <- 0
          CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
          PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
```

```
          DigitalCallValue[StateStep+1] <-
            max(DigitalCallValue[StateStep+1], IVDCall)
          DigitalPutValue[StateStep+1] <-
            max(DigitalPutValue[StateStep+1], IVDPut)
        }
        if(TimeStep == 2){ # Increment time to maturity and time to PV Div
          if(StateStep == 1){
            OMidCallValue <- CallValue[StateStep+1]
            OMidPutValue <- PutValue[StateStep+1]
            OMidDigitalCallValue <- DigitalCallValue[StateStep+1]
            OMidDigitalPutValue <- DigitalPutValue[StateStep+1]
            SMid <- S
          }
        }
      }
    }
    CV <- (OMidCallValue - CallValue[1])/(2.0*Delta)
    PV <- (OMidPutValue - PutValue[1])/(2.0*Delta)
    DCV <- (OMidDigitalCallValue - DigitalCallValue[1])/(2.0*Delta)
    DPV <- (OMidDigitalPutValue - DigitalPutValue[1])/(2.0*Delta)
    ABMOptionOutput <- list(CV, PV, DCV, DPV)
    names(ABMOptionOutput) <- c("ASCallTheta", "ASPutTheta",
                                "ASDigitalCallTheta", "ASDigitalPutTheta")
    return(ABMOptionOutput)
  })
}
#
# Theta Direct Enhanced
#
ABMASBINOptionThetaDirectEnh <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OriginalNumberOfSteps <- NumberOfSteps
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    TimeToMaturity <- TimeToMaturity + 2.0*Delta
    NumberOfSteps <- NumberOfSteps + 2
    OHigh <- OLow <- SHigh <- SLow <- 0
    CallValue <- numeric(NumberOfSteps+1)
    PutValue <- numeric(NumberOfSteps+1)
    DigitalCallValue <- numeric(NumberOfSteps+1)
    DigitalPutValue <- numeric(NumberOfSteps+1)
    Rate = InterestRate/100.0  # Local variable, decimal
    Dividend = DividendYield/100.0
    Sigma = Volatility              # Local variable, in DOLLARS
    # Delta = TimeToMaturity / NumberOfSteps
    PeriodRate = exp(Rate*Delta)
    PeriodDiv = exp(Dividend*Delta)
    PeriodRateMDiv = exp((Rate - Dividend)*Delta)
    R <- PeriodRateMDiv - 1
    Prob = EMMProbability/100.0
    A = Sigma*sqrt(Delta)/sqrt(Prob*(1-Prob))
    Up = (1 - Prob)*A
    Down = -Prob*A
    for (TimeStep in NumberOfSteps:0){
      TTM <- Delta*(NumberOfSteps - TimeStep)
      for (StateStep in 0:TimeStep){
        if(TimeStep == NumberOfSteps){
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          CallValue[StateStep+1] <- max(0, S - StrikePrice)
          PutValue[StateStep+1] <- max(0, StrikePrice - S)
          if(S > StrikePrice){
            DigitalCallValue[StateStep+1] <- DigitalPayout
            DigitalPutValue[StateStep+1] <- 0
```

```
          } else {
            DigitalCallValue[StateStep+1] <- 0
            DigitalPutValue[StateStep+1] <- DigitalPayout
          }
        } else {
          S <- StockPrice + StateStep*Up + (TimeStep - StateStep)*Down
          phi <- (S*R - Down)/(Up - Down)
          CallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*CallValue[StateStep+2] + (1 - phi)*CallValue[StateStep+1])
          PutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*PutValue[StateStep+2] + (1 - phi)*PutValue[StateStep+1])
          DigitalCallValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalCallValue[StateStep+2] +
               (1 - phi)*DigitalCallValue[StateStep+1])
          DigitalPutValue[StateStep+1] <- (1/PeriodRate) *
            (phi*DigitalPutValue[StateStep+2] +
               (1 - phi)*DigitalPutValue[StateStep+1])

          # Check for lower boundary violation
          IVCall <- max( 0, S*exp(-(DividendYield/100)*TTM) -
                            StrikePrice*exp(-(InterestRate/100)*TTM), S - StrikePrice )
          IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
                        S*exp(-(DividendYield/100)*TTM), StrikePrice - S )

          IVDCall <- 0
          IVDPut <- 0
          CallValue[StateStep+1] <- max(CallValue[StateStep+1], IVCall)
          PutValue[StateStep+1] <- max(PutValue[StateStep+1], IVPut)
          DigitalCallValue[StateStep+1] <-
            max(DigitalCallValue[StateStep+1], IVDCall)
          DigitalPutValue[StateStep+1] <-
            max(DigitalPutValue[StateStep+1], IVDPut)
        }
        if(TimeStep == 4){ # Increment time to maturity and time to PV Div
          if(StateStep == 2){
            OMidCallValue <- CallValue[StateStep+1]
            OMidPutValue <- PutValue[StateStep+1]
            OMidDigitalCallValue <- DigitalCallValue[StateStep+1]
            OMidDigitalPutValue <- DigitalPutValue[StateStep+1]
            SMid <- S
          }
        }
      }
    }
    CV <- (OMidCallValue - CallValue[1])/(4.0*Delta)
    PV <- (OMidPutValue - PutValue[1])/(4.0*Delta)
    DCV <- (OMidDigitalCallValue - DigitalCallValue[1])/(4.0*Delta)
    DPV <- (OMidDigitalPutValue - DigitalPutValue[1])/(4.0*Delta)
    ABMOptionOutput <- list(CV, PV, DCV, DPV)
    names(ABMOptionOutput) <- c("ASCallTheta", "ASPutTheta",
                                "ASDigitalCallTheta", "ASDigitalPutTheta")
    return(ABMOptionOutput)
  })
}
#
# Numerical Greeks
# Delta
ABMASBINOptionNDelta <- function(B){
  Original <- B$StockPrice
  Change <- (B$GreekIncrement/100.0)*Original
  SHigh <- Original + Change
  B$StockPrice <- SHigh
  OHigh <- ABMASOptionValue(B)
  SLow <- Original - Change
```

```
  B$StockPrice <- SLow
  OLow <- ABMASOptionValue(B)
  B$StockPrice <- Original
  CV <- (OHigh$CallValue - OLow$CallValue)/((SHigh - SLow))
  PV <- (OHigh$PutValue - OLow$PutValue)/((SHigh - SLow))
  DCV <- (OHigh$DigitalCallValue - OLow$DigitalCallValue)/((SHigh - SLow))
  DPV <- (OHigh$DigitalPutValue - OLow$DigitalPutValue)/((SHigh - SLow))
  ABMOptionOutput <- list(CV, PV, DCV, DPV)
  names(ABMOptionOutput) <- c("ASCallDelta", "ASPutDelta",
                              "ASDigitalCallDelta", "ASDigitalPutDelta")
  return( ABMOptionOutput )
}
#
# Gamma: Goes to zero for large time steps (UNSTABLE)
#
ABMASBINOptionNGamma <- function(B){
  Original <- B$StockPrice
  Change <- (B$GreekIncrement/100.0)*Original*100
  SHigh <- Original + Change
  B$StockPrice <- SHigh
  OHigh <- ABMASOptionValue(B)
  SLow <- Original - Change
  B$StockPrice <- SLow
  OLow <- ABMASOptionValue(B)
  B$StockPrice <- Original
  OMid <- ABMASOptionValue(B)
  CV <- ( (OHigh$CallValue - OMid$CallValue) -
            (OMid$CallValue - OLow$CallValue) )/(Change*Change)
  PV <- ( (OHigh$PutValue - OMid$PutValue) -
            (OMid$PutValue - OLow$PutValue) )/(Change*Change)
  DCV <- ( (OHigh$DigitalCallValue - OMid$DigitalCallValue) -
             (OMid$DigitalCallValue - OLow$DigitalCallValue) )/(Change*Change)
  DPV <- ( (OHigh$DigitalPutValue - OMid$DigitalPutValue) -
             (OMid$DigitalPutValue - OLow$DigitalPutValue) )/(Change*Change)
  ABMOptionOutput <- list(CV, PV, DCV, DPV)
  names(ABMOptionOutput) <- c("ASCallGamma", "ASPutGamma",
                              "ASDigitalCallGamma", "ASDigitalPutGamma")
  return( ABMOptionOutput )
}
# # Theta
ABMASBINOptionNTheta <- function(B){
  Original <- B$TimeToMaturity
  Change <- (B$GreekIncrement/100.0)*Original*(1/10)
  SHigh <- Original + Change
  B$TimeToMaturity <- SHigh
  OHigh <- ABMASOptionValue(B)
  SLow <- Original - Change
  B$TimeToMaturity <- SLow
  OLow <- ABMASOptionValue(B)
  B$TimeToMaturity <- Original
  CV <- -(OHigh$CallValue - OLow$CallValue)/((SHigh - SLow))
  PV <- -(OHigh$PutValue - OLow$PutValue)/((SHigh - SLow))
  DCV <- -(OHigh$DigitalCallValue - OLow$DigitalCallValue)/((SHigh - SLow))
  DPV <- -(OHigh$DigitalPutValue - OLow$DigitalPutValue)/((SHigh - SLow))
  ABMOptionOutput <- list(CV, PV, DCV, DPV)
  names(ABMOptionOutput) <- c("ASCallTheta", "ASPutTheta",
                              "ASDigitalCallTheta", "ASDigitalPutTheta")
  return( ABMOptionOutput )
}
#
# Vega
#
ABMASBINOptionNVega <- function(B){
  Original <- B$Volatility
```

```
  Change <- (B$GreekIncrement/100.0)*Original
  SHigh <- Original + Change
  B$Volatility <- SHigh
  OHigh <- ABMASOptionValue(B)
  SLow <- Original - Change
  B$Volatility <- SLow
  OLow <- ABMASOptionValue(B)
  B$Volatility <- Original
  CV <- (OHigh$CallValue - OLow$CallValue)/((SHigh - SLow))
  PV <- (OHigh$PutValue - OLow$PutValue)/((SHigh - SLow))
  DCV <- (OHigh$DigitalCallValue - OLow$DigitalCallValue)/((SHigh - SLow))
  DPV <- (OHigh$DigitalPutValue - OLow$DigitalPutValue)/((SHigh - SLow))
  ABMOptionOutput <- list(CV, PV, DCV, DPV)
  names(ABMOptionOutput) <- c("ASCallVega", "ASPutVega",
                              "ASDigitalCallVega", "ASDigitalPutVega")
  return( ABMOptionOutput )
}
#
# Rho
#
ABMASBINOptionNRho <- function(B){
  Original <- B$InterestRate
  Change <- (B$GreekIncrement/100.0)*Original
  SHigh <- Original + Change
  B$InterestRate <- SHigh
  OHigh <- ABMASOptionValue(B)
  SLow <- Original - Change
  B$InterestRate <- SLow
  OLow <- ABMASOptionValue(B)
  B$InterestRate <- Original
  CV <- (OHigh$CallValue - OLow$CallValue)/((SHigh - SLow))
  PV <- (OHigh$PutValue - OLow$PutValue)/((SHigh - SLow))
  DCV <- (OHigh$DigitalCallValue - OLow$DigitalCallValue)/((SHigh - SLow))
  DPV <- (OHigh$DigitalPutValue - OLow$DigitalPutValue)/((SHigh - SLow))
  ABMOptionOutput <- list(CV, PV, DCV, DPV)
  names(ABMOptionOutput) <- c("ASCallRho", "ASPutRho",
                              "ASDigitalCallRho", "ASDigitalPutRho")
  return( ABMOptionOutput )
}
```