Module 8.1: SRM GBM-Based Binomial Models
R Commentary

See module *Ch 8.1 SRM GBM Binomial OVM*. The test program is *SRM GBM Binomial OVM American-Style Test.R*. Technical functions for European-style options are contained in *ESGBMBINOVM With SRM Functions.R* and technical functions for American-style options are contained in *ASGBMBINOVM With SRM Functions.R*.

We now review selected snippets of R code.

SRM GBM Binomial OVM European-Style Test.R (Selected Excerpts and Output)

There are two ways to estimate the European-style option value based on the binomial model. You could estimate the terminal distribution, compute the expected cash flow, and then take the present value. Alternatively, you can use backward recursion like what is required with the binomial model. Both methods are provided, and the switch `TERMINAL` allows one to compare results.

```
# Full (AS) or Terminal ES method
TERMINAL <- TRUE
for(i in 1:NumberOfObservations){
  TStockPrice[i] <-LowerBoundSP + (i - 1)*StepSizeSP
  BINInputData$Style = 1 # 1-ES, 2-AS (AS code only)
  BINInputData$StockPrice = TStockPrice[i]
  BINInputData$Type = 1L
# Plain vanilla calls
  BINInputData$PayoutType = 1L
  if(TERMINAL){
    ESCallLB[i] <- ESBINOptionLowerBound(BINInputData)
```

This program only illustrates the sensitivity with respect to the stock price. Depending on your hardware, this program will take a while to run. Thus, we use a beeper to let us know when it is finished.

```
beep(sound = 3, print('Finished')) # fanfare
```

One can easily incorporate discrete cash dividend payments, but the code is cumbersome to manage and is not provided here.

8.1 SRM GBM Binomial OVM American-Style Test.R

The main test program is provided below.

```
# 8.1 SRM GBM Binomial OVM American-Style Test.R
# Geometric Brownian Motion
# Illustrating American-style binomial option valuation and
#  related functions in R
# rmarkdown::render("SRM GBM Binomial OVM American-Style 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)
```

Note that the R source call below itself calls the European-style function.

```
source('ASGBMBINOVM With SRM Functions.R')
# Test inputs
inputStockPrice = 100.0          # Need "input" as using variable names below
inputStrikePrice = 100.0         # In currency units, numeric
inputInterestRate = 5.0          # In percent
inputDividendYield = 0.0         # In percent
inputVolatility = 30.0           # In percent
```

```
inputTimeToMaturity = 1.0        # In fraction of year
inputType = 1L                   # 1 for call, -1 for put
# NEW for American-style (include ES for audit)
inputStyle = 1                   # 1 for European-style, 2 for American-style
inputNumberOfSteps = as.integer(250)   # Or use L: 1000L
inputPayoutType = 1L             # 1 Plain vanilla, 2 digital (digital not built)
inputEMMProbability = 50.0       # In percent
inputGreekIncrement = 0.1        # For numerical derivatives, % of variable
inputDigitalPayout = 100.0
LowerBoundSP = 20 # Analysis wrt stock price
UpperBoundSP = 180
# LowerBound = inputStockPrice*0.05
# UpperBound = inputStockPrice*2.0
# LowerBoundV = inputVolatility*0.5
# UpperBoundV = inputVolatility*1.5
# LowerBoundT = inputTimeToMaturity*0.5
# UpperBoundT = inputTimeToMaturity*1.5
NumberOfObservations = 101
StepSize = 1L # Analysis wrt number of steps
MinStep = 5L # Must be multiple of StepSize
MaxStep = 500L # Must be multiple of StepSize
# Full (AS) or Terminal ES method
TERMINAL <- TRUE
# Plot footers
TS = paste0('S=', inputStockPrice)
TX = paste0(',X=', inputStrikePrice)
TR = paste0(',r=', inputInterestRate)
Td = paste0(',d=', inputDividendYield, ',')
TV = paste0('Vol=', inputVolatility)
TT = paste0(',T=',inputTimeToMaturity)
TN = paste0(',N=',inputNumberOfSteps)
TDP = paste0(',DP=', inputDigitalPayout)
TI = paste0(',Incr.=', inputGreekIncrement)
sTitleBIN = paste0(TS,TX, TR, Td, TV, TT, TN)
sTitleBINDP = paste0(TS,TX, TR, Td, TV, TT, TN, TDP)
sTitleBINGk = paste0(TS,TX, TR, Td, TV, TT, TN, TI)
#
#  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")
#
```

It is always good to run newly created functions through a variety of tests.

```
# Selected functions test
#  ES - European-style
#  AS - American-style
#  LB - Lower bound
#  OV - Option value
#  C or P - Call or put
#  TD - Test derivative (standard method)
#  TDE - Test derivative (enhanced method)
#  TN - Test derivative (numerical method)
#
# Value test
BINInputData$Style = 1 # ES or AS (not used in functions below)
BINInputData$Type <- 1 # Call
ESLB <- ESBINOptionLowerBound(BINInputData)
ASLB <- ASBINOptionLowerBound(BINInputData)
```

```
ESOV <- ESBINOptionValue(BINInputData)
ASOV <- ASBINOptionValue(BINInputData)
ESLB; ASLB
ESOV; ASOV
BINInputData$Type <- -1 # Put
ESLB <- ESBINOptionLowerBound(BINInputData)
ASLB <- ASBINOptionLowerBound(BINInputData)
ESOV <- ESBINOptionValue(BINInputData)
ASOV <- ASBINOptionValue(BINInputData)
ESLB; ASLB
ESOV; ASOV
# Delta test
BINInputData$Style = 1
BINInputData$Type <- 1
ESCTD <- ESBINOptionDeltaDirect(BINInputData)
ASCTD <- ASBINOptionDeltaDirect(BINInputData)
ESCTDE <- ESBINOptionDeltaDirectEnh(BINInputData)
ASCTDE <- ASBINOptionDeltaDirectEnh(BINInputData)
ESCTN <- ESBINOptionDelta(BINInputData)
ASCTN <- ASBINOptionDelta(BINInputData)
ESCTD; ESCTDE; ESCTN
ASCTD; ASCTDE; ASCTN
BINInputData$Type <- -1
ESPTD <- ESBINOptionDeltaDirect(BINInputData)
ASPTD <- ASBINOptionDeltaDirect(BINInputData)
ESPTDE <- ESBINOptionDeltaDirectEnh(BINInputData)
ASPTDE <- ASBINOptionDeltaDirectEnh(BINInputData)
ESPTN <- ESBINOptionDelta(BINInputData)
ASPTN <- ASBINOptionDelta(BINInputData)
ESPTD; ESPTDE; ESPTN
ASPTD; ASPTDE; ASPTN
# Gamma test
BINInputData$Style = 1
BINInputData$Type <- 1
ESCTD <- ESBINOptionGammaDirect(BINInputData)
ASCTD <- ASBINOptionGammaDirect(BINInputData)
ESCTDE <- ESBINOptionGammaDirectEnh(BINInputData)
ASCTDE <- ASBINOptionGammaDirectEnh(BINInputData)
ESCTN <- ESBINOptionGamma(BINInputData)
ASCTN <- ASBINOptionGamma(BINInputData)
ESCTD; ESCTDE; ESCTN
ASCTD; ASCTDE; ASCTN
BINInputData$Type <- -1
ESPTD <- ESBINOptionGammaDirect(BINInputData)
ASPTD <- ASBINOptionGammaDirect(BINInputData)
ESPTDE <- ESBINOptionGammaDirectEnh(BINInputData)
ASPTDE <- ASBINOptionGammaDirectEnh(BINInputData)
ESPTN <- ESBINOptionGamma(BINInputData)
ASPTN <- ASBINOptionGamma(BINInputData)
ESPTD; ESPTDE; ESPTN
ASPTD; ASPTDE; ASPTN
# Theta test
BINInputData$Style = 1
BINInputData$Type <- 1
ESCTD <- ESBINOptionThetaDirect(BINInputData)
ASCTD <- ASBINOptionThetaDirect(BINInputData)
ESCTDE <- ESBINOptionThetaDirectEnh(BINInputData)
ASCTDE <- ASBINOptionThetaDirectEnh(BINInputData)
ESCTN <- ESBINOptionTheta(BINInputData)
ASCTN <- ASBINOptionTheta(BINInputData)
ESCTD; ESCTDE; ESCTN
ASCTD; ASCTDE; ASCTN
BINInputData$Type <- -1
ESPTD <- ESBINOptionThetaDirect(BINInputData)
```

```
ASPTD <- ASBINOptionThetaDirect(BINInputData)
ESPTDE <- ESBINOptionThetaDirectEnh(BINInputData)
ASPTDE <- ASBINOptionThetaDirectEnh(BINInputData)
ESPTN <- ESBINOptionTheta(BINInputData)
ASPTN <- ASBINOptionTheta(BINInputData)
ESPTD; ESPTDE; ESPTN
ASPTD; ASPTDE; ASPTN
# Vega test
BINInputData$Style = 1
BINInputData$Type <- 1
ESCTN <- ESBINOptionVega(BINInputData)
ASCTN <- ASBINOptionVega(BINInputData)
ESCTN
ASCTN
BINInputData$Type <- -1
ESPTN <- ESBINOptionVega(BINInputData)
ASPTN <- ASBINOptionVega(BINInputData)
ESPTN
ASPTN
# Rho test
BINInputData$Style = 1
BINInputData$Type <- 1
ESCTN <- ESBINOptionRho(BINInputData)
ASCTN <- ASBINOptionRho(BINInputData)
ESCTN
ASCTN
BINInputData$Type <- -1
ESPTN <- ESBINOptionRho(BINInputData)
ASPTN <- ASBINOptionRho(BINInputData)
ESPTN
ASPTN
#
# Analysis of Stock Price
#
StepSizeSP = (UpperBoundSP - LowerBoundSP)/(NumberOfObservations - 1)
TStockPrice <- c(1:NumberOfObservations)
CallIV <- c(1:NumberOfObservations)
PutIV <- c(1:NumberOfObservations)
ESCallLB <- c(1:NumberOfObservations)
ESCallUB <- c(1:NumberOfObservations)
ESCallValue <- c(1:NumberOfObservations)
ESCallDeltaDirect <- c(1:NumberOfObservations)
ESCallDeltaDirectEnh <- c(1:NumberOfObservations)
ESCallDeltaNG <- c(1:NumberOfObservations)
ESCallGammaDirect <- c(1:NumberOfObservations)
ESCallGammaDirectEnh <- c(1:NumberOfObservations)
ESCallGammaNG <- c(1:NumberOfObservations)
ESCallThetaDirect <- c(1:NumberOfObservations)
ESCallThetaDirectEnh <- c(1:NumberOfObservations)
ESCallThetaNG <- c(1:NumberOfObservations)
ESCallVegaNG <- c(1:NumberOfObservations)
ESCallRhoNG <- c(1:NumberOfObservations)
# Puts
ESPutLB <- c(1:NumberOfObservations)
ESPutUB <- c(1:NumberOfObservations)
ESPutValue <- c(1:NumberOfObservations)
ESPutDeltaDirect <- c(1:NumberOfObservations)
ESPutDeltaDirectEnh <- c(1:NumberOfObservations)
ESPutDeltaNG <- c(1:NumberOfObservations)
ESPutGammaDirect <- c(1:NumberOfObservations)
ESPutGammaDirectEnh <- c(1:NumberOfObservations)
ESPutGammaNG <- c(1:NumberOfObservations)
ESPutThetaDirect <- c(1:NumberOfObservations)
ESPutThetaDirectEnh <- c(1:NumberOfObservations)
```

```
ESPutThetaNG <- c(1:NumberOfObservations)
ESPutVegaNG <- c(1:NumberOfObservations)
ESPutRhoNG <- c(1:NumberOfObservations)
# American-style
ASCallLB <- c(1:NumberOfObservations)
ASCallUB <- c(1:NumberOfObservations)
ASCallValue <- c(1:NumberOfObservations)
ASCallDeltaDirect <- c(1:NumberOfObservations)
ASCallDeltaDirectEnh <- c(1:NumberOfObservations)
ASCallDeltaNG <- c(1:NumberOfObservations)
ASCallGammaDirect <- c(1:NumberOfObservations)
ASCallGammaDirectEnh <- c(1:NumberOfObservations)
ASCallGammaNG <- c(1:NumberOfObservations)
ASCallThetaDirect <- c(1:NumberOfObservations)
ASCallThetaDirectEnh <- c(1:NumberOfObservations)
ASCallThetaNG <- c(1:NumberOfObservations)
ASCallVegaNG <- c(1:NumberOfObservations)
ASCallRhoNG <- c(1:NumberOfObservations)
# Puts
ASPutLB <- c(1:NumberOfObservations)
ASPutUB <- c(1:NumberOfObservations)
ASPutValue <- c(1:NumberOfObservations)
ASPutDeltaDirect <- c(1:NumberOfObservations)
ASPutDeltaDirectEnh <- c(1:NumberOfObservations)
ASPutDeltaNG <- c(1:NumberOfObservations)
ASPutGammaDirect <- c(1:NumberOfObservations)
ASPutGammaDirectEnh <- c(1:NumberOfObservations)
ASPutGammaNG <- c(1:NumberOfObservations)
ASPutThetaDirect <- c(1:NumberOfObservations)
ASPutThetaDirectEnh <- c(1:NumberOfObservations)
ASPutThetaNG <- c(1:NumberOfObservations)
ASPutVegaNG <- c(1:NumberOfObservations)
ASPutRhoNG <- c(1:NumberOfObservations)
# Digital options
DigitalCallValue <- c(1:NumberOfObservations)
DigitalPutValue <- c(1:NumberOfObservations)
# # Full (AS) or Terminal ES method
# TERMINAL <- TRUE
for(i in 1:NumberOfObservations){
  TStockPrice[i] <-LowerBoundSP + (i - 1)*StepSizeSP
  BINInputData$Style = 1 # 1-ES, 2-AS (AS code only)
  BINInputData$StockPrice = TStockPrice[i]
  BINInputData$Type = 1L
# Plain vanilla calls
  BINInputData$PayoutType = 1L
  CallIV[i] <- OptionIntrinsicValue(BINInputData)
  if(TERMINAL){
    ESCallLB[i] <- ESBINOptionLowerBound(BINInputData)
    ESCallUB[i] <- ESBINOptionUpperBound(BINInputData)
    ESCallValue[i] <- ESBINOptionValue(BINInputData)
    ESCallDeltaDirect[i] <- ESBINOptionDeltaDirect(BINInputData)
    ESCallDeltaDirectEnh[i] <-
      ESBINOptionDeltaDirectEnh(BINInputData)
    ESCallDeltaNG[i] <- ESBINOptionDelta(BINInputData)
    ESCallGammaDirect[i] <- ESBINOptionGammaDirect(BINInputData)
    ESCallGammaDirectEnh[i] <-
      ESBINOptionGammaDirectEnh(BINInputData)
    ESCallGammaNG[i] <- ESBINOptionGamma(BINInputData)
    ESCallThetaDirect[i] <- ESBINOptionThetaDirect(BINInputData)
    ESCallThetaDirectEnh[i] <-
      ESBINOptionThetaDirectEnh(BINInputData)
    ESCallThetaNG[i] <- ESBINOptionTheta(BINInputData)
    ESCallVegaNG[i] <- ESBINOptionVega(BINInputData)
    ESCallRhoNG[i] <- ESBINOptionRho(BINInputData)
```

```
  } else {
    ESCallLB[i] <- ASBINOptionLowerBound(BINInputData)
    ESCallUB[i] <- ASBINOptionUpperBound(BINInputData)
    ESCallValue[i] <- ASBINOptionValue(BINInputData)
    ESCallDeltaDirect[i] <- ASBINOptionDeltaDirect(BINInputData)
    ESCallDeltaDirectEnh[i] <-
      ASBINOptionDeltaDirectEnh(BINInputData)
    ESCallDeltaNG[i] <- ASBINOptionDelta(BINInputData)
    ESCallGammaDirect[i] <- ASBINOptionGammaDirect(BINInputData)
    ESCallGammaDirectEnh[i] <-
      ASBINOptionGammaDirectEnh(BINInputData)
    ESCallGammaNG[i] <- ASBINOptionGamma(BINInputData)
    ESCallThetaDirect[i] <- ASBINOptionThetaDirect(BINInputData)
    ESCallThetaDirectEnh[i] <-
      ASBINOptionThetaDirectEnh(BINInputData)
    ESCallThetaNG[i] <- ASBINOptionTheta(BINInputData)
    ESCallVegaNG[i] <- ASBINOptionVega(BINInputData)
    ESCallRhoNG[i] <- ASBINOptionRho(BINInputData)
  }
# American-style
  BINInputData$Style = 2 # 1-ES, 2-AS (AS code only)
  ASCallLB[i] <- ASBINOptionLowerBound(BINInputData)
  ASCallUB[i] <- ASBINOptionUpperBound(BINInputData)
  ASCallValue[i] <- ASBINOptionValue(BINInputData)
  ASCallDeltaDirect[i] <- ASBINOptionDeltaDirect(BINInputData)
  ASCallDeltaDirectEnh[i] <-
    ASBINOptionDeltaDirectEnh(BINInputData)
  ASCallDeltaNG[i] <- ASBINOptionDelta(BINInputData)
  ASCallGammaDirect[i] <- ASBINOptionGammaDirect(BINInputData)
  ASCallGammaDirectEnh[i] <-
    ASBINOptionGammaDirectEnh(BINInputData)
  ASCallGammaNG[i] <- ASBINOptionGamma(BINInputData)
  ASCallThetaDirect[i] <- ASBINOptionThetaDirect(BINInputData)
  ASCallThetaDirectEnh[i] <-
    ASBINOptionThetaDirectEnh(BINInputData)
  ASCallThetaNG[i] <- ASBINOptionTheta(BINInputData)
  ASCallVegaNG[i] <- ASBINOptionVega(BINInputData)
  ASCallRhoNG[i] <- ASBINOptionRho(BINInputData)
# Plain vanilla puts
  BINInputData$Style = 1 # 1-ES, 2-AS (AS code only)
  BINInputData$Type = -1L
  PutIV[i] <- OptionIntrinsicValue(BINInputData)
  if(TERMINAL){
    ESPutLB[i] <- ESBINOptionLowerBound(BINInputData)
    ESPutUB[i] <- ESBINOptionUpperBound(BINInputData)
    ESPutValue[i] <- ESBINOptionValue(BINInputData)
    ESPutDeltaDirect[i] <- ESBINOptionDeltaDirect(BINInputData)
    ESPutDeltaDirectEnh[i] <-
      ESBINOptionDeltaDirectEnh(BINInputData)
    ESPutDeltaNG[i] <- ESBINOptionDelta(BINInputData)
    ESPutGammaDirect[i] <- ESBINOptionGammaDirect(BINInputData)
    ESPutGammaDirectEnh[i] <-
      ESBINOptionGammaDirectEnh(BINInputData)
    ESPutGammaNG[i] <- ESBINOptionGamma(BINInputData)
    ESPutThetaDirect[i] <- ESBINOptionThetaDirect(BINInputData)
    ESPutThetaDirectEnh[i] <-
      ESBINOptionThetaDirectEnh(BINInputData)
    ESPutThetaNG[i] <- ESBINOptionTheta(BINInputData)
    ESPutVegaNG[i] <- ESBINOptionVega(BINInputData)
    ESPutRhoNG[i] <- ESBINOptionRho(BINInputData)
  } else {
    ESPutLB[i] <- ASBINOptionLowerBound(BINInputData)
    ESPutUB[i] <- ASBINOptionUpperBound(BINInputData)
    ESPutValue[i] <- ASBINOptionValue(BINInputData)
```

```
    ESPutDeltaDirect[i] <- ASBINOptionDeltaDirect(BINInputData)
    ESPutDeltaDirectEnh[i] <-
      ASBINOptionDeltaDirectEnh(BINInputData)
    ESPutDeltaNG[i] <- ASBINOptionDelta(BINInputData)
    ESPutGammaDirect[i] <- ASBINOptionGammaDirect(BINInputData)
    ESPutGammaDirectEnh[i] <-
      ASBINOptionGammaDirectEnh(BINInputData)
    ESPutGammaNG[i] <- ASBINOptionGamma(BINInputData)
    ESPutThetaDirect[i] <- ASBINOptionThetaDirect(BINInputData)
    ESPutThetaDirectEnh[i] <-
      ASBINOptionThetaDirectEnh(BINInputData)
    ESPutThetaNG[i] <- ASBINOptionTheta(BINInputData)
    ESPutVegaNG[i] <- ASBINOptionVega(BINInputData)
    ESPutRhoNG[i] <- ASBINOptionRho(BINInputData)
  }
#American-style
  BINInputData$Style = 2 # 1-ES, 2-AS (AS code only)
  ASPutLB[i] = ASBINOptionLowerBound(BINInputData)
  ASPutUB[i] = ASBINOptionUpperBound(BINInputData)
  ASPutValue[i] = ASBINOptionValue(BINInputData)
  ASPutDeltaDirect[i] <- ASBINOptionDeltaDirect(BINInputData)
  ASPutDeltaDirectEnh[i] <-
    ASBINOptionDeltaDirectEnh(BINInputData)
  ASPutDeltaNG[i] <- ASBINOptionDelta(BINInputData)
  ASPutGammaDirect[i] <- ASBINOptionGammaDirect(BINInputData)
  ASPutGammaDirectEnh[i] <-
    ASBINOptionGammaDirectEnh(BINInputData)
  ASPutGammaNG[i] <- ASBINOptionGamma(BINInputData)
  ASPutThetaDirect[i] <- ASBINOptionThetaDirect(BINInputData)
  ASPutThetaDirectEnh[i] <-
    ASBINOptionThetaDirectEnh(BINInputData)
  ASPutThetaNG[i] <- ASBINOptionTheta(BINInputData)
  ASPutVegaNG[i] <- ASBINOptionVega(BINInputData)
  ASPutRhoNG[i] <- ASBINOptionRho(BINInputData)
}
BINInputData$StockPrice = inputStockPrice # Reset from previous analysis

source('SRM GBM Binomial OVM European Style Plots.R')
source('SRM GBM Binomial OVM American Style Plots.R')
source('SRM GBM Binomial OVM Comparison Plots.R')

beep(sound = 3, print('Finished')) # fanfare
```

ASGBMBINOVM With SRM Functions.R

American-style GBM-based binomial option valuation model with static risk measures functions are documented below. Note the valuation model functions are contained in a separate file, but sourced here.

```
# ASGBMBINOVM With SRM Functions.R
#  American-style, geometric Brownian motion, binomial OVM
# Present value function (See ESGBMBINOVM With SRM Functions.R)
# Binomial probability
source('ESGBMBINOVM With SRM Functions.R')
# Built on ASGBMBINOVM Functions.R
source('ASGBMBINOVM Functions.R')
#
# Delta Direct -- have to do full backward recursion
#
ASBINOptionDeltaDirect <- function(B){
  with(B, {
    # OriginalTimeToMaturity <- TimeToMaturity
    # OriginalStockPrice <- StockPrice
    OHigh <- OLow <- SHigh <- SLow <- 0
    N <- NumberOfSteps + 1
    OptionValue <- c(1:N)
```

```
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    Rate <- InterestRate/100.0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Sigma <- Volatility/100.0   # Local variable, in decimal
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    PeriodRate <- exp(Rate*Delta)
    PeriodDriftRate <- exp(DriftRate*Delta)
    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)
    for (TimeStep in NumberOfSteps:0){
      dTimeStep <- as.numeric(TimeStep)
      for (i in 0:TimeStep){
        S <- (Up^i) * (Down^(TimeStep-i)) * StockPrice
        if(TimeStep == NumberOfSteps){ # At expiration
          OptionValue[i+1] = max(0, Type*(S - StrikePrice))
        } else {                          # Prior to expiration
          B$TimeToMaturity <- TimeToMaturity - dTimeStep*Delta
          B$StockPrice <- S
          LowerBound <- ASBINOptionLowerBound(B)
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          OptionValue[i+1] = max(LowerBound, OptionValue[i+1])
        }
        if(TimeStep == 1){ # Increment time to maturity and time to PV Div
          if(i == 0){
            OLow <- OptionValue[i+1]
            SLow <- S
          }
          if(i == 1){
            OHigh <- OptionValue[i+1]
            SHigh <- S
          }
        }
      }
    }
    Value <- (OHigh - OLow) / (SHigh - SLow)
    return(Value)
  })
}
#
# Delta Direct Enhanced (+2 Delta) -- have to do full backward recursion
#
ASBINOptionDeltaDirectEnh <- function(B){
  with(B, {
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    TimeToMaturity <- TimeToMaturity + 2.0*Delta
    NumberOfSteps <- NumberOfSteps + 2
    OHigh <- OLow <- SHigh <- SLow <- 0
    N <- NumberOfSteps + 1
    OptionValue <- c(1:N)
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    Rate <- InterestRate/100.0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Sigma <- Volatility/100.0   # Local variable, in decimal
    PeriodRate <- exp(Rate*Delta)
    PeriodDriftRate <- exp(DriftRate*Delta)
```

```
    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)
    for (TimeStep in NumberOfSteps:0){
      dTimeStep <- as.numeric(TimeStep)
      for (i in 0:TimeStep){
        S <- (Up^i) * (Down^(TimeStep-i)) * StockPrice
        if(TimeStep == NumberOfSteps){ # At expiration
          OptionValue[i+1] = max(0, Type*(S - StrikePrice))
        } else { # Prior to expiration
          B$TimeToMaturity <- TimeToMaturity - dTimeStep*Delta
          B$StockPrice <- S
          LowerBound <- ASBINOptionLowerBound(B)
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          OptionValue[i+1] = max(LowerBound, OptionValue[i+1])
        }
        if(TimeStep == 2){
          if(i == 0){
            OLow <- OptionValue[i+1]
            SLow <- S
          }
          if(i == 2){
            OHigh <- OptionValue[i+1]
            SHigh <- S
          }
        }
      }
    }
    Value <- (OHigh - OLow) / (SHigh - SLow)
    return(Value)
  })
}
#
# Gamma Direct
#
ASBINOptionGammaDirect <- function(B){
  with(B, {
    OHigh <- OLow <- SHigh <- SLow <- SMid <- OMid <- 0
    N <- NumberOfSteps + 1
    OptionValue <- c(1:N)
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    Rate <- InterestRate/100.0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Sigma <- Volatility/100.0   # Local variable, in decimal
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    PeriodRate <- exp(Rate*Delta)
    PeriodDriftRate <- exp(DriftRate*Delta)
    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)
    for (TimeStep in NumberOfSteps:0){
      dTimeStep <- as.numeric(TimeStep)
      for (i in 0:TimeStep){
```

```
        S <- (Up^i) * (Down^(TimeStep-i)) * StockPrice
        if(TimeStep == NumberOfSteps){ # At expiration
          OptionValue[i+1] = max(0, Type*(S - StrikePrice))
        } else { # Prior to expiration
          B$TimeToMaturity <- TimeToMaturity - dTimeStep*Delta
          B$StockPrice <- S
          LowerBound <- ASBINOptionLowerBound(B)
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          OptionValue[i+1] = max(LowerBound, OptionValue[i+1])
        }
        if(TimeStep == 2){
          if(i == 0){
            OLow <- OptionValue[i+1]
            SLow <- S
          }
          if(i == 1){
            OMid <- OptionValue[i+1]
            SMid <- S
          }
          if(i == 2){
            OHigh <- OptionValue[i+1]
            SHigh <- S
          }
        }
      }
    }
    Value <- ( (OHigh - OMid)/(SHigh - SMid) -
      (OMid - OLow)/(SMid - SLow) ) / (0.5*(SHigh - SLow))
    return(Value)
  })
}
#
# Gamma Direct Enhanced
#
ASBINOptionGammaDirectEnh <- 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 <- SMid <- OMid <- 0
    N <- NumberOfSteps + 1
    OptionValue <- c(1:N)
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    Rate <- InterestRate/100.0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Sigma <- Volatility/100.0   # Local variable, in decimal
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    PeriodRate <- exp(Rate*Delta)
    PeriodDriftRate <- exp(DriftRate*Delta)
    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)
    for (TimeStep in NumberOfSteps:0){
      dTimeStep <- as.numeric(TimeStep)
```

```
      for (i in 0:TimeStep){
        S <- (Up^i) * (Down^(TimeStep-i)) * StockPrice
        if(TimeStep == NumberOfSteps){ # At expiration
          OptionValue[i+1] = max(0, Type*(S - StrikePrice))
        } else { # Prior to expiration
          B$TimeToMaturity <- TimeToMaturity - dTimeStep*Delta
          B$StockPrice <- S
          LowerBound <- ASBINOptionLowerBound(B)
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          OptionValue[i+1] = max(LowerBound, OptionValue[i+1])
        }
        if(TimeStep == 2){
          if(i == 0){
            OLow <- OptionValue[i+1]
            SLow <- S
          }
          if(i == 1){
            OMid <- OptionValue[i+1]
            SMid <- S
          }
          if(i == 2){
            OHigh <- OptionValue[i+1]
            SHigh <- S
          }
        }
      }
    }
    Value <- ( (OHigh - OMid)/(SHigh - SMid) -
      (OMid - OLow)/(SMid - SLow) ) / (0.5*(SHigh - SLow))
    return(Value)
  })
}
#
# Theta Direct
#
ASBINOptionThetaDirect <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OHigh <- OLow <- SHigh <- SLow <- SMid <- OMid <- 0
    N <- NumberOfSteps + 1
    OptionValue <- c(1:N)
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    Rate <- InterestRate/100.0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Sigma <- Volatility/100.0   # Local variable, in decimal
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    PeriodRate <- exp(Rate*Delta)
    PeriodDriftRate <- exp(DriftRate*Delta)
    # 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))
#
# Thetas corrupted unless S(0) = S(ud): Cox, Ross, Rubinstein
#
    Up <- exp(Sigma*sqrt(Delta))
    Down <- 1/Up
    Prob <- (exp(DriftRate*Delta) - Down) / (Up - Down)
    if(Up < PeriodRate || Down > PeriodRate)return(-99)
    dSteps <- as.numeric(NumberOfSteps)
```

```
    for (TimeStep in NumberOfSteps:0){
      dTimeStep <- as.numeric(TimeStep)
      for (i in 0:TimeStep){
        S <- (Up^i) * (Down^(TimeStep-i)) * StockPrice
        if(TimeStep == NumberOfSteps){  # At expiration
          OptionValue[i+1] = max(0, Type*(S - StrikePrice))
        } else { # Prior to expiration
          B$TimeToMaturity <- TimeToMaturity - TimeStep*Delta
          B$StockPrice <- S
          LowerBound <- ASBINOptionLowerBound(B)
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          OptionValue[i+1] = max(LowerBound, OptionValue[i+1])
        }
        if(TimeStep == 2){
          if(i == 1){
            OMid <- OptionValue[i+1]
            SMid <- S
          }
        }
      }
    }
    Value <- (OMid - OptionValue[1])/(2.0*Delta)
    return(Value)
  })
}
#
# Theta Direct Enhanced
#
ASBINOptionThetaDirectEnh <- function(B){
  with(B, {
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    TimeToMaturity <- TimeToMaturity + 2.0*Delta
    NumberOfSteps <- NumberOfSteps + 2
    OHigh <- OLow <- SHigh <- SLow <- SMid <- OMid <- 0
    N <- NumberOfSteps + 1
    OptionValue <- c(1:N)
    Sum <- 0
    Moneyness <- 0
    Value <- 0
    Rate <- InterestRate/100.0
    DriftRate <- (InterestRate - DividendYield)/100.0
    Sigma <- Volatility/100.0   # Local variable, in decimal
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
    PeriodRate <- exp(Rate*Delta)
    PeriodDriftRate <- exp(DriftRate*Delta)
    # 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))
#
# Thetas corrupted unless S(0) = S(ud): Cox, Ross, Rubinstein
#
    Up <- exp(Sigma*sqrt(Delta))
    Down <- 1/Up
    Prob <- (exp(DriftRate*Delta) - Down) / (Up - Down)
    if(Up < PeriodRate || Down > PeriodRate)return(-99)
    dSteps <- as.numeric(NumberOfSteps)
    for (TimeStep in NumberOfSteps:0){
      for (i in 0:TimeStep){
        S <- (Up^i) * (Down^(TimeStep-i)) * StockPrice
        if(TimeStep == NumberOfSteps){  # At expiration
          OptionValue[i+1] = max(0, Type*(S - StrikePrice))
        } else { # Prior to expiration
```

```
          B$TimeToMaturity <- TimeToMaturity - TimeStep*Delta
          B$StockPrice <- S
          LowerBound <- ASBINOptionLowerBound(B)
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          OptionValue[i+1] = max(LowerBound, OptionValue[i+1])
        }
      if(TimeStep == 4){
        if(i == 2){
          OMid <- OptionValue[i+1]
          SMid <- S
        }
      }
    }
  }
  Value <- (OMid - OptionValue[1])/(4.0*Delta)
  return(Value)
  })
}
#
# Numerical Greeks
#
# Delta
ASBINOptionDelta <- function(B){
  Original <- B$StockPrice
  Change <- (B$GreekIncrement/100.0)*Original
  High <- Original + Change
  B$StockPrice <- High
  OHigh <- ASBINOptionValue(B)
  Low <- Original - Change
  B$StockPrice <- Low
  OLow <- ASBINOptionValue(B)
  B$StockPrice <- Original
  OptionDelta <- (OHigh - OLow)/(High - Low)
  return( OptionDelta )
}
# Gamma: Goes to zero for large time steps (UNSTABLE)
ASBINOptionGamma <- function(B){
  Original <- B$StockPrice
  Change <- (B$GreekIncrement/100.0)*Original*100
  High <- Original + Change
  B$StockPrice <- High
  OHigh <- ASBINOptionValue(B)
  Low <- Original - Change
  B$StockPrice <- Low
  OLow <- ASBINOptionValue(B)
  B$StockPrice <- Original
  OMid <- ASBINOptionValue(B)
  OptionGamma <- ( (OHigh - OMid) - (OMid - OLow) )/(Change*Change)
  # return(OMid - OLow)
  return(OptionGamma)
}
# Theta
ASBINOptionTheta <- function(B){
  Original <- B$TimeToMaturity
  Change <- (B$GreekIncrement/100.0)*Original*(1/10)
  High <- Original + Change
  B$TimeToMaturity <- High
  OHigh <- ASBINOptionValue(B)
  Low <- Original - Change
  B$TimeToMaturity <- Low
  OLow <- ASBINOptionValue(B)
  B$TimeToMaturity <- Original
  OptionTheta <- (OHigh - OLow)/(High - Low)
```

```
  # Moving time to maturity up is maturity time down (-)
  return( -OptionTheta )
}
# Vega
ASBINOptionVega <- function(B){
  Original <- B$Volatility
  Change <- (B$GreekIncrement/100.0)*Original
  High <- Original + Change
  B$Volatility <- High
  OHigh <- ASBINOptionValue(B)
  Low <- Original - Change
  B$Volatility <- Low
  OLow <- ASBINOptionValue(B)
  B$Volatility <- Original
  OptionVega <- (OHigh - OLow)/(High - Low)
  return( OptionVega )
}
# Rho
ASBINOptionRho <- function(B){
  Original <- B$InterestRate
  Change <- (B$GreekIncrement/100.0)*Original
  High <- Original + Change
  B$InterestRate <- High
  OHigh <- ASBINOptionValue(B)
  Low <- Original - Change
  B$InterestRate <- Low
  OLow <- ASBINOptionValue(B)
  B$InterestRate <- Original
  OptionRho <- (OHigh - OLow)/(High - Low)
  return( OptionRho )
}
```