Module 8.2: SRM ABM-Based Binomial Models
R Commentary

See module *Ch 8.2 Valuation ABM Binomial OVM*. The main test program is *SRM ABM Binomial OVM Test.R*. Technical functions are contained in *ESABMBINOVM Backward Recursion With SRM Functions.R* and *ASABMBINOVM Backward Recursion With SRM Function.R*.

Valuation ABM Binomial OVM European-Style Test.R

This program examines a particular set of data for one option. We specifically examine the ABM case that calibrates to the GBM case of 30% volatility.

```
source('ESABMBINOVM Backward Recursion Function.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 = 29.88476829    # In dollars, annualized
inputTimeToMaturity = 1.0        # In fraction of year
inputType = 1L                   # 1 for call, -1 for put
inputNumberOfSteps = as.integer(250)
inputPayoutType = 1L             # 1 Plain vanilla, 2 digital
inputEMMProbability = 50.0       # In percent
inputDigitalPayout = 100.0
```

The analysis is provided in *ESABMBINOVM Backward Recursion With SRM Function.R*. The key to understanding this code is the outer loop runs backward and the option values are stored in the same vector. As it loops backward, one value in the vector is no longer referenced. The final answer is in the first element of the vector. The complete valuation method is reproduced here. The Greeks follow this code closely.

```
# European-style binomial option valuation function (ABM)
ABMESOptionValue = 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) )
          IVPut <- max(0, StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          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)
  })
}
```

ASABMBINOVM Backward Recursion Function.R

The complete file is reproduced here.

```
# ASABMBINOVM Backward Recursion Function.R
#
# American-style binomial option valuation function (ABM)
source('ESABMBINOVM Backward Recursion With SRM Functions.R')
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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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)
  })
}
# Lower bound
ASOptionLowerBound = function(B){
```

```
  with(B, {
    LowerBound = Type * StockPrice * PV1(TimeToMaturity, DividendYield) -
      Type * StrikePrice * PV1(TimeToMaturity, InterestRate)
    IntrinsicValue = max(0, Type * (StockPrice - StrikePrice))
    LowerBound = max(0, LowerBound, IntrinsicValue)
    return( LowerBound )
  })
}
# Upper bound
ASOptionUpperBound = function(B){
  with(B, {
    if(Type == 1)UpperBound = StockPrice
    if(Type == -1)UpperBound = StrikePrice
    return( UpperBound )
  })
}
#
# Delta Direct -- have to do full backward recursion with ABM
#
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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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
#
ABMASBINOptionDeltaDirectEnh <- function(B){
  with(B, {
    OriginalTimeToMaturity <- TimeToMaturity
    OriginalStockPrice <- StockPrice
    OriginalNumberOfSteps <- NumberOfSteps
    Delta <- TimeToMaturity / as.numeric(NumberOfSteps)
```

```
    B$TimeToMaturity <- TimeToMaturity + 2.0*Delta
    B$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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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){
          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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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){
          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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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){
          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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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 - CallValue[1])/(2.0*Delta)
    DPV <- (OMidDigitalPutValue - PutValue[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 early exercise or lower boundary violation
          IVCall <- max( 0, S - StrikePrice,
            S*exp(-(DividendYield/100)*TTM) -
            StrikePrice*exp(-(InterestRate/100)*TTM) )
          IVPut <- max(0, StrikePrice - S,
            StrikePrice*exp(-(InterestRate/100)*TTM) -
            S*exp(-(DividendYield/100)*TTM) )
          IVDCall <- 0
          IVDPut <- 0
          if(S > StrikePrice){
            IVDCall <- DigitalPayout
            IVDPut <- 0
          }
          if (S < StrikePrice){
            IVDCall <- 0
            IVDPut <- DigitalPayout
          }
          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 - CallValue[1])/(4.0*Delta)
    DPV <- (OMidDigitalPutValue - PutValue[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 )
}
```