Module 5.2: Geometric Brownian Motion-Based Binomial Models
R Commentary

See module *Ch 5.2 Valuation GBM Binomial OVM*. There are two main test programs, *Valuation GBM Binomial OVM European-Style Test.R* and *Valuation GBM Binomial OVM American-Style Test.R*. Technical functions are contained in *ESGBMBINOVM.R* and *ASGBMBINOVM.R*.

Valuation GBM Binomial OVM European-Style Test.R (Selected Excerpts and Output)
The functions used for valuing European-style options with the binomial model that converges to the terminal lognormal distribution consistent with GBM is sourced first. Note below the dividend yield is set to zero and the lattice type is the coherent model.

```
source('ESGBMBINOVM 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
inputNumberOfSteps = as.integer(500)   # Or use L: 1000L
inputPayoutType = 1L             # 1 Plain vanilla, 2 digital
inputEMMProbability = 50.0       # In percent
# Lattice Type: 0 - Cox, Ross, Rubenstein, 1 - Coherent, eventually add others
inputLatticeType = 1L
inputDigitalPayout = 100.0
```

This code produces several figures for European-style options discussed earlier.

Valuation GBM Binomial OVM American-Style Test.R (Selected Excerpts and Output)
The functions used for valuing American-style options with the binomial model that converges to the terminal lognormal distribution consistent with GBM is sourced first.

```
source('ASGBMBINOVM Functions.R')
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
inputNumberOfSteps = as.integer(500)   # Or use L: 1000L
inputPayoutType = 1L             # 1 Plain vanilla, 2 digital
inputEMMProbability = 50.0       # In percent
inputDigitalPayout = 100.0
```

The function to compute option values for European-style options is repeated below.

```
# ASGBMBINOVM Functions.R
#  American-style, geometric Brownian motion, binomial OVM
source('ESGBMBINOVM Functions.R')
# American-style lower bound
ASBINOptionLowerBound <- function(B){
  with(B, {
    LowerBound <- Type * (StockPrice * PV1(TimeToMaturity, DividendYield) -
      StrikePrice * PV1(TimeToMaturity, InterestRate) )
    IntrinsicValue = max(0, Type*(StockPrice - StrikePrice))
    Value = max(0, LowerBound, IntrinsicValue)
    return( Value )
  })
}
# American-style upper bound (Same as ES)
ASBINOptionUpperBound <- function(B){
  with(B, {
    if(Type == 1){
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
      UpperBound <- StockPrice
    }
    if(Type == -1){
      UpperBound <- StrikePrice
    }
    return( UpperBound )
  })
}
# Option valuation -- Binomial American-style
ASBINOptionValue = function(B){
  with(B, {
#
#     B <- BINInputData
#     attach(B)
#
    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)
    # N = NumberOfSteps
    OptionValue <- numeric(NumberOfSteps + 1)
# Lattice structure: Up, Down, and Prob
    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))
# Test that d<PeriodicRate<u otherwise quit
    if(Up < PeriodRate || Down > PeriodRate)return(-99)
# AS: Backward induction
    for (TimeStep in NumberOfSteps:0){
# Inner loop
      for (i in 0:TimeStep){
        dStateStep <- as.numeric(i)
        dTimeStep <- as.numeric(TimeStep)
        if(TimeStep == NumberOfSteps){  # At expiration
          if( (Type == 1) && (PayoutType == 1) ){    # Plain vanilla call
            Moneyness <- (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) *
              StockPrice - StrikePrice
          }
          if( (Type == 1) && (PayoutType == 2) ){    # Digital call
            Moneyness <- (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) *
              StockPrice - StrikePrice
            if(Moneyness > 0.0) Moneyness <- DigitalPayout
          }
          if( (Type == -1) && (PayoutType == 1) ){   # Plain vanilla put
            Moneyness <- StrikePrice -
              (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) * StockPrice
          }
          if( (Type == -1) && (PayoutType == 2) ){  # Digital put
            Moneyness  <- StrikePrice -
              (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) * StockPrice
            if(Moneyness > 0.0) Moneyness <- DigitalPayout
          }
          OptionValue[i+1] <- max(0, Moneyness)
        } else { # AS prior to expiration
          OptionValue[i+1] = (1.0/PeriodRate) * (Prob*OptionValue[i+2] +
            (1.0 - Prob)*OptionValue[i+1])
          if((Type == 1) && (PayoutType == 1)){    # Plain vanilla call
            MarketStockPrice = (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) *
```


© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
            StockPrice
          Moneyness = max(0, MarketStockPrice - StrikePrice)
          OptionValue[i+1] = max(Moneyness, OptionValue[i+1])
        }
        if( (Type == 1) && (PayoutType == 2) ){    # Digital call
          MarketStockPrice = (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) *
            StockPrice
          Moneyness = MarketStockPrice - StrikePrice
          if(Moneyness > 0.0) Moneyness <- DigitalPayout
          OptionValue[i+1] = max(Moneyness, OptionValue[i+1])
        }
        if((Type == -1) && (PayoutType == 1)){    # Plain vanilla put
          MarketStockPrice = (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) *
            StockPrice
          Moneyness = max(0, StrikePrice - MarketStockPrice)
          OptionValue[i+1] = max(Moneyness, OptionValue[i+1])
        }
        if( (Type == -1) && (PayoutType == 2) ){    # Digital put
          MarketStockPrice = (Up^dStateStep) * (Down^(dTimeStep-dStateStep)) *
            StockPrice
          Moneyness = StrikePrice - MarketStockPrice
          if(Moneyness > 0.0) Moneyness <- DigitalPayout
          OptionValue[i+1] = max(Moneyness, OptionValue[i+1])
        }
      }
    }
    Value = OptionValue[1]
  }
# Check lower boundary conditions
  LowerBound = Type * ( StockPrice * PV1(TimeToMaturity, DividendYield) -
    StrikePrice * PV1(TimeToMaturity, InterestRate) )
  LowerBound = max(0, LowerBound)
  IntrinsicValue = max(0, Type*(StockPrice - StrikePrice))
  Value = max(Value, LowerBound, IntrinsicValue)
  return(Value)
 })
}
```

This code produced the American-style results discussed earlier.

In the R code folder are two additional R programs that produce similar graphs except the strike price is varied and the X axis is the normalized strike price (= Strike Price/Stock price). Both axes are expressed in percent.

© 2023 Robert Brooks. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.