Package ‘fOptions’

November 16, 2017

Title Rmetrics - Pricing and Evaluating Basic Options
Date 2017-11-12
Version 3042.86
Author Diethelm Wuertz [aut],
Tobias Setz [cre],
Yohan Chalabi [ctb]
Maintainer Tobias Setz <tobias.setz@live.com>
Description Provides a collection of functions to
valuate basic options. This includes the generalized
Black-Scholes option, options on futures and options on
commodity futures.
Depends timeDate, timeSeries, fBasics
Imports graphics, methods, stats
Suggests RUnit, tcltk
LazyData yes
License GPL (>= 2)
URL http://www.rmetrics.org
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-11-16 22:46:13 UTC

R topics documented:

fOptions-package .................................................. 2
BasicAmericanOptions ........................................... 4
BinomialTreeOptions ............................................. 7
HestonNandiGarchFit ............................................ 10
HestonNandiOptions ............................................. 13
LowDiscrepancy .................................................. 15
MonteCarloOptions .............................................. 18
PlainVanillaOptions ............................................. 22
Description

The Rmetrics "Options" package is a collection of functions to valuate basic options.

Details

Package:    fOptions
Type:       Package
Version:    R 3.0.1
Date:       2014
License:    GPL Version 2 or later
Copyright:  (c) 1999-2014 Rmetrics Association
URL:        https://www.rmetrics.org

1 Introduction

The fOptions package provides function for pricing and evaluating basic options.

2 Plain Vanilla Option

This section provides a collection of functions to valuate plain vanilla options. Included are functions for the Generalized Black-Scholes option pricing model, for options on futures, some utility functions, and print and summary methods for options.

GBS*          the generalized Black-Scholes option
BlackScholesOption a synonyme for the GBSOption
Black76Option  options on Futures
MiltersenSchwartzOption options on commodity futures

NDF, CND, CBND  distribution functions

print           print method for Options
summary         summary method for Options
3 Binomial Tree Options

This section offers a collection of functions to valuate options in the framework of the Binomial tree option approach.

- CRRBinomialTreeOption CRR Binomial Tree Option
- JRBinomialTreeOption JR Binomial Tree Option
- TIANBinomialTreeOption TIAN Binomial Tree Option
- BinomialTreeOption Binomial Tree Option
- BinomialTreePlot Binomial Tree Plot

4 Monte Carlo Options

In this section we provide functions to valuate options by Monte Carlo methods. The functions include beside the main Monte Carlo Simulator, example functions to generate Monte Carlo price paths and to compute Monte Carlo price payoffs.

- sobolInnovations Example for scrambled Sobol innovations
- wienerPath Example for a Wiener price path
- plainVanillaPayoff Example for the plain vanilla option's payoff
- arithmeticAsianPayoff Example for the arithmetic Asian option's payoff
- MonteCarloOption Monte Carlo Simulator for options

5 Low Discrepancy Sequences

This section provides three types of random number generators for uniform and normal distributed deviates. These are pseudo random number generator and a halton and sobol generator for low discrepancy sequences.

- runif.pseudo Uniform pseudo random numbers
- rnorm.pseudo Normal pseudo random numbers
- runif.halton Uniform Halton sequence
- rnorm.halton Normal Halton sequence
- runif.sobol Uniform scrambled Sobol sequence
- rnorm.sobol Normal scrambled Sobol sequence
6 Heston Nandi Garch Fit

Here we provide functions to model the GARCH(1,1) price paths which underly Heston and Nandi’s option pricing model. The functions are:

- `hngarchSim` simulates a Heston-Nandi Garch(1,1) process
- `hngarchFit` fits parameters of a Heston Nandi Garch(1,1) model
- `hngarchStats` returns true moments of the log-Return distribution
- `print.hngarch` print method, \cr
- `summary.hngarch` diagnostic summary

7 Heston Nandi Garch Options

This section comes with functions to valuate Heston-Nandi options. Provided are functions to compute the option price and the delta and gamma sensitivities for call and put options.

- `HNGOption` Heston-Nandi GARCH(1,1) option price
- `HNGGreeks` Heston-Nandi GARCH(1,1) option sensitivities
- `HNGCharacteristics` combines option prices and sensitivities

About Rmetrics

The fOptions Rmetrics package is written for educational support in teaching “Computational Finance and Financial Engineering” and licensed under the GPL.

---

**BasicAmericanOptions**

**Valuation of Basic American Options**

**Description**

A collection and description of functions to valuate basic American options. Approximative formulas for American calls are given for the Roll, Geske and Whaley Approximation, for the Barone-Adesi and Whaley Approximation, and for the Bjerksund and Stensland Approximation.

The functions are:

- `RollGeskeWhaleyOption` Roll, Geske and Whaley Approximation,
- `BAWAmericanApproxOption` Barone-Adesi and Whaley Approximation,
- `BSAmericanApproxOption` Bjerksund and Stensland Approximation.
Usage

RollGeskeWhaleyOption(S, X, time1, Time2, r, D, sigma,
               title = NULL, description = NULL)
BAWAmericanApproxOption(TypeFlag, S, X, Time, r, b, sigma,
                       title = NULL, description = NULL)
BSAmericanApproxOption(TypeFlag, S, X, Time, r, b, sigma,
                       title = NULL, description = NULL)

Arguments

b                        the annualized cost-of-carry rate, a numeric value; e.g. 0.1 means 10% pa.
D                        a single dividend with time to dividend payout t1.
description              a character string which allows for a brief description.
r                        the annualized rate of interest, a numeric value; e.g. 0.25 means 25% pa.
S                        the asset price, a numeric value.
sigma                    the annualized volatility of the underlying security, a numeric value; e.g. 0.3
                        means 30% volatility pa.
Time                     the time to maturity measured in years, a numeric value.
time1, Time2             [RollGeskeWhaley*] - the first value measures time to dividend payout in years,
                        e.g. 0.25 denotes a quarter, and the second value measures time to maturity
                        measured in years, a numeric value; e.g. 0.5 means 6 months.
title                    a character string which allows for a project title.
TypeFlag                  a character string either "c" for a call option or a "p" for a put option.
X                        the exercise price, a numeric value.

Details

Roll-Geske-Whaley Option:

The function RollGeskeWhaleyOption valuates American calls on a stock paying a single dividend
with specified time to dividend payout according to the pricing formula derived by Roll, Geske and

Approximations for American Options:

The function BSAmericanApproxOption valuates American calls or puts on an underlying asset for
a given cost-of-carry rate according to the quadratic approximation method due to Barone-Adesi
and Whaley (1987). The function BSAmericanApproxOption valuates American calls or puts on
stocks, futures, and currencies due to the approximation method of Bjerksund and Stensland (1993).

Value

RollGeskeWhaleyOption
BAWAmericanApproxOption
return the option price, a numeric value.

\texttt{BSAmericanApproxOption}
returns a list with the following two elements: Premium the option price, and \texttt{TriggerPrice} the trigger price.

\textbf{Note}

The functions implement the algorithms to valuate basic American options as described in Chapter 1.4 of Haug’s Option Guide (1997).

\textbf{Author(s)}

Diethelm Wuertz for the Rmetrics R-port.

\textbf{References}


Geske R. (1979); \textit{A Note on an Analytical Formula for Unprotected American Call Options on Stocks with known Dividends}, Journal of Financial Economics 7, 63–81.


\textbf{Examples}

\begin{verbatim}
## All the examples are from Haug's Option Guide (1997)

## CHAPTER 1.4: ANALYTICAL MODELS FOR AMERICAN OPTIONS

## Roll-Geske-Whaley American Calls on Dividend Paying Stocks [Haug 1.4.1]
RollGeskeWhaleyOption(S = 80, X = 82, time1 = 1/4,  
Time2 = 1/3, r = 0.06, D = 4, sigma = 0.30)

## Barone-Adesi and Whaley Approximation for American Options  
# Options [Haug 1.4.2] vs. Black76 Option on Futures:
BWAmericanApproxOption(TypeFlag = "p", S = 100,  
X = 100, Time = 0.5, r = 0.10, b = 0, sigma = 0.25)
Black76Option(TypeFlag = "c", FT = 100, X = 100,  
Time = 0.5, r = 0.10, sigma = 0.25)

## Bjerksund and Stensland Approximation for American Options:
BSAmericanApproxOption(TypeFlag = "c", S = 42, X = 40,  
Time = 0.75, r = 0.04, b = 0.04–0.08, sigma = 0.35)
\end{verbatim}
Binomial Tree Option Model

Description

A collection and description of functions to valuate options in the framework of the Binomial tree option approach.

The functions are:

- **CRRBinomialTreeOption**: CRR Binomial Tree Option,
- **JRBinomialTreeOption**: JR Binomial Tree Option,
- **TIANBinomialTreeOption**: TIAN Binomial Tree Option,
- **BinomialTreeOption**: Binomial Tree Option,
- **BinomialTreePlot**: Binomial Tree Plot.

Usage

- **CRRBinomialTreeOption**(`TypeFlag = c("ce", "pe", "ca", "pa"), S, X, Time, r, b, sigma, n, title = NULL, description = NULL`)
- **JRBinomialTreeOption**(`TypeFlag = c("ce", "pe", "ca", "pa"), S, X, Time, r, b, sigma, n, title = NULL, description = NULL`)
- **TIANBinomialTreeOption**(`TypeFlag = c("ce", "pe", "ca", "pa"), S, X, Time, r, b, sigma, n, title = NULL, description = NULL`)
- **BinomialTreeOption**(`TypeFlag = c("ce", "pe", "ca", "pa"), S, X, Time, r, b, sigma, n, title = NULL, description = NULL`)
- **BinomialTreePlot**(`BinomialTreeValues, dx = -0.025, dy = 0.4, cex = 1, digits = 2, ...`)

Arguments

- **b**: the annualized cost-of-carry rate, a numeric value; e.g. 0.1 means 10% pa.
- **binomialtreevalues**: the return value from the BinomialTreeOption function.
- **cex**: a numerical value giving the amount by which the plotting text and symbols should be scaled relative to the default.
- **description**: a character string which allows for a brief description.
- **digits**: an integer value, how many digits should be displayed in the option tree?
- **dx**, **dy**: numerical values, an offset fine tuning for the placement of the option values in the option tree.
- **n**: number of time steps; an integer value.
- **r**: the annualized rate of interest, a numeric value; e.g. 0.25 means 25% pa.
- **S**: the asset price, a numeric value.
**Details**

**CRR Binomial Tree Model:**

Binomial models were first suggested by Cox, Ross and Rubinstein (1979), CRR, and then became widely used because of its intuition and easy implementation. Binomial trees are constructed on a discrete-time lattice. With the time between two trading events shrinking to zero, the evolution of the price converges weakly to a lognormal diffusion. Within this mode the European options value converges to the value given by the Black-Scholes formula.

**JR Binomial Tree Model:**

There exist many extensions of the CRR model. Jarrow and Rudd (1983), JR, adjusted the CRR model to account for the local drift term. They constructed a binomial model where the first two moments of the discrete and continuous time return processes match. As a consequence a probability measure equal to one half results. Therefore the CRR and JR models are sometimes attributed as equal jumps binomial trees and equal probabilities binomial trees.

**TIAN Binomial Tree Model:**

Tian (1993) suggested to match discrete and continuous local moments up to third order. Leisen and Reimer (1996) proved that the order of convergence in pricing European options for all three methods is equal to one, and thus the three models are equivalent.

**Value**

The option price, a numeric value.

**Note**

Note, the `BinomialTree` and `BinomialTreePlot` are preliminary implementations.

**Author(s)**

Diethelm Wuertz for the Rmetrics R-port.
BinomialTreeOptions

References


Examples

```c
## Cox–Ross–Rubinstein Binomial Tree Option Model:
# Example 14.1 from Hull’s Book:
CRRBinomialTreeOption(TypeFlag = "pa", S = 50, X = 50,
                    Time = 5/12, r = 0.1, b = 0.1, sigma = 0.4, n = 5)
# Example 3.1.1 from Haug’s Book:
CRRBinomialTreeOption(TypeFlag = "pa", S = 100, X = 95,
                    Time = 0.5, r = 0.08, b = 0.08, sigma = 0.3, n = 5)
# A European Call – Compare with Black Scholes:
CRRBinomialTreeOption(TypeFlag = "ce", S = 100, X = 100,
                    Time = 1, r = 0.1, b = 0.1, sigma = 0.25, n = 50)
GBSOption(TypeFlag = "c", S = 100, X = 100,
                    Time = 1, r = 0.1, b = 0.1, sigma = 0.25)@price

## CRR – JR – TIAN Model Comparison:
# Hull’s Example as Function of "n":
par(mfrow = c(2, 1), cex = 0.7)
steps = 50
CRROptionValue = JROptionValue = TIANOptionValue = rep(NA, times = steps)
for (n in 3:steps) {
    CRROptionValue[n] = CRRBinomialTreeOption(TypeFlag = "pa", S = 50,
                                               X = 50, Time = 0.4167, r = 0.1, b = 0.1, sigma = 0.4, n = n)@price
    JROptionValue[n] = JRBinomialTreeOption(TypeFlag = "pa", S = 50,
                                             X = 50, Time = 0.4167, r = 0.1, b = 0.1, sigma = 0.4, n = n)@price
    TIANOptionValue[n] = TIANBinomialTreeOption(TypeFlag = "pa", S = 50,
                                                X = 50, Time = 0.4167, r = 0.1, b = 0.1, sigma = 0.4, n = n)@price
}
plot(CRROptionValue[3:steps], type = "l", col = "red", ylab = "Option Value")
lines(JROptionValue[3:steps], col = "green")
lines(TIANOptionValue[3:steps], col = "blue")
# Add Result from BAW Approximation:
BAWValue = BAWAmericanApproxOption(TypeFlag = "p", S = 50, X = 50,
                                        Time = 0.4167, r = 0.1, b = 0.1, sigma = 0.4)@price
abline(h = BAWValue, lty = 3)
```
HestonNandiGarchFit

Heston-Nandi Garch(1,1) Modelling

Description

A collection and description of functions to model the GARCH(1,1) price paths which underly Heston and Nandi’s option pricing model.

The functions are:

- `hngarchSim` Simulates a Heston-Nandi Garch(1,1) process,
- `hngarchFit` MLE for a Heston Nandi Garch(1,1) model,
- `hngarchStats` True moments of the log-Return distribution,
- `print.hngarch` Print method,
- `summary.hngarch` Diagnostic summary.

Usage

```r
hngarchSim(model, n, innov, n.start, start.innov, rand.gen, ...)
```

```r
hngarchFit(x, model = list(lambda = -0.5, omega = var(x), alpha = 0.1 * var(x), beta = 0.1, gamma = 0, rf = 0), symmetric = TRUE, trace = FALSE, title = NULL, description = NULL, ...)
```

```r
hngarchStats(model)
```

```r
## S3 method for class 'hngarch'
print(x, ...)
```

## S3 method for class 'hngarch'
summary(object, ...)

Arguments

- `description` a brief description of the project of type character.
- `innov` [hngarchSim] - is a univariate time series or vector of innovations to produce the series. If not provided, innov will be generated using the random number generator specified.
by rand.gen. Missing values are not allowed. By default the normal random
number generator will be used.

model

a list of GARCH model parameters with the following entries: lambda, omega,
the constant coefficient of the variance equation, alpha the autoregressive coef-
ficient, beta the variance coefficient, gamma the asymmetry coefficient, and rf,
the risk free rate, numeric values.

n

[hngarchSim] -
is the length of the series to be simulated. The default value is 1000.

n.start

[garchSim] -
gives the number of start-up values to be discarded. The default value is 100.

object

[summary] -
a fitted HN-GARCH(1,1) time series object of class "hngarch" as returned from
the function hngarchFit.

rand.gen

[hngarchSim] -
is the function which is called to generate the innovations. Usually, rand.gen
will be a random number generator. Additional arguments required by the ran-
dom number generator rand.gen, usually the location, scale and/or shape pa-
rameter of the underlying distribution function, have to be passed through the
dots argument.

start.innov

[hngarchSim] -
is a univariate time series or vector of innovations to be used as start up values.
Missing values are not allowed.

symmetric

[hngarchFit] -
a logical, if TRUE a symmetric model is estimated, otherwise the parameters
are estimated for an asymmetric HN Garch(1,1) model.

title

a character string which allows for a project title.

trace

[hngarchFit] -
a logical value. Should the optimizarion be traced? If trace=FALSE, no tracing
is done of the iteration path.

x

[hngarchFit] -
an univariate vector or time series.
[print] -
a fitted HN-GARCH(1,1) time series object of class "hngarch" as returned from
the function hngarchFit.

additional arguments to be passed.

Details

Path Simulation:

The function hngarchSim simulates a Heston-Nandi Garch(1,1) process with structure parameters
specified through the list model(lambda, omega, alpha, beta, gamma, rf).

Parameter Estimation:
The function hngarchFit estimates by the maximum log-likelihood approach the parameters either for a symmetric or an asymmetric Heston-Nandi Garch(1,1) model from the log returns $x$ of a financial time series. For optimization R’s optim function is used. Additional optimization parameters may be passed through the ... argument.

**Diagnostic Analysis:**

The function summary.hngarch performs a diagnostic analysis and recalculates conditional variances and innovations from the time series.

**Calculation of Moments:**

The function hngarchStats calculates the first four true moments of the unconditional log return distribution for a stationary Heston-Nandi Garch(1,1) process with standard normally distributed innovations. In addition the persistence and the expectation values of sigma to the power 2, 4, 6, and 8 can be accessed.

**Value**

hngarchSim
returns numeric vector with the simulated time series points neglecting those from the first start.innov innovations.

hngarchFit
returns list with two entries: The estimated model parameters model, where model is a list of the parameters itself, and l1h the value of the log likelihood.

hngarchStats
returns a list with the following components: persistence, the value of the persistence, meansigma2, meansigma4, meansigma6, meansigma8, the expectation value of sigma to the power of 2, 4, 6, and 8, mean, variance, skewness, kurtosis, the mean, variance, skewness and kurtosis of the log returns.

summary.hngarch
returns list with the following elements: $h$, a numeric vector with the conditional variances, $z$, a numeric vector with the innovations.

**Author(s)**

Diethelm Wuertz for the Rmetrics R-port.

**References**

Examples

```r
## hngarchSim -
# Simulate a Heston Nandi Garch(1,1) Process:
# Symmetric Model - Parameters:
model = list(lambda = 4, omega = 8e-5, alpha = 6e-5,
            beta = 0.7, gamma = 0, rf = 0)
ts = hngarchSim(model = model, n = 500, n.start = 100)
par(mfrow = c(2, 1), cex = 0.75)
ts.plot(ts, col = "steelblue", main = "HN Garch Symmetric Model")
grid()

## hngarchFit -
# HN-GARCH log likelihood Parameter Estimation:
# To speed up, we start with the simulated model ...
mle = hngarchFit(model = model, x = ts, symmetric = TRUE)
mle

## summary.hngarch -
# HN-GARCH Diagnostic Analysis:
par(mfrow = c(3, 1), cex = 0.75)
summary(mle)

## hngarchStats -
# HN-GARCH Moments:
hngarchStats(mle$model)
```

---

**HestonNandiOptions**

**Option Price for the Heston-Nandi Garch Option Model**

**Description**

A collection and description of functions to valuate Heston-Nandi options. Included are functions to compute the option price and the delta and gamma sensitivities for call and put options.

The functions are:

- `hngoption`  
  Heston-Nandi GARCH(1,1) option price.
- `hnggreeks`  
  Heston-Nandi GARCH(1,1) option sensitivities.
- `hngcharacteristics`  
  Option prices and sensitivities.

**Usage**

- `hngoption(TypeFlag, model, S, X, Time.inDays, r.daily)`
- `hnggreeks(Selection, TypeFlag, model, S, X, Time.inDays, r.daily)`
- `hngcharacteristics(TypeFlag, model, S, X, Time.inDays, r.daily)`
Arguments

model a list of model parameters with the following entries: lambda, omega, alpha, beta, and gamma, numeric values.
r.daily the daily rate of interest, a numeric value; e.g. 0.25/252 means about 0.001% per day.
S the asset price, a numeric value.
Selection sensitivity to be computed, one of "delta", "gamma", "vega", "theta", "rho", or "coC", a string value.
Time.inDays the time to maturity measured in days, a numerical value; e.g. 5/252 means 1 business week.
TypeFlag a character string either "c" for a call option or a "p" for a put option.
X the exercise price, a numeric value.

Details

Option Values:

HNGOption calculates the option price, HNGGreeks allows to compute the option sensitivity Delta or Gamma, and HNGcharacteristics summarizes both in one function call.

Value

HNGOption
returns a list object of class "option" with $price denoting the option price, a numeric value, and $call a character string which matches the function call.

HNGOGreeks
returns the option sensitivity for the selected Greek, either "delta" or "gamma"; a numeric value.

HNGCharacteristics
returns a list with the following entries:

premium the option price, a numeric value.
delta the delta sensitivity, a numeric value.
gamma the gamma sensitivity, a numeric value.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

Heston S.L., Nandi S. (1997); A Closed-Form GARCH Option Pricing Model, Federal Reserve Bank of Atlanta.
Examples

```r
## model -
# Define the Model Parameters for a Heston-Nandi Option:
model = list(lambda = -0.5, omega = 2.3e-6, alpha = 2.9e-6,
             beta = 0.85, gamma = 184.25)
S = X = 100
Time.inDays = 252
r.daily = 0.05 / Time.inDays
sigma.daily = sqrt((model$omega + model$alpha) /
                   (1 - model$beta - model$alpha * model$gamma^2))
data.frame(S, X, r.daily, sigma.daily)

## HGOption -
# Compute HNG Call-Put and compare with GBS Call-Put:
HNG = GBS = Diff = NULL
for (TypeFlag in c("c", "p")) {
  HNG = c(HNG, HGOption(TypeFlag, model = model, S = S, X = X,
                   Time.inDays = Time.inDays, r.daily = r.daily)@price )
  GBS = c(GBS, GBSSOption(TypeFlag, S = S, X = X, Time = Time.inDays,
                          r = r.daily, b = r.daily, sigma = sigma.daily)@price )
Options = cbind(HNG, GBS, Diff = round(100*(HNG-GBS)/GBS, digits=2))
row.names(Options) <- c("Call", "Put")
data.frame(Options)

## HGNGreeks -
# Compute HNG Greeks and compare with GBS Greeks:
Selection = c("Delta", "Gamma")
HNG = GBS = NULL
for (i in 1:2){
  HNG = c(HNG, HGNGreeks(Selection[i], TypeFlag = "c", model = model,
                        S = 100, X = 100, Time = Time.inDays, r = r.daily )
  GBS = c(GBS, GBSSGreeks(Selection[i], TypeFlag = "c", S = 100, X = 100,
                        Time = Time.inDays, r = r.daily, b = r.daily, sigma = sigma.daily) )
Greeks = cbind(HNG, GBS, Diff = round(100*(HNG-GBS)/GBS, digits = 2))
row.names(Greeks) <- Selection
data.frame(Greeks)
```

Description

A collection and description of functions to compute Halton’s and Sobol’s low discrepancy sequences, distributed in form of a uniform or normal distribution.

The functions are:

- `runif.halton`: Uniform Halton sequence,
- `rnorm.halton`: Normal Halton sequence,
runif.sobol  Uniform scrambled Sobol sequence,
rnorm.sobol  Normal scrambled Sobol sequence,
runif.pseudo Uniform pseudo random numbers,
norma.pseudo Normal pseudo random numbers.

Usage

runif.halton(n, dimension, init)
rnorm.halton(n, dimension, init)
runif.sobol(n, dimension, init, scrambling, seed)
rnorm.sobol(n, dimension, init, scrambling, seed)
runif.pseudo(n, dimension, init)
rnorm.pseudo(n, dimension, init)

Arguments

dimension  an integer value, the dimension of the sequence. The maximum value for the Sobol generator is 1111.
init  a logical, if TRUE the sequence is initialized and restarts, otherwise not. By default TRUE.
n  an integer value, the number of random deviates.
scrambling  an integer value, if 1, 2 or 3 the sequence is scrambled otherwise not. If 1, Owen type type of scrambling is applied, if 2, Faure-Tezuka type of scrambling, is applied, and if 3, both Owen+Faure-Tezuka type of scrambling is applied. By default 0.
seed  an integer value, the random seed for initialization of the scrambling process. By default 4711. On effective if scrambling>0.

Details

Halton’s Low Discrepancy Sequences:
Calculates a matrix of uniform or normal deviated halton low discrepancy numbers.

Scrambled Sobol’s Low Discrepancy Sequences:
Calculates a matrix of uniform and normal deviated Sobol low discrepancy numbers. Optional scrambling of the sequence can be selected.

Pseudo Random Number Sequence:
Calculates a matrix of uniform or normal distributed pseudo random numbers. This is a helpful function for comparing investigations obtained from a low discrepancy series with those from a pseudo random number.
All generators return a numeric matrix of size \( n \) by dimension.

The global variables `runif.halton.seed` and `runif.sobol.seed` save the status to restart the generators. Note, that only one instance of a generator can be run at the same time.

The ACM Algorithm 659 implemented to generate scrambled Sobol sequences is under the License of the ACM restricted for academic and noncommercial usage. Please consult the ACM License agreement included in the doc directory.

P. Bratley and B.L. Fox for the Fortran Sobol Algorithm 659, S. Joe for the Fortran extension to 1111 dimensions, Diethelm Wuertz for the Rmetrics R-port.


Joe S., Kuo F.Y. (1998); *Remark on Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator*.

## Examples

```r
## *.halton -
par(mfrow = c(2, 2), cex = 0.75)
runif.halton(n = 10, dimension = 5)
hist(runif.halton(n = 5000, dimension = 1), main = "Uniform Halton",
     xlab = "x", col = "steelblue3", border = "white")
runif.halton(n = 10, dimension = 5)
hist(runif.halton(n = 5000, dimension = 1), main = "Normal Halton",
     xlab = "x", col = "steelblue3", border = "white")
```

```r
## *.sobol -
runif.sobol(n = 10, dimension = 5, scrambling = 3)
hist(runif.sobol(5000, 1, scrambling = 2), main = "Uniform Sobol",
      xlab = "x", col = "steelblue3", border = "white")
runif.sobol(n = 10, dimension = 5, scrambling = 3)
hist(runif.sobol(5000, 1, scrambling = 2), main = "Normal Sobol",
      xlab = "x", col = "steelblue3", border = "white")
```

```r
## *.pseudo -
runif.pseudo(n = 10, dimension = 5)
rnorm.pseudo(n = 10, dimension = 5)
```
MonteCarloOptions

Monte Carlo Valuation of Options

Description

A collection and description of functions to valuate options by Monte Carlo methods. The functions include beside the main Monte Carlo Simulator, example functions to generate Monte Carlo price paths and to compute Monte Carlo price payoffs.

The functions are:

- `sobolInnovations`: Example for scrambled Sobol innovations,
- `wienerPath`: Example for a Wiener price path,
- `plainVanillaPayoff`: Example for the plain vanilla option’s payoff,
- `arithmeticAsianPayoff`: Example for the arithmetic Asian option’s payoff,
- `montecarlooption`: Monte Carlo Simulator for options.

Usage

MonteCarloOption(delta.t, pathLength, mcSteps, mcLoops, init = TRUE, innovations.gen, path.gen, payoff.calc, antithetic = TRUE, standardization = FALSE, trace = TRUE, ...)

Arguments

- `antithetic`: a logical flag, should antithetic variates be used? By default TRUE.
- `delta.t`: the time step interval measured as a fraction of one year, by default one day, i.e. `delta.t = 1/360`.
- `init`: a logical flag, should the random number generator be initialized? By default TRUE.
- `innovations.gen`: a user defined function to generate the innovations, this can be the normal random number generator `rnorm.pseudo` with mean zero and variance one. For the usage of low discrepancy sequences alternatively `rnorm.halton` and `rnorm.sobol` can be called. The generator must deliver a normalized matrix of innovations with dimension given by the number of Monte Carlo steps and the path length. The first three arguments of the generator are the number of Monte Carlo steps and path length. The initialization flag `init`. Optional arguments can be passed through the argument ..., e.g. the type of scrambling for low discrepancy numbers.
- `mcLoops, mcSteps`: the number of Monte Carlo loops and Monte Carlo Steps. In total `mcLoops*mcSteps` samples are included in one MC simulation.
- `path.gen`: the user defined function to generate the price path. As the only input argument serves the matrix of innovations, the option parameters must be available as global variables.
蒙特卡洛选项

蒙特卡洛期权的计算方法

路径长度 (pathLength)
路径长度是价格路径的长度。它可以通过 floor(Time/delta.t) 来计算，其中 Time 是到期时间，单位为年。

支付函数 (payoff.calc)
支付函数是用户定义的函数，用于计算期权的支付。作为输入参数，它接受路径矩阵作为由路径生成器返回的路径矩阵。选项参数必须作为全局变量可用。

标准化 (standardization)
一个逻辑标志，表示一个循环中的创新是否应标准化？默认为 TRUE。

跟踪 (trace)
一个逻辑标志，表示蒙特卡洛模拟应被跟踪吗？默认为 TRUE。

...（省略部分）
额外的参数传递给创新生成器。

详情

创新

创新必须由用户定义的创新生成器创建。生成器必须返回一个包含 (随机) 创新的数值矩阵，大小为 mcSteps 乘以 pathLength。示例部分显示如何为逐次蒙特卡洛抽样生成 Scrambled Quasi Monte Carlo Sobol 数字。该包包含三个生成器： rnorm.pseudo, rnorm.halton 和 rnorm.sobol，这些生成器可以很容易地用于模拟。

价格路径

用户必须提供一个生成价格路径的函数。在示例部分，函数 wienerPath 创建了一个从随机创新的 Wiener 蒙特卡洛路径。Wiener 路径要求输入 b，年化成本-拥金率，和 sigma，年化波动率的基底证券，以计算路径的漂移和方差，这些变量必须是全局定义的。

支付函数

用户还必须提供一个计算期权支付值的函数。示例部分显示如何为普通 vanilla 期权和普通的算术亚洲期权编写支付计算器。作为输入参数，路径矩阵是必需的。再次，选项参数必须是全局可用的。

蒙特卡洛模拟器

模拟器是蒙特卡洛估值过程的核心。此模拟器执行 mcLoops 蒙特卡洛循环，每个循环有 mcSteps 蒙特卡洛步骤。在每个循环中，以下步骤会执行：首先，创新矩阵由指定的创新生成器创建（通常从正常随机数生成器或低分歧生成器构建），然后如果需要，假想创新会被添加（默认为 true），然后创新可以在每个循环中被标准化（默认为 false），然后计算每个循环中所有样本的平均支付。模拟可以被跟踪循环以循环设置参数 trace=TRUE。
MonteCarloOptions

Value

The user defined innovation generator
returns a numeric matrix of (random) innovations to build the Monte Carlo Paths.

The user defined path generator
returns a numeric matrix of the Monte Carlo paths for the calculation of the option’s payoffs. To be more precise, as an example the function returns for a Wiener process the matrix \((b-\sigma\sigma/2)\delta t + \sigma\delta \sqrt{\delta t}\)
where the first term corresponds to the drift and the second to the volatility.

The user defined payoff calculator,
returns the vector of the option’s payoffs calculated from the generated paths. As an example this becomes for an arithmetic Asian call option with a Wiener Monte Carlo path payoff \(= \exp(-r\text{Time})\max(SM-X, 0)\)
where \(SM = \text{mean}(S*\exp(\text{cumsum(path)))}\) and path denotes the MC price paths.

MonteCarloOption:
returns a vector with the option prices for each Monte Carlo loop.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

Jaeckel P. (2002); Monte Carlo Methods in Finance, John Wiley and Sons Ltd, 222 pp.

Examples

## How to perform a Monte Carlo Simulation?

## First Step:
# Write a function to generate the option's innovations.
# Use scrambled normal Sobol numbers:
sobolInnovations <- function(mcSteps, pathLength, init, ...) {
  # Create and return Normal Sobol Innovations:
  returnNorm.sobol(mcSteps, pathLength, init, ...)
}

## Second Step:
# Write a function to generate the option's price paths.
# Use a Wiener path:
wienerPath <- function(eps) {
  # Note, the option parameters must be globally defined!
MonteCarloOptions

```r
# Generate and return the Paths:
(b-sigma*sigma/2)*delta.t + sigma*sqrt(delta.t)*eps
}

## Third Step:
# Write a function for the option's payoff

# Example 1: use the payoff for a plain Vanilla Call or Put:
plainVanillaPayoff <- function(path) {
  # Note, the option parameters must be globally defined!
  # Compute the Call/Put Payoff Value:
  ST <- S*exp(sum(path))
  if (TypeFlag == "c") payoff <- exp(-r*Time)*max(ST-X, 0)
  if (TypeFlag == "p") payoff <- exp(-r*Time)*max(0, X-ST)
  # Return Value:
  payoff
}

# Example 2: use the payoff for an arithmetic Asian Call or Put:
arithmeticAsianPayoff <- function(path) {
  # Note, the option parameters must be globally defined!
  # Compute the Call/Put Payoff Value:
  SM <- mean(S*exp(cumsum(path)))
  if (TypeFlag == "c") payoff <- exp(-r*Time)*max(SM-X, 0)
  if (TypeFlag == "p") payoff <- exp(-r*Time)*max(0, X-SM)
  # Return Value:
  payoff
}

## Final Step:
# Set Global Parameters for the plain Vanilla / arithmetic Asian Options:
TypeFlag <- "c"; S <- 100; X <- 100
Time <- 1/12; sigma <- 0.4; r <- 0.10; b <- 0.1

# Do the Asian Simulation with scrambled random numbers:
mc <- MonteCarloOption(delta.t = 1/360, pathLength = 30, mcSteps = 5000,
  mcloops = 50, init = TRUE, innovations.gen = sobolInnovations,
  path.gen = wienerPath, payoff.calc = arithmeticAsianPayoff,
  antithetic = TRUE, standardization = FALSE, trace = TRUE,
  scrambling = 2, seed = 4711)

# Plot the MC Iteration Path:
par(mfrow = c(1, 1))
mcPrice <- cumsuc(mcc)/{(1:length(mcc))
plot(mcPrice, type = "l", main = "Arithmetic Asian Option",
  xlab = "Monte Carlo Loops", ylab = "Option Price")

# Compare with Turnbull-Wakeman Approximation:
if(FALSE) { # ... requires(fExoticOptions)
  TW <- TurnbullWakemanAsianApproxOption(
    TypeFlag = "c", S = 100, SA = 100, X = 100,
    Time = 1/12, time = 1/12, tau = 0 , r = 0.1,
    b = 0.1, sigma = 0.4)$price
```
Valuation of Plain Vanilla Options

Description

A collection and description of functions to valuate plain vanilla options. Included are functions for the Generalized Black-Scholes option pricing model, for options on futures, some utility functions, and print and summary methods for options.

The functions are:

- **GBS**
  - BlackScholesOption: the generalized Black-Scholes option,
  - Black76Option: a synonyme for the GBSOption,
  - MiltersenSchwartzOption: options on commodity futures,
  - NDF, CND, CBND: distribution functions,
  - print: print method for Options,
  - summary: summary method for Options.

Usage

```r
GBSOption(TypeFlag, S, X, Time, r, b, sigma, title = NULL, description = NULL)
GBSGreeks(Selection, TypeFlag, S, X, Time, r, b, sigma)
GBSCharacteristics(TypeFlag, S, X, Time, r, b, sigma)
GBSVolatility(price, TypeFlag, S, X, Time, r, b, tol, maxiter)
BlackScholesOption(...)
Black76Option(TypeFlag, FT, X, Time, r, sigma, title = NULL, description = NULL)
MiltersenSchwartzOption(TypeFlag, Pt, FT, X, time, Time, sigmas, sigmE, sigmaF, rhoSE, rhoSF, rhoEF, KappaE, KappaF, title = NULL, description = NULL)

NDF(x)
CND(x)
CBND(x1, x2, rho)
```

## S4 method for signature 'fOPTION'
show(object)
## S3 method for class 'fOPTION'

summary(object, ...)

## S3 method for class 'option'

print(x, ...)

## S3 method for class 'option'

summary(object, ...)

### Arguments

- **b**
  - the annualized cost-of-carry rate, a numeric value; e.g. 0.1 means 10% pa.

- **description**
  - a character string which allows for a brief description.

- **FT**
  - [Black76*][MiltersenSchwartz*] - the futures price, a numeric value.

- **KappaE, KappaF**
  - [MiltersenSchwartz*] - the speed of mean reversion of the forward interest rate (E), the speed of mean reversion of the convenience yield (F), a numeric value.

- **maxiter, tol**
  - [GBSVolatility*] - the maximum number of iterations and the tolerance to compute the root of the GBS volatility equation, see uniroot.

- **object**
  - an object of class "option".

- **price**
  - [GBSVolatility*] - the price of the GBS option, a numerical value.

- **Pt**
  - [MiltersenSchwartz*] - the zero coupon bond that expires on the option maturity; a numeric value.

- **r**
  - the annualized rate of interest, a numeric value; e.g. 0.25 means 25% pa.

- **rhoSE, rhoSF, rhoEF**
  - [MiltersenSchwartz*] - the correlations between the spot commodity price and the future convenience yield (SE), between the spot commodity price and the forward interest rate (SF), between the forward interest rate and the future convenience yield (EF), a numeric value.

- **S**
  - the asset price, a numeric value.

- **Selection**
  - [GBSGreeks] - sensitivity to be computed, one of "delta", "gamma", "vega", "theta", "rho", or "CoC", a string value.

- **sigma**
  - the annualized volatility of the underlying security, a numeric value; e.g. 0.3 means 30% volatility pa.

- **sigmaS, sigmaE, sigmaF**
  - [MiltersenSchwartz*] - numeric values, the annualized volatility of the spot commodity price (S), of the future convenience yield (E), and of the forward interest rate (F), e.g. 0.25 means 25% pa.

- **time, Time**
  - the time to maturity measured in years, a numeric value.
Plain Vanilla Options

title: a character string which allows for a project title.
TypeFlag: a character string either "c" for a call option or a "p" for a put option.

\[NDF][CND][CBND]\ -
the function argument x for the normal distribution function NDF and the cumulated normal distribution CND. The arguments for the bivariate function are named x1 and x2; rho is the correlation coefficient.

[print] -
the object x to be printed.

X: a numeric value, the exercise price.

... arguments to be passed.

Details

Generalized Black Scholes Options:

GBSOption calculates the option price, GBSSGreens calculates option sensitivities delta, theta, vega, rho, lambda and gamma, and GBSSCharacteristics does both. GBSVolatility computes the implied volatility.

Note, that setting \(b = r\) we get Black and Scholes’ stock option model, \(b = r - q\) we get Merton’s stock option model with continuous dividend yield \(q\), \(b = 0\) we get Black’s futures option model, and \(b = r - r_f\) we get Garman and Kohlhagen’s currency option model with foreign interest rate \(r_f\).

Options on Futures:

The Black760Option pricing formula is applicable for valuing European call and European put options on commodity futures. The exact nature of the underlying commodity varies and may be anything from a precious metal such as gold or silver to agricultural products.

The Miltersen Schwartz Option model is a three factor model with stochastic futures prices, term structures and convenience yields, and interest rates. The model is based on lognormal distributed commodity prices and normal distributed continuously compounded forward interest rates and future convenience yields.

Miltersen Schwartz Options:

The MiltersenSchwartzOption function allows for pricing options on commodity futures. The model is a three factor model with stochastic futures prices, term structures of convenience yields, and interest rates. The model is based on lognormal distributed commodity prices and normal distributed continuously compounded forward interest rates and futures convenience yields.

Distribution Functions:

The functions NDF, CND, and CBND compute values for the Normal density functions, for the normal probability function, and for the bivariate normal probability functions. The functions are implemented as described in the book of E.G. Haug.

Print and Summary Method:
PlainVanillaOptions

There are two methods to print and summarize an object of class "fOPTION" or of "option". The second is used for the older class representation.

Value

GBSOption
BlackScholesOption
returns an object of class "fOption".

GBSGreeks
returns the option sensitivity for the selected Greek, a numeric value.

GBSCharacteristics
returns a list with the following entries: premium, the option price, delta, the delta sensitivity, gamma, the gamma sensitivity, theta, the theta sensitivity, vega, the vega sensitivity, rho, the rho sensitivity, lambda, the lambda sensitivity.

GBSVolatility
returns the GBS option implied volatility for a given price.

Black76Option,
MiltersenSchwartzOption
return an object of class "fOPTION".

The option valuation programs return an object of class "fOPTION" with the following slots:

@call the function call.
@parameters a list with the input parameters.
@price a numeric value with the value of the option.
@title a character string with the name of the test.
@description a character string with a brief description of the test.

Note

The functions implement algorithms to valuate plain vanilla options and to compute option Greeks as described in Chapter 1 of Haug’s Option Guide (1997).

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References


Examples

```r
## All the examples are from Haug's Option Guide (1997)

## CHAPTER 1.1: ANALYTICAL FORMULAS FOR EUROPEAN OPTIONS:

## Black Scholes Option [Haug 1.1.1]
GBSOption(VariableFlag = "c", S = 60, X = 65, Time = 1/4, r = 0.08,
b = 0.08, sigma = 0.30)

## European Option on a Stock with Cash Dividends [Haug 1.1.2]
S0 = 100; r = 0.10; D1 = D2 = 2; t1 = 1/4; t2 = 1/2
S = S0 - 2*exp(-r*t1) - 2*exp(-r*t2)
GBSOption(VariableFlag = "c", S = S, X = 90, Time = 3/4, r = r, b = r,
sigma = 0.25)

## Options on Stock Indexes [Haug 1.2.3]
GBSOption(VariableFlag = "p", S = 100, X = 95, Time = 1/2, r = 0.10,
b = 0.10-0.05, sigma = 0.20)

## Option on Futures [Haug 1.1.4]
FuturesPrice = 19
GBSOption(VariableFlag = "c", S = FuturesPrice, X = 19, Time = 3/4,
r = 0.10, b = 0, sigma = 0.28)

## Currency Option [Haug 1.1.5]
r = 0.06; rf = 0.08
GBSOption(VariableFlag = "c", S = 1.5600, X = 1.6000,
Time = 1/2, r = 0.06, b = 0.06-0.08, sigma = 0.12)

## Delta of GBS Option [Haug 1.3.1]
GBS Greeks(Selection = "delta", VariableFlag = "c", S = 105, X = 100,
Time = 1/2, r = 0.10, b = 0, sigma = 0.36)

## Gamma of GBS Option [Haug 1.3.3]
GBS Greeks(Selection = "gamma", VariableFlag = "c", S = 55, X = 60,
Time = 0.75, r = 0.10, b = 0.10, sigma = 0.30)

## Vega of GBS Option [Haug 1.3.4]
GBS Greeks(Selection = "vega", VariableFlag = "c", S = 55, X = 60,
Time = 0.75, r = 0.10, b = 0.10, sigma = 0.30)

## Theta of GBS Option [Haug 1.3.5]
GBS Greeks(Selection = "theta", VariableFlag = "p", S = 430, X = 405,
Time = 0.8833, r = 0.07, b = 0.07-0.05, sigma = 0.20)

## Rho of GBS Option [Haug 1.3.5]
```
GBSGreeks(Selection = "rho", TypeFlag = "c", S = 72, X = 75, Time = 1, r = 0.09, b = 0.09, sigma = 0.19)

## CHAPTER 1.3 OPTIONS SENSITIVITIES:

## The Generalized Black Scholes Option Formula
GBSCharacteristics(TypeFlag = "p", S = 1.5600, X = 1.6000, Time = 1, r = 0.09, b = 0.09, sigma = 0.19)

## CHAPTER 1.5: RECENT DEVELOPMENTS IN COMMODITY OPTIONS

## Miltsersen Schwartz Option vs. Black76 Option on Futures:
MiltersenSchwartzOption(TypeFlag = "c", Pt = exp(-0.05/4), FT = 95, X = 80, time = 1/4, Time = 1/2, sigmaS = 0.2660, sigmaE = 0.2490, sigmaF = 0.0095, rhoSE = 0.805, rhoSF = 0.0805, rhoEF = 0.1243, KappaE = 1.045, KappaF = 0.200)
Black76Option(TypeFlag = "c", FT = 95, X = 80, Time = 1/2, r = 0.05, sigma = 0.266)
Index

*Topic math
  BasicAmericanOptions, 4
  BinomialTreeOptions, 7
  HestonNandiOptions, 13
  PlainVanillaOptions, 22
*Topic models
  HestonNandiGarchFit, 10
*Topic package
  fOptions-package, 2
*Topic programming
  LowDiscrepancy, 15
  MonteCarloOptions, 18

arithmeticAsianMCPayoff
  (MonteCarloOptions), 18

BasicAmericanOptions, 4
BNAmericanApproxOption
  (BasicAmericanOptions), 4
BinomialTreeOption
  (BinomialTreeOptions), 7
BinomialTreeOptions, 7
BinomialTreePlot (BinomialTreeOptions), 7
Black76Option (PlainVanillaOptions), 22
BlackScholesOption
  (PlainVanillaOptions), 22
BSAmericanApproxOption
  (BasicAmericanOptions), 4

CBND (PlainVanillaOptions), 22
CND (PlainVanillaOptions), 22
CRRBinomialTreeOption
  (BinomialTreeOptions), 7
fOPTION (PlainVanillaOptions), 22
fOPTION-class (PlainVanillaOptions), 22
fOptions (fOptions-package), 2
fOptions-package, 2

GBSCharacteristics
  (PlainVanillaOptions), 22
GBSGreeks (PlainVanillaOptions), 22
GBSOption (PlainVanillaOptions), 22
GBSVolatility (PlainVanillaOptions), 22
HestonNandiGarchFit, 10
HestonNandiOptions, 13
hngarchFit (HestonNandiGarchFit), 10
hngarchSim (HestonNandiGarchFit), 10
hngarchStats (HestonNandiGarchFit), 10
HNGCharacteristics
  (HestonNandiOptions), 13
HNGGreeks (HestonNandiOptions), 13
HNOption (HestonNandiOptions), 13
JRBinomialTreeOption
  (BinomialTreeOptions), 7
LowDiscrepancy, 15

MiltersenSchwartzOption
  (PlainVanillaOptions), 22
MonteCarloOption (MonteCarloOptions), 18
MonteCarloOptions, 18

NDF (PlainVanillaOptions), 22
plainVanillaMCPayoff
  (MonteCarloOptions), 18

print.hngarch (HestonNandiGarchFit), 10
print.option (PlainVanillaOptions), 22

rnorm.halton (LowDiscrepancy), 15
rnorm.pseudo (LowDiscrepancy), 15
rnorm.sobol (LowDiscrepancy), 15
RollGeskeWhaleyOption
  (BasicAmericanOptions), 4
runif.halton (LowDiscrepancy), 15
runif.pseudo (LowDiscrepancy), 15
runif.sobol (LowDiscrepancy), 15

show, fOPTION-method
(PplainVanillaOptions), 22
summary.fOPTION (PlainVanillaOptions),
22
summary.hngarch (HestonNandiGarchFit),
10
summary.option (PlainVanillaOptions), 22

TIANBinomialTreeOption
(BinomialTreeOptions), 7

wienerMCPath (MonteCarloOptions), 18