Package ‘derivmkts’

June 6, 2019

Title Functions and R Code to Accompany Derivatives Markets
Version 0.2.4
Author Robert McDonald [aut, cre, cph]
Maintainer Robert McDonald <rmcd1024@gmail.com>
Description A set of pricing and expository functions that should be useful in teaching a course on financial derivatives.
Imports graphics, stats, grDevices, mnormt
Depends R (>= 3.0.0)
Suggests highlight, markdown, knitr, rmarkdown, ggplot2, dplyr, tidyr
License MIT + file LICENSE
VignetteBuilder knitr, ggplot2, dplyr, tidyr
LazyData true
Encoding UTF-8
RoxygenNote 6.1.1
NeedsCompilation no
Repository CRAN
Date/Publication 2019-06-06 14:10:03 UTC

R topics documented:

 arthiasianmc .................................................. 2
 arithavgpricecv ........................................... 3
 asiangeomavg ............................................... 4
 barriers ...................................................... 5
 binom .......................................................... 7
 blksch .......................................................... 9
 bondsimple .................................................... 11
 compound ..................................................... 12
 geomasianmc ............................................... 13
 greeks .......................................................... 14
 implied ....................................................... 16
Description

Monte Carlo pricing calculations for European Asian options. `arithasianmc` and `geomasianmc` compute Monte Carlo prices for the full range of average price and average strike call and puts computes prices of a complete assortment of Arithmetic Asian options (average price call and put and average strike call and put)

Arithmetic average Asian option prices

Usage

`arithasianmc(s, k, v, r, tt, d, m, numsim=1000, printsds=False)`

Arguments

- `s`: Price of underlying asset
- `k`: Strike price of the option. In the case of average strike options, \( k/s \) is the multiplier for the average
- `v`: Volatility of the underlying asset price, defined as the annualized standard deviation of the continuously-compounded return
- `r`: Annual continuously-compounded risk-free interest rate
- `tt`: Time to maturity in years
- `d`: Dividend yield, annualized, continuously-compounded
- `m`: Number of prices in the average calculation
- `numsim`: Number of Monte Carlo iterations
- `printsds`: Print standard deviation for the particular Monte Carlo calculation

Value

Array of arithmetic average option prices, along with vanilla European option prices implied by the the simulation. Optionally returns Monte Carlo standard deviations.

See Also

Other Asian: `arithavgpricev`, `asiangeomavg`, `geomasianmc`
arithavgpricecv

Examples

s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; m=3; numsim=1e04
arithasianmc(s, k, v, r, tt, d, m, numsim, printsds=TRUE)

arithavgpricecv Control variate asian call price

Description

Calculation of arithmetic-average Asian call price using control variate Monte Carlo valuation

Usage

arithavgpricecv(s, k, v, r, tt, d, m, numsim)

Arguments

s  Price of underlying asset
k  Strike price of the option. In the case of average strike options, k/s is the multiplier for the average
v  Volatility of the underlying asset price, defined as the annualized standard deviation of the continuously-compounded return
r  Annual continuously-compounded risk-free interest rate
tt  Time to maturity in years
d  Dividend yield, annualized, continuously-compounded
m  Number of prices in the average calculation
numsim  Number of Monte Carlo iterations

Value

Vector of the price of an arithmetic-average Asian call, computed using a control variate Monte Carlo calculation, along with the regression beta used for adjusting the price.

See Also

Other Asian: arithasianmc, asiangeomavg, geomasianmc

Examples

s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; m=3; numsim=1e04
arithavgpricecv(s, k, v, r, tt, d, m, numsim)
asiangeomavg  

**Description**

Pricing functions for European Asian options based on geometric averages. `geomavgpricecall`, `geomavgpriceput`, `geomavgstrikecall` and `geomavgstrikeput` compute analytical prices of geometric Asian options using the modified Black-Scholes formula.

**Usage**

- `geomavgprice(s, k, v, r, tt, d, m, cont=FALSE)`
- `geomavgpricecall(s, k, v, r, tt, d, m, cont=FALSE)`
- `geomavgpriceput(s, k, v, r, tt, d, m, cont=FALSE)`
- `geomavgstrike(s, km, v, r, tt, d, m, cont=FALSE)`
- `geomavgstrikecall(s, km, v, r, tt, d, m, cont=FALSE)`
- `geomavgstrikeput(s, km, v, r, tt, d, m, cont=FALSE)`

**Arguments**

- `s` Price of underlying asset
- `k` Strike price of the option. In the case of average strike options, `k/s` is the multiplier for the average
- `v` Volatility of the underlying asset price, defined as the annualized standard deviation of the continuously-compounded return
- `r` Annual continuously-compounded risk-free interest rate
- `tt` Time to maturity in years
- `d` Dividend yield, annualized, continuously-compounded
- `m` Number of prices in the average calculation
- `cont` Boolean which when TRUE denotes continuous averaging
- `km` The strike multiplier, relative to the initial stock price, for an average price payoff. If the initial stock price is `s = 120` and `km = 115`, the payoff for an average strike call is

\[
Payoff = \max(ST - km/s * S_{Avg}, 0)
\]

**Value**

Option prices as a vector

**See Also**

Other Asian: `arithasianmc`, `arithavgpricecv`, `geomasianmc`
Examples

s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; m=3;
geomavgpricecall(s, k, v, r, tt, d, m)
geomavgpricecall(s, 38:42, v, r, tt, d, m)
geomavgpricecall(s, 38:42, v, r, tt, d, m, cont=TRUE)

barriers  Barrier option pricing

Description

This library provides a set of barrier binary options that are used to construct prices of barrier options. The nomenclature is that

- \"call\" and \"put\" refer to claims that are exercised when the asset price is above or below the strike;
- \"up\" and \"down\" refer to claims for which the barrier is above or below the current asset price; and
- \"in\" and \"out\" refer to claims that knock in or out

For example, for standard barrier options, calldownin refers to a knock-in call for which the barrier is below the current price, while putdownout refers to a knock-out put for which the barrier is below the current asset price.

For binary barrier options, \"ui\", \"di\" \"uo\", and \"do\" refer to up-and-in, down-and-in, up-and-out, and down-and-out options.

Rebate options pay \$1 if a barrier is reached. The barrier can be reached from above (\"d\") or below (\"d\"), and the payment can occur immediately (\"ur\" or \"dr\") or at expiration (\"driddeferred\" and \"urdeferred\")

callupin(s, k, v, r, tt, d, H) = assetuicall(s, k, v, r, tt, d, H) - k*cashuicall(s, k, v, r, tt, d, H)

Usage

callupin(s, k, v, r, tt, d, H)
callupout(s, k, v, r, tt, d, H)
putupin(s, k, v, r, tt, d, H)
putupout(s, k, v, r, tt, d, H)
calldownin(s, k, v, r, tt, d, H)
calldownout(s, k, v, r, tt, d, H)
putdownin(s, k, v, r, tt, d, H)
putdownout(s, k, v, r, tt, d, H)
uicall(s, k, v, r, tt, d, H) ## same as callupin
uocall(s, k, v, r, tt, d, H) ## same as callupout
uiput(s, k, v, r, tt, d, H) ## same as putupin
uoput(s, k, v, r, tt, d, H) ## same as putupout
dicall(s, k, v, r, tt, d, H) ## same as calldownin
Arguments

- **s**: Stock price
- **k**: Strike price of the option
- **v**: Volatility of the stock, defined as the annualized standard deviation of the continuously-compounded return
- **r**: Annual continuously-compounded risk-free interest rate
- **tt**: Time to maturity in years
- **d**: Dividend yield, annualized, continuously-compounded
- **H**: Barrier
- **perpetual**: Boolean for the case where an up or down rebate is infinitely lived. Default is FALSE.

Details

Returns a scalar or vector of option prices, depending on the inputs.

Value

The pricing functions return the price of a barrier claim. If more than one argument is a vector, the recycling rule determines the handling of the inputs.
Examples

s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; H=44
callupin(s, k, v, r, tt, d, H)

## following returns the same price as previous
assetuicall(s, k, v, r, tt, d, H) - k=cashuicall(s, k, v, r, tt, d, H)

## return option prices for different strikes putupin(s, k=38:42,
v, r, tt, d, H)

---

**binom**

**Binomial option pricing**

---

**Description**

`binomopt` using the binomial pricing algorithm to compute prices of European and American calls and puts.

**Usage**

```r
binomopt(s, k, v, r, tt, d, nstep = 10, american = TRUE,
          putopt=FALSE, specifyupdn=FALSE, crr=FALSE, jarrowrudd=FALSE,
          up=1.5, dn=0.5, returntrees=FALSE, returnparams=FALSE,
          returngreeks=FALSE)

binomplot(s, k, v, r, tt, d, nstep, putopt=FALSE, american=TRUE,
          plotvalues=FALSE, plotarrows=FALSE, drawstrike=TRUE,
          pointsize=4, ylimval=c(0,0),
          saveplot = FALSE, saveplotfn='binomialplot.pdf',
          crr=FALSE, jarrowrudd=FALSE, titles=TRUE, specifyupdn=FALSE,
          up=1.5, dn=0.5, returnprice=FALSE, logy=FALSE)
```

**Arguments**

- `s` Stock price
- `k` Strike price of the option
- `v` Volatility of the stock, defined as the annualized standard deviation of the continuously-compounded return
- `r` Annual continuously-compounded risk-free interest rate
- `tt` Time to maturity in years
- `d` Dividend yield, annualized, continuously-compounded
- `nstep` Number of binomial steps. Default is `nstep = 10`
- `american` Boolean indicating if option is American
- `putopt` Boolean TRUE is the option is a put
specifyupdn  Boolean, if TRUE, manual entry of the binomial parameters up and down. This overrides the crr and jarrowrudd flags

crr        TRUE to use the Cox-Ross-Rubinstein tree
jarrowrudd TRUE to use the Jarrow-Rudd tree
up, dn     If specifyupdn=TRUE, up and down moves on the binomial tree
returntrees If returntrees=TRUE, the list returned by the function includes four trees: for the price of the underlying asset (stree), the option price (oppricetree), where the option is exercised (exertree), and the probability of being at each node. This parameter has no effect if returnparams=FALSE, which is the default.
returnparams Return the vector of inputs and computed pricing parameters as well as the price
returngreeks Return time 0 delta, gamma, and theta in the vector greeks
plotvalues display asset prices at nodes
plotarrows draw arrows connecting pricing nodes
drawstrike draw horizontal line at the strike price
pointsixe CEX parameter for nodes
ylimval c(low, high) for ylim of the plot
saveplot boolean; save the plot to a pdf file named saveplotfn
saveplotfn file name for saved plot
titles automatically supply appropriate main title and x- and y-axis labels
returnprice if TRUE, the binomplot function returns the option price
logy (FALSE). If TRUE, y-axis is plotted on a log scale

Details

By default, binomopt returns an option price. Optionally, it returns a vector of the parameters used to compute the price, and if returntrees=TRUE it can also return the following matrices, all but but two of which have dimensionality (nstep + 1) × (nstep + 1):

stree the binomial tree for the price of the underlying asset.

oppricetree the binomial tree for the option price at each node

exertree the tree of boolean indicators for whether or not the option is exercised at each node

probtree the probability of reaching each node

delta at each node prior to expiration, the number of units of the underlying asset in the replicating portfolio. The dimensionality is (nstep) × (nstep)

bond at each node prior to expiration, the bond position in the replicating portfolio. The dimensionality is (nstep) × (nstep)

binomplot plots the stock price lattice and shows graphically the probability of being at each node (represented as the area of the circle at that price) and whether or not the option is optimally exercised there (green if yes, red if no), and optionally, ht, depending on the inputs.
**Value**

By default, binomopt returns the option price. If `returnparams=TRUE`, it returns a list where $price$ is the binomial option price and $params$ is a vector containing the inputs and binomial parameters used to compute the option price. Optionally, by specifying `returntrees=TRUE`, the list can include the complete asset price and option price trees, along with trees representing the replicating portfolio over time. The current delta, gamma, and theta are also returned. If `returntrees=False` and `returngreeks=TRUE`, only the current price, delta, gamma, and theta are returned. The function `binomplot` produces a visual representation of the binomial tree.

**Note**

By default, binomopt computes the binomial tree using up and down moves of

\[
u = \exp((r - d) \ast h + \sigma \sqrt{h})\]

and

\[
d = \exp((r - d) \ast h - \sigma \sqrt{h})\]

You can use different trees: There is a boolean variable `crr` to use the Cox-Ross-Rubinstein pricing tree, and you can also supply your own up and down moves with `specifyupdn=TRUE`. It’s important to realize that if you do specify the up and down moves, you are overriding the volatility parameter.

**Examples**

```r
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; nstep=15inomopt(s, k, v, r, tt, d, nstep, american=TRUE, putopt=TRUE)
binomopt(s, k, v, r, tt, d, nstep, american=TRUE, putopt=TRUE, returnparams=TRUE)

## matches Fig 10.8 in 3rd edition of Derivatives Markets
x <- \binomopt(110, 100, .3, .05, 1, 0.035, 3, american=TRUE,
returntrees=TRUE, returnparams=TRUE)
print(x$opprice)
print(x$delta)
print(x$gamma)

binomplot(s, k, v, r, tt, d, nstep, american=TRUE, putopt=TRUE)
binomplot(s, k, v, r, tt, d, nstep, american=FALSE, putopt=TRUE)
```

---

*blksch*  
*Black-Scholes option pricing*
Description

bscall and bsp put compute Black-Scholes call and put prices. The functions assetcall, assetput, cashcall, and cashput provide the prices of binary options that pay one share (the asset options) or $1 (the cash options) if at expiration the asset price exceeds the strike (the calls) or is below the strike (the puts). We have the identities

\[
\text{bscall}(s, k, v, r, tt, d) = \text{assetcall}(s, k, v, r, tt, d) - k \times \text{cashcall}(s, k, v, r, tt, d)
\]

\[
\text{bsp put}(s, k, v, r, tt, d) = k \times \text{cashput}(s, k, v, r, tt, d) - \text{assetput}(s, k, v, r, tt, d)
\]

Usage

bscall\((s, k, v, r, tt, d)\)
bsput\((s, k, v, r, tt, d)\)
assetcall\((s, k, v, r, tt, d)\)
cashcall\((s, k, v, r, tt, d)\)
assetput\((s, k, v, r, tt, d)\)
cashput\((s, k, v, r, tt, d)\)

Arguments

\begin{align*}
  s & \quad \text{Price of the underlying asset} \\
  k & \quad \text{Strike price} \\
  v & \quad \text{Volatility of the asset price, defined as the annualized standard deviation of the continuously-compounded return} \\
  r & \quad \text{Annual continuously-compounded risk-free interest rate} \\
  tt & \quad \text{Time to maturity in years} \\
  d & \quad \text{Dividend yield, annualized, continuously-compounded}
\end{align*}

Details

Returns a scalar or vector of option prices, depending on the inputs

Value

A Black-Scholes option price. If more than one argument is a vector, the recycling rule determines the handling of the inputs

Note

It is possible to specify the inputs either in terms of an interest rate and a "dividend yield" or an interest rate and a "cost of carry". In this package, the dividend yield should be thought of as the cash dividend received by the owner of the underlying asset, or (equivalently) as the payment received if the owner were to lend the asset.

There are other option pricing packages available for R, and these may use different conventions for specifying inputs. In fOptions, the dividend yield is replaced by the generalized cost of carry, which is the net payment required to fund a position in the underlying asset. If the interest rate is 10% and the dividend yield is 3%, the generalized cost of carry is 7% (the part of the interest
payment not funded by the dividend payment). Thus, using the GBS function from fOptions, these two expressions return the same price:

```r
bscall(s, k, v, r, tt, d)
foptions::GBSOption('c', S=s, K=k, Time=tt, r=r, b=r-d, sigma=v)
```

**Examples**

```r
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0;
bscall(s, k, v, r, tt, d)

## following returns the same price as previous
assetcall(s, k, v, r, tt, d) - k*cashcall(s, k, v, r, tt, d)

## return option prices for different strikes
bsput(s, k=38:42, v, r, tt, d)
```

---

**bondsimple**

**Simple Bond Functions**

**Description**

Basic yield, pricing, duration and convexity calculations. These functions perform simple present value calculations assuming that all periods between payments are the same length. Unlike bond functions in Excel, for example, settlement and maturity dates are not used. By default, duration is Macaulay duration.

**Usage**

```r
bondsimple::bondpv(coupon, mat, yield, principal, freq)
bondsimple::bondyield(price, coupon, mat, principal, freq)
bondsimple::duration(price, coupon, mat, principal, freq, modified)
bondsimple::convexity(price, coupon, mat, principal, freq)
```

**Arguments**

- **coupon**: annual coupon
- **mat**: maturity in years
- **yield**: annual yield to maturity. If freq > 1, the yield is freq times the per period yield.
- **principal**: maturity payment of the bond, in addition to the final coupon. Default value is $1,000. If the instrument is an annuity, set principal to zero.
- **freq**: number of payments per year.
- **price**: price of the bond
- **modified**: If true, compute modified duration, otherwise compute Macaulay duration. FALSE by default.
Value

Return price, yield, or duration/convexity.

Examples

coupon <- 6; mat <- 20; freq <- 2; principal <- 100; yield <- 0.045;

price <- bondpv(coupon, mat, yield, principal, freq) # 119.7263
bondyield(coupon, mat, price=price, principal, freq) # 0.045
duration(price, coupon, mat, principal, freq, modified=FALSE) # 12.5043
duration(price, coupon, mat, principal, freq, modified=TRUE) # 12.3928
convexity(price, coupon, mat, principal, freq) # 205.3245

compound

Description

A compound option is an option for which the underlying asset is an option. The underlying option (the option on which there is an option) in turn has an underlying asset. The definition of a compound option requires specifying

• whether you have the right to buy or sell an underlying option
• whether the underlying option (the option upon which there is an option) is a put or a call
• the price at which you can buy or sell the underlying option (strike price \( k_{co} \) — the strike on the compound option)
• the price at which you can buy or sell the underlying asset should you exercise the compound option (strike price \( k_{uo} \) — the strike on the underlying option)
• the date at which you have the option to buy or sell the underlying option (first exercise date, \( t_1 \))
• the date at which the underlying option expires, \( t_2 \)

Given these possibilities, you can have a call on a call, a put on a call, a call on a put, and a put on a put. The valuation procedure require knowing, among other things, the underlying asset price at which it will be worthwhile to acquire the underlying option.

Given the underlying option, there is a parity relationship: If you buy a call on a call and sell a call on a call, you have acquired the underlying call by paying the present value of the strike, \( k_{co} \).

Usage

binormsdist(x1, x2, rho)
optionsoncall(s, kuo, kco, v, r, t1, t2, d)
optionsonput(s, kuo, kco, v, r, t1, t2, d)
calloncall(s, kuo, kco, v, r, t1, t2, d, returnscritical)
callonput(s, kuo, kco, v, r, t1, t2, d, returnscritical)
putoncall(s, kuo, kco, v, r, t1, t2, d, returnscritical)
putonput(s, kuo, kco, v, r, t1, t2, d, returnscritical)
Arguments

s  
Price of the asset on which the underlying option is written

v  
Volatility of the underlying asset, defined as the annualized standard deviation of the continuously-compounded return

r  
Annual continuously-compounded risk-free interest rate

D  
Dividend yield of the underlying asset, annualized, continuously-compounded

kuo  
strike on the underlying option

kco  
strike on compound option (the price at which you would buy or sell the underlying option at time t1)

t1  
time until exercise for the compound option

t2  
time until exercise for the underlying option

x1, x2  
values at which the cumulative bivariate normal distribution will be evaluated

rho  
correlation between x1 and x2

d  
returns is 
(FALSE) boolean determining whether the function returns just the options price (the default) or the option price along with the asset price above or below which the compound option is exercised.

Value

The option price, and optionally, the stock price above or below which the compound option is exercised. The compound option functions are not vectorized, but the greeks function should work, apart from theta.

Note

The compound option formulas are not vectorized.
Arguments

- **s**: Price of underlying asset
- **k**: Strike price of the option. In the case of average strike options, $k/s$ is the multiplier for the average
- **v**: Volatility of the underlying asset price, defined as the annualized standard deviation of the continuously-compounded return
- **r**: Annual continuously-compounded risk-free interest rate
- **tt**: Time to maturity in years
- **d**: Dividend yield, annualized, continuously-compounded
- **m**: Number of prices in the average calculation
- **numsim**: Number of Monte Carlo iterations
- **printsds**: Print standard deviation for the particular Monte Carlo calculation

Value

Array of geometric average option prices, along with vanilla European option prices implied by the simulation. Optionally returns Monte Carlo standard deviations. Note that exact solutions for these prices exist, the purpose is to see how the Monte Carlo prices behave.

See Also

Other Asian: *arithasianmc, arithavgpricecv, asiangeomavg*

Examples

```r
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; m=3; numsim=1e04
gemasianmc(s, k, v, r, tt, d, m, numsim, printsds=FALSE)
```

---

greeks

*Calculate option Greeks*

Description

The functions *greeks* and *greeks2* provide two different calling conventions for computing a full set of option Greeks. *greeks* simply requires entering a pricing function with parameters. *greeks2* requires the use of named parameter entries. The function *bsopt* calls *greeks2* to produce a full set of prices and greeks for calls and puts. These functions are all vectorized, the only restriction being that the functions will produce an error if the recycling rule cannot be used safely (that is, if parameter vector lengths are not integer multiples of one another).

Usage

```r
greeks(f, complete=FALSE, long=FALSE, initcaps=TRUE)
# must used named list entries:
greeks2(fn, ...)
bsopt(s, k, v, r, tt, d)
```
Arguments

s  Price of underlying asset
k  Option strike price
v  Volatility of the underlying asset, defined as the annualized standard deviation of the continuously-compounded return
r  Annual continuously-compounded risk-free interest rate
tt  Time to maturity in years
d  Dividend yield of the underlying asset, annualized, continuously-compounded
fn  Pricing function name, not in quotes
f  Fully-specified option pricing function, including inputs which need not be named. For example, you can enter greeks(bscall(40, 40, .3, .08, .25, 0))
complete  FALSE. If TRUE, return a data frame with columns equal to input parameters, function name, premium, and greeks (each greek is a column). This is experimental and the output may change. Convert to long format using long=TRUE.
long  FALSE. Setting long=TRUE returns a long data frame, where each row contains input parameters, function name, and either the premium or one of the greeks. long=TRUE implies complete=TRUE
initcaps  TRUE. If true, capitalize names (e.g. "Delta" vs "delta")
...  Pricing function inputs, must be named, may either be a list or not

Details

Numerical derivatives are calculated using a simple difference. This can create numerical problems in edge cases. It might be good to use the package numDeriv or some other more sophisticated calculation, but the current approach works well with vectorization.

Value

A named list of Black-Scholes option prices and Greeks, or optionally (‘complete=TRUE’) a dataframe.

Note

The pricing function being passed to the greeks function must return a numeric vector. For example, callperpetual must be called with the option showbarrier=FALSE (the default). The pricing function call cannot contain a variable named ‘z91k25’.

Examples

s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0;
greeks(bscall(s, k, v, r, tt, d), complete=FALSE, long=FALSE, initcaps=TRUE)
greeks2(bscall, list(s=s, k=k, v=v, r=r, tt=tt, d=d))[[‘Delta’, ‘Gamma’], ]
bsopt(s, c35, 40, 45), v, r, tt, d)
bsopt(s, c35, 40, 45), v, r, tt, d)[[‘Call’]][c(‘Delta’, ‘Gamma’), ]
## plot Greeks for calls and puts for 500 different stock prices
##
## This plot can generate a "figure margins too large" error
## in RStudio

```r
k <- 100; v <- 0.30; r <- 0.08; tt <- 2; d <- 0
S <- seq(.5, 250, by=.5)
Call <- greeks(bcall(S, k, v, r, tt, d))
Put <- greeks(bput(S, k, v, r, tt, d))
y <- list(Call=Call, Put=Put)
par(mfrow=c(4, 4), mar=c(2, 2, 2, 2))  # create a 4x4 plot
for (i in names(y)) {
  for (j in rownames(y[[i]])) {
    # loop over greeks
    plot(S, y[[i]][j], main=paste(i, j), ylab=j, type='l')
  }
}
```

## Not run:
## Using complete option for calls
```r
call_long <- greeks(bcall(S, k, v, r, tt, d), long=TRUE)
```
```r
ggplot2::ggplot(call_long, aes(x=s, y=value)) +
  geom_line() + facet_wrap(~greek, scales='free')
```

## End(Not run)

---

**Black-Scholes implied volatility and price**

**Description**

`bscallimpvol` and `bsputimpvol` compute Black-Scholes implied volatilities. The functions `bscallimps` and `bsputimps`, compute stock prices implied by a given option price, volatility and option characteristics.

**Usage**

```r
bscallimpvol(s, k, r, tt, d, price)
bsputimpvol(s, k, r, tt, d, price)
bscallimps(s, k, v, r, tt, d, price)
bsputimps(s, k, v, r, tt, d, price)
```

**Arguments**

- `s`  Stock price
- `k`  Strike price of the option
- `v`  Volatility of the stock, defined as the annualized standard deviation of the continuously-compounded return
- `r`  Annual continuously-compounded risk-free interest rate
- `tt`  Time to maturity in years
- `d`  Dividend yield, annualized, continuously-compounded
- `price`  Option price when computing an implied value
Option pricing with jumps

Description

The functions cashjump, assetjump, and mertonjump return call and put prices, as vectors named "Call" and "Put", or "Call1", "Call2", etc. in case inputs are vectors. The pricing model is the Merton jump model, in which jumps are lognormally distributed.

Usage

assetjump(s, k, v, r, tt, d, lambda, alphaj, vj, complete)
cashjump(s, k, v, r, tt, d, lambda, alphaj, vj, complete)
mertonjump(s, k, v, r, tt, d, lambda, alphaj, vj, complete)

Arguments

s Stock price
k Strike price of the option
v Volatility of the stock, defined as the annualized standard deviation of the continuously-compounded return
\[ r \]  \text{Annual continuously-compounded risk-free interest rate}

\[ tt \]  \text{Time to maturity in years}

\[ d \]  \text{Dividend yield, annualized, continuously-compounded}

\[ \lambda \]  \text{Poisson intensity: expected number of jumps per year}

\[ \alpha_j \]  \text{Mean change in log price conditional on a jump}

\[ \nu_j \]  \text{Standard deviation of change in log price conditional on a jump}

\[ \text{complete} \]  \text{Return inputs along with prices, all in a data frame}

\section*{Details}

Returns a scalar or vector of option prices, depending on the inputs.

\section*{Value}

A vector of call and put prices computed using the Merton lognormal jump formula.

\section*{See Also}


bscall bsput

\section*{Examples}

\begin{verbatim}
 s <- 40; k <- 40; v <- 0.30; r <- 0.08; tt <- 2; d <- 0;
 lambda <- 0.75; alphaj <- -0.05; vj <- .35;
 bscall(s, k, v, r, tt, d)
 bspput(s, k, v, r, tt, d)
 mertonjump(s, k, v, r, tt, d, 0, 0, 0)
 mertonjump(s, k, v, r, tt, d, lambda, alphaj, vj)

 # following returns the same price as previous
 c(1, -1)*(assetjump(s, k, v, r, tt, d, lambda, alphaj, vj) -
 k*cashjump(s, k, v, r, tt, d, lambda, alphaj, vj))

 # return call prices for different strikes
 kseq <- 35:45
 cp <- mertonjump(s, kseq, v, r, tt, d, lambda, alphaj,
 vj)$call

 # Implied volatilities: Compute Black-Scholes implied volatilities
 # for options priced using the Merton jump model
 vimp <- sapply(1:length(kseq), function(i) bscallimpvol(s, kseq[i],
 r, tt, d, cp[i]))
 plot(kseq, vimp, main='Implied volatilities', xlab='Strike',
 ylab='Implied volatility', ylim=c(0.30, 0.50))
\end{verbatim}
**Perpetual American options**

**Description**

callperpetual and putperpetual compute prices of perpetual American options. The functions optionally return the exercise barriers (the prices at which the options are optimally exercised).

**Usage**

callperpetual(s, k, v, r, d, showbarrier)
putperpetual(s, k, v, r, d, showbarrier)

**Arguments**

- `s` : Price of the underlying asset
- `k` : Strike price
- `v` : Volatility of the asset price, defined as the annualized standard deviation of the continuously-compounded return
- `r` : Annual continuously-compounded risk-free interest rate
- `d` : Dividend yield, annualized, continuously-compounded
- `showbarrier` : Boolean (FALSE). If TRUE, the option price and exercise barrier are returned as a list

**Details**

Returns a scalar or vector of option prices, depending on the inputs

callperpetual(s, k, v, r, tt, d)

**Value**

Option price, and optionally the optimal exercise barrier.

**Note**

If the dividend yield is zero, a perpetual call is never exercised. The pricing function in this case will return the stock price, which is the limiting option price as the dividend yield goes to zero. Similarly, if the risk-free rate is zero, a perpetual put is never exercised. The pricing function will return the strike price in this case, which is the limiting value of the pricing function as the interest rate approaches zero.
Examples

```
s=40; k=40; v=0.30; r=0.08; d=0.02;
callperpetual(s, k, v, r, d)

putperpetual(s, c(35, 40, 45), v, r, d, showbarrier=TRUE)
```

---

**quincunx**  
*Quincunx simulation*

**Description**

*quincunx* simulates balls dropping down a pegboard with a 50% chance of bouncing right or left at each level. The balls accumulate in bins. If enough balls are dropped, the distribution approaches normality. This device is called a quincunx. See http://www.mathsisfun.com/data/quincunx.html

**Usage**

```
quincunx(n = 3, numballs = 20, delay = 0.1, probright = 0.5, plottrue = TRUE)
```

**Arguments**

- **n**  
  Integer The number of peg levels, default is 3
- **numballs**  
  Integer The number of balls dropped, default is 20
- **delay**  
  Numeric Number of seconds between ball drops. Set delay > 0 to see animation with delay seconds between dropped balls. If delay < 0, the simulation will run to completion without delays. If delay == 0, the user must hit <return> for the next ball to drop. The default is 0.1 second and can be set with the delay parameter.
- **probright**  
  Numeric The probability the ball bounces to the right; default is 0.5
- **plottrue**  
  Boolean If TRUE, the display will indicate bin levels if the distribution were normal. Default is TRUE

**Examples**

```
## These examples will not display correctly within RStudio unless
## the plot window is large
quincunx(delay=0)
quincunx(n=10, numballs=200, delay=0)
quincunx(n=20, numballs=200, delay=0, probright=0.7)
```
**simprice**

**Simulate asset prices**

**Description**

`simprice` computes simulated lognormal price paths, with or without jumps. Saves and restores random number seed.

`simprice(s0, v, r, tt, d, trials = 1, jump = FALSE, lambda = 0, alphaj = 0, vj = 0, seed = NULL, long = TRUE, scalar_v_is_stddev = TRUE)`

**Usage**

**Arguments**

- `s0` Initial price of the underlying asset
- `v` If scalar, default is volatility of the asset price, defined as the annualized standard deviation of the continuously-compounded return. The parameter `scalar_v_is_stddev` controls this behavior. If `v` is a square \( n \times n \) matrix, it is assumed to be the covariance matrix and `simprice` will return \( n \) simulated price series.
- `r` Annual continuously-compounded risk-free interest rate
- `tt` Time to maturity in years
- `d` Dividend yield, annualized, continuously-compounded
- `trials` number of simulated price paths
- `periods` number of equal-length periods in each simulated path
- `jump` boolean controlling use of jump parameters
- `lambda` expected number of jumps in one year (\( \lambda \times tt \)) is the Poisson parameter
- `alphaj` Expected continuously compounded jump percentage
- `vj` lognormal volatility of the jump amount
- `seed` random number seed
- `long` if TRUE, return a long-form dataframe with columns indicating the price, trial, and period. If FALSE, the returned data is wide, containing only prices: each row is a trial and each column is a period
- `scalar_v_is_stddev` if TRUE, scalar `v` is interpreted as the standard deviation; if FALSE, it is variance. Non-scalar `V` is always interpreted as a covariance matrix

**Value**

A dataframe with `trials` simulated stock price paths
Examples

# simple Monte Carlo option price example. Since there are two
# periods we can compute options prices for \code{tt} and
# \code{tt/2}
s0=40; k=40; v=0.30; r=0.08; tt=0.25; d=0;
st = simprice(s0, k, v, r, tt, d, trials=3, periods=2, jump=FALSE)
callprice1 = \exp(-r*tt/2) * \text{mean}(\text{pmax}(st[st$period==1,] - k, 0))
callprice2 = \exp(-r*tt) * \text{mean}(\text{pmax}(st[st$period==2,] - k, 0))
Index

*Topic datasets
  implied, 16
  .tol (implied), 16

arithasianmc, 2, 3, 4, 14
arithavgpricecv, 2, 3, 4, 14
asiangeomavg, 2, 3, 4, 14
assetcall (blksch), 9
assetdicall (barriers), 5
assetdiput (barriers), 5
assetdocall (barriers), 5
assetdoput (barriers), 5
assetjump (jumps), 17
assetput (blksch), 9
assetuicall (barriers), 5
assetuiput (barriers), 5
assetuocall (barriers), 5
assetuoput (barriers), 5

barriers, 5
binom, 7
binomial (binom), 7
binomopt (binom), 7
binomplot (binom), 7
binormsdist (compound), 12
blksch, 9
bondpv (bondsimple), 11
bondsimple, 11
bondyield (bondsimple), 11
bscall (blksch), 9
bscalllimps (implied), 16
bscallimpvol (implied), 16
bsopt (greeks), 14
bsput (blksch), 9
bsputimps (implied), 16
bsputimpvol (implied), 16

calldownin (barriers), 5
calldownout (barriers), 5
calloncall (compound), 12
callonput (compound), 12
callperpetual (perpetual), 19
callupin (barriers), 5
callupout (barriers), 5
cashcall (blksch), 9
cashdicall (barriers), 5
cashdiput (barriers), 5
cashdocall (barriers), 5
cashdoput (barriers), 5
cashjump (jumps), 17
cashput (blksch), 9
cashuicall (barriers), 5
cashuiput (barriers), 5
cashuocall (barriers), 5
cashuoput (barriers), 5
compound, 12
convexity (bondsimple), 11
dicall (barriers), 5
diput (barriers), 5
docall (barriers), 5
doput (barriers), 5
dr (barriers), 5
drdeferred (barriers), 5
duration (bondsimple), 11

gemmasianmc, 2–4, 13
gemavgprice (asiangemavg), 4
gemavgpricecall (asiangemavg), 4
gemavgpriceput (asiangemavg), 4
gemavgstrike (asiangemavg), 4
gemavgstrikecall (asiangemavg), 4
gemavgstrikeput (asiangemavg), 4
greeks, 14

greeks2 (greeks), 14
implied, 16

jumps, 17

mertonjump (jumps), 17
options on call (compound), 12
options on put (compound), 12
perpetual, 19
put down in (barriers), 5
put down out (barriers), 5
put on call (compound), 12
put on put (compound), 12
put perpetual (perpetual), 19
put up in (barriers), 5
put up out (barriers), 5
quincunx, 20
simprice, 21
ui call (barriers), 5
ui put (barriers), 5
uo call (barriers), 5
uo put (barriers), 5
ur (barriers), 5
ur deferred (barriers), 5