Package 'derivmkts'

August 5, 2015

Title functions and R code to accompany Derivatives Markets

Version 0.0.0.9000

Maintainer Robert McDonald <rmcd1024@gmail.com>

Description This package contains routines that should
be useful in teaching a course on financial derivatives.

Depends R (>= 3.0.0)

License GPL-2

LazyData true

R topics documented:

`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 ("drdeferred" and "urdeferred")

`bscall(s, k, v, r, tt, d)  = assetcall(s, k, v, r, tt, d) - k*cashcall(s, k, v, r, tt, d)`

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)
cashuicall(s, k, v, r, tt, d, H)
cashuiput(s, k, v, r, tt, d, H)
cashdicall(s, k, v, r, tt, d, H)
cashdiput(s, k, v, r, tt, d, H)
assetuicall(s, k, v, r, tt, d, H)
assetuiput(s, k, v, r, tt, d, H)
assetdicall(s, k, v, r, tt, d, H)
assetdiput(s, k, v, r, tt, d, H)
cashuocall(s, k, v, r, tt, d, H)
cashuoput(s, k, v, r, tt, d, H)
cashdocall(s, k, v, r, tt, d, H)
cashdoput(s, k, v, r, tt, d, H)
assetuocall(s, k, v, r, tt, d, H)
assetuoput(s, k, v, r, tt, d, H)
assetdocall(s, k, v, r, tt, d, H)
assetdoput(s, k, v, r, tt, d, H)
dr(s, v, r, tt, d, H)
ur(s, v, r, tt, d, H)
drdeferred(s, v, r, tt, d, H)
urdeferred(s, v, r, tt, d, H)
```

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

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)
```

---

blksch *Black-Scholes option pricing*

Description

bscall and bsput compute Black-Scholes call and put prices. The functions assetcall, assetput, cashcall, and cashput provide the prices of binary options that pay a 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

```
bscall(s, k, v, r, tt, d)  = assetcall(s, k, v, r, tt, d) - k*cashcall(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

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

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.

In other packages, for example fOption, 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:

```
bscall(s, k, v, r, tt, d)
```

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

Examples

```
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)
```

greeks *Calculate option Greeks*

Description

The functions greeks and greeks2 provide two different calling conventions for computing a full set of option Greeks. greeks requires the use of named list entries. greeks2 simply requires entering a pricing function with parameters. The function bsopt calls greeks 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 can not be used safely (that is, if parameter vector lengths are not integer multiples of one another.

Usage

```
bsopt(s, k, v, r, tt, d)
# must used named list entries:
greeks(fn, ...)
greeks2(f)
```

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
fn	Pricing function name, not in quotes
...	Pricing function inputs, may be named or not
f	Fully-specified option pricing function, including inputs (e.g., bscall(40, 40, .3, .08, .25, 0)), need not be named

Details

Numerical derivatives are calculated using a simple difference. This creates readily apparent problems in edge cases. A future version should make use of the package numDeriv or some other more sophisticated calculation.

Value

A named list of Black-Scholes option prices and Greeks.

Examples

```
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0;
greeks(bscall, list(s=s, k=k, v=v, r=r, tt=tt, d=d))
greeks2(bscall(s, k, v, r, tt, d))
bsopt(s, k, v, r, tt, d)

# plot Greeks for calls and puts for 500 different stock prices
k <- 100; v <- 0.30; r <- 0.08; tt <- 2; d <- 0
S <- seq(.5, 250, by=.5)
x <- bsopt(S, k, v, r, tt, d)
par(mfrow=c(4, 4))  ## create a 4x4 plot
for (i in c('Call', 'Put')) {
    for (j in rownames(x[[i]])) {  ## loop over greeks
        plot(S, x[[i]][j, ], main=paste(i, j), ylab=j, type='l')
    }
}
```

---

implied *Black-Scholes implied volatility and price*

Description

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

Usage

```
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

Format

```
num 1.49e-08
```

Details

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

Value

Implied volatility (for the "impvol" functions) or implied stock price (for the "impS") functions.

Note

Implied volatilties and stock prices do not exist if the price of the option exceeds no-arbitrage bounds. For example, if the interest rate is non-negative, a 40 strike put cannot have a price exceeding $40.

Examples

```
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0;
bscallimpvol(s, k, r, tt, d, 4)
bsputimpvol(s, k, r, tt, d, 4)
bscallimps(s, k, v, r, tt, d, 4)
bsputimps(s, k, v, r, tt, d, 4)
```

---

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. Default is 0.1 second.
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

```
quincunx()
quincunx(n=10, numballs=200, delay=0)
```

Index