Package 'derivmkts'

December 19, 2015

Title Functions and R Code to Accompany Derivatives Markets

Version 0.1.2.9000

License MIT + file LICENSE

Author 'Robert McDonald'

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.

Depends R (>= 3.0.0)

Suggests highlight,
markdown,
knitr,
rmarkdown

VignetteBuilder knitr

LazyData true

RoxygenNote 5.0.0

R topics documented:

`arithasianmc` *Asian Monte Carlo option pricing*

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`	Stock price
`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 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
`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: `arithavgpricecv`, `geomasianmc`, `geomavgprice`, `geomavgstrike`

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`	Stock price
`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 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
`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`, `geomasianmc`, `geomavgprice`, `geomavgstrike`

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)
```

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")

`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)
uocall(s, k, v, r, tt, d, H)
dicall(s, k, v, r, tt, d, H)
docall(s, k, v, r, tt, d, H)
uiput(s, k, v, r, tt, d, H)
uoput(s, k, v, r, tt, d, H)
diput(s, k, v, r, tt, d, H)
doput(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)
```

binom *Binomial option pricing*

Description

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

Usage

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

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
pointsize	CEX parameter for nodes
ylimval	c(low, high) for ylimit 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

Details

Returns an option price, a vector of the parameters used to compute the price. Optionally returns the following matrices, all but but two of which have dimensionality $(\text{nstep}+1) \times (\text{nstep}+1)$:

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 exercisd 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 $(\text{nstep}) \times (\text{nstep})$

bond at each node prior to expiration, the bond position in the replicating portfolio. The dimensionality is $(\text{nstep}) \times (\text{nstep})$

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

$$u = \exp((r-d) * h + \sigma\sqrt{h})$$

and

$$d = \exp((r - d) * h - \sigma\sqrt{h})$$

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

```
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; nstep=15

binomopt(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$oppricretree)
print(x$delta)
print(x$bond)

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 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.

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:

```
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)
```

`geomasianmc` *Geometric Asian option prices computed by Monte Carlo*

Description

Geometric average Asian option prices

Usage

```
geomasianmc(s, k, v, r, tt, d, m, numsim, printsds=FALSE)
```

Arguments

`s`	Stock price
`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 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
`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 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`, `geomavgprice`, `geomavgstrike`

Examples

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

`geomavgprice` *Geometric average price*

Description

Pricing functions for European Asian options based on geometric averages. `geomavgprice` and `geomavgstrike` compute analytical prices of geometric Asian options using the modified Black-Scholes formula.

Prices of geometric average-price call and put options

Usage

```
geomavgprice(s, k, v, r, tt, d, m, cont=FALSE)
```

Arguments

`s`	Stock price
`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 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
`m`	Number of prices in the average calculation
`cont`	Boolean which when TRUE denotes continuous averaging

Details

Geometric average asian options

Value

Call and put prices as a vector

See Also

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

Examples

```
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0; m=3;
geomavgprice(s, k, v, r, tt, d, m)
```

`geomavgstrike` *Geometric average-strike options*

Description

Prices of geometric average-strike call and put options

Usage

```
geomavgstrike(s, km, v, r, tt, d, m, cont=FALSE)
```

Arguments

`s`	Stock price
`km`	The strike mutiplier, 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 $$Payoff = max(ST - km/s * SAvg, 0)$$.
`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
`m`	Number of prices in the average calculation
`cont`	Boolean which when TRUE denotes continuous averaging

Value

Vector of call and put prices for geometric average strike options geomavgstrike(s, km, v, r, tt, d, m)

See Also

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

Examples

```
s=40; km=40; v=0.30; r=0.08; tt=0.25; d=0; m=3;
```

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 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)
greeks(f)
# must used named list entries:
greeks2(fn, ...)
```

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
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))
...	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.

Examples

```
s=40; k=40; v=0.30; r=0.08; tt=0.25; d=0;
greeks(bscall(s, k, v, r, tt, d))
greeks2(bscall, list(s=s, k=k, v=v, r=r, tt=tt, d=d))
greeks2(bscall, list(s=s, k=k, v=v, r=r, tt=tt, d=d))[c('Delta', 'Gamma'), ]
bsopt(s, k, v, r, tt, d)
bsopt(s, c(35, 40, 45), v, r, tt, d)
bsopt(s, c(35, 40, 45), v, r, tt, d)[['Call']][c('Delta', 'Gamma'), ]

## plot Greeks for calls and puts for 500 different stock prices
## Unfortunately, this plot will most likely not display in RStudio;
## it will generate a "figure margins too large" error
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

An object of class `numeric` of length 1.

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)
```

---

jumps *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)
cashjump(s, k, v, r, tt, d, lambda, alphaj, vj)
mertonjump(s, k, v, r, tt, d, lambda, alphaj, vj)
```

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
lambda	Poisson intensity: expected number of jumps per year
alphaj	Mean change in log price conditional on a jump
vj	Standard deviation of change in log price conditional on a jump

Details

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

Value

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

See Also

McDonald, Robert L., *Derivatives Markets*, 3rd Edition (2013) Chapter 24

bscall bsput

Examples

```
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)
bsput(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 <- 20:60
cp <- mertonjump(s, kseq, v, r, tt, d, lambda, alphaj,
    vj)[paste0('Call', 1:length(kseq))]

## 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))
```

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)
```

Index