Package ‘RND’

January 11, 2017

Type Package

Title Risk Neutral Density Extraction Package

Version 1.2

Date 2017-01-10

Author Kam Hamidieh <khamidieh@gmail.com>

Maintainer Kam Hamidieh <khamidieh@gmail.com>

Description Extract the implied risk neutral density from options using various methods.

Depends R (>= 3.0.1)

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2017-01-11 00:14:55

R topics documented:

`RND-package` *Risk Neutral Density Extraction Package*

Description

This package is a collection of various functions to extract the implied risk neutral density from option.

Details

Package:	RND
Type:	Package
Version:	1.2
Date:	2017-01-10
License:	GPL (>= 2)

Author(s)

Kam Hamidieh <khamidieh@gmail.com>

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
###
### You should see that all methods extract the same density!
###

r     = 0.05
te    = 60/365
s0    = 1000
sigma = 0.25
y     = 0.02

call.strikes.bsm   = seq(from = 500, to = 1500, by = 5)
market.calls.bsm   = price.bsm.option(r =r, te = te, s0 = s0,
                       k = call.strikes.bsm, sigma = sigma, y = y)$call

put.strikes.bsm    = seq(from = 500, to = 1500, by = 5)
market.puts.bsm    = price.bsm.option(r =r, te = te, s0 = s0,
                       k = put.strikes.bsm, sigma = sigma, y = y)$put

###
### See where your results will be outputted to...
###

getwd()


###
###  Running this may take a few minutes...
###
###  MOE(market.calls.bsm, call.strikes.bsm, market.puts.bsm,
###  put.strikes.bsm, s0, r , te, y, "bsm2")
###
```

---

approximate.max *Max Function Approximation*

Description

`approximate.max` gives a smooth approximation to the max function.

Usage

```
approximate.max(x, y, k = 5)
```

Arguments

x	the first argument for the max function
y	the second argument fot the max function

k — a tuning parameter. The larger this value, the closer the function output to a true max function.

Details

`approximate.max` approximates the max of x, and y as follows:

$$g(x,y) = \frac{1}{1+\exp(-k(x-y))}, \quad \max(x,y) \approx xg(x,y) + y(1-g(x,y))$$

Value

approximate maximum of x and y

Author(s)

Kam Hamidieh

References

Melick, W. R. and Thomas, C.P. (1997) Recovering an asset's implied pdf from option proces: An application to crude oil during the gulf crisis. *Journal of Financial and Quantitative Analysis*, 32(1), 91-115

Examples

```
#
# To see how the max function compares with approximate.max,
# run the following code.
#

i = seq(from = 0, to = 10, by = 0.25)
y = i - 5
max.values = pmax(0,y)
approximate.max.values = approximate.max(0,y,k=5)
matplot(i, cbind(max.values, approximate.max.values), lty = 1, type = "l",
 col=c("black","red"), main = "Max in Black, Approximate Max in Red")
```

---

bsm.objective *BSM Objective Function*

Description

`bsm.objective` is the objective function to be minimized in `extract.bsm.density`.

Usage

```
bsm.objective(s0, r, te, y, market.calls, call.strikes, call.weights = 1,
  market.puts, put.strikes, put.weights = 1, lambda = 1, theta)
```

Arguments

`s0`	current asset value
`r`	risk free rate
`te`	time to expiration
`y`	dividend yield
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`market.puts`	market calls (cheapest to most expensive)
`put.strikes`	strikes for the puts (smallest to largest)
`put.weights`	weights to be used for calls
`lambda`	Penalty parameter to enforce the martingale condition
`theta`	initial values for the optimization. This must be a vector of length 2: first component is μ, the lognormal mean of the underlying density, and the second component is $\sqrt{t}\sigma$ which is the time scaled volatility parameter of the underlying density.

Details

This function evaluates the weighted squared differences between the market option values and values predicted by the Black-Scholes-Merton option pricing formula.

Value

Objective function evalued at a specific set of values.

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
r     = 0.05
te    = 60/365
s0    = 1000
sigma = 0.25
y     = 0.01

call.strikes   = seq(from = 500, to = 1500, by = 25)
market.calls   = price.bsm.option(r =r, te = te, s0 = s0,
                 k = call.strikes, sigma = sigma, y = y)$call
```

```
put.strikes    = seq(from = 510, to = 1500, by = 25)
market.puts    = price.bsm.option(r =r, te = te, s0 = s0,
                 k = put.strikes, sigma = sigma, y = y)$put

###
### perfect initial values under BSM framework
###

mu.0     = log(s0) + ( r - y - 0.5 * sigma^2) * te
zeta.0   = sigma * sqrt(te)
mu.0
zeta.0

###
### The objective function should be *very* small
###

bsm.obj.val = bsm.objective(theta=c(mu.0, zeta.0), r = r, y=y, te = te, s0 = s0,
              market.calls = market.calls, call.strikes = call.strikes,
              market.puts  = market.puts,  put.strikes = put.strikes, lambda = 1)
bsm.obj.val
```

`compute.implied.volatility` *Compute Impied Volatility*

Description

`compute.implied.volatility` extracts the implied volatility for a call option.

Usage

```
compute.implied.volatility(r, te, s0, k, y, call.price, lower, upper)
```

Arguments

`r`	risk free rate
`te`	time to expiration
`s0`	current asset value
`k`	strike of the call option
`y`	dividend yield
`call.price`	call price
`lower`	lower bound of the implied volatility to look for
`upper`	upper bound of the implied volatility to look for

Details

The simple R `uniroot` function is used to extract the implied volatility.

Value

`sigma`	extratced implied volatility

Author(s)

Kam Hamidieh

References

J. Hull (2011) *Options, Futures, and Other Derivatives and DerivaGem Package* Prentice Hall, Englewood Cliffs, New Jersey, 8th Edition

R. L. McDonald (2013) *Derivatives Markets* Pearson, Upper Saddle River, New Jersey, 3rd Edition

Examples

```
#
# Create prices from BSM with various sigma's
#

r     =  0.05
y     =  0.02
te    =  60/365
s0    =  400

sigma.range = seq(from = 0.1, to = 0.8, by = 0.05)
k.range     = floor(seq(from = 300, to = 500, length.out = length(sigma.range)))
bsm.calls   = numeric(length(sigma.range))

for (i in 1:length(sigma.range))
{
  bsm.calls[i] = price.bsm.option(r = r, te = te, s0 = s0, k = k.range[i],
                                  sigma = sigma.range[i], y = y)$call
}
bsm.calls
k.range

#
# Computed implied sigma's should be very close to sigma.range.
#

compute.implied.volatility(r = r, te = te, s0 = s0, k = k.range, y = y,
                          call.price = bsm.calls, lower = 0.001, upper = 0.999)
sigma.range
```

dew *Edgeworth Density*

Description

dew is the probability density function implied by the Edgeworth expansion method.

Usage

```
dew(x, r, y, te, s0, sigma, skew, kurt)
```

Arguments

x	value at which the denisty is to be evaluated
r	risk free rate
y	dividend yield
te	time to expiration
s0	current asset value
sigma	volatility
skew	normalized skewness
kurt	normalized kurtosis

Details

This density function attempts to capture deviations from lognormal density by using Edgeworth expansions.

Value

density value at x

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

R. Jarrow and A. Rudd (1982) Approximate valuation for arbitrary stochastic processes. *Journal of Finanical Economics*, 10, 347-369

C.J. Corrado and T. Su (1996) S&P 500 index option tests of Jarrow and Rudd's approximate option valuation formula. *Journal of Futures Markets*, 6, 611-629

Examples

```
#
# Look at a true lognorma density & related dew
#
r       = 0.05
y       = 0.03
s0      = 1000
sigma   = 0.25
te      = 100/365
strikes = seq(from=600, to = 1400, by = 1)
v       = sqrt(exp(sigma^2 * te) - 1)
ln.skew = 3 * v + v^3
ln.kurt = 16 * v^2 + 15 * v^4 + 6 * v^6 + v^8

skew.4 = ln.skew * 1.50
kurt.4 = ln.kurt * 1.50

skew.5 = ln.skew * 0.50
kurt.5 = ln.kurt * 2.00

ew.density.4   = dew(x=strikes, r=r, y=y, te=te, s0=s0, sigma=sigma,
                       skew=skew.4, kurt=kurt.4)
ew.density.5   = dew(x=strikes, r=r, y=y, te=te, s0=s0, sigma=sigma,
                       skew=skew.5, kurt=kurt.5)
bsm.density    = dlnorm(x = strikes, meanlog = log(s0) + (r - y - (sigma^2)/2)*te,
                   sdlog = sigma*sqrt(te), log = FALSE)

matplot(strikes, cbind(bsm.density, ew.density.4, ew.density.5), type="l",
        lty=c(1,1,1), col=c("black","red","blue"),
        main="Black = BSM,  Red = EW 1.5 Times,  Blue = EW 0.50 & 2")
```

---

dgb *Generalized Beta Density*

Description

dgb is the probability density function of generalized beta distribution.

Usage

```
dgb(x, a, b, v, w)
```

Arguments

x	value at which the denisty is to be evaluated
a	power parameter > 0
b	scale paramter > 0
v	first beta paramter > 0
w	second beta parameter > 0

Details

Let B be a beta random variable with parameters v and w, then $Z = b(B/(1-B))^{1/a}$ is a generalized beta with parameters (a,b,v,w).

Value

density value at x

Author(s)

Kam Hamidieh

References

R.M. Bookstaber and J.B. McDonald (1987) A general distribution for describing security price returns. *Journal of Business*, 60, 401-424

X. Liu and M.B. Shackleton and S.J. Taylor and X. Xu (2007) Closed-form transformations from risk-neutral to real-world distributions *Journal of Business*, 60, 401-424

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# Just simple plot of the density
#

x = seq(from = 500, to = 1500, length.out = 10000)
a = 10
b = 1000
v = 3
w = 3
dx = dgb(x = x, a = a, b = b, v = v, w = w)
plot(dx ~ x, type="l")
```

---

dmln — *Density of Mixture Lognormal*

Description

`mln` is the probability density function of a mixture of two lognormal densities.

Usage

```
dmln(x, alpha.1, meanlog.1, meanlog.2, sdlog.1, sdlog.2)
```

Arguments

`x`	value at which the denisty is to be evaluated
`alpha.1`	proportion of the first lognormal. Second one is 1 - `alpha.1`
`meanlog.1`	mean of the log of the first lognormal
`meanlog.2`	mean of the log of the second lognormal
`sdlog.1`	standard deviation of the log of the first lognormal
`sdlog.2`	standard deviation of the log of the second lognormal

Details

mln is the density f(x) = alpha.1 * g(x) + (1 - alpha.1) * h(x), where g and h are densities of two lognormals with parameters (mean.log.1, sdlog.1) and (mean.log.2, sdlog.2) respectively.

Value

`out`	density value at x

Author(s)

Kam Hamidieh

References

B. Bahra (1996): Probability distribution of future asset prices implied by option prices. *Bank of England Quarterly Bulletin*, August 1996, 299-311

P. Soderlind and L.E.O. Svensson (1997) New techniques to extract market expectations from financial instruments. *Journal of Monetary Economics*, 40, 383-429

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# A bimodal risk neutral density!
#

mln.alpha.1   = 0.4
mln.meanlog.1 = 6.3
mln.meanlog.2 = 6.5
mln.sdlog.1   = 0.08
mln.sdlog.2   = 0.06

k  = 300:900
dx = dmln(x = k, alpha.1 = mln.alpha.1, meanlog.1 = mln.meanlog.1,
         meanlog.2 = mln.meanlog.2,
         sdlog.1 = mln.sdlog.1, sdlog.2 = mln.sdlog.2)
plot(dx ~ k, type="l")
```

`dmln.am` *Density of Mixture Lognormal for American Options*

Description

`mln.am` is the probability density function of a mixture of three lognormal densities.

Usage

```
dmln.am(x, u.1, u.2, u.3, sigma.1, sigma.2, sigma.3, p.1, p.2)
```

Arguments

`x`	value at which the denisty is to be evaluated
`u.1`	log mean of the first lognormal
`u.2`	log mean of the second lognormal
`u.3`	log mean of the third lognormal
`sigma.1`	log standard deviation of the first lognormal
`sigma.2`	log standard deviation of the second lognormal
`sigma.3`	log standard deviation of the third lognormal
`p.1`	weight assigned to the first density
`p.2`	weight assigned to the second density

Details

mln is density f(x) = p.1 * f1(x) + p.2 * f2(x) + (1 - p.1 - p.2) * f3(x), where f1, f2, and f3 are lognormal densities with log means u.1,u.2, and u.3 and standard deviations sigma.1, sigma.2, and sigma.3 respectively.

Value

`out`	density value at x

Author(s)

Kam Hamidieh

References

Melick, W. R. and Thomas, C. P. (1997). Recovering an asset's implied pdf from option prices: An application to crude oil during the gulf crisis. *Journal of Financial and Quantitative Analysis*, 32(1), 91-115.

Examples

```
###
### Just look at a generic density and see if it integrates to 1.
###

u.1     = 4.2
u.2     = 4.5
u.3     = 4.8
sigma.1 = 0.30
sigma.2 = 0.20
sigma.3 = 0.15
p.1     = 0.25
p.2     = 0.45
x = seq(from = 0, to = 250, by = 0.01)
y = dmln.am(x = x, u.1 = u.1, u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
            sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)

plot(y ~ x, type="l")
sum(y * 0.01)

###
### Yes, the sum is near 1.
###
```

---

dshimko *Density Implied by Shimko Method*

Description

`dshimko` is the probability density function implied by the Shimko method.

Usage

```
dshimko(r, te, s0, k, y, a0, a1, a2)
```

Arguments

r	risk free rate
te	time to expiration
s0	current asset value
k	strike at which volatility to be computed
y	dividend yield
a0	constant term in the quadratic polynomial
a1	coefficient term of k in the quadratic polynomial
a2	coefficient term of k squared in the quadratic polynomial

Details

The implied volatility is modeled as: $\sigma(k) = a_0 + a_1 k + a_2 k^2$

Value

density value at x

Author(s)

Kam Hamidieh

References

D. Shimko (1993) Bounds of probability. *Risk*, 6, 33-47

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# a0, a1, a2 values come from Shimko's paper.
#

r     =  0.05
y     =  0.02
a0    =  0.892
a1    = -0.00387
a2    =  0.00000445
te    =  60/365
s0    =  400
k     =  seq(from = 250, to = 500, by = 1)
sigma =  0.15

#
# Does it look like a proper density and intergate to one?
#

dx = dshimko(r = r, te = te, s0 = s0, k = k, y = y, a0 = a0, a1 = a1, a2 = a2)
plot(dx ~ k, type="l")

#
# sum(dx) should be about 1 since dx is a density.
#

sum(dx)
```

ew.objective — *Edgeworth Exapnsion Objective Function*

Description

`ew.objective` is the objective function to be minimized in `ew.extraction`.

Usage

```
ew.objective(theta, r, y, te, s0, market.calls, call.strikes, call.weights = 1,
  lambda = 1)
```

Arguments

`theta`	initial values for the optimization
`r`	risk free rate
`y`	dividend yield
`te`	time to expiration
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`lambda`	Penalty parameter to enforce the martingale condition

Details

This function evaluates the weighted squared differences between the market option values and values predicted by Edgworth based expansion of the risk neutral density.

Value

Objective function evalued at a specific set of values

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

R. Jarrow and A. Rudd (1982) Approximate valuation for arbitrary stochastic processes. *Journal of Finanical Economics*, 10, 347-369

C.J. Corrado and T. Su (1996) S&P 500 index option tests of Jarrow and Rudd's approximate option valuation formula. *Journal of Futures Markets*, 6, 611-629

Examples

```
r        = 0.05
y        = 0.03
s0       = 1000
sigma    = 0.25
te       = 100/365
k        = seq(from=800, to = 1200, by = 50)
v        = sqrt(exp(sigma^2 * te) - 1)
ln.skew = 3 * v + v^3
ln.kurt = 16 * v^2 + 15 * v^4 + 6 * v^6 + v^8

#
# The objective function should be close to zero.
# Also the weights are automatically set to 1.
#

market.calls.bsm = price.bsm.option(r = r, te = te, s0 = s0, k=k,
                     sigma=sigma, y=y)$call
ew.objective(theta = c(sigma, ln.skew, ln.kurt), r = r, y = y, te = te, s0=s0,
             market.calls = market.calls.bsm, call.strikes = k, lambda = 1)
```

`extract.am.density` *Mixture of Lognormal Extraction for American Options*

Description

`extract.am.density` extracts the mixture of three lognormals from American options.

Usage

```
extract.am.density(initial.values = rep(NA, 10), r, te, s0, market.calls,
  call.weights = NA, market.puts, put.weights = NA, strikes, lambda = 1,
  hessian.flag = F, cl = list(maxit = 10000))
```

Arguments

`initial.values`	initial values for the optimization
`r`	risk free rate
`te`	time to expiration
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.weights`	weights to be used for calls. Set to 1 by default.
`market.puts`	market calls (cheapest to most expensive)

`put.weights`	weights to be used for puts. Set to 1 by default.
`strikes`	strikes (smallest to largest)
`lambda`	Penalty parameter to enforce the martingale condition
`hessian.flag`	If FALSE then no Hessian is produced
`cl`	List of parameter values to be passed to the optimization function

Details

The extracted density is in the form of f(x) = p.1 * f1(x) + p.2 * f2(x) + (1 - p.1 - p.2) * f3(x), where f1, f2, and f3 are lognormal densities with log means u.1,u.2, and u.3 and standard deviations sigma.1, sigma.2, and sigma.3 respectively.

For the description of w.1 and w.2 see equations (7) & (8) of Melick and Thomas paper below.

Value

`w.1`	First weight, a number between 0 and 1, to weigh the option price bounds
`w.2`	Second weight, a number between 0 and 1, to weigh the option price bounds
`u.1`	log mean of the first lognormal
`u.2`	log mean of the second lognormal
`u.3`	log mean of the third lognormal
`sigma.1`	log sd of the first lognormal
`sigma.2`	log sd of the second lognormal
`sigma.3`	log sd of the third lognormal
`p.1`	weight assigned to the first density
`p.2`	weight assigned to the second density
`converge.result`	Captures the convergence result
`hessian`	Hessian Matrix

Author(s)

Kam Hamidieh

References

Melick, W. R. and Thomas, C. P. (1997). Recovering an asset's implied pdf from option prices: An application to crude oil during the gulf crisis. *Journal of Financial and Quantitative Analysis*, 32(1), 91-115.

Examples

```
###
### Try with synthetic data first.
###

r       = 0.01
te      = 60/365
w.1     = 0.4
w.2     = 0.25
u.1     = 4.2
u.2     = 4.5
u.3     = 4.8
sigma.1 = 0.30
sigma.2 = 0.20
sigma.3 = 0.15
p.1     = 0.25
p.2     = 0.45
theta   = c(w.1,w.2,u.1,u.2,u.3,sigma.1,sigma.2,sigma.3,p.1,p.2)
p.3     = 1 - p.1 - p.2

###
### Generate some synthetic American calls & puts
###

expected.f0   =  sum(c(p.1, p.2, p.3) * exp(c(u.1,u.2,u.3) +
                    (c(sigma.1, sigma.2, sigma.3)^2)/2) )
expected.f0

strikes = 50:150

market.calls = numeric(length(strikes))
market.puts  = numeric(length(strikes))

for (i in 1:length(strikes))
{

  if ( strikes[i] < expected.f0) {
    market.calls[i] = price.am.option(k = strikes[i], r = r, te = te, w = w.1, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$call.value

    market.puts[i]  = price.am.option(k = strikes[i], r = r, te = te, w = w.2, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$put.value
  }  else {

    market.calls[i] = price.am.option(k = strikes[i], r = r, te = te, w = w.2, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$call.value

    market.puts[i]  = price.am.option(k = strikes[i], r = r, te = te, w = w.1, u.1 = u.1,
```

```
                     u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                     sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$put.value
     }

}


###
### ** IMPORTANT **:  The code that follows may take 1-2 minutes.
###                   Copy and paste onto R console the commands
###                   that follow the greater sign >.
###
### Try the optimization with exact inital values.
### They should be close the actual initials.
###
#
# > s0      = expected.f0 * exp(-r * te)
# > s0
#
# > extract.am.density(initial.values = theta, r = r, te = te, s0 = s0,
#              market.calls = market.calls, market.puts = market.puts, strikes = strikes,
#                 lambda = 1, hessian.flag = FALSE)
#
# > theta
#
###
### Now try without our the correct initial values...
###
#
# > optim.obj.no.init = extract.am.density( r = r, te = te, s0 = s0,
#               market.calls = market.calls, market.puts = market.puts, strikes = strikes,
#                    lambda = 1, hessian.flag = FALSE)
#
# > optim.obj.no.init
# > theta
#
###
### We do get different values but how do the densities look like?
###
#
###
### plot the two densities side by side
###
#
# > x = 10:190
#
# > y.1 = dmln.am(x = x, p.1 = theta[9], p.2 = theta[10],
#          u.1 = theta[3], u.2 = theta[4], u.3 = theta[5],
#         sigma.1 = theta[6], sigma.2 = theta[7], sigma.3 = theta[8] )
#
# > o = optim.obj.no.init
#
# > y.2 = dmln.am(x = x, p.1 = o$p.1, p.2 = o$p.2,
```

```
#           u.1 = o$u.1, u.2 = o$u.2, u.3 = o$u.3,
#           sigma.1 = o$sigma.1, sigma.2 = o$sigma.2, sigma.3 = o$sigma.3 )
#
# > matplot(x, cbind(y.1,y.2), main = "Exact = Black, Approx = Red", type="l", lty=1)
#
###
### Densities are close.
###
```

---

extract.bsm.density   *Extract Lognormal Density*

Description

bsm.extraction extracts the parameters of the lognormal density as implied by the BSM model.

Usage

```
extract.bsm.density(initial.values = c(NA, NA), r, y, te, s0, market.calls,
  call.strikes, call.weights = 1, market.puts, put.strikes, put.weights = 1,
  lambda = 1, hessian.flag = F, cl = list(maxit = 10000))
```

Arguments

initial.values	initial values for the optimization
r	risk free rate
y	dividend yield
te	time to expiration
s0	current asset value
market.calls	market calls (most expensive to cheapest)
call.strikes	strikes for the calls (smallest to largest)
call.weights	weights to be used for calls
market.puts	market calls (cheapest to most expensive)
put.strikes	strikes for the puts (smallest to largest)
put.weights	weights to be used for puts
lambda	Penalty parameter to enforce the martingale condition
hessian.flag	if F, no hessian is produced
cl	list of parameter values to be passed to the optimization function

Details

If initial.values are not specified then the function will attempt to pick them automatically. cl is a list that can be used to pass parameters to the optim function.

Value

Let S_T with the lognormal random variable of the risk neutral density.

- `mu` mean of log(S_T)
- `zeta` sd of log(S_T)
- `converge.result` Did the result converge?
- `hessian` Hessian matrix

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

J. Hull (2011) *Options, Futures, and Other Derivatives and DerivaGem Package* Prentice Hall, Englewood Cliffs, New Jersey, 8th Edition

R. L. McDonald (2013) *Derivatives Markets* Pearson, Upper Saddle River, New Jersey, 3rd Edition

Examples

```
#
# Create some BSM Based options
#

r     = 0.05
te    = 60/365
s0    = 1000
sigma = 0.25
y     = 0.01

call.strikes   = seq(from = 500, to = 1500, by = 25)
market.calls   = price.bsm.option(r =r, te = te, s0 = s0,
                 k = call.strikes, sigma = sigma, y = y)$call

put.strikes    = seq(from = 510, to = 1500, by = 25)
market.puts    = price.bsm.option(r =r, te = te, s0 = s0,
                 k = put.strikes, sigma = sigma, y = y)$put

#
#  Get extract the parameter of the density
#

extract.bsm.density(r = r, y = y, te = te, s0 = s0, market.calls = market.calls,
               call.strikes = call.strikes,  market.puts = market.puts,
               put.strikes = put.strikes, lambda = 1, hessian.flag = FALSE)

#
```

```
# The extracted parameters should be close to these actual values:
#
actual.mu     = log(s0) + ( r - y - 0.5 * sigma^2) * te
actual.zeta   = sigma * sqrt(te)
actual.mu
actual.zeta
```

`extract.ew.density` *Extract Edgeworth Based Density*

Description

`ew.extraction` extracts the parameters for the density approximated by the Edgeworth expansion method.

Usage

```
extract.ew.density(initial.values = c(NA, NA, NA), r, y, te, s0, market.calls,
  call.strikes, call.weights = 1, lambda = 1, hessian.flag = F,
  cl = list(maxit = 10000))
```

Arguments

`initial.values`	initial values for the optimization
`r`	risk free rate
`y`	dividend yield
`te`	time to expiration
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`lambda`	Penalty parameter to enforce the martingale condition
`hessian.flag`	if F, no hessian is produced
`cl`	list of parameter values to be passed to the optimization function

Details

If `initial.values` are not specified then the function will attempt to pick them automatically. `cl` in form of a list can be used to pass parameters to the `optim` function.

Value

sigma	volatility of the underlying lognormal
skew	normalized skewness
kurt	normalized kurtosis
converge.result	Did the result converge?
hessian	Hessian matrix

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

R. Jarrow and A. Rudd (1982) Approximate valuation for arbitrary stochastic processes. *Journal of Finanical Economics*, 10, 347-369

C.J. Corrado and T. Su (1996) S&P 500 index option tests of Jarrow and Rudd's approximate option valuation formula. *Journal of Futures Markets*, 6, 611-629

Examples

```
#
#  ln.skew & ln.kurt are the normalized skewness and kurtosis of a true lognormal.
#

r       = 0.05
y       = 0.03
s0      = 1000
sigma   = 0.25
te      = 100/365
strikes = seq(from=600, to = 1400, by = 1)
v       = sqrt(exp(sigma^2 * te) - 1)
ln.skew = 3 * v + v^3
ln.kurt = 16 * v^2 + 15 * v^4 + 6 * v^6 + v^8

#
# Now "perturb" the lognormal
#

new.skew = ln.skew * 1.50
new.kurt = ln.kurt * 1.50

#
# new.skew & new.kurt should not be extracted.
# Note that weights are automatically set to 1.
#
```

```
market.calls      =  price.ew.option(r = r, te = te, s0 = s0, k=strikes, sigma=sigma,
                     y=y, skew = new.skew, kurt = new.kurt)$call
ew.extracted.obj  =  extract.ew.density(r = r, y = y, te = te, s0 = s0,
                     market.calls = market.calls, call.strikes = strikes,
                     lambda = 1, hessian.flag = FALSE)
ew.extracted.obj
```

`extract.gb.density` *Generalized Beta Extraction*

Description

`extract.gb.density` extracts the generalized beta density from market options.

Usage

```
extract.gb.density(initial.values = c(NA, NA, NA, NA), r, te, y, s0, market.calls,
  call.strikes, call.weights = 1, market.puts, put.strikes, put.weights = 1,
  lambda = 1, hessian.flag = F, cl = list(maxit = 10000))
```

Arguments

`initial.values`	initial values for the optimization
`r`	risk free rate
`te`	time to expiration
`y`	dividend yield
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`market.puts`	market calls (cheapest to most expensive)
`put.strikes`	strikes for the puts (smallest to largest)
`put.weights`	weights to be used for puts
`lambda`	Penalty parameter to enforce the martingale condition
`hessian.flag`	if F, no hessian is produced
`cl`	list of parameter values to be passed to the optimization function

Details

This function extracts the generalized beta density implied by the options.

Value

a	extracted power parameter
b	extracted scale paramter
v	extracted first beta paramter
w	extracted second beta parameter
converge.result	Did the result converge?
hessian	Hessian matrix

Author(s)

Kam Hamidieh

References

R.M. Bookstaber and J.B. McDonald (1987) A general distribution for describing security price returns. *Journal of Business*, 60, 401-424

X. Liu and M.B. Shackleton and S.J. Taylor and X. Xu (2007) Closed-form transformations from risk-neutral to real-world distributions *Journal of Business*, 60, 401-424

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# create some GB based calls and puts
#

r  = 0.03
te = 50/365
k  = seq(from = 800, to = 1200, by = 10)
a  = 10
b  = 1000
v  = 2.85
w  = 2.85
y  = 0.01
s0 = exp((y-r)*te) * b * beta(v + 1/a, w - 1/a)/beta(v,w)
s0

call.strikes = seq(from = 800, to = 1200, by = 10)
market.calls = price.gb.option(r = r, te = te, y = y, s0 = s0,
                               k = call.strikes, a = a, b = s0, v = v, w = w)$call

put.strikes  = seq(from = 805, to = 1200, by = 10)
market.puts  = price.gb.option(r = r, te = te, y = y, s0 = s0,
                               k = put.strikes, a = a, b = s0, v = v, w = w)$put
```

```
#
# Extraction...should match the a,b,v,w above. You will also get warning messages.
# Weigths are automatically set to 1.
#

extract.gb.density(r=r, te=te, y = y, s0=s0, market.calls = market.calls,
                call.strikes = call.strikes, market.puts = market.puts,
                put.strikes = put.strikes, lambda = 1, hessian.flag = FALSE)
```

`extract.mln.density` *Extract Mixture of Lognormal Densities*

Description

`mln.extraction` extracts the parameters of the mixture of two lognormals densities.

Usage

```
extract.mln.density(initial.values = c(NA, NA, NA, NA, NA), r, y, te, s0,
  market.calls, call.strikes, call.weights = 1, market.puts, put.strikes,
  put.weights = 1, lambda = 1, hessian.flag = F, cl = list(maxit = 10000))
```

Arguments

`initial.values`	initial values for the optimization
`r`	risk free rate
`y`	dividend yield
`te`	time to expiration
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`market.puts`	market calls (cheapest to most expensive)
`put.strikes`	strikes for the puts (smallest to largest)
`put.weights`	weights to be used for puts
`lambda`	Penalty parameter to enforce the martingale condition
`hessian.flag`	if F, no hessian is produced
`cl`	list of parameter values to be passed to the optimization function

Details

mln is the density f(x) = alpha.1 * g(x) + (1 - alpha.1) * h(x), where g and h are densities of two lognormals with parameters (mean.log.1, sdlog.1) and (mean.log.2, sdlog.2) respectively.

Value

`alpha.1`	extracted proportion of the first lognormal. Second one is 1 - `alpha.1`
`meanlog.1`	extracted mean of the log of the first lognormal
`meanlog.2`	extracted mean of the log of the second lognormal
`sdlog.1`	extracted standard deviation of the log of the first lognormal
`sdlog.2`	extracted standard deviation of the log of the second lognormal
`converge.result`	Did the result converge?
`hessian`	Hessian matrix

Author(s)

Kam Hamidieh

References

F. Gianluca and A. Roncoroni (2008) *Implementing Models in Quantitative Finance: Methods and Cases*

B. Bahra (1996): Probability distribution of future asset prices implied by option prices. *Bank of England Quarterly Bulletin*, August 1996, 299-311

P. Soderlind and L.E.O. Svensson (1997) New techniques to extract market expectations from financial instruments. *Journal of Monetary Economics*, 40, 383-4

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# Create some calls and puts based on mln and
# see if we can extract the correct values.
#


r         = 0.05
y         = 0.02
te        = 60/365
meanlog.1 = 6.8
meanlog.2 = 6.95
sdlog.1   = 0.065
sdlog.2   = 0.055
alpha.1   = 0.4


call.strikes = seq(from = 800, to = 1200, by = 10)
market.calls = price.mln.option(r = r, y = y, te = te, k = call.strikes,
               alpha.1 = alpha.1, meanlog.1 = meanlog.1, meanlog.2 = meanlog.2,
                                sdlog.1 = sdlog.1, sdlog.2 = sdlog.2)$call
```

```
s0 = price.mln.option(r = r, y = y, te = te, k = call.strikes, alpha.1 = alpha.1,
                      meanlog.1 = meanlog.1, meanlog.2 = meanlog.2,
                      sdlog.1 = sdlog.1, sdlog.2 = sdlog.2)$s0
s0
put.strikes  = seq(from = 805, to = 1200, by = 10)
market.puts  = price.mln.option(r = r, y = y, te = te, k = put.strikes,
                                alpha.1 = alpha.1, meanlog.1 = meanlog.1,
                                meanlog.2 = meanlog.2, sdlog.1 = sdlog.1,
                                sdlog.2 = sdlog.2)$put

###
### The extracted values should be close to the actual values.
###

extract.mln.density(r = r, y = y, te = te, s0 = s0, market.calls = market.calls,
               call.strikes = call.strikes, market.puts = market.puts,
               put.strikes = put.strikes, lambda = 1, hessian.flag = FALSE)
```

`extract.rates` *Extract Risk Free Rate and Dividend Yield*

Description

`extract.rates` extracts the risk free rate and the dividend yield from European options.

Usage

```
extract.rates(calls, puts, s0, k, te)
```

Arguments

`calls`	market calls (most expensive to cheapest)
`puts`	market puts (cheapest to most expensive)
`s0`	current asset value
`k`	strikes for the calls (smallest to largest)
`te`	time to expiration

Details

The extraction is based on the put-call parity of the European options. Shimko (1993) - see below - shows that the slope and intercept of the regression of the calls minus puts onto the strikes contains the risk free and the dividend rates.

Value

`risk.free.rate`
: extracted risk free rate

`dividend.yield`
: extracted dividend rate

Author(s)

Kam Hamidieh

References

D. Shimko (1993) Bounds of probability. *Risk*, 6, 33-47

Examples

```
#
# Create calls and puts based on BSM
#

r     = 0.05
te    = 60/365
s0    = 1000
k     = seq(from = 900, to = 1100, by = 25)
sigma = 0.25
y     = 0.01

bsm.obj = price.bsm.option(r =r, te = te, s0 = s0, k = k, sigma = sigma, y = y)

calls = bsm.obj$call
puts  = bsm.obj$put

#
# Extract rates should give the values of r and y above:
#

rates = extract.rates(calls = calls, puts = puts, k = k, s0 = s0, te = te)
rates
```

---

extract.shimko.density *Extract Risk Neutral Density based on Shimko's Method*

Description

`shimko.extraction` extracts the implied risk neutral density based on modeling the volatility as a quadratic function of the strikes.

Usage

```
extract.shimko.density(market.calls, call.strikes, r, y, te, s0, lower, upper)
```

Arguments

`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`r`	risk free rate
`y`	dividend yield
`te`	time to expiration
`s0`	current asset value
`lower`	lower bound for the search of implied volatility
`upper`	upper bound for the search of implied volatility

Details

The correct values for range of search must be specified.

Value

`implied.curve.obj`
: variable that holds a0, a1, and a2 which are the constant terms of the quadratic polynomial

`shimko.density`
: density evaluated at the strikes

`implied.volatilities`
: implied volatilities at each `call.strike`

Author(s)

Kam Hamidieh

References

D. Shimko (1993) Bounds of probability. *Risk*, 6, 33-47

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
#  Test the function shimko.extraction.  If BSM holds then a1 = a2 = 0.
#

r       =  0.05
y       =  0.02
```

```
te      =  60/365
s0      =  1000
k       =  seq(from = 800, to = 1200, by = 5)
sigma   =  0.25

bsm.calls = price.bsm.option(r = r, te = te, s0 = s0, k = k,
                             sigma = sigma, y = y)$call
extract.shimko.density(market.calls = bsm.calls, call.strikes = k, r = r, y = y, te = te,
                  s0 = s0, lower = -10, upper = 10)

#
# Note: a0 is about equal to sigma, and a1 and a2 are close to zero.
#
```

`fit.implied.volatility.curve`

Fit Implied Quadratic Volatility Curve

Description

`fit.implied.volatility.curve` estimates the coefficients of the quadratic equation for the implied volatilities.

Usage

```
fit.implied.volatility.curve(x, k)
```

Arguments

`x`	a set of implied volatilities
`k`	range of strikes

Details

This function estimates volatility σ as a quadratic function of strike k with the coefficents a_0, a_1, a_2: $\sigma(k) = a_0 + a_1 k + a_2 k^2$

Value

`a0`	constant term in the quadratic ploynomial
`a1`	coefficient term of k in the quadratic ploynomial
`a2`	coefficient term of k squared in the quadratic polynomial
`summary.obj`	statistical summary of the fit

Author(s)

Kam Hamidieh

References

D. Shimko (1993) Bounds of probability. *Risk*, 6, 33-47

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# Suppose we see the following implied volatilities and strikes:
#

implied.sigma = c(0.11, 0.08, 0.065, 0.06, 0.05)
strikes       = c(340, 360, 380, 400, 410)
tmp           = fit.implied.volatility.curve(x = implied.sigma, k = strikes)
tmp

strike.range = 340:410
plot(implied.sigma ~ strikes)
lines(strike.range, tmp$a0 + tmp$a1 * strike.range + tmp$a2 * strike.range^2)
```

---

gb.objective *Generalized Beta Objective*

Description

`gb.objective` is the objective function to be minimized in `extract.gb.density`.

Usage

```
gb.objective(theta, r, te, y, s0, market.calls, call.strikes, call.weights = 1,
  market.puts, put.strikes, put.weights = 1, lambda = 1)
```

Arguments

`theta`	initial values for optimization
`r`	risk free rate
`te`	time to expiration
`y`	dividend yield
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`market.puts`	market calls (cheapest to most expensive)

put.strikes	strikes for the puts (smallest to largest)
put.weights	weights to be used for puts
lambda	Penalty parameter to enforce the martingale condition

Details

This is the function minimized by `extract.gb.desnity` function.

Value

obj	value of the objective function

Author(s)

Kam Hamidieh

References

R.M. Bookstaber and J.B. McDonald (1987) A general distribution for describing security price returns. *Journal of Business*, 60, 401-424

X. Liu and M.B. Shackleton and S.J. Taylor and X. Xu (2007) Closed-form transformations from risk-neutral to real-world distributions *Journal of Business*, 60, 401-424

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# The objective should be very small!
# Note the weights are automatically
# set to 1.
#

r  = 0.03
te = 50/365
k  = seq(from = 800, to = 1200, by = 10)
a  = 10
b  = 1000
v  = 2.85
w  = 2.85
y  = 0.01
s0 = exp((y-r)*te) * b * beta(v + 1/a, w - 1/a)/beta(v,w)
s0

call.strikes = seq(from = 800, to = 1200, by = 10)
market.calls = price.gb.option(r = r, te = te, s0 = s0, y = y,
                        k = call.strikes, a = a, b = b, v = v, w = w)$call

put.strikes = seq(from = 805, to = 1200, by = 10)
```

```
market.puts = price.gb.option(r = r, te = te, s0 = s0, y = y,
                        k = put.strikes, a = a, b = b, v = v, w = w)$put

gb.objective(theta=c(a,b,v,w),r = r, te = te, y = y, s0 = s0,
             market.calls = market.calls, call.strikes = call.strikes,
             market.puts = market.puts, put.strikes = put.strikes, lambda = 1)
```

`get.point.estimate` *Point Estimation of the Density*

Description

`get.point.estimate` estimates the risk neutral density by center differentiation.

Usage

```
get.point.estimate(market.calls, call.strikes, r, te)
```

Arguments

`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`r`	risk free rate
`te`	time to expiration

Details

This is a non-parametric estimate of the risk neutral density. Due to center differentiation, the density values are not estimated at the highest and lowest strikes.

Value

`point.estimates`
values of the estimated density at each strike

Author(s)

Kam Hamidieh

References

J. Hull (2011) *Options, Futures, and Other Derivatives and DerivaGem Package* Prentice Hall, Englewood Cliffs, New Jersey, 8th Edition

Examples

```
###
### Recover the lognormal density based on BSM
###

r     = 0.05
te    = 60/365
s0    = 1000
k     = seq(from = 500, to = 1500, by = 1)
sigma = 0.25
y     = 0.01


bsm.calls = price.bsm.option(r =r, te = te, s0 = s0, k = k, sigma = sigma, y = y)$call
density.est = get.point.estimate(market.calls = bsm.calls,
              call.strikes = k, r = r , te = te)

len = length(k)-1
### Note, estimates at two data points (smallest and largest strikes) are lost
plot(density.est ~ k[2:len], type = "l")
```

`mln.am.objective` *Objective function for the Mixture of Lognormal of American Options*

Description

`mln.am.objective` is the objective function to be minimized in `extract.am.density`.

Usage

```
mln.am.objective(theta, s0, r, te, market.calls, call.weights = NA, market.puts,
  put.weights = NA, strikes, lambda = 1)
```

Arguments

`theta`	initial values for the optimization
`s0`	current asset value
`r`	risk free rate
`te`	time to expiration
`market.calls`	market calls (most expensive to cheapest)
`call.weights`	weights to be used for calls
`market.puts`	market calls (cheapest to most expensive)
`put.weights`	weights to be used for calls
`strikes`	strikes for the calls (smallest to largest)
`lambda`	Penalty parameter to enforce the martingale condition

Details

mln is density f(x) = p.1 * f1(x) + p.2 * f2(x) + (1 - p.1 - p.2) * f3(x), where f1, f2, and f3 are lognormal densities with log means u.1,u.2, and u.3 and standard deviations sigma.1, sigma.2, and sigma.3 respectively.

Value

`obj`	Value of the objective function

Author(s)

Kam Hamidieh

References

Melick, W. R. and Thomas, C. P. (1997). Recovering an asset's implied pdf from option prices: An application to crude oil during the gulf crisis. *Journal of Financial and Quantitative Analysis*, 32(1), 91-115.

Examples

```
r       = 0.01
te      = 60/365
w.1     = 0.4
w.2     = 0.25
u.1     = 4.2
u.2     = 4.5
u.3     = 4.8
sigma.1 = 0.30
sigma.2 = 0.20
sigma.3 = 0.15
p.1     = 0.25
p.2     = 0.45
theta   = c(w.1,w.2,u.1,u.2,u.3,sigma.1,sigma.2,sigma.3,p.1,p.2)

p.3 = 1 - p.1 - p.2
p.3
expected.f0   =  sum(c(p.1, p.2, p.3) * exp(c(u.1,u.2,u.3) +
                    (c(sigma.1, sigma.2, sigma.3)^2)/2) )
expected.f0

strikes = 30:170

market.calls = numeric(length(strikes))
market.puts  = numeric(length(strikes))

for (i in 1:length(strikes))
{

  if ( strikes[i] < expected.f0) {
```

```
    market.calls[i] = price.am.option(k = strikes[i], r = r, te = te, w = w.1, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$call.value

    market.puts[i]  = price.am.option(k = strikes[i], r = r, te = te, w = w.2, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$put.value
  }  else {

    market.calls[i] = price.am.option(k = strikes[i], r = r, te = te, w = w.2, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$call.value

    market.puts[i]  = price.am.option(k = strikes[i], r = r, te = te, w = w.1, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$put.value
     }

}

###
### Quickly look at the option values...
###

par(mfrow=c(1,2))
plot(market.calls ~ strikes, type="l")
plot(market.puts  ~ strikes, type="l")
par(mfrow=c(1,1))

###
### ** IMPORTANT **:  The code that follows may take a few seconds.
###                   Copy and paste onto R console the commands
###                   that follow the greater sign >.
###
###
### Next try the objective function.  It should be zero.
### Note: Let weights be the defaults values of 1.
###
#
# > s0      = expected.f0 * exp(-r * te)
# > s0
#
# > mln.am.objective(theta, s0 =s0, r = r, te = te, market.calls = market.calls,
#                  market.puts = market.puts, strikes = strikes, lambda = 1)
#
###
### Now directly try the optimization with perfect initial values.
###
#
#
# > optim.obj.with.synthetic.data = optim(theta, mln.am.objective, s0 = s0, r=r, te=te,
#               market.calls = market.calls, market.puts = market.puts, strikes = strikes,
#                  lambda = 1, hessian = FALSE , control=list(maxit=10000) )
```

```
#
# > optim.obj.with.synthetic.data
#
# > theta
#
###
### It does take a few seconds but the optim converges to exact theta values.
###
```

---

mln.objective *Objective function for the Mixture of Lognormal*

Description

`mln.objective` is the objective function to be minimized in `extract.mln.density`.

Usage

```
mln.objective(theta, r, y, te, s0, market.calls, call.strikes, call.weights,
  market.puts, put.strikes, put.weights, lambda = 1)
```

Arguments

`theta`	initial values for the optimization
`r`	risk free rate
`y`	dividend yield
`te`	time to expiration
`s0`	current asset value
`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`call.weights`	weights to be used for calls
`market.puts`	market calls (cheapest to most expensive)
`put.strikes`	strikes for the puts (smallest to largest)
`put.weights`	weights to be used for puts
`lambda`	Penalty parameter to enforce the martingale condition

Details

mln is the density f(x) = alpha.1 * g(x) + (1 - alpha.1) * h(x), where g and h are densities of two lognormals with parameters (mean.log.1, sdlog.1) and (mean.log.2, sdlog.2) respectively.

Value

`obj`	value of the objective function

Author(s)

Kam Hamidieh

References

F. Gianluca and A. Roncoroni (2008) *Implementing Models in Quantitative Finance: Methods and Cases*

B. Bahra (1996): Probability distribution of future asset prices implied by option prices. *Bank of England Quarterly Bulletin*, August 1996, 299-311

P. Soderlind and L.E.O. Svensson (1997) New techniques to extract market expectations from financial instruments. *Journal of Monetary Economics*, 40, 383-429

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# The mln objective function should be close to zero.
# The weights are automatically set to 1.
#

r  = 0.05
te = 60/365
y  = 0.02

meanlog.1 = 6.8
meanlog.2 = 6.95
sdlog.1   = 0.065
sdlog.2   = 0.055
alpha.1   = 0.4

# This is the current price implied by parameter values:
s0 = 981.8815

call.strikes = seq(from = 800, to = 1200, by = 10)
market.calls = price.mln.option(r=r, y = y, te = te, k = call.strikes,
               alpha.1 = alpha.1, meanlog.1 = meanlog.1, meanlog.2 = meanlog.2,
               sdlog.1 = sdlog.1, sdlog.2 = sdlog.2)$call

put.strikes  = seq(from = 805, to = 1200, by = 10)
market.puts  = price.mln.option(r = r, y = y, te = te, k = put.strikes,
               alpha.1 = alpha.1, meanlog.1 = meanlog.1, meanlog.2 = meanlog.2,
               sdlog.1 = sdlog.1, sdlog.2 = sdlog.2)$put

mln.objective(theta=c(alpha.1,meanlog.1, meanlog.2 , sdlog.1, sdlog.2),
               r = r, y = y, te = te, s0 = s0,
               market.calls = market.calls, call.strikes = call.strikes,
               market.puts = market.puts, put.strikes = put.strikes, lambda = 1)
```

MOE *Mother of All Extractions*

Description

`MOE` function extracts the risk neutral density based on all models and summarizes the results.

Usage

```
MOE(market.calls, call.strikes, market.puts, put.strikes, call.weights = 1,
  put.weights = 1, lambda = 1, s0, r, te, y, file.name = "myfile")
```

Arguments

`market.calls`	market calls (most expensive to cheapest)
`call.strikes`	strikes for the calls (smallest to largest)
`market.puts`	market calls (cheapest to most expensive)
`put.strikes`	strikes for the puts (smallest to largest)
`call.weights`	Weights for the calls (must be in the same order of calls)
`put.weights`	Weights for the puts (must be in the same order of puts)
`lambda`	Penalty parameter to enforce the martingale condition
`s0`	Current asset value
`r`	risk free rate
`te`	time to expiration
`y`	dividend yield
`file.name`	File names where analysis is to be saved. SEE DETAILS!

Details

The MOE function in a few key strokes extracts the risk neutral density via various methods and summarizes the results.

This function should only be used for European options.

NOTE: Three files will be produced: filename will have the pdf version of the results. file.namecalls.csv will have the predicted call values. file.nameputs.csv will have the predicted put values.

Value

`bsm.mu`	mean of log(S(T)), when S(T) is lognormal
`bsm.sigma`	SD of log(S(T)), when S(T) is lognormal
`gb.a`	extracted power parameter, when S(T) is assumed to be a GB rv
`gb.b`	extracted scale paramter, when S(T) is assumed to be a GB rv
`gb.v`	extracted first beta paramter, when S(T) is assumed to be a GB rv

`gb.w`	extracted second beta parameter, when S(T) is assumed to be a GB rv
`mln.alpha.1`	extracted proportion of the first lognormal. Second one is 1 - `alpha.1` in mixture of lognormals
`mln.meanlog.1`	extracted mean of the log of the first lognormal in mixture of lognormals
`mln.meanlog.2`	extracted mean of the log of the second lognormal in mixture of lognormals
`mln.sdlog.1`	extracted standard deviation of the log of the first lognormal in mixture of lognormals
`mln.sdlog.2`	extracted standard deviation of the log of the second lognormal in mixture of lognormals
`ew.sigma`	volatility when using the Edgeworth expansions
`ew.skew`	normalized skewness when using the Edgeworth expansions
`ew.kurt`	normalized kurtosis when using the Edgeworth expansions
`a0`	extracted constant term in the quadratic polynomial of Shimko method
`a1`	extracted coefficient term of k in the quadratic polynomial of Shimko method
`a2`	extracted coefficient term of k squared in the quadratic polynomial of Shimko method

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
###
### You should see that all methods extract the same density!
###

r     = 0.05
te    = 60/365
s0    = 1000
sigma = 0.25
y     = 0.02

strikes     = seq(from = 500, to = 1500, by = 5)
bsm.prices  = price.bsm.option(r =r, te = te, s0 = s0,
              k = strikes, sigma = sigma, y = y)

calls   = bsm.prices$call
puts    = bsm.prices$put

###
```

```
### See where your results will go...
###

getwd()


###
###  Running this may take 1-2 minutes...
###
### MOE(market.calls = calls, call.strikes = strikes, market.puts = puts,
###    put.strikes = strikes, call.weights = 1, put.weights = 1,
###    lambda = 1, s0 = s0, r = r, te = te, y = y, file.name = "myfile")
###
### You may get some warning messages.  This happens because the
###    automatic initial value selection sometimes picks values
###    that produce NaNs in the generalized beta density estimation.
###    These messages are often inconsequential.
###
```

`oil.2012.10.01` *West Texas Intermediate Crude Oil Options on 2013-10-01*

Description

This dataset contains West Texas Intermediate (WTI) crude oil options with 43 days to expiration at the end of the business day October 1, 2012. On October 1, 2012, WTI closed at 92.44.

Usage

```
data(oil.2012.10.01)
```

Format

A data frame with 332 observations on the following 7 variables.

`type` a factor with levels `C` for call option `P` for put option

`strike` option strike

`settlement` option settlement price

`openint` option open interest

`volume` trading volume

`delta` option delta

`impliedvolatility` option implied volatility

Source

CME posts sample data at: http://www.cmegroup.com/market-data/datamine-historical-data/endofday.html

Examples

```
data(oil.2012.10.01)
```

pgb *CDF of Generalized Beta*

Description

pgb is the cumulative distribution function (CDF) of a genaralized beta random variable.

Usage

```
pgb(x, a, b, v, w)
```

Arguments

`x`	value at which the CDF is to be evaluated
`a`	power parameter > 0
`b`	scale paramter > 0
`v`	first beta paramter > 0
`w`	second beta parameter > 0

Details

Let B be a beta random variable with parameters v and w. Then Z = b *(B/(1-B))^(1/a) is a generalized beta random variable with parameters (a,b,v,w).

Value

`out`	CDF value at x

Author(s)

Kam Hamidieh

References

R.M. Bookstaber and J.B. McDonald (1987) A general distribution for describing security price returns. *Journal of Business*, 60, 401-424

X. Liu and M.B. Shackleton and S.J. Taylor and X. Xu (2007) Closed-form transformations from risk-neutral to real-world distributions *Journal of Business*, 60, 401-424

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# What does the cdf of a GB look like?
#

a  = 1
b  = 10
v  = 2
w  = 2

x = seq(from = 0, to = 500, by = 0.01)
y = pgb(x = x, a = a, b = b, v = v, w = w)
plot(y ~ x, type = "l")
abline(h=c(0,1), lty=2)
```

---

price.am.option *Price American Options on Mixtures of Lognormals*

Description

`price.am.option` gives the price of a call and a put option at a set strike when the risk neutral density is a mixture of three lognormals.

Usage

```
price.am.option(k, r, te, w, u.1, u.2, u.3, sigma.1, sigma.2, sigma.3, p.1, p.2)
```

Arguments

`k`	Strike
`r`	risk free rate
`te`	time to expiration
`w`	Weight, a number between 0 and 1, to weigh the option price bounds
`u.1`	log mean of the first lognormal
`u.2`	log mean of the second lognoral
`u.3`	log mean of the second lognoral
`sigma.1`	log sd of the first lognormal
`sigma.2`	log mean of the second lognormal
`sigma.3`	log mean of the third lognormal
`p.1`	weight assigned to the first density
`p.2`	weight assigned to the second density

Details

mln is density f(x) = p.1 * f1(x) + p.2 * f2(x) + (1 - p.1 - p.2) * f3(x), where f1, f2, and f3 are lognormal densities with log means u.1,u.2, and u.3 and standard deviations sigma.1, sigma.2, and sigma.3 respectively.

Note: Different weight values, w, need to be assigned to whether the call or put is in the money or not. See equations (7) & (8) of Melick and Thomas paper below.

Value

`call.value`	American call value
`put.value`	American put value
`expected.f0`	Expected mean value of asset at expiration
`prob.f0.gr.k`	Probability asset values is greater than strike
`prob.f0.ls.k`	Probability asset value is less than strike
`expected.f0.f0.gr.k`	Expected value of asset given asset exceeds strike
`expected.f0.f0.ls.k`	Expected value of asset given asset is less than strike

Author(s)

Kam Hamidieh

References

Melick, W. R. and Thomas, C. P. (1997). Recovering an asset's implied pdf from option prices: An application to crude oil during the gulf crisis. *Journal of Financial and Quantitative Analysis*, 32(1), 91-115.

Examples

```
###
###  Set a few parameters and create some
###  American options.
###

r       = 0.01
te      = 60/365
w.1     = 0.4
w.2     = 0.25
u.1     = 4.2
u.2     = 4.5
u.3     = 4.8
sigma.1 = 0.30
sigma.2 = 0.20
sigma.3 = 0.15
p.1     = 0.25
```

```
p.2     = 0.45
theta   = c(w.1,w.2,u.1,u.2,u.3,sigma.1,sigma.2,sigma.3,p.1,p.2)

p.3 = 1 - p.1 - p.2
p.3
expected.f0   =  sum(c(p.1, p.2, p.3) * exp(c(u.1,u.2,u.3) +
                    (c(sigma.1, sigma.2, sigma.3)^2)/2) )
expected.f0

strikes = 30:170

market.calls = numeric(length(strikes))
market.puts  = numeric(length(strikes))

for (i in 1:length(strikes))
{

  if ( strikes[i] < expected.f0) {
    market.calls[i] = price.am.option(k = strikes[i], r = r, te = te, w = w.1, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$call.value

    market.puts[i]  = price.am.option(k = strikes[i], r = r, te = te, w = w.2, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$put.value
  }  else {

    market.calls[i] = price.am.option(k = strikes[i], r = r, te = te, w = w.2, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$call.value

    market.puts[i]  = price.am.option(k = strikes[i], r = r, te = te, w = w.1, u.1 = u.1,
                      u.2 = u.2, u.3 = u.3, sigma.1 = sigma.1, sigma.2 = sigma.2,
                      sigma.3 = sigma.3, p.1 = p.1, p.2 = p.2)$put.value
     }

}

###
### Quickly look at the option values...
###

par(mfrow=c(1,2))
plot(market.calls ~ strikes, type="l")
plot(market.puts  ~ strikes, type="l")
par(mfrow=c(1,1))
```

---

price.bsm.option *Price BSM Option*

Description

`bsm.option.price` computes the BSM European option prices.

Usage

```
price.bsm.option(s0, k, r, te, sigma, y)
```

Arguments

`s0`	current asset value
`k`	strike
`r`	risk free rate
`te`	time to expiration
`sigma`	volatility
`y`	dividend yield

Details

This function implements the classic Black-Scholes-Merton option pricing model.

Value

`d1`	value of `(log(s0/k) + (r - y + (sigma^2)/2) * te)/(sigma * sqrt(te))`
`d2`	value of `d1 - sigma * sqrt(te)`
`call`	call price
`put`	put price

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

J. Hull (2011) *Options, Futures, and Other Derivatives and DerivaGem Package* Prentice Hall, Englewood Cliffs, New Jersey, 8th Edition

R. L. McDonald (2013) *Derivatives Markets* Pearson, Upper Saddle River, New Jersey, 3rd Edition

Examples

```
#
# call should be 4.76, put should be 0.81, from Hull 8th, page 315, 316
#

r     = 0.10
te    = 0.50
```

```
s0    = 42
k     = 40
sigma = 0.20
y     = 0

bsm.option = price.bsm.option(r =r, te = te, s0 = s0, k = k, sigma = sigma, y = y)
bsm.option

#
# Make sure put-call parity holds, Hull 8th, page 351
#

(bsm.option$call - bsm.option$put) - (s0 * exp(-y*te) - k * exp(-r*te))
```

`price.ew.option` *Price Options with Edgeworth Approximated Density*

Description

`price.ew.option` computes the option prices based on Edgeworth approximated densities.

Usage

```
price.ew.option(r, te, s0, k, sigma, y, skew, kurt)
```

Arguments

`r`	risk free rate
`te`	time to expiration
`s0`	current asset value
`k`	strike
`sigma`	volatility
`y`	dividend rate
`skew`	normalized skewness
`kurt`	normalized kurtosis

Details

Note that this function may produce negative prices if `skew` and `kurt` are not well estimated from the data.

Value

`call`	Edgeworth based call
`put`	Edgeworth based put

Author(s)

Kam Hamidieh

References

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

R. Jarrow and A. Rudd (1982) Approximate valuation for arbitrary stochastic processes. *Journal of Finanical Economics*, 10, 347-369

C.J. Corrado and T. Su (1996) S&P 500 index option tests of Jarrow and Rudd's approximate option valuation formula. *Journal of Futures Markets*, 6, 611-629

Examples

```
#
# Here, the prices must match EXACTLY the BSM prices:
#

r       = 0.05
y       = 0.03
s0      = 1000
sigma   = 0.25
te      = 100/365
k       = seq(from=800, to = 1200, by = 50)
v       = sqrt(exp(sigma^2 * te) - 1)
ln.skew = 3 * v + v^3
ln.kurt = 16 * v^2 + 15 * v^4 + 6 * v^6 + v^8

ew.option.prices  =  price.ew.option(r = r, te = te, s0 = s0, k=k, sigma=sigma,
                                     y=y, skew = ln.skew, kurt = ln.kurt)
bsm.option.prices =  price.bsm.option(r = r, te = te, s0 = s0, k=k, sigma=sigma, y=y)

ew.option.prices
bsm.option.prices

###
### Now ew prices should be different as we increase the skewness and kurtosis:
###

new.skew = ln.skew * 1.10
new.kurt = ln.kurt * 1.10

new.ew.option.prices  =  price.ew.option(r = r, te = te, s0 = s0, k=k, sigma=sigma,
                                         y=y, skew = new.skew, kurt = new.kurt)
new.ew.option.prices
bsm.option.prices
```

`price.gb.option` *Generalized Beta Option Pricing*

Description

`price.gb.option` computes the price of options.

Usage

```
price.gb.option(r, te, s0, k, y, a, b, v, w)
```

Arguments

`r`	risk free interest rate
`te`	time to expiration
`s0`	current asset value
`k`	strike
`y`	dividend yield
`a`	power parameter > 0
`b`	scale paramter > 0
`v`	first beta paramter > 0
`w`	second beta parameter > 0

Details

This function is used to compute European option prices when the underlying has a generalized beta (GB) distribution. Let B be a beta random variable with parameters v and w. Then Z = b *(B/(1-B))^(1/a) is a generalized beta random variable with parameters with (a,b,v,w).

Value

`prob.1`	Probability that a GB random variable with parameters (a,b,v+1/a,w-1/a) will be above the strike
`prob.2`	Probability that a GB random variable with parameters (a,b,v,w) will be above the strike
`call`	call price
`put`	put price

Author(s)

Kam Hamidieh

References

R.M. Bookstaber and J.B. McDonald (1987) A general distribution for describing security price returns. *Journal of Business*, 60, 401-424

X. Liu and M.B. Shackleton and S.J. Taylor and X. Xu (2007) Closed-form transformations from risk-neutral to real-world distributions *Journal of Business*, 60, 401-424

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# A basic GB option pricing....
#

r  = 0.03
te = 50/365
s0 = 1000.086
k  = seq(from = 800, to = 1200, by = 10)
y  = 0.01
a  = 10
b  = 1000
v  = 2.85
w  = 2.85

price.gb.option(r = r, te = te, s0 = s0, k = k, y = y, a = a, b = b, v = v, w = w)
```

---

price.mln.option   *Price Options on Mixture of Lognormals*

Description

`mln.option.price` gives the price of a call and a put option at a strike when the risk neutral density is a mixture of two lognormals.

Usage

```
price.mln.option(r, te, y, k, alpha.1, meanlog.1, meanlog.2, sdlog.1, sdlog.2)
```

Arguments

`r`	risk free rate
`te`	time to expiration
`y`	dividend yield
`k`	strike

`alpha.1`	proportion of the first lognormal. Second one is 1 - `alpha.1`
`meanlog.1`	mean of the log of the first lognormal
`meanlog.2`	mean of the log of the second lognormal
`sdlog.1`	standard deviation of the log of the first lognormal
`sdlog.2`	standard deviation of the log of the second lognormal

Details

mln is the density f(x) = alpha.1 * g(x) + (1 - alpha.1) * h(x), where g and h are densities of two lognormals with parameters (mean.log.1, sdlog.1) and (mean.log.2, sdlog.2) respectively.

Value

`call`	call price
`put`	put price
`s0`	current value of the asset as implied by the mixture distribution

Author(s)

Kam Hamidieh

References

F. Gianluca and A. Roncoroni (2008) *Implementing Models in Quantitative Finance: Methods and Cases*

B. Bahra (1996): Probability distribution of future asset prices implied by option prices. *Bank of England Quarterly Bulletin*, August 1996, 299-311

P. Soderlind and L.E.O. Svensson (1997) New techniques to extract market expectations from financial instruments. *Journal of Monetary Economics*, 40, 383-429

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
#
# Try out a range of options
#

r  = 0.05
te = 60/365
k  = 700:1300
y  = 0.02
meanlog.1 = 6.80
meanlog.2 = 6.95
sdlog.1   = 0.065
sdlog.2   = 0.055
alpha.1   = 0.4
```

```
mln.prices = price.mln.option(r = r, y = y, te = te, k = k, alpha.1 = alpha.1,
  meanlog.1 = meanlog.1, meanlog.2 = meanlog.2, sdlog.1 = sdlog.1, sdlog.2 = sdlog.2)

par(mfrow=c(1,2))
plot(mln.prices$call ~ k)
plot(mln.prices$put  ~ k)
par(mfrow=c(1,1))
```

`price.shimko.option` *Price Option based on Shimko's Method*

Description

`price.shimko.option` prices a European option based on the extracted Shimko volatility function.

Usage

```
price.shimko.option(r, te, s0, k, y, a0, a1, a2)
```

Arguments

`r`	risk free rate
`te`	time to expiration
`s0`	current asset value
`k`	strike
`y`	dividend yield
`a0`	constant term in the quadratic polyynomial
`a1`	coefficient term of k in the quadratic polynomial
`a2`	coefficient term of k squared in the quadratic polynomial

Details

This function may produce negative option values when nonsensical values are used for a0, a1, and a2.

Value

`call`	call price
`put`	put price

Author(s)

Kam Hamidieh

References

D. Shimko (1993) Bounds of probability. *Risk*, 6, 33-47

E. Jondeau and S. Poon and M. Rockinger (2007): *Financial Modeling Under Non-Gaussian Distributions* Springer-Verlag, London

Examples

```
r       =  0.05
y       =  0.02
te      =  60/365
s0      =  1000
k       =  950
sigma   =  0.25
a0      =  0.30
a1      =  -0.00387
a2      =  0.00000445

#
#  Note how Shimko price is the same when a0 = sigma, a1=a2=0 but substantially
#  more when a0, a1, a2 are changed so the implied volatilies are very high!
#

price.bsm.option(r = r, te = te, s0 = s0, k = k, sigma = sigma, y = y)$call
price.shimko.option(r = r, te = te, s0 = s0, k = k, y = y,
                    a0 = sigma, a1 = 0, a2 = 0)$call
price.shimko.option(r = r, te = te, s0 = s0, k = k, y = y,
                    a0 = a0, a1 = a1, a2 = a2)$call
```

---

sp500.2013.04.19 *S&P 500 Index Options on 2013-04-19*

Description

This dataset contains S&P 500 options with 62 days to expiration at the end of the business day April 19, 2013. On April 19, 2013, S&P 500 closed at 1555.25.

Usage

```
data(sp500.2013.04.19)
```

Format

A data frame with 171 observations on the following 19 variables.

`bidsize.c` call bid size

`bid.c` call bid price

`ask.c` call ask price

`asksize.c` call ask size

`chg.c` change in call price

`impvol.c` call implied volatility

`vol.c` call volume

`openint.c` call open interest

`delta.c` call delta

`strike` option strike

`bidsize.p` put bid size

`bid.p` put bid price

`ask.p` put ask price

`asksize.p` put ask size

`chg.p` change in put price

`impvol.p` put implied volatility

`vol.p` put volume

`openint.p` put open interest

`delta.p` put delta

Source

http://www.cboe.com/DelayedQuote/QuoteTableDownload.aspx

Examples

```
data(sp500.2013.04.19)
```

`sp500.2013.06.24` *S&P 500 Index Options on 2013-06-24*

Description

This dataset contains S&P 500 options with 53 days to expiration at the end of the business day June 24, 2013. On June 24, 2013, S&P 500 closed at 1573.09.

Usage

```
data(sp500.2013.06.24)
```

Format

A data frame with 173 observations on the following 9 variables.

`bid.c` call bid price

`ask.c` call ask price

`vol.c` call volume

`openint.c` call open interest

`strike` option strike

`bid.p` put bid price

`ask.p` put ask price

`vol.p` put volume

`openint.p` put open interest

Source

http://www.cboe.com/DelayedQuote/QuoteTableDownload.aspx

Examples

```
data(sp500.2013.06.24)
```

`vix.2013.06.25` *VIX Options on 2013-06-25*

Description

This dataset contains VIX options with 57 days to expiration at the end of the business day June 25, 2013. On June 25, 2013, VIX closed at 18.21.

Usage

```
data(vix.2013.06.25)
```

Format

A data frame with 35 observations on the following 13 variables.

`last.c` closing call price

`change.c` change in call price from previous day

`bid.c` call bid price

`ask.c` call ask price

`vol.c` call volume

`openint.c` call open interest

`strike` option strike

`last.p` closing put price

`change.p` change in put price from previous day

`bid.p` put bid price

`ask.p` put ask price

`vol.p` put volume

`openint.p` put open interest

Source

http://www.cboe.com/DelayedQuote/QuoteTableDownload.aspx

Examples

```
data(vix.2013.06.25)
```

Index