Package 'solarr'

May 28, 2024

Type Package

Title Stochastic model for solar radiation data

Version 0.1.0

Author Beniamino Sartini

Maintainer Beniamino Sartini <beniamino.sartini2@unibo.it>

Description Implementation of stochastic models and option pricing on solar radiation data.

Depends R (>= 3.5.0),
ggplot2 (>= 3.4.0),
dplyr (>= 1.1.3),

Imports assertive (>= 0.3-6),
stringr (>= 1.5.0),
rugarch (>= 1.4.1),
dplyr (>= 1.1.3),
purrr (>= 1.0.2),
readr (>= 2.1.2),
tidyr (>= 1.2.0),
tibble (>= 3.2.1),
ggplot2 (>= 3.4.0),
lubridate (>= 1.8.0),
reticulate (>= 1.24),

Suggests DT,
knitr,
rmarkdown,
testthat (>= 2.1.0)

License GPL-3

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

R topics documented:

---

boot_dnorm_mix *Bootstrap Parameters of Normal Mixture*

Description

Bootstrap the parameter for the density function for a normal mixture with two components.

Usage

```
boot_dnorm_mix(
  x,
  params,
  B = 50,
  ci = 0.95,
  sample_perc = 0.8,
  loss = "kl",
  seed = 1,
  na.rm = TRUE
)
```

Arguments

x	vector
params	initial parameters
B	number of bootstraps
ci	confidence interval for empirical quantiles
loss	loss type. Can be ‘ml‘ for maximum likelihood or ‘kl‘ for kl_dist.
seed	random seed
na.rm	logical.

Examples

```
params <- c(mu1 = -2, mu2 = 2, sd1 = 3, sd2 = 1, p = 0.5)
n1 <- rnorm(t_bar, mean = params[1], sd = params[3])
n2 <- rnorm(t_bar, mean = params[2], sd = params[4])
Z <- rbinom(t_bar, 1, params[5])
x <- Z * n1 + (1-Z)*n2
boot_dnorm_mix(x, params = init_params,  B = 50, ci = 0.95, sample_perc = 0.8, loss = "ml")$par
boot_dnorm_mix(x, params = init_params,  B = 50, ci = 0.95, sample_perc = 0.8, loss = "kl")$par
```

CAMS — *CAMS Data (Location object)*

Description

CAMS Data (Location object)

Usage

```
CAMS(place, year_max = lubridate::year(Sys.Date()))
```

Arguments

place	place

Examples

```
object <- CAMS("Bologna")

place <- "Bologna"
```

clearsky.seasonalModel — *Seasonal Model Clear sky Radiation*

Description

Seasonal Model Clear sky Radiation

Usage

```
clearsky.seasonalModel(object, control = control.seasonalModel())
```

Examples

```
object <- CAMS("Amsterdam")
control = control.seasonalModel()
```

control — *Control function*

Description

Control function

Usage

```
control(object)
```

control.seasonalModel *seasonalModel control parameters*

Description

seasonalModel control parameters

Usage

```
## S3 method for class 'seasonalModel'
control(
  object,
  method = "II",
  include.intercept = TRUE,
  order = 1,
  seed = 1,
  delta_init = 1.1,
  tol = 30,
  lower = 0,
  upper = 1,
  by = 0.001,
  quiet = FALSE
)
```

control.solarEsscher *solarEsscher control parameters*

Description

solarEsscher control parameters

Usage

```
## S3 method for class 'solarEsscher'
control(
  nsim = 200,
  ci = 0.05,
  seed = 1,
  quiet = FALSE,
  n_key_points = 15,
  init_lambda = 0,
  lower_lambda = -1,
  upper_lambda = 1
)
```

control.solarModel *solarModel control parameters*

Description

solarModel control parameters

Usage

```
## S3 method for class 'solarModel'
control(
  object,
  loss = "ml",
  clearsky.model = control.seasonalModel(),
  mean.model = list(seasonalOrder = 1, arOrder = 2, include.intercept = FALSE),
  variance.model = list(seasonalOrder = 1, match_moments = FALSE),
  threshold = 0.001,
  quiet = FALSE
)
```

Arguments

`loss`	type of loss function for mixture model, ‘ml‘ stands for maximum-likelihood, while ‘kl‘ for KL-distance.
`mean.model`	a list of parameters
`variance.model`	a list of parameters
`threshold`	Threshold for the estimation of alpha and beta
`quiet`	logical, when ‘TRUE‘ the function will not display any message.

control.solarOption *solarOption control parameters*

Description

solarOption control parameters

Usage

```
## S3 method for class 'solarOption'
control(nyears = c(2010, 2022), K = 0, put = TRUE, B = discount_factor())
```

`desscher_mix` *Esscher transform of a Gaussian Mixture*

Description

Esscher transform of a Gaussian Mixture

Usage

```
desscher_mix(params = c(0, 1, 0, 1, 0.5))

pesscher_mix(params = c(0, 1, 0, 1, 0.5))
```

Arguments

`params`	Gaussian Mixture parameters, mu1, sigma1, mu2, sigma2, p.

`detect_season` *Detect Season*

Usage

```
detect_season(day_date = NULL)
```

Arguments

`day_date`	vector of dates in the format ‘ a character vector containing the correspondent season. Can be ‘spring‘, ‘summer‘, ‘autumn‘, ‘winter‘. Detect the season from a vector of dates detect_season("2040-01-31") detect_season(c("2040-01-31", "2023-04-01", "2015-09-02"))

`discount_factor` *Discount factor function*

Description

Discount factor function

Usage

```
discount_factor(r = 0.03)
```

Arguments

`r`	level of yearly constant risk-free rate

`dnorm_mix` *Normal Mixture Pdf*

Description

Probability density function for a normal mixture with two components

Usage

```
dnorm_mix(params)
```

Arguments

`params`	parameters of the two components, (mu1, mu2, sd1, sd2, p)

Value

a function of x.

Examples

```
params <- c(mu1 = -2, mu2 = 2, sd1 = 3, sd2 = 1, p = 0.5)
# Density function
pdf <- dnorm_mix(params)
```

`fit_dnorm_mix` *Fit Normal Mixture Pdf*

Description

Fit the parameter for the density function for a normal mixture with two components.

Usage

```
fit_dnorm_mix(x, params, loss = "ml", na.rm = TRUE)
```

Arguments

`x`	vector
`params`	initial parameters
`loss`	loss type. Can be ‘ml‘ for maximum likelihood or ‘kl‘ for kl_dist.

Examples

```
params <- c(mu1 = -2, mu2 = 2, sd1 = 3, sd2 = 1, p = 0.5)
n1 <- rnorm(t_bar, mean = params[1], sd = params[3])
n2 <- rnorm(t_bar, mean = params[2], sd = params[4])
Z <- rbinom(t_bar, 1, params[5])
x <- Z * n1 + (1-Z)*n2
fit_dnorm_mix(x, params = init_params, loss = "ml")$par
fit_dnorm_mix(x, params = init_params, loss = "kl")$par
```

from_list_to_parameters.solarModel

Convert a vector of parameter in a structured list

Description

Convert a vector of parameter in a structured list

Usage

```
from_list_to_parameters.solarModel(params_list)
```

from_radiant_to_degree

Conversion in Radiant or Degrees

Description

Convert an angle in radiant into an angle in degrees.

Usage

```
from_radiant_to_degree(x = NULL)

from_degree_to_radiant(x = NULL)
```

Arguments

x	numeric vector, angles in radiant or degrees.

Value

numeric numeric vector.

Examples

```
# convert 0.34 radiant in degrees
from_radiant_to_degree(0.34)

# convert 19.48 degree in radiant
from_degree_to_radiant(19.48)
```

getCAMS *get CAMS data*

Description

get CAMS data

Usage

```
getCAMS(place, lat, lon, alt, from = "2005-01-01", to = Sys.Date())
```

gumbel *Gumbel Random Variable*

Description

Probability density function for a gumbel random variable

Usage

```
dgumbel(x, mean = 0, scale = 1, log.p = FALSE, invert = FALSE)

pgumbel(
  x,
  mean = 0,
  scale = 1,
  log.p = FALSE,
  lower.tail = TRUE,
  invert = FALSE
)

qgumbel(
  p,
  mean = 0,
  scale = 1,
  log.p = FALSE,
  lower.tail = TRUE,
  invert = FALSE
)

rgumbel(n, mean = 0, scale = 1, invert = FALSE)
```

Arguments

x	vector of quantiles.
mean	vector of means.
scale	vector of scale parameter.
log.p	logical; if ‘TRUE‘, probabilities p are given as ‘log(p)‘.

invert	logical, use the inverted Gumbel distribution
lower.tail	logical; if TRUE (default), probabilities are ‘P[X < x]‘ otherwise, ‘P[X > x]‘.
p	vector of probabilities.
n	number of observations. If ‘length(n) > 1‘, the length is taken to be the number required.

Examples

```
x <- seq(-5, 5, 0.01)

# Density function
p <- dgumbel(x, mean = 0, scale = 1)
plot(x, p, type = "l")

# Distribution function
p <- pgumbel(x, mean = 0, scale = 1)
plot(x, p, type = "l")

# Quantile function
qgumbel(0.1)
pgumbel(qgumbel(0.1))

# Random Numbers
rgumbel(1000)
plot(rgumbel(1000), type = "l")
```

---

is_leap_year *Is leap year?*

Usage

```
is_leap_year(day_date)
```

Arguments

day_date dates vector in the format ‘

Boolean. ‘TRUE‘ if it is a leap year, ‘FALSE‘ otherwise.

Check if an year is leap (366 days) or not (365 days).

is_leap_year("2024-02-01") is_leap_year(c("2024-10-01", "2025-10-01")) is_leap_year("2029-02-01") `kl_dist` *Kullback–Leibler divergence*

Description

Compute the Kullback–Leibler distance between two probability measure.

Usage

```
kl_dist(p, q, quiet = FALSE)

kl_dist_cont(pdf_1, pdf_2, lower = -Inf, upper = Inf, quiet = FALSE)
```

Arguments

`p`	Numeric, probability vector. Usually, the empiric probabilities.
`q`	Numeric, probability vector. Usually, the model probabilities.
`quiet`	Boolean, default is 'TRUE'. When set to 'FALSE' the function will not display warnings.

Details

The function implements:

$$\sum_i p_i \log(\frac{p_i}{q_i}) \quad p_i, q_i > 0 \; \forall i$$

References

https://en.wikipedia.org/wiki/Kullback–Leibler_divergence

Examples

```
p <- dnorm(rnorm(100))
q <- dnorm(rnorm(100))
kl_dist(p, q)

pdf_1 <- function(x) dnorm(x, mean = 2, sd = 1)
pdf_2 <- function(x) dnorm(x, mean = -2, sd = 3)
kl_dist_cont(pdf_1, pdf_2, lower = -Inf, upper = Inf)
```

`kumaraswamy` *Kumaraswamy Random Variable*

Description

Probability functions for a Kumaraswamy random variable

Usage

```
dkumaraswamy(x, a = 1, b = 1, log.p = FALSE)

pkumaraswamy(x, a = 1, b = 1, log.p = FALSE, lower.tail = TRUE)

qkumaraswamy(p, a = 1, b = 1, log.p = FALSE, lower.tail = TRUE)

rkumaraswamy(n, a = 1, b = 1)
```

Arguments

`x`	vector of quantiles.
`a`	parameter.
`b`	parameter..
`log.p`	logical; if ‘TRUE‘, probabilities p are given as ‘log(p)‘.
`lower.tail`	logical; if TRUE (default), probabilities are ‘P[X < x]‘ otherwise, ‘P[X > x]‘.
`p`	vector of probabilities.
`n`	number of observations. If ‘length(n) > 1‘, the length is taken to be the number required.

`logLik.solarModel` *Compute the Log-likelihood for a Solar Model*

Description

Compute the Log-likelihood for a Solar Model

Usage

```
## S3 method for class 'solarModel'
logLik(model, params)
```

normality_test *Perform normality tests*

Description

Perform normality tests

Usage

```
normality_test(x = NULL, p_value = 0.05)
```

Arguments

x	vector
p_value	p.value

Value

a tibble

number_of_day *Number of Day*

Usage

```
number_of_day(day_date = NULL)
```

Arguments

day_date dates vector in the format ‘

Numeric vector with the number of the day during the year. Can vary from ‘1‘ up to ‘365‘ or ‘366‘.

Compute the number of day of the year given a vector of dates.

detect the number of the day in 2040-01-31 number_of_day("2040-01-31")

detect the number of the day for a vector of dates number_of_day(c("2040-01-31", "2023-04-01", "2015-09-02"))

optimize.solarModel *Log-likelihood optimazation for Solar Model*

Description

Log-likelihood optimazation for Solar Model

Usage

```
optimize.solarModel(model, maxit = 100, quiet = FALSE)
```

plot.solarModel *Plot solar model*

Description

Plot solar model

Usage

```
## S3 method for class 'solarModel'
plot(object, nplot = 1, plot_year = 2019)
```

plot.solarModelSimulation *Plot solar model simulations*

Description

Plot solar model simulations

Usage

```
## S3 method for class 'solarModelSimulation'
plot(data, object, nplot = 1, empiric = TRUE)
```

pnorm_mix *Normal Mixture Cdf*

Description

Probability distribution function for a normal mixture with two components.

Usage

```
pnorm_mix(params)
```

Arguments

params	parameters of the two components, (mu1, mu2, sd1, sd2, p)

Value

a function of x.

Examples

```
params <- c(mu1 = -2, mu2 = 2, sd1 = 3, sd2 = 1, p = 0.5)
# Density function
cdf <- pnorm_mix(params)
```

predict.seasonalModel *predict method for seasonalModel*

Description

predict method for seasonalModel

Usage

```
## S3 method for class 'seasonalModel'
predict(object, n = 1)
```

print.Location *Method Print for Location object*

Description

Method Print for Location object

Usage

```
## S3 method for class 'Location'
print(x)
```

Examples

```
object <- Location("Roma")
object
```

PUN *PUN*

Description

Function that computes the mean PUN.

Usage

```
PUN(nyear = NULL, nmonth = NULL, file = "data/df_GME_day.RData")
```

Arguments

`file`	path
`month`	Reference month.
`year`	Reference year.

Value

numeric, price in euros of a kwh

qnorm_mix *Normal Mixture Quantile*

Description

Quantile function for a normal mixture with two components.

Usage

```
qnorm_mix(params)
```

Arguments

params	parameters of the two components, (mu1, mu2, sd1, sd2, p)

Value

a function of p.

Examples

```
params <- c(mu1 = -2, mu2 = 2, sd1 = 3, sd2 = 1, p = 0.5)
# Density function
quantile_func <- qnorm_mix(params)
```

riccati_root *Riccati Square Root*

Description

Square root of a symmetric matrix.

Usage

```
riccati_root(x)
```

Arguments

x	symmetric matrix.

Examples

```
x <- matrix(c(1, 0.3, 0.3, 1), nrow = 2, byrow = TRUE)
riccati_root(x)
```

`seasonalModel` *Seasonal Model*

Description

Seasonal Model

Usage

```
seasonalModel(formula = "Yt ~ 1", order = 1, period = 365, data)
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted.
`order`	number of sine/cosine expansions.
`period`	periodicity period 2pi/365.
`data`	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which ‘lm‘ is called.

Examples

```
formula = "GHI ~ 1"
order = 1
period = 365
data = model$data
```

`simulate.solarModel` *Simulate scenarios of solar model*

Description

Simulate scenarios of solar model

Usage

```
## S3 method for class 'solarModel'
simulate(
  object,
  from = "2010-01-01",
  to = "2010-12-31",
  nsim = 1,
  lambda = 0,
  vol = NA,
  rf = FALSE,
  seed = 1,
  quiet = FALSE
```

```
)

scenario.solarModel(
  object,
  from = "2010-01-01",
  to = "2010-12-31",
  by = "1 month",
  nsim = 1,
  lambda = 0,
  vol = NA,
  rf = FALSE,
  seed = 1,
  quiet = FALSE
)
```

Arguments

`from`	scalar date, starting date for simulations.
`to`	scalar date, end date for simulations.
`nsim`	scalar integer, number of simulations.
`lambda`	scalar numeric, Esscher parameter. When ‘rf = FALSE‘, the input parameter ‘lambda‘ will be transformed in negative.
`vol`	scalar numeric, unconditional mean of GARCH(1,1) standard deviation. If ‘NA‘ will be used the estimated one.
`rf`	logical. When ‘TRUE‘ the AR(2) component will be set to zero.
`seed`	scalar integer, starting random seed.
`quiet`	logical

`snorm` *Skewed Normal*

Description

Probability for a skewed normal random variable.

Usage

```
dsnorm(x, mean = 0, sd = 1, skew = 0, log = FALSE)

psnorm(x, mean = 0, sd = 1, skew = 0, log.p = FALSE, lower.tail = TRUE)

qsnorm(p, mean = 0, sd = 1, skew = 0, log.p = FALSE, lower.tail = TRUE)

rsnorm(n, mean = 0, sd = 1, skew = 0)
```

Arguments

`x`	vector of quantiles.
`mean`	vector of means.
`sd`	vector of standard deviations.
`skew`	vector of skewness.
`log.p`	logical; if 'TRUE', probabilities p are given as 'log(p)'.
`lower.tail`	logical; if TRUE (default), probabilities are 'P[X < x]' otherwise, 'P[X > x]'.
`p`	vector of probabilities.
`n`	number of observations. If 'length(n) > 1', the length is taken to be the number required.

Examples

```
x <- seq(-5, 5, 0.01)

# Density function
p <- dsnorm(x, mean = 0, scale = 1)
plot(x, p, type = "l")

# Distribution function
p <- psnorm(x, mean = 0, scale = 1)
plot(x, p, type = "l")

# Quantile function
dsnorm(0.1)
psnorm(qsnorm(0.1))

# Random Numbers
rsnorm(1000)
plot(rsnorm(1000), type = "l")
```

`solarModel` *Solar Model*

Description

Solar Model

Usage

```
solarModel(object, control = control.solarModel())
```

Arguments

`object`	Location object, 'CAMS("Bologna")'
`control`	control settings, 'control.solarModel()'.

solarModel.monthly_mixture

Solar Normal Mixture Model

Description

Solar Normal Mixture Model

Usage

```
solarModel.monthly_mixture(data, loss = "ml", match_moments = FALSE)
```

Arguments

`data`	dataset with at least a column with ‘Month‘ and the target variable names ‘ut‘.
`loss`	character, type of loss function. Default is ‘ml‘ for maximum likelihood or can be ‘kl‘ for KL distance.
`match_moments`	logical.

solar_angle_minmax

solar_angle_minmax

Usage

```
solar_angle_minmax(
  lat = NULL,
  day_date = Sys.Date(),
  day_end = NULL,
  method = "cooper"
)
```

Arguments

`lat` integer, latitude.

`day_date` vector of dates in the format ‘

\itemday_endend date, if it is not NULL will be end date.

\itemmethodmethod used for computation of solar declination, can be ‘cooper‘ or ‘spencer‘.

a tibble.

Compute the solar angle for a latitude in different dates.

solar_angle_minmax(55.3, "2040-01-01", day_end = "2040-12-31")

solar_angle_minmax(55.3, c("2040-01-31", "2023-04-01", "2015-09-02"))

solar_clearsky_hourly *solar_clearsky_hourly*

Description

Compute the clearsky radiation for hourly data.

Usage

```
solar_clearsky_hourly(
  cosZ = NULL,
  G0 = NULL,
  altitude = 2.5,
  clime = c("No Correction", "Summer", "Winter", "Subartic Summer", "Tropical")
)
```

Value

a numeric vector containing the time adjustment in minutes.

Examples

```
# detect the season in 2040-01-31
solar_clearsky(cosZ = 1, altitude = 2.5, clime = c("No Correction", "Summer", "Winter", "Subartic Summer", "Tro
```

solar_clearsky_optimizer *Optimizer for Solar Clear sky*

Description

Find the best parameter delta for fitting clear sky radiation.

Usage

```
solar_clearsky_optimizer(GHI, G0, control = control.seasonalModel())
```

Arguments

GHI	vector of solar radiation
G0	vector of extraterrestrial radiation

Value

a numeric vector containing the fitted clear sky radiation.

`solar_extraterrestrial_radiation`
solar_extraterrestrial_radiation

Usage

```
solar_extraterrestrial_radiation(
  lat = NULL,
  day_date = Sys.Date(),
  day_end = NULL,
  method = "spencer"
)
```

Arguments

`lat`	latitude
`day_date`	vector of dates in the format ‘ \itemday_endend date, if it is not NULL will be end date. \itemmethodmethod used for computation of solar declination, can be ‘cooper‘ or ‘spencer‘. a numeric vector containing the time adjustment in minutes. compute the solar angle for a latitude in different times of the day. # detect the season in 2040-01-31 solar_extraterrestrial_radiation(42.23, "2022-05-01", day_end = "2022-05-31")

`solar_extraterrestrial_radiation_hourly`
solar_extraterrestrial_radiation_hourly

Usage

```
solar_extraterrestrial_radiation_hourly(
  lat = NULL,
  lon = NULL,
  day_date_time = NULL,
  day_time_end = NULL,
  altitude = 2.5,
  clime = "No Correction"
)
```

Arguments

`lat`	latitude
`day_date`	vector of dates in the format ‘ \itemday_endend date, if it is not NULL will be end date. \itemmethodmethod used for computation of solar declination, can be ‘cooper‘ or ‘spencer‘.

a numeric vector containing the time adjustment in minutes.

Compute the extraterrestrial hourly total radiation on an horizontal surface. Note that hottel clear sky max model is included in computation.

detect the season in 2040-01-31 solar_extraterrestrial_radiation_hourly(44.23, 11.20, day_date_time = "2040-01-01 00:00:00", day_time_end = "2040-01-03 00:00:00")

`solar_movements` *solar_movements*

Usage

```
solar_movements(
  lat = NULL,
  lon = NULL,
  day_date_time = NULL,
  day_time_end = NULL,
  method = "spencer"
)
```

Arguments

`lat`	latitude
`method`	method used for computation of solar declination, can be 'cooper' or 'spencer'.
`day_date`	vector of dates in the format ' \itemday_endend date, if it is not NULL will be end date. a numeric vector containing the time adjustment in minutes. compute the solar angle for a latitude in different times of the day. # detect the season in 2040-01-31 solar_movements(44.23, 11.20, day_date_time = "2040-01-01", day_time_end = "2040-01-03")

`solar_option_esscher_calibrator`
Calibrate Esscher Bounds and parameters

Description

Calibrate Esscher Bounds and parameters

Usage

```
solar_option_esscher_calibrator(
  model,
  sim,
  control_options = control.solarOption(),
  control = control.solarEsscher()
)
```

Examples

```
model
sim
control_options = control.solarOption()
control = control.solarEsscher()
```

---

solar_option_payoff_bootstrap

Bootstrap a fair price from historical data

Description

Bootstrap a fair price from historical data

Usage

```
solar_option_payoff_bootstrap(
  model,
  nsim = 500,
  ci = 0.05,
  seed = 1,
  control = control.solarOption()
)
```

Examples

```
model <- Location$model
nsim = 500
ci = 0.05
seed = 1
control = control.solarOption()
```

---

solar_option_payoff_historical

Payoff on Historical Data

Description

Payoff on Historical Data

Usage

```
solar_option_payoff_historical(
  data,
  nmonth = 1:12,
  control = control.solarOption()
)
```

Examples

```
data = model$data
nmonth = 1:12
control = control.solarOption()
```

---

solar_option_payoff_model   *Pricing function for a solar model (for all the year)*

Description

Pricing function for a solar model (for all the year)

Usage

```
solar_option_payoff_model(
  model,
  lambda = 0,
  vol = NA,
  nmonths = 1:12,
  control = control.solarOption()
)
```

Arguments

model	an object of the class ‘solarModel‘
lambda	Esscher parameter
vol	unconditional GARCH variance, when ‘NA‘ will be used the fitted one,

Examples

```
model
nmonths = 3
lambda = 0
vol = NA
control = control.solarOption()
lambda = 0
vol = 1
nmonth = 3
nday = 1
```

solar_option_payoff_scenarios

Payoff on Simulated Data

Description

Payoff on Simulated Data

Usage

```
solar_option_payoff_scenarios(
  sim,
  nmonth = 1:12,
  nsim = NULL,
  control = control.solarOption()
)
```

Examples

```
sim = Location$model$scenarios$P
nmonth = 1:12
nsim = NULL
control = control.solarOption()
```

solar_option_payoff_structure

Structure payoffs

Description

Structure payoffs

Usage

```
solar_option_payoff_structure(model, type = "sim", exact_daily_premium = TRUE)
```

solar_time_adjustment

solar_time_adjustment

Usage

```
solar_time_adjustment(day_date = NULL, day_end = NULL)
```

Arguments

day_date vector of dates in the format ‘

\itemday_endend date, if it is not NULL will be end date.

a numeric vector containing the time adjustment in seconds.

Compute the time adjustment for a date.

detect the season in 2040-01-31 solar_time_adjustment("2040-01-31")

detect the season in a vector of dates solar_time_adjustment(c("2040-01-31", "2023-04-01", "2015-09-02"))

`solar_time_constant` *solar_time_constant*

Description

Compute the solar constant for a date.

Usage

```
solar_time_constant(day_date = NULL, day_end = NULL, method = "spencer")
```

Arguments

`day_date`	vector of dates in the format ‘YYYY-MM-DD‘.
`day_end`	end date, if it is not ‘NULL‘ will be end date.
`method`	method used for computation, can be ‘cooper‘ or ‘spencer‘.

Value

a numeric vector containing the solar constant.

Examples

```
solar_time_constant("2040-01-31")
solar_time_constant(c("2040-01-31", "2023-04-01", "2015-09-02"))
```

`solar_time_declination` *solar_time_declination*

Usage

```
solar_time_declination(
  day_date = NULL,
  day_end = NULL,
  method = c("cooper", "spencer")
)
```

Arguments

day_date vector of dates in the format ‘

\itemday_endend date, if it is not NULL will be end date.

\itemmethodmethod used for computation, can be ‘cooper‘ or ‘spencer‘.

a numeric vector containing the solar declination in minutes.

Compute the solar declination for different dates.

detect the season in 2040-01-31 solar_time_declination("2040-01-01", day_end = "2040-12-31")

detect the season in a vector of dates solar_time_declination(c("2040-01-31", "2023-04-01", "2015-09-02"))

---

tnorm *Truncated Normal*

Description

Probability for a truncated normal random variable.

Usage

```
dtnorm(x, mean = 0, sd = 1, a = -3, b = 3, log.p = FALSE)

ptnorm(x, mean = 0, sd = 1, a = -3, b = 3, log.p = FALSE, lower.tail = TRUE)

qtnorm(p, mean = 0, sd = 1, a = -3, b = 3, log.p = FALSE, lower.tail = TRUE)

rtnorm(n, mean = 0, sd = 1, a = -100, b = 100)
```

Arguments

x	vector of quantiles.
mean	vector of means.
sd	vector of standard deviations.
a	lower bound.
b	upper bound.
log.p	logical; if ‘TRUE‘, probabilities p are given as ‘log(p)‘.
lower.tail	logical; if TRUE (default), probabilities are ‘P[X < x]‘ otherwise, ‘P[X > x]‘.
p	vector of probabilities.
n	number of observations. If ‘length(n) > 1‘, the length is taken to be the number required.

Examples

```
x <- seq(-5, 5, 0.01)

# Density function
p <- dtnorm(x, mean = 0, sd = 1, a = -1)
plot(x, p, type = "l")

# Distribution function
p <- ptnorm(x, mean = 0, sd = 1, b = 1)
plot(x, p, type = "l")

# Quantile function
dtnorm(0.1)
ptnorm(qtnorm(0.1))

# Random Numbers
rtnorm(1000)
plot(rtnorm(100, mean = 1, sd = 1, a = 1, b = 10), type = "l")
```

---

update.solarModel *Extract and update parameters for Solar Model*

Description

Extract and update parameters for Solar Model

Usage

```
## S3 method for class 'solarModel'
update(object, params = NULL)
```

---

updateCAMS *update CAMS data*

Description

update CAMS data

Usage

```
updateCAMS(
  place,
  lat,
  lon,
  alt,
  from = "2005-01-01",
  to = Sys.Date() - 1,
  CAMS_data = solarr::CAMS_data,
  quiet = FALSE
)
```

Index