Package ‘solarr’

July 17, 2024

Type Package

Title Stochastic model for solar radiation data

Version 0.2.0

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),
tibble (>= 3.2.1),

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),
nortest,
broom

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:

`clearskyModel` *Seasonal model for clear sky radiation*

Description

Fit a seasonal model for clear sky radiation in a location.

Usage

```
clearskyModel_fit(data, control = clearskyModel_control())
```

Arguments

`data`	dataset
`control`	list of control parameters. See ‘clearskyModel_control()‘ for details.

`clearskyModel_control` *Control for ‘clearskyModel‘*

Description

Control parameters for a ‘clearskyModel‘ object.

Usage

```
clearskyModel_control(
  method = "II",
  include.intercept = TRUE,
  order = 1,
  period = 365,
  seed = 1,
  delta_init = 1.1,
  tol = 30,
  lower = 0,
  upper = 1,
  by = 0.001,
  quiet = FALSE
)
```

Arguments

`method`	character, method for clearsky estimate, can be ‘I‘ or ‘II‘.
`include.intercept`	logical. When ‘TRUE‘, the default, the intercept will be included in the model.
`order`	numeric, of sine and cosine elements.
`period`	numeric, periodicity. The default is ‘2*base::pi/365‘.
`seed`	numeric, random seed for reproducibility. It is used to random impute the violations.

delta_init	Value for delta init in the clearsky model.
tol	integer, numeric tolerance for clearsky > GHI condition. Maximum number of violations admitted. Used in ‘clearskyModel_optimize()‘.
lower	numeric, lower bound for delta grid, see ‘clearskyModel_optimize()‘.
upper	numeric, upper bound for delta grid, see ‘clearskyModel_optimize()‘.
by	numeric, step for delta grid, see ‘clearskyModel_optimize()‘.
quiet	logical. When ‘FALSE‘, the default, the functions displays warning or messages.

---

clearskyModel_optimize *Optimizer for Solar Clear sky*

Description

Find the best parameter delta for fitting clear sky radiation.

Usage

```
clearskyModel_optimize(GHI, G0, control = clearskyModel_control())
```

Arguments

GHI	vector of solar radiation
G0	vector of extraterrestrial radiation
control	list of control parameters. See ‘clearskyModel_control()‘ for details.

Value

a numeric vector containing the fitted clear sky radiation.

---

clearskyModel_predict *Predict method*

Description

Predict method for the class ‘cleaskyModel‘.

Usage

```
clearskyModel_predict(object, n = 1)
```

Arguments

object	an object of the class ‘cleaskyModel‘
n	number of day of the year.

control_solarEsscher *solarEsscher control parameters*

Description

solarEsscher control parameters

Usage

```
control_solarEsscher(
  nsim = 200,
  ci = 0.05,
  seed = 1,
  quiet = FALSE,
  n_key_points = 15,
  init_lambda = 0,
  lower_lambda = -1,
  upper_lambda = 1
)
```

control_solarOption *solarOption control parameters*

Description

solarOption control parameters

Usage

```
control_solarOption(
  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, 0, 1, 1, 0.5))

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

Arguments

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

Examples

```
library(ggplot2)
grid <- seq(-5, 5, 0.01)
pdf_1 <- desscher_mix()(grid, h = 0)
pdf_2 <- desscher_mix()(grid, h = -0.5)
pdf_3 <- desscher_mix()(grid, h = 0.5)
ggplot()+
 geom_line(aes(grid, pdf_1), color = "black")+
 geom_line(aes(grid, pdf_2), color = "green")+
 geom_line(aes(grid, pdf_3), color = "red")

cdf_1 <- pesscher_mix()(grid, h = 0)
cdf_2 <- pesscher_mix()(grid, h = -0.5)
cdf_3 <- pesscher_mix()(grid, h = 0.5)
ggplot()+
  geom_line(aes(grid, cdf_1), color = "black")+
  geom_line(aes(grid, cdf_2), color = "green")+
  geom_line(aes(grid, cdf_3), color = "red")
```

---

detect_season *Detect the 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

`fit_dnorm_mix_em` *Fit the Gaussian mixture parameters with EM algorithm.*

Description

Fit the parameters for the density function of a Gaussian mixture with two components.

Usage

```
fit_dnorm_mix_em(
  x,
  params = NULL,
  abstol = 1e-30,
  maxit = 50000,
  match_moments = FALSE,
  na.rm = FALSE
)
```

Arguments

x	vector
params	initial parameters
maxit	maximum number of iterations.
match_moments	logical. When ‘TRUE‘, the parameters of the second distribution will be estimated such that the empirical first two moments of ‘x‘ matches the theoretical Gaussian mixture moments.
na.rm	logical. When ‘TRUE‘, the default, ‘NA‘ values will be excluded from the computations.
absotol	absolute level for convergence.

Examples

```
t_bar <- 1000
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_em(x, params = params)$par
fit_dnorm_mix_em(x, params = params)$par
```

`fit_dnorm_mix_ML` *Fit the Gaussian mixture parameters with Maximum likelihood.*

Description

Fit the parameters for the density function of a Gaussian mixture with two components.

Usage

```
fit_dnorm_mix_ml(x, params, match_moments = FALSE, na.rm = TRUE)
```

Arguments

`x`	vector
`params`	initial parameters.
`match_moments`	logical. When 'TRUE', the parameters of the second distribution will be estimated such that the empirical first two moments of 'x' matches the theoretical Gaussian mixture moments.
`na.rm`	logical. When 'TRUE', the default, 'NA' values will be excluded from the computations.

Examples

```
t_bar <- 1000
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_ml(x, params = params)$par
fit_dnorm_mix_ml(x, params = params)$par
```

`fit_dnorm_mix_monthly` *Fit a monthly Gaussian Mixture Pdf*

Description

Fit the monthly parameters for the density function of a Gaussian mixture with two components.

Usage

```
fit_dnorm_mix_monthly(x, date, algo = c("em", "ml"), ...)
```

Arguments

x	vector
date	vector of dates
algo	character, type of algorithm. Can be ‘em‘ for Expectation-maximization or ‘ml‘ for maximum likelihood.
loss	loss type. Can be ‘ml‘ for maximum likelihood or ‘kl‘ for kl_dist.
match_moments	when true the theoric moments will match the empirical ones.

---

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)

from_degree_to_radiant(x)
```

Arguments

x	numeric vector, angles in radiant or degrees.

Value

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)
```

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")

`ks_ts_test` *Two sample Kolmogorov Smirnov test for a time series*

Description

Two sample Kolmogorov Smirnov test for a time series

Usage

```
ks_ts_test(
  x,
  ci = 0.01,
  min_quantile = 0.015,
  max_quantile = 0.985,
  seed = 1,
  plot = FALSE
)
```

Arguments

x	a vector.
ci	p.value for rejection.
min_quantile	minimum quantile for the grid of values.
max_quantile	maximum quantile for the grid of values.
seed	random seed.
plot	when ‘TRUE‘ a plot is returned, otherwise a ‘tibble‘.

Value

when ‘plot = TRUE‘ a plot is returned, otherwise a ‘tibble‘.

---

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.

`norm_mix` *Gaussian mixture density*

Description

Probability density function for a Gaussian mixture with two components.

Usage

```
dnorm_mix(params)

pnorm_mix(params)

qnorm_mix(params)

rnorm_mix(n, params)
```

Arguments

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

Value

a density function.

Examples

```
params <- c(mu1 = -2, mu2 = 2, sd1 = 3, sd2 = 1, p = 0.5)
# Density function
pdf <- dnorm_mix(params)
# Distribution function
cdf <- dnorm_mix(params)
# Quantile function
q <- qnorm_mix(params)
# Random numbers
rnorm_mix(100, params)
```

`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. Compute the number of day of the year given a vector of dates. number_of_day("2040-01-31") number_of_day(c("2040-01-31", "2023-04-01", "2015-09-02"))

optimal_n_contracts *Compute optimal number of contracts*

Description

Compute optimal number of contracts

Usage

```
optimal_n_contracts(
  model,
  type = "model",
  premium = "Qr",
  nyear = 2021,
  tick = 0.06,
  efficiency = 0.2,
  n_panels = 2000,
  pun = 0.06
)
```

riccati_root *Riccati Root*

Description

Compute the square root of a symmetric matrix.

Usage

```
riccati_root(x)
```

Arguments

x	matrix, squared and symmetric.

Examples

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

seasonalModel_fit *Fit a seasonal model*

Description

Fit a seasonal model as a linear combination of sine and cosine functions.

Usage

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

Arguments

formula	formula, an object of class 'formula' (or one that can be coerced to that class). It is a symbolic description of the model to be fitted and can be used to include or exclude the intercept or external regressors in 'data'.
order	numeric, of sine and cosine elements.
period	numeric, periodicity. The default is '2*base::pi/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.

seasonalModel_predict *Predict method for the class 'seasonalModel'.*

Description

Predict method for the class 'seasonalModel'.

Usage

```
seasonalModel_predict(object, n = 1)
```

Arguments

object	object of the class 'seasonalModel'.
n	integer, number of day of the year.

`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`	logical; if ‘TRUE‘, probabilities are returned as ‘log(p)‘.
`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, sd = 1)
plot(x, p, type = "l")
# Distribution function
p <- psnorm(x, mean = 0, sd = 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_control — *Control parameters for a ‘solarModel‘ object*

Description

Control parameters for a ‘solarModel‘ object

Usage

```
solarModel_control(
  clearsky.model = clearskyModel_control(),
  mean.model = list(seasonalOrder = 1, arOrder = 2, include.intercept = FALSE),
  variance.model = list(seasonalOrder = 1, match_moments = FALSE),
  threshold = 0.001,
  algo = "em",
  quiet = FALSE
)
```

Arguments

`clearsky.model`	list with control parameters, see ‘clearskyModel_control()‘.
`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.

solarModel_fit — *Fit a model for solar radiation*

Description

Fit a model for solar radiation

Usage

```
solarModel_fit(object, ...)
```

Arguments

`object`	object with class ‘solarModel‘. See the function ‘solarModel_spec()‘. For example ‘solarModel_spec("Bologna")‘.
`...`	additional parameters for the function ‘fit_dnorm_mix_em()‘.

`solarModel_scenario` *Simulate multiple scenarios*

Description

Simulate multiple scenarios of solar radiation with a ‘solarModel‘ object.

Usage

```
solarModel_scenario(
  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`	character, start Date for simulations in the format ‘YYYY-MM-DD‘.
`to`	character, end Date for simulations in the format ‘YYYY-MM-DD‘.
`by`	character, steps for multiple scenarios, e.g. ‘1 day‘ (day-ahead simulations), ‘15 days‘, ‘1 month‘, ‘3 months‘, ecc. For each step are simulated ‘nsim‘ scenarios.
`nsim`	integer, number of simulations.
`lambda`	numeric, Esscher parameter. When ‘rf = FALSE‘, the input parameter ‘lambda‘ will be transformed in negative.
`vol`	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

`solarModel_simulate` *Simulate trajectories*

Description

Simulate trajectories of solar radiation with a ‘solarModel‘ object.

Usage

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

Arguments

`from`	date, starting date for simulations.
`to`	date, end date for simulations.
`nsim`	integer, number of simulations.
`lambda`	numeric, Esscher parameter. When ‘rf = FALSE‘, the input parameter ‘lambda‘ will be transformed in negative.
`vol`	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

`solarModel_spec` *Specification for a ‘solarModel‘ object*

Description

Specification for a ‘solarModel‘ object

Usage

```
solarModel_spec(
  place,
  year_max = NULL,
  from = NULL,
  to = NULL,
  CAMS_data = solarr::CAMS_data,
  control = solarModel_control()
)
```

Arguments

`place`	character, name for the selected location in ‘CAMS_data‘ list.
`year_max`	integer, maximum year in the dataset
`from`	character. Date in the format ‘YYYY-MM-DD‘. Minimum date in the data in ‘CAMS_data‘. If ‘NULL‘ will be used the maximum available.
`to`	character. Date in the format ‘YYYY-MM-DD‘. Maximum date in the data in ‘CAMS_data‘. If ‘NULL‘ will be used the maximum available.
`CAMS_data`	list with radiation data for different locations.
`control`	list with control parameters, see ‘control_solarModel()‘.

`solar_angle_minmax` *Solar angle minimum and maximum*

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 clear sky hourly*

Description

Compute the clear sky radiation for hourly data.

Usage

```
solar_clearsky_hourly(
  cosZ = NULL,
  G0 = NULL,
  altitude = 2.5,
  clime = "No Correction"
)
```

Arguments

cosZ	cosine angle of incidence
G0	extraterrestrial radiation
altitude	altitude in meters.
clime	correction for different climes, can be ‘No Correction‘, ‘Summer‘, ‘Winter‘, ‘Subartic Summer‘, ‘Tropical‘.

Value

a numeric vector containing the time adjustment in minutes.

Examples

```
solar_clearsky_hourly(cosZ = 0.4, G0 = 4, altitude = 2.5, clime = "No Correction")
```

---

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.

solar_extraterrestrial_radiation(42.23, "2022-05-01", day_end = "2022-05-31")

solar_movements *Solar movements*

Usage

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

Arguments

`lat`	latitude
`lon`	longitude
`day_date_time`	vector of dates in the format ‘ \itemday_time_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. 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()
)
```

Arguments

`model`	an object of the class ‘solarModel‘.
`sim`	simulations object.
`control_options`	control function, see ‘control_solarOption‘.
`control`	control function, see ‘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()
)
```

Arguments

model	an object of the class ‘solarModel‘.
nsim	number of simulation to bootstrap.
ci	confidence interval for quantile
seed	random seed.
control	control function, see ‘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()
)
```

Arguments

data	slot ‘data‘ from ‘solarModel‘ object.
nmonth	index for the months.
control	control function, see ‘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,
nmonths	index for the months.
control	control function, see ‘control_solarOption‘.

Examples

```
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()
)
```

Arguments

`sim`	slot ‘sim‘ from ‘solarModel‘ object.
`nmonth`	index for the months.
`nsim`	number of simulation to use.
`control`	control function, see ‘control_solarOption‘.

`solar_option_payoff_structure` *Structure payoffs*

Description

Structure payoffs

Usage

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

Arguments

`model`	an object of the class ‘solarModel‘.
`type`	can be ‘sim‘ or ‘model‘.
`exact_daily_premium`	when ‘TRUE‘ the historical premium is computed as daily average among all the years. Otherwise the monthly premium is computed and then divided by the number of days of the month.

`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.

solar_time_adjustment("2040-01-31") 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.

solar_time_declination("2040-01-01", day_end = "2040-12-31") solar_time_declination(c("2040-01-31", "2023-04-01", "2015-09-02"))

test_normality *Perform normality tests*

Description

Perform normality tests

Usage

```
test_normality(x = NULL, pvalue = 0.05)
```

Arguments

x	numeric, a vector of observation.
pvalue	numeric, the desiderd level of ‘p.value‘ at which the null hypothesis will be rejected.

Value

a tibble with the results of the normality tests.

Examples

```
set.seed(1)
x <- rnorm(1000, 0, 1) + rchisq(1000, 1)
test_normality(x)
x <- rnorm(1000, 0, 1)
test_normality(x)
```

tnorm *Truncated Normal*

Description

Probability for a truncated normal random variable.

Usage

```
dtnorm(x, mean = 0, sd = 1, a = -3, b = 3, log = 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	logical; if ‘TRUE‘, probabilities are returned as ‘log(p)‘.
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")
```

Index