Package ‘solarr’

September 16, 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,
tibble,
np

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),
lubridate (>= 1.8.0),
reticulate (>= 1.24),
nortest,
broom,
formula.tools

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(data, seasonal_data, control = control_clearskyModel())
```

Arguments

data	dataset
seasonal_data	dataset with two columns: 'n' with the number of the day in 1:365 and 'H0' with the extraterrestrial radiation.
control	list of control parameters. See control_clearskyModel for details.

clearskyModel_optimize *Optimizer for Solar Clear sky*

Description

Find the best parameter delta for fitting clear sky radiation.

Usage

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

Arguments

GHI	vector of solar radiation
G0	vector of extraterrestrial radiation
control	list of control parameters. See control_clearskyModel for details.

Value

a numeric vector containing the fitted clear sky radiation.

`clearskyModel_outliers` *clearskyModel_outliers*

Description

clearskyModel_outliers

Usage

```
clearskyModel_outliers(Ct, GHI, date, quiet = FALSE)
```

`control_clearskyModel` *Control parameters for a ‘clearskyModel‘ object*

Description

Control parameters for a ‘clearskyModel‘ object

Usage

```
control_clearskyModel(
  method = "II",
  include.intercept = TRUE,
  order = 1,
  period = 365,
  seed = 1,
  delta_init = 1.4,
  tol = 30,
  lower = 0,
  upper = 2,
  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 ‘365‘.
`seed`	numeric, random seed for reproducibility. It is used to random impute the violations.
`delta_init`	Value for delta init in the clear sky model.
`tol`	integer, tolerance for ‘clearsky > GHI‘ condition. Maximum number of violations admitted.

lower	numeric, lower bound for delta grid.
upper	numeric, upper bound for delta grid.
by	numeric, step for delta grid,
quiet	logical. When ‘FALSE‘, the default, the functions displays warning or messages.

Details

The parametes ‘tol‘, ‘lower‘, ‘upper‘ and ‘by‘ are used exclusively in `clearskyModel_optimize`.

`control_solarEsscher` *Control for Esscher calibration.*

Description

Control parameters for calibration of Esscher parameters.

Usage

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

Arguments

nsim	integer, number of simulations used to bootstrap the premium bounds.
ci	numeric, confidence interval for bootstrapping. See ‘solar_option_payoff_bootstrap()‘.
seed	integer, random seed for reproducibility.
n_key_points	integer, number of key points for interpolation.
init_lambda	numeric, initial value for the Esscher parameter.
lower_lambda	numeric, lower value for the Esscher parameter.
upper_lambda	numeric, upper value for the Esscher parameter.
quiet	logical

control_solarModel — *Control parameters for a 'solarModel' object*

Description

Control function for a solarModel

Usage

```
control_solarModel(
  clearsky.model = control_clearskyModel(),
  mean.model = list(seasonalOrder = 1, arOrder = 2, include.intercept = FALSE,
    monthly.mean = TRUE),
 variance.model = list(seasonalOrder = 1, unconditional_variance = NA, match_moments =
    FALSE, monthly.mean = TRUE, abstol = 1e-20, maxit = 100),
  threshold = 0.01,
  outliers_quantile = 0,
  quiet = FALSE
)
```

Arguments

`clearsky.model` list with control parameters, see `control_clearskyModel` for details.

`mean.model` a list of parameters. Available choices are:

- **'seasonalOrder'** An integer specifying the order of the seasonal component in the model. The default is '1'.
- **'arOrder'** An integer specifying the order of the autoregressive component in the model. The default is '2'.
- **'include.intercept'** When 'TRUE' the intercept will be included in the AR model. The dafault if 'FALSE'.
- **'monthly.mean'** When 'TRUE' a set of 12 monthly means parameters will be computed from the deseasonalized time series to center it perfectly around zero.

`variance.model` a list of parameters.

`threshold` numeric, threshold for the estimation of alpha and beta.

`outliers_quantile` quantile for outliers detection. If different from 0, the observations that are below the quantile at confidence levels 'outliers_quantile' and the observation above the quantile at confidence level 1-'outliers_quantile' will have a weight equal to zero and will be excluded from estimations.

`quiet` logical, when 'TRUE' the function will not display any message.

control_solarOption *Control parameters for a solar option*

Description

Control parameters for a solar option

Usage

```
control_solarOption(
  nyears = c(2005, 2023),
  K = 0,
  put = TRUE,
  leap_year = FALSE,
  B = discountFactor()
)
```

Arguments

nyears	numeric vector. Interval of years considered. The first element will be the minimum and the second the maximum years used in the computation of the fair payoff.
K	numeric, level for the strike with respect to the seasonal mean. The seasonal mean is multiplied by ‘exp(K)‘.
put	logical, when ‘TRUE‘, the default, the computations will consider a ‘put‘ contract. Otherwise a ‘call‘.
leap_year	logical, when ‘FALSE‘, the default, the year will be considered of 365 days, otherwise 366.
B	function. Discount factor function. Should take as input a number (in years) and return a discount factor.

desscher *Compute the Esscher transform of a pdf function*

Description

Compute the Esscher transform of a pdf function

Usage

```
desscher(pdf, theta = 0, lower = -Inf, upper = Inf, ...)
```

Arguments

pdf	density function
theta	esscher parameter
lower	lower bound for domain of the pdf.
upper	upper bound for domain of the pdf.

Value

A density function.

Examples

```
grid <- c(-3,-2,-1,0,1,2,3)
normal_pdf <- function(x) dnorm(x)
esscher_pdf_1 <- desscher_norm(theta = -0.1)
esscher_pdf_2 <- desscher(normal_pdf, theta = -0.1)

# Same result
esscher_pdf_1(grid)
esscher_pdf_2(grid)
```

`desscherMixture` *Esscher transform of a Gaussian Mixture*

Description

Esscher transform of a Gaussian Mixture

Usage

```
desscherMixture(means = c(0, 0), sd = c(1, 1), p = c(0.5, 0.5), theta = 0)

pesscherMixture(means = c(0, 0), sd = c(1, 1), p = c(0.5, 0.5), theta = 0)
```

Arguments

`means`	vector of means parameters.
`sd`	vector of std. deviation parameters.
`p`	vector of probability parameters.
`theta`	Esscher parameter, the default is zero.

Examples

```
library(ggplot2)
grid <- seq(-5, 5, 0.01)
pdf_1 <- desscherMixture(means = c(-3, 3), theta = 0)(grid)
pdf_2 <- desscherMixture(means = c(-3, 3), theta = -0.5)(grid)
pdf_3 <- desscherMixture(means = c(-3, 3), theta = 0.5)(grid)
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 <- pesscherMixture(means = c(-3, 3), theta = 0)(grid)
cdf_2 <- pesscherMixture(means = c(-3, 3), theta = -0.2)(grid)
cdf_3 <- pesscherMixture(means = c(-3, 3), theta = 0.2)(grid)
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*

Description

Detect the season from a vector of dates

Usage

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

Arguments

`day_date`	vector of dates in the format 'YYYY-MM-DD'.

Value

a character vector containing the correspondent season. Can be 'spring', 'summer', 'autumn', 'winter'.

Examples

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

`discountFactor` *Discount factor function*

Description

Discount factor function

Usage

```
discountFactor(r = 0.03, discrete = TRUE)
```

Arguments

`r`	level of yearly constant risk-free rate
`discrete`	logical, when 'TRUE', the default, discrete compounding will be used. Otherwise continuous compounding.

`dmixnorm` *Gaussian mixture random variable*

Description

Gaussian mixture density, distribution, quantile and random generator.

Usage

```
dmixnorm(means = rep(0, 2), sd = rep(1, 2), p = rep(1/2, 2))

pmixnorm(means = rep(0, 2), sd = rep(1, 2), p = rep(1/2, 2))

qmixnorm(means = rep(0, 2), sd = rep(1, 2), p = rep(1/2, 2))

rmixnorm(n, means = rep(0, 3), sd = rep(1, 3), p = rep(1/3, 3), seed = 1)
```

Arguments

`means`	vector of means parameters.
`sd`	vector of std. deviation parameters.
`p`	vector of probability parameters.
`n`	number of simulations
`x`	quantile

Value

A function of x

Examples

```
means = c(-3,0,3)
sd = rep(1, 3)
p = c(0.2, 0.3, 0.5)
# Density function
dmixnorm(means, sd, p)(3)
# Distribution function
dmixnorm(means, sd, p)(c(1,2,-3))
# Quantile function
qmixnorm()(0.2)
# Random numbers
rmixnorm(1000)
```

dsolarGHI *Density function for the GHI*

Description

Density function for the GHI

Distribution function for the GHI

Quantile function for the GHI

Random generator function for the GHI

Usage

```
dsolarGHI(x, Ct, alpha, beta, pdf_Yt)

psolarGHI(x, Ct, alpha, beta, pdf_Yt)

qsolarGHI(p, Ct, alpha, beta, pdf_Yt)

rsolarGHI(n, Ct, alpha, beta, pdf_Yt)
```

Arguments

`x, p`	value or probability.
`Ct`	clear sky radiation
`alpha`	transform params
`beta`	transform params
`pdf_Yt`	density of Yt

Examples

```
dsolarGHI(5, 7, 0.001, 0.9, function(x) dnorm(x))
dsolarGHI(6.993, 7, 0.001, 0.9, function(x) dnorm(x))
psolarGHI(6.993, 7, 0.001, 0.9, function(x) dnorm(x))
qsolarGHI(1, 7, 0.001, 0.9, function(x) dnorm(x))
qsolarGHI(c(0.05, 0.95), 7, 0.001, 0.9, function(x) dnorm(x))
rsolarGHI(10, 7, 0.001, 0.9, function(x) dnorm(x))
```

esscher_norm *Esscher density of a Gaussian random variable*

Description

Esscher density of a Gaussian random variable

Usage

```
desscher_norm(mean = 0, sd = 1, theta = 0)

pesscher_norm(mean = 0, sd = 1, theta = 0)
```

Arguments

mean	mean
sd	std. deviation
theta	Esscher parameter

Value

A density or distribution function.

Examples

```
grid <- seq(-3, 3, 0.5)
# Density
pdf <- desscher_norm(theta = -0.1)
pdf(grid)
desscher_norm(theta = 0.1)(grid)
# Distribution
cdf <- pesscher_norm(theta = -0.1)
cdf(grid)
pesscher_norm(theta = 0.1)(grid)
```

---

gaussianMixture *Gaussian mixture*

Description

Fit the parameters of a gaussian mixture with k-components.

Usage

```
gaussianMixture(
  x,
  means,
  sd,
  p,
  components = 2,
  prior_p = rep(NA, components),
  weights,
  maxit = 100,
  abstol = 1e-14,
  na.rm = FALSE
)
```

Arguments

x	vector
means	vector of initial means parameters.
sd	vector of initial std. deviation parameters.
p	vector of initial probability parameters.

components	number of components.
prior_p	prior probability for the k-state. If the k-component is not ‘NA‘ the probability will be considered as given and the parameter ‘p[k]‘ will be equal to ‘prior_p[k]‘.
weights	observations weights, if a weight is equal to zero the observation is excluded, otherwise is included with unitary weight. When ‘missing‘ all the available observations will be used.
maxit	maximum number of iterations.
na.rm	logical. When ‘TRUE‘, the default, ‘NA‘ values will be excluded from the computations.
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.
absotol	absolute level for convergence.

Value

list with clustered components and the optimal parameters.

Examples

```
means = c(-3,0,3)
sd = rep(1, 3)
p = c(0.2, 0.3, 0.5)
# Density function
pdf <- dmixnorm(means, sd, p)
# Distribution function
cdf <- pmixnorm(means, sd, p)
# Random numbers
x <- rgaussianMixture(1000, means, sd, p)
gaussianMixture(x$X, means, sd, p, components = 3)
gaussianMixture(x$X, means, sd, prior_p = p, components = 3)
```

---

gaussianMixture_monthly

Fit a monthly Gaussian Mixture Pdf (??NOT USED)

Description

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

Usage

```
gaussianMixture_monthly(x, date, means, sd, p, components = 2, prior_p, ...)
```

Arguments

x	vector
date	vector of dates
means	matrix of initial means with dimension ‘12 X components‘.
sd	matrix of initial std. deviations with dimension ‘12 X components‘.

p	matrix of initial p with dimension ‘12 X components‘. The rows must sum up to 1.
prior_p	matrix of prior probabilities for the each month. Any element that is different from ‘NA‘ will be not optimized and will be considered as given.
...	other parameters for the optimization function. See gaussianMixture for more details.

---

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")
```

`havDistance` *Haversine distance*

Description

Compute the Haversine distance between two points.

Usage

```
havDistance(lat_1, lon_1, lat_2, lon_2)
```

Arguments

`lat_1`	numeric, latitude of first point.
`lon_1`	numeric, longitude of first point.
`lat_2`	numeric, latitude of second point.
`lon_2`	numeric, longitude of second point.

Value

Numeric vector the distance in kilometers.

Examples

```
havDistance(43.3, 12.1, 43.4, 12.2)
havDistance(43.35, 12.15, 43.4, 12.2)
```

`IDW` *Inverse Distance Weighting Function*

Description

Return a distance weighting function

Usage

```
IDW(beta, d0)
```

Arguments

beta	parameter used in exponential and power functions.
d0	parameter used only in exponential function.

Details

When the parameter ‘d0‘ is not specified the function returned will be of power type otherwise of exponential type.

Examples

```
# Power weighting
IDW_pow <- IDW(2)
IDW_pow(c(2, 3,10))
IDW_pow(c(2, 3,10), normalize = TRUE)
# Exponential weighting
IDW_exp <- IDW(2, d0 = 5)
IDW_exp(c(2, 3,10))
IDW_exp(c(2, 3,10), normalize = TRUE)
```

`is_leap_year` *Is leap year?*

Description

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

Usage

```
is_leap_year(x)
```

Arguments

x	numeric value or dates vector in the format ‘YYYY-MM-DD‘.

Value

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

Examples

```
is_leap_year("2024-02-01")
is_leap_year(c(2023:2030))
is_leap_year(c("2024-10-01", "2025-10-01"))
is_leap_year("2029-02-01")
```

---

kernelRegression *Kernel regression*

Description

Fit a kernel regression.

Usage

```
kernelRegression(formula, data, ...)
```

Arguments

formula	formula
data	data
...	other parameters to be passed to. See np::npreg.

---

ks_test *Kolmogorov Smirnov test for a distribution*

Description

Kolmogorov Smirnov test for a distribution

Usage

```
ks_test(
  x,
  cdf,
  ci = 0.05,
  min_quantile = 0.015,
  max_quantile = 0.985,
  k = 1000,
  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.
k	finite value for approximation of infinite sum.
plot	when ‘TRUE‘ a plot is returned, otherwise a ‘tibble‘.
pdf	the theoric density function to use for comparison.

Value

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

`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 = 0.05,
  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.

Location *Generate a location*

Description

Generate a location

Usage

```
Location(
  place,
  nsim = 50,
  by = "1 month",
  exact_daily_premium = FALSE,
  measures = c("Q", "Qdw", "Qup"),
  control_model = control_solarModel(),
  control_options = control_solarOption(),
  control_esscher = control_solarEsscher(),
  seed = 1
)
```

---

makeSemiPositive *Make a matrix semi-definite positive*

Description

Make a matrix semi-definite positive

Usage

```
makeSemiPositive(x, neg_values = 1e-10)
```

Arguments

x	matrix, squared and symmetric.
neg_values	numeric, the eigenvalues lower the zero will be substituted with this value.

Examples

```
m <- matrix(c(2, 1.99, 1.99, 2), nrow = 2, byrow = TRUE)
makeSemiPositive(m)
```

`number_of_day` *Number of day*

Description

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

Usage

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

Arguments

`day_date`	dates vector in the format 'YYYY-MM-DD'.

Value

Numeric vector with the number of the day during the year.

Examples

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

`optionPayoff` *Option payoff function*

Description

Compute the payoffs of an option at maturity.

Usage

```
optionPayoff(x, strike = 0, v0 = 0, put = TRUE)
```

Arguments

`x`	numeric, vector of values at maturity.
`strike`	numeric, option strike.
`v0`	numeric, price of the option.
`put`	logical, when 'TRUE', the default, the payoff function is a put othewise a call.

Examples

```
optionPayoff(10, 9, 1, put = TRUE)
mean(optionPayoff(seq(0, 20), 9, 1, put = TRUE))
```

`pks` *Kolmogorov distribution function*

Description

Kolmogorov distribution function

Usage

```
pks(x, k = 100)
```

Arguments

x	a vector.
k	finite value for approximation of infinite sum.

Value

A probability, a numeric vector in 0, 1.

`qks` *Kolmogorov quantile function*

Description

Kolmogorov quantile function

Usage

```
qks(p, k = 100)
```

Arguments

k	finite value for approximation of infinite sum.
x	a vector of probabilities.

Value

A positive number.

`radiant` *Conversion in Radiant or Degrees*

Description

Convert angles in radiant into an angles 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 with angles in radiant or degrees.

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)
```

`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 a seasonal model*

Description

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

Usage

```
seasonalModel(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.

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")
```

---

solarEsscher_bounds *Calibrate Esscher Bounds and parameters*

Description

Calibrate Esscher Bounds and parameters

Usage

```
solarEsscher_bounds(
  model,
  control_options = control_solarOption(),
  control_esscher = control_solarEsscher()
)
```

Arguments

model	object with the class ‘solarModel‘. See the function solarModel for details.
control_options	control function. See control_solarOption for details.
control_esscher	control function. See control_solarEsscher for details.

solarEsscher_calibrator

Calibrator for Esscher parameter

Description

Calibrator for Esscher parameter

Usage

```
solarEsscher_calibrator(
  model,
  target_price,
  nmonths = 1:12,
  control_esscher = control_solarEsscher(),
  control_options = control_solarOption()
)
```

Arguments

model	object with the class 'solarModel'. See the function solarModel for details.
target_price	target price for the calibration.
nmonths	months used in the model computation.
control_esscher	control function. See control_solarEsscher for details.
control_options	control function. See control_solarOption for details.

solarEsscher_calibrator_month

Calibrate monthly Esscher parameter given the expected return

Description

Calibrator function for the monthly Esscher parameter of a solarOption given a desired level of expected return at maturity.

Usage

```
solarEsscher_calibrator_month(
  model,
  nmonth = 1,
  expected_return = 0,
  target_price = NA,
  control_esscher = control_solarEsscher(),
  control_options = control_solarOption()
)
```

Arguments

`model`	solar model
`nmonth`	month
`expected_return`	expected return at maturity. The benchmark for the ‘target_price‘ to match will be the mean cumulated net payoff on the last day of the month plus the model price paid under the Esscher measure. The return of the ‘target_price‘ with respect to the model price will match the parameter ‘expected_return‘. For example ‘0.01‘, ‘0.02‘, ecc.
`target_price`	alternative to the ‘expected_return‘ parameter. Submitting a ‘target_price‘ will imply that the ‘expected_return = 0‘ so that the model price under the Esscher measure matches the ‘target_price‘
`control_esscher`	control
`control_options`	control

`solarModel` *Fit a model for solar radiation*

Description

Fit a model for solar radiation

Usage

```
solarModel(spec)
```

Arguments

`spec`	an object with class ‘solarModelSpec‘. See the function `solarModel_spec` for details.

Examples

```
control <- control_solarModel(outliers_quantile = 0.0005)
spec <- solarModel_spec("Berlino", from="2005-01-01", to="2024-01-01", control_model = control)
model <- solarModel(spec)
```

solarModel_calibrator *Calibrator for solar Options*

Description

Calibrator for solar Options

Usage

```
solarModel_calibrator(
  model,
  nmonths = 1:12,
  control_options = control_solarOption()
)
```

solarModel_empiric_GM *Empiric Gaussian Mixture parameters*

Description

Empiric Gaussian Mixture parameters

Usage

```
solarModel_empiric_GM(model, match_moments = FALSE)
```

solarModel_loglik *Compute the log-likelihood of a ‘solarModel‘ object*

Description

Compute the log-likelihood of a ‘solarModel‘ object

Usage

```
solarModel_loglik(model, target = c("Yt", "GHI"), nmonths = 1:12)
```

Arguments

model	‘solarModel‘ object
nmonths	months to consider

---

solarModel_parameters *Extract the parameters of a ‘solarModel‘*

Description

Extract the parameters of a ‘solarModel‘

Usage

```
solarModel_parameters(model, as_tibble = FALSE)
```

Arguments

model	object with the class ‘solarModel‘. See the function solarModel for details.
as_tibble	logical, when ‘TRUE‘ the output will be converted in a tibble.

Value

a named list with all the parameters

Examples

```
spec <- solarModel_spec("Ferrara", from="2005-01-01", to="2020-01-01")
model <- solarModel(spec)
solarModel_parameters(model)
```

---

solarModel_scenario *Simulate multiple scenarios*

Description

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

Usage

```
solarModel_scenario(
  model,
  from = "2010-01-01",
  to = "2010-12-31",
  by = "1 month",
  nsim = 1,
  lambda = 0,
  seed = 1,
  quiet = FALSE
)
```

Arguments

model	object with the class ‘solarModel‘. See the function solarModel for details.
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.
seed	scalar integer, starting random seed.
quiet	logical

Examples

```
spec <- solarModel_spec("Ferrara", from="2005-01-01", to="2020-01-01")
model <- solarModel(spec)
solarModel_scenario(model, from = "2010-01-01", to = "2010-12-31", nsim = 2, by = "1 month")
```

---

solarModel_simulate *Simulate trajectories*

Description

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

Usage

```
solarModel_simulate(
  model,
  from = "2010-01-01",
  to = "2010-12-31",
  nsim = 1,
  lambda = 0,
  seed = 1,
  exclude_known = FALSE,
  quiet = FALSE
)
```

Arguments

model	object with the class ‘solarModel‘. See the function solarModel for details.
from	character, start Date for simulations in the format ‘YYYY-MM-DD‘.
to	character, end Date for simulations in the format ‘YYYY-MM-DD‘.
nsim	integer, number of simulations.
lambda	numeric, Esscher parameter.
seed	scalar integer, starting random seed.
quiet	logical

Examples

```
spec <- solarModel_spec("Ferrara", from="2005-01-01", to="2020-01-01")
model <- solarModel(spec)
solarModel_simulate(model, from = "2010-01-01", to = "2010-12-31", nsim = 1)
```

---

solarModel_spec   *Specification function for a 'solarModel'*

Description

Specification function for a 'solarModel'

Usage

```
solarModel_spec(
  place,
  min_date,
  max_date,
  from,
  to,
  CAMS_data = solarr::CAMS_data,
  control_model = control_solarModel()
)
```

Arguments

place	character, name of an element in the 'CAMS_data' list.
min_date	character. Date in the format 'YYYY-MM-DD'. Minimum date for the complete data. If 'missing' will be used the minimum data available.
max_date	character. Date in the format 'YYYY-MM-DD'. Maximum date for the complete data. If 'missing' will be used the maximum data available.
from	character. Date in the format 'YYYY-MM-DD'. Starting date to use for training data. If 'missing' will be used the minimum data available after filtering for 'min_date'.
to	character. Date in the format 'YYYY-MM-DD'. Ending date to use for training data. If 'missing' will be used the maximum data available after filtering for 'max_date'.
CAMS_data	named list with radiation data for different locations.
control_model	list with control parameters, see control_solarModel for details.

solarModel_test_residuals

Stationarity and distribution test (Gaussian mixture) for a 'solarModel'

Description

Stationarity and distribution test (Gaussian mixture) for a 'solarModel'

Usage

```
solarModel_test_residuals(model, seed = 1, nrep = 500, ...)
```

solarModel_update_GM

Update Gaussian Mixture parameters for a given month

Description

Update Gaussian Mixture parameters for a given month

Usage

```
solarModel_update_GM(model, params, nmonth)
```

solarModel_update_params

Update the parameters of a 'solarModel' object

Description

Update the parameters of a 'solarModel' object

Usage

```
solarModel_update_params(model, params)
```

Arguments

model	'solarModel' object
params	named list of parameters. See the function solarModel_parameters to structure the list of new parameters.

`solarOption_bootstrap` *Bootstrap a fair premium from historical data*

Description

Bootstrap a fair premium from historical data

Usage

```
solarOption_bootstrap(
  model,
  nsim = 500,
  ci = 0.05,
  seed = 1,
  control_options = control_solarOption()
)
```

Arguments

`model`	object with the class ‘solarModel‘. See the function `solarModel` for details.
`nsim`	number of simulation to bootstrap.
`ci`	confidence interval for quantile
`seed`	random seed.
`control_options`	control function, see `control_solarOption` for details.

Value

An object of the class ‘solarOptionPayoffBoot‘.

`solarOption_contracts` *Optimal number of contracts*

Description

Compute the optimal number of contracts given a particular setup.

Usage

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

Arguments

`model`	object with the class ‘solarModel‘. See the function `solarModel` for details.
`type`	character, method used for computing the premium. Can be ‘model‘ (Model with integral) or ‘sim‘ (Monte Carlo).
`premium`	character, premium used. Can be ‘P‘, ‘Qdw‘, ‘Qup‘, or ‘Q‘.
`nyear`	integer, actual year. The optimization will be performed excluding the year ‘nyear‘ and the following.
`tick`	numeric, conversion tick for the monetary payoff of a contract.
`efficiency`	numeric, mean efficiency of the solar panels.
`n_panels`	numeric, number of meters squared of solar panels.
`pun`	numeric, reference electricity price at which the energy is sold for computing the cash-flows.

`solarOption_historical` *Payoff on Historical Data*

Description

Payoff on Historical Data

Usage

```
solarOption_historical(
  model,
  nmonths = 1:12,
  control_options = control_solarOption()
)
```

Arguments

`model`	object with the class ‘solarModel‘. See the function `solarModel` for details.
`nmonths`	numeric, months of which the payoff will be computed.
`control_options`	control list, see `control_solarOption` for more details.

---

solarOption_implied_return *Implied expected return at maturity*

Description

Implied expected return at maturity

Usage

```
solarOption_implied_return(
  model,
  target_prices = NA,
  nmonths = 1:12,
  control_options = control_solarOption()
)
```

---

solarOption_model *Pricing function under the solar model*

Description

Pricing function under the solar model

Usage

```
solarOption_model(
  model,
  nmonths = 1:12,
  theta = 0,
  implvol = 1,
  control_options = control_solarOption()
)
```

Arguments

model	object with the class ‘solarModel‘. See the function solarModel for details.
nmonths	numeric, months of which the payoff will be computed.
theta	Esscher parameter
implvol	implied unconditional GARCH variance, the default is ‘1‘.
control_options	control list, see control_solarOption for more details.

`solarOption_model_spatial` *Pricing function under the solar model*

Description

Pricing function under the solar model

Usage

```
solarOption_model_spatial(
  object,
  lat,
  lon,
  nmonths = 1:12,
  theta = 0,
  implvol = 1,
  control_options = control_solarOption()
)
```

Arguments

`object`	a ‘spatialModel‘ object
`lat`	numeric, latitude of the point.
`lon`	numeric, longitude of the point.
`nmonths`	numeric, months of which the payoff will be computed.
`theta`	Esscher parameter
`implvol`	implied unconditional GARCH variance, the default is ‘1‘.
`control_options`	control list, see `control_solarOption` for more details.

`solarOption_scenario` *Payoff on Simulated Data*

Description

Payoff on Simulated Data

Usage

```
solarOption_scenario(
  sim,
  nmonths = 1:12,
  nsim,
  control_options = control_solarOption()
)
```

Arguments

`sim`	simulated scenarios with the function `solarModel_scenarios`.
`nmonths`	numeric, months of which the payoff will be computed.
`nsim`	number of simulation to use for computation.
`control_options`	control function, see `control_solarOption` for details.

`solarOption_structure` *Structure payoffs*

Description

Structure payoffs

Usage

```
solarOption_structure(model, type = "model", exact_daily_premium = TRUE)
```

Arguments

`model`	object with the class ‘solarModel‘. See the function `solarModel` for details.
`type`	method used for computing the premium. If ‘model‘, the default will be used the analytic model, otherwise with ‘sim‘ the monte carlo scenarios stored inside the ‘model$scenarios$P‘.
`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.

`solarRiskDriver` *Compute Solar Risk driver*

Description

Compute Solar Risk driver

Usage

```
solarRiskDriver(GHI, Ct)
```

Arguments

`GHI`	radiation time series
`Ct`	clear sky radiation time series

Details

The function computes:

$$x_t = 1 - \frac{GHI}{C_t}$$

Value

A risk drivers time series.

Examples

```
solarRiskDriver(8, 12)
solarRiskDriver(11, 12)
```

`solarTransform` *Solar Model transformation functions*

Description

Solar Model transformation functions

Solar Model transformation functions

Methods

Public methods:

- `solarTransform$new()`
- `solarTransform$Yt()`
- `solarTransform$Xt()`
- `solarTransform$GHI()`
- `solarTransform$Ct()`
- `solarTransform$Yt_bar()`
- `solarTransform$Xt_bar()`
- `solarTransform$GHI_bar()`
- `solarTransform$clone()`

Method `new()`: Solar Model transformation functions

Usage:

```
solarTransform$new(params, seasonal_model_Ct, seasonal_model_Yt)
```

Arguments:

`params` bounds parameters

`seasonal_model_Ct` seasonal model clearsky.

`seasonal_model_Yt` seasonal model Yt.

Method `Yt()`: Transformation from Xt to Yt

Usage:

```
solarTransform$Yt(Xt)
```

Arguments:

`Xt` risk driver in (alpha, alpha+beta)

Method `Xt()`**:** Transformation from Yt to Xt

Usage:

```
solarTransform$Xt(Yt)
```

Arguments:

`Yt` transformed risk driver in (-Inf, Inf)

Method `GHI()`**:** Solar radiation function

Usage:

```
solarTransform$GHI(x, t)
```

Arguments:

`x` risk driver in (alpha, alpha+beta).

`t` time index, number of day of the year.

Method `Ct()`**:** Seasonal function clear sky radiation

Usage:

```
solarTransform$Ct(t)
```

Arguments:

`t` time index, number of day of the year.

Method `Yt_bar()`**:** Seasonal function transformed risk driver

Usage:

```
solarTransform$Yt_bar(t)
```

Arguments:

`t` time index, number of day of the year.

Method `Xt_bar()`**:** Seasonal function risk driver

Usage:

```
solarTransform$Xt_bar(t)
```

Arguments:

`t` time index, number of day of the year.

Method `GHI_bar()`**:** Seasonal function solar radiation

Usage:

```
solarTransform$GHI_bar(t)
```

Arguments:

`t` time index, number of day of the year.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
solarTransform$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

solarTransform_GHI *Solar Model transformation function for GHI*

Description

Solar Model transformation function for GHI

Usage

```
solarTransform_GHI(x, Ct)
```

Arguments

x	risk driver time series in (0,1)
Ct	clear sky radiation time series

Value

A radiation time series.

Examples

```
Xt <- solarRiskDriver(8, 12)
solarTransform_GHI(Xt, 12)
```

solarTransform_params *Solar Model transformation from Xt to Yt*

Description

Compute optimal parameters given the threshold.

Usage

```
solarTransform_params(x, threshold = 0.01)
```

Arguments

x	series of Xt
threshold	param

solarTransform_Xt *Transformation function from Yt to Xt*

Description

Transformation function from Yt to Xt

Usage

```
solarTransform_Xt(Yt, alpha, beta)
```

Arguments

alpha	param
beta	param
y	transformed time series in (-infty, infty)

Examples

```
Yt <- solarTransform_Yt(0.5, 0.01, 0.9)
solarTransform_Xt(Yt, 0.01, 0.9)
```

solarTransform_Yt *Transformation function from Xt to Yt*

Description

Transformation function from Xt to Yt

Usage

```
solarTransform_Yt(x, alpha, beta)
```

Arguments

x	risk driver time series in (0,1)
alpha	param
beta	param

Examples

```
solarTransform_Yt(0.5, 0.01, 0.9)
solarTransform_Yt(0.5, 0.02, 0.94)
```

solar_angle_minmax *Solar angle minimum and maximum*

Description

Compute the solar angle for a latitude in different dates.

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 ‘YYYY-MM-DD‘.
day_end	end date, if it is not NULL will be end date.
method	method used for computation of solar declination, can be ‘cooper‘ or ‘spencer‘.

Value

a tibble.

Examples

```
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_extraterrestrial_radiation *Solar extraterrestrial radiation*

Description

Compute the solar angle for a latitude in different times of the day.

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 ‘YYYY-MM-DD‘
day_end	end date, if it is not NULL will be end date.
method	method used for computation of solar declination, can be ‘cooper‘ or ‘spencer‘.

Value

a numeric vector containing the time adjustment in minutes.

Examples

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

---

solar_monthly_mixture *Monthly Gaussian mixture with two components*

Description

Monthly Gaussian mixture with two components

Usage

```
solar_monthly_mixture(x, date, weights, match_moments = FALSE, prior_p, ...)
```

Arguments

x	arg
date	arg
weights	arg
match_moments	arg
...	arg

---

solar_movements *Solar movements*

Description

Compute the solar angle for a latitude in different times of the day.

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 ‘YYYY-MM-DD HH:MM:SS‘
`day_time_end`	end date, if it is not NULL will be end date.
`method`	method used for computation of solar declination, can be ‘cooper‘ or ‘spencer‘.

Value

a numeric vector containing the time adjustment in minutes.

Examples

```
solar_movements(44.23, 11.20, day_date_time = "2040-01-01", day_time_end = "2040-01-03")
```

---

solar_time_adjustment *Solar time adjustment*

Description

Compute the time adjustment for a date.

Usage

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

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.

Examples

```
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*

Description

Compute the solar declination for different dates.

Usage

```
solar_time_declination(
  day_date = NULL,
  day_end = NULL,
  method = c("cooper", "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 declination in minutes.

Examples

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

---

spatialGrid *Spatial Grid*

Description

Create a grid from a range of latitudes and longitudes.

Usage

```
spatialGrid(lat = c(43.7, 45.1), lon = c(9.2, 12.7), by = c(0.1, 0.1))
```

Arguments

`by`	step for longitudes and latitudes. If two values are specified the first will be used for latitudes and the second for longitudes
`range_lat`	vector with latitudes. Only the minimum and maximum values are considered.
`range_lon`	vector with longitudes. Only the minimum and maximum values are considered.

Value

a tibble with two columns ‘lat‘ and ‘lon‘.

Examples

```
spatialGrid(lat = c(43.7, 43.8), lon = c(12.5, 12.7), by = 0.1)
spatialGrid(lat = c(43.7, 43.75, 43.8), lon = c(12.6, 12.6, 12.7), by = c(0.05, 0.01))
```

---

spatialModel *Spatial model object*

Description

Spatial model object

Usage

```
spatialModel(locations, solarModels)
```

Arguments

`locations`	grid of locations
`solarModels`	list of ‘solarModel‘ objects

`spatialModel_combinations` *Compute all possible states*

Description

Compute all possible states

Usage

```
spatialModel_combinations(object, lat, lon)
```

Arguments

`object`	a ‘spatialModel‘ object
`lat`	numeric, latitude of the point.
`lon`	numeric, longitude of the point.

`spatialModel_interpolate` *Compute a solar model for a location*

Description

Compute a solar model for a location

Usage

```
spatialModel_interpolate(object, lat, lon, n = 4, quiet = FALSE, ...)
```

Arguments

`object`	a ‘spatialModel‘ object
`lat`	numeric, latitude of the point.
`lon`	numeric, longitude of the point.
`n`	number of neighborhoods
`quiet`	logical

spatialModel_interpolate_GHI

Interpolate the solar radiation for a location

Description

Interpolate the solar radiation for a location

Usage

```
spatialModel_interpolate_GHI(
  object,
  lat,
  lon,
  n = 4,
  day_date,
  quiet = FALSE,
  ...
)
```

Arguments

object	a ‘spatialModel‘ object
lat	numeric, latitude of the point.
lon	numeric, longitude of the point.
n	number of neighborhoods
day_date	day date for interpolation. If missing all the available dates will be used.
quiet	logical

spatialModel_neighborhoods

Find the n-closest neighborhoods of a point

Description

Find the n-closest neighborhoods of a point

Usage

```
spatialModel_neighborhoods(object, lat, lon, n = 4, beta = 2, d0)
```

Arguments

object	a ‘spatialModel‘ object
lat	numeric, latitude of the point.
lon	numeric, longitude of the point.
n	number of neighborhoods
beta	parameter used in exponential and power functions.
d0	parameter used only in exponential function.

spatialParameters *Spatial kernel regression*

Description

Fit kernel regression on all the parameters of a list containing ‘solarModels‘ at different coordinates.

Usage

```
spatialParameters(solarModels, quiet = FALSE)
```

Arguments

solarModels	a list containing ‘solarModels‘ objects.
quiet	logical

spatialParameters_predict *Predict method*

Description

Predict method for the class ‘spatialParameters‘.

Usage

```
spatialParameters_predict(object, lat, lon, as_tibble = FALSE, quiet = FALSE)
```

Arguments

object	an object of the class ‘spatialParameters‘. See clearskyModel.
lat	numeric latitude of the locations.
lon	numeric longitude of the locations.

spectralDistribution *Compute the spectral distribution for a black body*

Description

Compute the spectral distribution for a black body

Usage

```
spectralDistribution(lambda = NULL, measure = "nanometer")
```

Arguments

lambda	numeric, wave length in micrometers.
measure	character, measure of the irradiated energy. If ‘nanometer‘ the final energy will be in W/m2 x nanometer, otherwise if ‘micrometer‘ the final energy will be in W/m2 x micrometer.

`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")
```