# Package ‘reprDays’

October 27, 2021

**Version** 1.0

**Title** Representative days using PCA and hierarchical clustering

**Description**

Generates representative days for each season for a single region or across different regions. Methods are Principal Component Analysis (PCA) and for comparison also hierarchical clustering

**Depends** R (>= 3.1.0)

**Imports** corrplot, copula, fitdistrplus, spatstat, xts

**URL** `https://gitlab.psi.ch/energy-economics-group/representative-days`

**License** BSD

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Martin Densing [aut, cre] (<https://orcid.org/X>),
Yi Wan [ctb] (<https://orcid.org/X>)

**Maintainer** Martin Densing <martin.densing@psi.ch>

## R topics documented:

---

`reprDays-package` *reprDays: A package for computating representative days*

---

## Description

Package for article *Low-dimensional scenario generation method of solar and wind availability for representative days in energy modeling*, Applied Energy, in press. `https://doi.org/X`

## Details

The package provides different categories of functions:

### Import

- Read hourly solar and wind generation data, and optionally load data
  - Current data is from ENTSO-E's transparency platform. Data from countries DE and AT is converted to hourly. Data should all be in CET time-zone.
  - Provided data is of year 2017-2019 (State: Dec 2021)
  - Absolute wind and solar is converted to availabilities by using an auxiliary capacity data file
  - Absolute load data is normalized by dividing with high quantiles.
- Plot time-series of solar, wind (and load). Plot daily profiles per season and per country
- Correlations of wind, solar, and load

### PCA factor model

- Execute the PCA on the data vector
- Make a principal compoment factor analysis
- Generate representaive-days scenarios using the factor models,and plot the scenarios

### Analysis of accuracy of scenarios

Compare the scnearios with empirical data and incraeasing scenario number:

- Compare correlations
- Compare ECDF with empirical data
- Compare availability per season

### Hierarchical clustering using medoids

Generate representative days by clustering. Given a season, single and cross-regional scenarios can be generated

### Cross-regional scenarios

- Analysis of daily wind and solar correlations across regions (countries)
- Estimation of joint extreme solar and wind availabilities
- Fitting guassian copulas and t-copulas to daily cross-regional wind and solar data
- Random sampling of copulas
- Given a season, join daily cross-regional random samples with (aggregated from hourly to daily) regional scenarios to provide consistent hourly scenarios across regions

### Auxiliary function (selection)

- Aggregate availabilities to 96 (24*4) load-period availabilities and write to "avlBEM.csv" for use in BEM

## agggregate.avl.BEM *Average wind and solar availability to the 96 loadperiods of BEM per country*

### Description

A convenience function for BEM. Not required for PCA or clustering.

### Usage

```
agggregate.avl.BEM(xSWH)
```

### Arguments

| | |
|---|---|
| xSWH | Data in wide format, as returned by x.24wide |

### Value

Data.frame of solar and wind availability for each load period and for each country

- rows: 5 * 96 (countries x seasons x hours), e.g., "AT.WI-D-01"
- columns: "solar", "wind"

## avg.Cols *Replace some data columns by a single, averaged column*

### Description

Replace some data columns by a single, averaged column

### Usage

```
avg.Cols(
  x,
  avgCols = c("ATsolar", "DEsolar", "CHsolar", "ITsolar", "FRsolar"),
  newColname = "solar"
)
```

### Arguments

| | |
|---|---|
| x | xts time-series |
| avgCols | Data columns to be merged |
| newColname | Name of merged column |

## `avl.emp` *Seasonal availability factors from empirical data*

### Description

Seasonal availability factors from empirical data

### Usage

```
avl.emp(x)
```

### Arguments

| | |
|---|---|
| `x` | time-series of data (xts) |

## `avl.PCA` *Seasonal availability factors implied by PCA*

### Description

Seasonal availability factors implied by PCA

### Usage

```
avl.PCA(scnBEM)
```

### Arguments

| | |
|---|---|
| `scnBEM` | Data-frame |

### Value

avlPCA Array

## `binomSd` *Discretized standard binomial distribution*

### Description

Discretized standard binomial distribution

### Usage

```
binomSd(j)

binomSdProb(j)
```

### Arguments

| | |
|---|---|
| `j` | Number of disretization (yields j+1 values) |

### Value

`binomSd` returns values (j=0 yields 0); `binomSdProb` returns probabilities

### Examples

```
bimonSd(2)
binomSdProb(2)
```

## `clust.medoid` *Get the medoid of each cluster*

### Description

Get the medoid of each cluster

### Usage

```
clust.medoid(i, distmat, clusters)
```

### Arguments

| | |
|---|---|
| `i` | Index |
| `distmat` | Distance matrix between points |
| `clusters` | Clusters |

## `contour.plot` *Contour plot of correlation matrix*

### Description

Contour plot of correlation matrix

### Usage

```
contour.plot(correl, title.cont, cty, s, triang = TRUE, oldVer = FALSE)
```

### Arguments

| | |
|---|---|
| `correl` | Correlation matrix |
| `title.cont` | Title of contour plot |
| `cty` | Country or |-separated list of countries, used for labeling |
| `s` | Season, used for labeling |
| `triang` | Plot only upper triangle? (default: TRUE) |
| `oldVer` | Plot old version with other color codes, whereas new version asfixed levels? (default: TRUE) |

## `contour.wide24` *Contour plot of correlations of data in wide format*

### Description

Contour plot of correlations of data in wide format

### Usage

```
contour.wide24(
  xSWH,
  cty = "DE|IT",
  seas = c("SP", "SU"),
  yrRange = "2017-2019",
  upper.triangle = TRUE,
  subrange = FALSE,
  old.version = FALSE,
  plot2pdf = FALSE
)
```

### Arguments

| | |
|---|---|
| `xSWH` | Data in wide format, as returned by `x.24wide` |
| `cty` | Country, single or multiple countries (default: `"DE|IT"`) |
| `seas` | Season (single or multiple seasons (default: `c("SP","SU")`) |
| `yrRange` | Used in title of plot (default: `"2017-2019"`) |
| `upper.triangle` | Plot only upper triangle? (default: TRUE) |
| `subrange` | Plot only correlation between solar and wind (and not e.g. between hours of solar)? (default: FALSE) |
| `old.version` | Plot old version with other color codes, whereas new version asfixed levels? (default: TRUE) |

### Examples

```
## Not run:
contour.wide24(l.wide24, "DE", "SU", "2017-2019")

## End(Not run)
```

## daily.cor.cross *Daily cor-matrix of wind a solar between countries*

### Description

Daily cor-matrix of wind a solar between countries

### Usage

```
daily.cor.cross(x, seas = seasNames, plot2pdf = FALSE)
```

### Arguments

| | |
|---|---|
| x | Time series (xts class) containing hourly data with column names "ATsolar", etc. |
| seas | Season (seas = seasNames gives whole year) |

### Value

correlation matrix

### Examples

```
## Not run:
daily.cor.cross(x.d, "SU")
daily.cor.cross(x.d, seasNames)

## End(Not run)
```

## daily.cor.cross.simple *Simplified daily correlation matrix of wind and solar between countries*

### Description

Solar is no longer country-specific

### Usage

```
daily.cor.cross.simple(x, seas = seasNames, plot2pdf = FALSE)
```

### Arguments

| | |
|---|---|
| x | Time series (xts class) containing hourly data with column names "ATsolar", etc. |
| seas | Season (seas = seasNames gives whole year) |

### Value

correlation matrix

## `dayHourPerSeason` *Get data.frame of hourly wind+solar avl. per season in wide-format (hours are in columns)*

### Description

Get data.frame of hourly wind+solar avl. per season in wide-format (hours are in columns)

### Usage

```
dayHourPerSeason(x, cty, s, load = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | Time series (xts class) containing hourly data with column names "ATsolar", etc. |
| `cty` | Country |
| `s` | Seasons given as vector of month numbers (1-12) |

### Value

- 48-column data.frame of solar and wind for specified months
- if also load: 72-column df of solar, wind, and load)
- column names: "solarDE01", "solarDE02",...,"windDE24" ("loadDE24")
- Note: Length of columns can be different for different choices of seasons

### Examples

```
## Not run:
dayHourPerSeason("AT",3)
dayHourPerSeason("DE", 4:9) # summer halfyear
dayHourPerSeason("DE", c(1,2,3,10,11,12)) # winter halfyear

## End(Not run)
```

## `diff.avl` *Difference in Availability: PCA and Empirical (Analysis only for DE)*

### Description

Print differences for DE and plot

### Usage

```
## S3 method for class 'avl'
diff(avlPCA, avl, plot2pdf = FALSE)
```

### Arguments

| | |
|---|---|
| `avlPCA` | Array returned from `avl.PCA` |
| `avl` | Array returned from `avl.emp` |

## `discrWeibull` *Get values and probabilities of a discretized Weibull distribution*

### Description

Intervals are equally spaced from quantiles: `0.1/i` up to `1-0.1/i`. Values are the the mid-points of the intervals.

### Usage

```
discrWeibull(i, pWeib)
```

### Arguments

| | |
|---|---|
| `i` | Number of values |
| `pWeib` | Parameters from fitting returned by `fitWeibull` |

### Value

: Matrix with two columns: probability, value

## `facAnalysis` *Factor-Analysis*

### Description

Factor-Analysis of first three PC

### Usage

```
facAnalysis(
  fac,
  logFac1 = FALSE,
  logFac2 = FALSE,
  logFac3 = FALSE,
  plot2pdf = FALSE
)
```

### Arguments

| | |
|---|---|
| `fac` | Daily time series of factors as returned by `PCAfac` |

### Value

List *fitParam* with elements:

**"mean"** c(mF1, mF2, mF3)

**"std"** c(sF1, sF2, sF3)

**"weibull"** parWeibull (as returned by `fitWeibull`

## `fit.nC.and.sample` *Fit normal-copula from correlation matrix c.d.small, and sample*

### Description

Fit normal-copula from correlation matrix c.d.small, and sample

### Usage

```
fit.nC.and.sample(c.d.small, sample.size = 20)
```

### Arguments

| | |
|---|---|
| `c.d.small` | Time-series data (in paper: daily data) |
| `sample.size` | Number of random samples |

## `fit.tC.and.sample` *Fit t-copula to data, and sample*

### Description

Works only well with per season data. Attention: Symmetric copula.

### Usage

```
fit.tC.and.sample(x.d.small, sample.size = 20)
```

### Arguments

| | |
|---|---|
| `x.d.small` | Time-series data (in paper: daily data) |
| `sample.size` | Number of random samples |

## `fitWeibull` *Fit a Weibull distribution from data*

### Description

Before the fitting, the input data is shifted such that most of the values becomes positive, which may alos require an optional sign-reversion (if data was mainly nonnegative)

### Usage

```
fitWeibull(data)
```

### Arguments

| | |
|---|---|
| `data` | Vector of data |

## Value

Vector with components:

1. Shape and scale parameters of fitted Weibull distribution
2. Sign reversion and shift (applied before fitting)

---

`get.nearest` *describeIn scn.cross*

---

## Description

describeIn scn.cross

## Usage

```
get.nearest(x, y)
```

---

`hour` *Extract hours and months*

---

## Description

Extract hours and months

## Usage

```
hour(x)

month(x)
```

## Arguments

| | |
|---|---|
| `x` | Time-date, can be a vector of dates (format: POSIXct) |

## Value

Hour of day (= 0-23), or index of month (= 1-12)

## Examples

```
hour(as.POSIXct("2012-10-19"))
```

## `hourOfDay` *Get hour and year of a date*

### Description

Helper function to plot seasonal day-hour value

### Usage

```
hourOfDay(t)
```

### Arguments

| | |
|---|---|
| `t` | Date-time |

### Value

Hour and year (e.g. "02 2017")

## `l.stacked.All` *Data over seasons for all countries*

### Description

Used to prepare data for external use, e.g. for external clustering. Convert time series first to UTC to avoid duplicated hours

### Usage

```
l.stacked.All(x)

daysPerSeason(x, cty, seas)
```

### Arguments

| | |
|---|---|
| `x` | Time series with columns "ATsolar","ATwind",...,"ITwind" (xts-class) |
| `cty, seas` | Country and season |

### Value

List of data.frames over seasons. Given a season, the columns of the data.frame are "ATsolar", ..., "ITwind"; the rows are the hourly data in the season

### Functions

- `daysPerSeason`:

## `lambda.nonpar` *Non-parametric fit of lambda*

### Description

Best fits are for single season, e.g. "SU"

### Usage

```
lambda.nonpar(x)
```

### Arguments

| | |
|---|---|
| x | An xts-time series (for the paper, this is a daily time-series, usually denoted by x.d) |

### Examples

```
## Not run:
lambda.non.param(x.d)

## End(Not run)
```

## `lambda.t` *Lambda from t-distribution*

### Description

Lambda from t-distribution is symmetric: counterfactual

### Usage

```
lambda.t(x)

f.display(y, row.col.names)
```

### Arguments

| | |
|---|---|
| x | An xts-time series (for the paper, this is a daily time-series, usually denoted by x.d) |
| y | numeric argument |

### Value

rounded argument with row and column names set

### Functions

- `f.display`: Round numeric argument and set row and column names

## Examples

```
## Not run:
lambda.t(x.d)

## End(Not run)
```

---

`load.24` *Vector of load-period tags in a season*

---

## Description

Vector of load-period tags in a season

## Usage

```
load.24(seas, n = 24)
```

## Arguments

| | |
|---|---|
| `seas` | Season |
| `n` | Number of load periods in a day (default: 24) |

## Value

```
c("WI-D-01",...,"FA-D-24")
```

---

`PCAfac` *PCA for a single season and country*

---

## Description

There can be several seasons (together) and countries (additional dimension)

## Usage

```
PCAfac(
  xSWH,
  cty = "DE",
  season = "SP|SU",
  m = 4,
  deMean = FALSE,
  plot2pdf = FALSE,
  load = FALSE
)
```

## Arguments

| | |
|---|---|
| `xSWH` | List over seasons of data in wide format (day hours are in columns).Given a season, a list element is a data.frame with columns "ATsolar01", "ATsolar02' etc., and rows = observations in the season; as returned by `x.24wide` |
| `cty` | Country |
| `season` | Season |
| `m` | Number of loadings to plot |
| `deMean` | Should data first be de-meaned? (default: FALSE) |
| `plot2pdf` | Should plots be copied to pdf? (default: FALSE) |
| `load` | If load is included make a different pdf file name (default: FALSE) |

## Details

Log-normal factors gave usually bad results. Avoid!

## Value

list: 1. Factors implied by PCA over the input data points (rows), that is, a daily time series of factors (class: matrix); 2. pcaloads

## Examples

```
## Not run:
PCAsingle("DE|IT","SU")
PCAsingle("AT","SU")
PCAsingle("DE","WI|SP|SU|FA")

## End(Not run)
```

---

# `plot24` *Plot the seasonal (averaged) day-hour data*

---

## Description

The number of seasons is currently hardcoded: 4 seasons

## Usage

```
plot24(x, cols, yr = c("2017", "2018", "2019"), xpd = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Time series (xts class) containing hourly data with column names "ATsolar", etc.; as returned `sel.wind` |
| `cols` | column to plot |
| `yr` | list of years (default: `c("2017","2018","2019")`) |
| `xpd` | expand the dispay (required for plots of wind) |

### Examples

```
## Not run:
windows(); plot24(x,"ATsolar"); dev.print(pdf, "ATsolar.pdf")
windows(); plot24(x,"ATwind", TRUE)

## End(Not run)
```

## plotCor.Emp *Plot correlations of empirical data*

### Description

Plot correlations of empirical data

### Usage

```
plotCor.Emp(xSWH, cty = "DE", s = "SU", oldV = FALSE, plot2pdf = FALSE)
```

### Arguments

| | |
|---|---|
| xSWH | List of scenarios over countries of type *scnBEM* (see help scn.clust), as returned by scn.All |
| cty | Country |
| s | Season |

## plotCor.genClust *Clustering: Generate scenarios and plot correlations*

### Description

Clustering: Generate scenarios and plot correlations

### Usage

```
plotCor.genClust(
  xSWH,
  cty = "DE",
  s = "SU",
  n = 20,
  oldV = FALSE,
  plot2pdf = FALSE
)
```

### Arguments

| | |
|---|---|
| xSWH | List over seasons of data in wide format (day hours are in columns).Given a season, a list element is a data.frame with columns "ATsolar01", "ATsolar02' etc., and rows = observations in the season. |
| cty | Country |
| s | Season |

### Value

covariance-matrix

## plotCor.genPCA *PCA factor model: Generate scenarios and plot correlations*

### Description

PCA factor model: Generate scenarios and plot correlations

### Usage

```
plotCor.genPCA(
  xSWH,
  cty = "DE",
  s = "SU",
  J = c(5, 2, 1),
  oldV = FALSE,
  plot2pdf = FALSE,
  textArg
)
```

### Arguments

| | |
|---|---|
| `xSWH` | List over seasons of data in wide format (day hours are in columns).Given a season, a list element is a data.frame with columns "ATsolar01", "ATsolar02' etc., and rows = observations in the season. |
| `cty` | Country |
| `s` | Season |
| `J` | J+1 = Number of realizations of factors |
| `textArg` | Optional argument to insert in title of plot |

### Value

covariance-matrix

## plotECDF *Check with empirical distribution function*

### Description

Plot empirical distribution function of wind and solar. Select a season or entire year.

### Usage

```
plotECDF(
  scn.ws.or.comp,
  cty = "DE",
  s = seasNames,
  yrs = "2017-2019",
  comparePCA = FALSE,
  plot2pdf = FALSE,
  J.cases
)
```

### Arguments

| | |
|---|---|
| `scn.ws.or.comp` | List of scenarios over countries of type *scnBEM* (see help `scn.clust`). If sensitivity analysis (comparison): List of single-country scenarios over the different elements of `J.cases`. As returned by `scn.All` |
| `cty` | Country |
| `s` | Season (default is fully year: s=seasNames) |
| `yrs` | Used for plot title, range of years |
| `comparePCA` | Is it a sensitivity analysis (default: FALSE) |
| `J.cases` | Parameters of sensitivity analysis, used in title of plot |

### Examples

```
## Not run:
plotEmpirical(scn.ws) # full year
plotEmpirical(scn.ws, s="SU") # SU

## End(Not run)
```

---

## `plotScn` *Plot scenarios of PCA factor model*

---

### Description

Plot scenarios for selected country and season.

### Usage

```
plotScn(scnAll, cty = "DE", s = "SU")
```

### Arguments

| | |
|---|---|
| `scnAll` | List of scenarios over countries of type *scnBEM* (see help `scn.clust`), as returned by `scn.All` |
| `cty` | Country |
| `s` | Season |

## plotScnData *Plot scenarios generated from a PCA factor model*

### Description

The probability of a scenarios is proportional to the line width.

### Usage

```
plotScnData(scnData, cty, season, plot2pdf = FALSE)
```

### Arguments

| | |
|---|---|
| `scnData` | Scenarios as returnd by `scn.PCAfac` |

### Details

For the used rainbow palette you can also select start/end color (red = 0, yellow = 1/6, green = 2/6, cyan = 3/6, blue= 4/6 and magenta = 5/6) and saturation (s) and value (v): rainbow(n, s = 1, v = 1, start = 0, end = max(1, n - 1)/n, alpha = 1)

## reduce.cor *Reduce correlation matrix (remove cross-regional solar correlations)*

### Description

Reduce correlation matrix (remove cross-regional solar correlations)

### Usage

```
reduce.cor(x, seas)
```

### Arguments

| | |
|---|---|
| `x` | Time series (xts class) containing hourly data with column names "ATsolar", etc. |
| `seas` | Season (seas = seasNames gives whole year) |

scn.All *Create scenarios for all countries and all seasons*

## Description

Combination of previous functions: scn.PCAfac etc. Sensitivity analysis of a single country w.r.t different J is also possible

## Usage

```
scn.All(
  xSWH,
  J.cases = list(c(5, 2, 1)),
  thresMin = -0.1,
  thresMinDeMean = -0.01,
  weibFac1 = TRUE,
  comparePCA = FALSE,
  compareCty = "DE",
  logFac = c(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE)
)
```

## Arguments

| | |
|---|---|
| J.cases | List of vectors of factor realizations, e.g. list(c(5,2,1)); usually the list has just a element. Sensitvity analysis is also possible with more list elements |
| thresMin | Bound to identify a pathological, negative avl. scenario |
| thresMinDeMean | Different bound if data was de-meaned |
| weibFac1 | The first factor is fitted to weibull? (default: TRUE) |
| comparePCA | Sensitivity analysis? (default: FALSE) |
| compareCty | Country of sensitivity analysis (default: "DE") |
| logFac | Switches for log-normal factors (default: FALSE) |

## Value

List of scenarios over countries of type *scnBEM* (see help scn.clust). If sensitivity analysis: List of single-country scenarios over the different elements of J.cases

## Examples

```
## Not run:
scn.All(J.cases = list(c(5,2,1)))
scn.All(J.cases = list(c(5,0), c(5,2,0), c(5,2,1)))

## End(Not run)
```

## `scn.clust` *Cluster by hierarchical method using medoids*

### Description

Single countries or multiple countries can be selected, whereas only a single season can be specified. The agglomerative, hierarchical clustering uses a `method`, which is a dissimilarity measure between clusters (so-called Linkage):

`"ward.D2"` **(default)** Ward’s method. Minimizes variance of (potentially) merged clusters (ward "likes" to produce clusters of equal sizes; "ward.D" was coded wrongly)

`"average"` (= UPGMA): average distance between all points of two clusters

`"complete"` Highest distance between point-pairs of two clusters

`"single"` Smallest distance between point-pairs of two clusters

### Usage

```
scn.clust(
  xSWH,
  nclus = 20,
  cty = "DE",
  s = "SU",
  interactive = FALSE,
  crossCountry = FALSE,
  clustmethod = "ward.D2",
  distmeasure = "euclidean"
)
```

### Arguments

`xSWH` List over seasons of data in wide format (day hours are in columns). Given a season, a list element is a data.frame with columns "ATsolar01", "ATsolar02’ etc., and rows = observations in the season; as returned by `x.24wide`

`nclus` Number of clusters

`cty` Country (default: `"DE"`; several countries `"AT|CH|DE|FR|IT"`)

`s` Season (default: `"SU"`)

`interactive` Plot detailed diagonstic of clustering (dendogramm etc.)? (default: FALSE)

`crossCountry` Cluster across countries (`cty` must have several regions)? (default: FALSE)

`clustmethod` Cluster method, that is, the linkage (default: Ward’s method)

`distmeasure` Measure of distance between data points (default: `"euclidean"`). Other possible distances are e.g. "maximum" and "manhattan"

### Value

For a single region, a data.frame (called *scnBEM*) of scenarios in stacked format is returned (i.e. the 24 day hours of scenario values are in rows):

| **scn** | **country** | **loadp** | **prob** | **windAvl** | **solarAvl** |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| "o1" | "AT" | "WI-D-01" | 0.01 | 0.3 | 0.0 |
| "o1" | "AT" | "WI-D-02" | 0.01 | 0.3 | 0.0 |
| "o1" | "AT" | ... | 0.01 | ... | ... |
| "o1" | "AT" | "WI-D-24" | 0.01 | 0.3 | 0.1 |
| "o2" | "AT" | "WI-D-01" | 0.02 | 0.3 | 0.0 |

For multiple regions, a list over the countries is returned, where each list element is data.frame of format *scnBEM*

## `scn.clust.cross` *Clustering of cross-country scenarios over all seasons*

### Description

Stack the cross-country scenarios over all the seasons. Hence, increase the scenario index, because different seasons have independent scenarios in the electrictiy market model BEM

### Usage

```
scn.clust.cross(xSWH, nclust = 20)
```

### Arguments

`xSWH` List over seasons of data in wide format (day hours are in columns).Given a season, a list element is a data.frame with columns "ATsolar01", "ATsolar02’ etc., and rows = observations in the season; as returned by `x.24wide`

`nclust` Number of clusters

### Value

scenarios over all countries over all seasons

## `scn.cross` *Generate cross-country scenarios for seasonal data*

### Description

Generate cross-country scenarios for seasonal data

### Usage

```
scn.cross(scn.ws, s = "SU", rC, sample.size)
```

### Arguments

`scn.ws` List of scenarios over countries of type *scnBEM* (see help `scn.clust`)

`s` Season

`rC` random samples as returned by `fit.nC.and.sample` or `fit.tC.and.sample`

`sample.size` Sample size corresponding to rC (can in fact be determined by the dimensions of rC)

## scn.cross.plot *Plot cross-regional scenarios*

### Description

Plot cross-regional scenarios

### Usage

```
scn.cross.plot(scn.ws.new, s = "SU", sample.size = 20, plot2pdf = FALSE)
```

### Arguments

| | |
|---|---|
| `scn.ws.new` | Cross-regional scenarios as returned by `scn.cross` |

## scn.cross.write *Concatenate cross-regional scenarios over seasons and write to file*

### Description

Concatenate cross-regional scenarios over seasons and write to file

### Usage

```
scn.cross.write(scn.ws.new.year, sample.size = 20, tag = "")
```

### Arguments

| | |
|---|---|
| `scn.ws.new.year` | List of cross-regional scenarios over the seasons |
| `sample.size` | Sample size: required to make correct scenario numbering: "o1", "o2", ... |
| `tag` | String appended to file name |

### Examples

```
## Not run:
scn.cross.write(scn.ws.new.year, 20, "_tcopula")

## End(Not run)
```

## `scn.cross.year` *Make cross-regional scenarios for all seasons*

### Description

Make cross-regional scenarios for all seasons

### Usage

```
scn.cross.year(x, scn.ws, sample.size, tCop = TRUE)
```

### Arguments

`scn.ws` List of scenarios over countries of type *scnBEM* (see help `scn.clust`)

`sample.size` Sample size corresponding to rC (CAN PERHAPS BE DETERMINED FROM rC)

`tCop` Take t-copula instead of guassian copula? (default: TRUE)

`rC` random samples from copula, as rnC or rtC

### Value

List of cross-regional scenarios over the seasons

## `scn.PCAfac` *Generate scenarios given a PCA factor model*

### Description

Generate scenarios given a PCA factor model

### Usage

```
scn.PCAfac(
  fitParam,
  pcaloads,
  m.sw,
  J1 = 5,
  J2 = 1,
  J3 = 1,
  threshold = -0.01,
  dropNeg = TRUE,
  logFac1 = FALSE,
  logFac2 = FALSE,
  logFac3 = FALSE
)
```

### Arguments

`pcaloads` Loadings as returned by `PCAfac`

`m.sw` Mean values as returned by `PCAfac`

`J1` J1+1 = number of scenarios for factor 1

`J2` J2+1 = number of scenarios for factor 2

`J3` J3+1 = number of scenarios for factor 3

`threshold` Bound to identify a pathological, negative avl. scenario

`dropNeg` Should such negative scenario be dropped? (default: TRUE)

### Value

data-frame "scenData"

- row 1: probability of scenario
- row 2,...,48+1: scneario realization of wind and solar
- columns: "scn1", "scn2", ... "scn(smax)"

## `sel.wind` *Select for a country the wind: Onshore, Offshore, or On+Off?*

### Description

Select for a country the wind: Onshore, Offshore, or On+Off?

### Usage

```
sel.wind(x, windType, cty = "DE", load = FALSE)
```

### Arguments

`x` Time series (xts class) containing hourly data with column names ..., "DEONwind", "DEOFFwind", "DEONOFFwind"; as returned by `x.normalize` or `x.read`

`windType` 1: onshore, 2: offhsore, 3: total

`cty` Country (default: "DE")

### Value

Time series with selected column (named "DEwind")

## `set.cty.seas` *Set global parameters*

### Description

Initialize `ctyNames`, `seasNames`, `seasons`

### Usage

```
set.cty.seas(
  cty = c("AT", "CH", "DE", "FR", "IT"),
  seas = c("WI", "SP", "SU", "FA"),
  month.idx = c(c(12, 1, 2), 3:5, 6:8, 9:11)
)
```

### Arguments

| | |
|---|---|
| `cty` | Vector of country names |
| `seas` | Vector of season names |
| `month.idx` | Index of months of the seasons 1,...,12 |

### Value

Global parameters:

`ctyNames` `c("AT","CH","DE","FR","IT")`

`seasNames` `c("WI","SP","SU","FA")`

`seasons` Matrix: columns = "WI, "SP", "SU", "FA"; rows = month-index (1,...,12)

## `writeScnAll` *Write scenarios over seasons and countries into files*

### Description

Write scenarios over seasons and countries into files

### Usage

```
writeScnAll(scn.ws)
```

### Arguments

| | |
|---|---|
| `scn.ws` | List of scenarios over countries of type *scnBEM* (see help `scn.clust`), as returned by `scn.All` |

## `x.24wide` *Prepare data for PCA*

### Description

Assumption: The input time series is in CET, and we convert first to UTC. The conversion is required to get rid of the missing DST hours in the night. We do this before subsetting to a single year (else the last hour would be missed)

### Usage

```
x.24wide(x, yr = "2017/2019")
```

### Arguments

| | |
|---|---|
| x | Time series (xts class) containing hourly data with column names "ATsolar", etc. |

### Value

List over seasons. A list element contains data.frame of all countries in wide forma (the day hours are in different columns)

- data.frame has 240 columns (2*24 hours (solar+wind) * 5 countries) with names: "solarCH01", "solarCH02",. . . ,"windIT24"

## `x.cor` *Correlation between solar, wind, and (optionally) load*

### Description

Prints correlation for different years. Currently, it must be three years. If load is read-in: Print correlation-test of load/wind of first year

### Usage

```
x.cor(x, load = FALSE, yr = c("2017", "2018", "2019"))
```

### Arguments

| | |
|---|---|
| x | Time series (xts class) containing hourly data with column names "ATsolar", etc.; as returned by `sel.wind` |
| yr | List of three years |

## `x.daily` *Daily mean per season*

### Description

Daily mean per season

### Usage

```
x.daily(x, seas)
```

### Arguments

| | |
|---|---|
| `x` | xts-object |
| `seas` | Season. Season can be whole year (s = seasNames) |

## `x.normalize` *Convert x into availability factors (divide by capacity):*

### Description

Generation is divided by end-of-year capacity. Load is dvidided by a high quantile of each year. Currently only DE has also Offshore wind

### Usage

```
x.normalize(
  x,
  yearRead = c(2017, 2018, 2019),
  fn = "SolarWindcapacity.csv",
  load = FALSE,
  loadQuantile = 0.96
)
```

### Arguments

| | |
|---|---|
| `x` | Time series (xts class) containing hourly data with column names "ATsolar", etc.; as returned by `x.read` |
| `fn` | File name of solar and wind capacities in csv format, with rows as years (e.g. 2017), and columns as country and tech (e.g. "ATsolar") |

## x.plot *Plot wind, solar (and load, if read-in) per country and per year*

### Description

Plot wind, solar (and load, if read-in) per country and per year

### Usage

```
x.plot(x, cty, yr)
```

### Arguments

| | |
|---|---|
| x | Time series (xts class) containing hourly data with column names "ATsolar", etc.; as returned by x.normalize or x.read |
| cty | Country |
| yr | Year (or a range of years in xts-notation) |

### Examples

```
## Not run:
plotSolarWind(x,"CH","2017\2019")
plotSolarWind(x,"CH","2018")

## End(Not run)
```

## x.read *Load hourly wind, solar data (optionally also load)*

### Description

- The function works currently with only country DE having also offshore wind
- The wind and solar files must be named like "DE_Generation_2019 (ENTSO)HOURLY.csv"
- The load files must be named like "DE_Load_2019 (ENTSO)HOURLY.csv"

### Usage

```
x.read(
  fn.path = "Data/",
  yearRead = c(2017, 2018, 2019),
  load = FALSE,
  colSol = "Solar.",
  colWiOn = "Wind.Onshore.",
  colWiDEOff = "DEWind.Offshore.",
  colLoad = "load"
)
```

### Arguments

| | |
|---|---|
| `fn.path` | Path to ENTSO country files |

### Value

x Time series (xts class) containing hourly data with column names

- "ATsolar", "ATwind", . . . , "ITwind", "DEOFFwind", "DEONOFFwind"
- If also *load* is read: "ATsolar", "ATwind", "ATload",. . .

---

## `x24Seas` *Select columns from time series and simplify time index*

---

### Description

Select by year, by col-names, and reduce the time-index to the day-hour only

### Usage

```
x24Seas(z, cols, year)
```

### Arguments

| | |
|---|---|
| `z` | Time-series with time index like "23 2012" |
| `cols` | Vector of column names to select |
| `year` | Year (e.g. "2017") |

### Details

Helper function to plot seasonal day-hour value

### Value

Data in specified year and columns, where the time-index is the day hour

---

## `x24SeasonYears` *Calculate mean-hourly values per season over selected columns and years*

---

### Description

Helper function to plot seasonal day-hour value

### Usage

```
x24SeasonYears(x, cols, years = c("2017", "2018", "2019"))
```

### Arguments

`x` Time series (xts class) containing hourly data with column names "ATsolar", etc.; as returned `sel.wind`

`cols` Column-names of the time series to select (e.g. "ATsolar")

`yr` Years

### Value

List mean values over years and seasons. List elements have names like "SP2017", contain a time series, with columns "ATsolar", etc., and with rows the mean values over the day hours

# Index