Package 'repra'

October 6, 2014

Title Renewable Energy Probability Resource Assessment Tool (REPRA)

Version 0.4.2

Date 2014-09-30

Description This package includes methods to calculate resource adequacy metrics in power systems. These methods are based in the notion of loss-of-load probability (LOLP) and include the treatment of conventional and variable renewable generators.

License MIT + file LICENSE

URL https://github.com/NREL/repra

BugReports https://github.com/NREL/repra/issues

Depends R(>= 3.1.0)

Imports data.table, dplyr (>= 0.3), Rcpp, reshape2, ggplot2,assertthat

Suggests testthat, knitr

LinkingTo Rcpp (>= 0.11.1)

VignetteBuilder knitr

LazyData TRUE

Author Eduardo Ibanez [aut, cre],National Renewable Energy Laboratory [cph]

Maintainer Eduardo Ibanez <eduardo.ibanez@nrel.gov>

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-10-06 23:05:13

R topics documented:

`repra-package` *The Renewable Energy Probabilistic Resource Assessment tool*

Description

This package is the R implementation of the Renewable Energy Probabilistic Resource Assessment (REPRA) tool, developed at the National Renewable Energy Laboratory (NREL).

Details

More information is available in the help of the package functions. Additionally, two vignettes are included with this package and can be accessed with `browseVignettes("repra")` or `browseVignettes("sliding_window`

References

Eduardo Ibanez, Michael Milligan (2012). Impact of Transmission on Resource Adequacy in Systems with Wind and Solar Power. IEEE Power& Energy Society General Meeting, 22-26 July 2012, San Diego, CA. http://www.nrel.gov/docs/fy12osti/53482.pdf.

`calculate_elcc` *Calculate effective load carrying capability (ELCC)*

Description

Given a time data object (with net load data), find the effective load carrying capability (ELCC) to achieve the desired reliability metric. The function automatically applies these calculation for each scenario, level or aggregation and area.

Usage

```
calculate_elcc(time.data, outage.table, obj.metric = "LOLE",
  obj.value = 0.1, scale.load = FALSE, max.iter = 20, ignore = NULL,
  ...)
```

Arguments

time.data	Time series data formatted with format_timedata
outage.table	Outage table used in the lookup, created with outage_table
obj.metric	Metric to be used as objective: LOLE, LOLH, PeakLOLP, EUE (see details)
obj.value	Objective value to achieve for the selected metric
scale.load	During the iterations is the load scaled (TRUE) or the same value added to all times (FALSE)
max.iter	Maximum iterations to perform before stopping
ignore	Column names in time.data to ignore in the calculation of total VG data
...	Additional parameters passed to sliding_window

Details

The following metrics can be selected as an objective (through obj.metric):

- Daily loss of load expectation (LOLE): Calculated as the sum of the maximum daily LOLP
- Loss of load hours (LOLH): Sum of all the LOLP
- Maximum daily LOLP (PeakLOLP): Maximum LOLP value observed
- Expected unserved energy (EUE): Sum of EUE for all time steps

ELCC is calculated by modifying the load profile so that the final profile yields a certain reliability level. This can be done in two ways:

- by scaling the entire profile (e.g., by multiplying the entire time series by 1.05)
- by adding a constant number to the entire profiles (e.g., adding 100 MW)

scale.load needs to be set to TRUE or FALSE to achieve either one, respectively.

See Also

format_timedata and outage_table to create time.data and outage.table objects, respectively

sliding_window is used internally to extend time.data

calculate_metrics is used internally to evaluate the metrics

Examples

```
# Create outage table with 200 5-MW units
gens <- data.frame(Capacity = rep(5, 200),
                   EFOR = rep(0.08, 200))
out.table <- outage_table(gens)

# Create random load and wind data and format
tdata <- data.frame(Time = 1:8760,
                    Load = runif(8760, 450, 850),
                    Wind = runif(8760, 0, 100),
                    Wind2 = runif(8760, 0, 200))
```

```
td <- format_timedata(tdata)

# Calculate ELCC with both Wind and Wind2
calculate_elcc(td, out.table)

# Calculate ELCC with only Wind
calculate_elcc(td, out.table, ignore = "Wind2")
```

`calculate_metrics` *Calculate and summarize loss of load probability metrics*

Description

Given a time data object, summarize probabilities into metrics. The function automatically applies these calculation for each scenario, level of aggregation and area.

Usage

```
calculate_metrics(time.data, outage.table, raw = FALSE, ...)
```

Arguments

`time.data`	Time series data formatted with `format_timedata` (must have a `NetLoad` column, see details)
`outage.table`	Outage table used in the lookup, created with `outage_table`
`raw`	Return summary metrics (`FALSE`, default) or raw hourly output (`TRUE`)
`...`	Additional parameters passed to `sliding_window`

Details

The time data object must have a column called `NetLoad` (see the examples for an easy method to generate it).

Summary metrics include daily loss-of-load expectation (`LOLE`), loss-of-load hours (`LOLH`), maximum LOLP (`PeakLOLP`) and expected unserved energy (`EUE`).

See Also

`format_timedata` and `outage_table` to create `time.data` and `outage.table` objects, respectively

`sliding_window` is used internally to extend `time.data`

`calculate_elcc` uses this function to calculate ELCC

Examples

```
# Create outage table with 200 5-MW units
gens <- data.frame(Capacity = rep(5, 200),
                   EFOR = rep(0.08, 200))
out.table <- outage_table(gens)

# Create random load and wind data and format
tdata <- data.frame(Time = 1:8760,
                    Load = runif(8760, 450, 850),
                    Wind = runif(8760, 0, 100))
td <- format_timedata(tdata)

# Get metrics for net load (load - wind)
td2 <- td
td2$NetLoad <- td2$Load - td2$Wind
calculate_metrics(td2, out.table)

# Get metrics for just load
td3 <- td
td3$NetLoad <- td3$Load - td3$Wind
calculate_metrics(td3, out.table)

# Get raw data (i.e., not summarized)
calculate_metrics(td2, out.table, raw = TRUE)
```

---

capacity_value *Calculate capacity value for variable generation*

Description

Given time data series and an outage table, calculate the capacity value of variable generation (VG) time series. Capacity value is calculated by iteratively removing one type of VG at a time and recalculating effective load carrying capability (ELCC).

Usage

```
capacity_value(time.data, outage.table, VG.cols = NULL, marginal = FALSE,
  scale.first = FALSE, ...)
```

Arguments

time.data	Time series data (or subset) formatted with format_timedata
outage.table	Outage table used in the lookup, created with outage_table
VG.cols	Order in which VG capacity value is calculated (see details for more info)
marginal	Should the VG capacity value be calculated as the last resource (TRUE) or added progressively to the load (FALSE)?
scale.first	This triggers a special case to calculate capacity value. See details for more information.
...	Additional parameters passed to calculate_elcc and sliding_window

Details

This function calculates ELCC for the system with different combinations of VG. The metric and desired reliability metric can be set; see calculate_elcc for details and a list of parameters. Additionally, a sliding window can be used in the calculation (see calculate_elcc).

When calculating the capacity value of VG, the results depends on the order in which the different technologies are added into the system. The variable VG.cols is used to determine this order manually. If not determined the order will default to the order in which the VG columns appear in time.data.

This marginal parameter determines if the capacity value is calculated by taking each VG profile out of the total mix (when is set to TRUE) or by progressively stacking the VG profiles on top the conventional generators (when is set to FALSE). The latter option is the default and the order is determined by the VG.cols parameter or, if not set, by the order in which the columns appear in the time.data object.

If scale.first is set to TRUE the capacity value a special calculation is triggered. Before perfoming the calculations, the load is scaled so that the total mix (conventional generators and VG) provide the desired adequacy level, which is specified with the parameters passed to calculate_elcc. Once the load is scaled, the capacity value calculations are performed using the flat block method.

See Also

format_timedata and outage_table to create time.data and outage.table objects, respectively

sliding_window is used internally to extend time.data

calculate_elcc is used internally to evaluate ELCC

Examples

```
## Not run:
# Create outage table with 200 5-MW units
gens <- data.frame(Capacity = rep(5, 200),
                   EFOR = rep(0.08, 200))
out.table <- outage_table(gens)

# Create random load and wind data and format
tdata <- data.frame(Time = 1:8760,
                    Load = runif(8760, 450, 850),
                    Wind = runif(8760, 0, 100),
                    Wind2 = runif(8760, 0, 200))
td <- format_timedata(tdata)

# Calculate capacity value (both are equivalent)
capacity_value(td, out.table)
capacity_value(td, out.table, c("Wind", "Wind2"))

# Calculate capacity value of Wind2 first and Wind second
capacity_value(td, out.table, c("Wind", "Wind2"))

# Calculate capacity value with a sliding window that uses adjacent hours
capacity_value(td, out.table, win.h.size = c(-1, 1))
```

```
## End(Not run)
```

`format_timedata` *Format time data*

Description

Create valid data container that has aggregated load and variable generation (VG) time series for different areas and levels of aggregation.

Usage

```
format_timedata(data, levels = NULL, scenario = NULL, day.steps = 24)
```

Arguments

`data`	Data frame with the load and VG time series (see details for requirements)
`levels`	Optional data frame that contains levels of aggregation, e.g., BAA, transmission area or interconnection (see details for requirements)
`scenario`	Name of columns in `data` used to denote different scenarios
`day.steps`	Number of data points in a day (defaults to 24)

Details

The columns identified by `scenario` are used to perform calculations separately. This way one can run different sensitivities in the same calculation (e.g., to estimate the capacity value of wind with different penetration levels).

Requirements for `data`:

- The number of entries needs to be a multiple of `day.steps`
- A `Time` column that can be ordered (integer or time stamps)
- A `Load` column with load time series
- No column named `Level`, `NetLoad` or `VG` (they are reserved name)
- Optionally, if a column called `Area` exists, it will be used to separate areas
- All columns must contain numbers except for `Time`, `Area` and those in `scenario`

Requirements for `levels`:

- This parameter is optional
- If `levels` is provided, `data` must contain a column called `Area`
- The first column must contain the most granular level of data
- The column names in `levels` will become the names of the different levels of aggregation

Value

A data frame with load and VG data aggregated by scenario and different areas and levels of aggregation

See Also

outage_table is the function that creates outage tables

sliding_window is used internally by several functions to extend time data objects

Examples

```
# Create data for two days
tdata <- data.frame(Area = c(rep("A", 48), rep("B", 48)),
                    Time = 1:48,
                    Load = c(runif(48, 200, 250), runif(48, 400, 450)),
                    Wind = c(runif(48, 20, 25), runif(48, 40, 45)))
levs <- data.frame(BAA = c("A", "B"), Region = c("All", "All"))

# Format time data without and with different levels of aggregation
td1 <- format_timedata(tdata)
head(td1)
td2 <- format_timedata(tdata, levs)
head(td2)

# Format time data with a scenario column
tdata2 <- tdata
tdata2$Scenario <- "Scenario 1"
td3 <- format_timedata(tdata2, scenario = "Scenario")
head(td3)

# Format time data without Area column (minimum example)
tdata3 <- tdata
tdata2$Area <- NULL
td4 <- format_timedata(tdata3)
head(td4)
```

---

outage_table *Convolve generators to create outage tables*

Description

Allows to create outage tables for different areas and levels of aggregation for a list of capacities and equivalent forced outage rates (EFOR) for generators. The assumption is that the generator outages are independent from each other. The table can be used to quickly compute loss-of-load probability (LOLP) and expected unserved energy (EUE).

Usage

```
outage_table(generators, levels = NULL, threshold = 1e-06, round = 1)
```

Arguments

generators	Data frame with the list of generators (see details for requirements)
levels	Optional data frame that contains levels of aggregation, e.g., BAA, transmission area or interconnection (see details for requirements)
threshold	Table entries with LOLP below this value are ignored
round	Used to round capacities (default is 1 MW)

Details

Requirements for `generators`:

- The data frame must contain two columns: `Capacity` and `EFOR` (Other columns will be ignored)
- Optionally if a column called `Area` exists, it will be used to create separate tables within each area

Requirements for `levels`:

- This parameter is optional
- If `levels` is provided, `generators` must contain a column called `Area`
- The first column must contain the most granular level of data and compatible with columns `Area` in `generators`
- The column names in `levels` will become the names of the different levels of aggregation

Value

The result is a data.table object that is used to calculate LOLP metrics. For each area, the output presents the following columns: `Capacity`, `Prob`, `LOLP` and `BaseEUE`. The column `Capacity` is duplicated as (`Capacity2`) and is used internally by the LOLP calculation functions.

Given a load level `L`, the probability parameters are calculated as follows:

- Choose the first row with `L <= Capacity`
- The loss-of-load probability is the value in the `LOLP` for that row
- Expected unserved energy can be quickly calculated as `EUE = BaseEUE + (L - Capacity) * LOLP`. This is equivalent to calculating the sum of `(L - Capacity) * Prob` for all rows with `L < Capacity`

References

R. Billinton, and R. N. Allan, Reliability evaluation of power systems, New York: Plenum Press 1996.

See Also

`format_timedata` is the function that creates compatible time data objects

Examples

```
# Outage table with one 10-MW generator with 2% EFOR
outage_table(data.frame(Capacity = 10, EFOR = 0.02))

# Outage table with two generators (10 MW with 2% EFOR and 20 MW with 1% EFOR)
outage_table(data.frame(Capacity = c(10, 20), EFOR = c(0.02, 0.01)))

# List of generators and areas
gens <- data.frame(Area = c(rep("A", 10), rep("B", 5)),
                   Capacity = rep(60, 15),
                   EFOR = rep(0.08, 15))
levs <- data.frame(BAA = c("A", "B"), Region = c("All", "All"))

# Create a single outage table (without specifying area)
outage_table(gens[, c("Capacity", "EFOR")])

# Create an outage table for each 'Area'
outage_table(gens)

# Create table for each 'Area' and different and levels of aggregation in 'levs'
outage_table(gens, levs)
```

`repra.data` *Generator and time series data sets*

Description

`repra` includes sample data to showcase typical calculations. This data is contained in four data frames called `repratime`, `repragen`, `repraareas`, `repra capacity`. The data was adapted from the Western Electricity Coordinating Council's (WECC) Transmission Expansion Planning Policy Committee (TEPPC) 2024 Common Case (version 1.0). The data should be considered only for illustrative purposes and no conclusions should be made based on it.

Details

The data is presented for 8 regions in the Western Interconnection: Alberta, AZ-NM-NV (Arizona-New Mexico-Nevada), Basin, British Columbia, California North and South, NWPP (Northwest Power Pool), and RMPA (Rocky Mountain Power Authority). It is presented through four data frames, which are described in the following paragraphs.

`repratime` includes the load, wind, PV and CSP time series for the eight regions.

`repragen` includes the name, nameplate capacity, expected forced outage rate (EFOR) and area for all the generators (that are not wind, PV or CSP).

`repraareas` includes a little example of how the eight areas can be aggregated up to the interconnection level.

Finally, `repracapacities` summarize the installed capacities for wind, PV and CSP (which are useful to normalize capacity value, for example).

References

Western Electricity Coordinating Council (2014). Transmission Expansion Planning Policy Committee (TEPPC) 2024 Common Case (version 1.0). http://www.wecc.biz/committees/BOD/TEPPC/Pages/TAS_Datasets.aspx.

`sliding_window` *Expand time data series with a sliding window*

Description

This function allows to expand the time data series to include adjacent time steps and days, using a sliding window. The procedure is used to shift the variable generation time series, so calculations are not limited to the observed load and variable generation pairs. The objective is to further explore how the variable behaviour of these generators affect reliability calculations.

Usage

```
sliding_window(time.data, win.h.size = NULL, win.h.weight = NULL,
  win.d.size = NULL, win.d.weight = NULL)
```

Arguments

`time.data`	Time series data formatted with `format_timedata`
`win.h.size`	Vector to determine the size of the window for adjacent time steps
`win.h.weight`	Weights associated with adjacent hours
`win.d.size`	Vector to determine the size of the window for adjacent days
`win.d.weight`	Weights associated with adjacent days

Details

The sliding window can include adjacent time steps (typically adjacent hours) and/or adjacent days. Table 1 shows and example, with the current hour being hour 6 in day 5. If an hour is withing the extent of the sliding window it is marked with an x.

Table 1: Example of sliding window

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
Day 1										
Day 2										
Day 3			x	x	x	x	x	x	x	
Day 4			x	x	x	x	x	x	x	
Day 5			x	x	x	Current	x	x	x	
Day 6			x	x	x	x	x	x	x	
Day 7			x	x	x	x	x	x	x	
Day 8										
Day 9										

`win.h.size` determines how many adjacent time steps are included in the sliding window. By default, the window only includes the current hour. If specified, `win.h.size` must be a numeric vector with 2 values.

Similarly, `win.d.size` determines how many adjacent days are used in the window and defaults to the current day. If defined, it must be a vector with 2 integers.

For example, to reproduce the window in Table 1 set:

- `win.h.size = c(-3, 3)`
- `win.d.size = c(-2, 2)`

By default, all the data points in the sliding window are given the same probability in the calculations. It is possible to provide different weights to adjacent hours and adjacent days. This is achieved by using the `wind.h.weight` and `wind.d.weight` parameters, respectively. The lengths of these needs to be consistent with the corresponding sizes parameters. Table 2 shows an example of how the final weights are calculated, by setting:

- `wind.h.weight = c(0.1, 0.1, 0.2, 0.2, 0.2, 0.1, 0.1)`
- `wind.d.weight = c(0.1, 0.2, 0.4, 0.2 0.1)`

Figure 2: Example of sliding window with different weights

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	Day weight
Day 1											
Day 2											
Day 3			0.01	0.01	0.02	0.02	0.02	0.01	0.01		0.1
Day 4			0.02	0.02	0.04	0.04	0.04	0.02	0.02		0.2
Day 5			0.04	0.04	0.08	**0.08**	0.08	0.04	0.04		**0.4**
Day 6			0.02	0.02	0.04	0.04	0.04	0.02	0.02		0.2
Day 7			0.01	0.01	0.02	0.02	0.02	0.01	0.01		0.1
Day 8											
Day 9											
Hour weight			0.1	0.1	0.2	**0.2**	0.2	0.1	0.1		

Use of a sliding window is currently an open area of research. The authors suggest to use it as a sensitivitiy tool, especially when only a few years of data is available. Special care should be taken with solar data. Because of its daily pattern, a window with too many adjacent hours could return erroneous results.

A vignette in the package presents some additional examples and graphs for the use of this function. It can be accessed with `browseVignettes("sliding_window")`.

Value

An expanded data frame with the same format as `data.time`, but with a sliding window applied to the variable generation columns

See Also

format_timedata to create the time.data object

This function is called from calculate_metrics and calculate_elcc

Examples

```
# Create time data object
tdata <- data.frame(Area = c(rep("A", 48), rep("B", 48)),
                    Time = 1:48,
                    Load = c(runif(48, 200, 250), runif(48, 400, 450)),
                    Wind = c(runif(48, 20, 25), runif(48, 40, 45)))
td <- format_timedata(tdata)
head(td)

# If no data is provided, results remain intact
td2 <- sliding_window(td)

# Expand data for adjacent time steps (with equal and different weights)
td3 <- sliding_window(td, win.h.size = c(-1, 1))
td4 <- sliding_window(td, win.h.size = c(-1, 1), win.h.weight = c(1, 2, 1))

# Expand data for ajdacent days
td5 <- sliding_window(td, win.d.size = c(0, 1))

# Expand data for both adjacent times and days
td6 <- sliding_window(td, win.h.size = c(-1, 1), win.d.size = c(0, 1))
```

Index