Package ‘antaresProcessing’
February 26, 2020

Type Package
Title 'Antares' Results Processing
Version 0.18.0
Description Process results generated by 'Antares', a powerful open source software developed by RTE (Réseau de Transport d’Électricité) to simulate and study electric power systems (more information about 'Antares' here: <https://github.com/AntaresSimulatorTeam/Antares_Simulator>). This package provides functions to create new columns like net load, load factors, upward and downward margins or to compute aggregated statistics like economic surpluses of consumers, producers and sectors.

URL https://github.com/rte-antares-rpackage/antaresProcessing
BugReports https://github.com/rte-antares-rpackage/antaresProcessing/issues
License GPL (>= 2) | file LICENSE
LazyData TRUE
Depends antaresRead (>= 1.1.5)
Imports data.table, methods, stats
Suggests rhdf5 (>= 2.24.0), parallel, testthat, knitr, rmarkdown, covr
RoxygenNote 7.0.2
VignetteBuilder knitr
Encoding UTF-8
biocViews Infrastructure, DataImport
NeedsCompilation no

Author Veronique Bachelier [aut, cre], Jalal-Edine ZAWAM [aut], Francois Guillem [aut], Benoit Thieurmel [aut], Titouan Robert [aut], RTE [cph]

Maintainer Veronique Bachelier <veronique.bachelier@rte-france.com>
Repository CRAN
Date/Publication 2020-02-26 16:30:03 UTC
addDownwardMargin

**R topics documented:**

- addDownwardMargin ........................................ 2
- addExportAndImport ........................................ 3
- addLoadFactorLink .......................................... 4
- addNetLoad .................................................. 5
- addProcessingH5 ............................................. 6
- addUpwardMargin ............................................. 8
- compare ..................................................... 9
- externalDependency ......................................... 11
- getValues .................................................... 13
- loadFactor ................................................... 13
- mergeAllAntaresData ........................................ 15
- modulation ................................................... 15
- netLoadRamp .................................................. 17
- surplus ...................................................... 19
- surplusClusters ............................................. 21
- surplusSectors .............................................. 22
- synthesize ................................................... 24
- thermalGroupCapacities .................................... 25

**Index**

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>addDownwardMargin</td>
</tr>
</tbody>
</table>

*Add downward margins of areas*

**Description**

This function computes isolated and interconnected downward margins of areas and add them to an antaresData object.

**Usage**

```r
addDownwardMargin(x)
```

**Arguments**

- `x` An object of class antaresData created with `readAntares`

**Details**

For a given area, downward margin is equal to the thermal minimum production (due must run production and minimum stable power of production units) plus the fatal productions minus the load and the pumping capacity. More formally it is equal to:

\[
isolatedDownwardMargin = \text{thermalPMin} + \text{`H. ROR`} + \text{WIND} + \text{SOLAR} + \text{`MISC. NDG`} - \text{LOAD} - \text{pumpingCapacity}
\]

The variable `pumpingCapacity` is automatically created when pumped storage areas are removed with function `removeVirtualAreas`. If there is not any such area, `pumpingCapacity` is assumed to be equal to 0.
Interconnected downward margin is the isolated downward margin plus the exports minus the imports:

\[
\text{interconnectedDownwardMargin} = \text{isolatedDownwardMargin} + \text{BALANCE} - \text{ROW BAL}.
\]

Value

The function modifies its input by adding to it two new columns isolatedDownwardMargin and interconnectedDownwardMargin. For convenience it invisibly returns \( x \).

Examples

```r
## Not run:
# data required by the function
showAliases("downwardMargin")

mydata <- readAntares(select = "downwardMargin")
mydata <- removeVirtualAreas(mydata, getAreas(c("pump", "stor")))

addDownwardMargin(mydata)
names(mydata$areas)

## End(Not run)
```

---

**addExportAndImport**

*Export and import of areas or districts*

**Description**

This function computes the export and import of areas or districts and add it to an antaresData object.

**Usage**

```r
addExportAndImport(x, addCapacities = FALSE, opts = NULL)
```

**Arguments**

- **x**: an object of class "antaresDataList" created with the function readAntares. It has to contain some areas and all the links that are connected to these areas. Moreover the function removeVirtualAreas must be call before.
- **addCapacities**: If TRUE, export and import capacities are added.
- **opts**: opts
**addLoadFactorLink**

**Value**

`addExportAndImport` modifies its input by adding to it columns:

- `export`: export for an area or district
- `import`: import for an area or district
- `capExport`: capacity of export for an area or district, if `addCapacities` is set to TRUE
- `capImport`: capacity of import for an area or district, if `addCapacities` is set to TRUE

**Examples**

```
## Not run:
# Data required by the function
showAliases("exportsImports")

mydata <- readAntares(select = "exportsImports")
addExportAndImport(mydata)
names(mydata$areas)

## End(Not run)
```

---

**addLoadFactorLink**

*Load factors of link*

**Description**

This function computes the load factor of link and add it to an `antaresData` object.

**Usage**

`addLoadFactorLink(x)`

**Arguments**

- `x`: Object of class `antaresData` created with function `readAntares`. It must contain the columns `transCapacityDirect` and `transCapacityIndirect`.

**Value**

`addLoadFactorLink` modifies its input by adding to it two columns:

- `loadFactor`: Proportion of the installed capacity of a link that is effectively used:
  
  \[
  \text{loadFactor} = \frac{\text{FLOW LIN}}{\text{transCapacity}}
  \]

  Notice that `loadFactor` can be positive or negative according to the direction of the flow.

- `congestion`: 1 if the link is saturated (`loadFactor = +/-1`), 0 otherwise.

For convenience, the function invisibly returns the modified input.
addNetLoad

Examples

## Not run:
# Data required by the function
showAliases("loadFactorLink")

mydata <- readAntares(select = "loadFactorLink")
addLoadFactorLink(mydata)
names(mydata)

## End(Not run)

---

addNetLoad  

**Net load of areas**

Description

This function computes the net load of areas or districts and add it to an antaresData object. Net load is the load of an area minus productions that are not controlled: wind, solar, hydraulic run of river, etc. the production of clusters in must run mode is also subtracted by default.

Usage

```r
addNetLoad(x, ignoreMustRun = FALSE)
```

Arguments

- `x` An antaresData object created with readAntares. Unless ignoreMustRun is true, it must have a column mustRunTotal.
- `ignoreMustRun` If TRUE, the production in must run mode is not subtracted to the net load.

Value

addNetLoad modifies its input by adding to it a column "netLoad". For convenience, it invisibly returns the modified input. formula = LOAD - 'ROW BAL.' - PSP - 'MISC, NDG' - 'H. ROR' - WIND - SOLAR - mustRunTotal

Examples

## Not run:
# Data required by the function
showAliases("netLoad")

mydata <- readAntares(select = "netLoad")
addNetLoad(mydata)
names(mydata)

## End(Not run)
addProcessingH5  
*Add process results of antaresProcessing to an ANTARES .h5 files*

**Description**

In this version only hourly data can be enriched.

**Usage**

```r
addProcessingH5(
  opts = simOptions(),
  mcY = c("mcInd", "mcAll"),
  timeStep = "hourly",
  addNetLoad = FALSE,
  addDownwardMargin = FALSE,
  addUpwardMargin = FALSE,
  addExportAndImport = FALSE,
  addLoadFactorLink = FALSE,
  externalDependency = FALSE,
  loadFactor = FALSE,
  modulation = FALSE,
  netLoadRamp = FALSE,
  surplus = FALSE,
  surplusClusters = FALSE,
  thermalAvailabilities = FALSE,
  linkCapacity = FALSE,
  mustRun = FALSE,
  allProcess = FALSE,
  evalAreas = list(),
  evalLinks = list(),
  evalClusters = list(),
  evalDistricts = list(),
  nThreads = 1
)
```

**Arguments**

- `opts` simOptions obtain which setSimulationPath
- `mcY` character, "mcInd" or "mcAll".
- `timeStep` character, timeStep
- `addNetLoad` boolean refer to addNetLoad
- `addDownwardMargin` boolean refer to addDownwardMargin
- `addUpwardMargin` boolean refer to addUpwardMargin
addProcessingH5

addExportAndImport
  boolean refer to addExportAndImport
addLoadFactorLink
  boolean refer to addLoadFactorLink
externalDependency
  boolean refer to externalDependency
loadFactor
  boolean refer to loadFactor
modulation
  boolean refer to modulation
netLoadRamp
  boolean refer to netLoadRamp
surplus
  boolean refer to surplus
surplusClusters
  boolean refer to surplusClusters
thermalAvailabilities
  boolean Should the surplus of the last unit of a cluster be computed by surplusClusters. Should loadFactorAvailable be added to the result of loadFactor.
linkCapacity
  boolean should export and import capacities be computed by addExportAndImport.
mustRun
  boolean should the production in must run mode subtracted to the net load addNetLoad. Should the must run production be ignored in the computation of the netLoadRamp see netLoadRamp.
allProcess
  boolean All process in one argument.
evalAreas
  list, list of operation to evaluate in areas data
evalLinks
  list, list of operation to evaluate in links data
evalClusters
  list, list of operation to evaluate in clusters data
evalDistricts
  list, list of operation to evaluate in districts data
nThreads
  numeric, nThreads to use

Details

When you add a straitment, an alias is created. They can be used for request h5 file. See examples.

Available alias are :

- "Out_addDownwardMargin"
- "Out_addUpwardMargin"
- "Out_addExportAndImport"
- "Out_addLoadFactorLink"
- "Out_externalDependency"
- "Out_loadFactor"
- "Out_modulation"
- "Out_netLoadRamp"
- "Out_surplus"
- "Out_surplusClusters"
addUpwardMargin

Add upward margin of areas

Description

This function computes isolated and interconnected upward margins of areas and add them to an antaresData object.

Usage

addUpwardMargin(x)

Arguments

x An object of class antaresData created with readAntares
Details

For a given area and time step, isolated upward margin is the difference between the available production capacity plus the fatal productions and the load. More formally it is equal to:

\[
isolatedUpwardMargin = \left( \text{AVL DTG} + \text{generatingMaxPower} + \text{storageCapacity} \right) + \left( \text{'H. ROR'} + \text{WIND} + \text{SOLAR} + \text{'MISC. NDG'} \right) - \text{LOAD}
\]

NB: in Antares v6 (and earlier versions) \text{generatingMaxPower} is replaced by \text{hstorPMaxAvg}.

The variable \text{storageCapacity} is automatically created when pumped storage areas are removed with function \text{removeVirtualAreas}. If there is not any such area, \text{storageCapacity} is assumed to be equal to 0.

Interconnected upward margin is the isolated upward margin plus the imports and minus the exports:

\[
interconnectedUpwardMargin = isolatedUpwardMargin - \text{BALANCE} + \text{ROW BAL}.
\]

Value

The function modifies its input by adding to it two new columns \text{isolatedUpwardMargin} and \text{interconnectedUpwardMargin}. For convenience it invisibly returns \text{x}.

Examples

```r
## Not run:
# Data required by the function
showAliases("upwardMargin")

mydata <- readAntares(select = "upwardMargin")
mydata <- removeVirtualAreas(mydata, getAreas(c("pump", "stor"))

addUpwardMargin(mydata)
## End(Not run)
```

Description

\text{compare} has been designed to compare two surpluses created with function \text{surplus} but it can be used to compare the values of two tables of class \text{antaresData} that contain the same type of data.

Usage

\text{compare(x, y, method = c("diff", "ratio", "rate")}
Arguments

x  Table of class antaresData. x can be an antaresDataTable or antaresDataList.
y  Table of class antaresData. x can be an antaresDataTable or antaresDataList. It must contain the same type of data than 'x': if 'x' contains areas, it must contain areas, ... Moreover it has to have same time step and contain either synthetic or detailed results like 'x'.

method  Method used two compare the two tables. "diff" compute the difference between 'y' and 'x'. "ratio" computes the ratio between 'y' and 'x'. Finally, "rate" computes the rate of change between 'y' and 'x' (it is equal to the ratio between 'y' and 'x' minus one).

Value

a data.table of class antaresDataTable. It contains all shared rows and columns between 'x' and 'y'. The columns contain the statistic chosen: difference, ratio or rate of change.

Examples

## Not run:
# First simulation
studyPath <- "path/to/study/
setSimulationPath(studyPath, 1)
mydata1 <- readAntares("all", "all", synthesis = FALSE)
surplus1 <- surplus(mydata1, groupByDistrict = TRUE)

# Second simulation
setSimulationPath(studyPath, 2)
mydata2 <- readAntares("all", "all", synthesis = FALSE)
surplus2 <- surplus(mydata2, groupByDistrict = TRUE)

compare(surplus1, surplus2)

opts1 <- setSimulationPath(studyPath,-1)
mydata1<-readAntares(areas = "all", links = "all", select = c("allAreas", "allLinks"), mcYears = c(1), linkCapacity = TRUE)

opts2 <- setSimulationPath(studyPath,-2)
mydata2 <- readAntares(areas = "all", links = "all", select = c("allAreas", "allLinks"), mcYears = c(1), linkCapacity = TRUE)

opts3 <- setSimulationPath(studyPath,-3)
mydata3 <- readAntares(areas = "all", links = "all", select = c("allAreas", "allLinks"),
externalDependency

mcYears = c(1),
linkCapacity = TRUE)

opts4 <- setSimulationPath(studyPath, -4)
mydata4 <- readAntares(areas = "all",
links = "all",
select=c("allAreas", "allLinks"),
mcYears = c(1),
linkCapacity = TRUE)

opts5 <- setSimulationPath(studyPath, -5)
mydata5 <- readAntares(areas = "all",
links = "all",
select=c("allAreas", "allLinks"),
mcYears = c(1),
linkCapacity = TRUE)

resCompare1 <- compare(mydata2, mydata1, method = "diff")
resCompare2 <- compare(mydata3, mydata1, method = "diff")
resCompare3 <- compare(mydata4, mydata1, method = "diff")
resCompare4 <- compare(mydata5, mydata1, method = "diff")

listCompare <- list(resCompare1, resCompare2, resCompare3, resCompare4)

for (i in 1:length(listCompare)){
  listCompare[[i]] <- removeVirtualAreas(listCompare[[i]],
                                         storageFlexibility =
                                         getAreas(select = c("z_dsr", "y_mul", "pum", "tur")))
}

ml <- readRDS("path/to/mapLayout.rds")
plotMap(listCompare, ml)

## End(Not run)

---

### externalDependency

**External Dependencies in imports and exports**

**Description**

This function computes the dependency in imports and export for each area or districts at a given
time step. Dependency in imports represents moments where imports are required to have no loss
of load. Dependency in exports represents moments where exports are required to have no spilled
energy.

**Usage**

externalDependency(x, timeStep = "annual", synthesis = FALSE, opts = NULL)
Arguments

x  An object created with function readAntares. It must contain data for areas and/or districts. More specifically this function requires the columns generatingMaxPower (or hstorPMaxAvg for Antares v6 and earlier), and netLoad. To get these columns, one has to invoke readAntares with the parameter hydroStorageMaxPower = TRUE and addNetLoad (see examples). Moreover it needs to have a hourly time step. This object must also contain linkCapacity if there was virtual areas remove by removeVirtualAreas to be able to calculate pumping and storage capacities.

timeStep  Desired time step for the result.

synthesis  If TRUE, average external dependencies are returned. Else the function returns external dependencies per Monte-Carlo scenario.

opts  

Value

A data.table of class antaresDataTable with the following columns:

area  Area name.
timeId  Time id and other time columns.
pumping  capacity of pumping
storage  capacity of storage
exportsLevel  netLoad + pumping
importsLevel  netLoad - 'AVL DTG' - hydroStorageMaxPower - storage > 0
exportsFrequency  number of time step where this criteria is satisfied
criteria : netLoad + pumping < 0
importsFrequency  number of time step where this criteria is satisfied
criteria : netLoad - 'AVL DTG' - hydroStorageMaxPower - storage > 0

Examples

```r
## Not run:
# Data required by the function
showAliases("externalDependency")

mydata <- readAntares(select = "externalDependency")
addNetLoad(mydata)
externalDependency(mydata)

# if there are some virtual pumping/storage areas, remove them with
# removeVirtualAreas
mydata <- removeVirtualAreas(mydata, c("pumping", "storage"))
externalDependency(mydata, ignoreMustRun = TRUE)

## End(Not run)
```
**getValues**  
*Get values of a variable*

**Description**
Get all the values of a variable for some years Monte Carlo.

**Usage**
```
getValues(data = NULL, variable = NULL, mcyear = "all")
```

**Arguments**
- `data`: an object of class "antaresData" created with the function `readAntares`.
- `variable`: a variable of data.
- `mcyear`: set of mcYear.

**Examples**
```
## Not run:
mydata <- readAntares(areas="all",clusters="all", select="LOAD")
getValues(mydata$areas, variable="LOAD")
getValues(myData$clusters, variable = "production")

## End(Not run)
```

---

**loadFactor**  
*Load factors of clusters*

**Description**
This function computes the load factor and other related statistics for cluster of a study.

**Usage**
```
loadFactor(
  x,
  timeStep = "annual",
  synthesis = FALSE,
  clusterDesc = NULL,
  loadFactorAvailable = FALSE,
  opts = NULL
)
```
loadFactor

Arguments

x
Object of class antaresData created with function readAntares. It must contain hourly detailed results for clusters and has to contain the columns minGenModulation.

timeStep
Desired time step for the result.

synthesis
If TRUE, average surpluses are returned. Else the function returns surpluses per Monte-Carlo scenario.

clusterDesc
A table created with the function readClusterDesc. If this parameter is set to NULL (the default), then the function attempts to read the needed data in the same study as x.

loadFactorAvailable
Should loadFactorAvailable be added to the result?

opts
opts where clusterDesc will be read if null based on data

Value

a data.table of class antaresDataTable containing the following columns:

area
Area name

cluster
Cluster name

mcYear
Only if synthesis=FALSE. Id of the Monte-carlo scenario

timeId
Time id and other time variables

loadFactor
Load factor of the cluster. It represent the proportion of the installed capacity of a cluster that is effectively generate

Formula: production / (unitcount * nominalcapacity)

#'

loadFactorAvailable
Load factor of the cluster. It represent the proportion of the capacity available of a cluster that is effectively generate

Formula: production / thermalAvailability

propHoursMinGen
Proportion of hours when production is positive and all units of a cluster are either off, either producing at their minimum. This situation occurs when units are kept producing above the optimal level to avoid future startup costs or to satisfy the constraints generated by parameters "Min. up Time" or "Min gen. modulation".

Formula: mean(1 if production > 0 and production = max(min.stable.power * unitcount, minGenModulation * nominalcapacity * unitcount) else 0)

propHoursMaxGen
Proportion of hours when all units started produce at their maximal capacity.

Formula: mean(1 if production > 0 and production = NODU * nominalcapacity * (1 - spinning / 100))
mergeAllAntaresData

## Merge all antaresDataSets

### Description

Merge all antaresDataSets

### Usage

mergeAllAntaresData(dta)

### Arguments

- dta: antaresData

### Examples

```r
## Not run:
setSimulationPath("Mystud", 1)
dta <- readAntares(areas = "all", links = "all", clusters = "all", districts = "all")
dta <- mergeAllAntaresData(dta)
## End(Not run)
```

modulation

## Compute the modulation of cluster units

### Description

This function computes the modulation of cluster units or of sectors.
Usage

modulation(
  x,
  timeStep = "annual",
  synthesis = FALSE,
  by = c("cluster", "sector"),
  clusterDesc = NULL,
  opts = NULL
)

Arguments

x An antaresData object created with readAntares. It must contain the hourly detailed results for clusters if by = "cluster" or for areas and/or districts if by = "sector"
timeStep Desired time step for the result.
synthesis If TRUE, average surpluses are returned. Else the function returns surpluses per Monte-Carlo scenario.
by Should modulations computed by cluster or by sector? Possible values are "sector" and "cluster".
clusterDesc A table created with the function readClusterDesc. If is this parameter is set to NULL (the default), then the function attempts to read the needed data in the same study as x.
opts opts where clusterDesc will be read if null based on data

Value

A data.table of class antaresDataTable or a list of such tables with the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>Area name. If byDistrict=TRUE, this column is replaced by column district.</td>
</tr>
<tr>
<td>cluster</td>
<td>Cluster name. If by=&quot;sector&quot;, this column is replaced by column sector.</td>
</tr>
<tr>
<td>timeId</td>
<td>Time id and other time columns.</td>
</tr>
<tr>
<td>upwardModulation</td>
<td>Maximal absolute modulation of a cluster unit or of the sector, if timeStep is hourly.</td>
</tr>
<tr>
<td>downwardModulation</td>
<td>Maximal absolute modulation of a cluster unit or of the sector, if timeStep is hourly.</td>
</tr>
<tr>
<td>absoluteModulation</td>
<td>Maximal absolute modulation of a cluster unit or of the sector, if timeStep is hourly.</td>
</tr>
<tr>
<td>avg_upwardModulation</td>
<td>Average upward modulation of a cluster unit or of the sector, if timeStep is not hourly.</td>
</tr>
<tr>
<td>avg_downwardModulation</td>
<td>Average downward modulation of a cluster unit or of the sector, if timeStep is not hourly.</td>
</tr>
</tbody>
</table>
**netLoadRamp**

`avg_absoluteModulation`

Average absolute modulation of a cluster unit or of the sector, if `timeStep` is not hourly.

`max_upwardModulation`

Maximal upward modulation of a cluster unit or of the sector, if `timeStep` is not hourly.

`max_downwardModulation`

Maximal downward modulation of a cluster unit or of the sector, if `timeStep` is not hourly.

`max_absoluteModulation`

Maximal absolute modulation of a cluster unit or of the sector, if `timeStep` is not hourly.

Notice that if `by="cluster"`, the function computes the modulation per unit, i.e. the modulation of a cluster divided by the number of units of the cluster. On the opposite, if `by="sector"`, the function returns the modulation of the global production of the sector. Moreover, if parameter `x` contains area and district data, the function returns a list with components `areas` and `districts`.

**Examples**

```r
## Not run:
# data required by the function
showAliases("modulation")

mydata <- readAntares(select="modulation")

# Modulation of cluster units
modulation(mydata)

# Aggregate Monte-Carlo scenarios
modulation(mydata, synthesis = TRUE)

# Modulation of sectors
modulation(mydata, by = "sector")

# Modulation of sectors per district
modulation(mydata, by = "sector")

## End(Not run)
```

**Description**

This function computes the ramp of the consumption and the balance of areas and/or districts.
Usage

```r
netLoadRamp(
  x,
  timeStep = "hourly",
  synthesis = FALSE,
  ignoreMustRun = FALSE,
  opts = NULL
)
```

Arguments

- **x**: Object of class `antaresData` containing data for areas and/or districts. It must contain the column `BALANCE` and either the column "netLoad" or the columns needed to compute the net load see `addNetLoad`.
- **timeStep**: Desired time step for the result.
- **synthesis**: If TRUE, average surpluses are returned. Else the function returns surpluses per Monte-Carlo scenario.
- **ignoreMustRun**: Should the must run production be ignored in the computation of the net load?
- **opts**: opts where `clusterDesc` will be read if null based on data

Value

`netLoadRamp` returns a data.table or a list of data.tables with the following columns:

- **netLoadRamp**: Ramp of the net load of an area. If `timeStep` is not hourly, then these columns contain the average value for the given time step. Formula = `netLoad - shift(netLoad, fill = 0)`
- **balanceRamp**: Ramp of the balance of an area. If `timeStep` is not hourly, then these columns contain the average value for the given time step. Formula = `BALANCE - shift(BALANCE, fill = 0)`
- **areaRamp**: Sum of the two previous columns. If `timeStep` is not hourly, then these columns contain the average value for the given time step. Formula = `netLoadRamp + balanceRamp`
- **minNetLoadRamp**: Minimum ramp of the net load of an area, if `timeStep` is not hourly.
- **minBalanceRamp**: Minimum ramp of the balance of an area, if `timeStep` is not hourly.
- **minAreaRamp**: Minimum ramp sum of the sum of balance and net load, if `timeStep` is not hourly.
- **maxNetLoadRamp**: Maximum ramp of the net load of an area, if `timeStep` is not hourly.
- **maxBalanceRamp**: Maximum ramp of the balance of an area, if `timeStep` is not hourly.
- **maxAreaRamp**: Maximum ramp of the sum of balance and net load, if `timeStep` is not hourly.

For convenience the function invisibly returns the modified input.
surplus

Examples

## Not run:
# data required by the function
showAliases("netLoadRamp")

mydata <- readAntares(select="netLoadRamp")
netLoadRamp(mydata, timeStep = "annual")

## End(Not run)

---

surplus  Compute economic surplus

Description

This function computes the economic surplus for the consumers, the producers and the global surplus of an area.

Usage

surplus(
  x,
  timeStep = "annual",
  synthesis = FALSE,
  groupByDistrict = FALSE,
  hurdleCost = TRUE,
  opts = NULL
)

Arguments

x  an object of class "antaresDataList" created with the function readAntares. It has to contain some areas and all the links that are connected to these areas. Moreover it needs to have a hourly time step and detailed results.

timeStep  Desired time step for the result.

synthesis  If TRUE, average surpluses are returned. Else the function returns surpluses per Monte-Carlo scenario.

groupByDistrict  If TRUE, results are grouped by district.

hurdleCost  If TRUE, HURDLE COST will be removed from congestionFees.

opts  opts
Value

A data.table with the following columns:

area
Name of the area.

timeId
timeId and other time columns.

consumerSurplus
The surplus of the consumers of some area.
formula = (unsuppliedCost[area] - 'MRG. PRICE') * LOAD

producerSurplus
The surplus of the producers of some area.
formula = 'MRG. PRICE' * production - 'OP. COST'
Production includes "NUCLEAR", "LIGNITE", "COAL", "GAS", "OIL", "MIX. FUEL", "MISC. DTG", "H. STOR", "H. ROR", "WIND", "SOLAR" and "MISC. NDG"

rowBalanceSurplus
Surplus of the ROW balance.
Formula: 'MRG. PRICE' * 'ROW BAL.'

storageSurplus
Surplus created by storage/flexibility areas.
formula = storage * x$areas$'MRG. PRICE'

congestionFees
The congestion fees of a given area. It equals to half the congestion fees of the links connected to that area.
formula = (congestionFees - hurdleCost) / 2

globalSurplus
Sum of the consumer surplus, the producer surplus and the congestion fees.
formula = consumerSurplus + producerSurplus + storageSurplus + congestionFees + rowBalanceSurplus

Examples

```r
## Not run:
showAliases("surplus")

mydata <- readAntares(select="surplus")
surplus(mydata)

surplus(mydata, synthesis = TRUE)
surplus(mydata, synthesis = TRUE, groupByDistrict = TRUE)

## End(Not run)
```
surplusClusters

Compute the surplus of clusters

Description

This function computes the surplus of clusters of interest. The surplus of a cluster is equal to its production times the marginal cost of the area it belongs to minus variable, fixed and startup costs.

Usage

```r
surplusClusters(
  x,
  timeStep = "annual",
  synthesis = FALSE,
  surplusLastUnit = FALSE,
  clusterDesc = NULL,
  opts = NULL
)
```

Arguments

- **x**: An antaresData object created with readAntares. It must contain an element clusters and an element areas with at least the column MRG.PRICE.
- **timeStep**: Desired time step for the result.
- **synthesis**: If TRUE, average surpluses are returned. Else the function returns surpluses per Monte-Carlo scenario.
- **surplusLastUnit**: Should the surplus of the last unit of a cluster be computed? If TRUE, then x must have been created with the option thermalAvailabilities=TRUE in order to contain the required column "available units"
- **clusterDesc**: A table created with the function readClusterDesc. If this parameter is set to NULL (the default), then the function attempts to read the needed data in the same study as x.
- **opts**: opts where clusterDesc will be read if null based on data

Value

A data.table of class antaresDataTable with the following columns:

- **area**: Area name.
- **cluster**: Cluster name.
- **timeId**: Time id and other time columns.
- **variableCost**: Proportional costs of production of the cluster
  Formula = marginal cost * production
surplusSectors

fixedCost
Fixed costs of production of the cluster
Formula = NODU * fixed cost

startupCost
Start up costs of the cluster.

surplusPerUnit
Average surplus per unit of the cluster.
formula = ('MRG. PRICE' * production - opCost - startupCost) / unitcount

surplusLastUnit
Surplus of the last unit of the cluster.
formula = ('MRG. PRICE' * prodLastUnit - opCost / pmax(1, NODU) - startup.cost)

totalSurplus
Surplus of all units of the cluster.
formula = 'MRG. PRICE' * production - opCost - startupCost

economicGradient
Economic gradient of a cluster. It is equal to the surplus per unit divided by the capacity of a unit.
formula = surplusPerUnit / nominalcapacity

Examples

```
## Not run:
# Data required by the function:
showAliases("surplusClusters")
mydata <- readAntares(select = "surplusClusters")
surplusClusters(mydata)

# Computing the surplus of the last unit of a cluster requires the additional column "availableUnits". To add this column, one has to use parameter "thermalAvailabilities = TRUE" in readAntares.
mydata <- readAntares(select = c("surplusClusters", "thermalAvailabilities"))
surplusClusters(mydata, surplusLastUnit = TRUE)

## End(Not run)
```

surplusSectors

*Compute the surplus of sectors*

Description

This function computes the surplus of sectors for each area and time step. For sectors wind, solar, hydraulic storage and run of river, production costs are assumed to be equal to 0.
surplusSectors

Usage

surplusSectors(  
  x,  
  sectors = c("thermal", "renewable"),  
  timeStep = "annual",  
  synthesis = FALSE,  
  groupByDistrict = FALSE,  
  clusterDesc = NULL,  
  opts = NULL
)

Arguments

x Object of class antaresData created with readAntares. It needs to contain hourly detailed results of a simulation. Moreover, it must contain area data and if thermal sectors are required, cluster data.

sectors vector containing the name of the sectors for which surplus needs to be computed. Possible values are "thermal" for thermal sectors (nuclear, coal,...), "ren" for renewable energy and any column name that can be considered as a production (for instance production of virtual areas). It is assumed that the cost of these productions is equal to 0 as for renewable energies. If the parameter contains the value "thermal", then the parameter x has to contain cluster data.

timeStep Desired time step for the result.

synthesis If TRUE, average surpluses are returned. Else the function returns surpluses per Monte-Carlo scenario.

groupByDistrict If TRUE, results are grouped by district.

clusterDesc A table created with the function readClusterDesc. If this parameter is set to NULL (the default), then the function attempts to read the needed data in the same study as x.

opts opts

Value

A data.table of class "antaresData". It contains one column per sector containing the surplus of that sector for a given area and timeId.

Examples

## Not run:

# Data required by the function:
showAliases("surplusSectors")

mydata <- readAntares(select = "surplusSectors")
surplusSectors(mydata)

# Note that if the parameter "sectors" is modified, the function can require
# more or less data. For instance, if one only wants surplus for thermal sectors:
mydata <- readAntares(areas = "all", clusters = "all", synthesis = FALSE,
  select = "MRG. PRICE")
surplusSectors(mydata, sectors = "thermal")

## End(Not run)

synthesize

**Synthesize Monte-Carlo scenarios**

**Description**

This function takes as input an object of class antaresData containing detailed results of a simulation and creates a synthesis of the results. The synthesis contains the average value of each variable over Monte-Carlo scenarios and eventually other aggregated statistics.

**Usage**

```r
synthesize(x, ..., prefixForMeans = "", useTime = TRUE)
```

**Arguments**

- `x` an object of class antaresData created with `readAntares` and containing detailed results of an Antares simulation.
- `...` Additional parameters indicating which additional statistics to produce. See details to see how to specify them.
- `prefixForMeans` Prefix to add to the columns containing average values. If it is different than ",", a "_" is automatically added.
- `useTime` use times columns for synthesize.

**Details**

Additional statistics can be asked in three different ways:

1. A character string in "min", "max", "std", "median" or "qXXX" where "XXX" is a real number between 0 and 100. It will add for each column respectively the minimum or maximum value, the standard deviation, the median or a quantile.
2. A named argument whose value is a function or one of the previous aliases. For instance `med = median` will calculate the median of each variable. The name of the resulting column will be prefixed by "med_". Similarly, `1 = "q5"` will compute the 5 each variable and put the result in a column with name prefixed by "l_".
3. A named argument whose value is a list. It has to contain an element fun equal to a function or an alias and optionally an element only containing the names of the columns to which to apply the function. For instance med = list(fun = median, only = c("LOAD", "MRG. PRICE")) will compute the median of variables "LOAD" and "MRG. PRICE". The result will be stored in columns "med_LOAD" and "med_MRG. PRICE". The computation of custom statistics can take some time, especially with hourly data. To improve performance, prefer the third form and compute custom statistics only on a few variables.

Value

Synthetic version of the input data. It has the same structure as x except that column mcYear has been removed. All variables are averaged across Monte-Carlo scenarios and eventually some additional columns have been added corresponding to the requested custom statistics.

Examples

```r
## Not run:
mydata <- readAntares("all", timeStep = "annual")
synthesize(mydata)

# Add minimum and maximum for all variables
synthesize(mydata, "min", "max")

# Compute a custom statistic for all columns
synthesize(mydata, log = function(x) mean(log(1 + x)))

# Same but only for column "LOAD"
synthesize(mydata, 
    log = list(fun = function(x) mean(log(1 + x)),
               only = "LOAD"))

# Compute the proportion of time balance is positive
synthesize(mydata, propPos = list(fun = function(x) mean(x > 0),
                                  only = "BALANCE")

# Compute 95% confidence interval for the marginal price
synthesize(mydata, 
    l = list(fun = "q2.5", only = "MRG. PRICE"),
             u = list(fun = "q97.5", only = "MRG. PRICE"))

## End(Not run)
```
Description

compute thermal capacities from study

Usage

thermalGroupCapacities(opts = simOptions())

Arguments

  opts simOptions obtain which setSimulationPath
Index

addDownwardMargin, 2, 6
addExportAndImport, 3, 7
addLoadFactorLink, 4, 7
addNetLoad, 5, 6, 7, 12, 18
addProcessingH5, 6
addUpwardMargin, 6, 8

compare, 9

externalDependency, 7, 11

getValues, 13

loadFactor, 7, 13

mergeAllAntaresData, 15

modulation, 7, 15

netLoadRamp, 7, 17

readAntares, 2, 4, 8, 12, 14, 24
readClusterDesc, 14, 16, 21, 23
removeVirtualAreas, 2, 3, 9, 12

setSimulationPath, 6, 26

surplus, 7, 9, 19

surplusClusters, 7, 21

surplusSectors, 22

synthesize, 24

thermalGroupCapacities, 25