Package 'flexoptr'

February 18, 2021

Type Package

Title Optimise Energy Flexibilities

Version 1.0

Author Hendrik Obelöer

Maintainer <hendrik.obeloeer@haw-hamburg.de>

Description The goal of flexoptr is to provide a suite of functions to generalise and ease the modelling of energy flexibilities. By defining base parameters, the needs of a constrained flexibility are calculated, and optimised over price data. Several functions to facilitate the optimisation of more complex market and configuration analyses are also provided.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests testthat

Imports magrittr

R topics documented:

`adapt_constraints` *Adapt constraints to a charge input*

Description

Constraints built with `build_constraints` to incorporate the changes when a charge is added at an index. The values in the columns `cummax`, `cummin` and `dirmax` are updated accordingly.

Usage

```
adapt_constraints(constraints, index)
```

Arguments

`constraints`	a data frame of constraints, expected to resemble output of `build_constraints`.
`index`	a positive integer, the index where a unit of charge is added.

Value

the constraints data frame with updated values

`append_trades` *Extend a trade data frame with new trades*

Description

Extend a trade data frame with new trades

Usage

```
append_trades(new_trades, old_trades, time_offset = 0)
```

Arguments

`new_trades`	a data frame, formatted like the output of `describe_trades`.
`old_trades`	a data frame, formatted like the output of this function, can be NULL.
`time_offset`	the index of trades are assumed to be starting at 1, the offset is added to this index.

Value

a data frame, with all values from `new_trades` and `old_trades`

`build_constraints` *Calculate the cumulative minimum and maximum charge for a storage*

Description

Given the physical parameters, this function calculates the necessary, i.e lower limits, and possible, i.e. upper limits, for charging a storage.

Usage

```
build_constraints(
  cycles,
  state,
  capacity,
  loss_rate,
  charge_rate,
  parameters = NULL
)
```

Arguments

`cycles`	a positive integer, the number of cycles this function should consider
`state`	a positive integer, the starting state of energy in the storage
`capacity`	a positive integer, the maximum amount of energy that can be stored
`loss_rate`	a positive integer, the energy / cycle depleted from storage
`charge_rate`	a positive integer, the maximum energy / cycle which with the storage can be charged
`parameters`	a numerical, named vector, can substitute the use of the parameters `capacity`, `loss_rate`, and `charge_rate`. When used, all three values must be supplied by named values.

Details

Considering a steady loss rate in some kind of energy storage, this function calculates the cumulative minimal charge required to not go below zero charge.

In the same sense, a maximum cumulative charge is calculated which indicates the physical and realistic maximum of energy that could be put into the storage until it is full.

The function thinks in time cycles, where one would charge x amount of energy from the beginning of the cycle until the end of the same cycle.

Value

a data frame with cycle number, minimum, and maximum cumulative charge

Examples

```
build_constraints(10, 5, 20, 2, 4)
```

calc_revenue *Calculate the overall revenue of a trade data frame*

Description

Calculate the overall revenue of a trade data frame

Usage

```
calc_revenue(trade_df)
```

Arguments

trade_df	A data.frame as exported by describe_trades.

Value

A single number describing the revenue

describe_trades *Describe trades*

Description

Generate a trade list based on a previous and a new schedule. The function will analyse the differences between the schedules and infer the time and volume of the trades that occurred.

Usage

```
describe_trades(old_schedule, new_schedule, prices)
```

Arguments

old_schedule	a numerical vector
new_schedule	a numerical vector, must be same length as current_schedule.
prices	a numerical vector, must be same length as current_schedule.

Value

a data frame with columns describing the time at which a trade occurred (as index of old_schedule), volume and buy/sell price

Examples

```
schedule1 <- c(4, 0, 0, 3)
schedule2 <- c(0, 3, 0, 4)
new_prices <- c(70, 65, 80, 60)
describe_trades(schedule1, schedule2, new_prices)
```

filter_available_cycles *Select only viable charging times*

Description

A constraints data frame describes the physical limits of a charging process.

Usage

```
filter_available_cycles(constraints)
```

Arguments

constraints	a data frame with columns for cumulative minimum, cummin, cumulative maximum cummax, a direct maximum charge dirmax. The order of the rows is assumed to be strictly chronological without time lapses. The order of the data frame is assumed to represent the time hierarchy.

Details

The function selects the rows until cummin column first contains a value greater than zero (this row is also selected). Out of this selection only the rows with a cummax and a dirmax value greater than zero are selected.

Value

the original data with filtered rows.

See Also

build_constraints

format_da_prices *Make a day ahead price list*

Description

Reformats a vector of price data into a day ahead format that is compatible with optimise_schedule.

Usage

```
format_da_prices(prices)
```

Arguments

prices	a numerical vector of prices. Length must be a multiple of 24. Days with a change in daylight saving time can therefore not be handled.

Details

Data is grouped into groups of 24 and put into a list, the first element containing the first 24 prices, the next 23 elements being empty, the 25th element containing the next prices and so on.

Value

a list

See Also

optimise_schedule

Examples

```
some_prices <- rnorm(48, 20, 3)
format_da_prices(some_prices)
```

---

format_id_prices *Make an intra day price list*

Description

Reformats a data frame of index prices into a format compatible with optimise_schedule.

Usage

```
format_id_prices(pricetable, colnames)
```

Arguments

pricetable	a data frame with columns of price data. The order of the rows is assumed to be chronological. One row represents one hour.
colnames	a vector of strings. describes the names and order of columns that contain the price data. The first element represents the price one hour before market closure, the second element describes the price two hours before market closure and so on. The names cannot be repeated, so each hour needs to be coded into a separate column.

Details

Price data is typically formatted so that each row represents a product for a certain time. For each row there could a column describing the volume weighted average for the last hour, the last three hours and so on.

With this function, this data is transformed into a list format, where each element represents a time and contains the current prices at that point in time for the next hours. The current hour is assumed to be untradeable and therefore an NA is inserted for that time.

Value

the original data frame with the new price list as a further column

See Also

`optimise_schedule`

Examples

```
some_prices <- data.frame(
  time = c(1, 2, 3),
  id_1 = c(20, 30, 40),
  id_2 = c(22, 28, 39),
  id_3 = c(25, 27, 41)
)
format_id_prices(some_prices, c("id_1", "id_2", "id_3"))
```

---

identify_non_na_elements

Find the index of list elements that are not NA

Description

Find the index of list elements that are not NA

Usage

```
identify_non_na_elements(x)
```

Arguments

x	a list

Value

a vector of the numeric indexes where non-NA values are found

match_constraints *Limit constraints to respect reservations*

Description

This function allows to adapt constraints not only for a single index but for a range of indexes and a range of values.

Usage

```
match_constraints(constraints, fixed_schedule, untradeable = NULL)
```

Arguments

constraints	A data.frame with constraints.
fixed_schedule	A numeric vector describing which amounts should be applied to schedule. It is implicitly assumed that these values do not exceed the limits of the constraints.
untradeable	A numeric vector of indexes in which no trades can occur, so constraints are adapted accordingly.

Value

A data.frame of adapted, more limited constraints

Examples

```
some_constraints <- build_constraints(5, 2, 6, 1, 3)
already_scheduled <- c(0, 2, 2, 1, 0)
untradeable <- c(3, 4)
match_constraints(some_constraints, already_scheduled, untradeable)
```

optimise_constraints *Optimise a constrained schedule over price data*

Description

The order of the data frame and the price data is assumed to represent the time hierarchy.

Usage

```
optimise_constraints(constraints, prices, volume)
```

Arguments

constraints	a data frame with constraints, i.e. columns, for a cumulative minimum, cummin, a cumulative maximum cummax and a direct maximum charge dirmax.
prices	a vector of numeric prices, must have same length as rows of constraints.
volume	a positive integer, sets the number of units of energy that should be distributed over the possible times.

Value

A vector of integers representing the optimal schedule

See Also

optimise_schedule

Examples

```
sample_constraints <- build_constraints(10, 5, 20, 2, 4)
sample_prices <- sample.int(50, 10, replace = TRUE)
optimise_constraints(sample_constraints, sample_prices, 15)
```

---

optimise_schedule *Optimise a schedule with an iterating approach*

Description

The optimal possible schedule considering a set of prices a pre-existing schedule and physical constraints of a storage is calculated.

Usage

```
optimise_schedule(schedule, prices, parameters, shift, blocked = NULL)
```

Arguments

schedule	a numeric vector of the current schedule.
prices	a list of available prices. When an element consists only of NA-values (or a single), then that hour will not be iterated. When some price values inside a list element are NA, this will be interpreted to mean that only those hours are not tradeable.
parameters	a named vector of integers, including values for charge_rate, loss_rate, starting_state, and capacity.
shift	an integer, indicates the difference to the the sum of the current_schedule that is added or subtracted each iteration.
blocked	A logical vector indicating whether the schedule for that hour should be kept as is and not changed by the optimisation, same length as schedule.

Details

A current schedule is taken, the corresponding constraints are generated and then the optimal schedule in these constraints is calculated. The necessary trades are then recorded. This procedure is repeated as coded in the prices parameter.

The approach allows for overlapping optimisations, where the result of the previous run influences the outcome of the next - as is the case during a typical intra day optimisation.

Non-overlapping time frames are also handled, as is the case during a day ahead process where the shift represents the energy loss over 24 hours.

Value

A list with three elements, an optimised schedule, the states of the storage according to that new schedule and a data frame of corresponding trades.

See Also

`build_constraints`

---

shorten *Shorten a vector*

Description

Shorten a vector

Usage

```
shorten(x, by)
```

Arguments

`x`	a vector.
`by`	an integer value, giving the non-negative number of elements that should be left out, counted from the end of the vector.

Value

a vector

Examples

```
vec <- seq(10)
shorten(vec, 3)
```

simulate_marketing *Simulate different marketing scenarios for a set of parameters and prices*

Description

A single set of parameters, day ahead prices and intra day prices are used to construct the performance of marketing the parameters without storage on the day ahead market, with storage on the day ahead market, and with an intra day marketing on top of day ahead.

Usage

```
simulate_marketing(
  parameters,
  da_prices,
  id_prices,
  id_index_names,
  blocked_per_day = NULL,
  simplify = FALSE
)
```

Arguments

parameters	A named vector of parameters.
da_prices	A vector of day ahead prices, must be a multiple of 24.
id_prices	A data frame of intra day prices, different columns describe different indexes.
id_index_names	A vector of strings describing the order of columns for consecutive hours as encoded in the parameter id_prices.
blocked_per_day	An optional vector of 24 integers describing the power that is predetermined for certain hours and cannot be optimised.
simplify	Whether the output should be a simple data frame describing the revenues or a more complex list with the whole trade logs, schedule and state.

Value

A list of results for each of the three scenarios or a data frame with the respective revenues.

See Also

format_id_prices, format_da_prices, optimise_schedule

simulate_no_storage *Mock the output* optimise_schedule *for settings without storage*

Description

Mock the output optimise_schedule for settings without storage

Usage

```
simulate_no_storage(parameters, prices)
```

Arguments

parameters	a named vector, which must contain loss_rate and starting_state.
prices	a numeric vector.

Value

a list of schedule, state, ans trades. Compatible with optimise_schedule

split_schedule *Split a schedule in a fixed and a flexible part*

Description

The parameters schedule, reservations and available_prices should all be of same length.

Usage

```
split_schedule(schedule, reservations, available_prices)
```

Arguments

schedule	A numeric vector of the current schedule.
reservations	A numeric vector describing reserved amounts of power.
available_prices	A numeric vector. When an element is NA, this element is considered to be untradeable, therefore the schedule is taken as fixed for that index.

Value

A list of three vectors, the first element describes fixed parts of the schedule, the second the flexible parts, and a third contains the indexes where no trade is possible

Examples

```
my_schedule <- c(0, 0, 3, 2, 3, 4)
reserved_capacity <- c(TRUE, FALSE, FALSE, FALSE, TRUE, FALSE)
available_prices <- c(30, 40, 20, NA, NA, 5)
split_schedule(my_schedule, reserved_capacity, available_prices)
```

Index