Package 'amss'

June 6, 2017

Version 1.0.1

Date 2017-06-06

Title Agreggate Marketing System Simulator

Author Google Inc. <amss-opensource@google.com>

Maintainer Google Inc. <amss-opensource@google.com>

Depends R (>= 3.2.2)

Imports assertthat (>= 0.1.0.99), data.table (>= 1.9.6)

Suggests knitr (>= 1.14), testthat (>= 1.0.2)

VignetteBuilder knitr

Description Implementation of the Aggregate Marketing System Simulator. Functions for simulating aggregate time series marketing data, and for calculating associated ground truth for some relevant metrics.

License Apache License 2.0 | file LICENSE

RoxygenNote 5.0.1

R topics documented:

`CalculateROAS` *Calculate ROAS or mROAS.*

Description

This functions takes the original budget settings and a counterfactual budget setting. It reports the expected ratio between the total difference in revenue over all time points and the total difference in media spend over all time points.

Usage

```
CalculateROAS(object, new.budget = NULL,
  media.names = object$params$media.names, budget.periods = NULL,
  budget.proportion = rep(0, length(media.names)), t.start = 1,
  t.end = object$params$time.n, scaled.pop.size = 1e+18, min.reps = 2,
  max.coef.var = 0.01, max.margin.error = 0.01, max.time = 0,
  verbose = FALSE)
```

Arguments

`object`	amss.sim object containing simulated data
`new.budget`	table of new budgets for each budget period (row) and media channel (column)
`media.names`	if new.budget is NULL, adjust original budget of the media named here.
`budget.periods`	budget.periods over which to modify the budget. Default `NULL` will lead to all budget periods being modified.
`budget.proportion`	nonnegative numeric. When `new.budget` is NULL, it is calculated by setting the budget of the media channels specified in `media.names` to `budget.proportion` proportion of the original budget during the budget periods specified in `budget.periods`. The default proportion of 0 is used to calculate the average ROAS over the entire spend in the channel. Values such as 0.99 can be used to calculate the marginal ROAS.
`t.start`	time point to start generating data according to the new settings.
`t.end`	last time point to generate data according to the new settings. In scenarios with lag, this should extend past the last time point in the modified budget periods in order to include lagged effects in the calculation.
`scaled.pop.size`	`CalculateROAS` scales up the population size to reduce the variability of its estimates. This is equivalent to running the simulation for multiple repetitions to reduce variability. The default value should provide sufficient accuracy in most use cases. Extremely large values may result in numerical issues.
`min.reps`	integer representing the initial number of datasets to generate from each budget setting. The default value of 2 allows the user to make a rough check that the accuracy is indeed good under the chosen settings. This default was chosen

under the assumption that the default `scaled.pop.size` is large enough to accurately measure ROAS using 1 repetition, with the 2nd being used as confirmation of the accuracy. Higher precision and more accurate measurement of the precision can be achieved with more repetitions.

`max.coef.var` numeric, the target coefficient of variation. The function takes additional samples of the ROAS until it runs out of time, attains the target coefficient of variation, or attains the target margin of error.

`max.margin.error` numeric, the target margin of error. The function takes additional samples of the ROAS until it runs out of time, attains the target coefficient of variation, or attains the target margin of error.

`max.time` numeric, the number of minutes at which to cut off the function from taking additional samples beyond the initial sample generated according to `min.reps`. The function takes additional samples of the ROAS until it runs out of time, attains the target coefficient of variation, or attains the target margin of error. The default value of 0 forces the function to use precisely `min.reps` repetitions.

`verbose` boolean. If TRUE, output measures of the accuracy of the reported ROAS, including the full sample of ROAS values.

Value

numeric value for ROAS, or, if `verbose = TRUE`, a list with the roas, the 95 sample ROAS values.

`DefaultNatMigModule` *Model natural consumer behavior in the absence of marketing interventions.*

Description

This function models natural consumer behavior in the absence of marketing interventions. In particular, it models changes in consumer mindset over time that are outside of advertiser control, such as seasonal changes.

Usage

```
DefaultNatMigModule(data.dt, population, market.rate.trend = 1,
  market.rate.seas = 1, sat.decay = 1,
  prop.activity = rep(1/length(kActivityStates), length(kActivityStates)),
  prop.favorability = rep(1/length(kFavorabilityStates),
  length(kFavorabilityStates)), prop.loyalty = rep(1/length(kLoyaltyStates),
  length(kLoyaltyStates)),
  prop.availability = rep(1/length(kAvailabilityStates),
  length(kAvailabilityStates)), transition.matrices = list())
```

Arguments

`data.dt`	data.table with rows corresponding to segments and columns corresponding to variables; column `pop` for the number of people in each segment must be included.
`population`	constant specifying population size
`market.rate.trend`	the trend in market size, written as the proportion of the population to be considered potentially in the market, pending seasonal adjustments. If a vector, should match time.n in length. Defaults to 1, for full population participation in market.
`market.rate.seas`	the seasonal variation in market size, written as the proportion of the post-market-trend population in the market. For example, for market.rate.trend = 0.8 and market.rate.seas = 0.5, seasonal variation leaves 40 potentially in market according to market.rate.trend actually in market. If a vector, should match time.n in length. Defaults to 1 for full population participation in market.
`sat.decay`	single numeric value between 0 and 1, representing the geometric decay rate at which satiated individuals become unsatiated. Defaults to 1 for satiation lasting 1 time period for all individuals.
`prop.activity`	vector of nonnegative values summing to 1, representing the proportion of the population to be assigned to each activity state, given they are "responsive," i.e., "in.market" and "unsatiated."
`prop.favorability`	vector of nonnegative values summing to 1, representing the proportion of the population to be assigned to each favorability state, given they are not "loyal."
`prop.loyalty`	vector of nonnegative values summing to 1, representing the proportion of the population to be assigned to each loyalty state.
`prop.availability`	vector of nonnegative values summing to 1, representing the proportion of the population to be assigned to each availability state.
`transition.matrices`	list of matrices for each dimension of population segmentation that may be affected by marketing interventions. A named list with members 'activity', 'favorability', 'loyalty', and 'availability' is expected. By default, any missing members will have no effect. The transition matrices represent natural migration in these dimensions, and control how quickly the population returns to its equilibrium allocation across segments after marketing interventions.

Value

`invisible(NULL)`. `data.dt` is updated by reference.

`DefaultSalesModule` *Model advertiser and competitor sales.*

Description

Simulate consumer purchase behavior, and thus the advertiser's and its competitors' sales.

Usage

```
DefaultSalesModule(data.dt, price, mean.price = 0,
  advertiser.demand.intercept = list(),
  advertiser.demand.slope = list(favorability = rep(0,
  length(kFavorabilityStates))), competitor.demand.max = list(loyalty = c(1,
  0, 1)), competitor.demand.replacement = list(loyalty = c(0.5, 0, 1)),
  purchase.quantity.intercept = 1, purchase.quantity.slope = 0,
  purchase.quantity.competitor = 1, unit.cost = 0,
  advertiser.transitions = list(), competitor.transitions = list())
```

Arguments

`data.dt`	data.table with rows corresponding to population segments and columns corresponding to specific variables
`price`	numeric vector of product price over time. If the vector is shorter than the number of timepoints, it is repeated as necessary.
`mean.price`	numeric scaler, the mean of price over time. Defaults to zero.
`advertiser.demand.intercept`	list of numeric vectors corresponding to each brand state (favorability, loyalty, and availability). The product of multiplicands corresponding to a particular segment with 'purchase' activity state is the probability consumers in that segment will purchase the advertiser's product if the price is mean.price and there is no competition. Missing members of the list have no effect on the calculation.
`advertiser.demand.slope`	list of numeric vectors corresponding to each brand state (favorability, loyalty, and availability). The product of multiplicands corresponding to a particular segment with 'purchase' activity state is the linear decrease in the probability consumers in that segment will purchase the advertiser's product when the price increases by 1, when there is no competition. Missing members of the list have no effect on the calculation.
`competitor.demand.max`	list of numeric vectors corresponding to each brand state (favorability, loyalty, and availability). The product of multiplicands corresponding to a particular segment with 'purchase' activity state is the probability consumers in that segment will purchase a competitor's product when advertiser's product is too expensive to be a feasible choice. Missing members of the list have no effect on the calculation.

`competitor.demand.replacement`
: list of numeric vectors corresponding to each brand state (favorability, loyalty, and availability). The product of multiplicands corresponding to a particular segment specifies the degree to which advertiser and competitor sales are replacements for each other. At 1, competitor sales are unaffected by advertiser pricing, and competitor sales replace advertiser sales to the greatest degree possible. At 0, advertiser sales are unaffected by the presence of the competitor, and advertiser sales replace competitor sales to the greatest degree possible. Thus, a reasonble interpretation of consumer loyalty might set this parameter to `list(loyalty = c(0.5, 0.1, 0.9)`. Missing members of the list have no effect on the calculation.

`purchase.quantity.intercept`
: numeric, at least 1. Represents the average number of units bought by each consumer purchasing from the advertiser's brand, if price is mean.price.

`purchase.quantity.slope`
: numeric, generally >= 0. Represents the decrease in the average purchase quantity per consumer purchasing from the advertiser's brand given a unit increase in price. Missing members of the list have no effect on the calculation.

`purchase.quantity.competitor`
: average number of units bought by consumers purchasing a comeptitor's product. Must be at the least the default value of 1.

`unit.cost`
: numeric greater than 0, cost of goods sold, for one unit of the advertiser's product.

`advertiser.transitions`
: list of transition matrices for each brand state, specifying post-purchase changes in consumer mindset for those who purchased the advertiser's brand. A named list with members 'favorability', 'loyalty', and 'availability' is expected. Any missing members will have no effect. The default value, `list()` results in no post-purchase migration.

`competitor.transitions`
: list of transition matrices for each brand state, specifying post-purchase changes in consumer mindset for those who purchased a competitor's brand. A named list with members 'favorability', 'loyalty', and 'availability' is expected. Any missing members will have no effect. The default value, `list()` results in no post-purchase migration.

Value

`invisible(NULL)`. `data.dt` updated by reference.

`DefaultSearchMediaModule`

Model paid and/or organic search.

Description

Simulate the behavior of a paid and/or organic search, including observable variables (e.g., query volume, paid clicks, spend) and the effect on consumer mindset.

Usage

```
DefaultSearchMediaModule(data.dt, budget.index, budget,
  spend.cap.fn = function(time, budget, budget.indices) {     Inf },
  bid.fn = function(time, per.capita.budget, budget.indices) {     Inf },
  kwl.fn = function(time, per.capita.budget, budget.indices) {     1 },
  audience.membership = list(), query.rate = 1, cpc.min = 0,
  cpc.max = 1, ctr = list(), relative.effectiveness = c(0, 0, 1),
  transition.matrices = list())
```

Arguments

`data.dt`	data.table with rows corresponding to population segments and columns corresponding to specific variables
`budget.index`	vector specifying budget period each time point belongs to. For example, rep(1:4, each = 52) would correspond to 4 years of yearly budget periods.
`budget`	vector specifying the target spend for each budget period. For example, given the example `budget.index` from above, `budget = rep(1e6, 4)` would specify a budget of 1 million for each year.
`spend.cap.fn`	function mapping the current time, the budget, and the budget period to a spend cap for the current week. By default this is set to `Inf`, representing uncapped spend.
`bid.fn`	function mapping the current time, the per-capita budget over the population, and the budget period to a bid for the current week. By default this is set to `Inf`, so that the advertiser wins all auctions and will pay the maximum CPC.
`kwl.fn`	function mapping the current time, the per-capita budget over the population, and the budget period to the proportion of queries. that match the keyword list. By default this is the maximum value of 1. To specify the proportion of matching queries by population segment, have kwl.fn return a vector with entries for each segment.
`audience.membership`	list of multipliers used to calculate probability of audience membership. Each element of the list corresponds to a specific dimension of population segmentation. Multipliers corresponding to each dimension are multiplied to derive audience membership probability for each segment. A named list with members 'activity', 'favorability', 'loyalty', and 'availability' is expected. Each member is a numeric vector containing the multipliers to use for each state in the dimension. For example, if member "activity" is c(1, 0.5, 0.7), a multiplier of 0.7 should be used for all segments with activity state "purchase." By default, any missing members will have no effect.
`query.rate`	nonnegative numeric, or vector. Each member of the audience makes matching queries according to a Poisson process with this rate. A vector rate specifies the query rate at each time. Note that rate is the expected number of queries per person in the audience. Defaults to 1. Vector repeats as necessary, so that repeating patterns can be specified more simply.
`cpc.min`	minimum CPC, defaults to 1. Must be nonnegative. vector values are interpreted as the vector of minimum CPC's over time.

`cpc.max` maximum CPC. Must be at least as large as cpc.min. vector values are interpreted as the vector of maximum CPC's over time.

`ctr` list of multipliers for each dimension with an effect on the clickthrough rate (ctr). Values in each state are multiplied to derive the ctr for each population segment. A named list with members 'activity', 'favorability', 'loyalty', and 'availability' is expected. Each member is a numeric vector of the values for each state in that dimension. By default, any missing members will have no effect.

`relative.effectiveness` effectiveness, relative to the maximum effectiveness specified by the transition matrices, by volume type: organic only, paid impressions w/o paid click (click on organic result included), and paid clicks. Default to maximum (1) effectiveness for paid clicks, and no effect otherwise.

`transition.matrices` list of transition matrices for each dimension of population segmentation that may be affected by marketing interventions. A named list with members 'activity', 'favorability', 'loyalty', and 'availability' is expected. By default, any missing members will have no effect.

Value

`invisible(NULL)`. `data.dt` updated by reference.

`DefaultTraditionalMediaModule` *Model the effect of a traditional media channel.*

Description

Simulate the behavior of a traditional media channel, and generate associated observable variables such as media volume and spend.

Usage

```
DefaultTraditionalMediaModule(data.dt, budget.index, budget,
  audience.membership = list(), flighting = rep(1, length(budget.index)),
  unit.cost = 1, effectiveness.function = NULL, hill.ec = 1,
  hill.slope = 1, transition.matrices = list())
```

Arguments

`data.dt` data.table with rows corresponding to population segments and columns corresponding to specific variables

`budget.index` vector specifying budget period each time point belongs to. For example, rep(1:4, each = 52) would correspond to 4 years of yearly budget periods.

`budget` vector specifying the target spend for each budget period. For example, given the example `budget.index` from above, `budget = rep(1e6, 4)` would specify a budget of 1 million for each year.

`audience.membership`
: list of multipliers used to calculate probability of audience membership. Each element of the list corresponds to a specific dimension of population segmentation. Multipliers corresponding to each dimension are multiplied to derive audience membership probability for each segment. A named list with members 'activity', 'favorability', 'loyalty', and 'availability' is expected. Each member is a numeric vector containing the multipliers to use for each state in the dimension. For example, if member "activity" is c(1, 0.5, 0.7), a multiplier of 0.7 should be used for all segments with activity state "purchase." By default, any missing members will have no effect.

`flighting`
: specifies the relative amount to be spent on each time point within a budget period. For example, in a budget period of two weeks, `flighting = c(1,2)` specifies that twice 1/3 of the budget should be spent in the first week, and 2/3 in the second.

`unit.cost`
: positive numeric specifying expected unit cost per exposure.

`effectiveness.function`
: vectorized function mapping frequency to media effect (relative to transition matrices specifying maximum effect). The range of the function should be bounded between 0 and 1. Given the default value of NULL, the module will used the Hill transformation with parameters hill.ec and hill.slope.

`hill.ec`
: parameter controlling the scaling of frequency vs. effect. This is the EC50 of the Hill transformation.

`hill.slope`
: parameter controlling the scaling of frequency vs. effect. This is the maximum slope of the Hill transformation.

`transition.matrices`
: list of transition matrices for each dimension of population segmentation that may be affected by marketing interventions. A named list with members 'activity', 'favorability', 'loyalty', and 'availability' is expected. By default, any missing members will have no effect.

Value

`invisible(NULL)`. `data.dt` updated by reference.

`GetBudgetIdx` *Get the budget period assigned to each time interval.*

Description

Read the budget indices from an `amss.sim` object.

Usage

```
GetBudgetIdx(object)
```

Details

The budget indices specify which time intervals belong to the same budget period.

`OptimizeSpend` *Optimize the media budgets in a specified budget period.*

Description

Given a budget period and a set of constraints, find the budget setting and the associated media spend that maximizes the profit (revenue minus cost of production and advertising spend).

Usage

```
OptimizeSpend(object, budget.period = max(GetBudgetIdx(object)),
  t.start = match(budget.period, GetBudgetIdx(object)),
  t.end = object$params$time.n, lower.bound = 0, upper.bound = Inf,
  sum.lower.bound = 0, sum.upper.bound = Inf, scaled.pop.size = 1e+18)
```

Arguments

`object`	an object of class `amss.sim`
`budget.period`	numeric, the budget period to be optimized, the default being the most current budget period.
`t.start`	integer, the first time interval over which the result (profit) of the budget settings should be calculated.
`t.end`	integer, the last time interval over which the result (profit) of the budget settings should be calculated.
`lower.bound`	numeric vector, the lower bound on the budget for each media channel.
`upper.bound`	numeric vector, the upper bound on the budget for each media channel.
`sum.lower.bound`	numeric, the lower bound on the total advertising spend.
`sum.upper.bound`	numeric, the upper bound on the total advertising spend.
`scaled.pop.size`	numeric, the population is scaled to this size in order to increase the accuracy of estimated expected profit.

Details

See `DefaultSalesModule` for details on how the relationship between revenue, profit, units sold, and advertising spend is specified.

Value

`OptimizeBudget` returns a `list` with elemtnts

opt.spend the optimal spend in each media channel.

opt.budget the optimal budget in the specified budget period.

opt.profit the profit resulting from the optimal budget.

orig.profit the profit in the original dataset.

Note

A module does not necessarily force the spend in a budget period to match the budget. For example, in the paid search module, the budget is used as the lever that leads to increasing/decreasing search spend. Users should expect a monotonic relationship between budget and spend, but no more. The budget is useful as a parameter in simulation and optimization, as it is the lever moving advertiser-controlled settings in each media channel. The spend, which may depend on other factors outside of the advertiser's control, cannot be directly optimized; it is not a direct input into the simulator. However, any budget settings can be mapped to a corresponding media spend, and this is reported as the optimal spend. The optimal spend is more meaningful than the budget as a reporting metric, and is the key output of `OptimizeSpend`.

Examples

```
## Not run:
# Use the amss.sim object test.data from the testing suite.
# Find the optimal budget for the third budget period.
OptimizeSpend(test.data, budget.period = 3)

## End(Not run)
```

`population segmentation`

Population segmentation constants

Description

AMSS segments the population segments the population into groups based on each consumer's current mindset with regards to the category and the advertiser's brand. Aggregate changes in the population are tracked through the size of, i.e., the number of individuals belonging to, each population segment.

Format

An object of class `character` (all possible states in a single dimension) or `data.table` (each row specifying a valid combination of states in different dimensions).

Details

The consumer minset is defined along six dimensions. The first three specify the consumer's relationship with the category:

Market state describes whether the consumer should be considered part of the market for this category. Consumers with no interest are `out.of.market`; the rest are `in.market`. A consumer's market state may vary over time due to seasonal changes in consumer demand, but generally should not be affected by marketing interventions.

Satiation state tracks whether a consumer's demand for the product category is temporarily satisfied by a recent purchase.

Activity state tracks the consumer's progress along the path to purchase. Consumers may be in the `inactive`, `exploratory`, or `purchase` state. Consumers in different activity states will have different behaviors. For example, by default consumers outside the `purchase` state will never make a purchase. Activity state also affects media consumption; for example, individuals who are not `inactive` are generally more likely to make generic or branded search queries.

The last three dimensions describe the consumer's relationships with the advertiser's brand.

Brand favorability state specifies a consumer's awareness of and opinion of the advertiser's brand. Consumers are either `unaware`, or are aware and have an opinion of the brand ranging from `negative` to `favorable`, with intermediate favorabilitiy levels `neutral` and `somewhat favorable`.

Brand loyalty state specifies a consumer's loyalty status. A consumer may be a `switcher`, in which case he or she has no brand loyalty. Otherwise the consumer is either `loyal`, i.e., loyal to the advertiser's brand, or `competitor.loyal`.

Brand availability state refers to whether the advertiser's product is easily available to a particular consumer. For example, if the advertiser's distribution efforts only cover seventy percent of the population, then the thirty percent of the population not covered would be in the `low` brand availability state. The other options are `average` and `high` brand availability. Availability can refer to physical availability, i.e. the presence of the advertiser's product on store shelves. It could also refer to the mental availability (convenience) of the advertiser's brand. Thus brand availability can be affected by, say search ads that make the advertiser's brand the most prominent on the search results page, or by having the advertiser's product at eye-level in a store shelf.

The constants `kMarketStates`, `kSatiationStates`, `kActivityStates`, `kFavorabilityStates`, `kLoyaltyStates`, and `kAvailabilityStates` list the the possible states a consumer may take in each dimension as character vectors.

A consumer's mindset is summarized by the combination of states they take in each dimension. There are certain restrictions on which combinations of consumer states are possible. For example, only consumers who are both `in.market` and `unsatiated` can leave the `inactive` activity state. The `data.frame` `kCategoryStates` describes all valid combinations of market state, satiation state, and activity state, and thus lists all possible consumer mindsets with respect to the category in general. The `data.frame` `kBrandStates` describes all valid combinations of brand favorability, loyalty, and availability, given that only consumers with a `favorable` opinion of the brand can be `loyal`. Thus, `kBrandStates` lists all possible consumer mindsets with regards to the advertiser's brand.

A `data.table` of all valid consumer states is provided as `kAllStates`. It is the cross product of all category and brand states. Every consumer is assigned to one of these 198 states.

SimulateAMSS *Generate simulation objects under the AMSS framework.*

Description

Produces an amss.sim object that contains the simulated data and can be used to derive ground truth about the scenario.

Usage

```
SimulateAMSS(time.n, nat.mig.module = DefaultNatMigModule,
  nat.mig.params = list(), media.names = character(),
  media.modules = rep(list(DefaultTraditionalMediaModule),
  length(media.names)), media.params = rep(list(list()), length(media.names)),
  sales.module = DefaultSalesModule, sales.params = list(), ping = max(10,
  floor(time.n/10)), names.agg.const = NULL, names.agg.sum = NULL)
```

Arguments

time.n	number of timepoints.
nat.mig.module	specifications of class seq.specs, to be used to create the non-actionable drivers module and then generate its variables
nat.mig.params	any parameter values to pass to nat.mig.module
media.names	character vector of unique names of all media modules. ex: c("tv", "search")
media.modules	list of functions that simulate the behavior of each marketing intervention.
media.params	list of parameter value lists for each media module.
sales.module	function that models the sales in the category.
sales.params	list of any parameter values to pass to the sales module
ping	the spacing between time points at which to print a message updating the user on simulation progress.
names.agg.const	character vector of names of variables to surface in the observed data, aggregated using the first entry, since they are constant over the hidden states. By default, if not specified, SurfaceData() will pick up variables with names containing "price" and/or "budget", and the "pop.total" variable.
names.agg.sum	character vector of names of variables to surface in the observed data, aggregated using the function sum(). By default, SurfaceData() will pick up variables with names matching "revenue", "profit", "sales", "volume", and/or "spend".

Details

Objects of class `amss.sim` contain the full output from running a simulation scenario. This output includes the observed data, the complete dataset generated during the simulation process, and the parameters passed to the simulation function in order to generate the simulated data. The observed data is meant to be used by modelers. The complete dataset can be useful for users who want a more complete understanding of the forces operating in a simulation scenario. The parameter list is essential for generating future datasets based on the same, or slightly modified, simulation settings in order to obtain ground truth about the simulation scenario.

Value

an object of class `amss.sim`, containing

data the observed data.

data.full the full dataset, as a list of data.tables. Each `data.table` contains the data at the end of a time interval, by by population segment (row) and variable (column).

params the parameters used to generate the data.

`SimulateAR1` *Simulate AR1 time series*

Description

Function that outputs simulated AR1 time series with specified means, variances, and autocorrelations

Usage

```
SimulateAR1(n, stable.mu = 0, stable.sd = 1, autocor = 0)
```

Arguments

`n`	integer number of time points
`stable.mu`	means of the stable distribution for each variable
`stable.sd`	standard deviations of the stable distributions
`autocor`	autocorrelations for each time series

Value

vector realization of specified AR1 times series

See Also

Other Simulate.time.series: `SimulateCorrelated`, `SimulateDummy`, `SimulateSinusoidal`

`SimulateCorrelated` *Simulate correlated vectors.*

Description

Simulates a new vector x with a specified mean, standard deviation, and correlation with some other vector `v` by adding white noise and scaling.

Usage

```
SimulateCorrelated(v, cor.vx = 1, mu.x = 0, sigma.x = 1)
```

Arguments

`v`	vector to which the new data will be correlated.
`cor.vx`	numeric specifying correlation between v and the new vector x.
`mu.x`	numeric, mean of the new vector x.
`sigma.x`	numeric, standard deviation of the new vector x.

Value

a vector x with the specified mean, standard deviation, and correlation.

See Also

Other Simulate.time.series: `SimulateAR1`, `SimulateDummy`, `SimulateSinusoidal`

`SimulateDummy` *Simulate dummy (0-1) variables.*

Description

Create dummy (0-1) variables that repeat a requested pattern of 0's and 1's, with the option to scale.

Usage

```
SimulateDummy(n, pos.idx = NULL, period = n, amplitude = 1)
```

Arguments

`n`	integer number of time points
`pos.idx`	vector of indices where simulated vector should take positive values (as opposed to zero)
`period`	integer controlling periodicity.
`amplitude`	numeric value > 0 specifying the value of all postive entries in the return vector.

Value

specified vector

See Also

Other Simulate.time.series: `SimulateAR1`, `SimulateCorrelated`, `SimulateSinusoidal`

`SimulateSinusoidal` *Generate sinusoidal time series.*

Description

Function that outputs specified sinusoidal waves.

Usage

```
SimulateSinusoidal(n, period, max.loc = 1, vert.translation = 0,
  amplitude = 1, scale.x = FALSE)
```

Arguments

`n`	the length of the simulated vector.
`period`	the length of one full sinusoidal period.
`max.loc`	the index of the maximum of the sinusoidal curve.
`vert.translation`	a numeric for the vertical displacement of the sinusoidal curve from 0.
`amplitude`	numeric for the amplitude of the sinusoidal curve. Must be nonnegative.
`scale.x`	boolean. If TRUE, scale the sinusoidal curve to have mean 0 and standard deviation 1 before returning.

Value

specified sinusoidal curve as a vector

See Also

Other Simulate.time.series: `SimulateAR1`, `SimulateCorrelated`, `SimulateDummy`