Package 'SynthETIC'

August 8, 2020

Title Synthetic Experience Tracking Insurance Claims

Version 0.1.0

Author Benjamin Avanzi [aut], Greg Taylor [aut], Melantha Wang [aut, cre], Bernard Wong [aut]

Maintainer Melantha Wang <wang.melantha@gmail.com>

Imports stats, ggplot2, magrittr, rlang

Description Creation of an individual claims simulator which generates various features of non-life insurance claims. An initial set of test parameters, designed to mirror the experience of an Auto Liability portfolio, were set up and applied by default to generate a realistic test data set of individual claims (see vignette). The simulated data set then allows practitioners to back-test the validity of various reserving models and to prove and/or disprove certain actuarial assumptions made in claims modelling. The distributional assumptions used to generate this data set can be easily modified by users to match their experiences.

License GPL-3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, dplyr, RColorBrewer

VignetteBuilder knitr

Depends R (>= 2.10)

NeedsCompilation no

R topics documented:

`claims` *Construction of a* `claims` *Object*

Description

Contructs a `claims` object which stores all the simulated quantities.

Usage

```
claims(
  frequency_vector,
  occurrence_list,
  claim_size_list,
  notification_list,
  settlement_list,
  no_payments_list,
  payment_size_list,
  payment_delay_list,
  payment_time_list,
  payment_inflated_list
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`occurrence_list`	list of claim occurrence times.
`claim_size_list`	list of claim sizes.
`notification_list`	list of notification delays.
`settlement_list`	list of settlement delays.

`no_payments_list`	list of number of partial payments.
`payment_size_list`	(compound) list of payment size pattern (without inflation).
`payment_delay_list`	(compound) list of inter partial delays.
`payment_time_list`	(compound) list of payment times on a **continuous** time scale.
`payment_inflated_list`	(compound) list of payment size pattern (after inflation).

Value

Returns a `claims` object (mainly a reformat of the arguments as a list object), with the 10 components as listed above.

`claim_closure` *Claim Closure*

Description

Simulates and returns the closure/settlement delays of each of the claims occurring in each of the periods, assuming a Weibull distribution.

Usage

```
claim_closure(frequency_vector, claim_size_list, mean_function, cv_function)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`claim_size_list`	list of claim sizes.
`mean_function`	target mean of the Weibull distribution as a function of `claim_size` and `occurrence_period` (see Details for default mean function).
`cv_function`	target CoV of the Weibull distribution as a function of `claim_size` and `occurrence_period` (see Details for default CoV function).

Details

Claim settlement delay represents the delay from claim notification to closure. The epoch of closure is the sum of occurrence time, notification delay and settlement delay.

It is assumed that the settlement delay follows a Weibull distribution with parameters possibly dependent on claim size and occurrence period.

Recall that `ref_claim` is a packagewise-global variable that user is required to define by `set_parameters`.

By default, we assume that mean settlement delay (in quarters, but automatically converted to the relevant `time_unit` as defined in `set_parameters`) is porportional to

$$min(25, max(1, 6 + 4log[claim_size/(0.10 * ref_claim)]))$$

up to a scaling factor "a", which is dependent on `occurrence_perid`. Specifically,

$$a = min(0.85, 0.65 + 0.02 * (occurrence_period - 21))$$

if `claim_size` < (0.10 * ref_claim) and `occurrence_period` $\geq$ 21, and

$$a = max(0.85, 1 - 0.0075 * occurrence_period)$$

otherwise. The CoV of the settlement delay is constant at 60%, independent of the size and occurrence period of the claim.

Note that this function can create out-of-bound settlement dates. In these cases, the simulated epoch of occurrence of the transaction is maintained throughout the simulation of details of the claim concerned. Adjustments will only be made for the tabulation of results in `claim_output` and payment inflation.

Value

A list of settlement delays such that the ith component of the list gives the settlement delays for all claims that occurred in period i.

Examples

```
n_vector <- c(90, 79, 102, 78, 86, 88, 116, 84, 93, 104)

# Try constant mean/CoV function (i.e. independent of claim size)
setldel_mean <- function(claim_size, occurrence_period) {10}
setldel_cv <- function(claim_size, occurrence_period) {0.70}

setldel <- claim_closure(n_vector, claim_size(n_vector),
                         setldel_mean, setldel_cv)
setldel[[1]] # show settlement delay of claims originating from period 1
```

`claim_frequency` *Claim Frequency*

Description

Returns the number of insurance claims occurring in each of the periods.

Usage

```
claim_frequency(I = 40, E = 12000, freq = 0.03, cdf, range, ...)
```

Arguments

`I`	number of claims development periods considered.
`E`	**effective annual** exposure associated with each period (vector).
`freq`	expected frequency per unit exposure for each period (vector).
`cdf`	optional cumulative distribution function to be sampled from.
`range`	support of the custom `cdf`, with default (-1e200, 1e200).
`...`	other arguments/parameters to be passed onto `cdf`.

Details

Unless otherwise specified, `claim_frequency()` assumes the claim frequency follows a Poisson distribution with mean equal to the product of exposure `E` associated with period i and expected claim frequency `freq` per unit exposure for that period.

If no arguments are provided, by default `claim_frequency()` assumes a total of 40 development periods, constant exposure rate at 12000 per year and constant frequency at 0.03 per unit of exposure.

Pre-defined distribution functions such as `ppois` are supported.

Examples

```
no_period <- 40
exposure <- c(rep(12000, no_period))
exp_freq <- c(rep(0.03, no_period))
# returns the same result as claim_frequency()
claim_frequency(I = no_period, E = exposure, freq = exp_freq)

# some custom pre-defined distribution function
claim_frequency(I = 10, cdf = ppois, range = c(0, 1000), lambda = 80)
```

`claim_notification` *Claim Notification*

Description

Simulates and returns the notification/reporting delays of each of the claims occurring in each of the periods, assuming a Weibull distribution.

Usage

```
claim_notification(
  frequency_vector,
  claim_size_list,
  mean_function,
  cv_function
)
```

Arguments

`frequency_vector`
: a vector of claim frequencies for all the periods.

`claim_size_list`
: list of claim sizes.

`mean_function`
: target mean of the Weibull distribution as a function of `claim_size` and `occurrence_period` (see Details for default mean function).

`cv_function`
: target CoV (Coefficient of Variation) of the Weibull distribution as a function of `claim_size` and `occurrence_period` (see Details for default CoV function).

Details

Claim notification delay represents the delay from claim occurrence to claim reporting. It is assumed that the notification delay follows a Weibull distribution with parameters possibly dependent on claim size and occurrence period.

By default, we assume that mean notification delay (in quarters) is given by

$$min(3, max(1, 2 - [log(claim_size/(0.50 * ref_claim))]/3))$$

automatically converted to the relevant `time_unit` defined by user at the top of their script through `set_parameters`. Note that the `ref_claim` in the equation is another package-wise global variable that the user needs to define through `set_parameters` as it determines the monetary scale of the simulator. The CoV (Coefficient of Variation) of the notification delay is assumed to be constant at 70%, independent of the size and occurrence period of the claim.

Value

A list of notification delays such that the ith component of the list gives the notification delays for all claims that occurred in period i.

Examples

```
n_vector <- c(90, 79, 102, 78, 86, 88, 116, 84, 93, 104)

# Try constant mean/CoV function (i.e. independent of claim size)
notidel_mean <- function(claim_size, occurrence_period) {2}
notidel_cv <- function(claim_size, occurrence_period) {0.70}

notidel <- claim_notification(n_vector, claim_size(n_vector),
                              notidel_mean, notidel_cv)
notidel[[1]] # show notification for claims originating from period 1
```

`claim_occurrence` *Claim Occurrence Times*

Description

Returns the occurrence times of each of the claims occurring in each of the periods, assuming the occurrence time of any claim in period i is uniformly distributed between times $i - 1$ and i.

Usage

```
claim_occurrence(frequency_vector)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.

Value

A list of occurrence times such that the ith component of the list gives the claim occurrence time for all claims that occurred in period i.

Examples

```
n_vector <- c(90, 79, 102, 78, 86, 88, 116, 84, 93, 104)
# occurrence time for all claims originating from period 1
claim_occurrence(n_vector)[[1]]
```

`claim_output` *Loss Reserving Output*

Description

Outputs the full square of claim payments by occurrence period and development period. The upper left triangle represents the past, and the lower right triangle the unseen future.

Users can modify the aggregate level by providing an `aggregate_level` argument to the function. For example, setting `aggregate_level = 4` when working with calendar *quarters* produces a payment square by occurrence and development *year*.

Usage

```
claim_output(
  frequency_vector,
  payment_time_list,
  payment_size_list,
  aggregate_level = 1,
  incremental = TRUE
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`payment_time_list`	(compound) list of payment times (both the continous time scale and the discrete period versions work).
`payment_size_list`	(compound) list of payment size pattern (can be either with or without inflation).

`aggregate_level`	number of periods to be aggregated together; must be a divisor of the total number of periods under consideration (default 1).
`incremental`	logical; if true returns the incremental payment square, else returns the cumulative payment square.

Details

Remark on out-of-bound payment times: This function includes adjustment for out-of-bound transaction dates, by forcing payments that were projected to fall out of the maximum development period to be paid at the exact end of the maximum development period allowed. For example, if we consider 40 periods of development and a claim incurred in the interval (20, 21] was projected to have a payment at time 62.498210, then we would treat such a payment as if it occurred at time 60 for the purpose of tabulation.

Value

An array of claims payments.

Examples

```
attach(test_claims_object)
# a square of cumulative claims payments by accident and development quarters
CL <- claim_output(frequency_vector, payment_time_list, payment_size_list,
                   aggregate_level = 1, incremental = FALSE)
detach(test_claims_object)
```

`claim_payment_delay` *Inter-Partial Delays*

Description

Simulates and returns the inter-partial delays (i.e. the delay of one partial payment relative to the previous) of each payment for each of the claims occurring in each of the periods.

Usage

```
claim_payment_delay(
  frequency_vector,
  claim_size_list,
  no_payments_list,
  settlement_list,
  settlement_mean_function,
  simulate_delay_function
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`claim_size_list`	list of claim sizes.

`no_payments_list`
: list of number of partial payments.

`settlement_list`
: list of settlement delays.

`settlement_mean_function`
: target mean of the settlement delay as a function of `claim_size` and `occurrence_period` (see the documentation of `claim_closure` for more details).

`simulate_delay_function`
: a function that generates the payment delay pattern of a particular claim (as a vector of size = `no_pmt`), taking as input `no_pmt`, `claim_size`, `setldel`, `occurrence_period`, and `setldel_mean_function` (see Details for the default simulation algorithm).

Details

Returns a compound list structure such that the jth component of the ith sub-list gives the payment delay pattern (as a vector) for the jth claim of occurrence period i.

The default `simulate_delay_function` is split into 2 cases.

Case 1: claims with at least 4 partial payments. The simulation takes 2 steps.
First we sample the last payment delay from a Weibull distribution with mean = 1 quarter (automatically converted to the relevant `time_unit`, a global variable that the user is required to define at the top of their code) and CoV = 20%. Then we sample the remaining payment delays from a second Weibull distribution with CoV at 35% and

$$mean = settlement_mean_function(claim_size, occurrence_period)/no_pmt$$

where `settlement_mean_function()` is the function that we used in `claim_closure` to generate the settlement delays.

Case 2: claims with less than 4 partial payments. Proceed as in Case 1 but without separating out the simulation of the last payment delay (i.e. ignore step 1).

Examples

```
# set up
n_vector <- claim_frequency(I = 10)
claim_sizes <- claim_size(n_vector)
no_payments <- claim_payment_no(n_vector, claim_sizes)
setldel <- claim_closure(n_vector, claim_sizes)

# with default setting
pmtdel <- claim_payment_delay(n_vector, claim_sizes, no_payments, setldel)
pmtdel[[1]][[1]] # payment delays for claim 1 of occurrence period 1
```

`claim_payment_inflation`
Size of Partial Payments (With Inflation)

Description

Converts the (compound) list of constant-dollar-value payment sizes to a (compound) list of inflated payment sizes by applying inflation rates on a continuous time scale.

Compare with `claim_payment_size()` which generates the constant dollar amount of partial payment sizes. Note that the constant dollar values are as of time 0.

Usage

```
claim_payment_inflation(
  frequency_vector,
  payment_size_list,
  payment_time_list,
  occurrence_list,
  claim_size_list,
  base_inflation_vector,
  si_occurrence_function,
  si_payment_funtion
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the occurrence periods.
`payment_size_list`	(compound) list of payment size pattern (without inflation).
`payment_time_list`	(compound) list of payment times on a **continuous** time scale.
`occurrence_list`	(compound) list of occurrence times on a **continuous** time scale.
`claim_size_list`	list of claim sizes.
`base_inflation_vector`	vector showing **quarterly** base inflation rates (quarterly effective) for all the periods under consideration (default at nil base inflation).
`si_occurrence_function`	function of `occurrence_time` and `claim_size` that outputs the superimposed inflation index with respect to claim occurrence time (see Details for the default inflation function).
`si_payment_funtion`	function of `payment_time` and `claim_size` that outputs the superimposed inflation index with respect to payment time (see Details for the default inflation function).

Details

Returns a compound list structure such that the jth component of the ith sub-list gives the **inflated** payment pattern (as a vector) for the jth claim of occurrence period i.

By default we assume

- Nil base inflation.

- No superimposed inflation by (continuous) occurrence time for the first 20 quarters (converted to the relevant `time_unit`); beyond 20 quarters, the inflation index is given by

  $$1 - 0.4max(0, 1 - claim_size/(0.25 * ref_claim))$$

  where `ref_claim` is a package-wise global variable that user is required to define at the top of their code using `set_parameters`. The interpretation is that, due to some external change to the insurance scheme at the end of occurrence quarter 20, the smallest claims will reduce by up to 40% in size. This change will not impact claims exceeding `0.25*ref_claim` in size. The reduction varies linearly between these claim sizes.
- Superimposed inflation by (continuous) payment time operates at a period rate of

  $$\gamma * max(0, 1 - claim_size/ref_claim)$$

  where γ is equivalent to a 30% p.a. inflation rate (converted to the relevant `time_unit`). The interpretation is that, for claims of small size the payment time superimposed inflation tends to be very high (30% p.a.); whereas for claims exceeding `ref_claim` in dollar values as of $t = 0$, the payment time superimposed inflation is nil. The rate of inflation varies linearly between claim sizes of zero and `ref_claim`.

Remark on continuous inflation: We note that `SynthETIC` works with exact transaction times, so time has been measured continuously throughout the program. This allows us to apply inflation on a continous time scale too. For example, we asked the users to provide base inflation as a vector of quarterly base inflation rates, quarterly effective for all the periods under consideration. This data is generally available online (e.g. the Australian quarterly inflation is available on RBA's website - see link). We then interpolate the quarterly inflation rates to compute the addition of inflation by exact times. In the case of above, if we observed quarterly inflation rates of 0.6%, 0.5%, 0.7% and 0.3% for one particular year, then the base inflation applied to a payment at time $t = 1.82$ quarters will be $1.006 * 1.005^{0.82}$.

Remark on out-of-bound payment times: This function includes adjustment for out-of-bound transaction dates, by forcing payments that were projected to fall out of the maximum development period to be paid at the exact end of the maximum development period allowed. For example, if we consider 40 periods of development and a claim incurred in the interval (20, 21] was projected to have a payment at time 62.498210, then we would treat such a payment as if it occurred at time 60 for the purpose of inflation.

Examples

```
# remove SI occurrence and SI payment
SI_occurrence <- function(occurrence_time, claim_size) {1}
SI_payment <- function(payment_time, claim_size) {1}
# base inflation constant at 0.02 p.a. effective
base_inflation_vector <- rep((1 + 0.02)^(1/4) - 1, times = 40)
attach(test_claims_object)
payment_inflated_list <- claim_payment_inflation(
  frequency_vector, payment_size_list, payment_time_list,
  occurrence_list, claim_size_list, base_inflation_vector,
  SI_occurrence, SI_payment
)
detach(test_claims_object) # undo the attach
# inflated payments for claim 1 of occurrence period 1
payment_inflated_list[[1]][[1]]
```

`claim_payment_no` *Number of Partial Payments*

Description

Simulates and returns the number of partial payments required to settle each of the claims occurring in each of the periods.

Usage

```
claim_payment_no(
  frequency_vector,
  claim_size_list,
  simulate_no_pmt_function,
  claim_size_benchmark_1 = 0.0375 * .pkgenv$ref_claim,
  claim_size_benchmark_2 = 0.075 * .pkgenv$ref_claim
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`claim_size_list`	list of claim sizes.
`simulate_no_pmt_function`	a function that generates the number of partial payments associated with a particular claim, conditional on `claim_size` (see Details for the default simulation function).
`claim_size_benchmark_1`	a value below which claims are assumed to be settled with 1 or 2 payments, unless an alternative `simulate_no_pmt_function` is specified (default 0.0375 * `ref_claim`).
`claim_size_benchmark_2`	a second criterion below which claims are assumed to be settled with 2 or 3 payments, unless an alternative `simulate_no_pmt_function` is specified (default 0.075 * `ref_claim`).

Details

Returns a list structure such that the ith component of the list gives the number of partial payments required to settle each of the claims that occurred in period i. It is assumed that at least one payment is required i.e. no claims are settled without any single cash payment.

Let M represent the number of partial payments associated with a particular claim. The default `simulate_no_pmt_function` is set up such that if `claim_size` $\leq$ `claim_size_benchmark_1`,

$$Pr(M = 1) = Pr(M = 2) = 1/2;$$

if `claim_size_benchmark_1` < `claim_size` $\leq$ `claim_size_benchmark_2`,

$$Pr(M = 2) = 1/3, Pr(M = 3) = 2/3;$$

if `claim_size` > `claim_size_benchmark_2` then M is geometric with minimum 4 and mean

$$min(8, 4 + log(claim_size/claim_size_benchmark_2)).$$

Examples

```
n_vector <- claim_frequency(I = 10)
# with default simulate_no_pmt_function
no_payments <- claim_payment_no(n_vector, claim_size(n_vector))
no_payments[[1]] # number of payments for claims incurred in period 1
```

`claim_payment_size` *Size of Partial Payments (Without Inflation)*

Description

Simulates and returns the constant dollar amount of each partial payment (i.e.**without inflation**) for each of the claims occurring in each of the periods.

Usage

```
claim_payment_size(
  frequency_vector,
  claim_size_list,
  no_payments_list,
  simulate_amt_pmt_function
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`claim_size_list`	list of claim sizes.
`no_payments_list`	list of number of partial payments.
`simulate_amt_pmt_function`	a function that generates and returns the payment pattern of a particular claim (as a vector of size = `no_pmt`), conditional on `no_pmt` and `claim_size` (see Details for the default simulation function).

Details

Returns a compound list structure such that the jth component of the ith sub-list gives the payment pattern (as a vector) for the jth claim of occurrence period i.

The default `simulate_amt_pmt_function` is set up in three steps. First we sample the *complement* of the porportion of total claim size represented by the last two payments, from a Beta distribution with mean

$$1 - min(0.95, 0.75 + 0.04log[claim_size/(0.10 * ref_claim)])$$

where `ref_claim` is a package-wise global variable that we ask the user to define at the top of their code using `set_parameters`. CoV is assumed constant at 20%.

Next we simulate the porportion of last_two_pmts paid in the second last payment (*settlement of the claim*) from a Beta distribution with mean = 0.90 and CoV = 3%.

Lastly we sample the remaining payment proportions from a Beta distribution with mean

$$(1 - last_two_payments)/(no_pmt - 2)$$

and CoV = 10%, which is followed by a normalisation such that the proportions add up to 1.

In the cases where there are only 2 or 3 partial payments, proceed as if there were 4 or 5 payments respectively with last_two_payments = 0. The trivial case is when the claim is settled with a single payment, which must be of the same amount as the total claim size.

Explanation

Why did we set up a payment pattern as above?

The payment pattern is set up to reflect the typical pattern of a claim from an Auto liability line of business, which usually consists of:

1. (possibly) some small payments such as police reports, medical consultations and reports;
2. some more substantial payments such as hospitalisation, specialist medical procedures, equipment (e.g. prosthetics);
3. a final settlement with the claimant (usually the second last payment);
4. a smaller final payment, usually covering legal costs.

Claims in a number of other lines of business exhibit a similar structure, albeit with possible differences in the types of payment made.

Examples

```
# set up
n_vector <- claim_frequency(I = 10)
claim_sizes <- claim_size(n_vector)
no_payments <- claim_payment_no(n_vector, claim_sizes)

# with default simulate_amt_pmt_function
payments <- claim_payment_size(n_vector, claim_sizes, no_payments)
# partial payment sizes for claim 1 of occurrence period 1
payments[[1]][[1]]

# with some custom simulate_amt_pmt_function
# simplest case: (stochastically) equal amounts
my_func <- function(no_pmt, claim_size) {
  prop <- runif(no_pmt)
  prop <- prop/sum(prop)
  claim_size * prop
}
mypayments <- claim_payment_size(n_vector, claim_sizes, no_payments, my_func)
# partial payment sizes for claim 1 of occurrence period 1
mypayments[[1]][[1]]
```

`claim_payment_time` *Partial Payment Times (in Continuous Time Scale)*

Description

Converts the list of inter-partial delays to a list of payment times in continuous time scale. Set `discrete = TRUE` to get the payment times in calendar periods.

Usage

```
claim_payment_time(
  frequency_vector,
  occurrence_list,
  notification_list,
  payment_delay_list,
  discrete = FALSE
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`occurrence_list`	list of claim occurrence times.
`notification_list`	list of notification delays.
`payment_delay_list`	(compound) list of inter partial delays.
`discrete`	logical; if TRUE returns integer-valued payment times (default FALSE).

Details

Returns a compound list structure such that the jth component of the ith sub-list gives the payment time pattern (as a vector) for the jth claim of occurrence period i.

Note that, as in the case of `claim_closure`, this function can result in out-of-bound payment dates (i.e. payment times beyond the maximum number of development periods under consideration). In these cases, we retain the original simulated values for the simulation of other quantities, but we will make adjustments for such claims in the tabulation of results in `claim_output` and the payment inflation function `claim_payment_inflation`.

`claim_size` *Claim Size*

Description

Simulates and returns the size of each of the claims occurring in each of the periods, given its cumulative distribution function.

Note that `claim_size()` aims to model the claim sizes **without inflation**.

Usage

```
claim_size(frequency_vector, S_cdf, range = c(0, 1e+24), ...)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`S_cdf`	(optional) cumulative distribution function to be sampled from.
`range`	the possible set of values of claim sizes, with default (0, 1e24).
`...`	other arguments/parameters to be passed onto `S_cdf`.

Details

By default `claim_size()` assumes a left truncated power normal distribution: $S^0.2\, Normal(9.5, sd = 3)$, left truncated at 30. The truncation is done via resampling for rejected values.

Users can opt to use predefined distributions if desired. See Examples.

Value

A list of claim sizes such that the ith component of the list gives the sizes for all claims that occurred in period i.

Examples

```
n_vector <- c(90, 79, 102, 78, 86, 88, 116, 84, 93, 104)
claim_size(n_vector)[[1]] # gives the sizes for all
                          # all claims incurred in period 1

# with some custom pre-defined distribution function
claim_size(n_vector, stats::pweibull, shape = 4, scale = 100000)[[1]]
```

`cv` *Coefficient of Variation*

Description

Returns the observed coefficient of variation (CoV) of a given sample `x`.

If `na.rm` is true then missing values are removed before computation proceeds, as in the case of the `mean()` function.

Usage

```
cv(x, na.rm = TRUE)
```

Arguments

`x`	a numeric vector.
`na.rm`	a logical value indicating whether NA values should be stripped before the computation proceeds.

Details

The coefficient of variation is defined as is defined as the ratio of the standard deviation to the mean. It shows the extent of variability in relation to the mean of the population.

`cv()` estimates the CoV of a given sample by computing the ratio of the sample standard deviation (see `stats::sd`) to the sample mean.

Examples

```
cv(1:10)
```

---

generate_claim_dataset

Generate a Claims Dataset

Description

Generates a dataset of claims records that takes the same structure as `test_claim_dataset` included in this package, with each row representing a unique claim.

Usage

```
generate_claim_dataset(
  frequency_vector,
  occurrence_list,
  claim_size_list,
  notification_list,
  settlement_list,
  no_payments_list
)
```

Arguments

`frequency_vector`	a vector of claim frequencies for all the periods.
`occurrence_list`	list of claim occurrence times.
`claim_size_list`	list of claim sizes.
`notification_list`	list of notification delays.
`settlement_list`	list of settlement delays.
`no_payments_list`	list of number of partial payments.

Value

A dataframe that takes the same structure as `test_claim_dataset`.

See Also

test_claim_dataset

Examples

```
# demo only, in practice might generate claim dataset before simulating
# the partial payments
# this code generates the built-in test_claim_dataset
attach(test_claims_object)
claim_dataset <- generate_claim_dataset(
  frequency_vector, occurrence_list, claim_size_list, notification_list,
  settlement_list, no_payments_list
)
detach(test_claims_object)
```

---

generate_transaction_dataset

Generate a Transactions Dataset

Description

Generates a dataset of partial payment records that takes the same structure as `test_transaction_dataset` included in this package, with each row representing a unique payment.

Usage

```
generate_transaction_dataset(claims, adjust = FALSE)
```

Arguments

`claims`	an `claims` object containing all the simulated quantities, see `claims`.
`adjust`	if TRUE then the payment times will be forced to match with the maximum development period under consideration; default FALSE (which will produce out-of-bound payment times).

Value

A dataframe that takes the same structure as `test_transaction_dataset`.

See Also

test_transaction_dataset

Examples

```
# this generates the built-in test_transaction_dataset
transact_data <- generate_transaction_dataset(test_claims_object)
```

`get_Beta_parameters` *Estimating Beta Parameters*

Description

Returns the Beta parameters given the mean and the CoV of the target Beta distribution.

Usage

```
get_Beta_parameters(target_mean, target_cv)
```

Arguments

`target_mean`	mean of the target Beta distribution (between 0 and 1).
`target_cv`	CoV of the target Beta distribution.

Examples

```
get_Beta_parameters(target_mean = 0.5, target_cv = 0.20)
get_Beta_parameters(target_mean = 0.5,
                    target_cv = c(0.10, 0.20, 0.30))
```

`get_Weibull_parameters` *Estimating Weibull Parameters*

Description

Returns the Weibull shape and scale parameters given the mean and the CoV of the target Weibull distribution.

Usage

```
get_Weibull_parameters(target_mean, target_cv)
```

Arguments

`target_mean`	mean of the target Weibull distribution.
`target_cv`	CoV of the target Weibull distribution.

Examples

```
get_Weibull_parameters(target_mean = 100000, target_cv = 0.60)
get_Weibull_parameters(target_mean = c(100000, 200000, 300000),
                       target_cv = 0.60)
```

`plot.claims` *Plot of Cumulative Claims Payments (Incurred Pattern)*

Description

Generates a plot of cumulative claims paid (as a percentage of total amount incurred) as a function of development time for each occurrence period.

Usage

```
## S3 method for class 'claims'
plot(x, by_year = FALSE, inflated = TRUE, adjust = TRUE, ...)
```

Arguments

`x`	an object of class `claims` containing all the simulated quantities.
`by_year`	if `TRUE` returns a plot by occurrence year; otherwise returns a plot by occurrence period (default).
`inflated`	if `TRUE` shows a plot of payment pattern after inflation; otherwise shows a plot of discounted payment pattern.
`adjust`	if `TRUE` then the payment times will be forced to match with the maximum development period under consideration, otherwise the plot will see claims beyond the maximum development period; default `TRUE`.
`...`	other graphical parameters.

See Also

`claims`

Examples

```
plot(test_claims_object)
plot(test_claims_object, adjust = FALSE)
```

`return_parameters` *Get Current Parameters*

Description

Returns the current values of `ref_claim` and `time_unit` parameters, two packagewise-global variables used by all simulation functions within this package.

Usage

```
return_parameters(print = FALSE)
```

Arguments

`print`	logical; if TRUE prints a message.

Details

Returns and (optionally) prints the current values of `ref_claim` and `time_unit` parameters.

See Also

`set_parameters`

Examples

```
cur <- return_parameters()
cur
set_parameters(ref_claim = 200000, time_unit = 1/12) # monthly reserving
return_parameters(print = FALSE)
```

`set_parameters` *Set Packagewise Global Parameters for the Claims Simulator*

Description

Sets `ref_claim` and `time_unit` parameters for all the simulation functions within the `SynthETIC` package.

Usage

```
set_parameters(ref_claim = 2e+05, time_unit = 1/4)
```

Arguments

`ref_claim`	a reference value for the claim sizes (default 200000).
`time_unit`	time unit to work with, given as a fraction of a year; default calendar quarters (1/4).

Details

Those variables will be available to multiple functions in this package, but are kept local to the package environment (i.e. not accessible from the global environment). To extract the current values of the variables, use `return_parameters`.

We introduce the reference value `ref_claim` partly as a measure of the monetary unit and/or overall claims experience. The default distributional assumptions were set up with an Australian Auto Liability portfolio in mind. `ref_claim` then allows users to easily simulate a synthetic portfolio with similar claim pattern but in a different currency, for example. We also remark that users can alternatively choose to interpret `ref_claim` as a monetary unit. For example, one can set `ref_claim <-1000` and think of all amounts in terms of $1,000. However, in this case the default simulation functions will not work and users will need to supply their own set of functions and set the values as multiples of `ref_claim` rather than fractions as in the default setting.

We also require the user to input a `time_unit` (which should be given as a fraction of year), so that the default input parameters apply to contexts where the time units are no longer in quarters. In the default setting we have a `time_unit` of 1/4 i.e. we work with calendar quarters.

The default input parameters will update automatically with the choice of the two variables `ref_claim` and `time_unit`, which ensures that the simulator produce sensible results in contexts other than the default setting. We remark that both `ref_claim` and `time_unit` only affect the default simulation functions, and users can also choose to set up their own modelling assumptions for any of the modules to match their experiences even better. In the latter case, it is the responsibility of the user to ensure that their input parameters are compatible with their time units and claims experience. For example, if the time units are quarters, then claim occurrence rates must be quarterly.

See Also

See the vignette for this package for a full list of functions impacted by those two variables.

Examples

```
set_parameters(ref_claim = 200000, time_unit = 1/12) # monthly reserving
```

`simulate_cdf` *Inverse Tranform Sampling*

Description

Generates sample numbers at random from any probability distribution given its cumulative distribution function. Pre-defined distribution functions such as `pnorm` are supported.

See here for the algorithm.

Usage

```
simulate_cdf(n, cdf, range = c(-1e+200, 1e+200), ...)
```

Arguments

`n`	number of observations.
`cdf`	cumulative distribution function to be sampled from.
`range`	support of the given `cdf`.
`...`	other arguments/parameters to be passed onto `cdf`.

Examples

```
simulate_cdf(10, pnorm)
simulate_cdf(10, pbeta, shape1 = 2, shape2 = 2)
```

`test_claims_object` *Claims Data in List Format*

Description

A list containing a sample output from each of the simulation modules, in sequential order of the running of the modules. This is the test data generated when run with seed `20200131` at the top of the code.

Usage

```
test_claims_object
```

Format

A `claims` object with 10 components:

frequency_vector vector; number of claims for each occurrence period, see also `claim_frequency()`.

occurrence_list list; claim occurrence times for all claims that occurred in each of the period, see also `claim_occurrence()`.

claim_size_list list; claim sizes for all claims that occurred in each of the period, see also `claim_size()`.

notification_list list; notification delays for all claims that occurred in each of the period, see also `claim_notification()`.

settlement_list list; settlement delays for all claims that occurred in each of the period, see also `claim_closure()`.

no_payments_list list; number of partial payments for all claims that occurred in each of the period, see also `claim_payment_no()`.

payment_size_list (compound) list; sizes of partial payments (without inflation) for all claims that occurred in each of the period, see also `claim_payment_size()`.

payment_delay_list (compound) list; inter partial delays for all claims that occurred in each of the period, see also `claim_payment_delay()`.

payment_time_list (compound) list; payment times (on a continuous time scale) for all claims that occurred in each of the period, see also `claim_payment_time()`.

payment_inflated_list (compound) list; sizes of partial payments (with inflation) for all claims that occurred in each of the period, see also `claim_payment_inflation()`.

See Also

1. Claim occurrence: `claim_frequency`, `claim_occurrence`
2. Claim size: `claim_size`
3. Claim notification: `claim_notification`
4. Claim closure: `claim_closure`
5. Claim payment count: `claim_payment_no`
6. Claim payment size (without inflation): `claim_payment_size`
7. Claim payment time: `claim_payment_delay`, `claim_payment_time`
8. Claim inflation: `claim_payment_inflation`

Examples

```
test_claims_object$frequency_vector
```

---

test_claim_dataset *Claims Dataset*

Description

A dataset of 3,624 rows containing individual claims features.

Usage

```
test_claim_dataset
```

Format

A data frame with 3,624 rows and 7 variables:

claim_no claim number, which uniquely characterises each claim.

occurrence_period integer; period of ocurrence of the claim.

occurrence_time double; time of occurrence of the claim.

claim_size size of the claim (in constant dollar values).

notidel notification delay of the claim, i.e. time from occurrence to notification.

setldel settlement delay of the claim, i.e. time from notification to settlement.

no_payment number of partial payments required for the claim.

Examples

```
# see a distribution of payment counts
table(test_claim_dataset$no_payment)
```

---

test_transaction_dataset *Transactions Dataset*

Description

A dataset of 18,983 records of partial payments associated with the 3,624 claims in `test_claim_dataset`.

Usage

```
test_transaction_dataset
```

Format

A data frame with 18,983 rows and 12 variables:

claim_no claim number, which uniquely characterises each claim.

pmt_no payment number, identification number of partial payments in respect of a particular `claim_no`.

occurrence_period integer; period of ocurrence of the claim.

occurrence_time double; time of occurrence of the claim.

claim_size size of the claim (in constant dollar values).

notidel notification delay of the claim, i.e. time from occurrence to notification.

setldel settlement delay of the claim, i.e. time from notification to settlement.

payment_time double; time of payment (on a continuous time scale).

payment_period integer; time of payment (in calendar period).

payment_size size of the payment in constant dollar terms.

payment_inflated actual size of the payment (i.e. with inflation).

payment_delay inter partial delay associated with the payment.

Index