Package ‘SPLICE’

March 5, 2022

Title Synthetic Paid Loss and Incurred Cost Experience (SPLICE) Simulator

Version 1.1.0

Author Benjamin Avanzi [aut],
Greg Taylor [aut],
Melantha Wang [aut, cre],
William Ho [rev]

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

Imports stats, methods, zoo, lifecycle

Description An extension to the individual claim simulator called 'SynthETIC' (on CRAN), to simulate the evolution of case estimates of incurred losses through the lifetime of an insurance claim. The transactional simulation output now comprises key dates, and both claim payments and revisions of estimated incurred losses. An initial set of test parameters, designed to mirror the experience of a real insurance portfolio, were set up and applied by default to generate a realistic test data set of incurred histories (see vignette). However, the distributional assumptions used to generate this data set can be easily modified by users to match their experiences. Reference: Avanzi B, Taylor G, Wang M (2021) ``SPLICE: A Synthetic Paid Loss and Incurred Cost Experience Simulator'' <arXiv:2109.04058>.

License GPL-3

Encoding UTF-8

URL `https://github.com/agi-lab/SPLICE`

BugReports `https://github.com/agi-lab/SPLICE/issues`

LazyData true

RoxygenNote 7.1.2

Suggests knitr, rmarkdown, dplyr, RColorBrewer, actuar, ChainLadder

VignetteBuilder knitr

Depends R (>= 2.10), SynthETIC (>= 1.0.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-05 15:20:05 UTC

R topics documented:

`claim_history` *Development of Case Estimates*

Description

Consolidates payments and incurred revisions and returns a full transactional history of all the individual claims (transaction being either a payment or a case estimate revision).

Usage

```
claim_history(
  claims,
  majRev_list,
  minRev_list,
  k1 = 0.95,
  k2 = 0.95,
  base_inflation_vector = NULL,
  keep_all = FALSE
)
```

Arguments

`claims`	an `claims` object containing all the simulated quantities (other than those related to incurred loss), see `claims`.
`majRev_list`	nested list of major revision histories, see `claim_majRev`.
`minRev_list`	nested list of minor revision histories, see `claim_minRev`.
`k1`	maximum amount of cumulative claims paid as a proportion of total incurred estimate for major revisions; between 0 and 1.
`k2`	maximum amount of cumulative claims paid as a proportion of total incurred estimate for minor revisions; between 0 and 1.
`base_inflation_vector`	vector showing **quarterly** base inflation rates (quarterly effective) for all the periods under consideration (default is nil base inflation), should be consistent with the input inflation vector in `claim_payment_inflation`. If a single number is provided, the function will assume constant quarterly inflation.

keep_all	TRUE to keep the paid, outstanding payments, total incurred estimates just before the revision, FALSE to keep only the estimates right after the revision (_right).

Details

This function works to generate the full history of claims paid and incurred estimates by consolidating all the simulated revision quantities. It should be noted that in this consolidation step, we make the following adjustments:

- Major and minor revisions should not occur simultaneously. In the event that they do (which is only possible at the second last partial payment), the major revision takes precedence, and the minor revision be discarded. This will be reflected in the majRev and minRev components of the output list.
- Estimates of incurred loss are specified to be computed in reverse order, and it is necessary that the total incurred estimate is always strictly greater than the cumulative claims paid (except at the final paymen where equality holds). Hence we introduce k1 and k2 to make sure that the revised incurred estimates satisfy
  $$ky(t) \geq c(t)$$
  where $y(t)$ represents the total incurred estimate at delay t, k is a constant between 0 and 1, and $c(t)$ is the cumulative claims paid up to time t. When the raw simulated revision multipliers violate this requirement, the case estimates of the total incurred or the outstanding claim payments will be increased to make sure this condition always holds, i.e. this adjustment takes precedence over the raw simulated revision multipliers.
- **Inflation adjustment**: One can choose to ignore inflation in the incurred estimates (default), or to make allowance for it.
  - If base_inflation_vector == NULL (default), then all case estimates will be computed in values corresponding to time $t = 0$, i.e. the commencement of the first occurrence period.
  - If base_inflation_vector is provided, then the case estimators will include full superimposed inflation and base inflation only up to the date of the revision, i.e. there is an adjustment for the time elapsed since the immediately preceding revision and **no future base inflation** beyond the date of valuation.
  - If inflation is involved, it should be noted that the case estimator reviews the base inflation situation only in the process of making a revision. When *only* a payment is made, the insurer's system automatically writes down the outstanding liability on the assumption of no change in ultimate incurred amount.

Value

A nested list structure such that the *j*th component of the *i*th sub-list provides a full transactional history of the *j*th claim of occurrence period *i*. The "unit list" (i.e. the smallest, innermost sub-list) contains the following components:

txn_delay	Delays from notification to the transactions (payment or incurred revision).
txn_time	Times of the transactions (from the commencement of the first occurrence period).
txn_type	Types of the transactions, "Ma" for major revision, "Mi" for minor revision, "P" for payment, "PMa" for n
cumpaid_right	Cumulative claim payments immediately after each of the transactions (in the "right" continuous sense).
OCL_right	Case estimate of outstanding claim payments immediately after each of the transactions (in the "right" con

`incurred_right`	Case estimate of incurred loss immediately after each of the transactions (in the "right" continuous sense).
`minRev`	A list containing full history of minor revisions (frequency, time and revision size); `claim_minRev`.
`majRev`	A list containing full history of major revisions (frequency, time and revision size); see `claim_majRev`.

and optionally (by setting `keep_all = TRUE`),

`cumpaid_left`	Cumulative claim payments just before each of the transactions.
`OCL_left`	Case estimate of outstanding claim payments just before each of the transactions.
`incurred_left`	Case estimate of incurred loss just before each of the transactions.

`claim_majRev` *Major Revisions of Incurred Loss*

Description

A suite of functions that works together to simulate, in order, the (1) frequency, (2) time, and (3) size of major revisions of incurred loss, for each of the claims occurring in each of the periods.

Usage

```
claim_majRev_freq(
  claims,
  rfun,
  paramfun,
  frequency_vector = claims$frequency_vector,
  claim_size_list = claims$claim_size_list,
  ...
)

claim_majRev_time(
  claims,
  majRev_list,
  rfun,
  paramfun,
  claim_size_list = claims$claim_size_list,
  settlement_list = claims$settlement_list,
  payment_delay_list = claims$payment_delay_list,
  ...
)

claim_majRev_size(majRev_list, rfun, paramfun, ...)
```

Arguments

`claims`	an `claims` object containing all the simulated quantities (other than those related to incurred loss), see `claims`.
`rfun`	optional alternative random sampling function for: • `claim_majRev_freq`: the number of major revisions; • `claim_majRev_time`: the epochs of major revisions measured from claim notification; • `claim_majRev_size`: the sizes of the major revision multipliers. See Details for default.
`paramfun`	parameters for the random sampling function, as a function of other claim characteristics such as `claim_size`; see Details.
`frequency_vector`	a vector of claim frequencies for all the periods (not required if the `claims` argument is provided); see `claim_frequency`.
`claim_size_list`	list of claim sizes (not required if the `claims` argument is provided); see `claim_size`.
`...`	other arguments/parameters to be passed onto `paramfun`.
`majRev_list`	nested list of major revision histories (with non-empty revision frequencies).
`settlement_list`	list of settlement delays (not required if the `claims` argument is provided); see `claim_closure`.
`payment_delay_list`	(compound) list of inter partial delays (not required if the `claims` argument is provided); see `claim_payment_delay`.

Value

A nested list structure such that the *j*th component of the *i*th sub-list is a list of information on major revisions of the *j*th claim of occurrence period *i*. The "unit list" (i.e. the smallest, innermost sub-list) contains the following components:

`majRev_freq`	Number of major revisions of incurred loss [`claim_majRev_freq()`].
`majRev_time`	Time of major revisions (from claim notification) [`claim_majRev_time()`].
`majRev_factor`	Major revision multiplier of **incurred loss** [`claim_majRev_size()`].
`majRev_atP`	An indicator, `1` if the last major revision occurs at the time of the last major payment (i.e. second last paym

Details - `claim_majRev_freq` (Frequency)

Let K represent the number of major revisions associated with a particular claim. The notification of a claim is considered as a major revision, so all claims have at least 1 major revision ($K \geq 1$).

The default `majRev_freq_function` specifies that no additional major revisions will occur for claims of size smaller than or equal to `claim_size_benchmark` (0.075 * `ref_claim` by default). For claims above this threshold,

$$Pr(K = 2) \quad = 0.1 + 0.3 min(1, (claim_size - 0.075 * ref_claim)/0.925 * ref_claim)$$

$$\begin{aligned} Pr(K=3) &= 0.5 min(1, max(0, claim_size - 0.25 * ref_claim)/(0.75 * ref_claim)) \\ Pr(K=1) &= 1 - Pr(K=2) - Pr(K=3) \end{aligned}$$

where `ref_claim` is a package-wise global variable that user should define by `set_parameters` (if moving away from the default).

The idea is that major revisions are more likely for larger claims, and do not occur at all for the smallest claims. Note also that by default a claim may experience **up to a maximum of 2 major revisions** in addition to the one at claim notification. This is taken as an assumption in the default setting of `claim_majRev_size()`. If user decides to modify this assumption, they will need to take care of the part on the major revision size as well.

Details - `claim_majRev_time` (Time)

Let τ_k represent the epoch of the kth major revision (time measured from claim notification), $k = 1, ..., K$. As the notification of a claim is considered a major revision itself, we have $\tau_1 = 0$ for all claims.

The last major revision for a claim may occur at the time of the second last partial payment (which is usually the major settlement payment) with probability

$$0.2 min(1, max(0, (claim_size - ref_claim)/(14 * ref_claim)))$$

where `ref_claim` is a package-wise global variable that user should define by `set_parameters` (if moving away from the default).

Now, if there is a major revision at the time of the second last partial payment, then $\tau_k, k = 2, ..., K-1$ are sampled from a triangular distribution with parameters (see also `ptri`)

- `min` = `time_to_second_last_payment` / 3
- `max` = `time_to_second_last_payment`
- maximum density at `mode` = `time_to_second_last_payment` / 3.

Otherwise (i.e. no major revision at the time of the second last partial payment), $\tau_k, k = 2, ..., K$ are sampled from a triangular distribution with parameters

- `min` = `settlement_delay` / 3
- `max` = `settlement_delay`
- maximum density at `mode` = `settlement_delay` / 3.

Note that when there is a major revision at the time of the second last partial payment, `majRev_atP` (one of the output list components) will be set to be 1.

Details - `claim_majRev_size` (Revision Multiplier)

As mentioned in the frequency section ("Details - `claim_majRev_freq`"), the default function for the major revision multipliers assumes that there are only up to 2 major revisions (in addition to the one at claim notification) for all claims.

By default,

- the first major revision multiplier g_1 is simply 1 (no meaning);

- the second major revision multiplier g_2 is sampled from a lognormal distribution with parameters `meanlog` $= 1.8$ and `sdlog` $= 0.2$;
- the third major revision multiplier g_3 is sampled from a lognormal distribution with parameters `meanlog` $= 1 + 0.07(6 - g_2)$ and `sdlog` $= 0.1$. Note that the third major revision is likely to be smaller than the second.

The revision multipliers are subject to further constraints to ensure that the revised incurred estimate never falls below what has already been paid. This is dicussed in `claim_history`.

The major revision multipliers apply to the incurred loss estimates, that is, a revision multiplier of 2.54 means that at the time of the major revision the incurred loss increases by a factor of 2.54. We highlight this as **in the case of minor revisions, the multipliers will instead apply to outstanding claim amounts**, see `claim_minRev`.

See Also

`claims`

Examples

```
set.seed(1)
test_claims <- SynthETIC::test_claims_object
major <- claim_majRev_freq(test_claims)
major[[1]][[1]] # the "unit list" for the first claim

# update the timing information
major <- claim_majRev_time(test_claims, major)
# observe how this has changed
major[[1]][[1]]

# update the revision multipliers
major <- claim_majRev_size(major)
# again observe how this has changed
major[[1]][[1]]
```

`claim_minRev` *Minor Revisions of Outstanding Claim Payments*

Description

A suite of functions that works together to simulate, in order, the (1) frequency, (2) time, and (3) size of minor revisions of outstanding claim payments, for each of the claims occurring in each of the periods.

We separate the case of minor revisions that occur simultaneously with a partial payment (denoted _atP), and the ones that do not coincide with a payment (denoted _notatP).

Usage

```
claim_minRev_freq(
  claims,
  prob_atP = 0.5,
  rfun_notatP,
  paramfun_notatP,
  frequency_vector = claims$frequency_vector,
  settlement_list = claims$settlement_list,
  no_payments_list = claims$no_payments_list,
  ...
)

claim_minRev_time(
  claims,
  minRev_list,
  rfun_notatP,
  paramfun_notatP,
  settlement_list = claims$settlement_list,
  payment_delay_list = claims$payment_delay_list,
  ...
)

claim_minRev_size(
  claims,
  majRev_list,
  minRev_list,
  rfun,
  paramfun_atP,
  paramfun_notatP,
  settlement_list = claims$settlement_list,
  ...
)
```

Arguments

`claims`	an `claims` object containing all the simulated quantities (other than those related to incurred loss), see `claims`.
`prob_atP`	(optional) probability that a minor revision will occur at the time of a partial payment; default value 0.5.
`rfun_notatP`	optional alternative random sampling function for: • `claim_minRev_freq`: the number of minor revisions that occur at an epoch other than those of partial payments; • `claim_minRev_time`: the epochs of such minor revisions measured from claim notification; • `claim_minRev_size`: the sizes of the minor revision multipliers (common for _atP and _notatP, hence simply termed `rfun` in this case). See Details for default.

`paramfun_notatP`	parameters for the above random sampling function, as a function of other claim characteristics (e.g. `lambda` as a function of `claim_size` for an `rpois` simulation); see Examples.
`frequency_vector`	a vector of claim frequencies for all the periods (not required if the `claims` argument is provided); see `claim_frequency`.
`settlement_list`	list of settlement delays (not required if the `claims` argument is provided); see `claim_closure`.
`no_payments_list`	list of number of partial payments (not required if the `claims` argument is provided); see `claim_payment_no`.
`...`	other arguments/parameters to be passed onto `paramfun`.
`minRev_list`	nested list of minor revision histories (with non-empty revision frequencies).
`payment_delay_list`	(compound) list of inter partial delays (not required if the `claims` argument is provided); see `claim_payment_delay`.
`majRev_list`	nested list of major revision histories (with non-empty revision frequencies).
`rfun`	optional alternative random sampling function for the sizes of the minor revision multipliers (common for _atP and _notatP, hence simply termed `rfun` in this case).
`paramfun_atP`	parameters for `rfun` in `claim_minRev_size()` for minor revisions that occur at the time of a partial payment.

Value

A nested list structure such that the *j*th component of the *i*th sub-list is a list of information on minor revisions of the *j*th claim of occurrence period *i*. The "unit list" (i.e. the smallest, innermost sub-list) contains the following components:

`minRev_atP`	A vector of indicators showing whether there is a minor revision at each partial payment [`claim_m`
`minRev_freq_atP`	Number of minor revisions that occur simultaneously with a partial payment, numerically equals t
`minRev_freq_notatP`	Number of minor revisions that do not occur with a partial payment [`claim_minRev_freq()`].
`minRev_time_atP`	Time of minor revisions that occur simultaneously with a partial payment (time measured from cla
`minRev_time_notatP`	Time of minor revisions that do *not* occur simultaneously with a partial payment (time measured f
`minRev_factor_atP`	Minor revision multipliers of **outstanding claim payments** for revisions at partial payments [`clai`
`minRev_factor_notatP`	Minor revision multipliers of **outstanding claim payments** for revisions at any other times [`clair`

Details - `claim_minRev_freq` (Frequency)

Minor revisions may occur simultaneously with a partial payment, or at any other time.

For the former case, we sample the occurrence of minor revisions as Bernoulli random variables with default probability parameter `prob_atP` $= 1/2$.

For the latter case, by default we sample the number of (non payment simultaneous) minor revisions from a geometric distribution with mean = $min(3, setldel/4)$.

One can modify the above sampling distributions by plugging in their own `prob_atP` parameter and `rfun_notatP` function, where the former dictates the probability of incurring a minor revision at the time of a payment, and the latter simulates and returns the number of minor revisions at any other points in time, with possible dependence on the settlement delay of the claim and/or other claim characteristics.

Details - `claim_minRev_time` **(Time)**

For minor revisions that occur simultaneously with a partial payment, the revision times simply coincide with the epochs of the relevant partial payments.

For minor revisions that occur at a different time, by default the revision times are sampled from a uniform distribution with parameters `min` $= settlement_delay/6$ and `max` $= settlement_delay$.

One can modify the above sampling distribution by plugging in their own `rfun_notatP` and `paramfun_notatP` in `claim_minRev_time()`, which together simulate the epochs of minor revisions that do not coincide with a payment, with possible dependence on the settlement delay of the claim and/or other claim characteristics (see Examples).

Details - `claim_minRev_size` **(Revision Multiplier)**

The sampling distribution for minor revision multipliers is the same for both revisions that occur with and without a partial payment. In the default setting, we incorporate sampling dependence on the delay from notification to settlement (`setldel`), the delay from notification to the subject minor revisions (`minRev_time`), and the history of major revisions (in particular, the time of the second major revision).

Let τ denote the delay from notification to the epoch of the minor revision, and w the settlement delay. Then

- For $\tau \leq w/3$, the revision multiplier is sampled from a lognormal distribution with parameters `meanlog` $= 0.15$ and `sdlog` $= 0.05$ if preceded by a 2nd major revision, `sdlog` $= 0.1$ otherwise;
- For $w/3 < \tau \leq 2w/3$, the revision multiplier is sampled from a lognormal distribution with parameters `meanlog` $= 0$ and `sdlog` $= 0.05$ if preceded by a 2nd major revision, `sdlog` $= 0.1$ otherwise;
- For $\tau > 2w/3$, the revision multiplier is sampled from a lognormal distribution with parameters `meanlog` $= -0.1$ and `sdlog` $= 0.05$ if preceded by a 2nd major revision, `sdlog` $= 0.1$ otherwise.

Note that minor revisions tend to be upward in the early part of a claim's life, and downward in the latter part.

The revision multipliers are subject to further constraints to ensure that the revised incurred estimate never falls below what has already been paid. This is dicussed in `claim_history`.

Important note: Unlike the major revision multipliers which apply to the **incurred loss estimates**, the minor revision multipliers apply to the case estimate of **outstanding claim payments** i.e. a revision multiplier of 2.54 means that at the time of the minor revision the outstanding claims payment increases by a factor of 2.54.

See Also

claims

Examples

```
set.seed(1)
test_claims <- SynthETIC::test_claims_object

# generate major revisions (required for the simulation of minor revisions)
major <- claim_majRev_freq(test_claims)
major <- claim_majRev_time(test_claims, major)
major <- claim_majRev_size(major)

# generate frequency of minor revisions
minor <- claim_minRev_freq(test_claims)
minor[[1]][[1]] # the "unit list" for the first claim

# update the timing information
minor <- claim_minRev_time(test_claims, minor)
# observe how this has changed
minor[[1]][[1]]
# with an alternative sampling distribution e.g. triangular
minRev_time_notatP <- function(n, setldel) {
  sort(rtri(n, min = setldel/6, max = setldel, mode = setldel))
}
minor_2 <- claim_minRev_time(test_claims, minor, minRev_time_notatP)

# update the revision multipliers (need to generate "major" first)
minor <- claim_minRev_size(test_claims, major, minor)
```

---

generate_data *Generate Data of Varying Complexity*

Description

[Experimental]

Generates datasets under 5 scenarios of different levels of complexity (here "complexity" means the level of difficulty of analysis).

Usage

```
generate_data(
  n_claims_per_period,
  n_periods = 40,
  complexity = c(1:5),
  data_type = c("claims", "payments", "incurred"),
  random_seed = NULL,
  verbose = TRUE
)
```

Arguments

n_claims_per_period	**expected** number of claims per period (equals the total expected number of claims divided by n_periods).
n_periods	number of accident periods considered (equals number of claims development periods considered); default 40.
complexity	integer from 1 (simplest) to 5 (most complex); see Details.
data_type	a character vector specifying output data types. By default the function will output all 3 datasets (claims, payments, incurred), but the user may choose to output only a subset.
random_seed	optional seed for random number generation for reproducibility.
verbose	logical; if TRUE print a message about the data generated.

Details

generate_data() produces datasets of varying levels of complexity, where 1 represents the simplest, and 5 represents the most complex:

- 1 – simple, homogeneous claims experience, with zero inflation.
- 2 – slightly more complex than 1, with dependence of notification delay and settlement delay on claim size, and 2% p.a. base inflation.
- 3 – steady increase in claim processing speed over occurrence periods (i.e. steady decline in settlement delays).
- 4 – inflation shock at time 30 (from 0% to 10% p.a.).
- 5 – default distributional models, with complex dependence structures (e.g. dependence of settlement delay on claim occurrence period).

We remark that this by no means defines the limits of the complexity that can be generated with SPLICE. This function is provided for the convenience of users who wish to generate (a collection of) datasets under some representative scenarios. If more complex features are required, the user is free to modify the distributional assumptions (which, of course, requires more thoughts and coding) to achieve their purposes.

Value

A named list of dataframes:

claim_dataset	A dataset of claim records that takes the same structure as test_claim_dataset, with each row represe
payment_dataset	A dataset of partial payment records that takes the same structure as test_transaction_dataset, wit
incurred_dataset	A dataset of transaction records that tracks how the case estimates change over time. Takes the same str

See Also

generate_claim_dataset, generate_transaction_dataset, generate_incurred_dataset

Examples

```
# Generate datasets of full complexity
result <- generate_data(
  n_claims_per_period = 50, data_type = c('claims', 'payments'),
  complexity = 5, random_seed = 42)

# Save individual datasets
claims <- result$claim_dataset
payments <- result$payment_dataset

# Generate chain-ladder compatible dataset
CL_simple <- generate_data(
  n_claims_per_period = 50, data_type = 'claims', complexity = 1, random_seed = 42)

# To mute message output
CL_simple_2 <- generate_data(
  n_claims_per_period = 50, data_type = 'claims', verbose = FALSE, random_seed = 42)

# Ouput is reproducible with the same random_seed value
all.equal(CL_simple$claim_dataset, CL_simple_2$claim_dataset)
```

`generate_incurred_dataset` *Generate Incurred Dataset*

Description

Generates a dataset of transaction records that tracks how the case estimates of the total incurred, the outstanding claim payments, and the cumulative claims paid change over time. A sample dataset is included as part of the package, see `test_incurred_dataset`.

Usage

```
generate_incurred_dataset(claims, incurred_history)
```

Arguments

`claims`	an `claims` object containing all the simulated quantities, (other than those related to incurred loss), see `claims`.
`incurred_history`	the full history of incurred case estimates, see `claim_history`.

Value

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

See Also

`test_incurred_dataset`

`output_incurred` *Incurred Triangles*

Description

Outputs the full square of claims incurred 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 an incurred square by occurrence and development *year*.

We refer to the package vignette for examples on changing the aggregation granularity: `vignette("SPLICE-demo",package = "SPLICE")`

Usage

```
output_incurred(
  incurred_history,
  aggregate_level = 1,
  incremental = TRUE,
  future = TRUE
)
```

Arguments

`incurred_history`	the full history of incurred case estimates, see `claim_history`.
`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 incurred square, else returns the cumulative incurred square.
`future`	logical; if true (default) shows the full claim triangle (i.e. including claim payments in future periods), else shows only the past triangle.

Details

Remark on out-of-bound transaction times: This function includes adjustment for out-of-bound transaction dates, by forcing any transactions that were projected to fall out of the maximum development period to be counted as if they were made at the end of the limiting development period. For example, if we consider 40 periods of development and a claim of the 21st occurrence period was projected to have a major revision at time 62.498210, then we would treat such a revision as if it occurred at time 60 for the purpose of tabulation.

Value

An array of claims incurred to date.

`test_incurred_dataset` *Incurred Case Estimates Dataset*

Description

A dataset of 31,250 records of transactions (partial payments and incurred revisions) associated with the 3,624 claims described in `SynthETIC`'s `test_claim_dataset`. The _inflated version includes inflation adjustment in the case estimates, while the _noInf version excludes any inflation effects.

Usage

```
test_incurred_dataset_noInf

test_incurred_dataset_inflated
```

Format

A data frame with 31,250 rows and 9 variables:

claim_no claim number, which uniquely characterises each claim.

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

txn_time double; the "absolute" time of transaction (on a continuous time scale).

txn_delay delay from notification to the subject transaction.

txn_type character; nature of the transactions, "Ma" for major revision, "Mi" for minor revision, "P" for payment, "PMa" for major revision coincident with a payment, "PMi" for minor revision coincident with a payment.

incurred double; case estimate of total incurred loss immediately **after** the transaction.

OCL double; case estimate of outstanding claim payments immediately **after** the transaction.

cumpaid double; cumulative claim paid **after** the transaction.

multiplier revision multipliers (subject to further constraints documented in `claim_history`), `NA` for transactions that do not involve a revision. Note that major revision multipliers apply to the incurred losses, while minor revision multipliers apply to the outstanding claim payments.

See Also

`generate_incurred_dataset`

`triangular` *The Triangular Distribution*

Description

Density, distribution function, quantile function and random generation for the triangular distribution with parameters `min`, `max` and `mode`.

Usage

```
ptri(q, min, max, mode)

dtri(x, min, max, mode)

qtri(p, min, max, mode)

rtri(n, min, max, mode)
```

Arguments

`min`	vector of minimum values.
`max`	vector of maximum values.
`mode`	vector of modes.
`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.

Details

The triangular distribution with parameters `min`$= a$, `max` $= b$, `mode`$= c$ has density:

$$f(x) = \begin{array}{ll} \frac{2(x-a)}{(b-a)(c-a)} & \text{for } a < x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c < x \leq b \\ 0 & \text{otherwise} \end{array}$$

and distribution function:

$$F(x) = \begin{array}{ll} 0 & \text{for } x \leq a \\ \frac{(x-a)^2}{(b-a)(c-a)} & \text{for } a < x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & \text{for } c < x \leq b \\ 1 & \text{for } x > b \end{array}$$

for $a \leq c \leq b$.

Value

`dtri` gives the density, `ptri` gives the distribution function, `qtri` gives the quantile function, and `rtri` generates random deviates.

Examples

```
ptri(c(0, 1/2, 1), min = 0, max = 1, mode = 1/2)
dtri(c(0, 1/2, 1), min = 0, max = 1, mode = 1/2)
plot(function(x) dtri(x, min = 0, max = 1, mode = 1/2), 0, 1)
```

Index