Package: lifepack (via r-universe)

January 30, 2025

Title Insurance Reserve Calculations

Version 0.0.8

Description Calculates insurance reserves and equivalence premiums using advanced numerical methods, including the Runge-Kutta algorithm and product integrals for transition probabilities. This package is useful for actuarial analyses and life insurance modeling, facilitating accurate financial projections.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://oskarallerslev.r-universe.dev

RemoteUrl https://github.com/oskarallerslev/lifepack

RemoteRef HEAD

RemoteSha 0d62652261d2a878043f9eab890896fcd2b59f49

Contents

`equiv_premium` *Equivalence Premium*

Description

This function calculates the equivalence premium for an insurance contract.

Usage

```
equiv_premium(s, t, Lambda, R, dR, mu, r, n)
```

Arguments

`s`	Initial timepoint
`t`	End timepoint
`Lambda`	Intensity matrix
`R`	Reward matrix
`dR`	Differential of reward matrix
`mu`	Equivalence premium guess
`r`	Constant rate as a scalar
`n`	Number of steps for the Runge-Kutta algorithm

Value

A scalar

Examples

```
Lambda <- function(x) matrix(c(-0.1, 0.1, 0.05, -0.05), nrow = 2)
R <- function(x, mu) matrix(c(mu, 0, 0, mu), nrow = 2) # Corrected
dR <- function(x, mu) matrix(c(0.1, 0, 0, 0.1), nrow = 2)
equiv_premium(0, 80, Lambda, R, dR, 0.05, 0.03, 100)
```

`prodint` *Productintegral This function calculates the productintegral of a matrix from s to t with a given number of steps for the runge kutta*

Description

Productintegral This function calculates the productintegral of a matrix from s to t with a given number of steps for the runge kutta

Usage

```
prodint(A, s, t, n)
```

Arguments

A	intensity matrix
s	initial timepoint
t	end timepoint
n	number of steps for the runge kutta algorithm

Value

returns a matrix (if using an intensity matrix as A you are given the transition probabilities)

Examples

```
Lambda <- function(x) matrix(c(-0.1, 0.1, 0, -0.1), 2, 2)
prodint(Lambda, 0, 80, 100)
```

---

reserve *Reserve This function calculates the reserve given the reward matrix and some constant rate This function requires proper construction of reward matrix as specified in the lecture notes provided in the course Liv1 at the University of Copenhagen.*

Description

Reserve This function calculates the reserve given the reward matrix and some constant rate This function requires proper construction of reward matrix as specified in the lecture notes provided in the course Liv1 at the University of Copenhagen.

Usage

```
reserve(t, TT, Lambda, R, mu, r, n)
```

Arguments

t	initial timepoint
TT	end timepoint
Lambda	intensity matrix
R	reward matrix
mu	equivalence premium
r	constant rate as a scalar
n	number of steps for the Runge-Kutta algorithm

Value

Returns a matrix of statewise reserves

Examples

```
Lambda <- function(x) matrix(c(-0.1, 0.1, 0, -0.1), 2, 2)
R <- function(x, mu) matrix(c(0, 0, 0, mu), 2, 2)
reserve(0, 80, Lambda, R, 200000, 0.01, 1000)
```

`sreserve` *Reserve with Dynamic Rate*

Description

This function calculates the reserve matrix using a dynamic interest rate function. It extends the functionality of the `reserve` function by allowing the rate to vary over time.

Usage

```
sreserve(t, TT, Lambda, R, mu, r, n)
```

Arguments

`t`	Initial timepoint
`TT`	End timepoint
`Lambda`	Intensity matrix
`R`	Reward matrix
`mu`	Equivalence premium
`r`	A function of time that returns the interest rate
`n`	Number of steps for the Runge-Kutta algorithm

Value

A matrix representing statewise reserves

Examples

```
Lambda <- function(x) matrix(c(-0.1, 0.1, 0, -0.1), 2, 2)
R <- function(x, mu) matrix(c(0, 0, 0, mu), 2, 2)
rentefun <- function(x) { 0.01 + 0.001 * x }  # Dynamic interest rate
sreserve(0, 80, Lambda, R, 200000, rentefun, 1000)
```

Index