Package 'lifepack'

February 28, 2025

Title Insurance Reserve Calculations

Version 0.1.0

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

Imports Rcpp, RcppArmadillo

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Author Oskar Allerslev [aut, cre]

Maintainer Oskar Allerslev <Oskar.m1660@gmail.com>

Repository CRAN

Date/Publication 2025-02-28 22:10:02 UTC

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 (C++ optim)*

Description

This function calculates the product integral of a matrix function from s to t. It uses a Runge-Kutta method implemented in C++ for efficiency.

Usage

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

Arguments

A	A function returning a matrix (intensity matrix)
s	Initial timepoint
t	End timepoint
n	Number of steps for the Runge-Kutta algorithm

Value

A matrix (transition probabilities if A is an intensity matrix)

`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