

Package ‘mapfit’

September 28, 2026

Version 1.0.1

Title PH/MAP Parameter Estimation

Type Package

Maintainer Hiroyuki Okamura <okamu@hiroshima-u.ac.jp>

Description Estimation methods for phase-type distribution (PH) and Markovian arrival process (MAP) from empirical data (point and grouped data) and density function. The tool is based on the following researches:
Okamura et al. (2009) <[doi:10.1109/TNET.2008.2008750](https://doi.org/10.1109/TNET.2008.2008750)>,
Okamura and Dohi (2009) <[doi:10.1109/QEST.2009.28](https://doi.org/10.1109/QEST.2009.28)>,
Okamura et al. (2011) <[doi:10.1016/j.peva.2011.04.001](https://doi.org/10.1016/j.peva.2011.04.001)>,
Okamura et al. (2013) <[doi:10.1002/asmb.1919](https://doi.org/10.1002/asmb.1919)>,
Horvath and Okamura (2013) <[doi:10.1007/978-3-642-40725-3_10](https://doi.org/10.1007/978-3-642-40725-3_10)>,
Okamura and Dohi (2016) <[doi:10.15807/jorsj.59.72](https://doi.org/10.15807/jorsj.59.72)>.

Encoding UTF-8

License MIT + file LICENSE

RoxygenNote 7.3.3

Imports R6, deformula, Matrix, methods, Rcpp

URL <https://github.com/okamumu/mapfit>

BugReports <https://github.com/okamumu/mapfit/issues>

Suggests covr, ggplot2, testthat (>= 3.0.0)

LinkingTo Rcpp

Config/testthat/edition 3

NeedsCompilation yes

Author Hiroyuki Okamura [aut, cre] (ORCID:
<<https://orcid.org/0000-0001-6881-0593>>)

Repository CRAN

Date/Publication 2026-09-28 09:00:10 UTC

Contents

mapfit-package	3
AERHMMClass	3
AHerlangClass	7
as.gph	10
as.map	11
BCpAug89	11
cf1	12
cf1.param	12
cf1.param.linear	13
cf1.param.power	14
CF1Class	14
ctmc.st	16
data.frame.map.group	17
data.frame.map.time	17
data.frame.phase.group	18
data.frame.phase.time	19
dphase	20
emoptions	20
erhmm	21
erhmm.param	21
ERHMMClass	22
gmpp	25
GMMPPClass	26
gph.param	27
GPHClass	28
herlang	31
herlang.param	32
HErlangClass	33
map	36
map.acf	37
map.jmoment	38
map.mmoment	39
map.param	39
MAPClass	40
mapfit.group	43
mapfit.point	45
mmpp	46
ph	47
ph.bidiag	48
ph.coxian	48
ph.mean	49
ph.moment	50
ph.tridiag	51
ph.var	51
phfit.3mom	52
phfit.density	53

mapfit-package3

phfit.group55

phfit.point57

pphase59

rphase59

Index61

mapfit-package

mapfit: PH/MAP Parameter Estimation

Description

Estimation methods for phase-type distribution (PH) and Markovian arrival process (MAP) from empirical data (point and grouped data) and density function. The tool is based on the following researches: Okamura et al. (2009) [doi:10.1109/TNET.2008.2008750](#), Okamura and Dohi (2009) [doi:10.1109/QEST.2009.28](#), Okamura et al. (2011) [doi:10.1016/j.peva.2011.04.001](#), Okamura et al. (2013) [doi:10.1002/asmb.1919](#), Horvath and Okamura (2013) [doi:10.1007/9783642407253_10](#), Okamura and Dohi (2016) [doi:10.15807/jorsj.59.72](#).

Author(s)

Maintainer: Hiroyuki Okamura <okamu@hiroshima-u.ac.jp> ([ORCID](#))

See Also

Useful links:

- <https://github.com/okamumu/mapfit>
- Report bugs at <https://github.com/okamumu/mapfit/issues>

AERHMMClass

ErlangHMM for MAP with fixed phases

Description

ErlangHMM for MAP with fixed phases
ErlangHMM for MAP with fixed phases

Details

A special case of MAP.

Methods

Public methods:

- `AERHMMClass$alpha()`
- `AERHMMClass$shape()`
- `AERHMMClass$rate()`
- `AERHMMClass$P()`
- `AERHMMClass$xi()`
- `AERHMMClass$new()`
- `AERHMMClass$copy()`
- `AERHMMClass$size()`
- `AERHMMClass$df()`
- `AERHMMClass$print()`
- `AERHMMClass$mmoment()`
- `AERHMMClass$jmoment()`
- `AERHMMClass$acf()`
- `AERHMMClass$emfit()`
- `AERHMMClass$init()`
- `AERHMMClass$clone()`

Method `alpha()`: Get alpha

Usage:

`AERHMMClass$alpha()`

Returns: A vector of alpha

Method `shape()`: Get shape

Usage:

`AERHMMClass$shape()`

Returns: A vector of shapes

Method `rate()`: Get rate

Usage:

`AERHMMClass$rate()`

Returns: A vector of rates

Method `P()`: Get P

Usage:

`AERHMMClass$P()`

Returns: A matrix of P

Method `xi()`: Get exit rates

Usage:

`AERHMMClass$xi()`

Returns: A vector of exit rates

Method new(): Create an AERHMM

Usage:

AERHMMClass\$new(size, erhmm)

Arguments:

size An integer of the number of phases

erhmm An instance of ERHMM

Returns: An instance of AERHMM

Method copy(): copy

Usage:

AERHMMClass\$copy()

Returns: A new instance

Method size(): The number of components

Usage:

AERHMMClass\$size()

Returns: The number of components

Method df(): Degrees of freedom

Usage:

AERHMMClass\$df()

Returns: The degrees of freedom

Method print(): Print

Usage:

AERHMMClass\$print(...)

Arguments:

... Others

Method mmoment(): Marginal moments

Usage:

AERHMMClass\$mmoment(k, ...)

Arguments:

k An integer of degree

... Others

Returns: A vector of moments

Method jmoment(): Joint moments

Usage:

AERHMMClass\$jmoment(lag, ...)

Arguments:

lag An integer of lag

... Others

Returns: A matrix of moments

Method `acf()`: k-lag correlation

Usage:

`AERHMMClass$acf(...)`

Arguments:

... Others

Returns: A vector for k-lag correlation

Method `emfit()`: Run EM

Usage:

`AERHMMClass$emfit(data, options, ...)`

Arguments:

data A dataframe

options A list of options

... Others

Method `init()`: Initialize with data

Usage:

`AERHMMClass$init(data, ...)`

Arguments:

data A dataframe

... Others

options A list of options

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`AERHMMClass$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

AHerlangClass	<i>Hyper-Erlang distribution with a fixed phase</i>
---------------	---

Description

Hyper-Erlang distribution with a fixed phase

Hyper-Erlang distribution with a fixed phase

Details

A mixture of Erlang distributions. A subclass of PH distributions.

Methods

Public methods:

- [AHerlangClass\\$mixrate\(\)](#)
- [AHerlangClass\\$shape\(\)](#)
- [AHerlangClass\\$rate\(\)](#)
- [AHerlangClass\\$new\(\)](#)
- [AHerlangClass\\$copy\(\)](#)
- [AHerlangClass\\$size\(\)](#)
- [AHerlangClass\\$df\(\)](#)
- [AHerlangClass\\$moment\(\)](#)
- [AHerlangClass\\$print\(\)](#)
- [AHerlangClass\\$pdf\(\)](#)
- [AHerlangClass\\$cdf\(\)](#)
- [AHerlangClass\\$ccdf\(\)](#)
- [AHerlangClass\\$sample\(\)](#)
- [AHerlangClass\\$emfit\(\)](#)
- [AHerlangClass\\$init\(\)](#)
- [AHerlangClass\\$clone\(\)](#)

Method `mixrate()`: Get mixrate

Usage:

`AHerlangClass$mixrate()`

Returns: A vector of mixrate

Method `shape()`: Get shape

Usage:

`AHerlangClass$shape()`

Returns: A vector of shapes

Method `rate()`: Get rate

Usage:

AHHerlangClass\$rate()

Returns: A vector of rates

Method new(): Create a hyper-Erlang distribution with fixed phases

Usage:

AHHerlangClass\$new(size, herlang)

Arguments:

size An integer of the number of phases

herlang An instance of HErlang

Returns: An instance of AHHerlang

Method copy(): copy

Usage:

AHHerlangClass\$copy()

Returns: A new instance

Method size(): The number of components

Usage:

AHHerlangClass\$size()

Returns: The number of components

Method df(): Degrees of freedom

Usage:

AHHerlangClass\$df()

Returns: The degrees of freedom

Method moment(): Moments of HErlang

Usage:

AHHerlangClass\$moment(k, ...)

Arguments:

k A value to indicate the degrees of moments. k-th moment

... Others

Returns: A vector of moments from 1st to k-th moments

Method print(): Print

Usage:

AHHerlangClass\$print(...)

Arguments:

... Others

Method pdf(): PDF

Usage:

AHerlangClass\$pdf(x, ...)

Arguments:

x A vector of points

... Others

Returns: A vector of densities.

Method cdf(): CDF

Usage:

AHerlangClass\$cdf(q, ...)

Arguments:

q A vector of points

... Others

Returns: A vector of probabilities

Method ccdf(): Complementary CDF

Usage:

AHerlangClass\$ccdf(q, ...)

Arguments:

q A vector of points

... Others

Returns: A vector of probabilities

Method sample(): Make a sample

Usage:

AHerlangClass\$sample(...)

Arguments:

... Others

Returns: A sample of HErlang

Method emfit(): Run EM

Usage:

AHerlangClass\$emfit(data, options, ...)

Arguments:

data A dataframe

options A list of options

... Others

Method init(): Initialize with data

Usage:

AHerlangClass\$init(data, ...)

Arguments:

data A dataframe

... Others

options A list of options

Method clone(): The objects of this class are cloneable with this method.*Usage:*

AHerlangClass\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

as.gph*Convert from HErlang to GPH*

Description

Convert from hyper-Erlang distribution to the general PH distribution

Usage

as.gph(h)

Arguments

h An instance of HErlang

Value

An instance of GPH

Examples

```

#' ## create a hyper Erlang with specific parameters
(param <- herlang(shape=c(2,3), mixrate=c(0.3,0.7), rate=c(1.0,10.0)))

## convert to a general PH
as.gph(param)

```

as.map

Convert from ERHMM to MAP

Description

Convert from ERHMM to the general MAP

Usage

```
as.map(x)
```

Arguments

x An instance of ERHMM

Value

An instance of MAP

Examples

```
## create a hyper Erlang with specific parameters
(param <- erhmm(shape=c(2,3), alpha=c(0.3,0.7), rate=c(1.0,10.0)))

## convert to a general PH
as.map(param)
```

BCpAug89

Packet Trace Data

Description

The data contains packet arrivals seen on an Ethernet at the Bellcore Morristown Research and Engineering facility. Two of the traces are LAN traffic (with a small portion of transit WAN traffic), and two are WAN traffic. The original trace BC-pAug89 began at 11:25 on August 29, 1989, and ran for about 3142.82 seconds (until 1,000,000 packets had been captured). The trace BC-pOct89 began at 11:00 on October 5, 1989, and ran for about 1759.62 seconds. These two traces captured all Ethernet packets. The number of arrivals in the original trace is one million.

Format

BCpAug89 is a vector for the inter-arrival time in seconds for 1000 arrivals.

Source

The original trace data are published in <http://ita.ee.lbl.gov/html/contrib/BC.html>.

cf1	<i>Create CF1</i>
-----	-------------------

Description

Create an instance of CF1.

Usage

```
cf1(size, alpha, rate)
```

Arguments

size	An integer of the number of phases
alpha	A vector of initial probabilities
rate	A vector of rates

Value

An instance of CF1.

Examples

```
## create a CF1 with 5 phases
(param1 <- cf1(5))

## create a CF1 with 5 phases
(param1 <- cf1(size=5))

## create a CF1 with specific parameters
(param2 <- cf1(alpha=c(1,0,0), rate=c(1.0,2.0,3.0)))
```

cf1.param	<i>Create CF1 with data information</i>
-----------	---

Description

Create CF1 with the first moment of a given data. This function calls cf1.param.linear and cf1.param.power to determine CF1. After execute 5 EM steps, the model with the smallest LLF is selected.

Usage

```
cf1.param(data, size, options, ...)
```

Arguments

data	A dataframe
size	An integer for the number of phases
options	A list of options for EM steps
...	Others. This can provide additional options for EM steps.

Examples

```
## Generate group data
dat <- data.frame.phase.group(c(1,2,0,4), seq(0,10,length.out=5))

## Create an instance of CF1
p <- cf1.param(data=dat, size=5)
```

cf1.param.linear	<i>Determine CF1 parameters</i>
------------------	---------------------------------

Description

Determine CF1 parameters based on the linear rule.

Usage

```
cf1.param.linear(size, mean, s)
```

Arguments

size	An integer of the number of phases
mean	A value of mean of data
s	A value of fraction of minimum and maximum rates

Value

A list of alpha and rate

<code>cf1.param.power</code>	<i>Determine CF1 parameters</i>
------------------------------	---------------------------------

Description

Determine CF1 parameters based on the power rule.

Usage

```
cf1.param.power(size, mean, s)
```

Arguments

<code>size</code>	An integer of the number of phases
<code>mean</code>	A value of mean of data
<code>s</code>	A value of fraction of minimum and maximum rates

Value

A list of alpha and rate

<code>CF1Class</code>	<i>Canonical phase-type distribution</i>
-----------------------	--

Description

Canonical phase-type distribution

Canonical phase-type distribution

Details

A continuous distribution dominated by a continuous-time Markov chain. A random time is given by an absorbing time. In the CF1 (canonical form 1), the infinitesimal generator is given by a bi-diagonal matrix, and whose order is determined by the ascending order.

Super class

```
mapfit::GPHClass -> CF1Class
```

Methods**Public methods:**

- `CF1Class$rate()`
- `CF1Class$new()`
- `CF1Class$copy()`
- `CF1Class$print()`
- `CF1Class$sample()`
- `CF1Class$emfit()`
- `CF1Class$init()`
- `CF1Class$clone()`

Method `rate()`: Get rate

Usage:

`CF1Class$rate()`

Returns: An instance of rate

Method `new()`: Create a CF1

Usage:

`CF1Class$new(alpha, rate)`

Arguments:

alpha A vector of initial probability

rate A vector of rates

Returns: An instance of CF1

Method `copy()`: copy

Usage:

`CF1Class$copy()`

Returns: A new instance

Method `print()`: Print

Usage:

`CF1Class$print(...)`

Arguments:

... Others

Method `sample()`: Generate a sample of CF1

Usage:

`CF1Class$sample(...)`

Arguments:

... Others

Returns: A sample of CF1

Method `emfit()`: Run EM

Usage:

`CF1Class$emfit(data, options, ...)`

Arguments:

`data` A dataframe

`options` A list of options

`...` Others

Method `init()`: Initialize with data

Usage:

`CF1Class$init(data, options, ...)`

Arguments:

`data` A dataframe

`options` A list of options

`...` Others

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`CF1Class$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

ctmc.st

Markov stationary

Description

Compute the stationary vector with GTH

Usage

`ctmc.st(Q)`

Arguments

`Q` DTMC/CTMC kernel

Value

The stationary vector of DTMC/CTMC

data.frame.map.group *Create group data for map*

Description

Provide the data.frame for group data.

Usage

```
data.frame.map.group(counts, breaks, intervals, instants)
```

Arguments

counts	A vector of the number of samples
breaks	A vector of break points
intervals	A vector of differences of time
instants	A vector meaning whether a sample is observed at the end of break.

Value

A dataframe

Examples

```
t <- c(1,1,1,1,1)
n <- c(1,3,0,0,1)

dat <- data.frame.map.group(counts=n, intervals=t)
mean(dat)
print(dat)
```

data.frame.map.time *Create data for map*

Description

Provide a data.frame with samples.

Usage

```
data.frame.map.time(time, intervals)
```

Arguments

time	A vector for cumulative time
intervals	A vector for time intervals

Value

A dataframe

Note

- If both time and intervals are used, time is used.
- map.time is given by a special case of map.group.

Examples

```
x <- runif(10)

dat <- data.frame.map.time(time=x)
mean(dat)
print(dat)
```

data.frame.phase.group

Create group data for phase

Description

Provide the data.frame for group data.

Usage

```
data.frame.phase.group(counts, breaks, intervals, instants)
```

Arguments

counts	A vector of the number of samples
breaks	A vector of break points
intervals	A vector of differences of time
instants	A vector meaning whether a sample is observed at the end of break.

Value

A dataframe

Examples

```
dat <- data.frame.phase.group(counts=c(1,2,1,1,0,0,1,4))
print(dat)
mean(dat)
```

data.frame.phase.time *Create data for phase with weighted sample*

Description

Provide a data.frame with weighted samples.

Usage

```
data.frame.phase.time(x, weights)
```

Arguments

x	A vector of point (quantiles)
weights	A vector of weights

Value

A dataframe

Note

The point time is sorted and their differences are stored as the column of time

Examples

```
x <- runif(10)
w <- runif(10)

dat <- data.frame.phase.time(x=x, weights=w)
print(dat)
mean(dat)
```

dphase	<i>Probability density function of PH distribution</i>
--------	--

Description

Compute the probability density function (p.d.f.) for a given PH distribution

Usage

```
dphase(x, ph, log = FALSE, ...)
```

Arguments

x	A numeric vector of quantiles.
ph	An instance of PH distribution.
log	logical; if TRUE, densities y are returned as log(y)
...	Others.

Value

A vector of densities.

Examples

```
## create a PH with specific parameters
(phdist <- ph(alpha=c(1,0,0),
              Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
              xi=c(2,2,0)))

## p.d.f. for 0, 0.1, ..., 1
dphase(x=seq(0, 1, 0.1), ph=phdist)
```

emoptions	<i>EM Options</i>
-----------	-------------------

Description

A list of options for EM

Usage

```
emoptions()
```

Value

A list of options with default values

`erhmm`*Create ERHMM*

Description

Create an instance of ERHMM

Usage

```
erhmm(  
  size,  
  shape,  
  alpha = rep(1/length(shape), length(shape)),  
  rate = rep(1, length(shape)),  
  P = matrix(1/length(shape), length(shape), length(shape))  
)
```

Arguments

<code>size</code>	An integer of the number of phases
<code>shape</code>	A vector of shape parameters
<code>alpha</code>	A vector of initial probability (alpha)
<code>rate</code>	A vector of rate parameters
<code>P</code>	A matrix of transition probabilities

Value

An instance of ERHMM

Note

If shape is given, shape is used even though size is set.

`erhmm.param`*Determine ERHMM parameters*

Description

Determine ERHMM parameters with k-means.

Usage

```
erhmm.param(data, skel, ...)
```

Arguments

data	A dataframe
skel	An instance of ERHMM used as a skeleton
...	Others

Value

An instance of ERHMM

ERHMMClass	<i>ErlangHMM for MAP</i>
------------	--------------------------

Description

ErlangHMM for MAP

ErlangHMM for MAP

Details

A special case of MAP.

Methods**Public methods:**

- [ERHMMClass\\$alpha\(\)](#)
- [ERHMMClass\\$shape\(\)](#)
- [ERHMMClass\\$rate\(\)](#)
- [ERHMMClass\\$P\(\)](#)
- [ERHMMClass\\$xi\(\)](#)
- [ERHMMClass\\$new\(\)](#)
- [ERHMMClass\\$copy\(\)](#)
- [ERHMMClass\\$size\(\)](#)
- [ERHMMClass\\$df\(\)](#)
- [ERHMMClass\\$print\(\)](#)
- [ERHMMClass\\$mmoment\(\)](#)
- [ERHMMClass\\$jmoment\(\)](#)
- [ERHMMClass\\$acf\(\)](#)
- [ERHMMClass\\$emfit\(\)](#)
- [ERHMMClass\\$init\(\)](#)
- [ERHMMClass\\$clone\(\)](#)

Method [alpha\(\)](#): Get alpha

Usage:

ERHMMClass\$alpha()

Returns: A vector of alpha

Method shape(): Get shape

Usage:

ERHMMClass\$shape()

Returns: A vector of shapes

Method rate(): Get rate

Usage:

ERHMMClass\$rate()

Returns: A vector of rates

Method P(): Get P

Usage:

ERHMMClass\$P()

Returns: A matrix of P

Method xi(): Get exit rates

Usage:

ERHMMClass\$xi()

Returns: A vector of exit rates

Method new(): Create an ERHMM

Usage:

ERHMMClass\$new(alpha, shape, rate, P, xi)

Arguments:

alpha A vector of initial probability

shape A vector of shape parameters

rate A vector of rate parameters

P A matrix of transition probabilities

xi An exit rate vector

Returns: An instance of ERHMM

Method copy(): copy

Usage:

ERHMMClass\$copy()

Returns: A new instance

Method size(): The number of components

Usage:

ERHMMClass\$size()

Returns: The number of components

Method `df()`: Degrees of freedom

Usage:

`ERHMMClass$df()`

Returns: The degrees of freedom

Method `print()`: Print

Usage:

`ERHMMClass$print(...)`

Arguments:

... Others

Method `mmoment()`: Marginal moments

Usage:

`ERHMMClass$mmoment(k, ...)`

Arguments:

k An integer of degree

... Others

Returns: A vector of moments

Method `jmoment()`: Joint moments

Usage:

`ERHMMClass$jmoment(lag, ...)`

Arguments:

lag An integer of lag

... Others

Returns: A matrix of moments

Method `acf()`: k-lag correlation

Usage:

`ERHMMClass$acf(...)`

Arguments:

... Others

Returns: A vector for k-lag correlation

Method `emfit()`: Run EM

Usage:

`ERHMMClass$emfit(data, options, ...)`

Arguments:

data A dataframe

options A list of options

... Others

Method `init()`: Initialize with data

Usage:

`ERHMMClass$init(data, ...)`

Arguments:

`data` A dataframe

... Others

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`ERHMMClass$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

gmmpp

Create GMMPP

Description

Create an instance of GMMPP

Usage

`gmmpp(size, alpha, D0, D1)`

Arguments

<code>size</code>	An integer for the number of phases
<code>alpha</code>	A vector of initial probability
<code>D0</code>	An infinitesimal generator without arrivals
<code>D1</code>	An infinitesimal generator with arrivals

Value

An instance of GMMPP

Note

This function can omit several patterns of arguments. For example, `map(5)` omit the arguments `alpha`, `Q` and `xi`. In this case, the default values are assigned to them.

Examples

```
## create a map (full matrix) with 5 phases
(param1 <- gmmpp(5))

## create a map with specific parameters
(param2 <- gmmpp(alpha=c(1,0,0),
  D0=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
  D1=rbind(c(2,0,0),c(0,2,0),c(0,0,0))))
```

GMMPPClass

*GMMPP: Approximation for MAP***Description**

GMMPP: Approximation for MAP

GMMPP: Approximation for MAP

Details

A point process dominated by a continuous-time Markov chain.

Super class`mapfit::MAPClass -> GMMPPClass`**Methods****Public methods:**

- `GMMPPClass$new()`
- `GMMPPClass$copy()`
- `GMMPPClass$emfit()`
- `GMMPPClass$clone()`

Method new(): Create a MAP*Usage:*`GMMPPClass$new(alpha, D0, D1, xi)`*Arguments:*

alpha A vector of initial probability

D0 An infinitesimal generator

D1 An infinitesimal generator

xi An exit rate vector

Returns: An instance of MAP**Method copy():** copy

Usage:

GMMPPClass\$copy()

Returns: A new instance

Method emfit(): Run EM

Usage:

GMMPPClass\$emfit(data, options, ...)

Arguments:

data A dataframe

options A list of options

... Others

Method clone(): The objects of this class are cloneable with this method.

Usage:

GMMPPClass\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

gph.param

Generate GPH using the information on data

Description

Generate GPH randomly and adjust parameters to fit its first moment to the first moment of data.

Usage

```
gph.param(data, skel, ...)
```

Arguments

data	A dataframe
skel	An instance of skeleton of GPH.
...	Others

Value

An instance of GPH

Examples

```
## Create data
wsample <- rweibull(10, shape=2)
(dat <- data.frame.phase.time(x=wsample))

## Generate PH that is fitted to dat
(model <- gph.param(data=dat, skel=ph(5)))
```

GPHClass

General phase-type distribution

Description

General phase-type distribution

General phase-type distribution

Details

A continuous distribution dominated by a continuous-time Markov chain. A random time is given by an absorbing time.

Methods**Public methods:**

- `GPHClass$alpha()`
- `GPHClass$Q()`
- `GPHClass$xi()`
- `GPHClass$new()`
- `GPHClass$copy()`
- `GPHClass$size()`
- `GPHClass$df()`
- `GPHClass$moment()`
- `GPHClass$print()`
- `GPHClass$pdf()`
- `GPHClass$cdf()`
- `GPHClass$ccdf()`
- `GPHClass$sample()`
- `GPHClass$emfit()`
- `GPHClass$init()`
- `GPHClass$clone()`

Method `alpha()`: Get alpha*Usage:*`GPHClass$alpha()`*Returns:* A vector of alpha**Method** `Q()`: Get Q*Usage:*`GPHClass$Q()`*Returns:* A matrix of Q

Method `xi()`: Get xi

Usage:

`GPHClass$xi()`

Returns: A vector of xi

Method `new()`: Create a GPH

Usage:

`GPHClass$new(alpha, Q, xi)`

Arguments:

`alpha` A vector of initial probability

`Q` An infinitesimal generator

`xi` An exit rate vector

Returns: An instance of GPH

Method `copy()`: copy

Usage:

`GPHClass$copy()`

Returns: A new instance

Method `size()`: The number of phases

Usage:

`GPHClass$size()`

Returns: The number of phases

Method `df()`: Degrees of freedom

Usage:

`GPHClass$df()`

Returns: The degrees of freedom

Method `moment()`: Moments of GPH

Usage:

`GPHClass$moment(k, ...)`

Arguments:

`k` A value to indicate the degrees of moments. k-th moment

`...` Others

Returns: A vector of moments from 1st to k-th moments

Method `print()`: Print

Usage:

`GPHClass$print(...)`

Arguments:

`...` Others

Method pdf(): PDF*Usage:*`GPHClass$pdf(x, poisson.eps = 1e-08, ufactor = 1.01, ...)`*Arguments:*`x` A vector of points`poisson.eps` A value of tolerance error for uniformization`ufactor` A value of uniformization factor`...` Others*Returns:* A vector of densities.**Method cdf(): CDF***Usage:*`GPHClass$cdf(x, poisson.eps = 1e-08, ufactor = 1.01, ...)`*Arguments:*`x` A vector of points`poisson.eps` A value of tolerance error for uniformization`ufactor` A value of uniformization factor`...` Others*Returns:* A vector of probabilities**Method ccdf(): Complementary CDF***Usage:*`GPHClass$ccdf(x, poisson.eps = 1e-08, ufactor = 1.01, ...)`*Arguments:*`x` A vector of points`poisson.eps` A value of tolerance error for uniformization`ufactor` A value of uniformization factor`...` Others*Returns:* A vector of probabilities**Method sample(): Make a sample***Usage:*`GPHClass$sample(...)`*Arguments:*`...` Others*Returns:* A sample of GPH**Method emfit(): Run EM***Usage:*`GPHClass$emfit(data, options, ...)`*Arguments:*

data A dataframe
 options A list of options
 ... Others

Method `init()`: Initialize with data

Usage:

`GPHClass$init(data, ...)`

Arguments:

data A dataframe
 ... Others
 options A list of options

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`GPHClass$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

Note

This function provides the values of p.d.f. for PH distribution with the uniformization technique.

This function provides the values of c.d.f. for PH distribution with the uniformization technique.

This function provides the values of complementary c.d.f. for PH distribution with the uniformization technique.

herlang

Create HErlang distribution

Description

Create an instance of Hyper-Erlang distribution

Usage

```
herlang(
  size,
  shape,
  mixrate = rep(1/length(shape), length(shape)),
  rate = rep(1, length(shape))
)
```

Arguments

size	An integer of the number of phases
shape	A vector of shape parameters
mixrate	A vector of initial probability (mixrate)
rate	A vector of rate parameters

Value

An instance of HErlang

Note

If shape is given, shape is used even though size is set.

Examples

```
## create a hyper Erlang consisting of two Erlang
## with shape parameters 2 and 3.
(param1 <- herlang(shape=c(2,3)))

## create a hyper Erlang with specific parameters
(param2 <- herlang(shape=c(2,3), mixrate=c(0.3,0.7), rate=c(1.0,10.0)))

## convert to a general PH
as.gph(param2)

## p.d.f. for 0, 0.1, ..., 1
(dphase(x=seq(0, 1, 0.1), ph=param2))

## c.d.f. for 0, 0.1, ..., 1
(pphase(q=seq(0, 1, 0.1), ph=param2))

## generate 10 samples
(rphase(n=10, ph=param2))
```

herlang.param

Determine hyper-Erlang parameters

Description

Determine the hyper-Erlang parameters with k-means.

Usage

```
herlang.param(data, shape, ...)
```

Arguments

data	A dataframe
shape	A vector of shape parameters
...	Others

Value

An instance of HErlang

Examples

```
## Create data
wsample <- rweibull(10, shape=2)
(dat <- data.frame.phase.time(x=wsample))

## Generate PH that is fitted to dat
(model <- herlang.param(data=dat, shape=c(1,2,3)))
```

HErlangClass	<i>Hyper-Erlang distribution</i>
--------------	----------------------------------

Description

Hyper-Erlang distribution
Hyper-Erlang distribution

Details

A mixture of Erlang distributions. A subclass of PH distributions.

Methods**Public methods:**

- `HErlangClass$mixrate()`
- `HErlangClass$shape()`
- `HErlangClass$rate()`
- `HErlangClass$new()`
- `HErlangClass$copy()`
- `HErlangClass$size()`
- `HErlangClass$df()`
- `HErlangClass$moment()`
- `HErlangClass$print()`
- `HErlangClass$pdf()`
- `HErlangClass$cdf()`

- `HErlangClass$ccdf()`
- `HErlangClass$sample()`
- `HErlangClass$emfit()`
- `HErlangClass$init()`
- `HErlangClass$clone()`

Method `mixrate()`: Get mixrate

Usage:

`HErlangClass$mixrate()`

Returns: A vector of mixrate

Method `shape()`: Get shape

Usage:

`HErlangClass$shape()`

Returns: A vector of shapes

Method `rate()`: Get rate

Usage:

`HErlangClass$rate()`

Returns: A vector of rates

Method `new()`: Create a hyper-Erlang distribution

Usage:

`HErlangClass$new(mixrate, shape, rate)`

Arguments:

`mixrate` A vector of initial probability

`shape` A vector of shape parameters

`rate` A vector of rate parameters

Returns: An instance of HErlang

Method `copy()`: copy

Usage:

`HErlangClass$copy()`

Returns: A new instance

Method `size()`: The number of components

Usage:

`HErlangClass$size()`

Returns: The number of components

Method `df()`: Degrees of freedom

Usage:

`HErlangClass$df()`

Returns: The degrees of freedom

Method `moment()`: Moments of HErlang

Usage:

`HErlangClass$moment(k, ...)`

Arguments:

`k` A value to indicate the degrees of moments. k-th moment

... Others

Returns: A vector of moments from 1st to k-th moments

Method `print()`: Print

Usage:

`HErlangClass$print(...)`

Arguments:

... Others

Method `pdf()`: PDF

Usage:

`HErlangClass$pdf(x, ...)`

Arguments:

`x` A vector of points

... Others

Returns: A vector of densities.

Method `cdf()`: CDF

Usage:

`HErlangClass$cdf(q, ...)`

Arguments:

`q` A vector of points

... Others

Returns: A vector of probabilities

Method `ccdf()`: Complementary CDF

Usage:

`HErlangClass$ccdf(q, ...)`

Arguments:

`q` A vector of points

... Others

Returns: A vector of probabilities

Method `sample()`: Make a sample

Usage:

```
HErlangClass$sample(...)
```

Arguments:

... Others

Returns: A sample of HErlang

Method `emfit()`: Run EM

Usage:

```
HErlangClass$emfit(data, options, ...)
```

Arguments:

data A dataframe

options A list of options

... Others

Method `init()`: Initialize with data

Usage:

```
HErlangClass$init(data, ...)
```

Arguments:

data A dataframe

... Others

options A list of options

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
HErlangClass$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

map

Create MAP

Description

Create an instance of MAP

Usage

```
map(size, alpha, D0, D1)
```

Arguments

size	An integer for the number of phases
alpha	A vector of initial probability
D0	An infinitesimal generator without arrivals
D1	An infinitesimal generator with arrivals

Value

An instance of MAP

Note

This function can omit several patterns of arguments. For example, map(5) omit the arguments alpha, D0 D1 and xi. In this case, the default values are assigned to them.

Examples

```
## create a map (full matrix) with 5 phases
(param1 <- map(5))

## create a map with specific parameters
(param2 <- map(alpha=c(1,0,0),
               D0=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
               D1=rbind(c(2,0,0),c(0,2,0),c(0,0,0))))
```

map.acf

k-lag correlation of MAP

Description

Compute k-lag correlation

Usage

```
map.acf(map, ...)
```

Arguments

map	An instance of MAP
...	Others

Value

A vector of k-lag correlation

Examples

```
## create an MAP with specific parameters
(param1 <- map(alpha=c(1,0,0),
               D0=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-4)),
               D1=rbind(c(1,1,0),c(1,0,1),c(2,0,1))))

## create an ER-HMM with specific parameters
(param2 <- erhmm(shape=c(2,3), alpha=c(0.3,0.7),
```

```

        rate=c(1.0,10.0),
        P=rbind(c(0.3, 0.7), c(0.1, 0.9)))

map.acf(map=param1)
map.acf(map=param2)

```

map.jmoment

Joint moments of MAP

Description

Compute joint moments for a given MAP

Usage

```
map.jmoment(lag, map, ...)
```

Arguments

lag	An integer for lag
map	An instance of MAP
...	Others

Value

A vector of moments

Examples

```

## create an MAP with specific parameters
(param1 <- map(alpha=c(1,0,0),
               D0=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-4)),
               D1=rbind(c(1,1,0),c(1,0,1),c(2,0,1))))

## create an ER-HMM with specific parameters
(param2 <- erhmm(shape=c(2,3), alpha=c(0.3,0.7),
                 rate=c(1.0,10.0),
                 P=rbind(c(0.3, 0.7), c(0.1, 0.9))))

map.jmoment(lag=1, map=param1)
map.jmoment(lag=1, map=param2)

```

map.mmoment	<i>Marginal moments of MAP</i>
-------------	--------------------------------

Description

Compute up to k-th marginal moments for a given MAP

Usage

```
map.mmoment(k, map, ...)
```

Arguments

k	An integer for the moments to be computed
map	An instance of MAP
...	Others

Value

A vector of moments

Examples

```
## create an MAP with specific parameters
(param1 <- map(alpha=c(1,0,0),
               D0=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-4)),
               D1=rbind(c(1,1,0),c(1,0,1),c(2,0,1))))

## create an ER-HMM with specific parameters
(param2 <- erhmm(shape=c(2,3), alpha=c(0.3,0.7),
                 rate=c(1.0,10.0),
                 P=rbind(c(0.3, 0.7), c(0.1, 0.9))))

map.mmoment(k=3, map=param1)
map.mmoment(k=3, map=param2)
```

map.param	<i>Generate MAP using the information on data</i>
-----------	---

Description

Generate MAP randomly and adjust parameters to fit its first moment to the first moment of data.

Usage

```
map.param(data, skel, ...)
```

Arguments

data	A dataframe
skel	An instance of skeleton of MAP
...	Others

Value

An instance of MAP

MAPClass	<i>General Markovian arrival process</i>
----------	--

Description

General Markovian arrival process

General Markovian arrival process

Details

A point process dominated by a continuous-time Markov chain.

Methods**Public methods:**

- [MAPClass\\$alpha\(\)](#)
- [MAPClass\\$D0\(\)](#)
- [MAPClass\\$D1\(\)](#)
- [MAPClass\\$xi\(\)](#)
- [MAPClass\\$new\(\)](#)
- [MAPClass\\$copy\(\)](#)
- [MAPClass\\$size\(\)](#)
- [MAPClass\\$df\(\)](#)
- [MAPClass\\$print\(\)](#)
- [MAPClass\\$mmoment\(\)](#)
- [MAPClass\\$jmoment\(\)](#)
- [MAPClass\\$acf\(\)](#)
- [MAPClass\\$emfit\(\)](#)
- [MAPClass\\$init\(\)](#)
- [MAPClass\\$clone\(\)](#)

Method [alpha\(\)](#): Get alpha

Usage:

`MAPClass$alpha()`

Returns: A vector of alpha

Method D0(): Get D0

Usage:

MAPClass\$D0()

Returns: A matrix of D0

Method D1(): Get D1

Usage:

MAPClass\$D1()

Returns: A matrix of D1

Method xi(): Get xi

Usage:

MAPClass\$xi()

Returns: A vector of xi

Method new(): Create a MAP

Usage:

MAPClass\$new(alpha, D0, D1, xi)

Arguments:

alpha A vector of initial probability

D0 An infinitesimal generator

D1 An infinitesimal generator

xi An exit rate vector

Returns: An instance of MAP

Method copy(): copy

Usage:

MAPClass\$copy()

Returns: A new instance

Method size(): The number of phases

Usage:

MAPClass\$size()

Returns: The number of phases

Method df(): Degrees of freedom

Usage:

MAPClass\$df()

Returns: The degrees of freedom

Method print(): Print

Usage:

MAPClass\$print(...)

Arguments:

... Others

Method `mmoment()`: Marginal moments

Usage:

MAPClass\$mmoment(k, ...)

Arguments:

k An integer of degree

... Others

Returns: A vector of moments

Method `jmoment()`: Joint moments

Usage:

MAPClass\$jmoment(lag, ...)

Arguments:

lag An integer of lag

... Others

Returns: A matrix of moments

Method `acf()`: k-lag correlation

Usage:

MAPClass\$acf(...)

Arguments:

... Others

Returns: A vector for k-lag correlation

Method `emfit()`: Run EM

Usage:

MAPClass\$emfit(data, options, ...)

Arguments:

data A dataframe

options A list of options

... Others

Method `init()`: Initialize with data

Usage:

MAPClass\$init(data, ...)

Arguments:

data A dataframe

... Others

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
MAPClass$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

mapfit.group

MAP fitting with grouped data

Description

Estimates MAP parameters from grouped data.

Usage

```
mapfit.group(map, counts, breaks, intervals, instants, ...)
```

Arguments

map	An object of R6 class. The estimation algorithm is selected depending on this class.
counts	A vector of the number of points in intervals.
breaks	A vector for a sequence of points of boundaries of intervals. This is equivalent to <code>c(0, cumsum(intervals))</code> . If this is missing, it is assigned to <code>0:length(counts)</code> .
intervals	A vector of time lengths for intervals. This is equivalent to <code>diff(breaks)</code> . If this is missing, it is assigned to <code>rep(1, length(counts))</code> .
instants	A vector of integers to indicate whether sample is drawn at the last of interval. If instant is 1, a sample is drawn at the last of interval. If instant is 0, no sample is drawn at the last of interval. By using instant, point data can be expressed by grouped data. If instant is missing, it is given by <code>rep(0L, length(counts))</code> , i.e., there are no samples at the last of interval.
...	Further options for EM steps.

Value

Returns a list with components, which is an object of S3 class `mapfit.result`;

model	an object for estimated MAP class.
llf	a value of the maximum log-likelihood.
df	a value of degrees of freedom of the model.
aic	a value of Akaike information criterion.
iter	the number of iterations.
convergence	a logical value for the convergence of estimation algorithm.

ctime	computation time (user time).
data	an object for data class
aerror	a value of absolute error for llf at the last step of algorithm.
rerror	a value of relative error for llf at the last step of algorithm.
options	a list of options used in the fitting.
call	the matched call.

Note

Unlike `phfit.group`, missing counts (NA) are not supported in MAP fitting, and breaks must start at 0, i.e., left-truncated data cannot be used. Both of them would introduce an interval whose number of arrivals is unknown, which the estimation algorithm for MAP cannot handle.

Examples

```
## load trace data
data(BCpAug89)
BCpAug89s <- head(BCpAug89, 50)

## make grouped data
BCpAug89.group <- hist(cumsum(BCpAug89s),
                      breaks=seq(0, 0.15, 0.005),
                      plot=FALSE)

## MAP fitting for general MAP
(result1 <- mapfit.group(map=map(2),
                      counts=BCpAug89.group$counts,
                      breaks=BCpAug89.group$breaks))

## MAP fitting for MMPP
(result2 <- mapfit.group(map=mmpp(2),
                      counts=BCpAug89.group$counts,
                      breaks=BCpAug89.group$breaks))

## MAP fitting with approximate MMPP
(result3 <- mapfit.group(map=gmmpp(2),
                      counts=BCpAug89.group$counts,
                      breaks=BCpAug89.group$breaks))

## marginal moments for estimated MAP
map.mmoment(k=3, map=result1$model)
map.mmoment(k=3, map=result2$model)
map.mmoment(k=3, map=result3$model)

## joint moments for estimated MAP
map.jmoment(lag=1, map=result1$model)
map.jmoment(lag=1, map=result2$model)
map.jmoment(lag=1, map=result3$model)

## lag-k correlation
map.acf(map=result1$model)
```

```
map.acf(map=result2$model)
map.acf(map=result3$model)
```

mapfit.point	<i>MAP fitting with point data</i>
--------------	------------------------------------

Description

Estimates MAP parameters from point data.

Usage

```
mapfit.point(map, x, intervals, ...)
```

Arguments

map	An object for MAP. The estimation algorithm is selected depending on this class.
x	A vector for point data.
intervals	A vector for intervals.
...	Further options for fitting methods.

Value

Returns a list with components, which is an object of S3 class `mapfit.result`;

model	an object for estimated PH class.
llf	a value of the maximum log-likelihood.
df	a value of degrees of freedom of the model.
aic	a value of Akaike information criterion.
iter	the number of iterations.
convergence	a logical value for the convergence of estimation algorithm.
ctime	computation time (user time).
data	an object for data class
aerror	a value of absolute error for llf at the last step of algorithm.
rerror	a value of relative error for llf at the last step of algorithm.
options	a list of options used for fitting.
call	the matched call.

Examples

```
## load trace data
data(BCpAug89)
BCpAug89s <- head(BCpAug89, 50)

## MAP fitting for general MAP
(result1 <- mapfit.point(map=map(2), x=cumsum(BCpAug89s)))

## MAP fitting for MMPP
(result2 <- mapfit.point(map=mmpp(2), x=cumsum(BCpAug89s)))

## MAP fitting for ER-HMM
(result3 <- mapfit.point(map=erhmm(3), x=cumsum(BCpAug89s)))

## marginal moments for estimated MAP
map.mmoment(k=3, map=result1$model)
map.mmoment(k=3, map=result2$model)
map.mmoment(k=3, map=result3$model)

## joint moments for estimated MAP
map.jmoment(lag=1, map=result1$model)
map.jmoment(lag=1, map=result2$model)
map.jmoment(lag=1, map=result3$model)

## lag-k correlation
map.acf(map=result1$model)
map.acf(map=result2$model)
map.acf(map=result3$model)
```

mmpp

*Create an MMPP***Description**

Create an instance of MMPP (Markov-Modulated Poisson Process)

Usage

```
mmpp(size)
```

Arguments

size An integer for the number of phases

Value

An instance of MMPP

Note

MMPP is a MAP whose D1 is given by a diagonal matrix.

Examples

```
## create an MMPP with 5 phases
(param1 <- mmp(5))
```

ph	Create GPH distribution
----	-------------------------

Description

Create an instance of GPH

Usage

```
ph(size, alpha, Q, xi)
```

Arguments

size	An integer for the number of phases
alpha	A vector of initial probability
Q	An infinitesimal generator
xi	An exit rate vector

Value

An instance of GPH

Note

This function can omit several patterns of arguments. For example, `ph(5)` omit the arguments `alpha`, `Q` and `xi`. In this case, the default values are assigned to them.

Examples

```
## create a PH (full matrix) with 5 phases
(param1 <- ph(5))

## create a PH (full matrix) with 5 phases
(param1 <- ph(size=5))

## create a PH with specific parameters
(param2 <- ph(alpha=c(1,0,0),
              Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
              xi=c(2,2,0)))
```

ph.bidiag	<i>Create a bi-diagonal PH distribution</i>
-----------	---

Description

Create an instance of bi-diagonal PH distribution.

Usage

```
ph.bidiag(size)
```

Arguments

size	An integer for the number of phases
------	-------------------------------------

Value

An instance of bi-diagonal PH distribution

Note

Bi-diagonal PH distribution is the PH distribution whose infinitesimal generator is given by a upper bi-diagonal matrix. This is similar to canonical form 1. But there is no restriction on the order for diagonal elements.

Examples

```
## create a bidiagonal PH with 5 phases
(param1 <- ph.bidiag(5))
```

ph.coxian	<i>Create a Coxian PH distribution</i>
-----------	--

Description

Create an instance of coxian PH distribution.

Usage

```
ph.coxian(size)
```

Arguments

size	An integer for the number of phases
------	-------------------------------------

Value

An instance of coxian PH distribution

Note

Coxian PH distribution is the PH distribution whose infinitesimal generator is given by a upper bi-diagonal matrix. This is also called canonical form 3.

Examples

```
## create a Coxian PH with 5 phases
(param1 <- ph.coxian(5))
```

ph.mean	<i>Mean of PH distribution</i>
---------	--------------------------------

Description

Compute the mean of a given PH distribution.

Usage

```
ph.mean(ph, ...)
```

Arguments

ph	An instance of PH distribution
...	Others

Value

A value of mean

Examples

```
## create a PH with specific parameters
(param1 <- ph(alpha=c(1,0,0),
              Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
              xi=c(2,2,0)))

## create a CF1 with specific parameters
(param2 <- cf1(alpha=c(1,0,0), rate=c(1.0,2.0,3.0)))

## create a hyper Erlang with specific parameters
(param3 <- herlang(shape=c(2,3), mixrate=c(0.3,0.7), rate=c(1.0,10.0)))

## mean
```

```
ph.mean(param1)
ph.mean(param2)
ph.mean(param3)
```

ph.moment

Moments of PH distribution

Description

Generate moments up to k-th moments for a given PH distribution.

Usage

```
ph.moment(k, ph, ...)
```

Arguments

k	An integer for the moments to be computed
ph	An instance of PH distribution
...	Others

Value

A vector of moments

Examples

```
## create a PH with specific parameters
(param1 <- ph(alpha=c(1,0,0),
              Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
              xi=c(2,2,0)))

## create a CF1 with specific parameters
(param2 <- cf1(alpha=c(1,0,0), rate=c(1.0,2.0,3.0)))

## create a hyper Erlang with specific parameters
(param3 <- herlang(shape=c(2,3), mixrate=c(0.3,0.7), rate=c(1.0,10.0)))

## up to 5 moments
ph.moment(5, param1)
ph.moment(5, param2)
ph.moment(5, param3)
```

ph.tridiag	Create a tri-diagonal PH distribution
------------	---------------------------------------

Description

Create an instance of tri-diagonal PH distribution.

Usage

```
ph.tridiag(size)
```

Arguments

size	An integer for the number of phases
------	-------------------------------------

Value

An instance of tri-diagonal PH distribution

Note

Tri-diagonal PH distribution is the PH distribution whose infinitesimal generator is given by a tri-diagonal matrix (band matrix).

Examples

```
## create a tridiagonal PH with 5 phases  
(param1 <- ph.tridiag(5))
```

ph.var	Variance of PH distribution
--------	-----------------------------

Description

Compute the variance of a given PH distribution.

Usage

```
ph.var(ph, ...)
```

Arguments

ph	An instance of PH distribution
...	Others

Value

A value of variance

Examples

```
## create a PH with specific parameters
(param1 <- ph(alpha=c(1,0,0),
  Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
  xi=c(2,2,0)))

## create a CF1 with specific parameters
(param2 <- cf1(alpha=c(1,0,0), rate=c(1.0,2.0,3.0)))

## create a hyper Erlang with specific parameters
(param3 <- herlang(shape=c(2,3), mixrate=c(0.3,0.7), rate=c(1.0,10.0)))

## variance
ph.var(param1)
ph.var(param2)
ph.var(param3)
```

phfit.3mom

PH fitting with three moments

Description

Estimates PH parameters from three moments.

Usage

```
phfit.3mom(
  m1,
  m2,
  m3,
  method = c("Osogami06", "Bobbio05"),
  max.phase = 50,
  epsilon = sqrt(.Machine$double.eps)
)
```

Arguments

m1	A value of the first moment.
m2	A value of the second moment.
m3	A value of the third moment.
method	The name of moment matching method.
max.phase	An integer for the maximum number of phases in the method "Osogami06".
epsilon	A value of precision in the method "Osogami06".

Value

An object of GPH.

Note

The method "Osogami06" checks the first three moments on whether there exists a PH whose three moments match to them. In such case, the method "Bobbio05" often returns an error.

References

Osogami, T. and Harchol-Balter, M. (2006) Closed Form Solutions for Mapping General Distributions to Minimal PH Distributions. *Performance Evaluation*, **63**(6), 524–552.

Bobbio, A., Horvath, A. and Telek, M. (2005) Matching Three Moments with Minimal Acyclic Phase Type Distributions. *Stochastic Models*, **21**(2-3), 303–326.

Examples

```
## Three moment matching
## Moments of Weibull(shape=2, scale=1); (0.886227, 1.0, 1.32934)
(result1 <- phfit.3mom(0.886227, 1.0, 1.32934))

## Three moment matching
## Moments of Weibull(shape=2, scale=1); (0.886227, 1.0, 1.32934)
(result2 <- phfit.3mom(0.886227, 1.0, 1.32934, method="Bobbio05"))

## mean
ph.mean(result1)
ph.mean(result2)

## variance
ph.var(result1)
ph.var(result2)

## up to 5 moments
ph.moment(5, result1)
ph.moment(5, result2)
```

phfit.density

PH fitting with density function

Description

Estimates PH parameters from density function.

Usage

```
phfit.density(
  ph,
  f,
  deformula = deformula.zeroinf,
  weight.zero = 1e-12,
  weight.reltol = 1e-08,
  start.divisions = 8,
  max.iter = 12,
  ...
)
```

Arguments

ph	An object of R6 class. The estimation algorithm is selected depending on this class.
f	A function object for a density function.
deformula	An object for formulas of numerical integration. It is not necessary to change it when the density function is defined on the positive domain [0,infinity).
weight.zero	A absolute value which is regarded as zero in numerical integration.
weight.reltol	A value for precision of numerical integration.
start.divisions	A value for starting value of divisions in deformula.
max.iter	A value for the maximum number of iterations to increase divisions in deformula.
...	Options for EM steps, which is also used to send the arguments to density function.

Value

Returns a list with components, which is an object of S3 class `phfit.result`;

model	an object for estimated PH class.
llf	a value of the maximum log-likelihood (a negative value of the cross entropy).
df	a value of degrees of freedom of the model.
KL	a value of Kullback-Leibler divergence.
iter	the number of iterations.
convergence	a logical value for the convergence of estimation algorithm.
ctime	computation time (user time).
data	an object for data class
aerror	a value of absolute error for llf at the last step of algorithm.
rerror	a value of relative error for llf at the last step of algorithm.
options	a list of options.
call	the matched call.

Note

Any of density function can be applied to the argument `f`, where `f` should be defined `f <- function(x, ...)`. The first argument of `f` should be an integral parameter. The other parameters are set in the argument `...` of `phfit.density`. The truncated density function can also be used directly.

Examples

```
#####
##### truncated density
#####

## PH fitting for general PH
(result1 <- phfit.density(ph=ph(2), f=dnorm, mean=3, sd=1))

## PH fitting for CF1
(result2 <- phfit.density(ph=cf1(2), f=dnorm, mean=3, sd=1))

## PH fitting for hyper Erlang
(result3 <- phfit.density(ph=herlang(3), f=dnorm, mean=3, sd=1))

## mean
ph.mean(result1$model)
ph.mean(result2$model)
ph.mean(result3$model)

## variance
ph.var(result1$model)
ph.var(result2$model)
ph.var(result3$model)

## up to 5 moments
ph.moment(5, result1$model)
ph.moment(5, result2$model)
ph.moment(5, result3$model)
```

phfit.group

PH fitting with grouped data

Description

Estimates PH parameters from grouped data.

Usage

```
phfit.group(ph, counts, breaks, intervals, instants, ...)
```

Arguments

<code>ph</code>	An object of R6 class. The estimation algorithm is selected depending on this class.
<code>counts</code>	A vector of the number of points in intervals.
<code>breaks</code>	A vector for a sequence of points of boundaries of intervals. This is equivalent to <code>c(0, cumsum(intervals))</code> . If this is missing, it is assigned to <code>0:length(counts)</code> .
<code>intervals</code>	A vector of time lengths for intervals. This is equivalent to <code>diff(breaks)</code> . If this is missing, it is assigned to <code>rep(1, length(counts))</code> .
<code>instants</code>	A vector of integers to indicate whether sample is drawn at the last of interval. If instant is 1, a sample is drawn at the last of interval. If instant is 0, no sample is drawn at the last of interval. By using instant, point data can be expressed by grouped data. If instant is missing, it is given by <code>rep(0L, length(counts))</code> , i.e., there are no samples at the last of interval.
<code>...</code>	Further options for EM steps.

Value

Returns a list with components, which is an object of S3 class `phfit.result`;

<code>model</code>	an object for estimated PH class.
<code>llf</code>	a value of the maximum log-likelihood.
<code>df</code>	a value of degrees of freedom of the model.
<code>aic</code>	a value of Akaike information criterion.
<code>iter</code>	the number of iterations.
<code>convergence</code>	a logical value for the convergence of estimation algorithm.
<code>ctime</code>	computation time (user time).
<code>data</code>	an object for data class
<code>aerror</code>	a value of absolute error for llf at the last step of algorithm.
<code>rerror</code>	a value of relative error for llf at the last step of algorithm.
<code>options</code>	a list of options used in the fitting.
<code>call</code>	the matched call.

Note

In this method, we can handle truncated data using NA and Inf; `phfit.group(ph=cf1(5), counts=c(countsdata, NA), breaks=c(breakdata, +Inf))` NA means missing of count data at the corresponding interval, and Inf is allowed to put the last of breaks or intervals which represents a special interval [the last break point, infinity).

Examples

```
## make sample
wsample <- rweibull(n=100, shape=2, scale=1)
wgroup <- hist(x=wsample, breaks="fd", plot=FALSE)

## PH fitting for general PH
(result1 <- phfit.group(ph=ph(2), counts=wgroup$counts, breaks=wgroup$breaks))

## PH fitting for CF1
(result2 <- phfit.group(ph=cf1(2), counts=wgroup$counts, breaks=wgroup$breaks))

## PH fitting for hyper Erlang
(result3 <- phfit.group(ph=herlang(3), counts=wgroup$counts, breaks=wgroup$breaks))

## mean
ph.mean(result1$model)
ph.mean(result2$model)
ph.mean(result3$model)

## variance
ph.var(result1$model)
ph.var(result2$model)
ph.var(result3$model)

## up to 5 moments
ph.moment(5, result1$model)
ph.moment(5, result2$model)
ph.moment(5, result3$model)
```

phfit.point	<i>PH fitting with point data</i>
-------------	-----------------------------------

Description

Estimates PH parameters from point data.

Usage

```
phfit.point(ph, x, weights, ...)
```

Arguments

ph	An object of R6 class for PH. The estimation algorithm is selected depending on this class.
x	A vector for point data.
weights	A vector of weights for points.
...	Further options for fitting methods.

Value

Returns a list with components, which is an object of S3 class `phfit.result`;

<code>model</code>	an object for estimated PH class.
<code>llf</code>	a value of the maximum log-likelihood.
<code>df</code>	a value of degrees of freedom of the model.
<code>aic</code>	a value of Akaike information criterion.
<code>iter</code>	the number of iterations.
<code>convergence</code>	a logical value for the convergence of estimation algorithm.
<code>ctime</code>	computation time (user time).
<code>data</code>	an object for data class
<code>aerror</code>	a value of absolute error for llf at the last step of algorithm.
<code>rerror</code>	a value of relative error for llf at the last step of algorithm.
<code>options</code>	a list of options used for fitting.
<code>call</code>	the matched call.

Examples

```
## make sample
wsample <- rweibull(n=100, shape=2, scale=1)

## PH fitting for general PH
(result1 <- phfit.point(ph=ph(2), x=wsample))

## PH fitting for CF1
(result2 <- phfit.point(ph=cf1(2), x=wsample))

## PH fitting for hyper Erlang
(result3 <- phfit.point(ph=herlang(3), x=wsample))

## mean
ph.mean(result1$model)
ph.mean(result2$model)
ph.mean(result3$model)

## variance
ph.var(result1$model)
ph.var(result2$model)
ph.var(result3$model)

## up to 5 moments
ph.moment(5, result1$model)
ph.moment(5, result2$model)
ph.moment(5, result3$model)
```

pphase *Distribution function of PH distribution*

Description

Compute the cumulative distribution function (c.d.f.) for a given PH distribution

Usage

```
pphase(q, ph, lower.tail = TRUE, log.p = FALSE, ...)
```

Arguments

q	A numeric vector of quantiles.
ph	An instance of PH distribution.
lower.tail	logical; if TRUE, probabilities are $P(X \leq x)$, otherwise $P(X > x)$.
log.p	logical; if TRUE, probabilities p are returned as $\log(p)$.
...	Others

Value

A vector of densities

Examples

```
## create a PH with specific parameters
(phdist <- ph(alpha=c(1,0,0),
              Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
              xi=c(2,2,0)))

## c.d.f. for 0, 0.1, ..., 1
pphase(q=seq(0, 1, 0.1), ph=phdist)
```

rphase *Sampling of PH distributions*

Description

Generate a sample from a given PH distribution.

Usage

```
rphase(n, ph, ...)
```

Arguments

<code>n</code>	An integer of the number of samples.
<code>ph</code>	An instance of PH distribution.
<code>...</code>	Others

Value

A vector of samples.

Examples

```
## create a PH with specific parameters
(phdist <- ph(alpha=c(1,0,0),
              Q=rbind(c(-4,2,0),c(2,-5,1),c(1,0,-1)),
              xi=c(2,2,0)))

## generate 10 samples
rphase(n=10, ph=phdist)
```

Index

- * **datasets**
 - BCpAug89, [11](#)
- * **package**
 - mapfit-package, [3](#)
- AERHMMClass, [3](#)
- AHerlangClass, [7](#)
- as.gph, [10](#)
- as.map, [11](#)
- BCpAug89, [11](#)
- cf1, [12](#)
- cf1.param, [12](#)
- cf1.param.linear, [13](#)
- cf1.param.power, [14](#)
- CF1Class, [14](#)
- ctmc.st, [16](#)
- data.frame.map.group, [17](#)
- data.frame.map.time, [17](#)
- data.frame.phase.group, [18](#)
- data.frame.phase.time, [19](#)
- dphase, [20](#)
- emoptions, [20](#)
- erhmm, [21](#)
- erhmm.param, [21](#)
- ERHMMClass, [22](#)
- gmmpp, [25](#)
- GMMPPClass, [26](#)
- gph.param, [27](#)
- GPHClass, [28](#)
- herlang, [31](#)
- herlang.param, [32](#)
- HErlangClass, [33](#)
- map, [36](#)
- map.acf, [37](#)
- map.jmoment, [38](#)
- map.mmoment, [39](#)
- map.param, [39](#)
- MAPClass, [40](#)
- mapfit (mapfit-package), [3](#)
- mapfit-package, [3](#)
- mapfit.group, [43](#)
- mapfit.point, [45](#)
- mapfit::GPHClass, [14](#)
- mapfit::MAPClass, [26](#)
- mmpp, [46](#)
- ph, [47](#)
- ph.bidiag, [48](#)
- ph.coxian, [48](#)
- ph.mean, [49](#)
- ph.moment, [50](#)
- ph.tridiag, [51](#)
- ph.var, [51](#)
- phfit.3mom, [52](#)
- phfit.density, [53](#)
- phfit.group, [55](#)
- phfit.point, [57](#)
- pphase, [59](#)
- rphase, [59](#)