

Package ‘timescale’

March 24, 2022

Title A package for development time calculation and other time scalings

Version 0.1.0

Description The package provides tools to compute the time elapsed in a different scaling of time. The package was developed to compute development time of ectotherms in function of temperature, but has been built to accept any other time scaling. The package offers pre-defined models, but is also structured to accept any model defined by the user.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Suggests knitr,
rmarkdown,
testthat (>= 3.0.0),
ggplot2

VignetteBuilder knitr

R topics documented:

timescale-package	2
composeModel	2
interpolateCond	3
intervalUniroot	4
modelBriere1999	5
modelGDD	6
modelLinear	7
rleInterval	7
timeScale	8
timeShift	9
validityConditions	11
validityModel	11
validityScaledPeriod	12
validityScaledTime	13
validityTime	14
variableConditions	14
variableModel	15
vectIntegral	15
vectIntegrate	16

Index**17**

timescale-package	<i>timescale: A package for development time calculation and other time scalings</i>
-------------------	--------------------------------------------------------------------------------------

Description

The package provides tools to compute the time elapsed in a different scaling of time. The package was develop to compute development time of ectotherms in function of temperature, but has been built to accept any other time scaling. It is particularly useful for non-linear thermal response of development and allows interpolation of temperature.

The principal functions of the package are `timeScale` and `timeShift`, no other function require to be explicitly called. The function `timeScale` evaluate the time elapsed between two bounds (`x1`, `x2`) into a scaled domain, or perform the back-transformation. The function `timeShift` estimate the upper bounds `x2` that would correspond to a specified period elapsed in the scaled domain.

Installation and help

The package can be installed from github by entering the following command lines in R:
`devtools::install_github("aleblancbio/timescale", build_vignettes = TRUE).`

An overview of the package can be accessed through: `browseVignettes('timescale')`. To get information on a specific function, one can use `?` or the `help()` function in R.

Author(s)

Maintainer: Alexandre Leblanc <aleblanc.bio@gmail.com>

composeModel	<i>Compose a model with its variables representation</i>
--------------	----------------------------------------------------------

Description

Compose the function of the model and the functions representing its variables, `condModel` a list a function representing each variables.

Usage

```
composeModel(model, condModel, param = list(), control = list())
```

Arguments

<code>model</code>	character corresponding to the name of the rate model.
<code>condModel</code>	list of functions that represent conditions variables in function of time (<code>v</code>) as the only argument, and to be substituted into the variables of the model. The names attribute of the list correspond to the variable names.

Details

Note that condModel must correspond to a list of functions and not of function names. This is safer as these functions are usually not predefined, but generated using interpolateCond.

Value

Return a new function that only depend on time (represented as x).

Examples

```
#Example of a call with two variables
conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5))
condModel <- interpolateCond(conditions, method = "linear")
model <- "modelLinear"
param = list(a = 1, T0 = 10)

g <- composeModel(model, condModel, param)
x <- 1:20
g(x)
```

interpolateCond	<i>Interpolate conditions</i>
-----------------	-------------------------------

Description

Interpolate variables of conditions

Usage

```
interpolateCond(conditions, method)
```

Arguments

conditions	data.frame with columns named time and variables expected to correspond to those of the model.
method	character corresponding to the name of the interpolating method. Available methods include constant and linear.

Details

The function is a wrapper of approxfun.

Value

Return a list of functions that interpolate variable according to time. Functions have only v as an argument which represent the time vector at which the function is to be interpolated. The names attribute of the list correspond to the variable names.

Examples

```
#Example of a call with two variables
conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5), hr = rnorm(10, 10, 5))
f <- interpolateCond(conditions, method = "linear")
f
#Plotting the result for temp
plot(conditions$time,conditions$temp)
x <- seq(0,30,length.out = 100)
lines(x,f$temp(x), lty = 1)
```

intervalUniroot	<i>Interval uniroot</i>
-----------------	-------------------------

Description

Extend the uniroot function from stats by accepting intervals on which the function is constant. The function then return one of the endpoints of the interval.

Usage

```
intervalUniroot(
  f,
  lower,
  upper,
  constantLower,
  constantUpper,
  correction = "constantLower",
  ...
)
```

Arguments

f	a function for which to find the root, and passed to uniroot.
lower	the lower end-point of the interval to be searched for the root.
upper	the upper end-point of the interval to be searched for the root.
constantLower	a vector of lower limits of intervals on which the function is constant.
constantUpper	a vector of upper limits of intervals on which the function is constant.
correction	a character indicating to which value of the interval correct the root, either constantLower or constantUpper.
...	arguments to pass to the function uniroot.

Details

The function does not modify the searching behaviour of uniroot, but instead correct the root if it falls on a constant interval. Note than intervals are truncated at lower and upper if they fall outside the searching interval.

Value

Return only the root (a constant), other components returned by uniroot are dismissed.

Examples

```
#Define a function that reach zero at a +- delta
f <- function(x, a, delta){
  y <- rep(0,length = length(x))
  y[x <= a - delta] <- x[x <= a - delta] - (a - delta)
  y[x >= a + delta] <- x[x >= a + delta] - (a + delta)

  return(y)
}
#Define parameters and range on which the function is known to be constant
a <- 5
delta <- 1
constantLower <- (a - delta)
constantUpper <- (a + delta)
lower = - 10
upper = 10
correction = "constantLower"
intervalUniroot(f, lower, upper, constantLower, constantUpper, correction = correction, a = a, delta = delta)

correction = "constantUpper"
intervalUniroot(f, lower, upper, constantLower, constantUpper, correction = correction, a = a, delta = delta)

correction = "constantUpper"
lower = 4.5
upper = 5.5
intervalUniroot(f, lower, upper, constantLower, constantUpper, correction = correction, a = a, delta = delta)

uniroot(f, lower = lower, upper = upper, a = a, delta = delta)$root
```

modelBriere1999

Briere 1999 Model

Description

Correspond to the non-linear model of Briere et al. (1999), describing growth rate in function of temperature.

Usage

```
modelBriere1999(temp, param = list(a, T0, TL, m), control = list())
```

Arguments

temp	numeric vector of temperature (in celcius)
param	list of parameters named: T0 the base temperature under which no development occurs, TL the temperature above which no development occurs, a a normalizing factor such as development is completed at 1, m a constant determining the shape of the function.
control	list of arguments that control the behaviour of the model; empty, kept for consistency with the general structure of rate models.

Details

The function correspond to equation of (2) of Briere et al. (1999), the rate is $r(T) = a \cdot T \cdot (T - T_0) \cdot (T_L - T)^{1/m}$ if $T_0 \leq T \leq T_L$ and zero otherwise. Setting $m = 2$ correspond to equation (1) of the paper.

Value

return a vector of rate associated to each element of temp.

Examples

```
modelBriere1999(temp = seq(8,32,2), param = list(a = 1/100, T0 = 10, TL = 30, m = 2))
```

modelGDD	<i>Growing degree day (from daily temperature)</i>
----------	----------------------------------------------------

Description

Growing degree day model based on daily approximation of temperature.

Usage

```
modelGDD(Tmin, Tmax, param = list(T0), control = list(method = 1))
```

Arguments

Tmin	numeric vector of minimum daily temperature (in celcius)
Tmax	numeric vector of maximum daily temperature (in celcius)
param	list of parameters named T0, the base temperature under which no development occurs.
control	list of arguments that control the behaviour of the model; for this function, method can be selected among 1 and 2 (see details).

Details

Two methods, described McMaster and Wilhelm (1997) can be chosen. Method 1 define GDD for one day as $(T_{max} - T_{min}) / 2 - T_0$ if greater than zero and zero otherwise; Method 2 define GDD for one day as $(T_{max} - T_{min}) / 2 - T_0$ after converting Tmin and Tmax into T0 if they were smaller values.

Value

Return a vector of rate associated to each element of temp.

Reference

McMaster GS, Wilhelm WW. 1997. Growing degree-days: one equation, two interpretations. Agricultural and Forest Meteorology. 87: 291-300. Available from: [https://doi.org/10.1016/S0168-1923\(97\)00027-0](https://doi.org/10.1016/S0168-1923(97)00027-0).

Examples

```
modelGDD(Tmin = seq(0,20,2), Tmax = 5 + seq(0,20,2), param = list(T0 = 10), control = list(method = 1))
```

modelLinear	<i>Linear rate model in function of temperature</i>
-------------	-----------------------------------------------------

Description

Correspond to normalized growing degree day model, without daily approximation of temperature.

Usage

```
modelLinear(temp, param = list(a, T0), control = list())
```

Arguments

temp	numeric vector of temperature (in celcius)
param	list of parameters named T0, the base temperature under which no development occurs, and a a normalizing factor such as development is completed at 1 (i.e. a = 1/GDD).
control	list of arguments that control the behaviour of the model; empty, kept for consistency with the general structure of rate models.

Details

The normalizing parameter should be set to one if the model returns units such as growing degree days.

Value

return a vector of rate associated to each element of temp.

Examples

```
modelLinear(temp = seq(0,30,2), param = list(a = 1, T0 = 10))
```

rleInterval	<i>rle interval</i>
-------------	---------------------

Description

Define intervals limit (lower, upper) on which a vector (x) repeat a specified value at least n time (without any other values ocuring). The function relies on the rle function from base.

Usage

```
rleInterval(x, value, n)
```

Arguments

x	an atomic vector .
value	a single value to find in x.
n	the minimum of repetition to be considered.

Value

Return a list of two vectors (lower and upper) giving the position of the beginning and end of the intervals respecting the criteria.

Examples

```
x <- sample(1:3, size = 100, replace = TRUE)
rleInterval(x, value = 1, n = 1)
rleInterval(x, value = 1, n = 2)
rleInterval(x, value = 1, n = 100)
```

timeScale	<i>Time scale</i>
-----------	-------------------

Description

Evaluate the time elapsed between two bounds (x1, x2) into a scaled domain given a rate model and conditions that represent its variables.

Usage

```
timeScale(
  x1,
  x2,
  model,
  conditions,
  param = list(),
  control = list(),
  interpolation = "constant",
  inverse = FALSE,
  assignConstant = "lower"
)
```

Arguments

x1	numeric vector of initial time from which evaluate the scaling
x2	numeric vector of final time until which evaluate the scaling
model	character corresponding to the name of the rate model (see the details section).
conditions	data.frame with columns named time and variables expected to correspond to those of the model (see the details section).
param	list parameters of the model.
control	list of arguments that control the behaviour of the model.
interpolation	character corresponding to the name of the interpolating method for conditions. Available methods only include constant at the moment and is the default value.
inverse	logical indicating if the inverse operation (i.e. scaled time to time transformation) should be performed.
assignConstant	character indicating how to assign scaled time to time when it remains constant on a time interval. Choices are lower and upper for both end of the interval.

Details

Both x1 and x2 must be in the time range provided by conditions.

model is a function defined by the user that return the rate at which time elapse in the new scale. It takes variables, model parameters (param as a list) and optional arguments (under a list named control). All arguments of the model that is not either param or control are considered variables; therefore, any number of variables and any name can be chosen, with the exception of the word time). The model should accept numeric vectors of the same length as variable and return a value greater or equal to zero. Some models are already included in the package and can serve as template (modelGDD, modelLinear, modelBriere1999).

conditions is a data.frame that contains one column named time and other columns with the same name as the model variables. It represents the evolution of the model variable through time. Checks are made when calling timeScale and timeShift to ensure conditions and the model are compatible. Other checks are made on conditions, including time must be a strictly increasing numeric containing 0. Units of times must simply match the definition of the model, but day is a good choice.

For the inverse option, a specified scaled time can be associated on an interval of time instead of a unique value, if the rate specified by the model is zero. In this case the inverse option of timeScale would return either the first occurrence the scaled time was reached (if assignConstant = "lower") or the last ((if assignConstant = "upper"))

Value

Return a vector of the same length as x1 and x2 representing the scaled time elapsed between those values.

Examples

```
conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 25, 5))
model <- "modelLinear"
param = list(a = 1, T0 = 10)
x1 = rep(0,10)
x2 = seq(11,20,length.out = 10)
z2 <- timeScale(x1, x2, model = model, conditions = conditions, param = param)
z1 <- rep(0,10,length.out = 10)
timeScale(z1, z2, model = model, conditions = conditions, param = param, inverse = TRUE)
```

timeShift

Time shift

Description

Evaluate the time elapsed between two bounds (x1, x2) into a scaled domain given a rate model and conditions that represent its variables.

Usage

```
timeShift(
  x1,
  scaledPeriod,
  model,
  conditions,
  param = list(),
  control = list(),
  interpolation = "constant",
  assignConstant = "lower"
)
```

Arguments

<code>x1</code>	numeric vector of initial time from which evaluate the scaling
<code>scaledPeriod</code>	numeric vector of period in the scaled domain to add to <code>x1</code> .
<code>model</code>	character corresponding to the name of the rate model.
<code>conditions</code>	<code>data.frame</code> with columns named time and variables expected to correspond to those of the model.
<code>param</code>	list parameters of the model.
<code>control</code>	list of arguments that control the behaviour of the model.
<code>interpolation</code>	character corresponding to the name of the interpolating method for conditions. Available methods only include constant at the moment and is the default value.
<code>assignConstant</code>	character indicating how to assign scaled time to time when it remains constant on a time interval. Choices are lower and upper for both end of the interval.

Details

Note that `x1` and the calculated `x2` must be in the time range provided by conditions. See the details section of timescale for the structure of conditions and model.

Value

Return a vector of the same length as `x1` and `x2` representing the scaled time elapsed between those values.

Examples

```
#Setting entries
conditions <- data.frame(time = seq(0,50,length.out = 100), temp = rnorm(10, 10, 5))
model <- "modelLinear"
param = list(a = 1, T0 = 10)
x1 = seq(1,10,length.out = 10)
scaledPeriod = rep(10,10)
control = list()
inverse = FALSE

#Calculating x2 time values
x2 <- timeShift(x1, scaledPeriod = scaledPeriod, model = model, conditions = conditions, param = param)
x2
```

validityConditions	<i>Conditions validity</i>
--------------------	----------------------------

Description

Check the validity of conditions with respect to its structure and compatibility with a rate model for scaling time.

Usage

```
validityConditions(conditions, model)
```

Arguments

conditions	data.frame with time and variables expected to correspond to those of the model.
model	character corresponding to the name of the rate model.

Details

Variables in the model are all function inputs excluding param and control and those of conditions are the column names except time. The function test that: (1) column names include time; (2) column names exclude param and control, as those are reserved terms of the model function; (3) variable names are the same in the model and conditions.

Value

Return TRUE if the conditions are compatibles with the model, return an error otherwise.

Examples

```
conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5))
validityConditions(conditions, model = "modelLinear")
```

validityModel	<i>Model validity</i>
---------------	-----------------------

Description

Check the validity of a rate model for scaling time.

Usage

```
validityModel(model)
```

Arguments

model	character corresponding to the name of the rate model.
-------	--------------------------------------------------------

Details

Variables in the model are all function inputs excluding param and control. The function test that:
 (1) codetime is not a variable of the model; (2) the model has at least one variable.

Value

Return TRUE if the model is valid, return an error otherwise.

Examples

```
validityModel(model = "modelLinear")
```

validityScaledPeriod	<i>Scaled period validity</i>
----------------------	-------------------------------

Description

Check that adding a period in the scaled domain from a time vector (x1), return a vector of time the range of conditions time. Also check that x1 and scaledPeriod have the same length.

Usage

```
validityScaledPeriod(  
  x1,  
  scaledPeriod,  
  model,  
  conditions,  
  param = list(),  
  control = list(),  
  interpolation = "constant"  
)
```

Arguments

x1	numeric vector of initial time from which evaluate the scaling
scaledPeriod	numeric vector of period in the scaled domain to add to x1.
model	character corresponding to the name of the rate model.
conditions	data.frame with columns named time and variables expected to correspond to those of the model.
param	list parameters of the model.
control	list of arguments that control the behaviour of the model.
interpolation	character corresponding to the name of the interpolating method for conditions. Available methods only include constant at the moment and is the default value.

Value

Return TRUE if the condition is respected and an error otherwise.

Examples

```

conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5))
condModel <- interpolateCond(conditions, method = "linear")
model <- "modelLinear"
param = list(a = 1, T0 = 10)
x1 = seq(1,10,length.out = 10)
scaledPeriod = rep(10,10)
validityScaledPeriod(x1, scaledPeriod, model = model, conditions = conditions, param = param)

```

validityScaledTime	<i>Scaled time validity</i>
--------------------	-----------------------------

Description

Check that a scaled time vector (x) is included in the scaled time range of conditions.

Usage

```

validityScaledTime(
  x,
  model,
  conditions,
  param = list(),
  control = list(),
  interpolation = "constant"
)

```

Arguments

x	numeric vector representing scaled time.
model	character corresponding to the name of the rate model.
conditions	data.frame with columns named time and variables expected to correspond to those of the model.
param	list parameters of the model.
control	list of arguments that control the behaviour of the model.
interpolation	character corresponding to the name of the interpolating method for conditions. Available methods only include constant at the moment and is the default value.

Value

Return TRUE if the condition is respected and an error otherwise.

Examples

```

conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5))
condModel <- interpolateCond(conditions, method = "linear")
model <- "modelLinear"
param = list(a = 1, T0 = 10)
x = seq(1,10,length.out = 10)
scaledPeriod = rep(10,10)
validityScaledTime(x, model = model, conditions = conditions, param = param)

```

validityTime	<i>Time validity</i>
--------------	----------------------

Description

Check that a time vector (x) is included in the range of conditions time.

Usage

```
validityTime(x, conditions)
```

Arguments

x	numeric vector representing time.
conditions	data.frame with time and variables expected to correspond to those of the model.

Details

The check serve to ensure conditions can be represented through interpolation on x.

Value

Return TRUE if the condition is respected and an error otherwise.

Examples

```
conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5))
validityTime(x = 1:10,conditions)
```

variableConditions	<i>Model variables</i>
--------------------	------------------------

Description

Access variable from conditions data.frame.

Usage

```
variableConditions(conditions)
```

Arguments

conditions	data.frame with time and variables as other columns.
------------	------------------------------------------------------

Value

Return character vector made of the variable names.

Examples

```
conditions <- data.frame(time = seq(0,30,length.out = 10), temp = rnorm(10, 10, 5))
variableConditions(conditions)
```

variableModel	<i>Model variables</i>
---------------	------------------------

Description

Access variable from a rate model function.

Usage

```
variableModel(model)
```

Arguments

model	character corresponding to the name of the rate model.
-------	--------------------------------------------------------

Details

Variables in the model are all function inputs excluding param and control.

Value

Return TRUE if the conditions and models are compatibles and can be used to scale time, return an error otherwise.

Examples

```
variableModel(model = "modelLinear")
```

vectIntegral	<i>Integral (vectorized)</i>
--------------	------------------------------

Description

Extend the Integral function from pracma by accepting vector for the limits of integration.

Usage

```
vectIntegral(f, xmin, xmax, ...)
```

Arguments

f	a function to be integrated, that can be used by the function Integral.
xmin	numeric vector lower limits of integration.
xmax	numeric vector upper limits of integration.
...	arguments to pass to the function Integral.

Value

Return a vector, of the same length as xmin and upper, giving the associated value of the integral.

Examples

```
#Simple case
f <- function(x) (x)
xmin <- seq(1,10,length.out = 10)
xmax <- seq(11,20,length.out = 10)
vectIntegral(f, xmin, xmax)

#Oscillating case (integrating over 2*pi)
f <- function(x) sin(2*pi*x/100)
xmin <- seq(0,100,length.out = 10)
xmax <- xmin+100
vectIntegral(f, xmin, xmax)
```

vectIntegrate	<i>Integrate (vectorized)</i>
---------------	-------------------------------

Description

Extend the integrate function from stats by accepting vector for the limits of integration.

Usage

```
vectIntegrate(f, lower, upper, ...)
```

Arguments

f	a function to be integrated, that can be used by the function integrate.
lower	numeric vector lower limits of integration.
upper	numeric vector upper limits of integration.
...	arguments to pass to the function integrate.

Value

Return a vector, of the same length as lower and upper, giving the associated value of the integral, other components returned by integrate are dismissed.

Examples

```
f <- function(x) (x)
lower <- seq(1,10,length.out = 10)
upper <- seq(11,20,length.out = 10)
vectIntegrate(f, lower, upper)
```


Index

`composeModel`, [2](#)

`interpolateCond`, [3](#)
`intervalUniroot`, [4](#)

`modelBriere1999`, [5](#)
`modelGDD`, [6](#)
`modelLinear`, [7](#)

`rleInterval`, [7](#)

`timeScale`, [8](#)
`timescale (timescale-package)`, [2](#)
`timescale-package`, [2](#)
`timeShift`, [9](#)

`validityConditions`, [11](#)
`validityModel`, [11](#)
`validityScaledPeriod`, [12](#)
`validityScaledTime`, [13](#)
`validityTime`, [14](#)
`variableConditions`, [14](#)
`variableModel`, [15](#)
`vectIntegral`, [15](#)
`vectIntegrate`, [16](#)