

Package ‘SoilR’

July 2, 2014

Title Models of Soil Organic Matter Decomposition

Version 1.1-23

Date 2014-04-26

Author Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller
<mamueller@bgc-jena.mpg.de>

Maintainer Markus Mueller <mamueller@bgc-jena.mpg.de>

Description This package contains functions for modeling Soil Organic
Matter decomposition in terrestrial ecosystems.

License GPL-3

Depends deSolve,methods,parallel,RUnit

Suggests FME,lattice,MASS

LazyData TRUE

NeedsCompilation no

Repository CRAN

Date/Publication 2014-04-30 18:53:34

R topics documented:

SoilR-package	6
AbsoluteFractionModern-methods	7
AbsoluteFractionModern_from_Delta14C	7
AbsoluteFractionModern_from_Delta14C-methods	8
AbsoluteFractionModern_from_Delta14C_method__matrix	8
AbsoluteFractionModern_from_Delta14C_method__numeric	8
AbsoluteFractionModern_method__BoundFc	9
as.character	9
as.character-methods	9

as.character_method__BoundInFlux	10
as.character_method__TimeMap	10
availableParticleProperties-methods	10
availableParticleSets-methods	11
bacwaveModel	11
bind.C14curves	12
BoundFc	13
BoundFc-class	14
BoundFc-methods	15
BoundFc_method__data.frame_missing_missing_missing_character_function	15
BoundFc_method__data.frame_missing_missing_missing_character_missing	16
BoundFc_method__data.frame_missing_missing_numeric_character_function	16
BoundFc_method__data.frame_missing_missing_numeric_character_missing	17
BoundFc_method__function_numeric_numeric_missing_character_missing	17
BoundFc_method__function_numeric_numeric_numeric_character_missing	18
BoundInFlux	18
BoundInFlux-class	19
BoundInFlux-methods	20
BoundInFlux_method__data.frame_missing_missing_missing_missing	20
BoundInFlux_method__data.frame_missing_missing_numeric_function	21
BoundInFlux_method__function_numeric_numeric_missing_missing	21
BoundInFlux_method__function_numeric_numeric_numeric_missing	22
BoundInFlux_method__TimeMap_missing_missing_missing_missing	22
BoundLinDecompOp	23
BoundLinDecompOp-class	23
BoundLinDecompOp-methods	24
BoundLinDecompOp_method__ConstLinDecompOp	24
BoundLinDecompOp_method__function_numeric_numeric_missing	25
BoundLinDecompOp_method__function_numeric_numeric_numeric	25
BoundLinDecompOp_method__TimeMap_missing_missing_missing	26
C14Atm	26
C14Atm_NH	27
computeResults-methods	27
ConstFc	28
ConstLinDecompOp	28
ConstLinDecompOp-class	29
ConstLinDecompOp-methods	29
ConstLinDecompOp_method__matrix	30
DecompOp-class	30
DecompOp-methods	31
DecompOp_method__DecompOp	31
DecompOp_method__matrix	32
DecompOp_method__TimeMap	32
DecompositionOperator-class	33
Delta14C-methods	33
Delta14C_from_AbsoluteFractionModern	34
Delta14C_from_AbsoluteFractionModern-methods	34
Delta14C_from_AbsoluteFractionModern_method__matrix	35

Delta14C_from_AbsoluteFractionModern_method__numeric	35
Delta14C_method__BoundFc	36
deSolve.lsoda.wrapper	36
eCO2	37
fT.Arrhenius	38
fT.Century1	38
fT.Century2	39
fT.Daycent1	40
fT.Daycent2	41
fT.Demeter	42
fT.KB	42
fT.LandT	43
fT.linear	44
fT.Q10	45
fT.RothC	46
fT.Standcarb	47
fW.Candy	48
fW.Century	49
fW.Daycent1	50
fW.Daycent2	51
fW.Demeter	52
fW.Gompertz	52
fW.Moyano	53
fW.RothC	54
fW.Skopp	55
fW.Standcarb	56
GaudinskiModel14	57
GeneralModel	59
GeneralModel-methods	60
GeneralModel_14	60
GeneralModel_14-methods	63
GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_missing_missing_missing_logical	
GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_function_logical	
GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_function_missing	
GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_logical	
GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_missing	
GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_missing_ANY_numeric_missing_missing_missing	
GeneralModel_method__numeric_ANY_numeric	68
getAccumulatedRelease	69
getAccumulatedRelease-methods	70
getAccumulatedRelease_method__Model	70
getC	71
getC-methods	72
getC14	72
getC14-methods	73
getC14_method__Model_14	73
getC_method__Model	73
getDecompOp-methods	74

getDotOut-methods	74
getF14	74
getF14-methods	75
getF14C	75
getF14C-methods	75
getF14C_method__Model_14	76
getF14R	76
getF14R-methods	77
getF14R_method__Model_14	77
getF14_method__Model_14	78
getFormat-methods	78
getFormat_method__BoundFc	79
getFunctionDefinition	79
getFunctionDefinition-methods	80
getFunctionDefinition_method__BoundFc	80
getFunctionDefinition_method__BoundInFlux	81
getFunctionDefinition_method__BoundLinDecompOp	81
getFunctionDefinition_method__ConstLinDecompOp	81
getFunctionDefinition_method__DecompositionOperator	82
getFunctionDefinition_method__TimeMap	82
getInFluxes-methods	82
getInitialValues-methods	83
getMeanTransitTime	83
getMeanTransitTime-methods	84
getMeanTransitTime_method__ConstLinDecompOp	85
getNumberOfPools-methods	86
getOutputFluxes-methods	86
getOutputReceivers-methods	86
getParticleMonteCarloSimulator-methods	86
getReleaseFlux	87
getReleaseFlux-methods	87
getReleaseFlux14	88
getReleaseFlux14-methods	88
getReleaseFlux14_method__Model_14	89
getReleaseFlux_method__Model	90
getTimeRange	91
getTimeRange-methods	91
getTimeRange_method__BoundFc	92
getTimeRange_method__BoundInFlux	92
getTimeRange_method__BoundLinDecompOp	93
getTimeRange_method__ConstLinDecompOp	93
getTimeRange_method__DecompositionOperator	94
getTimeRange_method__TimeMap	94
getTimes	95
getTimes-methods	96
getTimes_method__Model	96
getTransferCoefficients-methods	96
getTransferMatrix-methods	97

getTransitTimeDistributionDensity	97
getTransitTimeDistributionDensity-methods	98
getTransitTimeDistributionDensity_method__ConstLinDecompOp	98
getValues-methods	99
HarvardForest14CO2	99
Hua2013	100
ICBMModel	101
InFlux-class	103
InFlux-methods	104
InFlux_method__InFlux	104
InFlux_method__TimeMap	105
initialize-methods	105
initialize_method__BoundFc	106
initialize_method__BoundInFlux	106
initialize_method__BoundLinDecompOp	107
initialize_method__ConstLinDecompOp	107
initialize_method__DecompositionOperator	108
initialize_method__Model	108
initialize_method__Model_14	112
initialize_method__TimeMap	114
IntCal09	115
IntCal13	116
linesCPool	117
Model	118
Model-class	119
Model-methods	120
Model_14	120
Model_14-class	121
Model_14-methods	122
Model_method__numeric_ANY_numeric	122
OnepModel	123
OnepModel14	124
ParallelModel	126
plot-methods	127
plotC14Pool	128
plotCPool	129
plot_method__Model	130
print-methods	130
print_method__Model	130
RespirationCoefficients	131
RothCModel	131
show-methods	133
show_method__Model	134
summary-methods	134
summary_method__Model	134
ThreepairMMmodel	135
ThreepFeedbackModel	136
ThreepFeedbackModel14	139

ThreepParallelModel	142
ThreepParallelModel14	144
ThreepSeriesModel	146
ThreepSeriesModel14	148
TimeMap-class	150
TimeMap.from.DataFrame	151
TimeMap.new	152
turnoverFit	152
TwopFeedbackModel	154
TwopFeedbackModel14	155
TwopMMmodel	157
TwopParallelModel	159
TwopParallelModel14	160
TwopSeriesModel	162
TwopSeriesModel14	164
Yasso07Model	165
YassoModel	167
[-methods	168
[[[-methods	169
[[<-methods	169
[_method__Model_character	169
[-methods	169

Index**170**

SoilR-package

*Models of Soil Organic Matter Decomposition***Description**

This package contains functions for modeling Soil Organic Matter decomposition in terrestrial ecosystems.

Details

Package: SoilR
 Title: Models of Soil Organic Matter Decomposition
 Version: 1.1-22
 Date: 2014-04-26
 Author: Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>
 Maintainer: Markus Mueller <mamueller@bgc-jena.mpg.de>
 License: GPL-3
 Depends: deSolve,methods,parallel,RUnit
 Suggests: FME,lattice,MASS
 LazyData: TRUE

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

AbsoluteFractionModern-methods

~~ Methods for Function AbsoluteFractionModern ~~

Description

~~ Methods for function AbsoluteFractionModern ~~

Methods

signature(F = "BoundFc") [AbsoluteFractionModern_method__BoundFc](#)

AbsoluteFractionModern_from_Delta14C

Converts its argument to an Absolute Fraction Modern representation

Description

The function returns an object of the same type as its input, which can be a number or a matrix.
Have a look at the methods for details.

Usage

```
AbsoluteFractionModern_from_Delta14C(delta14C)
```

Arguments

delta14C

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

AbsoluteFractionModern_from_Delta14C-methods

~~ *Methods for Function* AbsoluteFractionModern_from_Delta14C

~~

Description

~~ Methods for function AbsoluteFractionModern_from_Delta14C ~~

Methods

signature(delta14C = "matrix") [AbsoluteFractionModern_from_Delta14C_method__matrix](#)

signature(delta14C = "numeric") [AbsoluteFractionModern_from_Delta14C_method__numeric](#)

AbsoluteFractionModern_from_Delta14C_method__matrix

Converts from Delta14C to Absolute Fraction Modern

Description

This method produces a matrix of Delta14C values from a Matrix or number of Absolute Fraction Modern

Arguments

delta14C An object of class matrix containing the values in Delta14C format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

AbsoluteFractionModern_from_Delta14C_method__numeric

Converts from Delta14C to Absolute Fraction Modern

Description

Converts a number or vector containing Delta14C values to the appropriate Absolute Fraction Modern values. Have a look at the methods for details.

Arguments

delta14C A numeric object containing the values in Delta14C format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

AbsoluteFractionModern_method__BoundFc
convert to Absolute Fraction Normal values

Description

convert a BoundFc object containing values in any supported format to the appropriate Absolute Fraction Modern values.

Arguments

F object containing the values in any format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

as.character *creates a character representation of the object in question*

Description

This function computes the carbon content of the pools as function of time

Usage

as.character(object)

Arguments

object The object to be printed.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

as.character-methods *~~ Methods for Function as.character ~~*

Description

~~ Methods for function as.character ~~

Methods

signature(x = "BoundInFlux") [as.character_method__BoundInFlux](#)
signature(x = "TimeMap") [as.character_method__TimeMap](#)

as.character_method__BoundInFlux

convert BoundInFlux Objects to something printable.

Description

returns a string describing the object

Arguments

x An object

...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

as.character_method__TimeMap

convert TimeMap Objects to something printable.

Description

This method is needed to print a TimeMap object.

Arguments

x An Object of class time map

...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

availableParticleProperties-methods

~~ Methods for Function availableParticleProperties ~~

Description

All methods for function availableParticleProperties are intended for internal use inside the package only.

 availableParticleSets-methods

 ~~ *Methods for Function availableParticleSets* ~~

Description

All methods for function availableParticleSets are intended for internal use inside the package only.

 bacwaveModel

Implementation of the microbial model Bacwave (bacterial waves)

Description

This function implements the microbial model Bacwave (bacterial waves), a two-pool model with a bacterial and a substrate pool. It is a special case of the general nonlinear model.

Usage

```

bacwaveModel(t, umax = 0.063, ks = 3, theta = 0.23, Dmax = 0.26,
             kd = 14.5, kr = 0.4, Y = 0.44, ival = c(S0 = 0.5, X0 = 1.5),
             BGF = 0.15, ExuM = 8, ExuT = 0.8)
  
```

Arguments

t	vector of times (in hours) to calculate a solution.
umax	a scalar representing the maximal relative growth rate of bacteria (hr-1)
ks	a scalar representing the substrate constant for growth (ug C /ml soil solution)
theta	a scalar representing soil water content (ml solution/cm ³ soil)
Dmax	a scalar representing the maximal relative death rate of bacteria (hr-1)
kd	a scalar representing the substrate constant for death of bacteria (ug C/ml soil solution)
kr	a scalar representing the fraction of death biomass recycling to substrate (unitless)
Y	a scalar representing the yield coefficient for bacteria (ug C/ugC)
ival	a vector of length 2 with the initial values for the substrate and the bacterial pools (ug C/cm ³)
BGF	a scalar representing the constant background flux of substrate (ug C/cm ³ soil/hr)
ExuM	a scalar representing the maximal exudation rate (ug C/(hr cm ³ soil))
ExuT	a scalar representing the time constant for exudation, responsible for duration of exudation (1/hr).

Details

This implementation contains default parameters presented in Zelenev et al. (2000). It produces nonlinear damped oscillations in the form of a stable focus.

Value

An object of class NIModel that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Zelenev, V.V., A.H.C. van Bruggen, A.M. Semenov. 2000. "BACWAVE," a spatial-temporal model for traveling waves of bacterial populations in response to a moving carbon source in soil. *Microbial Ecology* 40: 260-272.

Examples

```
hours=seq(0,800,0.1)

#Run the model with default parameter values
bcmode1=bacwaveModel(t=hours)
Cpools=getC(bcmode1)

#Time solution
matplot(hours,Cpools,type="l",ylab="Concentrations",xlab="Hours",lty=1,ylim=c(0,max(Cpools)*1.2))
legend("topleft",c("Substrate", "Microbial biomass"),lty=1,col=c(1,2),bty="n")

#State-space diagram
plot(Cpools[,2],Cpools[,1],type="l",ylab="Substrate",xlab="Microbial biomass")

#Microbial biomass over time
plot(hours,Cpools[,2],type="l",col=2,xlab="Hours",ylab="Microbial biomass")
```

bind.C14curves

Binding of pre- and post-bomb Delta14C curves

Description

This function takes a pre- and a post-bomb curve, binds them together, and reports the results back either in years BP or AD.

Usage

```
bind.C14curves(prebomb, postbomb, time.scale)
```

Arguments

prebomb	A pre-bomb radiocarbon dataset. They could be either IntCal09 or IntCal13 .
postbomb	A post-bomb radiocarbon dataset. They could be any of the datasets in Hua2013 .
time.scale	A character indicating whether to report the results in years before present BP or anno domini AD.

Value

A data.frame with 3 columns: years in AD or BP, the atmospheric Delta14C value, the standard deviation of the Delta14C value.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
#Bind the IntCal13 dataset and Hua2013 for the NH Zone 1
bp=bind.C14curves(prebomb=IntCal13,postbomb=Hua2013$NHZone1,time.scale="BP")

plot(bp[,1:2],type="l")
plot(bp[,1:2],type="l",xlim=c(-100,100))

#Report results in years AD
ad=bind.C14curves(prebomb=IntCal13,postbomb=Hua2013$NHZone1,time.scale="AD")

plot(ad[,1:2],type="l")
plot(ad[,1:2],type="l",xlim=c(0,2010))
abline(v=1950,lty=2)
```

BoundFc	<i>generic constructor</i>
---------	----------------------------

Description

create a BoundFc object from different sources

Usage

```
BoundFc(map, starttime, endtime, lag, format, interpolation)
```

Arguments

```
map
starttime
endtime
lag
format
interpolation
```

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundFc-class

Class "BoundFc"

Description

Objects of this class contain a time dependent function describing the Atmospheric ^{14}C fraction and a format description, that allows to use the numeric valuest to be interpreted correctly in subsequent computations.

Slots

starttime: Object of class "numeric" ~~

endtime: Object of class "numeric" ~~

map: Object of class "function" ~~

lag: Object of class "numeric" ~~

format: Object of class "character" ~~

Methods

AbsoluteFractionModern signature(F = "BoundFc"): ...

Delta14C signature(F = "BoundFc"): ...

getFormat signature(object = "BoundFc"): ...

getFunctionDefinition signature(object = "BoundFc"): ...

getTimeRange signature(object = "BoundFc"): ...

initialize signature(.Object = "BoundFc"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("BoundFc")
```

BoundFc-methods *~~ Methods for Function BoundFc ~~*

Description

~~ Methods for function BoundFc ~~

Methods

```
signature(map = "data.frame", starttime = "missing", endtime = "missing", lag = "missing", format =
  BoundFc_method__data.frame_missing_missing_missing_character_function
signature(map = "data.frame", starttime = "missing", endtime = "missing", lag = "missing", format =
  BoundFc_method__data.frame_missing_missing_missing_character_missing
signature(map = "data.frame", starttime = "missing", endtime = "missing", lag = "numeric", format =
  BoundFc_method__data.frame_missing_missing_numeric_character_function
signature(map = "data.frame", starttime = "missing", endtime = "missing", lag = "numeric", format =
  BoundFc_method__data.frame_missing_missing_numeric_character_missing
signature(map = "function", starttime = "numeric", endtime = "numeric", lag = "missing", format =
  BoundFc_method__function_numeric_numeric_missing_character_missing
signature(map = "function", starttime = "numeric", endtime = "numeric", lag = "numeric", format =
  BoundFc_method__function_numeric_numeric_numeric_character_missing
```

BoundFc_method__data.frame_missing_missing_missing_character_function
constructor

Description

wrapper for `BoundFc_method__data.frame_missing_missing_numeric_character_missing` with the assumption `lag=0`

Arguments

map
format
interpolation

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundFc_method__data.frame_missing_missing_missing_character_missing
constructor

Description

wrapper for `BoundFc_method__data.frame_missing_missing_numeric_character_missing` with the assumption `lag=0` `interpolation = splinefun`

Arguments

map
format
interpolation

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundFc_method__data.frame_missing_missing_numeric_character_function
constructor

Description

the method constructs an object from a dataframe a timelag format using the given interpolating function

Arguments

map	A data frame containing exactly two columns: the first one is interpreted as time the second one is interpreted as atmospheric C14 fraction in the format mentioned
lag	a scalar describing the time lag. Positive Values shift the argument of the interpolation function forward in time. (retard its effect)
format	a string that specifies the format used to represent the atmospheric fraction. Possible values are "Delta14C" which is the default or "afn" the Absolute Fraction Normal representation
interpolation	A function that returns a function the default is splinefun. Other possible values are the linear interpolation approxfun or any self made function with the same interface.

Value

An object that contains the interpolation function and the limits of the time range where the function is valid. Note that the limits change according to the time lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundFc_method__data.frame_missing_missing_numeric_character_missing
constructor

Description

wrapper for [BoundFc_method__data.frame_missing_missing_numeric_character_missing](#) with the assumption interpolation =splinefun

Arguments

map
lag
format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundFc_method__function_numeric_numeric_missing_character_missing
constructor

Description

wrapper for [BoundFc_method__function_numeric_numeric_numeric_character_missing](#) with the assumption lag=0

Arguments

map
starttime
endtime
format

Value

An object that contains the interpolation function and the limits of the time range where the function is valid. Note that the limits change according to the time lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundFc_method__function_numeric_numeric_character_missing
constructor

Description

the method constructs an object from a function a timerange where it is valid and a format

Arguments

map	a function of one argument (time)
starttime	the point in time from which map is a valid representation
endtime	the point in time until which map is a valid representation
lag	a scalar describing the time lag. Positive Values shift the argument of the interpolation function forward in time. (retard its effect)
format	a string that specifies the format used to represent the atmospheric fraction. Possible values are "Delta14C" which is the default or "afn" the Absolute Fraction Normal representation

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundInFlux *generic constructor*

Description

create a BoundInFlux object from different sources

Usage

BoundInFlux(map, starttime, endtime, lag, interpolation)

Arguments

map
starttime
endtime
lag
interpolation

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundInFlux-class *Class "BoundInFlux"*

Description

defines a time dependent inputrate as function of time and including the domain where the function is well defined. This can be used to avoid interpolations out of range when mixing different time dependent data sets

Slots

starttime: Object of class "numeric" ~~
endtime: Object of class "numeric" ~~
map: Object of class "function" ~~
lag: Object of class "numeric" ~~

Extends

Class "[InFlux](#)", directly.

Methods

as.character signature(x = "BoundInFlux"): ...
getFunctionDefinition signature(object = "BoundInFlux"): ...
getTimeRange signature(object = "BoundInFlux"): ...
initialize signature(.Object = "BoundInFlux"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("BoundInFlux")
```

BoundInFlux-methods *~~ Methods for Function BoundInFlux ~~*

Description

~~ Methods for function BoundInFlux ~~

Methods

signature(map = "TimeMap", starttime = "missing", endtime = "missing", lag = "missing", interpolation = "linear")
[BoundInFlux_method__TimeMap_missing_missing_missing_missing](#)

signature(map = "data.frame", starttime = "missing", endtime = "missing", lag = "missing", interpolation = "linear")
[BoundInFlux_method__data.frame_missing_missing_missing_missing](#)

signature(map = "data.frame", starttime = "missing", endtime = "missing", lag = "numeric", interpolation = "linear")
[BoundInFlux_method__data.frame_missing_missing_numeric_function](#)

signature(map = "function", starttime = "numeric", endtime = "numeric", lag = "missing", interpolation = "linear")
[BoundInFlux_method__function_numeric_numeric_missing_missing](#)

signature(map = "function", starttime = "numeric", endtime = "numeric", lag = "numeric", interpolation = "linear")
[BoundInFlux_method__function_numeric_numeric_numeric_missing](#)

BoundInFlux_method__data.frame_missing_missing_missing_missing
constructor

Description

This function is another constructor of the class BoundInFlux.

Arguments

map A data frame; the first column is interpreted as time

Value

An object of class BoundInFlux that contains the interpolation function and the limits of the time range where the function is valid. Note that the limits change according to the time lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundInFlux_method__data.frame_missing_missing_numeric_function
constructor

Description

This function is another constructor of the class BoundInFlux.

Arguments

map A data frame; the first column is interpreted as time
lag lag time
interpolation function used for interpolation

Value

An object of class BoundInFlux that contains the interpolation function and the limits of the time range where the function is valid. Note that the limits change according to the time lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundInFlux_method__function_numeric_numeric_missing_missing
constructor

Description

the method constructs an object from its basic ingredients

Arguments

map
starttime
endtime

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundInFlux_method__function_numeric_numeric_numeric_missing
constructor

Description

the method constructs an object from its basic ingredients

Arguments

map
starttime
endtime
lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundInFlux_method__TimeMap_missing_missing_missing_missing
convert to BoundInFlux

Description

The method is used internally to convert TimeMap objects to BoundInFlux objects, since the use of TimeMap objects is now deprecated.

Arguments

map

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundLinDecompOp	<i>Generic constructor</i>
------------------	----------------------------

Description

Creates a LinearDecompositonOperator from different sources. Please look at the methods to see what kind of input is supported.

Usage

```
BoundLinDecompOp(map, starttime, endtime, lag)
```

Arguments

map
starttime
endtime
lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundLinDecompOp-class	<i>Class "BoundLinDecompOp"</i>
------------------------	---------------------------------

Description

NA

Slots

map: Object of class "function" ~~
lag: Object of class "numeric" ~~
starttime: Object of class "numeric" ~~
endtime: Object of class "numeric" ~~

Extends

Class "[DecompOp](#)", directly.

Methods

getFunctionDefinition signature(object = "BoundLinDecompOp"): ...

getTimeRange signature(object = "BoundLinDecompOp"): ...

initialize signature(.Object = "BoundLinDecompOp"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("BoundLinDecompOp")
```

BoundLinDecompOp-methods

~~ *Methods for Function BoundLinDecompOp* ~~

Description

~~ Methods for function BoundLinDecompOp ~~

Methods

signature(map = "ConstLinDecompOp", starttime = "ConstLinDecompOp", endtime = "ConstLinDecompOp", lag = "missing")
[BoundLinDecompOp_method__ConstLinDecompOp](#)

signature(map = "TimeMap", starttime = "missing", endtime = "missing", lag = "missing")
[BoundLinDecompOp_method__TimeMap_missing_missing_missing](#)

signature(map = "function", starttime = "numeric", endtime = "numeric", lag = "missing")
[BoundLinDecompOp_method__function_numeric_numeric_missing](#)

signature(map = "function", starttime = "numeric", endtime = "numeric", lag = "numeric")
[BoundLinDecompOp_method__function_numeric_numeric_numeric](#)

BoundLinDecompOp_method__ConstLinDecompOp

convert a ConstLinDecompOp to a BoundLinDecompOp

Description

The method creates a BoundLinDecompOp consisting of a constant time dependent function and the limits of its domain (starttime and endtime) set to -Inf and Inf respectively

Arguments

map

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundLinDecompOp_method__function_numeric_numeric_missing
a constructor

Description

This method creates a BoundLinDecompOp from a timedependent function and its domain

Arguments

map
starttime
endtime

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundLinDecompOp_method__function_numeric_numeric_numeric
a constructor

Description

This method creates a BoundLinDecompOp from a timedependent function and its domain

Arguments

map	a function
starttime	the begin of the time domain
endtime	the end of the time domain
lag	lag time

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

BoundLinDecompOp_method__TimeMap_missing_missing_missing
create a BoundLinDecompOp from a TimeMap

Description

The method is used internally to convert TimeMap objects to BoundLinDecompOp where the use of TimeMap is now deprecated.

Arguments

map

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

C14Atm *Atmospheric 14C fraction*

Description

Atmospheric 14C fraction in units of Delta14C for the bomb period in the northern hemisphere.

Usage

```
data(C14Atm)
```

Format

A data frame with 108 observations on the following 2 variables.

V1 a numeric vector

V2 a numeric vector

Note

This function will be deprecated soon. Please use [C14Atm_NH](#) or [Hua2013](#) instead.

Examples

```
#Notice that C14Atm is a shorter version of C14Atm_NH
plot(C14Atm_NH,type="l")
lines(C14Atm,col=2)
```

C14Atm_NH	<i>Post-bomb atmospheric 14C fraction</i>
-----------	---

Description

Atmospheric 14C concentrations for the post-bomb period expressed as Delta 14C in per mille. This dataset contains a combination of observations from locations in Europe and North America. It is representative for the Northern Hemisphere.

Usage

```
data(C14Atm_NH)
```

Format

A data frame with 111 observations on the following 2 variables.

YEAR a numeric vector with year of measurement.

Atmosphere a numeric vector with the Delta 14 value of atmospheric CO2 in per mil.

Examples

```
plot(C14Atm_NH, type="l")
```

```
computeResults-methods
```

```
~~ Methods for Function computeResults ~~
```

Description

All methods for function `computeResults` are intended for internal use inside the package only.

ConstFc	<i>creates an object containing the initial values for the 14C fraction needed to create models in SoilR</i>
---------	--

Description

The function returns an object of class ConstFc which is a building block for any 14C model in SoilR. The building blocks of a model have to keep information about the formats their data are in, because the high level function dealing with the models have to know. This function is actually a convenient wrapper for a call to R's standard constructor new, to hide its complexity from the user.

Usage

```
ConstFc(values = c(0), format = "Delta14C")
```

Arguments

values	a numeric vector
format	a character string describing the format e.g. "Delta14C"

Value

An object of class ConstFc that contains data and a format description that can later be used to convert the data into other formats if the conversion is implemented.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

ConstLinDecompOp	<i>Generic constructor</i>
------------------	----------------------------

Description

Creates a ConstantDecompositionOperator object from different sources. Please look at the different methods to see what kind of input is supported.

Usage

```
ConstLinDecompOp(mat)
```

Arguments

mat

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

ConstLinDecompOp-class
Class "ConstLinDecompOp"

Description

NA

Slots

mat: Object of class "matrix" ~~

Extends

Class "[DecompOp](#)", directly.

Methods

BoundLinDecompOp signature(map = "ConstLinDecompOp", starttime = "ANY", endtime = "ANY", lag = "ANY", ...)

getFunctionDefinition signature(object = "ConstLinDecompOp"): ...

getMeanTransitTime signature(object = "ConstLinDecompOp"): ...

getTimeRange signature(object = "ConstLinDecompOp"): ...

getTransitTimeDistributionDensity signature(object = "ConstLinDecompOp"): ...

initialize signature(.Object = "ConstLinDecompOp"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("ConstLinDecompOp")
```

ConstLinDecompOp-methods
~~ Methods for Function ConstLinDecompOp ~~

Description

~~ Methods for function ConstLinDecompOp ~~

Methods

signature(mat = "matrix") [ConstLinDecompOp_method__matrix](#)

ConstLinDecompOp_method__matrix
construct from matrix

Description

This method creates a ConstLinDecompOp from a matrix. The operator is assumed to act on the vector of carbon stocks by multiplication of the (time invariant) matrix from the left.

Arguments

mat

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

DecompOp-class *Class "DecompOp"*

Description

NA

Methods

DecompOp signature(object = "DecompOp"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("DecompOp")
```

DecompOp-methods *~~ Methods for Function DecompOp ~~*

Description

~~ Methods for function DecompOp ~~

Methods

signature(object = "DecompOp") [DecompOp_method__DecompOp](#)

signature(object = "TimeMap") [DecompOp_method__TimeMap](#)

signature(object = "matrix") [DecompOp_method__matrix](#)

DecompOp_method__DecompOp
pass through constructor

Description

This method handles the case that no actual construction is necessary since the argument is already of a subclass of DecompOp

Arguments

object

Value

the unchanged argument

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

DecompOp_method__matrix

creates a ConstLinDecompOp from a matrix

Description

The resulting operator is created by a call to the constructor of class ConstLinDecompOp

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

DecompOp_method__TimeMap

creates a BoundLinDecompOp from a TimeMap object

Description

The resulting operator is created by a call to the constructor of class BoundLinDecompOp. The method is used to ensure backward compatibility with the now deprecated TimeMap class.

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

DecompositionOperator-class
Class "DecompositionOperator"

Description

The new class implementing the same functionality is names BoundLinDecompOp

Slots

map: Object of class "function" ~~
 lag: Object of class "numeric" ~~
 starttime: Object of class "numeric" ~~
 endtime: Object of class "numeric" ~~

Extends

Class "[DecompOp](#)", directly.

Methods

getFunctionDefinition signature(object = "DecompositionOperator"): ...
getTimeRange signature(object = "DecompositionOperator"): ...
initialize signature(.Object = "DecompositionOperator"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("DecompositionOperator")
```

Delta14C-methods *~~ Methods for Function Delta14C ~~*

Description

~~ Methods for function Delta14C ~~

Methods

signature(F = "BoundFc") [Delta14C_method__BoundFc](#)

Delta14C_from_AbsoluteFractionModern

Converts its argument from an Absolute Fraction Modern to a Delta14C representation

Description

The function returns an object of the same type as its input, which can be of different type. Have a look at the methods for details.

Usage

```
Delta14C_from_AbsoluteFractionModern(AbsoluteFractionModern)
```

Arguments

AbsoluteFractionModern
A numeric object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Delta14C_from_AbsoluteFractionModern-methods

~~ *Methods for Function* Delta14C_from_AbsoluteFractionModern
~~

Description

~~ Methods for function Delta14C_from_AbsoluteFractionModern ~~

Methods

signature(AbsoluteFractionModern = "matrix") [Delta14C_from_AbsoluteFractionModern_method__matrix](#)
signature(AbsoluteFractionModern = "numeric") [Delta14C_from_AbsoluteFractionModern_method__numeric](#)

Delta14C_from_AbsoluteFractionModern_method__matrix

Converts Absolute Fraction Modern values to Delta14C

Description

This method produces a matrix of Delta14C values from a matrix of values in Absolute Fraction Modern.

Arguments

AbsoluteFractionModern

An object of class matrix containing the values in Absolute Fraction Modern format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Delta14C_from_AbsoluteFractionModern_method__numeric

Converts to Delta14C format

Description

This method produces Delta14C values from Absolute Fraction Modern Have a look at the methods for details.

Arguments

AbsoluteFractionModern

A numeric object containing the values in Absolute Fraction Modern format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Delta14C_method__BoundFc

convert to Absolute Fraction Normal values

Description

convert object containing values in any supported format to the appropriate Absolute Fraction Modern values.

Arguments

F object of containing the values in any format

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

deSolve.lsoda.wrapper *deSolve lsoda wrapper*

Description

The function serves as a wrapper for lsoda using a much simpler interface which allows the use of matrices in the definition of the derivative. To use lsoda we have to convert our vectors to lists, define tolerances and so on. This function does this for us , so we don't need to bother about it.

Usage

```
deSolve.lsoda.wrapper(t, ydot, startValues)
```

Arguments

t A row vector containing the points in time where the solution is sought.
ydot The function of y and t that computes the derivative for a given point in time and a column vector y.
startValues A column vector with the starting values.

Value

A matrix. Every column represents a pool and every row a point in time

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

eCO2

Soil CO2 efflux from incubation experiments

Description

A dataset with soil CO2 efflux measurements in two laboratory incubations at controlled temperature and moisture conditions.

Usage

```
data(eCO2)
```

Format

A data frame with the following 4 variables.

Sample Sample code: AK_T25 is a soil from a boreal forest in Alaska incubated at 25 degrees C. HN_T35 is a soil from a temperate German forest (Hainich National Park) incubated at 35 degrees C.

Days A numeric vector with the day of measurement.

eCO2mean A numeric vector with the accumulated release of CO2. Units in mg C g-1 soil C.

eCO2sd A numeric vector with the standard deviation of the accumulated release of CO2. Units in mg C g-1 soil C.

Details

Two laboratory incubation experiments were performed in April-May 2013 for a period of 42 days. Soil CO2 measurements were taken under controlled laboratory conditions. One soil was sampled at a boreal forest site (Caribou Poker Research Watershed, Alaska, USA) and the other from a temperate forest in Germany (Hainich National Park). For each soil 5 replicates were incubated and sampled. This dataset presents the mean and standard deviation of the measurements.

Examples

```
head(eCO2)
```

```
plot(eCO2mean~Days,data=eCO2,subset=Sample=="HN_T35",col=2)
points(eCO2mean~Days,data=eCO2,subset=Sample=="AK_T25",col=4)
legend("topleft",
       c("HN_T35: temperate forest soil","AK_T25: boreal forest soil"),
       pch=1,col=c(2,4),bty="n")
```

fT.Arrhenius	<i>Effects of temperature on decomposition rates according the Arrhenius equation</i>
--------------	---

Description

Calculates the effects of temperature on decomposition rates according to the Arrhenius equation.

Usage

```
fT.Arrhenius(Temp, A = 1000, Ea = 75000, Re = 8.3144621)
```

Arguments

Temp	A scalar or vector containing values of temperature (in degrees Kelvin) for which the effects on decomposition rates are calculated.
A	A scalar defining the pre-exponential factor.
Ea	A scalar defining the activation energy in units of J mol ⁻¹ .
Re	A scalar defining the universal gas constant in units of J K ⁻¹ mol ⁻¹ .

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
Temperature=273:300
plot(Temperature,fT.Arrhenius(Temperature),type="l",ylab="f(T) (unitless)", xlab="Temperature (K)",
     main="Effects of temperature on decomposition rates according to the Arrhenius equation")
```

fT.Century1	<i>Effects of temperature on decomposition rates according the the CENTURY model</i>
-------------	--

Description

Calculates the effects of temperature on decomposition rates according to the CENTURY model.

Usage

```
fT.Century1(Temp, Tmax = 45, Topt = 35)
```

Arguments

Temp	A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated.
Tmax	A scalar defining the maximum temperature in degrees C.
Topt	A scalar defining the optimum temperature for the decomposition process in degrees C.

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Burke, I. C., J. P. Kaye, S. P. Bird, S. A. Hall, R. L. McCulley, and G. L. Sommerville. 2003. Evaluating and testing models of terrestrial biogeochemistry: the role of temperature in controlling decomposition. Pages 235-253 in C. D. Canham, J. J. Cole, and W. K. Lauenroth, editors. Models in ecosystem science. Princeton University Press, Princeton.

Examples

```
Temperature=0:50
plot(Temperature,fT.Century1(Temperature),type="l",ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition rates according to the Century model")
```

fT.Century2	<i>Effects of temperature on decomposition rates according to the CENTURY model</i>
-------------	---

Description

Calculates the effects of temperature on decomposition rates according to the CENTURY model.

Usage

```
fT.Century2(Temp, Tmax = 45, Topt = 35)
```

Arguments

Temp	A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated.
Tmax	A scalar defining the maximum temperature in degrees C.
Topt	A scalar defining the optimum temperature for the decomposition process in degrees C.

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Adair, E. C., W. J. Parton, S. J. D. Grosse, W. L. Silver, M. E. Harmon, S. A. Hall, I. C. Burke, and S. C. Hart. 2008. Simple three-pool model accurately describes patterns of long-term litter decomposition in diverse climates. *Global Change Biology* 14:2636-2660.

Examples

```
Temperature=0:50
plot(Temperature, fT.Century2(Temperature), type="l",
     ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition rates according to the Century model")
```

fT.Daycent1

Effects of temperature on decomposition rates according to the DAYCENT model

Description

Calculates the effects of temperature on decomposition rates according to the DAYCENT model.

Usage

```
fT.Daycent1(Temp)
```

Arguments

Temp	A scalar or vector containing values of soil temperature for which the effects on decomposition rates are calculated
------	--

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Kelly, R. H., W. J. Parton, M. D. Hartman, L. K. Stretch, D. S. Ojima, and D. S. Schimel (2000), Intra-annual and interannual variability of ecosystem processes in shortgrass steppe, *J. Geophys. Res.*, 105.

Examples

```
Temperature=0:50
plot(Temperature, fT.Daycent1(Temperature), type="l", ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition rates according to the DAYCENT model")
```

fT.Daycent2	<i>Effects of temperature on decomposition rates according to the DAYCENT model</i>
-------------	---

Description

Calculates the effects of temperature on decomposition rates according to the Daycent/Century models.

Usage

```
fT.Daycent2(Temp)
```

Arguments

Temp	A scalar or vector containing values of soil temperature for which the effects on decomposition rates are calculated.
------	---

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Del Grosso, S. J., W. J. Parton, A. R. Mosier, E. A. Holland, E. Pendall, D. S. Schimel, and D. S. Ojima (2005), Modeling soil CO₂ emissions from ecosystems, *Biogeochemistry*, 73(1), 71-91.

Examples

```
Temperature=0:50
plot(Temperature, fT.Daycent2(Temperature), type="l",
     ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition rates according to the DAYCENT model")
```

fT.Demeter	<i>Effects of temperature on decomposition rates according to the DEMETER model</i>
------------	---

Description

Calculates the effects of temperature on decomposition rates according to the DEMETER model.

Usage

```
fT.Demeter(Temp, Q10 = 2)
```

Arguments

Temp	A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated
Q10	A scalar. Temperature coefficient Q10

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Foley, J. A. (1995), An equilibrium model of the terrestrial carbon budget, *Tellus B*, 47(3), 310-319.

Examples

```
Temperature=0:50  
plot(Temperature,fT.Demeter(Temperature),type="l",ylab="f(T) (unitless)",  
      main="Effects of temperature on decomposition rates according to the DEMETER model")
```

fT.KB	<i>Effects of temperature on decomposition rates according to a model proposed by M. Kirschbaum (1995)</i>
-------	--

Description

Calculates the effects of temperature on decomposition rates according to a model proposed by Kirschbaum (1995).

Usage

```
fT.KB(Temp)
```

Arguments

Temp	a scalar or vector containing values of soil temperature for which the effects on decomposition rates are calculated
------	--

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Kirschbaum, M. U. F. (1995), The temperature dependence of soil organic matter decomposition, and the effect of global warming on soil organic C storage, *Soil Biology and Biochemistry*, 27(6), 753-760.

Examples

```
Temperature=0:50
plot(Temperature,fT.KB(Temperature),type="l",ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition rates according to the Kirschbaum function")
```

fT.LandT

Effects of temperature on decomposition rates according to a function proposed by Lloyd and Taylor (1994)

Description

Calculates the effects of temperature on decomposition rates according to a function proposed by Lloyd and Taylor (1994).

Usage

```
fT.LandT(Temp)
```

Arguments

Temp	A scalar or vector containing values of soil temperature for which the effects on decomposition rates are calculated
------	--

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Lloyd, J., and J. A. Taylor (1994), On the Temperature Dependence of Soil Respiration, *Functional Ecology*, 8(3), 315-323.

Examples

```
Temperature=0:50
plot(Temperature,fT.LandT(Temperature),type="l",
     ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition
          rates according to the Lloyd and Taylor function")
```

fT.linear	<i>Effects of temperature on decomposition rates according to a linear model</i>
-----------	--

Description

Calculates the effects of temperature on decomposition rates according to a linear model.

Usage

```
fT.linear(Temp, a = 0.198306, b = 0.036337)
```

Arguments

Temp	A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated.
a	A scalar defining the intercept of the linear function.
b	A scalar defining the slope of the linear function.

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Adair, E. C., W. J. Parton, S. J. D. Grosse, W. L. Silver, M. E. Harmon, S. A. Hall, I. C. Burke, and S. C. Hart. 2008. Simple three-pool model accurately describes patterns of long-term litter decomposition in diverse climates. *Global Change Biology* 14:2636-2660.

Examples

```
Temperature=0:50
plot(Temperature, fT.linear(Temperature), type="l", ylab="f(T) (unitless)",
      main="Effects of temperature on decomposition rates according to a linear function")
```

fT.Q10	<i>Effects of temperature on decomposition rates according to a Q10 function</i>
--------	--

Description

Calculates the effects of temperature on decomposition rates according to the modified Van't Hoff function (Q10 function).

Usage

```
fT.Q10(Temp, k_ref = 1, T_ref = 10, Q10 = 2)
```

Arguments

Temp	A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated.
k_ref	A scalar representing the value of the decomposition rate at a reference temperature value.
T_ref	A scalar representing the reference temperature.
Q10	A scalar. Temperature coefficient Q10.

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
Temperature=0:50
plot(Temperature, fT.Q10(Temperature), type="l", ylab="f(T) (unitless)",
      main="Effects of temperature on decomposition rates according to a Q10 function")
lines(Temperature, fT.Q10(Temperature, Q10=2.2), col=2)
lines(Temperature, fT.Q10(Temperature, Q10=1.4), col=4)
legend("topleft", c("Q10 = 2", "Q10 = 2.2", "Q10 = 1.4"), lty=c(1,1,1), col=c(1,2,4), bty="n")
```

fT.RothC

Effects of temperature on decomposition rates according to the functions included in the RothC model

Description

Calculates the effects of temperature on decomposition rates according to the functions included in the RothC model.

Usage

```
fT.RothC(Temp)
```

Arguments

Temp A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated.

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Note

This function returns NA for Temp <= -18.3

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Jenkinson, D. S., S. P. S. Andrew, J. M. Lynch, M. J. Goss, and P. B. Tinker (1990), The Turnover of Organic Carbon and Nitrogen in Soil, Philosophical Transactions: Biological Sciences, 329(1255), 361-368.

Examples

```
Temperature=0:50  
plot(Temperature,fT.RothC(Temperature),type="l",ylab="f(T) (unitless)",  
      main="Effects of temperature on decomposition rates according to the RothC model")
```

fT.Standcarb	<i>Effects of temperature on decomposition rates according to the Stand-Carb model</i>
--------------	--

Description

Calculates the effects of temperature on decomposition rates according to the StandCarb model.

Usage

```
fT.Standcarb(Temp, Topt = 45, Tlag = 4, Tshape = 15, Q10 = 2)
```

Arguments

Temp	A scalar or vector containing values of temperature for which the effects on decomposition rates are calculated.
Topt	A scalar representing the optimum temperature for decomposition.
Tlag	A scalar that determines the lag of the response curve.
Tshape	A scalar that determines the shape of the response curve.
Q10	A scalar. Temperature coefficient Q10.

Value

A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Harmon, M. E., and J. B. Domingo (2001), A users guide to STANDCARB version 2.0: A model to simulate carbon stores in forest stands. Oregon State University, Corvallis.

Examples

```
Temperature=0:50
plot(Temperature,fT.Standcarb(Temperature),type="l",ylab="f(T) (unitless)",
     main="Effects of temperature on decomposition rates according to the StandCarb model")
```

fW.Candy

*Effects of moisture on decomposition rates according to the Candy model***Description**

Calculates the effects of water content and pore volume on decomposition rates.

Usage

```
fW.Candy(theta, PV)
```

Arguments

theta A scalar or vector containing values of volumetric soil water content.
 PV A scalar or vector containing values of pore volume.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

J. Bauer, M. Herbst, J.A. Huisman, L. Weihermüller, H. Vereecken. 2008. Sensitivity of simulated soil heterotrophic respiration to temperature and moisture reduction functions. *Geoderma*, Volume 145, Issues 1-2, 15 May 2008, Pages 17-27.

Examples

```
th=seq(0,1,0.01)
xi1=fW.Candy(theta=th,PV=0.4)
xi2=fW.Candy(theta=th,PV=0.6)
xi3=fW.Candy(theta=th,PV=0.8)
plot(th,xi1,type="l",main="Effects of soil water content and pore volume on decomposition rates",
      xlab="Volumetric soil water content (cm3 cm-3)",ylab=expression(xi))
lines(th,xi2,col=2)
lines(th,xi3,col=3)
legend("bottomright",c("Pore volume = 0.4", "Pore volume = 0.6", "Pore volume = 0.8"),lty=1,col=1:3)
```

fW.Century	<i>Effects of moisture on decomposition rates according to the CENTURY model</i>
------------	--

Description

Calculates the effects of precipitation and potential evapotranspiration on decomposition rates.

Usage

```
fW.Century(PPT, PET)
```

Arguments

PPT	A scalar or vector containing values of monthly precipitation.
PET	A scalar or vector containing values of potential evapotranspiration.

Value

A scalar or a vector containing the effects of precipitation and potential evapotranspiration on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Adair, E. C., W. J. Parton, S. J. D. Grosse, W. L. Silver, M. E. Harmon, S. A. Hall, I. C. Burke, and S. C. Hart (2008), Simple three-pool model accurately describes patterns of long-term litter decomposition in diverse climates, *Global Change Biology*, 14(11), 2636-2660. \ Parton, W. J., J. A. Morgan, R. H. Kelly, and D. S. Ojima (2001), Modeling soil C responses to environmental change in grassland systems, in *The potential of U.S. grazing lands to sequester carbon and mitigate the greenhouse effect*, edited by R. F. Follett, J. M. Kimble and R. Lal, pp. 371-398, Lewis Publishers, Boca Raton.

Examples

```
PPT=seq(0,1500,by=10)
PET=rep(1500,length(PPT))
PPT.PET=fW.Century(PPT,PET)
plot(PPT/PET,PPT.PET,
      ylab="f(PPT, PET) (unitless)",
      main="Effects of precipitation and potential evapotranspiration on decomposition rates")
```

fW.Daycent1	<i>Effects of moisture on decomposition rates according to the DAYCENT model</i>
-------------	--

Description

Calculates the effects of Soil Water Content on decomposition rates according to the Daycent Model.

Usage

```
fW.Daycent1(swc, a = 0.6, b = 1.27, c = 0.0012, d = 2.84, partd = 2.65,
            bulkd = 1, width = 1)
```

Arguments

swc	A scalar or vector with soil water content of a soil layer (cm).
a	Empirical coefficient. For fine textured soils a = 0.6. For coarse textured soils a = 0.55.
b	Empirical coefficient. For fine textured soils b = 1.27. For coarse textured soils b = 1.70.
c	Empirical coefficient. For fine textured soils c = 0.0012. For coarse textured soils c = -0.007.
d	Empirical coefficient. For fine textured soils d = 2.84. For coarse textured soils d = 3.22.
partd	Particle density of soil layer.
bulkd	Bulk density of soil layer (g/cm ³).
width	Thickness of a soil layer (cm).

Value

A data frame with values of water filled pore space (wfps) and effects of soil water content on decomposition rates. Both vectors are unitless.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Kelly, R. H., W. J. Parton, M. D. Hartman, L. K. Stretch, D. S. Ojima, and D. S. Schimel (2000), Intra-annual and interannual variability of ecosystem processes in shortgrass steppe, *J. Geophys. Res.*, 105.

Examples

```

swc=seq(0,0.8,by=0.01) # A sequence of values of soil water content
fine=fW.Daycent1(swc)
coarse=fW.Daycent1(swc,a=0.55,b=1.7,c=-0.007,d=3.22)

#This plot reproduces Figure 2b in Kelly et al. (2000)
plot(fine,type="l",xlim=c(0,1))
lines(coarse,lwd=2)
legend("topleft",c("coarse","fine"),lty=c(1,1),lwd=c(2,1),bty="n")

```

fW.Daycent2

Effects of moisture on decomposition rates according to the DAYCENT model

Description

Calculates the effects of volumetric water content on decomposition rates according to the Daycent/Century models.

Usage

```
fW.Daycent2(W, WP = 0, FC = 100)
```

Arguments

W	A scalar or vector of volumetric water content in percentage.
WP	A scalar representing the wilting point in percentage.
FC	A scalar representing the field capacity in percentage.

Value

A data frame with values of relative water content (RWC) and the effects of RWC on decomposition rates (fRWC).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Del Grosso, S. J., W. J. Parton, A. R. Mosier, E. A. Holland, E. Pendall, D. S. Schimel, and D. S. Ojima (2005), Modeling soil CO₂ emissions from ecosystems, *Biogeochemistry*, 73(1), 71-91.

Examples

```

W=10:90
fW=fW.Daycent2(W,WP=10,FC=90)
plot(fW,type="l",ylim=c(0,6)) #This plot reproduces Figure 1b, in del Grosso et al. (2005)

```

fW.Demeter	<i>Effects of moisture on decomposition rates according to the DEMETER model</i>
------------	--

Description

Calculates the effects of soil moisture on decomposition rates according to the DEMETER model.

Usage

```
fW.Demeter(M, Msat = 100)
```

Arguments

M	A scalar or vector containing values of soil moisture for which the effects on decomposition rates are calculated.
Msat	A scalar representing saturated soil moisture.

Value

A scalar or a vector containing the effects of moisture on decomposition rates (unitless).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Foley, J. A. (1995), An equilibrium model of the terrestrial carbon budget, *Tellus B*, 47(3), 310-319.

Examples

```
Moisture=0:100
plot(Moisture,fW.Demeter(Moisture),type="l",ylab="f(W) (unitless)",
     main="Effects of soil moisture on decomposition rates according to the DEMETER model")
```

fW.Gompertz	<i>Effects of moisture on decomposition rates according to the Gompertz function</i>
-------------	--

Description

Calculates the effects of water content on decomposition rates.

Usage

```
fW.Gompertz(theta, a = 0.824, b = 0.308)
```

Arguments

theta	A scalar or vector containing values of volumetric soil water content.
a	Empirical parameter
b	Empirical parameter

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

I. Janssens, S. Dore, D. Epron, H. Lankreijer, N. Buchmann, B. Longdoz, J. Brossaud, L. Montagnani. 2003. Climatic Influences on Seasonal and Spatial Differences in Soil CO₂ Efflux. In Valentini, R. (Ed.) Fluxes of Carbon, Water and Energy of European Forests. pp 235-253. Springer.

Examples

```
th=seq(0,1,0.01)
xi=fW.Gompertz(theta=th)
plot(th,xi,type="l",main="Effects of soil water content on decomposition rates",
      xlab="Volumetric soil water content (cm3 cm-3)",ylab=expression(xi))
```

fW.Moyano

Effects of moisture on decomposition rates according to the function proposed by Moyano et al. (2013)

Description

Calculates the effects of water content on decomposition rates.

Usage

```
fW.Moyano(theta, a = 3.11, b = 2.42)
```

Arguments

theta	A scalar or vector containing values of volumetric soil water content.
a	Empirical parameter
b	Empirical parameter

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

F. E. Moyano, S. Manzoni, C. Chenu. 2013 Responses of soil heterotrophic respiration to moisture availability: An exploration of processes and models. *Soil Biology and Biochemistry*, Volume 59, April 2013, Pages 72-85

Examples

```
th=seq(0,1,0.01)
xi=fW.Moyano(theta=th)
plot(th,xi,type="l",main="Effects of soil water content on decomposition rates",
      xlab="Volumetric soil water content (cm3 cm-3)",ylab=expression(xi))
```

fW.RothC

Effects of moisture on decomposition rates according to the RothC model

Description

Calculates the effects of moisture (precipitation and pan evaporation) on decomposition rates according to the RothC model.

Usage

```
fW.RothC(P, E, S.Thick = 23, pClay = 23.4, pE = 0.75, bare = FALSE)
```

Arguments

P	A vector with monthly precipitation (mm).
E	A vector with same length with open pan evaporation or evapotranspiration (mm).
S.Thick	Soil thickness in cm. Default for Rothamsted is 23 cm.
pClay	Percent clay.
pE	Evaporation coefficient. If open pan evaporation is used pE=0.75. If Potential evaporation is used, pE=1.0.
bare	Logical. Under bare soil conditions, bare=TRUE. Default is set under vegetated soil.

Value

A data.frame with accumulated top soil moisture deficit (Acc.TSMD) and the rate modifying factor b.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Coleman, K., and D. S. Jenkinson (1999), RothC-26.3 A model for the turnover of carbon in soil: model description and windows user guide (modified 2008), 47 pp, IACR Rothamsted, Harpenden.

Examples

```
P=c(74,59,62,51,52,57,34,55,58,56,75,71) #Monthly Precipitation (mm)
E=c(8,10,27,49,83,99,103,91,69,34,16,8) #Monthly open pan evaporation (mm)

Rothamsted=fW.RothC(P,E)
data.frame(month.name,P,E,0.75*E,P-0.75*E,Rothamsted)
# This reproduces Table 1 in the RothC documentation (Coleman and Jenkinson 1999)
```

fW.Skopp	<i>Effects of moisture on decomposition rates according to the function proposed by Skopp et al. 1990</i>
----------	---

Description

Calculates the effects of relative soil water content on decomposition rates.

Usage

```
fW.Skopp(rwc, alpha = 2, beta = 2, f = 1.3, g = 0.8)
```

Arguments

rwc	relative water content
alpha	Empirical parameter
beta	Empirical parameter
f	Empirical parameter
g	Empirical parameter

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

J. Skopp, M. D. Jawson, and J. W. Doran. 1990. Steady-state aerobic microbial activity as a function of soil water content. Soil Sci. Soc. Am. J., 54(6):1619-1625

Examples

```
th=seq(0,1,0.01)
xi=fW.Skopp(rwc=th)
plot(th,xi,type="l",main="Effects of soil water content on decomposition rates",
      xlab="Relative water content",ylab=expression(xi))
```

fW.Standcarb	<i>Effects of moisture on decomposition rates according to the StandCarb model</i>
--------------	--

Description

Calculates the effects of moisture on decomposition rates according to the StandCarb model.

Usage

```
fW.Standcarb(Moist, MatricShape = 5, MatricLag = 0, MoistMin = 30,
             MoistMax = 350, DiffuseShape = 15, DiffuseLag = 4)
```

Arguments

Moist	A scalar or vector containing values of moisture content of a litter or soil pool (%).
MatricShape	A scalar that determines when matric limit is reduced to the point that decay can begin to occur.
MatricLag	A scalar used to offset the curve to the left or right.
MoistMin	A scalar determining the minimum moisture content.
MoistMax	A scalar determining the maximum moisture content without diffusion limitations.
DiffuseShape	A scalar that determines the range of moisture contents where diffusion is not limiting.
DiffuseLag	A scalar used to shift the point when moisture begins to limit diffusion.

Value

A data frame with limitation due to water potential (MatricLimit), limitation due to oxygen diffusion (DiffuseLimit), and the overall limitation of moisture on decomposition rates (MoistDecayIndex).

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Harmon, M. E., and J. B. Domingo (2001), A users guide to STANDCARB version 2.0: A model to simulate carbon stores in forest stands. Oregon State University, Corvallis.

Examples

```

MC=0:500
DeadFoliage=fW.Standcarb(MC)
DeadBranch=fW.Standcarb(MC,MoistMax=200)
DeadWood=fW.Standcarb(MC,MoistMax=150)
StableSoil=fW.Standcarb(MC,MoistMin=15,MoistMax=100)
plot(MC,DeadFoliage$MoistDecayIndex,type="l",xlab="Moisture Content (%)",
     ylab="f(W) (unitless)",
     main="Effects of moisture on decomposition rates according to the StandCarb model")
lines(MC,DeadBranch$MoistDecayIndex,col=4)
lines(MC,DeadWood$MoistDecayIndex,col=3)
lines(MC,StableSoil$MoistDecayIndex,col=2)
legend("topright",c("Dead Foliage","Dead Branch","Dead Wood","Stable Soil"),
      lty=c(1,1,1),col=c(1,4,3,2),bty="n")

```

GaudinskiModel14

Implementation of a the six-pool C14 model proposed by Gaudinski et al. 2000

Description

This function creates a model as described in Gaudinski et al. 2000. It is a wrapper for the more general functions [GeneralModel_14](#) that can handle an arbitrary number of pools.

Usage

```

GaudinskiModel14(t, ks = c(kr = 1/1.5, koi = 1/1.5, koeal = 1/4,
  koeah = 1/80, kA1 = 1/3, kA2 = 1/75, kM = 1/110), C0 = c(FR0 = 390,
  C10 = 220, C20 = 390, C30 = 1370, C40 = 90, C50 = 1800, C60 = 560),
  F0_Delta14C = rep(0, 7), LI = 150, RI = 255, xi = 1, inputFc,
  lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)

```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 7 containing the decomposition rates for the 6 soil pools plus the fine-root pool.
C0	A vector of length 7 containing the initial amount of carbon for the 6 pools plus the fine-root pool.
F0_Delta14C	A vector of length 7 containing the initial amount of the radiocarbon fraction for the 7 pools as Delta14C values in per mil.
LI	A scalar or a data.frame object specifying the amount of litter inputs by time.
RI	A scalar or a data.frame object specifying the amount of root inputs by time.

<code>xi</code>	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
<code>inputFc</code>	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
<code>lambda</code>	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
<code>lag</code>	A positive integer representing a time lag for radiocarbon to enter the system.
<code>solver</code>	A function that solves the system of ODEs. This can be <code>euler</code> or any other user provided function with the same interface.
<code>pass</code>	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Gaudinski JB, Trumbore SE, Davidson EA, Zheng S (2000) Soil carbon cycling in a temperate forest: radiocarbon-based estimates of residence times, sequestration rates and partitioning fluxes. *Biogeochemistry* 51: 33-69

See Also

[ThreepParallelModel14](#), [ThreepFeedbackModel14](#)

Examples

```
years=seq(1901,2010,by=0.5)

Ex=GaudinskiModel14(
  t=years,
  ks=c(kr=1/3, koi=1/1.5, koeal=1/4, koeah=1/80, kA1=1/3, kA2=1/75, kM=1/110),
  inputFc=C14Atm_NH
)
R14m=getF14R(Ex)
C14m=getF14C(Ex)

plot(
  C14Atm_NH,
  type="l",
  xlab="Year",
  ylab=expression(paste(Delta^14,"C ", "(\u2030)")),
```

```

    xlim=c(1940,2010)
  )
  lines(years,C14m,col=4)
  points(HarvardForest14C02[1:11,1],HarvardForest14C02[1:11,2],pch=19,cex=0.5)
  points(HarvardForest14C02[12:173,1],HarvardForest14C02[12:173,2],pch=19,col=2,cex=0.5)
  points(HarvardForest14C02[158,1],HarvardForest14C02[158,2],pch=19,cex=0.5)
  lines(years,R14m,col=2)
  legend(
    "topright",
    c("Delta 14C Atmosphere",
      "Delta 14C SOM",
      "Delta 14C Respired"
    ),
    lty=c(1,1,1),
    col=c(1,4,2),
    bty="n"
  )
  ## We now show how to bypass soilR s parameter sanity check if necessary
  ## (e.g in for parameter estimation ) in functions
  ## which might call it with unreasonable parameters
  years=seq(1800,2010,by=0.5)
  Ex=GaudinskiModel14(
    t=years,
    ks=c(kr=1/3,koi=1/1.5,koeal=1/4,koeah=1/80,kA1=1/3,kA2=1/75,kM=1/110),
    inputFc=C14Atm_NH,
    pass=TRUE
  )

```

 GeneralModel

A general constructor

Description

Creates a Model object from different sources Have a look at the methods for details.

Usage

```
GeneralModel(t, A, ivList, inputFluxes, ...)
```

Arguments

```

t
A
ivList
inputFluxes
...

```

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

GeneralModel-methods *~~ Methods for Function GeneralModel ~~*

Description

~~ Methods for function GeneralModel ~~

Methods

signature(t = "numeric", A = "ANY", ivList = "numeric") [GeneralModel_method__numeric_ANY_numeric](#)

GeneralModel_14 *A general constructor*

Description

Creates a Model14 object from different sources Have a look at the methods for details.

Usage

```
GeneralModel_14(t, A, ivList, initialValF, inputFluxes, inputFc,
  Fc, di = -0.0001209681, lambda = -0.0001209681, solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE, ...)
```

Arguments

t	
A	
ivList	
initialValF	
inputFluxes	
inputFc	
Fc	
di	
lambda	
solverfunc	The function used by to actually solve the ODE system. This can be deSolve.lsoda.wrapper or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid
...	

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```

t_start=1960
t_end=2010
tn=220
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
n=3
At=new(Class="BoundLinDecompOp",
  t_start,
  t_end,
  function(t0){
    matrix(nrow=n,ncol=n,byrow=TRUE,
      c(-1, 0.1, 0,
        0.5, -0.4, 0,
        0, 0.2, -0.1)
    )
  }
)

c0=c(100, 100, 100)
F0=ConstFc(c(0,10,10),"Delta14C")
#constant inputrate
inputFluxes=new(
  "TimeMap",
  t_start,
  t_end,
  function(t0){matrix(nrow=n,ncol=1,c(10,10,10))}
)
# we have a dataframe representing the C_14 fraction
# note that the time unit is in years and the fraction is given in
# the Absolute Fraction Modern format.
# This means that all the other data provided are assumed to have the same value
# This is especially true for the decay constants to be specified later
inputFc=BoundFc(C14Atm_NH,format="Delta14C")
# add the C14 decay to the matrix which is done by a diagonal matrix which does not vary over time
# we assume a half life th=5730 years
th=5730
k=log(0.5)/th #note that k is negative and has the unit y^-1

mod=GeneralModel_14(t=t,A=At,ivList=c0,initialValF=F0,inputFluxes=inputFluxes,inputFc=inputFc,di=k)
#start plots
par(mfrow=c(3,2))
lt1=1; lt2=2; lt3=3
col1=1; col2=2; col3=3
# plot the C and C14 curves
Ct=getC(mod)
plot(t,Ct[,1],type="l",lty=lt1,col=col1,ylim=c(0,200),
  ylab="C stocks (arbitrary units)",xlab="Time")
lines(t,Ct[,2],type="l",lty=lt2,col=col2)
lines(t,Ct[,3],type="l",lty=lt3,col=col3,lwd=2)
legend(
  "topright",

```

```

    c("C in pool 1",
      "C in pool 2",
      "C in pool 3"
    ),
    lty=c(lt1,lt2,lt3),
    col=c(col1,col2,col3)
  )
  C14t=getC14(mod)
  plot(t,C14t[,1]/1000,type="l",lty=lt1,col=col1, ylim=c(0,100),
       ylab="14C stocks (arbitrary units)",xlab="Time")
  lines(t,C14t[,2]/1000,type="l",lty=lt2,col=col2)
  lines(t,C14t[,3]/1000,type="l",lty=lt3,col=col3)
  legend(
    "topright",
    c("14C in pool 1",
      "14C in pool 2",
      "14C in pool 3"
    ),
    lty=c(lt1,lt2,lt3),
    col=c(col1,col2,col3)
  )
#now plot the C14 Fraction in the atmosphere and compute the C14/C fraction of in the Soil

FC14=getF14(mod)
plot(C14Atm_NH, type="l",xlim=c(1960,2010))
lines(t,FC14[,1],lty=lt1,col=col1)
lines(t,FC14[,2],lt2,type="l",lty=lt2,col=col2)
lines(t,FC14[,3],type="l",lty=lt3,col=col3)
legend("topleft",c(
      expression(F[1]),
      expression(F[2]),
      expression(F[3])
    ),
    lty=c(lt1,lt2,lt3),col=c(col1,col2,col3))

#now compute the release flux
Rt=getReleaseFlux(mod)
plot(
  t,
  Rt[,1],
  type="l",
  lty=lt1,
  col=col1,
  ylab="C Release Flux (arbitrary units)",
  xlab="Time",
  ylim=c(0,50)
)
lines(t,Rt[,2],lt2,type="l",lty=lt2,col=col2)
lines(t,Rt[,3],type="l",lty=lt3,col=col3)
legend("topleft",c("RF1", "RF2", "RF3"),lty=c(lt1,lt2,lt3),col=c(col1,col2,col3))
#now compute the c14 release flux
R14t=getReleaseFlux14(mod)/1000

```

```

plot(
  t,
  R14t[,1],
  type="l",
  lty=lt1,
  col=col1,
  ylab="C14 Release Flux (arbitrary units)",
  xlab="Time"
)
lines(t,R14t[,2],lt2,type="l",lty=lt2,col=col2)
lines(t,R14t[,3],type="l",lty=lt3,col=col3)
legend("topleft",c(
  expression(RF[14]^1),
  expression(RF[14]^2),
  expression(RF[14]^3)
),
  ,lty=c(lt1,lt2,lt3),col=c(col1,col2,col3))

R14m=getF14R(mod)
C14m=getF14C(mod)
plot(C14Atm_NH, type="l",xlim=c(1960,2010),col=4)
lines(t,C14m)
lines(t,R14m,col=2)
legend(
  "topright",
  c("Atmosphere", "Mean SOM-14C", "Mean Release 14C"),
  lty=rep(1,3),
  col=c(4,1,2),
  bty="n"
)
par(mfrow=c(1,1))

```

GeneralModel_14-methods

~~ *Methods for Function* GeneralModel_14 ~~

Description

~~ Methods for function GeneralModel_14 ~~

Methods

```

signature(t = "numeric", A = "ANY", ivList = "numeric", initialValF = "ANY", inputFluxes = "ANY", i
  GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_missing_missing_missing_logical
signature(t = "numeric", A = "ANY", ivList = "numeric", initialValF = "ANY", inputFluxes = "ANY", i
  GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_function_logica
signature(t = "numeric", A = "ANY", ivList = "numeric", initialValF = "ANY", inputFluxes = "ANY", i
  GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_function_mis

```

64 *GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_missing_missing_missing_logical*

```
signature(t = "numeric", A = "ANY", ivList = "numeric", initialValF = "ANY", inputFluxes = "ANY", i
  GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_logical
signature(t = "numeric", A = "ANY", ivList = "numeric", initialValF = "ANY", inputFluxes = "ANY", i
  GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_missing
signature(t = "numeric", A = "ANY", ivList = "numeric", initialValF = "ANY", inputFluxes = "ANY", i
  GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_missing_ANY_numeric_missing_missing_missing
```

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_missing_missing_missing_logical
a constructor for class Model

Description

A wrapper for [Model_14](#) with pass=FALSE

Arguments

t	A vector containing the points in time where the solution is sought.
A	
ivList	
initialValF	
inputFluxes	
inputFc	
pass	

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_function_logical
a constructor for class Model_14

Description

This method tries to create a Model object from any combination of arguments that can be converted into the required set of building blocks for a model for n arbitrarily connected pools.

Arguments

t A vector containing the points in time where the solution is sought.
A
ivList
initialValF
inputFluxes
inputFc
di
solverfunc
pass

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

66GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_logical

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_function_missing
a constructor for class Model <- 14

Description

A wrapper for [Model_14](#) with pass=FALSE

Arguments

t
A
ivList
initialValF
inputFluxes
inputFc
di
solverfunc
pass

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_logical
a constructor for class Model_14

Description

A wrapper for [Model_14](#) with pass=FALSE

Arguments

t
A
ivList
initialValF
inputFluxes
inputFc
di
pass

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_missing67

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_ANY_missing_numeric_missing_missing_missing
a constructor for class Model

Description

A wrapper for [Model_14](#) with pass=FALSE

Arguments

t A vector containing the points in time where the solution is sought.
A
ivList
initialValF
inputFluxes
inputFc
di
pass

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

GeneralModel_14_method__numeric_ANY_numeric_ANY_ANY_missing_ANY_numeric_missing_missing_missing
a constructor for class Model

Description

A wrapper for [Model_14](#) with pass=FALSE

Arguments

t	A vector containing the points in time where the solution is sought.
A	
ivList	
initialValF	
inputFluxes	
Fc	
di	
pass	

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

GeneralModel_method__numeric_ANY_numeric
a constructor for class Model

Description

This method tries to create a Model object from any combination of arguments that can be converted into the required set of building blocks for a model for n arbitrarily connected pools.

Arguments

t	A vector containing the points in time where the solution is sought.
A	something that can be converted to any of the available DecompositionOperator classes
ivList	A vector containing the initial amount of carbon for the n pools. The length of this vector is equal to the number of pools and thus equal to the length of k. This is checked by an internal function.
inputFluxes	something that can be converted to any of the available InFlux classes
solverfunc	The function used by to actually solve the ODE system. This can be deSolve.lsoda.wrapper or any other user provided function with the same interface.
pass	Forces the constructor to create the model even if it is invalid

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

getAccumulatedRelease *Calculates the accumulated carbon release from the pools as a function of time*

Description

This function computes the accumulated carbon release of the given model as function of time.

Usage

```
getAccumulatedRelease(object)
```

Arguments

object	A Model object (e.g. of class Model or Model14) Have a look at the methods for details.
--------	---

Details

This function takes a Model object, calculates the release flux as specified by [getReleaseFlux](#), and integrates numerically the release flux up to each time in t.

Value

A $n \times m$ matrix of cumulative release fluxes with m columns representing the number of pools, and n rows representing the times as specified by the argument t in [GeneralModel](#) or other model creating functions.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

See examples in [Model](#), [GeneralModel](#), [GeneralModel_14](#), [TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#), etc.

getAccumulatedRelease-methods

~~ *Methods for Function getAccumulatedRelease* ~~

Description

~~ Methods for function getAccumulatedRelease ~~

Methods

signature(object = "Model") [getAccumulatedRelease_method_Model](#)

getAccumulatedRelease_method_Model

time integrals of release fluxes per pool

Description

The method integrates the release flux of every pool for all times in the interval specified by the model definition.

Arguments

object

Value

a matrix

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getC	<i>Calculates the C content of the pools</i>
------	--

Description

This function computes the carbon content of the pools as function of time. Have a look at the methods for details.

Usage

```
getC(object, as.closures = F)
```

Arguments

object some model object, the actual class depends on the method used.
as.closures if set to TRUE instead of a matrix a list of functions will be returned.

Details

This function takes a Model object, which represents a system of ODEs and solves the system for $C(t)$. The numerical solver used can be specified in the constructors of the Model classes e.g. [Model](#), [Model_14](#), [GeneralModel](#).

Value

A matrix with m columns representing the number of pools, and n rows representing the times as specified by the argument t in [GeneralModel](#) or another model creating function.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

See examples in [GeneralModel](#), [GeneralModel_14](#), [TwpParallelModel](#), [TwpSeriesModel](#), [TwpFeedbackModel](#), etc.

Examples

```
# This test function produces the example for testing the function getC
t=seq(0,10,by=0.1)
k=0.8
C0=100
In = 30

#Create two models with same arguments but calling two different model creation functions
Cmodel=OnepModel(t,k,C0,In)
C14model=OnepModel14(t,k,C0,In,F0=0,inputFc=IntCal09)
```

```
#getC can extract the amount of C from these two type of models
Ctmodel=getC(Cmodel)
Ctmodel14=getC(C14model)

#The output is identical because parameter values are the same
plot(t,Ctmodel,type="l")
lines(t,Ctmodel14,col=2,lty=2,lwd=2)
legend("topright",c("OnepModel output", "OnepModel14 output"),col=1:2,lty=1:2,bty="n")
```

getC-methods *~~ Methods for Function getC ~~*

Description

~~ Methods for function getC ~~

Methods

signature(object = "Model") [getC_method__Model](#)

getC14 *Calculates the mass of radiocarbon (14C fraction times C stock) in all pools*

Description

This function computes the mass of 14C (14C fraction times C stock) for all pools as function of time. Have a look at the methods for details.

Usage

```
getC14(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
getC14-methods      ~~ Methods for Function getC14 ~~
```

Description

~~ Methods for function getC14 ~~

Methods

```
signature(object = "Model_14") getC14\_method\_\_Model\_14
```

```
getC14_method__Model_14
      getC14 method Model 14
```

Description

This function computes the value for ^{14}C (mass or concentration) as function of time

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
getC_method__Model      getC method Model
```

Description

This function computes the value for C for each time and pool.

Arguments

object

Details

This function takes a Model object, which represents a system of ODEs of the form

$$\frac{d\mathbf{C}(t)}{dt} = \mathbf{I}(t) + \mathbf{A}(t)\mathbf{C}(t)$$

and solves the system for $\mathbf{C}(t)$. The numerical solver used can be specified in the constructor of the Model class e.g. [Model](#), [GeneralModel](#).

Value

A matrix with m columns representing the number of pools, and n rows representing the times as specified by the argument t in [Model](#), [GeneralModel](#) or another model creating function.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

See examples in [GeneralModel](#), [GeneralModel_14](#), [TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#), etc.

getDecompOp-methods *~~ Methods for Function getDecompOp ~~*

Description

All methods for function getDecompOp are intended for internal use inside the package only.

getDotOut-methods *~~ Methods for Function getDotOut ~~*

Description

All methods for function getDotOut are intended for internal use inside the package only.

getF14 *Calculates the 14C fraction of all pools*

Description

This function computes the radiocarbon fraction for all pools as function of time. Have a look at the methods for details.

Usage

```
getF14(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getF14-methods *~~ Methods for Function getF14 ~~*

Description

~~ Methods for function getF14 ~~

Methods

signature(object = "Model_14") [getF14_method_Model_14](#)

getF14C *Calculates the average radiocarbon fraction weighted by the mass of carbon*

Description

This function calculates the average radiocarbon fraction weighted by the mass of carbon at each time step Have a look at the methods for details.

Usage

```
getF14C(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getF14C-methods *~~ Methods for Function getF14C ~~*

Description

~~ Methods for function getF14C ~~

Methods

signature(object = "Model_14") [getF14C_method_Model_14](#)

`getF14C_method__Model_14`*read access to the models F14C variable*

Description

The model was created with a F14C object to describe the atmospheric 14C content. This method serves to investigate those settings from the model.

Arguments

object a Model_14 object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

`getF14R`*Calculates the average radiocarbon fraction weighted by the amount of carbon release*

Description

This function calculates the average radiocarbon fraction weighted by the amount of carbon release at each time step. Have a look at the methods for details.

Usage

```
getF14R(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
getF14R-methods      ~~ Methods for Function getF14R ~~
```

Description

```
~~ Methods for function getF14R ~~
```

Methods

```
signature(object = "Model_14") getF14R\_method\_\_Model\_14
```

```
getF14R_method__Model_14
      average radiocarbon fraction weighted by carbonrelease
```

Description

Calculates the average radiocarbon fraction weighted by the amount of carbon release at each time step. $\overline{F_R} = \frac{\sum_{i=1}^n {}^{14}R_i}{\sum_{i=1}^n R_i}$ Where ${}^{14}R_i(t)$ is the time dependent release of ${}^{14}C$ of pool i and $R_i(t)$ the release of all carbon isotopes of pool i . Since the result is always in Absolute Fraction Modern format we have to convert it to Delta14C

Arguments

```
object          an object
```

Value

A vector of length n with the value of $\overline{F_R}$ for each time step.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=TwopParallelModel14(t=years,ks=c(k1=1/2.8, k2=1/35),C0=c(200,5000),
      F0_Delta14C=c(0,0),In=LitterInput, gam=0.7,inputFc=C14Atm_NH,lag=2)

R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
```

```

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
legend("topright",c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2"),
      lty=c(1,1,1),col=c(1,4,4),lwd=c(1,1,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))

```

```
getF14_method__Model_14
```

radiocarbon

Description

Calculates the radiocarbon fraction for each pool at each time step.

Arguments

object

Value

A matrix of dimension $n \times m$; i.e. n time steps as rows and m pools as columns.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
getFormat-methods
```

~~ Methods for Function getFormat ~~

Description

~~ Methods for function getFormat ~~

Methods

signature(object = "BoundFc") [getFormat_method__BoundFc](#)

`getFormat_method__BoundFc`
extract the format string

Description

the function just yields the format as a string

Arguments

object object containing information about the format that could be Delta14C or AFM (Absolute Fraction Modern) for instance

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

`getFunctionDefinition` *getFunctionDefinition*

Description

Extracts the function definition (the R-function) from the argument Have a look at the methods for details.

Usage

`getFunctionDefinition(object)`

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getFunctionDefinition-methods

~~ *Methods for Function* getFunctionDefinition ~~

Description

~~ Methods for function getFunctionDefinition ~~

Methods

signature(object = "BoundFc") [getFunctionDefinition_method__BoundFc](#)

signature(object = "BoundInFlux") [getFunctionDefinition_method__BoundInFlux](#)

signature(object = "BoundLinDecompOp") [getFunctionDefinition_method__BoundLinDecompOp](#)

signature(object = "ConstLinDecompOp") [getFunctionDefinition_method__ConstLinDecompOp](#)

signature(object = "DecompositionOperator") [getFunctionDefinition_method__DecompositionOperator](#)

signature(object = "TimeMap") [getFunctionDefinition_method__TimeMap](#)

getFunctionDefinition_method__BoundFc

function definition

Description

extract the function definition (the R-function)

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getFunctionDefinition_method__BoundInFlux
getFunctionDefinition method BoundInFlux

Description

extract the function definition (the R-function) from the BoundInFlux

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getFunctionDefinition_method__BoundLinDecompOp
getFunctionDefinition method BoundLinDecompOp

Description

extract the function definition (the R-function)

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getFunctionDefinition_method__ConstLinDecompOp
creates a constant timedependent function and returns it

Description

The method creates a timedependent function from the existing matrix describing the operator

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getFunctionDefinition_method__DecompositionOperator
getFunctionDefinition method DecompositionOperator

Description

extract the function definition (the R-function)

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getFunctionDefinition_method__TimeMap
getFunctionDefinition method TimeMap

Description

extract the function definition (the R-function)

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getInFluxes-methods *~~ Methods for Function getInFluxes ~~*

Description

All methods for function getInFluxes are intended for internal use inside the package only.

 getInitialValues-methods

 ~~ *Methods for Function getInitialValues* ~~

Description

All methods for function `getInitialValues` are intended for internal use inside the package only.

 getMeanTransitTime *Access to the mean transit time*

Description

This generic function assembles methods to compute mean transit times. The nature of the results can change considerably depending on the arguments of the function. For an argument of class `model` it means something different than for an object of class `DecompOp`. To interpret them correctly refer also to the documentation of the methods.

Usage

```
getMeanTransitTime(object, inputDistribution)
```

Arguments

`object` a `DecompOp` Object.

`inputDistribution`

a vector of length equal to the number of pools. The entries are weights, which must sum to 1.

Details

The concept of mean transit time also known as mean residence time can be used to describe compartment models. In particular it is very frequently used to characterize linear time invariant compartment models in steady state. 1, 2, 3, 4. It is very important to note that `SoilR` is *not* limited to those. To integrate the concept into the more general context therefor requires some care. This starts with the definition. Assuming a time invariant system in steady state described by the (constant) distribution of inputs to the pools and constant decomposition and transfer rates one can define the mean transit time as the average time a package of carbon spends in the system from entry to exit. From `SoilR`'s general perspective this definition is ambiguous with regard to several points.

1. It does not take into account that the mean transfer time may change itself with time. For time invariant models in steady state this does not matter since the mean transit time turns out to be time invariant also but for a general model in `SoilR` input fluxes and decomposition coefficients can be time dependent and the system as a whole far from steady state.

2. It does not specify the set of particles contributing to the mean value. If the system is forever in a steady state it is possible to think of the average transit time of all particles but if the system changes with time such a definition would not be to usefull. To be able to compare time dependent models with real measurements the set of particles leaving the system at a certain point in time is a more natural choice

To incorporate the concept of transit times into SoilR we need to address these ambiguities. We also would like the new definition to agree with the old one in the special but often studied case of linear systems in steady state. We suggest the following Definition:

Given a system described by the complete history of inputs $\mathbf{I}(t)$ for $t \in (t_{start}, t_0)$ to all pools until time t_0 and the cumulative output $O(t_0)$ of all pools at time t_0 the mean transit time \bar{T}_{t_0} **of the system at time** t_0 is the average of the transit times of all particles leaving the system at time t_0

Remark:

For a system with several output channels one could define the mean transit time of particles leaving by this specific channel. Remark:

In future versions of SoilR it will be possible to compute a dynamic, time dependent mean transit time for objects of class `Model`. There is also a method that constructs a time invariant mean transit time by creting a time invariant model in steady state from an input flux distribution and a constant decomposition operators. This emphasizes that different methods for this function really answer different questions.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Manzoni, S., G.G. Katul, and A. Porporato. 2009. Analysis of soil carbon transit times and age distributions using network theories. *Journal of Geophysical Research-Biogeosciences* 114, DOI: 10.1029/2009JG001070.

Thompson, M.~V. and Randerson, J.~T.: Impulse response functions of terrestrial carbon cycle models: method and application, *Global Change Biology*, 5, 371–394, 10.1046/j.1365-2486.1999.00235.x, 1999.

Bolin, B. and Rodhe, H.: A note on the concepts of age distribution and transit time in natural reservoirs, *Tellus*, 25, 58–62, 1973.

Eriksson, E.: Compartment Models and Reservoir Theory, *Annual Review of Ecology and Systematics*, 2, 67–84, 1971.

getMeanTransitTime-methods

~~ *Methods for Function* getMeanTransitTime ~~

Description

~~ *Methods for function* getMeanTransitTime ~~

Methods

signature(object = "ConstLinDecompOp") [getMeanTransitTime_method__ConstLinDecompOp](#)

getMeanTransitTime_method__ConstLinDecompOp
compute the mean transit time

Description

This method computes the mean transit time for the linear time invariant system that can be constructed from the given operator and input distribution.

It relies on the method `getTransitTimeDistributionDensity` using the same arguments.

Arguments

object
inputDistribution

Details

To compute the mean transit time for the distribution we have to compute the integral

$$\bar{T} = \int_0^{\infty} T \cdot S_r \left(\frac{\vec{I}}{I}, 0, T \right) dT$$

for the numerically computed density. To avoid issues with numerical integration we don't use ∞ as upper limit but cut off the integration interval prematurely. For this purpose we calculate a maximum response time of the system as *Lasaga*

$$\tau_{cycle} = \frac{1}{|\min(\lambda_i)|}$$

where λ_i are non-zero eigenvalues of the matrix \mathbf{A} .

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Lasaga, A.: The kinetic treatment of geochemical cycles, *Geochimica et Cosmochimica Acta*, 44, 815 – 828, doi10.1016/0016-7037(80)90263-X, 1980.

getNumberOfPools-methods

~~ *Methods for Function* getNumberOfPools ~~

Description

All methods for function getNumberOfPools are intended for internal use inside the package only.

getOutputFluxes-methods

~~ *Methods for Function* getOutputFluxes ~~

Description

All methods for function getOutputFluxes are intended for internal use inside the package only.

getOutputReceivers-methods

~~ *Methods for Function* getOutputReceivers ~~

Description

All methods for function getOutputReceivers are intended for internal use inside the package only.

getParticleMonteCarloSimulator-methods

~~ *Methods for Function* getParticleMonteCarloSimulator ~~

Description

All methods for function getParticleMonteCarloSimulator are intended for internal use inside the package only.

getReleaseFlux	<i>Calculates the release of C from each pool</i>
----------------	---

Description

This function computes carbon release from each pool of the given model as function of time. Have a look at the methods for details.

Usage

```
getReleaseFlux(object)
```

Arguments

object An model object (the actual class depends on the method e.g. Model or Model14)

Details

This function takes a Model object, which represents a system of ODEs solves the system for $\mathbf{C}(t)$, calculates a diagonal matrix of release coefficients $\mathbf{R}(t)$, and computes the release flux as $\mathbf{R}(t)\mathbf{C}(t)$. The numerical solver used can be specified in the model creating functions like e.g. [Model](#).

Value

A matrix. Every column represents a pool and every row a point in time

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

See examples in [Model](#), [GeneralModel](#), [GeneralModel_14](#), [TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#), etc.

getReleaseFlux-methods

~~ Methods for Function getReleaseFlux ~~

Description

~~ Methods for function getReleaseFlux ~~

Methods

signature(object = "Model") [getReleaseFlux_method__Model](#)

getReleaseFlux14	<i>Calculates the mass of radiocarbon in the release flux (14C fraction times release flux)</i>
------------------	---

Description

This function computes the mass of radiocarbon in the release flux (14C fraction times release flux) as a function of time. Have a look at the methods for details.

Usage

```
getReleaseFlux14(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getReleaseFlux14-methods
~~ Methods for Function getReleaseFlux14 ~~

Description

~~ Methods for function getReleaseFlux14 ~~

Methods

signature(object = "Model_14") [getReleaseFlux14_method__Model_14](#)

```
getReleaseFlux14_method_Model_14
      14C respiration rate for all pools
```

Description

The function computes the ^{14}C release flux (mass per time) for all pools. Note that the respiration coefficients for ^{14}C do not change in comparison to the total C case. The fraction of ^{14}C lost by respiration is not greater for ^{14}C although the decay is faster due to the contribution of radioactivity.

Arguments

object an object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=TwopParallelModel14(
  t=years,
  ks=c(k1=1/2.8, k2=1/35),
  C0=c(200,5000),
  F0_Delta14C=c(0,0),
  In=LitterInput,
  gam=0.7,
  inputFc=C14Atm_NH,
  lag=2
)

R14=getReleaseFlux14(Ex)
C14=getC14(Ex)

par(mfrow=c(2,1))
plot(years, C14[,1], col=4,type="l",xlab="Year",ylab="14C(t)",ylim=c(min(C14),max(C14)))
lines(years, C14[,2],col=4,lwd=2)
legend("topright",c( "14C pool 1", "14C pool 2"),
      lty=c(1,1),col=c(1,4),lwd=c(1,1),bty="n")

plot(years, R14[,1], col=4,type="l",xlab="Year",ylab="14R(t)",ylim=c(min(R14),max(R14)))
lines(years, R14[,2],col=4,lwd=2)
legend("topright",c( "14C respiration pool 1", "14C respiration pool 2"),
      lty=c(1,1),col=c(1,4),lwd=c(1,1),bty="n")
par(mfrow=c(1,1))
```

 getReleaseFlux_method_Model

get the release rate for all pools

Description

The method computes the release of carbon per time for all points in time specified in the Model objects time slot.

Arguments

object an object of class Model created by a call to a constructor e.g. [Model](#), [GeneralModel](#) or other model creating functions.

Details

This function takes a Model object, which represents a system of ODEs

$$\frac{d\mathbf{C}(t)}{dt} = \mathbf{I}(t) + \mathbf{A}(t)\mathbf{C}(t)$$

solves the system for $\mathbf{C}(t)$, calculates the release coefficients $\mathbf{R}(t)$, and computes the release flux as $\mathbf{R}(t)\mathbf{C}(t)$. The numerical solver used can be specified in the model creating functions like e.g. [Model](#).

Value

A n x m matrix of release fluxes with m columns representing the number of pools, and n rows representing the time step as specified by the argument t in [Model](#), [GeneralModel](#) or another model creating function.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
# create the model
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)

Ex=TwoParallelModel(t,ks=c(k1=0.5,k2=0.2),C0=c(c10=100, c20=150),In=10,gam=0.7,xi=0.5)
# get the carbon stocks
Ct=getC(Ex)
# get the release rates
Rt=getReleaseFlux(Ex)
```

```

par(mfrow=c(2,1))
plot(t,rowSums(Ct),type="l",lwd=2,
      ylab="Carbon stocks (arbitrary units)",xlab="Time",ylim=c(0,sum(Ct[1,])))
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
legend("topright",c("Total C", "C in pool 1", "C in pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

plot(t,rowSums(Rt),type="l",ylab="Carbon released (arbitrary units)",
      xlab="Time",lwd=2,ylim=c(0,sum(Rt[1,])))
lines(t,Rt[,1],col=2)
lines(t,Rt[,2],col=4)
legend("topright",c("Total C release", "C release from pool 1", "C release from pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

par(mfrow=c(1,1))

```

```
getTimeRange
```

```
getTimeRange
```

Description

This function returns the time range of the given object. Have a look at the methods for details.

Usage

```
getTimeRange(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
getTimeRange-methods  ~~ Methods for Function getTimeRange ~~
```

Description

```
~~ Methods for function getTimeRange ~~
```

Methods

signature(object = "BoundFc") [getTimeRange_method__BoundFc](#)
signature(object = "BoundInFlux") [getTimeRange_method__BoundInFlux](#)
signature(object = "BoundLinDecompOp") [getTimeRange_method__BoundLinDecompOp](#)
signature(object = "ConstLinDecompOp") [getTimeRange_method__ConstLinDecompOp](#)
signature(object = "DecompositionOperator") [getTimeRange_method__DecompositionOperator](#)
signature(object = "TimeMap") [getTimeRange_method__TimeMap](#)

`getTimeRange_method__BoundFc`

ask for the boundaries of the underlying time interval

Description

The method returns the time range of the given object It is (probably mostly) used internally to make sure that time dependent functions retrieved from data are not used outside the interval where they are valid.

Arguments

object

Value

a vector of length two `c(t_min, t_max)` containing start and end time of the time interval for which the object has been defined.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

`getTimeRange_method__BoundInFlux`

time domain of the function

Description

The method returns a vector containing the start and end time where the interpolation is valid.

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getTimeRange_method__BoundLinDecompOp

ask for the boundaries of the underlying time interval

Description

The method returns the time range of the given object It is (probably mostly) used internally to make sure that time dependent functions retrieved from data are not used outside the interval where they are valid.

Arguments

object

Value

a vector of length two $c(t_{min}, t_{max})$ containing start and end time of the time interval for which the object has been defined.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getTimeRange_method__ConstLinDecompOp

return an (infinite) time range since the operator is constant

Description

some functions dealing with DecompOps in general rely on this so we have to implement it even though the timerange is always the same: (-inf,inf)

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getTimeRange_method__DecompositionOperator

ask for the boundaries of the underlying time interval

Description

The method returns the time range of the given object It is (probably mostly) used internally to make sure that time dependent functions retrieved from data are not used outside the interval where they are valid.

Arguments

object

Value

a vector of length two $c(t_{min}, t_{max})$ containing start and end time of the time interval for which the object has been defined.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getTimeRange_method__TimeMap

ask for the boundaries of the underlying time interval

Description

The method returns the time range of the given object It is probably mostly used internally to make sure that time dependent functions retrieved from data are not used outside the interval where they are valid.

Arguments

object An object of class TimeMap or one that inherits from TimeMap

Value

a vector of length two $c(t_{min}, t_{max})$ containing start and end time of the time interval for which the TimeMap object has been defined.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```

# set up some variables
t_start=0
t_end=10

# Create an explicit example of a TimeMap
c0=c(0.5, 0.5, 0.5)
inputFluxes=TimeMap.new(
  t_start,
  t_end,
  function(t0){matrix(nrow=n,ncol=1,c(0.0,0,0))}
)
# now it you only have the inputFluxes
# you can ask it for which time interval it was specified

print(getTimeRange(inputFluxes))

# Construct a less explicit example of a TimeMap with an object
# of class # BoundLinDecompOp which is subclass of TimeMap
n=3
At=new("BoundLinDecompOp",
  t_start,
  t_end,
  function(t0){
    matrix(nrow=n,ncol=n,byrow=TRUE,
      c(-0.2, 0, 0,
        0 , -0.3, 0,
        0, 0, -0.4/t0)
    )
  }
)
print(getTimeRange(At))

```

getTimes

Extracts the times argument

Description

This functions extracts the times argument from an argument of class NIModel

Usage

```
getTimes(object)
```

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getTimes-methods *~~ Methods for Function getTimes ~~*

Description

~~ Methods for function getTimes ~~

Methods

signature(object = "Model") [getTimes_method__Model](#)

getTimes_method__Model
getTimes method Model

Description

This functions extracts the times argument from an argument of class Model

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

getTransferCoefficients-methods
~~ Methods for Function getTransferCoefficients ~~

Description

All methods for function getTransferCoefficients are intended for internal use inside the package only.

 getTransferMatrix-methods

 ~~ Methods for Function getTransferMatrix ~~

Description

All methods for function `getTransferMatrix` are intended for internal use inside the package only.

getTransitTimeDistributionDensity

methods for transit time distributions

Description

According to [getMeanTransitTime](#) to we define the related density:

Usage

```
getTransitTimeDistributionDensity(object, inputDistribution,
  times)
```

Arguments

<code>object</code>	a protoDecompOp Object
<code>inputDistribution</code>	a vector of length equal to the number of pools. The entries are weights. That means that their sum must be equal to one!
<code>times</code>	the times for which the distribution density is sought

Details

Given a system described by the complete history of inputs $\mathbf{I}(t)$ for $t \in (t_{start}, t_0)$ to all pools until time t_0 and the cumulative output $O(t_0)$ of all pools at time t_0 the transit time density $\psi_{t_0}(T)$ **of the system at time t_0** is the probability density with respect to T implicitly defined by

$$\bar{T}_{t_0} = \int_0^{t-t_{start}} \psi_{t_0}(T) T dT$$

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Manzoni, S., G.G. Katul, and A. Porporato. 2009. Analysis of soil carbon transit times and age distributions using network theories. *Journal of Geophysical Research-Biogeosciences* 114, DOI: 10.1029/2009JG001070.

```
getTransitTimeDistributionDensity-methods
    ~~ Methods for Function getTransitTimeDistributionDensity
    ~~
```

Description

~~ Methods for function getTransitTimeDistributionDensity ~~

Methods

signature(object = "ConstLinDecompOp") [getTransitTimeDistributionDensity_method__ConstLinDecompOp](#)

```
getTransitTimeDistributionDensity_method__ConstLinDecompOp
    compute the TransitTimeDistributionDensity
```

Description

This method computes the probability density of the transit time of the linear time invariant system that can be constructed from the given operator and input distribution.

Arguments

object
inputDistribution

times

Details

In a forthcoming paper *SoilRv1.2* we derive the algorithm used in this implementation under the assumption of steady conditions having prevailed infinitely. We arrive at a formulation well known from the literature about time invariant linear systems, cited e.g. in *ManzoniJGR*.

The somehow amazing result is that the weight of the transit time density $\psi(T)$ for a *transit time* T for the steady state system is identical to the output $O(T)$ observed at time T of a *different* system which started with a normalized impulsive input $\frac{\vec{I}}{I}$ at time $T = 0$, where $I = \sum_{k=1}^m i_k$ is the cumulative input flux to all pools.

This fact simplifies the computation considerably. Translated into the language of an ode solver an impulsive input becomes a start vector $\frac{\vec{I}}{I}$ at time $T = 0$ and $O(T)$ the respiration related to the solution of the initial value problem observed at time T .

$$\psi(T) = S_r \left(\frac{\vec{I}}{I}, 0, T \right)$$

Note that from the perspective of the ode solver S_r depends on the decomposition operator and the distribution of the input among the pools only. It is therefore possible to implement the transit time distribution as a function of the decomposition operator and the fixed input flux distribution. To insure steady state conditions the decomposition operator is not allowed to be a true function of time. We therefore implement the method only for the subclass `ConstLinDecompOp`

Remark:

The decision to implement this method for `transitTimeDensity` especially for objects of class `ConstLinDecompOp` reflects the fact that an arbitrary model in SoilR is by no means bound to end up in steady state. To insure this we would have to ignore the input part of a user created model which would be confusing.

Remark:

In future versions of SoilR it will be possible to compute a dynamic, time dependent transit time distribution for objects of class `Model` with a time argument specifying for which time the distribution is sought. The steady state results computed here could then be reproduced with the user responsible for providing a model actually reaching it.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Manzoni, S., Katul, G.~G., and Porporato, A.: Analysis of soil carbon transit times and age distributions using network theories, *J. Geophys. Res.*, 114,

getValues-methods *~~ Methods for Function getValues ~~*

Description

All methods for function `getValues` are intended for internal use inside the package only.

HarvardForest14C02 *Delta14C in soil CO2 efflux from Harvard Forest*

Description

Measurements of Delta14C in soil CO2 efflux conducted at Harvard Forest, USA, between 1996 and 2010.

Usage

```
data(HarvardForest14C02)
```

Format

A data frame with the following 3 variables.

Year A numeric vector with the date of measurement in years

D14C A numeric vector with the value of the Delta 14C value measured in CO₂ efflux in per mil

Site A factor indicating the site where measurements were made. NWN: Northwest Near, Dry-down: Rainfall exclusion experiment.

Details

Samples for isotopic measurements of soil CO₂ efflux were collected from chambers that enclosed an air headspace in contact with the soil surface in the absence of vegetation using a closed dynamic chamber system to collect accumulated CO₂ in stainless steel traps with a molecular sieve inside. See Sierra et al. (2012) for additional details.

References

Sierra, C. A., Trumbore, S. E., Davidson, E. A., Frey, S. D., Savage, K. E., and Hopkins, F. M. 2012. Predicting decadal trends and transient responses of radiocarbon storage and fluxes in a temperate forest soil, *Biogeosciences*, 9, 3013-3028, doi:10.5194/bg-9-3013-2012

Examples

```
plot(HarvardForest14C02[,1:2])
```

Hua2013

Atmospheric radiocarbon for the period 1950-2010 from Hua et al. (2013)

Description

Atmospheric radiocarbon for the period 1950-2010 reported by Hua et al. (2013) for 5 atmospheric zones.

Usage

```
data(Hua2013)
```

Format

A [list](#) containing 5 data frames, each representing an atmospheric zone. The zones are: NHZone1: northern hemisphere zone 1, NHZone2: northern hemisphere zone 2, NHZone3: northern hemisphere zone 3, SHZone12: southern hemisphere zones 1 and 2, SHZone3: southern hemisphere zone 3. Each data frame contains a variable number of observations on the following 5 variables.

Year.AD Year AD

mean.Delta14C mean value of atmospheric radiocarbon reported as Delta14C

sd.Delta14C standard deviation of atmospheric radiocarbon reported as Delta14C
 mean.F14C mean value of atmospheric radiocarbon reported as fraction modern F14C
 sd.F14 standard deviation of atmospheric radiocarbon reported as fraction modern F14C

Details

This dataset corresponds to Table S3 from Hua et al. (2013). For additional details see the original publication.

Source

<https://journals.uair.arizona.edu/index.php/radiocarbon/article/view/16177>

References

Hua Q., M. Barbetti, A. Z. Rakowski. 2013. Atmospheric radiocarbon for the period 1950-2010. Radiocarbon 55(4):2059-2072.

Examples

```
plot(Hua2013$NHZone1$Year.AD, Hua2013$NHZone1$mean.Delta14C,
     type="l", xlab="Year AD", ylab=expression(paste(Delta^14, "C (\u2030)")))
lines(Hua2013$NHZone2$Year.AD, Hua2013$NHZone2$mean.Delta14C, col=2)
lines(Hua2013$NHZone3$Year.AD, Hua2013$NHZone3$mean.Delta14C, col=3)
lines(Hua2013$SHZone12$Year.AD, Hua2013$SHZone12$mean.Delta14C, col=4)
lines(Hua2013$SHZone3$Year.AD, Hua2013$SHZone3$mean.Delta14C, col=5)
legend(
  "topright",
  c(
    "Norther hemisphere zone 1",
    "Norther hemisphere zone 2",
    "Norther hemisphere zone 3",
    "Southern hemisphere zones 1 and 2",
    "Southern Hemispher zone 3"
  ),
  lty=1,
  col=1:5,
  bty="n"
)
```

Description

This function is an implementation of the Introductory Carbon Balance Model (ICBM). This is simply a two pool model connected in series.

Usage

```
ICBMModel(t, ks = c(k1 = 0.8, k2 = 0.00605), h = 0.13, r = 1.32,
           c0 = c(Y0 = 0.3, O0 = 3.96), In = 0, solver = deSolve.lsoda.wrapper,
           pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 2 with the decomposition rates for the young and the old pool.
h	Humification coefficient (transfer rate from young to old pool).
r	External (environmental or edaphic) factor.
c0	A vector of length 2 with the initial value of carbon stocks in the young and old pool.
In	Mean annual carbon input to the soil.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE forces the constructor to create the model even if it is invalid

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Andren, O. and T. Katterer. 1997. ICBM: The Introductory Carbon Balance Model for Exploration of Soil Carbon Balances. *Ecological Applications* 7:1226-1236.

See Also

[TwopSeriesModel](#)

Examples

```
# This example reproduces the simulations
# presented in Table 1 of Andren and Katterer (1997).
# First, the model is run for different values of the
# parameters representing different field experiments.
times=seq(0,20,by=0.1)
Bare=ICBMModel(t=times) #Bare fallow
pNpS=ICBMModel(t=times, h=0.125, r=1, c0=c(0.3,4.11), In=0.19+0.095) #+N +Straw
mNpS=ICBMModel(t=times, h=0.125, r=1.22, c0=c(0.3, 4.05), In=0.19+0.058) #-N +Straw
mNmS=ICBMModel(t=times, h=0.125, r=1.17, c0=c(0.3, 3.99), In=0.057) #-N -Straw
pNmS=ICBMModel(t=times, h=0.125, r=1.07, c0=c(0.3, 4.02), In=0.091) #+N -Straw
FM=ICBMModel(t=times, h=0.250, r=1.10, c0=c(0.3, 3.99), In=0.19+0.082) #Manure
SwS=ICBMModel(t=times, h=0.340, r=0.97, c0=c(0.3, 4.14), In=0.19+0.106) #Sewage Sludge
SS=ICBMModel(t=times, h=0.125, r=1.00, c0=c(0.25, 4.16), In=0.2) #Steady State

#The amount of carbon for each simulation is recovered with the function getC
```

```

CtBare=getC(Bare)
CtpNpS=getC(pNpS)
CtmNpS=getC(mNpS)
CtmNmS=getC(mNmS)
CtpNmS=getC(pNmS)
CtFM=getC(FM)
CtSwS=getC(SwS)
CtSS=getC(SS)

#This plot reproduces Figure 1 in Andren and Katterer (1997)
plot(times,
      rowSums(CtBare),
      type="l",
      ylim=c(0,8),
      xlim=c(0,20),
      ylab="Topsoil carbon mass (kg m-2)",
      xlab="Time (years)"
)
lines(times,rowSums(CtpNpS),lty=2)
lines(times,rowSums(CtmNpS),lty=3)
lines(times,rowSums(CtmNmS),lty=4)
lines(times,rowSums(CtpNmS),lwd=2)
lines(times,rowSums(CtFM),lty=2,lwd=2)
lines(times,rowSums(CtSwS),lty=3,lwd=2)
#lines(times,rowSums(CtSS),lty=4,lwd=2)
legend("topleft",
      c("Bare fallow",
        "+N +Straw",
        "-N +Straw",
        "-N -Straw",
        "+N -Straw",
        "Manure",
        "Sludge"
      ),
      lty=c(1,2,3,4,1,2,3),
      lwd=c(1,1,1,1,2,2,2),
      bty="n"
)

```

InFlux-class

Class "InFlux"

Description

NA

Methods

InFlux signature(object = "InFlux"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("InFlux")
```

InFlux-methods *~~ Methods for Function InFlux ~~*

Description

~~ Methods for function InFlux ~~

Methods

```
signature(object = "InFlux") InFlux\_method\_\_InFlux
signature(object = "TimeMap") InFlux\_method\_\_TimeMap
```

[InFlux_method__InFlux](#) *pass through constructor*

Description

This method handles the case that no actual construction is necessary since the argument is already of a subclass of InFlux

Arguments

object

Value

the unchanged argument

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

InFlux_method__TimeMap

create a BoundInFlux from a TimeMap object

Description

The method is used to ensure backward compatibility with the now deprecated TimeMap class. The resulting BoundInFlux is created by a call to the constructor BoundInFlux(object) of that class.

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

initialize-methods *~~ Methods for Function initialize ~~*

Description

~~ Methods for function initialize ~~

Methods

signature(.Object = "BoundFc") [initialize_method__BoundFc](#)

signature(.Object = "BoundInFlux") [initialize_method__BoundInFlux](#)

signature(.Object = "BoundLinDecompOp") [initialize_method__BoundLinDecompOp](#)

signature(.Object = "ConstLinDecompOp") [initialize_method__ConstLinDecompOp](#)

signature(.Object = "DecompositionOperator") [initialize_method__DecompositionOperator](#)

signature(.Object = "Model") [initialize_method__Model](#)

signature(.Object = "Model_14") [initialize_method__Model_14](#)

signature(.Object = "TimeMap") [initialize_method__TimeMap](#)

`initialize_method__BoundFc`
internal constructor (new)

Description

The function is probably called internally only but used by all other constructors. It calls a sanity check on its arguments and initializes the object.

Arguments

.Object
map
starttime
endtime
lag
format
interpolation

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

`initialize_method__BoundInFlux`
internal constructor

Description

This method is intended for internal use only, it may change with the internal representation of the class. In user code please use the generic constructor [BoundInFlux](#) instead.

Arguments

.Object
starttime
endtime
map
lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

initialize_method__BoundLinDecompOp
initialize called by (new)

Description

This is the internal constructor for objects of this class It is called by statements of the form `new("..)` which in turn is called by all other constructors which may have more convenient interfaces

Arguments

.Object
starttime
endtime
map
lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

initialize_method__ConstLinDecompOp
internal constructor

Description

This mehtod is intendet to be used internally. It may change in the future. For user code it is recommended to use one of the generic constructor `ConstLinDecompOp` instead.

Arguments

.Object
mat

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
initialize_method__DecompositionOperator
      initialize called by (new)
```

Description

This is the internal constructor for objects of the deprecated Class DecompositonOperator

Arguments

```
.Object
starttime
endtime
map
lag
```

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
initialize_method__Model
      internal constructor for Model objects
```

Description

Note that we encourage the use of more convenient constructors for the creation of model objects. Since this method is tightly coupled to the internal implementation of the class it is much more likely to change in the future than the other constructors, which can be kept stable much more easily in the future and are therefor encouraged for user code. This method implements R's initialize generic for objects of class Model. It is called whenever a new object of this class is created by a call to new with the first argument Model. It performs some sanity checks of its arguments and in case those tests pass returns an object of class Model. The checks can be turned off.(see arguments)

Arguments

```
.Object
times
mat          A decomposition Operator of some kind
initialValues
inputFluxes
solverfunc
pass
```

Details

Due to the mechanism of S4 object initialization (package "methods") new always calls initialize. (see the help pages for initialize and initialize-methods for details)

Value

an Object of class Model

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
require(RUnit)
# We present three possible scenarios:
# 1.) create an object from valid input
# 2.) try to build an Model object with unsound parameters and
#     show the savety net in action.
# 3.) force an unsound model to be created that would be rejected by default
# 4.) show some other insensible models being rejected
#
#1.) we first create a sensible model
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
A=new("BoundLinDecompOp",
     t_start,
     t_end,
     function(times){matrix(nrow=3,ncol=3,byrow=TRUE,
                           c(-1,  0,  0,
                             0.5, -2,  0,
                             0,  1, -0.5)
                           )
     }
)
I=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=1,byrow=TRUE,
          c(-1,  0,  0)
          )
  }
)
res=new("Model",t,mat=A,c(0,0,0),I)
print(class(res))
#2.)
# Now we present some examples where the constructor protests
# test.correctnessOfModel.impossibleCoefficients
```

```

t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)

A=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=3,byrow=TRUE,
           c(-1, 0, 0,
             1, -0.7, 0,
             0, 1, -0.5)
          )
  }
)
I=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=1,byrow=TRUE,
           c(-1, 0, 0)
          )
  }
)
checkException(
new("Model",t,A,c(0,0,0),I),
"correctnessOfModel should have returned FALSE
because the matrix values indicate unbiological
behavior (ruwsum should be smaller than zero),
but has not"
)
#3.)
# force it nevertheless
new("Model",t,A,c(0,0,0),I,pass=TRUE)

#4.) further examples
# test.correctnessOfModel.impossibleTimeRanges
mess="correctnessOfModel should have returned FALSE, but has not"
t_start=0
t_end=10
tdiff=t_end-t_start
tn=50
timestep=(tdiff)/tn
t=seq(t_start,t_end,timestep)

#we create an A(t) with sensible coefficients
#but where the time range begins to late

A=TimeMap.new(
  t_start+1/4*tdiff,
  t_end,

```

```

function(times){
  matrix(nrow=3,ncol=3,byrow=TRUE,
        c(-1,  0,  0,
          1, -0.7,  0,
          0,  0.5, -0.5)
  )
}
)
I=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=1,byrow=TRUE,
          c(-1,  0,  0)
    )
  }
)

checkException(new("Model",t,A,c(0,0,0),I),mess)
#now we do the same to the InFluxes(t) while A(t) is correct
A=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=3,byrow=TRUE,
          c(-1,  0,  0,
            1, -0.7,  0,
            0,  0.5, -0.5)
    )
  }
)
I=TimeMap.new(
  t_start+1/4*tdiff,
  t_end,
  function(times){
    matrix(nrow=3,ncol=1,byrow=TRUE,
          c(-1,  0,  0)
    )
  }
)
checkException(new("Model",t,A,c(0,0,0),I),mess)

#we create an A(t) with sensible coefficients
#but where the time range ends to early

A=TimeMap.new(
  t_start,
  t_end-1/4*tdiff,
  function(times){
    matrix(nrow=3,ncol=3,byrow=TRUE,
          c(-1,  0,  0,
            1, -0.7,  0,
            0,  0.5, -0.5)
    )
  }
)

```

```

    )
  }
)
I=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=1,byrow=TRUE,
           c(-1, 0, 0)
    )
  }
)
checkException(new("Model",t,A,c(0,0,0),I),mess)
#now we do the same to the InFluxes(t) while A(t) is correct
A=TimeMap.new(
  t_start,
  t_end,
  function(times){
    matrix(nrow=3,ncol=3,byrow=TRUE,
           c(-1, 0, 0,
             1, -0.7, 0,
             0, 0.5, -0.5)
    )
  }
)
I=TimeMap.new(
  t_start,
  t_end-1/4*tdiff,
  function(times){
    matrix(nrow=3,ncol=1,byrow=TRUE,
           c(-1, 0, 0)
    )
  }
)
checkException(new("Model",t,A,c(0,0,0),I),mess)

```

```
initialize_method__Model_14
```

An internal constructor for Model_14 objects not recommended to be used directly in user code.

Description

This method implements R's initialize generic for objects of class Model_14 and is not intended as part of the public interface to SoilR. It may change in the future as the classes implementing SoilR may. It is called whenever a new object of this class is created by a call to new with the first argument Model_14. It performs some sanity checks of its arguments and in case those tests pass returns an object of class Model_14 The checks can be turned off.(see arguments)

Arguments

.Object	the Model_14 object itself
times	The points in time where the solution is sought
mat	A decomposition Operator of some kind
initialValues	
initialValF	An object of class ConstFc containing a vector with the initial values of the radiocarbon fraction for each pool and a format string describing in which format the values are given.
inputFluxes	
c14Fraction	
c14DecayRate	
solverfunc	
pass	

Details

Due to the mechanism of S4 object initialization (package "methods") new always calls initialize. (see the help pages for initialize and initialize-methods for details)

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
require(RUnit)
# We present three possible scenarios:
# 1.) create an object from valid input
# 2.) try to build an Model_14 object with unsound parameters and
#    show the savety net in action.
# 3.) force an unsound model to be created that would be rejected by default

#1.)
# create the ingredients and assemble them to a Model_14 in the final step
t_start=1960
t_end=2010
tn=220
timestep=(t_end-t_start)/tn
## the time
t=seq(t_start,t_end,timestep)
## some Decomposition Operator
n=3
At=new(Class="BoundLinDecompOp",
       t_start,
       t_end,
       function(t0){
         matrix(nrow=n,ncol=n,byrow=TRUE,
               c(-1, 0.1, 0,
```

```

        0.5 , -0.4,  0,
        0,  0.2, -0.1)
    )
}
)

c0=c(100, 100, 100)

## Atmospheric C_14

F0=ConstFc(c(0,10,10),"Delta14C")

## constant inputrate
inputFluxes=new(
  "TimeMap",
  t_start,
  t_end,
  function(t0){matrix(nrow=n,ncol=1,c(10,10,10))}
)
# we have a dataframe representing the C_14 fraction
# note that the time unit is in years and the fraction is given in
# the Absolute Fraction Modern format.
# This means that all the other data provided are assumed to have the same value
# This is especially true for the decay constants to be specified later
Fc=BoundFc(C14Atm_NH,format="Delta14C")
# add the C14 decay to the matrix which is done by a diagonal
# matrix which does not vary over time
# we assume a half life th=5730 years
th=5730
k=log(0.5)/th #note that k is negative and has the unit y^-1

mod=new("Model_14",t,At,c0,F0,inputFluxes,Fc,k)

#2.) provoke failure by demanding extrapolation to times where
# the model ingrediens are not specified (10 years later than the input e.g)
t_toLong=seq(t_start,t_end+10,timestep)
checkException(new("Model_14",t_toLong,At,c0,F0,inputFluxes,Fc,k),
  "initialize must throw an exception because it is asked to build
  an invalid model"
)
#3.) force an unsound model by settign pass to TRUE
new("Model_14",t_toLong,At,c0,F0,inputFluxes,Fc,k,pass=TRUE)

```

```
initialize_method__TimeMap
```

initialize called by (new)

Description

This is the standard constructor for objects of this class It is called by statements of the form `new("TimeMap", start, end, f, lag)`

Arguments

.Object
starttime
endtime
map
lag

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

IntCal09

Northern Hemisphere atmospheric radiocarbon for the pre-bomb period

Description

Northern Hemisphere atmospheric radiocarbon calibration curve for the period 0 to 50,000 yr BP.

Usage

```
data(IntCal09)
```

Format

A data frame with 3522 observations on the following 5 variables.

CAL.BP Calibrated age in years Before Present (BP).

C14.age C14 age in years BP.

Error Error estimate for C14.age.

Delta.14C Delta.14C value in per mil.

Sigma Standard deviation of Delta.14C in per mil.

Details

Delta.14C is age-corrected as per Stuiver and Polach (1977). All details about the derivation of this dataset are provided in Reimer et al. (2009).

Source

<http://www.radiocarbon.org/IntCal09%20files/intcal09.14c>

References

P. Reimer, M. Baillie, E. Bard, A. Bayliss, J. Beck, P. Blackwell, C. Ramsey, C. Buck, G. Burr, R. Edwards, et al. 2009. IntCal09 and Marine09 radiocarbon age calibration curves, 0 - 50,000 years cal bp. Radiocarbon, 51(4):1111 - 1150.

M. Stuiver and H. A. Polach. 1977. Reporting of C-14 data. Radiocarbon, 19(3):355 - 363.

Examples

```
par(mfrow=c(2,1))
plot(IntCal09$CAL.BP, IntCal09$C14.age, type="l")
polygon(x=c(IntCal09$CAL.BP, rev(IntCal09$CAL.BP)),
y=c(IntCal09$C14.age+IntCal09$Error, rev(IntCal09$C14.age-IntCal09$Error)),
col="gray", border=NA)
lines(IntCal09$CAL.BP, IntCal09$C14.age)

plot(IntCal09$CAL.BP, IntCal09$Delta.14C, type="l")
polygon(x=c(IntCal09$CAL.BP, rev(IntCal09$CAL.BP)),
y=c(IntCal09$Delta.14C+IntCal09$Sigma, rev(IntCal09$Delta.14C-IntCal09$Sigma)),
col="gray", border=NA)
lines(IntCal09$CAL.BP, IntCal09$Delta.14C)
par(mfrow=c(1,1))
```

IntCal13

Atmospheric radiocarbon for the 0-50,000 yr BP period

Description

Atmospheric radiocarbon calibration curve for the period 0 to 50,000 yr BP. This is the most recent update to the internationally agreed calibration curve and supersedes [IntCal09](#).

Usage

```
data(IntCal13)
```

Format

A data frame with 5140 observations on the following 5 variables.

CAL.BP Calibrated age in years Before Present (BP).

C14.age C14 age in years BP.

Error Error estimate for C14.age.

Delta.14C Delta.14C value in per mil.

Sigma Standard deviation of Delta.14C in per mil.

Details

Delta.14C is age-corrected as per Stuiver and Polach (1977). All details about the derivation of this dataset are provided in Reimer et al. (2013).

Source

<http://www.radiocarbon.org/IntCal13%20files/intcal13.14c>

References

Reimer PJ, Bard E, Bayliss A, Beck JW, Blackwell PG, Bronk Ramsey C, Buck CE, Cheng H, Edwards RL, Friedrich M, Grootes PM, Guilderson TP, Hafliðason H, Hajdas I, Hatte C, Heaton TJ, Hogg AG, Hughen KA, Kaiser KF, Kromer B, Manning SW, Niu M, Reimer RW, Richards DA, Scott EM, Southon JR, Turney CSM, van der Plicht J. 2013. IntCal13 and MARINE13 radiocarbon age calibration curves 0-50000 years calBP. Radiocarbon 55(4): 1869-1887. DOI: 10.2458/azu_js_rc.55.16947

M. Stuiver and H. A. Polach. 1977. Reporting of C-14 data. Radiocarbon, 19(3):355 - 363.

Examples

```
plot(IntCal13$CAL.BP, IntCal13$C14.age-IntCal13$Error, type="l", col=2,
     xlab="cal BP", ylab="14C BP")
lines(IntCal13$CAL.BP, IntCal13$C14.age+IntCal13$Error, col=2)

plot(IntCal13$CAL.BP, IntCal13$Delta.14C+IntCal13$Sigma, type="l", col=2,
     xlab="cal BP", ylab="Delta14C")
lines(IntCal13$CAL.BP, IntCal13$Delta.14C-IntCal13$Sigma, col=2)
```

linesCPool	<i>Add lines with the output of getC14, getC, or getReleaseFlux to an existing plot</i>
------------	---

Description

This function adds lines to a plot with the C content, the C release, or Delta 14C value of each pool over time. Needs as input a matrix obtained after a call to [getC14](#), [getC](#), or [getReleaseFlux](#).

Usage

```
linesCPool(t, mat, col, ...)
```

Arguments

t	A vector containing the time points for plotting.
mat	A matrix object obtained after a call to getC14 , getC , or getReleaseFlux .
col	A color palette specifying color lines for each pool (columns of mat).
...	Other arguments passed to plot.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=ThreepFeedbackModel14(t=years,ks=c(k1=1/2.8, k2=1/35, k3=1/100),
                          C0=c(200,5000,500),F0=c(0,0,0), In=LitterInput,
                          a21=0.1,a12=0.01,a32=0.005,a23=0.001,inputFc=C14Atm_NH)

Ct=getC(Ex)

pal=rainbow(3)
plotCPool(t=years,mat=Ct,col=pal,xlab="Time (yrs)",
          ylab="Carbon stocks",ylim=c(min(Ct),max(Ct)))

LitterInput2=350

Ex2=ThreepFeedbackModel14(t=years,ks=c(k1=1/2.8, k2=1/35, k3=1/100),
                          C0=c(200,5000,500),F0=c(0,0,0), In=LitterInput2,
                          a21=0.1,a12=0.01,a32=0.005,a23=0.001,inputFc=C14Atm_NH)

Ct2=getC(Ex2)

linesCPool(t=years,mat=Ct2,col=pal,lwd=2)
```

Model

A general constructor

Description

Creates a Model object from different sources Have a look at the methods for details.

Usage

```
Model(t, A, ivList, inputFluxes, ...)
```

Arguments

t
A
ivList
inputFluxes
...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Model-class

Class "Model"

Description

serves as a fence to the interface of SoilR functions. So that later implementations can differ

Slots

times: Object of class "numeric" ~~
mat: Object of class "DecompOp" ~~
initialValues: Object of class "numeric" ~~
inputFluxes: Object of class "InFlux" ~~
solverfunc: Object of class "function" ~~

Methods

[signature(x = "Model", i = "character"): ...
getAccumulatedRelease signature(object = "Model"): ...
getC signature(object = "Model"): ...
getReleaseFlux signature(object = "Model"): ...
getTimes signature(object = "Model"): ...
initialize signature(.Object = "Model"): ...
plot signature(x = "Model"): ...
print signature(x = "Model"): ...
show signature(object = "Model"): ...
summary signature(object = "Model"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("Model")
```

Model-methods

~~ *Methods for Function Model* ~~

Description

~~ Methods for function Model ~~

Methods

signature(t = "numeric", A = "ANY", ivList = "numeric") [Model_method__numeric_ANY_numeric](#)

Model_14

A general constructor

Description

Creates a Model_14 object from different sources Have a look at the methods for details.

Usage

```
Model_14(t, A, ivList, initialValF, inputFluxes, inputFc, c14DecayRate,  
         solverfunc, pass)
```

Arguments

t
A
ivList
initialValF
inputFluxes
inputFc
c14DecayRate
solverfunc
pass

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Model_14-class	Class "Model_14"
----------------	------------------

Description

defines a representation of a 14C model

Slots

c14Fraction: Object of class "BoundFc" ~~
c14DecayRate: Object of class "numeric" ~~
initialValF: Object of class "ConstFc" ~~
times: Object of class "numeric" ~~
mat: Object of class "DecompOp" ~~
initialValues: Object of class "numeric" ~~
inputFluxes: Object of class "InFlux" ~~
solverfunc: Object of class "function" ~~

Extends

Class "[Model](#)", directly.

Methods

getC14 signature(object = "Model_14"): ...
getF14 signature(object = "Model_14"): ...
getF14C signature(object = "Model_14"): ...
getF14R signature(object = "Model_14"): ...
getReleaseFlux14 signature(object = "Model_14"): ...
initialize signature(.Object = "Model_14"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("Model_14")
```

Model_14-methods *~~ Methods for Function Model_14 ~~*

Description

All methods for function Model_14 are intended for internal use inside the package only.

Model_method__numeric_ANY_numeric
general constructor for class Model

Description

This method tries to create a Model object from any combination of arguments that can be converted into the required set of building blocks for a model for n arbitrarily connected pools.

Arguments

t	A vector containing the points in time where the solution is sought.
A	something that can be converted to any of the available DecompOp classes
ivList	A vector containing the initial amount of carbon for the n pools. The length of this vector is equal to the number of pools and thus equal to the length of k. This is checked by an internal function.
inputFluxes	something that can be converted to any of the available InFlux classes
solverfunc	The function used by to actually solve the ODE system. This can be deSolve.lsoda.wrapper or any other user provided function with the same interface.
pass	Forces the constructor to create the model even if it is invalid

Value

A model object that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel](#), [TwopSeriesModel](#), [TwopFeedbackModel](#)

`O nepModel`*Implementation of a one pool model*

Description

This function creates a model for one pool. It is a wrapper for the more general function [GeneralModel](#).

Usage

```
O nepModel(t, k, C0, In, xi = 1, solver = deSolve::lsoda.wrapper,  
           pass = FALSE)
```

Arguments

<code>t</code>	A vector containing the points in time where the solution is sought.
<code>k</code>	A scalar with the decomposition rate of the pool.
<code>C0</code>	A scalar containing the initial amount of carbon in the pool.
<code>In</code>	A scalar or a data.frame object specifying the amount of litter inputs by time.
<code>xi</code>	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
<code>solver</code>	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
<code>pass</code>	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. *Geoscientific Model Development* 5, 1045-1060.

See Also

[TwopParallelModel](#), [TwopFeedbackModel](#)

Examples

```
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
k=0.8
C0=100
In = 30
```

```
Ex=OnepModel(t,k,C0,In)
Ct=getC(Ex)
Rt=getReleaseFlux(Ex)
Rc=getAccumulatedRelease(Ex)
```

```
plot(
  t,
  Ct,
  type="l",
  ylab="Carbon stocks (arbitrary units)",
  xlab="Time (arbitrary units)",
  lwd=2
)
```

```
plot(
  t,
  Rt,
  type="l",
  ylab="Carbon released (arbitrary units)",
  xlab="Time (arbitrary units)",
  lwd=2
)
```

```
plot(
  t,
  Rc,
  type="l",
  ylab="Cumulative carbon released (arbitrary units)",
  xlab="Time (arbitrary units)",
  lwd=2
)
```

OnepModel14

Implementation of a one-pool C14 model

Description

This function creates a model for one pool. It is a wrapper for the more general function [GeneralModel_14](#).

Usage

```
OnepModel14(t, k, C0, F0_Delta14C, In, xi = 1, inputFc, lambda = -0.0001209681,
            lag = 0, solver = deSolve.lsoda.wrapper, pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
k	A scalar with the decomposition rate of the pool.
C0	A scalar containing the initial amount of carbon in the pool.
F0_Delta14C	A scalar containing the initial amount of the radiocarbon fraction in the pool in Delta_14C format.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object consisting of a function describing the fraction of C_14 in per mille. The first column will be assumed to contain the times.
lambda	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
lag	A (positive) scalar representing a time lag for radiocarbon to enter the system.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[OnepModel](#), [TwopParallelModel14](#), [TwopFeedbackModel14](#)

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700
```

```
Ex=OnepModel14(t=years,k=1/10,C0=500, F0=0,In=LitterInput, inputFc=C14Atm_NH)
C14t=getF14(Ex)
```

```

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
legend(
  "topright",
  c("Delta 14C Atmosphere", "Delta 14C in SOM"),
  lty=c(1,1),
  col=c(1,4),
  lwd=c(1,1),
  bty="n"
)

```

ParallelModel

ParallelModel

Description

This function creates a numerical model for n independent (parallel) pools that can be queried afterwards. It is used by the convenient wrapper functions [TwopParallelModel](#) and [ThreepParallelModel](#) but can also be used independently.

Usage

```

ParallelModel(times,
  coeffs_tm, startvalues, inputrates, solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE)

```

Arguments

<code>times</code>	A vector containing the points in time where the solution is sought.
<code>coeffs_tm</code>	A <code>TimeMap</code> object consisting of a vector valued function containing the decay rates for the n pools as function of time and the time range where this function is valid. The length of the vector is equal to the number of pools.
<code>startvalues</code>	A vector containing the initial amount of carbon for the n pools. «The length of this vector is equal to the number of pools and thus equal to the length of k . This is checked by the function.
<code>inputrates</code>	An object consisting of a vector valued function describing the inputs to the pools as functions of time TimeMap.new
<code>solverfunc</code>	The function used to actually solve the ODE system. This can be deSolve.lsoda.wrapper or any other user provided function with the same interface.
<code>pass</code>	if TRUE forces the constructor to create the model even if it is invalid

Value

a model object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```

t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
k=TimeMap.new(t_start,t_end,function(times){c(-0.5,-0.2,-0.3)})
c0=c(1, 2, 3)
#constant inputrates
inputrates=BoundInFlux(
  function(t){matrix(nrow=3,ncol=1,c(1,1,1))},
  t_start,
  t_end
)
mod=ParallelModel(t,k,c0,inputrates)
Y=getC(mod)
lt1=1 ;lt2=2 ;lt3=3
col1=1; col2=2; col3=3
plot(t,Y[,1],type="l",lty=lt1,col=col1,
ylab="C stocks",xlab="Time")
  lines(t,Y[,2],type="l",lty=lt2,col=col2)
  lines(t,Y[,3],type="l",lty=lt3,col=col3)
  legend(
"topleft",
c("C in pool 1",
"C in 2",
"C in pool 3"
),
lty=c(lt1,lt2,lt3),
col=c(col1,col2,col3)
)
Y=getAccumulatedRelease(mod)
plot(t,Y[,1],type="l",lty=lt1,col=col1,ylab="C release",xlab="Time")
  lines(t,Y[,2],lty=lt2,col=col2)
  lines(t,Y[,3],type="l",lty=lt3,col=col3)
  legend("topright",c("R1","R2","R3"),lty=c(lt1,lt2,lt3),col=c(col1,col2,col3))

```

plot-methods

~~ *Methods for Function plot* ~~

Description

~~ *Methods for function plot* ~~

Methods

signature(x = "Model") [plot_method__Model](#)

plotC14Pool *Plots the output of [getC14](#) for each pool over time*

Description

This function produces a plot with the Delta14C in the atmosphere and the Delta14C of each pool obtained after a call to [getC14](#).

Usage

```
plotC14Pool(t, mat, inputFc, col, ...)
```

Arguments

t	A vector containing the time points for plotting.
mat	A matrix object obtained after a call to getC14
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
col	A color palette specifying color lines for each pool (columns of mat).
...	Other arguments passed to plot.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=ThreepFeedbackModel14(
  t=years,ks=c(k1=1/2.8, k2=1/35, k3=1/100),
  C0=c(200,5000,500), F0=c(0,0,0),
  In=LitterInput, a21=0.1,a12=0.01,a32=0.005,a23=0.001,inputFc=C14Atm_NH
)
C14t=getF14(Ex)

pal=rainbow(3)
plotC14Pool(
  t=years,mat=C14t,inputFc=C14Atm_NH,
  col=pal,xlab="Time (yrs)",
  ylab="Delta 14C (per mil)",
  xlim=c(1950,2000)
)
```

plotCPool	<i>Plots the output of getC or getReleaseFlux for each pool over time</i>
-----------	---

Description

This function produces a plot with the C content or released C for each pool over time. Needs as input a matrix obtained after a call to [getC](#) or [getReleaseFlux](#).

Usage

```
plotCPool(t, mat, col, ...)
```

Arguments

t	A vector containing the time points for plotting.
mat	A matrix object obtained after a call to getC or getReleaseFlux
col	A color palette specifying color lines for each pool (columns of mat).
...	Other arguments passed to <code>link{plot}</code> .

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=ThreepFeedbackModel14(
  t=years,ks=c(k1=1/2.8, k2=1/35, k3=1/100),C0=c(200,5000,500),
  F0=c(0,0,0), In=LitterInput, a21=0.1,a12=0.01,a32=0.005,a23=0.001,
  inputFc=C14Atm_NH
)
Ct=getC(Ex)
Rt=getReleaseFlux(Ex)

pal=rainbow(3)
plotCPool(t=years,mat=Ct,col=pal,xlab="Time (yrs)",
          ylab="Carbon stocks",ylim=c(min(Ct),max(Ct)))
```

plot_method__Model *plot method Model*

Description

This function is a stub

Arguments

x

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

print-methods *~~ Methods for Function print ~~*

Description

~~ Methods for function print ~~

Methods

signature(x = "Model") [print_method__Model](#)

print_method__Model *print method Model*

Description

This function is a stub

Arguments

x

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

RespirationCoefficients

helper function to compute respiration coefficients

Description

This function computes the respiration coefficients as function of time for all pools according to the given matrix A

Usage

RespirationCoefficients(A)

Arguments

A A matrix valued function representing the model.

Value

A vector valued function of time containing the respiration coefficients for all pools.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

RothCModel

Implementation of the RothCModel

Description

This function implements the RothC model of Jenkinson et al. It is a wrapper for the more general function [GeneralModel](#).

Usage

```
RothCModel(t, ks = c(k.DPM = 10, k.RPM = 0.3, k.BIO = 0.66, k.HUM = 0.02,
  k.IOM = 0), C0 = c(0, 0, 0, 0, 2.7), In = 1.7, DR = 1.44,
  clay = 23.4, xi = 1, solver = deSolve.lsoda.wrapper, pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 5 containing the values of the decomposition rates for the different pools
C0	A vector of length 5 containing the initial amount of carbon for the 5 pools.
In	A scalar or data.frame object specifying the amount of litter inputs by time.
DR	A scalar representing the ratio of decomposable plant material to resistant plant material (DPM/RPM).
clay	Percent clay in mineral soil.
xi	A scalar or data.frame object specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Jenkinson, D. S., S. P. S. Andrew, J. M. Lynch, M. J. Goss, and P. B. Tinker. 1990. The Turnover of Organic Carbon and Nitrogen in Soil. *Philosophical Transactions: Biological Sciences* 329:361-368. Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. *Geoscientific Model Development* 5, 1045-1060.

See Also

[ICBMModel](#)

Examples

```
t=0:500
Ex=RothCModel(t)
Ct=getC(Ex)
Rt=getReleaseFlux(Ex)

plot(
  t,
  Ct[,1],
  type="l",
  col=1,
  ylim=c(0,25),
  ylab=expression(paste("Carbon stores (Mg C", ha^-1,")")),
```

```

      xlab="Time (years)",
      lwd=2
    )
lines(t,Ct[,2],col=2,lwd=2,lty=2)
lines(t,Ct[,3],col=3,lwd=2,lty=3)
lines(t,Ct[,4],col=4,lwd=2,lty=4)
lines(t,Ct[,5],col=5,lwd=2,lty=5)
lines(t,rowSums(Ct),lwd=2)
legend(
  "topright",
  c(
    "Pool 1, DPM",
    "Pool 2, RPM",
    "Pool 3, BIO",
    "Pool 4, HUM",
    "Pool 5, IOM",
    "Total Carbon"
  ),
  lty=c(1:5,1),
  lwd=rep(2,5),
  col=c(1,2,3,4,5,"black")
  ,bty="n"
)

plot(t,Rt[,1],type="l",ylim=c(0,2),ylab="Respiration (Mg C ha-1 yr-1)",xlab="Time")
lines(t,Rt[,2],col=2)
lines(t,Rt[,3],col=3)
lines(t,Rt[,4],col=4)
lines(t,Rt[,5],col=5)
lines(t,rowSums(Rt),lwd=2)
legend(
  "topright",
  c("Pool 1, DPM",
    "Pool 2, RPM",
    "Pool 3, BIO",
    "Pool 4, HUM",
    "Pool 5, IOM",
    "Total Respiration"
  ),
  lty=c(1,1,1,1,1,1),
  lwd=c(1, 1,1,1,1,2),
  col=c(1,2,3,4,5,1),
  bty="n"
)

```

 show-methods

 ~~ Methods for Function show ~~

Description

~~ Methods for function show ~~

Methods

signature(object = "Model") [show_method__Model](#)

show_method__Model *show method Model*

Description

This function is a stub

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

summary-methods *~~ Methods for Function summary ~~*

Description

~~ Methods for function summary ~~

Methods

signature(object = "Model") [summary_method__Model](#)

summary_method__Model *summary method Model*

Description

This function is a stub

Arguments

object

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

 ThreepairMMmodel *Implementation of a 6-pool Michaelis-Menten model*

Description

This function implements a 6-pool Michaelis-Menten model with pairs of microbial biomass and substrate pools.

Usage

```
ThreepairMMmodel(t, ks, kb, Km, r, Af = 1, ADD, ival)
```

Arguments

t	vector of times to calculate a solution.
ks	a vector of length 3 representing SOM decomposition rate (m ³ d ⁻¹ (gCB) ⁻¹)
kb	a vector of length 3 representing microbial decay rate (d ⁻¹)
Km	a vector of length 3 representing the Michaelis constant (g m ⁻³)
r	a vector of length 3 representing the respired carbon fraction (unitless)
Af	a scalar representing the Activity factor; i.e. a temperature and moisture modifier (unitless)
ADD	a vector of length 3 representing the annual C input to the soil (g m ⁻³ d ⁻¹)
ival	

Value

An object of class NIModel that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
days=seq(0,1000)

#Run the model with default parameter values
MMmodel=ThreepairMMmodel(t=days,ival=rep(c(100,10),3),ks=c(0.1,0.05,0.01),
                           kb=c(0.005,0.001,0.0005),Km=c(100,150,200),r=c(0.9,0.9,0.9),
                           ADD=c(3,1,0.5))

Cpools=getC(MMmodel)

#Time solution
matplot(days,Cpools,type="l",ylab="Concentrations",xlab="Days",lty=rep(1:2,3),
        ylim=c(0,max(Cpools)*1.2),col=rep(1:3,each=2),main="Multi-substrate microbial model")
```

```

legend("topright",c("Substrate 1", "Microbial biomass 1",
                    "Substrate 2", "Microbial biomass 2",
                    "Substrate 3", "Microbial biomass 3"),lty=rep(1:2,3),col=rep(1:3,each=2),
      bty="n")

#State-space diagram
plot(Cpools[,2],Cpools[,1],type="l",ylab="Substrate",xlab="Microbial biomass")
lines(Cpools[,4],Cpools[,3],col=2)
lines(Cpools[,6],Cpools[,5],col=3)
legend("topright",c("Substrate-Enzyme pair 1", "Substrate-Enzyme pair 2",
                    "Substrate-Enzyme pair 3"),col=1:3,lty=1,bty="n")

#Microbial biomass over time
plot(days,Cpools[,2],type="l",col=2,xlab="Days",ylab="Microbial biomass")

```

ThreepFeedbackModel *Implementation of a three pool model with feedback structure*

Description

This function creates a model for three pools connected with feedback. It is a wrapper for the more general function [GeneralModel](#).

Usage

```
ThreepFeedbackModel(t, ks, a21, a12, a32, a23, C0, In, xi = 1,
  solver = deSolve.lsoda.wrapper, pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 3 containing the values of the decomposition rates for pools 1, 2, and 3.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
a12	A scalar with the value of the transfer rate from pool 2 to pool 1.
a32	A scalar with the value of the transfer rate from pool 2 to pool 3.
a23	A scalar with the value of the transfer rate from pool 3 to pool 2.
C0	A vector containing the initial concentrations for the 3 pools. The length of this vector is 3
In	A data.frame object specifying the amount of litter inputs by time.
xi	A scalar or data.frame object specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

[ThreepParallelModel](#), [ThreepSeriesModel](#)

Examples

```
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
ks=c(k1=0.8,k2=0.4,k3=0.2)
C0=c(C10=100,C20=150, C30=50)
In = 60

Temp=rnorm(t,15,1)
TempEffect=data.frame(t,FT.Daycent1(Temp))

Ex1=ThreepFeedbackModel(t=t,ks=ks,a21=0.5,a12=0.1,a32=0.2,a23=0.1,C0=C0,In=In,xi=TempEffect)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plot(
  t,
  rowSums(Ct),
  type="l",
  ylab="Carbon stocks (arbitrary units)",
  xlab="Time (arbitrary units)",
  lwd=2,
  ylim=c(0,sum(Ct[51,]))
)
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
lines(t,Ct[,3],col=3)
legend(
  "topleft",
  c("Total C", "C in pool 1", "C in pool 2", "C in pool 3"),
  lty=c(1,1,1,1),
  col=c(1,2,4,3),
  lwd=c(2,1,1,1),
```

```

    bty="n"
  )

  plot(
    t,
    rowSums(Rt),
    type="l",
    ylab="Carbon released (arbitrary units)",
    xlab="Time (arbitrary units)",
    lwd=2,
    ylim=c(0, sum(Rt[51,]))
  )
  lines(t, Rt[,1], col=2)
  lines(t, Rt[,2], col=4)
  lines(t, Rt[,3], col=3)
  legend(
    "topleft",
    c("Total C release",
      "C release from pool 1",
      "C release from pool 2",
      "C release from pool 3"),
    lty=c(1,1,1,1),
    col=c(1,2,4,3),
    lwd=c(2,1,1,1),
    bty="n"
  )

  Inr=data.frame(t, Random.inputs=rnorm(length(t), 50, 10))
  plot(Inr, type="l")

  Ex2=ThreepFeedbackModel(t=t, ks=ks, a21=0.5, a12=0.1, a32=0.2, a23=0.1, C0=C0, In=Inr)
  Ctr=getC(Ex2)
  Rtr=getReleaseFlux(Ex2)

  plot(
    t,
    rowSums(Ctr),
    type="l",
    ylab="Carbon stocks (arbitrary units)",
    xlab="Time (arbitrary units)",
    lwd=2,
    ylim=c(0, sum(Ctr[51,]))
  )
  lines(t, Ctr[,1], col=2)
  lines(t, Ctr[,2], col=4)
  lines(t, Ctr[,3], col=3)
  legend("topright", c("Total C", "C in pool 1", "C in pool 2", "C in pool 3"),
        lty=c(1,1,1,1), col=c(1,2,4,3), lwd=c(2,1,1,1), bty="n")

  plot(t, rowSums(Rtr), type="l", ylab="Carbon released (arbitrary units)",
        xlab="Time (arbitrary units)", lwd=2, ylim=c(0, sum(Rtr[51,])))
  lines(t, Rtr[,1], col=2)
  lines(t, Rtr[,2], col=4)

```

```

lines(t,Rtr[,3],col=3)
legend(
  "topright",
  c("Total C release",
    "C release from pool 1",
    "C release from pool 2",
    "C release from pool 3"
  ),
  lty=c(1,1,1,1),
  col=c(1,2,4,3),
  lwd=c(2,1,1,1),
  bty="n")

```

ThreepFeedbackModel14 *Implementation of a three-pool C14 model with feedback structure*

Description

This function creates a model for three pools connected with feedback. It is a wrapper for the more general function [GeneralModel_14](#) that can handle an arbitrary number of pools with arbitrary connection. [GeneralModel_14](#) can also handle input data in different formats, while this function requires its input as Delta14C. Look at it as an example how to use the more powerful tool [GeneralModel_14](#) or as a shortcut for a standard task!

Usage

```

ThreepFeedbackModel14(t, ks, C0, F0_Delta14C, In, a21, a12, a32,
  a23,
  xi = 1, inputFc, lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)

```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 3 containing the decomposition rates for the 3 pools.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.
F0_Delta14C	#The format will be assumed to be Delta14C, so please take care that it is. A vector of length 3 containing the initial fraction of radiocarbon for the 3 pools in Delta14C format.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
a12	A scalar with the value of the transfer rate from pool 2 to pool 1.
a32	A scalar with the value of the transfer rate from pool 2 to pool 3.
a23	A scalar with the value of the transfer rate from pool 3 to pool 2.

<code>xi</code>	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
<code>inputFc</code>	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
<code>lambda</code>	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
<code>lag</code>	A positive scalar representing a time lag for radiocarbon to enter the system.
<code>solver</code>	A function that solves the system of ODEs. This can be <code>euler</code> or <code>ode</code> or any other user provided function with the same interface.
<code>pass</code>	if TRUE forces the constructor to create the model even if it is invalid. This is sometimes useful when SoilR is used by external packages for parameter estimation.

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[GeneralModel_14](#) [ThreepSeriesModel14](#), [ThreepParallelModel14](#)

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=100
k1=1/2; k2=1/10; k3=1/50
a21=0.9*k1
a12=0.4*k2
a32=0.4*k2
a23=0.7*k3

Feedback=ThreepFeedbackModel14(
  t=years,
  ks=c(k1=k1, k2=k2, k3=k3),
  C0=c(100,500,1000),
  F0_Delta14C=c(0,0,0),
  In=LitterInput,
  a21=a21,
  a12=a12,
  a32=a32,
  a23=a23,
  inputFc=C14Atm_NH
)
```

```

F.R14m=getF14R(Feedback)
F.C14m=getF14C(Feedback)
F.C14t=getF14(Feedback)

Series=ThreepSeriesModel14(
  t=years,
  ks=c(k1=k1, k2=k2, k3=k3),
  C0=c(100,500,1000),
  F0_Delta14C=c(0,0,0),
  In=LitterInput,
  a21=a21,
  a32=a32,
  inputFc=C14Atm_NH
)
S.R14m=getF14R(Series)
S.C14m=getF14C(Series)
S.C14t=getF14(Series)

Parallel=ThreepParallelModel14(
  t=years,
  ks=c(k1=k1, k2=k2, k3=k3),
  C0=c(100,500,1000),
  F0_Delta14C=c(0,0,0),
  In=LitterInput,
  gam1=0.6,
  gam2=0.2,
  inputFc=C14Atm_NH,
  lag=2
)
P.R14m=getF14R(Parallel)
P.C14m=getF14C(Parallel)
P.C14t=getF14(Parallel)

par(mfrow=c(3,2))
plot(
  C14Atm_NH,
  type="l",
  xlab="Year",
  ylab=expression(paste(Delta^14,"C ", "\u2030")),
  xlim=c(1940,2010)
)
lines(years, P.C14t[,1], col=4)
lines(years, P.C14t[,2], col=4, lwd=2)
lines(years, P.C14t[,3], col=4, lwd=3)
legend(
  "topright",
  c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),
  lty=rep(1,4),
  col=c(1,4,4,4),
  lwd=c(1,1,2,3),
  bty="n"
)

```

```

plot(C14Atm_NH,type="l",xlab="Year",
     ylab=expression(paste(Delta^14,"C ", "\u2030")),xlim=c(1940,2010))
lines(years,P.C14m,col=4)
lines(years,P.R14m,col=2)
legend("topright",c("Atmosphere", "Bulk SOM", "Respired C"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
     ylab=expression(paste(Delta^14,"C ", "\u2030")),xlim=c(1940,2010))
lines(years, S.C14t[,1], col=4)
lines(years, S.C14t[,2],col=4,lwd=2)
lines(years, S.C14t[,3],col=4,lwd=3)
legend("topright",c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),
      lty=rep(1,4),col=c(1,4,4,4),lwd=c(1,1,2,3),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
     ylab=expression(paste(Delta^14,"C ", "\u2030")),xlim=c(1940,2010))
lines(years,S.C14m,col=4)
lines(years,S.R14m,col=2)
legend("topright",c("Atmosphere", "Bulk SOM", "Respired C"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
     ylab=expression(paste(Delta^14,"C ", "\u2030")),xlim=c(1940,2010))
lines(years, F.C14t[,1], col=4)
lines(years, F.C14t[,2],col=4,lwd=2)
lines(years, F.C14t[,3],col=4,lwd=3)
legend("topright",c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),
      lty=rep(1,4),col=c(1,4,4,4),lwd=c(1,1,2,3),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
     ylab=expression(paste(Delta^14,"C ", "\u2030")),xlim=c(1940,2010))
lines(years,F.C14m,col=4)
lines(years,F.R14m,col=2)
legend("topright",c("Atmosphere", "Bulk SOM", "Respired C"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")

par(mfrow=c(1,1))

```

ThreepParallelModel *Implementation of a three pool model with parallel structure*

Description

The function creates a model for three independent (parallel) pools. It is a wrapper for the more general function [ParallelModel](#) that can handle an arbitrary number of pools.

Usage

```
ThreepParallelModel(t,
  ks, C0, In, gam1, gam2, xi = 1, solver = deSolve::lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 3 containing the decomposition rates for the 3 pools.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
gam1	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.
gam2	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 2.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. *Geoscientific Model Development* 5, 1045-1060.

See Also

[TwopParallelModel](#) and [ParallelModel](#)

Examples

```
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
```

```
Ex=ThreepParallelModel(t,ks=c(k1=0.5,k2=0.2,k3=0.1),
```

```

C0=c(c10=100, c20=150, c30=50), In=20, gam1=0.7, gam2=0.1, xi=0.5)
Ct=getC(Ex)

plot(t, rowSums(Ct), type="l", lwd=2,
     ylab="Carbon stocks (arbitrary units)", xlab="Time", ylim=c(0, sum(Ct[1,])))
lines(t, Ct[,1], col=2)
lines(t, Ct[,2], col=4)
lines(t, Ct[,3], col=3)
legend("topright", c("Total C", "C in pool 1", "C in pool 2", "C in pool 3"),
      lty=c(1,1,1,1), col=c(1,2,4,3), lwd=c(2,1,1,1), bty="n")

Rt=getReleaseFlux(Ex)
plot(t, rowSums(Rt), type="l", ylab="Carbon released (arbitrary units)",
     xlab="Time", lwd=2, ylim=c(0, sum(Rt[1,])))
lines(t, Rt[,1], col=2)
lines(t, Rt[,2], col=4)
lines(t, Rt[,3], col=3)
legend("topright", c("Total C release", "C release from pool 1",
                    "C release from pool 2", "C release from pool 3"),
      lty=c(1,1,1,1), col=c(1,2,4,3), lwd=c(2,1,1,1), bty="n")

```

ThreepParallelModel14 *Implementation of a three-pool C14 model with parallel structure*

Description

This function creates a model for two independent (parallel) pools. It is a wrapper for the more general function [GeneralModel_14](#) that can handle an arbitrary number of pools.

Usage

```

ThreepParallelModel14(t, ks, C0, F0_Delta14C, In, gam1, gam2,
  xi = 1, inputFc, lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)

```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 3 containing the decomposition rates for the 3 pools.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.
F0_Delta14C	A vector of length 3 containing the initial amount of the radiocarbon fraction for the 3 pools in Delta14C values in per mil.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
gam1	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.

gam2	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 2.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
lag	A positive scalar representing a time lag for radiocarbon to enter the system.
solver	A function that solves the system of ODEs. This can be <code>euler</code> or <code>ode</code> or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopSeriesModel14](#), [TwopFeedbackModel14](#)

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=ThreepParallelModel14(
  t=years,
  ks=c(k1=1/2.8, k2=1/35, k3=1/100),
  C0=c(200,5000,500),
  F0_Delta14C=c(0,0,0),
  In=LitterInput,
  gam1=0.7,
  gam2=0.1,
  inputFc=C14Atm_NH,
  lag=2
)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
```

```

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
lines(years, C14t[,3],col=4,lwd=3)
legend(
  "topright",
  c(
    "Delta 14C Atmosphere",
    "Delta 14C pool 1",
    "Delta 14C pool 2",
    "Delta 14C pool 3"
  ),
  lty=rep(1,4),
  col=c(1,4,4,4),
  lwd=c(1,1,2,3),
  bty="n"
)

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))

```

ThreepSeriesModel *Implementation of a three pool model with series structure*

Description

This function creates a model for three pools connected in series. It is a wrapper for the more general function [GeneralModel](#).

Usage

```
ThreepSeriesModel(t, ks, a21, a32, C0, In, xi = 1, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 3 containing the values of the decomposition rates for pools 1, 2, and 3.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
a32	A scalar with the value of the transfer rate from pool 2 to pool 3.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.
In	A scalar or data.frame object specifying the amount of litter inputs by time.

<code>xi</code>	A scalar or data.frame object specifying the external (environmental and/or edaphic) effects on decomposition rates.
<code>solver</code>	A function that solves the system of ODEs. This can be <code>euler</code> or <code>ode</code> or any other user provided function with the same interface.
<code>pass</code>	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

[ThreepParallelModel](#), [ThreepFeedbackModel](#)

Examples

```
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
ks=c(k1=0.8,k2=0.4,k3=0.2)
C0=c(C10=100,C20=150, C30=50)
In = 50

Ex1=ThreepSeriesModel(t=t,ks=ks,a21=0.5,a32=0.2,C0=C0,In=In,xi=fT.Q10(15))
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plot(t,rowSums(Ct),type="l",ylab="Carbon stocks (arbitrary units)",
      xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Ct[1,])))
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
lines(t,Ct[,3],col=3)
legend("topright",c("Total C","C in pool 1", "C in pool 2","C in pool 3"),
      lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")

plot(t,rowSums(Rt),type="l",ylab="Carbon released (arbitrary units)",
      xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Rt[1,])))
lines(t,Rt[,1],col=2)
lines(t,Rt[,2],col=4)
lines(t,Rt[,3],col=3)
```

```

legend("topright",c("Total C release","C release from pool 1",
  "C release from pool 2","C release from pool 3"),
  lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")

Inr=data.frame(t,Random.inputs=rnorm(length(t),50,10))
plot(Inr,type="l")

Ex2=ThreepSeriesModel(t=t,ks=ks,a21=0.5,a32=0.2,C0=C0,In=Inr)
Ctr=getC(Ex2)
Rtr=getReleaseFlux(Ex2)

plot(t,rowSums(Ctr),type="l",ylab="Carbon stocks (arbitrary units)",
  xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Ctr[1,])))
lines(t,Ctr[,1],col=2)
lines(t,Ctr[,2],col=4)
lines(t,Ctr[,3],col=3)
legend("topright",c("Total C","C in pool 1","C in pool 2","C in pool 3"),
  lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")

plot(t,rowSums(Rtr),type="l",ylab="Carbon released (arbitrary units)",
  xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Rtr[1,])))
lines(t,Rtr[,1],col=2)
lines(t,Rtr[,2],col=4)
lines(t,Rtr[,3],col=3)
legend("topright",c("Total C release","C release from pool 1",
  "C release from pool 2","C release from pool 3"),
  lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")

```

ThreepSeriesModel14 *Implementation of a three-pool C14 model with series structure*

Description

This function creates a model for three pools connected in series. It is a wrapper for the more general function [GeneralModel_14](#) that can handle an arbitrary number of pools.

Usage

```

ThreepSeriesModel14(t, ks, C0, F0_Delta14C, In, a21, a32, xi = 1,
  inputFc, lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)

```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 3 containing the decomposition rates for the 3 pools.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.

F0_Delta14C	A vector of length 3 containing the initial amount of the radiocarbon fraction for the 3 pools.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
a32	A scalar with the value of the transfer rate from pool 2 to pool 3 as Delta14C values in per mil.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
lag	A positive scalar representing a time lag for radiocarbon to enter the system.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[ThreepParallelModel14](#), [ThreepFeedbackModel14](#)

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=ThreepSeriesModel14(
  t=years,ks=c(k1=1/2.8, k2=1/35, k3=1/100),
  C0=c(200,5000,500), F0_Delta14C=c(0,0,0),
  In=LitterInput, a21=0.1, a32=0.01,inputFc=C14Atm_NH
)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",
```

```

      ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
lines(years, C14t[,3],col=4,lwd=3)
legend(
  "topright",
  c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2", "Delta 14C pool 3"),
  lty=rep(1,4),col=c(1,4,4,4),lwd=c(1,1,2,3),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))

```

TimeMap-class

Class "TimeMap"

Description

defines a (time dependent) mapping including the function definition and the ### domain where the function is well define. This can be used to avoid interpolations out of range when mixing different time dependent data sets

Slots

map: Object of class "function" ~~
lag: Object of class "numeric" ~~
starttime: Object of class "numeric" ~~
endtime: Object of class "numeric" ~~

Methods

BoundInFlux signature(map = "TimeMap", starttime = "missing", endtime = "missing", lag = "missing", i
...
BoundLinDecompOp signature(map = "TimeMap", starttime = "missing", endtime = "missing", lag = "miss
...
DecompOp signature(object = "TimeMap"): ...
InFlux signature(object = "TimeMap"): ...
as.character signature(x = "TimeMap"): ...
getFunctionDefinition signature(object = "TimeMap"): ...
getTimeRange signature(object = "TimeMap"): ...
initialize signature(.Object = "TimeMap"): ...

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
showClass("TimeMap")
```

TimeMap.from.DataFrame

TimeMap from DataFrame

Description

This function is another constructor of the class TimeMap.

Usage

```
TimeMap.from.DataFrame(dframe, lag = 0, interpolation = splinefun)
```

Arguments

dframe	A data frame containing exactly two columns: the first one is interpreted as time
lag	a scalar describing the time lag. Positive Values shift the argument of the interpolation function forward in time. (retard its effect)
interpolation	A function that returns a function the default is splinefun. Other possible values are the linear interpolation approxfun or any self made function with the same interface.

Value

An object of class TimeMap that contains the interpolation function and the limits of the time range where the function is valid. Note that the limits change according to the time lag this serves as a safeguard for Model which thus can check that all involved functions of time are actually defined for the times of interest

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

TimeMap.new *basic constructor of the class TimeMap.*

Description

A TimeMap is nothing more than an usual R-function of one argument augmented by the lower and upper boundary of the interval where it is defined.

Usage

```
TimeMap.new(t_start, t_end, f)
```

Arguments

t_start	A number marking the begin of the time domain where the function is valid
t_end	A number the end of the time domain where the function is valid
f	The time dependent function definition (a function in R's sense)

Value

An object of class TimeMap that can be used to describe models.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

turnoverFit *Estimation of the turnover time from a soil radiocarbon sample.*

Description

This function finds the best possible value of turnover time from a soil radiocarbon sample assuming a one pool model and annual litter inputs.

Usage

```
turnoverFit(obsC14, obsyr, In, C0 = 0, yr0 = 1900, Zone = "NHZone2",  
          plot = TRUE)
```


Arguments

obsC14	a scalar with the observed radiocarbon value in Delta14C of the soil sample.
obsyr	a scalar with the year in which the soil sample was taken.
In	a scalar or data.frame with the annual amount of litter inputs to the soil.
C0	a scalar with the initial amount of carbon stored at the beginning of the simulation.
yr0	The year at which simulations will start.
Zone	the hemispheric zone of atmospheric radiocarbon. Possible values are NHZone1: northern hemisphere zone 1, NHZone2: northern hemisphere zone 2, NHZone3: northern hemisphere zone 3, SHZone12: southern hemisphere zones 1 and 2, SHZone3: southern hemisphere zone 3. See Hua2013 for additional information.
plot	logical. Should the function produce a plot?

Details

This algorithm takes the observed values and a given amount of litter inputs, runs [OnepModel14](#), calculates the squared difference between predictions and observations, and uses [optimize](#) to find the minimum difference. If the turnover time is relatively short (< 50 yrs), it is safe to assume C0=0 because the soil will reach steady state within the simulation time. However, for longer turnover times it is recommended to use a value of C0 close to the steady state value.

Value

A scalar with the value of the turnover time that minimizes the difference between the prediction of a one pool model and the observed radiocarbon value.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

Examples

```
## Not run:  
# Calculate the turnover time for a sample from a temperate forest soil  
turnoverFit(obsC14=115.22, obsyr=2004.5, C0=2800, yr0=1900,  
            In=473, Zone="NHZone2")  
  
## End(Not run)
```

TwoPoolFeedbackModel *Implementation of a two pool model with feedback structure*

Description

This function creates a model for two pools connected with feedback. It is a wrapper for the more general function [GeneralModel](#).

Usage

```
TwoPoolFeedbackModel(t, ks, a21, a12, C0, In, xi = 1, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 2 with the values of the decomposition rate for pools 1 and 2.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
a12	A scalar with the value of the transfer rate from pool 2 to pool 1.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
In	A data.frame object specifying the amount of litter inputs by time.
xi	A scalar or data.frame object specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

[TwoParallelModel](#), [TwoSeriesModel](#)

Examples

```

#This example show the difference between the three types of two-pool models
times=seq(0,20,by=0.1)
ks=c(k1=0.8,k2=0.00605)
C0=c(C10=5,C20=5)

Temp=rnorm(times,15,2)
WC=runif(times,10,20)
TempEffect=data.frame(times,fT=fT.Daycent1(Temp))
MoistEffect=data.frame(times, fW=fW.Daycent2(WC)[2])

Inmean=1
InRand=data.frame(times,Random.inputs=rnorm(length(times),Inmean,0.2))
InSin=data.frame(times,Inmean+0.5*sin(times*pi*2))

Parallel=TwopParallelModel(t=times,ks=ks,C0=C0,In=Inmean,gam=0.9,
                           xi=(fT.Daycent1(15)*fW.Demeter(15)))
Series=TwopSeriesModel(t=times,ks=ks,a21=0.2*ks[1],C0=C0,In=InSin,
                       xi=(fT.Daycent1(15)*fW.Demeter(15)))
Feedback=TwopFeedbackModel(t=times,ks=ks,a21=0.2*ks[1],a12=0.5*ks[2],C0=C0,
                           In=InRand,xi=MoistEffect)

CtP=getC(Parallel)
CtS=getC(Series)
CtF=getC(Feedback)

RtP=getReleaseFlux(Parallel)
RtS=getReleaseFlux(Series)
RtF=getReleaseFlux(Feedback)

par(mfrow=c(2,1),mar=c(4,4,1,1))
plot(times,rowSums(CtP),type="l",ylim=c(0,20),ylab="Carbon stocks (arbitrary units)",xlab=" ")
lines(times,rowSums(CtS),col=2)
lines(times,rowSums(CtF),col=3)
legend("topleft",c("Two-pool Parallel","Two-pool Series","Two-pool Feedback"),
      lty=c(1,1,1),col=c(1,2,3),bty="n")

plot(times,rowSums(RtP),type="l",ylim=c(0,3),ylab="Carbon release (arbitrary units)", xlab="Time")
lines(times,rowSums(RtS),col=2)
lines(times,rowSums(RtF),col=3)
par(mfrow=c(1,1))

```

TwopFeedbackModel14 *Implementation of a two-pool C14 model with feedback structure*

Description

This function creates a model for two pools connected with feedback. It is a wrapper for the more general function [GeneralModel_14](#) that can handle an arbitrary number of pools.

Usage

```
TwoFeedbackModel14(t, ks, C0, F0_Delta14C, In, a21, a12, xi = 1,
  inputFc, lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 2 containing the decomposition rates for the 2 pools.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
F0_Delta14C	A vector of length 2 containing the initial amount of the radiocarbon fraction for the 2 pools as Delta14C values in per mil.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
a12	A scalar with the value of the transfer rate from pool 2 to pool 1.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
lag	A positive integer representing a time lag for radiocarbon to enter the system.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwoSeriesModel14](#), [TwoParallelModel14](#)

Examples

```

years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=TwopFeedbackModel14(t=years,ks=c(k1=1/2.8, k2=1/35),C0=c(200,5000),
                        F0_Delta14C=c(0,0),In=LitterInput, a21=0.1,a12=0.01,inputFc=C14Atm_NH)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
legend("topright",c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2"),
      lty=c(1,1,1),col=c(1,4,4),lwd=c(1,1,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))

```

TwopMMmodel

*Implementation of a two-pool Michaelis-Menten model***Description**

This function implements a two-pool Michaelis-Menten model with a microbial biomass and a substrate pool.

Usage

```
TwopMMmodel(t, ks = 1.8e-05, kb = 0.007, Km = 900, r = 0.6, Af = 1,
            ADD = 3.2, ival)
```

Arguments

t	vector of times (in days) to calculate a solution.
ks	a scalar representing SOM decomposition rate (m ³ d ⁻¹ (gCB) ⁻¹)
kb	a scalar representing microbial decay rate (d ⁻¹)
Km	a scalar representing the Michaelis constant (g m ⁻³)
r	a scalar representing the respired carbon fraction (unitless)
Af	a scalar representing the Activity factor; i.e. a temperature and moisture modifier (unitless)
ADD	a scalar representing the annual C input to the soil (g m ⁻³ d ⁻¹)
ival	

Details

This implementation is similar to the model described in Manzoni and Porporato (2007).

Value

An object of class NIModel that can be further queried.

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Manzoni, S, A. Porporato (2007). A theoretical analysis of nonlinearities and feedbacks in soil carbon and nitrogen cycles. *Soil Biology and Biochemistry* 39: 1542-1556.

Examples

```

days=seq(0,1000,0.5)

#Run the model with default parameter values
MMmodel=TwopMMmodel(t=days,ival=c(100,10))
Cpools=getC(MMmodel)

#Time solution
matplot(days,Cpools,type="l",ylab="Concentrations",xlab="Days",lty=1,ylim=c(0,max(Cpools)*1.2))
legend("topleft",c("SOM-C", "Microbial biomass"),lty=1,col=c(1,2),bty="n")

ks=0.000018
kb=0.007
r=0.6
ADD=3.2

#Analytical solution of fixed points
Cs=kb/((1-r)*ks)
abline(h=Cs,lty=2)

Cb=(ADD*(1-r))/(r*kb)
abline(h=Cb,lty=2,col=2)

#State-space diagram
plot(Cpools[,2],Cpools[,1],type="l",ylab="SOM-C",xlab="Microbial biomass")

#Microbial biomass over time
plot(days,Cpools[,2],type="l",col=2,xlab="Days",ylab="Microbial biomass")

#The default parameterization exhaust the microbial biomass.
#A different behavior is obtained by increasing ks and decreasing kb
MMmodel=TwopMMmodel(t=days,ival=c(972,304),Af=3,kb=0.0000001)
Cpools=getC(MMmodel)

```

```

matplot(days,Cpools,type="l",ylab="Concentrations",xlab="Days",lty=1,ylim=c(0,max(Cpools)*1.2))
legend("topleft",c("SOM-C", "Microbial biomass"),lty=1,col=c(1,2),bty="n")

plot(Cpools[,2],Cpools[,1],type="l",ylab="SOM-C",xlab="Microbial biomass")

plot(days,Cpools[,2],type="l",col=2,xlab="Days",ylab="Microbial biomass")

```

TwopParallelModel *Implementation of a two pool model with parallel structure*

Description

This function creates a model for two independent (parallel) pools. It is a wrapper for the more general function [ParallelModel](#) that can handle an arbitrary number of pools.

Usage

```
TwopParallelModel(t, ks, C0, In, gam, xi = 1, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 2 containing the decomposition rates for the 2 pools.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
gam	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

[ThreepParallelModel](#)

Examples

```
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)

Ex=TwoParallelModel(t,ks=c(k1=0.5,k2=0.2),C0=c(c10=100, c20=150),In=10,gam=0.7,xi=0.5)
Ct=getC(Ex)

plot(t,rowSums(Ct),type="l",lwd=2,
      ylab="Carbon stocks (arbitrary units)",xlab="Time",ylim=c(0,sum(Ct[1,])))
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
legend("topright",c("Total C","C in pool 1", "C in pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

Rt=getReleaseFlux(Ex)
plot(t,rowSums(Rt),type="l",ylab="Carbon released (arbitrary units)",
      xlab="Time",lwd=2,ylim=c(0,sum(Rt[1,])))
lines(t,Rt[,1],col=2)
lines(t,Rt[,2],col=4)
legend("topleft",c("Total C release","C release from pool 1", "C release from pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")
```

TwoParallelModel14 *Implementation of a two-pool C14 model with parallel structure*

Description

This function creates a model for two independent (parallel) pools. It is a wrapper for the more general function [GeneralModel_14](#) that can handle an arbitrary number of pools.

Usage

```
TwoParallelModel14(t, ks, C0, F0_Delta14C, In, gam, xi = 1,
  inputFc, lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset [C14Atm_NH](#) is 1900-2010.

ks	A vector of length 2 containing the decomposition rates for the 2 pools.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
F0_Delta14C	A vector of length 2 containing the initial amount of the fraction of radiocarbon for the 2 pools as Delta14C values in per mil.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
gam	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
lag	A positive scalar representing a time lag for radiocarbon to enter the system.
solver	A function that solves the system of ODEs. This can be <code>euler</code> or <code>ode</code> or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwoSeriesModel14](#), [TwoFeedbackModel14](#)

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=TwoParallelModel14(t=years,ks=c(k1=1/2.8, k2=1/35),C0=c(200,5000),
                      F0_Delta14C=c(0,0),In=LitterInput, gam=0.7,inputFc=C14Atm_NH,lag=2)

R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
```

```

lines(years, C14t[,2],col=4,lwd=2)
legend("topright",c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2"),
      lty=c(1,1,1),col=c(1,4,4),lwd=c(1,1,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))

```

TwopSeriesModel

Implementation of a two pool model with series structure

Description

This function creates a model for two pools connected in series. It is a wrapper for the more general function [GeneralModel](#).

Usage

```
TwopSeriesModel(t, ks, a21, C0, In, xi = 1, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 2 with the values of the decomposition rate for pools 1 and 2.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. *Geoscientific Model Development* 5, 1045-1060.

See Also

[TwoParallelModel](#), [TwoFeedbackModel](#)

Examples

```
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
ks=c(k1=0.8,k2=0.4)
a21=0.5
C0=c(C10=100,C20=150)
In = 30

Temp=rnorm(t,15,1)
TempEffect=data.frame(t,fT.Daycent1(Temp))

Ex1=TwoSeriesModel(t,ks,a21,C0,In,xi=TempEffect)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plot(t,rowSums(Ct),type="l",ylab="Carbon stocks (arbitrary units)",
      xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Ct[1,])))
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
legend("bottomright",c("Total C", "C in pool 1", "C in pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

plot(t,rowSums(Rt),type="l",ylab="Carbon released (arbitrary units)",
      xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Rt[1,])))
lines(t,Rt[,1],col=2)
lines(t,Rt[,2],col=4)
legend("topright",c("Total C release", "C release from pool 1", "C release from pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

Inr=data.frame(t,Random.inputs=rnorm(length(t),30,5))
plot(Inr)

Ex2=TwoSeriesModel(t,ks,a21,C0,In=Inr,xi=fT.Q10(15))
Ctr=getC(Ex2)
Rtr=getReleaseFlux(Ex2)

plot(t,rowSums(Ctr),type="l",ylab="Carbon stocks (arbitrary units)",
      xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Ctr[1,])))
lines(t,Ctr[,1],col=2)
lines(t,Ctr[,2],col=4)
```

```

legend("topright",c("Total C", "C in pool 1", "C in pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

plot(t,rowSums(Rtr),type="l",ylab="Carbon released (arbitrary units)",
      xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Rtr[1,])))
lines(t,Rtr[,1],col=2)
lines(t,Rtr[,2],col=4)
legend("topright",c("Total C release", "C release from pool 1", "C release from pool 2"),
      lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")

```

TwopSeriesModel14 *Implementation of a two-pool C14 model with series structure*

Description

This function creates a model for two pools connected in series. It is a wrapper for the more general function [GeneralModel_14](#) that can handle an arbitrary number of pools.

Usage

```

TwopSeriesModel14(t, ks, C0, F0_Delta14C, In, a21, xi = 1, inputFc,
  lambda = -0.0001209681, lag = 0, solver = deSolve.lsoda.wrapper,
  pass = FALSE)

```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 2 containing the decomposition rates for the 2 pools.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
F0_Delta14C	A vector of length 2 containing the initial amount of the radiocarbon fraction for the 2 pools as Delta14C values in per mil.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default $\lambda = -0.0001209681 \text{ y}^{-1}$. This has the side effect that all your time related data are treated as if the time unit was year.
lag	A (positive) scalar representing a time lag for radiocarbon to enter the system.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE Forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

See Also

[TwopParallelModel14](#), [TwopFeedbackModel14](#)

Examples

```
years=seq(1901,2009,by=0.5)
LitterInput=700

Ex=TwopSeriesModel14(t=years,ks=c(k1=1/2.8, k2=1/35),
                     C0=c(200,5000), F0_Delta14C=c(0,0),
                     In=LitterInput, a21=0.1,inputFc=C14Atm_NH)

R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",
     ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
legend("topright",c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2"),
      lty=c(1,1,1),col=c(1,4,4),lwd=c(1,1,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
      lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))
```

Description

This function creates a model for five pools as described in Tuomi et al. (2008)

Usage

```
Yasso07Model(t, ks = c(kA = 0.66, kW = 4.3, kE = 0.35, kN = 0.22,
  kH = 0.0033), p = c(p1 = 0.32, p2 = 0.01, p3 = 0.93, p4 = 0.34,
  p5 = 0, p6 = 0, p7 = 0, p8 = 0, p9 = 0.01, p10 = 0, p11 = 0,
  p12 = 0.92, pH = 0.04), C0, In, xi = 1, solver = deSolve.lsoda.wrapper,
  pass = FALSE)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 5 containing the values of the decomposition rates for each pool.
p	A vector of length 13 containing transfer coefficients among different pools.
C0	A vector containing the initial amount of carbon for the 5 pools. The length of this vector must be 5.
In	A single scalar or data.frame object specifying the amount of litter inputs by time
xi	A scalar or data.frame object specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or ode or any other user provided function with the same interface.
pass	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Tuomi, M., Thum, T., Jarvinen, H., Fronzek, S., Berg, B., Harmon, M., Trofymow, J., Sevanto, S., and Liski, J. (2009). Leaf litter decomposition-estimates of global variability based on Yasso07 model. *Ecological Modelling*, 220:3362 - 3371.

See Also

[ICBMModel](#), [RothCModel](#)

Examples

```
years=seq(0,50,0.1)
C0=rep(100,5)
In=0
```

```

Ex1=Yasso07Model(t=years,C0=C0,In=In)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plotCPool(years,Ct,col=1:5,xlab="years",ylab="C pool",
           ylim=c(0,max(Ct)))
legend("topright",c("xA","xW","xE","xN","xH"),lty=1,col=1:5,bty="n")

plotCPool(years,Rt,col=1:5,xlab="years",ylab="Respiration",ylim=c(0,50))
legend("topright",c("xA","xW","xE","xN","xH"),lty=1,col=1:5,bty="n")

```

YassoModel

Implementation of the Yasso model.

Description

This function creates a model for seven pools as described in Liski et al. (2005). Model not yet implemented due to lack of data in original publication: values of vector p not completely described in paper. 0.1 was assumed.

Usage

```

YassoModel(t, ks = c(a_fwl = 0.54, a_cwl = 0.03, k_ext = 0.48,
                    k_cel = 0.3, k_lig = 0.22, k_hum1 = 0.012, k_hum2 = 0.0012),
           p = c(fwl_ext = 0.1, cwl_ext = 0.1, fwl_cel = 0.1, cwl_cel = 0.1,
                fwl_lig = 0.1, cwl_lig = 0.1, pext = 0.05, pcel = 0.24,
                plig = 0.77, phum1 = 0.51), C0, In = c(u_fwl = 0.0758,
                u_cwl = 0.0866, u_nwl_cnwl_ext = 0.251 * 0.3, u_nwl_cnwl_cel = 0.251 *
                0.3, u_nwl_cnwl_lig = 0.251 * 0.3, 0, 0), xi = 1,
           solver = deSolve.lsoda.wrapper, pass = FALSE)

```

Arguments

<code>t</code>	A vector containing the points in time where the solution is sought.
<code>ks</code>	A vector of length 7 containing the values of the exposure and decomposition rates for each pool.
<code>p</code>	A vector of containing transfer coefficients among different pools.
<code>C0</code>	A vector containing the initial amount of carbon for the 7 pools. The length of this vector must be 7.
<code>In</code>	A vector of constant litter inputs.
<code>xi</code>	A scalar or data.frame object specifying the external (environmental and/or edaphic) effects on decomposition rates.
<code>solver</code>	A function that solves the system of ODEs. This can be <code>euler</code> or <code>ode</code> or any other user provided function with the same interface.
<code>pass</code>	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

References

Liski, J., Palosuo, T., Peltoniemi, M., and Sievanen, R. (2005). Carbon and decomposition model Yasso for forest soils. *Ecological Modelling*, 189:168-182.

See Also

[ThreepParallelModel](#), [ThreepSeriesModel](#)

Examples

```
years=seq(0,500,0.5)
C0=rep(100,7)

Ex1=YassoModel(t=years,C0=C0)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plotCPool(years,Ct,col=1:7,xlab="years",ylab="C pool",ylim=c(0,200))
legend("topright",c("fwl","cwl","ext","cel","lig","hum1","hum2"),lty=1,col=1:7,bty="n")

plotCPool(years,Rt,col=1:7,xlab="years",ylab="Respiration",ylim=c(0,50))
legend("topright",c("fwl","cwl","ext","cel","lig","hum1","hum2"),lty=1,col=1:7,bty="n")
```

[-methods

~~ *Methods for Function* [~~

Description

~~ Methods for function [~~

Methods

signature(x = "Model", i = "character") [\[_method_Model_character](#)

```
[[ -methods          ~ ~ Methods for Function [[ ~ ~
```

Description

All methods for function [[are intended for internal use inside the package only.

```
[[<- -methods      ~ ~ Methods for Function [[<- ~ ~
```

Description

All methods for function [[<- are intended for internal use inside the package only.

```
[_method__Model_character
                        (experimental) partially overload [ ] for models
```

Description

This method overloads the [] operator for Model objects but is not yet finished so the full interface for [] is not yet implemented and the behavior is very likely to change in future versions of SoilR

Arguments

x
i

Author(s)

Carlos A. Sierra <csierra@bgc-jena.mpg.de>, Markus Mueller <mamueller@bgc-jena.mpg.de>

```
$ -methods          ~ ~ Methods for Function $ ~ ~
```

Description

All methods for function \$ are intended for internal use inside the package only.

Index

*Topic **other possible keyword(s)**

[-methods, 168
[[[-methods, 169
[[[<--methods, 169
\$-methods, 169
AbsoluteFractionModern-methods, 7
AbsoluteFractionModern_from_Delta14C-methods, 8
as.character-methods, 9
availableParticleProperties-methods, 10
availableParticleSets-methods, 11
BoundFc-methods, 15
BoundInFlux-methods, 20
BoundLinDecompOp-methods, 24
computeResults-methods, 27
ConstLinDecompOp-methods, 29
DecompOp-methods, 31
Delta14C-methods, 33
Delta14C_from_AbsoluteFractionModern-methods, 34
GeneralModel-methods, 60
GeneralModel_14-methods, 63
getAccumulatedRelease-methods, 70
getC-methods, 72
getC14-methods, 73
getDecompOp-methods, 74
getDotOut-methods, 74
getF14-methods, 75
getF14C-methods, 75
getF14R-methods, 77
getFormat-methods, 78
getFunctionDefinition-methods, 80
getInFluxes-methods, 82
getInitialValues-methods, 83
getMeanTransitTime-methods, 84
getNumberOfPools-methods, 86

getOutputFluxes-methods, 86
getOutputReceivers-methods, 86
getParticleMonteCarloSimulator-methods, 86
getReleaseFlux-methods, 87
getReleaseFlux14-methods, 88
getTimeRange-methods, 91
getTimes-methods, 96
getTransferCoefficients-methods, 96
getTransferMatrix-methods, 97
getTransitTimeDistributionDensity-methods, 98
getValues-methods, 99
InFlux-methods, 104
initialize-methods, 105
Model-methods, 120
Model_14-methods, 122
plot-methods, 127
print-methods, 130
show-methods, 133
summary-methods, 134

*Topic **classes**

BoundFc-class, 14
BoundInFlux-class, 19
BoundLinDecompOp-class, 23
ConstLinDecompOp-class, 29
DecompOp-class, 30
DecompositionOperator-class, 33
InFlux-class, 103
Model-class, 119
Model_14-class, 121
TimeMap-class, 150

*Topic **datasets**

C14Atm, 26
C14Atm_NH, 27
eCO2, 37
HarvardForest14CO2, 99
Hua2013, 100

- IntCal09, 115
- IntCal13, 116
- *Topic **methods**
 - [-methods, 168
 - [[[-methods, 169
 - [[[<--methods, 169
 - \$-methods, 169
 - AbsoluteFractionModern-methods, 7
 - AbsoluteFractionModern_from_Delta14C-methods, 8
 - as.character-methods, 9
 - availableParticleProperties-methods, 10
 - availableParticleSets-methods, 11
 - BoundFc-methods, 15
 - BoundInFlux-methods, 20
 - BoundLinDecompOp-methods, 24
 - computeResults-methods, 27
 - ConstLinDecompOp-methods, 29
 - DecompOp-methods, 31
 - Delta14C-methods, 33
 - Delta14C_from_AbsoluteFractionModern-methods, 34
 - GeneralModel-methods, 60
 - GeneralModel_14-methods, 63
 - getAccumulatedRelease-methods, 70
 - getC-methods, 72
 - getC14-methods, 73
 - getDecompOp-methods, 74
 - getDotOut-methods, 74
 - getF14-methods, 75
 - getF14C-methods, 75
 - getF14R-methods, 77
 - getFormat-methods, 78
 - getFunctionDefinition-methods, 80
 - getInFluxes-methods, 82
 - getInitialValues-methods, 83
 - getMeanTransitTime-methods, 84
 - getNumberOfPools-methods, 86
 - getOutputFluxes-methods, 86
 - getOutputReceivers-methods, 86
 - getParticleMonteCarloSimulator-methods, 86
 - getReleaseFlux-methods, 87
 - getReleaseFlux14-methods, 88
 - getTimeRange-methods, 91
 - getTimes-methods, 96
 - getTransferCoefficients-methods, 96
 - getTransferMatrix-methods, 97
 - getTransitTimeDistributionDensity-methods, 98
 - getValues-methods, 99
 - InFlux-methods, 104
 - initialize-methods, 105
 - Model-methods, 120
 - Model_14-methods, 122
 - plot-methods, 127
 - print-methods, 130
 - show-methods, 133
 - summary-methods, 134
- *Topic **package**
 - SoilR-package, 6
 - [,Model,character-method([-methods), 168
 - [,NlModel,character-method([-methods), 168
 - [-methods, 168
 - [[,MCSim-method([[[-methods), 169
 - [[[-methods, 169
 - [[[<-,MCSim-method([[[<--methods), 169
 - [[[<--methods, 169
 - [_method__Model_character, 168, 169
 - \$_NlModel-method(\$-methods), 169
 - \$-methods, 169
 - AbsoluteFractionModern,BoundFc-method (AbsoluteFractionModern-methods), 7
 - AbsoluteFractionModern,ConstFc-method (AbsoluteFractionModern-methods), 7
 - AbsoluteFractionModern-methods, 7
 - AbsoluteFractionModern_from_Delta14C, 7
 - AbsoluteFractionModern_from_Delta14C,matrix-method (AbsoluteFractionModern_from_Delta14C-methods), 8
 - AbsoluteFractionModern_from_Delta14C,numeric-method (AbsoluteFractionModern_from_Delta14C-methods), 8
 - AbsoluteFractionModern_from_Delta14C-methods, 8
 - AbsoluteFractionModern_from_Delta14C_method__matrix, 8, 8
 - AbsoluteFractionModern_from_Delta14C_method__numeric, 8, 8

- AbsoluteFractionModern_method__BoundFc,
 7, 9
 as.character, 9
 as.character,BoundInFlux-method
 (as.character-methods), 9
 as.character,MCSim-method
 (as.character-methods), 9
 as.character,TimeMap-method
 (as.character-methods), 9
 as.character-methods, 9
 as.character_method__BoundInFlux, 9, 10
 as.character_method__TimeMap, 9, 10
 availableParticleProperties,MCSim-method
 (availableParticleProperties-methods),
 10
 availableParticleProperties-methods,
 10
 availableParticleSets,MCSim-method
 (availableParticleSets-methods),
 11
 availableParticleSets-methods, 11
 bacwaveModel, 11
 bind.C14curves, 12
 BoundFc, 13
 BoundFc,data.frame,missing,missing,missing,character,function-method
 (BoundFc-methods), 15
 BoundFc,data.frame,missing,missing,missing,character,function-method
 (BoundFc-methods), 15
 BoundFc,data.frame,missing,missing,numeric,character,function-method
 (BoundFc-methods), 15
 BoundFc,data.frame,missing,missing,numeric,character,missing-method
 (BoundFc-methods), 15
 BoundFc,function,numeric,numeric,missing,character,missing-method
 (BoundFc-methods), 15
 BoundFc,function,numeric,numeric,numeric,character,missing-method
 (BoundFc-methods), 15
 BoundFc-class, 14
 BoundFc-methods, 15
 BoundFc_method__data.frame_missing_missing_missing_missing_character_function,
 15, 15
 BoundFc_method__data.frame_missing_missing_missing_missing_character_missing,
 15, 16
 BoundFc_method__data.frame_missing_missing_numeric_character_missing,
 15, 16
 BoundFc_method__data.frame_missing_missing_numeric_character_missing,
 15-17, 17
 BoundFc_method__function_numeric_numeric_missing_missing_missing_missing_missing,
 15, 17
 BoundFc_method__function_numeric_numeric_numeric_numeric_character
 15, 17, 18
 BoundInFlux, 18, 106
 BoundInFlux,data.frame,missing,missing,missing,missing-method
 (BoundInFlux-methods), 20
 BoundInFlux,data.frame,missing,missing,numeric,function-me
 (BoundInFlux-methods), 20
 BoundInFlux,function,numeric,numeric,missing,missing-metho
 (BoundInFlux-methods), 20
 BoundInFlux,function,numeric,numeric,numeric,missing-metho
 (BoundInFlux-methods), 20
 BoundInFlux,TimeMap,missing,missing,missing,missing-method
 (BoundInFlux-methods), 20
 BoundInFlux-class, 19
 BoundInFlux-methods, 20
 BoundInFlux_method__data.frame_missing_missing_missing_mis
 20, 20
 BoundInFlux_method__data.frame_missing_missing_numeric_fun
 20, 21
 BoundInFlux_method__function_numeric_numeric_missing_mis
 20, 21
 BoundInFlux_method__function_numeric_numeric_numeric_mis
 20, 22
 BoundInFlux_method__TimeMap_missing_missing_missing_mis
 20, 22
 BoundLinDecompOp, 23
 BoundLinDecompOp,ConstLinDecompOp,ANY,ANY,ANY-method
 (BoundLinDecompOp-methods), 24
 BoundLinDecompOp,function,numeric,numeric,missing-method
 (BoundLinDecompOp-methods), 24
 BoundLinDecompOp,function,numeric,numeric,numeric-method
 (BoundLinDecompOp-methods), 24
 BoundLinDecompOp,TimeMap,missing,missing,missing-method
 (BoundLinDecompOp-methods), 24
 BoundLinDecompOp-class, 23
 BoundLinDecompOp-methods, 24
 BoundLinDecompOp_method__ConstLinDecompOp,
 24, 24
 BoundLinDecompOp_method__function_numeric_numeric_missing,
 24, 25
 BoundLinDecompOp_method__function_numeric_numeric_numeric,
 24, 25
 BoundLinDecompOp_method__TimeMap_missing_missing_missing,
 24, 26
 BoundLinDecompOp_method__TimeMap_missing_missing_missing,
 C14Atm, 26
 BoundLinDecompOp_method__TimeMap_missing_missing_missing,
 139, 144, 148,
 156, 160, 164

- computeResults,MCSim-method
(computeResults-methods), 27
- computeResults,NlModel-method
(computeResults-methods), 27
- computeResults-methods, 27
- ConstFc, 28
- ConstLinDecompOp, 28
- ConstLinDecompOp,matrix-method
(ConstLinDecompOp-methods), 29
- ConstLinDecompOp-class, 29
- ConstLinDecompOp-methods, 29
- ConstLinDecompOp_method__matrix, 29, 30

- DecompOp, 23, 29, 33
- DecompOp,DecompOp-method
(DecompOp-methods), 31
- DecompOp,matrix-method
(DecompOp-methods), 31
- DecompOp,TimeMap-method
(DecompOp-methods), 31
- DecompOp-class, 30
- DecompOp-methods, 31
- DecompOp_method__DecompOp, 31, 31
- DecompOp_method__matrix, 31, 32
- DecompOp_method__TimeMap, 31, 32
- DecompositionOperator-class, 33
- Delta14C,BoundFc-method
(Delta14C-methods), 33
- Delta14C,ConstFc-method
(Delta14C-methods), 33
- Delta14C-methods, 33
- Delta14C_from_AbsoluteFractionModern,
34
- Delta14C_from_AbsoluteFractionModern,matrix-method
(Delta14C_from_AbsoluteFractionModern-methods),
34
- Delta14C_from_AbsoluteFractionModern,numeric-method
(Delta14C_from_AbsoluteFractionModern-methods),
34
- Delta14C_from_AbsoluteFractionModern-methods,GeneralModel_14,numeric,ANY,numeric,ANY,ANY,ANY,missing,nu
34
(GeneralModel_14-methods), 63
- Delta14C_from_AbsoluteFractionModern_method__matrix,
34, 35
(GeneralModel_14-methods), 63
- Delta14C_from_AbsoluteFractionModern_method__numeric,
34, 35
(GeneralModel_14-methods), 63
- Delta14C_method__BoundFc, 33, 36
- deSolve.lsoda.wrapper, 36, 60, 69, 122,
126
- eCO2, 37
- euler, 58, 102, 123, 125, 132, 136, 140, 143,
145, 147, 149, 154, 156, 159, 161,
162, 164, 166, 167

- ft.Arrhenius, 38
- ft.Century1, 38
- ft.Century2, 39
- ft.Daycent1, 40
- ft.Daycent2, 41
- ft.Demeter, 42
- ft.KB, 42
- ft.LandT, 43
- ft.linear, 44
- ft.Q10, 45
- ft.RothC, 46
- ft.Standcarb, 47
- fW.Candy, 48
- fW.Century, 49
- fW.Daycent1, 50
- fW.Daycent2, 51
- fW.Demeter, 52
- fW.Gompertz, 52
- fW.Moyano, 53
- fW.RothC, 54
- fW.Skopp, 55
- fW.Standcarb, 56

- GaudinskiModel14, 57
- GeneralModel, 59, 70, 71, 73, 74, 87, 90, 123,
131, 136, 146, 154, 162
- GeneralModel,numeric,ANY,numeric-method
(GeneralModel-methods), 60
- GeneralModel-methods, 60
- GeneralModel_14, 57, 60, 70, 71, 74, 87, 124,
139, 140, 144, 148, 155, 160, 164
- GeneralModel_14,numeric,ANY,numeric,ANY,ANY,ANY,missing,mi
(GeneralModel_14-methods), 63
- GeneralModel_14,numeric,ANY,numeric,ANY,ANY,ANY,missing,nu
(GeneralModel_14-methods), 63
- GeneralModel_14,numeric,ANY,numeric,ANY,ANY,ANY,missing,nu
(GeneralModel_14-methods), 63
- GeneralModel_14,numeric,ANY,numeric,ANY,ANY,ANY,missing,nu
(GeneralModel_14-methods), 63
- GeneralModel_14,numeric,ANY,numeric,ANY,ANY,ANY,missing,nu
(GeneralModel_14-methods), 63
- GeneralModel_14,numeric,ANY,numeric,ANY,ANY,missing,ANY,nu
(GeneralModel_14-methods), 63
- GeneralModel_14-methods, 63

- GeneralModel_14_method__numeric_ANY_numeric_AggrANYANYModel_14-method
[63, 64](#) (getF14R-methods), [77](#)
- GeneralModel_14_method__numeric_ANY_numeric_AggrANYANYMethods, [77](#)
[63, 65](#) getF14R_method__Model_14, [77, 77](#)
- GeneralModel_14_method__numeric_ANY_numeric_AggrANYANYBoundInFlux-method
[63, 66](#) (getFormat-methods), [78](#)
- GeneralModel_14_method__numeric_ANY_numeric_AggrANYANYConstLinDecompOp-method
[64, 66](#) (getFormat-methods), [78](#)
- GeneralModel_14_method__numeric_ANY_numeric_AggrANYANYDecompositionOperator-method
[64, 67](#) (getFormat-methods), [78](#)
 getFormat_method__BoundFc, [78, 79](#)
- GeneralModel_14_method__numeric_ANY_numeric_AggrANYANYInFlux-method
[64, 68](#) getFunctionDefinition, BoundFc-method
- GeneralModel_method__numeric_ANY_numeric,
[60, 68](#) (getFunctionDefinition-methods),
[80](#)
- getAccumulatedRelease, [69](#)
- getAccumulatedRelease, Model-method
 (getAccumulatedRelease-methods),
[70](#)
- getAccumulatedRelease-methods, [70](#)
- getAccumulatedRelease_method__Model,
[70, 70](#)
- getC, [71, 117, 118, 129](#)
- getC, Model-method (getC-methods), [72](#)
- getC, NIModel-method (getC-methods), [72](#)
- getC-methods, [72](#)
- getC14, [72, 117, 118, 128](#)
- getC14, Model_14-method
 (getC14-methods), [73](#)
- getC14-methods, [73](#)
- getC14_method__Model_14, [73, 73](#)
- getC_method__Model, [72, 73](#)
- getDecompOp, NIModel-method
 (getDecompOp-methods), [74](#)
- getDecompOp-methods, [74](#)
- getDotOut, TransportDecompositionOperator-method
 (getDotOut-methods), [74](#)
- getDotOut-methods, [74](#)
- getF14, [74](#)
- getF14, Model_14-method
 (getF14-methods), [75](#)
- getF14-methods, [75](#)
- getF14_method__Model_14, [75, 78](#)
- getF14C, [75](#)
- getF14C, Model_14-method
 (getF14C-methods), [75](#)
- getF14C-methods, [75](#)
- getF14C_method__Model_14, [75, 76](#)
- getF14R, [76](#)
- getFunctionDefinition, BoundInFlux-method
 (getFunctionDefinition-methods),
[80](#)
- getFunctionDefinition, BoundLinDecompOp-method
 (getFunctionDefinition-methods),
[80](#)
- getFunctionDefinition, ConstLinDecompOp-method
 (getFunctionDefinition-methods),
[80](#)
- getFunctionDefinition, DecompositionOperator-method
 (getFunctionDefinition-methods),
[80](#)
- getFunctionDefinition, TimeMap-method
 (getFunctionDefinition-methods),
[80](#)
- getFunctionDefinition, TransportDecompositionOperator-method
 (getFunctionDefinition-methods),
[80](#)
- getFunctionDefinition-methods, [80](#)
- getFunctionDefinition_method__BoundFc,
[80, 80](#)
- getFunctionDefinition_method__BoundInFlux,
[80, 81](#)
- getFunctionDefinition_method__BoundLinDecompOp,
[80, 81](#)
- getFunctionDefinition_method__ConstLinDecompOp,
[80, 81](#)
- getFunctionDefinition_method__DecompositionOperator,
[80, 82](#)
- getFunctionDefinition_method__TimeMap,
[80, 82](#)
- getInFluxes, NIModel-method
 (getInFluxes-methods), [82](#)
- getInFluxes-methods, [82](#)

- getInitialValues, NlModel-method
(getInitialValues-methods), 83
- getInitialValues-methods, 83
- getMeanTransitTime, 83, 97
- getMeanTransitTime, ConstLinDecompOp-method
(getMeanTransitTime-methods), 84
- getMeanTransitTime-methods, 84
- getMeanTransitTime_method__ConstLinDecompOp, 85, 85
- getNumberOfPools, NlModel-method
(getNumberOfPools-methods), 86
- getNumberOfPools, TransportDecompositionOperator-method
(getNumberOfPools-methods), 86
- getNumberOfPools-methods, 86
- getOutputFluxes, NlModel-method
(getOutputFluxes-methods), 86
- getOutputFluxes-methods, 86
- getOutputReceivers, TransportDecompositionOperator, numeric-method
(getOutputReceivers-methods), 86
- getOutputReceivers-methods, 86
- getParticleMonteCarloSimulator, NlModel-method
(getParticleMonteCarloSimulator-methods), 86
- getParticleMonteCarloSimulator-methods, 86
- getReleaseFlux, 69, 87, 117, 118, 129
- getReleaseFlux, Model-method
(getReleaseFlux-methods), 87
- getReleaseFlux-methods, 87
- getReleaseFlux14, 88
- getReleaseFlux14, Model_14-method
(getReleaseFlux14-methods), 88
- getReleaseFlux14-methods, 88
- getReleaseFlux14_method__Model_14, 88, 89
- getReleaseFlux_method__Model, 87, 90
- getTimeRange, 91
- getTimeRange, BoundFc-method
(getTimeRange-methods), 91
- getTimeRange, BoundInFlux-method
(getTimeRange-methods), 91
- getTimeRange, BoundLinDecompOp-method
(getTimeRange-methods), 91
- getTimeRange, ConstLinDecompOp-method
(getTimeRange-methods), 91
- getTimeRange, DecompositionOperator-method
(getTimeRange-methods), 91
- getTimeRange, TimeMap-method
(getTimeRange-methods), 91
- getTimeRange, TransportDecompositionOperator-method
(getTimeRange-methods), 91
- getTimeRange-methods, 91
- getTimeRange_method__BoundFc, 92, 92
- getTimeRange_method__BoundInFlux, 92, 92
- getTimeRange_method__BoundLinDecompOp, 92, 93
- getTimeRange_method__ConstLinDecompOp, 92, 93
- getTimeRange_method__DecompositionOperator, 92, 94
- getTimeRange_method__TimeMap, 92, 94
- getTimes, 95
- getTimes, Model-method
(getTimes-methods), 96
- getTimes, NlModel-method
(getTimes-methods), 96
- getTimes-methods, 96
- getTimes_method__Model, 96, 96
- getTransferCoefficients, NlModel-method
(getTransferCoefficients-methods), 96
- getTransferCoefficients, TransportDecompositionOperator-method
(getTransferCoefficients-methods), 96
- getTransferCoefficients-methods, 96
- getTransferMatrix, TransportDecompositionOperator-method
(getTransferMatrix-methods), 97
- getTransferMatrix-methods, 97
- getTransitTimeDistributionDensity, 97
- getTransitTimeDistributionDensity, ConstLinDecompOp-method
(getTransitTimeDistributionDensity-methods), 98
- getTransitTimeDistributionDensity-methods, 98
- getTransitTimeDistributionDensity_method__ConstLinDecompOp, 98, 98
- getValues, ConstFc-method
(getValues-methods), 99
- getValues-methods, 99
- HarvardForest14CO2, 99
- Hua2013, 13, 26, 100, 153
- ICBModel, 101, 132, 166

- InFlux, [19](#)
- InFlux, InFlux-method (InFlux-methods), [104](#)
- InFlux, TimeMap-method (InFlux-methods), [104](#)
- InFlux-class, [103](#)
- InFlux-methods, [104](#)
- InFlux_method__InFlux, [104](#), [104](#)
- InFlux_method__TimeMap, [104](#), [105](#)
- initialize, BoundFc-method (initialize-methods), [105](#)
- initialize, BoundInFlux-method (initialize-methods), [105](#)
- initialize, BoundLinDecompOp-method (initialize-methods), [105](#)
- initialize, ConstFc-method (initialize-methods), [105](#)
- initialize, ConstLinDecompOp-method (initialize-methods), [105](#)
- initialize, DecompositionOperator-method (initialize-methods), [105](#)
- initialize, MCSim-method (initialize-methods), [105](#)
- initialize, Model-method (initialize-methods), [105](#)
- initialize, Model_14-method (initialize-methods), [105](#)
- initialize, NlModel-method (initialize-methods), [105](#)
- initialize, TimeMap-method (initialize-methods), [105](#)
- initialize, TransportDecompositionOperator-method (initialize-methods), [105](#)
- initialize-methods, [105](#)
- initialize_method__BoundFc, [105](#), [106](#)
- initialize_method__BoundInFlux, [105](#), [106](#)
- initialize_method__BoundLinDecompOp, [105](#), [107](#)
- initialize_method__ConstLinDecompOp, [105](#), [107](#)
- initialize_method__DecompositionOperator, [105](#), [108](#)
- initialize_method__Model, [105](#), [108](#)
- initialize_method__Model_14, [105](#), [112](#)
- initialize_method__TimeMap, [105](#), [114](#)
- IntCal09, [13](#), [115](#), [116](#)
- IntCal13, [13](#), [116](#)
- linesCPool, [117](#)
- list, [100](#)
- Model, [70](#), [71](#), [73](#), [74](#), [87](#), [90](#), [118](#), [121](#)
- Model, numeric, ANY, numeric-method (Model-methods), [120](#)
- Model-class, [119](#)
- Model-methods, [120](#)
- Model_14, [64](#), [66–68](#), [71](#), [120](#)
- Model_14, numeric, ANY, numeric, ConstFc, ANY, ANY, numeric, function (Model_14-methods), [122](#)
- Model_14, numeric, ANY, numeric, ConstFc, ANY, ANY, numeric, missing (Model_14-methods), [122](#)
- Model_14-class, [121](#)
- Model_14-methods, [122](#)
- Model_method__numeric_ANY_numeric, [120](#), [122](#)
- ode, [102](#), [123](#), [125](#), [132](#), [136](#), [140](#), [143](#), [145](#), [147](#), [149](#), [154](#), [156](#), [159](#), [161](#), [162](#), [164](#), [166](#), [167](#)
- OnepModel, [123](#), [125](#)
- OnepModel14, [124](#), [153](#)
- optimize, [153](#)
- ParallelModel, [126](#), [142](#), [143](#), [159](#)
- plot, ANY-method (plot-methods), [127](#)
- plot, MCSim-method (plot-methods), [127](#)
- plot, Model-method (plot-methods), [127](#)
- plot, NlModel-method (plot-methods), [127](#)
- plot-methods, [127](#)
- plot_method__Model, [128](#), [130](#)
- plotC14Pool, [128](#)
- plotCPool, [129](#)
- print, ANY-method (print-methods), [130](#)
- print, Model-method (print-methods), [130](#)
- print, NlModel-method (print-methods), [130](#)
- print-methods, [130](#)
- print_method__Model, [130](#), [130](#)
- RespirationCoefficients, [131](#)
- RothCModel, [131](#), [166](#)
- show, Model-method (show-methods), [133](#)
- show, NlModel-method (show-methods), [133](#)
- show-methods, [133](#)
- show_method__Model, [134](#), [134](#)
- SoilR (SoilR-package), [6](#)

SoilR-package, [6](#)
summary, ANY-method (summary-methods),
[134](#)
summary, Model-method (summary-methods),
[134](#)
summary, NlModel-method
(summary-methods), [134](#)
summary-methods, [134](#)
summary_method_Model, [134](#), [134](#)

ThreepairMMmodel, [135](#)
ThreepFeedbackModel, [136](#), [147](#)
ThreepFeedbackModel14, [58](#), [139](#), [149](#)
ThreepParallelModel, [126](#), [137](#), [142](#), [147](#),
[160](#), [168](#)
ThreepParallelModel14, [58](#), [140](#), [144](#), [149](#)
ThreepSeriesModel, [137](#), [146](#), [168](#)
ThreepSeriesModel14, [140](#), [148](#)
TimeMap-class, [150](#)
TimeMap.from.DataFrame, [151](#)
TimeMap.new, [126](#), [152](#)
turnoverFit, [152](#)
TwopFeedbackModel, [64](#), [65](#), [67–71](#), [74](#), [87](#),
[122](#), [123](#), [154](#), [163](#)
TwopFeedbackModel14, [125](#), [145](#), [155](#), [161](#),
[165](#)
TwopMMmodel, [157](#)
TwopParallelModel, [64](#), [65](#), [67–71](#), [74](#), [87](#),
[122](#), [123](#), [126](#), [143](#), [154](#), [159](#), [163](#)
TwopParallelModel14, [125](#), [156](#), [160](#), [165](#)
TwopSeriesModel, [64](#), [65](#), [67–71](#), [74](#), [87](#), [102](#),
[122](#), [154](#), [162](#)
TwopSeriesModel14, [145](#), [156](#), [161](#), [164](#)

Yasso07Model, [165](#)
YassoModel, [167](#)