Package ‘SoilR’

January 21, 2022

Title Models of Soil Organic Matter Decomposition
Version 1.2.105
Date 2022-01-21
Author Carlos A. Sierra, Markus Mueller
Maintainer Carlos A. Sierra <csierra@bgc-jena.mpg.de>
Description Functions for modeling Soil Organic Matter decomposition in terrestrial ecosystems with linear and nonlinear models.
License GPL-3
Depends R (>= 3.5.0), deSolve, methods
Imports igraph, assertthat, parallel, expm, sets, purrr
Suggests FME, lattice, MASS, knitr, rmarkdown, getopt, tinytex
LazyData TRUE
  S4ClassDefinitions.R TimeMap.R ScalarTimeMap.R PoolId.R
  PoolIndex.R PoolName.R InFlux_by_PoolIndex.R
  InFlux_by_PoolName.R OutFlux_by_PoolIndex.R
  OutFlux_by_PoolName.R InternalFlux_by_PoolIndex.R
  InternalFlux_by_PoolName.R PoolConnection_by_PoolIndex.R
  PoolConnection_by_PoolName.R ConstantInFluxRate_by_PoolIndex.R
  ConstantInFluxRate_by_PoolName.R InFluxes.R ConstInFluxes.R
  ConstantInFlux_by_PoolIndex.R
  ConstantInternalFluxRate_by_PoolIndex.R
  ConstantInternalFluxRate_by_PoolName.R
  ConstantOutFluxRate_by_PoolIndex.R
  ConstantOutFluxRate_by_PoolName.R
  InFluxList_by_PoolIndex.R
  InFluxList_by_PoolName.R
  StateIndependentInFluxList_by_PoolIndex.R
  StateIndependentInFluxList_by_PoolName.R
  OutFluxList_by_PoolIndex.R
  OutFluxList_by_PoolName.R
  InternalFluxList_by_PoolIndex.R
  InternalFluxList_by_PoolName.R
  ConstantInFluxList_by_PoolIndex.R
  ConstantInFluxList_by_PoolName.R
  ConstantInternalFluxRateList_by_PoolIndex.R
  ConstantInternalFluxRateList_by_PoolName.R
  ConstantOutFluxRateList_by_PoolIndex.R
  ConstantOutFluxRateList_by_PoolName.R
  Fc.R HelperFunctions.R
  BoundInFluxes.R deSolve.lsoda.wrapper.R
  RespirationCoefficients.R TransportDecompositionOperator.R
R topics documented:

SoilR-package ......................................................... 12
AbsoluteFractionModern ........................................... 13
AbsoluteFractionModern,BoundFc-method ......................... 13
AbsoluteFractionModern,ConstFc-method ........................... 14
AbsoluteFractionModern_from_Delta14C ......................... 14
AbsoluteFractionModern_from_Delta14C,matrix-method ........... 15
AbsoluteFractionModern_from_Delta14C,numeric-method .......... 15
add_plot .......................................................... 15
add_plot,TimeMap-method ......................................... 16
as.character,TimeMap-method ..................................... 16
as.numeric,InFluxList_by_PoolName-method ....................... 17
as.numeric,InternalFluxList_by_PoolName-method ................. 17
as.numeric,InternalFlux_by_PoolName-method ...................... 18
as.numeric,OutFluxList_by_PoolName-method ...................... 18
availableParticleProperties ......................................... 19
availableParticleProperties,MCSim-method ......................... 19
availableParticleSets ............................................... 20
availableParticleSets,MCSim-method ................................ 20
availableResidentSets ............................................. 21
availableResidentSets,MCSim-method ................................. 21
AWBmodel .......................................................... 22
bacwaveModel ....................................................... 24
bind.C14curves ...................................................... 25
BoundFc ............................................................. 26
BoundFc.character-method ......................................... 26
BoundFc.missing-method ............................................ 27
BoundInFluxes ........................................................ 27
BoundInFluxes-class .................................................. 28
BoundLinDecompOp .................................................... 28
BoundLinDecompOp,ANY-method ..................................... 29
BoundLinDecompOp,UnBoundLinDecompOp-method ..................... 29
BoundLinDecompOp-class ............................................. 30
by_PoolIndex ........................................................ 30
by_PoolIndex,ConstantInFluxRate_by_PoolName,ANY,ANY-method .... 31
by_PoolIndex,ConstantInternalFluxRateList_by_PoolName,ANY,ANY-method .... 32
by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method ........ 32
by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-method ........... 33
by_PoolIndex,ConstantOutFluxRate_by_PoolName,ANY,ANY-method .................. 33
by_PoolIndex.function.character.character-method ................ 34
by_PoolIndex,InFluxList_by_PoolName.character.character-method ......... 35
by_PoolIndex,InFlux_by_PoolName.character.character-method ............ 35
by_PoolIndex,InternalFluxList_by_PoolName.character.character-method .... 36
by_PoolIndex,InternalFlux_by_PoolName.character.character-method .......... 36
by_PoolIndex,OutFluxList_by_PoolName.character.character-method ........... 37
by_PoolIndex,OutFlux_by_PoolName.character.character-method .............. 37
by_PoolIndex,PoolConnection_by_PoolName,ANY,ANY-method ............... 38
by_PoolName ........................................................ 38
by_PoolName,ConstantInFluxList_by_PoolIndex-method ............... 39
by_PoolName,ConstantInFluxRate_by_PoolIndex-method ............... 39
R topics documented:

by_PoolName,ConstantInFlux_by_PoolIndex-method .......................... 40
by_PoolName,ConstantInternalFluxRateList_by_PoolIndex-method .......... 40
by_PoolName,ConstantInternalFluxRate_by_PoolIndex-method ............... 41
by_PoolName,ConstantOutFluxRateList_by_PoolIndex-method ............... 41
by_PoolName,ConstantOutFluxRate_by_PoolIndex-method .................... 42
C14Atm ....................................................... 42
C14Atm_NH .................................................... 43
CenturyModel .................................................... 43
CenturyModel14 .................................................. 45
check_duplicate_pool_names ................................................. 47
check_id_length ...................................................... 47
check_pool_ids ...................................................... 47
check_pool_ids,PooolConnection_by_PoolIndex,integer-method ............ 48
computeResults ....................................................... 48
computeResults,MCSim-method .............................................. 49
ConstantInFluxList_by_PoolIndex .......................................... 49
ConstantInFluxList_by_PoolIndex,ConstInFluxes-method .................... 50
ConstantInFluxList_by_PoolIndex,list-method ................................ 50
ConstantInFluxList_by_PoolIndex,numeric-method .......................... 51
ConstantInFluxList_by_PoolIndex-class .................................... 51
ConstantInFluxList_by_PoolName ........................................... 51
ConstantInFluxList_by_PoolName,class ..................................... 52
ConstantInFluxRate_by_PoolIndex ........................................... 52
ConstantInFluxRate_by_PoolIndex-class ..................................... 52
ConstantInFlux_by_PoolIndex ............................................... 53
ConstantInFlux_by_PoolName ............................................... 53
ConstantInternalFluxRateList_by_PoolIndex ................................ 54
ConstantInternalFluxRateList_by_PoolIndex,list-method .................... 54
ConstantInternalFluxRateList_by_PoolIndex,numeric-method .............. 55
ConstantInternalFluxRateList_by_PoolIndex-class ......................... 55
ConstantInternalFluxRateList_by_PoolName ................................ 55
ConstantInternalFluxRateList_by_PoolName,list-method .................... 56
ConstantInternalFluxRateList_by_PoolName-class ........................... 56
ConstantInternalFluxRate_by_PoolIndex .................................... 57
ConstantInternalFluxRate_by_PoolIndex,missing,missing,character,numeric-method 57
ConstantInternalFluxRate_by_PoolIndex,numeric,numeric,missing,numeric-method 58
ConstantInternalFluxRate_by_PoolIndex-class ................................ 58
ConstantInternalFluxRate_by_PoolName .................................... 59
ConstantInternalFluxRate_by_PoolName,character,character,missing,numeric-method 59
ConstantInternalFluxRate_by_PoolName,missing,missing,character,numerical-method 60
ConstantInternalFluxRate_by_PoolName-class ................................ 60
ConstantOutFluxRateList_by_PoolIndex ..................................... 61
ConstantOutFluxRateList_by_PoolIndex,list-method .......................... 61
ConstantOutFluxRateList_by_PoolIndex,numeric-method ..................... 62
ConstantOutFluxRateList_by_PoolIndex-class ................................ 62
ConstantOutFluxRate_by_PoolIndex ......................................... 62
ConstantOutFluxRate_by_PoolName ......................................... 62
ConstantOutFluxRate_by_PoolName,list-method ................................ 63
ConstantOutFluxRate_by_PoolName,numerical-method ........................ 63
ConstantOutFluxRate_by_PoolName-class .................................... 64
ConstantOutFluxRate_by_PoolIndex ......................................... 64
ConstantOutFluxRate_by_PoolIndex,numerical,numerical-method ............ 65
R topics documented:

ConstantOutFluxRate_by_PoolIndex-class ........................................... 65
ConstantOutFluxRate_by_PoolName-class ........................................... 65
ConstFc ......................................................................................... 66
ConstFc-class ............................................................................. 66
ConstInFluxes ................................................................................ 67
ConstInFluxes,ConstantInFluxList_by_PoolIndex,numeric-method ............... 67
ConstInFluxes,numeric,ANY-method ................................................. 68
ConstInFluxes-class ....................................................................... 68
ConstLinDecompOp .......................................................................... 69
ConstLinDecompOp.matrix,missing,missing,missing,missing-method .......... 69
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,ConstantOutFluxRateList_by_PoolIndex,numeric-method .................................................. 70
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,missing,numeric,missing-method ................................................................. 70
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolName,missing,character-method .................................. 71
ConstLinDecompOp,missing,missing,ConstantOutFluxRateList_by_PoolIndex,numeric,missing-method ................................................................. 71
ConstLinDecompOp-class .................................................................. 72
ConstLinDecompOpWithLinearScalarFactor .......................................... 72
ConstLinDecompOpWithLinearScalarFactor,matrix,missing,missing,missing,ScalarTimeMap-method ................................................................. 73
ConstLinDecompOpWithLinearScalarFactor-class ................................... 73
ConstLinDecompOp_by_PoolName ........................................................ 74
cycling .......................................................................................... 75
DecompOp-class ............................................................................. 75
DecompositionOperator-class .......................................................... 76
Delta14C ....................................................................................... 76
Delta14C,BoundFc-method ................................................................ 77
Delta14C,ConstFc-method ................................................................ 77
Delta14C_from_AbsoluteFractionModern .......................................... 77
Delta14C_from_AbsoluteFractionModern,matrix-method ....................... 78
Delta14C_from_AbsoluteFractionModern,numeric-method ..................... 78
deSolve.lsoda.wrapper ..................................................................... 79
eCO2 .............................................................................................. 79
deltaRatePerJump .......................................................................... 80
deltaRatePerTime ........................................................................... 81
euler .............................................................................................. 81
example.2DBoundInFluxesFromFunction ............................................. 82
example.2DBoundLinDecompOpFromFunction ....................................... 82
example.2DConstFc.Args .................................................................. 82
example.2DConstInFluxesFromVector ................................................. 83
example.2DGeneralDecompOpArgs .................................................... 83
example.2DInFluxes.Args .................................................................. 83
example.2DUnBoundLinDecompOpFromFunction ................................... 84
example.ConstlinDecompOpFromMatrix ............................................. 84
example.nestedTime2DMatrixList ..................................................... 84
example.Time2DArrayList ................................................................ 84
example.Time3DArrayList ................................................................ 85
example.TimeMapFromArray ........................................................... 85
Fc-class .......................................................................................... 85
FcAtm.from.Dataframe ..................................................................... 86
R topics documented:

- `fT.Arrhenius` ................................................. 86
- `fT.Century1` ................................................ 87
- `fT.Century2` ................................................ 87
- `fT.Daycent1` ................................................ 88
- `fT.Daycent2` ................................................ 89
- `fT.Demeter` .................................................. 89
- `fT.KB` ......................................................... 90
- `fT.LandT` ..................................................... 90
- `fT.linear` .................................................... 91
- `fT.Q10` ........................................................ 91
- `fT.RothC` ..................................................... 92
- `fT.Standcarb` ............................................... 93
- `fW.Candy` .................................................... 93
- `fW.Century` .................................................. 94
- `fW.Daycent1` ............................................... 94
- `fW.Daycent2` ............................................... 95
- `fW.Demeter` .................................................. 96
- `fW.Gompertz` ............................................... 97
- `fW.Moyano` .................................................. 97
- `fW.RothC` .................................................... 98
- `fW.Skopp` .................................................... 98
- `fW.Standcarb` ............................................... 99
- `GaudinskiModel14` ........................................ 100
- `GeneralDecompOp` ......................................... 102
- `GeneralDecompOp,DecompOp-method` ....................... 103
- `GeneralDecompOp,function-method` ......................... 103
- `GeneralDecompOp,list-method` ............................. 103
- `GeneralDecompOp,matrix-method` ........................... 104
- `GeneralDecompOp,TimeMap-method` ........................ 104
- `GeneralModel` ............................................... 104
- `GeneralModel_14` ......................................... 105
- `GeneralNlModel` ............................................ 107
- `GeneralPoolId` .............................................. 108
- `GeneralPoolId,character-method` ......................... 109
- `GeneralPoolId,numeric-method` ........................... 109
- `getAccumulatedRelease` .................................... 110
- `getAccumulatedRelease,Model-method` ................. 110
- `getC` ........................................................ 111
- `getC,Model-method` ....................................... 111
- `getC,Model_by_PoolNames-method` ....................... 112
- `getC,NlModel-method` ..................................... 113
- `getC14` ...................................................... 113
- `getC14,Model_14-method` ................................ 114
- `getCompartmentalMatrixFunc` .............................. 114
- `getCompartmentalMatrixFunc,BoundLinDecompOp,ANY,ANY-method` .................. 115
- `getCompartmentalMatrixFunc,ConstLinDecompOp,ANY,ANY-method` ........ 115
- `getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method` .... 115
- `getCompartmentalMatrixFunc,UnBoundNonLinDecompOp,ANY,ANY-method` .... 116
- `getCompartmentalMatrixFunc,UnBoundNonLinDecompOp_by_PoolNames,character,character-method` .... 116
- `getConstantCompartmentalMatrix` ......................... 117
- `getConstantCompartmentalMatrix,ConstLinDecompOp-method` .... 117
R topics documented:

getConstantCompartmentalMatrix,ConstLinDecompOpWithLinearScalarFactor-method 118
getConstantInFluxVector .............................................. 118
getConstantInFluxVector,ConstInFluxes-method ........................ 119
getConstantInternal FluxRateList by_PoolIndex .......................... 119
getConstantInternal FluxRateList by_PoolIndex,ConstLinDecompOp-method 120
getConstantOutFluxRateList by_PoolIndex ................................ 120
getConstantOutFluxRateList by_PoolIndex,ConstLinDecompOp-method 121
getConstLinDecompOp .................................................. 121
getConstLinDecompOp,ConstLinDecompOpWithLinearScalarFactor-method 122
getCumulativeC .......................................................... 122
getcumulativeC,NlModel-method ........................................ 123
getDecompOp .............................................................. 123
getDecompOp,Model-method .............................................. 124
getDecompOp,NlModel-method ........................................... 124
getDotOut ................................................................. 125
getDotOut,TransportDecompositionOperator-method ..................... 126
getF14 ................................................................. 126
getF14,Model_14-method ............................................... 127
getF14C ................................................................. 127
getF14C,Model_14-method .............................................. 128
getF14R ................................................................. 128
getF14R,Model_14-method .............................................. 129
getFormat ................................................................. 129
getFormat,Fc-method .................................................... 129
getFunctionDefinition .................................................... 130
getFunctionDefinition,ConstInFluxes-method ............................. 130
getFunctionDefinition,ConstLinDecompOp-method ........................ 131
getFunctionDefinition,ConstLinDecompOpWithLinearScalarFactor-method 131
getFunctionDefinition,DecompositionOperator-method .................. 131
getFunctionDefinition,InFluxList by_PoolIndex-method ................ 132
getFunctionDefinition,InFluxList by_PoolName-method ................ 132
getFunctionDefinition,StateDependentInFluxVector-method .......... 133
getFunctionDefinition,TimeMap-method ................................ 133
getFunctionDefinition,TransportDecompositionOperator-method ...... 133
getFunctionDefinition,UnBoundInFluxes-method ........................ 134
getFunctionDefinition,UnBoundLinDecompOp-method ................... 134
getInFluxes ............................................................. 135
getInFluxes,Model-method ............................................. 135
getInFluxes,NlModel-method .......................................... 136
getInitialValues .......................................................... 136
getInitialValues,NlModel-method ...................................... 136
getLinearScaleFactor .................................................... 137
getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method 137
getMeanTransitTime ...................................................... 138
getMeanTransitTime,ConstLinDecompOp-method .......................... 138
getNumberOfPools ....................................................... 139
getNumberOfPools,MCSim-method ....................................... 139
getNumberOfPools,NlModel-method .................................... 140
getNumberOfPools,TransportDecompositionOperator-method .......... 140
getOutputFluxes .......................................................... 140
getOutputFluxes,NlModel-method ...................................... 141
getOutputReceivers ...................................................... 141
R topics documented:

getOutputReceivers,TransportDecompositionOperator,numeric-method 142
getParticleMonteCarloSimulator 142
getParticleMonteCarloSimulator,NlModel-method 143
getReleaseFlux 143
getReleaseFlux,Model-method 144
getReleaseFlux,Model_by_PoolNames-method 144
getReleaseFlux,NlModel-method 145
getReleaseFlux14 145
getReleaseFlux14,Model_14-method 146
getRightHandSideOfODE 146
getRightHandSideOfODE,Model-method 147
getRightHandSideOfODE,Model_by_PoolNames-method 147
getSolution 148
getSolution,Model_by_PoolNames-method 148
g getTimeRange 149
g getTimeRange,ConstInFluxes-method 150
g getTimeRange,ConstLinDecompOp-method 150
g getTimeRange,ConstLinDecompOpWithLinearScalarFactor-method 150
g getTimeRange,DecompositionOperator-method 151
g getTimeRange,TimeMap-method 151
g getTimeRange,UnBoundInFluxes-method 151
g getTimeRange,UnBoundLinDecompOp-method 151
g getTimes 152
g getTimes,Model-method 153
g getTimes,Model_by_PoolNames-method 153
g getTimes,NlModel-method 154
g getTransferCoefficients 154
g getTransferCoefficients,NlModel-method 155
g getTransferCoefficients,TransportDecompositionOperator-method 155
g getTransferMatrix 155
g getTransferMatrixFunc 156
g getTransferMatrixFunc,TransportDecompositionOperator-method 156
g getTransitTimeDistributionDensity 157
g getTransitTimeDistributionDensity,ConstLinDecompOp-method 157
g getValues 158
g getValues,ConstFc-method 158
Graven2017 159
HarvardForest14CO2 160
Hua2013 160
ICBMModel 162
ICBM_N 163
incubation_experiment 164
InFlux 165
InFluxes 166
InFluxes,ConstantInFluxList_by_PoolIndex-method 166
InFluxes,function-method 167
InFluxes,InFluxes-method 167
InFluxes,list-method 167
InFluxes,numeric-method 168
InFluxes,StateIndependentInFluxList_by_PoolIndex-method 168
InFluxes,TimeMap-method 169
InFluxes-class 169
R topics documented:

Model_by_PoolNames-class .................................................. 202
NlModel-class ................................................................. 202
OnepModel ................................................................. 203
OnepModel14 ................................................................. 204
OutFlux ............................................................. 206
OutFluxList_by_PoolIndex .................................................. 206
OutFluxList_by_PoolIndex,list-method ............................... 207
OutFluxList_by_PoolIndex-class ..................................... 207
OutFluxList_by_PoolName .................................................. 207
OutFluxList_by_PoolName,list-method ............................... 208
OutFluxList_by_PoolName-class ..................................... 208
OutFlux_by_PoolIndex ...................................................... 209
OutFlux_by_PoolIndex,function,numeric-method .................. 209
OutFlux_by_PoolIndex-class .......................................... 210
OutFlux_by_PoolName ...................................................... 210
OutFlux_by_PoolName,function,character-method ................ 210
OutFlux_by_PoolName-class .......................................... 211
ParallelModel ................................................................. 211
pathEntropy ................................................................. 212
plot,MCSim-method .......................................................... 213
plot,Model-method .......................................................... 213
plot,Model_by_PoolNames-method .................................... 214
plot,NlModel-method .......................................................... 214
plot,TimeMap-method .......................................................... 214
plotC14Pool ................................................................. 215
plotCPool ................................................................. 215
plotPoolGraph ................................................................. 216
plotPoolGraph,SymbolicModel_by_PoolNames-method ............. 216
plotPoolGraphFromTupleLists ........................................... 217
PoolConnection_by_PoolIndex ............................................. 217
PoolConnection_by_PoolIndex,ANY,ANY,missing-method ........ 218
PoolConnection_by_PoolIndex,missing,missing,character-method 218
PoolConnection_by_PoolIndex-class .................................. 218
PoolConnection_by_PoolName ............................................. 219
PoolConnection_by_PoolName,ANY,ANY,missing-method ........ 219
PoolConnection_by_PoolName-class .................................. 219
PoolId-class ................................................................. 220
PoolIndex ................................................................. 220
PoolIndex,character-method ............................................ 221
PoolIndex,numeric-method ............................................. 221
PoolIndex,Poollndex-method ........................................... 221
PoolIndex,PoolName-method ........................................... 222
PoolIndex-class ............................................................. 222
PoolName ................................................................. 223
PoolName,character-method ............................................ 223
PoolName,Poollndex-method ........................................... 224
PoolName,PoolName-method ........................................... 224
PoolName-class ............................................................. 225
predefinedModels .............................................................. 225
print,NlModel-method ...................................................... 226
RespirationCoefficients .................................................. 226
RothCModel ................................................................. 227
SoilR-package

Description

The package allows you to study compartmental Soil models.

Details

The typical workflow consists of the following steps:

1. Create a model (run)
2. Inspect it

The simplest way of creating a model is to use one of the top level functions for predefined models: predefinedModels. The objects returned by these functions can be of different type, usually either

1. Model
2. Model_14

To inspect the behavior of a model object these classes provide several methods to be found in their respective descriptions. If none of the predefined models fits your needs you can assemble your own model. The functions that create it are the constructors of the above mentioned classes. By convention they have the same name as the class and are described here:

1. Model
2. \texttt{Model\_14}.

There is also a new way to describe models which could be very useful for the definition of nonlinear nonautonomous models with many pools. Parts of the new infrastructure, which is still under development, are used in the source code and examples of:

1. \texttt{CenturyModel}
2. \texttt{WangThreePoolNonAutonomous\_sym}

\begin{itemize}
\item \texttt{AbsoluteFractionModern,BoundFc-method}
\item \texttt{AbsoluteFractionModern,ConstFc-method}
\end{itemize}
## Description

automatic title

## Usage

```r
## S4 method for signature 'ConstFc'
AbsoluteFractionModern(F)
```

### Arguments

- `F`  
  object of class: `ConstFc`, no manual documentation

---

## Description

conversion

## Usage

```r
AbsoluteFractionModern_from_Delta14C(delta14C)
```

### Arguments

- `delta14C`  
  Object to be converted to `AbsoluteFractionModern`

### S4-methods

- `AbsoluteFractionModern_from_Delta14C, matrix-method`
- `AbsoluteFractionModern_from_Delta14C, numeric-method`
AbsoluteFractionModern_from_Delta14C, matrix-method

Description

automatic title

Usage

## S4 method for signature 'matrix'
AbsoluteFractionModern_from_Delta14C(delta14C)

Arguments

delta14C object of class: matrix, no manual documentation

AbsoluteFractionModern_from_Delta14C, numeric-method

Description

automatic title

Usage

## S4 method for signature 'numeric'
AbsoluteFractionModern_from_Delta14C(delta14C)

Arguments

delta14C object of class: numeric, no manual documentation

add_plot automatic title

Description

automatic title

Usage

add_plot(x, ...)

Arguments

x see method arguments
...
see method arguments
S4-methods

- add_plot,TimeMap-method

add_plot,TimeMap-method

 automatic title

Description

automatic title

Usage

## S4 method for signature 'TimeMap'
add_plot(x, ...)

Arguments

x object of class:TimeMap, no manual documentation
...
... no manual documentation

as.character,TimeMap-method

 automatic title

Description

automatic title

Usage

## S4 method for signature 'TimeMap'
as.character(x, ...)

Arguments

x object of class:TimeMap, no manual documentation
...
... no manual documentation
as.numeric(InFluxList_by_PoolName-method)

Convert to a numeric vector with the pool names as names

Description

Convert to a numeric vector with the pool names as names

Usage

## S4 method for signature 'InFluxList_by_PoolName'
as.numeric(x, y, t, time_symbol, ...)

Arguments

x  
object of class: InFluxList_by_PoolName, The list of fluxes. Every element contains a function that depends on a combination of of state variables and time.

y  
A vector indexed by the names of the state variables

t  
a number representing the current point in time

time_symbol  
The name of the time argument used in the definition of the flux functions

...  
no manual documentation

as.numeric,InternalFluxList_by_PoolName-method

Convert to a numeric vector with names of the form ’a->b’

Description

Convert to a numeric vector with names of the form ’a->b’

Usage

## S4 method for signature ’InternalFluxList_by_PoolName’
as.numeric(x, y, t, time_symbol, ...)

Arguments

x  
object of class: InternalFluxList_by_PoolName, The list of fluxes. Every element contains a function that depends on a combination of of state variables and time.

y  
A vector indexed by the names of the state variables

t  
a number representing the current point in time

time_symbol  
The name of the time argument used in the definition of the flux functions

...  
no manual documentation
as.numeric,InternalFlux_by_PoolName-method

*Convert to a numeric value with name of the form ‘a->b’*

**Description**

Convert to a numeric value with name of the form ‘a->b’

**Usage**

```r
## S4 method for signature 'InternalFlux_by_PoolName'
as.numeric(x, y, t, time_symbol, ...)
```

**Arguments**

- `x`: object of class `InternalFlux_by_PoolName`, The list of fluxes. Every element contains a function that depends on a combination of state variables and time.
- `y`: A vector indexed by the names of the state variables
- `t`: a number representing the current point in time
- `time_symbol`: The name of the time argument used in the definition of the flux functions
- `...`: no manual documentation

as.numeric,OutFluxList_by_PoolName-method

*Convert to a numeric vector with the pool names as names*

**Description**

Convert to a numeric vector with the pool names as names

**Usage**

```r
## S4 method for signature 'OutFluxList_by_PoolName'
as.numeric(x, y, t, time_symbol, ...)
```

**Arguments**

- `x`: object of class `OutFluxList_by_PoolName`, The list of fluxes. Every element contains a function that depends on a combination of state variables and time.
- `y`: A vector indexed by the names of the state variables
- `t`: a number representing the current point in time
- `time_symbol`: The name of the time argument used in the definition of the flux functions
- `...`: no manual documentation
availableParticleProperties

automatic title

Description

automatic title

Usage

availableParticleProperties(object)

Arguments

object see method arguments

S4-methods

- availableParticleProperties,MCSim-method

availableParticleProperties,MCSim-method

automatic title

Description

automatic title

Usage

## S4 method for signature 'MCSim'
availableParticleProperties(object)

Arguments

object object of class:MCSim, no manual documentation
**availableParticleSets**  
*automatic title*

---

**Description**

automatic title

**Usage**

```r
availableParticleSets(object)
```

**Arguments**

- `object` see method arguments

---

**S4-methods**

- `availableParticleSets,MCSim-method`

---

**availableParticleSets,MCSim-method**  
*automatic title*

---

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'MCSim'
availableParticleSets(object)
```

**Arguments**

- `object` object of class: `MCSim`, no manual documentation
availableResidentSets

---

**availableResidentSets**  *automatic title*

### Description
automatic title

### Usage
```r
availableResidentSets(object)
```

### Arguments
- **object**  see method arguments

### S4-methods
- *availableResidentSets,MCSim-method*

---

**availableResidentSets,MCSim-method**  *automatic title*

### Description
automatic title

### Usage
```r
## S4 method for signature 'MCSim'
availableResidentSets(object)
```

### Arguments
- **object**  object of class:MCSim, no manual documentation
Implementation of the microbial model AWB (Allison, Wallenstein, Bradford, 2010)

Description

This function implements the microbial model AWB (Allison, Wallenstein, Bradford, 2010), a four-pool model with a microbial biomass, enzyme, SOC and DOC pools. It is a special case of the general nonlinear model.

Usage

```
AWBmodel(  
  t,  
  V_M = 1e+08,  
  V_m = 1e+08,  
  r_B = 2e-04,  
  r_E = 5e-06,  
  r_L = 0.001,  
  a_BS = 0.5,  
  epsilon_0 = 0.63,  
  epsilon_s = -0.016,  
  Km_0 = 500,  
  Km_u0 = 0.1,  
  Km_s = 0.5,  
  Km_us = 0.1,  
  Ea = 47,  
  R = 0.008314,  
  Temp1 = 20,  
  Temp2 = 20,  
  ival = c(B = 2.19159, E = 0.0109579, S = 111.876, D = 0.00144928),  
  I_S = 0.005,  
  I_D = 0.005  
)
```

Arguments

- **t** vector of times (in hours) to calculate a solution.
- **V_M** a scalar representing the maximum rate of uptake (mg DOC cm\(^{-3}\) h\(^{-1}\)). Equivalent to \(V_{\text{maxuptake0}}\) in original paper.
- **V_m** a scalar representing the maximum rate of decomposition of SOM (mg SOM cm\(^{-3}\) h\(^{-1}\)). Equivalent to \(V_{\text{max0}}\) in original paper.
- **r_B** a scalar representing the rate constant of microbial death (h\(^{-1}\)). Equivalent to \(r_{\text{death}}\) in original publication.
- **r_E** a scalar representing the rate constant of enzyme production (h\(^{-1}\)). Equivalent to \(r_{\text{EnzProd}}\) in original publication.
- **r_L** a scalar representing the rate constant of enzyme loss (h\(^{-1}\)). Equivalent to \(r_{\text{EnzLoss}}\) in original publication.
- **a_BS** a scalar representing the fraction of the dead microbial biomass incorporated to SOC. \(\text{MICtoSOC}\) in original publication.
epsilon_0 a scalar representing the intercept of the CUE function (mg mg⁻¹). CUE_0 in original paper.

epsilon_s a scalar representing the slope of the CUE function (degree⁻¹). CUE_slope in original paper.

Km_θ a scalar representing the intercept of the half-saturation constant of SOC as a function of temperature (mg cm⁻³).

Km_uθ a scalar representing the intercept of the half saturation constant of uptake as a function of temperature (mg cm⁻³).

Km_s a scalar representing the slope of the half saturation constant of SOC as a function of temperature (mg cm⁻³ degree⁻¹).

Km_us a scalar representing the slope of the half saturation constant of uptake as a function of temperature (mg cm⁻³ degree⁻¹).

Ea a scalar representing the activation energy (kJ mol⁻¹).

R a scalar representing the gas constant (kJ mol⁻¹ degree⁻¹).

Temp1 a scalar representing the temperature in the output vector.

Temp2 a scalar representing the temperature in the transfer matrix.

ival a vector of length 4 with the initial values for the pools (mg cm⁻³).

I_S a scalar with the inputs to the SOC pool (mg cm⁻³ h⁻¹).

I_D a scalar with the inputs to the DOC pool (mg cm⁻³ h⁻¹).

Details

This implementation contains default parameters presented in Allison et al. (2010).

Value

An object of class NlModel that can be further queried.

References


Examples

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

#Run the model with default parameter values
bcmodel=AWBmodel(t=hours)
Cpools=getC(bcmodel)
##Time solution
# fixme mm:
# the next line causes trouble on Rforge Windows patched build
# matplot(hours,Cpools,type="l",ylab="Concentrations",xlab="Hours",lty=1,ylim=c(0,max(Cpools)*1.2))
##State-space diagram
plot(as.data.frame(Cpools))
```
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

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

Arguments

- \text{t} \quad \text{vector of times (in hours) to calculate a solution.}
- \text{umax} \quad \text{a scalar representing the maximum relative growth rate of bacteria (hr-1) }
- \text{ks} \quad \text{a scalar representing the substrate constant for growth (ug C/ml soil solution)}
- \text{theta} \quad \text{a scalar representing soil water content (ml solution/cm3 soil)}
- \text{Dmax} \quad \text{a scalar representing the maximal relative death rate of bacteria (hr-1) }
- \text{kd} \quad \text{a scalar representing the substrate constant for death of bacteria (ug C/ml soil solution)}
- \text{kr} \quad \text{a scalar representing the fraction of death biomass recycling to substrate (unitless)}
- \text{Y} \quad \text{a scalar representing the yield coefficient for bacteria (ug C/ugC)}
- \text{ival} \quad \text{a vector of length 2 with the initial values for the substrate and the bacterial pools (ug C/cm3)}
- \text{BGF} \quad \text{a scalar representing the constant background flux of substrate (ug C/cm3 soil/hr) }
- \text{ExuM} \quad \text{a scalar representing the maximal exudation rate (ug C/hr cm3 soil))}
- \text{ExuT} \quad \text{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 NlModel that can be further queried.

References


See Also

There are other predefinedModels and also more general functions like Model.

Examples

```r
hours=seq(0,800,0.1)
# Run the model with default parameter values
bcmodel=bacwaveModel(t=hours)
Cpools=getC(bcmodel)
# 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prebomb</td>
<td>A pre-bomb radiocarbon dataset. They could be either IntCal09 or IntCal13.</td>
</tr>
<tr>
<td>postbomb</td>
<td>A post-bomb radiocarbon dataset. They could be any of the datasets in Hua2013.</td>
</tr>
<tr>
<td>time.scale</td>
<td>A character indicating whether to report the results in years before present BP or anno domini AD.</td>
</tr>
</tbody>
</table>

Value

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

automatic title

Usage

BoundFc(format, ...)

Arguments

format see method arguments
...

S4-methods

- BoundFc,character-method
- BoundFc,missing-method

Description

automatic title

Usage

## S4 method for signature 'character'
BoundFc(format, ...)

Arguments

format object of class:character, no manual documentation
...

no manual documentation
Description

automatic title

Usage

```r
## S4 method for signature 'missing'
BoundFc(format, ...)
```

Arguments

- `format`: object of class `missing`, no manual documentation
- `...`: no manual documentation

BoundFc-class

*S4-class to represent atmospheric 14C concentration as scalar function of time.*

Description

As time dependent scalar function which remembers its domain (see `ScalarTimeMap`) and its format.

S4-superclasses (in the package)

- `ScalarTimeMap`
- `Fc`
- `TimeMap`

BoundInFluxes

*constructor for BoundInFluxes*

Description

The method internally calls `TimeMap` and expects the same kind of arguments

Usage

```r
BoundInFluxes(...)```

Arguments

- `...`: passed on to `TimeMap`
BoundInFluxes-class  

automatic title

Description

automatic title

S4-methods

S4-methods with superclasses (in the package) of class BoundInFluxes in their signature::

superclass InFluxes:
  • InFluxes,InFluxes-method

superclass TimeMap:
  • add_plot,TimeMap-method
  • as.character,TimeMap-method
  • GeneralDecompOp,TimeMap-method
  • getFunctionDefinition,TimeMap-method
  • getTimeRange,TimeMap-method
  • InFluxes,TimeMap-method
  • initialize,TimeMap-method
  • plot,TimeMap-method
  • TimeMap,TimeMap,ANY,ANY,ANY,ANY-method

S4-superclasses (in the package)

  • InFluxes
  • TimeMap

BoundLinDecompOp  

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

BoundLinDecompOp(map, ...)

Arguments

map  see method arguments
...
  see method arguments

S4-methods

  • BoundLinDecompOp,ANY-method
  • BoundLinDecompOp,UnBoundLinDecompOp-method
**Description**

The distinction between the classes `BoundLinDecompOp` and `UnBoundLinDecompOp` exist for those functions, that should be only defined for objects of class `UnBoundLinDecomp`. Many functions however do not need extra methods for objects of class `UnBoundLinDecompOp` and just treat it as a `BoundLinDecompOp` which is defined on the complete timeline (-Inf,+Inf).

With its default arguments this function converts its map argument to a `BoundLinDecompOp` with just this domain. This is the most frequent internal use case. If `starttime` and `endtime` are provided the domain of the operator will be restricted `[starttime,endtime]`.

**Usage**

```r
## S4 method for signature 'UnBoundLinDecompOp'
BoundLinDecompOp(map, starttime = -Inf, endtime = Inf)
```

**Arguments**

- `starttime` Begin of time interval map will be restricted to
- `endtime` End of time interval map will be restricted to
BoundLinDecompOp-class

A S4 class to represent a linear compartmental operator defined on time interval

Description

A S4 class to represent a linear compartmental operator defined on time interval

S4-methods

S4-methods with class BoundLinDecompOp in their signature::

• getCompartmentalMatrixFunc,BoundLinDecompOp,ANY,ANY-method

S4-methods with superclasses (in the package) of class BoundLinDecompOp in their signature::

superclass DecompOp:

• GeneralDecompOp,DecompOp-method

superclass TimeMap:

• add_plot,TimeMap-method
• as.character,TimeMap-method
• GeneralDecompOp,TimeMap-method
• getFunctionDefinition,TimeMap-method
• getTimeRange,TimeMap-method
• InFluxes,TimeMap-method
• initialize,TimeMap-method
• plot,TimeMap-method
• TimeMap,TimeMap,ANY,ANY,ANY,ANY-method

S4-superclasses (in the package)

• DecompOp
• TimeMap

by_PoolIndex  automatic title

Description

automatic title

Usage

by_PoolIndex(obj, poolNames, timeSymbol)
Arguments

- **obj**: see method arguments
- **poolNames**: see method arguments
- **timeSymbol**: see method arguments

S4-methods

- `by_PoolIndex,ConstantInFluxRate_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantOutFluxRate_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantInternalFluxRateList_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,InFlux_by_PoolName,character,character-method`
- `by_PoolIndex,InFluxList_by_PoolName,character,character-method`
- `by_PoolIndex,InternalFlux_by_PoolName,character,character-method`
- `by_PoolIndex,InternalFluxList_by_PoolName,character,character-method`
- `by_PoolIndex,OutFlux_by_PoolName,character,character-method`
- `by_PoolIndex,OutFluxList_by_PoolName,character,character-method`
- `by_PoolIndex,PoolConnection_by_PoolName,ANY,ANY-method`

Description

new object with the source pool id converted to a PoolIndex if necessary

Usage

```r
## S4 method for signature 'ConstantInFluxRate_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)
```

```r
## S4 method for signature 'ConstantInFluxRate_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)
```

```r
## S4 method for signature 'ConstantInFluxRate_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)
```

Arguments

- **obj**: object of class:ConstantInFluxRate_by_PoolName, no manual documentation
- **poolNames**: no manual documentation
by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method

convert to a list indexed by pool names

Description
convert to a list indexed by pool names

Usage
## S4 method for signature 'ConstantInternalFluxRateList_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)

Arguments
obj object of class:ConstantInternalFluxRateList_by_PoolName, no manual documentation
poolNames no manual documentation

by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method

new object with the source pool id converted to a PoolName if necessary

Description
new object with the source pool id converted to a PoolName if necessary

Usage
## S4 method for signature 'ConstantInternalFluxRate_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)

Arguments
obj object of class:ConstantInternalFluxRate_by_PoolName, no manual documentation
poolNames no manual documentation
by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-method

convert to a list indexed by pool names

Description

convert to a list indexed by pool names

Usage

## S4 method for signature 'ConstantOutFluxRateList_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)

Arguments

obj object of class:ConstantOutFluxRateList_by_PoolName, no manual documentation

poolNames no manual documentation

by_PoolIndex,ConstantOutFluxRate_by_PoolName,ANY,ANY-method

new object with the source pool id converted to a PoolIndex if necessary

Description

new object with the source pool id converted to a PoolIndex if necessary

Usage

## S4 method for signature 'ConstantOutFluxRate_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)

Arguments

obj object of class:ConstantOutFluxRate_by_PoolName, no manual documentation

poolNames no manual documentation
by_PoolIndex, function, character, character-method

---

**Description**

convert a function f of to f_vec

**Usage**

```r
## S4 method for signature 'function', character, character'
by_PoolIndex(obj, poolNames, timeSymbol)
```

**Arguments**

- **obj**: object of class `function`. For this method a function, whose formal arguments must have names that are elements of the union of poolNames and timeSymbol
- **poolNames**: object of class `character`. The ordered poolnames
- **timeSymbol**: object of class `character`. The name of the argument of obj that represents time.

**Value**

f_vec(vec,t) A new function that extracts the arguments of obj from a complete vector of state variables and the time argument t and applies the original function to these arguments. The ode solvers used by SoilR expect a vector valued function of the state vector and time that represents the derivative. The components of this vector are scalar functions of a vector argument and time. It is possible for the user to define such functions directly, but the definition always depends on the order of state variables. Furthermore these functions usually do not use the complete state vector but only some parts of it. It is much clearer more intuitive and less error prone to be able to define functions that have only formal arguments that are used. This is what this method is used for.

**Examples**

```r
leaf_resp=function(leaf_pool_content){leaf_pool_content*4}
leaf_resp(1)
poolNames=c(
  "some_thing",
  "some_thing_else",
  "some_thing_altogether",
  "leaf_pool_content"
)
leaf_resp_vec=by_PoolIndex(leaf_resp,poolNames,timeSymbol="t")
# The result is the same since the only the forth position in the vector
leaf_resp_vec(c(1,27,3,1),5)
```
Transform pool names to indices

**Description**

Transform pool names to indices

**Usage**

```r
## S4 method for signature 'InFlux_list_by_PoolName,character,character'
by_PoolIndex(obj, poolNames, timeSymbol)
```

**Arguments**

- `obj`: object of class: `InFlux_list_by_PoolName`, no manual documentation
- `poolNames`: object of class: `character`, no manual documentation
- `timeSymbol`: object of class: `character`, no manual documentation

Convert the pool names to indices

**Description**

Convert the pool names to indices

**Usage**

```r
## S4 method for signature 'InFlux_by_PoolName,character,character'
by_PoolIndex(obj, poolNames, timeSymbol)
```

**Arguments**

- `obj`: object of class: `InFlux_by_PoolName`, no manual documentation
- `poolNames`: object of class: `character`, no manual documentation
- `timeSymbol`: object of class: `character`, no manual documentation
by_PoolIndex,InternalFluxList_by_PoolName,character,character-method

Description

automatic title

Usage

## S4 method for signature 'InternalFluxList_by_PoolName,character,character'
by_PoolIndex(obj, poolNames, timeSymbol)

Arguments

obj object of class:InternalFluxList_by_PoolName, no manual documentation
poolNames object of class:character, no manual documentation
timeSymbol object of class:character, no manual documentation

by_PoolIndex,InternalFlux_by_PoolName,character,character-method

Description

automatic title

Usage

## S4 method for signature 'InternalFlux_by_PoolName,character,character'
by_PoolIndex(obj, poolNames, timeSymbol)

Arguments

obj object of class:InternalFlux_by_PoolName, no manual documentation
poolNames object of class:character, no manual documentation
timeSymbol object of class:character, no manual documentation
### by_PoolIndex,OutFluxList_by_PoolName,character,character-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'OutFluxList_by_PoolName,character,character'
by_PoolIndex(obj, poolNames, timeSymbol)
```

**Arguments**

- `obj` object of class `OutFluxList_by_PoolName`, no manual documentation
- `poolNames` object of class `character`, no manual documentation
- `timeSymbol` object of class `character`, no manual documentation

### by_PoolIndex,OutFlux_by_PoolName,character,character-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'OutFlux_by_PoolName,character,character'
by_PoolIndex(obj, poolNames, timeSymbol)
```

**Arguments**

- `obj` object of class `OutFlux_by_PoolName`, no manual documentation
- `poolNames` object of class `character`, no manual documentation
- `timeSymbol` object of class `character`, no manual documentation
by_PoolIndex,PoolConnection_by_PoolName,ANY,ANY-method

constructor from strings of the form 'x->y' new object with the source pool id and the destination pool id guaranteed to be of class PoolIndex

Description

converts the ids if necessary otherwise returns an identical object

Usage

```r
## S4 method for signature 'PoolConnection_by_PoolName,ANY,ANY'
by_PoolIndex(obj, poolNames)
```

Arguments

- `obj`: object of class: PoolConnection_by_PoolName, no manual documentation
- `poolNames`: no manual documentation

by_PoolName

automatic title

Description

automatic title

Usage

```r
by_PoolName(obj, poolNames)
```

Arguments

- `obj`: see method arguments
- `poolNames`: see method arguments

S4-methods

- `by_PoolName,ConstantInFlux_by_PoolIndex-method`
- `by_PoolName,ConstantInFluxList_by_PoolIndex-method`
- `by_PoolName,ConstantInFluxRate_by_PoolIndex-method`
- `by_PoolName,ConstantInternalFluxRate_by_PoolIndex-method`
- `by_PoolName,ConstantInternalFluxRateList_by_PoolIndex-method`
- `by_PoolName,ConstantOutFluxRate_by_PoolIndex-method`
- `by_PoolName,ConstantOutFluxRateList_by_PoolIndex-method`
by_PoolName,ConstantInFluxList_by_PoolIndex-method

convert to a list indexed by pool names

Description

convert to a list indexed by pool names

Usage

## S4 method for signature 'ConstantInFluxList_by_PoolIndex'
by_PoolName(obj, poolNames)

Arguments

obj object of class:ConstantInFluxList_by_PoolIndex, no manual documentation

poolNames no manual documentation

by_PoolName,ConstantInFluxRate_by_PoolIndex-method

new object with the source pool id converted to a PoolIndex if necessary

Description

new object with the source pool id converted to a PoolIndex if necessary

Usage

## S4 method for signature 'ConstantInFluxRate_by_PoolIndex'
by_PoolName(obj, poolNames)

Arguments

obj object of class:ConstantInFluxRate_by_PoolIndex, no manual documentation

poolNames no manual documentation
**by_PoolName,ConstantInFlux_by_PoolIndex-method**

new object with the source pool id converted to a PoolIndex if necessary

---

**Description**

new object with the source pool id converted to a PoolIndex if necessary

**Usage**

```r
## S4 method for signature 'ConstantInFlux_by_PoolIndex'
by_PoolName(obj, poolNames)
```

**Arguments**

- `obj` object of class: `ConstantInFlux_by_PoolIndex`, no manual documentation
- `poolNames` no manual documentation

---

**by_PoolName,ConstantInternalFluxRateList_by_PoolIndex-method**

convert to a list indexed by pool names

---

**Description**

convert to a list indexed by pool names

**Usage**

```r
## S4 method for signature 'ConstantInternalFluxRateList_by_PoolIndex'
by_PoolName(obj, poolNames)
```

**Arguments**

- `obj` object of class: `ConstantInternalFluxRateList_by_PoolIndex`, no manual documentation
- `poolNames` no manual documentation
by_PoolName,ConstantInternalFluxRate_by_PoolIndex-method

Description

new object with the source pool id converted to a PoolIndex if necessary

Usage

## S4 method for signature 'ConstantInternalFluxRate_by_PoolIndex'
by_PoolName(obj, poolNames)

Arguments

obj  object of class:ConstantInternalFluxRate_by_PoolIndex, no manual documentation
poolNames  no manual documentation

by_PoolName,ConstantOutFluxRateList_by_PoolIndex-method

Description

convert to a list indexed by pool names

Usage

## S4 method for signature 'ConstantOutFluxRateList_by_PoolIndex'
by_PoolName(obj, poolNames)

Arguments

obj  object of class:ConstantOutFluxRateList_by_PoolIndex, no manual documentation
poolNames  no manual documentation
Description

This method exists only for classes that do not contain functions of the state_variables since we cannot automatically translate functions with a state vector arguments to functions of the respective state variables which would require symbolic computations. The reverse direction is always possible and is therefore the preferred way to input rate functions that depend on state variables.

Usage

```r
## S4 method for signature 'ConstantOutFluxRate_by_PoolIndex'
by_PoolName(obj, poolNames)
```

Arguments

- `obj`: object of class `ConstantOutFluxRate_by_PoolIndex`, no manual documentation
- `poolNames`: no manual documentation

---

**C14Atm**

*Atmospheric 14C fraction*

Description

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

@note This dataset will be deprecated soon. Please use C14Atm_NH or Hua2013 instead.

Usage

```r
data(C14Atm)
```

Format

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

1. V1 a numeric vector

Examples

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

Post-bomb atmospheric 14C fraction

Description

Atmospheric 14C concentrations for the post-bomb period expressed as Delta 14C in per mile. 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.

1. YEAR a numeric vector with year of measurement.
2. Atmosphere a numeric vector with the Delta 14 value of atmospheric CO2 in per mil.

Examples

plot(C14Atm_NH,type="l")

CenturyModel

Implementation of the Century model

Description

This function implements the Century model as described in Parton et al. (1987).

Usage

CenturyModel(
  t,
  ks = c(STR.surface = 0.076, MET.surface = 0.28, STR.belowground = 0.094,
         MET.belowground = 0.35, ACT = 0.14, SLW = 0.0038, PAS = 0.00013),
  C0 = rep(0, 7),
  surfaceIn,
  soilIn,
  LN,
  Ls,
  clay = 0.2,
  silt = 0.45,
  xi = 1,
  xi.lag = 0,
  solver = deSolve.lsoda.wrapper
)

Arguments

- **t**: A vector containing the points in time where the solution is sought.
- **ks**: A vector of length 7 containing the values of the decomposition rates for the different pools. Units in per week.
- **C0**: A vector of length 7 containing the initial amount of carbon for the 7 pools.
- **surfaceIn**: A scalar or data.frame object specifying the amount of aboveground litter inputs to the soil surface by time (mass per area per week).
- **soilIn**: A scalar or data.frame object specifying the amount of belowground litter inputs to the soil by time (mass per area per week).
- **LN**: A scalar representing the lignin to nitrogen ratio of the plant residue inputs.
- **Ls**: A scalar representing the fraction of structural material that is lignin.
- **clay**: Proportion of clay in mineral soil.
- **silt**: Proportion of silt in mineral soil.
- **xi**: A scalar, data.frame, function or anything that can be converted to a scalar function of time, specifying the external (environmental and/or edaphic) effects on decomposition rates.
- **xi_lag**: A time shift/delay for the automatically created time dependent function xi(t).
- **solver**: A function that solves the system of ODEs. This can be euler or deSolve.lsoda.wrapper or any other user provided function with the same interface.

Details

This is one of the few examples that internally make use of the new infrastructure for flux based descriptions of models (see examples).

Value

A Model Object that can be further queried

References


See Also

- RothCModel. There are other predefinedModels and also more general functions like Model.

Examples

```r
mnths=seq(0,100)
APPT=50 # Assume 50 cm annual precipitation
Pmax=-40+7.7*APPT # Max aboveground production
Rmax=100+7.0*APPT # Max belowground production
abvgIn=Pmax/(Pmax+Rmax)
blgIn=Rmax/(Pmax+Rmax)

cm=CentryModel(t=mnths, surfaceIn = abvgIn, soilIn = blgIn, LN=0.5, Ls=0.1)
```
CenturyModel14

Implementation of a radiocarbon version of the Century model

Description

This function implements a radiocarbon version of the Century model as described in Parton et al. (1987).

Usage

CenturyModel14(
  t,
  ks = 52 * c(STR.surface = 0.076, MET.surface = 0.28, STR.belowground = 0.094,
              MET.belowground = 0.35, ACT = 0.14, SLW = 0.0038, PAS = 0.00013),
  C0 = rep(0, 7),
  surfaceIn,
  soilIn,
  F0_Delta14C,
  LN,
  ls,
  clay = 0.2,
  silt = 0.45,
  xi = 1,
  inputFc,
  lag = 0,
  lambda = -0.0001209681,
  xi_lag = 0,
  solver = deSolve.lsoda.wrapper
)

Arguments

- `t`: A vector containing the points in time where the solution is sought.
- `ks`: A vector of length 7 containing the values of the decomposition rates for the different pools. Units in per year.
- `C0`: A vector of length 7 containing the initial amount of carbon for the 7 pools.
- `surfaceIn`: A scalar or data.frame object specifying the amount of aboveground litter inputs to the soil surface by time (mass per area per year).
- `soilIn`: A scalar or data.frame object specifying the amount of belowground litter inputs to the soil by time (mass per area per year).
- `F0_Delta14C`: A vector of length 7 containing the initial fraction of radiocarbon for the 7 pools in Delta14C format.
- `LN`: A scalar representing the lignin to nitrogen ratio of the plant residue inputs.
Ls  A scalar representing the fraction of structural material that is lignin.
clay Proportion of clay in mineral soil.
silt Proportion of silt in mineral soil.
xi  A scalar, data.frame, function or anything that can be converted to a scalar function of time ScalarTimeMap object 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.
lag  A time shift/delay for the radiocarbon inputs
lambda Radioactive decay constant. By default lamda=-0.0001209681 y^-1 . This has the side effect that all your time related data are treated as if the time unit was year.
xi_lag A time shift/delay for the automatically created time dependent function xi(t)
solver A function that solves the system of ODEs. This can be euler or deSolve lsoda.wrapper or any other user provided function with the same interface.

Value
A Model Object that can be further queried

References

See Also
RothCModel. There are other predefinedModels and also more general functions like Model.

Examples
```
cal_yrs=seq(1900,2015, by=1/12)
APPT=50  # Assume 50 cm annual precipitation
Pmax=40+7.7*APPT  # Max aboveground production
Rmax=100+7.0*APPT  # Max belowground production
abvgIn=52*Pmax/(Pmax+Rmax)
blgIn=52*Rmax/(Pmax+Rmax)
AtmC14=Graven2017[,c("Year.AD", "NH")]

cm=CenturyModel14(t=cal_yrs, surfaceIn = abvgIn, soilIn = blgIn,
F0_Delta14C=rep(0,7), inputFc=AtmC14, LN=0.5, Ls=0.1)
C14t=getF14(cm)

poolNames=c("Surface structural", "Surface metabolic", "Belowground structural",
 "Belowground metabolic", "Active SOM", "Slow SOM", "Passive SOM")
plot(AtmC14, type="l", ylab="Delta 14C (per mil)"
matlines(cal_yrs,C14t, lty=1, col=2:8)
legend("topleft", poolNames, lty=1, col=2:8, bty="n")
```
check_duplicate_pool_names

helper function

Description
Check that poolNames are unique

Usage
check_duplicate_pool_names(poolNames)

Arguments
poolNames character vector which will be tested for duplicates

check_id_length helper function to check that the length of the argument is exactly 1

Description
helper function to check that the length of the argument is exactly 1

Usage
check_id_length(id)

Arguments
id Either a string or a number

check_pool_ids automatic title

Description
automatic title

Usage
check_pool_ids(obj, pools)

Arguments
obj see method arguments
pools see method arguments

S4-methods

• check_pool_ids, PoolConnection_by_PoolIndex, integer-method
check_pool_ids, PoolConnection_by_PoolIndex, integer-method

**Description**
automatic title

**Usage**

```r
## S4 method for signature 'PoolConnection_by_PoolIndex, integer'
check_pool_ids(obj, pools)
```

**Arguments**

- `obj`: object of class: `PoolConnection_by_PoolIndex`, no manual documentation
- `pools`: object of class: `integer`, no manual documentation

---

**computeResults**

**Description**
automatic title

**Usage**

```r
computeResults(object)
```

**Arguments**

- `object`: see method arguments

**S4-methods**

- `computeResults,MCSim-method`
computeResults,MCSim-method

## Description
automatic title

## Usage

```r
## S4 method for signature 'MCSim'
computeResults(object)
```

### Arguments

- `object`: object of class `MCSim`, no manual documentation

---

ConstantInFluxList_by_PoolIndex

*Generic constructor for the class with the same name*

## Description
Generic constructor for the class with the same name

## Usage

```r
ConstantInFluxList_by_PoolIndex(object)
```

### Arguments

- `object`: see method arguments

### S4-methods

- `ConstantInFluxList_by_PoolIndex,ConstInFluxes-method`
- `ConstantInFluxList_by_PoolIndex,list-method`
- `ConstantInFluxList_by_PoolIndex,numeric-method`
ConstantInFluxList_by_PoolIndex,ConstInFluxes-method

**constructor from** ConstInFluxes

---

**Description**

constructor from ConstInFluxes

**Usage**

```r
## S4 method for signature 'ConstInFluxes'
ConstantInFluxList_by_PoolIndex(object)
```

**Arguments**

object  
object of class: ConstInFluxes, An object of class ConstInFluxes

**Value**

An object of class ConstantInFluxList_by_PoolIndex

---

ConstantInFluxList_by_PoolIndex,list-method

**constructor from a normal list**

---

**Description**

constructor from a normal list

**Usage**

```r
## S4 method for signature 'list'
ConstantInFluxList_by_PoolIndex(object)
```

**Arguments**

object  
object of class: list, A list. Either a list of elements of type ConstantInFlux_by_PoolIndex or a list where the names of the elements are strings of the form '1->3' (for the flux rate from pool 1 to 2

**Value**

An object of class ConstantInFluxList_by_PoolIndex

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.
ConstantInFluxList_by_PoolIndex, numeric-method

Description

constructor from numeric vector

Usage

## S4 method for signature 'numeric'
ConstantInFluxList_by_PoolIndex(object)

Arguments

object object of class: numeric, no manual documentation

ConstantInFluxList_by_PoolIndex-class

Subclass of list that is guaranteed to contain only elements of type ConstantInFlux_by_PoolIndex

Description

Subclass of list that is guaranteed to contain only elements of type ConstantInFlux_by_PoolIndex

S4-methods

S4-methods with class ConstantInFluxList_by_PoolIndex in their signature::

• by_PoolName, ConstantInFluxList_by_PoolIndex-method
• ConstInFluxes, ConstantInFluxList_by_PoolIndex, numeric-method
• InFluxes, ConstantInFluxList_by_PoolIndex-method

ConstantInFluxList_by_PoolName

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

ConstantInFluxList_by_PoolName(object)

Arguments

object see method arguments
ConstantInFluxRate_by_PoolName

**Description**

Subclass of list that is guaranteed to contain only elements of type `ConstantIn Flux_by_PoolName`

**ConstantInFluxRate_by_PoolIndex-class**

Describes a flux rates.

**Description**

The purpose is to avoid creation of negative rates or in accidental confusion with fluxes. Instances are usually automatically created from data. If the state variables are known indices can be converted to pool names.

**S4-methods**

**S4-methods with class** ConstantInFluxRate_by_PoolIndex in their signature::

- by_PoolName,ConstantInFluxRate_by_PoolIndex-method

**ConstantInFluxRate_by_PoolName**

Constructor for the class with the same name

**Description**

Constructor for the class with the same name

**Usage**

ConstantInFluxRate_by_PoolName(destinationName, rate_constant)

**Arguments**

- **destinationName**
  
  Index of the receiving pool (positive integer)

- **rate_constant**
  
  Rate (Flux/content) positive real number
ConstantInFluxRate_by_PoolName-class

Describes a flux rates.

Description

The purpose is to avoid creation of negative rates or in accidental confusion with fluxes. Instances are usually automatically created from data. If the state variables are known indices can be converted to pool names.

The purpose is to avoid creation of negative rates or in accidental confusion with fluxes. Instances are usually automatically created from data. If the state variables are known indices can be converted to pool names.

The purpose is to avoid creation of lists that contain negative rates or in accidental confusion with list of fluxes. Instances are usually automatically created from data.

The purpose is to avoid creation of lists that contain negative rates or in accidental confusion with list of fluxes. Instances are usually automatically created from data.

S4-methods

S4-methods with class ConstantInFluxRate_by_PoolName in their signature::

- by_PoolIndex,ConstantInFluxRate_by_PoolName,ANY,ANY-method

ConstantInFlux_by_PoolIndex-class

class for a constant influx to a single pool identified by index

Description

class for a constant influx to a single pool identified by index

S4-methods

S4-methods with class ConstantInFlux_by_PoolIndex in their signature::

- by_PoolName,ConstantInFlux_by_PoolIndex-method

ConstantInFlux_by_PoolName-class

class for a constant influx to a single pool identified by pool name

Description

class for a constant influx to a single pool identified by pool name
Description

Generic constructor for the class with the same name

Usage

\[
\text{ConstantInternalFluxRateList\_by\_PoolIndex}(\text{object})
\]

Arguments

- `object` see method arguments

S4-methods

- `ConstantInternalFluxRateList\_by\_PoolIndex, list\_method`
- `ConstantInternalFluxRateList\_by\_PoolIndex, numeric\_method`

Description

constructor from a normal list

Usage

```
## S4 method for signature 'list'
ConstantInternalFluxRateList\_by\_PoolIndex(\text{object})
```

Arguments

- `object` object of class: `list`, A list. Either a list of elements of type `ConstantInternalFluxRate\_by\_PoolIndex` or a list where the names of the elements are strings of the form `1->3` (for the flux rate from pool 1 to 2)

Value

An object of class `ConstantInternalFluxRateList\_by\_PoolIndex`

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.
ConstantInternalFluxRateList_by_PoolIndex, numeric-method

Description

automatic title

Usage

## S4 method for signature 'numeric'
ConstantInternalFluxRateList_by_PoolIndex(object)

Arguments

object object of class: numeric, no manual documentation

ConstantInternalFluxRateList_by_PoolIndex-class

Describes a list of flux rates.

Description

The purpose is to avoid creation of lists that contain negative rates or in accidental confusion with
list of fluxes. Instances are usually automatically created from data

S4-methods

S4-methods with class ConstantInternalFluxRateList_by_PoolIndex in their signature::

• by_PoolName, ConstantInternalFluxRateList_by_PoolIndex-method
• ConstLinDecompOp, missing, ConstantInternalFluxRateList_by_PoolIndex, ConstantOutFluxRateList_by_PoolIndex-method
• ConstLinDecompOp, missing, ConstantInternalFluxRateList_by_PoolIndex, missing, numeric, missing-

ConstantInternalFluxRateList_by_PoolName

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

ConstantInternalFluxRateList_by_PoolName(object)

Arguments

object see method arguments
S4-methods

• ConstantInternalFluxRateList_by_PoolName,list-method

ConstantInternalFluxRateList_by_PoolName,list-method

Constructor from a normal list of fluxes

Description

Constructor from a normal list of fluxes

Usage

## S4 method for signature 'list'
ConstantInternalFluxRateList_by_PoolName(object)

Arguments

object object of class:list, A list. Either a list of elements of type ConstantInternalFluxRate_by_PoolName or a list where the names of the elements are strings of the form 'somePool->someOtherPool' (for the flux rate from pool somePool to someOtherPool)

Value

An object of class ConstantInternalFluxRateList_by_PoolName

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.

ConstantInternalFluxRateList_by_PoolName-class

Describes a list of flux rates.

Description

The purpose is to avoid creation of lists that contain negative rates or in accidental confusion with list of fluxes. Instances are usually automatically created from data

S4-methods

S4-methods with class ConstantInternalFluxRateList_by_PoolName in their signature::

• by_PoolIndex,ConstantInternalFluxRateList_by_PoolName,ANY,ANY-method
• ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolName
**ConstantInternalFluxRate_by_PoolIndex**

*Generic constructor for the class with the same name*

**Description**

Generic constructor for the class with the same name

**Usage**

```r
ConstantInternalFluxRate_by_PoolIndex(
    sourceIndex,
    destinationIndex,
    src_to_dest,
    rate_constant
)
```

**Arguments**

- `sourceIndex`: see method arguments
- `destinationIndex`: see method arguments
- `src_to_dest`: see method arguments
- `rate_constant`: see method arguments

**S4-methods**

- `ConstantInternalFluxRate_by_PoolIndex,missing,missing,character,numeric-method`
- `ConstantInternalFluxRate_by_PoolIndex,numeric,numeric,missing,numeric-method`

**ConstantInternalFluxRate_by_PoolIndex,missing,missing,character,numeric-method**

*constructor from strings of the form '1_to_2'*

**Description**

constructor from strings of the form '1_to_2'

**Usage**

```r
## S4 method for signature 'missing,missing,character,numeric'
ConstantInternalFluxRate_by_PoolIndex(src_to_dest, rate_constant)
```

**Arguments**

- `src_to_dest`: object of class: character, no manual documentation
- `rate_constant`: object of class: numeric, no manual documentation
ConstantInternalFluxRate_by_PoolIndex-class

Description

automatic title

Usage

```r
## S4 method for signature 'numeric,numeric,missing,numeric'
ConstantInternalFluxRate_by_PoolIndex(
  sourceIndex,
  destinationIndex,
  rate_constant
)
```

Arguments

- `sourceIndex`  object of class: numeric, no manual documentation
- `destinationIndex`  object of class: numeric, no manual documentation
- `rate_constant`  object of class: numeric, no manual documentation

Description

The class is used to dispatch specific methods for the creation of the compartmental matrix which is simplified in case of constant rates.

S4-methods

S4-methods with class ConstantInternalFluxRate_by_PoolIndex in their signature::

- `by_PoolName,ConstantInternalFluxRate_by_PoolIndex-method`

**ConstantInternalFluxRate_by_PoolName**

*Generic constructor for the class with the same name*

---

**Description**

Generic constructor for the class with the same name

**Usage**

```r
ConstantInternalFluxRate_by_PoolName(
    sourceName, 
    destinationName, 
    src_to_dest, 
    rate_constant
)
```

**Arguments**

- `sourceName` see method arguments
- `destinationName` see method arguments
- `src_to_dest` see method arguments
- `rate_constant` see method arguments

**S4-methods**

- `ConstantInternalFluxRate_by_PoolName,character,character,missing,numeric-method`
- `ConstantInternalFluxRate_by_PoolName,missing,missing,character,numeric-method`

---

**ConstantInternalFluxRate_by_PoolName,character,character,missing,numeric-method**

*constructor with argument conversion*

---

**Description**

constructor with argument conversion

**Usage**

```r
## S4 method for signature 'character,character,missing,numeric'
ConstantInternalFluxRate_by_PoolName(
    sourceName, 
    destinationName, 
    rate_constant
)
```
Arguments

sourceName  object of class: character, no manual documentation
destinationName  object of class: character, no manual documentation
rate_constant  object of class: numeric, no manual documentation

Description

constructor from strings of the form 'a->b'

Usage

## S4 method for signature '/quotesingle.Var
missing,missing,character,numeric
/quotesingle.Var
ConstantInternalFluxRate_by_PoolName(src_to_dest, rate_constant)

Arguments

src_to_dest  object of class: character, no manual documentation
rate_constant  object of class: numeric, no manual documentation

Description

S4-class to represent a constant internal flux rate with source and target indexed by name

S4-methods

S4-methods with class ConstantInternalFluxRate_by_PoolName in their signature::

• by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method
ConstantOutFluxRateList_by_PoolIndex

Generic constructor for the class with the same name

Description
Generic constructor for the class with the same name

Usage
ConstantOutFluxRateList_by_PoolIndex(object)

Arguments
object see method arguments

S4-methods
• ConstantOutFluxRateList_by_PoolIndex,list-method
• ConstantOutFluxRateList_by_PoolIndex,numeric-method

ConstantOutFluxRateList_by_PoolIndex,list-method
constructor from a normal list

Description
constructor from a normal list

Usage
## S4 method for signature 'list'
ConstantOutFluxRateList_by_PoolIndex(object)

Arguments
object object of class:list, A list. Either a list of elements of type ConstantOutFluxRate_by_PoolIndex or a list where the names of the elements are integer strings of the form '3' (for the flux rate from pool 3)

Value
An object of class ConstantOutFluxRateList_by_PoolIndex

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.
ConstantOutFluxRateList_by_PoolIndex, numeric-method

Description

automatic title

Usage

## S4 method for signature 'numeric'
ConstantOutFluxRateList_by_PoolIndex(object)

Arguments

object object of class numeric, no manual documentation

ConstantOutFluxRateList_by_PoolIndex-class

Describes a list of flux rates.

Description

The purpose is to avoid creation of lists that contain negative rates or in accidental confusion with list of fluxes. Instances are usually automatically created from data

S4-methods

S4-methods with class ConstantOutFluxRateList_by_PoolIndex in their signature::

• by_PoolName, ConstantOutFluxRateList_by_PoolIndex-method
• ConstLinDecompOp, missing, ConstantInternalFluxRateList_by_PoolIndex, ConstantOutFluxRateList_by_PoolName, numeric-method
• ConstLinDecompOp, missing, missing, ConstantOutFluxRateList_by_PoolIndex, numeric, missing-method

ConstantOutFluxRateList_by_PoolName

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

ConstantOutFluxRateList_by_PoolName(object)

Arguments

object see method arguments
S4-methods

- ConstantOutFluxRateList_by_PoolName,list-method
- ConstantOutFluxRateList_by_PoolName,numeric-method

Description

constructor from a normal list

Usage

```r
## S4 method for signature 'list'
ConstantOutFluxRateList_by_PoolName(object)
```

Arguments

- `object`: object of class: `list`. A list. Either a list of elements of type `ConstantOutFluxRate_by_PoolName` or a list where the names of the elements are integer strings of the form ‘3’ (for the flux rate from pool 3)

Value

An object of class `ConstantOutFluxRateList_by_PoolName`

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.

Description

automatic title

Usage

```r
## S4 method for signature 'numeric'
ConstantOutFluxRateList_by_PoolName(object)
```

Arguments

- `object`: object of class: `numeric`, no manual documentation
ConstantOutFluxRateList_by_PoolName-class

Describes a list of flux rates.

Description

The purpose is to avoid creation of lists that contain negative rates or in accidental confusion with list of fluxes. Instances are usually automatically created from data.

S4-methods

S4-methods with class ConstantOutFluxRateList_by_PoolName in their signature::

- by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-method
- ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolName-method

ConstantOutFluxRate_by_PoolIndex

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

ConstantOutFluxRate_by_PoolIndex(sourceIndex, rate_constant)

Arguments

sourceIndex     see method arguments
rate_constant   see method arguments

S4-methods

- ConstantOutFluxRate_by_PoolIndex,numeric,numeric-method
ConstantOutFluxRate_by_PoolIndex, numeric, numeric-method

Description

automatic title

Usage

## S4 method for signature 'numeric, numeric'
ConstantOutFluxRate_by_PoolIndex(sourceIndex, rate_constant)

Arguments

sourceIndex object of class: numeric, no manual documentation
rate_constant object of class: numeric, no manual documentation

ConstantOutFluxRate_by_PoolIndex-class

S4 Class to represent a single constant out-flux rate with the source pool specified by an index

Description

S4 Class to represent a single constant out-flux rate with the source pool specified by an index

S4-methods

S4-methods with class ConstantOutFluxRate_by_PoolIndex in their signature:

• by_PoolIndex, ConstantOutFluxRate_by_PoolIndex-method

ConstantOutFluxRate_by_PoolName-class

S4 Class to represent a single constant out-flux rate with the source pool specified by name

Description

S4 Class to represent a single constant out-flux rate with the source pool specified by name

S4-methods

S4-methods with class ConstantOutFluxRate_by_PoolName in their signature:

• by_PoolIndex, ConstantOutFluxRate_by_PoolName, ANY, ANY-method
constfc-class

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.

constfc-class

S4 class representing a constant ^14C fraction

Description

S4 class representing a constant ^14C fraction

S4-methods

S4-methods with class ConstFc in their signature::

- AbsoluteFractionModern,ConstFc-method
- Delta14C,ConstFc-method
- getValues,ConstFc-method

S4-methods with superclasses (in the package) of class ConstFc in their signature::

superclass Fc:
- getFormat,Fc-method

S4-superclasses (in the package)

- Fc
**ConstInFluxes**

### Description

automatic title

### Usage

```r
ConstInFluxes(map, numberOfPools)
```

### Arguments

- **map**  
  see method arguments
- **numberOfPools**  
  see method arguments

### S4-methods

- `ConstInFluxes,ConstantInFluxList_by_PoolIndex,numeric-method`
- `ConstInFluxes,numeric,ANY-method`

---

**ConstInFluxes,ConstantInFluxList_by_PoolIndex,numeric-method**

### Description

automatic title

### Usage

```r
## S4 method for signature 'ConstantInFluxList_by_PoolIndex,numeric'
ConstInFluxes(map, numberOfPools)
```

### Arguments

- **map**  
  object of class: `ConstantInFluxList_by_PoolIndex`, no manual documentation
- **numberOfPools**  
  object of class: `numeric`, no manual documentation
### Description

It is mainly used to dispatch S4-methods for computations that are valid only if the influx is constant. This knowledge can either be used to speed up computations or to decide if they are possible at all. E.g. the computation of equilibria for a model run requires autonomy of the model which requires the influxes to be time independent. If the model is linear and compartmental then the (unique) equilibrium can be computed. Accordingly a method with ConstInFluxes in the signature can be implemented, whereas none would be available for a general InFluxes argument.

### S4-methods

**S4-methods with class ConstInFluxes in their signature:**

- ConstantInFluxList_by_PoolIndex,ConstInFluxes-method
- getConstantInFluxVector,ConstInFluxes-method
- getFunctionDefinition,ConstInFluxes-method
- getTimeRange,ConstInFluxes-method

**S4-methods with superclasses (in the package) of class ConstInFluxes in their signature:**

**superclass InFluxes:**

- InFluxes,InFluxes-method
ConstLinDecompOp

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic constructor for the class with the same name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConstLinDecompOp(</td>
</tr>
<tr>
<td>mat,</td>
</tr>
<tr>
<td>internal_flux_rates,</td>
</tr>
<tr>
<td>out_flux_rates,</td>
</tr>
<tr>
<td>numberOfPools,</td>
</tr>
<tr>
<td>poolNames</td>
</tr>
<tr>
<td>)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat see method arguments</td>
</tr>
<tr>
<td>internal_flux_rates see method arguments</td>
</tr>
<tr>
<td>out_flux_rates see method arguments</td>
</tr>
<tr>
<td>numberOfPools see method arguments</td>
</tr>
<tr>
<td>poolNames see method arguments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S4-methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>* ConstLinDecompOp,matrix,missing,missing,missing,missing,missing-method</td>
</tr>
<tr>
<td>* ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,ConstantOutFluxRateList_by_PoolIndex-method</td>
</tr>
<tr>
<td>* ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,missing,numeric,missing-method</td>
</tr>
<tr>
<td>* ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolIndex-method</td>
</tr>
<tr>
<td>* ConstLinDecompOp,missing,missing,ConstantOutFluxRateList_by_PoolIndex,numeric,missing-method</td>
</tr>
<tr>
<td>* ConstLinDecompOp,matrix,missing,missing,missing,missing-method</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>## S4 method for signature 'matrix,missing,missing,missing,missing'</td>
</tr>
<tr>
<td>ConstLinDecompOp(mat)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat object of class:matrix, no manual documentation</td>
</tr>
</tbody>
</table>
Constructor

Description
Constructor

Usage
## S4 method for signature
## 'missing',
## ConstantInternalFluxRateList_by_PoolIndex,
## ConstantOutFluxRateList_by_PoolIndex,
## numeric,
## 'missing'
ConstLinDecompOp(internal_flux_rates, out_flux_rates, numberOfPools)

Arguments
internal_flux_rates
object of class: ConstantInternalFluxRateList_by_PoolIndex, no manual documentation
out_flux_rates
object of class: ConstantOutFluxRateList_by_PoolIndex, no manual documentation
numberOfPools
object of class: numeric, no manual documentation
alternative Constructor with pool names

Description
alternative Constructor with pool names

Usage

## S4 method for signature
## 'missing, ConstantInternalFluxRateList_by_PoolName, ConstantOutFluxRateList_by_PoolName, missing, character'
ConstLinDecompOp(internal_flux_rates, out_flux_rates, poolNames)

Arguments

internal_flux_rates  object of class: ConstantInternalFluxRateList_by_PoolName, no manual documentation
out_flux_rates       object of class: ConstantOutFluxRateList_by_PoolName, no manual documentation
poolNames           object of class: character, no manual documentation

Constructor

Description
Constructor

Usage

## S4 method for signature
## 'missing, missing, ConstantOutFluxRateList_by_PoolIndex, numeric, missing'
ConstLinDecompOp(out_flux_rates, numberOfPools)

Arguments

out_flux_rates  object of class: ConstantOutFluxRateList_by_PoolIndex, no manual documentation
numberOfPools   object of class: numeric, no manual documentation
**ConstLinDecompOp-class**

A class to represent a constant (=nonautonomous, linear) compartmental matrix or equivalently a combination of ordered constant internal flux rates and constant out flux rates.

---

**Description**

A class to represent a constant (=nonautonomous, linear) compartmental matrix or equivalently a combination of ordered constant internal flux rates and constant out flux rates.

---

**S4-methods**

**S4-methods with class** ConstLinDecompOp in their signature:

- `getCompartmentalMatrixFunc(ConstLinDecompOp, ANY, ANY-method)
- `getConstantCompartmentalMatrix(ConstLinDecompOp-method)
- `getConstantInternalFluxRateList_by_PoolIndex(ConstLinDecompOp-method)
- `getConstantOutFluxRateList_by_PoolIndex(ConstLinDecompOp-method)
- `getFunctionDefinition(ConstLinDecompOp-method)
- `getMeanTransitTime(ConstLinDecompOp-method)
- `getTimeRange(ConstLinDecompOp-method)
- `getTransitTimeDistributionDensity(ConstLinDecompOp-method)
- `initialize(ConstLinDecompOp-method)

**S4-methods with superclasses (in the package) of class** ConstLinDecompOp in their signature:

- `GeneralDecompOp, Decompo-method`

---

**S4-superclasses (in the package)**

- `DecompOp`

---

**ConstLinDecompOpWithLinearScalarFactor**

Generic constructor for the class with the same name

---

**Description**

Generic constructor for the class with the same name

---

**Usage**

```r
ConstLinDecompOpWithLinearScalarFactor(
  mat,
  internal_flux_rates,
  out_flux_rates,
  numberOfPools,
  xi
)
```
Arguments

- **mat**: see method arguments
- **internal_flux_rates**: see method arguments
- **out_flux_rates**: see method arguments
- **numberOfPools**: see method arguments
- **xi**: see method arguments

S4-methods

- **ConstLinDecompOpWithLinearScalarFactor, matrix, missing, missing, missing, ScalarTimeMap-method**

Description

 convert names of vectors or lists to class `ConstantOutFluxRate` convert names of vectors or lists to class `ConstantInternalFluxRate`

Usage

```r
## S4 method for signature 'matrix, missing, missing, missing, ScalarTimeMap'
ConstLinDecompOpWithLinearScalarFactor(mat, xi)
```

Arguments

- **mat**: object of class: `matrix`, no manual documentation
- **xi**: object of class: `ScalarTimeMap`, no manual documentation

Description

A class to represent a constant (=nonautonomous,linear) compartmental matrix with a time dependent (linear) scalar pre factor This is a special case of a linear compartmental operator/matrix
S4-methods

S4-methods with class `ConstLinDecompOpWithLinearScalarFactor` in their signature::

- `getConstantCompartmentalMatrix`, `ConstLinDecompOpWithLinearScalarFactor-method`
- `getConstLinDecompOp`, `ConstLinDecompOpWithLinearScalarFactor-method`
- `getFunctionDefinition`, `ConstLinDecompOpWithLinearScalarFactor-method`
- `getLinearScaleFactor`, `ConstLinDecompOpWithLinearScalarFactor-method`
- `getTimeRange`, `ConstLinDecompOpWithLinearScalarFactor-method`

S4-methods with superclasses (in the package) of class `ConstLinDecompOpWithLinearScalarFactor` in their signature::

superclass `DecompOp`:

- `GeneralDecompOp`, `DecompOp-method`

S4-superclasses (in the package)

- `DecompOp`

---

**ConstLinDecompOp_by_PoolName**

*Generic constructor for the class with the same name*

---

**Description**

Generic constructor for the class with the same name

**Usage**

`ConstLinDecompOp_by_PoolName(internal_flux_rates, out_flux_rates, poolNames)`

**Arguments**

- `internal_flux_rates` see method arguments
- `out_flux_rates` see method arguments
- `poolNames` see method arguments
cycling

Cycling analysis of compartmental matrices

Description
Computes the fundamental matrix N, and the expected number of steps from a compartmental matrix A

Usage
`cycling(A)`

Arguments
- A A compartmental linear square matrix with cycling rates in the diagonal and transfer rates in the off-diagonal.

Value
A list with 2 objects: the fundamental matrix N, and the expected number of steps Et.

See Also
- `systemAge`
- `DecompOp-class`
- `S4-methods`
- `S4-subclasses`

DecompOp-class S4-class to represent compartmental operators

Description
S4-class to represent compartmental operators

S4-methods
S4-methods with class DecompOp in their signature::
- GeneralDecompOp, DecompoOp-method

S4-subclasses
- DecompositionOperator
- ConstLinDecompOp
- ConstLinDecompOpWithLinearScalarFactor
- BoundLinDecompOp
- UnBoundLinDecompOp
- UnBoundNonLinDecompOp
DecompositionOperator-class

**automatic title**

---

**Description**

automatic title

**S4-methods**

**S4-methods with class DecompositionOperator in their signature::**

- getFunctionDefinition, DecompositionOperator-method
- getTimeRange, DecompositionOperator-method
- initialize, DecompositionOperator-method

**S4-methods with superclasses (in the package) of class DecompositionOperator in their signature::**

* superclass DecompOp:
  - GeneralDecompOp, DecompOp-method

**S4-superclasses (in the package)**

- DecompOp

---

**Delta14C**

**automatic title**

---

**Description**

automatic title

**Usage**

Delta14C(F)

**Arguments**

F see method arguments

**S4-methods**

- Delta14C,BoundFc-method
- Delta14C,ConstFc-method
Delta14C,BoundFc-method

Description

automatic title

Usage

## S4 method for signature 'BoundFc'
Delta14C(F)

Arguments

F object of class:BoundFc, no manual documentation

Delta14C,ConstFc-method

Description

automatic title

Usage

## S4 method for signature 'ConstFc'
Delta14C(F)

Arguments

F object of class:ConstFc, no manual documentation

Delta14C_from_AbsoluteFractionModern

Description

automatic title

Usage

Delta14C_from_AbsoluteFractionModern(AbsoluteFractionModern)
Arguments

AbsoluteFractionModern

see method arguments

S4-methods

- Delta14C_from_AbsoluteFractionModern,matrix-method
- Delta14C_from_AbsoluteFractionModern,numeric-method

Delta14C_from_AbsoluteFractionModern,matrix-method

automatic title

Description

automatic title

Usage

## S4 method for signature 'matrix'
Delta14C_from_AbsoluteFractionModern(AbsoluteFractionModern)

Arguments

AbsoluteFractionModern

object of class:matrix, no manual documentation

Delta14C_from_AbsoluteFractionModern,numeric-method

automatic title

Description

automatic title

Usage

## S4 method for signature 'numeric'
Delta14C_from_AbsoluteFractionModern(AbsoluteFractionModern)

Arguments

AbsoluteFractionModern

object of class:numeric, no manual documentation
**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.

---

**eCO2**

**Soil CO2 efflux from an incubation experiment**

**Description**

A dataset with soil CO2 efflux measurements from a laboratory incubation at controlled temperature and moisture conditions.

**Usage**

data(eCO2)

**Format**

A data frame with the following 3 variables.

- **Days**: A numeric vector with the day of measurement after the experiment started.
- **eCO2mean**: A numeric vector with the release flux of CO2. Units in ug C g\(^{-1}\) soil day\(^{-1}\).
- **eCO2sd**: A numeric vector with the standard deviation of the release flux of CO2-C. Units in ug C g\(^{-1}\) soil day\(^{-1}\).
A laboratory incubation experiment was performed in March 2014 for a period of 35 days under controlled conditions of temperature (15 degrees Celsius), moisture (30 percent soil water content), and oxygen levels (20 percent). Soil CO2 measurements were taken using an automated system for gas sampling connected to an infrared gas analyzer. The soil was sampled at a boreal forest site (Caribou Poker Research Watershed, Alaska, USA). This dataset presents the mean and standard deviation of 4 replicates.

**Examples**

```r
head(eCO2)
plot(eCO2[,1:2],type="o",ylim=c(0,50),ylab="CO2 efflux (ug C g^-1 soil day^-1)")
arrows(eCO2[,1],eCO2[,2]-eCO2[,3],eCO2[,1],eCO2[,2]+eCO2[,3], angle=90,length=0.3,code=3)
```

<table>
<thead>
<tr>
<th>entropyRatePerJump</th>
<th>Entropy rate per jump</th>
</tr>
</thead>
</table>

**Description**

Computes the entropy rate per jump of the Markov chain generated by the compartmental system.

**Usage**

```r
entropyRatePerJump(A, u)
```

**Arguments**

- **A**: A constant compartmental square matrix with cycling rates in the diagonal and transfer rates in the off-diagonal.
- **u**: A one-column matrix defining the amount of inputs per compartment.

**Value**

A scalar value with the entropy rate per jump.

**References**


**Examples**

```r
B6=matrix(c(-1,1,0,0,-1,1,0,0,-1),3,3); u6=matrix(c(1,0,0))
entropyRatePerJump(A=B6, u=u6)
```
Description

Computes the entropy rate per time of the Markov chain generated by the compartmental system.

Usage

entropyRatePerTime(A, u)

Arguments

A  
A constant compartmental square matrix with cycling rates in the diagonal and transfer rates in the off-diagonal.

u  
A one-column matrix defining the amount of inputs per compartment.

Value

A scalar value with the entropy rate per time.

References


Examples

B6=matrix(c(-1,1,0,0,-1,1,0,0,-1),3,3); u6=matrix(c(1,0,0))
entropyRatePerTime(A=B6, u=u6)

euler

description

This function can solve arbitrary first order ode systems with the euler forward method and an adaptive time-step size control given a tolerance for the deviation of a coarse and fine estimate of the change in y for the next time step. It is an alternative to deSolve.lsoda.wrapper and has the same interface. It is much slower than ode and should probably be considered less capable in solving stiff ode systems. However it has one main advantage, which consists in its simplicity. It is quite easy to see what is going on inside it. Whenever you don’t trust your implementation of another (more efficient but probably also more complex) ode solver, just compare the result to what this method computes.

Usage

euler(times, ydot, startValues)
Arguments

- **times**: 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 initial values.

---

**Description**

Create a 2-dimensional example of a BoundInFluxes object

**Usage**

`example.2DBoundInFluxesFromFunction()` 

**Value**

The returned object represents a time dependent Influx into a two pool model.

---

**Description**

An example used in tests and other examples.

**Usage**

`example.2DBoundLinDecompOpFromFunction()` 

---

**Description**

Create a 2-dimensional examples of a Influx objects from different arguments

**Usage**

`example.2DConstFc.Args()`
**example.2DConstInFluxesFromVector**

2D example for constant Influx

---

**Description**

An example used in tests and other examples.

**Usage**

```python
example.2DConstInFluxesFromVector()
```

**Value**

The returned object represents a time invariant constant influx into a two pool model.

---

**example.2DGeneralDecompOpArgs**

**example.2DGeneralDecompOpArgs**

---

**Description**

We present all possibilities to define a 2D `DecompOp-class`

**Usage**

```python
example.2DGeneralDecompOpArgs()
```

---

**example.2DInFluxes.Args**

**example.2DInFluxes.Args**

---

**Description**

Create a 2-dimensional examples of a Influx objects from different arguments

**Usage**

```python
example.2DInFluxes.Args()
```
example.2DUnBoundLinDecompOpFromFunction

**Description**

An example used in tests and other examples.

**Usage**

```java
example.2DUnBoundLinDecompOpFromFunction()
```

example.ConstlinDecompOpFromMatrix

**Description**

An example used in tests and other examples.

**Usage**

```java
example.ConstlinDecompOpFromMatrix()
```

example.nestedTime2DMatrixList

`create an example nested list that can be`

**Description**

An example used in tests and other examples.

**Usage**

```java
example.nestedTime2DMatrixList()
```

example.Time2DArrayList

`create an example TimeMap from 2D array`

**Description**

An example used in tests and other examples.

**Usage**

```java
example.Time2DArrayList()
```
example.Time3DArrayList

create an example TimeFrame from 3D array

Description

An example used in tests and other examples.

Usage

example.Time3DArrayList()

example.TimeMapFromArray

create an example TimeFrame from 3D array

Description

The function creates an example TimeMap that is used in other examples and tests.

Usage

example.TimeMapFromArray()

Fc-class

automatic title

Description

automatic title

S4-methods

S4-methods with class Fc in their signature::

• getFormat,Fc-method

S4-subclasses

• BoundFc
• ConstFc
**FcAtm.from.Dataframe**

**Description**

This function is deprecated constructor of the deprecated class FcAtm

**Usage**

FcAtm.from.Dataframe(dframe, lag = 0, interpolation = splinefun, format)

**Arguments**

dframe
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)

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.

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

**Value**

An object of the new class BoundFc that replaces FcAtm

**fT.Arrhenius**

**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 contents in units of J K^-1 mol^-1.
Value
A scalar or a vector containing the effects of temperature on decomposition rates (unitless).

Usage
\[ fT.Century1(\text{Temp}, \text{Tmax} = 45, \text{Topt} = 35) \]

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).

References

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

Usage
\[ fT.Century2(\text{Temp}, \text{Tmax} = 45, \text{Topt} = 35) \]
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).

References


### fT.Daycent2

**Effects of temperature on decomposition rates according to the DAYCENT model**

**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).

**References**

### fT.Demeter

**Effects of temperature on decomposition rates according to the DEMETER model**

**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).

**References**
**fT.KB**

*Effects of temperature on decomposition rates according to a model proposed by M. Kirschbaum (1995)*

**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).

**References**


---

**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).
References


---

**fT.linear**

*Effects of temperature on decomposition rates according to a linear model*

**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).

**References**


---

**fT.Q10**

*Effects of temperature on decomposition rates according to a Q10 function*

**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).

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 $\leq -18.3$

References

**fT.Standcarb**  
*Effects of temperature on decomposition rates according to the StandCarb model*

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

**Usage**
```r
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.

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

**References**

---

**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**
```r
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.
References


---

**fW.Century**

*Effects of moisture on decomposition rates according to the CENTURY model*

**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).

**References**


---

**fW.Daycent1**

*Effects of moisture on decomposition rates according to the DAYCENT model*

**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.

References


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).

References


---

**fW.Demeter**

*Effects of moisture on decomposition rates according to the DEMETER model*

Description

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

Usage

```r
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).

References

\textbf{fW.Gompertz}  
\textit{Effects of moisture on decomposition rates according to the Gompertz function}

\textbf{Description}
Calculates the effects of water content on decomposition rates.

\textbf{Usage}
\begin{verbatim}
fW.Gompertz(theta, a = 0.824, b = 0.308)
\end{verbatim}

\textbf{Arguments}
- theta: A scalar or vector containing values of volumetric soil water content.
- a: Empirical parameter
- b: Empirical parameter

\textbf{References}

\textbf{fW.Moyano}  
\textit{Effects of moisture on decomposition rates according to the function proposed by Moyano et al. (2013)}

\textbf{Description}
Calculates the effects of water content on decomposition rates.

\textbf{Usage}
\begin{verbatim}
fW.Moyano(theta, a = 3.11, b = 2.42)
\end{verbatim}

\textbf{Arguments}
- theta: A scalar or vector containing values of volumetric soil water content.
- a: Empirical parameter
- b: Empirical parameter

\textbf{References}
Function: 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**
```r
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.

**References**

Function: fW.Skopp

**Effects of moisture on decomposition rates according to the function proposed by Skopp et al. 1990**

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

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

- \( r_{WC} \): relative water content
- \( \alpha \): Empirical parameter
- \( \beta \): Empirical parameter
- \( f \): Empirical parameter
- \( g \): Empirical parameter

References


fw.Standcarb

Effects of moisture on decomposition rates according to the StandCarb model

Description

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

Usage

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

Arguments

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

References


---

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.
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 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. An alternative to the default is euler 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

References

See Also
There are other predefinedModels and also more general functions like Model.

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)"))), x1im=c(1940,2010)
lines(years,C14m,col=4)
points(HarvardForest14CO2[1:11,1],HarvardForest14CO2[1:11,2],pch=19,cex=0.5)
points(HarvardForest14CO2[12:173,1],HarvardForest14CO2[12:173,2],pch=19,col=2,cex=0.5)
points(HarvardForest14CO2[158,1],HarvardForest14CO2[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
)

---

GeneralDecompOp  A generic factory for subclasses of GeneralDecompOp

Description

A generic factory for subclasses of GeneralDecompOp

Usage

GeneralDecompOp(object)

Arguments

object see method arguments

S4-methods

- GeneralDecompOp,DecompOp-method
- GeneralDecompOp,function-method
- GeneralDecompOp,list-method
- GeneralDecompOp,matrix-method
- GeneralDecompOp,TimeMap-method
Description

This method takes and returns an (identical) object that inherits from \texttt{DecompOp}. It's purpose is to be able to call the generic function on arguments that are already

Usage

\begin{verbatim}
## S4 method for signature 'DecompOp'
GeneralDecompOp(object)
\end{verbatim}

Arguments

- \texttt{object} object of class: \texttt{DecompOp}, An object that already is of class \texttt{DecompOp}

Description

automatic title

Usage

\begin{verbatim}
## S4 method for signature `function`
GeneralDecompOp(object)
\end{verbatim}

Arguments

- \texttt{object} object of class: \texttt{function}, no manual documentation

Description

automatic title

Usage

\begin{verbatim}
## S4 method for signature 'list'
GeneralDecompOp(object)
\end{verbatim}

Arguments

- \texttt{object} object of class: \texttt{list}, no manual documentation
GeneralDecompOp, matrix-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'matrix'
GeneralDecompOp(object)
```

**Arguments**

- `object` object of class `matrix`, no manual documentation

GeneralDecompOp, TimeMap-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'TimeMap'
GeneralDecompOp(object)
```

**Arguments**

- `object` object of class `TimeMap`, no manual documentation

GeneralModel

**Description**

In previous SoilR Version GeneralModel was the function to create linear models, a task now fulfilled by the function `Model`. To ensure backward compatibility this function remains as a wrapper. In future versions it might take on the role of an abstract factory that produces several classes of models (i.e. autonomous or non-autonomous and linear or non-linear) depending on different combinations of arguments. It creates 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.
Usage

GeneralModel(
  t,
  A,
  ivList,
  inputFluxes,
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE,
  timeSymbol
)

Arguments

t A vector containing the points in time where the solution is sought.

A Anything that can be converted by GeneralDecompOp 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 InFluxes 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

timeSymbol A string (character vector of length 1) identifying the variable name

Value

A model object that can be further queried.

See Also

TwopParallelModel, TwopSeriesModel, TwopFeedbackModel

GeneralModel_14 create objects of class Model_14

Description

At the moment this is just a wrapper for the actual constructor Model_14 with additional support for some now deprecated parameters for backward compatibility. This role may change in the future to an abstract factory where the actual class of the created model will be determined by the supplied parameters.
Usage

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

Arguments

t          A vector containing the points in time where the solution is sought.
A          something that can be converted by GeneralDecompOp to any of the available subclasses of DecompOp.
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.
initialValF An object equal or equivalent to 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 something that can be converted by InFluxes to any of the available subclasses of InFluxes.
Fc          deprecated keyword argument, please use inputFc instead
inputFc     An object describing the fraction of C_14 in per mille (different formats are possible)
di          the rate at which C_14 decays radioactively. If you don’t provide a value here we assume the following value: k=-0.0001209681 y^-1. This has the side effect that all your time related data are treated as if the time unit was year. Thus beside time itself it also affects decay rates the inputrates and the output
solverfunc  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.
pass        Forces the constructor to create the model even if it is invalid

Value

A model object that can be further queried.

See Also

TwopParallelModel, TwopSeriesModel, TwopFeedbackModel
GeneralNlModel

Use this function to create objects of class NlModel.

Description

The function creates a numerical model for n arbitrarily connected pools. It is one of the constructors of class NlModel. It is used by some more specialized wrapper functions, but can also be used directly.

Usage

GeneralNlModel(
  t,
  TO,
  ivList,
  inputFluxes,
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE
)

Arguments

t A vector containing the points in time where the solution is sought.

TO A object describing the model decay rates for the n pools, connection and feedback coefficients. The number of pools n must be consistent with the number of initial values and input fluxes.

ivList A numeric vector containing the initial amount of carbon for the n pools. The length of this vector is equal to the number of pools.

inputFluxes A TimeMap object consisting of a vector valued function describing the inputs to the pools as functions of time TimeMap.new.

solverfunc The function used by to actually solve the ODE system.

pass Forces the constructor to create the model even if it is invalid. If set to TRUE, does not enforce the requirements for a biologically meaningful model, e.g. does not check if negative values of respiration are calculated.

Value

Tr=getTransferMatrix(Anl) #this is a function of C and t

# build the two models (linear and nonlinear) mod=GeneralModel(t, A, iv, inputrates, deSolve.lsoda.wrapper)
modnl=GeneralNlModel(t, Anl, iv, inputrates, deSolve.lsoda.wrapper)

Ynonlin=getC(modnl) lt1=2 lt2=4 plot(t,Ynonlin[,1],type="l",lty=lt1,col=1, ylab="Concentrations",xlab="Time",ylim=c(min(Ynonlin),max(Ynonlin)))
lines(t,Ynonlin[,2],type="l",lty=lt2,col=2) legend("topleft",c("Pool 1", "Pool 2"),lty=c(lt1,lt2),col=c(1,2))

See Also

GeneralModel.
Examples

t_start=0
t_end=20
tn=100
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
k1=1/2
k2=1/3
Km=0.5
nr=2

alpha=list()
alpha["1_to_2"] = function(C,t){
  1/5
}
alpha["2_to_1"] = function(C,t){
  1/6
}

f=function(C,t){
  # The only thing to take care of is that we release a vector of the same
  # size as C
  S=C[[1]]
  M=C[[2]]
  O=matrix(byrow=TRUE,nrow=2,c(k1*M*(S/(Km+S)),
k2*M))
  return(O)
}
Anl=new("TransportDecompositionOperator",t_start,Inf,nr,alpha,f)

c01=3
c02=2
iv=c(c01,c02)
inputrates=new("TimeMap",t_start,t_end,function(t){return(matrix(
nrow=nr,
ncol=1,
c( 2, 2)
))))

# we check if we can reproduce the linear decomposition operator from the
# nonlinear one

Description

automatic title
**Usage**

GeneralPoolId(id)

GeneralPoolId(id)

**Arguments**

id see method arguments

**S4-methods**

- GeneralPoolId,character-method
- GeneralPoolId,numeric-method

---

**GeneralPoolId,character-method**

*automatic title*

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'character'
GeneralPoolId(id)
```

**Arguments**

id object of class:character, no manual documentation

---

**GeneralPoolId,numeric-method**

*generic factory for this virtual class*

**Description**

the class returned depends on the method dispatched depending on the supplied arguments

**Usage**

```r
## S4 method for signature 'numeric'
GeneralPoolId(id)
```

**Arguments**

id object of class:numeric, no manual documentation
Description

The definite integral of the vector of release fluxes over time from start to t, computed for all t in the times argument the modelrun has been created with.

Usage

```r
## S4 method for signature 'Model'
getAccumulatedRelease(object)
```

Arguments

- **object**: object of class `Model`, A modelrun as produced by the constructors: `Model`, `Model_by_PoolNames`, `Model_14` the function `GeneralModel` or the functions listed in `predefinedModels`.

A model represents the initial value problem (IVP) for the contents of the pool consisting of

- The initial values of the pool content
- The system of ordinary differential equations, as dictated by the fluxes
- The times for which the solution of the IVP is evaluated.

Value

A matrix with as many columns as there are pools and as many rows as there are entries in the times argument the model has been build with.
**getC**  
*Calculates the content of the pools*

**Description**

This function computes the content of the pools as function of time. In the original (and most of the present) Models these are Carbon pools hence the name. Have a look at the methods for details.

**Usage**

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

**Arguments**

- **object**: A modelrun as produced by the constructors: `Model`, `Model_by_PoolNames`, `Model_14` the function `GeneralModel` or the functions listed in `predefinedModels`. A model represents the initial value problem (IVP) for the contents of the pool consisting of
  - The initial values of the pool content
  - The system of ordinary differential equations, as dictated by the fluxes
  - The times for which the solution of the IVP is evaluated.

- **as.closures**: see method arguments

**Value**

A matrix with m columns representing where m is the number of pools, and n rows where n is the number times as specified by the times of the model.

**S4-methods**

- `getC,Model-method`
- `getC,Model_by_PoolNames-method`
- `getC,NlModel-method`

---

**getC,Model-method**  
*Pool Contents for all times*

**Description**

Pool Contents for all times

The solution of the initial value problem (IVP) for the pool contents. Since the first models in SoilR had only Carbon pools the function name `getC` could be interpreted as referring to the C content. If the model includes other element cycles e.g. N or P this interpretation is no longer valid. In this case the C in 'getC' stands for 'content' since the function will always return the solution for all pools, regardless of the chemical element the author of the model associated them with.
Usage

## S4 method for signature 'Model'
getC(object)

Arguments

object object of class:Model, A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14 the function GeneralModel or the functions listed in predefinedModels.
A model represents the initial value problem (IVP) for the contents of the pool consisting of
• The initial values of the pool content
• The system of ordinary differential equations, as dictated by the fluxes
• The times for which the solution of the IVP is evaluated.

Value

A matrix with as many columns as there are pools and as many rows as there are entries in the times argument the model has been build with.

Description

Pool Contents for all times
The solution of the initial value problem (IVP) for the pool contents. Since the first models in SoilR had only Carbon pools the function name getC could be interpreted as referring to the C content. If the model includes other element cycles e.g. N or P this interpretation is no longer valid. In this case the C in 'getC' stands for 'content' since the function will always return the solution for all pools, regardless of the chemical element the author of the model associated them with.

Usage

## S4 method for signature 'Model_by_PoolNames'
getC(object)

Arguments

object object of class:Model_by_PoolNames, A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14 the function GeneralModel or the functions listed in predefinedModels.
A model represents the initial value problem (IVP) for the contents of the pool consisting of
• The initial values of the pool content
• The system of ordinary differential equations, as dictated by the fluxes
• The times for which the solution of the IVP is evaluated.
Value

A matrix with as many columns as there are pools and as many rows as there are entries in the times argument the model has been build with.

Description

Pool Contents for all times

The solution of the initial value problem (IVP) for the pool contents. Since the first models in SoilR had only Carbon pools the function name getC could be interpreted as referring to the C content. If the model includes other element cycles e.g. N or P this interpretation is no longer valid. In this case the C in 'getC' stands for 'content' since the function will always return the solution for all pools, regardless of the chemical element the author of the model associated them with.

Usage

```r
## S4 method for signature 'NlModel'
getC(object, as.closures = FALSE)
```

Arguments

- **object**: object of class: NlModel, no manual documentation
- **as.closures**: If TRUE will return the result as a list of approximating functions of time indexed by the pool number.

Value

If `as.closures` is FALSE (the default) the return value is a matrix with as many columns as there are pools and as many rows as there are entries in the times argument the model has been built with.

Description

Generic that yields the \(^{14}\)C content for all pools and all times

Usage

```r
getC14(object)
```

Arguments

- **object**: see method arguments

S4-methods

- `getC14.Model_14-method`
getC14,Model_14-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'Model_14'
getc14(object)
```

**Arguments**

- `object` object of class:Model_14, no manual documentation

getCompartmentalMatrixFunc

**Description**

automatic title

**Usage**

```r
getCompartmentalMatrixFunc(object, timeSymbol, state_variable_names)
```

**Arguments**

- `object` see method arguments
- `timeSymbol` see method arguments
- `state_variable_names` see method arguments

**S4-methods**

- `getCompartmentalMatrixFunc,BoundLinDecompOp,ANY,ANY-method`
- `getCompartmentalMatrixFunc,ConstLinDecompOp,ANY,ANY-method`
- `getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method`
- `getCompartmentalMatrixFunc,UnBoundNonLinDecompOp_by_PoolNames,character,character-method`
- `getCompartmentalMatrixFunc,UnBoundNonLinDecompOp,ANY,ANY-method`
getCompartmentalMatrixFunc,BoundLinDecompOp,ANY,ANY-method

Description
automatic title

Usage
## S4 method for signature 'BoundLinDecompOp,ANY,ANY'
getCompartmentalMatrixFunc(object)

Arguments
object object of class: BoundLinDecompOp, no manual documentation

getCompartmentalMatrixFunc,ConstLinDecompOp,ANY,ANY-method

Description
automatic title

Usage
## S4 method for signature 'ConstLinDecompOp,ANY,ANY'
getCompartmentalMatrixFunc(object)

Arguments
object object of class: ConstLinDecompOp, no manual documentation

getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method

Description
automatic title

Usage
## S4 method for signature 'TransportDecompositionOperator,ANY,ANY'
getCompartmentalMatrixFunc(object)

Arguments
object object of class: TransportDecompositionOperator, no manual documentation
Description

Extract the matrix valued function of time and state vector for the compartmental matrix.

Usage

```
## S4 method for signature 'UnBoundNonLinDecompOp,ANY,ANY'
getCompartmentalMatrixFunc(object)
```

Arguments

- `object` object of class: `UnBoundNonLinDecompOp`, no manual documentation

---

Description

Compartmental Matrix as function of the state vector and time.

Usage

```
## S4 method for signature
## 'UnBoundNonLinDecompOp_by_PoolNames,character,character'
getCompartmentalMatrixFunc(object, timeSymbol, state_variable_names)
```

Arguments

- `object` object of class: `UnBoundNonLinDecompOp_by_PoolNames`, An object of the class `UnBoundNonLinDecompOp_by_PoolNames` which is a representation equivalent to the compartmental matrix but independent of the order of state variables (pools) which therefore can be translated to any such ordering.
- `timeSymbol` object of class: `character`, The name of the argument representing time in the functions defining the fluxes in `object`
getConstantCompartmentalMatrix

state_variable_names
object of class: character. The vector of the names of the state variables. The argument object is a representation of the compartmental system as lists of fluxes (internal fluxes and out-fluxes) as functions of the state variables and time. This method translates it to a matrix based formulation specific to a given ordering of the state variables. It is assumed (and checked) that the names formal arguments of the flux functions in object are a subset of the names of state_variable_names. The method is used internally to translate the more intuitive (and more general) flux based description to the matrix based description required by the ode solvers.

getConstantCompartmentalMatrix

Description
automatic title

Usage
getConstantCompartmentalMatrix(object)

Arguments
object see method arguments

S4-methods

• getConstantCompartmentalMatrix, ConstLinDecompOp-method
• getConstantCompartmentalMatrix, ConstLinDecompOpWithLinearScalarFactor-method

generateComponentSet

Description
automatic title

Usage
## S4 method for signature 'ConstLinDecompOp'
generateComponentSet(object)

Arguments
object object of class: ConstLinDecompOp, no manual documentation
getConstantCompartmentalMatrix, ConstLinDecompOpWithLinearScalarFactor-method

Description

automatic title

Usage

## S4 method for signature 'ConstLinDecompOpWithLinearScalarFactor'
getConstantCompartmentalMatrix(object)

Arguments

object object of class: ConstLinDecompOpWithLinearScalarFactor, no manual documentation

getConstantInFluxVector

Description

automatic title

Usage

getConstantInFluxVector(object)

Arguments

object see method arguments

S4-methods

- getConstantInFluxVector, ConstInFluxes-method
getConstantInFluxVector, ConstInFluxes-method

Description

automatic title

Usage

## S4 method for signature 'ConstInFluxes'
getConstantInFluxVector(object)

Arguments

object object of class: ConstInFluxes, no manual documentation

getConstantInternalFluxRateList_by_PoolIndex

Description

automatic title

Usage

getConstantInternalFluxRateList_by_PoolIndex(object)

Arguments

object see method arguments

S4-methods

• getConstantInternalFluxRateList_by_PoolIndex, ConstLinDecompOp-method
getConstantOutFluxRateList_by_PoolIndex

Description

automatic title

Usage

## S4 method for signature 'ConstLinDecompOp'
getConstantOutFluxRateList_by_PoolIndex(object)

Arguments

object object of class: ConstLinDecompOp, no manual documentation

getConstantInternalFluxRateList_by_PoolIndex

Description

automatic title

Usage

getConstantInternalFluxRateList_by_PoolIndex(object)

Arguments

object see method arguments

S4-methods

- getConstantOutFluxRateList_by_PoolIndex, ConstLinDecompOp-method
getConstantOutFluxRateList_by_PoolIndex, ConstLinDecompOp-method

Description

automatic title

Usage

## S4 method for signature 'ConstLinDecompOp'
getConstantOutFluxRateList_by_PoolIndex(object)

Arguments

object object of class: ConstLinDecompOp, no manual documentation

getConstLinDecompOp automatic title

Description

automatic title

Usage

getConstLinDecompOp(object)

Arguments

object see method arguments

S4-methods

• getConstLinDecompOp, ConstLinDecompOpWithLinearScalarFactor-method
Description

automatic title

Usage

## S4 method for signature 'ConstLinDecompOpWithLinearScalarFactor'
getConstLinDecompOp(object)

Arguments

object object of class:ConstLinDecompOpWithLinearScalarFactor, no manual documentation

Description

automatic title

Usage

getcumulativeC(object)

Arguments

object see method arguments

S4-methods

- getCumulativeC,NlModel-method
**getCumulativeC,NlModel-method**

*automatic title*

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'NlModel'
getcumulativeC(object)
```

**Arguments**

- `object` object of class:NlModel, no manual documentation

---

**getDecompOp**

*automatic title*

**Description**

automatic title

**Usage**

`getDecompOp(object)`

**Arguments**

- `object` see method arguments

**S4-methods**

- `getDecompOp,Model-method`
- `getDecompOp,NlModel-method`
getDecompOp,Model-method

Extract the Compartmental Operator

Description

The method is usually used internally by other methods operating on models. The information it
yields has either been provided by the user in creating the modelrun or can be obtained by directly
transforming the arguments that were used.

Usage

## S4 method for signature 'Model'
getDecompOp(object)

Arguments

object  object of class:Model, A modelrun as produced by the constructors: Model,
Model_by_PoolNames, Model_14 the function GeneralModel or the functions
listed in predefinedModels.
A model represents the initial value problem (IVP) for the contents of the pool
consisting of
  • The initial values of the pool content
  • The system of ordinary differential equations, as dictated by the fluxes
  • The times for which the solution of the IVP is evaluated.

Value

The actual class of the result can vary. It will be a subclass of DecompOp. These objects are an
abstraction for a complete description of the fluxes in the pool system regardless of the form it
is provided in. The information contained in these objects is equivalent to the set of internal and
outward fluxes as functions of pool contents and time and sufficient to infer the "Compartmental
Matrix" as a matrix valued function of the same arguments. In the general case of a nonautonomous
nonlinear Model this function is a true function of both, the pool contents and time. In the case of
an non-autonomous linear model it is a function of time only, and in case of a autonomous linear
model it is a constant matrix. The vector valued function can be inferred by the generic function
getFunctionDefinition.

getDecompOp,NlModel-method

Extract the Compartmental Operator

Description

Extract the Compartmental Operator

Usage

## S4 method for signature 'NlModel'
getDecompOp(object)
getDotOut

Arguments

object object of class: NlModel, A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14, the function GeneralModel or the functions listed in predefinedModels.

A model represents the initial value problem (IVP) for the contents of the pool consisting of

• The initial values of the pool content
• The system of ordinary differential equations, as dictated by the fluxes
• The times for which the solution of the IVP is evaluated.

Value

The actual class of the result can vary. It will be a subclass of DecompOp. These objects are an abstraction for a complete description of the fluxes in the pool system regardless of the form it is provided in. The information contained in these objects is equivalent to the set of internal and outward fluxes as functions of pool contents and time and sufficient to infer the “Compartmental Matrix” as a matrix valued function of the same arguments. In the general case of a nonautonomous nonlinear Model this function is a true function of both, the pool contents and time. In the case of an non-autonomous linear model it is a function of time only, and in case of a autonomous linear model it is a constant matrix. The vector valued function can be inferred by the generic function getFunctionDefinition.

getDotOut automatic title

Description

automatic title

Usage

getDotOut(object)

Arguments

object see method arguments

S4-methods

• getDotOut,TransportDecompositionOperator-method
getDotOut,TransportDecompositionOperator-method

## S4 method for signature 'TransportDecompositionOperator'
getDotOut(object)

**Arguments**

- **object**: object of class `TransportDecompositionOperator`, no manual documentation

getF14

Generic that yields the $^{14}$C fraction for the content all pools and all times

**Description**

Generic that yields the $^{14}$C fraction for the content all pools and all times

**Usage**

getF14(object)

**Arguments**

- **object**: see method arguments

**S4-methods**

- `getF14,Model_14-method`
getF14, Model_14-method

---

**getF14, Model_14-method**

*automatic title*

---

### Description

automatic title

### Usage

```r
# S4 method for signature 'Model_14'
getF14(object)
```

#### Arguments

- `object` object of class `Model_14`, no manual documentation

---

### getF14C

*Generic that yields the \(^{14}C\) fraction for the cumulative content of all pools and all times*

---

#### Description

Generic that yields the \(^{14}C\) fraction for the cumulative content of all pools and all times

#### Usage

```r
getF14C(object)
```

#### Arguments

- `object` see method arguments

---

### S4-methods

- `getF14C, Model_14-method`
getF14C, Model_14-method

**Description**

Automatic title

**Usage**

```r
## S4 method for signature 'Model_14'
getF14C(object)
```

**Arguments**

- `object` object of class: Model_14, no manual documentation

---

getF14R

*Generic that yields the ^14C fraction for the release flux of all pools and all times*

**Description**

Generic that yields the ^14C fraction for the release flux of all pools and all times

**Usage**

```r
getF14R(object)
```

**Arguments**

- `object` see method arguments

**S4-methods**

- `getF14R, Model_14-method`
getF14R,Model_14-method

Description
automatic title

Usage
## S4 method for signature 'Model_14'
getF14R(object)

Arguments
object object of class:Model_14, no manual documentation

getFormat automatic title

Description
automatic title

Usage
getFormat(object)

Arguments
object see method arguments

S4-methods
• getFormat,Fc-method

getFormat,Fc-method automatic title

Description
automatic title

Usage
## S4 method for signature 'Fc'
getFormat(object)

Arguments
object object of class:Fc, no manual documentation
getFunctionDefinition,ConstInFluxes-method

Description

automatic title

Usage

getFunctionDefinition(object)

Arguments

object object of class: ConstInFluxes, no manual documentation

S4-methods

- getFunctionDefinition,ConstInFluxes-method
- getFunctionDefinition,ConstLinDecompOp-method
- getFunctionDefinition,ConstLinDecompOpWithLinearScalarFactor-method
- getFunctionDefinition,DecompositionOperator-method
- getFunctionDefinition,InFluxList_by_PoolIndex-method
- getFunctionDefinition,InFluxList_by_PoolName-method
- getFunctionDefinition,StateDependentInFluxVector-method
- getFunctionDefinition,TimeMap-method
- getFunctionDefinition,TransportDecompositionOperator-method
- getFunctionDefinition,UnBoundInFluxes-method
- getFunctionDefinition,UnBoundLinDecompOp-method

getFunctionDefinition,ConstInFluxes-method

Description

automatic title

Usage

## S4 method for signature 'ConstInFluxes'
getFunctionDefinition(object)

Arguments

object object of class: ConstInFluxes, no manual documentation
getFunctionDefinition,ConstLinDecompOp-method

Description
automatic title

Usage

```r
## S4 method for signature 'ConstLinDecompOp'
getFunctionDefinition(object)
```

Arguments

object object of class: ConstLinDecompOp, no manual documentation

getFunctionDefinition,ConstLinDecompOpWithLinearScalarFactor-method

helper function

Description
helper function

Usage

```r
## S4 method for signature 'ConstLinDecompOpWithLinearScalarFactor'
getFunctionDefinition(object)
```

Arguments

object object of class: ConstLinDecompOpWithLinearScalarFactor, no manual documentation

getFunctionDefinition,DecompositionOperator-method

automatic title

Description
automatic title

Usage

```r
## S4 method for signature 'DecompositionOperator'
getFunctionDefinition(object)
```

Arguments

object object of class: DecompositionOperator, no manual documentation
getFunctionDefinition,InFluxList_by_PoolIndex-method

Description
automatic title

Usage

## S4 method for signature 'InFluxList_by_PoolIndex'
getFunctionDefinition(object, numberOfPools)

Arguments

object object of class:InFluxList_by_PoolIndex, no manual documentation
numberOfPools no manual documentation

getFunctionDefinition,InFluxList_by_PoolName-method

Description
automatic title
automatic title

Usage

## S4 method for signature 'InFluxList_by_PoolName'
getFunctionDefinition(object, timeSymbol, poolNames)

Arguments

object object of class:InFluxList_by_PoolName, no manual documentation
timeSymbol no manual documentation
poolNames no manual documentation
getFunctionDefinition, StateDependentInFluxVector-method

Description
automatic title

Usage

## S4 method for signature 'StateDependentInFluxVector'
getFunctionDefinition(object)

Arguments

object
object of class: StateDependentInFluxVector, no manual documentation

getFunctionDefinition, TimeMap-method

Description
automatic title

Usage

## S4 method for signature 'TimeMap'
getFunctionDefinition(object)

Arguments

object
object of class: TimeMap, no manual documentation

getFunctionDefinition, TransportDecompositionOperator-method

Description
automatic title

Usage

## S4 method for signature 'TransportDecompositionOperator'
getFunctionDefinition(object)

Arguments

object
object of class: TransportDecompositionOperator, no manual documentation
### Description

Extracts the time dependent matrix valued function (compartmental matrix)

### Usage

```r
## S4 method for signature 'UnBoundLinDecompOp'
getFunctionDefinition(object)
```

### Arguments

- `object`: object of class `UnBoundLinDecompOp`, no manual documentation

### See Also

Other `UnBoundLinDecompOp_constructor`: `UnBoundLinDecompOp, function-method`
**getInFluxes**  
*Extract the influxes*

**Description**  
Extract the influxes

**Usage**  
`getInFluxes(object)`

**Arguments**  

- `object`  
  see method arguments

**S4-methods**

- `getInFluxes,Model-method`
- `getInFluxes,NlModel-method`

**getInFluxes,Model-method**

*Extract the InFluxes as provided during creation of the model*

**Description**  
Since the influxes had to be provided to create the model this method yields no new information that can not be obtained simpler. It is usually called internally by other functions.

**Usage**  
```
## S4 method for signature 'Model'
getInFluxes(object)
```

**Arguments**  

- `object`  
  object of class `Model`, A modelrun as produced by the constructors: `Model, Model_by_PoolNames, Model_14` the function `GeneralModel` or the functions listed in `predefinedModels`.  
  A model represents the initial value problem (IVP) for the contents of the pool consisting of
  - The initial values of the pool content
  - The system of ordinary differential equations, as dictated by the fluxes
  - The times for which the solution of the IVP is evaluated.
Description
automatic title

Usage
## S4 method for signature 'NlModel'
getInitialValues(object)

Arguments
object object of class:NlModel, no manual documentation

S4-methods
• getInitialValues,NlModel-method

Description
automatic title

Usage
getInitialValues(object)

Arguments
object see method arguments

S4-methods
• getInitialValues,NlModel-method

Description
automatic title

Usage
## S4 method for signature 'NlModel'
getInitialValues(object)

Arguments
object object of class:NlModel, no manual documentation
getLinearScaleFactor

Description

automatic title

Usage

getLinearScaleFactor(object)

Arguments

object see method arguments

S4-methods

• getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method

getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method

Description

automatic title

Usage

## S4 method for signature 'ConstLinDecompOpWithLinearScalarFactor'
getLinearScaleFactor(object)

Arguments

object object of class:ConstLinDecompOpWithLinearScalarFactor, no manual documentation
getMeanTransitTime

Description

automatic title

Usage

g MeinTransitTime(object, inputDistribution)

Arguments

object see method arguments
inputDistribution see method arguments

S4-methods

• getMeanTransitTime,ConstLinDecompOp-method
getNumberOfPools

Description

automatic title

Usage

getNumberOfPools(object)

Arguments

object see method arguments

S4-methods

- getNumberOfPools,MCSim-method
- getNumberOfPools,NlModel-method
- getNumberOfPools,TransportDecompositionOperator-method

getNumberOfPools,MCSim-method

Description

automatic title

Usage

## S4 method for signature 'MCSim'
getNumberOfPools(object)

Arguments

object object of class:MCSim, no manual documentation
getNumberOfPools,NlModel-method

Description

automatic title

Usage

## S4 method for signature 'NlModel'
getNumberOfPools(object)

Arguments

object object of class:NlModel, no manual documentation

getNumberOfPools,TransportDecompositionOperator-method

Description

automatic title

Usage

## S4 method for signature 'TransportDecompositionOperator'
getNumberOfPools(object)

Arguments

object object of class:TransportDecompositionOperator, no manual documentation

getOutputFluxes

Generic Function to obtain the fluxes out of of the pools

Description

Generic Function to obtain the fluxes out of of the pools

Usage

getOutputFluxes(object, as.closures = F)

Arguments

object see method arguments
as.closures see method arguments
getOutputFluxes,NlModel-method

Description

automatic title

Usage

## S4 method for signature 'NlModel'
getOutputFluxes(object, as.closures = F)

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>object of class:NlModel, no manual documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.closures</td>
<td>no manual documentation</td>
</tr>
</tbody>
</table>

getOutputReceivers

Description

automatic title

Usage

getOutputReceivers(object, i)

Arguments

<table>
<thead>
<tr>
<th>object</th>
<th>see method arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>see method arguments</td>
</tr>
</tbody>
</table>

S4-methods

- getOutputReceivers,TransportDecompositionOperator,numeric-method
getOutputReceivers, TransportDecompositionOperator, numeric-method

getParticleMonteCarloSimulator

Description

automatic title

Usage

## S4 method for signature 'TransportDecompositionOperator, numeric'
getOutputReceivers(object, i)

Arguments

object object of class: TransportDecompositionOperator, no manual documentation
i object of class: numeric, no manual documentation

getParticleMonteCarloSimulator

Description

automatic title

Usage

getParticleMonteCarloSimulator(object)

Arguments

object see method arguments

S4-methods

- getParticleMonteCarloSimulator, NLModel-method
getParticleMonteCarloSimulator,NlModel-method

Description

automatic title

Usage

## S4 method for signature 'NlModel'
getParticleMonteCarloSimulator(object)

Arguments

object object of class:NlModel, no manual documentation

getReleaseFlux

Generic Function to obtain the vector of release fluxes out of the pools for all times.

Description

Generic Function to obtain the vector of release fluxes out of the pools for all times.

Usage

getReleaseFlux(object)

Arguments

object see method arguments

S4-methods

• getReleaseFlux,Model-method
• getReleaseFlux,Model_by_PoolNames-method
• getReleaseFlux,NlModel-method
getReleaseFlux, Model-method

*The release fluxes [content][time] for all pools.*

Description

The release fluxes [content][time] for all pools.

Usage

```r
## S4 method for signature 'Model'
getReleaseFlux(object)
```

Arguments

object

object of class: Model. A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14 the function GeneralModel or the functions listed in predefinedModels. A model represents the initial value problem (IVP) for the contents of the pool consisting of

• The initial values of the pool content
• The system of ordinary differential equations, as dictated by the fluxes
• The times for which the solution of the IVP is evaluated.

Value

A matrix with as many columns as there are pools and as many rows as there are entries in the times argument the model has been build with.

getReleaseFlux, Model_by_PoolNames-method

*automatic title*

Description

automatic title

Usage

```r
## S4 method for signature 'Model_by_PoolNames'
getReleaseFlux(object)
```
Arguments

object object of class: Model_by_PoolNames. A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14 the function GeneralModel or the functions listed in predefinedModels. A model represents the initial value problem (IVP) for the contents of the pool consisting of

- The initial values of the pool content
- The system of ordinary differential equations, as dictated by the fluxes
- The times for which the solution of the IVP is evaluated.

Description

automatic title

Usage

## S4 method for signature 'NlModel'
getReleaseFlux(object)

Arguments

object object of class: NlModel, no manual documentation

Description

automatic title

Usage

getReleaseFlux14(object)

Arguments

object see method arguments

S4-methods

- getReleaseFlux14, Model_14-method
**getReleaseFlux14,Model_14-method**

*automatic title*

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'Model_14'
getReleaseFlux14(object)
```

**Arguments**

- `object` object of class: Model_14. no manual documentation

---

**getRightHandSideOfODE automatic title**

**Description**

automatic title

**Usage**

```r
g.getRightHandSideOfODE(object, timeSymbol, poolNames, numberOfPools)
```

**Arguments**

- `object` see method arguments
- `timeSymbol` see method arguments
- `poolNames` see method arguments
- `numberOfPools` see method arguments

**S4-methods**

- `getRightHandSideOfODE,Model-method`
- `getRightHandSideOfODE,Model_by_PoolNames-method`
Description

For non-linear models or models with state dependent influxes the returned function is a true function of state and time. For linear models with state independent influxes the returned function is in fact a function of time only.

Usage

```r
## S4 method for signature 'Model'
getRightHandSideOfODE(object)
```

Arguments

- `object` object of class: `Model`, no manual documentation

Value

A function $f(t)$

Description

This function is required by the ODE solvers.

Usage

```r
## S4 method for signature 'Model_by_PoolNames'
getRightHandSideOfODE(object)
```

Arguments

- `object` object of class: `Model_by_PoolNames`, The model
getSolution

Calculates all stocks all fluxes to, in and out of the compartment system and also their integrals over time

Description
Have a look at the methods for details.

Usage
getSolution(object, params, as.closures = F)

Arguments

- **object**
  A modelrun as produced by the constructors: `Model`, `Model_by_PoolNames`, `Model_14` the function `GeneralModel` or the functions listed in `predefinedModels`. A model represents the initial value problem (IVP) for the contents of the pool consisting of
  - The initial values of the pool content
  - The system of ordinary differential equations, as dictated by the fluxes
  - The times for which the solution of the IVP is evaluated.

- **params**
  see method arguments

- **as.closures**
  see method arguments

Value
A matrix with columns representing the name of the statevariable, flux and accumulated flux for every time as specified by the times of the model.

S4-methods

- `getSolution,Model_by_PoolNames-method`

getSolution,Model_by_PoolNames-method

All Fluxes and stocks for all times

Description
All Fluxes and stocks for all times

Usage
```r
## S4 method for signature 'Model_by_PoolNames'
getSolution(object, params)
```
**getTimeRange**

**Arguments**

object object of class: `Model_by_PoolNames`, A modelrun as produced by the constructors: `Model`, `Model_by_PoolNames`, `Model_14` the function `GeneralModel` or the functions listed in `predefinedModels`. A model represents the initial value problem (IVP) for the contents of the pool consisting of

- The initial values of the pool content
- The system of ordinary differential equations, as dictated by the fluxes
- The times for which the solution of the IVP is evaluated.

params no manual documentation

**Value**

A matrix with as many columns as there are pools and as many rows as there are entries in the `times` argument the model has been build with.

**Description**

automatic title

**Usage**

g getTimeRange(object)

**Arguments**

object see method arguments

**S4-methods**

- `getTimeRange,ConstInFluxes-method`
- `getTimeRange,ConstLinDecompOp-method`
- `getTimeRange,ConstLinDecompOpWithLinearScalarFactor-method`
- `getTimeRange,DecompositionOperator-method`
- `getTimeRange,TimeMap-method`
- `getTimeRange,UnBoundInFluxes-method`
- `getTimeRange,UnBoundLinDecompOp-method`
getTimeRange,ConstInFluxes-method

Description
automatic title

Usage
## S4 method for signature 'ConstInFluxes'
getTimeRange(object)

Arguments
object object of class:ConstInFluxes, no manual documentation

getTimeRange,ConstLinDecompOp-method

Description
automatic title

Usage
## S4 method for signature 'ConstLinDecompOp'
getTimeRange(object)

Arguments
object object of class:ConstLinDecompOp, no manual documentation

getTimeRange,ConstLinDecompOpWithLinearScalarFactor-method

Description
automatic title

Usage
## S4 method for signature 'ConstLinDecompOpWithLinearScalarFactor'
getTimeRange(object)

Arguments
object object of class:ConstLinDecompOpWithLinearScalarFactor, no manual documentation
Description
automatic title

Usage

## S4 method for signature 'DecompositionOperator'
getTimeRange(object)

Arguments
object object of class: DecompositionOperator, no manual documentation

Description
The time interval where the function is defined

Usage

## S4 method for signature 'TimeMap'
getTimeRange(object)

Arguments
object object of class: TimeMap, no manual documentation

Description
automatic title

Usage

## S4 method for signature 'UnBoundInFluxes'
getTimeRange(object)

Arguments
object object of class: UnBoundInFluxes, no manual documentation
### Description

Extracts the time interval for which the function is valid.

### Usage

```r
## S4 method for signature 'UnBoundLinDecompOp'
g getTimeRange(object)
```

### Arguments

- `object` object of class: `UnBoundLinDecompOp`, no manual documentation

### Description

automatic title

### Usage

```r
getTimes(object)
```

### Arguments

- `object` see method arguments

### S4-methods

- `getTimes,Model-method`
- `getTimes,Model_by_PoolNames-method`
- `getTimes,NlModel-method`
**getTimes,Model-method**

*Extract the times vector*

**Description**

Since the times had to be provided to create the model this method yields no new information. It is usually called internally by other functions that deal with models.

**Usage**

```r
## S4 method for signature 'Model'
getTimes(object)
```

**Arguments**

- `object` object of class:Model, A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14 the function GeneralModel or the functions listed in predefinedModels.

A model represents the initial value problem (IVP) for the contents of the pool consisting of:
- The initial values of the pool content
- The system of ordinary differential equations, as dictated by the fluxes
- The times for which the solution of the IVP is evaluated.

---

**getTimes,Model_by_PoolNames-method**

*Extract the times vector*

**Description**

Since the times had to be provided to create the model this method yields no new information. It is usually called internally by other functions that deal with models.

**Usage**

```r
## S4 method for signature 'Model_by_PoolNames'
getTimes(object)
```

**Arguments**

- `object` object of class:Model_by_PoolNames, A modelrun as produced by the constructors: Model, Model_by_PoolNames, Model_14 the function GeneralModel or the functions listed in predefinedModels.

A model represents the initial value problem (IVP) for the contents of the pool consisting of:
- The initial values of the pool content
- The system of ordinary differential equations, as dictated by the fluxes
- The times for which the solution of the IVP is evaluated.
getTimes,NlModel-method

Description
automatic title

Usage
## S4 method for signature 'NlModel'
getTimes(object)

Arguments
object object of class:NlModel, no manual documentation

getTransferCoefficients

Description
automatic title
automatic title

Usage
getTransferCoefficients(object, as.closures = F)

Arguments
object see method arguments
as.closures see method arguments

S4-methods
- getTransferCoefficients,NlModel-method
- getTransferCoefficients,TransportDecompositionOperator-method
getTransferCoefficients,NlModel-method

Description
automatic title

Usage
## S4 method for signature 'NlModel'
getTransferCoefficients(object, as.closures = F)

Arguments
object object of class:NlModel, no manual documentation
as.closures no manual documentation

getTransferCoefficients,TransportDecompositionOperator-method

Description
automatic title

Usage
## S4 method for signature 'TransportDecompositionOperator'
getTransferCoefficients(object)

Arguments
object object of class:TransportDecompositionOperator, no manual documentation

getTransferMatrix deprecated, use getTransferMatrixFunc instead

Description
deprecated, use getTransferMatrixFunc instead

Usage
getTransferMatrix(object)

Arguments
object A compartmental operator
getTransferMatrixFunc,TransportDecompositionOperator-method

---

getTransferMatrixFunc automatic title

---

**Description**

automatic title

**Usage**

getTransferMatrixFunc(object)

**Arguments**

object see method arguments

**S4-methods**

- getTransferMatrixFunc,TransportDecompositionOperator-method

---

getTransferMatrixFunc,TransportDecompositionOperator-method

---

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'TransportDecompositionOperator'
getTransferMatrixFunc(object)
```

**Arguments**

object object of class:TransportDecompositionOperator, no manual documentation
**getTransitTimeDistributionDensity**

*automatic title*

### Description

automatic title

### Usage

getTransitTimeDistributionDensity(object, inputDistribution, times)

### Arguments

- **object**
  - see method arguments
- **inputDistribution**
  - see method arguments
- **times**
  - see method arguments

### S4-methods

- `getTransitTimeDistributionDensity,ConstLinDecompOp-method`

---

**getTransitTimeDistributionDensity,ConstLinDecompOp-method**

*automatic title*

### Description

automatic title

### Usage

```r
## S4 method for signature 'ConstLinDecompOp'
getTransitTimeDistributionDensity(object, inputDistribution, times)
```

### Arguments

- **object**
  - object of class:ConstLinDecompOp, no manual documentation
- **inputDistribution**
  - no manual documentation
- **times**
  - no manual documentation
getValues method

### Description

automatic title

### Usage

```r
getValues(object)
```

### Arguments

- **object**: see method arguments

### S4-methods

- `getValues,ConstFc-method`

---

getValues,ConstFc-method

### Description

automatic title

### Usage

```r
## S4 method for signature 'ConstFc'
getValues(object)
```

### Arguments

- **object**: object of class: ConstFc, no manual documentation
Compiled records of radicarbon in atmospheric CO2 for historical simulations in CMIP6

Description

Historical Delta-14C in atmospheric CO2 used as forcing dataset for CMIP6 simulation experiments. Data is reported for three hemispheric zones, for the period 1850-2015.

Usage

data(Graven2017)

Format

A data frame with 166 rows and 4 variables.

- Year.AD Year (AD).
- NH Delta14C for the northern hemisphere, between 30N to 90N latitude. Values in per mil.
- Tropics Delta14C for the tropics, between 30N to 30S latitude. Values in per mil.
- SH Delta14C for the southern hemisphere, between 30S to 90S latitude. Values in per mil.

Details

All details about the derivation of this dataset are provided in Graven et al. (2017)

Author(s)

Carlos Sierra <csierra@bgc-jena.mpg.de>

Source

<https://doi.org/10.22033/ESGF/input4MIPs.1602>

References

Graven, Heather; Allison, Colin; Etheridge, David; Hammer, Samuel; Keeling, Ralph; Levin, Ingeborg; Meijer, Harro A. J.; Rubino, Mauro; Tans, Pieter; Trudinger, Cathy; Vaughn, Bruce; White, James (2017). Compiled Historical Record of Atmospheric Delta14CO2 version 2.0. Earth System Grid Federation. https://doi.org/10.22033/ESGF/input4MIPs.1602


Examples

```r
matplot(Graven2017[,1], Graven2017[,-1],type="l",
        lty=1, xlab="Year AD", ylab="Delta14C (per mil)", bty="n")
legend("topleft",names(Graven2017[,-1]), lty=1, col=1:3, bty="n")
```
### HarvardForest14CO2

*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**

HarvardForest14CO2

**Format**

A data frame with the following 3 variables.

1. Year A numeric vector with the date of measurement in years
2. D14C A numeric vector with the value of the Delta 14C value measured in CO2 efflux in per mil
3. Site A factor indicating the site where measurements were made. NWN: Northwest Near, Drydown: 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**


**Examples**

plot(HarvardForest14CO2[,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
doi: 10.2458/azu_js_rc.v55i2.16177

References

Examples
plot(Hua2013$NHZone1$Year.AD, Hua2013$NHZone1$mean.Delta14C, type="l", xlab="Year AD", ylab=expression(paste(Delta^{14},"C (\text{\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")
ICBMModel

Implementation of the Introductory Carbon Balance Model (ICBM)

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 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

References


See Also

There are other predefinedModels and also more general functions like Model.

Examples

# examples from external files
# inst/examples/exICBMModel.R exICBMModel_paper:

  # 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)

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),
by="n"
)
incubation_experiment

Description
This implementation follows the description in Katterer and Andren (2001, Eco Mod 136:191).

Usage
ICBM_N(
  i = 0.47,
  k_Y = 0.259,
  k_O = 0.0154,
  r_e = 1,
  e_Y = 0.362,
  h = 0.243,
  q_i = 18.8,
  q_b = 5
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>carbon input to the soil from plant production</td>
</tr>
<tr>
<td>k_Y</td>
<td>decomposition rate of young pool Y</td>
</tr>
<tr>
<td>k_O</td>
<td>decomposition rate of old pool O</td>
</tr>
<tr>
<td>r_e</td>
<td>external effects on decomposition rates</td>
</tr>
<tr>
<td>e_Y</td>
<td>yield efficiency of the soil organism community</td>
</tr>
<tr>
<td>h</td>
<td>humification coefficient. Fraction of outflux from Y that is not respired and enters O</td>
</tr>
<tr>
<td>q_i</td>
<td>C:N ratio of plant inputs</td>
</tr>
<tr>
<td>q_b</td>
<td>C:N ratio of soil organism biomass</td>
</tr>
</tbody>
</table>

incubation_experiment  Soil CO2 efflux from an incubation experiment, along with the soil mass and carbon concentration measurements.

Description
A dataset with soil CO2 efflux measurements from a laboratory incubation at controlled temperature and moisture conditions.

Usage
data(incubation_experiment)

Format
A list with 3 variables.
eCO2  A data.frame with the flux data.
c_concentrations  a vector with 3 measurement of the concentration of carbon in the soil.
soil_mass  the mass of the soil column in g
Details

The data.frame incubation_experiment$eCO2 has 3 columns.

Days  A numeric vector with the day of measurement after the experiment started.
eCO2mean  A numeric vector with the release flux of CO2. Units in ug C g-1 soil day-1.
eCO2sd  A numeric vector with the standard deviation of the release flux of CO2-C. Units in ug C g-1 soil day-1.

A laboratory incubation experiment was performed in March 2014 for a period of 35 days under controlled conditions of temperature (15 degrees Celsius), moisture (30 percent soil water content), and oxygen levels (20 percent). Soil CO2 measurements were taken using an automated system for gas sampling connected to an infrared gas analyzer. The soil was sampled at a boreal forest site (Caribou Poker Research Watershed, Alaska, USA). This dataset presents the mean and standard deviation of 4 replicates.

Examples

eCO2=incubation_experiment$eCO2
head(eCO2)

plot(eCO2[,1:2],type="o",ylim=c(0,50),ylab="CO2 efflux (ug C g-1 soil day-1")
arrows(eCO2[,1],eCO2[,2]-eCO2[,3],eCO2[,1],eCO2[,2]+eCO2[,3], angle=90,length=0.3,code=3)

InFlux

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

InFlux(map, ...)

Arguments

map  see method arguments
...
see method arguments
InFluxes

A generic factory for subclasses of InFluxes

Description
A generic factory for subclasses of InFluxes

Usage
InFluxes(object, numberOfPools)

Arguments
- object: see method arguments
- numberOfPools: see method arguments

S4-methods
- InFluxes,ConstantInFluxList_by_PoolIndex-method
- InFluxes,function-method
- InFluxes,InFluxes-method
- InFluxes,InFluxes-method
- InFluxes,list-method
- InFluxes,numeric-method
- InFluxes,StateIndependentInFluxList_by_PoolIndex-method
- InFluxes,TimeMap-method

InFluxes,ConstantInFluxList_by_PoolIndex-method

automatic title

Description
automatic title

Usage
## S4 method for signature 'ConstantInFluxList_by_PoolIndex'
InFluxes(object, numberOfPools)

Arguments
- object: object of class:ConstantInFluxList_by_PoolIndex, no manual documentation
- numberOfPools: no manual documentation
InFluxes, function-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'function'
InFluxes(object)
```

**Arguments**

- `object`: object of class: `function`, no manual documentation

InFluxes, InFluxes-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'InFluxes'
InFluxes(object)
```

**Arguments**

- `object`: object of class: `InFluxes`, no manual documentation

InFluxes, list-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'list'
InFluxes(object)
```

**Arguments**

- `object`: object of class: `list`, no manual documentation
InFluxes,numeric-method

Description

automatic title

Usage

## S4 method for signature 'numeric'
InFluxes(object)

Arguments

object object of class: numeric, no manual documentation

InFluxes,StateIndependentInFluxList_by_PoolIndex-method

Description

automatic title

Usage

## S4 method for signature 'StateIndependentInFluxList_by_PoolIndex'
InFluxes(object, numberOfPools)

Arguments

object object of class: StateIndependentInFluxList_by_PoolIndex, no manual documentation

numberOfPools no manual documentation basically produces a vector valued function from a list of scalar functions
InFluxes, TimeMap-method

automatic title

Description

automatic title

Usage

```r
## S4 method for signature 'TimeMap'
InFluxes(object)
```

Arguments

- `object` object of class: TimeMap, no manual documentation

InFluxes-class

A virtual S4-class representing (different subclasses) of in-fluxes to the system

Description

A virtual S4-class representing (different subclasses) of in-fluxes to the system

S4-methods

S4-methods with class InFluxes in their signature::

- InFluxes, InFluxes-method

S4-subclasses

- StateDependentInFluxVector
- ConstInFluxes
- BoundInFluxes
- UnBoundInFluxes
Description

Generic constructor for the class with the same name

Usage

InFluxList_by_PoolIndex(object)

Arguments

object see method arguments

S4-methods

- InFluxList_by_PoolIndex,list-method

InFluxList_by_PoolIndex,list-method

constructor from a normal list

Description

after checking the elements

Usage

## S4 method for signature 'list'
InFluxList_by_PoolIndex(object)

Arguments

object object of class:list, no manual documentation

S4-methods

S4-methods with class InFluxList_by_PoolIndex in their signature::

- getFunctionDefinition,InFluxList_by_PoolIndex-method
InFluxList_by_PoolName

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

InFluxList_by_PoolName(object)

Arguments

object see method arguments

S4-methods

• InFluxList_by_PoolName,list-method

InFluxList_by_PoolName,list-method

constructor from a normal list

Description

after checking the elements

Usage

## S4 method for signature 'list'
InFluxList_by_PoolName(object)

Arguments

object object of class: list, no manual documentation
InFluxList_by_PoolName-class

Class for a list of influxes indexed by the names of the target pools

Description

Class for a list of influxes indexed by the names of the target pools

S4-methods

S4-methods with class InFluxList_by_PoolName in their signature:

- as.numeric,InFluxList_by_PoolName-method
- by_PoolIndex,InFluxList_by_PoolName,character,character-method
- getFunctionDefinition,InFluxList_by_PoolName-method
- Model_by_PoolNames,missing,numeric,UnBoundNonLinDecompOp_by_PoolNames,numeric,InFluxList_by_PoolName-method

InFlux_by_PoolIndex

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

InFlux_by_PoolIndex(func, destinationIndex)

Arguments

func see method arguments
destinationIndex see method arguments

S4-methods

- InFlux_by_PoolIndex,Function,numeric-method
InFlux_by_PoolIndex, function, numeric-method

constructor from an ordered pair of PoolIndex (integer like) objects

Description

constructor from an ordered pair of PoolIndex (integer like) objects

Usage

## S4 method for signature 'function,numeric'
InFlux_by_PoolIndex(func, destinationIndex)

Arguments

- **func**: object of class: function, A function \( f(X,t) \) where \( X \) is a vector of the state variables. This form is required internally by the solvers and supported for backward compatibility with earlier versions of SoilR. Note that the function func given in this form can not be transformed to a different ordering of state variables, since the location of a state variable in the vector argument depends on a specific order and will be 'hardcoded' into your function. See InFlux_by_PoolName for the new, more powerful interface which allows subsequent reordering of the state variables by using the names of the state variables as formal arguments for func. In this case SoilR can infer (and later adapt) the vector argument form needed for the solvers.

- **destinationIndex**: object of class: numeric, no manual documentation

InFlux_by_PoolIndex-class

S4 class for the influx to a single pool identified by the index

Description

S4 class for the influx to a single pool identified by the index

InFlux_by_PoolName

Generic constructor for an influx to a single pool from an ordered pair of PoolName (string like) and function objects

Description

Generic constructor for an influx to a single pool from an ordered pair of PoolName (string like) and function objects

Usage

InFlux_by_PoolName(func, destinationName)
InFlux_by_PoolName-class

Arguments

func see method arguments
destinationName see method arguments

S4-methods

- InFlux_by_PoolName,function,character-method

InFlux_by_PoolName,function,character-method

Constructor from an ordered pair of PoolName (string like) and function objects

Description

Constructor from an ordered pair of PoolName (string like) and function objects

Usage

## S4 method for signature `function`,character'
InFlux_by_PoolName(func, destinationName)

Arguments

- func object of class: function, A function. The names of the formal arguments have to be a subset of the state variable names and the time symbol. This allows subsequent automatic reordering of the state variables. In the presence of a vector of state-variable-names the formulation can automatically be transformed to a function of a state VECTOR argument and time.

- destinationName object of class: character, PoolName (string like) object and a function

InFlux_by_PoolName-class

S4 class for the influx to a single pool identified by the name

Description

S4 class for the influx to a single pool identified by the name

S4-methods

S4-methods with class InFlux_by_PoolName in their signature:

- by_PoolIndex,InFlux_by_PoolName,character,character-method
### initialize,ConstLinDecompOp-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'ConstLinDecompOp'
initialize(.Object, mat = matrix())
```

**Arguments**

- `.Object`: object of class: `ConstLinDecompOp`, no manual documentation
- `mat`: no manual documentation

### initialize,DecompositionOperator-method

**Description**

automatic title

**Usage**

```r
## S4 method for signature 'DecompositionOperator'
initialize(
  .Object, 
  starttime = numeric(),
  endtime = numeric(),
  map = function(t) { t },
  lag = 0
)
```

**Arguments**

- `.Object`: object of class: `DecompositionOperator`, no manual documentation
- `starttime`: no manual documentation
- `endtime`: no manual documentation
- `map`: no manual documentation
- `lag`: no manual documentation
## Description

**automatic title**

**Usage**

```r
## S4 method for signature 'MCSim'
initialize(.Object, model = new(Class = "NLModel"), tasklist = list())
```

### Arguments

- `.Object`  
  - object of class: `MCSim`, no manual documentation
- `model`  
  - no manual documentation
- `tasklist`  
  - no manual documentation

---

## Description

*Internal method to supervise creation of objects of this class*

**Usage**

```r
## S4 method for signature 'Model'
initialize(
  .Object,
  times = c(0, 1),
  mat = ConstLinDecompOp(matrix(nrow = 1, ncol = 1, 0)),
  initialValues = numeric(),
  inputFluxes = BoundInFluxes(function(t) { return(matrix(nrow = 1, ncol = 1, 1)) }, 0, 1),
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

### Arguments

- `.Object`  
  - object of class: `Model`, no manual documentation
- `times`  
  - no manual documentation
- `mat`  
  - no manual documentation
- `initialValues`  
  - no manual documentation
initialize, Model_14-method

Internal method to supervise creation of objects of this class

Description

It is usually not necessary for user code to call this method. Its purpose is to enforce some sanity checks since it gets automatically called by new whenever an object of this class is created.

Usage

```r
## S4 method for signature 'Model_14'
initialize(
  .Object,
  times = c(0, 1),
  mat = ConstLinDecompOp(matrix(nrow = 1, ncol = 1, 0)),
  initialValues = numeric(),
  initialValF = ConstFc(values = c(0), format = "Delta14C"),
  inputFluxes = BoundInFluxes(function(t) { return(matrix(nrow = 1, ncol = 1, 1)) }, 0, 1),
  c14Fraction = BoundFc(function(t) { return(matrix(nrow = 1, ncol = 1, 1)) }, 0, 1),
  c14DecayRate = 0,
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

- `.Object` object of class: `Model_14`, no manual documentation
- `times` no manual documentation
- `mat` no manual documentation
- `initialValues` no manual documentation
- `initialValF` no manual documentation
- `inputFluxes` no manual documentation
- `c14Fraction` no manual documentation
- `c14DecayRate` no manual documentation
- `solverfunc` no manual documentation
- `pass` no manual documentation
### Description

automatic title

#### Usage

```r
## S4 method for signature 'Model_by_PoolNames'
initialize(  
  .Object,  
  times,  
  mat,  
  initialValues,  
  inputFluxes,  
  timeSymbol,  
  pass = FALSE,  
  solverfunc = deSolve.lsoda.wrapper
)
```

#### Arguments

- **.Object**: object of class: `Model_by_PoolNames`, no manual documentation
- **times**: no manual documentation
- **mat**: no manual documentation
- **initialValues**: no manual documentation
- **inputFluxes**: no manual documentation
- **timeSymbol**: no manual documentation
- **pass**: no manual documentation
- **solverfunc**: no manual documentation
### initialize, TimeMap-method

#### Usage

```r
## S4 method for signature 'NlModel'
initialize(
  .Object,
  times = c(0, 1),
  DepComp = new(Class = "TransportDecompositionOperator", 0, 1, function(t) {
    return(matrix(nrow = 1, ncol = 1, 0))
  }, function(t) {
    return(matrix(nrow = 1, ncol = 1, 0))
  }),
  initialValues = numeric(),
  inputFluxes = BoundInFluxes(function(t) {
    return(matrix(nrow = 1, ncol = 1, 1))
  }, 0, 1),
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

#### Arguments

- `.Object`: object of class: `NlModel`, no manual documentation
- `times`: no manual documentation
- `DepComp`: no manual documentation
- `initialValues`: no manual documentation
- `inputFluxes`: no manual documentation
- `solverfunc`: no manual documentation
- `pass`: no manual documentation

#### Description

automatic title

#### Usage

```r
## S4 method for signature 'TimeMap'
initialize(
  .Object,
  starttime = numeric(),
  endtime = numeric(),
  map = function(t) {
    t
  }
)
```

#### Arguments

- `.Object`: object of class: `TimeMap`, no manual documentation
- `starttime`: no manual documentation
- `endtime`: no manual documentation
- `map`: no manual documentation
initialize, TransportDecompositionOperator-method

automatic title

Description

automatic title

Usage

```r
## S4 method for signature 'TransportDecompositionOperator'
initialize(
  .Object,
  starttime = numeric(),
  endtime = numeric(),
  numberOfPools = 1,
  alpha = list(),
  f = function(t, O) { t },
  lag = 0
)
```

Arguments

<table>
<thead>
<tr>
<th>.Object</th>
<th>object of class: TransportDecompositionOperator, no manual documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>starttime</td>
<td>no manual documentation</td>
</tr>
<tr>
<td>endtime</td>
<td>no manual documentation</td>
</tr>
<tr>
<td>numberOfPools</td>
<td>no manual documentation</td>
</tr>
<tr>
<td>alpha</td>
<td>no manual documentation</td>
</tr>
<tr>
<td>f</td>
<td>no manual documentation</td>
</tr>
<tr>
<td>lag</td>
<td>no manual documentation</td>
</tr>
</tbody>
</table>

initialize, UnBoundInFluxes-method

automatic title

Description

automatic title

Usage

```r
## S4 method for signature 'UnBoundInFluxes'
initialize(.Object, map = function() {
})
```

Arguments

<table>
<thead>
<tr>
<th>.Object</th>
<th>object of class: UnBoundInFluxes, no manual documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>no manual documentation</td>
</tr>
</tbody>
</table>
Description

Automatic title

Usage

## S4 method for signature 'UnBoundLinDecompOp'
initialize(.Object, matFunc = function() {
  
})

Arguments

/Object/ object of class: UnBoundLinDecompOp, no manual documentation
/matFunc/ no manual documentation

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).
References


Examples

```r
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.

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.
- 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).
References


Examples

```r
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)
```

IntCal20

The IntCal20 northern hemisphere radiocarbon curve for the 0-55,000 yr BP period

Description

Atmospheric radiocarbon calibration curve for the period 0 to 55,000 yr BP for the northern hemisphere. This is the most recent update to the internationally agreed calibration curve and supersedes IntCal13.

Usage

data(IntCal20)

Format

A data frame with 9501 rows and 5 variables.

- CAL.BP   Calibrated age in years Before Present (BP).
- C14.age  C14 age in years BP.
- Delta.14C  Delta.14C value in per mil.

Details

All details about the derivation of this dataset are provided in Reimer et al. (2020).

Author(s)

Ingrid Chanca <ichanca@bgc-jena.mpg.de>
Source

<https://doi.org/10.1017/RDC.2020.41>

References


Examples

plot(IntCal20$CAL.BP, IntCal20$Delta.14C, type="l",
     xlab="cal BP", ylab="Delta14C (per mil)"
)

InternalFluxList_by_PoolIndex

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

InternalFluxList_by_PoolIndex(object)

Arguments

object see method arguments

S4-methods

• InternalFluxList_by_PoolIndex, list-method

InternalFluxList_by_PoolIndex, list-method

constructor from a normal list

Description

after checking the elements

Usage

## S4 method for signature 'list'
InternalFluxList_by_PoolIndex(object)

Arguments

object object of class:list, no manual documentation
**InternalFluxList_by_PoolIndex-class**

*S4-class for a list of internal fluxes with source and destination pool indices*

**Description**

S4-class for a list of internal fluxes with source and destination pool indices.

**InternalFluxList_by_PoolName**

*Generic constructor for the class with the same name*

**Description**

Generic constructor for the class with the same name.

**Usage**

```r
InternalFluxList_by_PoolName(object)
```

**Arguments**

- `object` see method arguments

**S4-methods**

- `InternalFluxList_by_PoolName,list-method`

**InternalFluxList_by_PoolName,list-method**

*constructor from a normal list*

**Description**

constructor from a normal list.

**Usage**

```r
## S4 method for signature 'list'
InternalFluxList_by_PoolName(object)
```

**Arguments**

- `object` object of class: `list`. A list. Either a list of elements of type `InternalFlux_by_PoolName` or a list where the names of the elements are strings of the form '1->3' (for the flux rate from pool 1 to 2).
An object of class `ConstantInFluxList_by_PoolIndex`
The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.

**Description**
S4-class for a list of internal fluxes with indexed by (source and destination pool) names

**S4-methods**
- `as.numeric,InternalFluxList_by_PoolName-method`
- `by_PoolIndex,InternalFluxList_by_PoolName,character,character-method`
- `UnBoundNonLinDecompOp_by_PoolNames,InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character-method`

**Description**
Generic constructor for the class with the same name

**Usage**
`InternalFlux_by_PoolIndex(func, sourceIndex, destinationIndex, src_to_dest)`

**Arguments**
- `func` see method arguments
- `sourceIndex` see method arguments
- `destinationIndex` see method arguments
- `src_to_dest` see method arguments

**S4-methods**
- `InternalFlux_by_PoolIndex,function,numeric,numeric,missing-method`
InternalFlux_by_PoolIndex, function, numeric, numeric, missing-method

constructor from an ordered pair of PoolIndex (integer like) objects and a function with vector argument

Description

constructor from an ordered pair of PoolIndex (integer like) objects and a function with vector argument

Usage

## S4 method for signature 'function', numeric, numeric, missing'

InternalFlux_by_PoolIndex(func, sourceIndex, destinationIndex)

Arguments

func object of class: function, A function \( f(X,t) \) where \( X \) is a vector of the state variables. This form is required internally by the solvers and supported for backward compatibility with earlier versions of SoilR. Note that the function func given in this form can not be transformed to a different ordering of state variables, since the location of a state variable in the vector argument depends on a specific order and will be 'hardcoded' into your function. See InternalFlux_by_PoolName for the new more powerful interface which allows subsequent reordering of the state variables by using the names of the state variables as formal arguments for func. In this case SoilR can infer (and later adapt) the vector argument form needed for the solvers. constructor from an ordered pair of PoolIndex (integer like) objects

sourceIndex object of class: numeric, no manual documentation

destinationIndex object of class: numeric, no manual documentation

InternalFlux_by_PoolIndex-class

S4-class for a single internal flux with source and destination pools specified by indices

Description

S4-class for a single internal flux with source and destination pools specified by indices
**InternalFlux_by_PoolName**  
*Generic constructor for the class with the same name*

**Description**  
Generic constructor for the class with the same name

**Usage**  
```
InternalFlux_by_PoolName(func, sourceName, destinationName, src_to_dest)
```

**Arguments**  
- **func**  
  see method arguments
- **sourceName**  
  see method arguments
- **destinationName**  
  see method arguments
- **src_to_dest**  
  see method arguments

**S4-methods**  
- `InternalFlux_by_PoolName,function,character,character,missing-method`
- `InternalFlux_by_PoolName,function,missing,missing,character-method`

**InternalFlux_by_PoolName,function,character,character,missing-method**  
*constructor from an ordered pair of PoolName (string like) objects and a function with the set of formal argument names forming a subset of the state_variable_names*

**Description**  
constructor from an ordered pair of PoolName (string like) objects and a function with the set of formal argument names forming a subset of the state_variable_names

**Usage**  
```
## S4 method for signature 'function',character,character,missing'
InternalFlux_by_PoolName(func, sourceName, destinationName)
```

**Arguments**  
- **func**  
  object of class: function, A real valued function describing the flux (mass/time) as function of (some of ) the state variables and time.
- **sourceName**  
  object of class: character, A string identifying the source pool of the flux
- **destinationName**  
  object of class: character, A string identifying the destination pool of the flux
## Description

auto

## Usage

```r
## S4 method for signature `function`, missing, missing, character'
InternalFlux_by_PoolName(func, src_to_dest)
```

## Arguments

- `func`: object of class: function, no manual documentation
- `src_to_dest`: object of class: character, no manual documentation

## Description

S4-class for a single internal flux with source and destination pools specified by name

## S4-methods

**S4-methods with class** `InternalFlux_by_PoolName` **in their signature::**

- `as.numeric`, `InternalFlux_by_PoolName-method`
- `by_PoolIndex`, `InternalFlux_by_PoolName`, `character`, `character-method`

## linearScalarModel

Implementation of a general model for linear non-autonomous systems with scalar modifiers

## Description

This function implements a linear model with scalar modifier for inputs and compartmental matrix.
linearScalarModel

Usage

\[
\text{linearScalarModel}(\text{t}, \text{A}, \text{C0}, \text{u}, \text{gamma}, \text{xi}, \text{xi}_{\text{lag}} = 0, \text{solver} = \text{deSolve.lsoda.wrapper})
\]

Arguments

- **t**: A vector containing the points in time where the solution is sought.
- **A**: A square (n x n) matrix with compartmental structure
- **C0**: A vector of length n containing the initial amount of carbon for the n pools.
- **u**: A vector of length n with constant mass inputs for the n pools.
- **gamma**: A scalar or data.frame object specifying the modifier for the mass inputs.
- **xi**: A scalar, data.frame, function or anything that can be converted to a scalar function of time \text{ScalarTimeMap} object specifying the external (environmental and/or edaphic) effects on decomposition rates.
- **xi_{lag}**: A time shift/delay for the automatically created time dependent function \text{xi}(t)
- **solver**: A function that solves the system of ODEs. This can be \text{euler} or \text{deSolve.lsoda.wrapper} or any other user provided function with the same interface.

Value

A Model Object that can be further queried

References


See Also

\text{RothCModel}. There are other \text{predefinedModels} and also more general functions like \text{Model}.

Examples

\[
t = \text{seq}(0, 52\times200, 1) \quad \# \text{ Fix me! Add an example.}
\]
**linesCPool**

Add lines with the output of `getC14`, `getC`, or `getReleaseFlux` to an existing plot

**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`.

**listProduct**

tensor product of lists

**Description**

Creates a list of all combinations of the elements of the input lists (like a "tensor product list ") The list elements can be of any class. The function is used in examples and tests to produce all possible combinations of arguments to a function. look at the tests for example usage

**Usage**

`listProduct(...)`

**Arguments**

- `...` lists

**Value**

a list of lists each containing one combinations of the elements of the input lists

**Examples**

`listProduct(list('a','b'),list(1,2))`
MCSim-class

Experimental Class for a Monte Carlo Simulation of particles leaving the pool

Description

Experimental Class for a Monte Carlo Simulation of particles leaving the pool

S4-methods

S4-methods with class MCSim in their signature::

- `[[,MCSim-method`
- `[[<-,MCSim-method`
- `availableParticleProperties,MCSim-method`
- `availableParticleSets,MCSim-method`
- `availableResidentSets,MCSim-method`
- `computeResults,MCSim-method`
- `getNumberOfPools,MCSim-method`
- `initialize,MCSim-method`
- `plot,MCSim-method`

Model

Constructor for class Model

Description

This function creates an object of class Model. The arguments can be given in different form as long as they can be converted to the necessary internal building blocks. (See the links)

Usage

Model(
  t,
  A,
  ivList,
  inputFluxes,
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE
)

Arguments

t A numeric vector containing the initial amount of carbon for the n pools. The length of this vector is equal to the number of pools. This is checked by an internal function.
inputFluxes something that can be converted by InFluxes to any of the available subclasses of InFluxes.
solverfunc The function used to actually solve the ODE system. The default is deSolve.lsoda.wrapper but you can also provide your own function that the same interface.
pass Forces the constructor to create the model even if it does not pass internal sanity checks

Details
This function Model wraps the internal constructor of class Model. The internal constructor requires the argument A to be of class DecompOp and argument inputFluxes to be of class InFluxes. Before calling the internal constructor Model calls GeneralDecompOp on its argument A and InFluxes on its argument inputFluxes to convert them into the required classes. Both are generic functions. Follow the links to see for which kind of inputs conversion methods are available. The attempted conversion allows great flexibility with respect to arguments and independence from the actual implementation. However if your code uses the wrong argument the error will most likely occur in the delegate functions. If this happens inspect the error message (or use traceback()) to see which function was called and try to call the constructor of the desired subclass explicitly with your arguments. The subclasses are linked in the class documentation DecompOp or InFluxes respectively.

Note also that this function checks its arguments quite elaborately and tries to detect accidental unreasonable combinations, especially concerning two kinds of errors.

1. unintended extrapolation of time series data
2. violations of mass balance by the DecompOp argument.

SoilR has a lot of unit tests which are installed with the package and are sometimes instructive as examples. To see example scenarios for parameter check look at:

Value
An object of class Model that can be queried by many methods to be found there.

See Also
This function is called by many of the predefinedModels.
Package functions called in the examples:
example.2DInFluxes.Args,
example.2DGeneralDecompOpArgs.

Examples
# vim: set ff=unix expandtab ts=2 sw=2:
test.all.possible.Model.arguments <- function(){
    # This example shows different kinds of arguments to the function Model.
    # The model objects we will build will share some common features.
    # - two pools
    # - initial values
    ivc<- c(5,6)
times <- seq(1,10,by=0.1)

    # The other parameters A and inputFluxes will be different
# The function Model will transform these arguments
# into objects of the classes required by the internal constructor.
# This leads to a number of possible argument types.
# We demonstrate some of the possibilities here.
# Let us first look at the choices for argument 'A'.

possibleAs <- example.2DGeneralDecompOpArgs()

Since "Model" will call "InFluxes" on its "inputFluxes"
argument there are again different choices
we have included a function in SoilR that produces 2D examples

possibleInfluxes <- example.2DInFluxes.Args()

We can build a lot of models from the possible combinations
# for instance
#m1 <- Model(
#  t=times,
#  A=matrix(ncol=2,byrow=TRUE,c(-0.1,0,0,-0.2)),
#  ivList=iv,
#  inputFluxes=possibleInfluxes$I.vec)

# We now produce all combinations of As and InputFluxes
combinations <- listProduct(possibleAs,possibleInfluxes)

# and a Model for each
models <- lapply(
  combinations,
  function(combi){
    #Model(t=times,A=combi$A,ivList=iv,inputFluxes=combi$I)
    Model(t=times,A=combi[[1]],ivList=iv,inputFluxes=combi[[2]])
  }
)

## lets check that we can compute something#
lapply(models,getC)

---

Model-class

S4 class representing a model run

Description

S4 class representing a model run

S4-methods

S4-methods with class Model in their signature::

- [,.Model,character,missing,missing-method
- getAccumulatedRelease,Model-method
- getC,Model-method
- getDecompOp,Model-method
- getInFluxes,Model-method
- getReleaseFlux,Model-method
Model_14

- getRightHandSideOfODE, Model-method
- getTimes, Model-method
- initialize, Model-method
- plot, Model-method

S4-subclasses

- Model_14

---

Model_14 general constructor for class Model_14

---

**Description**

This method tries to create an object from any combination of arguments that can be converted into the required set of building blocks for the Model_14 for n arbitrarily connected pools.

**Usage**

```r
Model_14(
  t,
  A,
  ivList,
  initialValF,
  inputFluxes,
  inputFc,
  c14DecayRate = -0.0001209681,
  solverfunc = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

**Arguments**

- **t** A vector containing the points in time where the solution is sought.
- **A** something that can be converted by GeneralDecompOp to any of the available subclasses of Decompo.
- **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.
- **initialValF** An object equal or equivalent to 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** something that can be converted by InFluxes to any of the available subclasses of InFluxes.
- **inputFc** An object describing the fraction of C_14 in per mille (different formats are possible)
- **c14DecayRate** the rate at which C_14 decays radioactively. If you don’t provide a value here we assume the following value: k=-0.0001209681 y^-1. This has the side effect that all your time related data are treated as if the time unit was year. Thus beside time itself it also affects decay rates the inputrates and the output
Model_14

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.

See Also

`TwopParallelModel`, `TwopSeriesModel`, `TwopFeedbackModel`

Examples

```r
# examples from external files
# inst/tests/requireSoilR/runit.all.possible.Model.arguments.R test.all.possible.Model.arguments:

# This example shows different kinds of arguments to the function Model.
# The model objects we will build will share some common features.
# - two pools
# - initial values
# iv<- c(5,6)
# - times
# times <- seq(1,10,by=0.1)
# The other parameters A and inputFluxes will be different
# The function Model will transform these arguments
# into objects of the classes required by the internal constructor.
# This leads to a number of possible argument types.
# We demonstrate some of the possibilities here.
# Let us first look at the choices for argument 'A'.

possibleAs <- example.2DGeneralDecompOpArgs()

# Since "Model" will call "InFluxes" on its "inputFluxes"
# argument there are again different choices
# we have included a function in SoilR that produces 2D examples

possibleInfluxes <- example.2DInFluxes.Args()
print(possibleInfluxes$I.vec)
# We can build a lot of models from the possible combinations
# for instance
#m1 <- Model(
#  t=times,
#  A=matrix(nrow=2,byrow=TRUE,c(-0.1,0,0,-0.2)),
#  ivList=iv,
#  inputFluxes=possibleInfluxes$I.vec)
## We now produce that all combinations of As and InputFluxes
combinations <- listProduct(possibleAs,possibleInfluxes)
print(length(combinations))
# an a Model for each
models <- lapply(
```

combinations,
  function(combi){
    #Model(t=times,A=combi$A,ivList=iv,inputFluxes=combi$I)
    Model(t=times,A=combi[[1]],ivList=iv,inputFluxes=combi[[2]])
  }
}

## lets check that we can compute something#
lapply(models,getC)

# This example describes the creation and use of a Model object that
# is defined by time dependent functions for decomposition and influx.
# The constructor of the Model-class (see ?Model)
# works for different combinations of
# arguments.
# Although Model (the constructor function for objects of this class
# accepts many many more convenient kinds of arguments,
# we will in this example call the constructor with arguments which
# are of the same type as one of the current internal
# representations in the
# Model object and create these arguments explicitly beforehand
# to demonstrate the approach with the most flexibility.
# We start with the Decomposition Operator.
# For this example we assume that we are able to describe the
# decomposition operator by explicit R functions that are valid
# for a finite time interval.
# Therefore we choose the appropriate sub class BoundLinDecompOp
# of DecompOp explicitly. (see ?'BoundLinDecompOp-class')
A=BoundLinDecompOp(
  ## We call the generic constructor (see ?BoundLindDecompOp)
  ## which has a method
  ## that takes a matrix-valued function of time as its first argument.
  ## (Although Model accepts time series data directly and
  ## will derive the internally used interpolating for you,
  ## the function argument could for instance represent the result
  ## of a very sophisticated interpolation performed by yourself)
  function(t){
    matrix(nrow=3,ncol=3,byrow=TRUE,
    c(-1, 0, 0,
     0.5, -2, 0,
     0, 1, sin(t)-1)
  },
  ## The other two arguments describe the time interval where the
  ## function is valid (the domain of the function)
  ## The interval will be checked against the domain of the InFlux
  ## argument of Model and against its 't' argument to avoid
  ## invalid computations outside the domain.
  ## (Inf and -Inf are possible values, but should only be used
  ## if the function is really valid for all times, which is
  ## especially untrue for functions resulting from interpolations,
  ## which are usually extremely misleading for arguments outside the
  ## domain covered by the data that has been used for the interpolation.)
  ## This is a safety net against wrong results origination from unitended EXTRApolation )
Model_14-class

S4-class to represent a ^14C model run

Description

S4-class to represent a ^14C model run

S4-methods

S4-methods with class Model_14 in their signature::

• getC14,Model_14-method
• getF14,Model_14-method
• getF14C,Model_14-method
• getF14R,Model_14-method
• getReleaseFlux14,Model_14-method
• initialize,Model_14-method

S4-methods with superclasses (in the package) of class Model_14 in their signature::

superclass Model:

• [,Model,character,missing,missing-method
Model_by_PoolNames

- getAccumulatedRelease, Model-method
- getC, Model-method
- getDecompOp, Model-method
- getInFluxes, Model-method
- getReleaseFlux, Model-method
- getRightHandSideOfODE, Model-method
- getTimes, Model-method
- initialize, Model-method
- plot, Model-method

S4-superclasses (in the package)
- Model

Model_by_PoolNames Constructor for Model_by_PoolNames

Description
Constructor for Model_by_PoolNames

Usage
Model_by_PoolNames(
  smod,
  times,
  mat,
  initialValues,
  inputFluxes,
  internal_fluxes,
  out_fluxes,
  timeSymbol,
  solverfunc
)

Arguments
smod see method arguments
times see method arguments
mat see method arguments
initialValues see method arguments
inputFluxes see method arguments
internal_fluxes see method arguments
out_fluxes see method arguments
timeSymbol see method arguments
solverfunc see method arguments
Value
A possibly nonlinear Model(run) that contains information about the pool names and connectivity of the pools and is therefore the preferred representation for new code.

S4-methods
- Model_by_PoolNames,missing,numeric,UnBoundNonLinDecompOp_by_PoolNames,numeric,InFluxList_by_PoolName,missing,missing,missing,missing-method
- Model_by_PoolNames,SymbolicModel_by_PoolNames,numeric,missing,numeric,missing,missing,missing,missing-method

Description
Create a model(run) described by fluxes
A flux and pool name based description is interesting for models where the traditional matrix based approach becomes difficult to manage:

1. For models with many pools the matrix representation makes the source code noisy and difficult to check.
2. Especially for nonlinear models, where the matrix is not only a function of time but also of the state vector the latter has to be decomposed in the user code.
3. Although mathematically equivalent the traditional matrix based representation is more opaque to automatic inspection by R. As a result it is not possible to automatically resolve the connectivity between the pools, or determine which pools have in/out-fluxes since for vector and matrix valued functions R can not determine which components are always zero.

The newer flux and pool name based approach has several advantages:

1. Instead of the whole matrix (nxn) only the existing fluxes have to be provided.
2. The fluxfunctions are scalar.
3. Nonlinear fluxfunctions can be written as functions of the state variable. The correct arguments are mapped automatically.
4. Since only the existing fluxes are provided the model structure can be found by inspection. E.g. connectivity graph can be drawn, which is very helpful to find mistakes in models with many pools.

Usage
```r
## S4 method for signature
## 'missing,
## numeric,
## UnBoundNonLinDecompOp_by_PoolNames,
## numeric,
## InFluxList_by_PoolName,
## missing,
## missing,
## missing,
## missing'
Model_by_PoolNames(times, mat, initialValues, inputFluxes)
```
Arguments

- **times**: object of class: numeric, no manual documentation
- **mat**: object of class: UnBoundNonLinDecompOp_by_PoolNames, UnBoundNonLinDecompOp_by_PoolNames
- **initialValues**: object of class: numeric, no manual documentation
- **inputFluxes**: object of class: InFluxList_by_PoolName, codeInFluxList_by_PoolName

Value

A possibly nonlinear Model(run) that contains information about the pool names and connectivity of the pools and is therefore the preferred representation for new code.

Description

Create a model(run) described by fluxes

A flux and pool name based description is interesting for models where the traditional matrix based approach becomes difficult to manage:

1. For models with many pools the matrix representation makes the source code noisy and difficult to check.
2. Especially for nonlinear models, where the matrix is not only a function of time but also of the state vector the latter has to be decomposed in the user code.
3. Although mathematically equivalent the traditional matrix based representation is more opaque to automatic inspection by R. As a result it is not possible to automatically resolve the connectivity between the pools, or determine which pools have in/out-fluxes since for vector and matrix valued functions R can not determine which components are always zero.

The newer flux and pool name based approach has several advantages:

1. Instead of the whole matrix (nxn) only the existing fluxes have to be provided.
2. The fluxfunctions are scalar.
3. Nonlinear fluxfunctions can be written as functions of the state variable. The correct arguments are mapped automatically.
4. Since only the existing fluxes are provided the model structure can be found by inspection. E.g. connectivity graph can be drawn, which is very helpful to find mistakes in models with many pools.

Usage

```r
## S4 method for signature
## 'SymbolicModel_by_PoolNames,
## numeric, missing, numeric, missing, missing, missing, missing,
## numeric, missing, missing, missing
## Create a model(run) described by fluxes
```
Model_by_PoolNames(smod, times, initialValues)

Arguments

smod: object of class: `SymbolicModel_by_PoolNames`, `SymbolicModel_by_PoolNames`
times: object of class: numeric, A vector
initialValues: object of class: numeric, no manual documentation

Value

A possibly nonlinear Model(run) that contains information about the pool names and connectivity of the pools and is therefore the preferred representation for new code.

Description

A model run based on flux functions

S4-methods

S4-methods with class `Model_by_PoolNames` in their signature:

- `getC,Model_by_PoolNames-method`
- `getReleaseFlux,Model_by_PoolNames-method`
- `getRightHandSideOfODE,Model_by_PoolNames-method`
- `getSolution,Model_by_PoolNames-method`
- `getTimes,Model_by_PoolNames-method`
- `initialize,Model_by_PoolNames-method`
- `plot,Model_by_PoolNames-method`

Description

deprecated class for a non-linear model run.
OnepModel

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

OnepModel(t, k, C0, In, xi = 1, solver = deSolve.lsoda.wrapper, pass = FALSE)

Arguments

t A vector containing the points in time where the solution is sought.

k A scalar with the decomposition rate of the pool.

C0 A scalar containing the initial amount of carbon in the pool.

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 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

References

See Also

There are other predefinedModels and also more general functions like Model.

Examples

```r

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 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 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

See Also

There are other predefinedModels and also more general functions like Model_14.

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"
)

---

**OutFlux**

*Generic constructor for the class with the same name*

**Description**

Generic constructor for the class with the same name

**Usage**

```
OutFlux(map, ...)
```

**Arguments**

- `map`: see method arguments
- `...`: see method arguments

---

**OutFluxList_by_PoolIndex**

*Generic constructor for the class with the same name*

**Description**

Generic constructor for the class with the same name

**Usage**

```
OutFluxList_by_PoolIndex(object)
```

**Arguments**

- `object`: see method arguments

**S4-methods**

- `OutFluxList_by_PoolIndex,list-method`
OutFluxList_by_PoolIndex, list-method

*constructor from a normal list*

**Description**

after checking the elements

**Usage**

```r
## S4 method for signature 'list'
OutFluxList_by_PoolIndex(object)
```

**Arguments**

- `object`: object of class `list`, no manual documentation

OutFluxList_by_PoolIndex-class

*A list of outfluxes*

**Description**

A list of outfluxes

OutFluxList_by_PoolName

*Generic constructor for the class with the same name*

**Description**

Generic constructor for the class with the same name

**Usage**

```r
OutFluxList_by_PoolName(object)
```

**Arguments**

- `object`: see method arguments

**S4-methods**

- `OutFluxList_by_PoolName, list-method`
OutFluxList_by_PoolName-class

constructor from a normal list

Description

constructor from a normal list

Usage

## S4 method for signature 'list'
OutFluxList_by_PoolName(object)

Arguments

object object of class: list. A list. Either a list of elements of type OutFlux_by_PoolName or a list where the names of the elements are integer strings.

Value

An object of class ConstantInFluxList_by_PoolIndex

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.

OutFluxList_by_PoolName-class

S4 class for a list of out-fluxes indexed by source pool name

Description

S4 class for a list of out-fluxes indexed by source pool name

S4-methods

S4-methods with class OutFluxList_by_PoolName in their signature::

• as.numeric,OutFluxList_by_PoolName-method
• by_PoolIndex,OutFluxList_by_PoolName,character,character-method
• UnBoundNonLinDecompOp_by_PoolNames,InternalFluxList_by_PoolName,OutFluxList_by_PoolName,cha
OutFlux_by_PoolIndex  

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

OutFlux_by_PoolIndex(func, sourceIndex)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>see method arguments</td>
</tr>
<tr>
<td>sourceIndex</td>
<td>see method arguments</td>
</tr>
</tbody>
</table>

S4-methods

- OutFlux_by_PoolIndex,function,numeric-method

----------

OutFlux_by_PoolIndex,function,numeric-method  

constructor from a PoolIndex (integer like) objects and a function with vector argument

Description

constructor from a PoolIndex (integer like) objects and a function with vector argument

Usage

## S4 method for signature 'function',numeric'

OutFlux_by_PoolIndex(func, sourceIndex)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>func</td>
<td>object of class:function, A function f(X,t) where X is a vector of the state variables. This form is required internally by the solvers and supported for backward compatibility with earlier versions of SoilR. Note that the function func given in this form can not be transformed to a different ordering of state variables, since the location of a state variable in the vector argument depends on a specific order and will be 'hardcoded' into your function. See OutFlux_by_PoolName for the new, more powerful interface which allows subsequent reordering of the state variables by using the names of the state variables as formal arguments for func. In this case SoilR can infer (and later adapt) the vector argument form needed for the solvers. constructor from an ordered pair of PoolIndex (integer like) objects</td>
</tr>
<tr>
<td>sourceIndex</td>
<td>object of class:numeric, no manual documentation</td>
</tr>
</tbody>
</table>
OutFlux_by_PoolName-class

S4 class for a single out-flux with source pool index

Description

S4 class for a single out-flux with source pool index

OutFlux_by_PoolName

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

OutFlux_by_PoolName(func, sourceName)

Arguments

func see method arguments
sourceName see method arguments

S4-methods

• OutFlux_by_PoolName,function,character-method

OutFlux_by_PoolName,function,character-method

constructor from a PoolName (integer like) object and a function

Description

constructor from a PoolName (integer like) object and a function

Usage

## S4 method for signature 'function',character
OutFlux_by_PoolName(func, sourceName)

Arguments

func object of class: function, A function. The names of the formal arguments have to be a subset of the state variable names and the time symbol. This allows subsequent automatic reordering of the state variables. In the presence of a vector of state variable names the formulation can automatically be transformed to a function of a state VECTOR argument and `time` constructor from an ordered pair of PoolName (integer like) objects
sourceName object of class: character, no manual documentation
OutFlux_by_PoolName-class

S4 class for a single out-flux with source pool name

Description
S4 class for a single out-flux with source pool name

S4-methods

S4-methods with class OutFlux_by_PoolName in their signature:

- by_PoolIndex,OutFlux_by_PoolName,character,character-method

ParallelModel

models for unconnected pools

Description
This function creates a (linear) 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

times A vector containing the points in time where the solution is sought.
coeffs_tm A TimeMap 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.
startvalues 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.
inputrates An object consisting of a vector valued function describing the inputs to the pools as functions of time TimeMap.new
solverfunc 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.
pass if TRUE forces the constructor to create the model even if it is invalid
Examples

```r
t_start=0
t_end=10
tn=50
timestep=(t_end-t_start)/tn
t=seq(t_start,t_end,timestep)
k=TimeMap(
  function(times){c(-0.5,-0.2,-0.3)},
  t_start,
  t_end
)
c0=c(1, 2, 3)
#constant inputrates
inputrates=BoundInFluxes(
  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],lty=lt3,col=col3)
legend("topright",c("R1","R2","R3"),lty=c(lt1,lt2,lt3),col=c(col1,col2,col3))
```

Description

Computes the entropy of particles passing through the whole network of compartments for a model at equilibrium

Usage

`pathEntropy(A, u)`
Arguments

A  A constant compartmental square matrix with cycling rates in the diagonal and transfer rates in the off-diagonal.
u  A one-column matrix defining the amount of inputs per compartment.

Value

A scalar value with the path entropy

References


Examples

B6=matrix(c(-1,1,0,0,-1,0,0,-1),3,3); u6=matrix(c(1,0,0))
pathEntropy(A=B6, u=u6)

Description

automatic title

Usage

## S4 method for signature 'MCSim'
plot(x, y, ...)

Arguments

x  object of class: MCSim, no manual documentation
y  no manual documentation
...  no manual documentation

Description

The method solves the model and plots the solutions. It is intended to provide a quick overview.

Usage

## S4 method for signature 'Model'
plot(x)

Arguments

x  object of class: Model, The model (run) the results of which are plotted
plot,Model_by_PoolNames-method

*Plot the graph of pool connections*

**Description**
Plot the graph of pool connections

**Usage**
```r
## S4 method for signature 'Model_by_PoolNames'
plot(x)
```

**Arguments**
- `x` object of class: `Model_by_PoolNames`, The model (run) the results of which are plotted

plot,NlModel-method

*automatic title*

**Description**
automatic title

**Usage**
```r
## S4 method for signature 'NlModel'
plot(x)
```

**Arguments**
- `x` object of class: `NlModel`, no manual documentation

plot,TimeMap-method

*automatic title*

**Description**
automatic title

**Usage**
```r
## S4 method for signature 'TimeMap'
plot(x, y, ...)
```

**Arguments**
- `x` object of class: `TimeMap`, no manual documentation
- `y` no manual documentation
- `...` no manual documentation
plotC14Pool

Plots the output of `getF14` 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 `getF14`.

**Usage**

```r
plotC14Pool(t, mat, inputFc, col, ...)
```

**Arguments**

- `t`: A vector containing the time points for plotting.
- `mat`: A matrix object obtained after a call to `getF14`.
- `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`.

---

plotCPool

Plots the output of `getC` or `getReleaseFlux` for each pool over time

**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**

```r
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 `link{plot}`.
plotPoolGraph

Generic plotter

Description

Generic plotter

Usage

plotPoolGraph(x)

Arguments

x An argument containing sufficient information about the connections between the pools as well as from and to the exterior.

S4-methods

• plotPoolGraph,SymbolicModel_by_PoolNames-method

plotPoolGraph,SymbolicModel_by_PoolNames-method

Plot the graph of pool connections

Description

Plot the graph of pool connections

Usage

## S4 method for signature 'SymbolicModel_by_PoolNames'
plotPoolGraph(x)

Arguments

x object of class:SymbolicModel_by_PoolNames. The modelrun the connection graph of which is plotted
plotPoolGraphFromTupleLists

Helper function to draw connectivity graphs

Description

Helper function to draw connectivity graphs

Usage

plotPoolGraphFromTupleLists(
  internalConnections,
  inBoundConnections,
  outBoundConnections
)

Arguments

internalConnections
  A list of tuples(source, dest) where src and dest are either both integers or both
  strings(poolnames)
inBoundConnections
  A list of either integers or strings (poolnames)
outBoundConnections
  A list of either integers or strings (poolnames) The function is used by the
  plotPoolGraph generic of the newer model classes SymbolicModel_by_PoolNames.

PoolConnection_by_PoolIndex

automatic title

Description

automatic title

Usage

PoolConnection_by_PoolIndex(source, destination, src_to_dest)

Arguments

source        see method arguments
destination   see method arguments
src_to_dest   see method arguments

S4-methods

- PoolConnection_by_PoolIndex, ANY, ANY, missing-method
- PoolConnection_by_PoolIndex, missing, missing, character-method
PoolConnection_by_PoolIndex, ANY, ANY, missing-method

constructor from an ordered pair of PoolId objects

Description

constructor from an ordered pair of PoolId objects

Usage

## S4 method for signature 'ANY, ANY, missing'
PoolConnection_by_PoolIndex(source, destination)

Arguments

source no manual documentation
destination no manual documentation

---

PoolConnection_by_PoolIndex, missing, missing, character-method

constructor from strings of the form '1_to_2'

Description

constructor from strings of the form '1_to_2'

Usage

## S4 method for signature 'missing, missing, character'
PoolConnection_by_PoolIndex(src_to_dest)

Arguments

src_to_dest object of class: character, no manual documentation

---

PoolConnection_by_PoolIndex-class

Objects that have a source and a destination described by integer like objects (of class PoolIndex)

Description

Examples are internal Fluxes and Fluxrates Their 'topologic' part and many related sanity checks are implemented here rather than in every function that uses fluxes or rates The methods are also essential for the translation from (internal) flux lists to the respective parts of compartmental matrices and back
Description
automatic title

Usage
PoolConnection_by_PoolName(source, destination, src_to_dest)

Arguments
- source: see method arguments
- destination: see method arguments
- src_to_dest: see method arguments

S4-methods
- `PoolConnection_by_PoolName,ANY,ANY,missing-method`

Description
constructor from an ordered pair of PoolName objects

Usage
```r
## S4 method for signature 'ANY,ANY,missing'
PoolConnection_by_PoolName(source, destination)
```

Arguments
- source: no manual documentation
- destination: no manual documentation

Description
Objects that have a source and a destination determined by a string like object of class PoolName

Examples are internal Fluxes and Fluxrates Their 'topologic' part and many related sanity checks are implemented here rather than in every function that uses fluxes or rates The methods are also essential for the translation from (internal) flux lists to the respective parts of compartmental matrices and back
PoolId-class  
common class for pool ids

Description
examples for ids are index or name

S4-subclasses
- PoolIndex
- PoolName

PoolIndex  
automatic title

Description
automatic title

Usage
PoolIndex(id, ...)

Arguments
id  see method arguments
...  see method arguments

S4-methods
- PoolIndex,character-method
- PoolIndex,numeric-method
- PoolIndex,PoolIndex-method
- PoolIndex,PoolName-method
**PoolIndex, character-method**

*construct from number string like '1' or '3'*

**Description**

construct from number string like '1' or '3'

**Usage**

```r
## S4 method for signature 'character'
PoolIndex(id)
```

**Arguments**

- `id` object of class: character, no manual documentation

**PoolIndex, numeric-method**

*construct from number*

**Description**

construct from number

**Usage**

```r
## S4 method for signature 'numeric'
PoolIndex(id)
```

**Arguments**

- `id` object of class: numeric, no manual documentation

**PoolIndex, PoolIndex-method**

*pass through constructor from an object of the same class*

**Description**

This is here to be able to call PoolIndex on a PoolIndex object without having to check before if it is necessary. the unnecessary poolNames argument will be ignored.

**Usage**

```r
## S4 method for signature 'PoolIndex'
PoolIndex(id, poolNames)
```
PoolIndex-class

Arguments

id
object of class: PoolIndex, no manual documentation

poolNames
no manual documentation

Description

convert to number like object

Usage

## S4 method for signature 'PoolName'
PoolIndex(id, poolNames)

Arguments

id
object of class: PoolName, no manual documentation

poolNames
no manual documentation

S4-methods

S4-methods with class PoolIndex in their signature::

• PoolIndex,PoolIndex-method
• PoolName,PoolIndex-method

S4-superclasses (in the package)

• PoolId
PoolName

Description

automatic title

Usage

PoolName(id, ...)

Arguments

<table>
<thead>
<tr>
<th>id</th>
<th>see method arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>see method arguments</td>
</tr>
</tbody>
</table>

S4-methods

- PoolName,character-method
- PoolName,PoolIndex-method
- PoolName,PoolName-method

---

PoolName,character-method

construct from string with checks

Description

construct from string with checks

Usage

## S4 method for signature 'character'

PoolName(id)

Arguments

| id | object of class:character, no manual documentation |
PoolName,PoolIndex-method

*convert to string like object*

**Description**

convert to string like object

**Usage**

```r
## S4 method for signature 'PoolIndex'
PoolName(id, poolNames)
```

**Arguments**

- `id` object of class: `PoolIndex`, no manual documentation
- `poolNames` no manual documentation

---

PoolName,PoolName-method

*pass through constructor from an object of the same class*

**Description**

This is here to be able to call PoolName on a PoolName object without having to test before if we have to.

**Usage**

```r
## S4 method for signature 'PoolName'
PoolName(id, poolNames)
```

**Arguments**

- `id` object of class: `PoolName`, no manual documentation
- `poolNames` no manual documentation
### PoolName-class

**class for pool-name-strings**

**Description**

used to control the creation of PoolName objects which have to be valid R identifiers and to dispatch pool name specific methods like conversion to pool indices

**S4-methods**

**S4-methods with class PoolName in their signature:**

- PoolIndex, PoolName-method
- PoolName, PoolName-method

**S4-superclasses (in the package)**

- PoolId

### predefinedModels

**PREDEFINED MODELS**

**Description**

<table>
<thead>
<tr>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>GaudinskiModel14</td>
</tr>
<tr>
<td>ICBMModel</td>
</tr>
<tr>
<td>OnepModel</td>
</tr>
<tr>
<td>OnepModel14</td>
</tr>
<tr>
<td>RothCModel</td>
</tr>
<tr>
<td>ThreeFeedbackModel</td>
</tr>
<tr>
<td>ThreeFeedbackModel14</td>
</tr>
<tr>
<td>ThreeParallelModel</td>
</tr>
<tr>
<td>ThreeParallelModel14</td>
</tr>
<tr>
<td>ThreeSeriesModel</td>
</tr>
<tr>
<td>ThreeSeriesModel14</td>
</tr>
<tr>
<td>TwoFeedbackModel</td>
</tr>
<tr>
<td>TwoFeedbackModel14</td>
</tr>
<tr>
<td>TwoParallelModel</td>
</tr>
<tr>
<td>TwoParallelModel14</td>
</tr>
<tr>
<td>TwoMMmodel</td>
</tr>
<tr>
<td>ThreepairMMmodel</td>
</tr>
<tr>
<td>TwoSeriesModel</td>
</tr>
<tr>
<td>TwoSeriesModel14</td>
</tr>
<tr>
<td>YassoModel</td>
</tr>
<tr>
<td>Yasso07Model</td>
</tr>
<tr>
<td>SeriesLinearModel</td>
</tr>
<tr>
<td>SeriesLinearModel14</td>
</tr>
<tr>
<td>CenturyModel</td>
</tr>
<tr>
<td>WangThreePoolNonAutonomous_sym</td>
</tr>
</tbody>
</table>
## S4 method for signature 'NlModel'

```r
print(x)
```

### Arguments

- `x`: object of class `NlModel`, no manual documentation

### Description

This function computes the respiration coefficients as function of time for all pools according to the given matrix function $A(t)$.

### Usage

```r
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.
RothCModel

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,
  FYM = 0,
  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.
  FYM A scalar or data.frame object specifying the amount of Farm Yard Manure 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 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

Value

A Model Object that can be further queried

References

See Also

There are other predefinedModels and also more general functions like Model.

Examples

t=0:500
Ex=RothCModel(t)
Ct=getC(Ex)
Rt=getReleaseFlux(Ex)

matplot(t,Ct,type="l",col=1:5, ylim=c(0,25),
ylab=expression(paste("Carbon stores (Mg C \"", ha^-1,\"\")")),
 xlab="Time (years)", lty=1)
lines(t,rowSums(Ct),lwd=2)
legend("topleft",
c("Pool 1, DPM",
 "Pool 2, RPM",
 "Pool 3, BIO",
 "Pool 4, HUM",
 "Pool 5, IOM",
 "Total Carbon"),
lty=1,
lwd=c(rep(1,5),2),
col=c(1:5,1),
bty="n"
)

ScalarTimeMap

Constructor for ScalarTimeMap-class

Description

Constructor for ScalarTimeMap-class

Usage

ScalarTimeMap(
  map,
  starttime,
  endtime,
  times,
  data,
  lag = 0,
  interpolation = splinefun,
  ...
)

Arguments

map see method arguments
starttime see method arguments
endtime see method arguments
ScalarTimeMap, data.frame, missing, missing, missing, missing-method

times see method arguments
data see method arguments
lag see method arguments
interpolation see method arguments
... see method arguments

S4-methods

- ScalarTimeMap, data.frame, missing, missing, missing, missing-method
- ScalarTimeMap, function, missing, missing, missing, missing-method
- ScalarTimeMap, function, numeric, numeric, missing, missing-method
- ScalarTimeMap, missing, missing, missing, missing, numeric-method
- ScalarTimeMap, missing, missing, missing, numeric, numeric-method

ScalarTimeMap, data.frame, missing, missing, missing, missing-method

constructor for data given as 2 column data.frame

Description
constructor for data given as 2 column data.frame

Usage
## S4 method for signature 'data.frame,missing,missing,missing,missing'
ScalarTimeMap(map, lag = 0, interpolation = splinefun)

Arguments
map object of class: data.frame, In this case a data.frame. Only the first two columns will be used
lag a (scalar) delay
interpolation the interpolation, usually splinefun or approxfun

ScalarTimeMap, function, missing, missing, missing, missing-method

manual constructor for just a function

Description
The interval will be set to [-Inf,Inf]

Usage
## S4 method for signature 'function',missing,missing,missing,missing'
ScalarTimeMap(map, lag = 0)
## Arguments

- **map**: object of class: function, no manual documentation
- **lag**: no manual documentation

### Description

manual constructor for a function and an interval

### Usage

```r
## S4 method for signature 'function',numeric,numeric,missing,missing'
ScalarTimeMap(map, starttime, endtime, lag = 0)
```

### Arguments

- **map**: object of class: function, no manual documentation
- **starttime**: object of class: numeric, no manual documentation
- **endtime**: object of class: numeric, no manual documentation
- **lag**: no manual documentation

### Description

special case for a time map from a constant

### Usage

```r
## S4 method for signature 'missing,missing,missing,missing,numeric'
ScalarTimeMap(starttime = -Inf, endtime = +Inf, data, lag = 0)
```

### Arguments

- **starttime**: object of class: missing, no manual documentation
- **endtime**: object of class: missing, no manual documentation
- **data**: object of class: numeric, no manual documentation
- **lag**: no manual documentation
ScalarTimeMap-times-numeric-method

constructor for data and times given as vectors

Description

constructor for data and times given as vectors

Usage

## S4 method for signature 'missing,missing,missing,numeric,numeric'
ScalarTimeMap(times, data, lag = 0, interpolation = splinefun)

Arguments

- **times**: object of class: numeric, (the times for the values in data)
- **data**: object of class: numeric, the values at times
- **lag**: a (scalar) delay
- **interpolation**: the interpolation, usually splinefun or approxfun

ScalarTimeMap-class

S4 class for a scalar time dependent function on a finite time interval

Description

S4 class for a scalar time dependent function on a finite time interval

S4-methods

S4-methods with class ScalarTimeMap in their signature:
- ConstLinDecompOpWithLinearScalarFactor, matrix, missing, missing, missing, ScalarTimeMap-method

S4-methods with superclasses (in the package) of class ScalarTimeMap in their signature:

  superclass TimeMap:
  - add_plot, TimeMap-method
  - as.character, TimeMap-method
  - GeneralDecompOp, TimeMap-method
  - getFunctionDefinition, TimeMap-method
  - getTimeRange, TimeMap-method
  - InFluxes, TimeMap-method
  - initialize, TimeMap-method
  - plot, TimeMap-method
  - TimeMap, TimeMap, ANY, ANY, ANY, ANY-method

S4-subclasses

- BoundFc
SeriesLinearModel

S4-superclasses (in the package)

- TimeMap

---

**SeriesLinearModel** General m-pool linear model with series structure

**Description**

This function creates a model for m number of pools connected in series. It is a wrapper for the more general function `GeneralModel`.

**Usage**

```r
SeriesLinearModel(
  t,
  m.pools,
  ki,
  Tij,
  C0,
  In,
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

**Arguments**

- **t**
  A vector containing the points in time where the solution is sought.

- **m.pools**
  An integer with the total number of pools in the model.

- **ki**
  A vector of length m containing the values of the decomposition rates for each pool i.

- **Tij**
  A vector of length m-1 with the transfer coefficients from pool j to pool i. The value of these coefficients must be in the range [0, 1].

- **C0**
  A vector of length m containing the initial amount of carbon for the m pools.

- **In**
  A 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 `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

**References**


**See Also**

There are other `predefinedModels` and also more general functions like `Model`.
SeriesLinearModel14

Examples

# A five-pool model
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, k4=0.1,k5=0.05)
Ts=c(0.5,0.2,0.2,0.1)
C0=c(C10=100,C20=150, C30=50, C40=50, C50=10)
In = 50
#
Ex1=SeriesLinearModel(t=t,m.pools=5,ki=ks,Tij=Ts,C0=C0,In=In,xi=fT.Q10(15))
Ct=getC(Ex1)
#
matplot(t,Ct,type="l",col=2:6,lty=1,ylim=c(0,sum(C0)))
lines(t,rowSums(Ct),lwd=2)
legend("topright",c("Total C","C in pool 1", "C in pool 2","C in pool 3",
"C in pool 4","C in pool 5"),
lty=1,col=1:6,lwd=c(2,rep(1,5)),bty="n")
ki A vector of length m containing the values of the decomposition rates for each pool i.
Tij A vector of length m-1 with the transfer coefficients from pool j to pool i. The value of these coefficients must be in the range [0, 1].
C0 A vector of length m containing the initial amount of carbon for the m pools.
F0_Delta14C A vector of length m containing the initial amount of the radiocarbon fraction for the m pools.
In A 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.
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 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 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

References

See Also
There are other predefinedModels and also more general functions like Model.

Examples
years=seq(1901,2009,by=0.5)
LitterInput=700
Ex=SeriesLinearModel14(
t=years, ki=c(k1=1/2.8, k2=1/35, k3=1/100), m.pools=3, C0=c(200,5000,500), F0_Delta14C=c(0,0,0), In=LitterInput, Tij=c(0.5, 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)
lines(years, C14t[,3],col=4,lwd=3)
legend("topright",
The SHCal20 southern hemisphere radiocarbon curve for the 0-55,000 yr BP period

Description

Atmospheric radiocarbon calibration curve for the period 0 to 55,000 yr BP for the southern hemisphere.

Usage

data(SHCal20)

Format

A data frame with 9501 rows and 5 variables.

CAL.BP  Calibrated age in years Before Present (BP).
C14.age  C14 age in years BP.
Delta.14C  Delta.14C value in per mil.

Details

All details about the derivation of this dataset are provided in Hogg et al. (2020).

Author(s)

Ingrid Chanca <ichanca@bgc-jena.mpg.de>

Source

<https://doi.org/10.1017/RDC.2020.59>

References

**Examples**

```r
plot(SHCal20$CAL.BP, SHCal20$Delta.14C, type="l",
     xlab="cal BP", ylab="Delta14C (per mil)"
```

---

**Description**

Automatic title

**Usage**

```r
## S4 method for signature 'NlModel'
show(object)
```

**Arguments**

- `object`: object of class `NlModel`, no manual documentation

---

**SoilR.F0.new**

deprecated function that used to create an object of class `SoilR.F0`

**Description**

The function internally calls the constructor of the replacement class `ConstFc-class`.

**Usage**

```r
SoilR.F0.new(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-class` that contains data and a format description that can later be used to convert the data into other formats if the conversion is implemented.
**StateDependentInFluxVector-class**

*Input vector that is a function of the pool content and time*

---

**Description**

Input vector that is a function of the pool content and time

**S4-methods**

- **S4-methods with class** StateDependentInFluxVector in their signature::
  - `getFunctionDefinition,StateDependentInFluxVector-method`

- **S4-methods with superclasses (in the package) of class** StateDependentInFluxVector in their signature::
  - **superclass** InFluxes:
    - `InFluxes,InFluxes-method`

**S4-superclasses (in the package)**

- `InFluxes`

---

**StateIndependentInFluxList_by_PoolIndex**

*Generic constructor for the class with the same name*

---

**Description**

Generic constructor for the class with the same name

**Usage**

`StateIndependentInFluxList_by_PoolIndex(object)`

**Arguments**

- `object` see method arguments

**S4-methods**

- `StateIndependentInFluxList_by_PoolIndex,list-method`
StateIndependentInFluxList_by_PoolIndex_class

**Description**

constructor from a normal list

**Usage**

```r
## S4 method for signature 'list'
StateIndependentInFluxList_by_PoolIndex(object)
```

**Arguments**

- `object` object of class: list. A list. Either a list of elements of type `StateIndependentInFlux_by_PoolIndex` or a list where the names of the elements are strings of the form '3' (for an in flux connected to pool 3)

**Value**

An object of class `StateIndependentInFluxList_by_PoolIndex`

The function checks if the elements are of the desired type or can be converted to it. It is mainly used internally and usually called by the front end functions to convert the user supplied arguments.

StateIndependentInFluxList_by_PoolIndex-class

**Description**

Subclass of list that is guaranteed to contain only elements of type `StateIndependentInFlux_by_PoolIndex`

**S4-methods**

S4-methods with class `StateIndependentInFluxList_by_PoolIndex` in their signature::

- `InFluxes,StateIndependentInFluxList_by_PoolIndex-method`
StateIndependentInFluxList_by_PoolName

_Generic constructor for the class with the same name_

**Description**

Generic constructor for the class with the same name

**Usage**

StateIndependentInFluxList_by_PoolName(object)

**Arguments**

- **object** see method arguments

StateIndependentInFlux_by_PoolIndex-class

_Constructor for the class with the same name_

**Description**

Constructor for the class with the same name

**Slots**

- destinationIndex
- flux

**state_variable_names**  _determine the minimum set of statevariables_

**Description**

determine the minimum set of statevariables

**Usage**

state_variable_names(object)

**Arguments**

- **object** The symbolic model description
SymbolicModel_by_PoolNames-class

A symbolic model description based on flux functions

Description

The set of flux functions along with the timesymbol is complete description of the structure

S4-methods

S4-methods with class SymbolicModel_by_PoolNames in their signature::

• Model_by_PoolNames,SymbolicModel_by_PoolNames,numerics,missing,numeric,missing,missing,missing,missing-

• plotPoolGraph,SymbolicModel_by_PoolNames-method

systemAge

System and pool age for constant compartment models

Description

Computes the density distribution and mean for the system and pool ages of a constant compartmental model in matrix representation

Usage

systemAge(A, u, a = seq(0, 100), q = c(0.05, 0.5, 0.95))

Arguments

A A constant compartmental square matrix with cycling rates in the diagonal and transfer rates in the off-diagonal.

u A one-column matrix defining the amount of inputs per compartment.

a A sequence of ages to calculate density functions

q A vector of probabilities to calculate quantiles of the system age distribution

Value

A list with 5 objects: mean system age, system age distribution, quantiles of system age distribution, mean pool-age, and pool-age distribution.

See Also

transitTime
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 (m3 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**: a vector of length 6 with the initial values of the SOM pools and the microbial biomass pools (g m-3)

Value

An object of class NlModel that can be further queried.

See Also

There are other predefinedModels and also more general functions like Model.

Examples

```r
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),
bt="n")
#State-space diagram
```
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
genral 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 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
References


See Also

There are other predefinedModels and also more general functions like Model.

Examples

```r
t_start=0
t_end=10	n=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,]))
)
```

Implementation of a three-pool C14 model with feedback structure

```r
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
ThreepFeedbackModel14

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 connections. 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**: A vector of length 3 containing the initial fraction of radiocarbon for the 3 pools in Delta14C format. The format will be assumed to be Delta14C, so please take care that it is.
- **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.
- **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 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 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. This is sometimes useful when SoilR is used by external packages for parameter estimation.

See Also

There are other predefinedModels and also more general functions like Model_14.

Examples

```r
#years=seq(1901,2009,by=0.5)
years=seq(1904,2009,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),
)```
\[ F_0 \_\Delta^{14}C = c(0, 0, 0), \]
\[ \text{In} = \text{Litter Input}, \]
\[ \text{gam1} = 0.6, \]
\[ \text{gam2} = 0.2, \]
\[ \text{inputFc} = \text{C14Atm\_NH}, \]
\[ \text{lag} = 2 \]

\[ \text{P.R14m} = \text{getF14R(Parallel)} \]
\[ \text{P.C14m} = \text{getF14C(Parallel)} \]
\[ \text{P.C14t} = \text{getF14(Parallel)} \]

\[ \text{par(mfrow} = c(3, 2)) \]
\[ \text{plot(} \]
\[ \text{C14Atm\_NH, } \]
\[ \text{type} = "l", \]
\[ \text{xlab} = "Year", \]
\[ \text{ylab} = \text{expression(paste(Delta^{14},"C ","(\u2030)"))}, \]
\[ \text{xlim} = c(1940, 2010) \]
\[ \} \]
\[ \text{lines(years, P.C14t[,1], col=4)} \]
\[ \text{lines(years, P.C14t[,2],col=4,lwd=2)} \]
\[ \text{lines(years, P.C14t[,3],col=4,lwd=3)} \]
\[ \text{legend("topright",} \]
\[ \text{c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),} \]
\[ \text{lty=rep(1,4),} \]
\[ \text{col=c(1,4,4,4),} \]
\[ \text{lwd=c(1,1,2,3),} \]
\[ \text{bty="n"}) \]

\[ \text{plot(C14Atm\_NH,type} = "l",xlab="Year",} \]
\[ \text{ylab=expression(paste(Delta^{14},"C ","(\u2030)")),xlim=c(1940,2010))} \]
\[ \text{lines(years,P.C14m,col=4)} \]
\[ \text{lines(years,P.R14m,col=2)} \]
\[ \text{legend("topright",c("Atmosphere","Bulk SOM", "Respired C"),} \]
\[ \text{lty=c(1,1,1),} \]
\[ \text{col=c(1,4,2),bty="n"}) \]

\[ \text{plot(C14Atm\_NH,type=} = "l",xlab="Year",} \]
\[ \text{ylab=expression(paste(Delta^{14},"C ","(\u2030)")),xlim=c(1940,2010))} \]
\[ \text{lines(years,S.C14t[,1], col=4)} \]
\[ \text{lines(years,S.C14t[,2],col=4,lwd=2)} \]
\[ \text{lines(years,S.C14t[,3],col=4,lwd=3)} \]
\[ \text{legend("topright",c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),} \]
\[ \text{lty=rep(1,4),col=c(1,4,4,4),lwd=c(1,1,2,3),bty="n"}) \]

\[ \text{plot(C14Atm\_NH,type} = "l",xlab="Year",} \]
\[ \text{ylab=expression(paste(Delta^{14},"C ","(\u2030)")),xlim=c(1940,2010))} \]
\[ \text{lines(years,S.R14m,col=2)} \]
\[ \text{legend("topright",c("Atmosphere","Bulk SOM", "Respired C"),} \]
\[ \text{lty=c(1,1,1),} \]
\[ \text{col=c(1,4,2),bty="n"}) \]

\[ \text{plot(C14Atm\_NH,type=} = "l",xlab="Year",} \]
\[ \text{ylab=expression(paste(Delta^{14},"C ","(\u2030)")),xlim=c(1940,2010))} \]
\[ \text{lines(years,F.C14t[,1], col=4)} \]
\[ \text{lines(years,F.C14t[,2],col=4,lwd=2)} \]
Implementation of a three pool model with parallel structure

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

```r
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 `deSolve.lsoda.wrapper` or any other user provided function with the same interface.
- `pass` Logical that forces the Model to be created even if the check suggest problems.
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 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 `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

See Also

There are other predefinedModels and also more general functions like Model_14.

Examples

```r
years=seq(1903,2009,by=0.5) # note that we
LitterInput=700

Ex=ThreeParallelModel14(
t=years,
ks=c(k1=1/2.8, k2=1/35, k3=1/100),
```
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,  # vector of time points
  ks,  # vector of decay constants
  a21,  # transfer constant from pool 2 to pool 1
  a32,  # transfer constant from pool 3 to pool 2
  C0,  # initial conditions
  F0_Delta14C,  # initial conditions of 14C
  In=LitterInput,  # input
  gam1,  # transfer constant
  gam2,  # transfer constant
  inputFc=C14Atm_NH,  # input function
  lag  # lag time
)
ThreepSeriesModel

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.
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 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

Value
A Model Object that can be further queried

References

See Also
There are other predefinedModels and also more general functions like Model.

Examples

t_start=0
t_end=10	n=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)
Implementation of a three-pool C14 model with series structure

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**

```r
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 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 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

Value A Model Object that can be further queried

See Also There are other predefinedModels and also more general functions like Model_14.

Examples

```r
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

Constructor for TimeMap-class

Description

Constructor for TimeMap-class

Usage

TimeMap(
  map,
  starttime,
  endtime,
  times,
  data,
  lag = 0,
  interpolation = splinefun,
  ...
)

Arguments

- map: see method arguments
- starttime: see method arguments
- endtime: see method arguments
- times: see method arguments
- data: see method arguments
- lag: see method arguments
- interpolation: see method arguments
- ...: see method arguments

S4-methods

- TimeMap, data.frame, missing, missing, missing, missing-method
- TimeMap, function, missing, missing, missing, missing-method
- TimeMap, function, numeric, numeric, missing, missing-method
- TimeMap, list, missing, missing, missing, missing-method
- TimeMap, missing, missing, missing, numeric, array-method
- TimeMap, missing, missing, missing, numeric, list-method
- TimeMap, missing, missing, missing, numeric, matrix-method
- TimeMap, missing, missing, numeric, numeric-method
- TimeMap, TimeMap, ANY, ANY, ANY, ANY-method
Description

automatic title

Usage

## S4 method for signature 'data.frame,missing,missing,missing,missing,missing'
TimeMap(map, lag = 0, interpolation = splinefun)

Arguments

map          object of class: data.frame, no manual documentation  
lag           no manual documentation  
interpolation  no manual documentation

Description

The interval will be set to [-Inf,Inf]

Usage

## S4 method for signature 'function,missing,missing,missing,missing'
TimeMap(map, lag = 0)

Arguments

map          object of class: function, no manual documentation  
lag           no manual documentation
TimeMap, function, numeric, numeric, missing, missing-method

manual constructor for a function and an interval

Description

manual constructor for a function and an interval

Usage

## S4 method for signature `function`, numeric, numeric, missing, missing'
TimeMap(map, starttime, endtime, lag = 0)

Arguments

map object of class: function, no manual documentation
starttime object of class: numeric, no manual documentation
endtime object of class: numeric, no manual documentation
lag no manual documentation

TimeMap, list, missing, missing, missing, missing-method

automatic title

Description

automatic title

Usage

## S4 method for signature `list, missing, missing, missing, missing'
TimeMap(map, lag = 0, interpolation = splinefun)

Arguments

map object of class: list, A nested list of the form list(times=l1, data=l2) where l1 is a vector or list of the time values and l2 is a list of numbers, vectors, matrices or arrays.
lag Time delay for the created function of time
interpolation The function used to compute the interpolation e.g splinefun
Interprets the received list as value table of a time dependent function
Description

automatic title

Usage

```r
## S4 method for signature 'missing,missing,missing,numeric,array'
TimeMap(times, data, lag = 0, interpolation = splinefun)
```

Arguments

times: object of class:numeric, no manual documentation
data: object of class:list, no manual documentation
lag: no manual documentation
interpolation: no manual documentation

Description

automatic title

Usage

```r
## S4 method for signature 'missing,missing,missing,numeric,list'
TimeMap(times, data, lag = 0, interpolation = splinefun)
```

Arguments

times: object of class:numeric, no manual documentation
data: object of class:list, no manual documentation
lag: no manual documentation
interpolation: no manual documentation
TimeMap

## S4 method for signature 'missing,missing,missing,numeric,matrix'
TimeMap(times, data, lag = 0, interpolation = splinefun)

### Arguments
- **times**: object of class:numeric, no manual documentation
- **data**: object of class:matrix, no manual documentation
- **lag**: no manual documentation
- **interpolation**: no manual documentation

Interpolates the data as function of times and remembers the limits of the time domain.

### Description
automatic title

### Usage
## S4 method for signature 'missing,missing,missing,numeric,numeric'
TimeMap(times, data, lag = 0, interpolation = splinefun)

### Arguments
- **times**: object of class:numeric, no manual documentation
- **data**: object of class:numeric, no manual documentation
- **lag**: no manual documentation
- **interpolation**: no manual documentation

Interpolates the data as function of times and remembers the limits of the time domain.
TimeMap-class

Description

automatic title

Usage

## S4 method for signature 'TimeMap,ANY,ANY,ANY,ANY'
TimeMap(map)

Arguments

map object of class: TimeMap, no manual documentation

Description

The class represents functions which are defined on a (possibly infinite) interval from [starttime,endtime]
Instances are usually created internally from data frames or lists provided by the user in the high level interfaces.

Details

The class is necessary to be able to detect unwanted extrapolation of time line data which might otherwise occur for some of the following reasons: SoilR allows to specify measured data for many of its arguments and computes the interpolating functions automatically. The functions returned by the standard R interpolation mechanisms like splinefun or approxfun do not provide a safeguard against accidental extrapolation. Internally SoilR converts nearly all data to time dependent functions e.g. to be used in ode solvers. So the information of the domain of the function has to be kept.

S4-methods

S4-methods with class TimeMap in their signature::

• add_plot, TimeMap-method
• as.character, TimeMap-method
• GeneralDecompOp, TimeMap-method
• getFunctionDefinition, TimeMap-method
• getTimeRange, TimeMap-method
• InFluxes, TimeMap-method
• initialize, TimeMap-method
• plot, TimeMap-method
• TimeMap, TimeMap, ANY, ANY, ANY, ANY-method

automatic title
S4-subclasses
- ScalarTimeMap
- BoundLinDecompOp
- TransportDecompositionOperator
- BoundInFluxes
- BoundFc

TimeMap.from.Dataframe

Description
This function is a deprecated 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 saveguard for Model which thus can check that all involved functions of time are actually defined for the times of interest

TimeMap.new

deprecated constructor of the class TimeMap.

Description
deprecated functions #oooooooooo use the generic TimeMap(...) instead

Usage
TimeMap.new(t_start, t_end, f)
**transitTime**

**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.

---

**TimeRangeIntersection**  
*The time interval where both functions are defined*

**Description**

The time interval where both functions are defined

**Usage**

TimeRangeIntersection(obj1, obj2)

**Arguments**

- **obj1**
  - An object on which getTimeRange can be called

- **obj2**
  - An object on which getTimeRange can be called

---

**transitTime**  
*Transit times for compartment models*

**Description**

Computes the density distribution and mean for the transit time of a constant compartmental model

**Usage**

transitTime(A, u, a = seq(0, 100), q = c(0.05, 0.5, 0.95))

**Arguments**

- **A**
  - A constant compartmental square matrix with cycling rates in the diagonal and transfer rates in the off-diagonal.

- **u**
  - A one-column matrix defining the amount of inputs per compartment.

- **a**
  - A sequence of ages to calculate density functions

- **q**
  - Vector of probabilities to calculate quantiles of the transit time distribution

**Value**

A list with 3 objects: mean transit time, transit time density distribution, and quantiles.

**See Also**

- systemAge
Description

automatic title

S4-methods

S4-methods with class `TransportDecompositionOperator` in their signature::

- `getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method`
- `getDotOut,TransportDecompositionOperator-method`
- `getFunctionDefinition,TransportDecompositionOperator-method`
- `getNumberOfPools,TransportDecompositionOperator-method`
- `getOutputReceivers,TransportDecompositionOperator,numeric-method`
- `getTransferCoefficients,TransportDecompositionOperator-method`
- `getTransferMatrixFunc,TransportDecompositionOperator-method`
- `initialize,TransportDecompositionOperator-method`

S4-methods with superclasses (in the package) of class `TransportDecompositionOperator` in their signature::

superclass `TimeMap`:

- `add_plot,TimeMap-method`
- `as.character,TimeMap-method`
- `GeneralDecompOp,TimeMap-method`
- `getFunctionDefinition,TimeMap-method`
- `getTimeRange,TimeMap-method`
- `Influxes,TimeMap-method`
- `initialize,TimeMap-method`
- `plot,TimeMap-method`
- `TimeMap,TimeMap,ANY,ANY,ANY,ANY-method`

S4-superclasses (in the package)

- `TimeMap`

---

turnoverFit

*Estimation of the turnover time from a radiocarbon sample.*

Description

This function finds two possible values of turnover time from radiocarbon sample assuming a one pool model with carbon at equilibrium.
Usage

```r
turnoverFit(obsC14, obsyr, yr0, Fatm, plot = TRUE, by = 0.5)
```

Arguments

- `obsC14`: a scalar with the observed radiocarbon value in Delta14C
- `obssyr`: a scalar with the year in which the sample was taken.
- `yr0`: The year at which simulations will start.
- `Fatm`: an atmospheric radiocarbon curve as data.frame. First column must be time.
- `plot`: logical. Should the function produce a plot?
- `by`: numeric. The increment of the sequence of years used in the simulations.

Details

This algorithm takes an observed radiocarbon value and runs `OnepModel14`, calculates the squared difference between predictions and observations, and uses `optimize` to find the minimum difference.

Value

A numeric vector with two values of the turnover time that minimize the difference between the prediction of a one pool model and the observed radiocarbon value.

---

TwopFeedbackModel

**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

```r
TwopFeedbackModel(
  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 deSolve.lsoda.wrapper or any other user provided function with the same interface.
passt	if TRUE forces the constructor to create the model even if it is invalid

Value

A Model Object that can be further queried

References


See Also

There are other predefinedModels and also more general functions like Model.

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=5,In=InRand,gam=0.9,xi=(fT.Daycent1(15)*fW.Demeter(15)))
Series=TwoPSeriesModel(t=times,ks=ks,a21=0.2*ks[1],C0=5,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=5,In=InRand,xi=MoistEffect)

CtP=getC(Parallel)
CtS=getC(Series)
CtF=getC(Feedback)
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

```r
TwopFeedbackModel14(
  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 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 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.

See Also
There are other predefinedModels and also more general functions like Model_14.

Examples

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

Ex=TwoopFeedbackModel14(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))
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 (m3 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 a vector of length 2 with the initial values of the SOM pool and the microbial biomass pool (g m-3)

Details

This implementation is similar to the model described in Manzoni and Porporato (2007).

Value

Microbial biomass over time

References


See Also

There are other predefinedModels and also more general functions like Model.
Examples

days = seq(0, 1000, 0.5)
MMmodel = TwoPMModel(t = days, ival = c(100, 10))
Cpools = getC(MMmodel)
matplot(days, Cpools[, 1], 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) wrong look at the sympy test print twoPMModel.pdf
Km = 900
Af = 1
Cs = kb * Km / ((Af + ks * (1 - r) - kb)
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")
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 = TwoPMModel(t = days, ival = c(972, 304), Af = 3, kb = 0.0000001)
Cpools = getC(MMmodel)

matplot(days, Cpools[, 1], 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 linear 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 TwoModel that can handle an arbitrary number of pools.

Usage

TwopParallelModel(
  t,    # required
  ks,    # required
  C0,    # required
  In,    # required
  gam,   # required
  xi = 1, # default = 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 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

References


See Also

There are other predefinedModels and also more general functions like Model.

Examples

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

```
TwopParallelModel14(
  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 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 `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.
Value

A Model Object that can be further queried

See Also

There are other predefinedModels and also more general functions like Model_14.

Examples

```r
lag <- 2
years=seq(1901+lag,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=lag)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14T(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

```r
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 `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

Value

A Model Object that can be further queried

References


See Also

There are other `predefinedModels` and also more general functions like `Model`.

Examples

```r
# Time
start=0
end=10
tn=50
timestep=(end-start)/tn
t=seq(start,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=TwopSeriesModel(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"),
1ty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")```
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**

```r
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 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 `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.
Value

A Model Object that can be further queried

See Also

There are other predefinedModels and also more general functions like Model_14.

Examples

```r
years <- seq(1901, 2009, by = 0.5)
LitterInput <- 700
# Ex = TwoSeriesModel14(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))
```

---

**UnBoundInFluxes**

*automatic title*

**Description**

*automatic title*

**Usage**

`UnBoundInFluxes(map)`

**Arguments**

- `map` see method arguments

**S4-methods**

- `UnBoundInFluxes, function-method`
UnBoundInFluxes, function-method

Description
automatic title

Usage

```r
## S4 method for signature 'function'
UnBoundInFluxes(map)
```

Arguments

map object of class: function, no manual documentation

UnBoundInFluxes-class automatic title

Description
automatic title

UnBoundLinDecompOp Generic constructor for the class with the same name

Description
Generic constructor for the class with the same name

Usage

UnBoundLinDecompOp(matFunc)

Arguments

matFunc see method arguments

S4-methods

- UnBoundLinDecompOp, function-method
UnBoundLinDecompOp\_function-method

*Generic constructor for the class with the same name*

---

**Description**

Generic constructor for the class with the same name

**Usage**

```r
## S4 method for signature 'function'
UnBoundLinDecompOp(matFunc)
```

**Arguments**

- `matFunc` object of class: function, no manual documentation

**See Also**

Other UnBoundLinDecompOp\_constructor: `getFunctionDefinition`, `UnBoundLinDecompOp-method`

---

UnBoundLinDecompOp\_class

*An S4 class to represent a linear nonautonomous compartmental matrix*

---

**Description**

An S4 class to represent a linear nonautonomous compartmental matrix

---

UnBoundNonLinDecompOp

*Generic constructor for the class with the same name*

---

**Description**

Generic constructor for the class with the same name

**Usage**

```r
UnBoundNonLinDecompOp(
  matFunc,
  internal\_fluxes,
  out\_fluxes,
  numberOfPools,
  state\_variable\_names,
  timeSymbol,
  operator
)
```
Arguments

- **matFunc**  
  see method arguments
- **internal_fluxes**  
  see method arguments
- **out_fluxes**  
  see method arguments
- **numberOfPools**  
  see method arguments
- **state_variable_names**  
  see method arguments
- **timeSymbol**  
  see method arguments
- **operator**  
  see method arguments

S4-methods

- `UnBoundNonLinDecompOp(function,missing,missing,missing,ANY,ANY,ANY-method)
- `UnBoundNonLinDecompOp,missing,missing,missing,missing,character,character,UnBoundNonLinDecompOp_by_PoolNames-method`
- `UnBoundNonLinDecompOp,missing,vector,vector,numeric,ANY,ANY,ANY-method`

UnBoundNonLinDecompOp(function,missing,missing,missing,ANY,ANY,ANY-method)

*Constructor for the class with the same name*

Description

Constructor for the class with the same name

Usage

```
## S4 method for signature `.Function`,missing,missing,missing,ANY,ANY,ANY`
UnBoundNonLinDecompOp(matFunc)
```

Arguments

- **matFunc**  
  object of class: function, A matrix valued function of the state vector and time

See Also

Other UnBoundNonLinDecompOp_constructor: `UnBoundNonLinDecompOp,missing,vector,vector,numeric,ANY,ANY-method`
convert to Indexed version

Description
convert to Indexed version

Usage
## S4 method for signature
## 'missing,'missing,'missing,'missing,'missing,'character,'character,'UnBoundNonLinDecompOp_by_PoolNames'
UnBoundNonLinDecompOp(state_variable_names, timeSymbol, operator)

Arguments
state_variable_names
object of class: character, no manual documentation

timeSymbol
object of class: character, no manual documentation

operator
object of class: UnBoundNonLinDecompOp_by_PoolNames, no manual documentation

Description
Constructor for the class with the same name

Usage
## S4 method for signature
## 'missing,vector,vector,numeric,ANY,ANY,ANY'
UnBoundNonLinDecompOp(internal_fluxes, out_fluxes, numberOfPools)

Arguments
internal_fluxes
object of class: vector, vector of elements of type InternalFlux_by_PoolName

out_fluxes
object of class: vector, vector of elements of type OutFlux_by_PoolName

numberOfPools
object of class: numeric, no manual documentation

See Also
Other UnBoundNonLinDecompOp constructor: UnBoundNonLinDecompOp, function,
An S4 class to represent a nonlinear nonautonomous compartmental matrix

Generic constructor for the class with the same name

A flux and pool name based representation of a possibly nonlinear and nonautonomous Compartmental Matrix

## S4 method for signature

```r
UnBoundNonLinDecompOp_by_PoolNames(internal_fluxes, out_fluxes, timeSymbol)
```

Arguments

- `internal_fluxes`: see method arguments
- `out_fluxes`: see method arguments
- `timeSymbol`: see method arguments

Usage

```r
# S4 method for signature
# 'InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character'
UnBoundNonLinDecompOp_by_PoolNames(internal_fluxes, out_fluxes, timeSymbol)
```
UnBoundNonLinDecompOp_by_PoolNames-class

Arguments

  internal_fluxes  
  object of class: InternalFluxList_by_PoolName, InternalFluxList_by_PoolName

  out_fluxes  
  object of class: OutFluxList_by_PoolName, OutFluxList_by_PoolName

  timeSymbol  
  object of class: character.

UnBoundNonLinDecompOp_by_PoolNames-class

An S4 class to represent the of nonlinear nonautonomous compartmental system independently of the order of state variables

Description

An S4 class to represent the of nonlinear nonautonomous compartmental system independently of the order of state variables

S4-methods

S4-methods with class UnBoundNonLinDecompOp_by_PoolNames in their signature:

  • getCompartmentalMatrixFunc, UnBoundNonLinDecompOp_by_PoolNames, character, character-method
  • Model_by_PoolNames, missing, numeric, UnBoundNonLinDecompOp_by_PoolNames, numeric, InFluxList_by_PoolName-method
  • UnBoundNonLinDecompOp, missing, missing, missing, missing, missing, character, character, UnBoundNonLinDecompOp_by_PoolNames-method

WangThreePoolNonAutonomous_sym

A non-autonomous version of the original Wang 3 pool model

Description

An Example based on the original non-linear autonomous model as described in Wang et al. (2014) with state_variables:

  1. C_l desc: litter carbon unit: "g C m^-2"
  2. C_s desc: soil organic matter unit: "g C m^-2"
  3. C_b: desc: microbial biomass unit: "g C m^-2"

Note that this is not a complete model run like most of the models in SoilR but a description of the fluxes that can be extended to a model run if initial values and times are specified. The default values are completely arbitrary. So is one time dependency that has been added to demonstrate that this is possible everywhere and every part of the model can become non-autonomous. At the moment the variable t is mostly ignored like in the original Wang Model except for the first influx to pool C_l.
Usage

WangThreePoolNonAutonomous_sym(
  alpha = 0.5,
  epsilon = 0.4,
  mu_b = 0.2,
  F_NPP = 3e+06,
  V_l = 0.5,
  V_s = 0.5,
  K_l = 1e+05,
  K_s = 100
)

Arguments

alpha   fraction of carbon influx that directly enters the soil organic matter pool
epsilon microbial growth efficiency
mu_b   turnover rate of microbial biomass per year, unit: "year^-1"
F_NPP   carbon influx into soil, unit: "g C*m^-2*year^-1"
V_l   maximum rate of litter carbon assimilation per unit microbial biomass per year
V_s   maximum rate of soil carbon assimilation per unit microbial biomass per year
K_l   half-saturation constant for litter carbon assimilation by microbial biomass
K_s   half-saturation constant for soil carbon assimilation by microbial biomass

Examples

# This is a working example which demonstrates some of the new functionality.
require('SoilR', quietly = TRUE)
smod <- WangThreePoolNonAutonomous_sym()
# (look at the source code of WangThreePoolNonAutonomous_sym )
plotPoolGraph(smod)
state_variable_names(smod)
# define initial values for the state variables
iv <- c(C_l=1000, C_b=5000, C_s=1000)
times <- seq(from=1, to=1000, by=10)
modrun <- Model_by_PoolNames( smod=smod, times=times, initialValues=iv)
sol <- getSolution(modrun)
# Let's see what we have computed
colnames(sol)
# shortcut overview plot for all phase plane projections and time lines
# of the pool contents
plot(data.frame(times=times, sol[,c('C_l', 'C_s', 'C_b')]))
# plot fluxes as functions of time
in_fluxes <- sol[,grep('influxes', colnames(sol))]
plot( times, sol[,c('influxes.C_l')] , type='l',
      ylim=c(min(in_fluxes), max(in_fluxes)) )
lines( times, sol[,c('influxes.C_l')] , type='l',
       ylim=c(min(in_fluxes), max(in_fluxes)) )
internal_fluxes <- sol[,grep('internal_fluxes', colnames(sol))]
plot( times, sol[,c('internal_fluxes.C_l->C_b')] , type='l' )
Yasso07Model

Implementation of the Yasso07 model

Description

This function creates a model for five pools as described in Tuomi et al. (2009)

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 = 0.035, p8 = 0.005, p9 = 0.01, p10 = 5e-04, p11 = 0.03, 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 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

Value

A Model Object that can be further queried

References

See Also

There are other predefinedModels and also more general functions like Model.

Examples

```r
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

```r
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_cwl_ext = 0.251 * 0.3, u_nwl_cwl_cel = 0.251 * 0.3, u_nwl_cwl_lig = 0.251 * 0.3, 0, 0),
  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 7 containing the values of the exposure and decomposition rates for each pool.
- `p` A vector of containing transfer coefficients among different pools.
A vector containing the initial amount of carbon for the 7 pools. The length of this vector must be 7.

In

A vector of constant litter inputs.

\( x_i \)

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 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

Value

A Model Object that can be further queried

References


See Also

There are other predefinedModels and also more general functions like Model.

Examples

```r
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")
```

Description

The method provides shortcuts and a unified interface to some of the methods that can be applied to a model. For a given model `M` the code `M['C']` is equivalent to `getC(M)` and `M['ReleaseFlux']` is equivalent to `getReleaseFlux(M)` `M['AccumulatedRelease']` is equivalent to `getAccumulatedRelease(M)`

Usage

```r
## S4 method for signature 'Model,character,missing,missing'
x[i]
```
Arguments

\[\text{x}\] object of class: \text{Model}, no manual documentation

\[\text{i}\] object of class: \text{character}, no manual documentation

Description

Automatic title

Usage

\[
\text{## S4 method for signature 'NlModel,character,ANY,ANY'}
\]

\[\text{x[i]}\]

Arguments

\[\text{x}\] object of class: \text{NlModel}, no manual documentation

\[\text{i}\] object of class: \text{character}, no manual documentation

Description

Automatic title

Usage

\[
\text{## S4 method for signature 'MCSim'}
\]

\[\text{x[i]}\]

Arguments

\[\text{x}\] object of class: \text{MCSim}, no manual documentation

\[\text{i}\] no manual documentation
[[<-,MCSim-method

Description

automatic title

Usage

## S4 replacement method for signature 'MCSim'
x[[i, j, ...]] <- value

Arguments

x object of class: MCSim, no manual documentation
i no manual documentation
j no manual documentation
... no manual documentation
value no manual documentation

$,NlModel-method

Description

automatic title

Usage

## S4 method for signature 'NlModel'
x$name

Arguments

x object of class: NlModel, no manual documentation
name no manual documentation
Index

* `UnBoundLinDecompOp_constructor`  
  getFunctionDefinition, UnBoundLinDecompOp-method, 134  
  UnBoundLinDecompOp, function-method, 277

* `UnBoundNonLinDecompOp_constructor`  
  UnBoundNonLinDecompOp, function, missing, missing, missing, ANY, ANY, ANY-method, 279

* `datasets`  
  C14Atm, 42  
  C14Atm_NH, 43  
  eCO2, 79  
  Graven2017, 159  
  HarvardForest14CO2, 160  
  Hua2013, 160  
  incubation_experiment, 164  
  IntCal09, 181  
  IntCal13, 182  
  IntCal20, 183  
  SHCal20, 235  

[, `Model`, character, missing, missing-method, 285  

[, `NlModel`, character, ANY, ANY-method, 286  

[[-, `MCSim-method`, 286  

$, `MCSim-method`, 287  

AbsoluteFractionModern, 13  
AbsoluteFractionModern, BoundFc-method, 13  
AbsoluteFractionModern, ConstFc-method, 14  
AbsoluteFractionModern_from_Delta14C, 14  
AbsoluteFractionModern_from_Delta14C, matrix-method, 15  
AbsoluteFractionModern_from_Delta14C, numeric-method, 15  
add_plot, 15  
add_plot, TimeMap-method, 16  
as.character, TimeMap-method, 16  

as.numeric, InFluxList_by_PoolName-method, 17  
as.numeric, InternalFlux_by_PoolName-method, 18  
as.numeric, InternalFluxList_by_PoolName-method, 17  
as.numeric, InternalFluxList_by_PoolName-method, 18  
availableParticleProperties, MCSim-method, 19  
availableParticleSets, 20  
availableParticleSets, MCSim-method, 20  
availableResidentSets, 21  
availableResidentSets, MCSim-method, 21  
AWBmodel, 22  

bacwaveModel, 24, 225  
bind.C14curves, 25  
BoundFc, 26, 85, 231, 261  
BoundFc, character-method, 26  
BoundFc, missing-method, 27  
BoundFc-class, 27  
BoundInFluxes, 27, 169, 261  
BoundInFluxes-class, 28  
BoundLinDecompOp, 28, 75, 261  
BoundLinDecompOp, ANY-method, 29  
BoundLinDecompOp, UnBoundLinDecompOp-method, 29  

BoundLinDecompOp-class, 30  
by_PoolIndex, 30  
by_PoolIndex, ConstantInFluxRate_by_PoolName, ANY, ANY-method, 31  
by_PoolIndex, ConstantInternalFluxRate_by_PoolName, ANY, ANY, ANY-method, 32  
by_PoolIndex, ConstantInternalFluxRateList_by_PoolName, ANY, ANY-method, 32  
by_PoolIndex, ConstantOutFluxRate_by_PoolName, ANY, ANY-method, 33  
by_PoolIndex, ConstantOutFluxRateList_by_PoolName, ANY, ANY-method, 33  
by_PoolIndex, function, character, character-method, 34
INDEX

by_PoolIndex, InFlux_by_PoolName, character, character-method
by_PoolIndex, InFluxList_by_PoolName, character, character-method
by_PoolIndex, InternalFlux_by_PoolName, character, character-method
by_PoolIndex, InternalFluxList_by_PoolName, character, character-method
by_PoolIndex, OutFlux_by_PoolName, character, character-method
by_PoolIndex, OutFluxList_by_PoolName, character, character-method
by_PoolIndex, PoolConnection_by_PoolName, ANY, ANY-method
by_PoolName
by_PoolName, ConstantInFlux_by_PoolIndex-method
by_PoolName, ConstantInFluxList_by_PoolIndex-method
by_PoolName, ConstantInFluxRate_by_PoolIndex-method
by_PoolName, ConstantInternalFluxRate_by_PoolIndex-method
by_PoolName, ConstantInternalFluxRateList_by_PoolIndex-method
by_PoolName, ConstantOutFluxRate_by_PoolIndex-method
by_PoolName, ConstantOutFluxRateList_by_PoolIndex-method
C14Atm
C14Atm_NH
CenturyModel
CenturyModel14
check_duplicate_pool_names
check_id_length
check_pool_ids
check_pool_ids; PoolConnection_by_PoolIndex, integer-method
computeResults
computeResults, MCSim-method
ConstantInFlux_by_PoolIndex
ConstantInFlux_by_PoolName
ConstantInFluxList_by_PoolIndex
ConstantInFluxList_by_PoolName
ConstantOutFluxRate_by_PoolIndex
ConstantOutFluxRate_by_PoolName
ConstantOutFluxRateList_by_PoolIndex
ConstantOutFluxRateList_by_PoolName
getInitialValues,NlModel-method, 136
getLinearScaleFactor, 137
getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method, 137
getMeanTransitTime, 138
getMeanTransitTime,ConstLinDecompOp-method, 138
getNumberOfPools, 139
getNumberOfPools,MCSim-method, 139
getNumberOfPools,NlModel-method, 140
getNumberOfPools,TransportDecompositionOperator-method, 140
getOutputFluxes, 140
getOutputFluxes,NlModel-method, 141
getOutputReceivers, 141
getOutputReceivers,TransportDecompositionOperator,numeric-method, 142
getParticleMonteCarloSimulator, 142
getParticleMonteCarloSimulator,NlModel-method, 143
getReleaseFlux, 143, 191, 215
getReleaseFlux,Model-method, 144
getReleaseFlux,Model_by_PoolNames-method, 144
getReleaseFlux,NlModel-method, 145
getReleaseFlux14, 145
getReleaseFlux14,Model_14-method, 146
getRightHandSideOfODE, 146
getRightHandSideOfODE,Model-method, 147
getRightHandSideOfODE,Model_by_PoolNames-method, 147
getSolution, 148
getSolution,Model_by_PoolNames-method, 148
timeRange, 149
timeRange,ConstInFluxes-method, 150
timeRange,ConstLinDecompOp-method, 150
timeRange,ConstLinDecompOpWithLinearScalarFactor-method, 150
timeRange,DecompositionOperator-method, 151
timeRange,TimeMap-method, 151
timeRange,UnBoundInFluxes-method, 151
timeRange,UnBoundLinDecompOp-method, 152
times, 152
times,Model-method, 153
times,Model_by_PoolNames-method, 153
getTimes, 154
getTransferCoefficients, 154
getTransferCoefficients,ConstLinDecompOpNlModel-method, 155
getTransferCoefficients,TransportDecompositionOperator-method, 155
getTransferMatrix, 155
getTransferMatrixFunc,TransportDecompositionOperator-method, 156
getTransitTimeDistributionDensity, 157
getTransitTimeDistributionDensity,ConstLinDecompOp-method, 157
getValues, 158
generate,ConstFc-method, 158
HarvardForest14CO2, 160
ICBM_N, 163
ICBMModel, 162, 225
incubation_experiment, 164
InFlux, 165
InFlux_by_PoolIndex, 172
InFlux_by_PoolIndex,Function,n numeric-method, 173
InFlux_by_PoolIndex-class, 173
InFlux_by_PoolName, 173
InFlux_by_PoolName,Function,character-method, 174
InFlux_by_PoolName-class, 174
InFluxes, 28, 68, 106, 166, 166, 193, 195, 237
InFluxes,ConstantInFluxList_by_PoolIndex-method, 166
InFluxes,Function-method, 167
InFluxes,InFluxes-method, 167
InFluxes,numeric-method, 168
InFluxes,StateIndependentInFluxList_by_PoolIndex-method, 168
InFluxes,TimeMap-method, 169
InFluxes-class, 169
InFluxList_by_PoolIndex, 170
InFluxList_by_PoolIndex,list-method, 170
InFluxList_by_PoolIndex-class, 170
InFluxList_by_PoolName, 171, 201
InFluxList_by_PoolName, list-method, 171
InFluxList_by_PoolName-class, 172
initialize,ConstLinDecompOp-method, 175
initialize, DecompositionOperator-method, 175
initialize, MCSim-method, 176
initialize, Model-method, 176
initialize, Model_14-method, 177
initialize, Model_by_PoolNames-method, 178
initialize, NlModel-method, 178
initialize, TimeMap-method, 179
initialize, TransportDecompositionOperator-method, 180
initialize, UnBoundInFluxes-method, 180
initialize, UnBoundLinDecompOp-method, 181
IntCal09, 25, 181
IntCal13, 25, 182, 183
IntCal20, 183
InternalFlux_by_PoolIndex, 186
InternalFlux_by_PoolIndex, function, numeric, numeric, missing-method, 187
InternalFlux_by_PoolIndex-class, 187
InternalFlux_by_PoolName, 185, 187, 188
InternalFlux_by_PoolName, function, character, character, missing-method, 188
InternalFlux_by_PoolName, function, missing, missing, character-method, 189
InternalFlux_by_PoolName-class, 189
InternalFluxList_by_PoolIndex, 184
InternalFluxList_by_PoolIndex, list-method, 184
InternalFluxList_by_PoolIndex-class, 185
InternalFluxList_by_PoolName, 185, 281
InternalFluxList_by_PoolName, list-method, 185
InternalFluxList_by_PoolName-class, 186
linearScalarModel, 189
linesCPool, 191
list, 161
listProduct, 191
MCSim-class, 192
Model, 12, 25, 44, 46, 101, 104, 110–112, 124, 125, 135, 144, 145, 148, 149, 153, 162, 190, 192, 193, 199, 204, 228, 232, 234, 241, 243, 249, 252, 265, 268, 270, 273, 284, 285
Model-class, 194
Model_14, 12, 13, 105, 110–112, 124, 125, 135, 144, 145, 148, 149, 153, 195,
PoolConnection_by_PoolIndex-class, 218
PoolConnection_by_PoolName, 219
PoolConnection_by_PoolName,ANY,ANY,missing-method, 219
PoolConnection_by_PoolName-class, 219
PoolId, 222, 225
PoolId-class, 220
PoolIndex, 220, 225
PoolIndex,character-method, 221
PoolIndex,numeric-method, 221
PoolIndex,PoolIndex-method, 221
PoolIndex,PoolName-method, 222
PoolIndex-class, 222
PoolName, 220, 223
PoolName,character-method, 223
PoolName,PoolIndex-method, 224
PoolName,PoolName-method, 224
PoolName-class, 225
print,NlModel-method, 226
RespirationCoefficients, 226
RothCModel, 44, 46, 190, 225, 227
ScalarTimeMap, 27, 44, 46, 190, 228, 261
ScalarTimeMap,data.frame,missing,missing,missing,missing-method, 229
ScalarTimeMap,function,missing,missing,missing,missing-method, 230
ScalarTimeMap,function,numerical,missing,missing,missing,missing-method, 231
ScalarTimeMap,missing,missing,missing,missing,missing,missing-method, 232
ScalarTimeMap,missing,missing,missing,missing,missing,missing-method, 233
ScalarTimeMap,missing,missing,missing,missing,missing,missing-method, 234
ScalarTimeMap-class, 228, 231
SeriesLinearModel, 225, 232
SeriesLinearModel14, 225, 233
SHCa120, 235
show,NlModel-method, 236
SoilR-package, 12
SoilR.F0.new, 236
state_variable_names, 239
StateDependentInFluxVector, 169
StateDependentInFluxVector
(StateDependentInFluxVector-class), 237
StateDependentInFluxVector-class, 237
StateIndependentInFlux_by_PoolIndex,
StateIndependentInFlux_by_PoolIndex-class, 238
StateIndependentInFluxList_by_PoolIndex,
StateIndependentInFluxList_by_PoolIndex-list-method, 239
StateIndependentInFluxList_by_PoolIndex-class, 239
SystemAge, 75, 240, 262
ThreepairMMmodel, 225, 241
ThreepFeedbackModel, 225, 242
ThreepFeedbackModel14, 225, 244
ThreepParallelModel, 211, 225, 248
ThreepParallelModel14, 225, 249
ThreepSeriesModel, 225, 251
ThreepSeriesModel14, 225, 253
TimeMap, 27, 28, 30, 232, 255, 263
TimeMap,data.frame,missing,missing,missing,missing-method, 256
TimeMap,function,missing,missing,missing,missing-method, 257
TimeMap,function,numerical,missing,missing,missing,missing-method, 258
TimeMap,missing,missing,missing,missing,missing,missing-method, 259
TimeMap,missing,missing,missing,missing,missing,missing-method, 260
TimeMap,missing,missing,missing,missing,missing,missing-method, 261
TimeMap,TimeMap,ANY,ANY,ANY,ANY-method, 262
transitTime, 240, 262
<table>
<thead>
<tr>
<th>Class/Function Name</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransportDecompositionOperator</td>
<td>261</td>
</tr>
<tr>
<td>TransportDecompositionOperator-class</td>
<td>263</td>
</tr>
<tr>
<td>turnoverFit</td>
<td>263</td>
</tr>
<tr>
<td>TwopFeedbackModel</td>
<td>105, 106, 196, 225, 264</td>
</tr>
<tr>
<td>TwopFeedbackModel14</td>
<td>225, 266</td>
</tr>
<tr>
<td>TwopMMmodel</td>
<td>225, 268</td>
</tr>
<tr>
<td>TwopParallelModel</td>
<td>105, 106, 196, 211, 225, 269</td>
</tr>
<tr>
<td>TwopParallelModel14</td>
<td>225, 271</td>
</tr>
<tr>
<td>TwopSeriesModel</td>
<td>105, 106, 196, 225, 272</td>
</tr>
<tr>
<td>TwopSeriesModel14</td>
<td>225, 274</td>
</tr>
<tr>
<td>UnBoundInFluxes</td>
<td>169, 275</td>
</tr>
<tr>
<td>UnBoundInFluxes-function-method</td>
<td>276</td>
</tr>
<tr>
<td>UnBoundInFluxes-class</td>
<td>276</td>
</tr>
<tr>
<td>UnBoundLinDecompOp</td>
<td>75, 276</td>
</tr>
<tr>
<td>UnBoundLinDecompOp-function-method</td>
<td>277</td>
</tr>
<tr>
<td>UnBoundLinDecompOp-class</td>
<td>277</td>
</tr>
<tr>
<td>UnBoundNonLinDecompOp-function,missing,missing,missing,missing,missing,ANY,ANY,ANY-method</td>
<td>278</td>
</tr>
<tr>
<td>UnBoundNonLinDecompOp,missing,missing,missing,missing,missing,missing,character,character,UnBoundNonLinDecompOp_by_PoolNames-method</td>
<td>279</td>
</tr>
<tr>
<td>UnBoundNonLinDecompOp-class,missing,vector,vector,numeric,ANY,ANY,ANY-method</td>
<td>279</td>
</tr>
<tr>
<td>UnBoundNonLinDecompOp_by_PoolNames</td>
<td>201, 280</td>
</tr>
<tr>
<td>UnBoundNonLinDecompOp_by_PoolNames,InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character-method</td>
<td>280</td>
</tr>
<tr>
<td>UnBoundNonLinDecompOp_by_PoolNames-class</td>
<td>281</td>
</tr>
<tr>
<td>WangThreePoolNonAutonomous_sym</td>
<td>13, 225, 281</td>
</tr>
<tr>
<td>Yasso07Model</td>
<td>225, 283</td>
</tr>
<tr>
<td>YassoModel</td>
<td>225, 284</td>
</tr>
</tbody>
</table>