Package ‘SoilR’

November 12, 2019

Title Models of Soil Organic Matter Decomposition

Version 1.1-93

Date 2017-05-04

Author Carlos A. Sierra, Markus Mueller

Maintainer Markus Mueller <mamueller@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

Suggests FME,lattice,MASS,knitr,rmarkdown

LazyData TRUE

VignetteBuilder knitr

Collate setGlobalVariables.R setOldClasses.R genericFunctions.R
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
StateDependentInFluxVector.R OutFluxList_by_PoolIndex.R
OutFluxList_by_PoolName.R InternalFluxList_by_PoolIndex.R
InternalFluxList_by_PoolName.R

ConstantInFluxList_by_PoolIndex.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
SoilR.F0__deprecated_in_1.2.R MCSim.R DecompOp.R
UnboundInflux.R UnBoundLinDecompOp.R BoundLinDecompOp.R
UnBoundNonLinDecompOp.R
DecompositionOperator_deprecated_in_1.2.R Model_by_PoolNames.R
Model.R Model_14.R NlModel.R AutonomousLinearModel.R
NonAutonomousLinearModel.R AWBmodel.R bacwaveModel.R
bind.C14curves.R BoundFc.R CenturyModel.R CenturyModel_new.R
ConstFc.R ConstLinDecompOp.R
ConstLinDecompOpWithLinearScaleFactor.R
UnBoundNonLinDecompOp_by_PoolNames.R cycling.R euler.R
example.2DBoundInFluxesFromFunction.R
example.2DBoundLinDecompOpFromFunction.R
example.2DConstFC.Args.R example.2DConstInFluxesFromVector.R
example.2DGeneralDecompOpArgs.R example.2DInFluxes.Args.R
example.2DUnBoundLinDecompOp.R
example.BoundLinDecompOpFromFunction.R
example.ConstlinDecompOpFromMatrix.R
example.nestedTime2DMatrixList.R
example.nestedTime3DArrayList.R example.Time2DArrayList.R
example.Time3DArrayList.R example.TimeMapFromArray.R
FcAtm_deprecated_in_1.2.R fT.Arrhenius.R fT.Century1.R
fT.Century2.R fT.Daycent1.R fT.Daycent2.R fT.Demeter.R fT.KB.R
fT.LandT.R fT.linear.R fT.Q10.R fT.RothC.R fT.Standcarb.R
function.R fW.Candy.R fW.Century.R fW.Daycent1.R fW.Daycent2.R
fW.Demeter.R fW.Gompertz.R fW.Moyano.R fW.RothC.R fW.Skopp.R
fW.Standcarb.R GeneralModel.R GaudinskiModel14.R
GeneralModel_14.R GeneralNlModel.R ICBMModel.R linesCPool.R
linMaker.R list.R listProduct.R makelink.R MeanAge.R MeanTT.R
NpYdot.R OnepModel.R OnepModel14.R ParallelModel.R
plotC14Pool.R plotCPool.R RothCModel.R SeriesLinearModel.R
SeriesLinearModel14.R solver.R spectralNorm.R systemAge.R
ThreepairMMmodel.R ThreepFeedbackModel.R
ThreepFeedbackModel14.R ThreepParallelModel.R
ThreepParallelModel14.R ThreepSeriesModel.R
ThreepSeriesModel14.R transitTime.R tupelList.R turnoverFit.R
makeListInstance.R pe.R pp.R plotPoolGraphFromTupleLists.R
checkTargetClassOfElements.R TimeRangeIntersection.R
TwopFeedbackModel.R TwopFeedbackModel14.R TwopMMmodel.R
TwopParallelModel.R TwopParallelModel14.R TwopSeriesModel.R
TwopSeriesModel14.R vecFuncMaker.R warnings.R Yasso07Model.R
YassoModel.R C14Atm_NH.R C14Atm.R Hua2013.R

IntCal09.R IntCal13.R incubation_experiment.R
HarvardForest14CO2.R getTransferMatrix.R

RoxygenNote 6.1.99.9001

Encoding UTF-8

R topics documented:

AbsoluteFractionModern . . . 12
AbsoluteFractionModern,BoundFc-method . . . 13
AbsoluteFractionModern,ConstFc-method . . . 13
AbsoluteFractionModern_from_Delta14C . . . 14
AbsoluteFractionModern_from_Delta14C,matrix-method . . . 14
AbsoluteFractionModern_from_Delta14C,numeric-method . . . 15
add_plot . . . 15
add_plot,TimeMap-method . . . 16
as.character,TimeMap-method . . . 16
availableParticleProperties . . . 17
availableParticleProperties,MCSim-method . . . 17
availableParticleSets . . . 18
availableParticleSets,MCSim-method . . . 18
availableResidentSets . . . 19
availableResidentSets,MCSim-method . . . 19
AWBmodel . . . 20
bacwaveModel . . . 22
bind.C14curves . . . 23
BoundFc . . . 24
BoundFc,character-method . . . 25
BoundFc,missing-method . . . 25
BoundFc-class . . . 26
BoundInFluxes . . . 26
BoundInFluxes-class . . . 26
BoundLinDecompOp . . . 26
BoundLinDecompOp,ANY-method . . . 27
BoundLinDecompOp,UnBoundLinDecompOp-method . . . 27
BoundLinDecompOp-class . . . 28
by_PoolIndex . . . 28
by_PoolIndex,ConstantInFluxRate_by_PoolName,ANY,ANY-method . . . 29
by_PoolIndex,ConstantInternalFluxRateList_by_PoolName,ANY,ANY-method . . . 29
by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method . . . 30
by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-method . . . 30
by_PoolIndex,ConstantOutFluxRate_by_PoolName,ANY,ANY-method . . . 31
by_PoolIndex,function,character,character-method . . . 31
by_PoolIndex,InFluxList_by_PoolName,character,character-method . . . 32
by_PoolIndex,InFlux_by_PoolName,character,character-method . . . 33
by_PoolIndex,InternalFluxList_by_PoolName,character,character-method . . . 33
by_PoolIndex,InternalFlux_by_PoolName,character,character-method . . . 34
by_PoolIndex,OutFluxList_by_PoolName,character,character-method . . . 34

by_PoolIndex,OutFlux_by_PoolName,character,character-method 35
by_PoolIndex,PoolConnection_by_PoolName,ANY,ANY-method 36
by_PoolName 36
by_PoolName,ConstantInFluxRate_by_PoolIndex-method 37
by_PoolName,ConstantInternalFluxRateList_by_PoolIndex-method 37
by_PoolName,ConstantInternalFluxRate_by_PoolIndex-method 38
by_PoolName,ConstantOutFluxRateList_by_PoolIndex-method 38
by_PoolName,ConstantOutFluxRate_by_PoolIndex-method 39
C14Atm 39
C14Atm_NH 40
CenturyModel 40
CenturyModel_new 42
check_duplicate_pool_names 44
check_id_length 44
check_pool_ids 45
check_pool_ids,PoolConnection_by_PoolIndex,integer-method 45
computeResults 46
computeResults,MCSim-method 46
computeResults,NlModel-method 47
ConstantInFluxList_by_PoolIndex 47
ConstantInFluxList_by_PoolIndex,ConstInFluxes-method 48
ConstantInFluxList_by_PoolIndex,list-method 48
ConstantInFluxList_by_PoolIndex,numeric-method 49
ConstantInFluxList_by_PoolIndex-class 49
ConstantInFluxList_by_PoolName 49
ConstantInFluxRate_by_PoolIndex-class 50
ConstantInFluxRate_by_PoolName-class 50
ConstantInFlux_by_PoolIndex-class 50
ConstantInternalFluxRateList_by_PoolIndex 50
ConstantInternalFluxRateList_by_PoolIndex,list-method 51
ConstantInternalFluxRateList_by_PoolIndex,numeric-method 51
ConstantInternalFluxRateList_by_PoolIndex-class 52
ConstantInternalFluxRateList_by_PoolName 52
ConstantInternalFluxRateList_by_PoolName,list-method 52
ConstantInternalFluxRateList_by_PoolName-class 53
ConstantInternalFluxRate_by_PoolIndex 53
ConstantInternalFluxRate_by_PoolIndex,missing,missing,character,numeric-method . . 54
ConstantInternalFluxRate_by_PoolIndex,numeric,numeric,missing,numeric-method . . 54
ConstantInternalFluxRate_by_PoolIndex-class 55
ConstantInternalFluxRate_by_PoolName 55
ConstantInternalFluxRate_by_PoolName,character,character,missing,numeric-method . 56
ConstantInternalFluxRate_by_PoolName,missing,missing,character,numeric-method . . 56
ConstantInternalFluxRate_by_PoolName-class 57
ConstantOutFluxRateList_by_PoolIndex 57
ConstantOutFluxRateList_by_PoolIndex,list-method 57
ConstantOutFluxRateList_by_PoolIndex,numeric-method 58
ConstantOutFluxRateList_by_PoolIndex-class 58
ConstantOutFluxRateList_by_PoolName 59

ConstantOutFluxRateList_by_PoolName,list-method 59
ConstantOutFluxRateList_by_PoolName,numeric-method 60
ConstantOutFluxRateList_by_PoolName-class 60
ConstantOutFluxRate_by_PoolIndex 61
ConstantOutFluxRate_by_PoolIndex,numeric,numeric-method 61
ConstantOutFluxRate_by_PoolIndex-class 62
ConstantOutFluxRate_by_PoolName-class 62
ConstFc 62
ConstFc-class 63
ConstInFluxes 63
ConstInFluxes,ConstantInFluxList_by_PoolIndex,numeric-method 63
ConstInFluxes,numeric,ANY-method 64
ConstInFluxes-class 64
ConstLinDecompOp 65
ConstLinDecompOp,matrix,missing,missing,missing,missing-method 65
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,ConstantOutFluxRateList_by_PoolIndex,nu
method 66
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,missing,numeric,missing-
method 66
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolName,n
method 67
ConstLinDecompOp,missing,missing,ConstantOutFluxRateList_by_PoolIndex,numeric,missing-
method 67
ConstLinDecompOp-class 68
ConstLinDecompOpWithLinearScalarFactor 68
ConstLinDecompOpWithLinearScalarFactor,matrix,missing,missing,missing,ScalarTimeMap-
method 69
ConstLinDecompOpWithLinearScalarFactor-class 69
ConstLinDecompOp_by_PoolName 69
cycling 70
DecompOp-class 70
DecompositionOperator-class 70
Delta14C 71
Delta14C,BoundFc-method 71
Delta14C,ConstFc-method 72
Delta14C_from_AbsoluteFractionModern 72
Delta14C_from_AbsoluteFractionModern,matrix-method 73
Delta14C_from_AbsoluteFractionModern,numeric-method 73
deSolve.lsoda.wrapper 74
eCO2 74
euler 75
example.2DBoundInFluxesFromFunction 76
example.2DBoundLinDecompOpFromFunction 76
example.2DConstFc.Args 76
example.2DConstInFluxesFromVector 77
example.2DGeneralDecompOpArgs 77
example.2DInFluxes.Args 77
example.2DUnBoundLinDecompOpFromFunction 78

example.ConstlinDecompOpFromMatrix 78
example.nestedTime2DMatrixList 78
example.Time2DArrayList 79
example.Time3DArrayList 79
example.TimeMapFromArray 79
Fc-class 79
FcAtm.from.Dataframe 80
fT.Arrhenius 80
fT.Century1 81
fT.Century2 82
fT.Daycent1 82
fT.Daycent2 83
fT.Demeter 84
fT.KB 84
fT.LandT 85
fT.linear 85
fT.Q10 86
fT.RothC 87
fT.Standcarb 87
fW.Candy 88
fW.Century 89
fW.Daycent1 89
fW.Daycent2 90
fW.Demeter 91
fW.Gompertz 92
fW.Moyano 92
fW.RothC 93
fW.Skopp 93
fW.Standcarb 94
GaudinskiModel14 95
GeneralDecompOp 97
GeneralDecompOp,DecompOp-method 98
GeneralDecompOp,function-method 98
GeneralDecompOp,list-method 99
GeneralDecompOp,matrix-method 99
GeneralDecompOp,TimeMap-method 100
GeneralModel 100
GeneralModel_14 101
GeneralNlModel 102
GeneralPoolId 104
GeneralPoolId,character-method 105
GeneralPoolId,numeric-method 105
getAccumulatedRelease 106
getAccumulatedRelease,Model-method 106
getC 107
getC,Model-method 107
getC,Model_by_PoolNames-method 108
getC,NlModel-method 108

getC14 109
getC14,Model_14-method 109
getCompartmentalMatrixFunc 110
getCompartmentalMatrixFunc,BoundLinDecompOp,ANY,ANY-method 110
getCompartmentalMatrixFunc,ConstLinDecompOp,ANY,ANY-method 111
getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method . . 111
getCompartmentalMatrixFunc,UnBoundNonLinDecompOp,ANY,ANY-method 112
getCompartmentalMatrixFunc,UnBoundNonLinDecompOp_by_PoolNames,character,character-method 112
getConstantCompartmentalMatrix 113
getConstantCompartmentalMatrix,ConstLinDecompOp-method 114
getConstantCompartmentalMatrix,ConstLinDecompOpWithLinearScalarFactor-method 114
getConstantInFluxVector 115
getConstantInFluxVector,ConstInFluxes-method 115
getConstantInternalFluxRateList_by_PoolIndex 116
getConstantInternalFluxRateList_by_PoolIndex,ConstLinDecompOp-method 116
getConstantOutFluxRateList_by_PoolIndex 117
getConstantOutFluxRateList_by_PoolIndex,ConstLinDecompOp-method 117
getConstLinDecompOp 118
getConstLinDecompOp,ConstLinDecompOpWithLinearScalarFactor-method 118
getCumulativeC 119
getCumulativeC,NlModel-method 119
getDecompOp 120
getDecompOp,Model-method 120
getDecompOp,NlModel-method 121
getDotOut 121
getDotOut,TransportDecompositionOperator-method 122
getF14 122
getF14,Model_14-method 123
getF14C 123
getF14C,Model_14-method 124
getF14R 124
getF14R,Model_14-method 125
getFormat 125
getFormat,Fc-method 126
getFunctionDefinition 126
getFunctionDefinition,ConstInFluxes-method 127
getFunctionDefinition,ConstLinDecompOp-method 128
getFunctionDefinition,ConstLinDecompOpWithLinearScalarFactor-method 128
getFunctionDefinition,DecompositionOperator-method 129
getFunctionDefinition,InFluxList_by_PoolIndex-method 129
getFunctionDefinition,InFluxList_by_PoolName-method 130
getFunctionDefinition,StateDependentInFluxVector-method 130
getFunctionDefinition,TimeMap-method 131
getFunctionDefinition,TransportDecompositionOperator-method 131
getFunctionDefinition,UnBoundInFluxes-method 132
getFunctionDefinition,UnBoundLinDecompOp-method 132
getInFluxes 133

getInFluxes,Model-method 133
getInFluxes,NlModel-method 134
getInitialValues 134
getInitialValues,NlModel-method 135
getLinearScaleFactor 135
getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method 136
getMeanTransitTime 136
getMeanTransitTime,ConstLinDecompOp-method 137
getNumberOfPools 137
getNumberOfPools,MCSim-method 138
getNumberOfPools,NlModel-method 138
getNumberOfPools,TransportDecompositionOperator-method 139
getOutputFluxes 139
getOutputFluxes,NlModel-method 140
getOutputReceivers 140
getOutputReceivers,TransportDecompositionOperator,numeric-method 141
getParticleMonteCarloSimulator 141
getParticleMonteCarloSimulator,NlModel-method 142
getReleaseFlux 142
getReleaseFlux,Model-method 143
getReleaseFlux,Model_by_PoolNames-method 143
getReleaseFlux,NlModel-method 144
getReleaseFlux14 144
getReleaseFlux14,Model_14-method 145
getRightHandSideOfODE 145
getRightHandSideOfODE,Model-method 146
getRightHandSideOfODE,Model_by_PoolNames-method 146
getTimeRange 147
getTimeRange,ConstInFluxes-method 147
getTimeRange,ConstLinDecompOp-method 148
getTimeRange,ConstLinDecompOpWithLinearScalarFactor-method 148
getTimeRange,DecompositionOperator-method 149
getTimeRange,TimeMap-method 149
getTimeRange,UnBoundInFluxes-method 150
getTimeRange,UnBoundLinDecompOp-method 150
getTimes 151
getTimes,Model-method 151
getTimes,NlModel-method 152
getTransferCoefficients 152
getTransferCoefficients,NlModel-method 153
getTransferCoefficients,TransportDecompositionOperator-method 153
getTransferMatrix 154
getTransferMatrixFunc 154
getTransferMatrixFunc,TransportDecompositionOperator-method 155
getTransitTimeDistributionDensity 155
getTransitTimeDistributionDensity,ConstLinDecompOp-method 156
getValues 156
getValues,ConstFc-method 157

HarvardForest14CO2 . . . 157
Hua2013 . . . 158
ICBMModel . . . 159
incubation_experiment . . . 161
InFlux . . . 162
InFluxes . . . 163
InFluxes,ConstantInFluxList_by_PoolIndex-method . . . 163
InFluxes,function-method . . . 164
InFluxes,InFluxes-method . . . 164
InFluxes,list-method . . . 165
InFluxes,numeric-method . . . 165
InFluxes,TimeMap-method . . . 166
InFluxes-class . . . 166
InFluxList_by_PoolIndex . . . 166
InFluxList_by_PoolIndex,list-method . . . 167
InFluxList_by_PoolIndex-class . . . 167
InFluxList_by_PoolName . . . 167
InFluxList_by_PoolName,list-method . . . 168
InFluxList_by_PoolName-class . . . 168
InFlux_by_PoolIndex . . . 168
InFlux_by_PoolIndex,function,numeric-method . . . 169
InFlux_by_PoolIndex-class . . . 169
InFlux_by_PoolName . . . 170
InFlux_by_PoolName,function,character-method . . . 170
InFlux_by_PoolName-class . . . 171
initialize,ConstLinDecompOp-method . . . 171
initialize,DecompositionOperator-method . . . 171
initialize,MCSim-method . . . 172
initialize,Model-method . . . 173
initialize,Model_14-method . . . 174
initialize,Model_by_PoolNames-method . . . 175
initialize,NlModel-method . . . 176
initialize,TimeMap-method . . . 177
initialize,TransportDecompositionOperator-method . . . 177
initialize,UnBoundInFluxes-method . . . 178
initialize,UnBoundLinDecompOp-method . . . 179
IntCal09 . . . 179
IntCal13 . . . 180
InternalFluxList_by_PoolIndex . . . 182
InternalFluxList_by_PoolIndex,list-method . . . 182
InternalFluxList_by_PoolIndex-class . . . 182
InternalFluxList_by_PoolName . . . 183
InternalFluxList_by_PoolName,list-method . . . 183
InternalFluxList_by_PoolName-class . . . 184
InternalFlux_by_PoolIndex . . . 184
InternalFlux_by_PoolIndex,function,numeric,numeric,missing-method . . . 184
InternalFlux_by_PoolIndex-class . . . 185
InternalFlux_by_PoolName . . . 185

InternalFlux_by_PoolName,function,character,character,missing-method 186
InternalFlux_by_PoolName,function,missing,missing,character-method 186
InternalFlux_by_PoolName-class 187
linesCPool 187
listProduct 187
MCSim-class 188
Model 188
Model-class 190
Model_14 191
Model_14-class 194
Model_by_PoolNames-class 194
NlModel-class 195
OnepModel 195
OnepModel14 196
OutFlux 198
OutFluxList_by_PoolIndex 198
OutFluxList_by_PoolIndex,list-method 199
OutFluxList_by_PoolIndex-class 199
OutFluxList_by_PoolName 199
OutFluxList_by_PoolName,list-method 200
OutFluxList_by_PoolName-class 200
OutFlux_by_PoolIndex 200
OutFlux_by_PoolIndex,function,numeric-method 201
OutFlux_by_PoolIndex-class 201
OutFlux_by_PoolName 202
OutFlux_by_PoolName,function,character-method 202
OutFlux_by_PoolName-class 203
ParallelModel 203
plot,MCSim-method 204
plot,Model-method 205
plot,NlModel-method 206
plot,TimeMap-method 206
plotC14Pool 207
plotCPool 207
plotPoolGraph 208
plotPoolGraph,Model-method 208
PoolConnection_by_PoolIndex 209
PoolConnection_by_PoolIndex,ANY,ANY,missing-method 209
PoolConnection_by_PoolIndex,missing,missing,character-method 210
PoolConnection_by_PoolIndex-class 210
PoolConnection_by_PoolName 210
PoolConnection_by_PoolName,ANY,ANY,missing-method 211
PoolConnection_by_PoolName-class 211
PoolId-class 212
PoolIndex 212
PoolIndex,character-method 212
PoolIndex,numeric-method 213
PoolIndex,PoolIndex-method 213

PoolIndex,PoolName-method . . . 214
PoolIndex-class . . . 214
PoolName . . . 214
PoolName,character-method . . . 215
PoolName,PoolIndex-method . . . 215
PoolName,PoolName-method . . . 216
PoolName-class . . . 216
predefinedModels . . . 216
print,Model-method . . . 217
print,NlModel-method . . . 218
RespirationCoefficients . . . 218
RothCModel . . . 219
ScalarTimeMap . . . 220
ScalarTimeMap,missing,missing,missing,missing,numeric-method . . . 221
ScalarTimeMap-class . . . 222
SeriesLinearModel . . . 222
SeriesLinearModel14 . . . 223
show,Model-method . . . 225
show,NlModel-method . . . 226
SoilR.F0.new . . . 226
StateDependentInFluxVector-class . . . 227
StateIndependentInFluxList_by_PoolIndex . . . 227
StateIndependentInFluxList_by_PoolIndex,list-method . . . 227
StateIndependentInFluxList_by_PoolIndex-class . . . 228
StateIndependentInFluxList_by_PoolName . . . 228
StateIndependentInFlux_by_PoolIndex-class . . . 228
summary,Model-method . . . 229
systemAge . . . 229
ThreepairMMmodel . . . 230
ThreepFeedbackModel . . . 231
ThreepFeedbackModel14 . . . 234
ThreepParallelModel . . . 238
ThreepParallelModel14 . . . 239
ThreepSeriesModel . . . 241
ThreepSeriesModel14 . . . 243
TimeMap . . . 245
TimeMap,data.frame,missing,missing,missing,missing-method . . . 246
TimeMap,function,missing,missing,missing,missing-method . . . 246
TimeMap,function,numeric,numeric,missing,missing-method . . . 247
TimeMap,list,missing,missing,missing,missing-method . . . 247
TimeMap,missing,missing,missing,numeric,array-method . . . 248
TimeMap,missing,missing,missing,numeric,list-method . . . 248
TimeMap,missing,missing,missing,numeric,matrix-method . . . 249
TimeMap,missing,missing,missing,numeric,numeric-method . . . 250
TimeMap,TimeMap,ANY,ANY,ANY,ANY-method . . . 250
TimeMap-class . . . 251
TimeMap.from.Dataframe . . . 251
TimeMap.new . . . 252

TimeRangeIntersection 252
transitTime 253
TransportDecompositionOperator-class 253
turnoverFit 254
TwopFeedbackModel 255
TwopFeedbackModel14 257
TwopMMmodel 258
TwopParallelModel 260
TwopParallelModel14 262
TwopSeriesModel 264
TwopSeriesModel14 265
UnBoundInFluxes 267
UnBoundInFluxes,function-method 267
UnBoundInFluxes-class 268
UnBoundLinDecompOp 268
UnBoundLinDecompOp,function-method 268
UnBoundLinDecompOp-class 269
UnBoundNonLinDecompOp 269
UnBoundNonLinDecompOp,function,missing,missing,missing,ANY,ANY,ANY-method 270
UnBoundNonLinDecompOp,missing,missing,missing,missing,character,character,UnBoundNonLinDecompOp_by_Po
method 270
UnBoundNonLinDecompOp,missing,vector,vector,numeric,ANY,ANY,ANY-method . . 271
UnBoundNonLinDecompOp-class 271
UnBoundNonLinDecompOp_by_PoolNames 272
UnBoundNonLinDecompOp_by_PoolNames,InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character-
method 272
UnBoundNonLinDecompOp_by_PoolNames-class 273
Yasso07Model 273
YassoModel 274
[,Model,character,missing,missing-method 276
[,NlModel,character,ANY,ANY-method 276
[[,MCSim-method 277
[[<-,MCSim-method 277
$,NlModel-method 278

Index **279**

`AbsoluteFractionModern` *automatic title*

Description

automatic title

Usage

```
AbsoluteFractionModern(F)
```

Arguments

`F`	: see method arguments

S4 methods for this generic

- `AbsoluteFractionModern,BoundFc-method`
- `AbsoluteFractionModern,ConstFc-method`

`AbsoluteFractionModern,BoundFc-method`
automatic title

Description

automatic title

Usage

```
## S4 method for signature 'BoundFc'
AbsoluteFractionModern(F)
```

Arguments

`F`	object of class:`BoundFc`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`AbsoluteFractionModern,ConstFc-method`
automatic title

Description

automatic title

Usage

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

Arguments

`F`	object of class:`ConstFc`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`AbsoluteFractionModern_from_Delta14C` *conversion param delta14C Object to be converted to AbsoluteFractionModern*

Description

conversion param delta14C Object to be converted to AbsoluteFractionModern

Usage

```
AbsoluteFractionModern_from_Delta14C(delta14C)
```

Arguments

`delta14C`	see method arguments

`AbsoluteFractionModern_from_Delta14C,matrix-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`delta14C`	object of class:`matrix`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`AbsoluteFractionModern_from_Delta14C,numeric-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`delta14C`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`add_plot` *automatic title*

Description

automatic title

Usage

```
add_plot(x, ...)
```

Arguments

`x`	: see method arguments
`...`	: see method arguments

S4 methods for this generic

- `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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`availableParticleProperties` *automatic title*

Description

automatic title

Usage

```
availableParticleProperties(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`availableParticleSets` *automatic title*

Description

automatic title

Usage

```
availableParticleSets(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `availableParticleSets,MCSim-method`

`availableParticleSets,MCSim-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`MCSim`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`availableResidentSets` *automatic title*

Description

automatic title

Usage

```
availableResidentSets(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `availableResidentSets,MCSim-method`

`availableResidentSets,MCSim-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`MCSim`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`AWBmodel` *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_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_max0 in original paper.
`r_B`	a scalar representing the rate constant of microbial death (h-1). Equivalent to r_death in original publication.
`r_E`	a scalar representing the rate constant of enzyme production (h-1). Equivalent to r_EnzProd in original publication.

`r_L`	a scalar representing the rate constant of enzyme loss (h-1). Equivalent to r_EnzLoss in original publication.
`a_BS`	a scalar representing the fraction of the dead microbial biomass incorporated to SOC. 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_0`	a scalar representing the intercept of the half-saturation constant of SOC as a function of temperature (mg cm-3).
`Km_u0`	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 containts default parameters presented in Allison et al. (2010).

Value

An object of class NlModel that can be further queried.

References

Allison, S.D., M.D. Wallenstein, M.A. Bradford. 2010. Soil-carbon response to warming dependent on microbial physiology. Nature Geoscience 3: 336-340.

Examples

```
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

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

Arguments

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

`ExuM`	a scalar representing the maximal exudation rate (ug C/(hr cm3 soil))
`ExuT`	a scalar representing the time constant for exudation, responsible for duration of exudation (1/hr).

Details

This implementation containts 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

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

See Also

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

Examples

```
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

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

Value

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

`BoundFc` *automatic title*

Description

automatic title

Usage

```
BoundFc(format, ...)
```

Arguments

`format`	: see method arguments
`...`	: see method arguments

S4 methods for this generic

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

`BoundFc,character-method`
automatic title

Description

automatic title

Usage

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

Arguments

`format`	object of class:`character`, : : no manual documentation
`...`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`BoundFc,missing-method`
automatic title

Description

automatic title

Usage

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

Arguments

`format`	object of class:`missing`, : : no manual documentation
`...`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`BoundFc-class` *automatic title*

Description

automatic title

`BoundInFluxes` *constructor for BoundInFluxes*

Description

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

Usage

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

Arguments

`...`	passed on to `TimeMap`

`BoundInFluxes-class` *automatic title*

Description

automatic title

`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

BoundLinDecompOp,ANY-method

automatic title

Description

automatic title

Usage

```
## S4 method for signature 'ANY'
BoundLinDecompOp(map, ...)
```

Arguments

`map`	: : no manual documentation
`...`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

BoundLinDecompOp,UnBoundLinDecompOp-method

A converter

Description

The destinction 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

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

Arguments

`map`	object of class:`UnBoundLinDecompOp`, An object of class UnBoundLinDecompOp
`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

---

`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 for this generic

- `by_PoolIndex,ConstantInFluxRate_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantInternalFluxRateList_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantOutFluxRate_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-method`
- `by_PoolIndex,function,character,character-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`

`by_PoolIndex,ConstantInFluxRate_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 '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,ConstantInternalFluxRateList_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`
convert a function f of to f_vec

Description

convert a function f of to f_vec

Usage

```
## 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 orginal 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

```
leaf_resp=function(leaf_pool_content){leaf_pool_content*4}
leaf_resp(1)
poolNames=c(
   "some_thing"
  ,"some_thing_else"
  ,"some_thing_altogther"
  ,"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)
```

---

by_PoolIndex,InFluxList_by_PoolName,character,character-method

automatic title

Description

automatic title

Usage

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

Arguments

obj	object of class:InFluxList_by_PoolName, : : no manual documentation
poolNames	object of class:character, : : no manual documentation
timeSymbol	object of class:character, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`by_PoolIndex,InFlux_by_PoolName,character,character-method`
automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`by_PoolIndex,InternalFluxList_by_PoolName,character,character-method`
automatic title

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

by_PoolIndex,InternalFlux_by_PoolName,character,character-method

automatic title

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

by_PoolIndex,OutFluxList_by_PoolName,character,character-method

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`by_PoolIndex,OutFlux_by_PoolName,character,character-method`
automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`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 guranteed to be of class PoolIndex*

Description

converts the ids if necessary otherwise returns an identical object

Usage

```
## 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

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

Arguments

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

S4 methods for this generic

- `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,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,ConstantInternalFluxRateList_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 '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
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 '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
convert to a list indexed by pool names

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

`by_PoolName,ConstantOutFluxRate_by_PoolIndex-method` *new object with the source pool id converted to a PoolName if necessary*

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

```
## 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

```
data(C14Atm)
```

Format

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

1. V1 a numeric vector

Examples

```
#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(k.STR = 0.094, k.MET = 0.35, k.ACT = 0.14, k.SLW = 0.0038, k.PAS = 0.00013),
  C0 = c(0, 0, 0, 0, 0),
  In,
  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 lenght 5 containing the values of the decomposition rates for the different pools. Units in per week.
`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 (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 `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 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

Parton, W.J, D.S. Schimel, C.V. Cole, and D.S. Ojima. 1987. Analysis of factors controlling soil organic matter levels in Great Plain grasslands. Soil Science Society of America Journal 51: 1173–1179. Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

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

Examples

```
t=seq(0,52*200,1) #200 years
LNcorn=0.17/0.004 # Values for corn clover reported in Parton et al. 1987
Ex=CenturyModel(t,LN=0.5,Ls=0.1,In=0.1)
Ct=getC(Ex)
Rt=getReleaseFlux(Ex)

matplot(t,Ct,type="l", col=1:5,lty=1,ylim=c(0,max(Ct)*2.5),
ylab=expression(paste("Carbon stores (kg C", ha^-1,")")),xlab="Time (weeks)")
lines(t,rowSums(Ct),lwd=2)
legend("topright", c("Structural litter","Metabolic litter",
"Active SOM","Slow SOM","Passive SOM","Total Carbon"),
lty=1,lwd=c(rep(1,5),2),col=c(1:5,1),bty="n")

matplot(t,Rt,type="l",lty=1,ylim=c(0,max(Rt)*3),ylab="Respiration (kg C ha-1 week-1)",xlab="Time")
lines(t,rowSums(Rt),lwd=2)
legend("topright", c("Structural litter","Metabolic litter",
"Active SOM","Slow SOM","Passive SOM","Total Respiration"),
lty=1,lwd=c(rep(1,5),2),col=c(1:5,1),bty="n")
```

CenturyModel_new *Implementation of the Century model*

Description

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

Usage

```
CenturyModel_new(
  t,
 ks = c(k.STR = 0.094, k.MET = 0.35, k.ACT = 0.14, k.SLW = 0.0038, k.PAS = 0.00013),
  C0 = c(0, 0, 0, 0, 0),
  In,
  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 lenght 5 containing the values of the decomposition rates for the different pools. Units in per week.
`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 (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 (`TimeMap` 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 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

Parton, W.J, D.S. Schimel, C.V. Cole, and D.S. Ojima. 1987. Analysis of factors controlling soil organic matter levels in Great Plain grasslands. Soil Science Society of America Journal 51: 1173–1179. Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

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

Examples

```
t=seq(0,52*200,1) #200 years
LNcorn=0.17/0.004 # Values for corn clover reported in Parton et al. 1987
Ex=CenturyModel(t,LN=0.5,Ls=0.1,In=0.1)
Ct=getC(Ex)
Rt=getReleaseFlux(Ex)

matplot(t,Ct,type="l", col=1:5,lty=1,ylim=c(0,max(Ct)*2.5),
ylab=expression(paste("Carbon stores (kg C", ha^-1,")")),xlab="Time (weeks)")
lines(t,rowSums(Ct),lwd=2)
legend("topright", c("Structural litter","Metabolic litter",
"Active SOM","Slow SOM","Passive SOM","Total Carbon"),
lty=1,lwd=c(rep(1,5),2),col=c(1:5,1),bty="n")
```

```
matplot(t,Rt,type="l",lty=1,ylim=c(0,max(Rt)*3),ylab="Respiration (kg C ha-1 week-1)",xlab="Time")
lines(t,rowSums(Rt),lwd=2)
legend("topright", c("Structural litter","Metabolic litter",
"Active SOM","Slow SOM","Passive SOM","Total Respiration"),
lty=1,lwd=c(rep(1,5),2),col=c(1:5,1),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 duplicats

`check_id_length` *helper function to check that the length of the argumetn is exactly 1*

Description

helper function to check that the length of the argumetn 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 for this generic

- `check_pool_ids,PoolConnection_by_PoolIndex,integer-method`

`check_pool_ids,PoolConnection_by_PoolIndex,integer-method` *automatic title*

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

computeResults *automatic title*

Description

automatic title

Usage

```
computeResults(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

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

computeResults,MCSim-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`MCSim`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

computeResults,NlModel-method

automatic title

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
computeResults(object)
```

Arguments

object	object of class:NlModel, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

ConstantInFluxList_by_PoolIndex

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

```
ConstantInFluxList_by_PoolIndex(object)
```

Arguments

object	see method arguments

`ConstantInFluxList_by_PoolIndex,ConstInFluxes-method`

constructor from ConstInFluxes

Description

constructor from ConstInFluxes

Usage

```
## 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

```
## 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`
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`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

---

`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_PoolIndex-class` *automatic title*

Description

automatic title

`ConstantInFluxRate_by_PoolName-class` *automatic title*

Description

automatic title

`ConstantInFlux_by_PoolIndex-class` *class for a constan influx to a single pool identified by index*

Description

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

`ConstantInternalFluxRateList_by_PoolIndex` *Generic constructor for the class with the same name*

Description

Generic constructor for the class with the same name

Usage

```
ConstantInternalFluxRateList_by_PoolIndex(object)
```

Arguments

`object`	see method arguments

`ConstantInternalFluxRateList_by_PoolIndex,list-method` *constructor from a normal list*

Description

constructor from a normal list

Usage

```
## S4 method for signature 'list'
ConstantInternalFluxRateList_by_PoolIndex(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` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`ConstantInternalFluxRateList_by_PoolIndex-class`
Subclass of list that is guaranteed to contain only elements of type ConstantInternalFluxRate_by_PoolIndex

---

Description

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

---

`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

`ConstantInternalFluxRateList_by_PoolName,list-method`
constructor from a normal list

Description

constructor from a normal list

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`

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

---

Description

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

```
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

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

```
## 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,numeric,numeric,missing,numeric-method

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`ConstantInternalFluxRate_by_PoolIndex-class` *S4 class representing a constant internal flux rate*

Description

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

`ConstantInternalFluxRate_by_PoolName` *Generic constructor for the class with the same name*

Description

Generic constructor for the class with the same name

Usage

```
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

ConstantInternalFluxRate_by_PoolName,character,character,missing,numeric-method
constructor with argument conversion

Description

constructor with argument conversion

Usage

```
## 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

ConstantInternalFluxRate_by_PoolName,missing,missing,character,numeric-method
constructor from strings of the form 'a->b'

Description

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

Usage

```
## S4 method for signature 'missing,missing,character,numeric'
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

`ConstantInternalFluxRate_by_PoolName-class`

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

Description

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

`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

`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` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`ConstantOutFluxRateList_by_PoolIndex-class` *Subclass of list that is guaranteed to contain only elements of type ConstantOutFluxRate_by_PoolIndex*

Description

The main purpose of the class is to be used in method signatures which would otherwise only indicate an object of class `'list'` in their signature an then check that the list contains the right kind of elements inside the function. Using this class the method signature becomes much more informative.

`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

`ConstantOutFluxRateList_by_PoolName,list-method` *constructor from a normal list*

Description

constructor from a normal list

Usage

```
## 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.

ConstantOutFluxRateList_by_PoolName,numeric-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

ConstantOutFluxRateList_by_PoolName-class

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

Description

The main purpose of the class is to be used in method signatures which would otherwise only indicate an object of class `'list'` in their signature an then check that the list contains the right kind of elements inside the function. Using this class the method signature becomes much more informative.

`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

`ConstantOutFluxRate_by_PoolIndex,numeric,numeric-method` *automatic title*

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`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

`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

`ConstFc` *creates an object containing the initial values for the 14C fraction needed to create models in SoilR*

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 iformation about the formats their data are in, because the high level function dealing wiht the models have to know. This function is actually a convienient 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 constan ^14C fraction*

Description

S4 class representing a constan ^14C fraction

ConstInFluxes *automatic title*

Description

automatic title

Usage

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

Arguments

map	: see method arguments
numberOfPools	: see method arguments

S4 methods for this generic

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

ConstInFluxes,ConstantInFluxList_by_PoolIndex,numeric-method *automatic title*

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`ConstInFluxes,numeric,ANY-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'numeric,ANY'
ConstInFluxes(map)
```

Arguments

`map`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`ConstInFluxes-class` *S4 class for a constant influx vector*

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.

`ConstLinDecompOp` *Generic constructor for the class with the same name*

Description

Generic constructor for the class with the same name

Usage

```
ConstLinDecompOp(
  mat,
  internal_flux_rates,
  out_flux_rates,
  numberOfPools,
  poolNames
)
```

Arguments

`mat`	see method arguments
`internal_flux_rates`	see method arguments
`out_flux_rates`	see method arguments
`numberOfPools`	see method arguments
`poolNames`	see method arguments

`ConstLinDecompOp,matrix,missing,missing,missing,missing-method` *Constructor*

Description

Constructor

Usage

```
## S4 method for signature 'matrix,missing,missing,missing,missing'
ConstLinDecompOp(mat)
```

Arguments

`mat`	object of class:`matrix`, no manual documentation

ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,ConstantOutFluxRateList_by_PoolInc *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

ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,missing,numeric,missing-method *Constructor*

Description

Constructor

Usage

```
## S4 method for signature
## 'missing,
##   ConstantInternalFluxRateList_by_PoolIndex,
##   missing,
##   numeric,
##   missing'
ConstLinDecompOp(internal_flux_rates, numberOfPools)
```

Arguments

`internal_flux_rates`	object of class:`ConstantInternalFluxRateList_by_PoolIndex`, no manual documentation
`numberOfPools`	object of class:`numeric`, no manual documentation

`ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolName`
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

`ConstLinDecompOp,missing,missing,ConstantOutFluxRateList_by_PoolIndex,numeric,missing-method`
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 (=nonautonomuous,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 (=nonautonomuous,linear) compartmental matrix or equivalently a combination of ordered constant internal flux rates and constant out flux rates.

`ConstLinDecompOpWithLinearScalarFactor`

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

```
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

`ConstLinDecompOpWithLinearScalarFactor,matrix,missing,missing,missing,ScalarTimeMap-method`
convert names of vectors or lists to class ConstantOutFluxRate convert names of vectors or lists to class ConstantInternalFluxRate

---

Description

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

Usage

```
## 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

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

---

Description

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

---

`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-class to represent compartmental operators*

Description

S4-class to represent compartmental operators

`DecompositionOperator-class`
automatic title

Description

automatic title

`Delta14C` *automatic title*

Description

automatic title

Usage

```
Delta14C(F)
```

Arguments

`F`	: see method arguments

S4 methods for this generic

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

`Delta14C,BoundFc-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`F`	object of class:`BoundFc`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

Delta14C,ConstFc-method

automatic title

Description

automatic title

Usage

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

Arguments

F	object of class:ConstFc, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

Delta14C_from_AbsoluteFractionModern

automatic title

Description

automatic title

Usage

```
Delta14C_from_AbsoluteFractionModern(AbsoluteFractionModern)
```

Arguments

AbsoluteFractionModern	: see method arguments

S4 methods for this generic

- 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`deSolve.lsoda.wrapper` *deSolve.lsoda.wrapper*

Description

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

Usage

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

Arguments

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

Value

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

`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.

Details

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

```
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)
```

`euler` *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.

`example.2DBoundInFluxesFromFunction` *example.2DBoundInFluxesFromFunction*

Description

Create a 2-dimensionsonal example of a BoundInFluxes object

Usage

```
example.2DBoundInFluxesFromFunction()
```

Value

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

`example.2DBoundLinDecompOpFromFunction` *example.2DBoundLinDecompOpFromFunction*

Description

An example used in tests and other examples.

Usage

```
example.2DBoundLinDecompOpFromFunction()
```

`example.2DConstFc.Args` *example.2DConstFc.Args*

Description

Create a 2-dimensionsonal 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

```
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

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

`example.2DInFluxes.Args`
example.2DInFluxes.Args

Description

Create a 2-dimensionsonal examples of Influxes objects from different arguments

Usage

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

`example.2DUnBoundLinDecompOpFromFunction` *example.2DUnBoundLinDecompOpFromFunction*

Description

An example used in tests and other examples.

Usage

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

`example.ConstlinDecompOpFromMatrix` *example.ConstlinDecompOpFromMatrix*

Description

An example used in tests and other examples.

Usage

```
example.ConstlinDecompOpFromMatrix()
```

`example.nestedTime2DMatrixList` *create an example nested list that can be*

Description

An example used in tests and other examples.

Usage

```
example.nestedTime2DMatrixList()
```

`example.Time2DArrayList` *create an example TimeMap from 2D array*

Description

An example used in tests and other examples.

Usage

```
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

FcAtm.from.Dataframe *FcAtm.from.Dataframe*

Description

This function is deprecated constructor of the deprecatied 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 secon 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 fracton. 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 *Effects of temperature on decomposition rates according the Arrhenius equation*

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 contant in units of J K^-1 mol^-1.

Value

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

`fT.Century1` *Effects of temperature on decomposition rates according the the CENTURY model*

Description

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

Usage

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

Arguments

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

Value

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

References

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

fT.Century2 — *Effects of temperature on decomposition rates according the the CENTURY model*

Description

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

Usage

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

Arguments

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

Value

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

References

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

fT.Daycent1 — *Effects of temperature on decomposition rates according to the DAYCENT model*

Description

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

Usage

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

Arguments

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

Value

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

References

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

`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

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

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

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

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

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

`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

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

`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

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

`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 vaule.
`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).

`fT.RothC` *Effects of temperature on decomposition rates according to the functions included in the RothC model*

Description

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

Usage

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

Arguments

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

Value

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

Note

This function returns NA for Temp <= -18.3

References

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

`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

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

Arguments

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

Value

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

References

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

`fW.Candy` *Effects of moisture on decomposition rates according to the Candy model*

Description

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

Usage

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

Arguments

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

References

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

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

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

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

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

`fW.Daycent2` *Effects of moisture on decomposition rates according to the DAYCENT model*

Description

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

Usage

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

Arguments

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

Value

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

References

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

`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

```
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

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

fW.Gompertz *Effects of moisture on decomposition rates according to the Gompertz function*

Description

Calculates the effects of water content on decomposition rates.

Usage

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

Arguments

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

References

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

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

Description

Calculates the effects of water content on decomposition rates.

Usage

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

Arguments

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

References

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

fW.RothC — *Effects of moisture on decomposition rates according to the RothC model*

Description

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

Usage

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

Arguments

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

Value

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

References

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

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

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

Arguments

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

References

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

`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

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

Arguments

`Moist`	A scalar or vector containing values of moisture content of a litter or soil pool (%).
`MatricShape`	A scalar that determines when matric limit is reduced to the point that decay can begin to occur.
`MatricLag`	A scalar used to offset the curve to the left or right.
`MoistMin`	A scalar determining the minimum moisture content.

`MoistMax`	A scalar determining the maximum moisture content without diffusion limitations.
`DiffuseShape`	A scalar that determines the range of moisture contents where diffusion is not limiting.
`DiffuseLag`	A scalar used to shift the point when moisture begins to limit diffusion.

Value

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

References

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

`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.
`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. 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

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

See Also

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)")),
xlim=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 nacessary
## (e.g in for parameter estimation ) in functions
## wchich 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

The class of the output depends on the provided arguments

Usage

```
GeneralDecompOp(object)
```

Arguments

`object`	see method arguments

`GeneralDecompOp,DecompOp-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'DecompOp'
GeneralDecompOp(object)
```

Arguments

`object`	object of class:`DecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`GeneralDecompOp,function-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature '`function`'
GeneralDecompOp(object)
```

Arguments

`object`	object of class:`function`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`GeneralDecompOp,list-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'list'
GeneralDecompOp(object)
```

Arguments

`object`	object of class:`list`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`GeneralDecompOp,matrix-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`matrix`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`GeneralDecompOp,TimeMap-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TimeMap`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`GeneralModel` *additional function to create Models*

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 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 lenght 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 radioactivly. 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 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`

`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 feed-back 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 funtions 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(mi
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
```

`GeneralPoolId` *automatic title*

Description

automatic title

automatic title

Usage

```
GeneralPoolId(id)

GeneralPoolId(id)
```

Arguments

`id`	: see method arguments

S4 methods for this generic

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

`GeneralPoolId,character-method`
automatic title

Description

automatic title

Usage

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

Arguments

`id`	object of class:`character`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`GeneralPoolId,numeric-method`
generic factory for this virtual class

Description

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

Usage

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

Arguments

`id`	object of class:`numeric`, no manual documentation

`getAccumulatedRelease` *automatic title*

Description

automatic title

Usage

```
getAccumulatedRelease(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getAccumulatedRelease,Model-method`

`getAccumulatedRelease,Model-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getC` *Generic Function to obtain the contents of the pools for all time steps*

Description

Generic Function to obtain the contents of the pools for all time steps

Usage

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

Arguments

`object`	see method arguments
`as.closures`	see method arguments

`getC,Model-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getC,Model_by_PoolNames-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_by_PoolNames`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getC,NlModel-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation
`as.closures`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getC14` *Generic that yields the ^14C content for all pools and all times*

Description

Generic that yields the ^14C content for all pools and all times

Usage

```
getC14(object)
```

Arguments

`object`	see method arguments

`getC14,Model_14-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model_14'
getC14(object)
```

Arguments

`object`	object of class:`Model_14`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getCompartmentalMatrixFunc`
automatic title

Description

automatic title

Usage

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

Arguments

`object`	: see method arguments
`timeSymbol`	: see method arguments
`state_variable_names`	: see method arguments

S4 methods for this generic

- `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`
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`BoundLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getCompartmentalMatrixFunc,ConstLinDecompOp,ANY,ANY-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TransportDecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getCompartmentalMatrixFunc,UnBoundNonLinDecompOp,ANY,ANY-method`

automatic title

Description

automatic title

automatic title

Usage

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

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

Arguments

`object`	object of class:`UnBoundNonLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getCompartmentalMatrixFunc,UnBoundNonLinDecompOp_by_PoolNames,character,character-method`

Compartmental Matrix as function of the state vector and time

---

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`
`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 metod 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`
automatic title

Description

automatic title

Usage

```
getConstantCompartmentalMatrix(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getConstantCompartmentalMatrix,ConstLinDecompOp-method`
- `getConstantCompartmentalMatrix,ConstLinDecompOpWithLinearScalarFactor-method`

getConstantCompartmentalMatrix,ConstLinDecompOp-method
automatic title

Description

automatic title

Usage

```
## S4 method for signature 'ConstLinDecompOp'
getConstantCompartmentalMatrix(object)
```

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getConstantCompartmentalMatrix,ConstLinDecompOpWithLinearScalarFactor-method
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOpWithLinearScalarFactor`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getConstantInFluxVector` *automatic title*

Description

automatic title

Usage

```
getConstantInFluxVector(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getConstantInFluxVector,ConstInFluxes-method`

`getConstantInFluxVector,ConstInFluxes-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstInFluxes`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getConstantInternalFluxRateList_by_PoolIndex` *automatic title*

Description

automatic title

Usage

```
getConstantInternalFluxRateList_by_PoolIndex(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getConstantInternalFluxRateList_by_PoolIndex,ConstLinDecompOp-method`

`getConstantInternalFluxRateList_by_PoolIndex,ConstLinDecompOp-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'ConstLinDecompOp'
getConstantInternalFluxRateList_by_PoolIndex(object)
```

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getConstantOutFluxRateList_by_PoolIndex` *automatic title*

Description

automatic title

Usage

```
getConstantOutFluxRateList_by_PoolIndex(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getConstantOutFluxRateList_by_PoolIndex,ConstLinDecompOp-method`

`getConstantOutFluxRateList_by_PoolIndex,ConstLinDecompOp-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getConstLinDecompOp` *automatic title*

Description

automatic title

Usage

```
getConstLinDecompOp(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getConstLinDecompOp,ConstLinDecompOpWithLinearScalarFactor-method`

`getConstLinDecompOp,ConstLinDecompOpWithLinearScalarFactor-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOpWithLinearScalarFactor`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getCumulativeC` *automatic title*

Description

automatic title

Usage

```
getCumulativeC(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getCumulativeC,NlModel-method`

`getCumulativeC,NlModel-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
getCumulativeC(object)
```

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getDecompOp *automatic title*

Description

automatic title

Usage

```
getDecompOp(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

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

getDecompOp,Model-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getDecompOp,NlModel-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getDotOut *automatic title*

Description

automatic title

Usage

```
getDotOut(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getDotOut,TransportDecompositionOperator-method`

getDotOut,TransportDecompositionOperator-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TransportDecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getF14

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

Description

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

Usage

```
getF14(object)
```

Arguments

`object`	see method arguments

`getF14,Model_14-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_14`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getF14C` *Generic that yields the ^14C fraction for the cumulative content of all pools and all times*

Description

Generic that yields the ^14C fraction for the cumulative content of all pools and all times

Usage

```
getF14C(object)
```

Arguments

`object`	see method arguments

getF14C,Model_14-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_14`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

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

```
getF14R(object)
```

Arguments

`object`	see method arguments

`getF14R,Model_14-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_14`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getFormat` *automatic title*

Description

automatic title

Usage

```
getFormat(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getFunctionDefinition *automatic title*

Description

automatic title

Usage

```
getFunctionDefinition(object, timeSymbol, poolNames, numberOfPools)
```

Arguments

object	: see method arguments
timeSymbol	: see method arguments
poolNames	: see method arguments
numberOfPools	: see method arguments

S4 methods for this generic

- `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` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstInFluxes`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getFunctionDefinition,ConstLinDecompOp-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getFunctionDefinition,ConstLinDecompOpWithLinearScalarFactor-method

helper function

Description

helper function

Usage

```
## 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

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

Arguments

`object`	object of class:`DecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getFunctionDefinition,InFluxList_by_PoolIndex-method` *automatic title*

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getFunctionDefinition,InFluxList_by_PoolName-method
automatic title

Description

automatic title

automatic title

Usage

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

## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getFunctionDefinition,StateDependentInFluxVector-method
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`StateDependentInFluxVector`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getFunctionDefinition,TimeMap-method`
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TimeMap`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getFunctionDefinition,TransportDecompositionOperator-method`
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TransportDecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getFunctionDefinition,UnBoundInFluxes-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'UnBoundInFluxes'
getFunctionDefinition(object)
```

Arguments

`object`	object of class:`UnBoundInFluxes`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getFunctionDefinition,UnBoundLinDecompOp-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`UnBoundLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getInFluxes` *automatic title*

Description

automatic title

Usage

```
getInFluxes(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

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

`getInFluxes,Model-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getInFluxes,NlModel-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
getInFluxes(object)
```

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getInitialValues` *automatic title*

Description

automatic title

Usage

```
getInitialValues(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- getInitialValues,NlModel-method

---

getInitialValues,NlModel-method *automatic title*

Description

automatic title

Usage

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

Arguments

object	object of class:NlModel, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

---

getLinearScaleFactor *automatic title*

Description

automatic title

Usage

```
getLinearScaleFactor(object)
```

Arguments

object	: see method arguments

S4 methods for this generic

- getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method

getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFactor-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOpWithLinearScalarFactor`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getMeanTransitTime *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	: see method arguments
`inputDistribution`	: see method arguments

S4 methods for this generic

- `getMeanTransitTime,ConstLinDecompOp-method`

`getMeanTransitTime,ConstLinDecompOp-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'ConstLinDecompOp'
getMeanTransitTime(object, inputDistribution)
```

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation
`inputDistribution`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getNumberOfPools` *automatic title*

Description

automatic title

Usage

```
getNumberOfPools(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

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

getNumberOfPools,MCSim-method
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`MCSim`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getNumberOfPools,NlModel-method
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getNumberOfPools,TransportDecompositionOperator-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TransportDecompositionOperator`, `: :` no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`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 *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation
`as.closures`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getOutputReceivers *automatic title*

Description

automatic title

Usage

```
getOutputReceivers(object, i)
```

Arguments

`object`	: see method arguments
`i`	: see method arguments

S4 methods for this generic

- `getOutputReceivers,TransportDecompositionOperator,numeric-method`

`getOutputReceivers,TransportDecompositionOperator,numeric-method` *automatic title*

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getParticleMonteCarloSimulator` *automatic title*

Description

automatic title

Usage

```
getParticleMonteCarloSimulator(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getParticleMonteCarloSimulator,NlModel-method`

getParticleMonteCarloSimulator,NlModel-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

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

getReleaseFlux,Model-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getReleaseFlux,Model_by_PoolNames-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_by_PoolNames`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getReleaseFlux,NlModel-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getReleaseFlux14` *automatic title*

Description

automatic title

Usage

```
getReleaseFlux14(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getReleaseFlux14,Model_14-method`

`getReleaseFlux14,Model_14-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_14`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getRightHandSideOfODE` *automatic title*

Description

automatic title

Usage

```
getRightHandSideOfODE(object, timeSymbol, poolNames, numberOfPools)
```

Arguments

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

S4 methods for this generic

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

`getRightHandSideOfODE,Model-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`getRightHandSideOfODE,Model_by_PoolNames-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model_by_PoolNames`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimeRange *automatic title*

Description

automatic title

Usage

```
getTimeRange(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

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

getTimeRange,ConstInFluxes-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstInFluxes`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimeRange,ConstLinDecompOp-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimeRange,ConstLinDecompOpWithLinearScalarFactor-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstLinDecompOpWithLinearScalarFactor`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimeRange,DecompositionOperator-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`DecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimeRange,TimeMap-method

The time interval where the function is defined

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

getTimeRange,UnBoundInFluxes-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`UnBoundInFluxes`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimeRange,UnBoundLinDecompOp-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`UnBoundLinDecompOp`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimes *automatic title*

Description

automatic title

Usage

```
getTimes(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getTimes,Model-method`
- `getTimes,NlModel-method`

getTimes,Model-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getTimes,NlModel-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

getTransferCoefficients

automatic title

Description

automatic title

automatic title

Usage

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

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

Arguments

`object`	: see method arguments
`as.closures`	: see method arguments

S4 methods for this generic

- `getTransferCoefficients,NlModel-method`
- `getTransferCoefficients,TransportDecompositionOperator-method`
- `getTransferCoefficients,NlModel-method`
- `getTransferCoefficients,TransportDecompositionOperator-method`

`getTransferCoefficients,NlModel-method`
automatic title

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getTransferCoefficients,TransportDecompositionOperator-method`
automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TransportDecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`getTransferMatrix` *deprecated, use getTransferMatrixFunc instead*

Description

deprecated, use getTransferMatrixFunc instead

Usage

```
getTransferMatrix(object)
```

Arguments

`object`	A compartmental operator

`getTransferMatrixFunc` *automatic title*

Description

automatic title

Usage

```
getTransferMatrixFunc(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getTransferMatrixFunc,TransportDecompositionOperator-method`

getTransferMatrixFunc,TransportDecompositionOperator-method

automatic title

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TransportDecompositionOperator`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

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 for this generic

- `getTransitTimeDistributionDensity,ConstLinDecompOp-method`

getTransitTimeDistributionDensity,ConstLinDecompOp-method
automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

getValues *automatic title*

Description

automatic title

Usage

```
getValues(object)
```

Arguments

`object`	: see method arguments

S4 methods for this generic

- `getValues,ConstFc-method`

getValues,ConstFc-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`ConstFc`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

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

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

Examples

```
plot(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

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

References

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

Examples

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

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 sougth.
ks	A vector of length 2 with the decomposition rates for the young and the old pool.
h	Humufication 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

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

See Also

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)
#lines(times,rowSums(CtSS),lty=4,lwd=2)
legend("topleft",
  c("Bare fallow",
    "+N +Straw",
    "-N +Straw",
    "-N -Straw",
    "+N -Straw",
    "Manure",
   "Sludge"
  ),
  lty=c(1,2,3,4,1,2,3),
  lwd=c(1,1,1,1,2,2,2),
  bty="n"
)
```

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 th 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

The actual class of the returned object depends on the arguments provided

Usage

```
InFluxes(object, numberOfPools)
```

Arguments

`object`	see method arguments
`numberOfPools`	see method arguments

`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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

InFluxes,function-method

automatic title

Description

automatic title

Usage

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

Arguments

object	object of class:`function`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

InFluxes,InFluxes-method

automatic title

Description

automatic title

Usage

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

Arguments

object	object of class:`InFluxes`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`InFluxes,list-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`list`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`InFluxes,numeric-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`numeric`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

InFluxes,TimeMap-method *automatic title*

Description

automatic title

Usage

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

Arguments

`object`	object of class:`TimeMap`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

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

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

Description

Generic constructor for the class with the same name

Usage

```
InFluxList_by_PoolIndex(object)
```

Arguments

`object`	see method arguments

`InFluxList_by_PoolIndex,list-method` *constructor from a normal list*

Description

after checking the elememts

Usage

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

Arguments

`object`	object of class:`list`, no manual documentation

`InFluxList_by_PoolIndex-class` *automatic title*

Description

automatic title

`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

`InFluxList_by_PoolName,list-method`
constructor from a normal list

Description

after checking the elememts

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

---

`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

`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 varaibles. 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 tranformed 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 theindex

Description

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

`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)
```

Arguments

`func`	see method arguments
`destinationName`	see method arguments

`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 s tate 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

`initialize,ConstLinDecompOp-method`
automatic title

Description

automatic title

Usage

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

Arguments

`.Object`	object of class:`ConstLinDecompOp`, : : no manual documentation
`mat`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`initialize,DecompositionOperator-method`
automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`initialize,MCSim-method`
automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

initialize,Model-method

automatic title

Description

automatic title

Usage

```
## 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
`inputFluxes`	: : no manual documentation
`solverfunc`	: : no manual documentation
`pass`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

initialize,Model_14-method

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

initialize,Model_by_PoolNames-method

automatic title

Description

automatic title

Usage

```
## S4 method for signature 'Model_by_PoolNames'
initialize(
  .Object,
  times,
  mat,
  initialValues,
  inputFluxes,
  solverfunc,
  pass,
  timeSymbol
)
```

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
`solverfunc`	: : no manual documentation
`pass`	: : no manual documentation
`timeSymbol`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

initialize,NlModel-method

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

initialize,TimeMap-method

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

initialize,TransportDecompositionOperator-method

automatic title

Description

automatic title

Usage

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

Arguments

`.Object`	object of class:`TransportDecompositionOperator`, : : no manual documentation
`starttime`	: : no manual documentation
`endtime`	: : no manual documentation
`numberOfPools`	: : no manual documentation
`alpha`	: : no manual documentation
`f`	: : no manual documentation
`lag`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`initialize,UnBoundInFluxes-method` *automatic title*

Description

automatic title

Usage

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

Arguments

`.Object`	object of class:`UnBoundInFluxes`, : : no manual documentation
`map`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`initialize,UnBoundLinDecompOp-method` *automatic title*

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`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

`Deltal.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

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

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

Examples

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

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

`IntCal13` *Atmospheric radiocarbon for the 0-50,000 yr BP period*

Description

Atmospheric radiocarbon calibration curve for the period 0 to 50,000 yr BP. This is the most recent update to the internationally agreed calibration curve and superseds `IntCal09`.

Usage

```
data(IntCal13)
```

Format

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

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

`C14.age` C14 age in years BP.

`Error` Error estimate for `C14.age`.

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

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

Details

`Deltal.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

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

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

Examples

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

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

`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

`InternalFluxList_by_PoolIndex,list-method` *constructor from a normal list*

Description

after checking the elememts

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 inidices*

Description

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

`InternalFluxList_by_PoolName` *Generic constructor for the class with the same name*

Description

Generic constructor for the class with the same name

Usage

```
InternalFluxList_by_PoolName(object)
```

Arguments

`object`	see method arguments

`InternalFluxList_by_PoolName,list-method` *constructor from a normal list*

Description

constructor from a normal list

Usage

```
## 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

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.

InternalFluxList_by_PoolName-class

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

Description

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

InternalFlux_by_PoolIndex

Generic constructor for the class with the same name

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

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 varaibles. 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 tranformed to a different ordering of state variables, since the location of a state variable in the vector argument depends on a specific orderand 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 wiht source and destination pools specified by indices

---

Description

S4-class for a single internal flux wiht 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

`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 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

`InternalFlux_by_PoolName,function,missing,missing,character-method`

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`InternalFlux_by_PoolName-class` *S4-class for a single internal flux wiht source and destination pools specified by name*

Description

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

---

`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 inputlists (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` *automatic title*

Description

automatic title

`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 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 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 analyse the errormassage (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. violoations of massbalance 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

```
# 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 choeices 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)
```

Model-class *S4 class representing a model run*

Description

S4 class representing a model run

`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

```
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 `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.
`inputFc`	An object describing the fraction of C_14 in per mille (different formats are possible)
`c14DecayRate`	the rate at which C_14 decays radioactivly. 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 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

```
# 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 choeices 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)

# inst/examples/ModelExamples.R CorrectNonautonomousLinearModelExplicit:

  # 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 convienient kinds of arguments,
  # we will in this example call the constructor whith arguments which
  # are of the same type as one of hte 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 ofperator  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 ?BoundLindDcompOp)
    ## 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 unitendet EXTRApolation )
    starttime=0,
    endtime=20
  )
  I=BoundInFluxes(
     ## The first argument is a vector-valued function of time
     function(t){
       matrix(nrow=3,ncol=1,byrow=TRUE,
           c(-1,    0,    0)
       )
     },
     ## The other two arguments describe the time interval where the
     ## function is valid (the domain of the function)
     starttime=0,
     endtime=40
  )
  ## No we specify the points in time where we want
  ## to compute results
  t_start=0
  t_end=10
  tn=50
  timestep <- (t_end-t_start)/tn
  times <- seq(t_start,t_end,timestep)
  ## and the start values
  sv=c(0,0,0)
  mod=Model(t=times,A,sv,I)

  ## No we use the model to compute some results
  getC(mod)
  getReleaseFlux(mod)
  #also look at the methods section of Model-class
```

`Model_14-class` *S4-class to represent a ^14C model run*

Description

S4-class to represent a ^14C model run

`Model_by_PoolNames-class` *automatic title*

Description

automatic title

`NlModel-class` *deprecated class for a non-linear model run.*

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

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

See Also

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)
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="Cummulative 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

`OutFluxList_by_PoolIndex,list-method`
constructor from a normal list

Description

after checking the elememts

Usage

```
## S4 method for signature 'list'
OutFluxList_by_PoolIndex(object)
```

Arguments

`object`	object of class:`list`, no manual documentation

`OutFluxList_by_PoolIndex-class`
automatic title

Description

automatic title

`OutFluxList_by_PoolName`
Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

```
OutFluxList_by_PoolName(object)
```

Arguments

`object`	see method arguments

`OutFluxList_by_PoolName,list-method`
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

`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

`func`	see method arguments
`sourceIndex`	see method arguments

`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

`func`	object of class:`function`, A function f(X,t) where X is a vector of the state varaibles. 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 tranformed to a different ordering of state variables, since the location of a state variable in the vector argument depends on a specific orderand 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
`sourceIndex`	object of class:`numeric`, no manual documentation

`OutFlux_by_PoolIndex-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

`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 aguemnts 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 stave variabl names the formulation can automatically be transformed to a function of a s tate 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

`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 convinient 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 funtions 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

```
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],lt2,type="l",lty=lt2,col=col2)
lines(t,Y[,3],type="l",lty=lt3,col=col3)
legend("topright",c("R1","R2","R3"),lty=c(lt1,lt2,lt3),col=c(col1,col2,col3))
```

---

plot,MCSim-method *automatic title*

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`plot,Model-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model'
plot(x)
```

Arguments

`x`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

plot,NlModel-method *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
plot(x)
```

Arguments

x	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

plot,TimeMap-method *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'TimeMap'
plot(x, y, ...)
```

Arguments

x	object of class:`TimeMap`, : : no manual documentation
y	: : no manual documentation
...	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

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

```
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

```
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.

`plotPoolGraph,Model-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model'
plotPoolGraph(x)
```

Arguments

`x`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`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 for this generic

- `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

`PoolConnection_by_PoolName` *automatic title*

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 for this generic

- `PoolConnection_by_PoolName,ANY,ANY,missing-method`

`PoolConnection_by_PoolName,ANY,ANY,missing-method`

constructor from an ordered pair of PoolName objects

Description

constructor from an ordered pair of PoolName objects

Usage

```
## S4 method for signature 'ANY,ANY,missing'
PoolConnection_by_PoolName(source, destination)
```

Arguments

`source`	no manual documentation
`destination`	no manual documentation

`PoolConnection_by_PoolName-class`

Objects that have a source and a destination determined by a string like object of class PoolName

---

Description

Examples are internal Fluxex 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

PoolIndex *automatic title*

Description

automatic title

Usage

```
PoolIndex(id, ...)
```

Arguments

id	: see method arguments
...	: see method arguments

S4 methods for this generic

- 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

```
## 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

```
## S4 method for signature 'numeric'
PoolIndex(id)
```

Arguments

id	object of class:`numeric`, no manual documentation

PoolIndex,PoolIndex-method

pass throug constructor fron an object of the same class

Description

This is here to be able to call PoolIndex on a PoolIndex ojbect without having to chech before if it is necessary. the unnecessary poolNames argument will be ignored

Usage

```
## S4 method for signature 'PoolIndex'
PoolIndex(id, poolNames)
```

Arguments

id	object of class:`PoolIndex`, no manual documentation
poolNames	no manual documentation

PoolIndex,PoolName-method *convert to number like object*

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

PoolIndex-class *S4 class for pool indices*

Description

used to dispatch pool index specific methods like conversion to names.

PoolName *automatic title*

Description

automatic title

Usage

```
PoolName(id, ...)
```

Arguments

id	: see method arguments
...	: see method arguments

S4 methods for this generic

- 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

```
## 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 throug constructor fron an object of the same class*

Description

This is here to be able to call PoolName on a PoolName ojbect without having to test before if we have to.

Usage

```
## 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

`predefinedModels` *PREDEFINED MODELS*

Description

`GaudinskiModel14`
`ICBMModel`
`OnepModel`
`OnepModel14`
`RothCModel`
`ThreepFeedbackModel`
`ThreepFeedbackModel14`
`ThreepParallelModel`
`ThreepParallelModel14`
`ThreepSeriesModel`

ThreepSeriesModel14
TwopFeedbackModel
TwopFeedbackModel14
TwopParallelModel
TwopParallelModel14
TwopMMmodel
ThreepairMMmodel
TwopSeriesModel
TwopSeriesModel14
YassoModel
bacwaveModel
Yasso07Model
SeriesLinearModel
SeriesLinearModel14
CenturyModel

`print,Model-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model'
print(x)
```

Arguments

x	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`print,NlModel-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
print(x)
```

Arguments

`x`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`RespirationCoefficients` *helper function to compute respiration coefficients*

Description

This function computes the respiration coefficients as function of time for all pools according to the given matrix function A(t)

Usage

```
RespirationCoefficients(A)
```

Arguments

`A`	A matrix valued function representing the model.

Value

A vector valued function of time containing the respiration coefficients for all pools.

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 lenght 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

Jenkinson, D. S., S. P. S. Andrew, J. M. Lynch, M. J. Goss, and P. B. Tinker. 1990. The Turnover of Organic Carbon and Nitrogen in Soil. Philosophical Transactions: Biological Sciences 329:361-368. Sierra, C.A., M. Mueller, S.E. Trumbore. 2012. Models of soil organic matter decomposition: the SoilR package version 1.0. Geoscientific Model Development 5, 1045-1060.

See Also

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` *automatic title*

Description

automatic title

Usage

```
ScalarTimeMap(
  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 for this generic

- `ScalarTimeMap,missing,missing,missing,missing,numeric-method`

`ScalarTimeMap,missing,missing,missing,missing,numeric-method` *constructor*

Description

special case for a time map from a constant

Usage

```
## 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-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

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

```
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 lenght 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

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

See Also

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

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")
```

SeriesLinearModel14 *General m-pool linear C14 model with series structure*

Description

This function creates a radiocarbon model for m number of pools connected in series. It is a wrapper for the more general function `GeneralModel_14`.

Usage

```
SeriesLinearModel14(
  t,
  m.pools,
  ki,
  Tij,
  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.
`m.pools`	An integer with the total number of pools in the model.
`ki`	A vector of lenght 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 containig 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

Sierra, C.A., M. Mueller, S.E. Trumbore. 2014. Modeling radiocarbon dynamics in soils: SoilR version 1.1. Geoscientific Model Development 7, 1919-1931.

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",
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))
```

`show,Model-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model'
show(object)
```

Arguments

`object`	object of class:Model, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags

with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`show,NlModel-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
show(object)
```

Arguments

`object`	object of class:`NlModel`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`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

```
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`
automatic title

Description

automatic title

`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

`StateIndependentInFluxList_by_PoolIndex,list-method`
constructor from a normal list

Description

constructor from a normal list

Usage

```
## 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`
Subclass of list that is guaranteed to contain only elements of type StateIndependentInFlux_by_PoolIndex

---

Description

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

---

`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`

`summary,Model-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model'
summary(object)
```

Arguments

`object`	object of class:`Model`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`systemAge` *System and pool age for compartment models*

Description

Computes the density distribution and mean for the system and pool ages of a SoilR model or a matrix representation of a compartmental model

Usage

```
systemAge(A, u, a = seq(0, 100), q = c(0.05, 0.5, 0.95))
```

Arguments

`A`	A compartmental linear 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

ThreepairMMmodel *Implementation of a 6-pool Michaelis-Menten model*

Description

This function implements a 6-pool Michaelis-Meneten 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

```
days=seq(0,1000)
#Run the model with default parameter values
MMmodel=ThreepairMMmodel(t=days,ival=rep(c(100,10),3),ks=c(0.1,0.05,0.01),
kb=c(0.005,0.001,0.0005),Km=c(100,150,200),r=c(0.9,0.9,0.9),
ADD=c(3,1,0.5))
Cpools=getC(MMmodel)
#Time solution
matplot(days,Cpools,type="l",ylab="Concentrations",xlab="Days",lty=rep(1:2,3),
ylim=c(0,max(Cpools)*1.2),col=rep(1:3,each=2),
main="Multi-substrate microbial model")
legend("topright",c("Substrate 1", "Microbial biomass 1",
"Substrate 2", "Microbial biomass 2",
"Substrate 3", "Microbial biomass 3"),
lty=rep(1:2,3),col=rep(1:3,each=2),
bty="n")
#State-space diagram
plot(Cpools[,2],Cpools[,1],type="l",ylab="Substrate",xlab="Microbial biomass")
lines(Cpools[,4],Cpools[,3],col=2)
lines(Cpools[,6],Cpools[,5],col=3)
legend("topright",c("Substrate-Enzyme pair 1","Substrate-Enzyme pair 2",
"Substrate-Enzyme pair 3"),col=1:3,lty=1,bty="n")
#Microbial biomass over time
plot(days,Cpools[,2],type="l",col=2,xlab="Days",ylab="Microbial biomass")
```

ThreepFeedbackModel *Implementation of a three pool model with feedback structure*

Description

This function creates a model for three pools connected with feedback. It is a wrapper for the more general function GeneralModel.

Usage

```
ThreepFeedbackModel(
  t,
  ks,
  a21,
  a12,
  a32,
  a23,
  C0,
  In,
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

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

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

See Also

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)
ks=c(k1=0.8,k2=0.4,k3=0.2)
C0=c(C10=100,C20=150, C30=50)
In = 60

Temp=rnorm(t,15,1)
TempEffect=data.frame(t,fT.Daycent1(Temp))

Ex1=ThreepFeedbackModel(t=t,ks=ks,a21=0.5,a12=0.1,a32=0.2,a23=0.1,C0=C0,In=In,xi=TempEffect)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plot(
t,
rowSums(Ct),
type="l",
ylab="Carbon stocks (arbitrary units)",
xlab="Time (arbitrary units)",
lwd=2,
ylim=c(0,sum(Ct[51,]))
)
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
lines(t,Ct[,3],col=3)
```

```
legend(
"topleft",
c("Total C","C in pool 1", "C in pool 2","C in pool 3"),
lty=c(1,1,1,1),
col=c(1,2,4,3),
lwd=c(2,1,1,1),
bty="n"
)

plot(
t,
rowSums(Rt),
type="l",
ylab="Carbon released (arbitrary units)",
xlab="Time (arbitrary units)",
lwd=2,
ylim=c(0,sum(Rt[51,]))
)
lines(t,Rt[,1],col=2)
lines(t,Rt[,2],col=4)
lines(t,Rt[,3],col=3)
legend(
"topleft",
c("Total C release",
"C release from pool 1",
"C release from pool 2",
"C release from pool 3"),
lty=c(1,1,1,1),
col=c(1,2,4,3),
lwd=c(2,1,1,1),
bty="n"
)

Inr=data.frame(t,Random.inputs=rnorm(length(t),50,10))
plot(Inr,type="l")

Ex2=ThreepFeedbackModel(t=t,ks=ks,a21=0.5,a12=0.1,a32=0.2,a23=0.1,C0=C0,In=Inr)
Ctr=getC(Ex2)
Rtr=getReleaseFlux(Ex2)

plot(
t,
rowSums(Ctr),
type="l",
ylab="Carbon stocks (arbitrary units)",
xlab="Time (arbitrary units)",
lwd=2,
ylim=c(0,sum(Ctr[51,]))
)
lines(t,Ctr[,1],col=2)
lines(t,Ctr[,2],col=4)
lines(t,Ctr[,3],col=3)
legend("topright",c("Total C","C in pool 1", "C in pool 2","C in pool 3"),
lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")

plot(t,rowSums(Rtr),type="l",ylab="Carbon released (arbitrary units)",
xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Rtr[51,])))
lines(t,Rtr[,1],col=2)
lines(t,Rtr[,2],col=4)
lines(t,Rtr[,3],col=3)
legend(
"topright",
c("Total C release",
"C release from pool 1",
"C release from pool 2",
"C release from pool 3"
),
lty=c(1,1,1,1),
col=c(1,2,4,3),
lwd=c(2,1,1,1),
bty="n")
```

ThreepFeedbackModel14 *Implementation of a three-pool C14 model with feedback structure*

Description

This function creates a model for three pools connected with feedback. It is a wrapper for the more general function GeneralModel_14 that can handle an arbitrary number of pools with arbitrary 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 externel packages for parameter estimation.

See Also

There are other predefinedModels and also more general functions like Model_14.

Examples

```
#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),
F0_Delta14C=c(0,0,0),
In=LitterInput,
gam1=0.6,
gam2=0.2,
inputFc=C14Atm_NH,
lag=2
)
P.R14m=getF14R(Parallel)
P.C14m=getF14C(Parallel)
P.C14t=getF14(Parallel)

par(mfrow=c(3,2))
plot(
C14Atm_NH,
type="l",
xlab="Year",
ylab=expression(paste(Delta^14,"C ","(\u2030)")),
xlim=c(1940,2010)
)
lines(years, P.C14t[,1], col=4)
lines(years, P.C14t[,2],col=4,lwd=2)
lines(years, P.C14t[,3],col=4,lwd=3)
legend(
"topright",
c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),
lty=rep(1,4),
col=c(1,4,4,4),
lwd=c(1,1,2,3),
bty="n"
)

plot(C14Atm_NH,type="l",xlab="Year",
ylab=expression(paste(Delta^14,"C ","(\u2030)")),xlim=c(1940,2010))
lines(years,P.C14m,col=4)
lines(years,P.R14m,col=2)
legend("topright",c("Atmosphere","Bulk SOM", "Respired C"),
lty=c(1,1,1), col=c(1,4,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
ylab=expression(paste(Delta^14,"C ","(\u2030)")),xlim=c(1940,2010))
lines(years, S.C14t[,1], col=4)
lines(years, S.C14t[,2],col=4,lwd=2)
lines(years, S.C14t[,3],col=4,lwd=3)
legend("topright",c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),
lty=rep(1,4),col=c(1,4,4,4),lwd=c(1,1,2,3),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
ylab=expression(paste(Delta^14,"C ","(\u2030)")),xlim=c(1940,2010))
lines(years,S.C14m,col=4)
lines(years,S.R14m,col=2)
legend("topright",c("Atmosphere","Bulk SOM", "Respired C"),
lty=c(1,1,1), col=c(1,4,2),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
ylab=expression(paste(Delta^14,"C ","(\u2030)")),xlim=c(1940,2010))
lines(years, F.C14t[,1], col=4)
lines(years, F.C14t[,2],col=4,lwd=2)
lines(years, F.C14t[,3],col=4,lwd=3)
legend("topright",c("Atmosphere", "Pool 1", "Pool 2", "Pool 3"),
lty=rep(1,4),col=c(1,4,4,4),lwd=c(1,1,2,3),bty="n")

plot(C14Atm_NH,type="l",xlab="Year",
ylab=expression(paste(Delta^14,"C ","(\u2030)")),xlim=c(1940,2010))
lines(years,F.C14m,col=4)
lines(years,F.R14m,col=2)
legend("topright",c("Atmosphere","Bulk SOM", "Respired C"),
lty=c(1,1,1), col=c(1,4,2),bty="n")


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

ThreepParallelModel *Implementation of a three pool model with parallel structure*

Description

The function creates a model for three independent (parallel) pools. It is a wrapper for the more general function `ParallelModel` that can handle an arbitrary number of pools.

Usage

```
ThreepParallelModel(
  t,
  ks,
  C0,
  In,
  gam1,
  gam2,
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

`t`	A vector containing the points in time where the solution is sought.
`ks`	A vector of length 3 containing the decomposition rates for the 3 pools.
`C0`	A vector of length 3 containing the initial amount of carbon for the 3 pools.
`In`	A scalar or a data.frame object specifying the amount of litter inputs by time.
`gam1`	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.
`gam2`	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 2.
`xi`	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
`solver`	A function that solves the system of ODEs. This can be `euler` or `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 chect suggest problems.

References

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

See Also

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=ThreepParallelModel(t,ks=c(k1=0.5,k2=0.2,k3=0.1),
C0=c(c10=100, c20=150,c30=50),In=20,gam1=0.7,gam2=0.1,xi=0.5)
Ct=getC(Ex)

plot(t,rowSums(Ct),type="l",lwd=2,
ylab="Carbon stocks (arbitrary units)",xlab="Time",ylim=c(0,sum(Ct[1,])))
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
lines(t,Ct[,3],col=3)
legend("topright",c("Total C","C in pool 1", "C in pool 2","C in pool 3"),
lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")

Rt=getReleaseFlux(Ex)
plot(t,rowSums(Rt),type="l",ylab="Carbon released (arbitrary units)",
xlab="Time",lwd=2,ylim=c(0,sum(Rt[1,])))
lines(t,Rt[,1],col=2)
lines(t,Rt[,2],col=4)
lines(t,Rt[,3],col=3)
legend("topright",c("Total C release","C release from pool 1",
"C release from pool 2","C release from pool 3"),
lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")
```

ThreepParallelModel14 *Implementation of a three-pool C14 model with parallel structure*

Description

This function creates a model for two independent (parallel) pools. It is a wrapper for the more general function GeneralModel_14 that can handle an arbitrary number of pools.

Usage

```
ThreepParallelModel14(
  t,
  ks,
  C0,
  F0_Delta14C,
  In,
  gam1,
  gam2,
  xi = 1,
  inputFc,
  lambda = -0.0001209681,
  lag = 0,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 3 containing the decomposition rates for the 3 pools.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.
F0_Delta14C	A vector of length 3 containing the initial amount of the radiocarbon fraction for the 3 pools in Delta14C values in per mil.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
gam1	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.
gam2	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 2.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default lambda=-0.0001209681 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(1903,2009,by=0.5) # note that we
LitterInput=700

Ex=ThreepParallelModel14(
t=years,
ks=c(k1=1/2.8, k2=1/35, k3=1/100),
C0=c(200,5000,500),
F0_Delta14C=c(0,0,0),
In=LitterInput,
gam1=0.7,
gam2=0.1,
inputFc=C14Atm_NH,
lag=2
)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
lines(years, C14t[,3],col=4,lwd=3)
legend(
"topright",
c(
"Delta 14C Atmosphere",
"Delta 14C pool 1",
"Delta 14C pool 2",
"Delta 14C pool 3"
),
lty=rep(1,4),
col=c(1,4,4,4),
lwd=c(1,1,2,3),
bty="n"
)

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

ThreepSeriesModel *Implementation of a three pool model with series structure*

Description

This function creates a model for three pools connected in series. It is a wrapper for the more general function GeneralModel.

Usage

```
ThreepSeriesModel(
  t,
  ks,
  a21,
  a32,
  C0,
  In,
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

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

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

See Also

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)
ks=c(k1=0.8,k2=0.4,k3=0.2)
C0=c(C10=100,C20=150, C30=50)
In = 50
```

```
Ex1=ThreepSeriesModel(t=t,ks=ks,a21=0.5,a32=0.2,C0=C0,In=In,xi=fT.Q10(15))
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plot(t,rowSums(Ct),type="l",ylab="Carbon stocks (arbitrary units)",
xlab="Time (arbitrary units)",lwd=2,ylim=c(0,sum(Ct[1,])))
lines(t,Ct[,1],col=2)
lines(t,Ct[,2],col=4)
lines(t,Ct[,3],col=3)
legend("topright",c("Total C","C in pool 1", "C in pool 2","C in pool 3"),
lty=c(1,1,1,1),col=c(1,2,4,3),lwd=c(2,1,1,1),bty="n")
```

ThreepSeriesModel14 *Implementation of a three-pool C14 model with series structure*

Description

This function creates a model for three pools connected in series. It is a wrapper for the more general function GeneralModel_14 that can handle an arbitrary number of pools.

Usage

```
ThreepSeriesModel14(
  t,
  ks,
  C0,
  F0_Delta14C,
  In,
  a21,
  a32,
  xi = 1,
  inputFc,
  lambda = -0.0001209681,
  lag = 0,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 3 containing the decomposition rates for the 3 pools.
C0	A vector of length 3 containing the initial amount of carbon for the 3 pools.
F0_Delta14C	A vector of length 3 containig 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

```
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

`TimeMap,data.frame,missing,missing,missing,missing-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'data.frame,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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`TimeMap,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'
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`, : : no manual documentation
`lag`	: : no manual documentation
`interpolation`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

TimeMap,missing,missing,missing,numeric,array-method

automatic title

Description

automatic title

Usage

```
## 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:array, : : no manual documentation
lag	: : no manual documentation
interpolation	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

TimeMap,missing,missing,missing,numeric,list-method

automatic title

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`TimeMap,missing,missing,missing,numeric,matrix-method` *automatic title*

Description

automatic title

Usage

```
## 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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

TimeMap,missing,missing,missing,numeric,numeric-method
automatic title

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

TimeMap,TimeMap,ANY,ANY,ANY,ANY-method
automatic title

Description

automatic title

Usage

```
## S4 method for signature 'TimeMap,ANY,ANY,ANY,ANY'
TimeMap(map)
```

Arguments

`map`	object of class:`TimeMap`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`TimeMap-class` *S4 class for a time dependent function*

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.

`TimeMap.from.Dataframe` *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 #################### use the generic TimeMap(...) instead

Usage

```
TimeMap.new(t_start, t_end, f)
```

Arguments

`t_start`	A number marking the begin of the time domain where the function is valid
`t_end`	A number the end of the time domain where the function is valid
`f`	The time dependent function definition (a function in R's sense)

Value

An object of class TimeMap that can be used to describe models.

`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 compartmental model

Usage

```
transitTime(A, u, a = seq(0, 100), q = c(0.05, 0.5, 0.95))
```

Arguments

`A`	A compartmental linear 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`

`TransportDecompositionOperator-class` *automatic title*

Description

automatic title

turnoverFit *Estimation of the turnover time from a soil radiocarbon sample.*

Description

This function finds the best possible value of turnover time from a soil radiocarbon sample assuming a one pool model and annual litter inputs.

Usage

```
turnoverFit(
  obsC14,
  obsyr,
  In,
  C0 = 0,
  yr0 = 1900,
  Zone = "NHZone2",
  plot = TRUE,
  by = 0.5
)
```

Arguments

`obsC14`	a scalar with the observed radiocarbon value in Delta14C of the soil sample.
`obsyr`	a scalar with the year in which the soil sample was taken.
`In`	a scalar or data.frame with the annual amount of litter inputs to the soil.
`C0`	a scalar with the initial amount of carbon stored at the begning of the simulation.
`yr0`	The year at which simulations will start.
`Zone`	the hemispheric zone of atmospheric radiocarbon. Possible values are NHZone1: northern hemisphere zone 1, NHZone2: northern hemisphere zone 2, NHZone3: northern hemisphere zone 3, SHZone12: southern hemisphere zones 1 and 2, SHZone3: southern hemisphere zone 3. See `Hua2013` for additional information.
`plot`	logical. Should the function produce a plot?
`by`	numeric. The increment of the sequence of years used in the simulations.

Details

This algorithm takes the observed values and a given amount of litter inputs, runs `OnepModel14`, calculates the squared difference between predictions and observations, and uses `optimize` to find the minimum difference. If the turnover time is relatively short (< 50 yrs), it is safe to assume C0=0 because the soil will reach steady state within the simulation time. However, for longer turnover times it is recommended to use a value of C0 close to the steady state value.

Value

A scalar with the value of the turnover time that minimizes the difference between the prediction of a one pool model and the observed radiocarbon value.

`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

```
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.
`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

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

See Also

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=C0,In=Inmean,gam=0.9,
xi=(fT.Daycent1(15)*fW.Demeter(15)))
Series=TwopSeriesModel(t=times,ks=ks,a21=0.2*ks[1],C0=C0,In=InSin,
xi=(fT.Daycent1(15)*fW.Demeter(15)))
Feedback=TwopFeedbackModel(t=times,ks=ks,a21=0.2*ks[1],a12=0.5*ks[2],C0=C0,
In=InRand,xi=MoistEffect)

CtP=getC(Parallel)
CtS=getC(Series)
CtF=getC(Feedback)

RtP=getReleaseFlux(Parallel)
RtS=getReleaseFlux(Series)
RtF=getReleaseFlux(Feedback)

par(mfrow=c(2,1),mar=c(4,4,1,1))
plot(times,rowSums(CtP),type="l",ylim=c(0,20),ylab="Carbon stocks (arbitrary units)",xlab=" ")
lines(times,rowSums(CtS),col=2)
lines(times,rowSums(CtF),col=3)
legend("topleft",c("Two-pool Parallel","Two-pool Series","Two-pool Feedback"),
lty=c(1,1,1),col=c(1,2,3),bty="n")

plot(times,rowSums(RtP),type="l",ylim=c(0,3),ylab="Carbon release (arbitrary units)", xlab="Time")
lines(times,rowSums(RtS),col=2)
lines(times,rowSums(RtF),col=3)
par(mfrow=c(1,1))
```

TwopFeedbackModel14 *Implementation of a two-pool C14 model with feedback structure*

Description

This function creates a model for two pools connected with feedback. It is a wrapper for the more general function GeneralModel_14 that can handle an arbitrary number of pools.

Usage

```
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=TwopFeedbackModel14(t=years,ks=c(k1=1/2.8, k2=1/35),C0=c(200,5000),
F0_Delta14C=c(0,0),In=LitterInput, a21=0.1,a12=0.01,inputFc=C14Atm_NH)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)

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

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

`TwopMMmodel` *Implementation of a two-pool Michaelis-Menten model*

Description

This function implements a two-pool Michaelis-Meneten 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

Manzoni, S, A. Porporato (2007). A theoretical analysis of nonlinearities and feedbacks in soil carbon and nitrogen cycles. Soil Biology and Biochemistry 39: 1542-1556.

See Also

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

Examples

```
days=seq(0,1000,0.5)
MMmodel=TwopMMmodel(t=days,ival=c(100,10))
Cpools=getC(MMmodel)
matplot(days,Cpools,type="l",ylab="Concentrations",xlab="Days",lty=1,ylim=c(0,max(Cpools)*1.2))
legend("topleft",c("SOM-C", "Microbial biomass"),lty=1,col=c(1,2),bty="n")
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=TwopMMmodel(t=days,ival=c(972,304) ,Af=3,kb=0.0000001)
Cpools=getC(MMmodel)

matplot(days,Cpools,type="l",ylab="Concentrations",xlab="Days",lty=1,ylim=c(0,max(Cpools)*1.2))
legend("topleft",c("SOM-C", "Microbial biomass"),lty=1,col=c(1,2),bty="n")

plot(Cpools[,2],Cpools[,1],type="l",ylab="SOM-C",xlab="Microbial biomass")

plot(days,Cpools[,2],type="l",col=2,xlab="Days",ylab="Microbial biomass")
```

`TwopParallelModel` *Implementation of a 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 `ParallelModel` that can handle an arbitrary number of pools.

Usage

```
TwopParallelModel(
  t,
  ks,
  C0,
  In,
  gam,
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 2 containing the decomposition rates for the 2 pools.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
gam	A scalar representing the partitioning coefficient, i.e. the proportion from the total amount of inputs that goes to pool 1.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be `euler` or `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

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

See Also

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")
```

TwopParallelModel14 *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

```
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=getF14(Ex)
par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
legend("topright",c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2"),
lty=c(1,1,1),col=c(1,4,4),lwd=c(1,1,2),bty="n")
plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))
```

TwopSeriesModel *Implementation of a two pool model with series structure*

Description

This function creates a model for two pools connected in series. It is a wrapper for the more general function GeneralModel.

Usage

```
TwopSeriesModel(
  t,
  ks,
  a21,
  C0,
  In,
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of length 2 with the values of the decomposition rate for pools 1 and 2.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
solver	A function that solves the system of ODEs. This can be euler or 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

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

See Also

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)
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"),
lty=c(1,1,1),col=c(1,2,4),lwd=c(2,1,1),bty="n")
```

TwopSeriesModel14 *Implementation of a two-pool C14 model with series structure*

Description

This function creates a model for two pools connected in series. It is a wrapper for the more general function GeneralModel_14 that can handle an arbitrary number of pools.

Usage

```
TwopSeriesModel14(
  t,
  ks,
  C0,
  F0_Delta14C,
  In,
  a21,
  xi = 1,
  inputFc,
  lambda = -0.0001209681,
  lag = 0,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

t	A vector containing the points in time where the solution is sought. It must be specified within the same period for which the Delta 14 C of the atmosphere is provided. The default period in the provided dataset C14Atm_NH is 1900-2010.
ks	A vector of length 2 containing the decomposition rates for the 2 pools.
C0	A vector of length 2 containing the initial amount of carbon for the 2 pools.
F0_Delta14C	A vector of length 2 containing the initial amount of the radiocarbon fraction for the 2 pools as Delta14C values in per mil.
In	A scalar or a data.frame object specifying the amount of litter inputs by time.
a21	A scalar with the value of the transfer rate from pool 1 to pool 2.
xi	A scalar or a data.frame specifying the external (environmental and/or edaphic) effects on decomposition rates.
inputFc	A Data Frame object containing values of atmospheric Delta14C per time. First column must be time values, second column must be Delta14C values in per mil.
lambda	Radioactive decay constant. By default lambda=-0.0001209681 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

```
years=seq(1901,2009,by=0.5)
LitterInput=700
#
Ex=TwopSeriesModel14(t=years,ks=c(k1=1/2.8, k2=1/35),
C0=c(200,5000), F0_Delta14C=c(0,0),
In=LitterInput, a21=0.1,inputFc=C14Atm_NH)
R14m=getF14R(Ex)
C14m=getF14C(Ex)
C14t=getF14(Ex)
#
par(mfrow=c(2,1))
plot(C14Atm_NH,type="l",xlab="Year",
ylab="Delta 14C (per mil)",xlim=c(1940,2010))
```

```
lines(years, C14t[,1], col=4)
lines(years, C14t[,2],col=4,lwd=2)
legend("topright",c("Delta 14C Atmosphere", "Delta 14C pool 1", "Delta 14C pool 2"),
lty=c(1,1,1),col=c(1,4,4),lwd=c(1,1,2),bty="n")
#
plot(C14Atm_NH,type="l",xlab="Year",ylab="Delta 14C (per mil)",xlim=c(1940,2010))
lines(years,C14m,col=4)
lines(years,R14m,col=2)
legend("topright",c("Delta 14C Atmosphere","Delta 14C SOM", "Delta 14C Respired"),
lty=c(1,1,1), col=c(1,4,2),bty="n")
par(mfrow=c(1,1))
```

UnBoundInFluxes *automatic title*

Description

automatic title

Usage

```
UnBoundInFluxes(map)
```

Arguments

map	: see method arguments

S4 methods for this generic

- UnBoundInFluxes,function-method

UnBoundInFluxes,function-method *automatic title*

Description

automatic title

Usage

```
## S4 method for signature '`function`'
UnBoundInFluxes(map)
```

Arguments

`map`	object of class:`function`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`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

`UnBoundLinDecompOp,function-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature '`function`'
UnBoundLinDecompOp(matFunc)
```

Arguments

`matFunc`	object of class:`function`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`UnBoundLinDecompOp-class` *automatic title*

Description

automatic title

`UnBoundNonLinDecompOp` *Generic constructor for the class with the same name*

Description

Generic constructor for the class with the same name

Usage

```
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

UnBoundNonLinDecompOp,function,missing,missing,missing,ANY,ANY,ANY-method

automatic title

Description

automatic title

Usage

```
## S4 method for signature '`function`,missing,missing,missing,ANY,ANY,ANY'
UnBoundNonLinDecompOp(matFunc)
```

Arguments

`matFunc`	object of class:`function`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

UnBoundNonLinDecompOp,missing,missing,missing,missing,character,character,UnBoundNonLinDecompOp_by_Po

convert to Indexed version

Description

convert to Indexed version

Usage

```
## S4 method for signature
## '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

`UnBoundNonLinDecompOp,missing,vector,vector,numeric,ANY,ANY,ANY-method` *constructor*

Description

constructor

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

`UnBoundNonLinDecompOp-class` *An S4 class to represent the operation nonlinear nonautonomuous compartmental matrix*

Description

An S4 class to represent the operation nonlinear nonautonomuous compartmental matrix

UnBoundNonLinDecompOp_by_PoolNames

Generic constructor for the class with the same name

Description

Generic constructor for the class with the same name

Usage

```
UnBoundNonLinDecompOp_by_PoolNames(internal_fluxes, out_fluxes, timeSymbol)
```

Arguments

`internal_fluxes`	see method arguments
`out_fluxes`	see method arguments
`timeSymbol`	see method arguments

UnBoundNonLinDecompOp_by_PoolNames,InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character-met

automatic title

Description

automatic title

Usage

```
## S4 method for signature
## 'InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character'
UnBoundNonLinDecompOp_by_PoolNames(internal_fluxes, out_fluxes, timeSymbol)
```

Arguments

`internal_fluxes`	object of class:`InternalFluxList_by_PoolName`, : : no manual documentation
`out_fluxes`	object of class:`OutFluxList_by_PoolName`, : : no manual documentation
`timeSymbol`	object of class:`character`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

UnBoundNonLinDecompOp_by_PoolNames-class

An S4 class to represent the of nonlinear nonautonomuous compartmental system independently of the order of state variables

Description

An S4 class to represent the of nonlinear nonautonomuous compartmental system independently of the order of state variables

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 lenght 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

Tuomi, M., Thum, T., Jarvinen, H., Fronzek, S., Berg, B., Harmon, M., Trofymow, J., Sevanto, S., and Liski, J. (2009). Leaf litter decomposition-estimates of global variability based on Yasso07 model. Ecological Modelling, 220:3362 - 3371.

See Also

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

Examples

```
years=seq(0,50,0.1)
C0=rep(100,5)
In=0

Ex1=Yasso07Model(t=years,C0=C0,In=In)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)

plotCPool(years,Ct,col=1:5,xlab="years",ylab="C pool",
ylim=c(0,max(Ct)))
legend("topright",c("xA","xW","xE","xN","xH"),lty=1,col=1:5,bty="n")

plotCPool(years,Rt,col=1:5,xlab="years",ylab="Respiration",ylim=c(0,50))
legend("topright",c("xA","xW","xE","xN","xH"),lty=1,col=1:5,bty="n")
```

`YassoModel` *Implementation of the Yasso model.*

Description

This function creates a model for seven pools as described in Liski et al. (2005). Model not yet implemented due to lack of data in original publication: values of vector p not completely described in paper. 0.1 was assumed.

Usage

```
YassoModel(
  t,
  ks = c(a_fwl = 0.54, a_cwl = 0.03, k_ext = 0.48, k_cel = 0.3, k_lig = 0.22, k_hum1 =
    0.012, k_hum2 = 0.0012),
  p = c(fwl_ext = 0.1, cwl_ext = 0.1, fwl_cel = 0.1, cwl_cel = 0.1, fwl_lig = 0.1,
    cwl_lig = 0.1, pext = 0.05, pcel = 0.24, plig = 0.77, phum1 = 0.51),
  C0,
  In = c(u_fwl = 0.0758, u_cwl = 0.0866, u_nwl_cnwl_ext = 0.251 * 0.3, u_nwl_cnwl_cel =
    0.251 * 0.3, u_nwl_cnwl_lig = 0.251 * 0.3, 0, 0),
  xi = 1,
  solver = deSolve.lsoda.wrapper,
  pass = FALSE
)
```

Arguments

t	A vector containing the points in time where the solution is sought.
ks	A vector of lenght 7 containing the values of the exposure and decomposition rates for each pool.
p	A vector of containing transfer coefficients among different pools.
C0	A vector containing the initial amount of carbon for the 7 pools. The length of this vector must be 7.
In	A vector of constatn litter inputs.
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

Liski, J., Palosuo, T., Peltoniemi, M., and Sievanen, R. (2005). Carbon and decomposition model Yasso for forest soils. Ecological Modelling, 189:168-182.

See Also

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

Examples

```
years=seq(0,500,0.5)
C0=rep(100,7)
#
Ex1=YassoModel(t=years,C0=C0)
Ct=getC(Ex1)
Rt=getReleaseFlux(Ex1)
#
plotCPool(years,Ct,col=1:7,xlab="years",ylab="C pool",ylim=c(0,200))
legend("topright",c("fwl","cwl","ext","cel","lig","hum1","hum2"),lty=1,col=1:7,bty="n")
#
plotCPool(years,Rt,col=1:7,xlab="years",ylab="Respiration",ylim=c(0,50))
legend("topright",c("fwl","cwl","ext","cel","lig","hum1","hum2"),lty=1,col=1:7,bty="n")
```

`[,Model,character,missing,missing-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'Model,character,missing,missing'
x[i]
```

Arguments

x	object of class:`Model`, : : no manual documentation
i	object of class:`character`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

`[,NlModel,character,ANY,ANY-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel,character,ANY,ANY'
x[i]
```

Arguments

x	object of class:`NlModel`, : : no manual documentation
i	object of class:`character`, : : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove ‘@param‘ tags for parameters that are no longer present in the source code and add ‘@param‘ tags with a default description for yet undocumented parameters. If you remove this ‘@autocomment‘ tag your comments will no longer be touched by the "update_autocomment_roclet".

[[,MCSim-method *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'MCSim'
x[[i]]
```

Arguments

`x`	object of class:`MCSim`, : : no manual documentation
`i`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

[[<-,MCSim-method *automatic title*

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 inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

`$,NlModel-method` *automatic title*

Description

automatic title

Usage

```
## S4 method for signature 'NlModel'
x$name
```

Arguments

`x`	object of class:`NlModel`, : : no manual documentation
`name`	: : no manual documentation inspection of the code. You can use the "update_auto_comment_roclet" to automatically adapt them to changes in the source code. This will remove '@param' tags for parameters that are no longer present in the source code and add '@param' tags with a default description for yet undocumented parameters. If you remove this '@autocomment' tag your comments will no longer be touched by the "update_autocomment_roclet".

Index

∗Topic **datasets**
C14Atm, 39
C14Atm_NH, 40
eCO2, 74
HarvardForest14CO2, 157
Hua2013, 158
incubation_experiment, 161
IntCal09, 179
IntCal13, 180
[,Model,character,missing,missing-method, 276
[,NlModel,character,ANY,ANY-method, 276
[[,MCSim-method, 277
[[<-,MCSim-method, 277
$,NlModel-method, 278

AbsoluteFractionModern, 12
AbsoluteFractionModern,BoundFc-method, 13
AbsoluteFractionModern,ConstFc-method, 13
AbsoluteFractionModern_from_Delta14C, 14
AbsoluteFractionModern_from_Delta14C,matrix-method, 14
AbsoluteFractionModern_from_Delta14C,numeric-method, 15
add_plot, 15
add_plot,TimeMap-method, 16
as.character,TimeMap-method, 16
availableParticleProperties, 17
availableParticleProperties,MCSim-method, 17
availableParticleSets, 18
availableParticleSets,MCSim-method, 18
availableResidentSets, 19
availableResidentSets,MCSim-method, 19
AWBmodel, 20

bacwaveModel, 22, *217*
bind.C14curves, 23
BoundFc, 24
BoundFc,character-method, 25
BoundFc,missing-method, 25
BoundFc-class, 26
BoundInFluxes, 26
BoundInFluxes-class, 26
BoundLinDecompOp, 26
BoundLinDecompOp,ANY-method, 27
BoundLinDecompOp,UnBoundLinDecompOp-method, 27
BoundLinDecompOp-class, 28
by_PoolIndex, 28
by_PoolIndex,ConstantInFluxRate_by_PoolName,ANY,ANY-method 29
by_PoolIndex,ConstantInternalFluxRate_by_PoolName,ANY,ANY- 30
by_PoolIndex,ConstantInternalFluxRateList_by_PoolName,ANY, 29
by_PoolIndex,ConstantOutFluxRate_by_PoolName,ANY,ANY-metho 31
by_PoolIndex,ConstantOutFluxRateList_by_PoolName,ANY,ANY-m 30
by_PoolIndex,function,character,character-method, 31
by_PoolIndex,InFlux_by_PoolName,character,character-method 33
by_PoolIndex,InFluxList_by_PoolName,character,character-me 32
by_PoolIndex,InternalFlux_by_PoolName,character,character- 34
by_PoolIndex,InternalFluxList_by_PoolName,character,charac 33
by_PoolIndex,OutFlux_by_PoolName,character,character-metho 35
by_PoolIndex,OutFluxList_by_PoolName,character,character-m 34
by_PoolIndex,PoolConnection_by_PoolName,ANY,ANY-method,

36
by_PoolName, 36
by_PoolName,ConstantInFluxRate_by_PoolIndex-method,
37
by_PoolName,ConstantInternalFluxRate_by_PoolIndex-method,
38
by_PoolName,ConstantInternalFluxRateList_by_PoolIndex-method,
37
by_PoolName,ConstantOutFluxRate_by_PoolIndex-method,
39
by_PoolName,ConstantOutFluxRateList_by_PoolIndex-method,
38

C14Atm, 39
C14Atm_NH, *39*, 40, *96*, *197*, *235*, *240*, *243*,
257, *262*, *266*
CenturyModel, 40, *217*
CenturyModel_new, 42
check_duplicate_pool_names, 44
check_id_length, 44
check_pool_ids, 45
check_pool_ids,PoolConnection_by_PoolIndex,integer-method,
45
computeResults, 46
computeResults,MCSim-method, 46
computeResults,NlModel-method, 47
ConstantInFlux_by_PoolIndex, *48*, *49*
ConstantInFlux_by_PoolIndex-class, 50
ConstantInFluxList_by_PoolIndex, 47, *48*,
183, *200*
ConstantInFluxList_by_PoolIndex,ConstInFluxes-method,
48
ConstantInFluxList_by_PoolIndex,list-method,
48
ConstantInFluxList_by_PoolIndex,numeric-method,
49
ConstantInFluxList_by_PoolIndex-class,
49
ConstantInFluxList_by_PoolName, 49
ConstantInFluxRate_by_PoolIndex-class,
50
ConstantInFluxRate_by_PoolName-class,
50
ConstantInternalFluxRate_by_PoolIndex,
51, *52*, 53
ConstantInternalFluxRate_by_PoolIndex,missing,missing,character,numeric-method,
54
ConstantInternalFluxRate_by_PoolIndex,numeric,numeric,missing,numeric-method,
54
ConstantInternalFluxRate_by_PoolIndex-class,
55
ConstantInternalFluxRate_by_PoolName,
52, *53*, 55
ConstantInternalFluxRate_by_PoolName,character,character,m
56
ConstantInternalFluxRate_by_PoolName,missing,missing,chara
56
ConstantInternalFluxRate_by_PoolName-class,
57
ConstantInternalFluxRateList_by_PoolIndex,
50, *51*
ConstantInternalFluxRateList_by_PoolIndex,list-method,
51
ConstantInternalFluxRateList_by_PoolIndex,numeric-method,
51
ConstantInternalFluxRateList_by_PoolIndex-class,
52
ConstantInternalFluxRateList_by_PoolName,
52, *53*
ConstantInternalFluxRateList_by_PoolName,list-method,
52
ConstantInternalFluxRateList_by_PoolName-class,
53
ConstantOutFluxRate_by_PoolIndex, *57*,
58, 61
ConstantOutFluxRate_by_PoolIndex,numeric,numeric-method,
61
ConstantOutFluxRate_by_PoolIndex-class,
62
ConstantOutFluxRate_by_PoolName, *59*, *60*
ConstantOutFluxRate_by_PoolName-class,
62
ConstantOutFluxRateList_by_PoolIndex,
57, *58*
ConstantOutFluxRateList_by_PoolIndex,list-method,
57
ConstantOutFluxRateList_by_PoolIndex,numeric-method,
58
ConstantOutFluxRateList_by_PoolIndex-class,
58
ConstantOutFluxRateList_by_PoolName,
59, 59
ConstantOutFluxRateList_by_PoolName,list-method,
59
ConstantOutFluxRateList_by_PoolName,numeric-method,
60
ConstantOutFluxRateList_by_PoolName-class,

60
ConstFc, 62
ConstFc-class, 63
ConstInFluxes, *48*, 63
ConstInFluxes,ConstantInFluxList_by_PoolIndex,numeric-method, 63
ConstInFluxes,numeric,ANY-method, 64
ConstInFluxes-class, 64
ConstLinDecompOp, 65
ConstLinDecompOp,matrix,missing,missing,missing,missing-method, 65
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,ConstantOutFluxRateList_by_PoolIndex 66
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolIndex,missing,numeric,missing-method, 66
ConstLinDecompOp,missing,ConstantInternalFluxRateList_by_PoolName,ConstantOutFluxRateList_by_PoolName,m 67
ConstLinDecompOp,missing,missing,ConstantOutFluxRateList_by_PoolIndex,numeric,missing-method, 67
ConstLinDecompOp-class, 68
ConstLinDecompOp_by_PoolName, 69
ConstLinDecompOpWithLinearScalarFactor, 68
ConstLinDecompOpWithLinearScalarFactor,matrix,missing,missing,missing,ScalarTimeMap-method, 69
ConstLinDecompOpWithLinearScalarFactor-class, 69
cycling, 70

DecompOp, *102*, *188*, *189*, *191*
DecompOp-class, 70
DecompositionOperator-class, 70
Delta14C, 71
Delta14C,BoundFc-method, 71
Delta14C,ConstFc-method, 72
Delta14C_from_AbsoluteFractionModern, 72
Delta14C_from_AbsoluteFractionModern,matrix-method, 73
Delta14C_from_AbsoluteFractionModern,numeric-method, 73
deSolve.lsoda.wrapper, *41*, *43*, 74, *75*, *101*, *102*, *160*, *189*, *191*, *195*, *197*, *203*, *219*, *222*, *224*, *232*, *235*, *238*, *240*, *242*, *244*, *255*, *258*, *261*, *263*, *264*, *266*, *273*, *275*

eCO2, 74
euler, *41*, *43*, 75, *96*, *160*, *195*, *197*, *219*, *222*, *224*, *232*, *235*, *238*, *240*, *242*, *244*, *255*, *258*, *261*, *263*, *264*, *266*, *273*, *275*
example.2DBoundInFluxesFromFunction, 76
example.2DBoundLinDecompOpFromFunction, 76
example.2DConstFc.Args, 76
example.2DConstInFluxesFromVector, 77
example.2DGeneralDecompOpArgs, 77, *189*
example.2DInFluxes.Args, 77, *189*
example.2DUnBoundLinDecompOpFromFunction, 78
example.ConstlinDecompOpFromMatrix, 78
example.nestedTime2DMatrixList, 78
example.Time2DArrayList, 79
example.Time3DArrayList, 79
example.TimeMapFromArray, 79

Fc-class, 79
FcAtm.from.Dataframe, 80
fT.Arrhenius, 80
fT.Century1, 81
fT.Century2, 82
fT.Daycent1, 82
fT.Daycent2, 83
fT.Demeter, 84
fT.KB, 84
fT.LandT, 85
fT.linear, 85
fT.Q10, 86
fT.RothC, 87
fT.Standcarb, 87
fW.Candy, 88
fW.Century, 89
fW.Daycent1, 89
fW.Daycent2, 90
fW.Demeter, 91
fW.Gompertz, 92
fW.Moyano, 92
fW.RothC, 93
fW.Skopp, 93
fW.Standcarb, 94

GaudinskiModel14, 95, *216*
GeneralDecompOp, 97, *101*, *102*, *188*, *189*, *191*
GeneralDecompOp,DecompOp-method, 98
GeneralDecompOp,function-method, 98

GeneralDecompOp,list-method, 99
GeneralDecompOp,matrix-method, 99
GeneralDecompOp,TimeMap-method, 100
GeneralModel, 100, *103*, *195*, *219*, *222*, *231*, *241*, *255*, *264*
GeneralModel_14, *95*, 101, *196*, *223*, *234*, *239*, *243*, *257*, *262*, *265*
GeneralNlModel, 102
GeneralPoolId, 104
GeneralPoolId,character-method, 105
GeneralPoolId,numeric-method, 105
getAccumulatedRelease, 106
getAccumulatedRelease,Model-method, 106
getC, 107, *187*, *207*
getC,Model-method, 107
getC,Model_by_PoolNames-method, 108
getC,NlModel-method, 108
getC14, 109, *187*
getC14,Model_14-method, 109
getCompartmentalMatrixFunc, 110
getCompartmentalMatrixFunc,BoundLinDecompOp,ANY,ANY-method, 110
getCompartmentalMatrixFunc,ConstLinDecompOp,ANY,ANY-method, 111
getCompartmentalMatrixFunc,TransportDecompositionOperator,ANY,ANY-method, 111
getCompartmentalMatrixFunc,UnBoundNonLinDecompOp,ANY,ANY-method, 112
getCompartmentalMatrixFunc,UnBoundNonLinDecompOp_by_PoolNames,character,character-method, 112
getConstantCompartmentalMatrix, 113
getConstantCompartmentalMatrix,ConstLinDecompOp-method, 114
getConstantCompartmentalMatrix,ConstLinDecompOpWithLinearScalarFactor-method, 114
getConstantInFluxVector, 115
getConstantInFluxVector,ConstInFluxes-method, 115
getConstantInternalFluxRateList_by_PoolIndex, 116
getConstantInternalFluxRateList_by_PoolIndex,ConstLinDecompOp-method, 116
getConstantOutFluxRateList_by_PoolIndex, 117
getConstantOutFluxRateList_by_PoolIndex,ConstLinDecompOp-method, 117
getConstLinDecompOp, 118
getConstLinDecompOp,ConstLinDecompOpWithLinearScalarFactor 118
getCumulativeC, 119
getCumulativeC,NlModel-method, 119
getDecompOp, 120
getDecompOp,Model-method, 120
getDecompOp,NlModel-method, 121
getDotOut, 121
getDotOut,TransportDecompositionOperator-method, 122
getF14, 122, *207*
getF14,Model_14-method, 123
getF14C, 123
getF14C,Model_14-method, 124
getF14R, 124
getF14R,Model_14-method, 125
getFormat, 125
getFormat,Fc-method, 126
getFunctionDefinition, 126
getFunctionDefinition,ConstInFluxes-method, 127
getFunctionDefinition,ConstLinDecompOp-method, 128
getFunctionDefinition,ConstLinDecompOpWithLinearScalarFact 128
getFunctionDefinition,DecompositionOperator-method, 129
getFunctionDefinition,InFluxList_by_PoolIndex-method, 129
getFunctionDefinition,InFluxList_by_PoolName-method, 130
getFunctionDefinition,StateDependentInFluxVector-method, 130
getFunctionDefinition,TimeMap-method, 131
getFunctionDefinition,TransportDecompositionOperator-metho 131
getFunctionDefinition,UnBoundInFluxes-method, 132
getFunctionDefinition,UnBoundLinDecompOp-method, 132
getInFluxes, 133
getInFluxes,Model-method, 133
getInFluxes,NlModel-method, 134
getInitialValues, 134
getInitialValues,NlModel-method, 135
getLinearScaleFactor, 135
getLinearScaleFactor,ConstLinDecompOpWithLinearScalarFacto

136
getMeanTransitTime, 136
getMeanTransitTime,ConstLinDecompOp-method, 137
getNumberOfPools, 137
getNumberOfPools,MCSim-method, 138
getNumberOfPools,NlModel-method, 138
getNumberOfPools,TransportDecompositionOperator-method, 139
getOutputFluxes, 139
getOutputFluxes,NlModel-method, 140
getOutputReceivers, 140
getOutputReceivers,TransportDecompositionOperator,numeric-method, 141
getParticleMonteCarloSimulator, 141
getParticleMonteCarloSimulator,NlModel-method, 142
getReleaseFlux, 142, *187*, *207*
getReleaseFlux,Model-method, 143
getReleaseFlux,Model_by_PoolNames-method, 143
getReleaseFlux,NlModel-method, 144
getReleaseFlux14, 144
getReleaseFlux14,Model_14-method, 145
getRightHandSideOfODE, 145
getRightHandSideOfODE,Model-method, 146
getRightHandSideOfODE,Model_by_PoolNames-method, 146
getTimeRange, 147
getTimeRange,ConstInFluxes-method, 147
getTimeRange,ConstLinDecompOp-method, 148
getTimeRange,ConstLinDecompOpWithLinearScalarFactor-method, 148
getTimeRange,DecompositionOperator-method, 149
getTimeRange,TimeMap-method, 149
getTimeRange,UnBoundInFluxes-method, 150
getTimeRange,UnBoundLinDecompOp-method, 150
getTimes, 151
getTimes,Model-method, 151
getTimes,NlModel-method, 152
getTransferCoefficients, 152
getTransferCoefficients,NlModel-method, 153
getTransferCoefficients,TransportDecompositionOperator-met
153
getTransferMatrix, 154
getTransferMatrixFunc, 154
getTransferMatrixFunc,TransportDecompositionOperator-metho
155
getTransitTimeDistributionDensity, 155
getTransitTimeDistributionDensity,ConstLinDecompOp-method, 156
getValues, 156
getValues,ConstFc-method, 157

HarvardForest14CO2, 157
Hua2013, *24*, *39*, 158, *254*

ICBMModel, 159, *216*
incubation_experiment, 161
InFlux, 162
InFlux_by_PoolIndex, 168
InFlux_by_PoolIndex,function,numeric-method, 169
InFlux_by_PoolIndex-class, 169
InFlux_by_PoolName, *169*, 170
InFlux_by_PoolName,function,character-method, 170
InFlux_by_PoolName-class, 171
InFluxes, *102*, *163*, 163, *189*, *191*
InFluxes,ConstantInFluxList_by_PoolIndex-method, 163
InFluxes,function-method, 164
InFluxes,InFluxes-method, 164
InFluxes,list-method, 165
InFluxes,numeric-method, 165
InFluxes,TimeMap-method, 166
InFluxes-class, 166
InFluxList_by_PoolIndex, 166
InFluxList_by_PoolIndex,list-method, 167
InFluxList_by_PoolIndex-class, 167
InFluxList_by_PoolName, 167
InFluxList_by_PoolName,list-method, 168
InFluxList_by_PoolName-class, 168
initialize,ConstLinDecompOp-method, 171
initialize,DecompositionOperator-method, 171
initialize,MCSim-method, 172
initialize,Model-method, 173

initialize,Model_14-method, 174
initialize,Model_by_PoolNames-method, 175
initialize,NlModel-method, 176
initialize,TimeMap-method, 177
initialize,TransportDecompositionOperator-method, 177
initialize,UnBoundInFluxes-method, 178
initialize,UnBoundLinDecompOp-method, 179
IntCal09, *24*, 179, *180*
IntCal13, *24*, 180
InternalFlux_by_PoolIndex, 184
InternalFlux_by_PoolIndex,function,numeric,numeric,missing-method, 184
InternalFlux_by_PoolIndex-class, 185
InternalFlux_by_PoolName, *183*, *185*, 185
InternalFlux_by_PoolName,function,character,character,missing-method, 186
InternalFlux_by_PoolName,function,missing,missing,character-method, 186
InternalFlux_by_PoolName-class, 187
InternalFluxList_by_PoolIndex, 182
InternalFluxList_by_PoolIndex,list-method, 182
InternalFluxList_by_PoolIndex-class, 182
InternalFluxList_by_PoolName, 183
InternalFluxList_by_PoolName,list-method, 183
InternalFluxList_by_PoolName-class, 184

linesCPool, 187
list, *158*
listProduct, 187

MCSim-class, 188
Model, *23*, *42*, *43*, *96*, *100*, *160*, *188*, 188, *189*, *195*, *220*, *223*, *224*, *230*, *232*, *238*, *242*, *256*, *259*, *261*, *264*, *274*, *275*
Model-class, 190
Model_14, *101*, 191, *197*, *235*, *240*, *244*, *258*, *263*, *266*
Model_14-class, 194
Model_by_PoolNames (Model_by_PoolNames-class), 194
Model_by_PoolNames-class, 194

NlModel-class, 195

OnepModel, 195, *216*
OnepModel14, 196, *216*, *254*
optimize, *254*
OutFlux, 198
OutFlux_by_PoolIndex, 200
OutFlux_by_PoolIndex,function,numeric-method, 201
OutFlux_by_PoolIndex-class, 201
OutFlux_by_PoolName, *200*, *201*, 202
OutFlux_by_PoolName,function,character-method, 202
OutFlux_by_PoolName-class, 203
OutFluxList_by_PoolIndex, 198
OutFluxList_by_PoolIndex,list-method, 199
OutFluxList_by_PoolIndex-class, 199
OutFluxList_by_PoolName, 199
OutFluxList_by_PoolName,list-method, 200
OutFluxList_by_PoolName-class, 200

ParallelModel, 203, *238*, *260*
plot,MCSim-method, 204
plot,Model-method, 205
plot,NlModel-method, 206
plot,TimeMap-method, 206
plotC14Pool, 207
plotCPool, 207
plotPoolGraph, 208
plotPoolGraph,Model-method, 208
PoolConnection_by_PoolIndex, 209
PoolConnection_by_PoolIndex,ANY,ANY,missing-method, 209
PoolConnection_by_PoolIndex,missing,missing,character-meth
210
PoolConnection_by_PoolIndex-class, 210
PoolConnection_by_PoolName, 210
PoolConnection_by_PoolName,ANY,ANY,missing-method, 211
PoolConnection_by_PoolName-class, 211
PoolId-class, 212
PoolIndex, 212
PoolIndex,character-method, 212
PoolIndex,numeric-method, 213
PoolIndex,PoolIndex-method, 213
PoolIndex,PoolName-method, 214
PoolIndex-class, 214

PoolName, 214
PoolName,character-method, 215
PoolName,PoolIndex-method, 215
PoolName,PoolName-method, 216
PoolName-class, 216
predefinedModels, *23*, *42*, *43*, *96*, *160*, *189*, *195*, *197*, 216, *220*, *223*, *224*, *230*, *232*, *235*, *238*, *240*, *242*, *244*, *256*, *258*, *259*, *261*, *263*, *264*, *266*, *274*, *275*
print,Model-method, 217
print,NlModel-method, 218

RespirationCoefficients, 218
RothCModel, *42*, *43*, *216*, 219

ScalarTimeMap, *41*, 220
ScalarTimeMap,missing,missing,missing,missing,numeric-method, 221
ScalarTimeMap-class, 222
SeriesLinearModel, *217*, 222
SeriesLinearModel14, *217*, 223
show,Model-method, 225
show,NlModel-method, 226
SoilR.F0.new, 226
StateDependentInFluxVector (*StateDependentInFluxVector-class*), 227
StateDependentInFluxVector-class, 227
StateIndependentInFlux_by_PoolIndex, *227*, *228*
StateIndependentInFlux_by_PoolIndex (*StateIndependentInFlux_by_PoolIndex-class*), 228
StateIndependentInFlux_by_PoolIndex-class, 228
StateIndependentInFluxList_by_PoolIndex, *227*, *228*
StateIndependentInFluxList_by_PoolIndex,list-method, 227
StateIndependentInFluxList_by_PoolIndex-class, 228
StateIndependentInFluxList_by_PoolName, 228
summary,Model-method, 229
systemAge, *70*, 229, *253*

ThreepairMMmodel, *217*, 230
ThreepFeedbackModel, *216*, 231
ThreepFeedbackModel14, *216*, 234
ThreepParallelModel, *203*, *216*, 238
ThreepParallelModel14, *216*, 239
ThreepSeriesModel, *216*, 241
ThreepSeriesModel14, *217*, 243
TimeMap, *26*, *43*, 245
TimeMap,data.frame,missing,missing,missing,missing-method, 246
TimeMap,function,missing,missing,missing,missing-method, 246
TimeMap,function,numeric,numeric,missing,missing-method, 247
TimeMap,list,missing,missing,missing,missing-method, 247
TimeMap,missing,missing,missing,numeric,array-method, 248
TimeMap,missing,missing,missing,numeric,list-method, 248
TimeMap,missing,missing,missing,numeric,matrix-method, 249
TimeMap,missing,missing,missing,numeric,numeric-method, 250
TimeMap,TimeMap,ANY,ANY,ANY,ANY-method, 250
TimeMap-class, *245*, 251
TimeMap.from.Dataframe, 251
TimeMap.new, *103*, *203*, 252
TimeRangeIntersection, 252
transitTime, *230*, 253
TransportDecompositionOperator-class, 253
turnoverFit, 254
TwopFeedbackModel, *101*, *102*, *192*, *217*, 255
TwopFeedbackModel14, *217*, 257
TwopMMmodel, *217*, 258
TwopParallelModel, *101*, *102*, *192*, *203*, *217*, 260
TwopParallelModel14, *217*, 262
TwopSeriesModel, *101*, *102*, *192*, *217*, 264
TwopSeriesModel14, *217*, 265

UnBoundInFluxes, 267
UnBoundInFluxes,function-method, 267
UnBoundInFluxes-class, 268
UnBoundLinDecompOp, 268
UnBoundLinDecompOp,function-method, 268
UnBoundLinDecompOp-class, 269
UnBoundNonLinDecompOp, 269

UnBoundNonLinDecompOp,function,missing,missing,missing,ANY,ANY,ANY-method,
270
UnBoundNonLinDecompOp,missing,missing,missing,missing,character,character,UnBoundNonLinDecompOp_by_Pool
270
UnBoundNonLinDecompOp,missing,vector,vector,numeric,ANY,ANY,ANY-method,
271
UnBoundNonLinDecompOp-class, 271
UnBoundNonLinDecompOp_by_PoolNames,
272
UnBoundNonLinDecompOp_by_PoolNames,InternalFluxList_by_PoolName,OutFluxList_by_PoolName,character-metho
272
UnBoundNonLinDecompOp_by_PoolNames-class,
273

Yasso07Model, *217*, 273
YassoModel, *217*, 274