Package ‘RxODE’

October 14, 2017

Version 0.6-1
Date 2017-10-09
Title Facilities for Simulating from ODE-Based Models
Author Matthew L. Fidler [aut], Melissa Hallow [aut], Wenping Wang [aut, cre]
Maintainer Wenping Wang <wwang8198@gmail.com>
Depends R (>= 3.2.0)
Suggests knitr, nlme, shiny, tcltk, testthat, devtools, covr,
markdown, SnakeCharmR, rSymPy, dplyr, tidyr, tibble, curl
Imports utils, methods, digest, rex, dparser (>= 0.1.7), brew,
memoise, magrittr, Rcpp (>= 0.12.3), inline, Matrix, R.utils
Description Facilities for running simulations from ordinary
differential equation (ODE) models, such as pharmacometrics and other
compartmental models. A compilation manager translates the ODE model
into C, compiles it, and dynamically loads the object code into R for
improved computational efficiency. An event table object facilitates
the specification of complex dosing regimens (optional) and sampling
schedules. NB: The use of this package requires both C and
Fortran compilers, for details on their use with R please see
Section 6.3, Appendix A, and Appendix D in the “R Administration and
Installation” manual. Also the code is mostly released under GPL. The
VODE and LSODA are in the public domain. The information is available
in the inst/COPYRIGHTS.
NeedsCompilation yes
VignetteBuilder knitr
SystemRequirements GNU make
License GPL (>= 2)
URL https://www.r-project.org,
https://github.com/nlmixrdevelopment/RxODE
RoxygenNote 6.0.1
LinkingTo dparser (>= 0.1.7), Rcpp (>= 0.12.3), RcppArmadillo (>=
0.5.600.2.0)
## R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.dosing</td>
<td>3</td>
</tr>
<tr>
<td>add.sampling</td>
<td>4</td>
</tr>
<tr>
<td>arrange_</td>
<td>5</td>
</tr>
<tr>
<td>coef.RxODE</td>
<td>5</td>
</tr>
<tr>
<td>distinct_</td>
<td>6</td>
</tr>
<tr>
<td>eventTable</td>
<td>6</td>
</tr>
<tr>
<td>filter_</td>
<td>9</td>
</tr>
<tr>
<td>gather_</td>
<td>9</td>
</tr>
<tr>
<td>genShinyApp.template</td>
<td>10</td>
</tr>
<tr>
<td>group_by_</td>
<td>11</td>
</tr>
<tr>
<td>mutate_</td>
<td>12</td>
</tr>
<tr>
<td>predict.RxODE</td>
<td>12</td>
</tr>
<tr>
<td>print.rxCoeffSolve</td>
<td>13</td>
</tr>
<tr>
<td>print.RxODE</td>
<td>13</td>
</tr>
<tr>
<td>rename_</td>
<td>14</td>
</tr>
<tr>
<td>rxAddReturn</td>
<td>14</td>
</tr>
<tr>
<td>rxChain</td>
<td>15</td>
</tr>
<tr>
<td>rxClean</td>
<td>15</td>
</tr>
<tr>
<td>rxCompile</td>
<td>16</td>
</tr>
<tr>
<td>rxCoutEcho</td>
<td>18</td>
</tr>
<tr>
<td>rxDelete</td>
<td>18</td>
</tr>
<tr>
<td>rxDfdy</td>
<td>19</td>
</tr>
<tr>
<td>rxDllLoaded</td>
<td>19</td>
</tr>
<tr>
<td>rxInv</td>
<td>20</td>
</tr>
<tr>
<td>rxKahanSum</td>
<td>20</td>
</tr>
<tr>
<td>rxLhs</td>
<td>21</td>
</tr>
<tr>
<td>rxLoad</td>
<td>21</td>
</tr>
<tr>
<td>rxLogifyModel</td>
<td>22</td>
</tr>
<tr>
<td>rxNeumaierSum</td>
<td>22</td>
</tr>
<tr>
<td>rxNorm</td>
<td>23</td>
</tr>
<tr>
<td>RxODE</td>
<td>23</td>
</tr>
<tr>
<td>rxOptions</td>
<td>28</td>
</tr>
<tr>
<td>rxPairwiseSum</td>
<td>29</td>
</tr>
<tr>
<td>rxParams</td>
<td>30</td>
</tr>
<tr>
<td>rxPermissive</td>
<td>30</td>
</tr>
<tr>
<td>rxProd</td>
<td>31</td>
</tr>
<tr>
<td>rxPythonFsum</td>
<td>32</td>
</tr>
<tr>
<td>rxRmPrint</td>
<td>33</td>
</tr>
<tr>
<td>rxRmSens</td>
<td>33</td>
</tr>
<tr>
<td>rxRtoolsBaseWin</td>
<td>34</td>
</tr>
<tr>
<td>rxSetProd</td>
<td>34</td>
</tr>
<tr>
<td>rxSetSum</td>
<td>35</td>
</tr>
</tbody>
</table>
**add.dosing**

*Add dosing to eventTable*

**Description**

This adds a dosing event to the event table. This is provided for piping syntax through magrittr.

**Usage**

```
add.dosing(eventTable, ...)
```

**Arguments**

- `eventTable` eventTable object
- `...` arguments sent to `eventTable$add.dosing`.
add.sampling

Value

eventTable with updated dosing (note the event table will be updated anyway)

Author(s)

Matthew L. Fidler

See Also

eventTable, RxODE

Add sampling to eventTable

Description

This adds a dosing event to the event table. This is provided for piping syntax through magrittr

Usage

add.sampling(eventTable, time, time.units = NA)

Arguments

eventTable An eventTable object
time a vector of time values (in time.units).
time.units an optional string specifying the time units. Defaults to the units specified when the EventTable was initialized.

Value

eventTable with updated sampling. (Note the event table will be updated even if you don’t reassign the eventTable)

Author(s)

Matthew L. Fidler

See Also

eventTable, RxODE
Description

compatibility function for dplyr

Usage

```r
## S3 method for class 'solveRxDll'
arrange_(.data, ...)
```

Arguments

- `.data` Solved equation, an solveRxDll object.
- `...` Additional arguments

_coef.RxODE_ Return the RxODE coefficients_

Description

This returns the parameters, state variables

Usage

```r
## S3 method for class 'RxODE'
coef(object, ...)

## S3 method for class 'RxCompilationManager'
coef(...) 

## S3 method for class 'solveRxODE'
coef(object, ...)

## S3 method for class 'rxDll'
coef(...) 
```

Arguments

- `object` is an RxODE object
- `...` ignored arguments
eventTable

Value

- a rxCoef object with the following
  - params is a list of strings for parameters for the RxODE object
  - state is a list of strings for the names of each state in the RxODE object.
  - ini is the model specified default values for the parameters.
  - RxODE is the referring RxODE object

Author(s)

Matthew L. Fidler

---

distinct_ for solveRxDll object

Description

compatability function for dplyr

Usage

```r
## S3 method for class 'solveRxDll'
distinct_(.data, ...)
```

Arguments

- `.data` Solved equation, an solveRxDll object.
- `...` Additional arguments

---

Create an event table object

Description

Initializes an object of class ‘EventTable’ with methods for adding and querying dosing and observation records

Usage

```r
eventTable(amount.units = NA, time.units = "hours")
```
Arguments

amount.units string denoting the amount dosing units, e.g., "mg", "ug". Default to NA to denote unspecified units. It could also be a solved RxODE object. In that case, eventTable(obj) returns the eventTable that was used to solve the RxODE object.

time.units string denoting the time units, e.g., "hours", "days". Default to "hours".

An eventTable is an object that consists of a data.frame storing ordered time-stamped events of an (unspecified) PK/PD dynamic system, units (strings) for dosing and time records, plus a list of functions to add and extract event records. Currently, events can be of two types: dosing events that represent inputs to the system and sampling time events that represent observations of the system with ‘amount.units’ and ‘time.units’, respectively. In the future, additional events may include resetting of state variables (compartments), for instance, to indicate time after “wash-out”, etc.

Value

A closure with the following list of functions:

get.EventTable returns the current event table.

add.dosing adds dosing records to the event table. Its arguments are
dose: numeric scalar, dose amount in amount.units;
nbr.doses: integer, number of doses;
dosing.interval: required numeric scalar, time between doses in time.units, defaults to 24 of time.units="hours";
dosing.to: integer, compartment the dose goes into (first compartment by default);
rate: for infusions, the rate of infusion (default is NULL, for bolus dosing);
start.time: required dosing start time;
do.sampling: logical, should observation sampling records be added at the dosing times? Defaults to FALSE.
amount.units: optional string indicating the dosing units. Defaults to NA to indicate as per the original EventTable definition.
time.units: optional string indicating the time units. Defaults to "hours" to indicate as per the original EventTable definition.

get.dosing returns a data.frame of dosing records.

clear.dosing clears or deletes all dosing from event table

add.sampling adds sampling time observation records to the event table. Its arguments are
time: a vector of time values (in time.units).
time.units an optional string specifying the time units. Defaults to the units specified when the EventTable was initialized.

get.sampling returns a data.frame of sampled observation records.

clear.sampling removes all sampling from event table.
get.obs.rec returns a logical vector indicating whether each event record represents an observation or not.
get.nobs returns the number of observation (not dosing) records.
get.units returns a two-element character vector with the dosing and time units, respectively.
copy makes a copy of the current event table. To create a copy of an event table object use qd2 <- qd$copy().

Author(s)
Melissa Hallow and Wenping Wang

See Also
RxODE

Examples

# create dosing and observation (sampling) events
# QD 50mg dosing, 5 days followed by 25mg 5 days
#
qd <- eventTable(amount.units = "mg", time.units = "days")
# qd$dadd.dosing(dose=50, nbr.doses=5, dosing.interval = 1, do.sampling=FALSE)
#
# sample the system's drug amounts hourly the first day, then every 12 hours
# for the next 4 days
qd$dadd.sampling(seq(from = 0, to = 1, by = 1/24))
qd$dadd.sampling(seq(from = 1, to = 5, by = 12/24))
#
# print(qd$get.dosing()) # table of dosing records
print(qd$get.nobs()) # number of observation (not dosing) records
#
# BID dosing, 5 days
bid <- eventTable("mg", "days") # only dosing
bid$dadd.dosing(dose=10000, nbr.doses=2*5,
    dosing.interval = 12, do.sampling=FALSE)
#
# Use the copy() method to create a copy (clone) of an existing
# event table (simple assignments just create a new reference to
# the same event table object (closure)).
#
bid.ext <- bid$copy() # three-day extension for a 2nd cohort
bid.ext$dadd.dosing(dose = 5000, nbr.doses = 2*3,
    start.time = 120, dosing.interval = 12, do.sampling = FALSE)
#
# You can also use the Piping operator to create a table

qd2 <- eventTable(amount.units="mg", time.units="days") %>%
    add.dosing(dose=50, nbr.doses=5, dosing.interval=1, do.sampling=FALSE) %>%
    add.sampling(seq(from=0, to=1, by=1 / 24)) %>%
add.sampling(seq(from=1, to=5, by=12 / 24))
# print(qd2$dosing())  # table of dosing records
print(qd2$get.nobs())  # number of observation (not dosing) records

# Note that piping with %>% will update the original table.
qd3 <- qd2 %>% add.sampling(seq(from=5, to=10, by=6 / 24))
print(qd2$get.nobs())
print(qd3$get.nobs())

---

**filter_**

*filter_for solveRxDll object*

**Description**

compatibility function for dplyr

**Usage**

```r
## S3 method for class 'solveRxDll'
filter_(data, ...)
```

**Arguments**

- `.data`  Solved equation, an solveRxDll object.
- `...`  Additional arguments

---

**gather_**

*gather_for solveRxDll object*

**Description**

compatibility function for tidyr

**Usage**

```r
## S3 method for class 'solveRxDll'
gather_(data, ...)
```

**Arguments**

- `.data`  Solved ODE, an solveRxDll object.
- `...`  Additional arguments
Generate an example (template) of a dosing regimen shiny app

Description

Create a complete shiny application for exploring dosing regimens given a (hardcoded) PK/PD model.

Usage

```r
genShinyApp.template(appDir = "shinyExample", verbose = TRUE,
    ODE.config = list(ode = "model", params = c(ka = 0.294), inits = c(eff = 1),
    stiff = TRUE, atol = 1e-08, rtol = 1e-06))

write.template.server(appDir)
write.template.ui(appDir, statevars)
```

Arguments

- `appDir` a string with a directory where to store the shiny app, by default is "shinyExample". The directory appDir will be created if it does not exist.
- `verbose` logical specifying whether to write messages as the shiny app is generated. Defaults to TRUE.
- `ODE.config` model name compiled and list of parameters sent to `rxSolve`.
- `statevars` List of statevars passed to to the `write.template.ui` function. This usually isn’t called directly.

A PK/PD model is defined using `rxode`, and a set of parameters and initial values are defined. Then the appropriate R scripts for the shiny’s user interface `ui.R` and the server logic `server.R` are created in the directory appDir.

The function evaluates the following PK/PD model by default:

```r
C2 = centr/V2;
C3 = peri/V3;
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
d/dt(peri) = Q*C2 - Q*C3;
d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
```

This can be changed by the `ODE.config` parameter.

To launch the shiny app, simply issue the `runApp(appDir)` R command.

Value

None, these functions are used for their side effects.
### group_by_

**Note**

These functions create a simple, but working example of a dosing regimen simulation web application. Users may want to modify the code to experiment creating shiny applications for their specific RxODE models.

**See Also**

RxODE, eventTable, and the package shiny (shiny.rstudio.com).

**Examples**

```r
## Not run:
# create the shiny app example (template)
genShinyApp.template(appDir = "myapp")
# run the shiny app
runApp("myapp")

## End(Not run)
```

---

#### group_by_

**group_by_ for solveRxDll object**

**Description**

compatbility function for dplyr

**Usage**

```r
## S3 method for class 'solveRxDll'
group_by_(.data, ...)
```

**Arguments**

- `.data`  
  Solved equation, an solveRxDll object.
- `...`  
  Additional arguments
predictrxode

**Description**
compatibility function for dplyr

**Usage**
```r
## S3 method for class 'solveRxDll'
mutate_(.data, ...)
```

**Arguments**
- `.data` Solved equation, an `solveRxDll` object.
- `...` Additional arguments

---

predict.RxODE

**Predict an RxODE object**

**Description**
predict solves the ordinary differential equations specified by RxODE object

**Usage**
```r
## S3 method for class 'RxODE'
predict(object, ...)
```

**Arguments**
- `object` An RxODE object
- `...` Solve arguments sent to `rxSolve`. See `rxSolve`.

**Author(s)**
Matthew L. Fidler
**print.rxCoefSolve**

*Print the rxCoefSolve object*

---

**Description**

This prints out the user supplied arguments for the rxCoef object.

**Usage**

```r
## S3 method for class 'rxCoefSolve'
print(x, ...)
```

**Arguments**

- `x` : rxCoefSolve object
- `...` : Other (ignored) parameters.

**Author(s)**

Matthew L.Fidler

---

**print.RxODE**

*Print information about the RxODE object.*

---

**Description**

This prints the model name and its status for being able to be solved.

**Usage**

```r
## S3 method for class 'RxODE'
print(x, ...)
```

```r
## S3 method for class 'RxCompilationManager'
print(x, ...)
```

**Arguments**

- `x` : An rxode object
- `...` : Ignored parameters

**Author(s)**

Matthew L.Fidler
rename_  

rename_for solveRxDll object

Description
compatability function for dplyr

Usage

```r
## S3 method for class 'solveRxDll'
rename_(.data, ...)
```

Arguments

- `.data` Solved equation, an solveRxDll object.
- `...` Additional arguments

rxAddReturn  

Add a return statement to a function.

Description
Add a return statement to a function.

Usage

```r
rxAddReturn(fn, ret = TRUE)
```

Arguments

- `fn` Function to deparse
- `ret` boolean stating if a return statement will be added.

Value

Function with parens removed and add a return statement.

Author(s)

Matthew L. Fidler
**rxChain**

**Description**

Add item to solved system of equations

**Usage**

```r
rxChain(obj1, obj2)
```

```r
## S3 method for class 'solveRxDll'
obj1 + obj2
```

**Arguments**

- `obj1` Solved object.
- `obj2` New object to be added/piped/chained to solved object.

**Value**

When `newObject` is an event table, return a new solved object with the new event table.

**Author(s)**

Matthew L. Fidler

---

**rxClean**

**Cleanup anonymous DLLs**

**Description**

This cleans up any DLLs created by text files

**Usage**

```r
rxClean(wd = getwd())
```

**Arguments**

- `wd` What directory should be cleaned

  This cleans up all files named `rx-*`, `dll` and associated files as well as `call_dvode.o` and associated files
Value

TRUE if successful

Author(s)

Matthew L. Fidler

---

rxCompile

Compile a model if needed

Description

This is the compilation workhorse creating the RxODE model DLL files.

Usage

rxCompile(model, dir, prefix, extraC = NULL, force = FALSE,
    modName = NULL, calcJac = NULL, calcSens = NULL,
    collapseModel = FALSE, ...)

## S3 method for class 'character'

rxCompile(model, dir, prefix = NULL, extraC = NULL,
    force = FALSE, modName = NULL, calcJac = NULL, calcSens = NULL,
    collapseModel = FALSE, ...)

## S3 method for class 'rxDll'

rxCompile(model, ...)

## S3 method for class 'RxODE'

rxCompile(model, ...)

Arguments

- **model**
  
  This is the ODE model specification. It can be:
  
  - a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.
  - a file name where the ODE system equation is contained
  - An ODE expression enclosed in \{\}  
  (see also the filename argument). For details, see the sections “Details” and “RxODE Syntax” below.

- **dir**
  
  This is the model directory where the C file will be stored for compiling. If unspecified, the C code is stored in a temporary directory, then the model is compiled and moved to the current directory. Afterwards the C code is removed. If specified, the C code is stored in the specified directory and then compiled in that directory. The C code is not removed after the DLL is created in the same directory. This can be useful to debug the c-code outputs.
prefix is a string indicating the prefix to use in the C based functions. If missing, it is calculated based on file name, or md5 of parsed model.

extraC Extra C code to include in the model. This can be useful to specify functions in the model. These C functions should usually take double precision arguments, and return double precision values.

force is a boolean stating if the (re)compile should be forced if RxODE detects that the models are the same as already generated.

modName a string to be used as the model name. This string is used for naming various aspects of the computations, including generating C symbol names, dynamic libraries, etc. Therefore, it is necessary that modName consists of simple ASCII alphanumeric characters starting with a letter.

calcJac boolean indicating if RxODE will calculate the Jacobian according to the specified ODEs.

calcSens boolean indicating if RxODE will calculate the sensitivities according to the specified ODEs.

collapseModel boolean indicating if RxODE will remove all LHS variables when calculating sensitivities.

... Other arguments sent to the rxTrans function.

Value

An rxDll object that has the following components

- dll DLL path
- model model specification
- .c A function to call C code in the correct context from the DLL using the .C function.
- .call A function to call C code in the correct context from the DLL using the .Call function.
- args A list of the arguments used to create the rxDll object.

Author(s)

Matthew L.Fidler

See Also

RxODE
**rxCoutEcho**

*Echo cout to console for a number*

**Description**

Echo cout to console for a number

**Usage**

```plaintext
rxCoutEcho(number)
```

**Arguments**

- **number**: number to output
  - @return nothing.

---

**rxDelete**

*Delete the DLL for the model*

**Description**

This function deletes the DLL, but doesn’t delete the model information in the object.

**Usage**

```plaintext
rxDelete(obj)
```

**Arguments**

- **obj**: RxODE family of objects

**Value**

A boolean stating if the operation was successful.

**Author(s)**

Matthew L. Fidler
**rxdfdy**

**Jacobain and parameter derivates**

---

**Description**

Return Jacobain and parameter derivates

**Usage**

rxdfdy(obj)

**Arguments**

- **obj**
  - RxODE family of objects

**Value**

A list of the jacobian parameters defined in this RxODE object.

**Author(s)**

Matthew L. Fidler

---

**rxdllloaded**

**Determine if the rxDll is loaded or not.**

---

**Description**

Determine if the rxDll is loaded or not.

**Usage**

rxdllloaded(x, retry = TRUE)

**Arguments**

- **x**
  - is a RxODE family of objects
- **retry**
  - is a flag to retry to load if the function can’t determine if the object is loaded or not...

**Value**

a boolean stating if the DLL is loaded

**Author(s)**

Matthew L. Fidler
**rxInv**

*Invert matrix using Rcpp Armadilo.*

**Description**

Invert matrix using Rcpp Armadilo.

**Usage**

```r
rxInv(matrix)
```

**Arguments**

- `matrix`: matrix to be inverted.

**Value**

inverse or pseudo inverse of matrix.

---

**rxKahanSum**

*Using the Kahan method, take a more accurate sum*

**Description**

Using the Kahan method, take a more accurate sum

**Usage**

```r
rxKahanSum(numbers)
```

**Arguments**

- `numbers`: A vector of numbers to sum.

**Value**

Sum of numbers

**References**

https://en.wikipedia.org/wiki/Kahan_summation_algorithm
**rxLhs**  
*Left handed Variables*

---

**Description**

This returns the model calculated variables

**Usage**

```r
rxLhs(obj)
```

**Arguments**

- `obj`  
  RxODE family of objects

**Value**

- a character vector listing the calculated parameters

**Author(s)**

Matthew L. Fidler

**See Also**

- `RxODE`

---

**rxLoad**  
*Load the DLL for the object*

---

**Description**

This loads the DLL into the current R session to allow C functions to be called in R.

**Usage**

```r
rxLoad(obj)
```

**Arguments**

- `obj`  
  a RxODE family of objects

**Author(s)**

Matthew L. Fidler
rxLogifyModel  Takes a model and expands it to log multiplication

Description
This is a numerical trick to help reduce multiplication errors when numbers are of very different magnitude.

Usage
rxLogifyModel(model, expand = TRUE)

Arguments
- **model**: RxODE model
- **expand**: Expand the symbolic expression using SymPy?

Value
Lines for expanded model. (but not compiled)

Author(s)
Matthew L. Fidler

rxNeumaierSum  Using the Neumaier method, take a more accurate sum

Description
Using the Neumaier method, take a more accurate sum

Usage
rxNeumaierSum(numbers)

Arguments
- **numbers**: A vector of numbers to sum.

Value
Sum of numbers, a bit more accurate than rxKahanSum

References
https://en.wikipedia.org/wiki/Kahan_summation_algorithm
**rxNorm**  
*Get the normalized model*

**Description**

This get the syntax preferred model for processing

**Usage**

```
rxNorm(obj, condition = NULL, removeInis, removeJac, removeSens)
```

**Arguments**

- `obj`: RxODE family of objects
- `condition`: Character string of a logical condition to use for subsetting the normalized model. When missing, and a condition is not set via `rxCondition`, return the whole code with all the conditional settings intact. When a condition is set with `rxCondition`, use that condition.
- `removeInis`: A boolean indicating if paramter initializations will be removed from the model
- `removeJac`: A boolean indicating if the Jacobians will be removed.
- `removeSens`: A boolean indicating if the sensitivities will be removed.

**Value**

Normalized Normal syntax (no comments)

**Author(s)**

Matthew L. Fidler

---

**RxODE**  
*Create an ODE-based model specification*

**Description**

Create a dynamic ODE-based model object suitably for translation into fast C code

**Usage**

```
RxODE(model, modName = basename(wd), wd = ifelse(RxODE.cache.directory == ",", getwd(), RxODE.cache.directory), filename = NULL, do.compile = NULL, 
extraC = NULL, debug = FALSE, calcJac = NULL, calcSens = NULL, 
collapseModel = FALSE, ...)
```
Arguments

Model

This is the ODE model specification. It can be:

- a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.
- a file name where the ODE system equation is contained
- An ODE expression enclosed in \{}

(see also the filename argument). For details, see the sections “Details” and “RxODE Syntax” below.

ModName

A string to be used as the model name. This string is used for naming various aspects of the computations, including generating C symbol names, dynamic libraries, etc. Therefore, it is necessary that modName consists of simple ASCII alphanumeric characters starting with a letter.

Wd

A character string with a working directory where to create a subdirectory according to modName. When specified, a subdirectory named after the “modName.d” will be created and populated with a C file, a dynamic loading library, plus various other working files. If missing, the files are created (and removed) in the temporary directory, and the RxODE DLL for the model is created in the current directory named rx_???.platform, for example rx_129f8f97fb94a87ca49ca8dafffe691e1e_i386.dll

Filename

A file name or connection object where the ODE-based model specification resides. Only one of model or filename may be specified.

do.compile

Logical specifying whether to carry out the parsing of the ODE into C code and its compilation. Default is TRUE

extraC

Extra C code to include in the model. This can be useful to specify functions in the model. These C functions should usually take double precision arguments, and return double precision values.

debug

A boolean indicating if the executable should be compiled with verbose debugging information turned on.

calcJac

A boolean indicating if RxODE will calculate the Jacobian according to the specified ODEs.

calcSens

A boolean indicating if RxODE will calculate the sensitivities according to the specified ODEs.

collapseModel

A boolean indicating if RxODE will remove all LHS variables when calculating sensitivities.

... any other arguments are passed to the function readLines, (e.g., encoding).

The “Rx” in the name RxODE is meant to suggest the abbreviation Rx for a medical prescription, and thus to suggest the package emphasis on pharmacometrics modeling, including pharmacokinetics (PK), pharmacodynamics (PD), disease progression, drug-disease modeling, etc.

The ODE-based model specification may be coded inside a character string or in a text file, see Section RxODE Syntax below for coding details. An internal RxODE compilation manager object translates the ODE system into C, compiles it, and dynamically loads the object code into the current R session. The call to RxODE produces an object of class RxODE which consists of a list-like structure (closure) with various member functions (see Section Value below).
For evaluating RxODE models, two types of inputs may be provided: a required set of time points for querying the state of the ODE system and an optional set of doses (input amounts). These inputs are combined into a single *event table* object created with the function `eventtable`.

**Value**

An object (closure) of class “RxODE” (see Chambers and Temple Lang (2001)) consisting of the following list of strings and functions:

- **modName**: the name of the model (a copy of the input argument).
- **model**: a character string holding the source model specification.
- **get.modelVars**: a function that returns a list with 3 character vectors, `params`, `state`, and `lhs` of variable names used in the model specification. These will be output when the model is computed (i.e., the ODE solved by integration).
- **solve**: this function solves (integrates) the ODE. This is done by passing the code to `rxSolve`. This is as if you called `rxSolve(RxODEobject, ...)`, but returns a matrix instead of a `rxSolve` object.
- **params**: a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;
- **events**: an `eventTable` object describing the input (e.g., doses) to the dynamic system and observation sampling time points (see `eventTable`);
- **inits**: a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);
- **stiff**: a logical (TRUE by default) indicating whether the ODE system is stiff or not.

For stiff ODE systems (stiff = TRUE), RxODE uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003).

For non-stiff systems (stiff = FALSE), RxODE uses DOP853, an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).

- **trans_abs**: a logical (FALSE by default) indicating whether to fit a transit absorption term (TODO: need further documentation and example);
- **atol**: a numeric absolute tolerance (1e-08 by default);
- **rtol**: a numeric relative tolerance (1e-06 by default).

The output of “solve” is a matrix with as many rows as there are sampled time points and as many columns as system variables (as defined by the ODEs and additional assignments in the RxODE model code).

- **isValid**: a function that (naively) checks for model validity, namely that the C object code reflects the latest model specification.
- **version**: a string with the version of the RxODE object (not the package).
dynLoad  a function with one force = FALSE argument that dynamically loads the object code if needed.
dynUnload a function with no argument that unloads the model object code.
cmpMgr  a “compilation manager” object, see rx.initCmpMgr.
delete  removes all created model files, including C and DDL files. The model object is no longer valid and should be removed, e.g., rm(m1).
run  deprecated, use solve.
parse  deprecated.
compile  deprecated.
get.index  deprecated.
getObj  internal (not user callable) function.

RxODE Syntax

An RxODE model specification consists of one or more statements terminated by semi-colons, ‘;’, and optional comments (comments are delimited by # and an end-of-line marker). NB: Comments are not allowed inside statements.

A block of statements is a set of statements delimited by curly braces, ‘{ ... }’. Statements can be either assignments or conditional if statements. Assignment statements can be: (1) “simple” assignments, where the left hand is an identifier (i.e., variable), (2) special “time-derivative” assignments, where the left hand specifies the change of that variable with respect to time e.g., d/dt(depot), or (3) special “jacobian” assignments, where the left hand specifies the change of the ODE with respect to one of the parameters, e.g. df(depot)/dy(kel). The “jacobian” assignments are not required, and are only useful for very stiff differential systems.

Expressions in assignment and ‘if’ statements can be numeric or logical (no character expressions are currently supported). Numeric expressions can include the following numeric operators (‘+’, ‘-’, ‘*’, ‘/’, ‘^’), and those mathematical functions defined in the C or the R math libraries (e.g., fabs, exp, log, sin). (Notice that the modulo operator ‘%’ is currently not supported.)

Identifiers in an RxODE model specification can refer to:

- state variables in the dynamic system (e.g., compartments in a pharmacokinetics/pharmacodynamics model);
- implied input variable, t (time), podo (oral dose, for absorption models), and tlast (last time point);
- model parameters, (ka rate of absorption, CL clearance, etc.);
- pi, for the constant pi.
- others, as created by assignments as part of the model specification.

Identifiers consists of case-sensitive alphanumeric characters, plus the underscore ‘_’ character. NB: the dot ‘.’ character is not a valid character identifier.

The values of these variables at pre-specified time points are saved as part of the fitted/integrated/solved model (see eventTable, in particular its member function add_sampling that defines a set of time points at which to capture a snapshot of the system via the values of these variables).

The ODE specification mini-language is parsed with the help of the open source tool DParser, Pleyvay (2015).
Author(s)

Melissa Hallow, Wenping Wang and Matthew Fidler

References


See Also

eventTable

Examples

# Step 1 - Create a model specification
ode <- "
# A 4-compartment model, 3 PK and 1 PD (effect) compartment
# (notice state variable names 'depot', 'centr', 'peri', 'eff')

C2 = centr/V2;
C3 = peri/V3;
d/dt(depot) = KA*depot;
d/dt(centr) = KA*depot - CL*C2*Q*C3; 
d/dt(peri) = Q*C2*Q*C3;
d/dt(eff) = kin*depot otherseff;
"

m1 <- RxODE(model = ode, modName = "m1")
print(m1)

# Step 2 - Create the model input as an EventTable, # including dosing and observation (sampling) events

# QD (once daily) dosing for 5 days.
qd <- eventTable(amount.units = "ug", time.units = "hours")
qd$add.dosing(dose = 10000, nbr.doses = 5, dosing.interval = 24)

# Sample the system hourly during the first day, every 8 hours # then after
qd$add.sampling(seq(from = 24+8, to = 5*24, by = 8))
# Step 3 - set starting parameter estimates and initial values of the state

```r
ttheta <-
  c(KA = .291, CL = 18.6, V2 = 40.2, Q = 10.5, V3 = 297.0, Kin = 1.0, Kout = 1.0, EC50 = 200.0)

# init state variable
inits <- c(0, 0, 0, 1);

# Step 4 - Fit the model to the data
qd.cp <- mI$solve(theta, events = qd, inits)

head(qd.cp)
```

# This returns a matrix. Note that you can also solve using name initial values. For example:
```r
inits <- c(eff = 1);
qd.cp <- solve(mI, theta, events = qd, inits);
```

---

### rxOptions

**Options for RxODE**

**Description**

This is a backend for rxPermissive (with `op.rx = 2`) and rxStrict (with `op.rx = 1`)

**Usage**

```r
rxOptions(exprL opL rx = NULL, silent = (regexp("/tests/testthat/", getwd(),
   fixed = TRUE) != 1), respect = FALSE,
rxclean = (regexp("/tests/testthat/", getwd(), fixed = TRUE) != 1),
cran = FALSE, on.validate = FALSE)
```

**Arguments**

- **expr**
  - Expression to evaluate in the permissive/strict environment. If unspecified, set the options for the current environment.

- **op.rx**
  - A numeric for strict (1) or permissive (2) syntax.

- **silent**
  - when true, also silence the syntax errors and interactive output (useful in testing).

- **respect**
  - when TRUE, respect any options that are specified. This is called at startup, but really should not be called elsewhere, otherwise the options are not changed.
rxPairwiseSum

rxclean when TRUE, call rxClean before and after the expr is called.
cran When specified and true, run on CRAN. Otherwise it is skipped on cran.
on.validate When TRUE run only when validating.

Details
When expr is missing and op.rx is NULL, this desplays the current RxODE options.

Author(s)
Matthew L. Fidler

----------

rxPairwiseSum Return an accurate floating point sum of values

Description
This was taken by NumPy and adapted for use here.

Usage
rxPairwiseSum(numbers)

Arguments
numbers A vector of numbers to sum.

Value
A sum of numbers with a rounding error of O(lg n) instead of O(n).

Author(s)
Matthew Fidler (R implementation), Julian Taylor, Nathaniel J Smith, and others in NumPy team.

References
https://github.com/juliantaylor/numpy/blob/b0bc01275cac04483e6df021211c1fa2ba65eaa3/numpy/core/src/umath/loops.c.src
https://github.com/numpy/numpy/pull/3685
### rxParams

**Parameters specified by the model**

**Description**

This returns the model’s parameters that are required to solve the ODE system.

**Usage**

```r
rxParams(obj)  
rxParam(obj)
```

**Arguments**

- `obj`: RxODE family of objects

**Value**

A character vector listing the parameters in the model.

**Author(s)**

Matthew L. Fidler

### rxPermissive

**Permissive or Strict RxODE syntax options**

**Description**

This sets the RxODE syntax to be permissive or strict.

**Usage**

```r
rxPermissive(expr, silent = (regexpr("/tests/testthat/", getwd(), fixed = TRUE) != -1), respect = FALSE, rxclean = (regexpr("/tests/testthat/", getwd(), fixed = TRUE) != -1), cran = FALSE, on.validate = FALSE)
```

```r
rxStrict(expr, silent = (regexpr("/tests/testthat/", getwd(), fixed = TRUE) != -1), respect = FALSE, rxclean = (regexpr("/tests/testthat/", getwd(), fixed = TRUE) != -1), cran = FALSE, on.validate = FALSE)
```
**Arguments**

- **expr**: Expression to evaluate in the permissive/strict environment. If unspecified, set the options for the current environment.
- **silent**: when true, also silence the syntax errors and interactive output (useful in testing).
- **respect**: when TRUE, respect any options that are specified. This is called at startup, but really should not be called elsewhere, otherwise the options are not changed.
- **rxclean**: when TRUE, call rxClean before and after the expr is called.
- **cran**: When specified and true, run on CRAN. Otherwise it is skipped on cran.
- **on.validate**: When TRUE run only when validating.

**Author(s)**

Matthew L. Fidler

---

**rxProd**

*Using RxODE’s default method, take a product*

**Description**

Using RxODE’s default method, take a product

**Usage**

`rxProd(numbers)`

**Arguments**

- **numbers**: A vector of numbers to sum.

**Value**

Product of numbers
rxPythonFsum

Return an accurate floating point sum of values

Description

This method avoids loss of precision by tracking multiple intermediate partial sums. Based on Python’s math.fsum

Usage

rxPythonFsum(numbers)

Arguments

numbers A vector of numbers to sum.

Value

Sum of numbers without loss of precision

The algorithm’s accuracy depends on IEEE-754 arithmetic guarantees and the typical case where the rounding mode is half-even. On some non-Windows builds, the underlying C library uses extended precision addition and may occasionally double-round an intermediate sum causing it to be off in its least significant bit.

Author(s)

Matthew Fidler (R implementation), Raymond Hettinger, Jonathan Shewchuk, Python Team

References

https://docs.python.org/2/library/math.html
https://code.activestate.com/recipes/393090/
https://github.com/python/cpython/blob/a0ce375e10b50f7606cb86b072fed7d8cd574fe7/Modules/mathmodule.c

**rxRmPrint**

Remove print statements

**Description**

Remove print statements

**Usage**

`rxRmPrint(x)`

**Arguments**

- `x` RxODE lines to remove

**Value**

RxODE with print lines removed.

**Author(s)**

Matthew L. Fidler

---

**rxRmSens**

Remove sensitivity equations

**Description**

Remove sensitivity equations

**Usage**

`rxRmSens(x)`

**Arguments**

- `x` RxODE lines to remove

**Value**

Lines with d/dt(rx_sens_....) removed.

**Author(s)**

Matthew L. Fidler
**rxRtoolsBaseWin**

*Return Rtools base*

**Description**

Return Rtools base

**Usage**

```
rxRtoolsBaseWin()
```

**Value**

Rtools base path, or "" on unix-style platforms.

**Author(s)**

Matthew L. Fidler

---

**rxSetProd**

*Choose the type of product to use in RxODE. These are used in the RxODE prod blocks*

**Description**

Choose the type of product to use in RxODE. These are used in the RxODE prod blocks

**Usage**

```
rxSetProd(type = c("long double", "double", "logify"))
```

**Arguments**

- **type**
  
  Product to use for prod() in RxODE blocks
  
  long double converts to long double, performs the multiplication and then converts back.
  
  double uses the standard double scale for multiplication.

**Value**

nothing

**Author(s)**

Matthew L. Fidler
**rxSetSum**

*Choose the type of sums to use for RxODE.*

**Description**

Choose the types of sums to use in RxODE. These are used in the RxODE `sum` blocks and the `rxSum` function.

**Usage**

```r
rxSetSum(type = c("pairwise", "fsum", "kahan", "neumaier", "c"))
```

**Arguments**

- `type`
  - Sum type to use for `rxSum` and `sum()` in RxODE code blocks.
  - pairwise uses the pairwise sum (fast, default)
  - fsum uses Python's fsum function (most accurate)
  - kahan uses kahan correction
  - neumaier uses Neumaier correction
  - c uses no correction, but default/native summing

**Value**

nothing

**Author(s)**

Matthew L. Fidler

---

**rxSetupMemoize**

*This sets the memoized functions.*

**Description**

To easily create a memoized function by adding a `.slow <- NULL` to the end of a function.

**Usage**

```r
rxSetupMemoize()
```

**Details**

For example, to memoize the function in the namespace `rxModelVars.character` you would add a line: `rxModelVars.character.slow <- NULL`

**Author(s)**

Matthew L. Fidler
**rxSolve**

*Solves a ODE equation*

**Description**

This uses RxODE family of objects, file, or model specification to solve a ODE system.

**Usage**

```r
rxSolve(object, params = NULL, events = NULL, inits = NULL, scale = c(),
  covs = NULL, stiff = TRUE, transit_abs = NULL, atol = 1e-06,
  rtol = 1e-06, maxsteps = 5000, hmin = 0, hmax = NULL, hini = 0,
  maxordn = 12, maxords = 5, ..., covs_interpolation = c("linear",
  "constant"), theta = numeric(), eta = numeric(), add.cov = FALSE)
```

```r
## S3 method for class 'RxODE'
rxSolve(object, params = NULL, events = NULL,
inits = NULL, scale = c(), covs = NULL, stiff = TRUE,
  transit_abs = NULL, atol = 1e-08, rtol = 1e-06, maxsteps = 5000,
  hmin = 0, hmax = NULL, hini = 0, maxordn = 12, maxords = 5, ...,
  covs_interpolation = c("linear", "constant"), theta = numeric(),
  eta = numeric(), matrix = FALSE, add.cov = FALSE)
```

```r
## S3 method for class 'solveRxODE'
rxSolve(object, params = NULL, events = NULL,
inits = NULL, scale = c(), covs = NULL, stiff = TRUE,
  transit_abs = NULL, atol = 1e-08, rtol = 1e-06, maxsteps = 5000,
  hmin = 0, hmax = NULL, hini = 0, maxordn = 12, maxords = 5, ...,
  covs_interpolation = c("linear", "constant"), theta = numeric(),
  eta = numeric(), matrix = FALSE, add.cov = FALSE)
```

**Arguments**

- **object**: is a either a RxODE family of objects, or a file-name with a RxODE model specification, or a string with a RxODE model specification.
- **params**: a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;
- **events**: an `eventTable` object describing the input (e.g., doses) to the dynamic system and observation sampling time points (see `eventTable`);
- **inits**: a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);
- **scale**: a numeric named vector with scaling for ode parameters of the system. The names must correspond to the parameter identifiers in the ODE specification. Each of the ODE variables will be divided by the scaling factor. For example `scale=(center=2)` will divide the center ODE variable by 2.
covs  a matrix or dataframe the same number of rows as the sampling points defined in the events eventTable. This is for time-varying covariates.

stiff  a logical (TRUE by default) indicating whether the ODE system is stiff or not. For stiff ODE systems (stiff = TRUE), RxODE uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003). For non-stiff systems (stiff = FALSE), RxODE uses DOP853, an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).

transit_abs boolean indicating if this is a transit compartment absorption

atol  a numeric absolute tolerance (1e-08 by default) used by the ODE solver to determine if a good solution has been achieved;

rtol  a numeric relative tolerance (1e-06 by default) used by the ODE solver to determine if a good solution has been achieved.

maxsteps maximum number of (internally defined) steps allowed during one call to the solver. (5000 by default)

hmin  The minimum absolute step size allowed. The default value is 0.

hmax  The maximum absolute step size allowed. The default checks for the maximum difference in times in your sampling and events, and uses this value. The value 0 is equivalent to infinite maximum absolute step size.

hini  The step size to be attempted on the first step. The default value is determined by the solver (when hini = 0)

maxordn  The maximum order to be allowed for the nonstiff (Adams) method. The default is 12. It can be between 1 and 12.

maxords  The maximum order to be allowed for the stiff (BDF) method. The default value is 5. This can be between 1 and 5.

...  Other arguments including scaling factors for each compartment. This includes S# = numeric will scale a compartment # by a dividing the compartment amount by the scale factor, like NONMEM.

covs_interpolation specifies the interpolation method for time-varying covariates. When solving ODEs it often samples times outside the sampling time specified in events. When this happens, the time varying covariates are interpolated. Currently this can be "linear" interpolation (the default), which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value. The other possibility is "constant", or Last observation carried forward. In this approach, the last observation of the covariate is considered the current value of the covariate.

theta  A vector of parameters that will be named THETA[#] and added to inits

eta  A vector of parameters that will be named ETA[#] and added to inits

add.cov  A boolean indicating if covariates should be added to the output matrix or data frame. By default this is disabled.

matrix  A boolean indicating if a matrix should be returned instead of the RxODE’s solved object
Value

An “rxSolve” solve object that stores the solved value in a matrix with as many rows as there are sampled time points and as many columns as system variables (as defined by the ODEs and additional assignments in the RxODE model code). It also stores information about the call to allow dynamic updating of the solved object.

The operations for the object are similar to a data-frame, but expand the $ and [["""\] access operators and assignment operators to resolve based on different parameter values, initial conditions, solver parameters, or events (by updating the time variable).

You can call the `eventTable` methods on the solved object to update the event table and resolve the system of equations.

Author(s)

Melissa Hallow, Wenping Wang and Matthew Fidler

References


See Also

RxODE

---

**rxState** State variables

**Description**

This returns the model’s compartments or states.

**Usage**

`rxState(obj, state)`

**Arguments**

<table>
<thead>
<tr>
<th>obj</th>
<th>RxODE family of objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>is a string indicating the state or compartment that you would like to lookup.</td>
</tr>
</tbody>
</table>
Value

If state is missing, return a character vector of all the states.
If state is a string, return the compartment number of the named state.

Author(s)

Matthew L. Fidler

See Also

RxODE

 rxSum

Using RxODE’s default method, take a sum

Description

Using RxODE’s default method, take a sum

Usage

rxSum(numbers)

Arguments

numbers A vector of numbers to sum.

Value

Sum of numbers

 rxSumProdModel

Recast model in terms of sum/prod

Description

Recast model in terms of sum/prod

Usage

rxSumProdModel(model, expand = FALSE, sum = FALSE, prod = TRUE, logify = FALSE)
rxSymDiag

Arguments

model RxODE model
expand Boolean indicating if the expression is expanded.
sum Use sum(...)
prod Use prod(...)
logify Use logify

Value

model string with prod(.) and sum(.) for all these operations.

Author(s)

Matthew L. Fidler

---

rxSymDiag Get the theta numbers for the diagonal elements.

Description

These start at element 1 like in R instead of element 0 like in C

Usage

rxSymDiag(x)

Arguments

x object created with rxSymInvCreate

Value

Theta numbers for diagonal elements

Author(s)

Matthew L. Fidler
rxSymInv

Get Omega and Omega^-1 and derivatives

Description
Get Omega and Omega^-1 and derivatives

Usage
rxSymInv(invobj, theta, pow = 0, dTheta = 0)

Arguments
invobj       rxSymInv object
theta        Thetas to be used for calculation
pow          Power of Omega, either 1 or -1 for inverse. If 0, return an environment with all
             possible derivatives and Omegas calculated
               the theta number to take the derivative. That is d(Omega)/dTheta. When dTheta
dTheta       = 0, just return the Omega or Omega^-1 matrix. Also can be 0.5. When 0.5 and
             pow is -1, return -1/2*log(det(omegaInv)).

Value
Matrix based on parameters or environment with all the matrixes calculated in variables omega,
            omegaInv, dOmega, dOmegaInv.

Author(s)
Matthew L. Fidler

rxSymPyFix
Fix SymPy expressions to be R parsable expressions

Description
Fix SymPy expressions to be R parsable expressions

Usage
rxSymPyFix(var)

Arguments
var          sympy expression
**rxSymPySensitivity**

**Value**
R valid expression

**Author(s)**
Matthew L. Fidler

---

**rxSymPyJacobian**  
*Calculate the full Jacobian for a model*

**Description**
This expands the model to calculate the Jacobian. This requires rSymPy.

**Usage**

```
rxSymPyJacobian(model)
```

**Arguments**

- `model`  
  RxODE family of objects

**Value**
RxODE syntax for model with Jacobian specified.

**Author(s)**
Matthew L. Fidler

---

**rxSymPySensitivity**  
*Calculate the sensitivity equations for a model*

**Description**
This expands the model to calculate sensitivities. This requires rSymPy.

**Usage**

```
rxSymPySensitivity(model, calcSens, calcJac = FALSE, keepState = NULL, collapseModel = FALSE)
```
**Arguments**

- **model**: RxODE family of objects
- **calcSens**: Either a logical or list of sensitivity parameters to calculate. When TRUE, calculate the sensitivities of all the known parameters. When FALSE raise an error.
- **calcJac**: A boolean that determines if the Jacobian should be calculated.
- **keepState**: State parameters to keep the sensitivities for.
- **collapseModel**: A boolean to collapse the model that each expression only depends on the unspecified parameters (instead on LHS quantities).

**Value**

Model syntax that includes the sensitivity parameters.

**Author(s)**

Matthew L. Fidler

---

**rxSymPyVersion**  
*Return the version of SymPy that is running*

**Description**

Return the version of SymPy that is running

**Usage**

```
rxSymPyVersion(numeric = TRUE)
```

**Arguments**

- **numeric**: boolean that specifies if the major and minor release should be a number.

**Value**

Version of sympy that is running.

**Author(s)**

Matthew L. Fidler
**rxSyncOptions**

**Sync options with RxODE variables**

**Description**

Accessing RxODE options via `getOption` slows down solving. This allows the options to be synced with variables.

**Usage**

```r
rxSyncOptions()
```

**Author(s)**

Matthew L. Fidler

---

**rxTrans**

**Translate the model to C code if needed**

**Description**

This function translates the model to C code, if needed.

**Usage**

```r
rxTrans(model, cFile = sprintf("%s.c", gsub("[.][^.]\*\$", ",", model)),
          extraC = NULL, modelName = NULL, md5 = "", modVars = FALSE, calcSens = NULL, calCJac = NULL,
          collapseModel = FALSE, ...)
```

```
## Default S3 method:
rxTrans(model, cFile = sprintf("%s.c", gsub("[.][^.]\*\$", ",", model)),
          extraC = NULL, modelPrefix = "", md5 = "",
          modelName = NULL, modVars = FALSE, calcSens = NULL, calcJac = NULL,
          collapseModel = FALSE, ...)
```

```
## S3 method for class 'character'
rxTrans(model, cFile = sprintf("%s.c", gsub("[.][^.]\*\$", ",", model)),
          extraC = NULL, modelPrefix = "", md5 = "",
          modelName = NULL, modVars = FALSE, calcSens = NULL, calcJac = NULL,
          collapseModel = FALSE, ...)
```
Arguments

model This is the ODE model specification. It can be:
- a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.
- a file name where the ODE system equation is contained
- An ODE expression enclosed in {} (see also the filename argument). For details, see the sections “Details” and “RxODE Syntax” below.

cFile The C file where the code should be output
extraC Extra c code to include in the model. This can be useful to specify functions in the model. These C functions should usually take double precision arguments, and return double precision values.
modelPrefix Prefix of the model functions that will be compiled to make sure that multiple RxODE objects can coexist in the same R session.
md5 Is the md5 of the model before parsing, and is used to embed the md5 into DLL, and then provide for functions like rxModelVars.
modName a string to be used as the model name. This string is used for naming various aspects of the computations, including generating C symbol names, dynamic libraries, etc. Therefore, it is necessary that modName consists of simple ASCII alphanumeric characters starting with a letter.
modVars returns the model variables instead of the named vector of translated properties.
calcSens boolean indicating if RxODE will calculate the sensitivities according to the specified ODEs.
calcJac boolean indicating if RxODE will calculate the Jacobain according to the specified ODEs.
collapseModel boolean indicating if RxODE will remove all LHS variables when calculating sensitivities.
... Ignored parameters.

Value

a named vector of translated model properties including what type of jacobian is specified, the C function prefix, as well as the C functions names to be called through the compiled model.

Author(s)

Matthew L. Fidler

See Also

RxODE, rxCompile.
**rxUnload**

*Unload the DLL for the object*

**Description**

This unloads the DLL in the R session so that the DLL can be deleted. All the C functions will no longer be accessible.

**Usage**

`rxUnload(obj)`

**Arguments**

- `obj` a RxODE family of objects

**Author(s)**

Matthew L. Fidler

---

**rxValidate**

*Validate RxODE*

**Description**

This allows easy validation/qualification of nlmixr by running the testing suite on your system.

**Usage**

`rxValidate(full = TRUE)`

`rxTest(full = TRUE)`

**Arguments**

- `full` Should a full validation be performed? (By default TRUE)

**Author(s)**

Matthew L. Fidler
**rxWinPythonSetup**  
*Setup Python and SymPy for windows*

**Description**

Setup Python and SymPy for windows

**Usage**

```r
rxWinPythonSetup()
```

**Author(s)**

Matthew L. Fidler

---

**rxWinRtoolsPath**  
*Setup Rtools path*

**Description**

Setup Rtools path

**Usage**

```r
rxWinRtoolsPath(rm.rtools = TRUE)
```

**Arguments**

- `rm.rtools` Remove the Rtools from the current path specs.

**Author(s)**

Matthew L. Fidler
### rxWinSetup

*Setup Windows components for RxODE*

**Description**

Setup Windows components for RxODE

**Usage**

```r
rxWinSetup(rm.rtools = TRUE)
```

**Arguments**

- `rm.rtools`: Remove Rtools from path?

**Author(s)**

Matthew L. Fidler

---

### sample_frac

*sample_frac for solveRxDll object*

**Description**

compatibility function for dplyr

**Usage**

```r
## S3 method for class 'solveRxDll'
sample_frac(tbl, ...)
```

**Arguments**

- `tbl`: Solved equation, an `solveRxDll` object.
- `...`: Additional arguments
**sample_n**

*sample_n for `solveRxDll` object*

---

**Description**

compatibility function for dplyr

**Usage**

```r
## S3 method for class 'solveRxDll'
sample_n(tbl, ...)
```

**Arguments**

- `tbl`  
  Solved equation, an `solveRxDll` object.
- `...`  
  Additional arguments

---

**select_**

*select for `solveRxDll` object*

---

**Description**

compatibility function for dplyr

**Usage**

```r
## S3 method for class 'solveRxDll'
select_(.data, ...)
```

**Arguments**

- `.data`  
  Solved equation, an `solveRxDll` object.
- `...`  
  Additional arguments
Description

compatibility function for tidyr

Usage

```r
## S3 method for class 'solveRxDll'
separate_(data, ...)
```

Arguments

data 
Solved ODE, an `solveRxDll` object.

... Additional arguments

Description

compatibility function for dplyr

Usage

```r
## S3 method for class 'solveRxDll'
slice_(..data, ...)
```

Arguments

..data 
Solved equation, an `solveRxDll` object.

... Additional arguments
solve.RxODE

Solve RxODE objects

Description
Solve RxODE objects

Usage

## S3 method for class 'RxODE'
solve(...)

## S3 method for class 'RxCompilationManager'
solve(...)

## S3 method for class 'solveRxDll'
solve(...)

## S3 method for class 'character'
solve(...)

## S3 method for class 'rxDll'
solve(...)

Arguments
... Additional arguments sent to rxSolve

Author(s)
Matthew L.Fidler

See Also
rxSolve

spread_ spread_for solveRxDll object

Description
compatability function for tidyr

Usage

## S3 method for class 'solveRxDll'
spread_(data, ...)

Arguments

- **data**: Solved ODE, an `solveRxDll` object.
- **...**: Additional arguments

Description

Compatibility function for dplyr

Usage

```r
## S3 method for class 'solveRxDll'
summarise_(.data, ...)
```

Arguments

- **.data**: Solved equation, an `solveRxDll` object.
- **...**: Additional arguments

summary.RxODE

*Print expanded information about the RxODE object.*

Description

This prints the expanded information about the RxODE object.

Usage

```r
## S3 method for class 'RxODE'
summary(object, ...)
```

```r
## S3 method for class 'RxCompilationManager'
summary(object, ...)
```

Arguments

- **object**: RxODE object
- **...**: Ignored parameters

Author(s)

Matthew L. Fidler
### transmute

**transmute** transmute for solveRxDll object

#### Description
compatability function for dplyr

#### Usage
```r
## S3 method for class 'solveRxDll'
transmute(.data, ...)
```

#### Arguments
- `.data` Solved equation, an solveRxDll object.
- `...` Additional arguments

### unite

**unite** unite for solveRxDll object

#### Description
compatability function for tidyr

#### Usage
```r
## S3 method for class 'solveRxDll'
unite(data, ...)
```

#### Arguments
- `data` Solved ODE, an solveRxDll object.
- `...` Additional arguments
update.solveRxODE

Update the solved object with any of the new parameters.

Description

Update the solved object with any of the new parameters.

Usage

### S3 method for class 'solveRxODE'

update(object, ...)

Arguments

- object: Object to be updated
- ...: Arguments to be updated, and resolved.

Author(s)

Matthew L. Fidler
Index

*Topic **Internal**
  - print.rxCoefSolve, 13
*Topic **data**
  - eventTable, 6
*Topic **models**
  - eventTable, 6
  - RxODE, 23
*Topic **nonlinear**
  - genShinyApp.template, 10
  - RxODE, 23
*Topic **simulation**
  - genShinyApp.template, 10
  + solveRxDll (rxChain), 15
  .C, 17
  .Call, 17

  - add.dosing, 3
  - add.sampling, 4
  - arrange_, 5

  - coef.RxCompilationManager (coef.RxODE), 5
  - coef.rxDll (coef.RxODE), 5
  - coef.RxODE, 5
  - coef.solveRxDODE (coef.RxODE), 5

  - distinct_, 6

  - eventTable, 4, 6, 11, 25–27, 36, 38

  - filter_, 9

  - gather_, 9
  - genShinyApp.template, 10
  - group_by_, 11

  - mutate_, 12

  - predict.RxODE, 12
  - print.rxCoefSolve, 13

  - print.RxCompilationManager (print.RxODE), 13
  - print.RxODE, 13

  - readLines, 24
  - rename_, 14
  - rx.initCmpMgr, 26
  - rxAddReturn, 14
  - rxChain, 15
  - rxClean, 15
  - rxCompile, 16, 45
  - rxCoutEcho, 18
  - rxDelete, 18
  - rxDfdy, 19
  - rxDllLoaded, 19
  - rxInv, 20
  - rxKahanSum, 20
  - rxLhs, 21
  - rxLoad, 21
  - rxLogifyModel, 22
  - rxModelVars, 45
  - rxNeumaierSum, 22
  - rxNorm, 23
  - RxODE, 4, 8, 10, 11, 17, 21, 23, 38, 39, 45
  - rxOptions, 28
  - rxPairwiseSum, 29
  - rxParam (rxParams), 30
  - rxParams, 30
  - rxPermissive, 30
  - rxProd, 31
  - rxPythonFsum, 32
  - rxRmPrint, 33
  - rxRmSens, 33
  - rxtoolsBaseWin, 34
  - rxSetProd, 34
  - rxSetSum, 35
  - rxSetupMemoize, 35
  - rxSolve, 10, 12, 25, 36, 51
  - rxState, 38
  - rxStrict (rxPermissive), 30
rxSum, 39
rxSumProdModel, 39
rxSymDiag, 40
rxSymInv, 41
rxSymPyFix, 41
rxSymPyJacobian, 42
rxSymPySensitivity, 42
rxSymPyVersion, 43
rxSyncOptions, 44
rxTest (rxValidate), 46
rxTrans, 17, 44
rxUnload, 46
rxValidate, 46
rxWinPythonSetup, 47
rxWinRtoolsPath, 47
rxWinSetup, 48

sample_frac, 48
sample_n, 49
select_, 49
separate_, 50
slice_, 50
solve.character (solve.RxODE), 51
solve.RxCompilationManager
  (solve.RxODE), 51
solve.rxDll (solve.RxODE), 51
solve.RxODE, 51
solve.solveRxDll (solve.RxODE), 51
spread_, 51
summarise_, 52
summary.RxCompilationManager
  (summary.RxODE), 52
summary.RxODE, 52
transmute_, 53
unite_, 53
update.solveRxODE, 54

write.template.server
  (genShinyApp.template), 10
write.template.ui, 10
write.template.ui
  (genShinyApp.template), 10