Package ‘rxode2’

October 20, 2022

**Version**  2.0.10

**Title**  Facilities for Simulating from ODE-Based Models

**Maintainer**  Matthew L. Fidler <matthew.fidler@gmail.com>

**Depends**  R (>= 4.0.0)

**Suggests**  Matrix, DT, covr, crayon, curl, digest, dplyr (>= 0.8.0), ggrepel, gridExtra, htmltools, knitr, learnr, microbenchmark, nlm, remotes, rmarkdown, scales, shiny, stringi, symengine, testthat, tidyr, usethis, vdiffr (>= 1.0), withr, xgxr, pillar, tibble, units (>= 0.6-0), rsconnect, devtools, patchwork, nlmixr2data, lifecycle

**Imports**  PreciseSums (>= 0.3), Rcpp (>= 0.12.3), backports, cli (>= 2.0.0), checkmate, ggplot2, inline, lotri (>= 0.4.0), magrittr, memoise, methods, rex, sys, tools, utils, rxode2ll (>= 2.0.9), rxode2et (>= 2.0.9), rxode2parse (>= 2.0.11), rxode2random (>= 2.0.9), data.table (>= 1.12.4), qs

**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.

**BugReports**  https://github.com/nlmixr2/rxode2/issues/

**NeedsCompilation**  yes

**VignetteBuilder**  knitr

**License**  GPL (>= 3)

RoxygenNote 7.2.1
Biarch true
LinkingTo rxode2parse, rxode2random, rxode2et, PreciseSums (>= 0.3), Rcpp, RcppArmadillo (>= 0.9.300.2.0), BH
Encoding UTF-8
LazyData true
Language en-US
Config/testthat/edition 3
Author Matthew L. Fidler [aut, cre] (<https://orcid.org/0000-0001-8538-6691>), Melissa Hallow [aut], Wenping Wang [aut], Zufar Mulyukov [ctb], Alan Hindmarsh [ctb], Arun Srinivasan [ctb], Awad H. Al-Mohy [ctb], Cleve Moler [ctb], Drew Schmidt [ctb], Ernst Hairer [ctb], Gerhard Wanner [ctb], Gilbert Stewart [ctb], Goro Fuji [ctb], Hadley Wickham [ctb], Jack Dongarra [ctb], Jim Bunch [ctb], Linda Petzold [ctb], Martin Maechler [ctb], Matt Dowle [ctb], Matteo Fasiolo [ctb], Morwenn [ctb], Nicholas J. Higham [ctb], Roger B. Sidje [ctb], Simon Frost [ctb], Yu Feng [ctb]
Repository CRAN
Date/Publication 2022-10-20 21:12:35 UTC

R topics documented:

```
.copyUi ................................................................. 5
.handleSingleErrTypeNormOrTFoceiBase .............................................. 6
.modelHandleModelLines .......................................................... 7
.quoteCallInfoLines ............................................................. 8
.rxLinCmtGen ..................................................................... 8
.rxWithOptions .................................................................... 9
.rxWithWd ........................................................................ 9
```
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>assertRxUi</td>
<td>10</td>
</tr>
<tr>
<td>erf</td>
<td>12</td>
</tr>
<tr>
<td>gammap</td>
<td>13</td>
</tr>
<tr>
<td>gammapDer</td>
<td>14</td>
</tr>
<tr>
<td>gammapInv</td>
<td>14</td>
</tr>
<tr>
<td>gammaq</td>
<td>15</td>
</tr>
<tr>
<td>gammaqInv</td>
<td>16</td>
</tr>
<tr>
<td>genShinyApp.template</td>
<td>17</td>
</tr>
<tr>
<td>getRxThreads</td>
<td>19</td>
</tr>
<tr>
<td>ini.rxUi</td>
<td>20</td>
</tr>
<tr>
<td>llikBeta</td>
<td>22</td>
</tr>
<tr>
<td>llikBinom</td>
<td>23</td>
</tr>
<tr>
<td>llikCauchy</td>
<td>24</td>
</tr>
<tr>
<td>llikChisq</td>
<td>25</td>
</tr>
<tr>
<td>llikExp</td>
<td>26</td>
</tr>
<tr>
<td>llikF</td>
<td>27</td>
</tr>
<tr>
<td>llikGamma</td>
<td>28</td>
</tr>
<tr>
<td>llikGeom</td>
<td>29</td>
</tr>
<tr>
<td>llikNbinom</td>
<td>30</td>
</tr>
<tr>
<td>llikNbinomMu</td>
<td>31</td>
</tr>
<tr>
<td>llikNorm</td>
<td>32</td>
</tr>
<tr>
<td>llikPois</td>
<td>34</td>
</tr>
<tr>
<td>llikT</td>
<td>35</td>
</tr>
<tr>
<td>llikUnif</td>
<td>36</td>
</tr>
<tr>
<td>llikWeibull</td>
<td>37</td>
</tr>
<tr>
<td>logit</td>
<td>38</td>
</tr>
<tr>
<td>lowergamma</td>
<td>39</td>
</tr>
<tr>
<td>model.function</td>
<td>40</td>
</tr>
<tr>
<td>odeMethodToInt</td>
<td>41</td>
</tr>
<tr>
<td>plot.rxSolve</td>
<td>42</td>
</tr>
<tr>
<td>probit</td>
<td>43</td>
</tr>
<tr>
<td>rxAllowUnload</td>
<td>44</td>
</tr>
<tr>
<td>rxAppendModel</td>
<td>44</td>
</tr>
<tr>
<td>rxAssignControlValue</td>
<td>46</td>
</tr>
<tr>
<td>rxAssignPtr</td>
<td>46</td>
</tr>
<tr>
<td>rxbeta</td>
<td>47</td>
</tr>
<tr>
<td>rxbinom</td>
<td>48</td>
</tr>
<tr>
<td>rxcauchy</td>
<td>49</td>
</tr>
<tr>
<td>rxchisq</td>
<td>51</td>
</tr>
<tr>
<td>rxClean</td>
<td>52</td>
</tr>
<tr>
<td>rxCompile</td>
<td>53</td>
</tr>
<tr>
<td>rxControlUpdateSens</td>
<td>55</td>
</tr>
<tr>
<td>rxCreateCache</td>
<td>56</td>
</tr>
<tr>
<td>rxD</td>
<td>56</td>
</tr>
<tr>
<td>rxDelete</td>
<td>57</td>
</tr>
<tr>
<td>rxDerived</td>
<td>57</td>
</tr>
<tr>
<td>rxDfdy</td>
<td>59</td>
</tr>
<tr>
<td>rxexp</td>
<td>60</td>
</tr>
</tbody>
</table>
### R topics documented:

- `rxf` ................................................................. 61
- `rxFun` ............................................................. 63
- `rxgamma` ........................................................... 64
- `rxgeom` ............................................................. 66
- `rxGetControl` ..................................................... 67
- `rxGetLin` ........................................................... 68
- `rxGetxode2` ......................................................... 68
- `rxHtml` ............................................................. 69
- `rxIndLinState` ..................................................... 70
- `rxIndLinStrategy` .................................................. 70
- `rxInv` ............................................................... 72
- `rxIsCurrent` ......................................................... 72
- `rxLhs` ............................................................... 73
- `rxLock` ............................................................. 73
- `rxnbinom` ........................................................... 74
- `rxNorm` ............................................................. 75
- `rxnormV` ........................................................... 76
- `rxode2` ............................................................. 77
- `rxOptExpr` .......................................................... 85
- `rxord` ............................................................... 86
- `rxParams` ........................................................... 87
- `rxPkg` ............................................................... 90
- `rxpois` ............................................................. 91
- `rxPp` ............................................................... 92
- `rxPreferredDistributionName` .................................. 93
- `rxProgress` ........................................................ 94
- `rxRemoveControl` ................................................ 95
- `rxRename` .......................................................... 96
- `rxReservedKeywords` ............................................. 97
- `rxS` ................................................................. 97
- `rxSetControl` ...................................................... 98
- `rxSetCovariateNamesForPiping` ................................ 98
- `rxSetIni0` .......................................................... 100
- `rxSetProd` .......................................................... 100
- `rxSetProgressBar` ................................................. 101
- `rxSetSum` .......................................................... 101
- `rxShiny` ........................................................... 102
- `rxSimThetaOmega` ................................................ 103
- `rxSolve` ........................................................... 106
- `rxState` ........................................................... 120
- `rxSumProdModel` .................................................... 120
- `rxSupportedFuns` .................................................. 121
- `rxSuppressMsg` .................................................... 122
- `rxSymInvChol` ..................................................... 123
- `rxSyncOptions` ..................................................... 124
- `rxSyntaxFunctions` ................................................ 124
- `rxt` ............................................................... 125
Description

This copies the rxode2 UI object so it can be modified

Usage

copyUi(ui)

Arguments

ui Original UI object

Value

Copied UI object

Author(s)

Matthew L. Fidler
Handle the single error for normal or t distributions

Usage

.handleSingleErrTypeNormOrTFoceiBase(
  env,
  pred1,
  errNum = 1L,
  rxPredLlik = TRUE
)

Arguments

env       Environment for the parsed model
pred1     The data.frame of the current error
errNum    The number of the error specification in the nlmixr2 model
rxPredLlik A boolean indicating if the log likelihood should be calculated for non-normal distributions. By default TRUE.

Value

A list of the lines added. The lines will contain

- rx_yj_ which is an integer that corresponds to the transformation type.
- rx_lambda_ is the transformation lambda
- rx_low_ The lower boundary of the transformation
- rx_hi_ The upper boundary of the transformation
- rx_pred_f_ The prediction function
- rx_pred_ The transformed prediction function
- rx_r_ The transformed variance

Author(s)

Matthew Fidler
.modelHandleModelLines

Handle model lines

Description
Handle model lines

Usage
.modelHandleModelLines(
  modelLines,
  rxui,
  modifyIni = FALSE,
  append = FALSE,
  auto = TRUE,
  envir
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelLines</td>
<td>The model lines that are being considered</td>
</tr>
<tr>
<td>rxui</td>
<td>The rxode2 UI object</td>
</tr>
<tr>
<td>modifyIni</td>
<td>Should the ini() be considered</td>
</tr>
<tr>
<td>append</td>
<td>This is a boolean to determine if the lines are appended in piping. The possible values for this is:</td>
</tr>
<tr>
<td></td>
<td>• TRUE which is when the lines are appended to the model instead of replaced (default)</td>
</tr>
<tr>
<td></td>
<td>• FALSE when the lines are replaced in the model</td>
</tr>
<tr>
<td></td>
<td>• NA is when the lines are pre-pended to the model instead of replaced</td>
</tr>
<tr>
<td>auto</td>
<td>This boolean tells if piping automatically selects the parameters should be characterized as a population parameter, between subject variability, or a covariate. When TRUE this automatic selection occurs. When FALSE this automatic selection is turned off and everything is added as a covariate (which can be promoted to a parameter with the ini statement). By default this is TRUE, but it can be changed by options(rxode2.autoVarPiping=FALSE).</td>
</tr>
<tr>
<td>envir</td>
<td>Environment for evaluation</td>
</tr>
</tbody>
</table>

Value
New UI

Author(s)
Matthew L. Fidler
.quoteCallInfoLines  Returns quoted call information

Description
Returns quoted call information

Usage
.quoteCallInfoLines(callInfo, envir = parent.frame())

Arguments
 callInfo  Call information
 envir     Environment for evaluation (if needed)

Value
Quote call information. for name=expression, change to name<-expression in quoted call list. For expressions that are within brackets ie {}, unlist the brackets as if they were called in one single sequence.

Author(s)
Matthew L. Fidler

.rxLinCmtGen  Internal function to generate the model variables for a linCmt() model

Description
Internal function to generate the model variables for a linCmt() model

Usage
.rxLinCmtGen(lenState, vars)

Arguments
 lenState  Length of the state
 vars     Variables in the model

Value
Model variables of expanded linCmt model
.rxWithOptions

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

---

**Description**
Temporarily set options then restore them while running code

**Usage**
.rxWithOptions(ops, code)

**Arguments**
- **ops**  list of options that will be temporarily set for the code
- **code**  The code to run during the sink

**Value**
value of code

**Examples**

```r
.rxWithOptions(list(digits = 21), {
  print(pi)
})

print(pi)
```

---

.rxWithWd

**Description**
Temporarily set options then restore them while running code

**Usage**
.rxWithWd(wd, code)

**Arguments**
- **wd**  working directory to temporarily set the system to while evaluating the code
- **code**  The code to run during the sink
assertRxUi

Value
value of code

Examples

`assertRxUi(tempdir(), {
    getwd()
})`

getwd()

assertRxUi Assert properties of the rxUi models

Description
Assert properties of the rxUi models

Usage

assertRxUi(model, extra = "", .var.name = .vname(model))
assertRxUiPrediction(model, extra = "", .var.name = .vname(model))
assertRxUiSingleEndpoint(model, extra = "", .var.name = .vname(model))
assertRxUiTransformNormal(model, extra = "", .var.name = .vname(model))
assertRxUiNormal(model, extra = "", .var.name = .vname(model))
assertRxUiMuRefOnly(model, extra = "", .var.name = .vname(model))
assertRxUiEstimatedResiduals(model, extra = "", .var.name = .vname(model))
assertRxUiPopulationOnly(model, extra = "", .var.name = .vname(model))
assertRxUiMixedOnly(model, extra = "", .var.name = .vname(model))
assertRxUiRandomOnIdOnly(model, extra = "", .var.name = .vname(model))

Arguments

model Model to check
extra Extra text to append to the error message (like "for focei")
.var.name [character(1)]
Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
**assertRxUi**

**Details**

These functions have different types of assertions

- **assertRxUi** – Make sure this is a proper rxode2 model (if not throw error)
- **assertRxUiSingleEndpoint** – Make sure the rxode2 model is only a single endpoint model (if not throw error)
- **assertRxUiTransformNormal** – This needs to be a normal or transformably normal residual distribution
- **assertRxUiNormal** – This needs to be a normal residual distribution
- **assertRxUiEstimatedResiduals** – This makes sure that the residual error parameter are estimated (not modeled).
- **assertRxUiPopulationOnly** – This makes sure the model is the population only model (no mixed effects)
- **assertRxUiMixedOnly** – This makes sure the model is a mixed effect model (not a population effect)
- **assertRxUiPrediction** – This makes sure the model has predictions
- **assertRxUiMuRefOnly** – This make sure that all the parameters are mu-referenced
- **assertRxUiRandomOnIdOnly** – This makes sure there is only random effects at the ID level

**Value**

the rxUi model

**Author(s)**

Matthew L. Fidler

**Examples**

```R
one.cmt <- function() {
  ini(
    ## You may label each parameter with a comment
    tka <- 0.45 # Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  )
  model(
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
  )
}
```
\[ v \leftarrow \exp(tv + \text{eta.v}) \]
\[ \text{linCmt()} \sim \text{add(add.sd)} \]

```r
})
}

assertRxUi(one.cmt)
# assertRxUi(rnorm) # will fail
assertRxUiSingleEndpoint(one.cmt)
```

---

**erf**  
*Error function*

**Description**
Error function

**Usage**
erf(x)

**Arguments**
x vector or real values

**Value**
erf of x

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

**Examples**
erf(1.0)
Gammap: normalized lower incomplete gamma function

Description

This is the gamma_p from the boost library

Usage

Gammap(a, z)

Arguments

a          The numeric 'a' parameter in the normalized lower incomplete gamma
z          The numeric 'z' parameter in the normalized lower incomplete gamma

Details

The gammap function is given by:

\[
\text{gammap} = \frac{\text{lowergamma}(a, z)}{\gamma(a)}
\]

Value

gammap results

Author(s)

Matthew L. Fidler

Examples

gammap(1, 3)
gammap(1:3, 3)
gammap(1, 1:3)
Description
This is the gamma_p_derivative from the boost library

Usage
\texttt{gammapDer(a, z)}

Arguments
\begin{itemize}
  \item \texttt{a} The numeric \textquote{a} parameter in the upper incomplete gamma
  \item \texttt{z} The numeric \textquote{z} parameter in the upper incomplete gamma
\end{itemize}

Value
lowergamma results

Author(s)
Matthew L. Fidler

Examples
\begin{itemize}
  \item \texttt{gammapDer(1:3, 3)}
  \item \texttt{gammapDer(1, 1:3)}
\end{itemize}

Description
gammapInv and gammapInva: Inverses of normalized gammap function

Usage
\begin{itemize}
  \item \texttt{gammapInv(a, p)}
  \item \texttt{gammapInva(x, p)}
\end{itemize}
Arguments

- $a$: The numeric ‘a’ parameter in the upper incomplete gamma
- $p$: The numeric ‘p’ parameter in the upper incomplete gamma
- $x$: The numeric ‘x’ parameter in the upper incomplete gamma

Details

With the equation:

$$p = \text{gammap}(a, x)$$

The ‘gammapInv’ function returns a value ‘$x$’ that satisfies the equation above.

The ‘gammapInva’ function returns a value ‘$q$’ that satisfies the equation above.

NOTE: gammapInva is slow

Value

inverse gammap results

Author(s)

Matthew L. Fidler

Examples

```r
gammapInv(1:3, 0.5)
gammapInv(1, 1:3 / 3.1)
gammapInv(1:3, 1:3 / 3.1)
gammapInva(1:3, 1:3 / 3.1)
```

---

**Gammaq**: normalized upper incomplete gamma function

Description

This is the gamma_q from the boost library

Usage

```r
gammaq(a, z)
```

Arguments

- $a$: The numeric ‘a’ parameter in the normalized upper incomplete gamma
- $z$: The numeric ‘z’ parameter in the normalized upper incomplete gamma
Details

The gamma q function is given by:
\[ \gammaq = \frac{\text{uppergamma}(a, z)}{\gamma(a)} \]

Value

gammaq results

Author(s)

Matthew L. Fidler

Examples

\[ \gammaq(1, 3) \]
\[ \gammaq(1:3, 3) \]
\[ \gammaq(1, 1:3) \]

\[
\begin{array}{ll}
\text{gammaqInv} & \quad \text{gammaqInv and gammaqInva: Inverses of normalized gammaq function} \\
\end{array}
\]

Description

gammaqInv and gammaqInva: Inverses of normalized gammaq function

Usage

\[ \gammaqInv(a, q) \]
\[ \gammaqInva(x, q) \]

Arguments

\[ a \quad \text{The numeric 'a' parameter in the upper incomplete gamma} \]
\[ q \quad \text{The numeric 'q' parameter in the upper incomplete gamma} \]
\[ x \quad \text{The numeric 'x' parameter in the upper incomplete gamma} \]

Details

With the equation:
\[ q = \gammaq(a, x) \]
The 'gammaqInv' function returns a value 'x' that satisfies the equation above
The 'gammaqInva' function returns a value 'a' that satisfies the equation above
NOTE: gammaqInva is slow
Value

inverse gamma results

Author(s)

Matthew L. Fidler

Examples

gammaqInv(1:3, 0.5)
gammaqInv(1, 1:3 / 3)
gammaqInv(1:3, 1:3 / 3.1)
gammaqInv(1:3, 1:3 / 3.1)

Description

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

Usage

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

write.template.server(appDir)

write.template.ui(appDir, statevars)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appDir</td>
<td>a string with a directory where to store the shiny app, by default is &quot;shinyExample&quot;. The directory appDir will be created if it does not exist.</td>
</tr>
<tr>
<td>verbose</td>
<td>logical specifying whether to write messages as the shiny app is generated. Defaults to TRUE.</td>
</tr>
<tr>
<td>ODE.config</td>
<td>model name compiled and list of parameters sent to rxSolve().</td>
</tr>
</tbody>
</table>
statevars List of statevars passed to the write.template.ui() function. This usually isn’t called directly.

A PK/PD model is defined using rxode2(), 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:

\[
\begin{align*}
C2 &= \text{centr} / V2; \\
C3 &= \text{peri} / V3; \\
d/dt(\text{depot}) &= -KA \times \text{depot}; \\
d/dt(\text{centr}) &= KA \times \text{depot} - CL \times C2 - Q \times C2 + Q \times C3; \\
d/dt(\text{peri}) &= Q \times C2 - Q \times C3; \\
d/dt(\text{eff}) &= \text{Kin} - Kout \times (1 - C2 / (EC50 + C2)) \times \text{eff};
\end{align*}
\]

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.

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 rxode2 models.

See Also

rxode2(), eventTable(), and the package shiny (https://shiny.rstudio.com).

Examples

```r
# remove myapp when the example is complete
on.exit(unlink("myapp", recursive = TRUE, force = TRUE))
# create the shiny app example (template)
genShinyApp.template(appDir = "myapp")
# run the shiny app
if (requireNamespace("shiny", quietly=TRUE)) {
  library(shiny)
  runApp("myapp") # Won't launch in environments without browsers
}
```
**getRxThreads**

Get/Set the number of threads that rxode2 uses

### Description

Get/Set the number of threads that rxode2 uses

### Usage

- `getRxThreads( verbose = FALSE )`
- `setRxThreads( threads = NULL, percent = NULL, throttle = NULL )`
- `rxCores( verbose = FALSE )`

### Arguments

- **verbose**
  - Display the value of relevant OpenMP settings
- **threads**
  - NULL (default) rereads environment variables. 0 means to use all logical CPUs available. Otherwise a number >= 1
- **percent**
  - If provided it should be a number between 2 and 100; the percentage of logical CPUs to use. By default on startup, 50 percent.
- **throttle**
  - 2 (default) means that, roughly speaking, a single thread will be used when number subjects solved for is <=2, 2 threads when the number of all points is <=4, etc. The throttle is to speed up small data tasks (especially when repeated many times) by not incurring the overhead of managing multiple threads. The throttle will also suppress sorting which ID will be solved first when there are (nsubject solved)*throttle <= nthreads. In rxode2 this sorting occurs to minimize the time for waiting for another thread to finish. If the last item solved has a long solving time, all the other solving have to wait for that last costly solving to occur. If the items which are likely to take more time are solved first, this wait is less likely to have an impact on the overall solving time.
  - In rxode2 the IDs are sorted by the individual number of solving points (largest first). It also has a C interface that allows these IDs to be resorted by total time spent solving the equation. This allows packages like nlmixr to sort by solving time if needed.
  - Overall the the number of threads is throttled (restricted) for small tasks and sorting for IDs are suppressed.

### Value

- number of threads that rxode2 uses
ini.rxUi

Init block for rxode2/nlmixr models

Description

The ini block controls initial conditions for 'theta' (fixed effects), 'omega' (random effects), and 'sigma' (residual error) elements of the model.

Usage

## S3 method for class 'rxUi'
ini(x, ..., envir = parent.frame())

## S3 method for class function
ini(x, ..., envir = parent.frame())

## S3 method for class 'rxode2'
ini(x, ..., envir = parent.frame())

## S3 method for class 'rxModelVars'
ini(x, ..., envir = parent.frame())

ini(x, ..., envir = parent.frame())

## Default S3 method:
ini(x, ...)

Arguments

x     expression
...
   Other expressions for ini() function
envir the environment in which unevaluated model expressions is to be evaluated. May also be NULL, a list, a data frame, a pairlist or an integer as specified to sys.call.

Details

'theta' and 'sigma' can be set using either <- or = such as tvCL <- 1 or equivalently tvCL = 1. 'omega' can be set with a ~.

Parameters can be named or unnamed (though named parameters are preferred). A named parameter is set using the name on the left of the assignment while unnamed parameters are set without an assignment operator. tvCL <- 1 would set a named parameter of tvCL to 1. Unnamed parameters are set using just the value, such as 1.

For some estimation methods, lower and upper bounds can be set for 'theta' and 'sigma' values. To set a lower and/or upper bound, use a vector of values. The vector is c(lower, estimate, upper). The vector may be given with just the estimate (c(estimate)), the lower bound and
estimate (c(lower, estimate)), or all three (c(lower, estimate, upper)). To set an estimate and upper bound without a lower bound, set the lower bound to -Inf, c(-Inf, estimate, upper). When an estimation method does not support bounds, the bounds will be ignored with a warning.

'omega' values can be set as a single value or as the values of a lower-triangular matrix. The values may be set as either a variance-covariance matrix (the default) or as a correlation matrix for the off-diagonals with the standard deviations on the diagonals. Names may be set on the left side of the \( \sim \). To set a variance-covariance matrix with variance values of 2 and 3 and a covariance of -2.5 use \( \sim c(2, 2.5, 3) \). To set the same matrix with names of iivKa and iivCL, use iivKa + iivCL \( \sim c(2, 2.5, 3) \). To set a correlation matrix with standard deviations on the diagonal, use cor() like iivKa + iivCL \( \sim \) cor(\( -0.5, 3 \)).

Values may be fixed (and therefore not estimated) using either the name fixed at the end of the assignment or by calling fixed() as a function for the value to fix. For 'theta' and 'sigma', either the estimate or the full definition (including lower and upper bounds) may be included in the fixed setting. For example, the following are all effectively equivalent to set a 'theta' or 'sigma' to a fixed value (because the lower and upper bounds are ignored for a fixed value): tvCL <- fixed(1), tvCL <- fixed(0, 1), tvCL <- fixed(0, 1, 2), tvCL <- c(0, fixed(1), 2), or tvCL <- c(0, 1, fixed). For 'omega' assignment, the full block or none of the block must be set as fixed. Examples of setting an 'omega' value as fixed are: iivKa ~ fixed(1), iivKa + iivCL ~ fixed(1, 2, 3), or iivKa + iivCL ~ c(1, 2, 3, fixed). Anywhere that fixed is used, FIX, FIXED, or fix may be used equivalently.

For any value, standard mathematical operators or functions may be used to define the value. For example, exp(2) and 24*30 may be used to define a value anywhere that a number can be used (e.g. lower bound, estimate, upper bound, variance, etc.).

Values may be labeled using the label() function after the assignment. Labels are used to make reporting easier by giving a human-readable description of the parameter, but the labels do not have any effect on estimation. The typical way to set a label so that the parameter tvCL has a label of "Typical Value of Clearance (L/hr)" is tvCL <- 1; label("Typical Value of Clearance (L/hr)").

\texttt{rxode2/nlmixr2} will attempt to determine some back-transformations for the user. For example, \( \text{CL} \leftarrow \exp(\text{tvCL}) \) will detect that tvCL must be back-transformed by \( \exp() \) for easier interpretation. When you want to control the back-transformation, you can specify the back-transformation using backTransform() after the assignment. For example, to set the back-transformation to \( \exp() \), you can use tvCL <- 1; backTransform(\( \exp() \)).

\textbf{Value}

Ini block

\textbf{Author(s)}

Matthew Fidler
llikBeta  

*Calculate the log likelihood of the binomial function (and its derivatives)*

---

**Description**

Calculate the log likelihood of the binomial function (and its derivatives)

**Usage**

```r
llikBeta(x, shape1, shape2, full = FALSE)
```

**Arguments**

- `x` Observation
- `shape1, shape2` non-negative parameters of the Beta distribution.
- `full` Add the data frame showing `x`, mean, sd as well as the `fx` and derivatives

**Details**

In an `rxode2()` model, you can use `llikBeta()` but you have to use all arguments. You can also get the derivative of `shape1` and `shape2` with `llikBetaDshape1()` and `llikBetaDshape2()`.

**Value**

Data frame with `fx` for the log pdf value of with `dShape1` and `dShape2` that has the derivatives with respect to the parameters at the observation time-point

**Author(s)**

Matthew L. Fidler

**Examples**

```r
x <- seq(1e-4, 1 - 1e-4, length.out = 21)
llikBeta(x, 0.5, 0.5)
llikBeta(x, 1, 3, TRUE)
```

```r
et <- et(seq(1e-4, 1-1e-4, length.out=21))
et$shape1 <- 0.5
et$shape2 <- 1.5
```

```r
model <- rxode2(
```
llikBinom

```r
fx <- llikBeta(time, shape1, shape2)
dShape1 <- llikBetaDshape1(time, shape1, shape2)
dShape2 <- llikBetaDshape2(time, shape1, shape2)
```

```r
rxSolve(model, et)
```

---

### llikBinom

*Calculate the log likelihood of the binomial function (and its derivatives)*

#### Description

Calculate the log likelihood of the binomial function (and its derivatives)

#### Usage

```r
llikBinom(x, size, prob, full = FALSE)
```

#### Arguments

- **x**: Number of successes
- **size**: Size of trial
- **prob**: Probability of success
- **full**: Add the data frame showing x, mean, sd as well as the fx and derivatives

#### Details

In an `rxode2()` model, you can use `llikBinom()` but you have to use all arguments. You can also get the derivative of `prob` with `llikBinomDprob()`

#### Value

Data frame with `fx` for the pdf value of with `dProb` that has the derivatives with respect to the parameters at the observation time-point

#### Author(s)

Matthew L. Fidler
Examples

\[
\text{llikBinom}(46:54, 100, 0.5)
\]
\[
\text{llikBinom}(46:54, 100, 0.5, \text{TRUE})
\]

# In rxode2 you can use:

et <- et(46:54)
et$size <- 100
et$prob <- 0.5

model <- rxode2({
  fx <- llikBinom(time, size, prob)
  dProb <- llikBinomDprob(time, size, prob)
})

rxSolve(model, et)

llikCauchy

log likelihood of Cauchy distribution and it's derivatives (from stan)

Description

log likelihood of Cauchy distribution and it's derivatives (from stan)

Usage

llikCauchy(x, location = 0, scale = 1, full = FALSE)

Arguments

x Observation
dlocation, scale
full Add the data frame showing x, mean, sd as well as the fx and derivatives

Details

In an rxode2() model, you can use llikCauchy() but you have to use all arguments. You can also get the derivative of location and scale with llikCauchyDlocation() and llikCauchyDscale().

Value

data frame with fx for the log pdf value of with dLocation and dScale that has the derivatives with respect to the parameters at the observation time-point
**llikChisq**

log likelihood and derivatives for chi-squared distribution

**Description**

log likelihood and derivatives for chi-squared distribution

**Usage**

```r
llikChisq(x, df, full = FALSE)
```

**Arguments**

- **x**: variable that is distributed by chi-squared distribution
- **df**: degrees of freedom (non-negative, but can be non-integer).
- **full**: Add the data frame showing x, mean, sd as well as the fx and derivatives

**Details**

In an rxdode2() model, you can use llikChisq() but you have to use the x and df arguments. You can also get the derivative of df with llikChisqDdf().

**Examples**

```r
x <- seq(-3, 3, length.out = 21)
llikCauchy(x, 0, 1)
llikCauchy(x, 3, 1, full=TRUE)
et <- et(-3, 3, length.out=10)
et$location <- 0
et$scale <- 1
model <- rxode2({
  fx <- llikCauchy(time, location, scale)
  dLocation <- llikCauchyDlocation(time, location, scale)
  dScale <- llikCauchyDscale(time, location, scale)
})
rxSolve(model, et)
```
llikExp

log likelihood and derivatives for exponential distribution

Description

log likelihood and derivatives for exponential distribution

Usage

llikExp(x, rate, full = FALSE)

Arguments

x variable that is distributed by exponential distribution
rate vector of rates.
full Add the data frame showing x, mean, sd as well as the fx and derivatives

Details

In an rxode2() model, you can use llikExp() but you have to use the x and rate arguments. You can also get the derivative of rate with llikExpDrate().
**llikF**

log likelihood and derivatives for F distribution

**Value**

data frame with $fx$ for the log pdf value of with $dRate$ that has the derivatives with respect to the rate parameter the observation time-point

**Author(s)**

Matthew L. Fidler

**Examples**

```r
llikExp(1, 1:3)
llikExp(1, 1:3, full=TRUE)

# You can use rxode2 for these too:
et <- et(1:3)
et$x <- 1
model <- rxode2({
  fx <- llikExp(x, time)
  dRate <- llikExpDrate(x, time)
})
rxSolve(model, et)
```

---

**llikF**

log likelihood and derivatives for F distribution

**Description**

log likelihood and derivatives for F distribution

**Usage**

```r
llikF(x, df1, df2, full = FALSE)
```

**Arguments**

- **x**: variable that is distributed by f distribution
- **df1, df2**: degrees of freedom. Inf is allowed.
- **full**: Add the data frame showing x, mean, sd as well as the fx and derivatives

**Details**

In an `rxode2()` model, you can use `llikF()` but you have to use the x and rate arguments. You can also get the derivative of df1 and df2 with `llikFDdf1()` and `llikFDdf2()`. 

Value

data frame with fx for the log pdf value of with dDf1 and dDf2 that has the derivatives with respect to the df1/df2 parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

```r
x <- seq(0.001, 5, length.out = 100)
llikF(x^2, 1, 5)

model <- rxode2(
  fx <- llikF(time, df1, df2)
  dMean <- llikFDdf1(time, df1, df2)
  dSd <- llikFDdf2(time, df1, df2)
)

et <- et(x)
et$df1 <- 1
et$df2 <- 5
rxSolve(model, et)
```

llikGamma
log likelihood and derivatives for Gamma distribution

Description

log likelihood and derivatives for Gamma distribution

Usage

```r
llikGamma(x, shape, rate, full = FALSE)
```

Arguments

- `x`: variable that is distributed by gamma distribution
- `shape`: this is the distribution’s shape parameter. Must be positive.
- `rate`: this is the distribution’s rate parameters. Must be positive.
- `full`: Add the data frame showing x, mean, sd as well as the fx and derivatives
Details

In an `rxode2()` model, you can use `llikGamma()` but you have to use the x and rate arguments. You can also get the derivative of shape or rate with `llikGammaDshape()` and `llikGammaDrate()`.

Value

data frame with fx for the log pdf value of with dProb that has the derivatives with respect to the prob parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

```r
llikGamma(1, 1, 10)
# You can use this in `rxode2` too:
et <- et(seq(0.001, 1, length.out=10))
et$shape <- 1
et$rate <- 10
model <- rxode2({
  fx <- llikGamma(time, shape, rate)
  dShape <- llikGammaDshape(time, shape, rate)
  dRate <- llikGammaDrate(time, shape, rate)
})
rxSolve(model, et)
```

llikGeom

log likelihood and derivatives for Geom distribution

Description

log likelihood and derivatives for Geom distribution

Usage

```r
llikGeom(x, prob, full = FALSE)
```

Arguments

- **x**: variable distributed by a geom distribution
- **prob**: probability of success in each trial. \( 0 < \text{prob} < 1 \).
- **full**: Add the data frame showing x, mean, sd as well as the fx and derivatives
Details

In an `rxode2()` model, you can use `llikGeom()` but you have to use the x and rate arguments. You can also get the derivative of prob with `llikGeomDprob()`.

Value

data frame with fx for the log pdf value of with dProb that has the derivatives with respect to the prob parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

```r
llikGeom(1:10, 0.2)
et <- et(1:10)
et$prob <- 0.2
model <- rxode2({
  fx <- llikGeom(time, prob)
  dProb <- llikGeomDprob(time, prob)
})
rxSolve(model, et)
```

llikNbinom `llikNbinom` Calculate the log likelihood of the negative binomial function (and its derivatives)

Description

Calculate the log likelihood of the negative binomial function (and its derivatives)

Usage

```r
llikNbinom(x, size, prob, full = FALSE)
```

Arguments

- `x` Number of successes
- `size` Size of trial
- `prob` probability of success
- `full` Add the data frame showing x, mean, sd as well as the fx and derivatives
Details

In an rxode2() model, you can use llikNbinom() but you have to use all arguments. You can also get the derivative of prob with llikNbinomDprob().

Value

data frame with fx for the pdf value of with dProb that has the derivatives with respect to the parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

llikNbinom(46:54, 100, 0.5)

llikNbinom(46:54, 100, 0.5, TRUE)

# In rxode2 you can use:

et <- et(46:54)
et$size <- 100
et$prob <- 0.5

model <- rxode2(
  fx <- llikNbinom(time, size, prob)
  dProb <- llikNbinomDprob(time, size, prob)
)

rxSolve(model, et)
Arguments

- **x**: Number of successes
- **size**: Size of trial
- **mu**: μ parameter for negative binomial
- **full**: Add the data frame showing x, mean, sd as well as the fx and derivatives

Details

In an `rxode2()` model, you can use `llikNbinomMu()` but you have to use all arguments. You can also get the derivative of μ with `llikNbinomMuDmu()`

Value

- Data frame with `fx` for the pdf value of with `dProb` that has the derivatives with respect to the parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

```r
llikNbinomMu(46:54, 100, 40)
llikNbinomMu(46:54, 100, 40, TRUE)
et <- et(46:54)
et$size <- 100
et$mu <- 40
model <- rxode2(
  fx <- llikNbinomMu(time, size, mu)
  dProb <- llikNbinomMuDmu(time, size, mu)
)
rxSolve(model, et)
```

```r
llikNorm
Log likelihood for normal distribution
```

Description

Log likelihood for normal distribution
Usage

llikNorm(x, mean = 0, sd = 1, full = FALSE)

Arguments

x  Observation
mean  Mean for the likelihood
sd  Standard deviation for the likelihood
full  Add the data frame showing x, mean, sd as well as the fx and derivatives

Details

In an rxode2() model, you can use llikNorm() but you have to use all arguments. You can also get the derivatives with llikNormDmean() and llikNormDsd()

Value

data frame with fx for the pdf value of with dMean and dSd that has the derivatives with respect to the parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

llikNorm(0)

llikNorm(seq(-2,2,length.out=10), full=TRUE)

# With rxode2 you can use:

et <- et(-3, 3, length.out=10)
et$mu <- 0
et$sigma <- 1

model <- rxode2({
  fx <- llikNorm(time, mu, sigma)
  dMean <- llikNormDmean(time, mu, sigma)
  dSd <- llikNormDsd(time, mu, sigma)
})

ret <- rxSolve(model, et)
ret
llikPois

log-likelihood for the Poisson distribution

Description

log-likelihood for the Poisson distribution

Usage

llikPois(x, lambda, full = FALSE)

Arguments

x  non negative integers
lambda  non-negative means
full  Add the data frame showing x, mean, sd as well as the fx and derivatives

Details

In an rxode2() model, you can use llikPois() but you have to use all arguments. You can also get the derivatives with llikPoisDlambda()

Value

data frame with fx for the pdf value of with dLambda that has the derivatives with respect to the parameters at the observation time-point

Author(s)

Matthew L. Fidler

Examples

llikPois(0:7, lambda = 1)

llikPois(0:7, lambda = 4, full=TRUE)

# In rxode2 you can use:

et <- et(0:10)
et$lambda <- 0.5

model <- rxode2({
  fx <- llikPois(time, lambda)
  dLambda <- llikPoisDlambda(time, lambda)
})
**llikT**

Log likelihood of T and it's derivatives (from stan)

**Description**

Log likelihood of T and it's derivatives (from stan)

**Usage**

llikT(x, df, mean = 0, sd = 1, full = FALSE)

**Arguments**

- **x**: Observation
- **df**: degrees of freedom (> 0, maybe non-integer). df = Inf is allowed.
- **mean**: Mean for the likelihood
- **sd**: Standard deviation for the likelihood
- **full**: Add the data frame showing x, mean, sd as well as the fx and derivatives

**Details**

In an `rxode2()` model, you can use `llikT()` but you have to use all arguments. You can also get the derivative of df, mean and sd with `llikTdf()`, `llikTDmean()` and `llikTdsd()`.

**Value**

data frame with fx for the log pdf value of with dDf dMean and dSd that has the derivatives with respect to the parameters at the observation time-point

**Author(s)**

Matthew L. Fidler

**Examples**

```r
x <- seq(-3, 3, length.out = 21)
llikT(x, 7, 0, 1)
llikT(x, 15, 0, 1, full = TRUE)
et <- et(-3, 3, length.out = 10)
et$nu <- 7
```
et$mean <- 0
et$sd <- 1

model <- rxode2{
  fx <- llikT(time, nu, mean, sd)
dDf <- llikTdf(time, nu, mean, sd)
dMean <- llikTmean(time, nu, mean, sd)
dSd <- llikTsd(time, nu, mean, sd)
}

rxSolve(model, et)

---

**llikUnif**

*log likelihood and derivatives for Unif distribution*

**Description**

log likelihood and derivatives for Unif distribution

**Usage**

`llikUnif(x, alpha, beta, full = FALSE)`

**Arguments**

- `x` variable distributed by a uniform distribution
- `alpha` is the lower limit of the uniform distribution
- `beta` is the upper limit of the distribution
- `full` Add the data frame showing x, mean, sd as well as the fx and derivatives

**Details**

In an `rxode2()` model, you can use `llikUnif()` but you have to use the `x` and `rate` arguments. You can also get the derivative of `alpha` or `beta` with `llikUnifDalpha()` and `llikUnifDbeta()`.

**Value**

data frame with `fx` for the log pdf value of with `dProb` that has the derivatives with respect to the `prob` parameters at the observation time-point

**Author(s)**

Matthew L. Fidler
**llikWeibull**

*log likelihood and derivatives for Weibull distribution*

**Examples**

```r
llikUnif(1, -2, 2)

et <- et(seq(1,1, length.out=4))
et$alpha <- -2
et$beta <- 2

model <- rxode2({
  fx <- llikUnif(time, alpha, beta)
  dAlpha<- llikUnifDalpha(time, alpha, beta)
  dBeta <- llikUnifDbeta(time, alpha, beta)
})

rxSolve(model, et)
```

**Description**

log likelihood and derivatives for Weibull distribution

**Usage**

```r
llikWeibull(x, shape, scale, full = FALSE)
```

**Arguments**

- `x` variable distributed by a Weibull distribution
- `shape, scale` shape and scale parameters, the latter defaulting to 1.
- `full` Add the data frame showing x, mean, sd as well as the fx and derivatives

**Details**

In an `rxode2()` model, you can use `llikWeibull()` but you have to use the `x` and `rate` arguments. You can also get the derivative of `shape` or `scale` with `llikWeibullDshape()` and `llikWeibullDscale()`.

**Value**

data frame with `fx` for the log pdf value of with `dProb` that has the derivatives with respect to the `prob` parameters at the observation time-point

**Author(s)**

Matthew L. Fidler
logit

logit and inverse logit (expit) functions

Description

logit and inverse logit (expit) functions

Usage

logit(x, low = 0, high = 1)

expit(alpha, low = 0, high = 1)

logitNormInfo(mean = 0, sd = 1, low = 0, high = 1, abs.tol = 1e-06, ...)

probitNormInfo(mean = 0, sd = 1, low = 0, high = 1, abs.tol = 1e-06, ...)

Arguments

x Input value(s) in range [low,high] to translate -Inf to Inf
low Lowest value in the range
high Highest value in the range
alpha Infinite value(s) to translate to range of [low, high]
mean logit-scale mean
sd logit-scale standard deviation
abs.tol absolute accuracy requested.
... other parameters passed to integrate()
Details

logit is given by:
\[ \text{logit}(p) = -\log(1/p - 1) \]
where:
\[ p = x - \text{low}/\text{high} - \text{low} \]
expit is given by:
\[ \text{expit}(p, \text{low}, \text{high}) = (\text{high} - \text{low})/(1+\exp(-\alpha)) + \text{low} \]
The `logitNormInfo()` gives the mean, variance and coefficient of variability on the untransformed scale.

Value

values from logit and expit

Examples

logit(0.25)
expit(-1.09)

logitNormInfo(logit(0.25), sd = 0.1)

logitNormInfo(logit(1, 0, 10), sd = 1, low = 0, high = 10)

lowergamma

lowergamma: upper incomplete gamma function

Description

This is the tgamma_lower from the boost library

Usage

lowergamma(a, z)

Arguments

a The numeric 'a' parameter in the upper incomplete gamma
z The numeric 'z' parameter in the upper incomplete gamma

Details

The lowergamma function is given by:
\[ \text{lowergamma}(a, z) = \int_0^z t^{a-1} \cdot e^{-t} \, dt \]
Value
lowergamma results

Author(s)
Matthew L. Fidler

Examples
lowergamma(1, 3)
lowergamma(1:3, 3)
lowergamma(1, 1:3)

Description
Model block for rxode2/nlmixr models

Usage

```r
## S3 method for class 'function'
model(x, ..., append = FALSE, auto = TRUE, envir = parent.frame())

## S3 method for class 'rxUi'
model(x, ..., append = FALSE, auto = TRUE, envir = parent.frame())

## S3 method for class 'rxode2'
model(x, ..., append = FALSE, auto = TRUE, envir = parent.frame())

## S3 method for class 'rxModelVars'
model(x, ..., append = FALSE, auto = TRUE, envir = parent.frame())

model(
  x,
  ...
  append = FALSE,
  auto = getOption("rxode2.autoVarPiping", TRUE),
  envir = parent.frame()
)

## Default S3 method:
model(x, ..., append = FALSE, envir = parent.frame())
```
odeMethodToInt

Conversion between character and integer ODE integration methods for rxode2

Description

If NULL is given as the method, all choices are returned as a named vector.

Usage

odeMethodToInt(method = c("liblsoda", "lsoda", "dop853", "indLin"))
Arguments

method The method for solving ODEs. Currently this supports:
- "liblsoda" thread safe lsoda. This supports parallel thread-based solving, and ignores user Jacobian specification.
- "lsoda" – LSODA solver. Does not support parallel thread-based solving, but allows user Jacobian specification.
- "indLin" – Solving through inductive linearization. The rxode2 dll must be setup specially to use this solving routine.

Value

An integer for the method (unless the input is NULL, in which case, see the details)

Description

Plot rxode2 objects

Usage

## S3 method for class 'rxSolve'
plot(x, y, ..., log = "", xlab = "Time", ylab = ")

## S3 method for class 'rxSolveConfint1'
plot(x, y, ..., xlab = "Time", ylab = ", log = "")

## S3 method for class 'rxSolveConfint2'
plot(x, y, ..., xlab = "Time", ylab = ", log = "")

Arguments

x rxode2 object to plot
y Compartments or left-hand-side values to plot either as a bare name or as a character vector
... Ignored
log Should "" (neither x nor y), "x", "y", or "xy" (or "yx") be log-scale?
xlab, ylab The x and y axis labels

Value

A ggplot2 object
**probit**

See Also

Other `rxode2` plotting: `rxTheme()`

---

**probit**  *probit and inverse probit functions*

**Description**

probit and inverse probit functions

**Usage**

```r
probit(x, low = 0, high = 1)
probitInv(x, low = 0, high = 1)
```

**Arguments**

- `x` Input value(s) in range `[low,high]` to translate -Inf to Inf
- `low` Lowest value in the range
- `high` Highest value in the range

**Value**

values from `probit`, `probitInv` and `probitNormInfo`

**Examples**

```r
probit(0.25)
probitInv(-0.674)
probitNormInfo(probit(0.25), sd = 0.1)
probitNormInfo(probit(1, 0, 10), sd = 1, low = 0, high = 10)
```
**rxAllowUnload**  
*Allow unloading of dlls*

**Description**

Allow unloading of dlls

**Usage**

```plaintext
rxAllowUnload(allow)
```

**Arguments**

- **allow**: boolean indicating if garbage collection will unload of rxode2 dlls.

**Value**

Boolean allow; called for side effects

**Author(s)**

Matthew Fidler

**Examples**

```plaintext
# Garbage collection will not unload un-used rxode2 dlls
rxAllowUnload(FALSE);

# Garbage collection will unload unused rxode2 dlls
rxAllowUnload(TRUE);
```

**rxAppendModel**  
*Append two rxui models together*

**Description**

Append two rxui models together

**Usage**

```plaintext
rxAppendModel(model1, model2)
```

**Arguments**

- **model1**: rxUi model 1
- **model2**: rxUi model 2
Value

New model with both models appended together

Author(s)

Matthew L. Fidler

Examples

```r
cocmt <- function() {
  ini({
    tka <- exp(0.45) # Ka
    tcl <- exp(1) # Cl
    tv <- exp(3.45); # log V
    ## the label("Label name") works with all models
    add.sd <- 0.7
  })
  model({
    ka <- tka
    cl <- tcl
    v <- tv
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  } }
}

idr <- function() {
  ini({
    tkin <- log(1)
    tkout <- log(1)
    tic50 <- log(10)
    gamma <- fix(1)
    idr.sd <- 1
  })
  model({
    kin <- exp(tkin)
    kout <- exp(tkout)
    ic50 <- exp(tic50)
    d/dt(eff) <- kin - kout*(1-ceff^gamma/(ic50^gamma+ceff^gamma))
    eff ~ add(idr.sd)
  } }
}

rxAppendModel(cocmt %>% model(ceff=cp,append=TRUE), idr)
```
**rxAssignControlValue**  
Assign Control Variable

**Description**

Assign Control Variable

**Usage**

```r
rxAssignControlValue(ui, option, value)
```

**Arguments**

- `ui`: rxode2 ui function
- `option`: Option name in the control to modify
- `value`: Value of control to modify

**Value**

Nothing; called for the side effects

**Author(s)**

Matthew L. Fidler

---

**rxAssignPtr**  
Assign pointer based on model variables

**Description**

Assign pointer based on model variables

**Usage**

```r
rxAssignPtr(object = NULL)
```

**Arguments**

- `object`: rxode2 family of objects

**Value**

nothing, called for side effects
Simulate beta variable from threefry generator

Description

Care should be taken with this method not to encounter the birthday problem, described https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage

```r
rbeta(shape1, shape2, n = 1L, ncores = 1L)
```

Arguments

- `shape1, shape2`: non-negative parameters of the Beta distribution.
- `n`: number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `ncores`: Number of cores for the simulation

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

beta random deviates

Examples

```r
## Use threefry engine
```
rxbinom(0.5, 0.5, n = 10) # with rxbeta you have to explicitly state n
rxbeta(5, 1, n = 10, ncores = 2) # You can parallelize the simulation using openMP
rxbeta(1, 3)

## This example uses `rxbeta` directly in the model
rx <- rxode2({
a <- rxbeta(2, 2)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)

---

**rxbinom**  
Simulate Binomial variable from threefry generator

**Description**

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

`rxbinom(size, prob, n = 1L, ncores = 1L)`

**Arguments**

- `size` number of trials (zero or more).
- `prob` probability of success on each trial.
- `n` number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `ncores` Number of cores for the simulation
  - `rxnorm` simulates using the threefry sitmo generator.
  - `rxnormV` used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of `rxnorm`. It is no longer supported in `rxode2({})` blocks.
Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the rxode2 environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the rxode2 engine with rxSetSeed().

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

binomial random deviates

Examples

```r
## Use threefry engine

rxbinom(10, 0.9, n = 10) # with rxbinom you have to explicitly state n
rxbinom(3, 0.5, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxbinom(4, 0.7)

## This example uses `rxbinom` directly in the model

rx <- rxode2({
  a <- rxbinom(1, 0.5)
})

et <- et(1, id = 1:2)

s <- rxSolve(rx, et)
```

---

**rxcauchy**

Simulate Cauchy variable from threefry generator

Description

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.
Usage

```r
taxcauchy(location = 0, scale = 1, n = 1L, ncores = 1L)
```

Arguments

- `location`, `scale` location and scale parameters.
- `n` number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `ncores` Number of cores for the simulation

`rxnorm` simulates using the threefry sitmo generator.
`rxnormV` used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of `rxnorm`. It is no longer supported in `rxode2()` blocks.

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

Cauchy random deviates

Examples

```r
## Use threefry engine

rcauchy(0, 1, n = 10) # with rcauchy you have to explicitly state n
rcauchy(0.5, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rcauchy(3)

## This example uses 'rcauchy' directly in the model

rx <- rxode2(
  a <- rcauchy(2)
)

et <- et(1, id = 1:2)
```
s <- rxSolve(rx, et)

---

**rxchisq**  
*Simulate chi-squared variable from threelfry generator*

---

**Description**

Care should be taken with this method not to encounter the birthday problem, described [here](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threelfry, this currently generates one random deviate from the uniform distribution to seed the engine threelfry and then run the code.

**Usage**

```
rxchisq(df, n = 1L, ncores = 1L)
```

**Arguments**

- `df`  
  degrees of freedom (non-negative, but can be non-integer).
- `n`  
  number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `ncores`  
  Number of cores for the simulation

**Values used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of `rxnorm`. It is no longer supported in `rxode2()` blocks**

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`.

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

**Value**

chi squared random deviates
Examples

```r
## Use threefry engine
rxchisq(0.5, n = 10) # with rxchisq you have to explicitly state n
rxchisq(5, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxchisq(1)

## This example uses `rxchisq` directly in the model
rx <- rxode2(
  a <- rxchisq(2)
)
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

---

**rxClean**

Cleanup anonymous DLLs by unloading them

Description

This cleans up any rxode2 loaded DLLs

Usage

```r
rxClean(wd)
```

Arguments

- `wd` What directory should be cleaned; (DEPRECATED), this no longer does anything.

This unloads all rxode2 anonymous dlls.

Value

TRUE if successful

Author(s)

Matthew L. Fidler
rxCompile

Compile a model if needed

Description

This is the compilation workhorse creating the rxode2 model DLL files.

Usage

rxCompile(model, dir, prefix, force = FALSE, modName = NULL, package = NULL, ...)

## S3 method for class 'rxModelVars'
rxCompile(model, dir = NULL, prefix = NULL, force = FALSE, modName = NULL, package = NULL, ...)

## S3 method for class 'character'
rxCompile(model, dir = NULL, prefix = NULL, force = FALSE, modName = NULL, package = NULL, ...)

## S3 method for class 'rxDll'
rxCompile(model, ...)

## S3 method for class 'rxode2'
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 `rxode2` 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.

force is a boolean stating if the (re)compile should be forced if `rxode2` 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.

package Package name for pre-compiled binaries.

... Other arguments sent to the `rxTrans()` function.

Value

An rxDll object that has the following components

• dllDLL path
• model model specification
• .cA function to call C code in the correct context from the DLL using the `.C()` function.
• .callA function to call C code in the correct context from the DLL using the `.Call()` function.
• argsA list of the arguments used to create the rxDll object.

Author(s)

Matthew L. Fidler

See Also

`rxode2()`
**rxControlUpdateSens**

This updates the tolerances based on the sensitivity equations

**Description**

This assumes the normal ODE equations are the first equations and the ODE is expanded by the forward sensitivities or other type of sensitivity (like adjoint)

**Usage**

```
rxControlUpdateSens(rxControl, sensCmt = NULL, ncmt = NULL)
```

**Arguments**

- `rxControl`: Input list or `rxControl` type of list
- `sensCmt`: Number of sensitivity compartments
- `ncmt`: Number of compartments

**Value**

Updated `rxControl` where $\texttt{atol}$, $\texttt{rtol}$, $\texttt{ssAtol}$, $\texttt{ssRtol}$ are updated with different sensitivities for the normal ODEs (first) and a different sensitivity for the larger compartments (sensitivities).

**Author(s)**

Matthew L. Fidler

**Examples**

```r
tmp <- rxControl()
tmp2 <- rxControlUpdateSens(tmp, 3, 6)
tmp2$atol
tmp2$rtol
tmp2$ssAtol
tmp2$ssRtol
```
**rxCreateCache**

*This will create the cache directory for rxode2 to save between sessions*

**Description**

When run, if the `R_user_dir` for rxode2’s cache isn’t present, create the cache

**Usage**

```r
rxCreateCache()
```

**Value**

Nothing

**Author(s)**

Matthew Fidler

---

**rxD**

*Add to rxode2’s derivative tables*

**Description**

Add to rxode2’s derivative tables

**Usage**

```r
rxD(name, derivatives)
```

**Arguments**

- **name**: Function Name
- **derivatives**: A list of functions. Each function takes the same number of arguments as the original function. The first function will construct the derivative with respect to the first argument; The second function will construct the derivative with respect to the second argument, and so on.

**Value**

Nothing

**Author(s)**

Matthew Fidler
Examples

## Add an arbitrary list of derivative functions
## In this case the fun(x,y) is assumed to be 0.5*x^2+0.5*y^2

```r
rxD("fun", list(
    function(x, y) {
        return(x)
    },
    function(x, y) {
        return(y)
    }
))
```

### Description

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

### Usage

```r
rxDelete(obj)
```

### Arguments

- **obj**
  - `rxode2` family of objects

### Value

A boolean stating if the operation was successful.

### Author(s)

Matthew L. Fidler

### Description

This calculates the derived parameters based on what is provided in a data frame or arguments.

### Usage

```r
rxDerived(..., verbose = FALSE, digits = 0)
```
Arguments

The input can be:

- A data frame with PK parameters in it; This should ideally be a data frame with one pk parameter per row since it will output a data frame with one PK parameter per row.
- PK parameters as either a vector or a scalar

verbose

boolean that when TRUE provides a message about the detected pk parameters and the detected compartmental model. By default this is FALSE.

digits

represents the number of significant digits for the output; If the number is zero or below (default), do not round.

Value

Return a data.frame of derived PK parameters for a 1-, 2-, or 3-compartment linear model given provided clearances and volumes based on the inferred model type.

The model parameters that will be provided in the data frame are:

- vc: Central Volume (for 1-, 2- and 3- compartment models)
- kel: First-order elimination rate (for 1-, 2-, and 3-compartment models)
- k12: First-order rate of transfer from central to first peripheral compartment; (for 2- and 3-compartment models)
- k21: First-order rate of transfer from first peripheral to central compartment, (for 2- and 3-compartment models)
- k13: First-order rate of transfer from central to second peripheral compartment; (3-compartment model)
- k31: First-order rate of transfer from second peripheral to central compartment (3-compartment model)
- vp: Peripheral Volume (for 2- and 3-compartment models)
- vp2: Peripheral Volume for 3rd compartment (3-compartment model)
- vss: Volume of distribution at steady state; (1-, 2-, and 3-compartment models)
- t12alpha: \( t_{1/2,\alpha} \); (1-, 2-, and 3-compartment models)
- t12beta: \( t_{1/2,\beta} \); (2- and 3-compartment models)
- t12gamma: \( t_{1/2,\gamma} \); (3-compartment model)
- alpha: \( \alpha \); (1-, 2-, and 3-compartment models)
- beta: \( \beta \); (2- and 3-compartment models)
- gamma: \( \beta \); (3-compartment model)
- A: true A; (1-, 2-, and 3-compartment models)
- B: true B; (2- and 3-compartment models)
- C: true C; (3-compartment model)
- fracA: fractional A; (1-, 2-, and 3-compartment models)
- fracB: fractional B; (2- and 3-compartment models)
- fracC: fractional C; (3-compartment model)
RxDfdy

Author(s)

Matthew Fidler and documentation from Justin Wilkins, <justin.wilkins@occams.com>

References

Shafer S. L. CONVERT.XLS


Examples

```r
## Note that rxode2 parses the names to figure out the best PK parameter
params <- rxDerived(cl = 29.4, v = 23.4, Vp = 114, vp2 = 4614, q = 270, q2 = 73)

## That is why this gives the same results as the value before
params <- rxDerived(CL = 29.4, V1 = 23.4, V2 = 114, V3 = 4614, Q2 = 270, Q3 = 73)

## You may also use micro-constants alpha/beta etc.
params <- rxDerived(k12 = 0.1, k21 = 0.2, k13 = 0.3, k31 = 0.4, kel = 10, v = 10)

## or you can mix vectors and scalars
params <- rxDerived(CL = 29.4, V = 1:3)

## If you want, you can round to a number of significant digits
## with the `digits` argument:
params <- rxDerived(CL = 29.4, V = 1:3, digits = 2)
```

rxDfdy

Jacobian and parameter derivatives

Description

Return Jacobian and parameter derivatives

Usage

```r
rxDfdy(obj)
```

Arguments

- `obj` 
  rxode2 family of objects
Value

A list of the jacobian parameters defined in this rxode2 object.

Author(s)

Matthew L. Fidler

See Also

Other Query model information: `rxInits()`, `rxLhs()`, `rxModelVars()`, `rxParams()`, `rxState()`

---

**rxexp**

*Simulate exponential variable from threefry generator*

Description

Care should be taken with this method not to encounter the birthday problem, described https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage

`rxexp(rate, n = 1L, ncores = 1L)`

Arguments

- **rate**: vector of rates.
- **n**: number of observations. If `length(n) > 1`, the length is taken to be the number required.
- **ncores**: Number of cores for the simulation

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the rxode2 environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the rxode2 engine with `rxSetSeed()`

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.
Value

exponential random deviates

Examples

```r
## Use threefry engine
rxexp(0.5, n = 10) # with rxexp you have to explicitly state n
rxexp(5, n = 10, ncores = 2) # You can parallelize the simulation using openMP
rxexp(1)

## This example uses `rxexp` directly in the model
rx <- rxode2({
a <- rxexp(2)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

---

**rxf**  
Simulate F variable from threefry generator

Description

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage

```r
rxf(df1, df2, n = 1L, ncores = 1L)
```

Arguments

- `df1, df2`  
degrees of freedom. Inf is allowed.
- `n`  
number of observations. If `length(n) > 1`, the length is taken to be the number required.
ncores  Number of cores for the simulation

rxnorm simulates using the threefry sitmo generator.

rxnormV used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of rxnorm. It is no longer supported in rxode2({}) blocks.

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the rxode2 environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the rxode2 engine with rxSetSeed()

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

f random deviates

Examples

```r
## Use threefry engine

rxf(0.5, 0.5, n = 10) # with rxf you have to explicitly state n
rxf(5, 1, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxf(1, 3)

## This example uses ‘rxf’ directly in the model

rx <- rxode2({
  a <- rxf(2, 2)
})

et <- et(1, id = 1:2)

s <- rxSolve(rx, et)
```
rxFun

Add user function to rxode2

Description

This adds a user function to rxode2 that can be called. If needed, these functions can be differentiated by numerical differences or by adding the derivatives to rxode2's internal derivative table with rxD()

Usage

rxFun(name, args, cCode)

rxRmFun(name)

Arguments

name This gives the name of the user function
args This gives the arguments of the user function
cCode This is the C-code for the new function

Value

nothing

Author(s)

Matthew L. Fidler

Examples

## Right now rxode2 is not aware of the function f
## Therefore it cannot translate it to symengine or
## Compile a model with it.
try(rxode2("a=fun(a,b,c)"))

## Note for this approach to work, it cannot interfere with C
## function names or reserved rxode2 speical terms. Therefore
## f(x) would not work since f is an alias for bioaviability.

fun <- "
double fun(double a, double b, double c) {
    return a*a+b*a+c;
}
"  ## C-code for function
rxFun("fun", c("a", "b", "c"), fun) ## Added function

## Now rxode2 knows how to translate this function to symengine
rxToSE("fun(a,b,c)"

## And will take a central difference when calculating derivatives
rxFromSE("Derivative(fun(a,b,c),a)"

## Of course, you could specify the derivative table manually
rxD("fun", list(
    function(a, b, c) {
        paste0("2\times", a, " +", b)
    },
    function(a, b, c) {
        return(a)
    },
    function(a, b, c) {
        return("0.0")
    }
))

rxFromSE("Derivative(fun(a,b,c),a)"

# You can also remove the functions by `rxRmFun`
rxRmFun("fun")

---

**rxgamma**

*Simulate gamma variable from threefry generator*

**Description**

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxgamma(shape, rate = 1, n = 1L, ncores = 1L)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
<td>The shape of the gamma random variable</td>
</tr>
<tr>
<td>rate</td>
<td>an alternative way to specify the scale.</td>
</tr>
</tbody>
</table>
number of observations. If length(n) > 1, the length is taken to be the number required.

ncore

Number of cores for the simulation. 

\texttt{rxnorm} simulates using the threefry sitmo generator. 

\texttt{rxnormV} used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of \texttt{rxnorm}. It is no longer supported in \texttt{rxode2} blocks.

\textbf{Details}

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the \texttt{rxode2} environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the \texttt{rxode2} engine with \texttt{rxSetSeed()}

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

\textbf{Value}

gamma random deviates

\textbf{Examples}

### Use threefry engine

\begin{verbatim}
rxgamma(0.5, n = 10) # with rxgamma you have to explicitly state n
rxgamma(5, n = 10, ncores = 2) # You can parallelize the simulation using openMP
rxgamma(1)
\end{verbatim}

### This example uses `rxbeta` directly in the model

\begin{verbatim}
rx <- rxode2{
  a <- rxgamma(2)
}

et <- et(1, id = 1:2)

s <- rxSolve(rx, et)
\end{verbatim}
Simulate geometric variable from threelfry generator

Description

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threelfry, this currently generates one random deviate from the uniform distribution to seed the engine threelfry and then run the code.

Usage

```r
rxgeom(prob, n = 1L, ncores = 1L)
```

Arguments

- `prob`: probability of success in each trial. \(0 < \text{prob} \leq 1\).
- `n`: number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `ncores`: Number of cores for the simulation.

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`.

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

geometric random deviates

Examples

```r
## Use threelfry engine
```
rxgeom(0.5, n = 10) # with rxgeom you have to explicitly state n
rxgeom(0.25, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxgeom(0.75)

## This example uses `rxgeom` directly in the model

rx <- rxode2({
  a <- rxgeom(0.24)
})

et <- et(1, id = 1:2)
s <- rxSolve(rx, et)

---

**rxGetControl**

rxGetControl option from ui

**Description**

rxGetControl option from ui

**Usage**

rxGetControl(ui, option, default)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ui</td>
<td>rxode2 ui object</td>
</tr>
<tr>
<td>option</td>
<td>Option to get</td>
</tr>
<tr>
<td>default</td>
<td>Default value</td>
</tr>
</tbody>
</table>

**Value**

Option (if present) or default value

**Author(s)**

Matthew L. Fidler
**rxGetLin**

*Get the linear compartment model true function*

**Description**

Get the linear compartment model true function

**Usage**

```r
rxGetLin(
  model,
  linCmtSens = c("linCmtA", "linCmtB", "linCmtC"),
  verbose = 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 `rxode2` Syntax below.
- `linCmtSens` The method to calculate the linCmt() solutions
- `verbose` When `TRUE` be verbose with the linear compartmental model

**Value**

model with linCmt() replaced with linCmtA()

**Author(s)**

Matthew Fidler

---

**rxGetrxode2**

*Get rxode2 model from object*

**Description**

Get rxode2 model from object

**Usage**

```r
rxGetrxode2(obj)
```
**Arguments**

obj  rxode2 family of objects

**Value**

rxode2 model

---

**Description**

Format rxSolve and related objects as html.

**Usage**

```
rxHtml(x, ...)
```

## S3 method for class 'rxSolve'

```
rxHtml(x, ...)
```

**Arguments**

x  rxode2 object

...  Extra arguments sent to kable

**Value**

html code for rxSolve object

**Author(s)**

Matthew L. Fidler
### rxIndLinState

*Set the preferred factoring by state*

**Description**

Set the preferred factoring by state

**Usage**

```r
rxIndLinState(preferred = NULL)
```

**Arguments**

- `preferred` A list of each state's preferred factorization

**Value**

Nothing

**Author(s)**

Matthew Fidler

---

### rxIndLinStrategy

*This sets the inductive linearization strategy for matrix building*

**Description**

When there is more than one state in a ODE that cannot be separated this specifies how it is incorporated into the matrix exponential.

**Usage**

```r
rxIndLinStrategy(strategy = c("curState", "split"))
```

**Arguments**

- `strategy` The strategy for inductive linearization matrix building
  - `curState` Prefer parameterizing in terms of the current state, followed by the first state observed in the term.
  - `split` Split the parameterization between all states in the term by dividing each by the number of states in the term and then adding a matrix term for each state.
Value

Nothing

Author(s)

Matthew L. Fidler

---

**rxIndLin**

*Inductive linearization solver*

**Description**

Inductive linearization solver

**Arguments**

- **cSub** = Current subject number
- **op** • rxode2 solving options
- **tp** • Prior time point/time zero
- **yp** • Prior state; vector size = neq; Final state is updated here
- **tf** • Final Time
- **InfusionRate** = Rates of each compartment; vector size = neq
- **on** Indicator for if the compartment is "on"
- **cache** 0 = no Cache When doIndLin == 0, cache > 0 = nInf-1
- **ME** the rxode2 matrix exponential function
- **IndF** The rxode2 Inductive Linearization function F

**Value**

Returns a status for solving

1 = Successful solve

-1 = Maximum number of iterations reached when doing inductive linearization
**rxInv**

Invert matrix using RcppArmadillo.

**Description**

Invert matrix using RcppArmadillo.

**Usage**

`rxInv(matrix)`

**Arguments**

- `matrix`: matrix to be inverted.

**Value**

inverse or pseudo inverse of matrix.

---

**rxIsCurrent**

Checks if the rxode2 object was built with the current build

**Description**

Checks if the rxode2 object was built with the current build

**Usage**

`rxIsCurrent(obj)`

**Arguments**

- `obj`: rxode2 family of objects

**Value**

boolean indicating if this was built with current rxode2
**rxLhs**  
*Left handed Variables*

**Description**  
This returns the model calculated variables

**Usage**  
`rxLhs(obj)`

**Arguments**  
- `obj`: rxode2 family of objects

**Value**  
a character vector listing the calculated parameters

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

**See Also**  
`rxode2`  
Other Query model information: `rxDfdy()`, `rxInits()`, `rxModelVars()`, `rxParams()`, `rxState()`

---

**rxLock**  
*Lock/unlocking of rxode2 dll file*

**Description**  
Lock/unlocking of rxode2 dll file

**Usage**  
`rxLock(obj)`  
`rxUnlock(obj)`

**Arguments**  
- `obj`: A rxode2 family of objects

**Value**  
nothing; called for side effects
Simulate Binomial variable from threefry generator

Description

Care should be taken with this method not to encounter the birthday problem, described https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage

rxnbinom(size, prob, n = 1L, ncores = 1L)

Arguments

size target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution). Must be strictly positive, need not be integer.

prob probability of success in each trial. 0 < prob <= 1.

n number of observations. If length(n) > 1, the length is taken to be the number required.

ncores Number of cores for the simulation

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the rxode2 environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the rxode2 engine with rxSetSeed().

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

negative binomial random deviates. Note that rxbinom2 uses the mu parameterization an the rxbinom uses the prob parameterization (mu=size/(prob+size))
**Examples**

```r
## Use threefry engine

rxnbinom(10, 0.9, n = 10) # with rxbinom you have to explicitly state n
rxnbinom(3, 0.5, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxnbinom(4, 0.7)

# use mu parameter
rxnbinomMu(40, 40, n = 10)

## This example uses `rxbinom` directly in the model

rx <- rxode2({
  a <- rxnbinom(10, 0.5)
})
et <- et(1, id = 1:100)
s <- rxSolve(rx, et)

rx <- rxode2({
  a <- rxnbinomMu(10, 40)
})
s <- rxSolve(rx, et)
```

---

**rxNorm**

*Get the normalized model*

**Description**

This get the syntax preferred model for processing

**Usage**

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

**Arguments**

- **obj**
  - rxode2 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 parameter initialization 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

rxnormV  Simulate random normal variable from threefry generator

Description
Simulate random normal variable from threefry generator

Usage
rxnormV(mean = 0, sd = 1, n = 1L, ncores = 1L)
rxnorm(mean = 0, sd = 1, n = 1L, ncores = 1L)

Arguments
mean  vector of means.
sd  vector of standard deviations.
n  number of observations
ncores  Number of cores for the simulation
rxnorm simulates using the threefry sitmo generator.
rxnormV used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of rxnorm. It is no longer supported in rxode2(()) blocks

Value
normal random number deviates
Examples

## Use threefry engine

rxnorm(n = 10) # with rxnorm you have to explicitly state n
rxnorm(n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxnorm(2, 3) ## The first 2 arguments are the mean and standard deviation

## This example uses `rxnorm` directly in the model

rx <- rxode2({
  a <- rxnorm()
})

et <- et(1, id = 1:2)

s <- rxSolve(rx, et)

---

### rxode2

**Create an ODE-based model specification**

#### Description

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

#### Usage

```r
rxode2(
  model,
  modName = basename(wd),
  wd = getwd(),
  filename = NULL,
  extraC = NULL,
  debug = FALSE,
  calcJac = NULL,
  calcSens = NULL,
  collapseModel = FALSE,
  package = NULL,
  ...
)
```

...,

linCmntSens = c("linCmtA", "linCmtB", "linCmtC"),
indLin = FALSE,
verbose = FALSE,
fullPrint = getOption("rxode2.fullPrint", 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 `rxode2 Syntax` below.

**modxName**

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` 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 rxode2 DLL for the model is created in the current directory named `rx_????_platform`, for example `rx_129f8f97fb94a87ca49ca8dafe691e1e_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.

`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` is a boolean indicating if the executable should be compiled with verbose debugging information turned on.

`calcJac` boolean indicating if rxode2 will calculate the Jacobain according to the specified ODEs.

`calcSens` boolean indicating if rxode2 will calculate the sensitivities according to the specified ODEs.

`collapseModel` boolean indicating if rxode2 will remove all LHS variables when calculating sensitivities.

`package` Package name for pre-compiled binaries.

... ignored arguments.

`linCmtSens` The method to calculate the linCmt() solutions

`indLin` Calculate inductive linearization matrices and compile with inductive linearization support.

`verbose` When TRUE be verbose with the linear compartmental model

`fullPrint` When using printf within the model, if TRUE print on every step (except ME/indLin), otherwise when FALSE print only when calculating the d/dt

Details

The Rx in the name `rxode2` 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 `rxode2 Syntax` below for coding details. An internal `rxode2` 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 `rxode2` produces an object of class `rxode2` which consists of a list-like structure (environment) with various member functions (see Section `Value` below).

For evaluating `rxode2` 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()` or `et()`.
An rxode2 model specification consists of one or more statements optionally terminated by semicolons ; and optional comments (comments are delimited by # and an end-of-line).
A block of statements is a set of statements delimited by curly braces { ... }.
Statements can be either assignments, conditional if/else if/else, while loops (can be exited by break), special statements, or printing statements (for debugging/testing)
Assignment statements can be:
• **simple** assignments, where the left hand is an identifier (i.e., variable)
• special **time-derivative** assignments, where the left hand specifies the change of the amount in the corresponding state variable (compartment) with respect to time e.g., d/dt(depot):
• special **initial-condition** assignments where the left hand specifies the compartment of the initial condition being specified, e.g. depot(0) = 0
• special model event changes including **bioavailability** (f(depot)=1), **lag time** (alag(depot)=0), **modeled rate** (rate(depot)=2) and **modeled duration** (dur(depot)=2). An example of these model features and the event specification for the modeled infusions the rxode2 data specification is found in rxode2 events vignette.
• special **change point syntax, or model times**. These model times are specified by mtime(var)=time
• special **Jacobian-derivative** assignments, where the left hand specifies the change in the compartment ode with respect to a variable. For example, if d/dt(y) = dy, then a Jacobian for this compartment can be specified as df(y)/dy(dy) = 1

Note that assignment can be done by =, <- or ~.

When assigning with the ~ operator, the **simple assignments** and **time-derivative** assignments will not be output.

Special statements can be:

• **Compartment declaration statements**, which can change the default dosing compartment and the assumed compartment number(s) as well as add extra compartment names at the end (useful for multiple-endpoint nlmixr models); These are specified by cmt(compartmentName)
• **Parameter declaration statements**, which can make sure the input parameters are in a certain order instead of ordering the parameters by the order they are parsed. This is useful for keeping the parameter order the same when using 2 different ODE models. These are specified by param(par1, par2,...)

An example model is shown below:

```
# simple assignment
C2 = centr/V2;

# time-derivative assignment
d/dt(centr) = F*KA*depot - CL*C2 - Q*C2 + Q*C3;
```

Expressions in assignment and if statements can be numeric or logical.
Numeric expressions can include the following numeric operators \(+\), \(-\), \(*\), \(/\), \(^\) and those mathematical functions defined in the C or the R math libraries (e.g., \(\text{fabs}\), \(\text{exp}\), \(\text{log}\), \(\text{sin}\), \(\text{abs}\)).

You may also access the R’s functions in the R math libraries, like \(\text{lgammafn}\) for the log gamma function.

The \texttt{rxode2} syntax is case-sensitive, i.e., \texttt{ABC} is different than \texttt{abc}, \texttt{Abc}, \texttt{ABc}, etc.

**Identifiers:**

Like R, Identifiers (variable names) may consist of one or more alphanumeric, underscore \(_\) or period \(\.) characters, but the first character cannot be a digit or underscore \(_\).

Identifiers in a model specification can refer to:

- State variables in the dynamic system (e.g., compartments in a pharmacokinetics model).
- Implied input variable, \(t\) (time), \(t_{\text{last}}\) (last time point), and \(podo\) (oral dose, in the undocumented case of absorption transit models).
- Special constants like \(\pi\) or R’s predefined constants.
- Model parameters (e.g., \(\text{ka}\) rate of absorption, \(\text{CL}\) clearance, etc.)
- Others, as created by assignments as part of the model specification; these are referred as LHS (left-hand side) variable.

Currently, the \texttt{rxode2} modeling language only recognizes system state variables and “parameters”, thus, any values that need to be passed from R to the ODE model (e.g., age) should be either passed in the \texttt{params} argument of the integrator function \texttt{rxSolve()} or be in the supplied event data-set.

There are certain variable names that are in the \texttt{rxode2} event tables. To avoid confusion, the following event table-related items cannot be assigned, or used as a state but can be accessed in the \texttt{rxode2} code:

- \texttt{cmt}
- \texttt{dvid}
- \texttt{addl}
- \texttt{ss}
- \texttt{rate}
- \texttt{id}

However the following variables are cannot be used in a model specification:

- \texttt{evid}
- \texttt{ii}

Sometimes \texttt{rxode2} generates variables that are fed back to \texttt{rxode2}. Similarly, \texttt{nlmixr} generates some variables that are used in \texttt{nlmixr} estimation and simulation. These variables start with the either the \texttt{rx} or \texttt{nlmixr} prefixes. To avoid any problems, it is suggested to not use these variables starting with either the \texttt{rx} or \texttt{nlmixr} prefixes.

**Logical Operators:**

Logical operators support the standard R operators \(==\), \(!=\) \(<=\) \(>=\) and \(<\). Like R these can be in \texttt{if()} or \texttt{while()} statements, \texttt{ifelse()} expressions. Additionally they can be in a standard assignment. For instance, the following is valid:

\[
\text{cov1} = \text{covm*(sexf} \text{=} \text{"female"}) + \text{covm*(sexf} \text{=} \text{"female"})
\]
Notice that you can also use character expressions in comparisons. This convenience comes at a cost since character comparisons are slower than numeric expressions. Unlike R, `as.numeric` or `as.integer` for these logical statements is not only not needed, but will cause an syntax error if you try to use the function.

**Value**

An object (environment) of class `rxode2` (see Chambers and Temple Lang (2001)) consisting of the following list of strings and functions:

- `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(rxode2object, ...)`, 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`), `rxode2` 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`), `rxode2` 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 \texttt{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 \texttt{rxode2} 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 \texttt{rxode2} 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.
* `delete` removes all created model files, including C and DLL files. The model object is no longer valid and should be removed, e.g., `rm(m1)`.
* `run` deprecated, use `solve`.
* `get.index` deprecated.
* `getObj` internal (not user callable) function.

**Author(s)**

Melissa Hallow, Wenping Wang and Matthew Fidler

**References**


**See Also**

\texttt{eventTable()}, \texttt{et()}, \texttt{add.sampling()}, \texttt{add.dosing()}
Examples

```r
# Step 1 - Create a model specification
ode <- 
  "# A 4-compartment model, 3 PK and a 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*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
"

m1 <- rxode(model = ode)
print(m1)
```

```r
# 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$add.sampling(0:24)
qd$add.sampling(seq(from = 24 + 8, to = 5 * 24, by = 8))

# Step 3 - set starting parameter estimates and initial
# values of the state

theta <- 
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 <- m1$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(m1, theta, events = qd, inits)
print(qd.cp)

plot(qd.cp)

# You can also directly simulate from a nlmixr model
f <- function() {
  ini(
    KA <- .291
    CL <- 18.6
    V2 <- 40.2
    Q <- 10.5
    V3 <- 297.0
    Kin <- 1.0
    Kout <- 1.0
    EC50 <- 200.0
  )
  model(
    # A 4-compartment model, 3 PK and a 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*C2 + Q*C3
    d/dt(peri) <- Q*C2 - Q*C3
    d/dt(eff) <- Kin - Kout*(1-C2/(EC50+C2))*eff
    eff(0) <- 1
  )
)}

u <- f()

# this pre-compiles and displays the simulation model
u$simulationModel

qd.cp <- solve(u, qd)
print(qd.cp)

---

**rxOptExpr**  
Optimize **rxode2** for computer evaluation

**Description**

This optimizes **rxode2** code for computer evaluation by only calculating redundant expressions once.
Usage

rxOptExpr(x, msg = "model")

Arguments

x
rxode2 model that can be accessed by rxNorm

msg
This is the name of type of object that rxode2 is optimizing that will in the
message when optimizing. For example "model" will produce the following
message while optimizing the model:
finding duplicate expressions in model...

Value

Optimized rxode2 model text. The order and type lhs and state variables is maintained while the
evaluation is sped up. While parameters names are maintained, their order may be modified.

Author(s)

Matthew L. Fidler

---

rxord

Simulate ordinal value

Description

Simulate ordinal value

Usage

rxord(...)  

Arguments

...  
the probabilities to be simulated. These should sum up to a number below one.

Details

The values entered into the 'rxord’ simulation will simulate the probability of falling each group.
If it falls outside of the specified probabilities, it will simulate the group (number of probabilities
specified + 1)

Value

A number from 1 to the (number of probabilities specified + 1)

Author(s)

Matthew L. Fidler
Examples

# This will give values 1, and 2
rxord(0.5)
rxord(0.5)
rxord(0.5)
rxord(0.5)

# This will give values 1, 2 and 3
rxord(0.3, 0.3)
rxord(0.3, 0.3)
rxord(0.3, 0.3)

rxParams

Parameters specified by the model

Description

This returns the model’s parameters that are required to solve the ODE system, and can be used to pipe parameters into an rxode2 solve

Usage

rxParams(obj, ...)

## S3 method for class 'rxode2'
rxParams(
  obj,
  constants = TRUE,
  ..., 
  params = NULL,
  inits = NULL,
  iCov = NULL,
  keep = NULL,
  thetaMat = NULL,
  omega = NULL,
  dfSub = NULL,
  sigma = NULL,
  dfObs = NULL,
  nSub = NULL,
  nStud = NULL
)

## S3 method for class 'rxSolve'
rxParams(
  obj,
```r
rxParams

## S3 method for class 'rxEt'
rxParams(
  obj,
  ...
)

Arguments

obj          rxode2 family of objects
...          Other arguments including scaling factors for each compartment. This includes
             $S# = \text{numeric}$ will scale a compartment # by a dividing the compartment amount
             by the scale factor, like NONMEM.
constants     is a boolean indicating if constants should be included in the list of parameters.  
             Currently rxode2 parses constants into variables in case you wish to change them 
             without recompiling the rxode2 model.
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;
inits         a vector of initial values of the state variables (e.g., amounts in each compart-
             ment), and the order in this vector must be the same as the state variables (e.g.,
```

```
PK/PD compartments);

iCov  A data frame of individual non-time varying covariates to combine with the events dataset by merge.

keep  Columns to keep from either the input dataset or the iCov dataset. With the iCov dataset, the column is kept once per line. For the input dataset, if any records are added to the data LOCF (Last Observation Carried forward) imputation is performed.

thetaMat  Named theta matrix.

omega  Estimate of Covariance matrix. When omega is a list, assume it is a block matrix and convert it to a full matrix for simulations. When omega is NA and you are using it with a rxode2 ui model, the between subject variability described by the omega matrix are set to zero.

dfSub  Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

sigma  Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system. When sigma is NA and you are using it with a rxode2 ui model, the unexplained variability described by the sigma matrix are set to zero.

dfObs  Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

nSub  Number between subject variabilities (ETAs) simulated for every realization of the parameters.

nStud  Number virtual studies to characterize uncertainty in estimated parameters.

Value

When extracting the parameters from an rxode2 model, a character vector listing the parameters in the model.

Author(s)

Matthew L.Fidler

See Also

Other Query model information: rxDfdy(), rxInit(), rxLhs(), rxModelVars(), rxState()
rxPkg  

*Creates a package from compiled rxode2 models*

**Description**

Creates a package from compiled rxode2 models

**Usage**

```r
rxPkg(
  ...,
  package,
  wd = getwd(),
  action = c("install", "build", "binary", "create"),
  license = c("gpl3", "lgpl", "mit", "agpl3"),
  name = "Firstname Lastname",
  fields = list()
)
```

**Arguments**

- `...`: Models to build a package from
- `package`: 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 rxode2 DLL for the model is created in the current directory named `rx_????_platform`, for example `rx_129f8f97fb94a87ca49ca8dafe691e1e_i386.dll`
- `wd`: 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 rxode2 DLL for the model is created in the current directory named `rx_????_platform`, for example `rx_129f8f97fb94a87ca49ca8dafe691e1e_i386.dll`
- `action`: Type of action to take after package is created
- `license`: Is the type of license for the package.
- `name`: Full name of author
- `fields`: A named list of fields to add to `DESCRIPTION`, potentially overriding default values. See `use_description()` for how you can set personalized defaults using package options.

**Value**

This function returns nothing and is used for its side effects

**Author(s)**

Matthew Fidler
### Description

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the `sitmo threefry`, this currently generates one random deviate from the uniform distribution to seed the engine `threefry` and then run the code.

### Usage

```r
rxpois(lambda, n = 1L, ncores = 1L)
```

### Arguments

- `lambda` vector of (non-negative) means.
- `n` number of random values to return.
- `ncores` Number of cores for the simulation

`rxnorm` simulates using the `threefry` `sitmo` generator.

`rxnormV` used to simulate with the `vandercorput` simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of `rxnorm`. It is no longer supported in `rxode2` blocks.

### Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`.

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

### Value

`poisson` random number deviates

### Examples

```
## Use threefry engine

rxpois(lambda = 3, n = 10) # with `rxpois` you have to explicitly state n
rxpois(lambda = 3, n = 10, ncores = 2) # You can parallelize the simulation using openMP
```
rxpois(4) ## The first arguments are the lambda parameter

## This example uses `rxpois` directly in the model

```r
rx <- rxode2({
a <- rxpois(3)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

---

**rxPp**

*Simulate a from a Poisson process*

**Description**

Simulate a from a Poisson process

**Usage**

```r
rxPp(
  n, lambda, gamma = 1, prob = NULL, t0 = 0, tmax = Inf, randomOrder = FALSE
)
```

**Arguments**

- **n**
  - Number of time points to simulate in the Poisson process
- **lambda**
  - Rate of Poisson process
- **gamma**
  - Asymmetry rate of Poisson process. When gamma=1.0, this simulates a homogeneous Poisson process. When gamma<1.0, the Poisson process has more events early, when gamma > 1.0, the Poisson process has more events late in the process.
  - When gamma is non-zero, the tmax should not be infinite but indicate the end of the Poisson process to be simulated. In most pharamcometric cases, this will be the end of the study. Internally this uses a rate of:
    - \( l(t) = \lambda \times \gamma(t/tmax)^{\gamma-1} \)
prob
When specified, this is a probability function with one argument, time, that gives the probability that a Poisson time \( t \) is accepted as a rejection time.

t0
the starting time of the Poisson process
tmax
the maximum time of the Poisson process
randomOrder
when TRUE randomize the order of the Poisson events. By default (FALSE) it returns the Poisson process is in order of how the events occurred.

Value
This returns a vector of the Poisson process times; If the dropout is >= tmax, then all the rest of the times are = tmax to indicate the dropout is equal to or after tmax.

Author(s)
Matthew Fidler

Examples

```r
## Sample homogenous Poisson process of rate 1/10
rxPp(10, 1 / 10)

## Sample inhomogenous Poisson rate of 1/10
rxPp(10, 1 / 10, gamma = 2, tmax = 100)

## Typically the Poisson process times are in a sequential order,
## using randomOrder gives the Poisson process in random order
rxPp(10, 1 / 10, gamma = 2, tmax = 10, randomOrder = TRUE)

## This uses an arbitrary function to sample a non-homogenous Poisson process
rxPp(10, 1 / 10, prob = function(x) {
  1 / x
})
```

---

**rxPreferredDistributionName**

*Change distribution name to the preferred distribution name term*

**Description**
This is determined by the internal preferred condition name list .errIdenticalDists

**Usage**

`rxPreferredDistributionName(dist)`
**Arguments**

- `dist`: This is the input distribution

**Value**

Preferred distribution term

**Author(s)**

Matthew Fidler

**Examples**

```r
rxPreferredDistributionName("dt")
rxPreferredDistributionName("add")
# can be vectorized
rxPreferredDistributionName(c("add","dt"))
```

---

**Description**

`rxProgress` sets up the progress bar

**Usage**

```r
rxProgress(num, core = 0L)
rxTick()
rxProgressStop(clear = TRUE)
rxProgressAbort(error = "Aborted calculation")
```

**Arguments**

- `num`: Total number of operations to track
- `core`: Number of cores to show. If below 1, don’t show number of cores
- `clear`: Boolean telling if you should clear the progress bar after completion (as if it wasn’t displayed). By default this is TRUE
- `error`: With `rxProgressAbort` this is the error that is displayed
Details

rxTick is a progress bar tick
rxProgressStop stop progress bar
rxProgressAbort shows an abort if rxProgressStop wasn’t called.

Value

All return NULL invisibly.

Author(s)

Matthew L. Fidler

Examples

```r
f <- function() {
  on.exit({
    rxProgressAbort()
  })
  rxProgress(100)
  for (i in 1:100) {
    rxTick()
    Sys.sleep(1 / 100)
  }
  rxProgressStop()
}

f()
```

---

**rxRemoveControl**

*rxRemoveControl options for UI object*

Description

rxRemoveControl options for UI object

Usage

`rxRemoveControl(ui)`

Arguments

ui rxode2 ui object

Value

Nothing, called for side effects
Author(s)
Matthew L. Fidler

---

**rxRename**

*Rename items inside of a rxode2 ui model*

**Description**

`rxRename()` changes the names of individual variables, lhs, and ode states using `new_name = old_name` syntax.

**Usage**

```
rxRename(.data, ..., envir = parent.frame())

rename.rxUi(.data, ...)

rename.function(.data, ...)
```

**Arguments**

- `.data` `rxode2` ui function, named data to be consistent with `dplyr::rename()`
- `...` `rename items`
- `envir` Environment for evaluation

**Value**

New model with items renamed

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

**Examples**

```r
ocmt <- function() {
inl(
  tka <- exp(0.45) # Ka
  tcl <- exp(1) # Cl
  ## This works with interactive models
  ## You may also label the preceding line with label("label text")
  tv <- exp(3.45) # log V
  ## the label("Label name") works with all models
  add.sd <- 0.7
)
model({
  ka <- tka
```
cl <- tcl
v <- tv
d/dt(depot) = -ka * depot
d/dt(center) = ka * depot - cl / v * center
cp = center / v
cp ~ add(add.sd)
}
}
ocmt %>% rxRename(cpParent=cp)

---

**rxReservedKeywords**

A list and description of Rode supported reserved keywords

**Description**

A list and description of Rode supported reserved keywords

**Usage**

```r
rxReservedKeywords
```

**Format**

A data frame with 3 columns and 98 or more rows

<table>
<thead>
<tr>
<th>Reserved Name</th>
<th>Reserved Keyword Name</th>
<th>Meaning</th>
<th>Alias</th>
<th>Keyword Alias</th>
</tr>
</thead>
</table>

**rxS**

Load a model into a symengine environment

**Description**

Load a model into a symengine environment

**Usage**

```r
rxS(x, doConst = TRUE, promoteLinSens = FALSE)
```

**Arguments**

- `x` rxode2 object
- `doConst` Load constants into the environment as well.
- `promoteLinSens` Promote solved linear compartment systems to sensitivity-based solutions.
rxSetCovariateNamesForPiping

Assign covariates for piping

Description

Assign covariates for piping

Usage

rxSetCovariateNamesForPiping(covariates = NULL)

Value

Nothing, called for side effects

Author(s)

Matthew L. Fidler

rxSetControl

rxSetControl options for UI object

Description

rxSetControl options for UI object

Usage

rxSetControl(ui, control)

Arguments

ui rxode2 ui object
control Default value

Value

Nothing, called for side effects

Author(s)

Matthew Fidler
### Arguments

covariates: NULL (for no covariates), or the list of covariates. nlmixr uses this function to set covariates if you pipe from a nlmixr fit.

### Value

Nothing, called for side effects

### Author(s)

Matthew L. Fidler

### Examples

```r
# First set the name of known covariates
# Note this is case sensitive
rxSetCovariateNamesForPiping(c("WT", "HT", "TC"))

one.compartment <- function() {
  ini({
    tka <- 0.45; label("Log Ka")
    tcl <- 1; label("Log Cl")
    tv <- 3.45; label("Log V")
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.err ~ 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    d / dt(depot) <- -ka * depot
    d / dt(depot) <- -ka * depot
    d / dt(center) <- ka * depot - cl / v * center
    cp <- center / v
    cp ~ add(add.err)
  })
}

# now TC is detected as a covariate instead of a population parameter
one.compartment %>%
  model({ka <- exp(tka + eta.ka + TC * cov.C)})

# You can turn it off by simply adding it back
rxSetCovariateNamesForPiping()
one.compartment %>%
```
model({ka <- exp(tka + eta.ka + TC * cov.C)})

# The covariates you set with `rxSetCovariateNamesForPiping()`
# are turned off every time you solve (or fit in nlmixr)

---

**rxSetIni0**

*Set Initial conditions to time zero instead of the first observed/dosed time*  

**Description**  
Set Initial conditions to time zero instead of the first observed/dosed time  

**Usage**  

```r  
rxSetIni0(ini0 = TRUE)  
```

**Arguments**  

- `ini0`  
  When TRUE (default), set initial conditions to time zero. Otherwise the initial conditions are the first observed time.  

**Value**  

the boolean ini0, though this is called for its side effects

---

**rxSetProd**

*Defunct setting of product*  

**Description**  
Defunct setting of product  

**Usage**  

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

**Arguments**  

- `type`  
  used to be type of product  

**Value**  

nothing
**rxSetProgressBar**

Set timing for progress bar

**Usage**

```r
rxSetProgressBar(seconds = 1)
```

**Arguments**

- **seconds**
  
  This sets the number of seconds that need to elapse before drawing the next segment of the progress bar. When this is zero or below this turns off the progress bar.

**Value**

- nothing, used for side effects

**Author(s)**

Matthew Fidler

---

**rxSetSum**

Defunct setting of sum

**Usage**

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

**Arguments**

- **type**
  
  used to be type of product

**Value**

- nothing
Use Shiny to help develop an rxode2 model

Description
Use Shiny to help develop an rxode2 model

Usage
rxShiny(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ..., 
  data = data.frame()
)

## S3 method for class 'rxSolve'
rxShiny(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ..., 
  data = data.frame()
)

## Default S3 method:
rxShiny(
  object = NULL,
  params = NULL,
  events = NULL,
  inits = NULL,
  ..., 
  data = data.frame()
)

Arguments

object A rxode2 family of objects. If not supplied a 2-compartment indirect effect model is used. If it is supplied, use the model associated with the rxode2 object for the model exploration.

params Initial parameters for model

events Event information (currently ignored)

inits Initial estimates for model
Other arguments passed to rxShiny. Currently doesn’t do anything.

data
Any data that you would like to plot. If the data has a time variable as well as a compartment or calculated variable that matches the rxode2 model, the data will be added to the plot of a specific compartment or calculated variable.

Value
Nothing: Starts a shiny server

Author(s)
Zufar Mulyukov and Matthew L. Fidler

rxSimThetaOmega Simulate Parameters from a Theta/Omega specification

Description
Simulate Parameters from a Theta/Omega specification

Usage
rxSimThetaOmega(
  params = NULL,
  omega = NULL,
  omegaDf = NULL,
  omegaLower = as.numeric(c(R_NegInf)),
  omegaUpper = as.numeric(c(R_PosInf)),
  omegaIsChol = FALSE,
  omegaSeparation = "auto",
  omegaXform = 1L,
  nSub = 1L,
  thetaMat = NULL,
  thetaLower = as.numeric(c(R_NegInf)),
  thetaUpper = as.numeric(c(R_PosInf)),
  thetaDf = NULL,
  thetaIsChol = FALSE,
  nStud = 1L,
  sigma = NULL,
  sigmaLower = as.numeric(c(R_NegInf)),
  sigmaUpper = as.numeric(c(R_PosInf)),
  sigmaDf = NULL,
  sigmaIsChol = FALSE,
  sigmaSeparation = "auto",
  sigmaXform = 1L,
  nCoresRV = 1L,
  nObs = 1L,
\begin{verbatim}
  dfSub = 0,
  dfObs = 0,
  simSubjects = TRUE,
  simVariability = as.logical(c(NA_LOGICAL))
)

Arguments

params       Named Vector of rxode2 model parameters
omega        Estimate of Covariance matrix. When omega is a list, assume it is a block matrix
              and convert it to a full matrix for simulations. When omega is NA and you are
              using it with a rxode2 ui model, the between subject variability described by
              the omega matrix are set to zero.
omegaDf      The degrees of freedom of a t-distribution for simulation. By default this is NULL
              which is equivalent to Inf degrees, or to simulate from a normal distribution
              instead of a t-distribution.
omegaLower   Lower bounds for simulated ETAs (by default -Inf)
omegaUpper   Upper bounds for simulated ETAs (by default Inf)
omegaIsChol  Indicates if the omega supplied is a Cholesky decomposed matrix instead of the
              traditional symmetric matrix.
omegaSeparation
              Omega separation strategy
              Tells the type of separation strategy when simulating covariance with parameter
              uncertainty with standard deviations modeled in the thetaMat matrix.

• "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter eta equal to the degrees of freedom nu by (nu-1)/2
• "separation" simulates from the identity inverse Wishart covariance matrix with nu degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10
• "auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.
omegaXform   When taking omega values from the thetaMat simulations (using the separation strategy for covariance simulation), how should the thetaMat values be turned int standard deviation values:

• identity This is when standard deviation values are directly modeled by the params and thetaMat matrix
• variance This is when the params and thetaMat simulates the variance that are directly modeled by the thetaMat matrix
• log This is when the params and thetaMat simulates log(sd)
• nlmixrSqrt This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the x^2 modeled along the diagonal. This only works with a diagonal matrix.
\end{verbatim}
• **nlmixrLog** This is when the `params` and `thetaMat` simulates the inverse cholesky decomposed matrix with the `exp(x^2)` along the diagonal. This only works with a diagonal matrix.
• **nlmixrIdentity** This is when the `params` and `thetaMat` simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.

**nSub**
Number between subject variabilities (ETAs) simulated for every realization of the parameters.

**thetaMat**
Named theta matrix.

**thetaLower**
Lower bounds for simulated population parameter variability (by default -Inf)

**thetaUpper**
Upper bounds for simulated population unexplained variability (by default Inf)

**thetaDf**
The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.

**thetaIsChol**
Indicates if the `theta` supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.

**nStud**
Number virtual studies to characterize uncertainty in estimated parameters.

**sigma**
Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system. When `sigma` is NA and you are using it with a `rxode2 ui` model, the unexplained variability described by the `sigma` matrix are set to zero.

**sigmaLower**
Lower bounds for simulated unexplained variability (by default -Inf)

**sigmaUpper**
Upper bounds for simulated unexplained variability (by default Inf)

**sigmaDf**
Degrees of freedom of the sigma t-distribution. By default it is equivalent to Inf, or a normal distribution.

**sigmaIsChol**
Boolean indicating if the `sigma` is in the Cholesky decomposition instead of a symmetric covariance

**sigmaSeparation**
Separation strategy for `sigma`;
Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the `thetaMat` matrix.

• "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter `eta` equal to the degrees of freedom `nu` by `(nu-1)/2`
• "separation" simulates from the identity inverse Wishart covariance matrix with `nu` degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10
• "auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.

**sigmaXform**
When taking `sigma` values from the `thetaMat` simulations (using the separation strategy for covariance simulation), how should the `thetaMat` values be turned into standard deviation values:
• identity This is when standard deviation values are directly modeled by the \( \text{params} \) and \( \text{thetaMat} \) matrix

• variance This is when the \( \text{params} \) and \( \text{thetaMat} \) simulates the variance that are directly modeled by the \( \text{thetaMat} \) matrix

• log This is when the \( \text{params} \) and \( \text{thetaMat} \) simulates \( \log(\text{sd}) \)

• nlmixrSqrt This is when the \( \text{params} \) and \( \text{thetaMat} \) simulates the inverse cholesky decomposed matrix with the \( x^2 \) modeled along the diagonal. This only works with a diagonal matrix.

• nlmixrLog This is when the \( \text{params} \) and \( \text{thetaMat} \) simulates the inverse cholesky decomposed matrix with the \( \exp(x^2) \) along the diagonal. This only works with a diagonal matrix.

• nlmixrIdentity This is when the \( \text{params} \) and \( \text{thetaMat} \) simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.

**nCoresRV** Number of cores used for the simulation of the sigma variables. By default this is 1. To reproduce the results you need to run on the same platform with the same number of cores. This is the reason this is set to be one, regardless of what the number of cores are used in threaded ODE solving.

**nObs** Number of observations to simulate (with sigma matrix)

**dfSub** Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

**dfObs** Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

**simSubjects** boolean indicated rxode2 should simulate subjects in studies (TRUE, default) or studies (FALSE)

**simVariability** determines if the variability is simulated. When NA (default) this is determined by the solver.

**Value**

a data frame with the simulated subjects

**Author(s)**

Matthew L. Fidler

---

**rxSolve** *Solving & Simulation of a ODE/solved system (a options) equation*

**Description**

This uses rxode2 family of objects, file, or model specification to solve a ODE system. There are many options for a solved rxode2 model, the first are the required object, and events with the some-times optional params and inits.
rxSolve

Usage

rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  scale = NULL,
  method = c("liblsoda", "lsoda", "dop853", "indLin"),
  sigdig = NULL,
  atol = 1e-08,
  rtol = 1e-06,
  maxsteps = 70000L,
  hmin = 0,
  hmax = NA_real_,
  hmaxSd = 0,
  hini = 0,
  maxordn = 12L,
  maxords = 5L,
  ..., 
  cores,
  covsInterpolation = c("locf", "linear", "nocb", "midpoint"),
  addCov = TRUE,
  sigma = NULL,
  sigmaDf = NULL,
  sigmaLower = -Inf,
  sigmaUpper = Inf,
  nCoresRV = 1L,
  sigmaIsChol = FALSE,
  sigmaSeparation = c("auto", "lkj", "separation"),
  sigmaXform = c("identity", "variance", "log", "nlmixrSqrt", "nlmixrLog",
                 "nlmixrIdentity"),
  nDisplayProgress = 10000L,
  amountUnits = NA_character_,
  timeUnits = "hours",
  theta = NULL,
  thetaLower = -Inf,
  thetaUpper = Inf,
  eta = NULL,
  addDosing = FALSE,
  stateTrim = Inf,
  updateObject = FALSE,
  omega = NULL,
  omegaDf = NULL,
  omegaIsChol = FALSE,
  omegaSeparation = c("auto", "lkj", "separation"),
  omegaXform = c("variance", "identity", "log", "nlmixrSqrt", "nlmixrLog",
                 "nlmixrIdentity"),
  omegaLower = -Inf,
omegaUpper = Inf,
nSub = 1L,
thetaMat = NULL,
thetaDf = NULL,
thetaIsChol = FALSE,
nStud = 1L,
dfSub = 0,
dfObs = 0,
returnType = c("rxSolve", "matrix", "data.frame", "data.frame.TBS", "data.table",
               "tbl", "tibble"),
seed = NULL,
nsim = NULL,
minSS = 10L,
maxSS = 1000L,
infSSstep = 12,
strictSS = TRUE,
istateReset = TRUE,
subsetNonmem = TRUE,
maxAtolRtolFactor = 0.1,
from = NULL,
to = NULL,
by = NULL,
length.out = NULL,
iCov = NULL,
keep = NULL,
indLinPhiTol = 1e-07,
indLinPhiM = 0L,
indLinMatExpType = c("expokit", "Al-Mohy", "arma"),
indLinMatExpOrder = 6L,
drop = NULL,
idFactor = TRUE,
mxhnil = 0,
hmxi = 0,
warnIdSort = TRUE,
warnDrop = TRUE,
ssAtol = 1e-08,
ssRtol = 1e-06,
safeZero = TRUE,
sumType = c("pairwise", "fsum", "kahan", "neumaier", "c"),
prodType = c("long double", "double", "logify"),
sensType = c("advan", "autodiff", "forward", "central"),
linDiff = c(tlag = 1.5e-05, f = 1.5e-05, rate = 1.5e-05, dur = 1.5e-05, tlag2 =
           1.5e-05, f2 = 1.5e-05, rate2 = 1.5e-05, dur2 = 1.5e-05),
linDiffCentral = c(tlag = TRUE, f = TRUE, rate = TRUE, dur = TRUE, tlag2 = TRUE, f2 =
                    TRUE, rate2 = TRUE, dur2 = TRUE),
resample = NULL,
resampleID = TRUE,
maxwhile = 1e+05,
atolSens = 1e-08,
rtolSens = 1e-06,
ssAtolSens = 1e-08,
ssRtolSens = 1e-06,
simVariability = NA,
nLLikAlloc = NULL,
useStdPow = FALSE
)

## S3 method for class '
function
\n' rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...
  theta = NULL,
  eta = NULL
)

## S3 method for class 'rxUi'
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...
  theta = NULL,
  eta = NULL
)

## S3 method for class 'nlmixr2FitData'
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...
  theta = NULL,
  eta = NULL
)

## S3 method for class 'nlmixr2FitCore'
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,

...,
  theta = NULL,
  eta = NULL
)

## Default S3 method:
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...

  theta = NULL,
  eta = NULL
)

## S3 method for class 'rxSolve'
update(object, ...)

## S3 method for class 'rxode2'
predict(object, ...)

## S3 method for class 'rxSolve'
predict(object, ...)

## S3 method for class 'rxEt'
predict(object, ...)

## S3 method for class 'rxParams'
predict(object, ...)

## S3 method for class 'rxode2'
simulate(object, nsim = 1L, seed = NULL, ...)

## S3 method for class 'rxSolve'
simulate(object, nsim = 1L, seed = NULL, ...)

## S3 method for class 'rxParams'
simulate(object, nsim = 1L, seed = NULL, ...)

## S3 method for class 'rxSolve'
solve(a, b, ...)

## S3 method for class 'rxUi'
solve(a, b, ...)

## S3 method for class 'rxode2'
solve(a, b, ...)
## S3 method for class 'rxParams'
solve(a, b, ...)

## S3 method for class 'rxEt'
solve(a, b, ...)

rxControl(..., params = NULL, events = NULL, inits = NULL)

### Arguments

- **object**
  is a either a rxode2 family of objects, or a file-name with a rxode2 model specification, or a string with a rxode2 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=c(center=2) will divide the center ODE variable by 2.

- **method**
  The method for solving ODEs. Currently this supports:
  - "liblsoda" thread safe lsoda. This supports parallel thread-based solving, and ignores user Jacobian specification.
  - "lsoda" – LSODA solver. Does not support parallel thread-based solving, but allows user Jacobian specification.
  - "dop853" – DOP853 solver. Does not support parallel thread-based solving nor user Jacobian specification
  - "indLin" – Solving through inductive linearization. The rxode2 dll must be setup specially to use this solving routine.

- **sigdig**
  Specifies the "significant digits" that the ode solving requests. When specified this controls the relative and absolute tolerances of the ODE solvers. By default the tolerance is \(0.5 \times 10^\text{-}\text{sigdig} - 2\) for regular ODEs. For the sensitivity equations the default is \(0.5 \times 10^\text{-}\text{sigdig} - 1.5\) (sensitivity changes only applicable for liblsoda). This also controls the atol/rtol of the steady state solutions. The ssAtol/ssRtol is \(0.5 \times 10^\text{-}\text{sigdig}\) and for the sensitivities \(0.5 \times 10^\text{-}\text{sigdig} + 0.625\). By default this is unspecified (NULL) and uses the standard atol/rtol.

- **atol**
  a numeric absolute tolerance (1e-8 by default) used by the ODE solver to determine if a good solution has been achieved; This is also used in the solved linear model to check if prior doses do not add anything to the solution.
rtol  a numeric relative tolerance (1e-6 by default) used by the ODE solver to determine if a good solution has been achieved. This is also used in the solved linear model to check if prior doses do not add anything to the solution.

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. When \( \text{hmax}=\text{NA} \) (default), uses the average difference + hmaxSd*sd in times and sampling events. The hmaxSd is a user specified parameter and which defaults to zero. When \( \text{hmax} = \text{NULL} \) \text{rxode2} uses the maximum difference in times in your sampling and events. The value 0 is equivalent to infinite maximum absolute step size.

hmaxSd  The number of standard deviations of the time difference to add to hmax. The default is 0

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

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 \( S# = \text{numeric} \) will scale a compartment # by a dividing the compartment amount by the scale factor, like NONMEM.

cores  Number of cores used in parallel ODE solving. This is equivalent to calling \text{setRxThreads()}\)

covsInterpolation  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, which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value.
- "constant" – Last observation carried forward (the default).
- "NOCB" – Next Observation Carried Backward. This is the same method that NONMEM uses.
- "midpoint" Last observation carried forward to midpoint; Next observation carried backward to midpoint.

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

sigma  Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system. When sigma is NA and you are using it with a \text{rxode2} ui model, the unexplained variability described by the sigma matrix are set to zero.

sigmaDf  Degrees of freedom of the sigma t-distribution. By default it is equivalent to Inf, or a normal distribution.
**rxSolve**

- **sigmaLower**: Lower bounds for simulated unexplained variability (by default -Inf)
- **sigmaUpper**: Upper bounds for simulated unexplained variability (by default Inf)
- **nCoresRV**: Number of cores used for the simulation of the sigma variables. By default this is 1. To reproduce the results you need to run on the same platform with the same number of cores. This is the reason this is set to be one, regardless of what the number of cores are used in threaded ODE solving.
- **sigmaIsChol**: Boolean indicating if the sigma is in the Cholesky decomposition instead of a symmetric covariance
- **sigmaSeparation**: separation strategy for sigma;
  
  Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the thetaMat matrix.
- **"lkj"**: simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter eta equal to the degrees of freedom nu by (nu-1)/2
- **"separation"**: simulates from the identity inverse Wishart covariance matrix with nu degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10
- **"auto"**: chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.
- **sigmaXform**: When taking sigma values from the thetaMat simulations (using the separation strategy for covariance simulation), how should the thetaMat values be turned into standard deviation values:
  - **identity**: This is when standard deviation values are directly modeled by the params and thetaMat matrix
  - **variance**: This is when the params and thetaMat simulates the variance that are directly modeled by the thetaMat matrix
  - **log**: This is when the params and thetaMat simulates log(sd)
  - **nlmixrSqrt**: This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the x^2 modeled along the diagonal. This only works with a diagonal matrix.
  - **nlmixrLog**: This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the exp(x^2) along the diagonal. This only works with a diagonal matrix.
  - **nlmixrIdentity**: This is when the params and thetaMat simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.
- **nDisplayProgress**: An integer indicating the minimum number of c-based solves before a progress bar is shown. By default this is 10,000.
- **amountUnits**: This supplies the dose units of a data frame supplied instead of an event table. This is for importing the data as an rxode2 event table.
- **timeUnits**: This supplies the time units of a data frame supplied instead of an event table. This is for importing the data as an rxode2 event table.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta</td>
<td>A vector of parameters that will be named THETA[#] and added to parameters</td>
</tr>
<tr>
<td>thetaLower</td>
<td>Lower bounds for simulated population parameter variability (by default -Inf)</td>
</tr>
<tr>
<td>thetaUpper</td>
<td>Upper bounds for simulated population unexplained variability (by default Inf)</td>
</tr>
<tr>
<td>eta</td>
<td>A vector of parameters that will be named ETA[#] and added to parameters</td>
</tr>
<tr>
<td>addDosing</td>
<td>Boolean indicating if the solve should add rxode2 EVID and related columns.</td>
</tr>
<tr>
<td></td>
<td>This will also include dosing information and estimates at the doses. Be</td>
</tr>
<tr>
<td></td>
<td>default, rxode2 only includes estimates at the observations. (default FALSE)</td>
</tr>
<tr>
<td></td>
<td>When addDosing is NULL, only include EVID=0 on solve and exclude any model-times</td>
</tr>
<tr>
<td></td>
<td>or EVID=2. If addDosing is NA the classic rxode2 EVID events are returned.</td>
</tr>
<tr>
<td></td>
<td>When addDosing is TRUE add the event information in NONMEM-style format; If</td>
</tr>
<tr>
<td></td>
<td>subsetNonmem=FALSE rxode2 will also include extra event types (EVID) for</td>
</tr>
<tr>
<td></td>
<td>ending infusion and modeled times:</td>
</tr>
<tr>
<td></td>
<td>• EVID=-1 when the modeled rate infusions are turned off (matches rate=-1)</td>
</tr>
<tr>
<td></td>
<td>• EVID=-2 When the modeled duration infusions are turned off (matches rate=-2)</td>
</tr>
<tr>
<td></td>
<td>• EVID=-10 When the specified rate infusions are turned off (matches rate&gt;0)</td>
</tr>
<tr>
<td></td>
<td>• EVID=-20 When the specified dur infusions are turned off (matches dur&gt;0)</td>
</tr>
<tr>
<td></td>
<td>• EVID=101,102,103,... Modeled time where 101 is the first model time, 102</td>
</tr>
<tr>
<td></td>
<td>is the second etc.</td>
</tr>
<tr>
<td>stateTrim</td>
<td>When amounts/concentrations in one of the states are above this value, trim</td>
</tr>
<tr>
<td></td>
<td>them to be this value. By default Inf. Also trims to -stateTrim for large</td>
</tr>
<tr>
<td></td>
<td>negative amounts/concentrations. If you want to trim between a range say c(0,</td>
</tr>
<tr>
<td></td>
<td>20000000) you may specify 2 values with a lower and upper range to make sure</td>
</tr>
<tr>
<td></td>
<td>all state values are in the reasonable range.</td>
</tr>
<tr>
<td>updateObject</td>
<td>This is an internally used flag to update the rxode2 solved object (when supply-</td>
</tr>
<tr>
<td></td>
<td>ing an rxode2 solved object) as well as returning a new object. You probably</td>
</tr>
<tr>
<td></td>
<td>should not modify it’s FALSE default unless you are willing to have unexpected</td>
</tr>
<tr>
<td></td>
<td>results.</td>
</tr>
<tr>
<td>omega</td>
<td>Estimate of Covariance matrix. When omega is a list, assume it is a block matrix</td>
</tr>
<tr>
<td></td>
<td>and convert it to a full matrix for simulations. When omega is NA and you are</td>
</tr>
<tr>
<td></td>
<td>using it with a rxode2 ui model, the between subject variability described by</td>
</tr>
<tr>
<td></td>
<td>the omega matrix are set to zero.</td>
</tr>
<tr>
<td>omegaDf</td>
<td>The degrees of freedom of a t-distribution for simulation. By default this is</td>
</tr>
<tr>
<td></td>
<td>NULL which is equivalent to Inf degrees, or to simulate from a normal distribution</td>
</tr>
<tr>
<td></td>
<td>instead of a t-distribution.</td>
</tr>
<tr>
<td>omegaIsChol</td>
<td>Indicates if the omega supplied is a Cholesky decomposed matrix instead of the</td>
</tr>
<tr>
<td></td>
<td>traditional symmetric matrix.</td>
</tr>
<tr>
<td>omegaSeparation</td>
<td>Omega separation strategy</td>
</tr>
<tr>
<td></td>
<td>Tells the type of separation strategy when simulating covariance with param-</td>
</tr>
<tr>
<td></td>
<td>eter uncertainty with standard deviations modeled in the thetaMat matrix.</td>
</tr>
<tr>
<td></td>
<td>• &quot;lkj&quot; simulates the correlation matrix from the rLJKJ1 matrix with the dis-</td>
</tr>
<tr>
<td></td>
<td>tribution parameter eta equal to the degrees of freedom nu by (nu-1)/2</td>
</tr>
</tbody>
</table>
"separation" simulates from the identity inverse Wishart covariance matrix with \( \nu \) degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10.

"auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.

**omegaXform**

When taking \( \omega \) values from the \( \theta \)Mat simulations (using the separation strategy for covariance simulation), how should the \( \theta \)Mat values be turned int standard deviation values:

- **identity** This is when standard deviation values are directly modeled by the \( \text{params} \) and \( \theta \)Mat matrix
- **variance** This is when the \( \text{params} \) and \( \theta \)Mat simulates the variance that are directly modeled by the \( \theta \)Mat matrix
- **log** This is when the \( \text{params} \) and \( \theta \)Mat simulates \( \log(\text{sd}) \)
- **nlmixrSqrt** This is when the \( \text{params} \) and \( \theta \)Mat simulates the inverse cholesky decomposed matrix with the \( \chi^2 \) modeled along the diagonal. This only works with a diagonal matrix.
- **nlmixrLog** This is when the \( \text{params} \) and \( \theta \)Mat simulates the inverse cholesky decomposed matrix with the \( \exp(\chi^2) \) along the diagonal. This only works with a diagonal matrix.
- **nlmixrIdentity** This is when the \( \text{params} \) and \( \theta \)Mat simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.

**omegaLower**

Lower bounds for simulated ETAs (by default -Inf)

**omegaUpper**

Upper bounds for simulated ETAs (by default Inf)

**nSub**

Number between subject variabilities (ETAs) simulated for every realization of the parameters.

**thetaMat**

Named theta matrix.

**thetaDf**

The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.

**thetaIsChol**

Indicates if the \( \theta \) supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.

**nStud**

Number virtual studies to characterize uncertainty in estimated parameters.

**dfSub**

Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

**dfObs**

Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

**returnType**

This tells what type of object is returned. The currently supported types are:

- "rxSolve" (default) will return a reactive data frame that can change easily change different pieces of the solve and update the data frame. This is the currently standard solving method in rnode2, is used for \( rxSolve(object, ...) \), solve(object,...).
- "data.frame" – returns a plain, non-reactive data frame; Currently very slightly faster than returnType="matrix"
- "matrix" – returns a plain matrix with column names attached to the solved object. This is what is used object$run as well as object$solve
- "data.table" – returns a data.table; The data.table is created by reference (ie setDt()), which should be fast.
- "tbl" or "tibble" returns a tibble format.

**seed**
an object specifying if and how the random number generator should be initialized

**nsim**
represents the number of simulations. For rxode2, if you supply single subject event tables (created with [eventTable()])

**minSS**
Minimum number of iterations for a steady-state dose

**maxSS**
Maximum number of iterations for a steady-state dose

**infSSstep**
Step size for determining if a constant infusion has reached steady state. By default this is large value, 12.

**strictSS**
Boolean indicating if a strict steady-state is required. If a strict steady-state is (TRUE) required then at least minSS doses are administered and the total number of steady states doses will continue until maxSS is reached, or atol and rtol for every compartment have been reached. However, if ODE solving problems occur after the minSS has been reached the whole subject is considered an invalid solve. If strictSS is FALSE then as long as minSS has been reached the last good solve before ODE solving problems occur is considered the steady state, even though either atol, rtol or maxSS have not been achieved.

**istateReset**
When TRUE, reset the ISTATE variable to 1 for lsoda and liblsoda with doses, like deSolve; When FALSE, do not reset the ISTATE variable with doses.

**subsetNonmem**
subset to NONMEM compatible EVIDs only. By default TRUE.

**maxAtolRtolFactor**
The maximum atol/rtol that FOCEi and other routines may adjust to. By default 0.1

**from**
When there is no observations in the event table, start observations at this value. By default this is zero.

**to**
When there is no observations in the event table, end observations at this value. By default this is 24 + maximum dose time.

**by**
When there are no observations in the event table, this is the amount to increment for the observations between from and to.

**length.out**
The number of observations to create if there isn’t any observations in the event table. By default this is 200.

**iCov**
A data frame of individual non-time varying covariates to combine with the events dataset by merge.

**keep**
Columns to keep from either the input dataset or the iCov dataset. With the iCov dataset, the column is kept once per line. For the input dataset, if any records are added to the data LOCF (Last Observation Carried forward) imputation is performed.
indLinPhiTol  the requested accuracy tolerance on exponential matrix.
indLinPhiM   the maximum size for the Krylov basis
indLinMatExpType
This is the matrix exponential type that is used for rxode2. Currently the following are supported:
• Al-Mohy  Uses the exponential matrix method of Al-Mohy Higham (2009)
• arma       Use the exponential matrix from RcppArmadillo
• expokit    Use the exponential matrix from Roger B. Sidje (1998)

indLinMatExpOrder
an integer, the order of approximation to be used, for the Al-Mohy and expokit values. The best value for this depends on machine precision (and slightly on the matrix). We use 6 as a default.
drop       Columns to drop from the output
idFactor   This boolean indicates if original ID values should be maintained. This changes the default sequentially ordered ID to a factor with the original ID values in the original dataset. By default this is enabled.
mxhnil     maximum number of messages printed (per problem) warning that $T + H = T$ on a step ($H$ = step size). This must be positive to result in a non-default value. The default value is 0 (or infinite).
hmxi       inverse of the maximum absolute value of $H$ to be used. hmxi = 0.0 is allowed and corresponds to an infinite hmax1 (default). hmxi and hmxi may be changed at any time, but will not take effect until the next change of $H$ is considered. This option is only considered with method="liblsoda".
warnIdSort Warn if the ID is not present and rxode2 assumes the order of the parameters/iCov are the same as the order of the parameters in the input dataset.
warnDrop   Warn if column(s) were supposed to be dropped, but were not present.
ssAtol     Steady state atol convergence factor. Can be a vector based on each state.
ssRtol     Steady state rtol convergence factor. Can be a vector based on each state.
safeZero   Use safe zero divide and log routines. By default this is turned on but you may turn it off if you wish.
sumType    Sum type to use for sum() in rxode2 code blocks.
pairwise uses the pairwise sum (fast, default)
fsum uses the PreciseSum package's fsum function (most accurate)
kahan uses Kahan correction
neumaier uses Neumaier correction
c uses no correction: default/native summing
prodType   Product to use for prod() in rxode2 blocks
long double converts to long double, performs the multiplication and then converts back.
double uses the standard double scale for multiplication.
sensType   Sensitivity type for linCmt() model:
advan Use the direct advan solutions
autodiff Use the autodiff advan solutions
forward Use forward difference solutions
central Use central differences
linDiff

This gives the linear difference amount for all the types of linear compartment model parameters where sensitivities are not calculated. The named components of this numeric vector are:

- "lag" Central compartment lag
- "f" Central compartment bioavailability
- "rate" Central compartment modeled rate
- "dur" Central compartment modeled duration
- "lag2" Depot compartment lag
- "f2" Depot compartment bioavailability
- "rate2" Depot compartment modeled rate
- "dur2" Depot compartment modeled duration

linDiffCentral

This gives the which parameters use central differences for the linear compartment model parameters. The are the same components as linDiff.

resample

A character vector of model variables to resample from the input dataset; This sampling is done with replacement. When NULL or FALSE no resampling is done. When TRUE resampling is done on all covariates in the input dataset.

resampleID

boolean representing if the resampling should be done on an individual basis TRUE (ie. a whole patient is selected) or each covariate is resampled independent of the subject identifier FALSE. When resampleID=TRUE correlations of parameters are retained, where as when resampleID=FALSE ignores patient covariate correlations. Hence the default is resampleID=TRUE.

maxwhile

represents the maximum times a while loop is evaluated before exiting. By default this is 100000

atolSens

Sensitivity atol, can be different than atol with liblsoda. This allows a less accurate solve for gradients (if desired)

rtolSens

Sensitivity rtol, can be different than rtol with liblsoda. This allows a less accurate solve for gradients (if desired)

ssAtolSens

Sensitivity absolute tolerance (atol) for calculating if steady state has been achieved for sensitivity compartments.

ssRtolSens

Sensitivity relative tolerance (rtol) for calculating if steady state has been achieved for sensitivity compartments.

simVariability

determines if the variability is simulated. When NA (default) this is determined by the solver.

nLlikAlloc

The number of log likelihood endpoints that are used in the model. This allows independent log likelihood per endpoint in focell for nlmixr2. It likely shouldn’t be set, though it won’t hurt anything if you do (just may take up more memory for larger allocations).

useStdPow

This uses C’s pow for exponentiation instead of R’s R_pow or R_pow_di. By default this is FALSE.

a

when using solve(), this is equivalent to the object argument. If you specify object later in the argument list it overwrites this parameter.

b

when using solve(), this is equivalent to the params argument. If you specify params as a named argument, this overwrites the output.
Details

The rest of the document focus on the different ODE solving methods, followed by the core solving method’s options, rxode2 event handling options, rxode2’s numerical stability options, rxode2’s output options, and finally internal rxode2 options or compatibility options.

Value

An “rxSolve” solve object that stores the solved value in a special data.frame or other type as determined by returnType. By default this has 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 rxode2 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)

Matthew Fidler, Melissa Hallow and Wenping Wang

References

"New Scaling and Squaring Algorithm for the Matrix Exponential", by Awad H. Al-Mohy and Nicholas J. Higham, August 2009


See Also

rxode2()
**rxState**

*State variables*

**Description**

This returns the model’s compartments or states.

**Usage**

```r
rxState(obj = NULL, state = NULL)
```

**Arguments**

- `obj`  
  rxode2 family of objects
- `state`  
  is a string indicating the state or compartment that you would like to lookup.

**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**

- `rxode2()`

Other Query model information: `rxDfdy()`, `rxInits()`, `rxLhs()`, `rxModelVars()`, `rxParams()`

---

**rxSumProdModel**

*Recast model in terms of sum/prod*

**Description**

Recast model in terms of sum/prod

**Usage**

```r
rxSumProdModel(model, expand = FALSE, sum = TRUE, prod = TRUE)
```
**Arguments**

- **model**  
  rxode2 model

- **expand**  
  Boolean indicating if the expression is expanded.

- **sum**  
  Use `sum(...)`

- **prod**  
  Use `prod(...)`

**Value**

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

**Author(s)**

Matthew L. Fidler

---

### rxSupportedFuns

*Get list of supported functions*

**Description**

Get list of supported functions

**Usage**

```r
rxSupportedFuns()
```

**Value**

list of supported functions in rxode2

**Examples**

```r
rxSupportedFuns()
```
rxSuppressMsg

Respect suppress messages

Description

This turns on the silent REprintf in C when suppressMessages() is turned on. This makes the REprintf act like messages in R, they can be suppressed with suppressMessages()

Usage

rxSuppressMsg()

Value

Nothing

Author(s)

Matthew Fidler

Examples

# rxSupressMsg() is called with rxode2()

# Note the errors are output to the console
try(rxode2("d/dt(matt)=/3"), silent = TRUE)

# When using suppressMessages, the output is suppressed
suppressMessages(try(rxode2("d/dt(matt)=/3"), silent = TRUE))

# In rxode2, we use REprintf so that interrupted threads do not crash R
# if there is a user interrupt. This isn't captured by R's messages, but
# This interface allows the `suppressMessages()` to suppress the C printing
# as well

# If you want to suppress messages from rxode2 in other packages, you can use
# this function
rxSymInvChol  Get Omega^-1 and derivatives

Description
Get Omega^-1 and derivatives

Usage
rxSymInvChol(
invObjOrMatrix,
theta = NULL,
type = "cholOmegaInv",
thetaNumber = 0L
)

Arguments
invObjOrMatrix  Object for inverse-type calculations. If this is a matrix, setup the object for inversion rxSymInvCholCreate() with the default arguments and return a reactive s3 object. Otherwise, use the inversion object to calculate the requested derivative/inverse.
theta  Thetas to be used for calculation. If missing (NULL), a special s3 class is created and returned to access Omega^-1 objects as needed and cache them based on the theta that is used.
type  The type of object. Currently the following types are supported:
• cholOmegaInv gives the Cholesky decomposition of the Omega Inverse matrix.
• omegaInv gives the Omega Inverse matrix.
• d(omegaInv) gives the d(Omega^-1) with respect to the theta parameter specified in thetaNumber.
• d(D) gives the d(diagonal(Omega^-1)) with respect to the theta parameter specified in the thetaNumber parameter
thetaNumber  For types d(omegaInv) and d(D), the theta number that the derivative is taken against. This must be positive from 1 to the number of thetas defining the Omega matrix.

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

Author(s)
Matthew L. Fidler
### rxSyncOptions

**Sync options with rxode2 variables**

**Description**

Accessing rxode2 options via `getOption` slows down solving. This allows the options to be synced with variables.

**Usage**

```r
rxSyncOptions(setDefaults = c("none", "permissive", "strict"))
```

**Arguments**

- `setDefaults` This will setup rxode2’s default solving options with the following options:
  - "none" leave the options alone
  - "permissive" This is a permissive option set similar to R language specifications.
  - "strict" This is a strict option set similar to the original `rxode2()`. It requires semicolons at the end of lines and equals for assignment

**Value**

nothing; called for side effects

**Author(s)**

Matthew L. Fidler

### rxSyntaxFunctions

**A list and description of Rode supported syntax functions**

**Description**

A list and description of Rode supported syntax functions

**Usage**

```r
rxSyntaxFunctions
```

**Format**

A data frame with 3 columns and 98 or more rows

- **Function** Reserved function Name
- **Description** Description of function
- **Aliases** Function Aliases
**rxt**  
_Simulate student t variable from threefry generator_

**Description**
Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```r
rxt(df, n = 1L, ncores = 1L)
```

**Arguments**
- **df**: degrees of freedom (> 0, maybe non-integer). `df = Inf` is allowed.
- **n**: number of observations. If `length(n) > 1`, the length is taken to be the number required.
- **ncores**: Number of cores for the simulation

**rxnorm** simulates using the threefry sitmo generator.
**rxnormV** used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of `rxnorm`. It is no longer supported in `rxode2` blocks.

**Details**
Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`.

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

**Value**
- t-distribution random numbers

**Examples**

```r
## Use threefry engine
```
rxt(df = 3, n = 10) # with rxt you have to explicitly state n
rxt(df = 3, n = 10, ncores = 2) # You can parallelize the simulation using openMP

rxt(4) ## The first argument is the df parameter

## This example uses `rxt` directly in the model

rx <- rxode2({
a <- rxt(3)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)

---

**rxTempDir**

*Get the rxode2 temporary directory*

**Description**

Get the rxode2 temporary directory

**Usage**

```r
rxTempDir()
```

**Value**

rxode2 temporary directory.

---

**rxTheme**

*rxTheme is the ggplot2 theme for rxode2 plots*

**Description**

rxTheme is the ggplot2 theme for rxode2 plots

**Usage**

```r
rxTheme(
    base_size = 11,
    base_family = "",
    base_line_size = base_size/22,
    base_rect_size = base_size/22,
    grid = TRUE
)
```
Arguments
- **base_size**: base font size, given in pts.
- **base_family**: base font family
- **base_line_size**: base size for line elements
- **base_rect_size**: base size for rect elements
- **grid**: a Boolean indicating if the grid is on (TRUE) or off (FALSE). This could also be a character indicating x or y.

Value
- ggplot2 theme used in rxode2

See Also
- Other rxode2 plotting: `plot.rxSolve()`

Description
- rxode2 to symengine environment

Usage
```
rxToSE(x, envir = NULL, progress = FALSE, promoteLinSens = TRUE)
```

```
.rxToSE(x, envir = NULL, progress = FALSE)
```

```
rxFromSE(x, unknownDerivatives = c("forward", "central", "error"))
```

```
.rxFromSE(x)
```

Arguments
- **x**: expression
- **envir**: default is NULL; Environment to put symengine variables in.
- **progress**: shows progress bar if true.
- **promoteLinSens**: Promote solved linear compartment systems to sensitivity-based solutions.
- **unknownDerivatives**: When handling derivatives from unknown functions, the translator will translate into different types of numeric derivatives. The currently supported methods are:
  - 'forward' for forward differences
  - 'central' for central differences
  - 'error' for throwing an error for unknown derivatives
Value

An rxode2 symengine environment

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,
  modelPrefix = "",
  md5 = "",
  modName = NULL,
  modVars = FALSE,
  ...
)

## Default S3 method:
rxTrans(
  model,
  modelPrefix = "",
  md5 = "",
  modName = NULL,
  modVars = FALSE,
  ...
)

## S3 method for class 'character'
rxTrans(
  model,
  modelPrefix = "",
  md5 = "",
  modName = NULL,
  modVars = 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 rxode2 Syntax below.

modelPrefix

Prefix of the model functions that will be compiled to make sure that multiple rxode2 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.

... Ignored parameters.

Value

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

Author(s)

Matthew L.Fidler

See Also

rxode2(), rxCompile().

Description

Compress/Decompress rxode2 ui

Usage

rxUiDecompress(ui)

rxUiCompress(ui)
Arguments

ui    
rxode2 ui object

Value

A compressed or decompressed rxui object

Author(s)

Matthew L. Fidler

Examples

```r
one.cmt <- function() {
  ini({
    ## You may label each parameter with a comment
    tka <- 0.45 # Log Ka
    tcl <- log(c(0, 2.7, 100)) # Log Cl
    ## This works with interactive models
    ## You may also label the preceding line with label("label text")
    tv <- 3.45; label("log V")
    ## the label("Label name") works with all models
    eta.ka ~ 0.6
    eta.cl ~ 0.3
    eta.v ~ 0.1
    add.sd <- 0.7
  })
  model({
    ka <- exp(tka + eta.ka)
    cl <- exp(tcl + eta.cl)
    v <- exp(tv + eta.v)
    linCmt() ~ add(add.sd) | tmp
  })
}

f <- rxode2(one.cmt)
print(class(f))
print(is.environment(f))

f <- rxUiDecompress(f)
print(class(f))
print(is.environment(f))

f <- rxUiCompress(f)
print(class(f))
print(is.environment(f))
```
Description

S3 for getting information from UI model

Usage

```r
## S3 method for class 'cmtLines'
rxUiGet(x, ...)

## S3 method for class 'dvidLine'
rxUiGet(x, ...)

## S3 method for class 'paramsLine'
rxUiGet(x, ...)

## S3 method for class 'simulationSigma'
rxUiGet(x, ...)

## S3 method for class 'simulationModel'
rxUiGet(x, ...)

## S3 method for class 'theta'
rxUiGet(x, ...)

## S3 method for class 'lstChr'
rxUiGet(x, ...)

## S3 method for class 'omega'
rxUiGet(x, ...)

## S3 method for class 'funTxt'
rxUiGet(x, ...)

## S3 method for class 'allCovs'
rxUiGet(x, ...)

## S3 method for class 'muRefTable'
rxUiGet(x, ...)

## S3 method for class 'multipleEndpoint'
rxUiGet(x, ...)
```
## S3 method for class 'funPrint'
rxUiGet(x, ...)

## S3 method for class 'fun'
rxUiGet(x, ...)

## S3 method for class 'md5'
rxUiGet(x, ...)

## S3 method for class 'ini'
rxUiGet(x, ...)

## S3 method for class 'iniFun'
rxUiGet(x, ...)

## S3 method for class 'modelFun'
rxUiGet(x, ...)

## S3 method for class 'modelDesc'
rxUiGet(x, ...)

## S3 method for class 'thetaLower'
rxUiGet(x, ...)

## S3 method for class 'thetaUpper'
rxUiGet(x, ...)

## Default S3 method:
rxUiGet(x, ...)

### Arguments

- **x**: list of (UIenvironment, exact). UI environment is the parsed function for rxode2. exact is a boolean that says if an exact match is required.
- **...**: Other arguments

### Value

value that was requested from the UI object

### Author(s)

Matthew Fidler
Description

Care should be taken with this method not to encounter the birthday problem, described [here](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage

```r
rxunif(min = 0, max = 1, n = 1L, ncores = 1L)
```

Arguments

- `min, max` lower and upper limits of the distribution. Must be finite.
- `n` number of observations. If `length(n) > 1`, the length is taken to be the number required.
- `ncores` Number of cores for the simulation

`rxnorm` simulates using the threefry sitmo generator.

`rxnormV` used to simulate with the vandercorput simulator, but since it didn’t satisfy the normal properties it was changed to simple be an alias of `rxnorm`. It is no longer supported in `rxode2` blocks.

Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the `rxode2` environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the `rxode2` engine with `rxSetSeed()`.

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value

uniform random numbers

Examples

```r
## Use threefry engine
```
rxunif(min = 0, max = 4, n = 10) # with rxunif you have to explicitly state n
rxunif(min = 0, max = 4, n = 10, ncores = 2) # You can parallelize the simulation using openMP
rxunif()

## This example uses `rxunif` directly in the model

```
rx <- rxode2({
a <- rxunif(0, 3)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

---

---

### rxUnloadAll

Unloads all rxode2 compiled DLLs

**Description**

Unloads all rxode2 compiled DLLs

**Usage**

```
rxUnloadAll()
```

**Value**

- List of rxode2 dlls still loaded
- boolean of if all rxode2 dlls have been unloaded

**Examples**

```
print(rxUnloadAll())
```
**rxUse**

*Use model object in your package*

---

**Description**

Use model object in your package

**Usage**

```
rxUse(obj, overwrite = TRUE, compress = "bzip2", internal = FALSE)
```

**Arguments**

- `obj` model to save.
- `overwrite` By default, `use_data()` will not overwrite existing files. If you really want to do so, set this to `TRUE`.
- `compress` Choose the type of compression used by `save()`. Should be one of "gzip", "bzip2", or "xz".
- `internal` If this is run internally. By default this is FALSE

**Value**

Nothing: This is used for its side effects and shouldn’t be called by a user

---

**rxValidate**

*Validate rxode2 This allows easy validation/qualification of nlmixr by running the testing suite on your system.*

---

**Description**

Validate rxode2 This allows easy validation/qualification of nlmixr by running the testing suite on your system.

**Usage**

```
rxValidate(type = NULL, skipOnCran = TRUE)
rxTest(type = NULL, skipOnCran = TRUE)
```

**Arguments**

- `type` Type of test or filter of test type, When this is an expression, evaluate the contents, respecting `skipOnCran`
- `skipOnCran` when `TRUE` skip the test on CRAN.
rxweibull

Simulate Weibull variable from threefry generator

Description
Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/). Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage
rxweibull(shape, scale = 1, n = 1L, ncores = 1L)

Arguments
shape, scale  shape and scale parameters, the latter defaulting to 1.
n  number of observations. If length(n) > 1, the length is taken to be the number required.
ncores  Number of cores for the simulation

Details
Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to either run all simulations in the rxode2 environment once (therefore one seed or series of seeds for the whole simulation), pre-generate all random variables used for the simulation, or seed the rxode2 engine with rxSetSeed()

Internally each ID is seeded with a unique number so that the results do not depend on the number of cores used.

Value
Weibull random deviates
Examples

## Use threefry engine

with rxweibull you have to explicitly state n
rxweibull(shape = 1, scale = 4, n = 10)

You can parallelize the simulation using openMP
rxweibull(shape = 1, scale = 4, n = 10, ncores = 2)

rxweibull(3)

This example uses `rxweibull` directly in the model

```
rx <- rxode2({
a <- rxweibull(1, 3)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

Description

This is a dosing geom that shows the vertical lines where a dose occurs

Usage

```
stat_amt(  
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

```
geom_amt(  
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
```

```
```
stat_amt

    inherit.aes = TRUE,
    ...)

Arguments

mapping Set of aesthetic mappings created by \texttt{aes()} or \texttt{aes()}. If specified and \texttt{inherit.aes} = \texttt{TRUE} (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data The data to be displayed in this layer. There are three options:
If \texttt{NULL}, the default, the data is inherited from the plot data as specified in the call to \texttt{ggplot()}. A \texttt{data.frame}, or other object, will override the plot data. All objects will be fortified to produce a data frame. See \texttt{fortify()} for which variables will be created.
A function will be called with a single argument, the plot data. The return value must be a \texttt{data.frame}, and will be used as the layer data. A function can be created from a formula (e.g. \texttt{\~ head(.x, 10)}).

position Position adjustment, either as a string, or the result of a call to a position adjustment function.

show.legend logical. Should this layer be included in the legends? \texttt{NA}, the default, includes if any aesthetics are mapped. \texttt{FALSE} never includes, and \texttt{TRUE} always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If \texttt{FALSE}, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. \texttt{borders()}. ... Other arguments passed on to \texttt{layer()}. These are often aesthetics, used to set an aesthetic to a fixed value, like \texttt{colour = "red"} or \texttt{size = 3}. They may also be parameters to the paired geom/stat.

Details

Requires the following aesthetics:

- \texttt{x} representing the x values, usually time
- \texttt{amt} representing the dosing values; They are missing or zero when no dose is given

Value

This returns a \texttt{stat_amt} in context of a \texttt{ggplot2} plot

Examples

```
library(rxode2)
library(units)
```
## Model from RxODE tutorial

```r
mod1 <- rxode2(
  KA=2.94E-01,  
  CL=1.86E+01,  
  V2=4.02E+01,  
  Q=1.05E+01,   
  V3=2.97E+02,  
  Kin=1,        
  Kout=1,       
  EC50=200,     
  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
)
```

## These are making the more complex regimens of the rxode2 tutorial

```r
## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000, ii=12, until=set_units(5, "days"))

## qd for 5 days
qd <- et(timeUnits="hr") %>%
  et(amt=20000, ii=24, until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days
et <- seq(bid,qd) %>%
  et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et, addDosing=TRUE)
```

# by default dotted and under-stated
plot(bidQd, C2) + geom_amt(aes(amt=amt))

# of course you can make it a bit more visible
plot(bidQd, C2) + geom_amt(aes(amt=amt), col="red", lty=1, size=1.2)

---

### stat_cens

This is a censoring geom that shows the left or right censoring specified in the nlmixr input data-set or fit
Usage

```r
stat_cens(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  width = 0.01,
  ...
)
```

```r
geom_cens(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  width = 0.01,
  ...
)
```

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a `formula` (e.g. `~ head(.x, 10)`).

- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.

- **show.legend**: logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

- **inherit.aes**: If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

- **width**: represents the width (in \ censoring box

- **...**: Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
Details

Requires the following aesthetics:

- \( x \) Represents the independent variable, often the time scale
- \( y \) represents the dependent variable
- \( \text{CENS} \) for the censoring information; (-1 right censored, 0 no censoring or 1 left censoring)
- \( \text{LIMIT} \) which represents the corresponding limit

Will add boxes representing the areas of the fit that were censored.

Value

This returns a ggplot2 stat

Description

This prints the expanded information about the rxode2 object.

Usage

```r
# S3 method for class 'rxode2'
summary(object, ...)
```

Arguments

- `object` rxode2 object
- `...` Ignored parameters

Value

object is returned

Author(s)

Matthew L. Fidler
### update.rxUi

**Update for rxUi**

**Description**
Update for rxUi

**Usage**
```r
## S3 method for class 'rxUi'
update(object, ..., envir = parent.frame())
```

**Arguments**
- `object`: rxode2 UI object
- `...`: Lines to update
- `envir`: Environment for evaluating ini() style calls

**Value**
a new rxode2 updated UI object

### uppergamma

**uppergamma: upper incomplete gamma function**

**Description**
This is the tgamma from the boost library

**Usage**
```r
uppergamma(a, z)
```

**Arguments**
- `a`: The numeric 'a' parameter in the upper incomplete gamma
- `z`: The numeric 'z' parameter in the upper incomplete gamma

**Details**
The uppergamma function is given by:

\[
uppergamma(a, z) = \int_z^{\infty} t^{a-1} e^{-t} dt
\]

**Value**
uppergamma results
Author(s)
Matthew L. Fidler

Examples

\texttt{uppergamma}(1, 3)
\texttt{uppergamma}(1:3, 3)
\texttt{uppergamma}(1, 1:3)
Index

* Internal
  * odeMethodToInt, 41
  * plot.rxSolve, 42
* Nonlinear regression
  * rxode2, 77
* ODE models
  * rxode2, 77
* Ordinary differential equations
  * rxode2, 77
* PK/PD
  * genShinyApp.template, 17
* Pharmacodynamics (PD)
  * rxode2, 77
* Pharmacokinetics (PK)
  * rxode2, 77
* Query model information
  * rxDfdy, 59
  * rxLhs, 73
  * rxParams, 87
  * rxState, 120
* datasets
  * rxReservedKeywords, 97
  * rxSyntaxFunctions, 124
* models
  * rxode2, 77
* nonlinear
  * genShinyApp.template, 17
  * rxode2, 77
* pharmacometrics
  * genShinyApp.template, 17
* rxode2 plotting
  * plot.rxSolve, 42
  * rxTheme, 126
* simulation
  * genShinyApp.template, 17
  * .C(), 54
  * .Call(), 54
  * .copyUI, 5
  * .handleSingleErrTypeNormOrTFoceiBase, 6
  * .modelHandleModelLines, 7
  * .quoteCallInfoLines, 8
  * .rxFromSE (rxToSE), 127
  * .rxInCmtGen, 8
  * .rxToSE (rxToSE), 127
  * .rxWithOptions, 9
  * .rxWithWd, 9
  * add.dosing(), 83
  * add.sampling(), 83
  * aes(), 138, 140
  * aes_(), 138, 140
  * assertRxUi, 10
  * assertRxUiEstimatedResiduals (assertRxUi), 10
  * assertRxUiMixedOnly (assertRxUi), 10
  * assertRxUiMuRefOnly (assertRxUi), 10
  * assertRxUiNormal (assertRxUi), 10
  * assertRxUiPopulationOnly (assertRxUi), 10
  * assertRxUiPrediction (assertRxUi), 10
  * assertRxUiRandomOnIdOnly (assertRxUi), 10
  * assertRxUiSingleEndpoint (assertRxUi), 10
  * assertRxUiTransformNormal (assertRxUi), 10
  * borders(), 138, 140
  * erf, 12
  * et(), 79, 83
  * eventTable(), 18, 79, 83, 111, 119
  * expit (logit), 38
  * fortify(), 138, 140
  * gammap, 13
  * gammapDer, 14
  * gammapInv, 14
<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gammapInva</code> (gammapInv)</td>
<td>14</td>
</tr>
<tr>
<td><code>gammaq</code></td>
<td>15</td>
</tr>
<tr>
<td><code>gammaqInv</code></td>
<td>16</td>
</tr>
<tr>
<td><code>gammaqInva</code> (gammaqInv)</td>
<td>16</td>
</tr>
<tr>
<td><code>genShinyApp.template</code></td>
<td>17</td>
</tr>
<tr>
<td><code>geom_amt</code> (stat_amt)</td>
<td>137</td>
</tr>
<tr>
<td><code>geom_cens</code> (stat_cens)</td>
<td>139</td>
</tr>
<tr>
<td><code>getRxThreads</code></td>
<td>19</td>
</tr>
<tr>
<td><code>ggplot()</code></td>
<td>138, 140</td>
</tr>
<tr>
<td><code>ini (ini.rxUi)</code></td>
<td>20</td>
</tr>
<tr>
<td><code>ini.rxUi</code></td>
<td>20</td>
</tr>
<tr>
<td><code>layer()</code></td>
<td>138, 140</td>
</tr>
<tr>
<td><code>llikBeta</code></td>
<td>22</td>
</tr>
<tr>
<td><code>llikBinom</code></td>
<td>23</td>
</tr>
<tr>
<td><code>llikCauchy</code></td>
<td>24</td>
</tr>
<tr>
<td><code>llikChisq</code></td>
<td>25</td>
</tr>
<tr>
<td><code>llikExp</code></td>
<td>26</td>
</tr>
<tr>
<td><code>llikF</code></td>
<td>27</td>
</tr>
<tr>
<td><code>llikGamma</code></td>
<td>28</td>
</tr>
<tr>
<td><code>llikGeom</code></td>
<td>29</td>
</tr>
<tr>
<td><code>llikNbinom</code></td>
<td>30</td>
</tr>
<tr>
<td><code>llikNbinomMu</code></td>
<td>31</td>
</tr>
<tr>
<td><code>llikPois</code></td>
<td>32</td>
</tr>
<tr>
<td><code>llikT</code></td>
<td>35</td>
</tr>
<tr>
<td><code>llikUnif</code></td>
<td>36</td>
</tr>
<tr>
<td><code>llikWeibull</code></td>
<td>37</td>
</tr>
<tr>
<td><code>logit</code></td>
<td>38</td>
</tr>
<tr>
<td><code>logitNormInfo (logit)</code></td>
<td>38</td>
</tr>
<tr>
<td><code>lowergamma</code></td>
<td>39</td>
</tr>
<tr>
<td><code>model (model.function)</code></td>
<td>40</td>
</tr>
<tr>
<td><code>model.function</code></td>
<td>40</td>
</tr>
<tr>
<td><code>odeMethodToInt</code></td>
<td>41</td>
</tr>
<tr>
<td><code>plot.rxSolve</code></td>
<td>42, 127</td>
</tr>
<tr>
<td><code>plot.rxSolveConfint1</code> (plot.rxSolve)</td>
<td>42</td>
</tr>
<tr>
<td><code>predict.rxEt (rxSolve)</code></td>
<td>106</td>
</tr>
<tr>
<td><code>predict.rxode2 (rxSolve)</code></td>
<td>106</td>
</tr>
<tr>
<td><code>predict.rxParams (rxSolve)</code></td>
<td>106</td>
</tr>
<tr>
<td><code>predict.rxSolve (rxSolve)</code></td>
<td>106</td>
</tr>
<tr>
<td><code>probit</code></td>
<td>43</td>
</tr>
<tr>
<td><code>probitInv (probit)</code></td>
<td>43</td>
</tr>
<tr>
<td><code>probitNormInfo (logit)</code></td>
<td>38</td>
</tr>
<tr>
<td><code>rename.function (rxRename)</code></td>
<td>96</td>
</tr>
<tr>
<td><code>rename.rxUi (rxRename)</code></td>
<td>96</td>
</tr>
<tr>
<td><code>rxAllowUnload</code></td>
<td>44</td>
</tr>
<tr>
<td><code>rxAppendModel</code></td>
<td>44</td>
</tr>
<tr>
<td><code>rxAssignControlValue</code></td>
<td>46</td>
</tr>
<tr>
<td><code>rxAssignPtr</code></td>
<td>46</td>
</tr>
<tr>
<td><code>rxbeta</code></td>
<td>47</td>
</tr>
<tr>
<td><code>rxbinom</code></td>
<td>48</td>
</tr>
<tr>
<td><code>rxcauchy</code></td>
<td>49</td>
</tr>
<tr>
<td><code>rxchisq</code></td>
<td>51</td>
</tr>
<tr>
<td><code>rxClean</code></td>
<td>52</td>
</tr>
<tr>
<td><code>rxCompile</code></td>
<td>53</td>
</tr>
<tr>
<td><code>rxCompile()</code></td>
<td>129</td>
</tr>
<tr>
<td><code>rxControl (rxSolve)</code></td>
<td>106</td>
</tr>
<tr>
<td><code>rxControlUpdateSens</code></td>
<td>55</td>
</tr>
<tr>
<td><code>rxCores</code> (getRxThreads)</td>
<td>19</td>
</tr>
<tr>
<td><code>rxCreateCache</code></td>
<td>56</td>
</tr>
<tr>
<td><code>rxD</code></td>
<td>56</td>
</tr>
<tr>
<td><code>rxD()</code></td>
<td>63</td>
</tr>
<tr>
<td><code>rxDelete</code></td>
<td>57</td>
</tr>
<tr>
<td><code>rxDerived</code></td>
<td>57</td>
</tr>
<tr>
<td><code>rxDerived</code></td>
<td>57</td>
</tr>
<tr>
<td><code>rxDfdy</code></td>
<td>59, 73, 89, 120</td>
</tr>
<tr>
<td><code>rexexp</code></td>
<td>60</td>
</tr>
<tr>
<td><code>rxf</code></td>
<td>61</td>
</tr>
<tr>
<td><code>rxFromSE (rxToSE)</code></td>
<td>127</td>
</tr>
<tr>
<td><code>rxFun</code></td>
<td>63</td>
</tr>
<tr>
<td><code>rxgamma</code></td>
<td>64</td>
</tr>
<tr>
<td><code>rxgeom</code></td>
<td>66</td>
</tr>
<tr>
<td><code>rxGetControl</code></td>
<td>67</td>
</tr>
<tr>
<td><code>rxGetLin</code></td>
<td>68</td>
</tr>
<tr>
<td><code>rxGetrxode2</code></td>
<td>68</td>
</tr>
<tr>
<td><code>rxHtml</code></td>
<td>69</td>
</tr>
<tr>
<td><code>rxIndLin_</code></td>
<td>71</td>
</tr>
<tr>
<td><code>rxIndLinState</code></td>
<td>70</td>
</tr>
<tr>
<td><code>rxIndLinStrategy</code></td>
<td>70</td>
</tr>
<tr>
<td><code>rxInits</code></td>
<td>60, 73, 89, 120</td>
</tr>
<tr>
<td><code>rxInv</code></td>
<td>72</td>
</tr>
<tr>
<td><code>rxIsCurrent</code></td>
<td>72</td>
</tr>
<tr>
<td><code>rxLhs</code></td>
<td>60, 73, 89, 120</td>
</tr>
<tr>
<td><code>rxLock</code></td>
<td>73</td>
</tr>
<tr>
<td><code>rxModelVars</code></td>
<td>60, 73, 89, 120</td>
</tr>
<tr>
<td><code>rxModelVars()</code></td>
<td>129</td>
</tr>
<tr>
<td><code>rxnbinom</code></td>
<td>74</td>
</tr>
<tr>
<td><code>rxnbinomMu (rxnbinom)</code></td>
<td>74</td>
</tr>
<tr>
<td><code>rxNorm</code></td>
<td>75</td>
</tr>
<tr>
<td><code>rxnorm (rxnormV)</code></td>
<td>76</td>
</tr>
<tr>
<td><code>rxnormV</code></td>
<td>76</td>
</tr>
<tr>
<td><code>RxODE (rxode2)</code></td>
<td>77</td>
</tr>
<tr>
<td><code>rxode (rxode2)</code></td>
<td>77</td>
</tr>
</tbody>
</table>
INDEX

rxode2, 73, 77
rxode2(), 18, 54, 119, 120, 129
rxOptExpr, 85
rxord, 86
rxParam (rxParams), 87
rxParams, 60, 73, 87, 120
rxPkg, 90
rxpois, 91
rxPp, 92
rxPreferredDistributionName, 93
rxProgress, 94
rxProgressAbort (rxProgress), 94
rxProgressStop (rxProgress), 94
rxRemoveControl, 95
rxRename, 96
rxReservedKeywords, 97
rxRmFun (rfun), 63
rxS, 97
rxSetControl, 98
rxSetCovariateNamesForPiping, 98
rxSetIni0, 100
rxSetProd, 100
rxSetProgressBar, 101
rxSetSum, 101
rxShiny, 102
rxSimThetaOmega, 103
rxSolve, 106
rxSolve(), 17
rxState, 60, 73, 89, 120
rxSumProdModel, 120
rxSupportedFuns, 121
rxSupressMsg, 122
rxSymInvChol, 123
rxSymInvCholCreate(), 123
rxSyncOptions, 124
rxSyntaxFunctions, 124
rxt, 125
rxTempDir, 126
rxTest (rxValidate), 135
rxTheme, 43, 126
rxTick (rxProgress), 94
rxToSE, 127
rxTrans, 128
rxTrans(), 54
rxUiCompress (rxUiDecompress), 129
rxUiDecompress, 129
rxUiGet (rxUiGet.cmlines), 131
rxUiGet.cmlines, 131
rxunif, 133
rxUnloadAll, 134
rxUnlock (rxLock), 73
rxUse, 135
rxValidate, 135
rxweibull, 136
save(), 135
setRxThreads (getRxThreads), 19
setRxThreads(), 112
simulate.rxode2 (rxSolve), 106
simulate.rxParams (rxSolve), 106
simulate.rxSolve (rxSolve), 106
solve.rxEt (rxSolve), 106
solve.rxode2 (rxSolve), 106
solve.rxParams (rxSolve), 106
solve.rxSolve (rxSolve), 106
solve.rxUi (rxSolve), 106
statamt, 137
stat_cens, 139
summary.rxode2, 141
update.rxSolve (rxSolve), 106
update.rxUi, 142
uppergamma, 142
use_description(), 90
vname, 10
write.template.server
   (genShinyApp.template), 17
write.template.ui
   (genShinyApp.template), 17
write.template.ui(), 18