Package ‘rxode2’

January 31, 2024

Version 2.1.2
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,
nlme, remotes, rlang, rmarkdown, scales, shiny, stringi,
symengine, testthat, tidyR, usethis, vdiffR (>= 1.0), withr,
xgxr, pillar, tibble, units (>= 0.6-0), rsconnect, devtools,
patchwork, nlmixr2data, lifecycle, kableExtra
Imports PreciseSums (>= 0.3), Rcpp (>= 0.12.3), backports, cli (>= 2.0.0), checkmate, ggplot2 (>= 3.4.0), inline, lotri (>= 0.4.0), magrittr, memoise, methods, rex, sys, tools, utils,
rxode2ll (>= 2.0.9), rxode2et (>= 2.0.9), rxode2parse (>= 2.0.16), rxode2random (>= 2.0.11), 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)
https://github.com/nlmixr2/rxode2/
RoxygenNote 7.3.1
Biarch true
LinkingTo rxode2parse (>= 2.0.12), rxode2random, PreciseSums (>= 0.3), Rcpp, RcppArmadillo (>= 0.9.300.2.0), BH
Encoding UTF-8
LazyData true
Language en-US

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], Hadley Wickham [ctb], Jack Dongarra [ctb], Jim Bunch [ctb], Linda Petzold [ctb], Martin Maechler [ctb], Matt Dowle [ctb], Matteo Fasiolo [ctb], Nicholas J. Higham [ctb], Roger B. Sidje [ctb], Simon Frost [ctb], Yu Feng [ctb], Bill Denney [ctb] (<https://orcid.org/0000-0002-5759-428X>)

Repository CRAN
Date/Publication 2024-01-30 23:30:02 UTC

R topics documented:

.copyUi .................................................. 5
.handleSingleErrTypeNormOrTFoceiBase .............................. 6
.matchesLangTemplate ............................................. 7
.modelHandleModelLines ........................................... 7
.quoteCallInfoLines ............................................... 8
.rxLinCmtGen .................................................... 9
.rxWithOptions .................................................. 9
.rxWithWd ....................................................... 10
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.as.ini</td>
<td>11</td>
</tr>
<tr>
<td>.as.model</td>
<td>13</td>
</tr>
<tr>
<td>.as.rxUi</td>
<td>15</td>
</tr>
<tr>
<td>.assertRxUi</td>
<td>16</td>
</tr>
<tr>
<td>.binomProbs</td>
<td>18</td>
</tr>
<tr>
<td>.erf</td>
<td>21</td>
</tr>
<tr>
<td>.gammap</td>
<td>21</td>
</tr>
<tr>
<td>.gammapDer</td>
<td>22</td>
</tr>
<tr>
<td>.gammapInv</td>
<td>23</td>
</tr>
<tr>
<td>.gammapq</td>
<td>24</td>
</tr>
<tr>
<td>.gammapqInv</td>
<td>25</td>
</tr>
<tr>
<td>.genShinyApp.template</td>
<td>26</td>
</tr>
<tr>
<td>.getRxThreads</td>
<td>27</td>
</tr>
<tr>
<td>.ini.rxUi</td>
<td>28</td>
</tr>
<tr>
<td>.ini&lt;-</td>
<td>31</td>
</tr>
<tr>
<td>.llikBeta</td>
<td>31</td>
</tr>
<tr>
<td>.llikBinom</td>
<td>32</td>
</tr>
<tr>
<td>.llikCauchy</td>
<td>34</td>
</tr>
<tr>
<td>.llikChisq</td>
<td>35</td>
</tr>
<tr>
<td>.llikExp</td>
<td>36</td>
</tr>
<tr>
<td>.llikF</td>
<td>37</td>
</tr>
<tr>
<td>.llikGamma</td>
<td>38</td>
</tr>
<tr>
<td>.llikGeom</td>
<td>39</td>
</tr>
<tr>
<td>.llikNbinom</td>
<td>40</td>
</tr>
<tr>
<td>.llikNbinomMu</td>
<td>42</td>
</tr>
<tr>
<td>.llikNorm</td>
<td>43</td>
</tr>
<tr>
<td>.llikPois</td>
<td>44</td>
</tr>
<tr>
<td>.llikT</td>
<td>45</td>
</tr>
<tr>
<td>.llikUnif</td>
<td>46</td>
</tr>
<tr>
<td>.llikWeibull</td>
<td>47</td>
</tr>
<tr>
<td>.logit</td>
<td>48</td>
</tr>
<tr>
<td>.lowergamma</td>
<td>50</td>
</tr>
<tr>
<td>.meanProbs</td>
<td>51</td>
</tr>
<tr>
<td>.model.function</td>
<td>53</td>
</tr>
<tr>
<td>.model&lt;-</td>
<td>55</td>
</tr>
<tr>
<td>.modelExtract</td>
<td>55</td>
</tr>
<tr>
<td>.odeMethodToInt</td>
<td>58</td>
</tr>
<tr>
<td>.plot.rxSolve</td>
<td>59</td>
</tr>
<tr>
<td>.probit</td>
<td>59</td>
</tr>
<tr>
<td>.rxAllowUnload</td>
<td>60</td>
</tr>
<tr>
<td>.rxAppendModel</td>
<td>61</td>
</tr>
<tr>
<td>.rxAssignControlValue</td>
<td>62</td>
</tr>
<tr>
<td>.rxAssignPtr</td>
<td>63</td>
</tr>
<tr>
<td>.rxbeta</td>
<td>63</td>
</tr>
<tr>
<td>.rxbinom</td>
<td>64</td>
</tr>
<tr>
<td>.rxcauchy</td>
<td>66</td>
</tr>
<tr>
<td>.rxchisq</td>
<td>67</td>
</tr>
<tr>
<td>.rxClean</td>
<td>68</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>rxCompile</td>
<td>69</td>
</tr>
<tr>
<td>rxControlUpdateSens</td>
<td>71</td>
</tr>
<tr>
<td>rxCreateCache</td>
<td>72</td>
</tr>
<tr>
<td>rxD</td>
<td>72</td>
</tr>
<tr>
<td>rxDelete</td>
<td>73</td>
</tr>
<tr>
<td>rxDfdy</td>
<td>74</td>
</tr>
<tr>
<td>rxexp</td>
<td>74</td>
</tr>
<tr>
<td>rxf</td>
<td>76</td>
</tr>
<tr>
<td>rxFun</td>
<td>77</td>
</tr>
<tr>
<td>rXgamma</td>
<td>80</td>
</tr>
<tr>
<td>rxgeom</td>
<td>81</td>
</tr>
<tr>
<td>rxGetControl</td>
<td>83</td>
</tr>
<tr>
<td>rxGetLin</td>
<td>83</td>
</tr>
<tr>
<td>rxGetxode2</td>
<td>84</td>
</tr>
<tr>
<td>rxHtml</td>
<td>85</td>
</tr>
<tr>
<td>rxIndLinState</td>
<td>85</td>
</tr>
<tr>
<td>rxIndLinStrategy</td>
<td>86</td>
</tr>
<tr>
<td>rxIndLin_</td>
<td>86</td>
</tr>
<tr>
<td>rxInv</td>
<td>87</td>
</tr>
<tr>
<td>rxIsCurrent</td>
<td>88</td>
</tr>
<tr>
<td>rxLhs</td>
<td>88</td>
</tr>
<tr>
<td>rxLock</td>
<td>89</td>
</tr>
<tr>
<td>rxnbinom</td>
<td>89</td>
</tr>
<tr>
<td>rxNorm</td>
<td>91</td>
</tr>
<tr>
<td>rxnormV</td>
<td>92</td>
</tr>
<tr>
<td>rxode2</td>
<td>93</td>
</tr>
<tr>
<td>rxode2&lt;-</td>
<td>112</td>
</tr>
<tr>
<td>rxOptExpr</td>
<td>114</td>
</tr>
<tr>
<td>rxord</td>
<td>115</td>
</tr>
<tr>
<td>rxParams</td>
<td>116</td>
</tr>
<tr>
<td>rxPkg</td>
<td>118</td>
</tr>
<tr>
<td>rxpois</td>
<td>119</td>
</tr>
<tr>
<td>rxPp</td>
<td>120</td>
</tr>
<tr>
<td>rxPreferredDistributionName</td>
<td>122</td>
</tr>
<tr>
<td>rxProgress</td>
<td>123</td>
</tr>
<tr>
<td>rxRemoveControl</td>
<td>124</td>
</tr>
<tr>
<td>rxRename</td>
<td>124</td>
</tr>
<tr>
<td>rxReservedKeywords</td>
<td>126</td>
</tr>
<tr>
<td>rxResidualError</td>
<td>126</td>
</tr>
<tr>
<td>rxS</td>
<td>127</td>
</tr>
<tr>
<td>rxSetControl</td>
<td>128</td>
</tr>
<tr>
<td>rxSetCovariateNamesForPiping</td>
<td>128</td>
</tr>
<tr>
<td>rxSetPipingAuto</td>
<td>130</td>
</tr>
<tr>
<td>rxSetProd</td>
<td>131</td>
</tr>
<tr>
<td>rxSetProgressBar</td>
<td>131</td>
</tr>
<tr>
<td>rxSetSum</td>
<td>132</td>
</tr>
<tr>
<td>rxShiny</td>
<td>132</td>
</tr>
<tr>
<td>rxSimThetaOmega</td>
<td>133</td>
</tr>
</tbody>
</table>
This copies the rxode2 UI object so it can be modified

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

.handleSingleErrTypeNormOrTFOceiBase

Handle the single error for normal or t distributions

Description

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
.matchesLangTemplate
Check if a language object matches a template language object

Description

• If template == str2lang("."), it will match anything.
• If template == str2lang(".name"), it will match any name.
• If template == str2lang(".call()"), it will match any call.

Usage

.matchesLangTemplate(x, template)

Arguments

x
The object to check

template
The template object it should match

Value

TRUE if it matches, FALSE, otherwise

Examples

.matchesLangTemplate(str2lang("d/dt(foo)"), str2lang("d/dt(.name)"))
.matchesLangTemplate(str2lang("d/dt(foo)"), str2lang("d/foo(.name)"))
.matchesLangTemplate(str2lang("d/dt(foo)"), str2lang("d/.")))
Arguments
modelLines  The model lines that are being considered
rxui       The rxode2 UI object
modifyIni  Should the ini() be considered
append     This is a boolean to determine if the lines are appended in piping. The possible values for this is:
            • TRUE which is when the lines are appended to the model instead of replaced
            • FALSE when the lines are replaced in the model (default)
            • NA is when the lines are pre-pended to the model instead of replaced
            • lhs expression, which will append the lines after the last observed line of the expression lhs
auto       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).
cov        is a character vector of variables that should be assumed to be covariates. This will override automatic promotion to a population parameter estimate (or an eta)
envir      Environment for evaluation

Value
New UI

Author(s)
Matthew L. Fidler

Returns quoted call information

Description
Returns quoted call information

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

Arguments
callInfo    Call information
envir       Environment for evaluation (if needed)
iniDf       The parent model iniDf when piping in a ini block (NULL otherwise)
.rxLinCmtGen

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

Author(s)
Matthew L. Fidler

---

.rxWithOptions Temporarily set options then restore them while running code

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  Temporarily set options then restore them while running code

---

Description

Temporarily set options then restore them while running code

Usage

```r
.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

Value

value of code

Examples

```r
.rxWithWd(tempdir(), {
  getwd()
})

getwd()
```
**as.ini**

**Turn into an ini block for initialization**

**Description**

Turn into an ini block for initialization

**Usage**

```r
as.ini(x)
```

## S3 method for class 'character'
```r
as.ini(x)
```

## S3 method for class 'data.frame'
```r
as.ini(x)
```

## S3 method for class 'call'
```r
as.ini(x)
```

## S3 method for class 'lotriFix'
```r
as.ini(x)
```

## S3 method for class 'matrix'
```r
as.ini(x)
```

## Default S3 method:
```r
as.ini(x)
```

**Arguments**

- `x`  
  Item to convert to a rxode2/nlmixr2 ui ini expression

**Value**

rxode2 ini expression

**Author(s)**

Matthew L. Fidler

**Examples**

```r
ini <- quote(ini(
  tka <- log(1.57)
  tcl <- log(2.72)
  tv <- log(31.5)
```
eta.ka ~ 0.6
eta.cl ~ 0.3
eta.v ~ 0.1
add.sd <- 0.7
))

as.ini(l)

m <- lotri({
  eta.ka ~ 0.6
  eta.cl ~ 0.3
  eta.v ~ 0.1
})

as.ini(m)

one.compartment <- function() {
  ini({
    tka <- log(1.57)
    tcl <- log(2.72)
    tv <- log(31.5)
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  })
}

as.ini(one.compartment)

ui <- one.compartment()
as.ini(ui)

ui$iniDf

as.ini(ui$iniDf)

ini <- c("ini({
    "tka <- log(1.57)",
    "tcl <- log(2.72)",
    "tv <- log(31.5)",
    "eta.ka ~ 0.6",
    "eta.cl ~ 0.3",
    "eta.v ~ 0.1",
    "add.sd <- 0.7",
    "})")

as.ini(ini)

ini <- paste(ini, collapse="\n")

as.ini(ini)

---

as.model

**Turn into a model expression**

---

**Description**

Turn into a model expression

**Usage**

as.model(x)

## S3 method for class 'character'
as.model(x)

## S3 method for class 'call'
as.model(x)

## S3 method for class 'list'
as.model(x)

## Default S3 method:
as.model(x)

**Arguments**

x item to convert to a model(()) expression
Value

model expression

Author(s)

Matthew L. Fidler

Examples

```r
model <- quote(model({
  ka <- exp(tka + eta.ka)
  cl <- exp(tcl + eta.cl)
  v <- exp(tv + eta.v)
  d/dt(depot) = -ka * depot
  d/dt(center) = ka * depot - cl / v * center
  cp = center / v
  cp ~ add(add.sd)
}))

as.model(model)

one.compartment <- function() {
  ini({
    tka <- log(1.57)
    tcl <- log(2.72)
    tv <- log(31.5)
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  })
}

as.model(one.compartment)

ui <- one.compartment()

as.model(ui)

model <- c("model({",
  "ka <- exp(tka + eta.ka)",
  "cl <- exp(tcl + eta.cl)",
  "v <- exp(tv + eta.v)",
  "d/dt(depot) = -ka * depot",
  "d/dt(center) = ka * depot - cl / v * center",
  "cp = center / v",
  "cp ~ add(add.sd)"
})
```

"v <- exp(tv + eta.v)",
"d/dt(depot) = -ka * depot",
"d/dt(center) = ka * depot - cl / v * center",
"cp = center / v",
"cp ~ add(add.sd)",
"})")

as.model(model)
model <- paste(model, collapse="\n")

as.model(model)

---

**Description**

As rxode2 ui

**Usage**

as.rxUi(x)

## S3 method for class 'rxode2'
as.rxUi(x)

## S3 method for class 'rxode2tos'
as.rxUi(x)

## S3 method for class 'rxModelVars'
as.rxUi(x)

## S3 method for class `\'function\'`
as.rxUi(x)

## S3 method for class 'rxUi'
as.rxUi(x)

## Default S3 method:
as.rxUi(x)

**Arguments**

x Object to convert to rxUi object
assertRxUi

Value

rxUi object (or error if it cannot be converted)

Author(s)

Matthew L. Fidler

Examples

mod1 <- function() {
  ini(
    # central
    KA=2.94E-01
    CL=1.86E+01
    V2=4.02E+01
    # peripheral
    Q=1.05E+01
    V3=2.97E+02
    # effects
    Kin=1
    Kout=1
    EC50=200
  )
  model(
    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
    eff(0) <- 1
    d/dt(eff) <- Kin - Kout*(1-C2/(EC50+C2))*eff
  )
}

as.rxUi(mod1)

---

assertRxUi

Assert properties of the rxUi models

Description

Assert properties of the rxUi models
assertRxUi

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     Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.

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 – Make sure that the model residual distribution is normal or transformably normal
- assertRxUiNormal – Make sure that the model residual distribution is normal
- assertRxUiEstimatedResiduals – Make sure that the residual error parameters are estimated (not modeled).
- assertRxUiPopulationOnly – Make sure the model is the population only model (no mixed effects)
- assertRxUiMixedOnly – Make sure the model is a mixed effect model (not a population effect, only)
- assertRxUiPrediction – Make sure the model has predictions
- assertRxUiMuRefOnly – Make sure that all the parameters are mu-referenced
- assertRxUiRandomOnIdOnly – Make sure there are only random effects at the ID level

Value

the rxUi model

Author(s)

Matthew L. Fidler

Examples

```r
one.cmt <- function() {
  ini({
    tka <- 0.45; label("Ka")
    tcl <- log(c(0, 2.7, 100)); label("Cl")
    tv <- 3.45; label("V")
    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)
  })
  }

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

assertRxUiSingleEndpoint(one.cmt)
```

---

**binomProbs**

*Calculate expected confidence bands with binomial sampling distribution*

Description

This is meant to perform in the same way as quantile() so it can be a drop in replacement for code using quantile() but using distributional assumptions.
Usage

binomProbs(x, ...)

## Default S3 method:
binomProbs(
  x,
  probs = c(0.025, 0.05, 0.5, 0.95, 0.975),
  na.rm = FALSE,
  names = TRUE,
  onlyProbs = TRUE,
  n = 0L,
  m = 0L,
  pred = FALSE,
  piMethod = c("lim"),
  M = 5e+05,
  tol = .Machine$double.eps^0.25,
  ciMethod = c("wilson", "wilsonCorrect", "agrestiCoull", "wald", "wc", "ac"),
  ...)

Arguments

x numeric vector whose mean and probability based confidence values are wanted, NA and NaN values are not allowed in numeric vectors unless na.rm is TRUE.

... Arguments passed to default method, allows many different methods to be applied.

probs numeric vector of probabilities with values in the interval 0 to 1, inclusive. When 0, it represents the maximum observed, when 1, it represents the maximum observed. When 0.5 it represents the expected probability (mean).

na.rm logical; if true, any NA and NaN's are removed from x before the quantiles are computed.

names logical; if true, the result has a names attribute.

onlyProbs logical; if true, only return the probability based confidence interval/prediction interval estimates, otherwise return extra statistics.

n integer/integerish; this is the n used to calculate the prediction or confidence interval. When n=0 (default) use the number of non-NA observations. When calculating the prediction interval, this represents the number of observations used in the input ("true") distribution.

m integer. When using the prediction interval this represents the number of samples that will be observed in the future for the prediction interval.

pred Use a prediction interval instead of a confidence interval. By default this is FALSE.

piMethod gives the prediction interval method (currently only lim) from Lu 2020

M number of simulations to run for the LIM PI.

tol tolerance of root finding in the LIM prediction interval
ciMethod gives the method for calculating the confidence interval.
Can be:
- "argestiCoull" or "ac" – Agresti-Coull method. For a 95\% interval, this method does not use the concept of "adding 2 successes and 2 failures," but rather uses the formulas explicitly described in the following link: https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval#Agresti-Coull_Interval.
- "wilson" – Wilson Method
- "wilsonCorrect" or "wc" – Wilson method with continuity correction
- "wald" – Wald confidence interval or standard z approximation.

Details
It is used for confidence intervals with rxode2 solved objects using confint(mean="binom")

Value
By default the return has the probabilities as names (if named) with the points where the expected distribution are located given the sampling mean and standard deviation. If onlyProbs=FALSE then it would prepend mean, variance, standard deviation, minimum, maximum and number of non-NA observations.

Author(s)
Matthew L. Fidler

References

Examples
```r
x <- rbinom(7001, p=0.375, size=1)
binomProbs(x)

# you can also use the prediction interval
binomProbs(x, pred=TRUE)

# Can get some extra statistics if you request onlyProbs=FALSE
binomProbs(x, onlyProbs=FALSE)
```
x[2] <- NA_real_

binomProbs(x, onlyProbs=FALSE)

binomProbs(x, na.rm=TRUE)

---

**erf**

*Error function*

**Description**

Error function

**Usage**

```r
erf(x)
```

**Arguments**

- `x` vector or real values

**Value**

`erf` of `x`

**Author(s)**

Matthew L. Fidler

**Examples**

```r
erf(1.0)
```

---

**gammap**

*Gammap: normalized lower incomplete gamma function*

**Description**

This is the `gamma_p` from the boost library

**Usage**

```r
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 gamma \( p \) function is given by:

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

Value

\( \)  

Author(s)

Matthew L. Fidler

Examples

- \( \text{gammap}(1, 3) \)
- \( \text{gammap}(1:3, 3) \)
- \( \text{gammap}(1, 1:3) \)

Description

This is the gamma \( p \) derivative from the boost library

Usage

\( \text{gammapDer}(a, z) \)

Arguments

- \( a \)  
  The numeric ‘a’ parameter in the upper incomplete gamma

- \( z \)  
  The numeric ‘z’ parameter in the upper incomplete gamma

Value

\( \)  

Author(s)

Matthew L. Fidler
**Examples**

```
  gammapDer(1:3, 3)
  gammapDer(1, 1:3)
```

---

**Description**

`gammapInv` and `gammapInvA`: Inverses of normalized gamma function

**Usage**

```
  gammapInv(a, p)
  gammapInvA(x, p)
```

**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 = 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

**Value**

inverse gamma results

**Author(s)**

Matthew L. Fidler
gammaq

Examples

\[ \text{gammaqInv}(1:3, 0.5) \]
\[ \text{gammaqInv}(1, 1:3 / 3.1) \]
\[ \text{gammaqInv}(1:3, 1:3 / 3.1) \]
\[ \text{gammaqInv}(1:3, 1:3 / 3.1) \]

---

**Gammaq: normalized upper incomplete gamma function**

**Description**

This is the gamma_q from the boost library

**Usage**

\[ \text{gammaq}(a, z) \]

**Arguments**

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

**Details**

The gamma q function is given by:

\[ \text{gammaq} = \text{uppergamma}(a, z) / \text{gamma}(a) \]

**Value**

gammaq results

**Author(s)**

Matthew L. Fidler

**Examples**

\[ \text{gammaq}(1, 3) \]
\[ \text{gammaq}(1:3, 3) \]
\[ \text{gammaq}(1, 1:3) \]
### Description

*gammaqInv and gammaqInva: Inverses of normalized gamma function*

### Usage

- `gammaqInv(a, q)`
- `gammaqInva(x, q)`

### Arguments

- **a**: The numeric ‘a’ parameter in the upper incomplete gamma
- **q**: The numeric ‘q’ parameter in the upper incomplete gamma
- **x**: The numeric ‘x’ parameter in the upper incomplete gamma

### Details

With the equation:

\[ q = \text{gammaq}(a, x) \]

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 gammaq results

### Author(s)

Matthew L. Fidler

### Examples

- `gammaqInv(1:3, 0.5)`
- `gammaqInv(1, 1:3 / 3)`
- `gammaqInv(1:3, 1:3 / 3.1)`
- `gammaqInva(1:3, 1:3 / 3.1)`
Generate an example (template) of a dosing regimen shiny app

Description

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

Usage

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

Arguments

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

A PK/PD model is defined using `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:

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

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

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

Value

None, these functions are used for their side effects.
getRxThreads

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.posit.co).

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

```r
getRxThreads(verbose = FALSE)

setRxThreads(threads = NULL, percent = NULL, throttle = NULL)

rxCores(verbose = FALSE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>Display the value of relevant OpenMP settings</td>
</tr>
<tr>
<td>threads</td>
<td>NULL (default) rereads environment variables. 0 means to use all logical CPUs available. Otherwise a number ( \geq 1 )</td>
</tr>
<tr>
<td>percent</td>
<td>If provided it should be a number between 2 and 100; the percentage of logical CPUs to use. By default on startup, 50 percent.</td>
</tr>
</tbody>
</table>
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 min-
imize the time for waiting for another thread to finish. If the last item solved is
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

init.rxUi  

Ini 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(), append = NULL)

## Default S3 method:
ini(x, ..., envir = parent.frame(), append = NULL)

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

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.
append

Reorder theta parameters. NULL means no change to parameter order. A parameter name (as a character string) means to put the new parameter after the named parameter. A number less than or equal to zero means to put the parameter at the beginning of the list. A number greater than the last parameter number means to put the parameter at the end of the list.

Details

The \function{ini()} function is used in two different ways. The main way that it is used is to set the initial conditions and associated attributes (described below) in a model. The other way that it is used is for updating the initial conditions in a model, often using the pipe operator.

'\(\theta\)' and '\(\sigma\)' can be set using either \code{<-} or \code{=} such as \code{tvCL <- 1} or equivalently \code{tvCL = 1}. '\(\omega\)' can be set with a \code{~} such as \code{etaCL ~ 0.1}.

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. \code{tvCL <- 1} would set a named parameter of \code{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 \code{c(lower, estimate, upper)}. The vector may be given with just the estimate \code{c(estimate),} the lower bound and estimate \code{c(lower, estimate)}, or all three \code{c(lower, estimate, upper)}. To set an estimate and upper bound without a lower bound, set the lower bound to \(-\infty\). \code{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 \code{~}. To set a variance-covariance matrix with variance values of \code{2} and \code{3} and a covariance of -2.5 use \code{~c(2, 2.5, 3)}. To set the same matrix with names of \code{iivKa} and \code{iivCL}, use \code{iivKa + iivCL~c(2, 2.5, 3)}. To set a correlation matrix with standard deviations on the diagonal, use \code{cor()} like \code{iivKa + iivCL~cor(2, -0.5, 3)}.

Values may be fixed (and therefore not estimated) using either the name \code{fixed} at the end of the assignment or by calling \code{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): \code{tvCL <- fixed(1), tvCL <- fixed(0, 1)}, \code{tvCL <- fixed(0, 1, 2), tvCL <- c(0, fixed(1), 2)}, or \code{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: \code{iivKa~fixed(1), iivKa + iivCL~fixed(1, 2, 3)}, or \code{iivKa + iivCL~c(1, 2, 3, fixed)}. Anywhere that \code{fixed} is used, \code{FIX}, \code{FIXED}, or \code{fix} may be used equivalently.

For any value, standard mathematical operators or functions may be used to define the value. For example, \code{log(2)} and \code{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 \code{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 \code{tvCL} has a
label of "Typical Value of Clearance (L/hr)" is tvCL <- 1; label("Typical Value of Clearance (L/hr)").

rxode2/nlmixr2 will attempt to determine some back-transformations for the user. For example, CL <- exp(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()).

Value

ini block

Author(s)

Matthew Fidler

See Also

Other Initial conditions: zeroRe()

Examples

# Set the ini() block in a model
one.compartment <- function() {
  ini(
    tka <- log(1.57); label("Ka")
    tcl <- log(2.72); label("Cl")
    tv <- log(31.5); label("V")
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  )
}

# Use piping to update initial conditions
one.compartment %>% ini(tka <- log(2))
one.compartment %>% ini(tka <- label("Absorption rate, Ka (1/hr)"))
# Move the tka parameter to be just below the tv parameter (affects parameter # summary table, only)
one.compartment %>% ini(tka <- label("Absorption rate, Ka (1/hr)"), append = "tv")
# When programming with rxode2/nlmixr2, it may be easier to pass strings in
# to modify the ini
ini<- one.compartment %>% ini("tka <- log(2)"")

\scrollbar{ini<-}{Assign the ini block in the rxode2 related object}

**Description**

Assign the ini block in the rxode2 related object

**Usage**

ini(x, envir = environment(x)) <- value

**Arguments**

- **x**: rxode2 related object
- **envir**: Environment where assignment occurs
- **value**: Value of the object

**Value**

rxode2 related object

**Author(s)**

Matthew L. Fidler

\scrollbar{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**

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)
et <- et(seq(1e-4, 1-1e-4, length.out=21))
et$shape1 <- 0.5
et$shape2 <- 1.5
model <- function() {
  model({
    fx <- llikBeta(time, shape1, shape2)
    dShape1 <- llikBetaDshape1(time, shape1, shape2)
    dShape2 <- llikBetaDshape2(time, shape1, shape2)
  })
}
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

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

**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**

```R
llikBinom(46:54, 100, 0.5)
llikBinom(46:54, 100, 0.5, TRUE)
# In rxode2 you can use:
et <- et(46:54)
et$size <- 100
et$prob <- 0.5

model <- function() {
  model({
    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

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

Arguments

- **x**: Observation
- **location, scale**: location and scale parameters.
- **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

Author(s)

Matthew L. Fidler

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 <- function() {
  model({
    fx <- llikCauchy(time, location, scale)
    dLocation <- llikCauchyDlocation(time, location, scale)
  })
}
```
likelihood and derivatives for chi-squared distribution

Description

log likelihood and derivatives for chi-squared distribution

Usage

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 rxode2() 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().

Value

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

Author(s)

Matthew L. Fidler

Examples

llikChisq(1, df = 1:3, full=TRUE)
llikChisq(1, df = 6:9)
et <- et(1:3)
et$x <- 1
model <- function() {
  model({
    fx <- llikChisq(x, time)
    dDf <- llikChisqDdf(x, time)
  })
}

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().

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

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 <- function() {
  model({
    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

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 <- function()
  model({
    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 \( f_x \) for the log pdf value of with \( d\text{Prob} \) that has the derivatives with respect to the \( \text{prob} \) 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 <- function() {
  model({
    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 \( f_x \) and derivatives
Details

In an rxd2() 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

llikGeom(1:10, 0.2)

et <- et(1:10)
et$prob <- 0.2

model <- function() {
  model({
    fx <- llikGeom(time, prob)
    dProb <- llikGeomDprob(time, prob)
  })
}

rxSolve(model, et)

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

llikNbinom(x, size, prob, full = FALSE)
\textbf{llikNbinom}

\textbf{Arguments}

\begin{itemize}
\item \textbf{x} \quad \text{Number of successes}
\item \textbf{size} \quad \text{Size of trial}
\item \textbf{prob} \quad \text{Probability of success}
\item \textbf{full} \quad \text{Add the data frame showing x, mean, sd as well as the fx and derivatives}
\end{itemize}

\textbf{Details}

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

\textbf{Value}

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

\textbf{Author(s)}

Matthew L. Fidler

\textbf{Examples}

\begin{verbatim}
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 <- function() {
    model({
        fx <- llikNbinom(time, size, prob)
        dProb <- llikNbinomDprob(time, size, prob)
    })
}
rxSolve(model, et)
\end{verbatim}
llikNbinomMu  

*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
llikNbinomMu(x, size, mu, full = FALSE)
```

**Arguments**

- `x`: Number of successes
- `size`: Size of trial
- `mu`: `mu` 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 `mu` 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)
```

```r
et <- et(46:54)
et$size <- 100
et$mu <- 40

model <- function() {
  fx <- llikNbinomMu(time, size, mu)
  dProb <- llikNbinomMuDmu(time, size, mu)
}
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 <- function(){
  model(
    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

```r
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 <- function() {
  model({
    fx <- llikPois(time, lambda)
    dLambda <- llikPoisDlambda(time, lambda)
  })
}

rxSolve(model, et)

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 llikTDdf(), llikTDmean() and llikTDsd().

Value

data frame with fx for the log pdf value of with dF df 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 <- function() {
  model({
    fx <- llikT(time, nu, mean, sd)
    dDf <- llikTDdf(time, nu, mean, sd)
    dMean <- llikTDmean(time, nu, mean, sd)
    dSd <- llikTDsd(time, nu, mean, sd)
  })
}

rxSolve(model, et)
```

llikUnif

log likelihood and derivatives for Unif distribution

Description
log likelihood and derivatives for Unif distribution

Usage

```r
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

Examples

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

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

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

rxSolve(model, et)
```

llikWeibull

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

Examples

```r
llikWeibull(1, 1, 10)
# rxode2 can use this too:

et <- et(seq(0.001, 1, length.out=10))
et$shape <- 1
et$scale <- 10

model <- function() {
  model({
    fx <- llikWeibull(time, shape, scale)
    dShape <- llikWeibullDshape(time, shape, scale)
    dScale <- llikWeibullDscale(time, shape, scale)
  })
}

rxSolve(model, et)
```

logit and inverse logit (expit) functions

Description

logit and inverse logit (expit) functions
Usage

\texttt{logit(x, low = 0, high = 1)}
\texttt{expit(alpha, low = 0, high = 1)}
\texttt{logitNormInfo(mean = 0, sd = 1, low = 0, high = 1, abs.tol = 1e-06, ...)}
\texttt{probitNormInfo(mean = 0, sd = 1, low = 0, high = 1, abs.tol = 1e-06, ...)}

Arguments

\begin{itemize}
  \item \texttt{x} \quad \text{Input value(s) in range [low,high] to translate -Inf to Inf}
  \item \texttt{low} \quad \text{Lowest value in the range}
  \item \texttt{high} \quad \text{Highest value in the range}
  \item \texttt{alpha} \quad \text{Infinite value(s) to translate to range of [low, high]}
  \item \texttt{mean} \quad \text{logit-scale mean}
  \item \texttt{sd} \quad \text{logit-scale standard deviation}
  \item \texttt{abs.tol} \quad \text{absolute accuracy requested.}
  \item \texttt{...} \quad \text{other parameters passed to \texttt{integrate()}}
\end{itemize}

Details

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

Value

values from logit and expit

Examples

\begin{itemize}
  \item \texttt{logit(0.25)}
  \item \texttt{expit(-1.09)}
  \item \texttt{logitNormInfo(logit(0.25), sd = 0.1)}
  \item \texttt{logitNormInfo(logit(1, 0, 10), sd = 1, low = 0, high = 10)}
\end{itemize}
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)
meanProbs

Calculate expected confidence bands or prediction interval with normal or t sampling distribution

Description

The generic function meanProbs produces expected confidence bands under either the t distribution or the normal sampling distribution. This uses qnorm() or qt() with the mean and standard deviation.

Usage

meanProbs(x, ...)

## Default S3 method:
meanProbs(
  x,
  probs = seq(0, 1, 0.25),
  na.rm = FALSE,
  names = TRUE,
  useT = TRUE,
  onlyProbs = TRUE,
  pred = FALSE,
  n = 0L,
  ...
)

Arguments

x numeric vector whose mean and probability based confidence values are wanted, NA and NaN values are not allowed in numeric vectors unless `na.rm` is `TRUE`.

... Arguments passed to default method, allows many different methods to be applied.

probs numeric vector of probabilities with values in the interval from 0 to 1.

na.rm logical; if true, any NA and NaN’s are removed from x before the quantiles are computed.

names logical; if true, the result has a names attribute.

useT logical; if true, use the t-distribution to calculate the confidence-based estimates. If false use the normal distribution to calculate the confidence based estimates.

onlyProbs logical; if true, only return the probability based confidence interval estimates, otherwise return

pred logical; if true use the prediction interval instead of the confidence interval

n integer/integerish; this is the n used to calculate the prediction or confidence interval. When n=0 (default) use the number of non-NA observations.
Details

For a single probability, p, it uses either:

mean + qt(p, df=n)*sd/sqrt(n)

or

mean + qnorm(p)*sd/sqrt(n)

The smallest observation corresponds to a probability of 0 and the largest to a probability of 1 and the mean corresponds to 0.5.

The mean and standard deviation of the sample is calculated based on Welford’s method for a single pass.

This is meant to perform in the same way as quantile() so it can be a drop in replacement for code using quantile() but using distributional assumptions.

Value

By default the return has the probabilities as names (if named) with the points where the expected distribution are located given the sampling mean and standard deviation. If onlyProbs=FALSE then it would prepend mean, variance, standard deviation, minimum, maximum and number of non-NA observations.

Author(s)

Matthew L. Fidler

Examples

quantile(x<- rnorm(1001))
meanProbs(x)

# Can get some extra statistics if you request onlyProbs=FALSE
meanProbs(x, onlyProbs=FALSE)

x[2] <- NA_real_

meanProbs(x, onlyProbs=FALSE)

quantile(x<- rnorm(42))

meanProbs(x)

meanProbs(x, useT=FALSE)
Description

Model block for rxode2/nlmixr models

Usage

```r
## S3 method for class 'function'
model(
  x,
  ..., 
  append = NULL, 
  auto = getOption("rxode2.autoVarPiping", TRUE), 
  cov = NULL, 
  envir = parent.frame()
)

## S3 method for class 'rxUi'
model(
  x,
  ..., 
  append = NULL, 
  auto = getOption("rxode2.autoVarPiping", TRUE), 
  cov = NULL, 
  envir = parent.frame()
)

## S3 method for class 'rxode2'
model(
  x,
  ..., 
  append = NULL, 
  auto = getOption("rxode2.autoVarPiping", TRUE), 
  cov = NULL, 
  envir = parent.frame()
)

## S3 method for class 'rxModelVars'
model(
  x,
  ..., 
  append = NULL, 
  auto = getOption("rxode2.autoVarPiping", TRUE), 
  cov = NULL, 
  envir = parent.frame()
)
```
model.function

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

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

Arguments

x      model expression
...
append This is a boolean to determine if the lines are appended in piping. The possible values for this is:
  • TRUE which is when the lines are appended to the model instead of replaced
  • FALSE when the lines are replaced in the model (default)
  • NA is when the lines are pre-pended to the model instead of replaced
  • lhs expression, which will append the lines after the last observed line of the expression lhs
auto   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).
cov    is a character vector of variables that should be assumed to be covariates. This will override automatic promotion to a population parameter estimate (or an eta)
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.

Value

Model block with ini information included. ini must be called before model block

Author(s)

Matthew Fidler
model<-  

Assign the model block in the rxode2 related object

Description

Assign the model block in the rxode2 related object

Usage

model(x, envir = environment(x)) <- value

Arguments

x  rxode2 related object
envir  Environment where assignment occurs
value  Value of the object

Value

rxode2 related object

Author(s)

Matthew L. Fidler

modelExtract

Extract model lines from a rxui model

Description

Extract model lines from a rxui model

Usage

modelExtract(
  x,
  ...,  
  expression = FALSE,
  endpoint = FALSE,
  lines = FALSE,
  envir = parent.frame()
)

## S3 method for class `function`
modelExtract(
modelExtract

```
x, ...
expression = FALSE,
endpoint = FALSE,
lines = FALSE,
envir = parent.frame()
)

## S3 method for class 'rxUi'
modelExtract(
x,
...
expression = FALSE,
endpoint = FALSE,
lines = FALSE,
envir = parent.frame()
)

## S3 method for class 'rxode2'
modelExtract(
x,
...
expression = FALSE,
endpoint = FALSE,
lines = FALSE,
envir = parent.frame()
)

## S3 method for class 'rxModelVars'
modelExtract(
x,
...
expression = FALSE,
endpoint = FALSE,
lines = FALSE,
envir = parent.frame()
)

## Default S3 method:
modelExtract(
x,
...
expression = FALSE,
endpoint = FALSE,
lines = FALSE,
envir = parent.frame()
)
```
Arguments

- **x**: model to extract lines from
- **...**: variables to extract. When it is missing, it will extract the entire model (conditioned on the endpoint option below)
- **expression**: return expressions (if TRUE) or strings (if FALSE)
- **endpoint**: include endpoint. This can be:
  - NA – Missing means include both the endpoint and non-endpoint lines
  - TRUE – Only include endpoint lines
  - FALSE – Only include non-endpoint lines
- **lines**: is a boolean. When TRUE this will add the lines as an attribute to the output value ie attr(, "lines")
- **envir**: Environment for evaluating variables

Value

expressions or strings of extracted lines. Note if there is a duplicated lhs expression in the line, it will return both lines

Author(s)

Matthew L. Fidler

Examples

```r
one.compartment <- function() {
  ini({
    tka <- 0.45 # Log Ka
    tcl <- 1 # Log Cl
    tv <- 3.45   # Log V
    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)
    d/dt(depot) <- -ka * depot
    d/dt(center) <- ka * depot - cl / v * center
    cp <- center / v
    cp ~ add(add.sd)
  })
}

f <- one.compartment()

modelExtract(f, cp)
```
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.
- "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.

Value

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

Plot rxode2 objects

Description

Plot rxode2 objects

Usage

```r
## 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

See Also

Other rxode2 plotting: `rxTheme()`

probit  

probit and inverse probit functions

Description

probit and inverse probit functions
Usage

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

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

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

# 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**

```r
rxAppendModel(..., common = TRUE)
```

**Arguments**

- `...`: models to append together
- `common`: boolean that determines if you need a common value to bind

**Value**

New model with both models appended together

**Author(s)**

Matthew L. Fidler

**Examples**

```r
ocmt <- 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
  })
}
```
\[
\begin{align*}
  d/dt(\text{depot}) &\leftarrow -ka \times \text{depot} \\
  d/dt(\text{center}) &\leftarrow ka \times \text{depot} - cl / v \times \text{center} \\
  cp &\leftarrow \text{center} / v \\
  cp &\sim \text{add}(\text{add.sd}) \\
\end{align*}
\]

\[
\begin{align*}
  \text{idr} &\leftarrow \text{function}() \\
  \text{ini}(& \\
    \text{tkin} &\leftarrow \log(1) \\
    \text{tkout} &\leftarrow \log(1) \\
    \text{tic50} &\leftarrow \log(10) \\
    \gamma &\leftarrow \text{fix}(1) \\
    \text{idr.sd} &\leftarrow 1 \\
  \\
  \text{model}(& \\
    \text{kin} &\leftarrow \exp(\text{tkin}) \\
    \text{kout} &\leftarrow \exp(\text{tkout}) \\
    \text{ic50} &\leftarrow \exp(\text{tic50}) \\
    d/dt(\text{eff}) &\leftarrow \text{kin} - \text{kout} \times (1 - \text{ceff}^\gamma / (\text{ic50}^\gamma + \text{ceff}^\gamma)) \\
    \text{eff} &\sim \text{add}(\text{idr.sd}) \\
  \\
  \text{rxAppendModel}(\text{ocmt} \%\% \text{model}(\text{ceff}=cp, \text{append=}\text{TRUE}), \text{idr})
\end{align*}
\]

---

**rxAssignControlValue**  
*Assign Control Variable*

**Description**

Assign Control Variable

**Usage**

\[
\text{rxAssignControlValue}(\text{ui}, \text{option}, \text{value})
\]

**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
**rxAssignPtr**

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

**Description**
Assign pointer based on model variables

**Usage**
```
rxAssignPtr(object = NULL)
```

**Arguments**
- `object` rxode2 family of objects

**Value**
nothing, called for side effects

---

**rxbeta**

**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/](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**
```
rxbeta(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

`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

beta random deviates

Examples

```r
## Use threefry engine
rxbeta(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 <- function() {
  model({
    a <- rxbeta(2, 2)
  })
}
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

<table>
<thead>
<tr>
<th>rxbinom</th>
<th>Simulate Binomial variable from threefry generator</th>
</tr>
</thead>
</table>

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

\texttt{rxbinom(size, prob, n = 1L, ncores = 1L)}

Arguments

\begin{description}
\item[size] number of trials (zero or more).
\item[prob] probability of success on each trial.
\item[n] number of observations. If \texttt{length(n) > 1}, the length is taken to be the number required.
\item[ncores] Number of cores for the simulation
\end{description}

\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.

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.

Value

binomial random deviates

Examples

\begin{verbatim}
## Use threefry engine

rxbinom(10, 0.9, n = 10) \# with \texttt{rxbinom} you have to explicitly state \texttt{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 <- function() {
  model({
    a <- rxbinom(1, 0.5)
  })
}
\end{verbatim}
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
rxcauchy(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

**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

## Use threefry engine

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

```
rxcauchy(3)
```

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

```r
rx <- function() {
  model({
    a <- rxcauchy(2)
  })
}
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

---

*rxchisq*  
*Simulate chi-squared 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
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

*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

chi squared random deviates

Examples

## 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 <- function() {
  model({
    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**

```
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, ...)
```

```
## 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

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

Author(s)
Matthew L. Fidler

See Also
rxode2()

Description
This updates the tolerances based on the sensitivity equations

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 $atol, $rtol, $ssAtol $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

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

rxCreateCache()

Value

nothing

Author(s)

Matthew Fidler

rxD

Add to rxode2’s derivative tables

Description

Add to rxode2’s derivative tables

Usage

rxD(name, derivatives)
**rxDelete**

Delete the DLL for the model

**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.

---

**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**

```r
## 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

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

**Author(s)**

Matthew L. Fidler

**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 [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
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

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

exponential random deviates

Examples

## Use threefry engine

```r
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
```

```r
rxexp(1)
```

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

```r
rx <- function() {
  model({
    a <- rxexp(2)
  })
}
```

```r
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```
### 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
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 threelfry 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 <- function() {
  model({
    a <- rxf(2, 2)
  })
}
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

### Description

Add/Create C functions for use in rxode2

### Usage

`rxFun(name, args, cCode)`

`rxRmFun(name)`

### Arguments

- **name**: This can either give the name of the user function or be a simple R function that you wish to convert to C. If you have rxode2 convert the R function to C, the name of the function will match the function name provided and the number of arguments will match the R function provided. Hence, if you are providing an R function for conversion to C, the rest of the arguments are implied.

- **args**: This gives the arguments of the user function

- **cCode**: This is the C-code for the new function
Examples

## Right now rxode2 is not aware of the function fun
## 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 special terms. Therefore
## f(x) would not work since f is an alias for bioavailability.

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*", 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")

# you can also use R functions directly in rxode2

gg <- function(x, y) {
    x + y
f <- rxode2({
  z = gg(x, y)
})

e <- et(1:10) |> as.data.frame()
e$x <- 1:10
e$y <- 21:30

rxSolve(f, e)

# Note that since it touches R, it can only run single-threaded.
# There are also requirements for the function:
# 1. It accepts one value per argument (numeric)
# 2. It returns one numeric value

# If it is a simple function (like gg) you can also convert it to C
# using rxFun and load it into rxode2

rxFun(gg)

rxSolve(f, e)

# to stop the recompile simply reassign the function
f <- rxode2(f)

rxSolve(f, e)

rxRmFun("gg")

rm(gg)
rm(f)

# You can also automatically convert a R function to R code (and
# calculate first derivatives)

fun <- function(a, b, c) {
  a^2+b*a+c
}

rxFun(fun)

# You can see the R code if you want with rxC

message(rxC("fun"))

# you can also remove both the function and the
# derivatives with rxRmFun("fun")
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

\[ \text{rxgamma}(\text{shape}, \text{rate} = 1, n = 1L, \text{ncores} = 1L) \]

Arguments

- `shape`: The shape of the gamma random variable
- `rate`: an alternative way to specify the scale.
- `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

gamma random deviates
Examples

```r
## Use threefry engine
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)

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

---

### `rxgeom`

_Simulate geometric 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**

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

**Arguments**

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

`rxnorm` simulates using the threefry sitmo generator.
\textit{rxgeometric} 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 \textit{rxnorm}. It is no longer supported in \textit{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 \textit{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 \textit{rxode2} engine with \textit{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 threecry 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 <- function() {
  model({
    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**

```r
rxGetControl(ui, option, default)
```

**Arguments**

- **ui**: rxode2 ui object
- **option**: Option to get
- **default**: Default value

**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
**rxHtml**

*Format rxSolve and related objects as html.*

---

**Description**

Format rxSolve and related objects as html.

**Usage**

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

## S3 method for class 'rxSolve'

```r
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 an 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**

```r
rxInv(matrix)
```

**Arguments**

```r
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()`, `rxInit()`, `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

**rxnbinom**

*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**

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

```
rxnbinomMu(size, mu, 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. $0 < \text{prob} \leq 1$.

- **n**
  
  number of observations. If $\text{length}(n) > 1$, 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

mu alternative parametrization via mean: see ‘Details’.

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 <- function() {
  model({
    a <- rxnbinom(10, 0.5)
  })
}

et <- et(1, id = 1:100)
s <- rxSolve(rx, et)

rx <- function() {
  model(

```
\begin{verbatim}
a <- rxnbinomMu(10, 40)
}

s <- rxSolve(rx, et)
\end{verbatim}

---

**rxNorm**  
*Get the normalized model*

**Description**  
This get the syntax preferred model for processing

**Usage**  
\[ rxNorm(obj, condition = \text{NULL}, \text{removeInis}, \text{removeJac}, \text{removeSens}) \]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>rxode2 family of objects</td>
</tr>
<tr>
<td>condition</td>
<td>Character string of a logical condition to use for subsetting the normalized model. When missing, and a condition is not set via <code>rxCondition</code>, return the whole code with all the conditional settings intact. When a condition is set with <code>rxCondition</code>, use that condition.</td>
</tr>
<tr>
<td>removeInis</td>
<td>A boolean indicating if parameter initialization will be removed from the model</td>
</tr>
<tr>
<td>removeJac</td>
<td>A boolean indicating if the Jacobians will be removed.</td>
</tr>
<tr>
<td>removeSens</td>
<td>A boolean indicating if the sensitivities will be removed.</td>
</tr>
</tbody>
</table>

**Value**  
Normalized Normal syntax (no comments)

**Author(s)**  
Matthew L. Fidler
Simulate random normal variable from threefry generator

**Description**

Simulate random normal variable from threefry generator

**Usage**

```r
rxnormV(mean = 0, sd = 1, n = 1L, ncores = 1L)
```

```r
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**

```r
## 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 <- function() {
  model({
    a <- rxnorm()
  })
}

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

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

Usage

\[
s \leftarrow \text{rxSolve}(rx, et)
\]

**rxode2**

Create an ODE-based model specification

### Description

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

### Usage

\[
\text{rxode2}(\text{model}, \text{modName} = \text{basename(wd)}, \text{wd} = \text{getwd}(), \text{filename} = \text{NULL}, \text{extraC} = \text{NULL}, \text{debug} = \text{FALSE}, \text{calcJac} = \text{NULL}, \text{calcSens} = \text{NULL}, \text{collapseModel} = \text{FALSE}, \text{package} = \text{NULL}, \ldots, \text{linCmtSens} = c(\text{"linCmtA"}, \text{"linCmtB"}, \text{"linCmtC"}), \text{indLin} = \text{FALSE}, \text{verbose} = \text{FALSE}, \text{fullPrint} = \text{getOption}(\text{"rxode2.fullPrint"}, \text{FALSE}), \text{envir} = \text{parent.frame}())
\]

\[
\text{RxODE}(\text{model}, \text{modName} = \text{basename(wd)}, \text{wd} = \text{getwd}(), \text{filename} = \text{NULL}, \text{extraC} = \text{NULL}, \text{debug} = \text{FALSE}, \text{calcJac} = \text{NULL}, \text{calcSens} = \text{NULL}, \text{collapseModel} = \text{FALSE}, \text{package} = \text{NULL}, \ldots, \text{linCmtSens} = c(\text{"linCmtA"}, \text{"linCmtB"}, \text{"linCmtC"}), \text{indLin} = \text{FALSE}, \ldots)
\]
verbose = FALSE,
fullPrint = getOption("rxode2.fullPrint", FALSE),
envir = parent.frame()
)

rxode(
    model,
    modName = basename(wd),
    wd = getwd(),
    filename = NULL,
    extraC = NULL,
    debug = FALSE,
    calcJac = NULL,
    calcSens = NULL,
    collapseModel = FALSE,
    package = NULL,
    ...
    linCntSens = c("linCmtA", "linCmtB", "linCmtC"),
    indLin = FALSE,
    verbose = FALSE,
    fullPrint = getOption("rxode2.fullPrint", FALSE),
    envir = parent.frame()
)

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.

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

wd 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 Jacobian 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.

linCntSens

The method to calculate the linCnt() 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 \( \frac{d}{dt} \)

evr

is the environment to look for R user functions (defaults to parent environment)

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 four places:

- Inside a `rxode2({})` block statements:

```r
library(rxode2)
mod <- rxode2(
  # simple assignment
  C2 <- centr/V2

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

## using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
## In file included from /usr/share/R/include/R.h:71,
## from /home/matt/R/x86_64-pc-linux-gnu-library/4.3/rxode2parse/include/rxode2parse.h:33,
## from /home/matt/R/x86_64-pc-linux-gnu-library/4.3/rxode2parse/include/rxode2_model_shared.h:3,
## from rx_80ab028288eddd16733200578a7fac4b_.c:117:
## /usr/share/R/include/R_ext/Complex.h:80:6: warning: ISO C99 doesn't support unnamed structs/unions
## 80 |    
## | ^
Inside a `rxode2("")` string statement:

```r
mod <- rxode2("#
# simple assignment
C2 <- centr/V2

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

## using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
## In file included from /usr/share/R/include/R.h:71,
## from /home/matt/R/x86_64-pc-linux-gnu-library/4.3/rxode2parse/include/rxode2parse.h:33,
## from /home/matt/R/x86_64-pc-linux-gnu-library/4.3/rxode2parse/include/rxode2_model_shared.h:9,
## from rx_16fc28e9f2ad308e65c8fb7a9b53fcd_1_.c:117:
## /usr/share/R/include/R_ext/Complex.h:80:6: warning: ISO C99 doesn’t support unnamed structs/unions
## 80 | |
## | ^

In a file name to be loaded by `rxode2`:

```r
writeLines("#
# simple assignment
C2 <- centr/V2

# time-derivative assignment
d/dt(centr) <- F*KA*depot - CL*C2 - Q*C2 + Q*C3;",
"modelFile.rxode2")
```

```r
mod <- rxode2(filename="modelFile.rxode2")
```

```r
unlink("modelFile.rxode2")
```

In a model function which can be parsed by `rxode2`:

```r
mod <- function() {
model({

  # simple assignment
  C2 <- centr/V2

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

```r
mod <- rxode2(mod) # or simply mod() if the model is at the end of the function
```

# These model functions often have residual components and initial
# (`ini({})`) conditions attached as well. For example the
# theophylline model can be written as:
one.compartment <- function() {
  ini(
    tka <- 0.45  # Log Ka
    tcl <- 1  # Log Cl
    tv <- 3.45   # Log V
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  )
}

# after parsing the model
mod <- one.compartment()

For the block statement, character string or text file an internal rkode2 compilation manager translates the ODE system into C, compiles it and loads it into the R session. The call to rkode2 produces an object of class rkode2 which consists of a list-like structure (environment) with various member functions.

For the last type of model (a model function), a call to rkode2 creates a parsed rkode2 ui that can be translated to the rkode2 compilation model.

mod$simulationModel

## using C compiler: 'gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0'
## In file included from /usr/share/R/include/R.h:71,
## from /home/matt/R/x86_64-pc-linux-gnu-library/4.3/rkode2parse/include/rkode2parse.h:33,
## from /home/matt/src/rkode2/inst/include/rkode2.h:9,
## from /home/matt/R/x86_64-pc-linux-gnu-library/4.3/rkode2parse/include/rkode2_model_shared.h:3,
## from rx_e3306ec84cb8151ac5b0c27ef3dbbe7_.c:117:
## /usr/share/R/include/R_ext/Complex.h:80:6: warning: ISO C99 doesn't support unnamed structs/unions [-Wpedantic]
## 80 | };
## | ^
## 
## rkode2 2.1.2 model named rx_e3306ec84cb8151ac5b0c27ef3dbbe7 model (ready).
## $x$state: depot, center
## $x$stateExtra: cp
## $x$params: tka, tcl, tv, add.sd, eta.ka, eta.cl, eta.v, rxerr.cp
## $x$lhs: ka, cl, v, cp, ipredSim, sim
This is the same type of function required for `nlmixr2` estimation and can be extended and modified by model piping. For this reason will be focused on in the documentation.

This basic 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. There may be some advantage to obtaining the solution or specifying the Jacobian for very stiff ODE systems. However, for the few stiff systems we tried with LSODA, this actually slightly slowed down the solving.

Note that assignment can be done by =, <- or ~.
When assigning with the \( \sim \) operator, the **simple assignments** and **time-derivative** assignments will not be output. Note that with the \texttt{rxode2} model functions assignment with \( \sim \) can also be overloaded with a residual distribution specification.

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 \texttt{nlmixr} models); These are specified by \texttt{cmt(\textit{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 \texttt{param(\textit{par1}, \textit{par2}, \ldots)}

An example model is shown below:

```r
# 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., \texttt{fabs}, \texttt{exp}, \texttt{log}, \texttt{sin}, \texttt{abs}).

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

The \texttt{rxode2} syntax is case-sensitive, i.e., ABC is different than abc, Abc, 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, \texttt{t} (time), \texttt{tlast} (last time point), and \texttt{podo} (oral dose, in the undocumented case of absorption transit models).
- Special constants like \texttt{pi} or R’s predefined constants.
- Model parameters (e.g., ka rate of absorption, CL clearance, etc.)
- Others, as created by assignments as part of the model specification; these are referred as \textit{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:
• cmt
• dvid
• addl
• ss
• rate
• id

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

Sometimes rxode2 generates variables that are fed back to rxode2. Similarly, nlmixr2 generates some variables that are used in nlmixr estimation and simulation. These variables start with the either the rx or nlmixr prefixes. To avoid any problems, it is suggested to not use these variables starting with either the rx or nlmixr prefixes.

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

\[
cov1 = \text{covm*(sexf == "female") + covm*(sexf != "female")}
\]

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, \text{as.numeric} or \text{as.integer} for these logical statements is not only not needed, but will cause an syntax error if you try to use the function.

Supported functions:
All the supported functions in rxode2 can be seen with the \text{rxSupportedFuns()}. A brief description of the built-in functions are in the following table:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma(x)</td>
<td>The Gamma function</td>
</tr>
<tr>
<td>lgamma(x)</td>
<td>Natural logarithm of absolute value of gamma function</td>
</tr>
<tr>
<td>digamma(x)</td>
<td>First derivative of lgamma</td>
</tr>
<tr>
<td>trigamma(x)</td>
<td>Second derivative of lgamma</td>
</tr>
<tr>
<td>tetragamma(x)</td>
<td>Third derivative of lgamma</td>
</tr>
<tr>
<td>pentagamma(x)</td>
<td>Fourth derivative of lgamma</td>
</tr>
<tr>
<td>psigamma(x, deriv)</td>
<td>n-th derivative of Psi, the digamma function, which is the derivative of lgammafn. In other words, digamma(x) is the same as psigamma(x, 0), trigamma(x) == psigamma(x, 1), etc.</td>
</tr>
<tr>
<td>cospi(x)</td>
<td>(\cos(\pi x))</td>
</tr>
<tr>
<td>sinpi(x)</td>
<td>(\sin(\pi x))</td>
</tr>
<tr>
<td>tanpi(x)</td>
<td>(\tan(\pi x))</td>
</tr>
<tr>
<td>beta(a, b)</td>
<td>Beta function</td>
</tr>
<tr>
<td>lbeta(a, b)</td>
<td>log Beta function</td>
</tr>
<tr>
<td>bessel_i(x, nu, expo)</td>
<td>Bessel function type I with index nu</td>
</tr>
<tr>
<td>bessel_j(x, nu)</td>
<td>Bessel function type J with index nu</td>
</tr>
<tr>
<td>bessel_k(x, ku, expo)</td>
<td>Bessel function type K with index nu</td>
</tr>
<tr>
<td>bessel_y(x, nu)</td>
<td>Bessel function type Y with index nu</td>
</tr>
</tbody>
</table>
R_pow(x, y) \quad x^y
R_pow_di(x, I) \quad x^y \quad y \text{ is an integer}
log1pmx \quad \log(1+x) - x
log1pexp \quad \log(1+\exp(x))
expm1(x) \quad \exp(x) - 1
lgamma1p(x) \quad \log(\text{gamma}(x+1))
sign(x) \quad \text{Compute the signum function where sign}(x) \text{ is } 1, 0, -1
fsign(x, y) \quad \abs(x) \times \text{sign}(y)
fprec(x, digits) \quad x \text{ rounded to digits (after the decimal point, used by signif())}
fround(x, digits) \quad \text{Round, used by R's round()}
frunc(x) \quad \text{Truncated towards zero}
abs(x) \quad \text{absolute value of } x
sin(x) \quad \text{sine of } x
cos(x) \quad \text{cos of } x
tan(x) \quad \text{tan of } x
factorial(x) \quad \text{factorial of } x
lfactorial(x) \quad \log(\text{factorial}(x))
log10(x) \quad \log \text{ base } 10
log2(x) \quad \log \text{ base } 2
pnorm(x) \quad \text{Normal CDF of } x
qnorm(x) \quad \text{Normal pdf of } x
probit(x, low=0, hi=1) \quad \text{Probit (normal pdf) of } x \text{ transforming into a range}
probitInv(q, low=0, hi=1) \quad \text{Inverse probit of } x \text{ transforming into a range}
acos(x) \quad \text{Inverse cosine}
asin(x) \quad \text{Inverse sine}
atan(x) \quad \text{Inverse tangent}
atan2(a, b) \quad \text{Four quadrant inverse tangent}
sinh(x) \quad \text{Hyperbolic sine}
cosh(x) \quad \text{Hyperbolic cosine}
tanh(x) \quad \text{Hyperbolic tangent}
floor(x) \quad \text{Downward rounding}
ceil(x) \quad \text{Upward rounding}
logit(x, low=0, hi=1) \quad \text{Logit transformation of } x \text{ transforming into a range}
expit(x, low=0, hi=1) \quad \text{expit transformation in range}
gammaq(a, z) \quad \text{Normalized incomplete gamma from boost}
gammaqInv(a, q) \quad \text{Normalized incomplete gamma inverse from boost}
ifelse(cond, trueValue, falseValue) \quad \text{if else function}
gammmap(a, z) \quad \text{Normalized lower incomplete gamma from boost}
gammmapInv(a, p) \quad \text{Inverse of Normalized lower incomplete gamma from boost}
gammmapInvb(x, p) \quad \text{Inverse of Normalized lower incomplete gamma from boost}
rxnorm(x) \quad \text{Generate one deviate of from a normal distribution for each observation scale}
rxnormV(x) \quad \text{Generate one deviate from low discrepancy normal for each observation}
rxcauchy \quad \text{Generate one deviate from the cauchy distribution for each observation}
rxchisq \quad \text{Generate one deviate from the chisq distribution for each observation}
rxexp \quad \text{Generate one deviate from the exponential distribution for each observation}
rxf \quad \text{Generate one deviate from low discrepancy normal for each observation}
rxgamma \quad \text{Generate one deviate from the gamma distribution for each observation}
rxbeta \quad \text{Generate one deviate from the beta distribution for each observation}
rxgeom Generate one deviate from the geometric distribution for each observation
rxpois Generate one deviate from the poission distribution for each observation
rxt Generate one deviate from the t distribution for each observation
tad() or tad(x) Time after dose (tad()) or time after dose for a compartment tad(cmt)
tafd() or tafd(x) Time after first dose (tafd()) or time after first dose for a compartment tafd(cmt)
dosenum() Dose Number
tlast() or tlast(cmt) Time of Last dose; This takes into consideration any lag time, so if there is a dose at time 3 and a lag of 1, the time of last dose would be 4.
tfirst() or tfirst(cmt) Time since first dose or time since first dose of a compartment
prod(...) product of terms; This uses PreciseSums so the product will not have as much floating point errors
sum(...) sum of terms; This uses PreciseSums so the product will not have as much floating point errors
max(...) maximum of a group of numbers
min(...) Min of a group of numbers
lag(parameter, number=1) Get the lag of an input parameter; You can specify a number of lagged observations
lead(parameter, number=2) Get the lead of an input parameter; You can specify a number of lead observation
diff(par, number=1) Get the difference between the current parameter and the last parameter; Can change the number
first(par) Get the first value of an input parameter
last(par) Get the last value of an input parameter
transit() The transit compartment pseudo function
is.na() Determine if a value is NA
is.nan() Determine if a value is NaN
is.infinite() Check to see if the value is infinite
rinorm(x) Generate one deviate of from a normal distribution for each individual
rinormV(x) Generate one deviate from low discrepancy normal for each individual
ricauchy Generate one deviate from the cauchy distribution for each individual
richisq Generate one deviate from the chi square distribution for each individual
riexp Generate one deviate from the exponential distribution for each individual
rif Generate one deviate from low discrepancy normal for each individual
rigamma Generate one deviate from the gamma distribution for each individual
ribeta Generate one deviate from the beta distribution for each individual
rigeom Generate one deviate from the geometric distribution for each individual
ropois Generate one deviate from the poission distribution for each individual
rit Generate one deviate from the t distribution for each individual
simeps Simulate EPS from possibly truncated sigma matrix. Will take sigma matrix from the current study
simeta Simulate ETA from possibly truncated omega matrix. Will take the omega matrix from the current study

Note that lag(cmt) = is equivalent to alag(cmt) = and not the same as = lag(wt)

**Reserved keywords:**

There are a few reserved keywords in a rxode2 model. They are in the following table:

<table>
<thead>
<tr>
<th>Reserved Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>solver time</td>
</tr>
<tr>
<td>podo</td>
<td>In Transit compartment models, last dose amount</td>
</tr>
<tr>
<td>tlast</td>
<td>Time of Last dose</td>
</tr>
<tr>
<td>M_E</td>
<td>Exp(1)</td>
</tr>
<tr>
<td>M_LOG2E</td>
<td>log2(e)</td>
</tr>
<tr>
<td>M_LOG10E</td>
<td>log10(e)</td>
</tr>
<tr>
<td>M_LN2</td>
<td>log(2)</td>
</tr>
</tbody>
</table>
Note that `rxFlag` will always output 11 or `calc_lhs` since that is where the final variables are calculated, though you can tweak or test certain parts of `rxode2` by using this flag.

**Residual functions when using `rxode2` functions:**

In addition to ~ hiding output for certain types of output, it also is used to specify a residual output or endpoint when the input is an `rxode2` model function (that includes the residual in the `model({})` block).

These specifications are of the form:

```
var ~ add(add.sd)
```

Indicating the variable `var` is the variable that represents the individual central tendencies of the model and it also represents the compartment specification in the data-set.

You can also change the compartment name using the `|` syntax, that is:

```
var ~ add(add.sd) | cmt
```

In the above case `var` represents the central tendency and `cmt` represents the compartment or `dvid` specification.

**Transformations:**

For normal and related distributions, you can apply the transformation on both sides by using some keywords/functions to apply these transformations.

```
Transformation rxode2/nlmixr2 code
```
Box-Cox  +boxCox(lambda)
Yeo-Johnson +yeoJohnson(lambda)
logit-normal +logitNorm(logit.sd, low, hi)
probit-normal +probitNorm(probid.sd, low, hi)
log-normal +lnorm(lnorm.sd)

By default for the likelihood for all of these transformations is calculated on the \textbf{untransformed} scale.
For bounded variables like logit-normal or probit-normal the low and high values are defaulted to 0 and 1 if missing.
For models where you wish to have a proportional model on one of these transformation you can replace the standard deviation with \texttt{NA}
To allow for more transformations, \texttt{lnorm()}, \texttt{probitNorm()} and \texttt{logitNorm()} can be combined the variance stabilizing \texttt{yeoJohnson()} transformation.

\textit{Normal and t-related distributions:}
For the normal and t-related distributions, we wanted to keep the ability to use skewed distributions additive and proportional in the t/cauchy-space, so these distributions are specified differently in comparison to the other supported distributions within \texttt{nlmixr2}:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>How to Add</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal (log-likelihood)</td>
<td>+dnorm()</td>
<td>cc ~ add(add.sd) + dnorm()</td>
</tr>
<tr>
<td>T-distribution</td>
<td>+dt(df)</td>
<td>cc ~ a dd(add.sd) + dt(df)</td>
</tr>
<tr>
<td>Cauchy (t with df=1)</td>
<td>+dcauchy()</td>
<td>cc ~ add(add.sd) + dcauchy()</td>
</tr>
</tbody>
</table>

Note that with the normal and t-related distributions \texttt{nlmixr2} will calculate \texttt{cwres} and \texttt{npde} under the normal assumption to help assess the goodness of the fit of the model.
Also note that the \texttt{+dnorm()} is mostly for testing purposes and will slow down the estimation procedure in \texttt{nlmixr2}. We suggest not adding it (except for explicit testing). When there are multiple endpoint models that mix non-normal and normal distributions, the whole problem is shifted to a log-likelihood method for estimation in \texttt{nlmixr2}.

\textit{Notes on additive + proportional models:}
There are two different ways to specify additive and proportional models, which we will call \texttt{combined1} and \texttt{combined2}, the same way that Monolix calls the two distributions (to avoid between software differences in naming).
The first, \texttt{combined1}, assumes that the additive and proportional differences are on the standard deviation scale, or:
\[ y = f + (a + b \cdot f^c) \cdot \text{err} \]
The second, \texttt{combined2}, assumes that the additive and proportional differences are combined on a variance scale:
\[ y = f + \sqrt{(a^2 + b^2 \cdot f^{2c})} \cdot \text{err} \]
The default in \texttt{nlmixr2/rxode2} if not otherwise specified is \texttt{combined2} since it mirrors how adding 2 normal distributions in statistics will add their variances (not the standard deviations).
However, the \texttt{combined1} can describe the data possibly even better than \texttt{combined2} so both are possible options in \texttt{rxode2/nlmixr2}.

\textit{Distributions of known likelihoods:}
For residuals that are not related to normal, t-distribution or cauchy, often the residual specification is of the form:

cmt ~ dbeta(alpha, beta)

Where the compartment specification is on the left handed side of the specification.

For generalized likelihood you can specify:

ll(cmt) ~ llik specification

**Ordinal likelihoods:**

Finally, ordinal likelihoods/simulations can be specified in 2 ways. The first is:

err ~ c(p0, p1, p2)

Here `err` represents the compartment and `p0` is the probability of being in a specific category:

<table>
<thead>
<tr>
<th>Category</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>p0</code></td>
</tr>
<tr>
<td>1</td>
<td><code>p1</code></td>
</tr>
<tr>
<td>2</td>
<td><code>p2</code></td>
</tr>
<tr>
<td>3</td>
<td>1-<code>p0</code>-<code>p1</code>-<code>p2</code></td>
</tr>
</tbody>
</table>

It is up to the model to ensure that the sum of the `p` values are less than 1. Additionally you can write an arbitrary number of categories in the ordinal model described above. It seems a little odd that `p0` is the probability for category 1 and sometimes scores are in non-whole numbers. This can be modeled as follows:

err ~ c(p0=0, p1=1, p2=2, 3)

Here the numeric categories are specified explicitly, and the probabilities remain the same:

<table>
<thead>
<tr>
<th>Category</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>p0</code></td>
</tr>
<tr>
<td>1</td>
<td><code>p1</code></td>
</tr>
<tr>
<td>2</td>
<td><code>p2</code></td>
</tr>
<tr>
<td>3</td>
<td>1-<code>p0</code>-<code>p1</code>-<code>p2</code></td>
</tr>
</tbody>
</table>

**General table of supported residual distributions:**

In general all that are supported are in the following table (available in `rxode2::rxResidualError`)

<table>
<thead>
<tr>
<th>Error model</th>
<th>Functional Form</th>
<th>Transformation</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>None</td>
<td></td>
<td>var ~ add(add.sd)</td>
</tr>
<tr>
<td>proportional</td>
<td>None</td>
<td></td>
<td>var ~ prop(prop.sd)</td>
</tr>
<tr>
<td>power</td>
<td>None</td>
<td></td>
<td>var ~ pow(pow.sd, exponent)</td>
</tr>
<tr>
<td>additive+proportional</td>
<td>combined1</td>
<td>None</td>
<td>var ~ add(add.sd) + prop(prop.sd) + combined1()</td>
</tr>
<tr>
<td>additive+proportional</td>
<td>combined2</td>
<td>None</td>
<td>var ~ add(add.sd) + prop(prop.sd) + combined2()</td>
</tr>
<tr>
<td>additive+power</td>
<td>combined1</td>
<td>None</td>
<td>var ~ add(add.sd) + pow(pow.sd, exponent) + combined1()</td>
</tr>
<tr>
<td>additive+power</td>
<td>combined2</td>
<td>None</td>
<td>var ~ add(add.sd) + pow(pow.sd, exponent) + combined2()</td>
</tr>
<tr>
<td>constant</td>
<td>log</td>
<td></td>
<td>var ~ lnorm(add.sd)</td>
</tr>
<tr>
<td>proportional</td>
<td>log</td>
<td></td>
<td>var ~ lnorm(NA) + prop(prop.sd)</td>
</tr>
<tr>
<td>power</td>
<td>log</td>
<td></td>
<td>var ~ lnorm(NA) + pow(pow.sd, exponent)</td>
</tr>
<tr>
<td>additive+proportional</td>
<td>combined1</td>
<td>log</td>
<td>var ~ lnorm(add.sd) + prop(prop.sd) + combined1()</td>
</tr>
</tbody>
</table>
additive+proportional  combined2  log  var ~ lnorm(add.sd) + prop(prop.sd) + combined2
additive+power   combined1  log  var ~ lnorm(add.sd) + pow(pow.sd, exponent) + combined1
additive+power   combined2  log  var ~ lnorm(add.sd) + pow(pow.sd, exponent) + combined2
constant         boxCox   boxCox var ~ boxCox(lambda) + add(add.sd)
proportional     boxCox   boxCox var ~ boxCox(lambda) + prop(prop.sd)
power            boxCox   boxCox var ~ boxCox(lambda) + add(add.sd) + prop(prop.sd)
additive+proportional combined1 boxCox boxCox var ~ boxCox(lambda) + add(add.sd) + prop(prop.sd)
additive+proportional combined2 boxCox boxCox var ~ boxCox(lambda) + add(add.sd) + prop(prop.sd)
additive+power   combined1 boxCox boxCox var ~ boxCox(lambda) + add(add.sd) + pow(pow.sd, exponent)
additive+power   combined2 boxCox boxCox var ~ boxCox(lambda) + add(add.sd) + pow(pow.sd, exponent)
constant         yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + add(add.sd)
proportional     yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + prop(prop.sd)
power            yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + pow(pow.sd, exponent)
additive+proportional combined1 yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + add(add.sd) + prop(prop.sd)
additive+proportional combined2 yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + add(add.sd) + prop(prop.sd)
additive+power   combined1 yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + add(add.sd) + pow(pow.sd, exponent)
additive+power   combined2 yeoJohnson yeoJohnson var ~ yeoJohnson(lambda) + add(add.sd) + pow(pow.sd, exponent)
constant         logit     logit var ~ logitNorm(logit.sd)
proportional     logit     logit var ~ logitNorm(NA) + prop(prop.sd)
power            logit     logit var ~ logitNorm(NA) + pow(pow.sd, exponent)
additive+proportional combined1 logit  logit var ~ logitNorm(logit.sd) + prop(prop.sd)
additive+proportional combined2 logit  logit var ~ logitNorm(logit.sd) + prop(prop.sd)
additive+power   combined1 logit  logit var ~ logitNorm(logit.sd) + pow(pow.sd, exponent)
additive+power   combined2 logit  logit var ~ logitNorm(logit.sd) + pow(pow.sd, exponent)
additive         yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(logit.sd)
proportional     yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(NA) + prop(prop.sd)
power            yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(NA) + pow(pow.sd, exponent)
additive+proportional combined1 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(logit.sd) + prop(prop.sd)
additive+proportional combined2 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(logit.sd) + prop(prop.sd)
additive+power   combined1 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(logit.sd) + pow(pow.sd, exponent)
additive+power   combined2 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + logitNorm(logit.sd) + pow(pow.sd, exponent)
constant         probit    probit var ~ probitNorm(probit.sd)
proportional     probit    probit var ~ probitNorm(NA) + prop(prop.sd)
power            probit    probit var ~ probitNorm(NA) + pow(pow.sd, exponent)
additive+proportional combined1 probit  probit var ~ probitNorm(probit.sd) + prop(prop.sd)
additive+proportional combined2 probit  probit var ~ probitNorm(probit.sd) + prop(prop.sd)
additive+power   combined1 probit  probit var ~ probitNorm(probit.sd) + pow(pow.sd, exponent)
additive+power   combined2 probit  probit var ~ probitNorm(probit.sd) + pow(pow.sd, exponent)
additive         yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(probit.sd)
proportional     yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(NA) + probitNorm(probit.sd)
power            yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(NA) + probitNorm(NA) + probitNorm(probit.sd)
additive+proportional combined1 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(probit.sd) + probitNorm(probit.sd)
additive+proportional combined2 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(probit.sd) + probitNorm(probit.sd)
additive+power   combined1 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(probit.sd) + probitNorm(probit.sd)
additive+power   combined2 yeoJohnson(yeom()) yeoJohnson(yeom()) var ~ yeoJohnson(lambda) + probitNorm(probit.sd) + probitNorm(probit.sd)
constant+t       None      None var ~ add(add.sd) + dt(df)
proportional+t   None      None var ~ prop(prop.sd) + dt(df)
power+t          None      None var ~ pow(pow.sd, exponent) + dt(df)
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy} 
\text{power+cauchy} 
\text{additive+proportional+cauchy} 
\text{additive+power+cauchy} 
\text{constant+cauchy} 
\text{proportional+cauchy}
proportional+cauchy
power+cauchy
additive+proportional+cauchy combined1
additive+proportional+cauchy combined2
additive+power+cauchy combined1
additive+power+cauchy combined2
additive+cauchy
proportional+cauchy
power+cauchy
additive+proportional+cauchy combined1
additive+proportional+cauchy combined2
additive+power+cauchy combined1
additive+power+cauchy combined2
poisson
binomial
beta
chisq
exponential
uniform
weibull
gamma
geometric
negative binomial form #1
negative binomial form #2
ordinal probability
log-likelihood

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 `\dQuote{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 `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 `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., `\dQuote{rm(m1)}`.
* `run` deprecated, use `solve`.
* `get.index` deprecated.
* `getObj` internal (not user callable) function.
Creating rxode2 models

NA

Author(s)

Melissa Hallow, Wenping Wang and Matthew Fidler

References


See Also

eventTable(), et(), add.sampling(), add.dosing()

Examples

```r
mod <- 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
  )
}
```

m1 <- rxode2(mod)
print(m1)

# Step 2 - Create the model input as an EventTable, # including dosing and observation (sampling) events
# QD (once daily) dosing for 5 days.
qd <- et(amountUnits = "ug", timeUnits = "hours") %>%
et(amt = 10000, addl = 4, ii = 24)

# Sample the system hourly during the first day, every 8 hours # then after
qd <- qd %>% et(0:24) %>%
et(from = 24 + 8, to = 5 * 24, by = 8)

# Step 3 - solve the system
qd.cp <- rxSolve(m1, qd)
head(qd.cp)

---

### `rxode2`<-

*Set the function body of an `rxUi` object while retaining other object information (like data)*

#### Description

Set the function body of an `rxUi` object while retaining other object information (like data)

#### Usage

```r
rxode2(x, envir = environment(x)) <- value
```

## S3 replacement method for class `"function"`
```r
rxode2(x, envir = environment(x)) <- value
```

## Default S3 replacement method:
```r
rxode(x, envir = environment(x)) <- value
```
```r
RxODE(x, envir = environment(x)) <- value
```
Arguments

- `x` The rxUi object
- `envir` environment where the assignment occurs
- `value` the value that will be assigned

Value

The rxode2 ui/function

Examples

```r
one.compartment <- function() {
  ini(
    tka <- log(1.57); label("Ka")
    tcl <- log(2.72); label("Cl")
    tv <- log(31.5); label("V")
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  )
}

two.compartment <- function() {
  ini(
    lka <- 0.45 ; label("Absorption rate (Ka)")
    lcl <- 1 ; label("Clearance (CL)")
    lvc <- 3 ; label("Central volume of distribution (V)")
    lvp <- 5 ; label("Peripheral volume of distribution (Vp)")
    lq <- 0.1 ; label("Intercompartmental clearance (Q)")
    propSd <- 0.5 ; label("Proportional residual error (fraction)")
  )
  model(
    ka <- exp(lka)
    cl <- exp(lcl)
    vc <- exp(lvc)
    vp <- exp(lvp)
    q <- exp(lq)
    kel <- cl/vc
    k12 <- q/vc
    k21 <- q/vp
  )
}
```
rxOptExpr

\[
\begin{align*}
d/dt(\text{depot}) & \leftarrow -ka*\text{depot} \\
d/dt(\text{central}) & \leftarrow ka*\text{depot} - kel*\text{central} - k12*\text{central} + k21*\text{peripheral1} \\
d/dt(\text{peripheral1}) & \leftarrow k12*\text{central} - k21*\text{peripheral1} \\
\text{cp} & \leftarrow \text{central} / \text{vc} \\
\text{cp} & \sim \text{prop(propSd)}
\end{align*}
\]

ui <- rxode2(one.compartment)

rxode2(ui) <- two.compartment

---

### 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)
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**

```r
rxParams(obj, ...)
```

```r
## 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
)
```

```r
## S3 method for class 'rxSolve'
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 'rxEt'
rxParams(
  obj, 
  ..., 
  params = NULL, 
  inits = NULL, 
  iCov = NULL, 
  keep = NULL, 
  thetaMat = NULL, 
  omega = NULL, 
  sigma = NULL, 
  dfSub = NULL, 
  dfObs = NULL, 
  nSub = NULL, 
  nStud = NULL 
)

rxParam(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 indicting 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 compartment), 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(), rxInits(), rxLhs(), rxModelVars(), rxState()

rxPkg

Description
Creates a package from compiled rxode2 models

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

- **package**: String of the package name to create
- **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

---

**rxpois**  
*Simulate random Poisson 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
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

```r
## 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
rx <- function() {
  model({
    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.
- `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 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)
```

```r
## Sample inhomogenous Poisson rate of 1/10
```
## rxPreferredDistributionName

Change distribution name to the preferred distribution name term

### Description

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

### Usage

```r
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
taxProgress(num, core = 0L)
taxTick()
taxProgressStop(clear = TRUE)
taxProgressAbort(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**

---

**Description**

rxRemoveControl options for UI object

**Usage**

```r
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**

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

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

```r
rename.rxUi(.data, ...)
```

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

```r
## S3 method for class 'rxUi'
xRename(.data, ...)
```

```r
## S3 method for class `function`
xRename(.data, ...)
```

```r
## Default S3 method:
xRename(.data, ...)
```

**Arguments**

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

**Details**

This is similar to dplyr's `rename()` function. When dplyr is loaded, the s3 methods work for the ui objects.

Note that the `.rxRename()` is the internal function that is called when renaming and is likely not what you need to call unless you are writing your own extension of the function.

**Value**

New model with items renamed

**Author(s)**

Matthew L. Fidler

**Examples**

```r
ocmt <- function() {
  ini({
    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 rxode2 supported reserved keywords_

### Description
A list and description of rxode2 supported reserved keywords

### Usage
```
rxReservedKeywords
```

### Format
A data frame with 3 columns and 31 rows

- **Reserved Name**  Reserved Keyword Name
- **Meaning**        Reserved Keyword Meaning
- **Alias**          Keyword Alias

---

**rxResidualError**

_A description of Rode2 supported residual errors_

### Description
A description of Rode2 supported residual errors

### Usage
```
rxResidualError
```
Format

- **Error model**: A description of the type of residual error
- **Functional Form**: For additive and proportional what functional form is used
- **Transformation**: The type of transformation that is done on the DV and the prediction
- **code**: Example code for the residual error type
- **addProp**: The type of add+prop residual error default that would be equivalent
- **lhs**: what the left handed side of the specification represents, either a response variable, or a compartment specification

---

**rxS**

*Load a model into a symengine environment*

---

Description

Load a model into a symengine environment

Usage

```
rxS(x, doConst = TRUE, promoteLinSens = FALSE, envir = parent.frame())
```

Arguments

- **x**: rxode2 object
- **doConst**: Load constants into the environment as well.
- **promoteLinSens**: Promote solved linear compartment systems to sensitivity-based solutions.
- **envir**: default is NULL; Environment to put symengine variables in.

Value

rxode2/symengine environment

Author(s)

Matthew Fidler
**rxSetCovariateNamesForPiping**

*Assign covariates for piping*

**Description**

Assign covariates for piping

**Usage**

```r
rxSetCovariateNamesForPiping(covariates = NULL)
```

**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

---

**rxSetControl**

*rxSetControl options for UI object*

**Description**

rxSetControl options for UI object

**Usage**

```r
rxSetControl(ui, control)
```

**Arguments**

- **ui**
  - rxode2 ui object
- **control**
  - Default value

**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)
```
`rxSetPipingAuto`  
*Set the variables for the model piping automatic covariate selection*

**Description**

Set the variables for the model piping automatic covariate selection

**Usage**

```r
rxSetPipingAuto(
  thetamodelVars = rex::rex(or("tv", "t", "pop", "POP", "Pop", "TV", "T", "cov", "err", "eff")),
  covariateExceptions = rex::rex(start, or("wt", "sex", "crcl", "kout"), end),
)
```

**Arguments**

- `thetamodelVars`  
  This is the prefixes for the theta model variables in a regular expression
- `covariateExceptions`  
  This is a regular expression of covariates that should always be covariates
- `etaParts`  
  This is the list of eta prefixes/post-fixes that identify a variable as a between subject variability

**Details**

This is called once at startup to set the defaults, though you can change this if you wish so that piping can work differently for your individual setup

**Value**

Nothing, called for side effects

**Author(s)**

Matthew L. Fidler
rxSetProd  Defunct setting of product

Description
Defunct setting of product

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

Arguments
  type  used to be type of product

Value
  nothing

rxSetProgressBar  Set timing for progress bar

Description
Set timing for progress bar

Usage
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*

**Description**

Defunct setting of sum

**Usage**

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

**Arguments**

- `type` used to be type of product

**Value**

nothing

---

**rxShiny**  
*Use Shiny to help develop an rxode2 model*

**Description**

Use Shiny to help develop an rxode2 model

**Usage**

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

```r
## 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
- **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

---

### 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,
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 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 \( \chi^2 \) 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 \( \exp(\chi^2) \) 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 \(-\infty\))

thetaUpper Upper bounds for simulated population unexplained variability (by default \(\infty\))

thetaDf The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to \(\infty\) 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 \(\text{NA} \) 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 \(-\infty\))

sigmaUpper Upper bounds for simulated unexplained variability (by default \(\infty\))

sigmaDf Degrees of freedom of the sigma t-distribution. By default it is equivalent to \(\infty\), 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 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.
- 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.

**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

Options, Solving & Simulation of an ODE/solved system

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.

Usage

```r
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,
naTimeHandle = c("ignore", "warn", "error"),
add1KeepsCov = FALSE,
add1DropSs = TRUE,
ssAtDoseTime = TRUE,
ss2cancelAllPending = FALSE,
envir = parent.frame()
)

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

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

## S3 method for class 'rxode2tos'
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ..., 
  theta = NULL,
  eta = NULL,
  envir = parent.frame()
)

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

## S3 method for class 'nlmixr2FitCore'
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ..., 
  theta = NULL,
  eta = NULL,
  envir = parent.frame()
)

## Default S3 method:
rxSolve(
  object,
params = NULL,
envents = NULL,
inits = NULL,
...
theta = NULL,
eta = NULL,
envir = parent.frame()
)

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

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

## S3 method for class "function"
predict(object, ...)

## S3 method for class 'rxUi'
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 "function"
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,
  envir = parent.frame()
)

### 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 $0.5 \times 10^{-(\text{sigdig}-2)}$ for regular ODEs. For the sensitivity equations the default is $0.5 \times 10^{-\text{sigdig}-1.5}$ (sensitivity changes only...
applicable for liblsoda). This also controls the atol/rtol of the steady state solutions. The ssAtol/ssRtol is $0.5 \times 10^{-\text{sigdig}}$ and for the sensitivities $0.5 \times 10^{-\text{sigdig}+0.625}$. 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 $hmax=\text{NA}$ (default), uses the average difference + $hmaxSd \times sd$ in times and sampling events. The $hmaxSd$ is a user specified parameter and which defaults to zero. When $hmax=\text{NULL}$ 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 $hini = 0$)

- **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.

- **cores**: Number of cores used in parallel ODE solving. This is equivalent to calling `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 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.

**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.

**theta**

A vector of parameters that will be named \( \text{THETA}[\#] \) and added to parameters.

**thetaLower**

Lower bounds for simulated population parameter variability (by default \(-\infty\))

**thetaUpper**

Upper bounds for simulated population unexplained variability (by default \(\infty\))

**eta**

A vector of parameters that will be named \( \text{ETA}[\#] \) and added to parameters.

**addDosing**

Boolean indicating if the solve should add rxode2 EVID and related columns. This will also include dosing information and estimates at the doses. Be default, rxode2 only includes estimates at the observations. (default \(\text{FALSE}\)). When \(\text{addDosing}\) is \(\text{NULL}\), only include \(\text{EVID}=0\) on solve and exclude any model-times or \(\text{EVID}=2\). If \(\text{addDosing}\) is \(\text{NA}\) the classic rxode2 EVID events are returned. When \(\text{addDosing}\) is \(\text{TRUE}\) add the event information in NONMEM-style format; If \(\text{subsetNonmem}=\text{FALSE}\) rxode2 will also include extra event types (EVID) for ending infusion and modeled times:

- \(\text{EVID}=-1\) when the modeled rate infusions are turned off (matches \(\text{rate}=-1\))
- \(\text{EVID}=-2\) When the modeled duration infusions are turned off (matches \(\text{rate}=-2\))
- \(\text{EVID}=-10\) When the specified rate infusions are turned off (matches \(\text{rate}>0\))
- \(\text{EVID}=-20\) When the specified dur infusions are turned off (matches \(\text{dur}>0\))
- \(\text{EVID}=101,102,103,\ldots\) Modeled time where 101 is the first model time, 102 is the second etc.

**stateTrim**

When amounts/concentrations in one of the states are above this value, trim them to be this value. By default \(\infty\). Also trims to \(-\text{stateTrim}\) for large negative amounts/concentrations. If you want to trim between a range say \(c(0, 2000000)\) you may specify 2 values with a lower and upper range to make sure all state values are in the reasonable range.

**updateObject**

This is an internally used flag to update the rxode2 solved object (when supplying an rxode2 solved object) as well as returning a new object. You probably should not modify it’s \(\text{FALSE}\) default unless you are willing to have unexpected results.

**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 \(\text{NA}\) 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 \(\text{NULL}\) which is equivalent to \(\infty\) degrees, or to simulate from a normal distribution instead of a t-distribution.

**omegaIsChol**

Indicates if the omega supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
Omega separation strategy

Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the \texttt{thetaMat} matrix.

- "lkj" simulates the correlation matrix from the \texttt{rLKJ1} matrix with the distribution parameter \texttt{eta} equal to the degrees of freedom \texttt{nu} by \((\texttt{nu} - 1)/2\)
- "separation" simulates from the identity inverse Wishart covariance matrix with \texttt{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.

When taking omega values from the \texttt{thetaMat} simulations (using the separation strategy for covariance simulation), how should the \texttt{thetaMat} values be turned into standard deviation values:

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

Lower bounds for simulated ETAs (by default -Inf)

Upper bounds for simulated ETAs (by default Inf)

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

Named theta matrix.

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

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

Number virtual studies to characterize uncertainty in estimated parameters.

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 rxode2. 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 a 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:

- 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). hmin 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 foci 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

naTimeHandle  Determines what time of handling happens when the time becomes NA: current options are:
• ignore this ignores the NA time input and passes it through.
• warn (default) this will produce a warning at the end of the solve, but continues solving passing through the NA time
• error this will stop this solve if this is not a parallel solved ODE (otherwise stopping can crash R)

addlKeepsCov  This determines if the additional dosing items repeats the dose only (FALSE) or keeps the covariates at the record of the dose (TRUE)

addlDropSs  When there are steady state doses with an addl specification the steady state flag is dropped with repeated doses (when TRUE) or retained (when FALSE)

ssAtDoseTime  Boolean that when TRUE back calculates the steady concentration at the actual time of dose, otherwise when FALSE the doses are shifted

ss2cancelAllPending  When TRUE the SS=2 event type cancels all pending doses like SS=1. When FALSE the pending doses not canceled with SS=2 (the infusions started before SS=2 occurred are canceled, though).

evir  is the environment to look for R user functions (defaults to parent environment)
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

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()
```
**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**

```r
# 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
Sync options with rxode2 variables

Description
Accessing rxode2 options viagetOption slows down solving. This allows the options to be synced with variables.

Usage
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

A list and description of Rode supported syntax functions

Description
A list and description of Rode supported syntax functions

Usage
rxSyntaxFunctions

Format
A data frame with 3 columns and 98 rows

Function  Reserved function Name
Description Description of function
Aliases  Function Aliases
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

\[
\text{rxt}(\text{df}, n = 1L, \text{ncores} = 1L)
\]

Arguments

- \text{df} \quad \text{degrees of freedom} (> 0, maybe non-integer). \text{df} = \text{Inf} is allowed.
- \text{n} \quad \text{number of observations}. If \text{length(n)} > 1, the length is taken to be the number required.
- \text{ncores} \quad \text{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
```
\texttt{rxTempDir}

\begin{verbatim}
xrt(df = 3, n = 10, ncores = 2) # You can parallelize the simulation using openMP

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

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

rx <- function() {
  model({
    a <- rxt(3)
  })
}

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

s <- rxSolve(rx, et)
\end{verbatim}

\begin{description}
\item[rxTempDir] Get the rxode2 temporary directory
\end{description}

\begin{verbatim}
xrTheme

rxTheme is the ggplot2 theme for rxode2 plots

rxTheme(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22,
  grid = TRUE
)
\end{verbatim}

\begin{description}
\item[rxTheme] rxTheme is the ggplot2 theme for rxode2 plots
\end{description}
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

```r
rxToSE(
  x,
  envir = NULL,
  progress = FALSE,
  promoteLinSens = TRUE,
  parent = parent.frame()
)
```

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

```r
rxFromSE(
  x,
  unknownDerivatives = c("forward", "central", "error"),
  parent = parent.frame()
)
```

```r
.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.
- **parent**: is the parent environment to look for R-based user functions
- **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:
```r
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)

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))

---

**rxUiGet.cmtLines**  
*S3 for getting information from UI model*

**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 'symengineModelNoPrune'
rxUiGet(x, ...)

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

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

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

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

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

Value

value that was requested from the UI object

Author(s)

Matthew Fidler
**rxunif**

Simulate uniform 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**

```
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 <- function() {
  model({
    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**

```r
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**

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

```r
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.
Value
nothing

Author(s)
Matthew L. Fidler

---

**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**

```r
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
  
  `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**

Weibull random deviates
Examples

```r
## 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 <- function() {
  model({
    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

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

```r
gem_amt(
  mapping = NULL,
  data = NULL,
```
Arguments

mapping

Set of aesthetic mappings created by `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 naming the adjustment (e.g. "jitter" to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

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()`.

...

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

Value

This returns a `stat_amt` in context of a `ggplot2` plot
library(rxode2)
library(units)

## Model from RxODE tutorial
mod1 <- function() {
    ini({
        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
    })
    model({
        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

## 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, linewidth=1.2)
stat_cens  Censoring geom/stat

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

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

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(). 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 adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

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.

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().

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
- CENS for the censoring information; (-1 right censored, 0 no censoring or 1 left censoring)
- LIMIT which represents the corresponding limit

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

Value

This returns a ggplot2 stat

summary.rxode2  Print expanded information about the rxode2 object.

Description

This prints the expanded information about the rxode2 object.

Usage

## 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

`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:

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

Value

uppergamma results

Author(s)

Matthew L. Fidler

Examples

\begin{align*}
\text{uppergamma}(1, 3) \\
\text{uppergamma}(1:3, 3) \\
\text{uppergamma}(1, 1:3)
\end{align*}

---

zeroRe  

Set random effects and residual error to zero

Description

Set random effects and residual error to zero

Usage

zeroRe(object, which = c("omega", "sigma"), fix = TRUE)

Arguments

- object: The model to modify
- which: The types of parameters to set to zero
- fix: Should the parameters be fixed to the zero value?

Value

The object with some parameters set to zero

Author(s)

Bill Denney
See Also

Other Initial conditions: `ini.rxUi()`

Examples

```r
one.compartment <- function() {
  ini({
    tka <- log(1.57); label("Ka")
    tcl <- log(2.72); label("Cl")
    tv <- log(31.5); label("V")
    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)
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  })
}
zeroRe(one.compartment)
```
Index

* Initial conditions
  ini.rxUi, 28
  zeroRe, 175
* Internal
  .matchesLangTemplate, 7
  odeMethodToInt, 58
  plot.rxSolve, 59
* Nonlinear regression
  rxode2, 93
* ODE models
  rxode2, 93
* Ordinary differential equations
  rxode2, 93
* PK/PD
  genShinyApp.template, 26
* Pharmacodynamics (PD)
  rxode2, 93
* Pharmacokinetics (PK)
  rxode2, 93
* Query model information
  rxDfdy, 74
  rxLhs, 88
  rxParams, 116
  rxState, 151
* datasets
  rxReservedKeywords, 126
  rxResidualError, 126
  rxSyntaxFunctions, 155
* models
  rxode2, 93
* nonlinear
  genShinyApp.template, 26
  rxode2, 93
* pharmacometrics
  genShinyApp.template, 26
* rxode2 plotting
  plot.rxSolve, 59
  rxTheme, 157
* simulation
  genShinyApp.template, 26
  .C(), 71
  .Call(), 71
  .copyUi, 5
  handleSingleErrTypeNormOrTFoceiBase, 6
  .matchesLangTemplate, 7
  modelHandleModelLines, 7
  quoteCallInfolines, 8
  rxFromSE(rxToSE), 158
  rxLinCmtGen, 9
  rxRename(rxRename), 124
  rxToSE(rxToSE), 158
  rxWithOptions, 9
  rxWithWd, 10
  add.dosing(), 111
  add.sampling(), 111
  aes(), 170, 172
  as.ini, 11
  as.model, 13
  as.rxUi, 15
  assertRxUi, 16
  assertRxUiEstimatedResiduals (assertRxUi), 16
  assertRxUiMixedOnly (assertRxUi), 16
  assertRxUiMuRefOnly (assertRxUi), 16
  assertRxUiNormal (assertRxUi), 16
  assertRxUiPopulationOnly (assertRxUi), 16
  assertRxUiPrediction (assertRxUi), 16
  assertRxUiRandomOnIdOnly (assertRxUi), 16
  assertRxUiSingleEndpoint (assertRxUi), 16
  assertRxUiTransformNormal (assertRxUi), 16
  binomProbs, 18
  borders(), 170, 173
write.template.ui
   (genShinyApp.template), 26
write.template.ui(), 26

zeroRe, 30, 175