Package ‘rxode2’

May 17, 2022

Version 2.0.7

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, data.table (>= 1.12.4), digest, dplyr (>= 0.8.0), ggrelp, gridExtra, htmltools, knitr, learnr, microbenchmark, nlme, remotes, rlang, rmarkdown, scales, shiny, stringi, symengine, testthat, tidyr, usethis, vdiffrr (>= 1.0), withr, xgxr, pillar, tibble, units (>= 0.6-0), rsconnect, devtools, patchwork, nlmixr2data

Imports PreciseSums (>= 0.3), Rcpp (>= 0.12.3), assertthat, backports, checkmate, cli (>= 2.0.0), dparser (>= 0.1.8), ggplot2, inline, lotri (>= 0.4.0), magrittr, memoise, methods, rex, qs, sys, tools, utils

Description Facilities for running simulations from ordinary differential equation (‘ODE’) models, such as pharmacometrics and other compartmental models. A compilation manager translates the ODE model into C, compiles it, and dynamically loads the object code into R for improved computational efficiency. An event table object facilitates the specification of complex dosing regimens (optional) and sampling schedules. NB: The use of this package requires both C and Fortran compilers, for details on their use with R please see Section 6.3, Appendix A, and Appendix D in the ‘R Administration and Installation’ manual. Also the code is mostly released under GPL. The ‘VODE’ and ‘LSODA’ are in the public domain. The information is available in the inst/COPYRIGHTS.

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

NeedsCompilation yes

VignetteBuilder knitr

License GPL (>= 3)

RoxygenNote 7.1.2
Biarch true
LinkingTo dparser (>= 1.3.1-0), PreciseSums (>= 0.3), RcppEigen (>= 0.3.3.3.0), RcppArmadillo (>= 0.9.300.2.0), sitmo, StanHeaders (>= 2.21.0), BH
Encoding UTF-8
LazyData true
Language en-US
Config/testthat/edition 3
Author Matthew L. Fidler [aut, cre] (<https://orcid.org/0000-0001-8538-6691>), Melissa Hallow [aut], Wenping Wang [aut], Zufar Mulyukov [ctb], Alan Hindmarsh [ctb], Awad H. Al-Mohy [ctb], Matt Dowle [ctb], Cleve Moler [ctb], David Cooley [ctb], Drew Schmidt [ctb], Arun Srinivasan [ctb], Ernst Hairer [ctb], Gerhard Wanner [ctb], Goro Fuji [ctb], Hadley Wickham [ctb], Jack Dongarra [ctb], Linda Petzold [ctb], Martin Maechler [ctb], Matteo Fasiolo [ctb], Morwenn [ctb], Nicholas J. Higham [ctb], Roger B. Sidje [ctb], Simon Frost [ctb], Kevin Ushey [ctb], Yu Feng [ctb]
Repository CRAN
Date/Publication 2022-05-17 17:50:02 UTC

R topics documented:

.copyUi ......................................................... 5
.handleSingleErrTypeNormOrTFoceiBase ................................ 6
.modelHandleModelLines ............................................. 6
.quoteCallInfoLines ............................................... 7
.rxLinCmtGen ..................................................... 8
.rxWithOptions .................................................. 9
R topics documented:

.rxWithWd .................................................. 9
add.dosing .................................................. 10
add.sampling .............................................. 13
as.et ......................................................... 15
assertRxUi .................................................. 16
cvPost ....................................................... 18
erf ............................................................. 21
et ............................................................. 21
etExpand ..................................................... 26
etRbind ....................................................... 27
etRep ......................................................... 30
etSeq ........................................................ 33
eventTable .................................................. 36
gammmap ................................................... 38
gammmapDer ................................................ 39
gammmapInv ............................................... 39
gammaq ...................................................... 40
gammaqInv ................................................ 41
gammap ...................................................... 42
getRxThreads ............................................ 44
ini.rxUi ...................................................... 45
logit .......................................................... 46
lowergamma ................................................ 48
model.function .......................................... 49
phi ............................................................. 50
plot.rxSolve ............................................... 51
probit ......................................................... 51
rinvchisq ................................................ 52
rxAllowUnload ........................................... 53
rxAppendModel ............................................ 54
rxAssignControlValue .................................... 55
rxAssignPtr ............................................... 56
rxbeta ....................................................... 56
rxbinom ..................................................... 57
rcaucahy .................................................... 59
rxChbindStudyIndividual .................................. 60
rxchisq ...................................................... 61
rxClean .................................................... 63
rxCompile ................................................ 63
rxControlUpdateSens .................................... 65
rxCreateCache ........................................... 66
rxD ........................................................ 67
rxDelete .................................................. 68
rxDerived .................................................. 68
rxDfdy ....................................................... 70
rxEvid ...................................................... 71
rxexp ........................................................ 72
rf ............................................................. 73
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rxFun</td>
<td>74</td>
</tr>
<tr>
<td>rxgamma</td>
<td>76</td>
</tr>
<tr>
<td>rgeom</td>
<td>77</td>
</tr>
<tr>
<td>rxGetControl</td>
<td>79</td>
</tr>
<tr>
<td>rxGetLin</td>
<td>79</td>
</tr>
<tr>
<td>rxGetrxode2</td>
<td>80</td>
</tr>
<tr>
<td>rxGetSeed</td>
<td>81</td>
</tr>
<tr>
<td>rxHtml</td>
<td>82</td>
</tr>
<tr>
<td>rxIndLinState</td>
<td>82</td>
</tr>
<tr>
<td>rxIndLinStrategy</td>
<td>83</td>
</tr>
<tr>
<td>rxIndLin_</td>
<td>83</td>
</tr>
<tr>
<td>rxInv</td>
<td>84</td>
</tr>
<tr>
<td>rxIsCurrent</td>
<td>85</td>
</tr>
<tr>
<td>rxLhs</td>
<td>85</td>
</tr>
<tr>
<td>rxLock</td>
<td>86</td>
</tr>
<tr>
<td>rxNorm</td>
<td>86</td>
</tr>
<tr>
<td>rxnorm</td>
<td>87</td>
</tr>
<tr>
<td>rxode2</td>
<td>88</td>
</tr>
<tr>
<td>rxOptExpr</td>
<td>96</td>
</tr>
<tr>
<td>rxord</td>
<td>97</td>
</tr>
<tr>
<td>rxParms</td>
<td>98</td>
</tr>
<tr>
<td>rxPkg</td>
<td>100</td>
</tr>
<tr>
<td>rxpois</td>
<td>101</td>
</tr>
<tr>
<td>rxPp</td>
<td>102</td>
</tr>
<tr>
<td>rxPreferredDistributionName</td>
<td>104</td>
</tr>
<tr>
<td>rxProgress</td>
<td>105</td>
</tr>
<tr>
<td>rxRandNV</td>
<td>106</td>
</tr>
<tr>
<td>rxRateDur</td>
<td>107</td>
</tr>
<tr>
<td>rxRemoveControl</td>
<td>107</td>
</tr>
<tr>
<td>rxRename</td>
<td>108</td>
</tr>
<tr>
<td>rxReservedKeywords</td>
<td>109</td>
</tr>
<tr>
<td>rxRmvn</td>
<td>109</td>
</tr>
<tr>
<td>rxS</td>
<td>112</td>
</tr>
<tr>
<td>rxSetControl</td>
<td>113</td>
</tr>
<tr>
<td>rxSetCovariateNamesForPiping</td>
<td>113</td>
</tr>
<tr>
<td>rxSetIni0</td>
<td>115</td>
</tr>
<tr>
<td>rxSetProd</td>
<td>115</td>
</tr>
<tr>
<td>rxSetProgressBar</td>
<td>116</td>
</tr>
<tr>
<td>rxSetSeed</td>
<td>116</td>
</tr>
<tr>
<td>rxSetSum</td>
<td>118</td>
</tr>
<tr>
<td>rxShiny</td>
<td>118</td>
</tr>
<tr>
<td>rxSimThetaOmega</td>
<td>120</td>
</tr>
<tr>
<td>rxSolve</td>
<td>123</td>
</tr>
<tr>
<td>rxStack</td>
<td>136</td>
</tr>
<tr>
<td>rxState</td>
<td>137</td>
</tr>
<tr>
<td>rxSumProdModel</td>
<td>138</td>
</tr>
<tr>
<td>rxSupportedFuns</td>
<td>138</td>
</tr>
<tr>
<td>rxSuppressMsg</td>
<td>139</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)

Arguments

env Environment for the parsed model
pred1 The data.frame of the current error

Value

A list of the lines added. The lines will contain

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

Author(s)

Matthew Fidler

.modelHandleModellines

Handle model lines

Description

Handle model lines
Usage

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

Arguments

- **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 (default)
  - FALSE when the lines are replaced in the model
  - NA is when the lines are pre-pended to the model instead of replaced
- **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).
- **envir**: Environment for evaluation

Value

New UI

Author(s)

Matthew L. Fidler

Description

Returns quoted call information
Usage

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

Arguments

callInfo  Call information
envir     Environment for evaluation (if needed)

Value

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

Author(s)

Matthew L. Fidler

---

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

Description

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

Usage

.rxLinCmtGen(lenState, vars)

Arguments

lenState  Length of the state
vars      Variables in the model

Value

Model variables of expanded linCmt model

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
.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
.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()
```

---

**add.dosing**

*Add dosing to eventTable*

Description

This adds a dosing event to the event table. This is provided for piping syntax through magrittr. It can also be accessed by `eventTable$add.dosing(...)`

Usage

```r
add.dosing(
  eventTable,
  dose,
  nbr.doses = 1L,
  dosing.interval = 24,
  dosing.to = 1L,
  rate = NULL,
  amount.units = NA_character_,
  start.time = 0,
  do.sampling = FALSE,
  time.units = NA_character_,
  ...
)
```

Arguments

- `eventTable` eventTable object; When accessed from object it would be `eventTable$`
- `dose` numeric scalar, dose amount in `amount.units`;
- `nbr.doses` integer, number of doses;
- `dosing.interval` required numeric scalar, time between doses in `time.units`, defaults to 24 of `time.units="hours"`;
- `dosing.to` integer, compartment the dose goes into (first compartment by default);
- `rate` for infusions, the rate of infusion (default is NULL, for bolus dosing);
add.dosing

amount.units optional string indicating the dosing units. Defaults to NA to indicate as per the original EventTable definition.

start.time required dosing start time;

do.sampling logical, should observation sampling records be added at the dosing times? Defaults to FALSE.

time.units optional string indicating the time units. Defaults to "hours" to indicate as per the original EventTable definition.

Other parameters passed to et().

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

Author(s)

Matthew L. Fidler
Matthew L Fidler, Wenping Wang

References


See Also
eventTable, add.sampling, add.dosing, et, etRep, etRbind, rxode2

Examples

library(rxode2)
library(units)

## Model from rxode2 tutorial
mod1 <-rxode2(
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) =-KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
\[
\begin{align*}
\frac{d}{dt}(peri) &= Q \cdot C2 - Q \cdot C3; \\
\frac{d}{dt}(eff) &= Kin - Kout \cdot (1 - C2/(EC50 + C2)) \cdot eff;
\end{align*}
\]

```r
## 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)
plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")
qd <- et(timeUnits = "hr") %>%
et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")
et <- seq(infusion, qd)
infusionQd <- rxSolve(mod1, et)
plot(infusionQd, C2)

## 2wk-on, 1wk-off
qd <- et(timeUnits = "hr") %>%
et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- seq(qd, set_units(1, "weeks"), qd) %>%
add.sampling(set_units(seq(0, 5.5, by=0.005), weeks))
wkOnOff <- rxSolve(mod1, et)
plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function
qd <- et(timeUnits = "hr") %>%
et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
```
et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
    add.sampling(set_units(seq(0, 12.5,by=0.005),"weeks"))

repCycle4 <- rxSolve(mod1, et)

plot(repCycle4, C2)

---

**add.sampling**

*Add sampling to eventTable*

**Description**

This adds a dosing event to the event table. This is provided for piping syntax through magrittr. It can also be accessed by `eventTable$add.sampling()`.

**Usage**

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

**Arguments**

- `eventTable`: An `eventTable` object. When accessed from object it would be `eventTable$`
- `time`: a vector of time values (in `time.units`).
- `time.units`: an optional string specifying the time units. Defaults to the units specified when the `EventTable` was initialized.

**Value**

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

**Author(s)**

Matthew L Fidler, Wenping Wang

**References**


**See Also**

`eventTable, add.sampling, add.dosing, et, etRep, etRbind, rxode2`
library(rxode2)
library(units)

## Model from rxode2 tutorial
mod1 <- rxode2({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) =-KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

## These are making the more complex regimens of the rxode2 tutorial
## 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)
plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days
## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")
et <- seq(infusion,qd)
infusionQd <- rxSolve(mod1, et)
plot(infusionQd, C2)

## 2wk-on, 1wk-off
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- seq(qd, set_units(1,"weeks"), qd) %>%
    add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))
wkOnOff <- rxSolve(mod1, et)
plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
    add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))
repCycle4 <- rxSolve(mod1, et)
plot(repCycle4, C2)

---

**as.et**  
*Coerce object to data.frame*

**Description**  
Coerce object to data.frame

**Usage**  
```r
as.et(x, ...)
```

### Default S3 method:
```
as.et(x, ...)
```

**Arguments**

- `x`  
  Object to coerce to et.
- `...`  
  Other parameters
Value

An event table

 assertRxUi

Assert properties of the rxUi models

Description

Assert properties of the rxUi models

Usage

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

Arguments

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

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)
assertRxUi

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

**Value**

the rxUi model

**Author(s)**

Matthew L. Fidler

**Examples**

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

assertRxUi(one.cmt)
# assertRxUi(rnorm) # will fail
```
Sample a covariance Matrix from the Posterior Inverse Wishart distribution.

**Description**

Note this Inverse wishart rescaled to match the original scale of the covariance matrix.

**Usage**

```r
cvPost(
  nu,
  omega,
  n = 1L,
  omegaIsChol = FALSE,
  returnChol = FALSE,
  type = c("invWishart", "lkj", "separation"),
  diagXformType = c("log", "identity", "variance", "nlmixrSqrt", "nlmixrLog", "nlmixrIdentity")
)
```

**Arguments**

- `nu` Degrees of Freedom (Number of Observations) for covariance matrix simulation.
- `omega` Either the estimate of covariance matrix or the estimated standard deviations in matrix form each row forming the standard deviation simulated values.
- `n` Number of Matrices to sample. By default this is 1. This is only useful when omega is a matrix. Otherwise it is determined by the number of rows in the input omega matrix of standard deviations.
- `omegaIsChol` is an indicator of if the omega matrix is in the Cholesky decomposition. This is only used when codetype="invWishart"
- `returnChol` Return the Cholesky decomposition of the covariance matrix sample. This is only used when codetype="invWishart"
- `type` The type of covariance posterior that is being simulated. This can be:
  - `invWishart` The posterior is an inverse wishart; This allows for correlations between parameters to be modeled. All the uncertainty in the parameter is captured in the degrees of freedom parameter.
  - `lkj` The posterior separates the standard deviation estimates (modeled outside and provided in the omega argument) and the correlation estimates. The correlation estimate is simulated with the `rLKJ1()`. This simulation uses the relationship `eta=(nu-1)/2`. This is relationship based on the proof of
the relationship between the restricted LKJ-distribution and inverse wishart distribution (XXXXXX). Once the correlation posterior is calculated, the estimated standard deviations are then combined with the simulated correlation matrix to create the covariance matrix.

- **separation** Like the \( \text{lkj} \) option, this separates out the estimation of the correlation and standard deviation. Instead of using the LKJ distribution to simulate the correlation, it simulates the inverse wishart of the identity matrix and converts the result to a correlation matrix. This correlation matrix is then used with the standard deviation to calculate the simulated covariance matrix.

**diagXformType** Diagonal transformation type. These could be:

- **log** The standard deviations are log transformed, so the actual standard deviations are \( \exp(\omega) \)
- **identity** The standard deviations are not transformed. The standard deviations are not transformed; They should be positive.
- **variance** The variances are specified in the \( \omega \) matrix; They are transformed into standard deviations.
- **nlmixrSqrt** These standard deviations come from an nlmixr \( \omega \) matrix where \( \text{diag}(\text{chol}(\text{inv}(\omega))) = x^2 \)
- **nlmixrLog** These standard deviations come from an nlmixr \( \omega \) matrix where \( \text{diag}(\text{chol}(\text{solve}(\omega))) = \exp(x) \)
- **nlmixrIdentity** These standard deviations come from an nlmixr \( \omega \) matrix where \( \text{diag}(\text{chol}(\text{solve}(\omega))) = x \)

The nlmixr transformations only make sense when there is no off-diagonal correlations modeled.

**Details**

If your covariance matrix is a 1x1 matrix, this uses an scaled inverse chi-squared which is equivalent to the Inverse Wishart distribution in the uni-directional case.

In general, the separation strategy is preferred for diagonal matrices. If the dimension of the matrix is below 10, \( \text{lkj} \) is numerically faster than separation method. However, the \( \text{lkj} \) method has densities too close to zero (XXXX) when the dimension is above 10. In that case, though computationally more expensive separation method performs better.

For matrices with modeled covariances, the easiest method to use is the inverse Wishart which allows the simulation of correlation matrices (XXXX). This method is more well suited for well behaved matrices, that is the variance components are not too low or too high. When modeling non-linear mixed effects modeling matrices with too high or low variances are considered sub-optimal in describing a system. With these rules in mind, it is reasonable to use the inverse Wishart.

**Value**

a matrix (\( n=1 \)) or a list of matrices (\( n > 1 \))

**Author(s)**

Matthew L.Fidler & Wenping Wang
References


Examples

```r
## Sample a single covariance.
draw1 <- cvPost(3, matrix(c(1, .3, .3, 1), 2, 2))

## Sample 3 covariances
set.seed(42)
draw3 <- cvPost(3, matrix(c(1, .3, .3, 1), 2, 2), n = 3)

## Sample 3 covariances, but return the cholesky decomposition
set.seed(42)
draw3c <- cvPost(3, matrix(c(1, .3, .3, 1), 2, 2), n = 3, returnChol = TRUE)

## Sample 3 covariances with lognormal standard deviations via LKJ
## correlation sample
cvPost(3, sapply(1:3, function(...) { rnorm(10) }), type = "lkj")

## or return cholesky decomposition
cvPost(3, sapply(1:3, function(...) { rnorm(10) }), type = "lkj",
returnChol = TRUE)

## Sample 3 covariances with lognormal standard deviations via separation
## strategy using inverse Wishart correlation sample
cvPost(3, sapply(1:3, function(...) { rnorm(10) }), type = "separation")

## or returning the cholesky decomposition
cvPost(3, sapply(1:3, function(...) { rnorm(10) }),
type = "separation",
returnChol = TRUE)
```
erf

**Error function**

**Description**
Error function

**Usage**
erf(x)

**Arguments**
x vector or real values

**Value**
erf of x

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

**Examples**
erf(1.0)

---

et

**Event Table Function**

**Description**
Event Table Function

**Usage**
et(x, ..., envir = parent.frame())

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

## S3 method for class 'rxSolve'
et(x, ..., envir = parent.frame())

## S3 method for class 'rxParams'
et(x, ..., envir = parent.frame())
## Default S3 method:

```r
et(
  x, 
  ..., 
  time, 
  amt, 
  evid, 
  cmt, 
  ii, 
  addl, 
  ss, 
  rate, 
  dur, 
  until, 
  id, 
  amountUnits, 
  timeUnits, 
  addSampling, 
  envir = parent.frame(), 
  by = NULL, 
  length.out = NULL 
)
```

### Arguments

- **x**
  - This is the first argument supplied to the event table. This is named to allow `et` to be used in a pipe-line with arbitrary objects.
- **...**
  - Times or event tables. They can also be one of the named arguments below.
- **envir**
  - The environment in which `expr` is to be evaluated. May also be `NULL`, a list, a data frame, a pairlist or an integer as specified to `sys.call`.
- **time**
  - Time is the time of the dose or the sampling times. This can also be unspecified and is determined by the object type (list or numeric/integer).
- **amt**
  - Amount of the dose. If specified, this assumes a dosing record, instead of a sampling record.
- **evid**
  - Event ID; This can be:
    - 0: An observation. This can also be specified as `evid=obs`
    - 1: A dose observation. This can also be specified as `evid=dose`
    - 2: A non-dose event. This can also be specified as `evid=other`
    - 3: A reset event. This can also be specified as `evid=reset`
    - 4: Dose and reset event. This can also be specified as `evid=doseReset` or `evid=resetDose`

Note a reset event resets all the compartment values to zero and turns off all infusions.
Compartment name or number. If a number, this is an integer starting at 1. Negative compartments turn off a compartment. If the compartment is a name, the compartment name is changed to the correct state/compartment number before running the simulation. For a compartment named "-cmt" the compartment is turned off.

Can also specify `cmt` as `dosing.to`, `dose.to`, `doseTo`, `dosingTo`, and `state`.

When specifying a dose, this is the inter-dose interval for ss, addl and until options (described below).

The number of additional doses at an inter-dose interval after one dose.

Steady state flag; It can be one of:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This dose is not a steady state dose</td>
</tr>
<tr>
<td>1</td>
<td>This dose is a steady state dose with the between/inter-dose interval of ii</td>
</tr>
<tr>
<td>2</td>
<td>Superposition steady state</td>
</tr>
</tbody>
</table>

When ss=2 the steady state dose that uses the super-position principle to allow more complex steady states, like 10 mg in the morning and 20 mg at night, or dosing at 8 am 12 pm and 8 pm instead of every 12 hours. Since it uses the super positioning principle, it only makes sense when you know the kinetics are linear.

All other values of SS are currently invalid.

When positive, this is the rate of infusion. Otherwise:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No infusion is on this record</td>
</tr>
<tr>
<td>-1</td>
<td>Modeled rate (in rxode2: rate(cmt) =); Can be et(rate=model).</td>
</tr>
<tr>
<td>-2</td>
<td>Modeled duration (in rxode2: dur(cmt) =); Can be et(dur=model) or et(rate=dur).</td>
</tr>
</tbody>
</table>

When a modeled bioavailability is applied to positive rates (rate > 0), the duration of infusion is changed. This is because the data specify the rate and amount, the only thing that modeled bioavailability can affect is duration.

If instead you want the modeled bioavailability to increase the rate of infusion instead of the duration of infusion, specify the dur instead or model the duration with rate=2.

Duration of infusion. When amt and dur are specified the rate is calculated from the two data items. When dur is specified instead of rate, the bioavailability changes will increase rate instead of duration.

This is the time until the dosing should end. It can be an easier way to figure out how many additional doses are needed over your sampling period.

A integer vector of IDs to add or remove from the event table. If the event table
is identical for each ID, then you may expand it to include all the IDs in this vector. All the negative IDs in this vector will be removed.

amountUnits  The units for the dosing records (amt)
timeUnits  The units for the time records (time)
addSampling  This is a boolean indicating if a sampling time should be added at the same time as a dosing time. By default this is FALSE.
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.

Value
A new event table

Author(s)
Matthew L Fidler, Wenping Wang

References

See Also

eventoTable, add.sampling, add.dosing, et, etRep, etRbind, rxode2

Examples

library(rxode2)
library(units)

## Model from rxode2 tutorial
mod1 <- rxode2(
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) =-KA*depot;
et

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)
plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>%
et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)
plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>%
et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- seq(qd, set_units(1,"weeks"), qd) %>%
add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)
plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

qd <-et(timeUnits = "hr") %>%
et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- etRep(qd, times=4, wait=set_units(1, "weeks")) %>%
  add.sampling(set_units(seq(0, 12.5, by=0.005), weeks))
repCycle4 <- rxSolve(mod1, et)
plot(repCycle4, C2)

---

etExpand

Expand additional doses

Description

Expand additional doses

Usage

etExpand(et)

Arguments

et  Event table to expand additional doses for.

Value

New event table with addl doses expanded

Author(s)

Matthew Fidler

Examples

ev <- et(amt = 3, ii = 24, until = 240)
print(ev)
etExpand(ev)  # expands event table, but doesn't modify it
print(ev)
ev$expand()  # expands the current event table and saves it in ev
etRbind: Combining event tables

Description
Combining event tables

Usage
etRbind(
  ..., 
  samples = c("use", "clear"),
  waitII = c("smart", "+ii"),
  id = c("merge", "unique")
)

## S3 method for class 'rxEt'
rbind(..., deparse.level = 1)

Arguments

... The event tables and optionally time between event tables, called waiting times in this help document.

samples How to handle samples when repeating an event table. The options are:
  • "clear" Clear sampling records before combining the datasets
  • "use" Use the sampling records when combining the datasets

waitII This determines how waiting times between events are handled. The options are:
  • "smart" This "smart" handling of waiting times is the default option. In this case, if the waiting time is above the last observed inter-dose interval in the first combined event table, then the actual time between doses is given by the wait time. If it is smaller than the last observed inter-dose interval, the time between event tables is given by the inter-dose interval + the waiting time between event tables.
  • "+ii" In this case, the wait time is added to the inter-dose interval no matter the length of the wait time or inter-dose interval

id This is how rbind will handle IDs. There are two different types of options:
  • merge with id="merge", the IDs are merged together, overlapping IDs would be merged into a single event table.
  • unique with id="unique", the IDs will be renumbered so that the IDs in all the event tables are not overlapping.

deparse.level The deparse.level of a traditional rbind is ignored.
Value

An event table

Author(s)

Matthew L Fidler
Matthew L Fidler, Wenping Wang

References


See Also

eventTable, add.sampling, add.dosing, et, etRep, etRbind, rxode2

Examples

```r
library(rxode2)
library(units)

## Model from rxode2 tutorial
mod1 <-rxode2({
    KA=2.94E-01;
    CL=1.86E+01;
    V2=4.02E+01;
    Q=1.05E+01;
    V3=2.97E+02;
    Kin=1;
    Kout=1;
    EC50=200;
    C2 = centr/V2;
    C3 = peri/V3;
    d/dt(depot) =-KA*depot;
    d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
    d/dt(peri) = Q*C2 - Q*C3;
    d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

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

## 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)
plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")
et <- seq(infusion, qd)
infusionQd <- rxSolve(mod1, et)
plot(infusionQd, C2)

## 2wk-on, 1wk-off
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- seq(qd, set_units(1,"weeks"), qd) %>%
    add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))
wkOnOff <- rxSolve(mod1, et)
plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
    add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))
repCycle4 <- rxSolve(mod1, et)
plot(repCycle4, C2)
etRep

Repeat an rxode2 event table

Description

Repeat an rxode2 event table

Usage

etRep(
  x,
  times = 1,
  length.out = NA,
  each = NA,
  n = NULL,
  wait = 0,
  id = integer(0),
  samples = c("clear", "use"),
  waitII = c("smart", "+ii"),
  ii = 24
)

## S3 method for class 'rxEt'
rep(x, ...)

Arguments

x An rxode2 event table

times Number of times to repeat the event table

length.out Invalid with rxode2 event tables, will throw an error if used.

each Invalid with rxode2 event tables, will throw an error if used.

n The number of times to repeat the event table. Overrides times.

wait Waiting time between each repeated event table. By default there is no waiting, or wait=0

id A integer vector of IDs to add or remove from the event table. If the event table is identical for each ID, then you may expand it to include all the IDs in this vector. All the negative IDs in this vector will be removed.

samples How to handle samples when repeating an event table. The options are:

  • "clear" Clear sampling records before combining the datasets
  • "use" Use the sampling records when combining the datasets

waitII This determines how waiting times between events are handled. The options are:
"smart" This "smart" handling of waiting times is the default option. In this case, if the waiting time is above the last observed inter-dose interval in the first combined event table, then the actual time between doses is given by the wait time. If it is smaller than the last observed inter-dose interval, the time between event tables is given by the inter-dose interval + the waiting time between event tables.

"+ii" In this case, the wait time is added to the inter-dose interval no matter the length of the wait time or inter-dose interval

When specifying a dose, this is the inter-dose interval for ss, add1 and until options (described below).

Times or event tables. They can also be one of the named arguments below.

Value
An event table

Author(s)
Matthew L Fidler, Wenping Wang

References

See Also
eventTable, add.sampling, add.dosing, et, etRep, etRbind, rxode2

Examples

library(rxode2)
library(units)

## Model from rxode2 tutorial
mod1 <-rxode2({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
d/dt(depot) = -KA*depot;
d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
d/dt(peri) = Q*C2 - Q*C3;
d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;

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

## 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)
plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>%
  et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")
et <- seq(infusion, qd)
infusionQd <- rxSolve(mod1, et)
plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>%
  et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- seq(qd, set_units(1, "weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5, by=0.005), weeks))
wkOnOff <- rxSolve(mod1, et)
plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function
etSeq <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

plot(repCycle4, C2)

---

### etSeq

**Sequence of event tables**

**Description**

This combines a sequence of event tables.

**Usage**

etSeq(..., samples = c("clear", "use"), waitII = c("smart", "+ii"), ii = 24)

## S3 method for class 'rxEt'

seq(...)

**Arguments**

- **...** The event tables and optionally time between event tables, called waiting times in this help document.
- **samples** How to handle samples when repeating an event table. The options are:
  - "clear" Clear sampling records before combining the datasets
  - "use" Use the sampling records when combining the datasets
- **waitII** This determines how waiting times between events are handled. The options are:
  - "smart" This "smart" handling of waiting times is the default option. In this case, if the waiting time is above the last observed inter-dose interval in the first combined event table, then the actual time between doses is given by the wait time. If it is smaller than the last observed inter-dose interval, the time between event tables is given by the inter-dose interval + the waiting time between event tables.
  - "+ii" In this case, the wait time is added to the inter-dose interval no matter the length of the wait time or inter-dose interval
- **ii** If there was no inter-dose intervals found in the event table, assume that the interdose interval is given by this ii value. By default this is 24.
Details
This sequences all the event tables in added in the argument list . . . . By default when combining the event tables the offset is at least by the last inter-dose interval in the prior event table (or ii). If you separate any of the event tables by a number, the event tables will be separated at least the wait time defined by that number or the last inter-dose interval.

Value
An event table

Author(s)
Matthew L Fidler, Wenping Wang

References

See Also
eventTable, add.sampling, add.dosing, et, etRep, etRbind, rxode2

Examples

library(rxode2)
library(units)

## Model from rxode2 tutorial
mod1 <-rxode2({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) =-KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

## These are making the more complex regimens of the rxode2 tutorial
## 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)
plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")
et <- seq(infusion, qd)
infusionQd <- rxSolve(mod1, et)
plot(infusionQd, C2)

## 2wk-on, 1wk-off
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- seq(qd, set_units(1,"weeks"), qd) %>%
    add.sampling(set_units(seq(0, 5.5,by=0.005), weeks))
wkOnOff <- rxSolve(mod1, et)
plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function
qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")
et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
    add.sampling(set_units(seq(0, 12.5,by=0.005), weeks))
repCycle4 <- rxSolve(mod1, et)
plot(repCycle4, C2)

---

**eventTable**

Create an event table object

---

**Description**

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

**Usage**

`eventTable(amount.units = NA, time.units = NA)`

**Arguments**

- `amount.units`: string denoting the amount dosing units, e.g., “mg”, “ug”. Default to NA to denote unspecified units. It could also be a solved rxode2 object. In that case, `eventTable(obj)` returns the eventTable that was used to solve the rxode2 object.
- `time.units`: string denoting the time units, e.g., “hours”, “days”. Default to “hours”.

An **eventTable** is an object that consists of a data.frame storing ordered time-stamped events of an (unspecified) PK/PD dynamic system, units (strings) for dosing and time records, plus a list of functions to add and extract event records. Currently, events can be of two types: dosing events that represent inputs to the system and sampling time events that represent observations of the system with ‘amount.units’ and ‘time.units’, respectively.

**Value**

A modified data.frame with the following accessible functions:

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

expand() Expands the event table for multi-subject solving. This is done by qd$expand(400)
for a 400 subject data expansion

Author(s)
Matthew Fidler, Melissa Hallow and Wenping Wang

See Also
et(), rxode2()

Examples

```r
# create dosing and observation (sampling) events
# QD 50mg dosing, 5 days followed by 25mg 5 days
#
qd <- eventTable(amount.units = "mg", time.units = "days")
qd$dadd.dosing(dose = 50, nbr.doses = 5, dosing.interval = 1, do.sampling = FALSE)
# sample the system's drug amounts hourly the first day, then every 12 hours
# for the next 4 days
qd$dadd.sampling(seq(from = 0, to = 1, by = 1 / 24))
qd$dadd.sampling(seq(from = 1, to = 5, by = 12 / 24))
# print(qd$get.dosing()) # table of dosing records
print(qd$get.nobs()) # number of observation (not dosing) records
#
# BID dosing, 5 days
# bid <- eventTable("mg", "days") # only dosing
bid$dadd.dosing( 
  dose = 10000, nbr.doses = 2 * 5, 
  dosing.interval = 12, do.sampling = FALSE 
)
#
# Use the copy() method to create a copy (clone) of an existing
# event table (simple assignments just create a new reference to
# the same event table object (closure)).
#
bid.ext <- bid$copy() # three-day extension for a 2nd cohort
bid.ext$dadd.dosing( 
  dose = 5000, nbr.doses = 2 * 3, 
  start.time = 120, dosing.interval = 12, do.sampling = FALSE 
)
#
# You can also use the Piping operator to create a table
qd2 <- eventTable(amount.units = "mg", time.units = "days") %>%
  add.dosing(dose = 50, nbr.doses = 5, dosing.interval = 1, do.sampling = FALSE) %>%
  add.sampling(seq(from = 0, to = 1, by = 1 / 24)) %>%
```
```
add.sampling(seq(from = 1, to = 5, by = 12 / 24)) # print(qd2$get.dosing()) # table of dosing records
print(qd2$get.nobs()) # number of observation (not dosing) records

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

---

**gammap**

Gammap: normalized lower incomplete gamma function

---

**Description**

This is the gamma_p from the boost library

**Usage**

gammap(a, z)

**Arguments**

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

**Details**

The gamma p function is given by:

gammap = lowergamma(a, z)/gamma(a)

**Value**

gammap results

**Author(s)**

Matthew L. Fidler

**Examples**

```r
gammap(1, 3)
gammap(1:3, 3)
gammap(1, 1:3)
```


**gammapDer**  

**gammapDer: derivative of gammap**

**Description**

This is the gamma\_p\_derivative from the boost library

**Usage**

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**

lowergamma results

**Author(s)**

Matthew L. Fidler

**Examples**

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

---

**gammapInv**  

**gammapInv and gammapInva: Inverses of normalized gammap function**

**Description**

gammapInv and gammapInva: Inverses of normalized gammap function

**Usage**

```r
  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:
\[ p = \text{gammapp}(a, x) \]
The 'gammapiInv' function returns a value 'x' that satisfies the equation above.
The 'gammapiInva' function returns a value 'q' that satisfies the equation above.

**NOTE**: gammapiInva is slow

Value

inverse gammapi results

Author(s)

Matthew L. Fidler

Examples

```r
gammapiInv(1:3, 0.5)
gammapiInv(1, 1:3 / 3.1)
gammapiInv(1:3, 1:3 / 3.1)
gammapiInva(1:3, 1:3 / 3.1)
```

---

**gammaq**

*Gammaq: normalized upper incomplete gamma function*

Description

This is the gamma_q from the boost library

Usage

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

Arguments

- **a**: The numeric 'a' parameter in the normalized upper incomplete gamma
- **z**: The numeric 'z' parameter in the normalized upper incomplete gamma
gammaqInv

Details
The gamma q function is given by:
gammaq = uppergamma(a, z)/gamma(a)

Value
gammaq results

Author(s)
Matthew L. Fidler

Examples

gammaq(1, 3)
gammaq(1:3, 3)
gammaq(1, 1:3)

gammaqInv and gammaqInva: Inverses of normalized gammaq function

Description
gammaqInv and gammaqInva: Inverses of normalized gammaq function

Usage

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

Arguments

<table>
<thead>
<tr>
<th>a</th>
<th>The numeric ‘a’ parameter in the upper incomplete gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>The numeric ‘q’ parameter in the upper incomplete gamma</td>
</tr>
<tr>
<td>x</td>
<td>The numeric ‘x’ parameter in the upper incomplete gamma</td>
</tr>
</tbody>
</table>

Details
With the equation:
q = gammaq(a, x)
The ‘gammaqInv’ function returns a value ‘x’ that satisfies the equation above
The ‘gammaqInva’ function returns a value ‘a’ that satisfies the equation above
NOTE: gammaqInva is slow
Value
inverse gammaq results

Author(s)
Matthew L. Fidler

Examples

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

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

Usage

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

Arguments

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

A PK/PD model is defined using \texttt{rxode2()}, and a set of parameters and initial values are defined. Then the appropriate R scripts for the shiny’s user interface \texttt{ui.R} and the server logic \texttt{server.R} are created in the directory \texttt{appDir}.

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

\begin{verbatim}
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;
\end{verbatim}

This can be changed by the \texttt{ODE.config} parameter.

To launch the shiny app, simply issue the \texttt{runApp(appDir)} R command.

**Value**

None, these functions are used for their side effects.

**Note**

These functions create a simple, but working example of a dosing regimen simulation web application. Users may want to modify the code to experiment creating shiny applications for their specific \texttt{rxode2} models.

**See Also**

\texttt{rxode2()}, \texttt{eventTable()}, and the package \texttt{shiny} (https://shiny.rstudio.com).

**Examples**

```r
# remove myapp when the example is complete
on.exit(unlink("myapp", recursive = TRUE, force = TRUE))
# create the shiny app example (template)
genShinyApp.template(appDir = "myapp")
# run the shiny app
if (requireNamespace("shiny", quietly=TRUE)) {
  library(shiny)
  # runApp("myapp") # Won't launch in environments without browsers
}
```
getRxThreads  
Get/Set the number of threads that rxode2 uses

**Description**

Get/Set the number of threads that rxode2 uses

**Usage**

```r
getRxThreads(VERBOSE = FALSE)
```

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

```r
rxCores(VERBOSE = FALSE)
```

**Arguments**

- **verbose**  
  Display the value of relevant OpenMP settings
- **threads**  
  NULL (default) rereads environment variables. 0 means to use all logical CPUs available. Otherwise a number >= 1
- **percent**  
  If provided it should be a number between 2 and 100; the percentage of logical CPUs to use. By default on startup, 50 percent.
- **throttle**  
  2 (default) means that, roughly speaking, a single thread will be used when number subjects solved for is <=2, 2 threads when the number of all points is <=4, etc. The throttle is to speed up small data tasks (especially when repeated many times) by not incurring the overhead of managing multiple threads.

The throttle will also suppress sorting which ID will be solved first when there are (nsubject solved)*throttle <= nthreads. In rxode2 this sorting occurs to minimize the time for waiting for another thread to finish. If the last item solved 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
The ini block controls initial conditions for 'theta' (fixed effects), 'omega' (random effects), and 'sigma' (residual error) elements of the model.

Usage

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

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

Arguments

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

Details

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

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

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

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

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

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

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

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

logit

logit and inverse logit (expit) functions

Description

logit and inverse logit (expit) functions
Usage

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

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

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

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

Arguments

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

Details

logit is given by:
\[ \text{logit}(p) = -\log\left(\frac{1}{p} - 1\right) \]

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 logitNormInfo() gives the mean, variance and coefficient of variability on the untransformed scale.

Value

values from logit and expit

Examples

logit(0.25)

expit(-1.09)

logitNormInfo(logit(0.25), sd = 0.1)

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

\[ \text{lowergamma}(a, z) = \int_0^z t^{a-1} e^{-t} dt \]

Value

lowergamma results

Author(s)

Matthew L. Fidler

Examples

lowergamma(1, 3)
lowergamma(1:3, 3)
lowergamma(1, 1:3)
Description
Model block for rxode2/nlmixr models

Usage

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

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

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

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

Arguments

x model expression
...
Other arguments
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 (default)
  • FALSE when the lines are replaced in the model
  • NA is when the lines are pre-pended to the model instead of replaced
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).
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

---

**phi**

*Cumulative distribution of standard normal*

**Description**

Cumulative distribution of standard normal

**Usage**

\[ \text{phi}(q) \]

**Arguments**

- \( q \) vector of quantiles.

**Value**

cumulative distribution of standard normal distribution

**Author(s)**

Matthew Fidler

**Examples**

```r
# phi is equivalent to pnorm(x)
phi(3)
```

```r
# See
pnorm(3)
```

# This is provided for NONMEM-like compatibility in rxode2 models
plot.rxSolve

Plot rxode2 objects

Description

Plot rxode2 objects

Usage

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

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

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

Arguments

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

Value

A ggplot2 object

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)

rinvchisq Scaled Inverse Chi Squared distribution

Description
Scaled Inverse Chi Squared distribution

Usage
rinvchisq(n = 1L, nu = 1, scale = 1)

Arguments
n Number of random samples
nu degrees of freedom of inverse chi square
scale Scale of inverse chi squared distribution (default is 1).

Value
a vector of inverse chi squared deviates.
Examples

rinvchisq(3, 4, 1) ## Scale = 1, degrees of freedom = 4
rinvchisq(2, 4, 2) ## Scale = 2, degrees of freedom = 4

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

rxAppendModel(model1, model2)

Arguments

model1  rxUi model 1
model2  rxUi model 2

Value

New model with both models appended together

Author(s)

Matthew L. Fidler

Examples

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
    d/dt(depot) = -ka * depot
    d/dt(center) = ka * depot - cl / v * center
    cp = center / v
    cp ~ add(add.sd)
  })
}

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

rxAppendModel(ocmt %>% model(ceff=cp,append=TRUE), idr)

---

###(rxAssignControlValue  Assign Control Variable)

**Description**

Assign Control Variable

**Usage**

rxAssignControlValue(ui, option, value)

**Arguments**

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

**Value**

Nothing; called for the side effects

**Author(s)**

Matthew L. Fidler
### rxAssignPtr

Assign pointer based on model variables

**Description**

Assign pointer based on model variables

**Usage**

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

**Arguments**

- `object`:
  - `rxode2` family of objects

**Value**

- nothing, called for side effects

### rxbeta

Simulate beta 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
rxbeta(shape1, shape2, n = 1L, ncores = 1L)
```

**Arguments**

- `shape1`:
  - non-negative parameters of the Beta distribution.
- `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` uses the vander-corput generator
**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().

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**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 <- rxode2(
  function()
    a <- rxbeta(2, 2)

  )

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

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

---

**rxbinom** Simulate Binomial variable from threefry generator

**Description**

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

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

Arguments

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

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()}

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

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 ‘\texttt{rxbinom}’ directly in the model
rx <- rxode2({
  a <- rxbinom(1, 0.5)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
\end{verbatim}
Described Cauchy variable from threefry generator

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

**Arguments**
- **location**: location and scale parameters.
- **scale**: location and scale parameters.
- **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

**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().

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

Cauchy random deviates
Examples

```r
## Use threefry engine
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
rx <- rxode2({
  a <- rxcauchy(2)
})
et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```

---

**rxBindStudyIndividual**

Bind the study parameters and individual parameters

**Description**

Bind the study parameters and individual parameters

**Usage**

```r
rxBindStudyIndividual(studyParameters, individualParameters)
```

**Arguments**

- **studyParameters**
  These are the study parameters, often can be generated by sampling from a population. This can be either a matrix or a data frame

- **individualParameters**
  A data frame of individual parameters

**Value**

Data frame that can be used in rxode2 simulations
Author(s)
Matthew Fidler

Examples

# Function for converting coefficient of covariance into a variance
logCv <- function(x){log((x/100)^2+1)}

set.seed(32)
nSub <- 100
nStud <- 10

# define theta
theta <- c(lka=log(0.5), # log ka
          lCl=log(5), # log Cl
          lV=log(300) # log V
        )

# define theta Matrix
thetaMat <- lotri(lCl ~ logCv(5),
                 lV ~ logCv(5),
                 lka ~ logCv(5))

nev <- nSub*nStud

ev1 <- data.frame(COV1=rnorm(nev,50,30),COV2=rnorm(nev,75,10),
                   COV3=sample(c(1.0,2.0),nev,replace=TRUE))

tmat <- rxRmvn(nStud, theta[[dimnames(thetaMat)[[1]]]], thetaMat)

rxCbindStudyIndividual(tmat, ev1)

---

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/. 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

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 threerefry sitmo generator; **rxnormV** uses the vander-corput generator

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()`.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

Value

Chi squared random deviates

Examples

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

rxchisq(1)
```

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

This cleans up any rxode2 loaded DLLs

**Usage**

```r
rxClean(wd)
```

**Arguments**

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

This unloads all rxode2 anonymous dlls.

**Value**

TRUE if successful

**Author(s)**

Matthew L. Fidler

---

**rxCompile**

`Compile a model if needed`

**Description**

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

**Usage**

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

### S3 method for class 'rxModelVars'

```r
rxCompile(
```

```r
```
model,
dir = NULL,
prefix = NULL,
force = FALSE,
modName = NULL,
package = NULL,
...)

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

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

## S3 method for class 'rxode2'
rxCompile(model, ...)

Arguments

model This is the ODE model specification. It can be:
  • a string containing the set of ordinary differential equations (ODE) and
    other expressions defining the changes in the dynamic system.
  • a file name where the ODE system equation is contained
An ODE expression enclosed in `{}`
(see also the filename argument). For details, see the sections “Details” and
rxode2 Syntax below.
dir This is the model directory where the C file will be stored for compiling.
If unspecified, the C code is stored in a temporary directory,
then the model is compiled and moved to the current directory.
Afterwards the C code is removed.

If specified, the C code is stored in the specified directory
and then compiled in that directory. The C code is not removed
after the DLL is created in the same directory. This can be
useful to debug the c-code outputs.

prefix is a string indicating the prefix to use in the C based functions. If missing, it is
calculated based on file name, or md5 of parsed model.
force is a boolean stating if the (re)compile should be forced if rxode2 detects that the models are the same as already generated.

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

package Package name for pre-compiled binaries.

Other arguments sent to the rxTrans() function.

Value

An rxDll object that has the following components

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

Author(s)

Matthew L. Fidler

See Also

rxode2()

rxControlUpdateSens This updates the tolerances based on the sensitivity equations

Description

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

Usage

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

Arguments

rxControl Input list or rxControl type of list
sensCmt Number of sensitivity compartments
ncmt Number of compartments
Value

Updated rxControl where $atol$, $rtol$, $ssAtol$ $ssRtol$ 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
Description
Add to `rxode2`’s derivative tables

Usage
```
rxD(name, derivatives)
```

Arguments

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

Value
nothing

Author(s)
Matthew Fidler

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

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

**Author(s)**

Matthew L. Fidler

---

### rxDerived

*Calculate derived parameters for the 1-, 2-, and 3-compartment linear models.*

**Description**

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

**Usage**

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

**Arguments**

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

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

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

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

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

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

Author(s)

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

References

Shafer S. L. CONVERT.XLS
Examples

```r
## Note that rxode2 parses the names to figure out the best PK parameter
params <- rxDerived(cl = 29.4, v = 23.4, Vp = 114, vp2 = 4614, q = 270, q2 = 73)
## That is why this gives the same results as the value before
params <- rxDerived(CL = 29.4, V1 = 23.4, V2 = 114, V3 = 4614, Q2 = 270, Q3 = 73)
## You may also use micro-constants alpha/beta etc.
params <- rxDerived(k12 = 0.1, k21 = 0.2, k13 = 0.3, k31 = 0.4, kel = 10, v = 10)
## or you can mix vectors and scalars
params <- rxDerived(CL = 29.4, V = 1:3)
## If you want, you can round to a number of significant digits
## with the `digits` argument:
params <- rxDerived(CL = 29.4, V = 1:3, digits = 2)
```

**rxDfdy**

_Jacobian and parameter derivatives_

**Description**

Return Jacobain and parameter derivatives

**Usage**

`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()`
Description
This is to make an EVID more readable by non pharmacometricians. It displays what each means and allows it to be displayed in a tibble.

Usage
rxEvid(x)
as.rxEvid(x)

## S3 method for class 'rxEvid'
c(x, ...)

## S3 method for class 'rxEvid'
x[...]

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

## S3 method for class 'rxEvid'
x[[...]]

## S3 method for class 'rxRateDur'
c(x, ...)

## S3 method for class 'rxEvid'
format(x, ...)

## S3 method for class 'rxRateDur'
format(x, ...)

## S3 method for class 'rxEvid'
print(x, ...)

Arguments
x Item to be converted to a rxode2 EVID specification.

... Other parameters

Value
rxEvid specification
Examples

```r
taxEvid(1:7)
```

---

**rxexp**

*Simulate exponential variable from threefry generator*

Description

Care should be taken with this method not to encounter the birthday problem, described [https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/](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** uses the vandercorput generator

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()`

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

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
rxexp(1)
```

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

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

---

### rxf

Simulate F variable from threefry generator

---

Description

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

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>df1</td>
<td>degrees of freedom. Inf is allowed.</td>
</tr>
<tr>
<td>df2</td>
<td>degrees of freedom. Inf is allowed.</td>
</tr>
<tr>
<td>n</td>
<td>number of observations. If length(n) &gt; 1, the length is taken to be the number required.</td>
</tr>
<tr>
<td>ncores</td>
<td>Number of cores for the simulation</td>
</tr>
</tbody>
</table>

`rnxnorm` simulates using the threefry sitmo generator; `rnxnormV` uses the vandercorput generator.
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()

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

Value

f random deviates

Examples

```r
## Use threefry engine

rxf(0.5, 0.5, n = 10) # with rxf you have to explicitly state n
rxf(5, 1, n = 10, ncores = 2) # You can parallelize the simulation using openMP
rxf(1, 3)

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

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

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

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

---

**rxFun**

Add user function to rxode2

Description

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

\texttt{rxFun(name, args, cCode)}

\texttt{rXRmFun(name)}

Arguments

\begin{itemize}
\item \textbf{name} \hspace{1cm} This gives the name of the user function
\item \textbf{args} \hspace{1cm} This gives the arguments of the user function
\item \textbf{cCode} \hspace{1cm} This is the C-code for the new function
\end{itemize}

Value

\textbf{nothing}

Author(s)

Matthew L. Fidler

Examples

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

## Note for this approach to work, it cannot interfere with C
## function names or reserved rxode2 speical terms. Therefore
## f(x) would not work since f is an alias for bioaviability.
fun <- "
double fun(double a, double b, double c) {
    return a*a+b*a+c;
}
"  # C-code for function

rxFun("fun", c("a", "b", "c"), fun)  # Added function

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

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

## Of course, you could specify the derivative table manually
rxD("fun", list(
    function(a, b, c) {

\end{verbatim}
```r
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")
```

---

**rxgamma**

*Simulate gamma variable from threefry generator*

---

**Description**

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

---

**Usage**

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

---

**Arguments**

- `shape`: shape and scale parameters. Must be positive, scale strictly.
- `rate`: an alternative way to specify the scale.
- `scale`: shape and scale parameters. Must be positive, scale strictly.
- `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` uses the vander-corput generator
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().

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

Value

gamma random deviates

Examples

## Use threefry engine

```r
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

```r
rx <- rxode2(
  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/. 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 <= 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 uses the vander-corput generator

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()

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

Value

geometric random deviates

Examples

## Use threefry engine

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

rxgeom(0.75)

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

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

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

s <- rxSolve(rx, et)
**rxGetControl**

**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

rxGetrxode2(obj)

Arguments

obj  rxode2 family of objects

Value

rxode2 model
rxGetSeed

Get the rxode2 seed

Description
Get the rxode2 seed

Usage
rxGetSeed()

Value
rxode2 seed state or -1 when the seed isn’t set

See Also
rxSetSeed, rxWithSeed, rxWithPreserveSeed

Examples

# without setting seed
rxGetSeed()

# Now set the seed
rxSetSeed(42)
rxGetSeed()

rxnorm()

rxGetSeed()

# don’t use the rxode2 seed again
rxSetSeed(-1)
rxGetSeed()

rxnorm()

rxGetSeed()
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'
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
**rxIndLinStrategy**

**Author(s)**

Matthew Fidler

---

**rxIndLinStrategy**  
*This sets the inductive linearization strategy for matrix building*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

Nothing

**Author(s)**

Matthew L. Fidler

---

**rxIndLin**  
*Inductive linearization solver*

---

**Description**

Inductive linearization solver
**Arguments**

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

**Value**

Returns a status for solving
1 = Successful solve
-1 = Maximum number of iterations reached when doing inductive linearization

---

**rxInv**

*Invert matrix using RcppArmadillo.*

**Description**

Invert matrix using RcppArmadillo.

**Usage**

`rxInv(matrix)`

**Arguments**

- **matrix** matrix to be inverted.

**Value**

inverse or pseudo inverse of matrix.
**rxIsCurrent**

Checks if the rxode2 object was built with the current build

**Description**

Checks if the rxode2 object was built with the current build

**Usage**

```r
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**

```r
rxLhs(obj)
```

**Arguments**

- `obj`: rxode2 family of objects

**Value**

a character vector listing the calculated parameters

**Author(s)**

Matthew L. Fidler

**See Also**

- `rxode2`

Other Query model information: `rxDfdy()`, `rxInits()`, `rxModelVars()`, `rxParams()`, `rxState()`
**rxLock**

*Lock/unlocking of rxode2 dll file*

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

**Usage**

rxLock(obj)

rxUnlock(obj)

**Arguments**

obj A rxode2 family of objects

**Value**
nothing; called for side effects

---

**rxNorm**

*Get the normalized model*

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

**Usage**

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

**Arguments**

obj rxode2 family of objects

collection Character string of a logical condition to use for subsetting the normalized model. When missing, and a condition is not set via rxCondition, return the whole code with all the conditional settings intact. When a condition is set with rxCondition, use that condition.

removeInis A boolean indicating if parameter initialization will be removed from the model

removeJac A boolean indicating if the Jacobians will be removed.

removeSens A boolean indicating if the sensitivities will be removed.

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

**Simulate random normal variable from threefry/vandercorput generator**

**Author(s)**

Matthew L. Fidler

**Usage**

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

```r
rxnormV(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* uses the vandercorput generator

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

## Use vandercorput generator

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

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

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

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

---

**rxode2**

*Create an ODE-based model specification*

**Description**

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

**Usage**

```r
rxode2(
  model,
  modName = basename(wd),
  wd = getwd(),
  filename = NULL,
  extraC = NULL,
  debug = FALSE,
  calcJac = NULL,
  calcSens = NULL,
  collapseModel = FALSE,
  package = NULL,
  ...
  linCmtSens = c("linCmtA", "linCmtB", "linCmtC"),
  indLin = FALSE,
  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.

**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.
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.

linCmtSens
The method to calculate the linCmt() solutions

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

verbose
When TRUE be verbose with the linear compartmental model

Details

The Rx in the name rxode2 is meant to suggest the abbreviation Rx for a medical prescription, and thus to suggest the package emphasis on pharmacometrics modeling, including pharmacokinetics (PK), pharmacodynamics (PD), disease progression, drug-disease modeling, etc.

The ODE-based model specification may be coded inside a character string or in a text file, see Section rxode2 Syntax below for coding details. An internal rxode2 compilation manager object translates the ODE system into C, compiles it, and dynamically loads the object code into the current R session. The call to rxode2 produces an object of class rxode2 which consists of a list-like structure (environment) with various member functions (see Section Value below).

For evaluating rxode2 models, two types of inputs may be provided: a required set of time points for querying the state of the ODE system and an optional set of doses (input amounts). These inputs are combined into a single event table object created with the function eventTable() or et().

An rxode2 model specification consists of one or more statements optionally terminated by semicolons ; and optional comments (comments are delimited by # and an end-of-line).

A block of statements is a set of statements delimited by curly braces, { ... }.

Statements can be either assignments, conditional if/else if/else, while loops (can be exited by break), special statements, or printing statements (for debugging/testing).

Assignment statements can be:
• **simple** assignments, where the left hand is an identifier (i.e., variable)

• special **time-derivative** assignments, where the left hand specifies the change of the amount in the corresponding state variable (compartment) with respect to time e.g., \( \frac{d}{dt}(\text{depot}) \):

• special **initial-condition** assignments where the left hand specifies the compartment of the initial condition being specified, e.g. \( \text{depot}(0) = 0 \)

• special model event changes including bioavailability \((f(\text{depot})=1)\), lag time \((\text{lag}(\text{depot})=0)\), modeled rate \((\text{rate}(\text{depot})=2)\) and modeled duration \((\text{dur}(\text{depot})=2)\). 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 \( \text{mtime(var)}=\text{time} \)

• special **Jacobian-derivative** assignments, where the left hand specifies the change in the compartment ode with respect to a variable. For example, if \( \frac{d}{dt}(y) = dy \), then a Jacobian for this compartment can be specified as \( \frac{df(y)}{dy} \). 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 \( =, \leftarrow \) or \( \sim \).

When assigning with the \( \sim \) operator, the **simple assignments** and **time-derivative** assignments will not be output.

Special statements can be:

• **Compartment declaration statements**, which can change the default dosing compartment and the assumed compartment number(s) as well as add extra compartment names at the end (useful for multiple-endpoint nlmixr models); These are specified by \( \text{cmt(compartmentName)} \)

• **Parameter declaration statements**, which can make sure the input parameters are in a certain order instead of ordering the parameters by the order they are parsed. This is useful for keeping the parameter order the same when using 2 different ODE models. These are specified by \( \text{param(par1, par2,\ldots)} \)

An example model is shown below:

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

# time-derivative assignment
\frac{d}{dt}(\text{centr}) = F*KA*\text{depot} - CL*C2 - Q*C2 + Q*C3;
```

Expressions in assignment and if statements can be numeric or logical.

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

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

The 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, \( t \) (time), \( t_{\text{last}} \) (last time point), and \( \text{podo} \) (oral dose, in the undocumented case of absorption transit models).
• Special constants like \( \pi \) or R’s predefined constants.
• Model parameters (e.g., \( \text{ka} \) rate of absorption, \( \text{CL} \) clearance, etc.)
• Others, as created by assignments as part of the model specification; these are referred as \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:

- \texttt{cmt}
- \texttt{dvid}
- \texttt{add1}
- \texttt{ss}
- \texttt{rate}
- \texttt{id}

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

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

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

**Logical Operators:**

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

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

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

**Value**

An object (environment) of class \texttt{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 `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., `rm(m1)`.
* `run` deprecated, use `solve`.
* `get.index` deprecated.
* `getObj` internal (not user callable) function.

### Author(s)
Melissa Hallow, Wenping Wang and Matthew Fidler

### References


### See Also
`eventTable()`, `et()`, `add.sampling()`, `add.dosing()`

### Examples

```r
# Step 1 - Create a model specification
ode <- "
# A 4-compartment model, 3 PK and a PD (effect) compartment
# (notice state variable names 'depot', 'centr', 'peri', 'eff')
C2 = centr/V2;
C3 = peri/V3;
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
```

\[
\frac{d}{dt}(\text{peri}) = Q \cdot C2 - Q \cdot C3; \\
\frac{d}{dt}(\text{eff}) = \text{Kin} - \text{Kout} \cdot (1 - C2/(\text{EC50} + C2)) \cdot \text{eff};
\]

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

# Step 2 - Create the model input as an EventTable,  
# including dosing and observation (sampling) events  
# QD (once daily) dosing for 5 days.
qd <- eventTable(amount.units = "ug", time.units = "hours")  
qd$add.dosing(dose = 10000, nbr.doses = 5, dosing.interval = 24)  

# Sample the system hourly during the first day, every 8 hours  
# then after
qd$add.sampling(0:24)  
qd$add.sampling(seq(from = 24 + 8, to = 5 * 24, by = 8))

# Step 3 - set starting parameter estimates and initial  
# values of the state
theta <-  
c(
  KA = .291, CL = 18.6, 
  V2 = 40.2, Q = 10.5, V3 = 297.0, 
  Kin = 1.0, Kout = 1.0, EC50 = 200.0
)

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

# Step 4 - Fit the model to the data
qd.cp <- m1$solve(theta, events = qd, inits)

head(qd.cp)

# This returns a matrix. Note that you can also  
# solve using name initial values. For example:
inits <- c(eff = 1)  
qd.cp <- solve(m1, theta, events = qd, inits)  
print(qd.cp)

plot(qd.cp)

# You can also directly simulate from a nlmixr model
f <- function() {
inli({
  KA <- .291
  CL <- 18.6
})
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 rxd2 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)
```
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,
  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,
  dfSub = NULL,
  sigma = 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# = numeric will scale a compartment # by a dividing the compartment amount
 by the scale factor, like NONMEM.

constants is a boolean indicating if constants should be included in the list of parameters. Currently rxode2 parses constants into variables in case you wish to change them
without recompiling the rxode2 model.

params a numeric named vector with values for every parameter in the ODE system; the
names must correspond to the parameter identifiers used in the ODE specification;

inits a vector of initial values of the state variables (e.g., amounts in each compart-
ment), and the order in this vector must be the same as the state variables (e.g.,
PK/PD compartments);

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

keep Columns to keep from either the input dataset or the iCov dataset. With the iCov
data set, 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.

**df0bs**

Degrees of freedom to sample the unexplained variability matrix from the in-
verse 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`

*Creates a package from compiled `rxode2` models*

**Description**

Creates a package from compiled `rxode2` models

**Usage**

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

```
... Models to build a package from
package 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_129f8f97fb94a87ca49ca8d9e691e1e_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/. Since the sitmo threelfry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

Usage

`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` uses the vander-corput generator
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()

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

Value

poisson random number deviates

Examples

### Use threefry engine

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

rxpois(4) ## The first arguments are the lambda parameter

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

rx <- rxode2({
  a <- rxpois(3)
})

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

s <- rxSolve(rx, et)

---

**rxPp**

Simulate a from a Poisson process

Description

Simulate a from a Poisson process
Usage

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

Arguments

- **n**: Number of time points to simulate in the Poisson process
- **lambda**: Rate of Poisson process
- **gamma**: Asymmetry rate of Poisson process. When gamma=1.0, this simulates a homogeneous Poisson process. When gamma<1.0, the Poisson process has more events early; when gamma>1.0, the Poisson process has more events late in the process.
  
  When gamma is non-zero, the tmax should not be infinite but indicate the end of the Poisson process to be simulated. In most pharamcometric cases, this will be the end of the study. Internally this uses a rate of:
  
  $$l(t) = \lambda \gamma (t/tmax)^{\gamma-1}$$
  
- **prob**: When specified, this is a probability function with one argument, time, that gives the probability that a Poisson time $t$ is accepted as a rejection time.
- **t0**: the starting time of the Poisson process
- **tmax**: the maximum time of the Poisson process
- **randomOrder**: when TRUE randomize the order of the Poisson events. By default (FALSE) it returns the Poisson process 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 homogeneous Poisson process of rate 1/10
rxPp(10, 1 / 10)
## Sample inhomogenous Poisson rate of 1/10
```
rxPreferredDistributionName

Description

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

Usage

rxPreferredDistributionName(dist)

Arguments

dist This is the input distribution

Value

Preferred distribution term

Author(s)

Matthew Fidler

Examples

rxPreferredDistributionName("dnorm")

rx PreferredDistributionName("add")

# can be vectorized

rxPreferredDistributionName(c("add","dnorm"))
**rxProgress**

**rxode2 progress bar functions**

### Description

`rxProgress` sets up the progress bar

### Usage

```r
taxProgress(num, core = 0L)
taxTick()
taxProgressStop(clear = TRUE)
taxProgressAbort(error = "Aborted calculation")
```

### Arguments

- `num`: Tot 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({
    on.exit({
      rxProgressAbort()
    })
  })
  rxProgress(100)
  for (i in 1:100) {
```
rxRandNV

Create a random "normal" matrix using vandercorput generator

Description

Create a random "normal" matrix using vandercorput generator

Usage

rxRandNV(nrow = 1, ncol = 1)

Arguments

nrow Number of rows
ncol Number of Columns

Value

Matrix of random numbers

Author(s)

Matthew Fidler

Examples

rxRandNV(1, 1)
rxRandNV(3, 2)
### rxRateDur

**Description**

This is primarily to display information about rate

**Usage**

```r
rxRateDur(x)
```

```r
## S3 method for class 'rxRateDur'
## method for class 'rxRateDur'
## method for class 'rxRateDur'
```

**Arguments**

- `x` : rxRateDur data
- `...` : Other parameters

**Value**

rxRateDur object

---

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

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

**Arguments**

`data`  
`rxode2` ui function, named data to be consistent with `dplyr::rename()`

`...`  
rename items

`envir`  
Environment for evaluation

**Value**

New model with items renamed

**Author(s)**

Matthew L. Fidler

**Examples**

```r
ocmt <- function() {
  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 Rode supported reserved keywords*

---

**Description**

A list and description of Rode supported reserved keywords

**Usage**

`rxReservedKeywords`

**Format**

A data frame with 3 columns and 98 or more rows

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

---

**rxRmvn**

*Simulate from a (truncated) multivariate normal*

---

**Description**

This is simulated with the fast, thread-safe threelfry simulator and can use multiple cores to generate the random deviates.
Usage

rxRmvn(
    n,
    mu = NULL,
    sigma,
    lower = -Inf,
    upper = Inf,
    ncores = 1,
    isChol = FALSE,
    keepNames = TRUE,
    a = 0.4,
    tol = 2.05,
    nlTol = 1e-10,
    nlMaxiter = 100L
)

Arguments

n Number of random row vectors to be simulated OR the matrix to use for simulation (faster).

mu mean vector

sigma Covariance matrix for multivariate normal or a list of covariance matrices. If a list of covariance matrix, each matrix will simulate n matrices and combine them to a full matrix

lower is a vector of the lower bound for the truncated multivariate norm

upper is a vector of the upper bound for the truncated multivariate norm

ncores Number of cores used in the simulation

isChol A boolean indicating if sigma is a cholesky decomposition of the covariance matrix.

keepNames Keep the names from either the mean or covariance matrix.

a threshold for switching between methods; They can be tuned for maximum speed; There are three cases that are considered:

case 1: a < l < u

case 2: l < u < -a

case 3: otherwise

where l=lower and u = upper

tol When case 3 is used from the above possibilities, the tol value controls the acceptance rejection and inverse-transformation;

When abs(u-l)>tol, uses accept-reject from randn

nlTol Tolerance for newton line-search

nlMaxiter Maximum iterations for newton line-search
Value

If \( n==\text{integer} \) (default) the output is an \((n \times d)\) matrix where the \(i\)-th row is the \(i\)-th simulated vector.

If \( \text{is.matrix}(n) \) then the random vector are store in \( n \), which is provided by the user, and the function returns \text{NULL} invisibly.

Author(s)

Matthew Fidler, Zdravko Botev and some from Matteo Fasiolo

References


The thread safe multivariate normal was inspired from the \texttt{mvnfast} package by Matteo Fasiolo

[https://CRAN.R-project.org/package=mvnfast](https://CRAN.R-project.org/package=mvnfast)

The concept of the truncated multivariate normal was taken from Zdravko Botev Botev (2017) doi:10.1111/rssb.12162 and Botev and L’Ecuyer (2015) doi:10.1109/WSC.2015.7408180 and converted to thread safe simulation;

Examples

```r
## From mvnfast
## Unlike mvnfast, uses threelfry simulation

# Creating covariance matrix
mcov <- tcrossprod(tmp, tmp)

set.seed(414)
rxRmvn(4, 1:d, mcov)

A <- matrix(NA, 4, d)
class(A) <- "numeric" # This is important. We need the elements of A to be of class "numeric".

set.seed(414)
rxRmvn(A, 1:d, mcov, ncores = 2) # This returns NULL ...
```
A # ... but the result is here

## You can also simulate from a truncated normal:

```r
rxRmvn(10, 1:d, mcov, lower = 1:d - 1, upper = 1:d + 1)
```

# You can also simulate from different matrices (if they match
# dimensions) by using a list of matrices.

```r
matL <- lapply(1:4, function(...) {
  tmp <- matrix(rnorm(d^2), d, d)
  tcrossprod(tmp, tmp)
})
```

```r
rxRmvn(4, setNames(1:d, paste0("a", 1:d)), matL)
```

---

### `rxS`

Load a model into a symengine environment

**Description**

Load a model into a symengine environment

**Usage**

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

**Arguments**

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

**Value**

rxode2/symengine environment

**Author(s)**

Matthew Fidler
**rxSetControl**

**rxSetControl options for UI object**

**Description**

rxSetControl options for UI object

**Usage**

rxSetControl(ui, control)

**Arguments**

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

**Value**

Nothing, called for side effects

**Author(s)**

Matthew L. Fidler

---

**rxSetCovariateNamesForPiping**

*Assign covariates for piping*

**Description**

Assign covariates for piping

**Usage**

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
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(err <- 0.7)
  })
}

# now TC is detected as a covariate instead of a population parameter

one.compartment %>%
  model({ka <- exp(tka + eta.ka + TC * cov.C)})

# You can turn it off by simply adding it back

rxSetCovariateNamesForPiping()

one.compartment %>%
  model({ka <- exp(tka + eta.ka + TC * cov.C)})

# The covariates you set with `rxSetCovariateNamesForPiping()`
# are turned off every time you solve (or fit in nlmixr)
```
**rxSetIni0**

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

### Description

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

### Usage

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

### Arguments

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

### Value

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

---

**rxSetProd**

*Defunct setting of product*

### Description

Defunct setting of product

### Usage

```
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

---

**rxSetSeed**  
*Set the parallel seed for rxode2 random number generation*

**Description**

This sets the seed for the rxode2 parallel random number generation. If set, then whenever a seed is set for the threefry or vandercorput simulation engine, it will use this seed, increment for the number of seeds and continue with the sequence the next time the random number generator is called.

**Usage**

`rxSetSeed(seed)`

**Arguments**

- **seed**  
  An integer that represents the rxode2 parallel and internal random number generator seed. When positive, use this seed for random number generation and increment and reseed any parallel or new engines that are being called. When negative, turn off the rxode2 seed and generate a seed from the R’s uniform random number generator. Best practice is to set this seed.
**Details**

In contrast, when this is not called, the time that the vandercorput or threefry simulation engines are seeded it comes from a uniform random number generated from the standard R random seed. This may cause a duplicate seed based on the R seed state. This means that there could be correlations between simulations that do not exist. This will avoid the birthday problem picking exactly the same seed using the seed state of the R random number generator. The more times the seed is called, the more likely this becomes.

**Value**

Nothing, called for its side effects

**Author(s)**

Matthew Fidler

**References**

JD Cook. (2016). Random number generator seed mistakes. [https://tinyurl.com/m62v3kv9](https://tinyurl.com/m62v3kv9)

**See Also**

rxGetSeed, rxWithSeed, rxWithPreserveSeed

**Examples**

```r
rxSetSeed(42)
# seed with generator 42
rxnorm()
# Use R's random number generator
rnorm(1)

rxSetSeed(42)
# reproduces the same number
rxnorm()
# But R's random number is not the same
rnorm(1)
# If we reset this to use the R's seed
# (internally rxode2 uses a uniform random number to span seeds)
# This can lead to duplicate sequences and seeds
rxSetSeed(-1)
# Now set seed works for both.
```
# This is not recommended, but illustrates the different types of
# seeds that can be generated.

set.seed(42)
rxnorm()
rnorm(1)
set.seed(42)
rxnorm()
rnorm(1)

---

**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

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

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

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

Arguments

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

params  Initial parameters for model

events  Event information (currently ignored)

inits  Initial estimates for model

...  Other arguments passed to rxShiny. Currently doesn’t do anything.

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

Value

Nothing: Starts a shiny server
**rxSimThetaOmega**

*Simulate Parameters from a Theta/Omega specification*

**Description**

Simulate Parameters from a Theta/Omega specification

**Usage**

```r
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
\textbf{rxSimThetaOmega}

- **\texttt{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 \texttt{omega} is \texttt{NA} and you are using it with a \texttt{rxode2} ui model, the between subject variability described by the \texttt{omega} matrix are set to zero.

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

- **\texttt{omegaLower}**
  
  Lower bounds for simulated ETAs (by default \texttt{-Inf})

- **\texttt{omegaUpper}**
  
  Upper bounds for simulated ETAs (by default \texttt{Inf})

- **\texttt{omegaIsChol}**
  
  Indicates if the \texttt{omega} supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.

- **\texttt{omegaSeparation}**
  
  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 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.

- **\texttt{omegaXform}**
  
  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:

  - \texttt{identity} This is when standard deviation values are directly modeled by the \texttt{params} and \texttt{thetaMat} matrix
  
  - \texttt{variance} This is when the \texttt{params} and \texttt{thetaMat} simulates the variance that are directly modeled by the \texttt{thetaMat} matrix
  
  - \texttt{log} This is when the \texttt{params} and \texttt{thetaMat} simulates \texttt{log(sd)}
  
  - \texttt{nlmixrSqrt} This is when the \texttt{params} and \texttt{thetaMat} simulates the inverse cholesky decomposed matrix with the \texttt{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 \texttt{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.

- **\texttt{nSub}**
  
  Number between subject variabilities (ETAs) simulated for every realization of the parameters.

- **\texttt{thetaMat}**
  
  Named theta matrix.
thetaLower  Lower bounds for simulated population parameter variability (by default -Inf)
thetaUpper  Upper bounds for simulated population unexplained variability (by default Inf)
thetaDf     The degrees of freedom of a t-distribution for simulation. By default this is NULL
            which is equivalent to Inf degrees, or to simulate from a normal distribution
            instead of a t-distribution.
thetaIsChol Indicates if the theta supplied is a Cholesky decomposed matrix instead of the
            traditional symmetric matrix.
nStud       Number virtual studies to characterize uncertainty in estimated parameters.
sigma      Named sigma covariance or Cholesky decomposition of a covariance matrix.
            The names of the columns indicate parameters that are simulated. These are
            simulated for every observation in the solved system. When sigma is NA and
            you are using it with a rxode2 ui model, the unexplained variability described
            by the sigma matrix are set to zero.
sigmaLower  Lower bounds for simulated unexplained variability (by default -Inf)
sigmaUpper  Upper bounds for simulated unexplained variability (by default Inf)
sigmaDf     Degrees of freedom of the sigma t-distribution. By default it is equivalent to
            Inf, or a normal distribution.
sigmaIsChol Boolean indicating if the sigma is in the Cholesky decomposition instead of a
            symmetric covariance
sigmaSeparation
            separation strategy for sigma;
            Tells the type of separation strategy when simulating covariance with parameter
            uncertainty with standard deviations modeled in the thetaMat matrix.
            • "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution
              parameter eta equal to the degrees of freedom nu by (nu-1)/2
            • "separation" simulates from the identity inverse Wishart covariance matrix with
              nu degrees of freedom. This is then converted to a covariance matrix and augmented
              with the modeled standard deviations. While computationally more complex than the
              "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10
            • "auto" chooses "lkj" when the dimension of the matrix is less than 10
              and "separation" when greater than equal to 10.
sigmaXform  When taking sigma values from the thetaMat simulations (using the separation
            strategy for covariance simulation), how should the thetaMat values be turned
            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

da data frame with the simulated subjects

Author(s)

Matthew L. Fidler

Value

a data frame with the simulated subjects

Author(s)

Matthew L. Fidler

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,
etta = 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,
indLinPhiToI = 1e-07,
indLinPhiIM = 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)
## S3 method for class 'function'
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...,
  theta = NULL,
  eta = NULL
)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

object is a either a rxode2 family of objects, or a file-name with a rxode2 model specification, or a string with a rxode2 model specification.

params a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;

events an eventTable object describing the input (e.g., doses) to the dynamic system and observation sampling time points (see eventTable());

inits a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);

scale a numeric named vector with scaling for ode parameters of the system. The names must correspond to the parameter identifiers in the ODE specification. Each of the ODE variables will be divided by the scaling factor. For example scale=c(center=2) will divide the center ODE variable by 2.

method The method for solving ODEs. Currently this supports:

- "liblsoda" thread safe lsoda. This supports parallel thread-based solving, and ignores user Jacobian specification.
- "lsoda" – LSODA solver. Does not support parallel thread-based solving, but allows user Jacobian specification.
- "dop853" – DOP853 solver. Does not support parallel thread-based solving nor user Jacobain 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 and steady-state solutions the default is $0.5 \times 10^{-(\text{sigdig}-1.5)}$ (sensitivity changes only applicable for liblsoda). 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=NA (default), uses the average difference $+ hmaxSd*sd$ in times and sampling events. The hmaxSd is a user specified parameter and which defaults to zero. When hmax=NULL rxode2 uses the maximum difference in times in your sampling and events. The value 0 is equivalent to infinite maximum absolute step size.
The number of standard deviations of the time difference to add to hmax. The default is 0.

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

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

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

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

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.

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

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.

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

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

Upper bounds for simulated unexplained variability (by default Inf)

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.

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 `THETA[#]` and added to parameters

**thetaLower**

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

**thetaUpper**

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

**eta**

A vector of parameters that will be named `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. By default, `rxode2` only includes estimates at the observations. (default `FALSE`). When
addDosing is NULL, only include EVID=0 on solve and exclude any model-times or EVID=2. If addDosing is NA the classic rxode2 EVID events are returned. When addDosing is TRUE add the event information in NONMEM-style format; if subsetNonmem=FALSE rxode2 will also include extra event types (EVID) for ending infusion and modeled times:

- EVID=-1 when the modeled rate infusions are turned off (matches rate=-1)
- EVID=-2 When the modeled duration infusions are turned off (matches rate=-2)
- EVID=-10 When the specified rate infusions are turned off (matches rate>0)
- EVID=-20 When the specified dur infusions are turned off (matches dur>0)
- EVID=101,102,103,... 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 Inf. Also trims to -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 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 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.

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

omegaSeparation Omega separation strategy
Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the thetaMat matrix.

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

omegaXform When taking omega values from the thetaMat simulations (using the separation strategy for covariance simulation), how should the thetaMat values be turned int standard deviation values:
• identity This is when standard deviation values are directly modeled by the params and thetaMat matrix.
• variance This is when the params and thetaMat simulates the variance that are directly modeled by the thetaMat matrix.
• log This is when the params and thetaMat simulates \(\log(sd)\).
• nlmixrSqrt This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the \(x^2\) modeled along the diagonal. This only works with a diagonal matrix.
• 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.

omegaLower Lower bounds for simulated ETAs (by default -Inf)
omegaUpper Upper bounds for simulated ETAs (by default Inf)
nSub Number between subject variabilities (ETAs) simulated for every realization of the parameters.
thetaMat Named theta matrix.
thetaDf The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.
thetaIsChol Indicates if the theta supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
nStud Number virtual studies to characterize uncertainty in estimated parameters.
dfSub Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
dfObs Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
returnType This tells what type of object is returned. The currently supported types are:
  • "rxSolve" (default) will return a reactive data frame that can change easily change different pieces of the solve and update the data frame. This is the currently standard solving method in rxode2, is used for \texttt{rxSolve(object, \ldots)}, \texttt{solve(object, \ldots)},
  • "data.frame" – returns a plain, non-reactive data frame; Currently very slightly faster than \texttt{returnType="matrix"}
  • "matrix" – returns a plain matrix with column names attached to the solved object. This is what is used \texttt{object$run} as well as \texttt{object$solve}
  • "data.table" – returns a data.table; The data.table is created by reference (ie setDt()), which should be fast.
  • "tbl" or "tibble" returns a tibble format.
seed an object specifying if and how the random number generator should be initialized
nsim represents the number of simulations. For rxode2, if you supply single subject event tables (created with `eventTable()`)

minSS Minimum number of iterations for a steady-state dose

maxSS Maximum number of iterations for a steady-state dose

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

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

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

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

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

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

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

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

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

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

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

indLinPhiTol the requested accuracy tolerance on exponential matrix.

indLinPhiM the maximum size for the Krylov basis

indLinMatExpType This is them matrix exponential type that is use for rxode2. Currently the following are supported:

- Al-Mohy Uses the exponential matrix method of Al-Mohy Higham (2009)
- arma Use the exponential matrix from RcppArmadillo
- expokit Use the exponential matrix from Roger B. Sidje (1998)
indLinMatExpOrder

an integer, the order of approximation to be used, for the Al-Mohy and expokit values. The best value for this depends on machine precision (and slightly on the matrix). We use 6 as a default.

drop

Columns to drop from the output

idFactor

This boolean indicates if original ID values should be maintained. This changes the default sequentially ordered ID to a factor with the original ID values in the original dataset. By default this is enabled.

mxhnil

maximum number of messages printed (per problem) warning that T + H = T on a step (H = step size). This must be positive to result in a non-default value. The default value is 0 (or infinite).

hmxi

inverse of the maximum absolute value of H to be used. hmxi = 0.0 is allowed and corresponds to an infinite hmax1 (default). hmxi and hmximay be changed at any time, but with

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
rxSolve

• "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.

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 dataframe 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()`

---

### rxStack

**Stack a solved object for things like ggplot**

**Description**
Stack a solved object for things like ggplot

**Usage**

```r
rxStack(Data, vars = NULL)
```

**Arguments**

- `Data` is a `rxode2` object to be stacked.
- `vars` Variables to include in stacked data; By default this is all the variables when `vars` is NULL.

**Value**
Stacked data with `value` and `trt`, where `value` is the values and `trt` is the state and `lhs` variables.
rxState

Author(s)
Matthew Fidler

---

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
taxSumProdModel(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

# 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
Get Omega^-1 and derivatives

**Usage**

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

**Arguments**

- `invObjOrMatrix`: Object for inverse-type calculations. If this is a matrix, setup the object for inversion `rxSymInvCholCreate()` with the default arguments and return a reactive s3 object. Otherwise, use the inversion object to calculate the requested derivative/inverse.

- `theta`: Thetas to be used for calculation. If missing (NULL), a special s3 class is created and returned to access Omega^-1 objects as needed and cache them based on the theta that is used.

- `type`: The type of object. Currently the following types are supported:
  - `cholOmegaInv` gives the Cholesky decomposition of the Omega Inverse matrix.
  - `omegaInv` gives the Omega Inverse matrix.
  - `d(omegaInv)` gives the `d(Omega^-1)` with respect to the theta parameter specified in `thetaNumber`.
  - `d(D)` gives the `d(diagonal(Omega^-1))` with respect to the theta parameter specified in the `thetaNumber` parameter.

- `thetaNumber`: For types `d(omegaInv)` and `d(D)`, the theta number that the derivative is taken against. This must be positive from 1 to the number of thetas defining the Omega matrix.

**Value**

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

**Author(s)**

Matthew L. Fidler
**rxSyncOptions**  
*Sync options with rxode2 variables*

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

nothing; called for side effects

**Author(s)**

Matthew L. Fidler

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

**Description**

A list and description of Rode supported syntax functions

**Usage**

```r	rxSyntaxFunctions
```

**Format**

A data frame with 3 columns and 98 or more rows

- **Function**  
  Reserved function Name
- **Description**  
  Description of function
- **Aliases**  
  Function Aliases
**rxt**  
*Simulate student t variable from threefry generator*

**Description**

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

**Usage**

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

**Arguments**

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

**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()`

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

t-distribution random numbers

**Examples**

```r
## Use threefry engine

rxt(df = 3, n = 10) # with rxt you have to explicitly state n
rxt(df = 3, n = 10, ncores = 2) # You can parallelize the simulation using openMP
```
\texttt{rxTempDir} \quad \textit{Get the rxode2 temporary directory}

\textbf{Description}

Get the rxode2 temporary directory

\textbf{Usage}

\texttt{rxTempDir()}

\textbf{Value}

rxode2 temporary directory.

\texttt{rxTheme} \quad \textit{rxTheme is the ggplot2 theme for rxode2 plots}

\textbf{Description}

rxTheme is the ggplot2 theme for rxode2 plots

\textbf{Usage}

\texttt{rxTheme(}
    \hspace{0.5em} \texttt{base_size = 11},
    \hspace{0.5em} \texttt{base_family = "",}
    \hspace{0.5em} \texttt{base_line_size = base_size/22,}
    \hspace{0.5em} \texttt{base_rect_size = base_size/22,}
    \hspace{0.5em} \texttt{grid = TRUE}
\texttt{)}
Arguments

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

Value

ggplot2 theme used in rxode2

See Also

Other rxode2 plotting: **plot.rxSolve()**

---

**Description**

rxode2 to symengine environment

**Usage**

- `rxToSE(x, envir = NULL, progress = FALSE, promoteLinSens = TRUE)`
- `.rxToSE(x, envir = NULL, progress = FALSE)`
- `rxFromSE(x, unknownDerivatives = c("forward", "central", "error"))`
- `.rxFromSE(x)`

**Arguments**

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

**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'

```r
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

S3 for getting information from UI model
Usage

## 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, ...)

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, ...)

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

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

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

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

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

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

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

**Arguments**

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

**Value**

value that was requested from the UI object

**Author(s)**

Matthew Fidler

---

**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**

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

- **min**: lower and upper limits of the distribution. Must be finite.
- **max**: lower and upper limits of the distribution. Must be finite.
- **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** uses the vander-corput generator

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()**

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

Value

uniform random numbers

Examples

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

rxunif()

## This example uses `rxunif` directly in the model
rx <- rxode2(
  a <- rxunif(0, 3)
)

et <- et(1, id = 1:2)
s <- rxSolve(rx, et)
```
**rxUnloadAll**  
*Unloads all rxode2 compiled DLLs*

**Description**
Unloads all rxode2 compiled DLLs

**Usage**
```
rxUnloadAll()
```

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

**Examples**
```
print(rxUnloadAll())
```

---

**rxUse**  
*Use model object in your package*

**Description**
Use model object in your package

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

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

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

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

**Description**

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

**Usage**

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

**Arguments**

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

**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**: shape and scale parameters, the latter defaulting to 1.
- **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

The `rxnorm` simulates using the threefry sitmo generator; the `rxnormV` uses the vandercorput generator.

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()`.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

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 <- rxode2({
  a <- rxweibull(1, 3)
})

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

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

**Setup Windows components for rxode2**

**Description**

Setup Windows components for rxode2

**Usage**

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

**Arguments**

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

**Value**

nothing, used for its side effects

**Author(s)**

Matthew L. Fidler

---

### rxWithSeed

**Preserved seed and possibly set the seed**

**Description**

Preserved seed and possibly set the seed

**Usage**

```r
rxWithSeed(
  seed,
  code,
  rxseed = rxGetSeed(),
  kind = "default",
  normal.kind = "default",
  sample.kind = "default"
)
```

```r
rxWithPreserveSeed(code)
```
Arguments

seed  R seed to use for the session

code  Is the code to evaluate

rxseed  is the rnode2 seed that is being preserved

kind  character or NULL. If kind is a character string, set R's RNG to the kind desired. Use "default" to return to the R default. See ‘Details’ for the interpretation of NULL.

normal.kind  character string or NULL. If it is a character string, set the method of Normal generation. Use "default" to return to the R default. NULL makes no change.

sample.kind  character string or NULL. If it is a character string, set the method of discrete uniform generation (used in sample, for instance). Use "default" to return to the R default. NULL makes no change.

Value

returns whatever the code is returning

See Also

rxGetSeed, rxSetSeed

Examples

```
rxGetSeed()
rxWithSeed(1, {
  print(rxGetSeed())
  rxnorm()
  print(rxGetSeed())
  rxnorm()
}, rxseed=3)
```

<table>
<thead>
<tr>
<th>stat_amt</th>
<th>Dosing/Amt geom/stat</th>
</tr>
</thead>
</table>

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

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()` or `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).
- **position**: Position adjustment, either as a string, or the result of a call to a position adjustment function.
- **show.legend**: logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
- **...**: 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

Examples

```r
library(rxode2)
library(units)

## Model from RxODE tutorial
mod1 <- rxode2({
  KA=2.94E-01
  CL=1.86E+01
  V2=4.02E+01
  Q=1.05E+01
  V3=2.97E+02
  Kin=1
  Kout=1
  EC50=200
  C2 = centr/V2
  C3 = peri/V3
  d/dt(depot) =-KA*depot
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3
  d/dt(peri) = Q*C2 - Q*C3
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff
})

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

## 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) %>%
  seq(0, 11*24, length.out=100));

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

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

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

---

**stat_cens**

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**

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

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

**Arguments**

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

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

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

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

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

width  
represents the width (in \ censoring box

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

Details

Requires the following aesthetics:

- `x` Represents the independent variable, often the time scale
- `y` represents the dependent variable
- `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

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

Arguments

- `object`  
  rxode2 object
- `...`  
  Ignored parameters
update.rxUi

Value

object is returned

Author(s)

Matthew L. Fidler

Description

Update for rxUi

Usage

## 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

uppergamma(1, 3)
uppergamma(1:3, 3)
uppergamma(1, 1:3)
Index

* Internal
  plot.rxSolve, 51
* Nonlinear regression
  eventTable, 36
  rxode2, 88
* ODE models
  rxode2, 88
* Ordinary differential equations
  rxode2, 88
* PK/PD
  genShinyApp.template, 42
* Pharmacodynamics (PD)
  eventTable, 36
  rxode2, 88
* Pharmacokinetics (PK)
  eventTable, 36
  rxode2, 88
* Query model information
  rxDfdy, 70
  rxLhs, 85
  rxParams, 98
  rxState, 137
* datasets
  rxReservedKeywords, 109
  rxSyntaxFunctions, 141
* data
  eventTable, 36
* models
  eventTable, 36
  rxode2, 88
* nonlinear
  genShinyApp.template, 42
  rxode2, 88
* ordinary differential equations
  eventTable, 36
* pharmacometrics
  genShinyApp.template, 42
* rxode2 plotting
  plot.rxSolve, 51

  rxTheme, 143
* simulation
  genShinyApp.template, 42
  .C(), 65
  .Call(), 65
  .copyUi, 5
  .handleSingleErrTypeNormOrTfoceiBase, 6
  .modelHandleModellines, 6
  .quoteCallInfoLines, 7
  .rxFromSE (rxToSE), 144
  .rxLinCmtGen, 8
  .rxToSE (rxToSE), 144
  .rxWithOptions, 9
  .rxWithWd, 9
  [.rxEvid (rxEvid), 71
  [.rxRateDur (rxRateDur), 107
  [[.rxEvid (rxEvid), 71
  [[.rxRateDur (rxRateDur), 107
  add.dosing, 10, 11, 13, 24, 28, 31, 34
  add.dosing(), 36, 94
  add.sampling, 11, 13, 13, 24, 28, 31, 34
  add.sampling(), 36, 94
  aes(), 155, 157
  aes_(), 155, 157
  as.character.rxEvid (rxEvid), 71
  as.character.rxRateDur (rxRateDur), 107
  as.et, 15
  as.rxEvid (rxEvid), 71
  as.rxRateDur (rxRateDur), 107
  assertRxUi, 16
  assertRxUiEstimatedResiduals
    (assertRxUi), 16
  assertRxUiMixedOnly (assertRxUi), 16
  assertRxUiMuRefOnly (assertRxUi), 16
  assertRxUiNormal (assertRxUi), 16
  assertRxUiPopulationOnly (assertRxUi),
  16
  assertRxUiPrediction (assertRxUi), 16
INDEX

assertRxUiRandomOnIdOnly (assertRxUi), 16
assertRxUiSingleEndpoint (assertRxUi), 16
assertRxUiTransformNormal (assertRxUi), 16

borders(), 155, 158
c.rxEvid (rxEvid), 71
c.rxRateDur (rxEvid), 71
cvPost, 18

evironment, 22
erf, 21
et, 11, 13, 21, 24, 28, 31, 34
et(), 11, 37, 90, 94
etExpand, 26
etRbind, 11, 13, 24, 27, 28, 31, 34
etRep, 11, 13, 24, 28, 30, 31, 34
etSeq, 33
eventTable, 11, 13, 24, 28, 31, 34, 36
eventTable(), 43, 90, 94, 128, 136
expit (logit), 46

format.rxEvid (rxEvid), 71
format.rxRateDur (rxEvid), 71

fortify(), 155, 157
gammap, 38
gammapDer, 39
gammapInv, 39
gammapInv (gammapInv), 39
gammaq, 40
gammaqInva (gammaqInv), 41
gammaqInv, 41
rbind.rxEt (etRbind), 27
rename.function (rxRename), 108

rinvchisq, 52
rLKJ1(), 18
rxAllowUnload, 53
rxAppendModel, 54
rxAssignControlValue, 55

rxAssignPtr, 56
rxbeta, 56
rxbeta, 56
rxCores (getRxThreads), 44
rxCreateCache, 66
rxDerived, 68

rxCreateCache, 66
rxDerived, 68
rxDerived, 68
rxD, 67
rxD(), 74
rxDelete, 68
rxDfdy, 70, 85, 100, 137
rxEvid, 71
rxexp, 72
rxf, 73
rxFromSE (rxToSE), 144

rxFun, 74
rxgamma, 76
INDEX

rxgeom, 77
rxGetControl, 79
rxGetLin, 79
rxGetrxdode2, 80
rxGetSeed, 81
rxHtml, 82
rxIndLin, 83
rxIndLinState, 82
rxIndLinStrategy, 83
rxInits, 70, 85, 100, 137
rxInv, 84
rxIsCurrent, 85
rxLhs, 70, 85, 100, 137
rxLock, 86
rxModelVars, 70, 85, 100, 137
rxModelVars(), 146
rxNorm, 86
rxmlnorm, 87
rxnormM(rxmlnorm), 87
RxODE (rxode2), 88
rxode (rxode2), 88
rxode2, 11, 13, 24, 28, 31, 34, 85, 88
rxode2(), 37, 43, 65, 136, 137, 146
rxOptExpr, 96
rxord, 97
rxParam (rxParams), 98
rxParams, 70, 85, 98, 137
rxPkg, 100
rxPois, 101
rxPp, 102
rxPreferredDistributionName, 104
rxProgress, 105
rxProgressAbort (rxProgress), 105
rxProgressStop (rxProgress), 105
rxRandNV, 106
rxRateDur, 107
rxRemoveControl, 107
rxRename, 108
rxReservedKeywords, 109
rxRmFun (rxFun), 74
rxRmvn, 109
rxS, 112
rxSetControl, 113
rxSetCovariateNamesForPiping, 113
rxSetIni0, 115
rxSetProd, 115
rxSetProgressBar, 116
rxSetSeed, 116
rxSetSum, 118
rxShiny, 118
rxSimThetaOmega, 120
rxSolve, 123
rxSolve(), 42
rxStack, 136
rxState, 70, 85, 100, 137
rxSumProdModel, 138
rxSupportedFuns, 138
rxSuppressModel, 139
rxSymInvChol, 140
rxSymInvCholCreate(), 140
rxSyncOptions, 141
rxSyntaxFunctions, 141
rxt, 142
rxTempDir, 143
rxTest (rxValidate), 151
rxTheme, 51, 143
rxTick (rxProgress), 105
rxToSE, 144
rxTrans, 145
rxTrans(), 65
rxiUiGet (rxiUiGet.cmtLines), 146
rxiUiGet.cmtLines, 146
rxunif, 148
rxUnloadAll, 150
rxUnlock (rxLock), 86
rxUse, 150
rxValidate, 151
rxweibull, 151
rxWinSetup, 153
rxWithPreserveSeed (rxWithSeed), 153
rxWithSeed, 153
sample, 154
save(), 150
seq.rxEt (etSeq), 33
setRxThreads (getRxThreads), 44
setRxThreads(), 129
simulate.rxode2 (rxSolve), 123
simulate.rxParams (rxSolve), 123
simulate.rxSolve (rxSolve), 123
solve.rxEt (rxSolve), 123
solve.rxode2 (rxSolve), 123
solve.rxParams (rxSolve), 123
solve.rxSolve (rxSolve), 123
solve.rxUi (rxSolve), 123
stat_amt, 154
stat_cens, 157
summary.rxode2, 158
sys.call, 22

update.rxSolve (rxSolve), 123
update.rxUi, 159
uppergamma, 159
use_description(), 101

vname, 16

write.template.server
  (genShinyApp.template), 42
write.template.ui
  (genShinyApp.template), 42
write.template.ui(), 43