Package ‘mrgsim.sa’

November 30, 2020

Type Package
Title Sensitivity Analysis with ‘mrgsolve’
Version 0.1.0
Maintainer Kyle Baron <kylebtwin@imap.cc>
Description Perform sensitivity analysis on ordinary differential equation based models, including ad-hoc graphical analyses based on structured sequences of parameters as well as local sensitivity analysis. Functions are provided for creating inputs, simulating scenarios and plotting outputs.
License GPL (>= 2)
URL https://github.com/kylebaron/mrgsim.sa
BugReports https://github.com/kylebaron/mrgsim.sa/issues
Suggests testthat, knitr, rmarkdown
Imports withr, purrr, dplyr, assertthat, rlang, ggplot2, tidyselect, tidyr, methods, tibble, patchwork
Encoding UTF-8
Language en-US
Depends mrgsolve
RoxygenNote 7.1.1
Collate 'utils.R' 'parseq.R' 'AAA.R' 'lsa.R' 'sens_each.R'
'sens_grid.R' 'sens_plot.R' 'sens_run.R' 'seq.R'
NeedsCompilation no
Author Kyle Baron [aut, cre]
Repository CRAN
Date/Publication 2020-11-30 11:20:02 UTC

R topics documented:

lsa ................................................................. 2
mrgsim.sa ......................................................... 3
lsa

Perform local sensitivity analysis

Description

Perform local sensitivity analysis

Usage

```r
lsa(mod, par, var, fun = .lsa_fun, eps = 1e-08, ...)
lsa_plot(x, ...)
```

Arguments

- `mod` a mrgsolve model object
- `par` parameter names as character vector or comma-separated string
- `var` output names (compartment or capture) as character vector or comma-separated string
- `fun` generating simulated for sensitivity analysis (see details)
- `eps` parameter change value for sensitivity analysis
- `...` passed to `plot.lsa()`
- `x` output from `lsa()`

Value

A tibble with class `lsa`. 
Examples

```r
mod <- mrgsolve::house(delta=0.1)
par <- "CL,VC,KA"
var <- "CP"
dose <- ev(amt = 100)
fun <- function(mod, ...) mrgsolve::mrgsim_e(mod, dose, output="df")
out <- lsa(mod, par, var, fun)
head(out)
lsa_plot(out)
```

---

**mrgsim.sa**  
**Sensitivity Analysis with 'mrgsolve'**

**Description**

Perform local sensitivity analysis on ordinary differential equation based models, including ad-hoc graphical analyses based on sequences of parameters as well as local sensitivity analysis. Functions are provided for creating inputs, simulating scenarios and plotting outputs.

**Details**

- Local sensitivity analysis: `lsa()
- Run ad-hoc sensitivity analyses: `sens_each(), sens_grid(), sens_run()
  - Use `sens_each_data()` and `sens_grid_data()` to pass in data sets
- Parameter sequence generation:
  - In a pipeline: `parseq_cv(), parseq_fct(), parseq_range(), parseq_manual()
  - Stand alone: `seq_cv(), seq_fct(), seq_geo(), seq_even()``

---

**parseq_cv**  
**Generate a sequence of parameters based on CV**

**Description**

Generate a sequence of parameters based on CV

**Usage**

```
parseq_cv(mod, ..., .cv = 30, .n = 5, .nsd = 2, .digits = NULL)
```
parseq_fct

Generate a sequence of parameters

Description

Generate a sequence of parameters

Usage

parseq_fct(mod, ..., .n = 5, .factor = 2, .geo = TRUE, .digits = NULL)

parseq_factor(mod, ..., .n = 5, .factor = 2, .geo = TRUE, .digits = NULL)
Arguments

- `mod` a model object
- `...` unquoted parameter names
- `.n` number of parameters to simulate between the minimum and maximum parameter values
- `.factor` a numeric vector used to divide and multiply the parameter value thus generating the minimum and maximum parameter values, respectively, for the sequence; if `.factor` is length 1 it will be recycled to length 2; the first value is used to divide the nominal value generating the minimum value; the second value is used to multiply the nominal value generating the maximum value
- `.geo` if TRUE a geometric sequence is generated (evenly spaced from min to max on log scale); otherwise, the sequence is evenly spaced on Cartesian scale
- `.digits` if numeric, the number of significant digits in the parameter sensitivity values are set using `signif()`

Details

- `.n` is passed to `seq_fct()` as `n`
- `.factor` is passed to `seq_fct()` as `factor`

See Also

- `parseq_cv()`, `parseq_range()`, `parseq_manual()`

Examples

```r
mod <- mrgsolve::house()

mod %>%
  parseq_fct(CL, VC) %>%
  sens_each()
```

Description

Simulation helper to manually specify parameter sequences

Usage

`parseq_manual(mod, ...)`
parseq_range

Arguments

mod  mrgsolve model object
...

named numeric vectors of parameter values to simulate; names must correspond
to parameters in the model object

See Also

parseq_cv(), parseq_range(), parseq_fct()

Examples

mod <- mrgsolve::house()

mod %>%
  parseq_manual(CL = c(0.5, 1, 1.5)) %>%
  sens_each()

Description

Simulation helper to generate a sequence of parameters from a range

Usage

parseq_range(mod, ..., .n = 5, .geo = TRUE, .digits = NULL)

Arguments

mod  mrgsolve model object
...

unquoted parameter names,

.n  number of values to simulate for each parameter sequence

.geo  if TRUE generate a geometric sequence; otherwise, generate a sequence evenly

spaced on Cartesian scale; see seq_geo()

.digits  if numeric, the number of significant digits in the parameter sensitivity values

are set using signif()

Details

• .n is passed to seq_geo() as n

See Also

parseq_cv(), parseq_fct(), parseq_manual()
parseq_reference

Examples

```r
mod <- mrgsolve::house()

mod %>%
  parseq_range(CL = c(0.5, 1), VC = c(10, 40)) %>%
  sens_each()
```

---

**parseq_reference**  
*Set reference values for each parameter*

**Description**

Set reference values for each parameter

**Usage**

```r
parseq_reference(mod, auto = TRUE)
```

**Arguments**

- `mod`: a model object
- `auto`: if TRUE then the model parameter list is used

---

**select_par**  
*Identify parameters in a model for sensitivity analysis*

**Description**

Identify parameters in a model for sensitivity analysis

**Usage**

```r
select_par(mod, ...)
```

**Arguments**

- `mod`: an mrgsolve model object
- `...`: unquoted parameter names

**Examples**

```r
mod <- mrgsolve::house()
select_par(mod, CL, VC)
```
**select_sens**  
*Select sensitivity runs from a sens_each object*

**Description**
Select sensitivity runs from a sens_each object

**Usage**

```r
select_sens(x, dv_name = NULL, p_name = NULL)
```

**Arguments**
- `x` a sens_each object
- `dv_name` character names of dependent variables to select
- `p_name` character names of parameters to select

**Examples**

```r
library(dplyr)
mod <- mrgsolve::house()
out1 <- mod %>% parseq_factor(CL,VC) %>% sens_each()
out2 <- select_sens(out1, dv_name = "CP", p_name = "CV")
```

---

**sens_fun**  
*Run an ad-hoc sensitivity analysis*

**Description**
Use `sens_each()` to examine sequences of parameters one at a time. Use `sens_grid()` to examine all combinations of sequences of parameters. The `sens_each_data()` and `sens_grid_data()` variants allow you to pass in a data set to simulate from.

**Usage**

```r
sens_each(mod, idata = NULL, ...)
sens_each_data(mod, data, idata = NULL, ...)
sens_grid(mod, idata = NULL, ...)
sens_grid_data(mod, data, idata = NULL, ...)
```
sens_plot

Arguments

mod an mrgsolve model object (usually read in with mrgsolve::mread())
idata included only to prevent users from passing through; the function will create an
ida set if appropriate
... passed to mrgsolve::mrksim_d()
data a simulation input data set (see mrgsolve::data_set())

Value

A tibble-like object with class sens_each or sens_grid, depending on the vary method that was
used. These objects will look just like a tibble, but they can be plotted with sens_plot().

See Also

sens_plot()

Examples

mod <- mrgsolve::house()
dose <- mrgsolve::ev(amt = 100)
out_each <- parseq_cv(mod, CL, VC) %>% sens_each()
out_grid <- parseq_cv(mod, CL, VC) %>% sens_grid()

sens_plot(data, ...)  

Plot sensitivity analysis results

Description

Plot sensitivity analysis results

Usage

sens_plot(data, ...)

## S3 method for class 'sens_each'
sens_plot(
  data,
  dv_name,
  logy = FALSE,
  ncol = NULL,
  lwd = 0.8,
  digits = 3,
  plot_ref = TRUE,
xlab = "time",
    ylab = dv_name[1],
    grid = FALSE,
    ...
  )

## S3 method for class 'sens_grid'
sens_plot(
  data,
  dv_name,
  digits = 2,
  ncol = NULL,
  lwd = 0.8,
  logy = FALSE,
  plot_ref = TRUE,
  ...
)

Arguments

data     output from sens_each() or sens_grid()
...       arguments passed on to methods
dv_name  output column name to plot
logy     if TRUE, y-axis is transformed to log scale
ncol     passed to ggplot2::facet_wrap()
lwd      passed to ggplot2::geom_line()
digits   used to format numbers on the strips
plot_ref if TRUE, then the reference case will be plotted in a black dashed line
xlab     x-axis title
ylab     y-axis title
grid     if TRUE, plots from the sens_each method will be passed through patchwork::wrap_plots()

Examples

mod <- mrgsolve::house()

dose <- mrgsolve::ev(amt = 100)
out <- sens_run(mod, sargs = list(events = dose), par = "CL,VC")
sens_plot(out, dv_name = "CP")
Run ad-hoc parameter sensitivity analyses with mrgsolve

Usage

sens_run(
  mod, 
  par = NULL, 
  var = NULL, 
  method = c("factor", "cv", "range", "manual"), 
  vary = c("each", "grid"), 
  ..., 
  sargs = list()
)

Arguments

mod       a mrgsolve model object
par       parameter names for sensitivity analysis; this can be a character vector or a 
          comma-separated string (see examples)
var       names of model output variables to include in simulated output; this could be 
          the name of a compartment or another output derived inside of the model (e.g. 
          DV or CP or logV, but is specific to what is coded into mod)
method    parameter sequence generation method
vary      use each to vary one parameter at a time or grid to vary all combinations of 
          parameters
...       passed to method function
sargs     a named list of arguments passed to sens_each() or sens_grid() and eventually to mrgsolve::mrgsim()

Examples

mod <- mrgsolve::house()

dose <- mrgsolve::ev(amt = 100)

sens_run(
  mod, 
  par = "CL,VC", 
  method = "cv", 
  vary = "each", 
  sargs = list(events = dose)
)
seq_cv

Generate a sequence based on coefficient of variation

Description
Generate a sequence based on coefficient of variation

Usage
seq_cv(point, cv = 30, n = 5, nsd = 2, digits = NULL)

Arguments
- point: reference parameter value
- cv: coefficient of variation
- n: number of values to simulate in the sequence
- nsd: number of standard deviations defining the range of simulated parameter values
- digits: number of significant digits in the answer; if NULL (the default) all digits are retained

Examples
seq_cv(10)

seq_even

Generate evenly spaced sequence

Description
Generate evenly spaced sequence

Usage
seq_even(from, to, n = 5, digits = NULL)

Arguments
- from: passed to base::seq()
- to: passed to base::seq()
- n: passed to base::seq() as length.out
- digits: number of significant digits in the answer; if NULL (the default) all digits are retained
**seq_fct**

Generate a sequence by fold increase and decrease from a point

**Description**

Generate a sequence by fold increase and decrease from a point

**Usage**

```r
seq_fct(point, n = 5, factor = c(3, 3), geo = TRUE, digits = NULL)
```

**Arguments**

- `point`: a numeric vector of length 1
- `n`: number of elements in the sequence
- `factor`: an integer vector of length 1 or 2; if length 1, values will be recycled to length 2; the first number used to divide point to generate the minimum value in the sequence; the second number is used to multiply point to generate the maximum value in the sequence
- `geo`: if TRUE, `seq_geo()` is used to generate the sequence; otherwise, `seq_even()` is used to generate the sequence
- `digits`: number of significant digits in the answer; if NULL (the default) all digits are retained

**Examples**

```r
seq_fct(10)
```

**seq_geo**

Generate a geometric sequence of parameter values

**Description**

Generate a geometric sequence of parameter values

**Usage**

```r
seq_geo(from, to, n = 5, digits = NULL)
```
Arguments
from passed to `base::seq()`
to passed to `base::seq()`
n passed to `base::seq()` as length.out
digits number of significant digits in the answer; if NULL (the default) all digits are retained

Examples
`seq_geo(1,10,10)`
Index

base::seq(), 12, 14

ggplot2::facet_wrap(), 10
ggplot2::geom_line(), 10

lsa, 2
lsa(), 3
lsa_plot(lsa), 2

mrgsim.sa, 3
mrgsolve::data_set(), 9
mrgsolve::mread(), 9
mrgsolve::mrgsim(), 11
mrgsolve::mrgsim_d(), 9

parseq_cv, 3
parseq_cv(), 3, 5, 6
parseq_factor(parseq_fct), 4
parseq_fct, 4
parseq_fct(), 3, 4, 6
parseq_manual, 5
parseq_manual(), 3–6
parseq_range, 6
parseq_range(), 3–6
parseq_reference, 7
patchwork::wrap_plots(), 10
plot.lsa(), 2

select_par, 7
select_sens, 8
sens_each(sens_fun), 8
sens_each(), 3, 10, 11
sens_each_data(sens_fun), 8
sens_each_data(), 3
sens_fun, 8
sens_grid(sens_fun), 8
sens_grid(), 3, 10, 11
sens_grid_data(sens_fun), 8
sens_grid_data(), 3
sens_plot, 9
sens_plot(), 9

sens_run, 11
sens_run(), 3
seq_cv, 12
seq_cv(), 3, 4
seq_even, 12
seq_even(), 3, 13
seq_fct, 13
seq_fct(), 3, 5
seq_geo, 13
seq_geo(), 3, 6, 13
signif(), 4–6