Package ‘readsdr’

April 5, 2024

Type Package
Title Translate Models from System Dynamics Software into 'R'
Version 0.3.0
Description The goal of 'readsdr' is to bridge the design capabilities from
specialised System Dynamics software with the powerful numerical tools
offered by 'R' libraries. The package accomplishes this goal by parsing
'XMILE' files ('Vensim' and 'Stella') models into 'R' objects to construct
networks (graph theory); 'ODE' functions for 'Stan'; and inputs to simulate
via 'deSolve' as described in Duggan (2016) <doi:10.1007/978-3-319-34043-2>.
License MIT + file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 7.3.1
Suggests testthat (>= 2.1.0), igraph, knitr, rmarkdown, ggplot2,
tidyrr, truncnorm
Imports stringr, xml2, purrr, dplyr, rlang, stringi, magrittr, stats,
deSolve, utils, future, future.apply, progressr
BugReports https://github.com/jandraor/readsdr/issues
VignetteBuilder knitr
Depends R (>= 4.1.0)
NeedsCompilation no
Author Jair Andrade [aut, cre] (<https://orcid.org/0000-0002-1412-7868>)
Maintainer Jair Andrade <jair.albert.andrade@gmail.com>
Repository CRAN
Date/Publication 2024-04-05 15:50:02 UTC

R topics documented:

create_stan_function .............................................. 2
expit ......................................................... 3
create_stan_function

Create a Stan's ODE function from an XMILE file

Description

create_stan_function returns a string with the code for a Stan's ODE function

Usage

create_stan_function(
  filepath,
  func_name,
  pars = NULL,
  override.consts = NULL,
  additional_funs = NULL
)
Arguments

- filepath: A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile
- func_name: A string for naming the ODE function
- pars: A character vector that indicates which constants will be considered as parameters in the ODE function
- overrideconsts: A list in which each element is a name-value pair that replaces values of constants.
- additional_funs: A vector of strings. Each string corresponds to a user-defined function.

Details

This function extracts the xml from the file specified via filepath to generate the code for an equivalent model in Stan.

Value

A string with the code containing the model’s equations in the format required by Stan.

Examples

```r
path <- system.file("models", "SIR.stmx", package = "readsdr")
create_stan_function(path, "my_model")
```

---

Expit transformation

**Description**

Expit transformation

**Usage**

```r
expit(x)
```

**Arguments**

- x: A real number

**Value**

A number in the range 0 to 1

**Examples**

```r
expit(-3)
```
extract_timeseries_stock

Extract the values over time of a stock from a Stan fit

Description
Extract the values over time of a stock from a Stan fit

Usage
extract_timeseries_stock(stock_name, posterior_df, all_stocks, ODE_output)

Arguments
- stock_name: A string that indicates the stock’s name for which the function will construct the timeseries.
- posterior_df: A Stan fit object converted into a data frame.
- all_stocks: A vector of strings that contains the names of all the stocks in the model. This vector must have the same order as the differential equations in the Stan code.
- ODE_output: A string that indicates the name of the variable where model’s output in stored in Stan.

Value
A data frame

Examples
posterior_df <- data.frame(`yhat[1,2]` = rep(0, 2), `yhat[2,2]` = rep(1, 2),
                           check.names = FALSE)
stocks <- c("S1", "S2")
exttract_timeseries_stock("S2", posterior_df, stocks, "yhat")

extract_timeseries_var

Extract the values over time of a variable from a Stan fit

Description
Extract the values over time of a variable from a Stan fit

Usage
extract_timeseries_var(var_name, posterior_df)
inv

Arguments

var_name A string that indicates the variable’s name for which the function will construct the timeseries.

posterior_df A Stan fit object converted into a data frame

Value

A data frame

Examples

posterior_df <- data.frame('var[1]' = rep(0, 2), 'var[2]' = rep(1, 2), check.names = FALSE)
extract_timeseries_var("var", posterior_df)

inv

Inverse of a number

Description

Inverse of a number

Usage

inv(x)

Arguments

x A real number

Value

A real number

Examples

inv(0.5) # Should return 2
logit  

Logit transformation

Description
Logit transformation

Usage
logit(p)

Arguments
p  A real number that represents a probability

Value
An unconstrained real number

Examples
logit(0.5)

Maryland

Influenza in Maryland during the 1918 pandemic

Description
Influenza in Maryland during the 1918 pandemic

Usage
Maryland

Format
A data frame with 91 rows and 6 columns:

<table>
<thead>
<tr>
<th>Date</th>
<th>Baltimore</th>
<th>Cumberland</th>
<th>Lonaconing</th>
<th>Frederick</th>
<th>Salisbury</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cases reported in the Baltimore</td>
<td>Cases reported in the Cumberland</td>
<td>Cases reported in the Lonaconing</td>
<td>Cases reported in the Frederick</td>
<td>Cases reported in the Salisbury</td>
<td></td>
</tr>
</tbody>
</table>

Source
<https://doi.org/10.2307/4575056>
read xmile

Description

read_xmile returns a list for constructing deSolve functions and graphs.

Usage

read_xmile(filepath, stock_list = NULL, const_list = NULL, graph = FALSE)

Arguments

filepath
A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile.

stock_list
A list in which each element’s name is the name of the stock to override and the element’s value correspond to the new init value.

const_list
A list in which each element’s name is the name of the constant to override and the element’s value correspond to the new value.

graph
A boolean parameter that indicates whether read_xmile returns a graph for the model.

Details

This function extracts the xml from the file specified via filepath to generate a list of objects. Such a list contains a summary of the model, the inputs for simulating through deSolve, and the inputs for creating an igraph object.

Value

This function returns a list with three elements. The first element, description, is a list that contains the simulation parameters, and the names and equations (including graphical functions) for each stock or level, variable and constant. The second element, deSolve_components, is a list that contains initial values, constants and the function for simulating via deSolve. The third element (optional), igraph contains the data.frames for creating a graph with igraph.

Examples

path <- system.file("models", "SIR.stmx", package = "readsdr")
read_xmile(path)
sd_Bayes  
Create Stan file for Bayesian inference

Description
Create Stan file for Bayesian inference

Usage
sd_Bayes(
  filepath,
  meas_mdl,
  estimated_params,
  data_params = NULL,
  data_inits = NULL,
  const_list = NULL,
  forecast = FALSE
)

Arguments
filepath  
A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile

meas_mdl  
A list of strings. Each string corresponds to a sampling statement written in Stan language.

estimated_params  
A list of lists. Each sublist describes each parameter that will be estimated in the inference stage. To construct this description, the user can avail of the function ‘sd_prior’.

data_params  
An optional string vector defining which model parameters will be configured through the Stan data block. That is, the user will provide fixed values for such parameters at every Stan run.

data_inits  
An optional string vector defining which model parameters that only affect initial values (of stocks) will be configured through the Stan data block. That is, the user will provide fixed values for such parameters at every Stan run.

const_list  
A list in which each element’s name is the name of the constant to override and the element’s value correspond to the new value.

forecast  
An optional boolean that indicates whether the Stan file supports a forecast. If TRUE, the data block requires the user to supply an integer value for n_fcst. This variable corresponds to the number of periods that will be predicted.

Value
A string
Negative binomial measurement component

While this package aims to avoid making decisions for users whenever possible, I have taken the liberty to automate the transformation of phi (the concentration parameter) when using the Negative Binomial distribution (alternative parameterisation) as a measurement component. sd_Bayes() automatically creates an inverse phi parameter for computational efficiency, which will be subject to inference (instead of phi). Additionally, I have provided a default prior for this inv_phi but users can override it as needed.

Time

Simulation of the ordinary differential equation (ODE) model starts at time 0.

Examples

filepath <- system.file("models/", "SEIR.stmx", package = "readsdr")
mm1 <- "y ~ neg_binomial_2(net_flow(C), phi)"
meas_mdl <- list(mm1)
estimated_params <- list(
  sd_prior("par_beta", "lognormal", c(0, 1)),
  sd_prior("par_rho", "beta", c(2, 2)),
  sd_prior("I0", "lognormal", c(0, 1), "init"))
sd_Bayes(filepath, meas_mdl, estimated_params)

sd_constants

Summarise the information of a model’s constants in a data frame

Description

Summarise the information of a model’s constants in a data frame

Usage

sd_constants(mdl)

Arguments

mdl A list which is the output from read_xmile.

Value

A data frame.

Examples

path <- system.file("models", "SIR.stmx", package = "readsdr")
mdl <- read_xmile(path)
sd_constants(mdl)
sd_data_generator_fun  Function factory for SBC

Description
Function factory for SBC

Usage
sd_data_generator_fun(
  filepath,
  estimated_params,
  meas_mdl,
  start_time = NULL,
  stop_time = NULL,
  timestep = NULL,
  integ_method = "euler"
)

Arguments
filepath  A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile
estimated_params  A list of lists. Each sublist describes each parameter that will be estimated in the inference stage. To construct this description, the user can avail of the function 'sd_prior'.
meas_mdl  A list of strings. Each string corresponds to a sampling statement written in Stan language.
start_time  A number indicating the time at which the simulation begins.
stop_time  A number indicating the time at which the simulation ends.
timestep  A number indicating the time interval for the simulation. Also known as dt.
integ_method  A string indicating the integration method. It can be either "euler" or "rk4"

Value
A function.

Examples
filepath <- system.file("models/", "SEIR.stmx", package = "readsdr")
meas_mdl <- list("y ~ poisson(net_flow(C))")
estimated_params <- list(
  sd_prior("par_beta", "lognormal", c(0, 1)),
  sd_prior("par_rho", "beta", c(2, 2)),
  sd_prior("I0", "lognormal", c(0, 1), "init")
)
 sd_data_generator_fun(filepath, estimated_params, meas_mdl)
sd_fixed_delay

Description
Fixed delay

Usage
sd_fixed_delay(var, time, delay, init, .memory)

Arguments
- var: A string that indicates the delayed variable.
- time: A number that indicates current simulation time.
- delay: A number that indicates the delay time.
- init: A number that indicates the function’s output value of at the start of the simulation.
- .memory: A data frame that keeps past values of delayed variables.

Value
A number.

Examples
 memoria <- data.frame(time = 3, inflow = 3)
rownames(.memory) <- 3
sd_fixed_delay("inflow", 5, 2, 0, .memory)

sd_impact_inputs

Description
Construct inputs for performing structural analysis via the impact method

Usage
sd_impact_inputs(desc_list)

Arguments
- desc_list: Element 'description' from the list returned by read_xmile()
**Value**

A list of three elements. The first element, `flows`, is a data frame that lists all the stock-flow links in the model. Further, this data frame describes the equation that governs the link and whether the link is an inflow (+) or an outflow (-). The second element, `pathways`, is a data frame that lists all the pathways among stocks. The third element, `velocities`, is a data frame in which each row corresponds to a stock. Each row consists of two columns (name & equation).

**Examples**

```r
filepath <- system.file("models/", "SIR.stmx", package = "readsdr")
mdl <- read_xmile(filepath)
desc_list <- mdl$description
sd_impact_inputs(desc_list)
```

---

**sd_interpret_estimates**

*Interpret estimates*

**Description**

Interpret estimates

**Usage**

```r
sd_interpret_estimates(estimates, par_list)
```

**Arguments**

- `estimates`: A list or data frame
- `par_list`: A list

**Value**

A data frame

**Examples**

```r
estimates <- c(par_beta = 0,
               par_rho = 0.8472979,
               I0 = 0,
               inv_phi = -2.302585)
par_list <- list(list(par_name = "par_beta",
                     par_trans = "exp"),
                 list(par_name = "par_rho",
                      par_trans = "expit"),
                 list(par_name = "I0",
                      par_trans = "exp"),
```
Generate a log-likelihood function for an SD model

Usage

sd_loglik_fun(filepath, unknown_pars, meas_data_mdl, neg_log = FALSE, supplied_pars = NULL, start_time = NULL, stop_time = NULL, timestep = NULL, integ_method = "euler", const_list = NULL)

Arguments

filepath A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xml files. They must be exported from Vensim with extension .xmile.

unknown_pars A list of lists. Each second-level list contains at least the element name name, which corresponds to the parameter's name subject to estimation. In addition to the element name, users can incorporate in the sub-list the elements min and max. The value of min can only be 0, whereas the value of max can only be 1.

meas_data_mdl A list of lists. Each second-level list corresponds to a sampling statement along with its measurements. Here is an example:

list(formula = "y ~ neg_binomial_2(net_flow(C), phi)", measurements = 1:10))

neg_log A boolean that indicates whether the log-likelihood function returns a positive or negative value. If TRUE, the function returns a positive value (for minimisation optimisers). If FALSE, the function returns the original log-likelihood.

supplied_pars A string vector indicating the name of parameters whose values will be supplied to the function. These values will not be subject to optimisation.

start_time A number indicating the time at which the simulation begins.

stop_time A number indicating the time at which the simulation ends.
**timestep**  
A number indicating the time interval for the simulation. Also known as dt.

**integ_method**  
A string indicating the integration method. It can be either "euler" or "rk4".

**const_list**  
A list in which each element’s name is the name of the constant to override and the element’s value correspond to the new value.

**Value**

A list of three elements. The first element, `fun`, corresponds to the log likelihood function. The second element, `par_names`, indicates the order in which the unknowns are returned. The third element, `sim_params`, corresponds to the simulation parameters (start time, stop time, and the integration step or dt) employed by the solver function.

**Examples**

```r
filepath <- system.file("models/", "SEIR.stmx", package = "readsdr")
unknown_pars <- list(list(par_name = "par_beta", min = 0))
meas_data_mdl <- list(list(formula = "y ~ neg_binomial_2(net_flow(C), phi)",
                          measurements = 1:10))
fun_obj <- sd_loglik_fun(filepath, unknown_pars, meas_data_mdl, neg_log = FALSE,
                         start_time = 0, stop_time = 10, timestep = 1/32)
```

---

**sd_measurements**  
Generate measurements

**Description**

Generate measurements

**Usage**

```r
sd_measurements(  
n_meas,  
meas_model,  
ds_inputs,  
start_time = NULL,  
stop_time = NULL,  
timestep = NULL,  
integ_method = "euler"
)
```

**Arguments**

- **n_meas**  
  Number of measurements. An integer.
- **meas_model**  
  Measurement model. A list of strings, in which each string corresponds to a sampling statement in Stan language.
- **ds_inputs**  
  A list of deSolve inputs generated by read_xmile
- **start_time**  
  A number indicating the time at which the simulation begins.
sd_net_change

stop_time  A number indicating the time at which the simulation ends.
timestep   A number indicating the time interval for the simulation. Also known as dt.
integ_method  A string indicating the integration method. It can be either "euler" or "rk4"

Value
A data frame.

Examples
filepath <- system.file("models/", "SEIR.stmx", package = "readsdr")
mdl <- read_xmile(filepath)

mm1 <- "y ~ poisson(C)"
meas_model <- list(mm1)

sd_measurements(n_meas = 2,
meas_model = meas_model,
d_s_inputs = mdl$deSolve_components,
start_time = 0,
stop_time = 10,
timestep = 1/16,
integ_method = "rk4")

ds_net_change(sim_df, cumulative_var)

Description
Estimate the net change of a stock in discrete times

Usage
sd_net_change(sim_df, cumulative_var)

Arguments
sim_df  A data frame with the simulation output
cumulative_var  A string that indicates to which variable the discrete change will be estimated

Value
A data frame.

Examples
test_output <- data.frame(time = seq(0, 2, by = 0.25),
C = c(0, rep(5, 4), rep(20, 4)))
sd_net_change(test_output, "C")
sd_posterior_fun

Description
Posterior function

Usage
sd_posterior_fun(
    filepath,
    meas_data_mdl,
    estimated_params,
    start_time = NULL,
    stop_time = NULL,
    timestep = NULL,
    integ_method = "euler",
    const_list = NULL
)

Arguments
filepath
A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile

meas_data_mdl
A list of lists. Each second-level list corresponds to a sampling statement along with its measurements. Here is an example:

```r
list(formula = "y ~ neg_binomial_2(net_flow(C), phi)", measurements = 1:10))
```

estimated_params
A list of lists. Each sublist describes each parameter that will be estimated in the inference stage. To construct this description, the user can avail of the function 'sd_prior'.

start_time
A number indicating the time at which the simulation begins.

stop_time
A number indicating the time at which the simulation ends.

timestep
A number indicating the time interval for the simulation. Also known as dt.

integ_method
A string indicating the integration method. It can be either "euler" or "rk4"

const_list
A list in which each element’s name is the name of the constant to override and the element’s value correspond to the new value.

Value
A function
Examples

```
filepath <- system.file("models/", "SEIR.stmx", package = "readsdr")
meas_data_mdl <- list(list(formula = "y ~ neg_binomial_2(net_flow(C), phi)",
                         measurements = 1:10))

estimated_params <- list(
    sd_prior("par_beta", "lognormal", c(0, 1)),
    sd_prior("par_rho", "beta", c(2, 2)),
    sd_prior("I0", "lognormal", c(0, 1), "init"))

fun <- sd_posterior_fun(filepath, meas_data_mdl, estimated_params)
```

---

### Description

SD prior

### Usage

```
sd_prior(par_name, dist, dist_pars, type = "constant", min_0 = FALSE)
```

### Arguments

- `par_name`: A string
- `dist`: A string
- `dist_pars`: A vector
- `type`: A string. It can be either 'constant' or 'init'. It is 'constant' by default. 'init' refers to parameters that have only affect the model at time 0.
- `min_0`: An optional boolean indicating whether the prior has a lower bound at zero. In the current implementation, this parameter only has an effect on normal priors.

### Value

A list

### Examples

```
sd_prior("par_beta", "lognormal", c(0, 1))
sd_prior("par_rho", "normal", c(0, 1), min_0 = TRUE)
```
Prior predictive checks

Usage

\[sd\_prior\_checks(\]
\hspace{1em}filepath, \hspace{1em}
\hspace{1em}meas\_mdl, \hspace{1em}
\hspace{1em}estimated\_params, \hspace{1em}
\hspace{1em}n\_draws, \hspace{1em}
\hspace{1em}start\_time = NULL, \hspace{1em}
\hspace{1em}stop\_time = NULL, \hspace{1em}
\hspace{1em}timestep = NULL, \hspace{1em}
\hspace{1em}integ\_method = "euler"\]

Arguments

- filepath: A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile.
- meas_mdl: A list of strings. Each string corresponds to a sampling statement written in Stan language.
- estimated_params: A list of lists. Each sublist describes each parameter that will be estimated in the inference stage. To construct this description, the user can avail of the function `sd_prior`.
- n_draws: An integer that indicates how many time-series will be returned.
- start_time: A number indicating the time at which the simulation begins.
- stop_time: A number indicating the time at which the simulation ends.
- timestep: A number indicating the time interval for the simulation. Also known as dt.
- integ_method: A string indicating the integration method. It can be either "euler" or "rk4".

Value

A list of two data frames.
**Examples**

```r
filepath <- system.file("models/", "SEIR.stmx", package = "readsdr")
meas_mdl <- list("y ~ neg_binomial_2(net_flow(C), phi)")
estimated_params <- list(
  sd_prior("par_beta", "lognormal", c(0, 1)),
  sd_prior("par_rho", "beta", c(2, 2)),
  sd_prior("I0", "lognormal", c(0, 1), "init"))
sd_prior_checks(filepath, meas_mdl, estimated_params, n_draws = 2,
  start_time = 0, stop_time = 5,
  integ_method = "rk4", timestep = 1/32)
```

---

**sd_pulse_s**

Replicate the behaviour of the PULSE function from Stella

**Description**

This function must be placed inside the object that will be passed as the argument `func` to deSolve’s `ode` function.

**Usage**

```r
sd_pulse_s(time, volume, start_p, interval)
```

**Arguments**

- `time` A number
- `volume` A number
- `start_p` A number
- `interval` A number

**Value**

A number

**Examples**

```r
timestep <- function() 0.25 # replicates timestep() from deSolve
sd_pulse_s(2, 1, 2, 0)
```
sd_pulse_train  PULSE TRAIN

Description

PULSE TRAIN

Usage

sd_pulse_train(time, start_pulse, duration_pulse, repeat_pt, end_pulse)

Arguments

time  A numeric argument that indicates the current simulation time
start_pulse  A numeric argument that indicates the start of the pulse
duration_pulse  A numeric argument that indicates the width of the pulse
repeat_pt  A numeric argument that indicates the repetition pattern
end_pulse  A numeric argument that indicates the end of the sequence

Value

1 during the pulse, 0 otherwise.

Examples

sd_pulse_train(5, 5, 3, 10, 20)

sd_pulse_v  Replicate the behaviour of the PULSE function from Vensim

Description

Replicate the behaviour of the PULSE function from Vensim

Usage

sd_pulse_v(time, startPulse, duration)

Arguments

time  A number
startPulse  A number
duration  A number
`sd_sensitivity_run`  

**Value**  
A number

**Examples**

```r  
timestep <- function() 0.25 # replicates timestep() from deSolve  
sd_pulse_v(1, 1, 2)  
```

---

**sd_sensitivity_run**  
Perform a sensitivity run on a System Dynamics model

**Description**

`sd_sensitivity_run` returns a data frame with the simulation of a model for several iterations of different inputs.

**Usage**

```r  
sd_sensitivity_run(  
  ds_inputs,  
  consts_df = NULL,  
  stocks_df = NULL,  
  start_time = NULL,  
  stop_time = NULL,  
  timestep = NULL,  
  integ_method = "euler",  
  multicore = FALSE,  
  n_cores = NULL,  
  reporting_interval = 1  
)
```

**Arguments**

- **ds_inputs**  
  A list of deSolve inputs generated by read_xmile
- **consts_df**  
  A data frame that contains the values of constants to simulate. Each column corresponds to a constant and each row to an iteration. If `stocks_df` is also supplied, both data frames must have the same number of rows.
- **stocks_df**  
  A data frame that contains the initial value of stocks to be explored. Each column corresponds to a stock and each row to an iteration. If `consts_df` is also supplied, both data frames must have the same number of rows.
- **start_time**  
  A number indicating the time at which the simulation begins.
- **stop_time**  
  A number indicating the time at which the simulation ends.
- **timestep**  
  A number indicating the time interval for the simulation. Also known as `dt`.
- **integ_method**  
  A string indicating the integration method. It can be either "euler" or "rk4".
- **multicore**  
  A boolean value that indicates whether the process is parallelised.
sd_simulate

An integer indicating the number of cores for the parallel run.

reporting_interval
A real number indicating the interval at which the simulation results are returned. The default is set to 1. For instance, if the simulation runs from 0 to 10. This function returns the results at times 0, 1, 2, ..., 10.

Value
A data frame

Examples

```r
path <- system.file("models", "SIR.stmx", package = "readsdr")
ds_inputs <- xml_to_deSolve(path)
consts_df <- data.frame(i = c(0.25, 0.30))
sd_sensitivity_run(ds_inputs, consts_df)
```

sd_simulate

Simulate a System Dynamics model

Description
Simulate a System Dynamics model

Usage

```r
sd_simulate(
  ds_inputs,
  start_time = NULL,
  stop_time = NULL,
  timestep = NULL,
  integ_method = "euler"
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds_inputs</td>
<td>A list of deSolve inputs generated by read_xmile</td>
</tr>
<tr>
<td>start_time</td>
<td>A number indicating the time at which the simulation begins.</td>
</tr>
<tr>
<td>stop_time</td>
<td>A number indicating the time at which the simulation ends.</td>
</tr>
<tr>
<td>timestep</td>
<td>A number indicating the time interval for the simulation. Also known as dt.</td>
</tr>
<tr>
<td>integ_method</td>
<td>A string indicating the integration method. It can be either &quot;euler&quot; or &quot;rk4&quot;</td>
</tr>
</tbody>
</table>

Value
A data frame
**sd_stocks**

**Examples**

```r
path <- system.file("models", "SIR.stmx", package = "readsdr")
ds_inputs <- xmile_to_deSolve(path)
sd_simulate(ds_inputs, 0, 1, 0.25, "rk4")
```

---

**sd_stocks**

*Summarise the information of a model’s stocks in a data frame*

**Description**

Summarise the information of a model’s stocks in a data frame

**Usage**

```r
sd_stocks(mdl)
```

**Arguments**

- **mdl**
  - A list which is the output from read_xmile.

**Value**

A data frame.

**Examples**

```r
path <- system.file("models", "SIR.stmx", package = "readsdr")
mdl <- read_xmile(path)
sd_stocks(mdl)
```

---

**sd_what_if_from_time**

*What if from time t we change the value of some parameters*

**Description**

What if from time t we change the value of some parameters

**Usage**

```r
ds_what_if_from_time(
  time,
  up_to_time = Inf,
  par_list,
  ds_inputs,
  start_time = NULL,
  stop_time = NULL,
  timestep = NULL,
  integ_method = "euler"
)
```
stan_ode_function

Arguments

time          Time at which the parameter values change
up_to_time    Time from which the original values are restored.
par_list      A list that indicates which parameters change from time t. For instance, if you wanted to change the value of parameter c to 4, you would provide the list(c = 4)
ds_inputs    A list of deSolve inputs generated by read_xmile
start_time    A number indicating the time at which the simulation begins.
stop_time     A number indicating the time at which the simulation ends.
timestep      A number indicating the time interval for the simulation. Also known as dt.
ingeg_method  A string indicating the integration method. It can be either "euler" or "rk4"

Value

A data frame

Examples

filepath <- system.file("models/", "SIR.stmx", package = "readsdr")
mdl <- read_xmile(filepath)
ds_components <- mdl$deSolve_components
output <- sd_what_if_from_time(3, Inf, list(c = 4), ds_components)

stan_ode_function   Create Stan ODE function

Description

Create Stan ODE function

Usage

stan_ode_function(
  filepath,           <- system.file("models/", "SIR.stmx", package = "readsdr")
  func_name,          <- read_xmile(filepath)
  pars = NULL,        <- mdl$deSolve_components
  const_list = NULL,  <- sd_what_if_from_time(3, Inf, list(c = 4), ds_components)
  extra_funs = NULL,
  XMILE_structure
)

Value

A data frame
xmile_to_deSolve

Arguments

filepath A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile
func_name A string for naming the ODE function
pars A character vector that indicates which constants will be considered as parameters in the ODE function
const_list A list in which each element’s name is the name of the constant to override and the element’s value correspond to the new value.
extra_funs A vector of strings. Each string corresponds to a user-defined function.
XMILE_structure A list.

Value
A string with the code containing a function with the model’s equations in the format required by cmdstan 2.24+

Examples

path <- system.file("models", "SIR.stmx", package = "readsdr")
stan_ode_function(path, "my_model")

Description

xmile_to_deSolve returns a list that serves as an input for deSolve’s ODE function.

Usage

xmile_to_deSolve(filepath)

Arguments

filepath A string that indicates a path to a file with extension .stmx or .xmile. Vensim files (.mdl) are not xmile files. They must be exported from Vensim with extension .xmile

Details
This function extracts the xml from the file specified via filepath to generate a list with the necessary elements to simulate with deSolve.
Value

This function returns a list with at least four elements. *stocks*, a numeric vector that contains initial values. *consts*, a numeric vector with the model’s constants. *func*, the function that wraps the model’s equations. *sim_params*, a list with control parameters. If the model includes a table or graphical function, this function returns the element *graph_funs*, a list with these functions.

Examples

```r
path <- system.file("models", "SIR.stmx", package = "readsdr")
xmile_to_deSolve(path)
```
Index

* datasets
  Maryland, 6
create_stan_function, 2
deSolve, 7, 25
expit, 3
extract_timeseries_stock, 4
extract_timeseries_var, 4
igraph, 7
inv, 5
logit, 6
Maryland, 6
read_xmile, 7
sd_Bayes, 8
sd_constants, 9
sd_data_generator_fun, 10
sd_fixed_delay, 11
sd_impact_inputs, 11
sd_interpret_estimates, 12
sd_loglik_fun, 13
sd_measurements, 14
sd_net_change, 15
sd_posterior_fun, 16
sd_prior, 17
sd_prior_checks, 18
sd_pulse_s, 19
sd_pulse_train, 20
sd_pulse_v, 20
sd_sensitivity_run, 21
sd_simulate, 22
sd_stocks, 23
sd_what_if_from_time, 23
stan_ode_function, 24
xmile_to_deSolve, 25