Package ‘DSAIRM’

February 11, 2019

Type Package

Title Dynamical Systems Approach to Immune Response Modeling

Description A collection of 'shiny' apps that allow for the simulation and
exploration of various within-host immune response scenarios.
The purpose of the package is to help individuals learn
about within-
host infection and immune response modeling from a dynamical systems perspective.
All apps include explanations of the underlying models and instructions on
what to do with the models. The development of this package was partially sup-
ported by NIH grant U19AI117891.

Version 0.5.5

Date 2019-02-11

Maintainer Andreas Handel <ahandel@uga.edu>

License GPL-3

Encoding UTF-8

LazyData TRUE

Imports adaptivetau (>= 2.2), boot (>= 1.3-20), deSolve (>= 1.20),
dplyr (>= 0.7.4), ggplot2 (>= 2.2.1), gridExtra (>= 2.3), lhs
(>= 0.15), nloptr (>= 1.0.4), stats (>= 3.4), utils (>= 3.4),
XML (>= 3.98)

Depends R (>= 3.4), shiny (>= 1.0)

Suggests covr, knitr (>= 1.15), pkgdown, rmarkdown (>= 1.10), testthat

RoxygenNote 6.1.1

VignetteBuilder knitr

URL https://ahgroup.github.io/DSAIRM,
https://github.com/ahgroup/DSAIRM/

BugReports https://github.com/ahgroup/DSAIRM/issues

NeedsCompilation no

Author Andreas Handel [aut, cre] (<https://orcid.org/0000-0002-4622-1146>),
Spencer Hall [ctb],
Sina Solaimanpour [ctb],
Henok Woldu [ctb]

Repository CRAN

Date/Publication 2019-02-11 21:33:37 UTC

R topics documented:

DSAIRM *DSAIRM: A package to learn about Dynamical Systems Approaches to Immune Response Modeling*

Description

The DSAIRM package provides a number of Shiny apps that simulate various within-host infection and immune response dynamics models. By manipulating the models and working through the instructions provided within the Shiny UI, you can learn about some important concepts in immune response modeling. You will also learn how models can be used to study such concepts.

Package Structure

The package is structured in a modular way. Each Shiny app calls an underlying function (which in turn might call other functions). The structure of the package allows you to interact with the models in 3 ways:

1. Start the main menu of the package by calling dsairmmenu(). Pick a Shiny app corresponding to a model/topic, explore it through the corresponding Shiny UI. The UI contains information about the model and a list of tasks to try. This is the main intended use of this package.

2. Call each simulator function directly from the R console, without going through the Shiny app. Each model simulator function is called simulate_XXX and is documented. See the 'Further Information' tab for a given Shiny app to find the names of the different simulation functions.

Calling the functions directly allows you more flexibility. For instance you could write a few lines of extra R code to loop over some model parameter, instead of the manual setting through the sliders in the Shiny app. This gives you more options, but requires being able to write a little bit of R code.

3. Find the code for a simulator function you are interested in and modify it to your needs. This provides the most flexibility in what you can do with this package, and you can end up with any model you need, but for that you need to know or learn some more R coding. To make it easy to get the source code for the simulator functions, they are located in a subdirectory called 'simulator-functions' inside the main package folder.

`dsairmmenu` *The main menu for the DSAIRM package*

Description

This function opens a Shiny app with a menu that will allow the user to run the different simulations.

Usage

```
dsairmmenu()
```

Details

Run this function with no arguments to start the main menu (a Shiny app) for DSAIRM.

Author(s)

Andreas Handel

Examples

```
## Not run: dsairmmenu()
```

generate_documentation — *A helper function for the UI part of the Shiny apps*

Description

This function take the documentation provided as html file and extracts sections to generate the tabs with content for each Shiny app. This is a helper function and only useful for this package.

Usage

```
generate_documentation(docfilename)
```

Arguments

docfilename	of html file containing the documentation

Details

This function is called by the Shiny UIs to populate the documentation and information tabs.

Value

tablist A list of tabs for display in a Shiny UI.

Author(s)

Andreas Handel

generate_plots — *A helper function that takes result from the simulators and produces plots*

Description

This function generates plots to be displayed in the Shiny UI. This is a helper function. This function processes results returned from the simulation, supplied as a list.

Usage

```
generate_plots(res)
```

Arguments

`res` A list structure containing all simulation results that are to be plotted. The length of the list indicates the number of separate plots to make. Each list entry corresponds to one plot and needs to contain the following information/elements:
1. A data frame called "dat" or "ts". If the data frame is "ts" it is assumed to be a time series and by default a line plot will be produced and labeled Time/Numbers. For plotting, the data needs to be in a format with one column called xvals, one column yvals, one column called varnames that contains names for different variables. Varnames needs to be a factor variable or will be converted to one. If a column 'varnames' exist, it is assumed the data is in the right format. Otherwise it will be transformed. An optional column called IDvar can be provided for further grouping (i.e. multiple lines for stochastic simulations). If plottype is 'mixedplot' an additional column called 'style' indicating line or point plot for each variable is needed.
2. Meta-data for the plot, provided in the following variables:
optional: plottype - One of "Lineplot" (default is nothing is provided),"Scatterplot","Boxplot", "Mixedplot".
optional: xlab, ylab - Strings to label axes.
optional: xscale, yscale - Scaling of axes, valid ggplot2 expression, e.g. "identity" or "log10".
optional: xmin, xmax, ymin, ymax - Manual min and max for axes.
optional: makelegend - TRUE/FALSE, if legend should be added to plot. Assume true if not provided.
optional: legendtitle - Legend title, if NULL/not supplied, default is used
optional: legendlocation - if "right" is specified, top right. anythbing else or nothing will place it top left.
optional: linesize - Width of line, numeric, i.e. 1.5, 2, etc. set to 1.5 if not supplied.
optional: title - A title for each plot.

Details

This function is called by the Shiny server to produce plots returned to the Shiny UI. Create plots run the simulation with default parameters just call the function: result <- simulate_basicbacteria()

Value

A plot structure for display in a Shiny UI.

Author(s)

Andreas Handel

`generate_shinyinput` *A helper function that takes a model and generates the shiny UI elements for the analyze tab*

Description

This function generates numeric shiny UI inputs for a supplied model. This is a helper function called by the shiny app.

Usage

```
generate_shinyinput(mbmodel, otherinputs = NULL, packagename)
```

Arguments

`mbmodel`	a name of a function for which to build inputs
`otherinputs`	a list of other shiny inputs to include
`packagename`	name of package using this function

Details

This function is called by the Shiny app to produce the Shiny input UI elements. mbmodel is assumed to be the name of a function which will be parsed to create UI elements. Non-numeric arguments of functions are removed.

Value

A renderUI object that can be added to the shiny output object for display in a Shiny UI

Author(s)

Andreas Handel

`generate_text` *A helper function that takes result from the simulators and produces text output*

Description

This function generates text to be displayed in the Shiny UI. This is a helper function. This function processes results returned from the simulation, supplied as a list.

Usage

```
generate_text(res)
```

Arguments

`res` A list structure containing all simulation results that are to be processed. This function is meant to be used together with generate_plots() and requires similar input information. See the generate_plots() function for most details. Specific entries for this function are 'maketext', 'showtext' and 'finaltext'. If 'maketext' is set to TRUE (or not provided) the function processes the data corresponding to each plot and reports min/max/final values (lineplots) or correlation coefficient (scatterplot) If 'maketext' is FALSE, no text based on the data is generated. If the entries 'showtext' or 'finaltext' are present, their values will be returned for each plot or for all together. The overall message of finaltext should be in the 1st plot.

Details

This function is called by the Shiny server to produce output returned to the Shiny UI.

Value

HTML formatted text for display in a Shiny UI.

Author(s)

Andreas Handel

---

run_model — *A function that runs a DSAIRM/DSAIDE app*

Description

This function takes a model and model settings and runs it. It runs the simulation determined by the model settings and returns simulation results.

Usage

```
run_model(modelsettings, modelfunction)
```

Arguments

`modelsettings` a list with model settings. needs to contain list elements with names and values for all inputs expected by simulation function. Also needs to contain an element plotscale to indicate which axis should be on a log scale (x, y or both), a list element nplots to indicate number of plots that should be produced when calling the generate_plot function with the result, and a list element modeltype which specifies what kind of model should be run. Currently one of (_ode_, _discrete_, _stochastic_, _usanalysis_, _modelexploration_, _fit_). Stochastic models also need an nreps list entry to indicate numer of repeat simulations.

`modelfunction` The name of a simulation function to be run with the indicated settings.

Details

This function runs a model for specific settings. It is similar to analyze_model in the modelbuilder package.

Value

A vectored list named "result" with each main list element containing the simulation results in a dataframe called dat and associated metadata required for generate_plot and generate_text functions. Most often there is only one main list entry (result[[1]]) for a single plot/text.

---

simulate_basicbacteria_discrete
Basic Bacteria model - discrete

Description

A basic bacteria infection model with 2 compartments, implemented as discrete time simulation. The model tracks bacteria and an immune response dynamics. The processes modeled are bacteria growth, death and killing by the immune response, and immune response activation and decay.

Usage

```
simulate_basicbacteria_discrete(B = 10, I = 1, g = 1, Bmax = 1e+06,
  dB = 0.1, k = 1e-07, r = 0.001, dI = 1, tstart = 0,
  tfinal = 30, dt = 0.05)
```

Arguments

B	: starting value for bacteria : numeric
I	: starting value for immune response : numeric
g	: maximum rate of bacteria growth : numeric
Bmax	: bacteria carrying capacity : numeric
dB	: bacteria death rate : numeric
k	: rate of bacteria killing by immune reesponse : numeric
r	: immune response growth rate : numeric
dI	: immune response decay rate : numeric
tstart	: start time of simulation : numeric
tfinal	: final time of simulation : numeric
dt	: time step : numeric

Details

The model includes bacteria and an immune response. The processes are bacteria growth, death and killing by the immune response, and immune response activation and decay. This is a predator-prey type model. The model is implemented as a set of discrete-time, deterministic equations, coded as a for-loop. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

The function returns the output as a list. The time-series from the simulation is returned as a dataframe saved as list element `ts`. The `ts` dataframe has one column per compartment/variable. The first column is time.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. have negative values for parameters), the code will likely abort with an error message.

Examples

```
# To run the simulation with default parameters:
result <- simulate_basicbacteria_discrete()
```

`simulate_basicbacteria_ode` *Basic Bacteria model - ODE*

Description

A basic bacteria infection model with 2 compartments, implemented as set of ODEs. The model tracks bacteria and an immune response dynamics. The processes modeled are bacteria growth, death and killing by the immune response, and immune response activation and decay.

Usage

```
simulate_basicbacteria_ode(B = 10, I = 1, g = 1, Bmax = 1e+05,
  dB = 0.1, k = 1e-06, r = 0.001, dI = 1, tstart = 0,
  tfinal = 30, dt = 0.05)
```

Arguments

`B`	: starting value for bacteria : numeric
`I`	: starting value for immune response : numeric
`g`	: maximum rate of bacteria growth : numeric
`Bmax`	: bacteria carrying capacity : numeric
`dB`	: bacteria death rate : numeric

k	: rate of bacteria killing by immune reesponse : numeric
r	: immune response growth rate : numeric
dI	: immune response decay rate : numeric
tstart	: start time of simulation : numeric
tfinal	: final time of simulation : numeric
dt	: times for which result is returned : numeric

Details

The model includes bacteria and an immune response. The processes are bacteria growth, death and killing by the immune response, and immune response activation and decay. This is a predator-prey type model. The model is implemented as a set of ordinary differential equations (ODE) using the deSolve package. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

The function returns the output as a list. The time-series from the simulation is returned as a dataframe saved as list element `ts`. The `ts` dataframe has one column per compartment/variable. The first column is time.

Notes

The parameter dt only determines the times the solution is returned and plotted, it is not the internal time step for the differential equation solver. The latter is set automatically by the ODE solver.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. have negative values for parameters), the code will likely abort with an error message.

Examples

```
# To run the simulation with default parameters:
result <- simulate_basicbacteria_ode()
# To run the simulation with different parameter or starting values,
# supply the ones you want to change.
# all other parameters will be kept at their default values shown in the function call above
result <- simulate_basicbacteria_ode(B = 100, g = 0.5, dI = 2)
```

simulate_basicmodel_fit

Fitting a simple viral infection models to influenza data

Description

This function runs a simulation of a compartment model using a set of ordinary differential equations. The model describes a simple viral infection system.

Usage

```
simulate_basicmodel_fit(U = 1e+05, I = 0, V = 1, X = 1, n = 0,
  dU = 0, dI = 1, g = 1, p = 10, plow = 0.001, phigh = 1000,
  psim = 10, b = 1e-05, blow = 1e-06, bhigh = 0.001,
  bsim = 1e-04, dV = 2, dVlow = 0.001, dVhigh = 1000, dVsim = 10,
  usesimdata = TRUE, noise = 0.001, iter = 100, solvertype = 1)
```

Arguments

`U`	: initial number of uninfected target cells : numeric
`I`	: initial number of infected target cells : numeric
`V`	: initial number of infectious virions : numeric
`X`	: initial level of immune response : numeric
`n`	: rate of uninfected cell production : numeric
`dU`	: rate at which uninfected cells die : numeric
`dI`	: rate at which infected cells die : numeric
`g`	: unit conversion factor : numeric
`p`	: rate at which infected cells produce virus : numeric
`plow`	: lower bound for p : numeric
`phigh`	: upper bound for p : numeric
`psim`	: rate at which infected cells produce virus for simulated data : numeric
`b`	: rate at which virus infects cells : numeric
`blow`	: lower bound for infection rate : numeric
`bhigh`	: upper bound for infection rate : numeric
`bsim`	: rate at which virus infects cells for simulated data : numeric
`dV`	: rate at which infectious virus is cleared : numeric
`dVlow`	: lower bound for virus clearance rate : numeric
`dVhigh`	: upper bound for virus clearance rate : numeric
`dVsim`	: rate at which infectious virus is cleared for simulated data : numeric
`usesimdata`	: set to TRUE if simulated data should be fitted, FALSE otherwise : logical
`noise`	: noise to be added to simulated data : numeric
`iter`	: max number of steps to be taken by optimizer : numeric
`solvertype`	: the type of solver/optimizer to use (1-3) : numeric

Details

A simple compartmental ODE model mimicking acute viral infection is fitted to data. Data can either be real or created by running the model with known parameters and using the simulated data to determine if the model parameters can be identified. The fitting is done using solvers/optimizers from the nloptr package (which is a wrapper for the nlopt library). The package provides access to a large number of solvers. Here, we only implement 3 solvers, namely 1 = NLOPT_LN_COBYLA, 2 = NLOPT_LN_NELDERMEAD, 3 = NLOPT_LN_SBPLX For details on what those optimizers are and how they work, see the nlopt/nloptr documentation.

Value

The function returns a list containing as elements the best fit time series data frame, the best fit parameters, the data and the final SSR

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values, the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the Shiny app documentation corresponding to this function for more details on this model.

Examples

```
# To run the code with default parameters just call the function:
## Not run: result <- simulate_basicmodel_fit()
# To apply different settings, provide them to the simulator function, like such:
result <- simulate_basicmodel_fit(iter = 5)
```

---

simulate_basicvirus_ode — *Basic Virus model - ODE*

Description

A basic virus infection model with 3 compartments, implemented as ODEs. The model tracks uninfected and infected target cells and free virus. The processes modeled are infection, virus production, uninfected cell birth and death, infected cell and virus death.

Usage

```
simulate_basicvirus_ode(U = 1e+05, I = 0, V = 1, n = 0, dU = 0,
  dI = 1, dV = 2, b = 2e-05, p = 5, g = 1, tstart = 0,
  tfinal = 30, dt = 0.1)
```

Arguments

`U`	: Starting value for uninfected cells : numeric
`I`	: Starting value for infected cells : numeric
`V`	: Starting value for virus : numeric
`n`	: Rate of new uninfected cell replenishment : numeric
`dU`	: Rate at which uninfected cells die : numeric
`dI`	: Rate at which infected cells die : numeric
`dV`	: Rate at which virus is cleared : numeric
`b`	: Rate at which virus infects cells : numeric
`p`	: Rate at which infected cells produce virus : numeric
`g`	: Possible conversion factor for virus units : numeric
`tstart`	: Start time of simulation : numeric
`tfinal`	: Final time of simulation : numeric
`dt`	: Times for which result is returned : numeric

Details

The model is implemented as a set of ordinary differential equations (ODE) using the deSolve package. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

The function returns the output as a list. The time-series from the simulation is returned as a dataframe saved as list element `ts`. The `ts` dataframe has one column per compartment/variable. The first column is time.

Notes

The parameter dt only determines for which times the solution is returned, it is not the internal time step. The latter is set automatically by the ODE solver.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. have negative values for parameters), the code will likely abort with an error message.

Examples

```
# To run the simulation with default parameters:
result <- simulate_basicvirus_ode()
```

`simulate_basicvirus_stochastic` *Stochastic simulation of a compartmental acute virus infection model*

Description

Simulation of a stochastic model with the following compartments: Uninfected target cells (U), Infected cells (I), virus (V).

Usage

```
simulate_basicvirus_stochastic(U = 10000, I = 0, V = 5, n = 0,
  dU = 0, b = 1e-04, dI = 1, p = 10, dV = 2, rngseed = 100,
  tfinal = 30)
```

Arguments

`U`	: initial number of target cells. Needs to be an integer : numeric
`I`	: initial number of wild-type infected cells. Needs to be an integer. : numeric
`V`	: initial number of resistant virus. Needs to be an integer. : numeric
`n`	: rate of uninfected cell production : numeric
`dU`	: rate of uninfected cell removal : numeric
`b`	: level/rate of infection of cells : numeric
`dI`	: rate of infected cell death : numeric
`p`	: virus production rate : numeric
`dV`	: virus removal rate : numeric
`rngseed`	: seed for random number generator to allow reproducibility : numeric
`tfinal`	: Final time of simulation : numeric

Details

A compartmental ID model with several states/compartments is simulated as a stochastic model using the adaptive tau algorithm as implemented by ssa.adaptivetau() in the adaptivetau package. See the manual of this package for more details. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

A list. The list has only one element called ts. ts contains the time-series of the simulation. The 1st column of ts is Time, the other columns are the model variables.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. have I0 > PopSize or any negative values or fractions > 1), the code will likely abort with an error message.

Examples

```
# To run the simulation with default parameters just call the function:
result <- simulate_basicvirus_stochastic()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_basicvirus_stochastic(U = 1e3, dI = 0.1)
# You should then use the simulation result returned from the function, like this:
plot(result$ts[,"time"],result$ts[,"V"],xlab='Time',ylab='Virus',type='l')
```

---

simulate_confint_fit *Fit a simple viral infection model and compute confidence intervals*

Description

This function runs a simulation of a compartment model using a set of ordinary differential equations. The model describes a simple viral infection system.

Usage

```
simulate_confint_fit(U = 1e+05, I = 0, V = 10, n = 0, dU = 0,
  dI = 2, p = 0.01, g = 0, b = 0.01, blow = 1e-06,
  bhigh = 1000, dV = 2, dVlow = 0.001, dVhigh = 1000, iter = 20,
  nsample = 10, rngseed = 100, parscale = "lin")
```

Arguments

`U`	: initial number of uninfected target cells : numeric
`I`	: initial number of infected target cells : numeric
`V`	: initial number of infectious virions : numeric
`n`	: rate of uninfected cell production : numeric
`dU`	: rate at which uninfected cells die : numeric
`dI`	: rate at which infected cells die : numeric
`p`	: rate at which infected cells produce virus : numeric
`g`	: unit conversion factor : numeric
`b`	: rate at which virus infects cells : numeric
`blow`	: lower bound for infection rate : numeric
`bhigh`	: upper bound for infection rate : numeric
`dV`	: rate at which infectious virus is cleared : numeric
`dVlow`	: lower bound for virus clearance rate : numeric
`dVhigh`	: upper bound for virus clearance rate : numeric
`iter`	: max number of steps to be taken by optimizer : numeric
`nsample`	: number of samples for conf int determination : numeric
`rngseed`	: seed for random number generator to allow reproducibility : numeric
`parscale`	: 'lin' or 'log' to fit parameters in linear or log space : character

Details

A simple compartmental ODE model mimicking acute viral infection is fitted to data. Data can either be real or created by running the model with known parameters and using the simulated data to determine if the model parameters can be identified. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

The function returns a list containing the best fit time series, the best fit parameters for the data, the final SSR, and the bootstrapped confidence intervals.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values), the code will likely abort with an error message.

Examples

```
# To run the code with default parameters just call the function:
## Not run: result <- simulate_confint_fit()
# To apply different settings, provide them to the simulator function, like such:
result <- simulate_confint_fit(iter = 5, nsample = 5)
```

---

simulate_drugresistance_stochastic

Stochastic simulation of a compartmental acute virus infection model with treatment and drug resistant strain

---

Description

Simulation of a stochastic model with the following compartments: Uninfected target cells (U), Infected with wild-type/sensitive and untreated (Is), infected with resistant (Ir), wild-type virus (Vs), resistant virus (Vr).

Usage

```
simulate_drugresistance_stochastic(U = 1e+05, Is = 0, Ir = 0,
  Vs = 10, Vr = 0, b = 1e-05, dI = 1, e = 0, m = 1e-04,
  p = 100, c = 4, f = 0.1, rngseed = 100, tfinal = 100)
```

Arguments

`U`	: initial number of target cells : numeric
`Is`	: initial number of wild-type infected cells : numeric
`Ir`	: initial number of resistant infected cells : numeric
`Vs`	: initial number of wild-type virus : numeric

`Vr`	: initial number of resistant virus : numeric
`b`	: level/rate of infection of cells : numeric
`dI`	: rate of infected cell death : numeric
`e`	: efficacy of drug : numeric
`m`	: fraction of resistant mutants created : numeric
`p`	: virus production rate : numeric
`c`	: virus removal rate : numeric
`f`	: fitness cost of resistant virus : numeric
`rngseed`	: seed for random number generator to allow reproducibility : numeric
`tfinal`	: maximum simulation time : numeric

Details

A compartmental ID model with several states/compartments is simulated as a stochastic model using the adaptive tau algorithm as implemented by ssa.adaptivetau in the adpativetau package. See the manual of this package for more details. The function returns the time series of the simulated disease as output matrix, with one column per compartment/variable. The first column is time.

Value

A list. The list has only one element called ts. ts contains the time-series of the simulation. The 1st column of ts is Time, the other columns are the model variables.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. have I0 > PopSize or any negative values or fractions > 1), the code will likely abort with an error message.

Author(s)

Andreas Handel

References

See the manual for the adaptivetau package for details on the algorithm. The implemented model is loosely based on: Handel et al 2007 PLoS Comp Bio "Neuraminidase Inhibitor Resistance in Influenza: Assessing the Danger of Its Generation and Spread"

Examples

```
# To run the simulation with default parameters just call the function:
result <- simulate_drugresistance_stochastic()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_drugresistance_stochastic(tfinal = 200, e = 0.5)
# You should then use the simulation result returned from the function, like this:
plot(result$ts[,"time"],result$ts[,"Vs"],xlab='Time',ylab='Uninfected cells',type='l')
```

simulate_modelcomparison_fit *Fitting 2 simple viral infection models to influenza data*

Description

This function runs a simulation of a compartment model using a set of ordinary differential equations. The model describes a simple viral infection system in the presence of drug treatment. The user provides initial conditions and parameter values for the system. The function simulates the ODE using an ODE solver from the deSolve package. The function returns a matrix containing time-series of each variable and time.

Usage

```
simulate_modelcomparison_fit(U = 1e+05, I = 0, V = 1, X = 1,
  dI = 1, dV = 2, g = 0, p = 10, k = 1e-06, a = 1e-05,
  alow = 1e-06, ahigh = 1e-04, b = 1e-05, blow = 1e-06,
  bhigh = 0.001, r = 1, rlow = 0.1, rhigh = 2, dX = 1,
  dXlow = 0.1, dXhigh = 10, fitmodel = 1, iter = 100)
```

Arguments

U	: initial number of uninfected target cells : numeric
I	: initial number of infected target cells : numeric
V	: initial number of infectious virions : numeric
X	: initial level of immune response : numeric
dI	: rate at which infected cells die : numeric
dV	: rate at which infectious virus is cleared : numeric
g	: unit conversion factor : numeric
p	: rate at which infected cells produce virus : numeric
k	: rate of killing of infected cells by T-cells (model 1) or virus by Ab (model 2) : numeric
a	: activation of T-cells (model 1) or growth of antibodies (model 2) : numeric
alow	: lower bound for activation rate : numeric
ahigh	: upper bound for activation rate : numeric
b	: rate at which virus infects cells : numeric
blow	: lower bound for infection rate : numeric
bhigh	: upper bound for infection rate : numeric
r	: rate of T-cell expansion (model 1) : numeric
rlow	: lower bound for expansion rate : numeric
rhigh	: upper bound for expansion rate : numeric

dX	: rate at which antibodies decay (model 2) : numeric
dXlow	: lower bound for decay rate : numeric
dXhigh	: upper bound for decay rate : numeric
fitmodel	: fitting model 1 or 2 : numeric
iter	: max number of steps to be taken by optimizer : numeric

Details

Two simple compartmental ODE models mimicking acute viral infection with T-cells (model 1) or antibodies (model 2) are fitted to data.

Value

The function returns a list containing the best fit timeseries, the best fit parameters, the data and the AICc for the model.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values), the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the Shiny app documentation corresponding to this function for more details on this model.

Examples

```
# To run the code with default parameters just call the function:
## Not run: result <- simulate_modelcomparison_fit()
# To apply different settings, provide them to the simulator function, like such:
result <- simulate_modelcomparison_fit(iter = 5, fitmodel = 1)
```

---

simulate_modelexploration

Simulation to illustrate parameter scan of the basic bacteria model #'

Description

This function simulates the simple bacteria model ODE for a range of parameters. The function returns a data frame containing the parameter that has been varied and the outcomes (see details).

Usage

```
simulate_modelexploration(B = 10, I = 1, g = 1, Bmax = 1e+06,
  dB = 0.1, k = 1e-07, r = 0.001, dI = 1, tstart = 0,
  tfinal = 200, dt = 0.1, samples = 10, parmin = 1e-08,
  parmax = 1e-05, samplepar = "k", pardist = "lin")
```

Arguments

B	: Starting value for bacteria : numeric
I	: Starting value for immune response : numeric
g	: Maximum rate of bacteria growth : numeric
Bmax	: Bacteria carrying capacity : numeric
dB	: Bacteria death rate : numeric
k	: Bacteria kill rate : numeric
r	: Immune response growth rate : numeric
dI	: Immune response decay rate : numeric
tstart	: Start time of simulation : numeric
tfinal	: Final time of simulation : numeric
dt	: Times for which result is returned : numeric
samples	: Number of values to run between pmin and pmax : numeric
parmin	: Lower value for varied parameter : numeric
parmax	: Upper value for varied parameter : numeric
samplepar	: Name of parameter to be varied : character
pardist	: spacing of parameter values, can be either 'lin' or 'log' : character

Details

##this code illustrates how to do analyze a simple model. A simple 2 compartment ODE model (the simple bacteria model introduced in the app of that name) is simulated for different parameter values. This function runs the simple bacterial infection model for a range of parameters. The user can specify which parameter is sampled, and the simulation returns for each parameter sample the peak and final value for B and I. Also returned is the varied parameter and an indicator if steady state was reached.

Value

The function returns the output as a list, list element 'dat' contains the data frame with results of interest. The first column is called xvals and contains the values of the parameter that has been varied as specified by 'samplepar'. The remaining columns contain peak and steady state values of bacteria and immune response, Bpeak, Ipeak, Bsteady and Isteady. A final boolean variable 'steady' is returned for each simulation. It is TRUE if the simulation reached steady state, otherwise FALSE.

Notes

The parameter dt only determines for which times the solution is returned, it is not the internal time step. The latter is set automatically by the ODE solver.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter values or fractions > 1), the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the shiny app documentation corresponding to this simulator function for more details on this model.

Examples

```
# To run the simulation with default parameters just call the function:
## Not run: res <- simulate_modelexploration()
# To choose parameter values other than the standard one, specify them, like such:
res <- simulate_modelexploration(tfinal=100, samples=5, samplepar='Bmax', parmin=1e1, parmax=1e5)
# You should then use the simulation result returned from the function, like this:
plot(res$dat[,"xvals"],res$data[,"Bpeak"],xlab='Parameter values',ylab='Peak Bacteria',type='l')
```

---

simulate_modelvariants_ode

Simulation of a viral infection model with immune response The simulation illustrates different alternative models.

Description

This function runs a simulation of a compartment model using a set of ordinary differential equations. The user provides initial conditions and parameter values for the system. The function simulates the ODE using an ODE solver from the deSolve package. The function returns a matrix containing time-series of each variable and time.

Usage

```
simulate_modelvariants_ode(U = 1e+05, I = 0, V = 10, F = 0,
  A = 0, n = 0, dU = 0, dI = 1, dV = 4, b = 1e-05, p = 1000,
  pF = 1, dF = 1, f1 = 1e-04, f2 = 0, f3 = 0, Fmax = 1000,
  sV = 1e-10, k1 = 0.001, k2 = 0, k3 = 0, a1 = 1000, a2 = 0,
  a3 = 0, hV = 1e-10, k4 = 0.001, k5 = 0, k6 = 0, sA = 1e-10,
  dA = 0.1, tstart = 0, tfinal = 30, dt = 0.05)
```

Arguments

U	: initial number of uninfected target cells : numeric
I	: initial number of infected target cells : numeric
V	: initial number of infectious virions : numeric
F	: initial level of innate response : numeric
A	: initial level of adaptive response : numeric
n	: rate of uninfected cell production : numeric
dU	: rate of natural death of uninfected cells : numeric
dI	: rate at which infected cells die : numeric
dV	: rate at which infectious virus is cleared : numeric
b	: rate at which virus infects cells : numeric
p	: rate at which infected cells produce virus : numeric
pF	: rate of innate response production in absence of infection : numeric
dF	: rate of innate response removal in absence of infection : numeric
f1	: growth of innate response alternative 1 : numeric
f2	: growth of innate response alternative 2 : numeric
f3	: growth of innate response alternative 3 : numeric
Fmax	: maximum level of innate response in alternative 1 : numeric
sV	: saturation of innate response growth in alternative 2 and 3 : numeric
k1	: action of innate response alternative 1 : numeric
k2	: action of innate response alternative 2 : numeric
k3	: action of innate response alternative 3 : numeric
a1	: growth of adaptive response alternative 1 : numeric
a2	: growth of adaptive response alternative 2 : numeric
a3	: growth of adaptive response alternative 3 : numeric
hV	: saturation of adaptive response growth in alternative 2 and 3 : numeric
k4	: action of adaptive response alternative 1 : numeric
k5	: action of adaptive response alternative 2 : numeric
k6	: action of adaptive response alternative 3 : numeric
sA	: saturation of adaptive response killing for alternative action 2 : numeric
dA	: adaptive immune response decay : numeric
tstart	: Start time of simulation : numeric
tfinal	: Final time of simulation : numeric
dt	: Times for which result is returned : numeric

Details

A compartmental infection model is simulated as a set of ordinary differential equations, using an ode solver from the deSolve package.

Value

The function returns the output from the odesolver as a matrix, with one column per compartment/variable. The first column is time.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values), the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the Shiny app documentation corresponding to this simulator function for more details on this model. See the manual for the deSolve package for details on the underlying ODE simulator algorithm.

Examples

```
# To run the simulation with default parameters just call the function:
result <- simulate_modelvariants_ode()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_modelvariants_ode(V = 100, k1 = 0 , k2 = 0, k3 = 1e-4)
# You should then use the simulation result returned from the function, like this:
plot(result$ts[,"time"],result$ts[,"V"],xlab='Time',ylab='Virus',type='l',log='y')
```

---

simulate_pkpdmodel_ode

PkPd Virus model

Description

This function runs a simulation of the basic 3 compartment virus infection model including the pharmacokinetics and pharmacodynamics of a drug. The user provides initial conditions and parameter values for the system. The function simulates the ODE using an ODE solver from the deSolve package.

Usage

```
simulate_pkpdmodel_ode(U = 1e+07, I = 0, V = 1, n = 0, dU = 0,
  dI = 1, dV = 2, b = 2e-07, g = 1, p = 5, C0 = 2, dC = 0.5,
  C50 = 1, k = 2, Emax = 1, txstart = 10, txinterval = 5,
  tstart = 0, tfinal = 30, dt = 0.1)
```

Arguments

`U`	: initial number of uninfected target cells : numeric
`I`	: initial number of infected target cells : numeric
`V`	: initial number of infectious virions : numeric
`n`	: rate of new uninfected cell replenishment : numeric
`dU`	: rate at which uninfected cells die : numeric
`dI`	: rate at which infected cells die : numeric
`dV`	: rate at which infectious virus is cleared : numeric
`b`	: rate at which virus infects cells : numeric
`g`	: unit conversion factor : numeric
`p`	: rate at which infected cells produce virus : numeric
`C0`	: drug dose given at specified times : numeric
`dC`	: drug concentration decay rate : numeric
`C50`	: drug concentration at which effect is half maximum : numeric
`k`	: steepness of concentration-dependent drug effect : numeric
`Emax`	: maximum drug effect (0-1) : numeric
`txstart`	: time of drug treatment start : numeric
`txinterval`	: time between drug doses : numeric
`tstart`	: Start time of simulation : numeric
`tfinal`	: Final time of simulation : numeric
`dt`	: Times for which result is returned : numeric

Details

A basic virus infection model with drug PkPd

A simple compartmental model is simulated as a set of ordinary differential equations, using an ode solver from the deSolve package. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

A list. The list has only one element called ts. ts contains the time-series of the simulation. The 1st column of ts is Time, the other columns are the model variables.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values), the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the Shiny app documentation corresponding to this simulator function for more details on this model. See the manual for the deSolve package for details on the underlying ODE simulator algorithm.

Examples

```
# To run the simulation with default parameters just call the function:
result <- simulate_pkpdmodel_ode()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_pkpdmodel_ode(V = 100, txstart = 10, n = 1e5, dU = 1e-2)
# You should then use the simulation result returned from the function, like this:
plot(result$ts[,"time"],result$ts[,"V"],xlab='Time',ylab='Virus',type='l',log='y')
```

---

simulate_usanalysis *Simulation to illustrate uncertainty and sensitivity analysis*

Description

This function performs uncertainty and sensitivity analysis using the simple, continuous-time basic bacteria model.

Usage

```
simulate_usanalysis(Bmin = 1, Bmax = 10, Imin = 1, Imax = 10,
  Bmaxmin = 1e+05, Bmaxmax = 1e+06, dBmin = 0.1, dBmax = 0.1,
  kmin = 1e-07, kmax = 1e-07, rmin = 0.001, rmax = 0.001,
  dImin = 1, dImax = 2, gmean = 0.5, gvar = 0.1, samples = 10,
  rngseed = 100, tstart = 0, tfinal = 200, dt = 0.1)
```

Arguments

Bmin	: lower bound for initial bacteria numbers : numeric
Bmax	: upper bound for initial bacteria numbers : numeric
Imin	: lower bound for initial immune response : numeric
Imax	: upper bound for initial immune response : numeric
Bmaxmin	: lower bound for maximum bacteria load : numeric
Bmaxmax	: upper bound for maximum bacteria load : numeric
dBmin	: lower bound for bacteria death rate : numeric
dBmax	: upper bound for bacteria death rate : numeric
kmin	: lower bound for immune response kill rate : numeric
kmax	: upper bound for immune response kill rate : numeric
rmin	: lower bound for immune response growth rate : numeric
rmax	: upper bound for immune response growth rate : numeric

dImin	: lower bound for immune response death rate : numeric
dImax	: upper bound for immune response death rate : numeric
gmean	: mean for bacteria growth rate : numeric
gvar	: variance for bacteria growth rate : numeric
samples	: number of LHS samples to run : numeric
rngseed	: seed for random number generator : numeric
tstart	: Start time of simulation : numeric
tfinal	: Final time of simulation : numeric
dt	: times for which result is returned : numeric

Details

A simple 2 compartment ODE model (the simple bacteria model introduced in the app of that name) is simulated for different parameter values. The user provides ranges for the initial conditions and parameter values and the number of samples. The function does Latin Hypercube Sampling (LHS) of the parameters and runs the basic bacteria ODE model for each sample. Distribution for all parameters is assumed to be uniform between the min and max values. The only exception is the bacteria growth parameter, which is assumed to be gamma distributed with the specified mean and variance. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

The function returns the output as a list. The list element 'dat' contains a data frame. The simulation returns for each parameter sample the peak and final value for B and I. Also returned are all parameter values as individual columns and an indicator stating if steady state was reached. A final variable 'steady' is returned for each simulation. It is TRUE if the simulation did reach steady state, otherwise FALSE.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter values or fractions > 1), the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the Shiny app documentation corresponding to this simulator function for more details on this model.

Examples

```
# To run the simulation with default parameters just call the function:
## Not run: result <- simulate_usanalysis()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_usanalysis(dImin = 0.1, dImax = 10, samples = 5, tfinal = 50)
# You should then use the simulation result returned from the function, like this:
plot(result$dat[,"dI"],result$dat[,"Bpeak"],xlab='values for d',ylab='Peak Bacteria',type='l')
```

---

simulate_virusandir_ode

Simulation of a viral infection model with an immune response

---

Description

This function runs a simulation of a compartment model which tracks uninfected and infected cells, virus, innate immune response, T-cells, B-cells and antibodies. The model is implemented as set of ordinary differential equations using the deSolve package.

Usage

```
simulate_virusandir_ode(U = 1e+05, I = 0, V = 10, T = 0, B = 1,
  A = 0, n = 0, dU = 0, dI = 1, dV = 4, b = 1e-05, p = 1000,
  sF = 0.01, kA = 1e-05, kT = 1e-05, pF = 1, dF = 1, gF = 1,
  Fmax = 1000, hV = 1e-06, hF = 1e-05, gB = 1, gT = 1e-04,
  rT = 0.5, rA = 10, dA = 0.2, tstart = 0, tfinal = 30,
  dt = 0.05)
```

Arguments

U	: initial number of uninfected target cells : numeric
I	: initial number of infected target cells : numeric
V	: initial number of infectious virions : numeric
T	: initial number of T cells : numeric
B	: initial number of B cells : numeric
A	: initial number of antibodies : numeric
n	: rate of new uninfected cell replenishment : numeric
dU	: rate at which uninfected cells die : numeric
dI	: rate at which infected cells die : numeric
dV	: rate at which infectious virus is cleared : numeric
b	: rate at which virus infects cells : numeric
p	: rate at which infected cells produce virus : numeric
sF	: strength of innate response at reducing virus production : numeric
kA	: rate of virus removal by antibodies : numeric

kT	: rate of infected cell killing by T cells : numeric
pF	: rate of innate response production in absence of infection : numeric
dF	: rate of innate response removal in absence of infection : numeric
gF	: rate of innate response growth during infection : numeric
Fmax	: maximum level of innate response : numeric
hV	: innate growth saturation constant : numeric
hF	: B-cell growth saturation constant : numeric
gB	: maximum growth rate of B cells : numeric
gT	: T-cell induction rate : numeric
rT	: T-cell expansion rate : numeric
rA	: rate of antibody production by B cells : numeric
dA	: rate of antibody decay : numeric
tstart	: start time of simulation : numeric
tfinal	: final time of simulation : numeric
dt	: times for which result is returned : numeric

Details

A compartmental infection model is simulated as a set of ordinary differential equations, using an ode solver from the deSolve package. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

A list. The list has only one element, called ts. ts contains the time-series of the simulation. The 1st column of ts is time, the other columns are the model variables.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values), the code will likely abort with an error message.

Author(s)

Andreas Handel

See Also

See the Shiny app documentation corresponding to this simulator function for more details on this model. See the manual for the deSolve package for details on the underlying ODE simulator algorithm.

Examples

```
# To run the simulation with default parameters just call the function:
result <- simulate_virusandir_ode()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_virusandir_ode(V = 100, tfinal = 50, n = 1e5, dU = 1e-2, kT=1e-7)
# You should then use the simulation result returned from the function, like this:
plot(result$ts[,"time"],result$ts[,"V"],xlab='Time',ylab='Virus',type='l',log='y')
```

---

simulate_virusandtx_ode *Simulation of a basic viral infection model with drug treatment*

Description

This function runs a simulation of a compartment model using a set of ordinary differential equations. The model describes a simple viral infection system in the presence of drug treatment. The user provides initial conditions and parameter values for the system. The function simulates the ODE using an ODE solver from the deSolve package. The function returns a list containing timeseries of each variable and time. inspired by a study on HCV and IFN treatment (Neumann et al. 1998, Science)

Usage

```
simulate_virusandtx_ode(U = 1e+05, I = 0, V = 1, n = 10000,
  dU = 0.1, dI = 1, dV = 2, b = 1e-05, p = 10, g = 1, f = 0,
  e = 0, tstart = 0, tfinal = 30, dt = 0.1, steadystate = FALSE,
  txstart = 0)
```

Arguments

U	: initial number of uninfected target cells : numeric
I	: initial number of infected target cells : numeric
V	: initial number of infectious virions : numeric
n	: rate of uninfected cell replenishment : numeric
dU	: rate at which uninfected cells die : numeric
dI	: rate at which infected cells die : numeric
dV	: rate at which infectious virus is cleared : numeric
b	: rate at which virus infects cells : numeric
p	: rate at which infected cells produce virus : numeric
g	: conversion between experimental and model virus units : numeric
f	: strength of cell infection reduction by drug : numeric
e	: strength of virus production reduction by drug : numeric
tstart	: Start time of simulation : numeric

tfinal	: Final time of simulation : numeric
dt	: times for which result is returned : numeric
steadystate	: start simulation at steady state : logical
txstart	: time at which treatment starts : numeric

Details

A simple compartmental model is simulated as a set of ordinary differential equations, using an ode solver from the deSolve package. if the steadystate input is set to TRUE, the starting values for U, I and V are set to their steady state values. Those steady state values are computed from the parameter values. See the Basic Virus Model To-do section for an explanation. In this case, user supplied values for U0, I0, V0 are ignored. This code is part of the DSAIRM R package. For additional model details, see the corresponding app in the DSAIRM package.

Value

A list. The list has only one element called ts. ts contains the time-series of the simulation. The 1st column of ts is Time, the other columns are the model variables.

Warning

This function does not perform any error checking. So if you try to do something nonsensical (e.g. specify negative parameter or starting values), the code will likely abort with an error message.

Examples

```
# To run the simulation with default parameters just call the function:
result <- simulate_virusandtx_ode()
# To choose parameter values other than the standard one, specify them, like such:
result <- simulate_virusandtx_ode(V = 100, tfinal = 100, n = 1e5, dU = 1e-2)
# You should then use the simulation result returned from the function, like this:
plot(result$ts[,"time"],result$ts[,"V"],xlab='Time',ylab='Virus',type='l',log='y')
```

Index