Package ‘seeds’

July 14, 2020

Type  Package
Title  Estimate Hidden Inputs using the Dynamic Elastic Net
Version  0.9.1
Description  Algorithms to calculate the hidden inputs of systems of differential equations. These hidden inputs can be interpreted as a control that tries to minimize the discrepancies between a given model and taken measurements. The idea is also called the Dynamic Elastic Net, as proposed in the paper ``Learning (from) the errors of a systems biology model'' (Engelhardt, Froelich, Kschischo 2016) <doi:10.1038/srep20772>. To use the experimental SBML import function, the 'rsbml' package is required. For installation I refer to the official 'rsbml' page: <https://bioconductor.org/packages/release/bioc/html/rsbml.html>.

Maintainer  Tobias Newmiwaka <tobias.newmiwaka@gmail.com>
URL  https://github.com/Newmi1988/seeds
BugReports  https://github.com/Newmi1988/seeds/issues
Depends  R (>= 3.5.0)

biocViews
Imports  deSolve (>= 1.20), pracma (>= 2.1.4), Deriv (>= 3.8.4), Ryacas, stats, graphics, methods, mvtnorm, matrixStats, statmod, coda, MASS, ggplot2, tidyr, dplyr, Hmisc, R.utils, callr
Suggests  knitr, rmarkdown, rsbml
RoxygenNote  7.1.1
VignetteBuilder  knitr
License  MIT + file LICENSE
Encoding  UTF-8
LazyData  true
NeedsCompilation  no
Author  Tobias Newmiwaka [aut, cre],
        Benjamin Engelhardt [aut]
R topics documented:

seeds-package .............................................. 3
BDEN ........................................................ 4
confidenceBands ............................................ 6
createCompModel ........................................... 7
DEN .......................................................... 7
estiStates .................................................... 10
GIBBS_update ................................................ 11
hiddenInputs .................................................. 12
importSBML .................................................... 12
LOGLIKELIHOOD_func ........................................ 13
MCMC_component .......................................... 14
Model ......................................................... 16
nominalSol .................................................... 16
odeEquations-class ........................................ 17
odeModel-class .............................................. 18
optimal_control_gradient_descent .......................... 19
outputEstimates ............................................ 20
plot,resultsSeeds,missing-method .......................... 21
plotAnno ...................................................... 22
print,resultsSeeds-method .................................. 23
res ............................................................. 23
resultsSeeds-class ........................................... 24
setInitState ................................................... 24
setInput ..................................................... 25
setMeas ...................................................... 26
setMeasFunc .................................................. 27
setModelEquation .......................................... 28
setParms ...................................................... 29
setSd .......................................................... 30
SETTINGS ..................................................... 31
uvbData ...................................................... 32
uvbModel ..................................................... 33

Index ......................................................... 34
seeds-package  

seeds-package  

seeds: Estimate Hidden Inputs using the Dynamic Elastic Net

Description

Algorithms to calculate the hidden inputs of systems of differential equations. These hidden inputs can be interpreted as a control that tries to minimize the discrepancies between a given model and taken measurements. The idea is also called the Dynamic Elastic Net, as proposed in the paper "Learning (from) the errors of a systems biology model" (Engelhardt, Froelich, Kschischo 2016) <doi:10.1038/srep20772>. To use the experimental SBML import function, the 'rsbml' package is required. For installation I refer to the official 'rsbml' page: <https://bioconductor.org/packages/release/bioc/html/rsbml.html>.

Details

The first algorithm (DEN) calculates the needed equations using the Deriv function of the Deriv package. The process is implemented through the use of the S4 class odeEquations-class.

The conjugate gradient based algorithm uses a greedy algorithm to estimate a sparse control that tries to minimize the discrepancies between a given 'nominal model given the measurements (e.g from an experiment). The algorithm the ode uses deSolve to calculate the hidden inputs w based on the adjoint equations of the ODE-System.

The adjoint equations are calculated using the ode function of the deSolve package. For the usage of the algorithm please look into the examples and documentation given for the functions.

The second algorithm is called Bayesian Dynamic Elastic Net (BDEN). The BDEN as a new and fully probabilistic approach, supports the modeler in an algorithmic manner to identify possible sources of errors in ODE based models on the basis of experimental data. THE BDEN does not require pre-specified hyper-parameters. BDEN thus provides a systematic Bayesian computational method to identify target nodes and reconstruct the corresponding error signal including detection of missing and wrong molecular interactions within the assumed model. The method works for ODE based systems even with uncertain knowledge and noisy data.

DEN  a greedy algorithm to calculate a sparse control
BDEN  a basian mcmc approach

Author(s)

Maintainer: Tobias Newmiwaka <tobias.newmiwaka@gmail.com>

Authors:

- Benjamin Engelhardt <engelhar@bit.uni-bonn.de>

References

Benjamin Engelhardt, Holger Froehlich, Maik Kschischo Learning (from) the errors of a systems biology model, *Nature Scientific Reports*, 6, 20772, 2016 <https://www.nature.com/articles/srep20772>
See Also

Useful links:

- https://github.com/Newmi1988/seeds

BDEN

Bayesian Dynamic Elastic Net

Description

Full Bayesian algorithm to detect hidden inputs in ODE based models. The algorithm is an extension of the Dynamic Elastic Net algorithm (Engelhardt et al. 2016) inspired by the Elastic-Net Regression.

Usage

BDEN(
  odeModel,
  settings,
  mcmc_component,
  loglikelihood_func,
  gibbs_update,
  ode_sol,
  NegativeStates = FALSE,
  numbertrialsstep = 15,
  numbertrialsseps = NA,
  numbertrialsinner = 25,
  lambda = 0.001,
  Grad_correct = 0,
  alpha = c(1, 1, 1, 1),
  beta_init = c(1, 1, 1, 1),
  printstatesignore = FALSE
)

Arguments

odeModel a object of class odeModel from the package seeds. The class saves the details of an experiment for easier manipulation and analysis.

settings initial model specific settings (automatically calculated based on the nominal model and data)

mcmc_component sampling algorithm

loglikelihood_func likelihood function

gibbs_update gibbs algorithm
ode_sol  ode solver
NegativeStates  Negative states are allowed
numbertrialsstep
  number of gibbs updates per timepoint. This should be at least 10. Values have
direct influence on the runtime.
numbertrialsseps
  number of samples per mcmc step. This should be greater than numberStates*500. Values
have direct influence on the runtime.
numbertrialinner
  number of inner samples. This should be greater 15 to guarantee a reasonable
exploration of the sample space. Values have direct influence on the runtime.
lambda  initial shrinkage parameter.
Grad_correct  correction factor for initial sigma estimate
alpha  mcmc tuning parameter (weighting of observed states)
beta_init  mcmc tuning parameter (weighting of observed states)
printstatesignore
  states ignored in final output (default = FALSE)

Details

Ordinary differential equations (ODEs) are a popular approach to quantitatively model molecular
networks based on biological knowledge. However, such knowledge is typically restricted.
Wrongly modeled biological mechanisms as well as relevant external influence factors that are not
included into the model likely manifest in major discrepancies between model predictions and
experimental data. Finding the exact reasons for such observed discrepancies can be quite challenging
in practice. In order to address this issue we suggest a Bayesian approach to estimate hidden in-
fluences in ODE based models. The method can distinguish between exogenous and endogenous
hidden influences. Thus, we can detect wrongly specified as well as missed molecular interac-
tions in the model. The BDEN as a new and fully probabilistic approach, supports the modeler
in an algorithmic manner to identify possible sources of errors in ODE based models on the basis
of experimental data. THE BDEN does not require pre-specified hyper-parameters. BDEN thus
provides a systematic Bayesian computational method to identify target nodes and reconstruct the
corresponding error signal including detection of missing and wrong molecular interactions within
the assumed model. The method works for ODE based systems even with uncertain knowledge and
noisy data. In contrast to approaches based on point estimates the Bayesian framework incorporates
the given uncertainty and circumvents numerical pitfalls which frequently arise from optimization
methods (Engelhardt et al. 2017).

For a complete example of the usage take a look into the vignette of the package.

Value

returns a results-object with default plot function

Examples

data(bden_uvb)
confidenceBands <- BDEN(odeModel = Model,
lambda = .001,
beta_init = c(1,1,1,1,1),
numbertrialsstep = 15,
numbertrialseps = 2000,
numbertrialinner = 10)

confidenceBands <- Get the estimated confidence bands for the bayesian method

Description

Get the estimated confidence bands for the bayesian method

Usage

confidenceBands(resultsSeeds, slot, ind)
## S4 method for signature 'list,character,numeric'
confidenceBands(resultsSeeds, slot, ind)
## S4 method for signature 'list,character,missing'
confidenceBands(resultsSeeds, slot, ind)
## S4 method for signature 'resultsSeeds,character,missing'
confidenceBands(resultsSeeds, slot, ind)

Arguments

resultsSeeds A object of the class resultsSeeds, which is returned from the algorithms.
slot Specifies the slot. Options are "states", "hiddenInputs", "outputs"
ind A numeric indicating the index of a resultsSeeds-Object in a list. If not set the last listed object will be used.

Value

A dataframe containing the confidence bands of the estiamted states, hidden inputs and outputs

Examples

data(uvb_res)
confidenceBands(res, slot = "states", ind = 2)
createCompModel

Create compilable c-code of a model

Description

Writes a c file that can be compiled for faster solution with the ode solver. The file created is formatted to be used with the dynamic elastic net. A hidden input is added to every component of the state vector.

Usage

createCompModel(modelFunc, parameters, bden, nnStates)

Arguments

modelFunc  
   a R-function that can be solved with deSolve. External input of the system should be declared with 'u'. To ensure that the function is working use the most general state-space representation.

parameters  
   a vector describing the parameters of the system. If names are missing the function tries to extract the declared parameters from the model function.

bden  
   a boolean that indicates if the c-file is used for the mcmc algorithm, default value is 'FALSE'

nnStates  
   a bit vector indicating the states that should be non negative

Value

None

Note

On the usage of compiled code in conjunction with deSolve take a look into the vignette 'R Package deSolve, Writing Code in Compiled Languages' of the package.

DEN        
Greedy method for estimating a sparse solution

Description

The sparse gradient dynamic elastic net calculates controls based on a first optimization with gradient descent. It should result in a sparse vector of hidden inputs. These hidden inputs try to minimize the discrepancy between a given model and the taken measurements.
Usage

DEN(
  odeModel, alphaStep, Beta, alpha1, alpha2, x0, optW, measFunc, measData, sd, epsilon, parameters, systemInput, modelFunc, greedyLogical, plotEstimates, conjGrad, cString, nnStates, verbose)

Arguments

odeModel a object of class odeModel from the package seeds. The class saves the details of an experiment for easier manipulation and analysis.
alphaStep the starting stepsize for the gradient descent a fitting stepsize will be calculated based on a backtracking line search if the algorithm converges to slow use a bigger stepsize
Beta scaling parameter for the backtracking to approximate the stepsize of the gradient descent. Is set to 0.8 if no value is given to the function
alpha1 L1-norm parameter of the dynamic elastic net approach, is set to zero for this algorithm
alpha2 L2-norm parameter of the dynamic elastic net approach used for regulation purposes
x0 initial state of the ODE system. Can be supplied with the odeModel class.
optW a vector that indicates for which knots of the network a input should be calculated. The default is all nodes.
measFunc a R-Function that is used for measurement of the states if the system is not completely measurable; an empty argument will result in the assumption that all states of the system are measurable. Can be supplied by the odeModel parameter.
measData a table that contains the measurements of the experiment. Used to calculate the needed inputs. Can be supplied with the odeModel class.
sd  
Standard deviation of the measurement. Is used to weight the errors of the estimates in the cost function. Optional parameter. Can be supplied with the odeModel class. Should contain the time in the first column.

epsilon  
Parameter that defines the stopping criteria for the algorithm, in this case percent change in cost function $J[w]$

parameters  
Vector or named vector that contains the parameters of the ODE equation. Can be supplied with the odeModel class.

systemInput  
A dataset that describes the external input of the system. The time steps should be given in the first column for the interpolation.

modelFunc  
a R-Function that states the ODE system for which the hidden inputs should be calculated. Can be supplied with the odeModel class.

greedyLogical  
a boolean that states if the greedy approach should be used; if set to FALSE the algorithm will only use perform a calculation of the inputs for all knots without a sparse solution.

plotEstimates  
Boolean that indicated if the current estimate should be plotted.

conjGrad  
Boolean that indicates the usage of conjugate gradient method over the normal steepest descent. Defaults to true if not specified.

cString  
Optional parameter: A string that represents constants, can be used to calculate a hidden input for a component that gradient is zero.

nnStates  
A bit vector indicating the states that should be non negative. Default behaviour will calculate positive and negative states. Can be supplied with the odeModel class.

verbose  
Boolean indicating if an output in the console should be created to display the gradient descent steps.

Details
This algorithm uses a greedy approach to calculate the hidden inputs. Starting with a first estimation of the hidden inputs the algorithm tries to optimize set of hidden inputs based on the area under the curve from the first run. The algorithm stops if a set of hidden gives a lower cost than a set with additional hidden inputs.

For a complete example of the usage take a look into the vignette of the package.

Value
returns a list of results objects. The default plot function can be used to plot the results.

Examples

```r
data(uvbModel)

results <- DEN(odeModel = uvbModel, alphaStep = 500, alpha2 = 0.0001,
               epsilon = 0.2, plotEstimates = TRUE)
```
estiStates  

Get the estimated states

Description

Get the estimated states

Usage

estiStates(resultsSeeds, ind)

## S4 method for signature 'list,numeric'
estiStates(resultsSeeds, ind)

## S4 method for signature 'list,missing'
estiStates(resultsSeeds, ind)

## S4 method for signature 'resultsSeeds,missing'
estiStates(resultsSeeds, ind)

Arguments

resultsSeeds  

A object of the class resultsSeeds, which is returned from the algorithms.

ind  

A numeric indicating the index of a resultsSeeds-Object in a list. If not set the last listed object will be used.

Value

Dataframe containing the estimated states

Examples

data(uvb_res)
estiStates(res)


**Description**

Algorithm implemented according to Engelhardt et al. 2017. The BDEN defines a conditional Gaussian prior over each hidden input. The scale of the variance of the Gaussian prior is a strongly decaying and smooth distribution peaking at zero, which depends on parameters Lambda2, Tau and Sigma. The parameter Tau is itself given by an exponential distribution (one for each component of the hidden influence vector) with parameters Lambda1. In consequence, sparsity is dependent on the parameter vector Lambda1, whereas smoothness is mainly controlled by Lambda2. These parameters are drawn from hyper-priors, which can be set in a non-informative manner or with respect to prior knowledge about the degree of shrinkage and smoothness of the hidden influences (Engelhardt et al. 2017).

**Usage**

```r
GIBBS_update(D, EPS_inner, R, ROH, SIGMA_0, n, SIGMA, LAMBDA2, LAMBDA1, TAU)
```

**Arguments**

- `D`: diagonal weight matrix of the current Gibbs step
- `EPS_inner`: row-wise vector of current hidden influences \([tn,tn+1]\)
- `R`: parameter for needed for the Gibbs update (for details see Engelhardt et al. 2017)
- `ROH`: parameter for needed for the Gibbs update (for details see Engelhardt et al. 2017)
- `SIGMA_0`: prior variance of the prior for the hidden influences
- `n`: number of system states
- `SIGMA`: current variance of the prior for the hidden influences (calculated during the Gibbs update)
- `LAMBDA2`: current parameter (smoothness) needed for the Gibbs update (for details see Engelhardt et al. 2017)
- `LAMBDA1`: current parameter (sparsity) needed for the Gibbs update (for details see Engelhardt et al. 2017)
- `TAU`: current parameter (smoothness) needed for the Gibbs update (for details see Engelhardt et al. 2017)

**Details**

The function can be replaced by an user defined version if necessary

**Value**

A list of updated Gibbs parameters; i.e. Sigma, Lambda1, Lambda2, Tau
**hiddenInputs**

*Get the estimated hidden inputs*

**Description**

Get the estimated hidden inputs

**Usage**

```r
hiddenInputs(resultsSeeds, ind)
```

## S4 method for signature 'list,numeric'

```r
hiddenInputs(resultsSeeds, ind)
```

## S4 method for signature 'list,missing'

```r
hiddenInputs(resultsSeeds, ind)
```

## S4 method for signature 'resultsSeeds,missing'

```r
hiddenInputs(resultsSeeds, ind)
```

**Arguments**

- `resultsSeeds` A object of the class 'resultsSeeds', which is returned from the algorithms.
- `ind` A numeric indicating the index of a 'resultsSeeds'-Object in a list. If not set the last listed object will be used.

**Value**

Dataframe containing the estimated hidden inputs

**Examples**

```r
data(uvb_res)
hiddenInputs(res[[2]])
```

**importSBML**

*Import SBML Models using the Bioconductor package 'rsbml'*

**Description**

A simple function for importing sbml models from a extensive markup language file.
Usage

importSBML(filename, times, meas_input)

Arguments

filename name of the import file. Should be located in the working directory.
times timestep at which the function should be evaluated
meas_input measurements have to be given in order to analyze the data

Value

returns a odeModel object

Examples

## Not run:

```r
t <- uvbData[,1]
y <- uvbData[,1:3]
modelFile <- system.file("extdata","BIOMD0000000545_url.xml", package = "seeds")

# generate an odeModel object
uvb <- importSBML(modelFile, times = t, meas = y)
```

## End(Not run)

---

**LOGLIKELIHOOD_func**

Calculates the Log Likelihood for a new sample given the current state (i.e. \( \log[ L(G|x)P(G) ] \))

Description

Algorithm implemented according to Engelhardt et al. 2017. The function can be replaced by an user defined version if necessary.

Usage

LOGLIKELIHOOD_func(
pars,
Step,
OBSERVATIONS,
x_0,
parameters,
EPS_inner,
)
MCMC_component

\begin{verbatim}
INPUT, 
D, 
GIBBS_PAR, 
k, 
MU_JUMP, 
SIGMA_JUMP, 
eps_new, 
objectivfunc 
)

Arguments

pars sampled hidden influence for state k (w_new) at time tn+1
Step time step of the sample algorithm corresponding to the given vector of time points
OBSERVATIONS observed values at the given time step/point
x_\theta initial values at the given time step/point
parameters model parameters estimates
EPS_inner current hidden inputs at time tn
INPUT discrete input function e.g. stimuli
D diagonal weight matrix of the current Gibbs step
GIBBS_PAR GIBBS_PAR["BETA"] and GIBBS_PAR["ALPHA"]; prespecified or calculated vector of state weights
k number state corresponding to the given hidden influence (w_new)
MU_JUMP mean of the normal distributed proposal distribution
SIGMA_JUMP variance of the normal distributed proposal distribution
eps_new current sample vector of the hidden influences (including all states)
objectivfunc, link function to match observations with modeled states

Value

returns the log-likelihood for two given hidden inputs
\end{verbatim}

---

MCMC_component | Componentwise Adapted Metropolis Hastings Sampler

Description

Algorithm implemented according to Engelhardt et al. 2017.
MCMC_component

Usage

MCMC_component(
    LOGLIKELIHOOD_func,
    STEP_SIZE,  
    STEP_SIZE_INNER,  
    EPSILON,  
    JUMP_SCALE,  
    STEP,  
    OBSERVATIONS,
    Y0,  
    INPUTDATA,  
    PARAMETER,  
    EPSILON_ACT,  
    SIGMA,  
    DIAG,  
    GIBBS_par,  
    N,  
    BURNIN,  
    objective
)

Arguments

LOGLIKELIHOOD_func
    likelihood function

STEP_SIZE
    number of samples per mcmc step. This should be greater than numberStates*500. Values have direct influence on the runtime.

STEP_SIZE_INNER
    number of inner samples. This should be greater 15 to guarantee a reasonable exploration of the sample space. Values have direct influence on the runtime.

EPSILON
    vector of hidden influences (placeholder for customized version)

JUMP_SCALE
    ODE system

STEP
    time step of the sample algorithm corresponding to the given vector of time points

OBSERVATIONS
    observed state dynamics e.g. protein concentrations

Y0
    initial values of the system

INPUTDATA
    discrete input function e.g. stimuli

PARAMETER
    model parameters estimates

EPSILON_ACT
    vector of current hidden influences

SIGMA
    current variance of the prior for the hidden influences (calculated during the Gibbs update)

DIAG
    diagonal weight matrix of the current Gibbs step

GIBBS_par
    GIBBS_PAR["BETA"] and GIBBS_PAR["ALPHA"]; prespecified or calculated vector of state weights
nominalSol

The function can be replaced by an user defined version if necessary.

Value

A matrix with the sampled hidden inputs (row-wise)

Model

Test dataset for demonstrating the bden algorithm.

Description

Dataset is identical with the example for the bden algorithm from the vignette. It contains an object of odeModel that describes the uvb network.

Usage

data(bden_uvb)

Format

An object of class odeModel of length 1.

nominalSol

Calculate the nominal solution of the model

Description

After an model is defined it can be evaluated. This returns the numerical solution for the state equation before hidden inputs are calculated.

Usage

nominalSol(odeModel)

## S4 method for signature 'odeModel'
nominalSol(odeModel)

Arguments

odeModel a object of the class ode model describing the experiment
odeEquations-class

A S4 class used to handle formatting ODE-Equation and calculate the needed functions for the seeds-algorithm

Value

Returns a s4 class object containing the needed equations for the costate equation

Slots

modelStr  a vector of strings describing the ODE
measureStr a vector of strings representing the equation of the measurement function
origEq    a vector of strings containing the original model function
measureFunction  a vector of strings containing the original measurement function
costateEq  a vector of strings describing the costate equation
JhT       a matrix of strings describing the jacobian matrix of the measurement function
jacobian   a matrix of strings representing the jacobian matrix model equations
costFunction a string containing the cost function
hamiltonian a string representing the Hamilton function of the model
odeModel-class

A class to store the important information of an model.

Description

The slots are used to store the important information of an model. The class is used to create object for the two algorithms implemented in seeds. Methods are implemented to easily calculate the nominal solution of the model and change the details of the saved model. The numerical solutions are calculated using the deSolve - package.

Value

an object of class odeModel which defines the model

Slots

func  A function containing the ode-equations of the model. For syntax look at the given examples of the deSolve package.
times  timesteps at which the model should be evaluated
parms  the parameters of the model
input  matrix containing the inputs with the time points
measFunc  function that converts the output of the ode solution
y  initial (state) values of the ODE system, has to be a vector
meas  matrix with the (experimental) measurements of the system
sd  optional standard deviations of the measurements, is used by the algorithms as weights in the costfunction
custom  customized link function
nnStates  bit vector that indicates if states should be observed by the root function
nnTollerance  tolerance at which a function is seen as zero
resetValue  value a state should be set to by an event
optimal_control_gradient_descent

estimating the optimal control using the dynamic elastic net

Description

estimating the optimal control using the dynamic elastic net

Usage

optimal_control_gradient_descent(
  alphaStep,
  armijoBeta,
  x0,
  parameters,
  alpha1,
  alpha2,
  measData,
  constStr,
  SD,
  modelFunc,
  measFunc,
  modelInput,
  optW,
  origAUC,
  maxIteration,
  plotEsti,
  conjGrad,
  eps,
  nnStates,
  verbose
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphaStep</td>
<td>starting value of the stepsize for the gradient descent, will be calculate to minimize the cost function by backtracking algorithm</td>
</tr>
<tr>
<td>armijoBeta</td>
<td>scaling of the alphaStep to find a approximately optimal value for the stepsize</td>
</tr>
<tr>
<td>x0</td>
<td>initial state of the ode system</td>
</tr>
<tr>
<td>parameters</td>
<td>parameters of the ODE-system</td>
</tr>
<tr>
<td>alpha1</td>
<td>L1 cost term scalar</td>
</tr>
<tr>
<td>alpha2</td>
<td>L2 cost term scalar</td>
</tr>
<tr>
<td>measData</td>
<td>measured values of the experiment</td>
</tr>
<tr>
<td>constStr</td>
<td>a string that represents constrains, can be used to calculate a hidden input for a component that gradient is zero</td>
</tr>
</tbody>
</table>
### outputEstimates

**SD** standard deviation of the experiment; leave empty if unknown; matrix should contain the timesteps in the first column

**modelFunc** function that describes the ODE-system of the model

**measFunc** function that maps the states to the outputs

**modelInput** an dataset that describes the external input of the system

**optW** vector that indicated at which knots of the network the algorithm should estimate the hidden inputs

**origAUC** AUCs of the first optimization; only used by the algorithm

**maxIteration** a upper bound for the maximal number of iterations

**plotEsti** boolean that controls of the current estimates should be plotted

**congGrad** boolean that indicates the usage of conjugate gradient method over the normal steepest descent

**eps** criteria for stopping the algorithm

**nnStates** a bit vector indicating the states that should be non negative

**verbose** Boolean indicating if an output in the console should be created to display the gradient descent steps

### Value

A list containing the estimated hidden inputs, the AUCs, the estimated states and resulting measurements and the cost function

### Description

Get the estimated outputs

### Usage

```r
outputEstimates(resultsSeeds, ind)
```

## S4 method for signature 'list,numeric'

```r
outputEstimates(resultsSeeds, ind)
```

## S4 method for signature 'list,missing'

```r
outputEstimates(resultsSeeds, ind)
```

## S4 method for signature 'resultsSeeds,missing'

```r
outputEstimates(resultsSeeds, ind)
```
Arguments

resultsSeeds  A object of the class 'resultsSeeds', which is returned from the algorithms.
ind  A numeric indicating the index of a 'resultsSeeds'-Object in a list. If not set the last listed object will be used.

Value

Dataframe with estimated measurements.

Examples

data(uvb_res)
outputEstimates(res[[2]])

plot,resultsSeeds,missing-method

Plot method for the S4 class resultsSeeds

Description

A standardized plot function to display the results of the algorithms. Both algorithms should result in objects of the class resultsSeeds. The results can be plotted using the plot-function.

Usage

## S4 method for signature 'resultsSeeds,missing'
plot(x, y)

Arguments

x  an object of type resultsSeeds or a list of these objects. If a list is given the last entry will be plotted.
y  ...

Value

A list of plots showing the results of the algorithm

Examples

data(uvb_res)
plot(res[[2]])
plotAnno

Create annotated plot

Description

Create a annotated plot with given state and measurement names. The plots are equal to the output of the normal plot function.

Usage

plotAnno(x, stateAnno, measAnno)

## S4 method for signature 'resultsSeeds'
plotAnno(x, stateAnno, measAnno)

## S4 method for signature 'list'
plotAnno(x, stateAnno, measAnno)

Arguments

x an object of type resultsSeeds which contains the results of the algorithms
stateAnno a character vector describing the names of the states
measAnno a character vector describing the names of the measurements

Value

Plots of the results with the provided annotation

Examples

data(uvb_res)

statesAnno <- c("x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9", "x10", "x11", "x12", "x13")
measAnno <- c("y1", "y2", "y3", "y4", "y5")

plotAnno(res[[2]], stateAnno = statesAnno, measAnno = measAnno)
print.resultsSeeds-method

A default printing function for the resultsSeeds class

Description
This function overwrites the default print function and is used for objects of the class resultsSeeds. The print function gives the basic information about the results seeds object. The default printout is the estimated states and the calculated hidden inputs.

Usage
## S4 method for signature 'resultsSeeds'
print(x)

Arguments

x
an object of the class resultsSeeds

Value
Returns a short summary of the important results

Examples

data(ubv_res)

plot(res[[2]])

res Results from the uvb dataset for examples

Description
Data from running the estimation of hidden inputs from the UVB-G Protein demo. This data is used for demonstration the different functions of the package.

Usage
data(ubv_res)

Format
An object of class list of length 2.
resultsSeeds-class  Results Class for the Algorithms

Description

A S4 class that collects the results of the two algorithms. The class also is equipped with functions for easily plotting and extracting the different results.

Value

A object of class resultsSeeds collecting all the results of the algorithm

Slots

stateNominal  data.frame containing the states of the nominal model
stateEstimates  data.frame containing the state estimates
stateUnscertainLower  lower bound of the estimated states as calculated by the baysian method
stateUnscertainUpper  upper bound of the estimated states as calculated by the baysian method
hiddenInputEstimates  estimated hidden input
hiddenInputUncertainLower  lower bounds of the estimated hidden inputs
hiddenInputUncertainUpper  upper bounds of the estimated hidden inputs
outputEstimates  estimated measurements resulting from the control of the hidden inputs
outputEstimatesUncLower  lower bound of the confidence bands of the estimated output
outputEstimatesUncUpper  upper bound of the confidence bands of the estimated output
Data  the given measurements
DataError  standard deviation of the given measurements

setInitState  Set the vector with the initial (state) values

Description

Set the vector with the initial (state) values

Usage

setInitState(odeModel, y)

## S4 method for signature 'odeModel'
setInitState(odeModel, y)
setInput

Arguments

odeModel      an object of the class odeModel
y             vector with the initial values

Value

an object of odeModel

Examples

data("uvbModel")
x0 = c(0.2,10,2,0,20,0,0,0,0,4.2,0.25,20,0)
newModel <- setInitState(uvbModel, y = x0)
Examples

data("uvbModel")

model_times <- uvbModel@times
input <- rep(0,length(model_times))

input_Dataframe <- data.frame(t = model_times, u = input)

newModel <- setInput(odeModel = uvbModel,input = input_Dataframe)

---

setMeas

set measurements of the model

Description

The odeModel object stores all important information. Measurements of the objects can be set directly by addressing the slot, or with this function.

Usage

setMeas(odeModel, meas)

### S4 method for signature 'odeModel'
setMeas(odeModel, meas)

Arguments

odeModel an object of the class odeModel
meas measurements of the model, a matrix with measurements of the model and the corresponding time values

Value

an object of odeModel

Examples

data(uvbData)
data(uvbModel)

measurements <- uvbData[,1:6]

newModel <- setMeas(odeModel = uvbModel, meas = measurements)
Description

For a given model a measurement equation can be set. If no measurement function is set the states become the output of the system. The function should be defined as in the example below.

Usage

```r
setMeasFunc(odeModel, measFunc, custom)
```

## S4 method for signature 'odeModel,'function',missing'
setMeasFunc(odeModel, measFunc, custom)

## S4 method for signature 'odeModel,'function',logical'
setMeasFunc(odeModel, measFunc, custom)

Arguments

- **odeModel**: an object of the class odeModel
- **measFunc**: measurement function of the model. Has to be a R functions.
- **custom**: custom indexing for the measurement function (used by the baysian method)

Value

- an object of odeModel

Examples

```r
data("uvbModel")

uvbMeasure <- function(x) {
  y1 = 2*x[,5] + x[,4] + x[,8]
  y2 = 2*x[,5] + 2* x[,3] + x[,1]
  y3 = x[,6]
  y4 = x[,11]
  y5 = x[,4]

  return(cbind(y1,y2,y3,y4,y5))
}

newModel <- setMeasFunc(odeModel = uvbModel, measFunc = uvbMeasure)
```
setModelEquation

Set the model equation

Description

Set the model equation of the system in an odeModel object. Has to be a function that can be used with the deSolve package.

Usage

setModelEquation(odeModel, func)

## S4 method for signature 'odeModel'
setModelEquation(odeModel, func)

Arguments

odeModel an object of the class odeModel
func function describing the ode equation of the model

Value

an object of odeModel

Examples

data("uvbModel")

uvbModelEq <- function(t,x,parameters) {
  with (as.list(parameters),{
    dx1 = ((-2) * ((ka1 * (x[1]^2) * (x[4]^2)) - (kd1 * x[5])) +
           (-2) * ((ka2 * (x[1]^2) * x[2]) - (kd2 * x[3])) +
           ((ks1 *((1) + (uv * n3 * (x[11] + fhy3_s)))) -
            (kdr1 * ((1) + (n1 * uv)) * x[13])))
    dx2 = ((-1) * ((ka2*(x[1]^2) * x[2]) - (kd2 * x[3])) +
           (-1) * ((ka4 * x[2] * x[12]) - (kd4 * x[13])))
    dx3 = (((ka2 * (x[1]^2) * x[2]) - (kd2* x[3])))
    dx4 = ((-2) * (k1*(x[4]^2)) + (2) * (k2 * x[6]) +
           (-2) * ((ka1 * (x[1]^2)* (x[4]^2)) - (kd1 * x[5])) +
           (-1)* (ka3 * x[4] *x[7]))
    dx5 = (((ka1 * (x[1]^2) * (x[4]^2)) -(kd1 * x[5])))
    dx6 = ((-1) * (k2 * x[6]) + (k1 * (x[4]^2)) +(kd3 * (x[8]^2)))
    dx7 = (((-1) * (ka3 * x[4] * x[7]) + (ks2 * (((1) + (uv * x[5]))) -
            (kdr2 * x[7])) + (2) * (kd3 * (x[8]^2)))
    dx8 = ((-2) * (kd3 * x[8]^2) + (ka3 * x[4] * x[7]))
    dx9 = 0
    dx10 = 0
    dx11 = (((ks3 * (((1) + (n2 * uv))) - (kdr3 * (((x[3] / (kdr3a + x[3])) +

[... continued code output ...]
\[
\begin{align*}
\end{align*}
\]
\[
dx12 = ((-1) * (ka4 * x[2] * x[12]) + (kd4 * x[13]))
\]
\[
dx13 =((ka4 * x[2] * x[12]) - (kd4 * x[13]))
\]

\[
\text{list}(c(dx1,dx2,dx3,dx4,dx5,dx6,dx7,dx8,dx9,dx10,dx11,dx12,dx13))
\]

```r
setModelEquation(uvbModel, uvbModelEq)
```

---

**setParms**

*Set the model parameters*

**Description**

A method to set the model parameters of an odeModel object.

**Usage**

```r
setParms(odeModel, parms)
```

## S4 method for signature 'odeModel,numeric'

```r
setParms(odeModel, parms)
```

**Arguments**

- `odeModel` an object of the class odeModel
- `parms` a vector containing the parameters of the model

**Value**

an object of odeModel

**Examples**

```r
data("uvbModel")
newParas <- c(k1=0.23,
ks1=4.0526,
kdr1=0.1,
kdr2=0.2118,
k1=0.0043,
k2=161.62,
ka1=0.0372,
ka2=0.0611,
ka3=4.7207,
k1=94.3524,
k2=50.6973,
```
kd3=0.5508,
ks3=0.4397,
kdr3=1.246,
uv=1,
ka4=10.1285,
kd4=1.1999,
n1=3,
n2=2,
n3=3.5,
kdr3a=0.9735,
kdr3b=0.406,
ksr=0.7537,
fhy3_s=5)

ewModel <- setParms(odeModel = uvbModel, parms = newParas)

---

### setSd

**Set the standard deviation of the measurements**

**Description**

With multiple measurements a standard deviation can be calculated for every point of measurement. The standard deviation is used to weigh the estimated data points in the cost function.

**Usage**

```r
setSd(odeModel, sd)
```

```r
## S4 method for signature 'odeModel'
setSd(odeModel, sd)
```

**Arguments**

- **odeModel**
  - an object of the class odeModel

- **sd**
  - a matrix with the standard deviations of the measurements

**Value**

an object of odeModel

**Examples**

```r
data(uvbData)
data(uvbModel)

sd_uvb <- uvbData[,7:11]
```
```r
newModel <- setSd(odeModel = uvbModel, sd = sd_uv)
```

## Automatic Calculation of optimal Initial Parameters

### Description

Implemented according to Engelhardt et al. 2017.

### Usage

```r
SETTINGS(
  VARIANCE,
  N,
  BETA_LAMDBA,
  alphainit,
  betainit,
  R = c(1000, 1000),
  ROH = c(10, 10)
)
```

### Arguments

- **VARIANCE**: standard error of the observed stat dynamics (per time point)
- **N**: number of system states
- **BETA_LAMDBA**: mcmc tuning parameter (weighting of observed states)
- **alphainit**: mcmc tuning parameter (weighting of observed states)
- **betainit**: mcmc tuning parameter (weighting of observed states)
- **R**: mcmc tuning parameter
- **ROH**: mcmc tuning parameter

### Details

The function can be replaced by an user defined version if necessary.

### Value

A list of optimal initial parameters; i.e. R, Roh, Alpha, Beta, Tau, Lambda1, Lambda2
Description

A data frame containing simulated values of the UVB Signaling pathway. The error of the system is synthetic and is added to the states x3 and x11. The model is taken from the works of Ouyang et al. [https://doi.org/10.1073/pnas.1412050111](https://doi.org/10.1073/pnas.1412050111)

Usage

uvbData

Format

An object of class \texttt{data.frame} with 8 rows and 11 columns.

Details

A data frame with 8 rows and 11 columns

- \texttt{t} time in fractions of an hour
- \texttt{y1} total amounts of UVR8 monomers
- \texttt{y2} total amounts of COP1 monomers
- \texttt{y3} total amounts of UVR8 dimers
- \texttt{y4} concentration of elongated hypocotyl 5 (HY5) protein
- \texttt{y5} concentration measured of UVR8 monomers
- \texttt{y1std} standard deviation of the first measurement
- \texttt{y2std} standard deviation of the second measurement
- \texttt{y3std} standard deviation of the third measurement
- \texttt{y4std} standard deviation of the fourth measurement
- \texttt{y5std} standard deviation of the fifth measurement

Source

[https://doi.org/10.1073/pnas.1412050111](https://doi.org/10.1073/pnas.1412050111)
uvbModel

An object of the odeModel Class

Description

Object is used for demonstrating the functions of the odeModel Class. It is used in the demos for the uvb signaling pathway.

Usage

data(uvbModel)

Format

An object of class odeModel of length 1.
Index

* datasets
  - Model, 16
  - res, 23
  - uvbData, 32
  - uvbModel, 33

BDEN, 3, 4

confidenceBands, 6
confidenceBands, list, character, missing-method
  (confidenceBands), 6
confidenceBands, list, character, numeric-method
  (confidenceBands), 6
confidenceBands, resultsSeeds, character, missing-method
  (confidenceBands), 6
createCompModel, 7

DEN, 3, 7
Deriv, 3

estiStates, 10
estiStates, list, missing-method
  (estiStates), 10
estiStates, list, numeric-method
  (estiStates), 10
estiStates, resultsSeeds, missing-method
  (estiStates), 10

GIBBS_update, 11

hiddenInputs, 12
hiddenInputs, list, missing-method
  (hiddenInputs), 12
hiddenInputs, list, numeric-method
  (hiddenInputs), 12
hiddenInputs, resultsSeeds, missing-method
  (hiddenInputs), 12

importSBML, 12

LOGLIKELIHOOD_func, 13

MCMC_component, 14
Model, 16

nominalSol, 16
nominalSol, odeModel-method
  (nominalSol), 16
ode, 3, 7
odeEquations (odeEquations-class), 17
odeEquations-class, 17
odeModel, 8
odeModel (odeModel-class), 18
odeModel-class, 18
optimal_control_gradient_descent, 19
outputEstimates, 20
outputEstimates, list, missing-method
  (outputEstimates), 20
outputEstimates, list, numeric-method
  (outputEstimates), 20
outputEstimates, resultsSeeds, missing-method
  (outputEstimates), 20

plot, 21
plot, resultsSeeds, missing-method, 21
plotAnno, 22
plotAnno, list-method (plotAnno), 22
plotAnno, resultsSeeds-method
  (plotAnno), 22
print, resultsSeeds
  (print, resultsSeeds-method), 23
print, resultsSeeds-method, 23

res, 23
resultsSeeds (resultsSeeds-class), 24
resultsSeeds-class, 24

seeds (seeds-package), 3
seeds-package, 3
setInitState, 24
setInitState, odeModel-method
  (setInitState), 24
setInput, 25
setInput, odeModel-method (setInput), 25
setMeas, 26
setMeas, odeModel-method (setMeas), 26
setMeasFunc, 27
setMeasFunc, odeModel, function, logical-method (setMeasFunc), 27
setMeasFunc, odeModel, function, missing-method (setMeasFunc), 27
setModelEquation, 28
setModelEquation, odeModel-method (setModelEquation), 28
setParms, 29
setParms, odeModel, numeric-method (setParms), 29
setSd, 30
setSd, odeModel-method (setSd), 30
SETTINGS, 31

uvbData, 32
uvbModel, 33