Package ‘pauwels2014’

February 20, 2015

Type Package

Title Bayesian Experimental Design for Systems Biology.

Version 1.0

Date 2014-08-20

Author Edouard Pauwels

Maintainer Edouard Pauwels <pauwelsed@gmail.com>

Description Implementation of a Bayesian active learning strategy to carry out sequential experimental design in the context of biochemical network kinetic parameter estimation. This package gathers functions and pre-computed data sets to reproduce results presented in Pauwels E. et. al published in BMC Systems Biology, 2014. Scripts are given to compute all results from scratch or to draw pictures based on pre-computed data sets.

License GPL-3

Depends deSolve, ggplot2, R (>= 2.10)

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-23 08:12:04

R topics documented:

pauwels2014-package 2
active_design 3
add_infinitesimal 4
add_noise 5
armijo 6
BFGS_special 6
compute_gradient 7
compute_gradient_coordinate 7
compute_mean_risks 8
dmvnorm 9
dream6_design 9
estimate_risk_dream6 10
estimate_risk_out_all 11

eval_kn_log_like 13
eval_log_like_knobj 14
experiment_list1 15
exps 17
generate_our_knowledge 18
generate_sample 21
knobjs 22
log_likelihood 23
log_normalize 24
log_prior 25
observables 26
proj_grad 27
random_design 28
read_knobjs 29
reverse_params 30
risk_theta_fun 31
risk_theta_vect 32
sample_function 33
sample_function_multi_mod_weight 33
sample_function_single_mod 35
simulate_experiment 36
simulate_experiment_no_transform 37
transform_params 38

Index **39**

`pauwels2014-package` *Reproduce numerical experiments*

Description

This package gathers functions and pre-computed results to reproduce the results presented in a research paper (see vignette). Scripts are given to compute all results from scratch or to draw pictures based on pre-computed results. Adapting this to other networks would require quite an amount of work. The vignette gives practical examples and points out the functions to modify in order to adapt to other settings.

Details

Package:	pauwels-2014
Type:	Package
Version:	1.0
Date:	2014-08-20
License:	GPL-3

Makes extensive use of ode solver from package `deSolve`. The vignette provides detailed examples.

Author(s)

Edouard Pauwels

References

Edouard Pauwels, Christian Lajaunie, and Jean-Philippe Vert. A bayesian active learning strategy for sequential experimental design in systems biology. BMC Systems Biology, 2014. To appear.

Karline Soetaert, Thomas Petzoldt, R. Woodrow Setzer (2010). Solving Differential Equations in R: Package deSolve Journal of Statistical Software, 33(9), 1–25. http://www.jstatsoft.org/v33/i09/

`active_design` *Simulates the active design process.*

Description

Simulates the active design process.

Usage

```
active_design(knobj, sample_function,
seed, credits = 5000, file_to_save = NULL, verbose = T)
```

Arguments

`knobj`	A knowledge list. See `knobjs`.
`sample_function`	A sample function that takes a knowledge list as argument and outputs a sample from the associated posterior in a design matrix.
`seed`	A random number generator seed.
`credits`	Total credit to be spent.
`file_to_save`	A file where the updated knowledge list should be saved at each step.
`verbose`	Should the process print information about on going computation.

Details

This implements the procedure described in the paper. If a file name is provided, the resulting object will be saved at the corresponding location at each step of the process. The function requires the global variables `experiment_list1` and `observables` to be available.

Value

An updated knowledge list.

Author(s)

Edouard Pauwels

See Also

knobjs, sample_function_multi_mod_weight, sample_function_single_mod, experiment_list1, observables, sample_function.

Examples

```
data(exps)
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
    knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
    observables[["mrnaLow"]]$obs
   ]
 )
)
knobj$experiments <- paste("nothing", "mrnaLow")


## Decrease parameter values for the example
knobj$global_parameters$max_it <- 2
knobj$global_parameters$n_simu_weights <- 2
knobj$global_parameters$sample_burn_in <- 5
knobj$global_parameters$sample_to_keep1 <- 2
knobj$global_parameters$n_multi_mod <- 2
knobj$global_parameters$final_sample <- 2
knobj$global_parameters$final_sample_design <- 2

## Run the active design (this takes quite some time)
#knobj <- active_design(knobj,
# sample_function_single_mod, seed = 1, credits = 400)
#
```

---

add_infinitesimal *Finite difference function*

Description

Adds a small deterministic amount to a vector.

add_noise *Noise generative process for the simulations*

Description

Specifies a noise generative process for the simulations. This describes how the true dynamics of the system is perturbed by noise.

Usage

```
add_noise(data_theta_Ts)
```

Arguments

`data_theta_Ts`	A time series data matrix, the first column representing time and the remaining columns representing time course of various quantities of interest.

Details

The default generative process is to add independant gaussian heteoscedastic noise to all columns, except the first one representing time. The noise model is gaussian with variance of the form `(0.01 + 0.04 * m^2)` where `m` is the mean.

Value

A time series data matrix of the same size as the input.

Author(s)

Edouard Pauwels

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Generate a time cours matrix
tempCourse <- simulate_experiment(
 knobj$global_parameters$true_params_T,
 knobj,
 experiment_list1$nothing
)[
 knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
 observables[["mrnaLow"]]$obs
]
```

```
## Add noise to the time course matrix
add_noise(tempCourse)
```

armijo *Performs armijo line searcc*

Description

Implements Armijo condition

BFGS_special *An implementation of BFGS method for posterior maximization.*

Description

Gradients are computed using finite differences.

Usage

```
BFGS_special(init, knobj, fun_like, verbose = FALSE)
```

Arguments

`init`	An initial value of the parameter to be optimized.
`knobj`	A knowledge list. See `knobjs`.
`fun_like`	A function to compute posterior value. See `eval_log_like_knobj`
`verbose`	Print progresses of the local search?

Details

The step size are chosen using Armijo's rule. Special checks are performed to avoid numerical instabilities in the differential equation solver.

Value

A list with the following entries:

`theta`	The local optimum found by the method.
`fail`	A boolean representing wither the local search failed or not due to numerical problems.

Author(s)

Edouard Pauwels

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
   knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
   observables[["mrnaLow"]]$obs
  ]
 )
)
knobj$experiments <- paste("nothing", "mrnaLow")

theta <- rep( 50, length(knobj$global_parameters$param_names) )
names(theta) <- knobj$global_parameters$param_names

## Only perform 5 iterations
knobj$global_parameters$max_it <- 5

temp <- BFGS_special(theta, knobj, eval_log_like_knobj)
temp$theta
```

`compute_gradient` *Finite difference function*

Description

Computes a gradient based on finite difference approximation

`compute_gradient_coordinate` *Finite difference function*

Description

Comute the derivative of a single coordinate based on finite difference approximation.

compute_mean_risks *Compute an average risk as a function of credit spent*

Description

Used to summarize information contained in lists given by `read_knobjs`.

Usage

```
compute_mean_risks(mean_risks, legend)
```

Arguments

`mean_risks`	A list of tables as given by `read_knobjs`.
`legend`	A label to be associated to the output data frame.

Details

Summarizes results in a data frame for further ploting.

Value

A data frame with four columns: `cost`, the budget spent, numerical, `risk`, the associated risk, numerical, `type`, given by the `legend` argument and `chain` which is an integer associated to each table in the list argument.

Author(s)

Edouard Pauwels

See Also

`read_knobjs`

Examples

```
data(knobjs)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)

data(exps)
temp <- read_knobjs(paste("knobjActMult", 1:10, sep=""))
mean_risks <- compute_mean_risks(temp, "A title")
mean_risks
```

`dmvnorm`	*Gaussian multivariate density*

Description

Gaussian multivariate density.

`dream6_design`	*Simulates the active design process using the comparison criterion (see article for details).*

Description

Simulates the active design process.

Usage

```
dream6_design(knobj, sample_function,
seed, credits = 5000, file_to_save = NULL, verbose = T)
```

Arguments

`knobj`	A knowledge list. See `knobjs`.
`sample_function`	A sample function that takes a knowledge list as argument and outputs a sample from the associated posterior in a design matrix.
`seed`	A random number generator seed.
`credits`	Total credit to be spent.
`file_to_save`	A file where the updated knowledge list should be saved at each step.
`verbose`	Should the process print information about on going computation.

Details

This implements the active design procedure using the criterion used by the wining strategy for DREAM6 challenge. If a file name is provided, the resulting object will be saved at the corresponding location at each step of the process. The function requires the global variables `experiment_list1` and `observables` to be available.

Value

An updated knowledge list.

Author(s)

Edouard Pauwels

See Also

knobjs, sample_function_multi_mod_weight, sample_function_single_mod, experiment_list1, observables, sample_function.

Examples

```
data(exps)
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
   knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
   observables[["mrnaLow"]]$obs
  ]
 )
)
knobj$experiments <- paste("nothing", "mrnaLow")


## Decrease parameter values for the example
knobj$global_parameters$max_it <- 2
knobj$global_parameters$n_simu_weights <- 2
knobj$global_parameters$sample_burn_in <- 5
knobj$global_parameters$sample_to_keep1 <- 2
knobj$global_parameters$n_multi_mod <- 2
knobj$global_parameters$final_sample <- 2
knobj$global_parameters$final_sample_design <- 2

## Run the active design (this takes quite some time)
#knobj <- dream6_design(knobj, sample_function_single_mod, seed = 1, credits = 400)
```

---

estimate_risk_dream6 *Expected risk estimation (comparison with litterature).*

Description

Implements the strategy of Dream6 challenge winning team.

Usage

```
estimate_risk_dream6(thetas, knobj, experiment_fun)
```

Arguments

thetas	A sample from the posterior associated to the knoweldge list knobjs.
knobj	A knowledge list. See knobjs.
experiment_fun	A function that represents the molecular perturbation to be performed. See experiment_list1.

Details

The global variable observables should be defined.

Value

A dataframe with the following columns

Measurement	Factor representing possible measurements. See observables.
Risk	The risk associated to this measurement.
Cost	The cost associated to this measurement.

Author(s)

Edouard Pauwels

Examples

```
data(experiment_list1)
data(observables)
data(knobjs)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)

knobjActMult1$global_parameters$n_simu_weights <- 3

estimate_risk_dream6(knobjActMult1$datas[[1]]$thetas[1:10,],
knobjActMult1, experiment_list1$nothing)
```

---

estimate_risk_out_all *Expected risk estimation.*

Description

Estimates the expected risk associated to a given experiment for all possible observations to be performed.

Usage

```
estimate_risk_out_all(thetas, knobj, experiment_fun)
```

Arguments

`thetas`	A sample from the posterior associated to the knoweldge list `knobjs`.
`knobj`	A knowledge list. See `knobjs`.
`experiment_fun`	A function that represents the molecular perturbation to be performed. See `experiment_list1`.

Details

This implements the risk estimation procedure described in the paper. We use importance weighting to perform computation based on a single posterior sample. The global variable `observables` should be defined.

Value

A dataframe with the following columns

`Measurement`	Factor representing possible measurements. See `observables`.
`Risk`	The risk associated to this measurement.
`Cost`	The cost associated to this measurement.

Author(s)

Edouard Pauwels

Examples

```
data(experiment_list1)
data(observables)
data(knobjs)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k],
knobjs[[k]], envir = .GlobalEnv)
}
)

knobjActMult1$global_parameters$n_simu_weights <- 3

estimate_risk_out_all(knobjActMult1$datas[[1]]$thetas[1:10,],
knobjActMult1, experiment_list1$nothing)
```

`eval_kn_log_like` *Evaluates the likelihood of a parameter value*

Description

This evaluates the likelihood of a parameter value by comparing corresponding kinetics to a given time course matrix.

Usage

```
eval_kn_log_like(theta, initial_conditions, data, knobj,
 fail_incoming = F, simu = NULL, fit = F)
```

Arguments

`theta`	A parameter named numeric vector.
`initial_conditions`	Initial conditions named numeric vector.
`data`	A time course matrix
`knobj`	A knowledge list. See `knobjs`
`fail_incoming`	A boolean indicating wether an error message is given by the ode solver.
`simu`	The simulated time course matrix corresponding to the parameter `theta`. If it is not provided, it will be computed by a call to `simulate_experiment_no_transform`.
`fit`	A parameter to be passed to the likelihood function. It indicates wether further prior information about the smoothness of the dynamical time course should be considered. This is used to guide local search posterior maximization methods.

Details

The comparison is made based on the user defined `log_likelihood` function.

Value

A numerical value if `fail_incoming == FALSE`. A list containing a `res` numerical slot and a `fail` boolean slot representing weither the ode solver failed or not.

Author(s)

Edouard Pauwels

See Also

`log_likelihood`, `knobjs`, `simulate_experiment_no_transform`

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
    knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
    observables[["mrnaLow"]]$obs
   ]
 )
)

eval_kn_log_like(
 knobj$global_parameters$true_params,
 knobj$global_parameters$initial_conditions,
 knobj$datas[[1]]$data, knobj )
```

`eval_log_like_knobj` *Posterior function.*

Description

Computes the posterior value associated to a given parameter value for a given knowledge list.

Usage

```
eval_log_like_knobj(theta, knobj, fail_incoming = F, fit = F)
```

Arguments

`theta`	A parameter named numeric vector.
`knobj`	A knowledge list. See `knobjs`
`fail_incoming`	A boolean indicating wether an error message is given by the ode solver.
`fit`	A parameter to be passed to the likelihood function. It indicates wether further prior information about the smoothness of the dynamical time course should be considered. This is used to guide local search posterior maximization methods.

Details

The function computes the log prior first and then the likelihood associated to all the time course data found in the `knobj$datas` slot. The likelihood terms are summed. The prior term and the likelihood terms are weighted, weights being the inverse the number of observations they represent. This is necessary in order to give comparable contributions to low resolution and high resolution experiments.

Value

A numerical value if `fail_incoming == FALSE`. A list containing a `res` numerical slot and a `fail` boolean slot representing wether the ode solver failed or not.

Author(s)

Edouard Pauwels

See Also

`log_prior`, `knobjs`, `eval_kn_log_like`

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
    knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
    observables[["mrnaLow"]]$obs
   ]
 )
)

eval_log_like_knobj(knobj$global_parameters$true_params_T, knobj)
```

`experiment_list1` *Molecular perturbations.*

Description

A list of functions that represent possible molecular perturbations to be performed on the network.

Usage

```
data(experiment_list1)
```

Details

A list of 10 functions. Each function takes as input a named numeric parameter vector `theta` and an initial condition named numeric vector `initial_conditions`. Each function outputs a list with the following slots:

- `theta`: A modified parameter value
- `initial_conditions`: A modified initial condition value
- `cost`: The cost of the associated molecular perturbation

The 10 functions perform the following operations on their input (parameters numbering might not be always coherent with names):

- `delete_gene6`
  - `theta`: set `pro6_strength` and `rbs6_strength`
  - `initial_condtions`: set `p6` initial concentration to 0
  - `cost`: 800
- delete_gene7
  - `theta`: set `pro7_strength` and `rbs8_strength`
  - `initial_condtions`: set `p7` initial concentration to 0
  - `cost`: 800
- `delete_gene8`
  - `theta`: set `pro7_strength` and `rbs8_strength`
  - `initial_condtions`: set `p7` initial concentration to 0
  - `cost`: 800
- `knockdown_gene6`
  - `theta`: multiply `mrna6_degradation_rate` by 10
  - `initial_condtions`: Nn operation
  - `cost`: 350
- `knockdown_gene7`
  - `theta`: multiply `mrna7_degradation_rate` by 10
  - `initial_condtions`: no operation
  - `cost`: 350
- `knockdown_gene8`
  - `theta`: multiply `mrna8_degradation_rate` by 10
  - `initial_condtions`: no operation
  - `cost`: 350
- `decrease_rbs_gene6`
  - `theta`: divide `rbs6_strength` by 10
  - `initial_condtions`: no operation

    - cost: 350

- decrease_rbs_gene7
    - theta: divide rbs6_strength by 10
    - initial_condtions: no operation
    - cost: 350
- decrease_rbs_gene8
    - theta: divide rbs7_strength by 10
    - initial_condtions: no operation
    - cost: 350
- nothing
    - theta: no operation
    - initial_condtions: no operation
    - cost: 0

Examples

```
data(experiment_list1)
data(knobjs)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)

theta <- knobjActMult1$datas[[1]]$thetas[1,]
thetaT <- knobjActMult1$transform_params(theta)

temp <- experiment_list1$delete_gene7(thetaT, knobjActMult1$global_parameters$initial_conditions)

rbind(temp$theta, thetaT)
rbind(temp$initial_conditions, knobjActMult1$global_parameters$initial_conditions)
```

---

exps *List of possible experiments*

Description

An experiment associates a molecular perturbation and a quantity to observe.

Usage

```
data(exps)
```

Format

A data frame that combines all possible experiments of experiment_list1 to all possible observations of observables.

Details

The data frame contains 49 entries of the following variables

- Measurement: a factor with levels "mrnaHigh", "mrnaLow", "p6", "p7", "p8". See observables
- Cost: a numeric vector. The cost of the corresponding experiment and observation to be performed.
- exp: a factor with levels "decrease_rbs_gene6", "decrease_rbs_gene7", "decrease_rbs_gene8", "delete_gene6", "delete_gene7", "delete_gene8", "knockdown_gene6", "knockdown_gene7", "knockdown_gene8", "nothing". See experiment_list1 for a description of the corresponding perturbations

This should be 50 experiments by default (10 experiments and 5 observables). However, we always initialise the design process with experiment nothing and observable mrnaLow, this leaves 49 possible experiments left. This object is used by the random_design function.

Examples

```
data(exps)
exps[1,]
```

---

generate_our_knowledge

Initialize a knowledge list.

Description

This function defines a new knowledge list. Its main purpose is to give a value to all the global parameters that have to be set up before running the simulations.

Usage

```
generate_our_knowledge(transform_params, global_parameters, datas)
```

Arguments

transform_params	A function that transform parameter values. See transform_params.
global_parameters	A list of global parameters. See the details section for the default value.
datas	Potential data to be put in the datas slot.

Details

There are four main slots in the knowledge list the first one is `transform_params` which is by default a log transformation function that allows to work in log space. See `transform_params`. The second slot, `global_parameters` contains various global parameters necessary for the simulation to proceed. The default value of those parameters is given in the next section. See also the vignette for details. The third slot is named `datas`. It is empty by default, but is intended to receive datasets, experiment function, posterior sample and expected risk estimates for each step of the simulation. The vignette provides more details about this.

Value

A list that agregates the input. The default value of the global parameter is as follows:

`atol = 1e-6`	deSolve parameter (see `deSolve`)
`rtol = 1e-6`	deSolve parameter (see `deSolve`)
`tspan = seq(0,100,0.5)`	deSolve parameter, the time points at which all simulated kinetic quantities should be evaluated
`max_step = 50`	deSolve parameter (see `deSolve`)
`max_it = 200`	optimization parameter, the maximum number of iteration in the for `BFGS_special` function
`tol = 1e-3`	optimization parameter, a stoping criterion based on gradient norm tolerance for `BFGS_special` function
`beta = 2`	optimization parameter, the division to perform on the step size when armijo's rule fails in `BFGS_special` function
`c = 0.0001`	optimization parameter, armijo's rule second parameter for `BFGS_special` function
`n_multi_mod_weight = 20`	sampling parameter, number of times the local search/sample operations are repeated in `sample_function_multi_mod_weight` function
`max_log_like = - 700`	sampling parameter, a lower bound under which parameter values are systematically after the BFGS search in `sample_function_multi_mod_weight` function and in `sample_function_single_mod` function
`centrality_ratio = 0.4`	sampling parameter, allows to only keep parameter values leading to reasonable datafits by filtering those which lead to kinetics trajectories that do not pass in the "middle" of the data in `sample_function_multi_mod_weight` function and in `sample_function_single_mod` function
`sample_burn_in = 5000`	sampling parameter, size of the burn in sample to be ignored in the Metropolis Hasting algorithm algorithm in `generate_sample`
`sample_to_keep1 = 10000`	sampling parameter, sample size to be further sampled in MH algorithm in `generate_sample`

`sample_step = 1`
: sampling parameter, the mean MH step length in `generate_sample` function

`final_sample = 10000`
: sampling parameter, the final sample size provided by sampling functions `sample_function_multi_mod_` and `sample_function_single_mod`. This is the size of the sample that is to be stored in the knowledge list.

`final_sample_design = 100`
: size of a subsample to be used for risk estimation in `active_design` strategy.

`n_simu_weights = 100`
: number of noise simulations required to estimate the weights in risk estimation in `active_design` strategy.

`initial_conditions = c(g6 = 1, p6 = 1,p7 = 1,p8 = 1, v6_mrna = 0,v7_mrna = 0,v8_mrna = 0)`
: simulation parameter, the initial condition default value

`n_params = 9`
: simulation parameter, number of free parameters

`param_names = c("p_degradation_rate", "r6_Kd", "r11_Kd", "pro6_strength", "pro7_strength", "pro9_st`
: simulation parameters, the names the free parameters

`params = c(p_degradation_rate = 1, r6_Kd = 1, r11_Kd=1, pro6_strength = 1, pro7_strength = 1, pro9_`
: simulation parameters, an instance of free parameter named numeric vector

`true_params = c(mrna6_degradation_rate =1, mrna7_degradation_rate =1, mrna8_degradation_rate =1, p_`
: simulation parameters, the true parameter to be estimated and used for simulations in the physical space

`true_params_T = c(p_degradation_rate = 50, r6_Kd = 56.9162224661803, r11_Kd = 55.0171665943997, pro`
: simulation parameters, the true parameter to be estimated and used for simulations in log space, such that `transform_params(true_params_T) = true_params`

,

`dllname = "pauwels2014"`
: simulation parameters, the name of the shared object which contains the function to be bassed to `deSolve` solver. See the vignette and `deSolve` package for more details. The default is `"pauwels2014"` as the source for the example of the article is provided in this package.

Author(s)

Edouard Pauwels

Examples

```
knobj <- generate_our_knowledge(transform_params)
knobj
```

`generate_sample` *An implementation of the Metropolis Hasting algorithm*

Description

This is an implementation of MH algorithm to sample from the posterior distribution. The proposal is a mixture between a gaussian proposal and a single coordinate proposal. The step size is diminished when the rejection rate is too high.

Usage

```
generate_sample(theta, knobj, N = 500, step = 1, verbose = F)
```

Arguments

`theta`	An initialization parameter named numeric vector
`knobj`	A knowledge list. See `knobjs`.
`N`	The total sample size.
`step`	The proposal distribution expected step length.
`verbose`	Should the sampling process print information about itself?

Details

The posterior is evaluated using `eval_log_like_knobj` function.

Value

A posterior sample matrix, each row representing a parameter named numeric vector.

Author(s)

Edouard Pauwels

See Also

`eval_log_like_knobj`, `knobjs`

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
  manip = experiment_list1$nothing,
  data = add_noise(
   simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
     knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
     observables[["mrnaLow"]]$obs
    ]
  )
)

generate_sample(knobj$global_parameters$params * 50, knobj, N = 10)
```

knobjs *Knowledge lists*

Description

A list of knwoledge list

Usage

```
data(knobjs)
```

Format

A list of lists that were generated using `generate_our_knowledge` function and updated using different strategy functions and sampling functions. The name of the object defines which strategy and which sampling function was used:

- `Rand`: `random_design` strategy
- `Act`: `active_design` strategy
- `Dream6`: `dream6_design` strategy
- `Sing`: `sample_function_single_mod` sampling function
- `Mult`: `sample_function_multi_mod_weight` sampling function

The integer at the end of the name is the random number generator seed used to perform the corresponding simulation.

Details

The update performed by the strategies consists in filling the `experiments` slot of `knobj` with the sequence experiments performed. See `exps`. The corresponding data is available in the `datas` slot. This slot is a list. Each element of this list is a list with the following entries

- `manip`: the molecular perturbation performed, an element of `experiment_list1`
- `data`: the associated noisy observations, based on the chosen observable, an element of `observables`

- `thetas`: a posterior sample generated by the corresponding `sample_function`.
- `risks`: the risk associated to all potential new experiments computed by `estimate_risk_out_all`

The knowledge lists updated using `random_design` strategy do not provide any risk information.

Examples

```
data(knobjs)
names(knobjs)

sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)
ls()
```

`log_likelihood` *User defined likelihood function.*

Description

Noise is assumed to be independent for each entry. The default likelihood assumes the same heteroscedastic noise as the model used in `add_noise`.

Usage

```
log_likelihood(simu, simu_subset, data, fit = F)
```

Arguments

`simu`	Simulated time courses.
`simu_subset`	Subset of the simulated time course which relates to observed data.
`data`	Observed data.
`fit`	Should smoothness prior information about simulation added to the prior or not?

Details

The noise model is gaussian with variance of the form `(0.01 + 0.04 * m^2)` where `m` is the mean.

Value

A numerical value.

Author(s)

Edouard Pauwels

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

simu <- simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)

simu_subset <- simu[
 knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
 observables[["mrnaLow"]]$obs
]

data <- add_noise(simu_subset)

log_likelihood(simu, simu_subset, data)
```

log_normalize *Normalize in log space*

Description

Binding to a C implementation of addition and normalization in log space.

Usage

```
log_normalize(vals, maxi = max(vals), ID = which.max(vals)[1])
```

Arguments

vals	Log-values to be normalize
maxi	The maximum value of vals
ID	One entry of vals which has the highest value.

Value

A list with three slots. First one is a numerical vector with values normalized so that they sum to 1. The second is the same normalized vector in log space. The last one is the log of the normalization constant. See example.

Author(s)

Edouard Pauwels

Examples

```
log_x <- rnorm(10)

temp <- log_normalize(log_x)

exp(temp[[2]]) - temp[[1]]

sum(exp(temp[[2]]))

log_x - temp[[3]] - temp[[2]]
```

`log_prior` *User defined log prior*

Description

User defined log prior based on the parameter value only.

Usage

```
log_prior(theta)
```

Arguments

`theta`	A parameter named numeric vector.

Details

The default is a gaussian prior with large covariance matrix. This is actually a prior on the parameters in log space. The prior can can thus be thought of as a log normal prior.

Value

A numerical value

Author(s)

Edouard Pauwels

Examples

```
theta <- 50 + sample(c(0,1), size = 34, replace = TRUE)
log_prior(theta)
```

`observables` *Observable quantities of the model*

Description

A list of quantities that can be observed. Each entry describes the quantity to observe and its time resolution as well as the cost of the corresponding observation.

Usage

```
data(observables)
```

Format

A list with the follwing entries: `"p6"`, `"p7"`, `"p8"`, `"mrnaHigh"`, `"mrnaLow"`. Each of those entries is a list with the following slots:

- `name`: (chr) name of the observable
- `obs`: (chr) the subset of kinetic variables to observe
- `reso`: (num) time points to observe
- `cost`: (nul) the cost of observing this quantity

Details

The complete description of the list is as follows

- `"p6"`:
  - `name`: `"p6"`
  - `obs`: `c("time", "p6")`
  - `reso`: from `0` to `100` by steps of `0.5`
  - `cost`: 400
- `"p7"`:
  - `name`: `"p7"`
  - `obs`: `c("time", "p7")`
  - `reso`: from `0` to `100` by steps of `0.5`
  - `cost`: 400
- `"p8"`:
  - `name`: `"p8"`
  - `obs`: `c("time", "p8")`
  - `reso`: from `0` to `100` by steps of `0.5`
  - `cost`: 400
- `"mrnaHigh"`:
  - `name`: `"mrnaHigh"`

  - obs: c("time", "v6_mrna", "v7_mrna", "v8_mrna")
  - reso: from 0 to 100 by steps of 2
  - cost: 1000
- "mrnaLow":
  - name: "mrnaLow"
  - obs: c("time", "v6_mrna", "v7_mrna", "v8_mrna")
  - reso: from 0 to 100 by steps of 4
  - cost: 500

Examples

```
data(observables)
data(knobjs)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)
data(exps)

theta <- knobjActMult1$datas[[1]]$thetas[1,]
thetaT <- knobjActMult1$transform_params(theta)

temp <- simulate_experiment_no_transform(thetaT,
knobjActMult1$global_parameters$initial_conditions, knobjActMult1)

observable <- observables$mrnaLow
temp[temp[,1] %in% observable$reso, colnames(temp) %in% observable$obs]
```

---

proj_grad *Least square on the positive orthant*

Description

A projected gradient scheme for least square estimation on the positive orthant.

random_design — *Simulates a randim design process.*

Description

Simulates a randim design process.

Usage

```
random_design(knobj, sample_function, exps, seed, credits = 5000,
 file_to_save = NULL, verbose = T)
```

Arguments

`knobj`	A knowledge list. See `knobjs`.
`sample_function`	A sample function that takes a knowledge list as argument and outputs a sample from the associated posterior in a design matrix.
`exps`	A dataframe representing possible experiments to be performed with their cost. See `exps`.
`seed`	A random number generator seed.
`credits`	Total credit to be spent.
`file_to_save`	A file where the updated knowledge list should be saved at each step.
`verbose`	Should the process print information about on going computation.

Details

This implements a strategy consisting in choosing experiments randomly. If a file name is provided, the resulting object will be saved at the corresponding location at each step of the process.

Value

An updated knowledge list.

Author(s)

Edouard Pauwels

See Also

`knobjs`, `sample_function_multi_mod_weight`, `sample_function_single_mod`, `experiment_list1`, `observables`, `sample_function`, `exps`

Examples

```
data(exps)
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
    knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
    observables[["mrnaLow"]]$obs
   ]
 )
)
knobj$experiments <- paste("nothing", "mrnaLow")


## Decrease parameter values for the example
knobj$global_parameters$max_it <- 2
knobj$global_parameters$n_simu_weights <- 3
knobj$global_parameters$sample_burn_in <- 5
knobj$global_parameters$sample_to_keep1 <- 10
knobj$global_parameters$sample_to_keep2 <- 10
knobj$global_parameters$n_multi_mod_weight <- 2
knobj$global_parameters$final_sample <- 5
knobj$global_parameters$final_sample_design <- 5

## Run the random design (this takes quite some time)
#knobj <- random_design(knobj, sample_function_single_mod, exps, seed = 1, credits = 400)
```

---

read_knobjs *Summarizes pre-computed results.*

Description

Allows to reproduce the figures of the paper (see also vignette).

Usage

```
read_knobjs(objs)
```

Arguments

objs	A table of strings representing names of objects given in knobjs.

Details

The function computes the risk of the estimated posterior mean as a function of credit spent for each object.

Value

A list of tables, one for each string. For each of them, the first line is the risk of the posterior sample mean and the second line is the credit spent.

Author(s)

Edouard Pauwels

See Also

knobjs

Examples

```
data(knobjs)
data(exps)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)

read_knobjs(paste("knobjActMult", 1:10, sep=""))
```

`reverse_params` *Transform log space parameters back to the original space*

Description

Least square minimization to map back log transformed parameters to their original values

`risk_theta_fun` *Risk function*

Description

Compute the risk between two different parameter values.

Usage

```
risk_theta_fun(theta1, theta2, n_params)
```

Arguments

`theta1`	A first parameter array.
`theta2`	A second parameter array.
`n_params`	Number of parameters to be estimated in the two arrays.

Details

The default is averaged squared difference between log values of parameters

Value

A numerical value.

Author(s)

Edouard Pauwels

Examples

```
theta1 <- sample(1:10000, size = 100)
theta2 <- sample(1:10000, size = 100)
n <- 50

risk_theta_fun(theta1, theta2, n)
```

risk_theta_vect *Expected risk based on a posterior sample*

Description

The function computes the expected risk based on the empirical distribution defined by the sample thetas_trans.

Usage

```
risk_theta_vect(thetas_trans, n_params)
```

Arguments

thetas_trans	A matrix which row represent parameter values in physical space (after applying transform_params).
n_params	The number of parameters to be estimated

Value

A numerical value

Author(s)

Edouard Pauwels

See Also

risk_theta_fun

Examples

```
data(knobjs)
sapply(
1:length(knobjs),
function(k){
assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
}
)

thetas <- knobjActMult1$datas[[1]]$thetas[1:10,]

thetas_trans <- t(apply(thetas, 1, transform_params))
risk_theta_vect(thetas_trans, 9)
```

`sample_function` *Generates posterior samples*

Description

This generates a posterior sample based on the data and global parameters defined in its input.

Usage

```
sample_function(knobj)
```

Arguments

`knobj`	A knowledge list. See `knobjs`.

Details

This function is the same as, `sample_function_multi_mod_weight`. This is the sampling function used by design strategies. It can be changed to modify the sampling strategy (e.g. `sample_function_single_mod`).

Value

A matrix which row represent a parameter named numeric vector in log space. Its distribution is supposed to be closed to the posterior distribution defined by knobj.

Author(s)

Edouard Pauwels

See Also

`sample_function_multi_mod_weight`, `sample_function_single_mod`

`sample_function_multi_mod_weight` *Sample function visiting multiple modes of the posterior*

Description

Generate a posterior sample using multiple local search maximization and sampling based on different initializations.

Usage

```
sample_function_multi_mod_weight(knobj)
```

Arguments

knobj	A knowledge list. See knobjs.

Details

The parameters governing the local search and sampling behaviour are defined in the global_parameters slot of the knobjs argument. The function consists in using the BFGS_special function to find an initialization for the Metropolis Hasting algorithm implented by generate_sample. This is done multiple times. This procedure is also applied to previous sample points. All those sample are aggregated and the resulting sample is chosen randomly based on the associated posterior values.

Value

A matrix which rows represent a named numeric vector of parameters

Author(s)

Edouard Pauwels

See Also

sample_function, BFGS_special, generate_sample

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
    knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
    observables[["mrnaLow"]]$obs
   ]
 )
)

## Decrease parameter values for the example
knobj$global_parameters$max_it <- 2
knobj$global_parameters$n_multi_mod <- 2
knobj$global_parameters$sample_burn_in <- 5
knobj$global_parameters$sample_to_keep1 <- 100
knobj$global_parameters$final_sample <- 100
knobj$global_parameters$final_sample_est <- 100
```

```
#thetas <- sample_function_multi_mod_weight(knobj)
#thetas
```

---

sample_function_single_mod

Sample function visiting a single mode of the posterior.

Description

Generate a posterior sample using a single local search maximization and sampling.

Usage

```
sample_function_single_mod(knobj)
```

Arguments

knobj	A knowledge list. See `knobjs`.

Details

The parameters governing the local search and sampling behaviour are defined in the `global_parameters` slot of the `knobjs` argument. The function consists in using the `BFGS_special` function to find an initialization for the Metropolis Hasting algorithm implented by `generate_sample`. This is done a single time.

Value

A matrix which rows represent a named numeric vector of parameters

Author(s)

Edouard Pauwels

See Also

`sample_function`, `BFGS_special`, `generate_sample`

Examples

```
data(experiment_list1)
data(observables)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

## Initialize with some data
knobj$datas[[1]] <- list(
 manip = experiment_list1$nothing,
 data = add_noise(
  simulate_experiment(knobj$global_parameters$true_params_T, knobj, experiment_list1$nothing)[
   knobj$global_parameters$tspan %in% observables[["mrnaLow"]]$reso,
   observables[["mrnaLow"]]$obs
  ]
 )
)

## Decrease parameter values for the example
knobj$global_parameters$max_it <- 2
knobj$global_parameters$sample_burn_in <- 5
knobj$global_parameters$sample_to_keep1 <- 10
knobj$global_parameters$final_sample <- 5
knobj$global_parameters$final_sample_est <- 5

thetas <- sample_function_single_mod(knobj)
thetas
```

simulate_experiment *Simulates the dynamics of a molecular perturbation*

Description

This function simulates the kinetics of the system, in accordance with a chosen molecular perturbation.

Usage

```
simulate_experiment(theta, knobj, experiment_fun)
```

Arguments

theta	A named numeric parameter vector
knobj	A knowledge list. See knobjs.
experiment_fun	An element of experiment_list1 that represents a molecular perturbation

Details

Transforms parameter values and initial conditions in order to pass it to deSolve ode solver.

Author(s)

Edouard Pauwels

See Also

transform_params, experiment_list1, simulate_experiment_no_transform

Examples

```
data(experiment_list1)

## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

temp <- simulate_experiment(knobj$global_parameters$true_params,
 knobj, experiment_list1$nothing)

head(temp)
```

---

simulate_experiment_no_transform

Link to the ode solver.

Description

Simulate the kinetics of the system using the ode solver of package `deSolve`.

Usage

```
simulate_experiment_no_transform(theta, initial_conditions, knobj)
```

Arguments

`theta`	A parameter vector in physical space to be passed to the solver
`initial_conditions`	A vector of initial conditions to be passed to the solver
`knobj`	A knowledge list. See `knobjs`

Details

The solver parameters are given by the `global_parameters` slot of `knobj`. The solver will call an executable which is defined in the `src` directory of the package under the name `model0_simplified_mrna_rates`. The number and order of parameters in `theta` and initial conditions in `initial_conditions` should correspond to the definition of the dynamical system as described in this file. See the vignette and manual of package `deSolve`.

Value

A `deSolve` table with attributes.

Author(s)

Edouard Pauwels

Examples

```
## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

temp <- simulate_experiment_no_transform(knobj$global_parameters$true_params,
 knobj$global_parameters$initial_conditions, knobj)

head(temp)
```

`transform_params` *User defined parameter transformation function.*

Description

Allows to go from log space to physical space for kinetic parameters. Known parameter values are also added here.

Usage

```
transform_params(pars)
```

Arguments

`pars`	A named numeric parameter vector

Details

The number and order of resulting parameters should be the same as the one expected by the `simulate_experiment_no_transform` function.

Value

A named numeric parameter vector

Author(s)

Edouard Pauwels

See Also

`simulate_experiment`, `simulate_experiment_no_transform`, `estimate_risk_out_all`

Examples

```
## Generate the knowledge object with correct parameter value
knobj <- generate_our_knowledge(transform_params)

knobj$global_parameters$true_params_T
transform_params(knobj$global_parameters$true_params_T)
```

Index

∗Topic **Design strategies**
- active_design, 3
- dream6_design, 9
- random_design, 28

∗Topic **Model specific functions**
- add_noise, 5
- experiment_list1, 15
- generate_our_knowledge, 18
- log_likelihood, 23
- log_prior, 25
- risk_theta_fun, 31
- risk_theta_vect, 32
- transform_params, 38

∗Topic **Model specific objects**
- exps, 17
- knobjs, 22

∗Topic **Optimization functions**
- add_infinitesimal, 4
- armijo, 6
- BFGS_special, 6
- compute_gradient, 7
- compute_gradient_coordinate, 7
- compute_mean_risks, 8
- dmvnorm, 9
- proj_grad, 27
- reverse_params, 30

∗Topic **Risk estimation functions**
- risk_theta_vect, 32

∗Topic **Risk estimation function**
- estimate_risk_dream6, 10
- estimate_risk_out_all, 11

∗Topic **Sampling functions**
- generate_sample, 21
- sample_function, 33
- sample_function_multi_mod_weight, 33
- sample_function_single_mod, 35

∗Topic **Simulation functions**
- eval_kn_log_like, 13
- eval_log_like_knobj, 14
- simulate_experiment, 36
- simulate_experiment_no_transform, 37

∗Topic **datasets**
- experiment_list1, 15
- exps, 17
- knobjs, 22
- observables, 26

∗Topic **package**
- pauwels2014-package, 2
- read_knobjs, 29

active_design, 3, *20*, *22*
add_infinitesimal, 4
add_noise, 5, *23*
armijo, 6

BFGS_special, 6, *19*, *34, 35*

compute_gradient, 7
compute_gradient_coordinate, 7
compute_mean_risks, 8

dmvnorm, 9
dream6_design, 9, *22*

estimate_risk_dream6, 10
estimate_risk_out_all, 11, *23*, *38*
eval_kn_log_like, 13, *15*
eval_log_like_knobj, *6*, 14, *21*
experiment_list1, *3, 4*, *9–12*, 15, *18*, *22*, *28*, *36*
exps, 17, *22*, *28*

generate_our_knowledge, 18, *22*
generate_sample, *19, 20*, 21, *34, 35*

knobjs, *3, 4*, *6*, *9–15*, *21*, 22, *28–30*, *33–37*

log_likelihood, *13*, 23

log_normalize, 24
log_prior, *15*, 25

observables, *3, 4*, *9–12*, *18*, *22*, 26, *28*

pauwels2014 *(*pauwels2014-package*)*, 2
pauwels2014-package, 2
proj_grad, 27

random_design, *18*, *22, 23*, 28
read_knobjs, *8*, 29
reverse_params, 30
risk_theta_fun, 31, *32*
risk_theta_vect, 32

sample_function, *4*, *10*, *23*, *28*, 33, *34, 35*
sample_function_multi_mod_weight, *4*, *10*, *19, 20*, *22*, *28*, *33*, 33
sample_function_single_mod, *4*, *10*, *19, 20*, *22*, *28*, *33*, 35
simulate_experiment, 36, *38*
simulate_experiment_no_transform, *13*, *36*, 37, *38*

transform_params, *18, 19*, *32*, *36*, 38