Bayesian experimental design for systems biology: reproduce results

Abstract

We present details about how the functions provided in `pauwels2014` can be used to reproduce the results presented in [1]. We also describe how the system, model specifications and experimental settings were represented. The goal is to provide one with the capacity to reproduce research results to potentially adapt the method to other settings.

Contents

1 Introduction

The purpose of `pauwels2014` package is to reproduce the experimental results presented in [1]. We focus on reproducing the comparison between different simulations based on a small network made of 3 genes with 6 variables kinetics explicitly modeled and 9 parameters to infer.

The first section shows how to reproduce the graphics based on pre-computed simulations. We provide pre-computed results because of the high computational cost of repeating experiments multiple times. We also describe the commands and scripts that were used to compute these results. We used parallel processes to run multiple simulations involving different parameter sampling and design strategies. This considerably reduces the total computational time of getting these results. We provide scripts and commands that were used to run multiple simulations.

In a second section, we describe more specifically the encoding of the dynamical system, noise model and estimation schemes that we considered. The purpose is to ease the adaptation of the content of this

package to other dynamical systems, noise models, parameter estimation algorithms, sampling schemes and design strategies in order to perform reproducible comparisons.

The package makes extensive use of `ode` solver provided in package `deSolve` [3]. We use the graphics libary `ggplot2` [5] to produce our graphics. Those two packages should be installed to use `pauwels2014` package, for example in the `R` interpreter

```
install.packages("ggplot2")
```

`pauwels2014` is distributed as an `R` package source archive. The installation requires a working `C` compiler correctly interfaced with `R`.

2 Results of the paper

2.1 Plot precomputed simulation results

The following code will load all the precomputed datasets and plot the associated results as presented in figure 1.

```
> library(pauwels2014)
> data(knobjs)
> sapply(
+                         1:length(knobjs),
+                         function(k){
+                                 assign(names(knobjs)[k], knobjs[[k]], envir = .GlobalEnv)
+                         }
+ )
```

```
                  [,1]        [,2]        [,3]        [,4]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,8      List,8
experiments       Character,8 Character,8 Character,8 Character,8
                  [,5]        [,6]        [,7]        [,8]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,8      List,8
experiments       Character,8 Character,8 Character,8 Character,8
                  [,9]        [,10]       [,11]       [,12]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,7      List,8
experiments       Character,8 Character,8 Character,7 Character,8
                  [,13]       [,14]       [,15]       [,16]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,7      List,8      List,8      List,8
experiments       Character,7 Character,8 Character,8 Character,8
                  [,17]       [,18]       [,19]       [,20]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,8      List,8
experiments       Character,8 Character,8 Character,8 Character,8
                  [,21]       [,22]       [,23]       [,24]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,6      List,6      List,5      List,6
experiments       Character,6 Character,6 Character,5 Character,6
                  [,25]       [,26]       [,27]       [,28]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,6      List,6      List,5      List,7
```

```
experiments       Character,6 Character,6 Character,5 Character,7
                  [,29]       [,30]
transform_params  ?           ?
global_parameters List,25     List,25
datas             List,6      List,5
experiments       Character,6 Character,5
```

```
> data(exps)
> mean_risks_act_mult <- read_knobjs( sprintf("knobjActMult%s", 1:10) )
> mean_risks_dream6_mult <- read_knobjs( sprintf("knobjDream6Mult%s", 1:10) )
> mean_risks_rand_mult <- read_knobjs( sprintf("knobjRandMult%s", 1:10) )
> mean_risks_act_mult <- compute_mean_risks(mean_risks_act_mult, "Bayesian active")
> mean_risks_dream6_mult <- compute_mean_risks(mean_risks_dream6_mult, "Dream6")
> mean_risks_rand_mult <- compute_mean_risks(mean_risks_rand_mult, "Random")
> data_to_plot <- rbind(mean_risks_act_mult,mean_risks_dream6_mult, mean_risks_rand_mult)
> ggplot(data = data_to_plot, aes(x=cost, y=risk)) +
+   theme_bw() +
+   facet_grid(.~type) +
+   scale_y_log10() +
+   geom_line(aes(group = chain), alpha = 0.2) +
+   geom_point() +
+   stat_smooth(method = "loess", colour = "black") +
+   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

The `knobj` section of the reference manual describes what these datasets are and gives the list of the datasets of this kind available in the package. `knobj` stands for knoledge list, `Act` stands for active design, `Rand` stands for random design, `Mult` stands for multimodal, `Sing` stands for unimodal and the integer is a random number generator seed used to run the simulation.

2.2 Re-run the simulations

2.2.1 Direct approach

The simulation can be re-run easily using the functions provided in the package. The default parameters allow to reproduce the pre-computed simulations that were used in the last section. However, this can be quite long. For example `knobjActMult1` can be re-computed using the following commands

```
> ## Load datasets
> data(exps)
> data(experiment_list1)
> data(observables)
> ## Initialize a knowledge list
> knobj <- generate_our_knowledge(transform_params)
> 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")
> ## Update the knowledge list
> knobjActMult1 <- active_design(knobj, sample_function_multi_mod_weight, seed = 1, credits = 5000)
```

In total, there were 30 simulations which are represented in figure 1. They can all be re-run using similar scripts, choosing different design strategies, sampling strategies and random seeds.

2.2.2 Using scripts

We describe here the two scripts that were used to launch multiple simulation. First we define a script that we call `perform_simu_one.R` and that allows to control the behaviour of the simulation through the shell.

```
> ## Simulates the active learning experimental design with
> ## A simplified network
> ## Arguments to be passed
> ## active: 0 or 1
> ## multimodal search: 0 or 1
> ## random seed: integer
> ## Results are saved during the process
>
> args <- commandArgs(trailingOnly=TRUE)
> if(length(args) != 3){
+   stop("Three argument should be passed: bool active (0 or 1), bool multimod (0 or 1), int seed")
+ }
> active <- as.integer(as.numeric(args[1]))
> multi_mod <- as.logical(as.numeric(args[2]))
> seed <- as.integer(args[3])
> ## Define the home directory
> home <- "some_working_directory"
> ## Where to save the results
> file_to_save <- paste(
+   home,
+   "results_subdirectory/",
+   "knobj",
+   c("Rand","Act", "Dream6")[active +1], c("Sing","Mult")[multi_mod +1],
+   seed,
+   sep=""
+ )
> ## Load datasets and library
> set.seed(seed)
> library(pauwels2014)
> data(exps)
> data(experiment_list1)
> data(observables)
> ## Initialize a knowledge list
> knobj <- generate_our_knowledge(transform_params)
> 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")
> ## Chosse a sample function
> if(multi_mod ){
+   sample_function <- sample_function_multi_mod_weight
+ }else{
+   sample_function <- sample_function_single_mod
+ }
> ## Update the knowledge list with the chosen strategy
```

```
> if(active==1){
+         knobj <- active_design(knobj, sample_function, seed, credits = 5000, file_to_save = file_to_save)
+ }
> if(active==0){
+         knobj <- random_design(knobj, sample_function, exps, seed, credits = 5000, file_to_save = file_to_sa
+ }
> if(active==2){
+         knobj <- dream6_design(knobj, sample_function, exps, seed, credits = 5000, file_to_save = file_to_sa
+ }
>
```

This script is available from the package and can be called from the shell. The `knobj` lists available from the package were computed using the following shell commands that could be passed to a resource manager for example.

```
> home <- "some_home/"
> path_to_R <- "some_path/R"
> n_seed <- 10
> seeds <- 1:n_seed
> active <- c(rep(2,n_seed),rep(1,2*n_seed),rep(0,n_seed))
> multi_mod <- c(rep(1,2*n_seed),rep(0,n_seed),rep(1,n_seed))
> commands <-
+   sprintf(
+     "%s --vanilla --args '%s' '%s' '%s' < %s",
+     path_to_R,
+     active,
+     multi_mod,
+     seeds,
+     paste(home, "perform_simu_one.R",
+     sep = ""
+   )
+ )
> print(commands)
 [1] "some_path/R --vanilla --args '2' '1' '1' < some_home/perform_simu_one.R"
 [2] "some_path/R --vanilla --args '2' '1' '2' < some_home/perform_simu_one.R"
 [3] "some_path/R --vanilla --args '2' '1' '3' < some_home/perform_simu_one.R"
 [4] "some_path/R --vanilla --args '2' '1' '4' < some_home/perform_simu_one.R"
 [5] "some_path/R --vanilla --args '2' '1' '5' < some_home/perform_simu_one.R"
 [6] "some_path/R --vanilla --args '2' '1' '6' < some_home/perform_simu_one.R"
 [7] "some_path/R --vanilla --args '2' '1' '7' < some_home/perform_simu_one.R"
 [8] "some_path/R --vanilla --args '2' '1' '8' < some_home/perform_simu_one.R"
 [9] "some_path/R --vanilla --args '2' '1' '9' < some_home/perform_simu_one.R"
[10] "some_path/R --vanilla --args '2' '1' '10' < some_home/perform_simu_one.R"
[11] "some_path/R --vanilla --args '1' '1' '1' < some_home/perform_simu_one.R"
[12] "some_path/R --vanilla --args '1' '1' '2' < some_home/perform_simu_one.R"
[13] "some_path/R --vanilla --args '1' '1' '3' < some_home/perform_simu_one.R"
[14] "some_path/R --vanilla --args '1' '1' '4' < some_home/perform_simu_one.R"
[15] "some_path/R --vanilla --args '1' '1' '5' < some_home/perform_simu_one.R"
[16] "some_path/R --vanilla --args '1' '1' '6' < some_home/perform_simu_one.R"
[17] "some_path/R --vanilla --args '1' '1' '7' < some_home/perform_simu_one.R"
[18] "some_path/R --vanilla --args '1' '1' '8' < some_home/perform_simu_one.R"
[19] "some_path/R --vanilla --args '1' '1' '9' < some_home/perform_simu_one.R"
[20] "some_path/R --vanilla --args '1' '1' '10' < some_home/perform_simu_one.R"
[21] "some_path/R --vanilla --args '1' '0' '1' < some_home/perform_simu_one.R"
[22] "some_path/R --vanilla --args '1' '0' '2' < some_home/perform_simu_one.R"
[23] "some_path/R --vanilla --args '1' '0' '3' < some_home/perform_simu_one.R"
[24] "some_path/R --vanilla --args '1' '0' '4' < some_home/perform_simu_one.R"
[25] "some_path/R --vanilla --args '1' '0' '5' < some_home/perform_simu_one.R"
[26] "some_path/R --vanilla --args '1' '0' '6' < some_home/perform_simu_one.R"
```

```
[27] "some_path/R --vanilla --args '1' '0' '7' < some_home/perform_simu_one.R"
[28] "some_path/R --vanilla --args '1' '0' '8' < some_home/perform_simu_one.R"
[29] "some_path/R --vanilla --args '1' '0' '9' < some_home/perform_simu_one.R"
[30] "some_path/R --vanilla --args '1' '0' '10' < some_home/perform_simu_one.R"
[31] "some_path/R --vanilla --args '0' '1' '1' < some_home/perform_simu_one.R"
[32] "some_path/R --vanilla --args '0' '1' '2' < some_home/perform_simu_one.R"
[33] "some_path/R --vanilla --args '0' '1' '3' < some_home/perform_simu_one.R"
[34] "some_path/R --vanilla --args '0' '1' '4' < some_home/perform_simu_one.R"
[35] "some_path/R --vanilla --args '0' '1' '5' < some_home/perform_simu_one.R"
[36] "some_path/R --vanilla --args '0' '1' '6' < some_home/perform_simu_one.R"
[37] "some_path/R --vanilla --args '0' '1' '7' < some_home/perform_simu_one.R"
[38] "some_path/R --vanilla --args '0' '1' '8' < some_home/perform_simu_one.R"
[39] "some_path/R --vanilla --args '0' '1' '9' < some_home/perform_simu_one.R"
[40] "some_path/R --vanilla --args '0' '1' '10' < some_home/perform_simu_one.R"
```

3 Problem specifications

In this section, we describe more some important aspects of the problem and related functions from the package. This should ease the process of adapting this code to different contexts.

3.1 Design strategy

A design strategy consists in choosing experiments sequentially. They consists in a molecular perturbation from `experiment_list1` and an observable from `observables`. Once this is chosen, the corresponding noisy simulation of the studied dynamical system is provided as a dataset and the simulation can continue. This is run until an initial credit budget has been spent. The objective is to estimate kinetic parameters of the underlying system as well as possible.

3.1.1 Random design

This is a baseline strategy. The experiments are chosen randomly among all possible experiments. The only task to perform at each stage is to estimate the parameters based on the data at hand. `random_design` simulate the process of following this strategy.

3.1.2 Active design

In order to perform active design, we implemented the Bayesian experimental design strategy described in the paper. The only difference with the previous strategy is that the experiment are chosen based on the estimation of an expected risk which is provided by the `estimate_risk_out_all` function. The parameter sampling schemes used for this strategy are the same as in the case of the random design strategy. The function `active_design` simulates the proposed active design strategy.

3.1.3 Dream6 design

We implemented the experimental design criterion given in [4]. We use the same sampling scheme as for the two previous strategies and estimate an expeced value of the criterion using function `estimate_risk_dream6`. `dream6_design` simulates this design strategy.

3.1.4 Organisation of this section

We present parts of the problem that could easily be modified by the user in order to adapt to a new setting. We first specify the probabilistic model and inference strategies in 3.2. Section 3.3 contains a desciption of how dynamics, molecular perturbations, observables, and stochastic perturbations were encoded in order to perform our simulation.

3.2 Model specificication and inference

3.2.1 Probabilistic model

We describe here the probabilistic model that was used for parameter inference. We first specify a `log_prior` function which is a large variance Gaussian prior.

```
> log_prior
```

```
function (theta)
{
    -1/2 * sum((theta - 50)^2/10000 + log(100))/length(theta)
}
<environment: namespace:pauwels2014>
```

We also need to specify a likelihood function. We use the same noise model as the one used in the generative noise process. The likelihood function allows to add prior information by penalizing large variations and large values of the kinetic variable which dynamics correspond to a chosen parameter value. It is preferable to use this kind of options when local maximization of the posterior is performed in order to avoid areas of the parameter space which lead to numerical instabilities in solving the corresponding initial value problems (see paper and supplementary information for details).

```
> log_likelihood
```

```
function (simu, simu_subset, data, fit = F)
{
    if ((sum(is.na(simu)) > 0)) {
        res <- -Inf
    }
    else {
        res <- -sum(((simu_subset - data)^2/((0.01 + (0.2 * simu_subset)^2)) +
            log((0.01 + (0.2 * simu_subset)^2)))[, -1])/2
        if (!fit) {
            res <- res - sum((simu_subset - data)^2/(10) + log(10))/2
            res <- res - 1/2 * max(simu[, -1]^2/1e+05 + log(1e-05))
            n <- nrow(simu)
            derivs <- simu[-c(1, 2), -1] - simu[-c(n - 1, n),
                -1]
            n1 <- nrow(derivs)
            second_derivs <- (derivs[-c(1, 2), ] - derivs[-c(n1 -
                1, n1), ])/2
            res <- res - 1/2 * (max(derivs^2) + max(second_derivs^2))
        }
        if (is.na(res)) {
            res <- -Inf
        }
    }
    res
}
<environment: namespace:pauwels2014>
```

Details about the arguments and the value of those functions are found in the manual of this package.

3.2.2 Parameter inference and sampling

Details are given in annex B of the paper. The first step of the generation of a sample is to find a local posterior maximum using `BFGS_special` which implements the BFGS quasi-newton method. The local maximum is used as an initialization point for an implementation of the Metropolis Hasting algorithm in function `generate_sample`. The process is repeated several time and a gaussian mixture model approximation is used to recombine samples. The function that performs all these steps is `sample_function_multi_mod_weight`. The parameters governing the behaviour of the local maximization function and the sampling function are found in `knobj$global_parameters`. The manual provides more details about those functions and their parameters.

In order to change the sampling strategy, a function of the same kind as `sample_function_multi_mod_weight` (e.g. `sample_function_single_mod`) should be saved as `sample_function`.

3.2.3 Risk

The last element of the decision process is the risk function. The risk is a dissimilarity function between two parameter values. In our simulations we implemented the following risk function.

```
> risk_theta_fun

function (theta1, theta2, n_params)
{
    1/n_params * sum(log(theta1/theta2)^2)
}
<environment: namespace:pauwels2014>
```

3.3 System specifications

We describe here how are encoded the simulation functions that define system dynamics, how they can be modified and what can be observed from this system.

3.3.1 Dynamics of the network

The dynamics of the small network we simulate obeys the following ordinary differential equation.

$$
\begin{aligned}
[\mathrm{as7}] &= \frac{\left(\frac{[\mathrm{p6}]}{\mathrm{r12_{Kd}}}\right)^{\mathrm{r12_h}}}{1+\left(\frac{[\mathrm{p6}]}{\mathrm{r12_{Kd}}}\right)^{\mathrm{r12_h}}} \\
[\mathrm{as9}] &= \frac{\left(\frac{[\mathrm{p6}]}{\mathrm{r11_{Kd}}}\right)^{\mathrm{r11_h}}}{1+\left(\frac{[\mathrm{p6}]}{\mathrm{r11_{Kd}}}\right)^{\mathrm{r11_h}}} \\
[\mathrm{rs7}] &= \frac{1}{1+\left(\frac{[\mathrm{p7}]}{\mathrm{r6_{Kd}}}\right)^{\mathrm{r6_h}}} \\
[\mathrm{g6}] &= 1 \\
[\mathrm{g7}] &= [\mathrm{as7}]\cdot[\mathrm{rs7}] \\
[\mathrm{g9}] &= [\mathrm{as9}] \\
\frac{\mathrm{d}([\mathrm{p6}])}{\mathrm{d}t} &= \mathrm{rbs6_{strength}}\cdot[\mathrm{v6_{mrna}}] - \mathrm{p6_{degradationRate}}\cdot[\mathrm{p6}] \\
\frac{\mathrm{d}([\mathrm{p7}])}{\mathrm{d}t} &= \mathrm{rbs8_{strength}}\cdot[\mathrm{v7_{mrna}}] - \mathrm{p7_{degradationRate}}\cdot[\mathrm{p7}] \\
\frac{\mathrm{d}([\mathrm{p8}])}{\mathrm{d}t} &= \mathrm{rbs7_{strength}}\cdot[\mathrm{v8_{mrna}}] - \mathrm{p8_{degradationRate}}\cdot[\mathrm{p8}] \\
\frac{\mathrm{d}([\mathrm{v6_{mrna}}])}{\mathrm{d}t} &= \mathrm{pro6_{strength}}\cdot[\mathrm{g6}] - \mathrm{v6_{mrnaDegradationRate}}\cdot[\mathrm{v6_{mrna}}] \\
\frac{\mathrm{d}([\mathrm{v7_{mrna}}])}{\mathrm{d}t} &= \mathrm{pro7_{strength}}\cdot[\mathrm{g9}] - \mathrm{v7_{mrnaDegradationRate}}\cdot[\mathrm{v7_{mrna}}] \\
\frac{\mathrm{d}([\mathrm{v8_{mrna}}])}{\mathrm{d}t} &= \mathrm{pro9_{strength}}\cdot[\mathrm{g7}] - \mathrm{v8_{mrnaDegradationRate}}\cdot[\mathrm{v8_{mrna}}]
\end{aligned}
$$

With kinetic variables [p6], [p7], [p8], [v6$_{\mathrm{mrna}}$], [v7$_{\mathrm{mrna}}$], [v8$_{\mathrm{mrna}}$] and kinetic parameters

$$\begin{aligned}
&\mathrm{v6}_{\mathrm{mrnaDegradationRate}}\\
&\mathrm{v7}_{\mathrm{mrnaDegradationRate}}\\
&\mathrm{v8}_{\mathrm{mrnaDegradationRate}}\\
&\mathrm{p6}_{\mathrm{degradationRate}}\\
&\mathrm{p7}_{\mathrm{degradationRate}}\\
&\mathrm{p8}_{\mathrm{degradationRate}}\\
&\mathrm{pro6}_{\mathrm{strength}}\\
&\mathrm{pro7}_{\mathrm{strength}}\\
&\mathrm{pro9}_{\mathrm{strength}}\\
&\mathrm{rbs6}_{\mathrm{strength}}\\
&\mathrm{rbs7}_{\mathrm{strength}}\\
&\mathrm{rbs8}_{\mathrm{strength}}\\
&\mathrm{r6}_{\mathrm{Kd}}, \mathrm{r6}_{\mathrm{h}}\\
&\mathrm{r11}_{\mathrm{Kd}}, \mathrm{r11}_{\mathrm{h}}\\
&\mathrm{r12}_{\mathrm{Kd}}, \mathrm{r12}_{\mathrm{h}}
\end{aligned}$$

Among them, we suppose that we have

$$\mathrm{v6}_{\mathrm{mrnaDegradationRate}} = \mathrm{v7}_{\mathrm{mrnaDegradationRate}} = \mathrm{v8}_{\mathrm{mrnaDegradationRate}} = 1$$

and

$$\mathrm{p6}_{\mathrm{degradationRate}} = \mathrm{p7}_{\mathrm{degradationRate}} = \mathrm{p8}_{\mathrm{degradationRate}}.$$

We suppose also that we know the r_{h} parameters,

$$\mathrm{r6}_{\mathrm{h}} = 4$$

$$\mathrm{r11}_{\mathrm{h}} = 2$$

$$\mathrm{r12}_{\mathrm{h}} = 2$$

as well as $\mathrm{r12}_{\mathrm{Kd}} = 0.2$. See also the `transform_params` function of the package. The parameter value used for simulation is found in `knobj$global_parameters$true_params_T`.

```
> knobj <- generate_our_knowledge(transform_params)
> knobj$global_parameters$true_params_T
```

```
p_degradation_rate              r6_Kd             r11_Kd
          50.00000           56.91622           55.01717
     pro6_strength      pro7_strength      pro9_strength
          50.00000           48.38483           59.60569
     rbs6_strength      rbs7_strength      rbs8_strength
          61.64950           61.64950           61.64950
```

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

```
mrna6_degradation_rate mrna7_degradation_rate mrna8_degradation_rate
                  1.00                   1.00                   1.00
    p_degradation_rate                  r6_Kd                  r6_h
                  0.10                   2.60                   4.00
                r11_Kd                  r11_h                 r12_Kd
                  2.00                   2.00                   0.20
                r12_h          pro6_strength          pro7_strength
                  2.00                   1.00                   0.80
         pro9_strength          rbs6_strength          rbs7_strength
                  3.77                   5.00                   5.00
         rbs8_strength
                  5.00
```

We encode this system in a compiled shared object which can be passed to the ode solver of the package `deSolve`. See the vignette [2] for more details about this. The `C` code used to represent these dynamics is available in `src/model0_simplified_mrna_rates.c` of the source of this package. It contains the follwing instructions:

```
/* file model0_simplified_mrna_rates.c */
/* Encodes a simplified network dynamics and structure(gene 6,7,8)*/

#include <R.h>
#include <math.h>
static double parms[16];
#define mrna6_degradation_rate parms[0]
#define mrna7_degradation_rate parms[1]
#define mrna8_degradation_rate parms[2]
#define p_degradation_rate parms[3]
#define r6_Kd parms[4]
#define r6_h parms[5]
#define r11_Kd parms[6]
#define r11_h parms[7]
#define r12_Kd parms[8]
#define r12_h parms[9]
#define pro6_strength parms[10]
#define pro7_strength parms[11]
#define pro9_strength parms[12]
#define rbs6_strength parms[13]
#define rbs7_strength parms[14]
#define rbs8_strength parms[15]

/* initializer */
void initmod(void (* odeparms)(int *, double *))
{
    int N=16;
    odeparms(&N, parms);
}

/* Derivatives and 1 output variable */
void derivs (int *neq, double *t, double *y, double *ydot,
             double *yout, int *ip)
{
    if (ip[0] <1) error("nout should be at least 1");

    double as7, as9;
    double rs7;
    double g6, g7, g9;

    /* Rules (skipping the pro* and rbs* variables which are constant and do not enter */
    /* the equations anyway */
    /* y[0] = g6=cte  y[1]=p6  y[2]=p7  y[3]=p8,  */
    /* y[4] = v6_mrna  y[5] = v7_mrna  y[6]=v8_mrna  */

    as7 = pow( y[1]/r12_Kd , r12_h ) /(1+ pow( y[1]/r12_Kd , r12_h ) );
    as9 =  pow( y[1]/r11_Kd , r11_h ) /(1 +  pow( y[1]/r11_Kd , r11_h ) );
    rs7 = 1/(1+ pow( y[2]/r6_Kd , r6_h ) );
    g6 = y[0];         /* cte */
    g7 = as7 * rs7;
    g9 = as9;

    /* mrna update (be carrefull with promotor numerotation !!) */
```

```
    ydot[4] = pro6_strength * g6 - mrna6_degradation_rate * y[4]; /* d(v6) */
    ydot[5] = pro7_strength * g9 - mrna7_degradation_rate * y[5]; /* d(v7) */
    ydot[6] = pro9_strength * g7 - mrna8_degradation_rate * y[6]; /* d(v8) */

    /* protein update (g9=y[0]=cte). ribosome and promoteur numbering  */

    ydot[0] = 0.;
    ydot[1] = rbs6_strength * y[4] - p_degradation_rate * y[1];  /* d(p6) */
    ydot[2] = rbs8_strength * y[5] - p_degradation_rate * y[2];  /* d(p7) */
    ydot[3] = rbs7_strength * y[6] - p_degradation_rate * y[3];  /* d(p8) */
    yout[0] = 1;
}
```

Note that the order of the parameters given by `transfom_params` function is the same as the order of the parameters expected by the `C` representation of the dynamics.

3.3.2 Obervables and perturbations

Observables are quantities that can be observed. The description of an observable is made through a list. Details are provided in the manual. The source code used to generate the `observables` data file from the package is as follows.

```
> superHighRes <-  0:200/2
> highRes <- 0:50 * 2
> lowRes <- 0:25 * 4
> observables <- list(
+   list(name = "p6", obs = c("time", "p6"),reso = superHighRes, cost = 400),
+   list(name = "p7", obs = c("time", "p7"),reso = superHighRes, cost = 400),
+   list(name = "p8", obs = c("time", "p8"),reso = superHighRes, cost = 400),
+   list(name = "mrnaHigh", obs = c("time", "v6_mrna", "v7_mrna", "v8_mrna"), reso = highRes, cost = 1000),
+   list(name = "mrnaLow", obs = c("time", "v6_mrna", "v7_mrna", "v8_mrna"), reso = lowRes, cost = 500)
+ )
> names(observables) <- sapply(observables, FUN = function(x){x$name})
```

Perturbations are modeled through functions that act on parameter and initial condition values. Details about these functions can be found in the manual of the package. The source code that was used to generate `experiment_list1` and `exps` data file is as follows.

```
> delete_gene6 <- function(theta, init){
+   theta[names(theta) %in% c("pro6_strength","rbs6_strength")] <- 0
+   init[names(init) == "p6"] <- 0
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 800
+   res
+ }
> delete_gene7 <- function(theta, init){
+   theta[names(theta) %in% c("pro7_strength","rbs8_strength")] <- 0
+   init[names(init) == "p7"] <- 0
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 800
+   res
+ }
> delete_gene8 <- function(theta, init){
+   theta[names(theta) %in% c("pro9_strength","rbs7_strength")] <- 0
+   init[names(init) == "p8"] <- 0
+   res <- c()
+   res$theta <- theta
```

```
+   res$initial_conditions <- init
+   res$cost <- 800
+   res
+ }
> knockdown_gene6 <- function(theta, init){
+   theta[names(theta) %in% "mrna6_degradation_rate"] <- 10
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 350
+   res
+ }
> knockdown_gene7 <- function(theta, init){
+   theta[names(theta) %in% "mrna7_degradation_rate"] <- 10
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 350
+   res
+ }
> knockdown_gene8 <- function(theta, init){
+   theta[names(theta) %in% "mrna8_degradation_rate"] <- 10
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 350
+   res
+ }
> decrease_rbs_gene6 <- function(theta, init){
+   theta[names(theta) %in% "rbs6_strength"] <- theta[names(theta) %in% "rbs6_strength"] / 10
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 450
+   res
+ }
> decrease_rbs_gene7 <- function(theta, init){
+   theta[names(theta) %in% "rbs8_strength"] <- theta[names(theta) %in% "rbs8_strength"] / 10
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 450
+   res
+ }
> decrease_rbs_gene8 <- function(theta, init){
+   theta[names(theta) %in% "rbs7_strength"] <- theta[names(theta) %in% "rbs7_strength"] / 10
+   res <- c()
+   res$theta <- theta
+   res$initial_conditions <- init
+   res$cost <- 450
+   res
+ }
> nothing <- function(theta, init){
+   list(theta = theta, initial_conditions = init, cost = 0)
+ }
> experiment_list1 <- c()
> experiment_list1$delete_gene6 <- delete_gene6
> experiment_list1$delete_gene7 <- delete_gene7
> experiment_list1$delete_gene8 <- delete_gene8
```

```
> experiment_list1$knockdown_gene6 <- knockdown_gene6
> experiment_list1$knockdown_gene7 <- knockdown_gene7
> experiment_list1$knockdown_gene8 <- knockdown_gene8
> experiment_list1$decrease_rbs_gene6 <- decrease_rbs_gene6
> experiment_list1$decrease_rbs_gene7 <- decrease_rbs_gene7
> experiment_list1$decrease_rbs_gene8 <- decrease_rbs_gene8
> experiment_list1$nothing <- nothing
> exps <- lapply(names(experiment_list1), function(name){
+     t(sapply(observables, function(obs){
+         cbind(obs$name, obs$cost + experiment_list1[[name]](0,0)$cost, name)
+       }
+     ))
+   }
+ )
> exps <- Reduce(exps, f = function(s,t){rbind(s,t)})
> exps <- data.frame(Measurement = exps[,1], Cost = as.numeric(exps[,2]), exp = exps[,3])
```

3.3.3 Noise generative process

This describes how noise affects the true dynamics of the system. The noise model we have is an independant heteroscedastic centered Gaussian noise which variance has the expression $0.01 + 0.04x^2$ where x is the non corrupted data. The function `add_noise` specifies this process.

```
> add_noise

function (data_theta_Ts)
{
    temp <- cbind(data_theta_Ts[, 1], matrix(rnorm(n = (ncol(data_theta_Ts) -
        1) * nrow(data_theta_Ts), mean = as.numeric(data_theta_Ts[,
        -1]), sd = sqrt(0.01 + 0.04 * as.numeric(data_theta_Ts[,
        -1])^2)), nrow(data_theta_Ts), ncol(data_theta_Ts) -
        1))
    dimnames(temp) <- dimnames(data_theta_Ts)
    temp
}
<environment: namespace:pauwels2014>
```

4 Conclusion

The purpose of this package is to allow users to reproduce the results reported in our article. One of the objectives of the development of active experimental design strategies is to provide methods that could be used by non specialists, we consider that reproducibility is a key issue in this realm.

In addition, this package could be used as a benchmark to compare newly proposed design strategy. As implementation is readilly available, it should not cost much to compare new parameter estimation schemes or design strategies provided that they can be implemented in a way intelligible for a computer.

References

[1] 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.

[2] Karline Soetaert, Thomas Petzoldt, and R Woodrow Setzer. R package desolve, writing code in compiled languages. *R package vignette, URL http://CRAN. R-project. org/package= deSolve*, 2009.

[3] Karline Soetaert, Thomas Petzoldt, and R. Woodrow Setzer. Solving differential equations in r: Package desolve. *Journal of Statistical Software*, 33(9):1–25, 2010.

[4] Bernhard Steiert, Andreas Raue, Jens Timmer, and Clemens Kreutz. Experimental design for parameter estimation of gene regulatory networks. *PLoS ONE*, 7(7):e40052, 07 2012.

[5] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009.

```
                  [,1]        [,2]        [,3]        [,4]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,8      List,8
experiments       Character,8 Character,8 Character,8 Character,8
                  [,5]        [,6]        [,7]        [,8]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,8      List,8
experiments       Character,8 Character,8 Character,8 Character,8
                  [,9]        [,10]       [,11]       [,12]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,7      List,8
experiments       Character,8 Character,8 Character,7 Character,8
                  [,13]       [,14]       [,15]       [,16]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,7      List,8      List,8      List,8
experiments       Character,7 Character,8 Character,8 Character,8
                  [,17]       [,18]       [,19]       [,20]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,8      List,8      List,8      List,8
experiments       Character,8 Character,8 Character,8 Character,8
                  [,21]       [,22]       [,23]       [,24]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,6      List,6      List,5      List,6
experiments       Character,6 Character,6 Character,5 Character,6
                  [,25]       [,26]       [,27]       [,28]
transform_params  ?           ?           ?           ?
global_parameters List,25     List,25     List,25     List,25
datas             List,6      List,6      List,5      List,7
experiments       Character,6 Character,6 Character,5 Character,7
                  [,29]       [,30]
transform_params  ?           ?
global_parameters List,25     List,25
datas             List,6      List,5
experiments       Character,6 Character,5
```