EvoPER – An R package for applying evolutionary computation methods in the parameter estimation of individual-based models implemented in Repast

Antonio Prestes García[1] and Alfonso Rodríguez-Patón[1]

[1]**Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, Madrid, Spain**

ABSTRACT

Individual-based models are complex and they normally have an elevated number of input parameters which must be tuned in order to reproduce the experimental or observed data as accurately as possible. Hence one of the weakest points of such kind of models is the fact that rarely the modeler has the enough information about the correct values or even the acceptable range for the input parameters. Therefore, several parameter combinations must be checked to find an acceptable set of input factors minimizing the deviations of simulated and observed data. In practice, most of the times, is computationally unfeasible to traverse the complete search space to check all parameter combination in order to find the best of them. That is precisely the kind of combinatorial problem suitable for evolutionary computation techniques. In this work we present the EvoPER, an R package for simplifying the parameter estimation using evolutionary computation techniques. The current version of EvoPER includes implementations of PSO, SA and ACO algorithms for parameter estimation of models generated with the open source agent-based modeling toolkit Repast.

Keywords: Individual-Based Modeling, Parameter Estimation, Evolutionary Computation, Systems Biology

INTRODUCTION

The Individual-based modeling and simulation is a powerful methodology which is having more and more adoptions between researchers and practitioners of distinct branches of ecological modeling and microbial consortia. Certainly one of the main reasons for the success of this approach is the extreme simplicity for capturing micro-level properties, stochasticity and spatially complex phenomena without the requirement of a high level of mathematical background (Grimm and Railsback (2005)). But the counterpart of that facility to build complex models, is the difficulty to make credible results which can be attributed in part to the fact that modelers are prone to circumvent a thoroughly analysis for simulation output.

There are several reasons for the situation previously mentioned. The first and perhaps most important is that modeling and simulation is a vast discipline with a broad and complex body of knowledge having a theoretical background under the surface (Minsky (1965); Zeigler et al. (2000); Boccara (2003)) which are not completely mastered from modelers coming from disperse domains like biology, ecology or even computer science. Of course, this should not be an obstacle for the development of good models by the practitioners. We believe that the availability of easy tools for the model tuning and analysis which efficiently encapsulate such complex subject can greatly help to improve the quality and significance of simulation results.

In the next sections we will describe the scope and the usage examples of the EvoPER R package which has been developed for facilitating the tasks of estimating the parameters of Individuals-based models.

BACKGROUND

The terms model calibration and parameter estimation, although informally are used interchangeably and being functionally similar are semantically distinct entities having a different scope and objectives. In order to provide a more formal definition of these terms let us briefly define the basic structure of a mathematical model. A model is normally expressed as some form of the algebraic composition expressing the relationship between of three element types, namely the independent variables, the dependent or the state variables and finally the constants. Therefore, for the sake of simplicity, a model expressing some linear relationship between variables is shown bellow

$$y = \alpha + \beta x$$

where x and y are independent and the state variable respectively and α and β are the model constants. The model constants are referred as the model parameters which necessarily do not have to have any correspondence to some element in the system being modeled (Beck and Arnold (1977)). The direct problem is, being known the model structure and also knowing the independent variables and the parameters, to estimate the value of state variable. Of course this oversimplified case is rarely seen when modeling real systems, especially when dealing with biological systems. In addition, in the most cases the constants and the independent variables are impossible to observe directly being also unknown the right model structure for representing the system under study.

Usually the only value elucidated experimentally or backed by observations of some population data is the state variable; therefore, the parameters which are the structural part of model must be estimated having as the only reference, the measurements of dependent variable. Hence the term **calibration** can be defined as the procedure to where the values of state variable ”y” are compared to the known standard values, let's say ”Y”, which in the context of biological research are those sampled from population true values Zeigler et al. (2000).

On the other hand, the **parameter estimation** is the task of estimating the values of the constants of a model and it can be seen somehow as an inverse problem, since we are using the reference values Y in order to determine the suitable values for the model constants (Ashyraliyev et al. (2009); Beck and Arnold (1977)). The parameter estimation procedure implicitly encompasses the calibration process as, in order to discover the values for the constants the model outputs must be checked to the reference values. Thus the problem can be also stated as an optimization problem, just because the process requires the search for the minimum values of some function $f(y_i, Y_i)$ measuring the distance between y_i and Y_i which are the simulated and the reference values respectively.

The function measuring how close are the observed and the reference values is the goodness of fit metric for assessing how well the model is able to reproduce the reference data. In other words, the metric gives a numerical hint about how close are the output of model to the reference data. There are fundamentally three approaches to define the goodness of fit for a model (Thiele et al. (2014)). The first approach is based on using acceptable ranges for the model outputs being the most straightforward one. That approach is also known as categorical calibration and works defining intervals for the model output values and when the output falls inside the interval it is considered as having a good fit. One of the main drawback of this approach is the fact that it is not possible to determine how close are the model and the reference data. The second metric relies on measuring the differences between simulated and observed values, being the least squares the most commonly used method for computing the quality of fit (Beck and Arnold (1977)). Finally, that last approach requires the use of likelihood functions. It is hard to implement and requires that the underlying distribution must be known.

In order to explore the search space, the calibration process requires many model executions as well as many evaluations of goodness of fit function over the output data in order to find the best estimation for the model parameters. This is a computationally expensive task, especially in the case of Individual-based models, as the problem bounds increases with model complexity and the number of input parameters which must be tested. Roughly speaking there are basically two different approaches for generating the sample points required for estimating parameters. The first of them is based on the definition of sampling schemes such as Monte Carlo, Factorial designs or the Latin Hypercube sampling that works by generating an a priori set of samples in the search space, that is to say, a set of parameter combinations for running model with all of these sampling points (Thiele et al. (2014); Viana (2013)). On the other hand, in the case of optimization methods, we have to generate an initial set of points sampled from the input space and modify them dynamically to search for neighboring solutions which could approximate better to the minima. The exact method depends on the evolutionary algorithm chosen for parameter estimation.

DESCRIPTION

In order to facilitate the parameter estimation task of Individual-based models we introduce the GNU R (R Core Team (2015)) package **EvoPER** - Evolutionary Parameter Estimation for Repast, an open source project intended to facilitate de adoption and application of evolutionary optimization methods and algorithms to the parameter estimation of IBMs developed using the Repast Symphony framework North et al. (2013). The EvoPER package is released under the MIT license being the binaries available for download from CRAN (`https://cran.r-project.org/web/packages/evoper/`) and the complete source code for the project can be found on GitHub (`https://github.com/antonio-pgarcia/evoper`).

The package EvoPER provides implementations of common evolutionary algorithms specially crafted for search the optimum values for Individual-based models developed in Repast Simphony. Current version of EvoPER package supports the Particle Swarm Optimization (PSO) (Kennedy and Eberhart (1995)), the Simulated Annealing (SA) (Kirkpatrick et al. (1983)) and the Ant Colony Optimization (ACO) (Dorigo et al. (2006)) algorithms for parameter estimation. We also plan to support more algorithms in future versions. All of these algorithms use some kind of natural or physical system analogy having each of them subtleties making them suitable for different types of problems. Nonetheless, despite of the differences in the natural metaphor chosen all algorithms share an important aspect which is that the search space is traversed downhill but allowing uphill moves in order to avoid to get trapped in a local optimum far from the global one.

The basic PSO algorithm uses the idea of particles moving in a multidimensional search space being the direction controlled by the *velocity*. The velocity has two components, one towards to the direction of best value of particle p_i and other towards to the best value found in the neighborhood of particle p_i (Kennedy and Eberhart (1995)). The behavior and convergence of the algorithm is controlled by the particle population size and by the ϕ_1, ϕ_2 parameters which respectively controls the particle acceleration towards the local and the neighbor best. The algorithm implementation and the default values for the algorithm parameters follows the guidelines and standard values provided by (Clerc (2012)).

On the other hand, the Simulated Annealing uses the idea of cooling scheme to control how the problem solutions are searched. The algorithm generates an initial solution and then iterates, searching for neighbor solutions accepting new solutions when they are better than the current solution or with some probability P which is function of current temperature and the cost of solutions. Important parameters are the initial temperature T_0, the final temperature and the cooling scheme Kirkpatrick et al. (1983). In our implementation the default function for temperature update is $T = \alpha T$, being α the parameter controlling how fast the temperature is decremented.

The package designed using an object-oriented approach being structured around the classes representing the objective function to be minimized. These classes are the basic input for the optimization algorithms available on the EvoPER package. There is a parent class called *ObjectiveFunction* with two subclasses, namely the *PlainFunction* and the *RepastFunction*. The purpose of the first subclass is allow the user run the optimization algorithms to their own mathematical functions, the second subclass encapsulates the Repast Model calls and perform the parameter estimation. A brief description of package classes and the main methods is given in Table 1.

Table 1. The EvoPER classes for encapsulating the objective function for parameter estimation.

Class name	Methods	Description
ObjectiveFunction		The base class in hierarchy providing the skeleton for running the optimization algorithms.
	Parameter	Sets a model parameter with range between a minimum and a maximum values.
	GetParameter	Returns a previously defined parameter.
	Evaluate	Evaluate the objective function.
	Value	Returns the value of last objective function evaluation.
PlainFunction		Allows the optimization of plain functions implemented in R.
	initialize	Class constructor. Requires any R function as parameter. For instance $f < -function(x_1, x_2)\{(1-x_1)^2 + 100(x_2 - x_1^2)^2\}$
	Evaluate	Override superclass method to the specific function call.
RepastFunction		Wrapper the Repast Model
	initialize	Requires the model directory, an aggregated data source, the simulation time and a user defined cost function.
	Evaluate	Override superclass method to the specific function call.

The object oriented approach allows the easy extension of the package for other types of Individual-based modeling tools or methods. As can be seen in Table 1 the only requirement to apply the methods contained in the EvoPER package is to extend the *ObjectiveFunction* class and override the Evaluate method to support the new parameter estimation target. One of the useful aspects of EvoPER implementation is the possibility to specify constraints in the search space by individually setting lower and upper bounds for every parameter being analyzed using the *ObjectiveFunction$Parameter(name, min, max)* method. That is an important point for limiting the parameter values only to the acceptable biological range.

The workflow for carry out the parameter estimation consists in a simple sequence of steps. First, an object instance of any *ObjectiveFunction* subclasses must be created and properly initialized. As mentioned previously, currently we have two options available for parameter estimation: one for simple functions which could be used for testing purposes (*PlainFunction*) and another for estimating parameters of Repast models (*RepastFunction*). Once the objective function has been initialized, the required parameters must be provided with the appropriate lower and upper bounds. Finally, the *extremize* function can be applied to the previously defined function. The required parameters are the optimization method and the objective function instance. The function has a third optional parameter for providing the custom options for the underlying optimization method.

The optimization functions and its accessory helper functions are shown in the Table 2 for providing an overview on the package contents, the package is in continuous improvement and development therefore the list could change over the time. The package manual will be the most updated source of information for the package contents.

Table 2. The partial list of EvoPER optimization functions for parameter estimation.

Function	Description
abm.pso	The function call for running the particle swarm optimization method. The parameters are the *ObjectiveFunction* and an instance of *Options* class whith the suitable parameter set.
pso.neighborhood.K2	This neighborhood function returns two neighbors of particle x_i, where the neighbors are the particles x_{i-1} and x_{i+1} using a ring topology Zambrano-Bigiarini et al. (2013).
pso.neighborhood.K4	Returns four neighbors of particle x_i using a von Neumann neighborhood function.
pso.neighborhood.KN	Return the whole set of particles. The neighborhood is a complete graph.
pso.Velocity	Calculate the particle velocity Poli et al. (2007)
pso.chi	Calculate the constriction coefficient Poli et al. (2007).
initSolution	Creates a random initial population of size N for the model parameters.
enforceBounds	Verify the upper and lower limits of every parameter
abm.saa	The Simulated Annealing implementation. The parameters required are the *ObjectiveFunction* and an instance of *Options* class whith the suitable parameter set.
saa.neighborhood1	Generate a neighborhood solution for simulated annealing perturbing randomly one value from current best solution and using the distance parameter.
saa.neighborhoodH	Generate a neighborhood solution for simulated annealing perturbing randomly the half of values of current best solution.
saa.neighborhoodN	Generate a neighborhood solution for simulated annealing perturbing randomly all values of current best solution.
extremize	This is a wrapper encapsulating the calls for all parameter estimation methods. The parameters are the optimization method, the *ObjectiveFunction* and an instance of *Options* class whith the suitable parameter set.

Most of the aspects implemented in the optimization code are standard and, perhaps the only points which are specific to the EvoPER package, are the neighborhood function for *pso.neighborhood.K4* and *saa.neighborhood*. The von Neumann neighborhood for particle swarm optimization is generated using a topology created converting the linear collections of particles to a matrix using the R code `m <- matrix(seq(1,N),nrow=(ceiling(sqrt(N))))` where N is the swarm size.

In the case of neighborhood solution for Simulated Annealing we have used the following logic for generating new solutions: first we pick randomly the parameters to be perturbed[1] and update them using the expression $S' = S + S * U(-1,1) * distance$ where S', S, U and $distance$ are respectively the new neighbor solution, the current solution, a uniform random number between $[-1,1]$ and the desired distance from current solution.

The package provides acceptable default values for most of parameters related to the optimization method in use. In spite of the fact that the parameter estimation functions can be called directly, the users should use the function *extremize(m, f, o)* which is the standard entry point for the optimization methods. As has been mentioned previously, the function has three parameters, which are respectively the method (*m*), the objective function (*f*) and the options (*o*). Only the first two are required and the third is optional. When the options parameter is not provided the default values are used. If setting different from the default values are required, the user must pass an instance of the corresponding option class. For example, if more iterations are required for PSO method an instance of *OptionsPSO* must be created and the method *setValue(”iterations”, value)* with the appropriate value. Many other parameters can be customized in order to fit the specific needs for the model being analyzed such as the neighborhood functions or the temperature update for the simulated annealing.

[1]Our implemented neighborhood functions allows to choose from 1, 1/2 n or n, being n the number of parameters

EXAMPLES

In this section we will show some small and illustrative examples about how to use the EvoPER package for estimating the model parameters. It is worth mentioning that although the package is oriented to the application of evolutionary optimization methods to the parameter estimation of models developed using Repast Simphony it can also be used to minimize basic mathematical functions. In the following example shown in Figure 1 we demonstrate the package usage applying it to the two variables Rosenbrock's function.

```
# Step 0
rm(list=ls())
set.seed(161803398)
library(evoper)

# Step 1
rosenbrock2<- function(x1, x2) { (1 - x1)^2 + 100 * (x2 - x1^2)^2 }

# Step 2
objective<- PlainFunction$new(rosenbrock2)

# Step 3
objective$Parameter(name="x1",min=-100,max=100)
objective$Parameter(name="x2",min=-100,max=100)

# Step 4
results<- extremize("pso", objective)
```

Figure 1. A simple example for minimizing the Rosenbrock's function using the EvoPER package.

As can be seen in Figure 1 the **step 1** shows the definition of a simple function to be minimized; the **step 2** demonstrate how to create an instance of *PlainFunction* class; in the **step 3** the parameter ranges for each function's parameter is provided and finally in the **step 4** the EvoPER *extremize* function is used to minimize the objective function. The results of running the example are shown in Figure 2 where can be seen the estimated parameters, the value of fitness function, the execution time and the number of times the function has been evaluated.

```
> system.time(results<- extremize("pso", f))
   user  system elapsed
   1.50    0.00    1.53
> results
        x1       x2 pset      fitness
1 1.000762 1.001341    4 3.948505e-06
> f$stats()
     total_evals converged
[1,]        1616         1
```

Figure 2. The R console output session showing the results of running the previous example.

One of important aspects is that the syntax is simple and consistent independent of the function for which parameters are being estimated. In next example shown in Figure 3 we can observe the simplicity for running the optimization code for Repast parameter estimation. As can be seen the same steps are required: (1) create the function to minimize based on the model characteristics; (2) Create the *RepastFunction* instance with model data; (3) Initialize the model parameters with the acceptable ranges and (4) Run the optimization function. In this example we are basically trying to find the best combination of model parameters which minimize the differences between the observed and the simulated data for the variable *Rate* and the method used is the normalized root mean square deviation.

Finally, in the last example show in Figure 4 we want to show an example on how to craft the cost function for tuning the model parameters in order to accomplish a specific output. Specifically, a simple toy model representing the Lotka-Volterra, also known predator-prey is presented and we want to estimate the parameters required to make the output oscillate with an approximate period of twenty-four hours. This model, despite of being developed for modeling the predator and prey relationship, has a broad range of applications and can be used for representing a many types of ecological and biological interactions

```
# Step 0
rm(list=ls())
set.seed(161803398)
library(evoper)

# Step 1
my.cost<- function(params, results) {
   Rate<- AoE.NRMSD(results$simulated, results$experimental)
   criteria<- cbind(Rate)
   return(criteria)
}

# Step 2
objective<- RepastFunction$new("/usr/models/BactoSim","ds::Output",300,my.cost)

# Step 3
objective$Parameter(name="cyclePoint",min=1,max=90)
objective$Parameter(name="conjugationCost",min=0,max=100)
objective$Parameter(name="pilusExpressionCost",min=0,max=100)
objective$Parameter(name="gamma0",min=1,max=10)

# Step 4
results<- extremize("saa", objective)
```

Figure 3. **The minimum code required to accomplish the parameter estimation for a repast model.**

(Shonkwiler (2008)). The parameters we are trying to estimate are $c1$, $c2$, $c3$ and $c4$ which represent respectively the growth rate of prey, the predation rate, the predation effect on predator growth rate and finally the death rate of predator. The session output is presented in Figure 5 where the values for the parameters required to produce oscillations with the desired period are shown. The Figure 6 shows graphically the results for the tuned parameters.

```
# Step 0
rm(list=ls())
set.seed(161803398)
library(evoper)

# Step 1
my.cost<- function(params, results) {
   predators<- AoE.NRMSD(period(results$predators),24)
   criteria<- cbind(predators)
   return(criteria)
}

# Step 2
objective<- RepastFunction$new("/usr/models/PredatorPrey","ds::population",180,my.cost)

# Step 3
f$Parameter(name="c1",min=0.5,max=8)
f$Parameter(name="c2",min=0.5,max=8)
f$Parameter(name="c3",min=0.5,max=8)
f$Parameter(name="c4",min=0.5,max=8)

# Step 4
results<- extremize("pso", objective)
```

Figure 4. **Tuning the oscillation period of predator-prey model.**

CONCLUSIONS

The systematic parameter estimation should be a fundamental part of individual-based modeling but it is normally omitted by modelers. One of the main reasons is the relative complexity of available methods and the lack of simple tools for the practitioners which usually come from different domains with different backgrounds. Individual-based models are complex and non-linear and the evaluation of model's input parameters is precisely the kind of combinatorial optimization problem for which evolutionary computation provides good results.

In this work we have introduced the set of features available on EvoPER package alongside with

```
> system.time(results<- extremize("pso", f))
   user  system elapsed
   91.2    0.00   91.29
> results
         c1        c2       c3       c4 pset      fitness
1 0.6305862 0.8146169 1.192911 1.611731    4  5.01271e-03
> f$stats()
     total_evals converged
[1,]         800         0
```

Figure 5. The R console output session showing the results of running predator-prey model in Figure 4.

some brief usage cases. The package is being developed bearing in mind the idea of minimizing the effort required to the application of sophisticated methods in the parameter estimation process of Individual-based models. This package will allow the modelers to try different alternatives without having to code ad hoc and complex integration code to the existent packages.

ACKNOWLEDGMENTS

This work was supported by the European FP7 - ICT - FET EU research project: 612146 (PLASWIRES "Plasmids as Wires" project) `www.plaswires.eu` and by Spanish Government (MINECO) research grant TIN2012-36992.

REFERENCES

Ashyraliyev, M., Fomekong-Nanfack, Y., Kaandorp, J. A., and Blom, J. G. (2009). Systems biology: Parameter estimation for biochemical models.

Beck, J. V. and Arnold, K. J. (1977). *Parameter estimation in engineering and science*. Wiley series in probability and mathematical statistics. Wiley, New York.

Boccara, N. (2003). *Modeling Complex Systems (Graduate Texts in Contemporary Physics)*. Springer, 1 edition.

Clerc, M. (2012). Standard Particle Swarm Optimisation.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

Grimm, V. and Railsback, S. F. (2005). *Individual-based Modeling and Ecology: (Princeton Series in Theoretical and Computational Biology)*. Princeton University Press, Princeton.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948 vol.4.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):pp. 671–680.

Minsky, M. (1965). Matter, Mind and Models. In *Proceedings of IFIP Congress 65*, pages 45–49.

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling*, 1(1):3.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Shonkwiler, R. W. (2008). *Mathematical Biology: An Introduction with Maple and Matlab*. Springer Publishing Company, Incorporated, 2nd edition.

Thiele, J. C., Kurth, W., and Grimm, V. (2014). Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and 'R'. *Journal of Artificial Societies and Social Simulation*, 17(3).

Viana, F. A. C. (2013). Things You Wanted to Know About the Latin Hypercube Design and Were Afraid to Ask. *10th World Congress on Structural and Multidisciplinary Optimization*, pages 1–9.

Zambrano-Bigiarini, M., Clerc, M., and Rojas, R. (2013). Standard Particle Swarm Optimisation 2011 at CEC-2013: A baseline for future PSO improvements. In *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pages 2337–2344.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of Modeling and Simulation, Second Edition*. Academic Press, 2 edition.

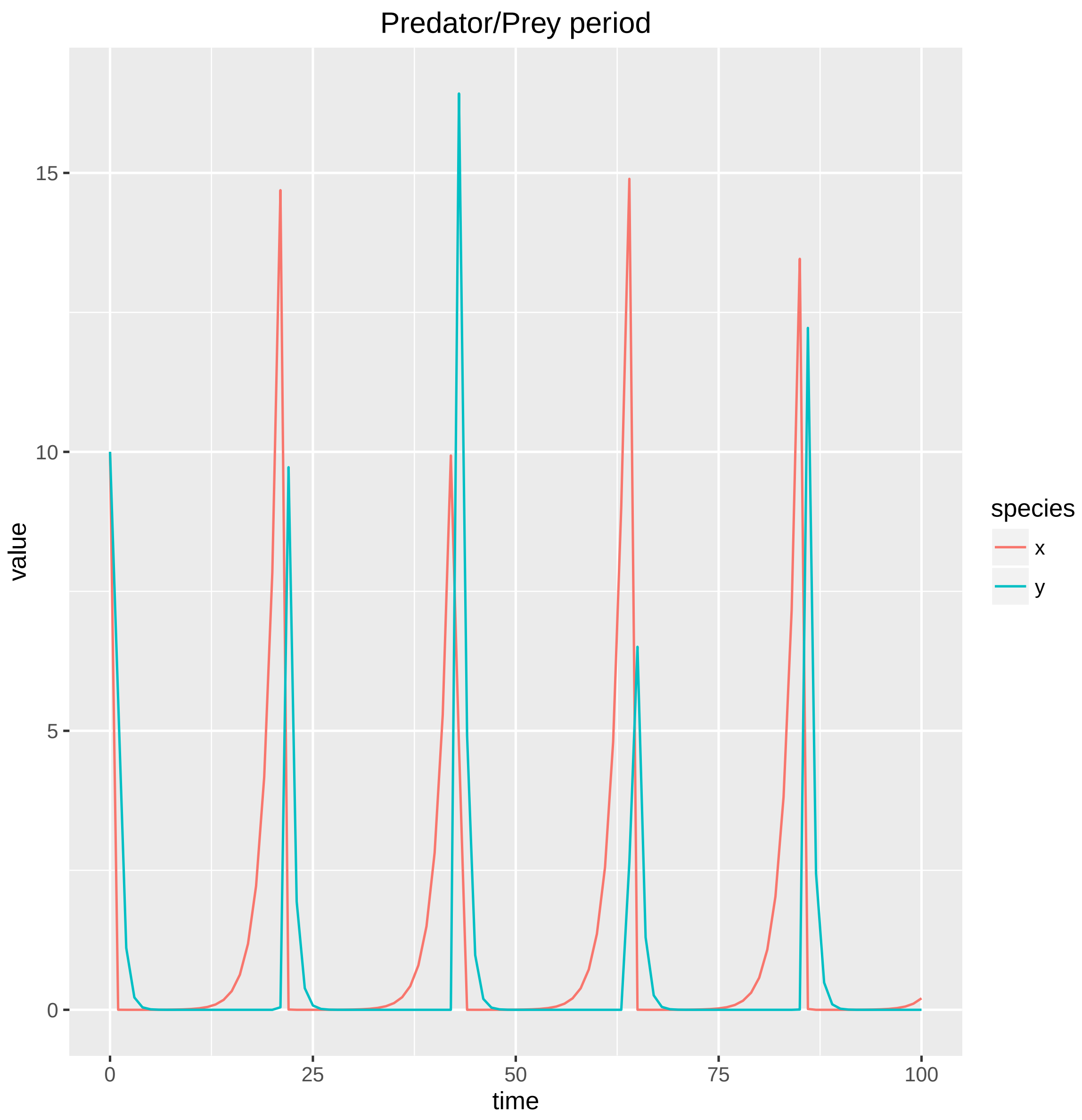

Figure 6. **An example of tuning the oscillation period of predator-prey model. The objective function can be tweaked to reproduce any desired output behavior. The most common one is to assess the quality of fit between simulated and experimental data but it is no limited and can be used to find parameter combinations which generate some global behavior. In this figure we can observe how *x* and *y* species, respectively the prey and predator components oscillates with an approximated period of 24 hours using the parameter combination shown in Figure 5.**