OPTFERM – A COMPUTATIONAL PLATFORM FOR THE OPTIMIZATION OF FERMENTATION PROCESSES

Orlando Rocha[1,2,3], Paulo Maia[1,2], Isabel Rocha[1,3], Miguel Rocha[2]
[1]IBB – Institute for Biotechnology and Bioengineering / Centre for Biological Engineering
[2]CCTC – Computer Science and Technology Center / Dep. Informatics - Universidade do Minho
Campus de Gualtar, 4710-057 Braga, Portugal
E-mails: {orocha,pmaia,irocha}@deb.uminho.pt, mrocha@di.uminho.pt
[3]Biotempo, Lda., Avepark – Zona Industrial Gandra, Apartado 4152, 4806–909, Caldas Taipas, Portugal

KEYWORDS
Fermentation processes, open-source software, process simulation and optimization, Evolutionary Computation, Differential Evolution

ABSTRACT

We present *OptFerm*, a computational platform for the simulation and optimization of fermentation processes. The aim of this project is to offer a platform-independent, user-friendly, open-source and extensible environment for Bioengineering process optimization that can be used to increase productivity. This tool is focused in optimizing a feeding trajectory to be fed into a fed-batch bioreactor and to calculate the best concentration of nutrients to initiate the fermentation. Also, a module for the estimation of kinetic and yield parameters has been developed, allowing the use of experimental data obtained from batch or fed-batch fermentations to reach the best possible model setup.
The software was built using a component-based modular development methodology, using Java as the programming language. *AIBench,* a Model-View-Control based application framework was used as the basis to implement the different data objects and operations, as well as their graphical user interfaces. Also, this allows the tool to be easily extended with new modules, currently being developed.

INTRODUCTION

Nowadays, several products such as antibiotics, proteins, amino-acids and other chemicals are produced using fermentation processes. Due to the rise of petroleum prices and the incentive to replace petroleum derivatives by "green products", many traditional processes have been replaced by new biotechnological ones. Consequently, an effort to improve biotechnological techniques has been verified. Recombinant DNA applications were conceived to produce new microorganisms, while several computational tools have been designed and implemented for modelling and simulation of metabolic pathways of the cell (Pettinen et al. 2005). All share a common purpose: to increase the production yield and get a higher purity of the final product.
To optimize the productivity of a biological process, in the majority of the cases, two different steps have to be addressed: firstly, a selection and genetic improvement of the organism strain is accomplished; in a second step, the best conditions of the fermentation process are identified, such as the initial nutrient concentrations, operating modes, feeding profiles for fermentations, temperature and pH.
In industry, the second step is mostly done experimentally using trial-and-error heuristics (Kawohl et al. 2007). Although there are several tools to study, simulate and optimize cellular pathways, there is still a clear lack of tools to perform the optimization of fermentation processes.
Fermentation processes are affected by biochemical and chemical phenomena such as the chemical interactions between components, concentrations of substrates, products and biomass, and environmental conditions like temperature, pH and dissolved oxygen concentration (Tzoneva 2006; Zhang 2008). The complex dynamic behavior and the unpredictable effects of these factors increase the difficulty of establishing accurate models to describe the real systems (Benjamin et al. 2008). However, new methods to control, predict and optimize bioprocesses have been proposed.
The OptFerm platform was developed using the Java programming language, with the aim of being a user-friendly, extensible and platform-independent tool. It was designed to allow the user to evaluate and compare several different methods for the tasks of simulation, optimization and parameter estimation, in the context of fermentation processes. The aim is to allow users to improve process productivity, achieving better results in reduced times.
The available optimization algorithms in this tool were developed and validated in previous work by the authors, namely Evolutionary Algorithms (Rocha et al. 2004, 2007; Mendes et al 2006, 2008), Differential Evolution (Mendes et al. 2006, 2008) and Simulated Annealing (Rocha et al. 2007). Any of these algorithms can be used in feed optimization or parameter estimation. Metaheuristic optimization approaches are used, since the underlying problems are typically quite complex. OptFerm is available in the following website: http://darwin.di.uminho.pt/optferm.

MAJOR FEATURES

The main aim of the OptFerm software is to provide specific computational tools for the simulation and optimization of fermentation processes. The tools should enable its users to use several methods and parameter configurations, thus saving time in performing expensive wet experiments.

Fermentation models

The basis for all operations available within OptFerm are the models of the fermentation processes. The internal representation of a model is based on ordinary differential equations (ODEs). In OptFerm, model information can be divided in two main entities, a *Process* and a *Function:*

- *Process* – contains information on the state variables such as names, initial values and upper and lower limits, and the objective function for optimization purposes.
- *Function* – keeps the kinetic parameters (names, values and limits), kinetic reactions and the ODEs that describe the current problem dynamics.

The kinetic reactions and the ODEs are defined separately, allowing any type of kinetic equations to be defined for a given set of ODEs. The user can apply constraints to limit or impose a condition when a value of a state variable or kinetic reaction is exceeded. The kinetics functions can be implemented using any of the control flow statements in Java, demanding some knowledge of the programming language, but allowing a greater flexibility.
The dynamical model describing the state variables behavior along time is described by a set of ODEs (see the case study). There are only two restrictions in the definition of the model: it is necessary to associate a substrate feeding rate parameter and a dilution rate factor has to be associated to all differential equations, with the exception of the equation describing volume/ weight variations.
Currently, the ODEs and kinetics have to be written in the Java language. The definition of a new model requires the implementation of two classes: one for the Process and the other for the Function. The structure of these classes is always the same, since they are based on a common interface. After the compilation of a model, the different data values associated with it are considered as default data and cannot be modified. Nevertheless, new instances can be created with different sets of values for different parameters. Indeed, when a Project is created, new sets of initial values for state variables, model parameters and feeding profiles can be defined and kept for future use.

Simulation

Regarding the process simulation, the user has the ability to test various combinations of the initial values for state variables, parameters and experimental or hypothetical feeding trajectories along time. Furthermore, it is possible to perform simulations with feed trajectories obtained from optimization. Likewise, after executing the estimation of model parameters, the results are immediately accessible and can be used to perform a simulation. The simulation results can also be compared with experimental data. The results are displayed via graphs, where each state variable or kinetic rate can be visualized separately. These figures can be exported as JPEG files. Simulations are performed by running a numerical integration process, using a linearly implicit-explicit Runge-Kutta scheme or a constant Runge-Kutta scheme, included in OdeToJava (Ascher et al. 1997).

Optimization

Three types of operations can be performed: the optimization of a simple feeding trajectory, of the feeding trajectory plus initial conditions or of the feeding trajectory plus final time (Rocha et al, 2004). In the first case, the ideal amount of substrate to be fed into the reactor per time unit along time is determined; the second scenario allows determining the best initial concentrations for each selected state variable, while in the third case the optimal duration of the fermentation is also provided.
The minimum and maximum pump limits can be defined by the user and these values are used as constraints on the optimization operations. Some preferences related with the algorithms can be modified by the user, such as the number of iterations, the population size, the discretization step and an interpolation factor. This factor is used to reduce the solution size, so that feeding values are defined only at certain equally spaced points. A report on the optimization operations performed can be generated, describing the conditions that were used and the results obtained.

Parameter estimation

To perform the estimation of parameters, a simple GUI is available, where the various estimation options are easily understandable. It is possible to fix certain parameters or to assert that certain state variables should be ignored during the estimation (this is important because if a state variable has null values over time, the objective function is affected, causing a numerical error). The results are presented in graphs or tables and both can be saved to files. As with feed optimization, a report can be generated. The fitting is performed by minimizing a total cost function that represents the adjustment between experimental and simulated data:

$$Total\ Cost = \sum_{i=1}^{n}\left(\frac{1}{N_p}\sum_{j=1}^{p}\left(\frac{\xi_{sim,ij}-\xi_{\exp,ij}}{\overline{\xi}_{\exp,ij}}\right)^2\right) \quad (1)$$

where $\xi_{sim,ij}$ represents the simulated data and $\xi_{\exp,ij}$ the experimental data for the state variable ξ (n is the number of state variables) for every point (p is the total of data points). The difference is divided by an average value $\overline{\xi}_{\exp,ij}$ with the purpose of giving the same importance to all state variables.

IMPLEMENTATION ISSUES

OptFerm is built in a modular way, using a component-based approach to software development. AIBench, a general purpose Java application framework for scientific software development, was used to manage the data objects and execute the operations, also making the linkage with the graphical interface. All information related with AIBench can be found in http://www.aibench.org/.
AIBench is a MVC (Model-View-Control) based Java application developed by the University of Vigo, with the collaboration of the authors. It uses a plug-in engine, which provides the capability to load or unload operations, allowing to create applications based in software modules. All applications developed with AIBench are structured through two main concepts: *datatypes*, defining data structures used in the application and *operations* describing functions receiving input objects and creating output objects.
To implement OptFerm, it was necessary to define the corresponding datatypes and operations. A general schema of OptFerm structure is shown in Figure 1.
A *datatype* is a Java class that specifies the internal representation of an object, in which simple data or complex data (other datatypes) are incorporated. It may be considered

as a container. They can be used or created during the various operations.

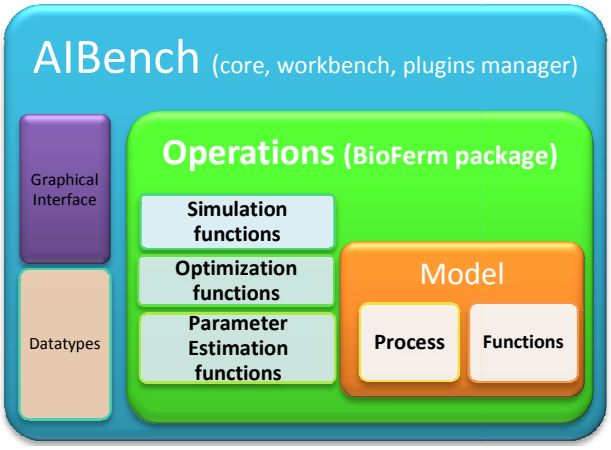

Figure 1: The general internal structure of *OptFerm*

In OptFerm, the datatypes were structured as (Figure 2):

- **Project** – it is the basic datatype; when a Project is created, each of the objects shown in Figure 2 is instantiated. A Project is directly related to a model (it has to contain one Model object and cannot contain more than a single one). A project has a list of Simulation, Optimization and Parameter Estimation results. These lists are extended, during the execution of each operation.
- **Model** – Within each Model there are four different datatypes, as shown in Figure 2, namely: State Variables, Kinetic Parameters, Feed Data and Experimental Data. These datatypes are of type List, in which a new set of initial values for the state variables, parameters, feed profiles or an experimental dataset can be added to the list. Consequently, different combinations of state variables, feed profiles and kinetic parameters can be used in the simulation, optimization and parameter estimation operations, without the need to change the internal structure of the model.
- **Simulation, Optimization and Estimation Results** – these are datatypes of type List. After the execution of each of these operations, a new object is created containing the results. The conditions that were used in these operations are saved, such as state variables, model parameters, feed profiles and experimental data sets.

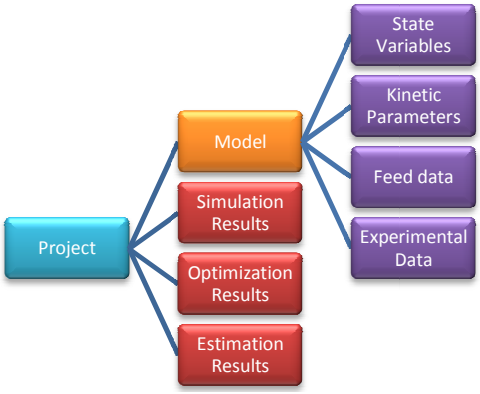

Figure 2: Structure of the Datatypes within a Project

All datatypes are organized in a *Clipboard* and presented to the user as a tree. The data contained inside the datatypes can be accessed through viewers, graphical interfaces where data is presented in tables, graphics or other suitable means.

In terms of the source code organization, a main library gathers the various packages with the simulation, optimization and estimation functions and a description of the models. A module containing specific optimization routines were created for feed optimization and related tasks. This module uses JECoLi (Java Evolutionary Computation Library; http://darwin.di.uminho.pt/jecoli) that contains generic optimization routines based on metaheuristic search algorithms. Some adaptations had to be made to adapt these algorithms to support feed optimizations, as explained in Rocha et al (2004) and Mendes et al (2006). Three algorithms belonging to the main groups of Evolutionary Algorithms, Differential Evolution and Simulated Annealing are used to perform optimizations.

A package for kinetic parameter estimation was developed, using the same optimization routines. Some modifications were made to enable the user to perform estimations without needing to modify the internal structure of the models. Functions to import/ export data were also implemented.

CASE STUDY

The case study is related to production of ethanol by *Saccharomyces cerevisiae* , described by Chen and Huang (1990). The purpose is to explain in a descriptive way the most important features of OptFerm and not to make any study of the used model. Due to space constraints only Simulation and Optimization operations are considered.

The model represents a fed-batch bioreactor system and encompasses the following equations (Chen e Hwang 1990):

$$\frac{dx_1}{dt} = g_1 x_1 - u\frac{x_1}{x_4} \quad (2)$$

$$\frac{dx_2}{dt} = -10 g_1 x_1 + u\frac{150 - x_2}{x_4} \quad (3)$$

$$\frac{dx_3}{dt} = g_2 x_1 - u\frac{x_3}{x_4} \quad (4)$$

$$\frac{dx_4}{dt} = u \quad (5)$$

where x_1, x_2 and x_3 are the cell mass, substrate and ethanol concentrations (g/L), x_4 the volume of the reactor (L) and u the feeding rate (L/h). Kinetic variables are given by:

$$g_1 = \frac{0.408}{1+\frac{x_3}{16}}\frac{x_2}{0.22+x_2} \quad (6)$$

$$g_2 = \frac{1}{1+\frac{x_3}{71.5}}\frac{x_2}{0.44+x_2} \quad (7)$$

The objective function was set to obtain a maximum of x_3 when the maximum of reactor capacity (x_4) is reached:

$$prod = x_3(T_f)x_4(T_f) \quad (8)$$

where T_f is the final time.

Model edition

The first step was to define the *Process* Java class and the *Function* Java class. The ODEs (equations 2 to 5) are converted into the equations presented in Figure 3. This represents a function that receives the present time value and an array of state variables calculated in the previous iteration. Next, it calls the *updateKineticCoefs(t)* method to calculate new values for the kinetic variables at time *t*, and then it calculates and returns an array containing the new values for the state variables at time *t*.

```
public double[] f(double t, double[] x)
{
    double[] xp = new double[x.length];
    updateKineticCoefs(t);
    kinetics(x[1], x[2]); // X2, X3
    double u = feed(t);

    xp[0]= kCoefs[0]*x[0] - u*(x[0]/ x[3]);
    xp[1]= -kCoefs[0]/ modelPars[0] * x[0] +
           u /x[3] * (modelPars[1]-x[1]);
    xp[2]= kCoefs[1]*x[0] - u * ( x[2]/ x[3] );
    xp[3]= u;

    return(xp);
}
```

Figure 3: How ODEs are defined in the Function Java class

The kinetic equations 6 and 7 have to be converted to the Java language as shown in Figure 4 . The variables g_1 and g_2 at each iteration are saved in an array *kCoefs*, and these values are used later in the ODEs. The *modelPars* are the kinetic parameters defined in the *Function* java class as well.

```
public void kinetics (double S, double P)
{
    kCoefs[0] = (modelPars[2]/(1.0 +
        (P/modelPars[3]))) * (S/(modelPars[4]+S));
    kCoefs[1] = (modelPars[5] / (1.0 + (P/modelPars[6])))
        * (S/(modelPars[7]+S));
}
```

Figure 4: How kinetic variables are defined in Java

An objective function must be defined, describing the purpose of the optimization. The aim was to obtain the maximum of ethanol (x_3) and equation 8 was used, being defined in the *Process* class as the *productivity* method:

```
public double productivity (double tf)
{
   double prod = u[2][endPoint] * u[3][endPoint];
   return prod;
}
```

Figure 5: The objective function in the Java language

OptFerm ClipBoard

After defining the process and function classes, these are compiled and are ready to use in OptFerm. A new project is created and all initial Datatypes are displayed (Figure 6). They are presented as a tree structure, and the data contained can be viewed by simply clicking over the datatypes. All functionalities are displayed in the menus.
Different sets of initial values for state variables, kinetic parameters, feeding profiles and experimental data can be created and added to the clipboard (Figure 7). Data can also be removed from the clipboard. The internal data of these new sets can be modified when necessary, and the user can save or load previously saved data.

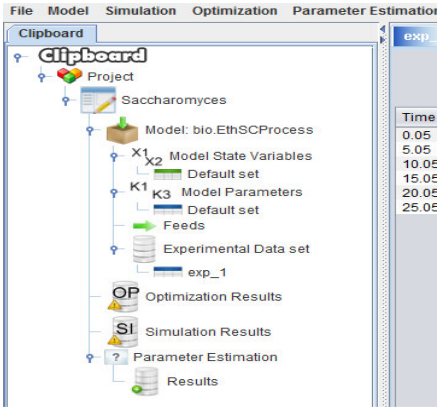

Figure 6: Example of OptFerm Clipboard

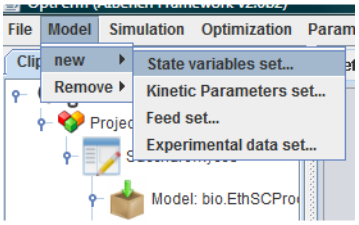

Figure 7: Menus and sub-menus of the OptFerm toolbox: example on how a new set of state variables can be created

Simulation

An interface is presented to the user with all options to perform simulations (Figure 8). A *Project*, the initial values of state variables and kinetic parameters have to be selected. It is possible to select between feeding profiles that had been defined by the user and the ones resulting from optimization procedures. After performing a parameter estimation, the model parameters are also available to be used.

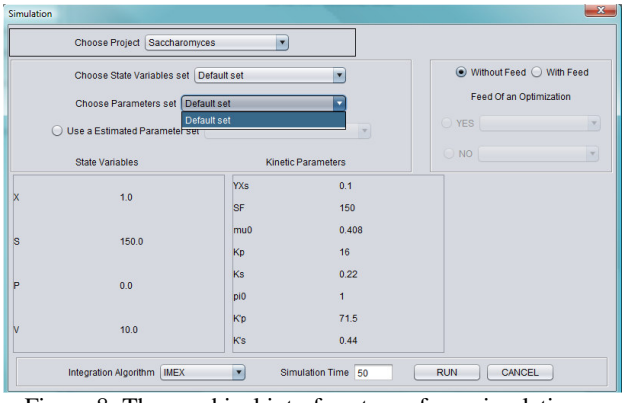

Figure 8: The graphical interface to perform simulations

After performing a simulation, the results are displayed in a graph (Figure 9). The state variables or kinetic rates can be visualized. The right panel displays the parameters used.

Optimization

To perform an optimization, a panel is presented (Figure 10). On this panel, several options can be selected and the available sets of initial values for state variables and model parameters are displayed.

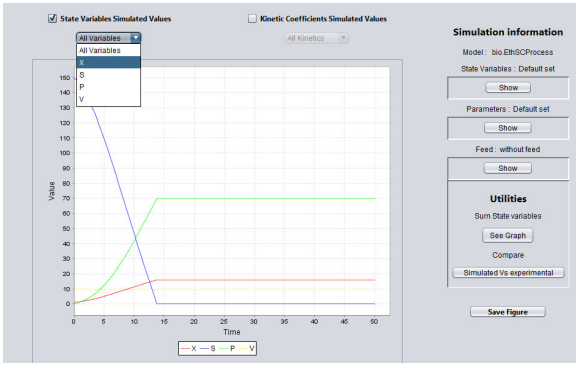

Figure 9: How simulation results are presented to the user

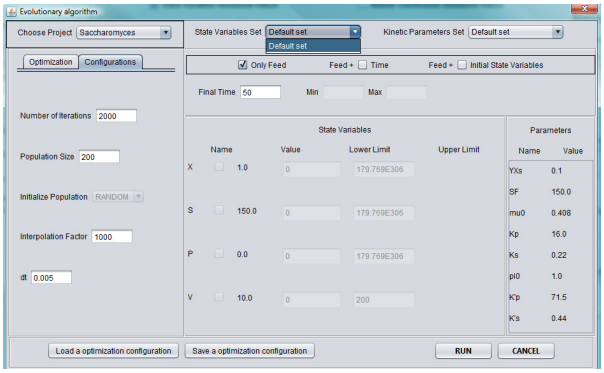

Figure 10: Graphical interface to execute optimization tasks.

After performing the optimization, the results are displayed as shown in Figure 11. A graph and a table are used to show the optimized feed trajectory. Information about the objective function is displayed, as well as the best value. The user can also see the parameters used in the optimization.

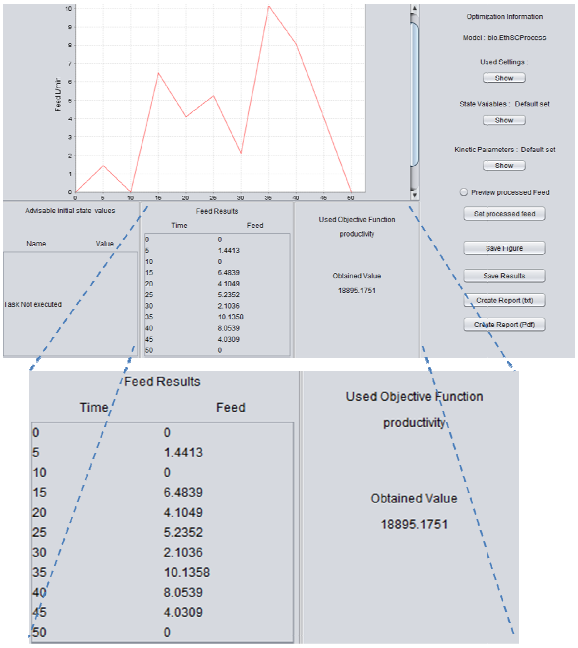

Figure 11: Results of the performed optimizations

CONCLUSIONS AND FURTHER WORK

The aim of the *OptFerm* software was not to replace bioprocess optimization by trial-and-error approach, but to reduce the number of trials that are necessary to achieve the best results. So, with this tool the user is able to analyze the robustness of a fed-batch model, compare simulated data with experimental data, determine unknown parameters and optimize a feeding profile to be fed into a bioreactor.

The current software version has a major limitation: the absence of a graphical interface to visualize and edit models. This feature will be available in a future version. The user can still create the corresponding Java classes describing the model by differential equations and kinetic reactions. In future versions, functions for exporting/importing models in SBML (System Biology Markup Language) format will be implemented. Because OptFerm is implemented inside AIBench framework that has a plug-in concept, new functionalities or algorithms can be easily integrated.

REFERENCES

Ascher,U.M. et al. (1997) Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.*, 25, 151–167.

Benjamin,K.K. et al. (2008) Genetic Algorithms Using for a Batch Fermentation Process Identification. *Journal of Applied Sciences*, 8, 2272–2278.

Chen,C. e Hwang,C. (1990) Optimal Control Computation for Differential-algebraic Process Systems with General Constraints. *Chemical Engin. Communications*, 97, 9.

Kawohl,M. et al. (2007) Model based estimation and optimal control of fed-batch fermentation processes for the production of antibiotics. *Chemical Engineering and Processing: Process Intensification*, 46, 1223–1241.

Mendes, R., Rocha, M., Rocha, I., Ferreira, E.C. (2006) A Comparison of Algorithms for the Optimization of Fermentation Processes. Proc. IEEE Conf. Evolutionary Computation, pp. 7371-7378, Vancouver, Canada, 2006.

Mendes,R. et al. (2008) Differential Evolution for the Offline and Online Optimization of Fed-Batch Fermentation Processes. In UK Chakraborty (ed.), *Advances in Differential Evolution*, ch.13, pp.299-318, Springer, 2008.

Pettinen,A. et al. (2005) Simulation tools for biochemical networks: evaluation of performance and usability. *Bioinformatics*, 21, 357–363.

Rocha, M., J. Neves, I. Rocha, E.C. Ferreira (2004) Evolutionary Algorithms for Optimal Control in Fed-batch Fermentation Processes, In LNCS 3005, pp.84-93, Coimbra, Portugal, Springer, 2004.

Rocha,M., Mendes, R., Maia, P. Pinto, J.P., Rocha, I., Ferreira, E.C. (2007) Evaluating Simulated Annealing Algorithms in the Optimization of Bacterial Strains. *Proc. EPIA 2007,* pp. 473-484, Springer, 2007.

Tzoneva,R. (2006) Method for Optimal Control Calculation of a Fed-batch Fermentation Process. *Proc. Mediterr. Conf. Control and Automation, 2006,* pp. 1–6.

Zhang,H. (2008) Optimal control of a fed-batch yeast fermentation process based on Least Square Support Vector Machine. *Intern. J. Engin. Systems Modelling Simulation*,1:63–68.