Engineering Letters, 14:2, EL_14_2_6 (Advance online publication: 16 May 2007)

Genetic Programming for Symbolic Regression of Chemical Process Systems

B. V. Babu*, *Member, IAENG*, and S. Karthik
Chemical Engineering Department, Birla Institute of Technology and Science (BITS), Pilani-333 031.

***Abstract*—The novel evolutionary artificial intelligence formalism namely, genetic programming (GP) a branch of genetic algorithms is utilized to develop mathematical models based on input-output data, instead of conventional regression and neural network modeling techniques which are commonly used for this purpose. This paper summarizes the available MATLAB toolboxes and their features. Glucose to gluconic acid batch bioprocess has been modeled using both GPLAB and hybrid approach of GP and Orthogonal Least Square method (GP OLS). GP OLS which is capable of pruning of trees has generated parsimonious expressions simpler to GPLAB, with high fitness values and low mean square error which is an indicative of the good prediction accuracy. The capability of GP OLS to generate non-linear input-output dynamic systems has been tested using an example of fed-batch bioreactor. The simulation and GP model prediction results indicate GP OLS is an efficient and fast method for predicting the order and structure for non-linear input and output model.**

***Index Terms*—About four key words or phrases in alphabetical order, separated by commas.**

I. Introduction

The increasing emphasis on product 'quality', economic process performance and environmental issues in the chemical and allied industries is placing significant demands on existing operational procedures. Enhanced process performance generally requires increased process knowledge, with mathematical models being the most common means of representing this knowledge. While it may be possible to develop a model using a detailed knowledge of the physics and chemistry of a system, there are a number of drawbacks to this approach. Industrial process systems are often extremely complex and non-linear in nature, thus it may take a considerable amount of time and effort to develop a realistic model [1,2]. Moreover, in many instances simplifying assumptions have to be made in order to provide a tractable solution. A first-principles model will, therefore, often be costly to develop and may be subject to inaccuracies.

* Corresponding Author
Assistant Dean-ESD &
Professor & Head-Chemical Engineering Department
Ph: +91-1596-245073 Ext. 205
Fax: +91-1596-244183
E-mail: bvbabu@bits-pilani.ac.in
http://discovery.bits-pilani.ac.in/discipline/chemical/BVb/

However, if an accurate process model were available, then many of the benefits of improved process operability would be achievable. The current trend within the process industries is to use data based modeling techniques to develop accurate, cost-effective input-output process descriptions [3]. The popular techniques may be divided into two categories. The first are based on the use of various statistical techniques and regression analysis, while the second involves the use of artificial neural networks.

The data-driven identification of these models involves the following tasks:

1) *Structure selection*.
2) *Input sequence design*.
3) *Noise modeling*.
4) *Parameter estimation*.
5) *Model validation*.

Genetic programming (GP), which is an evolutionary approach, is used to develop nonlinear models of chemical process systems using only plant input-output data.

Genetic programming is different from all other approaches to artificial intelligence, machine learning, neural networks, adaptive systems, reinforcement learning, or automated logic in all (or most) of the following seven ways (www.genetic-programming.com/sevendiffs.html):

1) Representation: Genetic programming overtly conducts it search for a solution to the given problem in program space.
2) Role of point-to-point transformations in the search: Genetic programming does not conduct its search by transforming a single point in the search space into another single point, but instead transforms a set of points into another set of points.
3) Role of hill climbing in the search: Genetic programming does not rely exclusively on greedy hill climbing to conduct its search, but instead allocates a certain number of trials, in a principled way, to choices that are known to be inferior.
4) Role of determinism in the search: Genetic programming conducts its search probabilistically.
5) Role of an explicit knowledge base: None.
6) Role of formal logic in the search: None.
7) Underpinnings of the technique: Biologically inspired.

A. *Modeling of glucose to gluconic acid bioprocess*

A new batch fermentation technique proposed for the production of gluconic acid from glucose wherein *A. niger* immobilized on a support matrix consisting of a cellulosic fabric has resulted in higher yields [4]. The improved overall productivity from this technique is primarily due to the enhanced interaction between the dissolved oxygen and the fungal mycelia. Enhancement in the said interaction is effected via a continuous substrate dripping mechanism and not by the mechanical agitation as used in the free-cell fermentation. The main objective in this dissertation is to develop a mathematical model of the new glucose to gluconic acid batch fermentation process. For developing the fermentor model, experimental data incorporating the effects of the substrate (glucose) and biomass concentrations, and the dissolved oxygen content, have been used.

1. Experimental Details

For developing the GP based model for the glucose to gluconic acid bioprocess, experimental input-output data from the fermenter were used. In these experiments, the gluconic acid producing strain *Aspergillus niger* NCIM 545 had been utilized.

1.1 Fermentation Medium for Immobilized Mycelia

Anhydrous purified glucose (100 g), MgSO4â7H2O (0.035g), KH2PO4 (0.05 g), and 0.1 g of (NH4)2HPO4 were dissolved in 1 L of water. The pH of this medium was adjusted to 6.0 using 1MH2SO4. A woven cellulosic fabric support (69 _ 8.5 _ 0.6 cm) with void volume of approximately 140 mL was sterilized at 15 psi for 60 min.

1.2 Submerged Fermentation

Submerged fermentation utilizing the immobilized culture was carried out in a modified locally fabricated batch fermenter. In the fermenter, the matrix with fully grown *A. niger* was folded in a spiral shape. For preventing mycelial recirculation, the upper end of the fixed bed was closed by the filter mesh. The batch reactor was drained after the substrate reached its lowest concentration.

1.3 Maintaining Oxygen Partial Pressure

A constant flow of air was used to maintain the oxygen partial pressure and a Dissolved Oxygen (DO) probe (Ingold, 170-ppm type DO amplifier) was used for measuring the dissolved oxygen concentration.

1.4 Glucose and Gluconic Acid Analyses

Feed and the unconverted glucose were analyzed by the dinitrosalicyclic acid method [5], and the gluconic acid concentration in the bioreactor was measured by titrating against 6 N NaOH.

B. Problems involved in modeling of glucose to gluconic acid bioprocess

The glucose to gluconic acid bioconversion using *A. niger* immobilized on the cellulosic micro fibrils involves complicated reaction and mass transfer phenomena. Development of a phenomenological (“first principles”) process model has therefore become a difficult task since the physicochemical phenomena underlying the bioconversion and the associated kinetic and transport mechanisms are not well-understood. Also, it has been observed that the process dynamics is nonlinear [5]. This has made the modeling task even more complex. In view of these difficulties, a novel artificial intelligence based paradigm, namely, genetic programming (GP) [6] has been employed here for modeling the fermenter. The principal advantage of the GP-formalism is that it automatically arrives at an empirical closed-form mathematical model relating process inputs and outputs exclusively from the historic process input-output data. Consequently, the detailed knowledge of the process phenomenology (reaction kinetics and mass transfer mechanisms) is not necessary in the GP-based process modeling.

The main objective in this paper is to develop a mathematical model of the new glucose to gluconic acid batch fermentation process. For developing the fermenter model, experimental data incorporating the effects of the substrate (glucose) and biomass concentrations, and the dissolved oxygen content, have been used.

C. Dynamic modeling of fed-batch bioreactor

This example considers a fed batch reactor that has been studied by Park and Ramirez [7]. The system is described by the following differential equations.

$$\frac{dx_1}{dt} = g_1(x_2 - x_1) - (u/x_5)x_1 \qquad (1)$$

$$\frac{dx_2}{dt} = g_2 x_3 - (u/x_5)x_2 \qquad (2)$$

$$\frac{dx_3}{dt} = g_3 x_3 - (u/x_5)x_3 \qquad (3)$$

$$\frac{dx_4}{dt} = -7.3 g_3 x_3 + (u/x_5)(20 - x_4) \qquad (4)$$

$$\frac{dx_5}{dt} = u \qquad (5)$$

where

$$g_3 = 21.87 x_4 /((x_4 + 0.4)(x_4 + 62.5))$$

$$g_2 = x_4 e^{-5x_4} /(0.1 + x_4)$$

$$g_1 = 4.75 g_3 /(0.12 + g_3)$$

$X(0)=[0\ 0\ 1\ 5\ 1]^T$
and the constraint is
$0 \le u \le 10$

The model was simulated to obtain the feed rate, which in turn was used to model the fed-batch bioreactor.

D. Objective and Scope of Study

The most well known and simple to apply statistical techniques assume that any relationships between input and output variables are linear and that the data itself is normally distributed. Unfortunately, industrial systems are normally highly non-linear and the data obtained from such processes generally do not conform to normal distributions. Nevertheless, numerous methods can be used to implement a systematic data analysis methodology and can help to establish the basic characteristics of the process.

A novel approach which offers a useful alternative to these established methodologies is genetic programming (GP) which has been used to develop nonlinear models of chemical process systems using only plant input-output data. This methodology not only performs symbolic regression to determine the appropriate structure, complexity of the required model but can also explain the physics of the system.

The work presented in this dissertation considers fulfilling the following objectives:

1) Application of GP for regression: A study has been made on genetic programming (GP). Different GP toolboxes have been utilized and their merits and demerits have been explored.
2) Modeling of Bioreactor using GP: GP based formalism for the glucose to gluconic acid bioprocess has been developed using the experimental input – output data from the fermentor.

Dynamic modeling of chemical system: GP based dynamic model for a fed – batch reactor, using pre-validated and simulated data.

II. Genetic Programming (GP)

The performance of an individual organism in its environment determines the likelihood of it passing on its genetic material to future generations. This basic biological principle is known as Darwinian survival of the fittest, and has inspired a class of algorithms known as Genetic Algorithms (GA's) [8,9,10]. GA's, attempt to find the best solution to a problem by mimicking the process of evolution in nature. Thus, a typical algorithm will 'breed' a population of individuals that represent possible solutions to a particular problem.

GA's are not appropriate for symbolic regression problems where the structure and parameters of a model are to be determined simultaneously. This is because GA's generally use fixed length binary strings to code potential solutions to a problem. Clearly this is unsuitable for symbolic regression, where the model structure is allowed to vary during evolution. However, GP is a closely related approach that does lend itself to the implementation of symbolic regression.

GP differs from GA's by utilizing the following:

1) Tree structured variable length chromosomes (rather than chromosomes of fixed length and structure).
2) Chromosomes coded in a problem specific fashion (that can usually be executed in their current form) rather than binary strings.
3) Genetic operators that preserve the syntax of the tree Structured chromosomes during 'reproduction'.

A. *Preparatory Steps for Genetic Programming*

Genetic programming starts from a high-level statement of the requirements of a problem and attempts to produce a computer program that solves the problem. The human user communicates the high-level statement of the problem to the genetic programming system by performing certain well-defined preparatory steps.

The five major preparatory steps for the basic version of genetic programming require the human user to specify are:

1) The set of terminals (e.g., the independent variables of the problem, zero-argument functions, and random constants) for each branch of the to-be-evolved program,
2) The set of primitive functions for each branch of the to-be-evolved program,
3) The fitness measure (for explicitly or implicitly measuring the fitness of individuals in the population),
4) Certain parameters for controlling the run, and
5) The termination criterion and method for designating the result of the run.

The figure below shows the five major preparatory steps for the basic version of genetic programming. The preparatory steps (shown at the top of the Fig. 1) are the human-supplied input to the genetic programming system. The computer program (shown at the bottom) is the output of the genetic programming system.

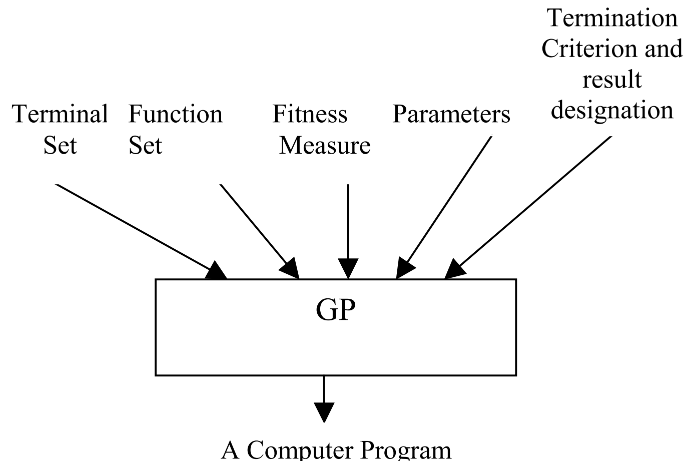

Fig. 1 Five major steps for GP [11]

The first two preparatory steps specify the ingredients that are available to create the computer programs. A run of genetic programming is a competitive search among a diverse population of programs composed of the available functions and terminals

1. *Function set and Terminal*

The identification of the function set and terminal set for a particular problem (or category of problems) is usually a straightforward process. For some problems, the function set may consist of merely the arithmetic functions of addition, subtraction, multiplication, and division as well as a conditional branching operator. The terminal set may consist of the program's external inputs (independent variables) and numerical constants. This function set and terminal set is useful for a wide variety of problems (and corresponds to the basic operations found in virtually every general-purpose digital computer).

For many other problems, the ingredients include specialized functions and terminals. For example, if the goal is to get genetic programming to automatically program a robot to mop the entire floor of an obstacle-laden room, the human user must tell genetic programming what the robot is capable of doing. For example, the robot may be capable of executing functions such as moving, turning, and swishing the mop.

If the goal is the automatic creation of a controller, the function set may consist of signal-processing functions that operates on time-domain signals, including integrators, differentiators, leads, lags, gains, adders, subtractors, and the like. The terminal set may consist of signals such as the reference signal and plant output. Once the human user has identified the primitive ingredients for a problem of controller synthesis, the same function set and terminal set can be used to automatically synthesize a wide variety of different controllers.

2. *Fitness measure*

The third preparatory step concerns the fitness measure for the problem. The fitness measure specifies what needs to be done. The fitness measure is the primary mechanism for communicating the high-level statement of the problem's requirements to the genetic programming system. The first two preparatory steps define the search space whereas the fitness measure implicitly specifies the search's desired goal.

3. *Control Parameters*

The fourth and fifth preparatory steps are administrative. The fourth preparatory step entails specifying the control parameters for the run. The most important control parameter is the population size. In practice, the user may choose a population size that will produce a reasonably large number of generations in the amount of computer time we are willing to devote to a problem (as opposed to, say, analytically choosing the population size by somehow analyzing a problem's fitness landscape). Other control parameters include the probabilities of performing the genetic operations, the maximum size for programs, and other details of the run.

4. *Termination*

The fifth preparatory step consists of specifying the termination criterion and the method of designating the result of the run. The termination criterion may include a maximum number of generations to be run as well as a problem-specific success predicate. In practice, one may manually monitor and manually terminate the run when the values of fitness for numerous successive best-of-generation individuals appear to have reached a plateau. The single best-so-far individual is then harvested and designated as the result of the run.

5. *Running GP*

After the human user has performed the preparatory steps for a problem, the run of genetic programming can be launched. Once the run is launched, a series of well-defined, problem-independent execution steps are executed.

B. *Executional Steps in GP*

Genetic programming typically starts with a population of randomly generated computer programs composed of the available programmatic ingredients. Genetic programming iteratively transforms a population of computer programs into a new generation of the population by applying analogs of naturally occurring genetic operations. These operations are applied to individual(s) selected from the population. The individuals are probabilistically selected to participate in the genetic operations based on their fitness (as measured by the fitness measure provided by the human user in the third preparatory step). The iterative transformation of the population is executed inside the main generational loop of the run of genetic programming.

GP uses four steps to solve problems:

1 Generate an initial population of random compositions of the functions and terminals of the problem (Computer programs).

2 Execute each program in the population and assign it a fitness value according to how well it solves the problem.

3 Create a new population of computer programs:

  1.1. Copy the best existing programs.

  1.2. Create new computer programs by mutation.

  1.3. Create new computer programs by crossover (sexual reproduction).

4 The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming.

III. Genetic Programming Toolboxes in MATLAB

MATLAB is a widely used programming environment available for a large number of computer platforms. Its programming language is simple and easy to learn, yet fast and powerful in mathematical calculus. Furthermore, its extensive and straightforward data visualization tools make it a very appealing programming environment. Toolboxes are collections of optimized, application specific functions, which extend the MATLAB environment and provide a solid foundation on which to build.

A. *GPLAB*

GPLAB is a genetic programming toolbox for MATLAB. Versatile, generalist and easily extendable, it can be used by all types of users, from the layman to the advanced researcher.

1. *Operational Structure*

The architecture of GPLAB follows a highly modular and parameterized structure, which different users may use at various levels of depth and insight [12].

1.1. Main modules

The operational structure of GPLAB has 3 main operation modules, namely SETVARS, GENPOP and GENERATION, and each represents an interaction point with the user.

1.1.1. GENPOP

This module generates the initial population (INIT POP) and calculates its fitness (FITNESS). The individuals in GPLAB are tree structures initialized with one of three available initialization methods – Full, Grow, Ramped Half-and-Half. The functions available to build the trees include some protected functions, plus any MATLAB function that verify closure. The terminals include a random number generator and all the variables necessary, created in runtime. Fitness is, by default, the sum of absolute differences between the obtained and expected results in all fitness cases. The lower the fitness value, the better the individual. This is the standard for symbolic regression problems ("regfitness").

GEN POP is called by the user. It starts by requesting some parameter initializations to SET VARS, and finishes by passing the execution to GENERATION. If the user only requests the creation of the initial generation, GENERATION is not used.

1.1.2. Generation

This module creates a new generation of individuals by applying the genetic operators to the previous population (OPERATORS). Standard tree crossover and tree mutation are the two genetic operators available as plug and play functions. They must have a pool of parents to choose from, created by a SAMPLING method, which may or may not base its choice on the EXPECTED number of offspring of each individual. Four sampling methods (Roulette, SUS, Tournament, Lexicographic Parsimony Pressure Tournament) and three methods for calculating the expected number of offspring (Absolute, Rank85, Rank89) are available as plug and play functions, and any combination of the two can be used. The genetic operators create new individuals until a new population is filled, a number determined by the generation gap.

Calculating fitness is followed by the SURVIVAL module, where the individuals that enter the new generation are chosen according to the elitism level parameter. The GENERATION module repeats itself until the stop condition is fulfilled, or when the maximum generation is reached. Several stop conditions can be used simultaneously. This module can be called either by the user or by GEN POP.

1.1.3. Set Vars

This module either initializes the parameters with the default values or updates them with the user settings. Besides the parameters directly related to the execution of the algorithm, other parameters affect the output of its results. SET VARS can be called either by the user or by a request for parameter initialization from GEN POP.

1.2. Parameters in Genetic Programming Toolbox

1.2.1. Tree Initialization

The initial population of trees, created in runtime in the beginning of a GPLAB run, is done by choosing random functions and terminals from the respective sets. There are three different methods available in GPLAB, used in the plug and play fashion, and each of them uses either the standard procedure based on depth, or the new variation based on size, i.e., number of nodes, depending on the parameter depth nodes ('1' for depth, '2' for size):

'fullinit' - This is the Full method. In the standard procedure, the new tree receives non terminal (internal) nodes until the initial tree depth is reached - the last depth level is limited to terminal nodes. As a result, trees initialized with this method will be perfectly balanced with all the branches of the same length. If size is used instead of depth, internal nodes are chosen until the size of the new tree is close to the specified size, and only then terminals are chosen. Unlike the standard procedure, the size variation may not be able to create trees with the exact size specified, but only close (never exceeding).

'growinit' - This is the Grow method. In the standard procedure, each new node is randomly chosen between terminals and non terminals, except nodes at the initial tree depth level, which must be terminals. Tree created with this method may be very unbalanced, with some branches much longer than others.

'rampedinit' - This is the Ramped Half-and-Half method. In the standard procedure, an equal number of individuals are initialized for each depth between 2 and the initial tree depth value. For each depth level considered, half of the individuals are initialized using the Full method, and the other half using the Grow method. The population of trees resulting from this initialization method is very diverse, with balanced and unbalanced trees of several different depths.

1.2.2. Functions

As any genetic programming algorithm, GPLAB needs functions and terminals to create the population, in this case the parse trees that represent individual functions. GPLAB can use any MATLAB function that verifies closure, plus some protected and logical functions and the if-then-else statement, also available as part of the toolbox. The user indicates which functions the algorithm should use by setting the parameter variable functions. Table 1 contains information on the available toolbox functions. All the functions described in Table 1 are used in the plug and play fashion.

Table 1 Protected and logical function using Matlab

Protected Function	MATLAB Function	Input Arguments	Output Argument
Division	mydivide	a,b	a (if b=0) a/b (otherwise)
Square root	mysqrt	a	0 (if a<=0) Sqrt(a) (otherwise)
Power	mypower	a,b	a^b (if a^b is a valid non-complex number) 0 (otherwise)
Natural logarithm	mylog	a	O (if a = 0) log(abs(a)) (otherwise)
Base 2 Logarithm	mylog2	a	O (if a = 0) log2(abs(a)) (otherwise)
Base 10 Logarithm	mylog10	a	O (if a = 0) log10(abs(a)) (otherwise)
If – then – else Statement	myif	a,b,c	eval (c) (if eval (a) = 0) eval (b) otherwise
Negation of AND	Nand	a,b	not(and(a,b))

The advanced users who want to build and us their own functions only have to implement them as MATLAB functions (and make sure the input arguments can be either scalars or vectors – see MATLAB user's manual) and declare them using one of the toolbox functions

```
params=setfunctions(params,'func1',2,'func2',1);
params=addfunctions(params,'func1',2,'func2',1);
```

setfunctions defines the set of available functions as containing functions 'func1' and 'func2', replacing any other functions previously declared. 'func1' has arity 2 - it needs two input arguments; 'func2' has arity 1. Any number of functions can be declared at one time, by adding more arguments to *setfunctions*. *addfunction*s accepts the same arguments but adds the declared functions to the already defined set, keeping the previously declared functions untouched. *setfunctions* and *addfunction*s are friendly substitutes to directly setting the parameter variable functions. The declaration of genetic operators is done similarly.

1.2.3. Terminals

GPLAB can use any constant as a terminal, plus a random number between 0 and 1, generated in runtime, as the function 'rand' with null arity. The declaration of terminals is done similarly to the declaration of functions, by using friendly substitutes to directly setting the parameter variable terminals. For example, to declare the constant '1' and the random number generator as members of the set of terminals.

Params = setterminals(params,'rand','1');

Unlike in setfunctions, there is no need to indicate the arity, which is always null. To add a new terminal to an already declared set of terminals

Params = addterminals(params,'new_terminal');

Any number of terminals can be declared or added at one time, by adding more input arguments.

Variables needed to evaluate the fitness cases are also part of the set of available terminals for the algorithm to work with, and these can only be generated (automatically) in the beginning of the run, according to the settings of the parameters numvars and autovars:

numvars=[] and autovars='0' - the parameter numvars is automatically filled with 0 and no variables are generated. This setting is appropriate for artificial ant problems.

numvars=[] and autovars='1' - the parameter numvars is automatically filled with the number of columns of the input data set and these many variables are generated. This setting is appropriate for symbolic regression and parity problems.

numvars=x - customized setting, where x is the number of variables generated, corresponding to the x first columns of the input data set.

1.2.4. Genetic Operators

GPLAB may use any number of genetic operators to create new individuals. A proportion of individuals, specified in parameter reproduction, may also be copied into the next generation without suffering the action of the operators. Tree crossover and tree mutation are the genetic operators provided by GPLAB, implemented as follows:

Crossover: In tree crossover, random nodes are chosen from both parent trees, and the respective branches are swapped creating two offspring. There is no bias towards choosing internal or terminal nodes as the crossing sites.

Mutation: In tree mutation, a random node is chosen from the parent tree and substituted by a new random tree created with the terminals and functions available. This new random tree is created with the Grow initialization method and obeys the size/depth restrictions imposed on the trees created for the initial generation. Although these are the only genetic operators provided, the addition of others is straightforward. Genetic operator is simply a MATLAB function used as a plug and play device to module OPERATOR, and the declaration of its existence to the algorithm is made similarly to the setting of functions and terminals, with one of the toolbox functions

Params = setoperators(params,'operator1',2,2,'operator2',2,1);
Params=addoperators(params,'operator1',2,2,'operator2',2,1);

Some of the tree manipulation functions available are:

maketree(level,functions,arities,exactlevel,depthnodes) – this function returns a new random tree no deeper/bigger than level, using the functions with respective arities. If exactlevel is true, the new tree will be initialized using the Full method; otherwise, it will be initialized using the Grow method. *depthnodes* indicates whether restrictions are to be applied in tree depth or tree size (number of nodes)

tree2str (tree) – returns the string that tree represents

findnode(tree,x) – returns the subtree of tree with root on ode number x. The nodes are numbered depth-first

swapnode(tree,x,node) – returns the result of swapping node number x in tree for node

tree2str(tree) – returns the translation of tree into a string

treelevel(tree) – returns the depth of tree

nodes(tree) – returns the number of nodes of tree

intronnodes(tree,params,data,state) – returns the number of introns of tree. Needs the variables params, data and state.

1.2.5. Selection for Reproduction

Genetic operators need parent individuals to produce their children. In GPLAB these parents are selected according to one of four sampling methods, as indicated in the parameter variable sampling:

'Roulette' - This method acts as if a roulette with random pointers is spun, and each individual owns a portion of the roulette that corresponds to its expected number of children.

'SUS' - This method also relies on the roulette, but the pointers are equally spaced.

'Tournament' - This method chooses each parent by randomly drawing a number of individuals from the population and selecting only the best of them.

'Lexictour' - This method implements lexicographic parsimony pressure. Like in 'tournament', a random number of individuals are chosen from the population and the best of them is chosen. The main difference is, if two individuals are equally fit, the shortest one (the tree with less nodes) is chosen as the best. This technique has shown to effectively control bloat in different types of problems.

1.2.6. Expected number of children

Some sampling procedures choose the parents based on their expected number of children, while others only need to know which are better than which. Likewise, the calculation of the expected number of children may use the actual fitness values or simply their rank in the population. The parameter variable expected determines with method is used for calculating the expected number of children for each individual. This calculation is performed only if the selection for reproduction so requires. Three different methods are available in GPLAB:

'Absolute' - the expected number of children for each individual is proportional to its absolute fitness value (it is equal to its normalized, or relative, fitness) [13].

'Rank85' - the expected number of children for each individual is based on its rank in the population [4].

'Rank89' - the expected number of children for each individual is based on its rank in the population and on the state of the algorithm (how far it is from the maximum allowed generation). The differentiation between individuals increases in later generations.

1.2.7. Measuring Fitness

'regfitness' - calculates, for each individual, the sum of the absolute difference between the expected output value and the value returned by the individual on all fitness cases. The best individuals are the ones that return values less different than the expected values - the ones with a lower fitness. This function should be used with the parameter lowerisbetter set to '1'. When regfitness is used, all the fitness values stored in the algorithm's variables are rounded to a certain number of decimal places, given by the parameter precision. This is meant to avoid rounding errors that affect the comparison of two different individuals who have the same fitness.

1.2.8. Survival

After producing gengap new individuals for the new population, GPLAB enters the SURVIVAL module where, from the current population plus all the new children, a number of individuals is chosen to form the new population. One of four elitism levels may be used, indicated in the parameter variable survival:

'replace' - The children replace the parent population completely, even if they are worse individuals than their parents. This option is not elitist at all.

'keepbest' - The best individual from both parents and children is kept for the new population, independently of being a parent or a child. The remaining places in the new population are occupied by children only. If not all children produced can be used in the new population, due to size constraints, the worst are discarded.

'halfelitism' - Half of the new population will be occupied by the best individuals chosen from both parents and children. The remaining places will be occupied by the best children still available.

'totalelitism' - The best individuals from both parents and children are chosen to fill the new population. The survival module is in fact elitist, even when the non elitism option is chosen. If GPLAB is operating in batch mode, the best children are always chosen, and the worst discarded.

1.2.9. Stop Conditions

GPLAB will run until the maximum generation indicated by the user is reached, or until a stop condition is reached. Stop conditions are defined by setting the parameter variable hits. One hit is a tuple [f d] where f is the percentage of fitness cases that must obey the stop condition and d is the definition of the stop condition itself, meaning that the result obtained by the best individual in the population must be no lower than the expected result minus d% (of the expected result) and no higher that the expected result plus d%. The default value of hits is [100 0], which means "stop if the best individual produces exact results in all fitness cases". [50 10] would mean "st op if the best individual produces results within minus or plus 10% of the expected results, in at least 50% of the fitness cases". Several stop conditions can be used, by adding rows to the hits variable. If the two previous stop conditions were to be used concurrently, hits should be set to [100 0; 50 10]. GPLAB tests each stop condition, starting with the first row, until one is satisfied or all have been tested. It is possible not to use any stop condition (hits= []), in which case GPLAB will only stop when reaching the maximum number of generations allowed.

B. Genetic Programming using Orthogonal Least Square Method

Genetic Programming (GP) is used to generate nonlinear input-output models of dynamical systems that are represented in a tree structure. The main idea is to apply Orthogonal Least Squares algorithm (OLS) to estimate the contribution of the branches of the tree to the accuracy of the model. This method results in more robust and interpretable models [13].

In practice, a model which gives good prediction performance on the training data may be over-parameterized and may contain unnecessary, complex terms. The penalty function handles this difficulty, because it decreases fitness values of trees that have complex terms. However, parameters of this penalty term are not easy to determine and the penalty function does not provide efficient solution for this difficulty. An efficient solution may be the elimination of complex and unnecessary terms from the model. For linear-in-parameters models it can be done by the Orthogonal Least Squares (OLS) algorithm.

1. Orthogonal Least Square Method Algorithm (OLS)

The great advantage of using linear-in-parameters models is that the Least Squares Method (LS) can be used for the identification of the model parameters, which is much less computationally demanding than other nonlinear optimization algorithms, since the optimal p = $[p_1, \ldots\ldots, p_M]^T$ parameter vector can be analytically calculated:

$$p = (F^{-1}F)^T Fy \tag{6}$$

where y = $[y(1),\ldots\ldots, y(N)]^T$ is the measured output vector, and the F regression matrix is:

$$F = \begin{pmatrix} F_1(x(1)) & F_M(x(1)) \\ \hline \\ \hline F_1(x(N)) & F_M(x(N)) \end{pmatrix} \tag{7}$$

In case most of process systems certain input and output interactions will be redundant and hence components in the ANOVA decomposition could be ignored, which can result in more parsimonious representations. The OLS algorithm is an effective algorithm to determine which terms are significant in a linear-in-parameters model. The OLS introduces the error reduction ratio (*err*) which is a measure of the decrease in the variance of output by a given term.

The compact matrix form corresponding to the linear-in-parameters model is y = Fp + e*;* where the **F** is the regression matrix, **p** is the parameter vector, **e** is the error vector. The OLS technique transforms the columns of the **F** matrix into a set of orthogonal basis vectors in order to inspect the individual contributions of each term.

The OLS algorithm assumes that the regression matrix F can be orthogonally decomposed as F = WA, where A is an $M \times M$ upper triangular matrix (it means $A_{i,j} = 0$ if $i > j$) and W is an $N \times M$ matrix with orthogonal columns in the sense that $W^T W = D$ is a diagonal matrix. (N is the length of y vector and M is the number of regressors). After this decomposition one can calculate the OLS auxiliary parameter vector g as

$$\mathbf{g} = \mathbf{D}^{-1}\mathbf{W}^T\mathbf{y}, \qquad (8)$$

where g_i is the corresponding element of the OLS solution vector. The output variance $(y^T y)/N$ can be explained as

$$y^T y = \sum_{i=1}^{M} g_i^2 w_i^T w_i + e^T e \qquad (9)$$

Thus the error reduction ratio, $[err]_i$ of F_i term can be expressed as

$$[err]_i = \frac{g_i^2 w_i^T w_i}{y^T y}. \qquad (10)$$

This ratio offers a simple mean for order and selects the model terms of a linear-in-parameters model according to their contribution to the performance of the model.

2. *GP and OLS*

During the operation of GP the hybrid GP OLS algorithm generates a lot of potential solutions in the form of a tree-structure. These trees may have terms (subtrees) that contribute more or less to the accuracy of the model.

The concept is the following: firstly the trees (the individual members of the population) are decomposed to subtrees (function terms of the linear - in- parameters models), then the error reduction ratios of these function terms are calculated; finally the less significant term(s) is/are eliminated. This “tree pruning” method is realized in every fitness evaluation before the calculation of the fitness values of the trees. The main goal of the application of this approach is to transform the trees to simpler trees which are more transparent, but their accuracy is close to the original trees. Because the further goal is to preserve the original structure of the trees as far as it possible (because the genetic programming works with the tree structure). This method always guarantees that the elimination of one or more function terms of the model can be done by ”pruning” the corresponding subtrees, so that, there is no need for structural rearrangement of the tree after this operation.

The proposed approach has been implemented in MATLAB that is the most widely applied rapid prototyping system. The aim of the toolbox is the data-based identification of static and dynamic models, since the approach proposed in this paper is can also be applied for static nonlinear equation discovery. At the development of the toolbox special attention has been given to the identification of dynamical input-output models. Hence, the generated model equations can be simulated to get one- and/or n-step ahead predictions. The toolbox is freeware, and it is downloadable from the website of the authors: ***www.fmt.veim.hu/softcomp***. Important parameters utilized in GP, are summarized in below Table 2

Table 2 Parameters used in GP – OLS toolbox

Population size	50
Maximum number of evaluated individuals	2500
Type of selection	Roulette-wheel
Type of mutation	Point-mutation
Type of crossover	One-point (2 parents)
Type of replacement	Elitist
Generation gap	0.9
Probability of crossover	0.5
Probability of mutation	0.5
Probability of changing terminal -non-terminal nodes (vice versa) during mutation	0.25

The individuals are represented by a structure called population variable.
Structure:

popu.generation	Scalar integer	Generation number
popu.symbols	Structure	Equation symbols
popu.size	Scalar integer	Number of individuals = nind
popu.chrom	Cells(1xnind)	Chromosomes

The chrom contains the chromosomes (individuals): equations encoded by trees.
Structure:

chrom{i}.fitness	Scalar integer	Fitness value
chrom{i}.mse	Scalar integer	Mean Square Error
chrom{i}.tree	Structure	The tree encoding equation

The popu.chrom{i} contains the 'chromosome' of i-th individual of the population.
The tree encodes the equation
Structure:

tree.maxsize	Scalar integer	Max. number of nodes, nn
tree.nodetyp	Vector of integers (nn x 1)	Type of nodes
tree.node	Vector of integers (nn x 1)	Nodes
tree.param	Vector of reals (nn/2 x 1)	Linear parameters
tree.paramn	Scalar integer	Number of parameters

The maxsize is equal the maximum number of nodes (= $2^{\text{max. deepth}}$-1).The nodetype contains the type of nodes. The value may be 1 or 2. If 1: operator, if 2: terminal node. The vector contains the nodes of tree in a structured way from top to down, from left to right. The node contains the index of the symbol assigned to the nodes. The vector is structured in the same way as nodetype variable. The indexes point to the gene.symbols variable. E.g. if nodetype(i) = m and node(i) = n then the i-th node is gene.symlist{m}{n}. The param contains the parameters of linear-in-parameters model represented by the tree. The vector is structured as the GP-OLS toolbox extracts the model-terms from the tree.

The symbols contains the equation symbols, namely the operators and variables. Structure:

Symbols {1}	Cells of strings	Operator symbols (internal nodes)
Symbols {2}	Cells of strings	Variables symbols (terminator nodes)

The symbols {1} contain the operator symbols. E.g. symbols {1} = {'+','*','*sqrt'}.

The symbol {2} contains the terminal symbols. E.g. symbols {2} = {‘u (k-1)',’u (k-2)'}.

The GP-OLS identification directly uses the symbols {1}, but not the symbols {2}. During the identification, only the size of the symbols {2} is used, it must be equal to the number of columns of X, i.e. length (symbols {2}) = size(X, 2) (it is important!). The gpols_result function uses the symbols {2} directly to write the equation.

GP- OLS matlab toolbox contain the following important functions

3.1 gpols_init: To make the population structure and to generate initial individuals

Syntax:

popu = gpols_init(popusize,maxtreedepth,symbols);

popu	Structure (population)	Resulted initial population
popusize	Scalar integer	Number of individuals (size of population)
maxtreedepth	Scalar integer	Maximum number of tree levels (tree depth)
symbols	Cells of strings	Equation symbols

This function initializes the population variable, clears and re-initializes its structure.

This function is called at the beginning of your program. This function does NOT evaluate the random generated (initial) individuals, so gpols_evaluate function needs to be called. The popu is the initialized population variable. The symbol is the list of operator and terminator strings.

Example:

popu = gpn_init(50,gene);
popu = gpn_evaluate(popu,[1:50],X,Y,[],optv);

3.2 gpols_evaluate: To evaluate the initial population.

Syntax:

newpopu = gpols_evaluate(popu,ixv,X,Y,Q,opte);

newpopu	Structure (population)	Resulted population (with fitness values)
popu	Structure (population)	Population
ixv	Vector of integers	Indexes of individuals must be evaluated
X	Matrix of reals (np x nr)	Input data for evaluation
Y	Vector of reals (np x 1)	Output data for evaluation
Q	Vector of reals (np x 1)	Optional weighting vector for X and Y
opte	Vector (4 x 1 or 1 x 4)	Options vector

This function evaluates the individuals of population. It determines the linear parameters, applies OLS to eliminate subtrees, and evaluates the fintess-value of resulted individuals (trees).

The popu is the population variable, which contains the individuals must be evaluated.

The ixv contains the indexes of individuals which must be evaluated. If. e.g. ixv = [1 2 5 6] then it means that the 1-th, 2-nd, 5-th, and 6-th member of population will be evaluated.

The X is regression matrix and the Y is the desired output vector. The terminal nodes of trees refer to X, the estimated output is compared to Y.

If Q vector is used, than the LS estimation problem modifies to ETˇdiag(Q)ˇE, where E = Y - Ye, Ye is the estimated output, Ye = XˇΘ.

The opte contains some parameters:

opte(1:2): a1 and a2 parameters of tree-size penalty function if they are zeros then the penalty is not used,
opt(3): OLS threshold value (0-1 or integer greater than 1),
opt(4): set 1 if you want polynomial evaluation, else 0.

3.3 gpols_mainloop: To execute one evolutionary-loop and step to next generation.

Syntax:

[newpopu,evnum] = gpols_mainloop(popu,X,Y,Q,opt);

newpopu	Structure (population)	New generation of population
evnum	Scalar integer	Number of function evaulations
popu	Structure (population)	Population
X	Matrix of reals (np x nr)	Input data for evaluation
Y	Vector of reals (np x 1)	Output data for evaluation
Q	Vector of reals (np x 1)	Optional weighting vector for X and Y
opt	Vector (1x10 or 10x1)	Options vector

This function execute one evolutionary-loop and generates the next generation of population.The user should call this function iteratively.

The popu is the population variable, which contains the individuals of current generation.

The X, Y, and Q are the regression matrix, output vector and weighting vector, respectively (see gpols_evaluation).

The opt contains the options:

opt(1): ggap, generation gap (0-1)
opt(2): pc, probability of crossover (0-1)
opt(3): pm, probability of mutation (0-1)
opt(4): sels, selection mode (0: roulette wheel, 1: total random, 2,3,..: tournament selection (with tournament size = sels)
opt(5): rmode, mode of tree-recombination (1 or 2)
opt(6): a1, first penalty parameter (default = 0)
opt(7): a2, second penalty parameter (default = 0)
opt(8): OLS threshold value (default = 0)
opt(9): if 1: polynomial evaluation, else normal evaluation (default = 0)
opt(10): if 1: always evaluate all individuals, else evaluate only new individuals (default = 0)

3.4 gpols_result: To write the current solution and the final result.

Syntax:

[sout,tree] = gpols_result(popu,info);

sout	String	Information string
tree	Structure (tree)	The best solution
popu	Structure (population)	The input population
info	Scalar integer	Asked information (0,1 or 2)

This function gets some information about the best member of the population variable.

Use info = 0 and popu = [] at the very beginning of your program to get a header string for info = 1: "Iter Fitness Solution".

Use info = 1 after every iteration to get a short information, e.g. "7. 0.729125 (y(k-2))+((u(k-1))*(u(k-1)))".

Use info = 2 at the end of your program to get detailed information about the final solution.
E.g.:

```
"fitness: 0.729125, mse: 0.462421
-0.638898 * (y(k-2)) +
0.493203 * ((u(k-1))*(u(k-1))) +
-0.069117"
```

If info > 0, than the function gives back the tree structure of the best solution (tree).
Example:

```
disp(gpols_result([],0));
gpols_mainloop(popu,X,Y,[],opt);
disp(gpols_result(popu,1));
disp(gpols_result(popu,2));
```

IV. Results and Discussion

In this section the modeling capabilities of both GPLAB and GPOLS have been tested and illustrated. Because GP is a stochastic optimization algorithm ten independent runs were executed for each method, while the best from all runs was considered.

A. Model of Glucose to Gluconic acid bio process

In this section the modeling capabilities of both GPLAB and GPOLS have been tested and illustrated. Because GP is a stochastic optimization algorithm ten independent runs were executed for each method, while the best from all runs was considered.

For obtaining the GP-based model, process data from 46 runs were used. The data set (see Table 3) comprises values of the three operating variables, namely, glucose concentration (g/L) (x_1), biomass concentration (g/L) (x_2), and dissolved oxygen (DO) concentration (mg/L) (x_3), and the corresponding values of the process output variable, i.e., gluconic acid concentration (y) [4]. The *normalized* data set was partitioned into the training set (batches 1-23) and the test set (batches 24-46). While the training set was used for computing the fitness of the GP-searched expressions, the test set was used to cross-validate the expressions. The objective of cross-validation is to test the prediction (generalization) ability of the GP-searched expressions on a data set different from the set used for obtaining the expression. To secure an overall optimal data-fitting expression, the GP procedure was repeated 100 times by employing different seed values for the pseudo-random number generator. In each such repeated run, a different mathematical expression was searched by the GP.

1. GP model predicted by GPLAB:

Parameters used:

Tree Initialization:	‘rampedinit’,
Functions:	‘plus, minus, times and mypower’
Terminals:	‘x1, x2, x3, rand, 10’
Cross over probability:	‘0.95’
Mutation Probability:	‘0.05’
Selection for Reproduction:	‘Roulette’
Expected number of children:	‘Rank89’
Measuring fitness:	‘regfitness’
Generations:	250
Fitness:	0.971358

Table 3 Experimental Data Utilized for Building GP [4]

batch	glucose concn (x_1) (g/L)	biomass concn (x_2) (g/L)	DO (x_3) (mg/L)	gluconic acid concn (y) (g/L)	Gluconic acid yield (y_{gt}) (%) [a]
1	100.0	1.00	10.0	6.416	5.90
2	150.0	2.00	10.0	48.015	29.42
3	200.0	2.00	15.0	27.100	20.76
4	150.0	2.50	15.0	57.946	35.51
5	150.0	3.00	15.0	57.389	35.16
6	120.0	2.00	25.0	36.262	27.77
7	120.0	2.00	30.0	45.020	34.48
8	150.0	2.00	30.0	94.424	57.86
9	150.0	3.00	25.0	80.486	49.32
10	150.0	2.00	40.0	128.907	78.99
11	150.0	2.00	45.0	146.036	89.48
12	150.0	2.00	50.0	154.230	94.50
13	180.0	2.00	50.0	175.525	89.63
14	150.0	3.00	40.0	129.006	79.05
15	150.0	2.50	50.0	154.360	94.58
16	150.0	2.50	55.0	152.440	93.41
17	150.0	2.50	60.0	148.940	91.26
18	160.0	2.50	60.0	163.067	93.67
19	175.0	3.00	55.0	176.490	92.69
20	160.0	3.00	60.0	162.420	93.30
21	180.0	3.00	60.0	172.598	88.13
22	150.0	3.00	60.0	151.280	92.70
23	100.0	3.00	60.0	21.803	20.04
24	100.0	2.00	10.0	6.670	6.13
25	120.0	2.50	10.0	22.952	17.58
26	100.0	2.00	15.0	7.829	7.20
27	150.0	2.00	15.0	57.261	35.09
28	120.0	2.00	20.0	31.486	24.12
29	150.0	2.00	20.0	66.900	40.99
30	150.0	2.50	20.0	67.449	41.33
31	150.0	3.00	20.0	67.328	41.25
32	150.0	2.00	35.0	111.328	68.22
33	150.0	2.50	30.0	95.988	58.82
34	150.0	3.00	30.0	94.707	58.03
35	150.0	2.50	40.0	129.930	79.61
36	150.0	3.00	35.0	111.604	68.38
37	150.0	2.00	60.0	152.430	93.40
38	120.0	2.00	60.0	73.502	56.30
39	150.0	3.00	45.0	144.651	88.63
40	180.0	2.50	55.0	179.064	91.94
41	150.0	3.00	50.0	152.890	93.68
42	180.0	2.50	60.0	174.483	89.09
43	150.0	3.00	55.0	154.230	94.50
44	166.0	3.00	60.0	169.450	93.82
45	165.0	3.00	60.0	167.910	93.53
46	162.0	3.00	60.0	164.870	93.54

[a] Gluconic acid percentage yield, $y_{gt} = 100\, y/1.088x_1$

Model equation predicted:
mypower(times(times(mydivide(X1,mypower(times(mypower(times(X1,X3),minus(0.82359,minus(X1,mypower(X3,X1)))),

times(minus(X1,minus(0.47932,X3)),X3)),times(times(mydivide(X1,mypower(times(mypower(mypower(times(X1,X3),minus(0.82359,minus(X1,X3))),minus(0.82359,minus(times(X3,X2),X3))),times(minus(X1,minus(0.47932,X3)),X3)),times(X1,times(minus(0.82359,minus(0.47932,X3)),X1)))),X3),times(minus(0.82359,minus(0.47932,X3)),X1)))),X3),times(minus(0.82359,minus(0.47932,X3)),X1)),0.38498)

2. GP model predicted by GP-OLS

The experimental data was split in to training set (1-23) and test set (24-46)

Parameters used:

Population size:	50.
Maximum number of evaluated individuals:	50
Type of selection:	roulette-wheel
Type of mutation point:	mutation
Type of crossover:	one-point (2 parents)
Type of replacement:	elitist
Generation gap:	0.9
Probability of crossover:	0.5
Probability of mutation:	0.5
Probability of changing terminal - non-terminal:	0.25 nodes (vice versa) during mutation
Functions:	+,-, /,*,*exp, +exp,-exp,/exp
Terminals:	x1, x2, x3

The best fit obtained is:
Fitness: 0.969914
MSE: 0.040065

The GP-based model obtained is:
0.232508 * (x1) +2.067329 * (((x3)*(x1))/exp((x3)*(x1))) + 0.028817
Elapsed Time: 256.938 seconds

To secure an overall optimal data-fitting expression, the GP procedure was repeated 100 times by employing different seed values for the pseudo-random number generator. In such repeated runs, the best mathematical expression was searched seven times by GP - OLS.

The high fitness values are indicative of the good prediction accuracy and generalization ability of the GP-based fermentor model. A comparison of the model predicted and actual process output values for the training and test set data is presented in Figures 2-7. As the prediction is evolutionary, following best models are obtained.

Model 1: Fitness: 0.957347, MSE: 0.027591
0.517192 * ((x1)-exp(((x3)*exp((x3)*exp(x3)))*(((x3)-exp(x1))*(x3)))) + 0.517702
Time=288.328000

Model 2: Fitness: 0.972972, MSE: 0.032064
-2.276953 * (x3) +2.323131 * ((x3)-exp((x1)-exp((x1)*exp(x3)))) + 0.888878
Time=222.235000

Model 3: Fitness: 0.969914, MSE: 0.040065
0.232508 * (x1) + 2.067329 * (((x1)*(x3))/exp((x1)*(x3))) + 0.028817
Time=205.875000

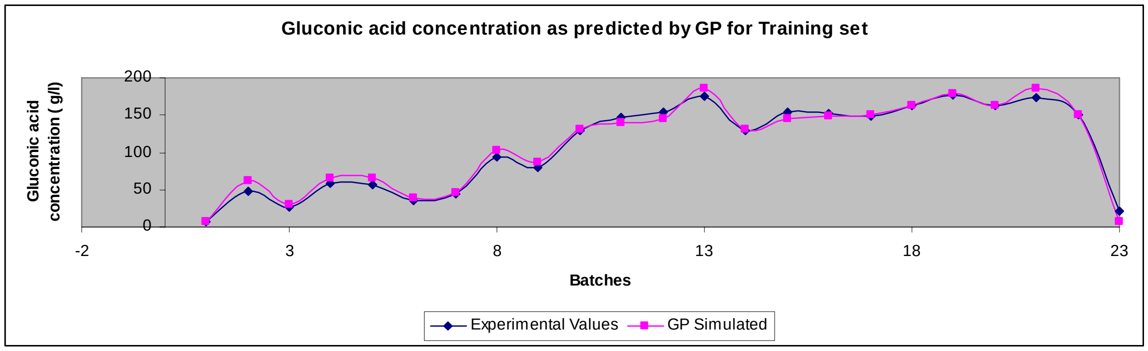

Fig. 2 Gluconic acid concentration as predicted by GP for Training set (Model 1)

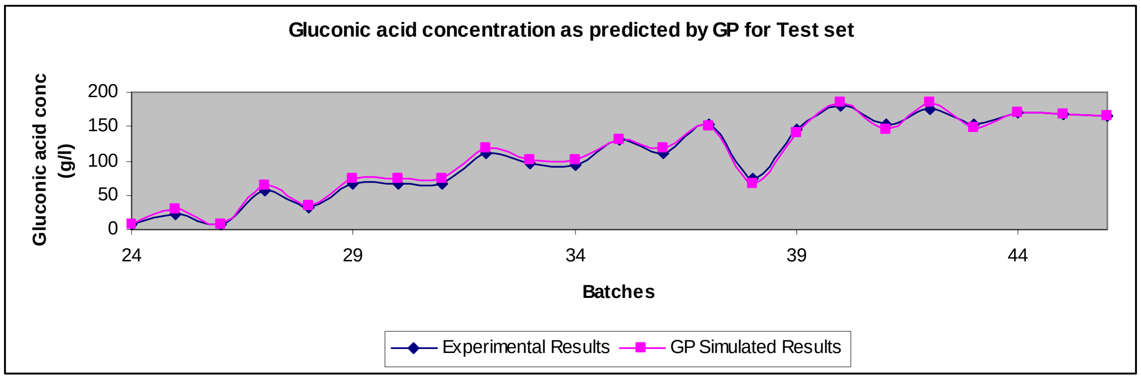

Fig. 3 Gluconic acid concentration as predicted by GP for Test set (Model 1)

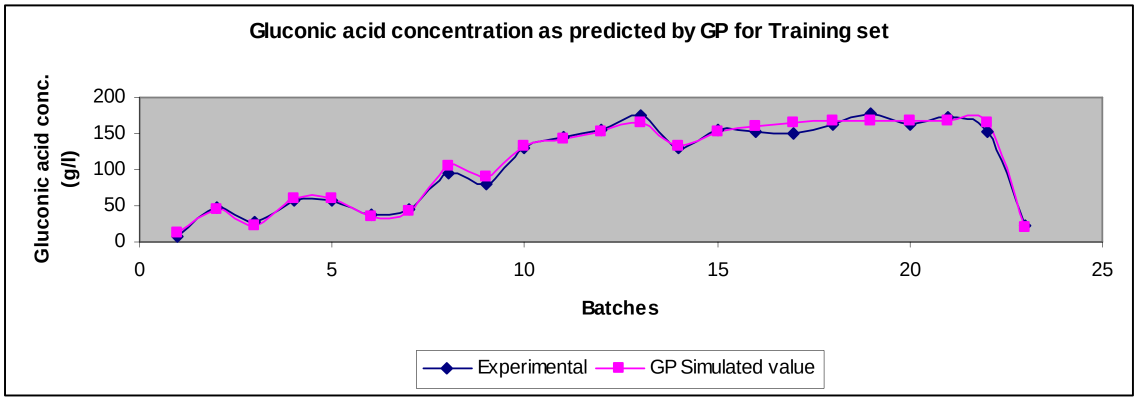

Fig. 4 Gluconic acid concentration as predicted by GP for Training set (Model 2)

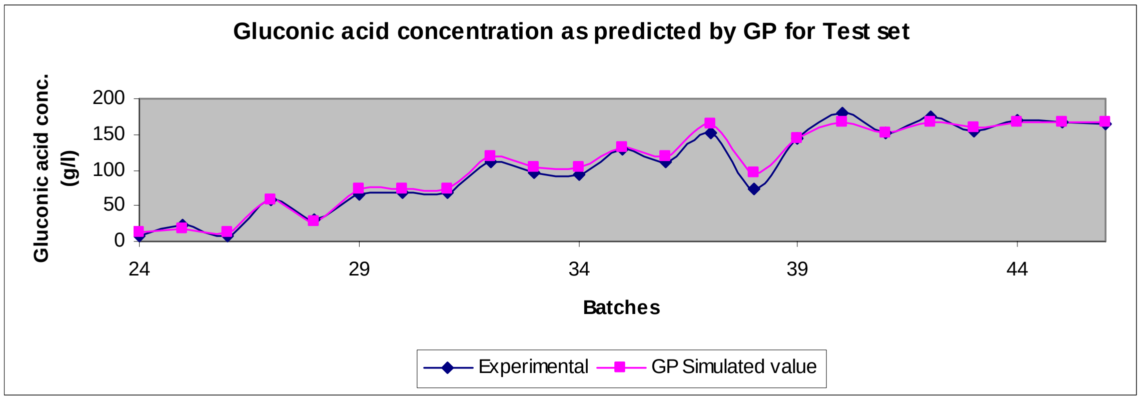

Fig. 5 Gluconic acid concentration as predicted by GP for Test set (Model 2)

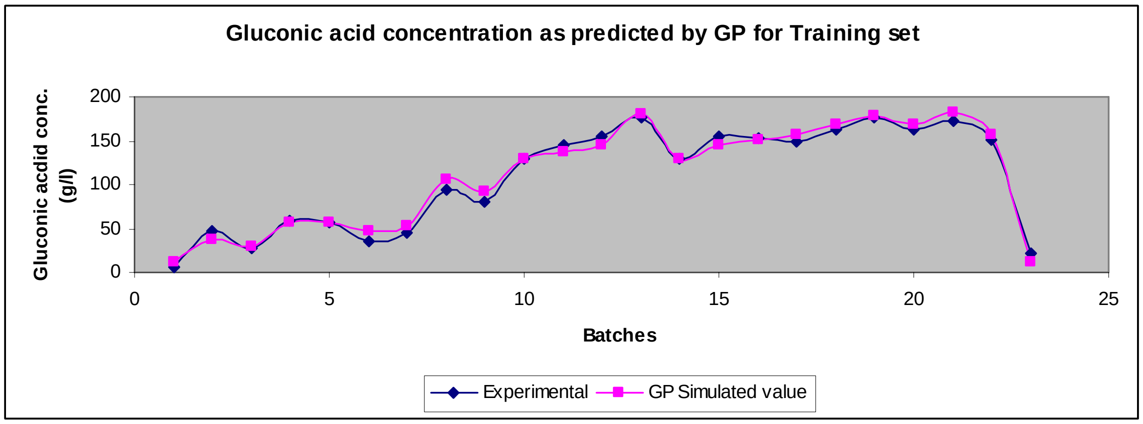

Fig. 6 Gluconic acid concentration as predicted by GP for Training set (Model 3)

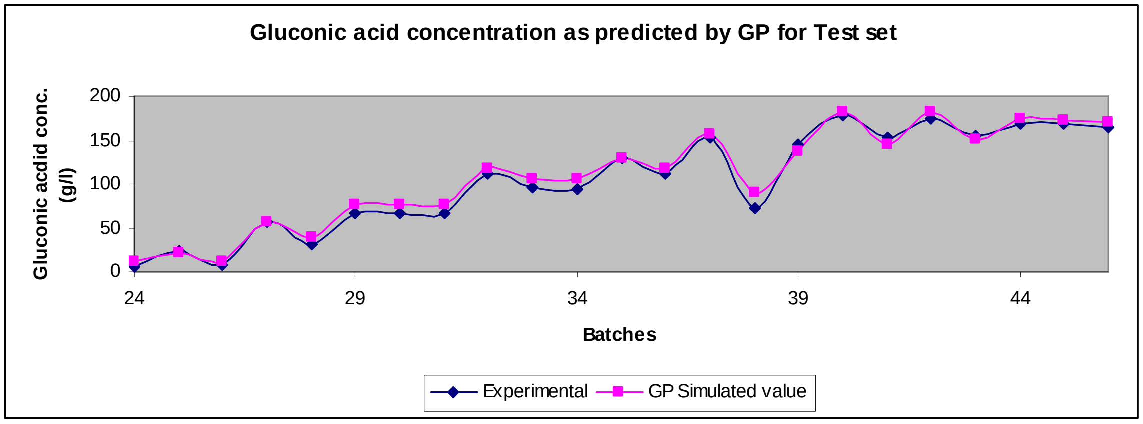

Fig. 7 Gluconic acid concentration as predicted by GP for Test set (Model 3)

B. *Dynamic modeling of fed-batch bioreactor*

Parameters used:	
Population size:	50
Maximum number of evaluated individuals:	20
Type of selection:	roulette-wheel
Type of mutation point:	mutation
Type of crossover:	one-point (2 parents)
Type of replacement:	elitist
Generation gap:	0.9
Probability of crossover:	0.5
Probability of mutation:	0.5
Probability of changing terminal - non-terminal: nodes (vice versa) during mutation	0.25
Functions:	+,-
Terminals:	u(k-i),y(k-i)

The best fit obtained is:
Fitness: 0.915976
MSE: 0.000002
Elapsed Time: 9 seconds

Table 4 Comparison of output of simulated value and GP model for Fed-Batch Bioreactor

Sl. No.	Time (h)	Feed rate (l/h)	Secreted protein conc. (g/l) (Simulated)	Secreted protein conc. (g/l) (GP model prediction)
1	0	0.146	0	0
2	1	0.19708	0	0
3	2	0.26604	0	0.00011
4	3	0.35911	0	0.00011
5	4	0.48476	0	0.00011
6	5	0.65437	0	0.00011
7	6	0.88332	0	0.00011
8	7	1.19239	0	0.00011
9	8	1.6096	0	0.00011
10	9	0	0.00102	0.00011
11	10	0	0.03237	0.03237
12	11	0	0.03237	0.03248
13	12	0	0.03237	0.03248
14	13	0	0.03237	0.03248
15	14	10	0.03237	0.03248
16	15	10	0.06847	0.06847
17	15.1393	10	0.065539	0.06547
18	15.2393	10	0.062306	0.06268
19	15.3393	10	0.059273	0.05959
20	15.4393	10	0.056519	0.05669
21	15.5393	10	0.05394	0.05407
22	15.7393	10	0.04942	0.05171
23	15.8393	10	0.04743	0.04953
24	159393	10	0.04556	0.04754
25	16	10	0.0445	0.04567

GP model :
-2.281562 * (((u(k-2))*(y(k-1)))*(u(k-2))) + 0.111181 * ((u(k-1))*(y(k-1))) + 22.699909 * ((u(k-2))*(y(k-1))) + 0.000111

The comparison of output of Simulated and GP model for Fed – Batch Bioreactor has been summarized in Table 4.

V. Conclusions

This limited but intensive study can be concluded with the following significant observations:

1 The study of two test examples using the MATLAB toolboxes for genetic programming shows the potentiality of the algorithm. Following salient points were observed during this study:
   1.1. Genetic programming is successful in giving the best solution, in all the situations.
   1.2. Genetic programming gives more than one equation to a given set of input output data, with varying fitness and mean square error. Thus, presenting user more choices to choose from.
   1.3. Crossover and mutation rates are selected based on trial runs.
   1.4. GPLAB though versatile with its features, lacks pruning of branches in trees. Hence produces more complex equations as compared to GP OLS which uses Orthogonal least square (OLS) method to prune branches in trees and is able to give simpler accurate and high fitness equations. The best example is the fermentor modeling, in which GPLAB gave a equation which was impossible to discern and GP OLS gave a simpler model of an excellent fit.

2 Modeling of glucose and gluconic acid bioprocess was done using experimental input-output data. The best equation from 100 test runs was obtained. The high fitness values are indicative of the good prediction accuracy and generalization ability of the GP-based fermenter model. Based on the simulations following significant points were observed.
   2.1. GP-OLS was able to remove the uncontributing branches as compared to GPLAB, to give rise to a simple equation in 500 generations.
   2.2. A comparison of the model predicted and actual process output values for the training and test data shows an accurate prediction and generalization ability.
   2.3. The model predicted by GP OLS, does not contain biomass concentration (x_2) term, clearly indicating the acute contribution to the model. Hence, the biomass concentration need not be a measured quantity.

3 The capability of GP OLS to generate non-linear input-output dynamic systems has been tested using two test examples. In one of the examples a fed batch bioreactor that has been studied by Park and Ramirez [5] has been considered. The simulations were carried out using ODE23s subroutine (MATLAB Library). The program was checked to be error free. The data generated was used to obtain the GP dynamic model. The simulation and GP model prediction results indicate GP OLS is an efficient and fast method for

predicting the order and structure for non-linear input and output model.

References

[1] C. Rhodes and M. Morari, “Determining the model order of non-linear input/output systems”, *AIChE Journal*, vol. 44, 1998, pp 151 – 163.

[2] J. Madar, J. Abonyi and F. Szeifert, “Model order Selection of nonlinear Input Output models a Clustering based approach”, *Journal of Process Control,* vol. 14, 2004, pp. 593 – 602.

[3] B. McKay, M. Willis and G. Barton, “Steady-state Modeling of Chemical Process Systems using Genetic Programming”, *Computers and Chemical Engineering,* vol. 21, 1997, pp 981 – 996.

[4] J. Cheema, N. V. Sankpal, S. S. Tambe and B. D. Kulkarni, “Genetic Programming Assisted Stochastic Optimization Strategies for Optimization of Glucose to Gluconic Acid Fermentation”, *Biotechnology Progress*, vol.18, 2002, pp 1356 – 1365.

[5] G. L. Miller, “Use of Dinitrosalicyclic Acid Reagent for Determination of Reducing Sugar”, *Anal. Chem.*, vol. 31, 1959, pp 426-429.

[6] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, Cambridge, MA, 1992.

[7] S. Park and W. F. Ramirez, “Optimal production of Secreted Protein in Fed-Batch Reactors”, *AIChE Journal*, vol. 34, 2003, pp 1550 – 1558.

[8] D. E. Goldberg, *Genetic Algorithms in search, Optimization, and Machine learning*, Addison-Wesley, MA, 1989.

[9] B. V. Babu, *Process Plant Simulation*, Oxford University Press, New York, 2004.

[10] G. C. Onwubolu and B. V. Babu, *New Optimization Techniques in Engineering*, Springer-Verlag, Germany, 2004.

[11] Home page of Genetic Programming Inc. “Preparatory Steps of Genetic Programming”, Available online as on May, 2006 at *http://www.genetic-programming.com/gppreparatory.html*.

[12] S. Silva , “RGPLAB: A Genetic Programming Toolbox for MATLAB”

[13] J. Madar., J. Abonyi, and F. Szeifert, “Genetic Programming for the Identification of Nonlinear Input-Output Models”, GP OLS toolbox for MATLAB, 2005.

Dr. B. V. Babu borned in Tuni of Andhra Pradesh in India on February 15, 1961. He did his B.Tech in chemical engineering from A.U. College of Engineering, Andhra University, Waltair, Andhra Pradesh, India in 1983; and subsequently did his M.Tech in chemical engineering from C.I.T. of Bharatiar University, Tamil Nadu, India in 1985. He completed his Ph.D. in hydrodynamics and heat transfer in gas-liquid cocurrent downflow through packed beds from I.I.T. Bombay, India in 1993. He became a Member (M) of IAENG.

He has 21 years of teaching, research, administration, and consultancy experience. He is currently working as a Professor and Head of Chemical Engineering Department & Assistant Dean of Engineering Services Division at Birla Institute of Technology and Science (BITS), Pilani, Rajasthan, India. He has around 120 research publications in various International Journals and Conference Proceedings to his credit. He has published three books: 1. New Optimization Techniques in Engineering (Heidelberg, Germany: Springer-Verlag, 2004); 2. Process Plant Simulation (New Delhi, India, Oxford University Press, 2004); 3. Environmental Management Systems (Pilani, India, BITS-Pilani, 2005). He guided 3 PhD students, 28 ME Dissertation students and 24 Thesis students and around 170 Project students. He is currently guiding 6 PhD candidates, 2 Dissertation students and 9 Project students. He currently has 2 research projects from UGC & DST. His research interests include Evolutionary Computation (Population-based search algorithms for optimization of highly complex and non-linear engineering problems), Environmental Engineering, Biomass Gasification, Energy Integration, Artificial Neural Networks, Nano Technology, and Modeling & Simulation.

Prof. Babu is Life member of Indian Institute of Chemical Engineers (IIChE), Life member of Indian Society for Technical Education (ISTE), Life member of Institution of Engineers (IE), Fellow of International Congress of Chemistry and Environment (ICCE), Life member of Indian Environmental Association (IEA), Life member of Society of Operations Management (SOM), Associate Member of International Society for Structural and Multidisciplinary Optimization (ISSMO), and Member of International Institute of Informatics and Systemics (IIIS). He is the recipient of National Technology Day (11th May, 2003) Award given by CSIR, India obtained in recognition of the research work done in the area of ‘A New Concept in Differential Evolution (DE) – Nested DE’. He is Editorial Board Member of three International Journals ‘Energy Education Science & Technology’, ‘Research Journal of Chemistry and Environment’, and ‘International Journal of Computer, Mathematical Sciences and Applications’. He is the referee & expert reviewer of 19 International Journals (Chemical Engineering Science, Computers and Chemical Engineering, International Journal of Heat and Mass Transfer, Chemical Engineering Journal, IEEE Transactions on Evolutionary Computation, European Journal of Operations Research, IEEE Transactions on Systems Man and Cybernetics, Industrial & Engineering Chemistry Research, International Journal of Systems Science, ASME Journal of Electronic Packaging, Bioresource Technology, Applied Mathematical Modelling, Waste Management, International Journal of Environment and Pollution, Progress in Energy and Combustion Science, Journal of The Indian Institute of Science, Materials and Manufacturing Processes, Transactions on Internet Research, The Proceedings of the Pakistan Academy of Sciences). He is also on the Programme Committees as an expert reviewer at many International Conferences. He reviewed three books of McGraw Hill, Oxford University Press, and Tata McGraw Hill publishers. He is PhD Examiner for one candidate and PhD Thesis Reviewer for 3 Candidates. Prof. Babu is the Organizing Secretary for “National Conference on Environmental Conservation (NCEC-2006)” held at BITS-Pilani during September 1-3, 2006. He also organized invited special sessions at two international conferences (CIRAS-2003 and SCI-2004).

Mr. S. Karthik is a M.E. student at BITS-Pilani during 2003-2005 and did his Dissertation work with Prof. B.V.Babu.