



HENRIQUE CARVALHO DE CASTRO

**A MODIFIED MGGP ALGORITHM FOR STRUCTURE
SELECTION OF NARMAX MODELS**

LAVRAS – MG

2021

HENRIQUE CARVALHO DE CASTRO

**A MODIFIED MGGP ALGORITHM FOR STRUCTURE SELECTION OF NARMAX
MODELS**

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Engenharia de Sistemas e Automação, para a obtenção do título de Mestre.

Prof. DSc. Bruno Henrique Groenner Barbosa
Orientador

**LAVRAS – MG
2021**

**Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca
Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).**

Castro, Henrique Carvalho de.

A Modified MGGP Algorithm for Structure Selection of
NARMAX Models / Henrique Carvalho de Castro. - 2021.

98 p. : il.

Orientador(a): Bruno Henrique Groenner Barbosa.

Dissertação (mestrado acadêmico) - Universidade Federal de
Lavras, 2021.

Bibliografia.

1. Identificação de Sistemas. 2. Modelos NARMAX. 3.
Computação Evolucionária. I. Barbosa, Bruno Henrique Groenner.
II. Título.

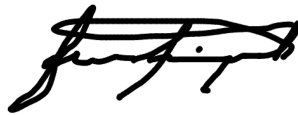
HENRIQUE CARVALHO DE CASTRO

**A MODIFIED MGGP ALGORITHM FOR STRUCTURE SELECTION OF NARMAX
MODELS**

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Engenharia de Sistemas e Automação, para a obtenção do título de Mestre.

APROVADA em 29 de março de 2021.

Prof. DSc. Bruno Henrique Groenner Barbosa UFLA
Prof. DSc. Erivelton Geraldo Nepomuceno UFSJ
Prof. DSc. Danton Diego Ferreira UFLA



Prof. DSc. Bruno Henrique Groenner Barbosa
Orientador

**LAVRAS – MG
2021**

Acknowledgements

I am very thankful for Bruno's guidance, without which this work would not be possible. For my friends from the lab, who made the labour pleasurable. And for my friends and family from "real life", who make the life real.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

I don't want the cheese; I just want to get out of the trap.

(SPANISH PROVERB)

ABSTRACT

In the area of system identification, the input-output *Nonlinear Autoregressive Moving Average with Exogenous Variables* (NARMAX) models are of great interest. The most challenging task faced when working with such models is to select the appropriate model structure that best represent the underlying system in the data. This structure selection is usually made via *Error Reduction Ratio* (ERR)-based algorithms. These algorithms suffer from the curse of dimensionality when high degree of nonlinearity and long term dependencies are required. Further, some nonlinearities require specific functions or terms in the model structure to be reproduced, i.e. the hysteretic behavior. The ERR-based algorithm may leave these fundamental terms out of the selected structure. Alternatively, *Evolutionary Algorithms* (EAs) can be used to perform the structure selection process. They are methods that evolves a population of individuals through generations (or epochs) via selection, mutation, and reproduction phenomena. In the case of system identification, each individual would be a candidate model. This dissertation proposes the hybridization of an EA called *Multi-Gene Genetic Programming* (MGGP) with an ERR-based algorithm to perform the identification process even for those cases in which specific functions are required. In total, four experiments are performed. The first two experiments analyse noise level and soft input problems using stochastic test systems to generate data. As result we verify that the increment of equation noise level does not interfere in the structure selection outcome and that the hybridization MGGP/ERR is beneficial in comparison with the standalone MGGP for the *soft input* problem. The MGGP/ERR yields more parsimonious models that perform better in free-run simulation. The third experiment is the identification of a hydraulic pumping system benchmark. It is shown that the MGGP/ERR is able to explore a wide range in search space for which the traditional ERR-based algorithm would require a very high computational power. And finally, the last experiment is the identification of a piezoelectric actuator, which is characterized by the hysteretic behavior. It is included specific functions in the search space so that the MGGP/ERR is able to identify hysteresis. A novel and easy-to-use toolbox based on Python was developed and is available under GPL.

Keywords: Nonlinear system identification. Multi-gene genetic programming. Error reduction ratio. NARMAX models.

RESUMO

Na área de identificação de sistemas, os modelos de entrada-saída NARMAX (*Nonlinear Autoregressive Moving Average with Exogenous Variables*) são de grande interesse. A tarefa mais desafiadora quando se trabalha com esses modelos é a seleção da estrutura adequada do modelo que melhor represente o sistema subjacente aos dados. Normalmente, essa seleção de estrutura é feita por meio de algoritmos baseados no critério ERR (*Error Reduction Ratio*). Esses algoritmos sofrem com a maldição da dimensionalidade quando são requeridos alto grau de não linearidade e dependências de longo prazo. Ademais, algumas não linearidades necessitam de funções ou termos específicos na estrutura do modelo para serem reproduzidas, *i.e.*, o comportamento de histerese. O algoritmo baseado em ERR pode deixar esses termos fundamentais fora da estrutura selecionada. Alternativamente, Algoritmos Evolucionários (AE) podem ser usados para realizar o processo de seleção de estrutura. Eles são métodos que evoluem uma população de indivíduos através das gerações por meio dos fenômenos de seleção, mutação e reprodução. No caso da identificação de sistemas, cada indivíduo seria um candidato a modelo. Essa dissertação propõe a hibridização de um EA chamado MGGP (*Multi-Gene Genetic Programming*) com um algoritmo baseado em ERR para desempenhar o processo de identificação mesmo naqueles casos em que funções específicas são requeridas. No total, são realizados quatro experimentos. Os dois primeiros analisam os problemas de nível de ruído e entrada suave utilizando sistemas de teste estocásticos para gerar os dados. Como resultado, verificamos que o incremento do nível de ruído na equação não interfere no resultado da seleção de estrutura e que a hibridização MGGP/ERR é benéfica em comparação com o MGGP autônomo para o problema de entrada suave. O MGGP/ERR produz modelos mais parcimoniosos que apresentam melhor desempenho em simulação livre. O terceiro experimento é a identificação de um benchmark de sistema de bombeamento hidráulico. É mostrado que o MGGP/ERR é capaz de explorar um amplo espaço de busca para o qual um método tradicional baseado em ERR requeriria um poder computacional muito alto. E finalmente, o último experimento é a identificação de um atuador piezoelétrico, que se caracteriza pelo comportamento de histerese. São incluídas funções específicas no espaço de busca de tal forma que o MGGP/ERR seja capaz de identificar a histerese. Uma *toolbox* nova e fácil de usar baseada em Python foi desenvolvida e está disponível sob Licença Pública Geral.

Palavras-chave: Identificação de sistemas não lineares. Programação genética multi-gene. Taxa de redução de erro. Modelos NARMAX

LIST OF TABLES

Table 5.1 – Number of times the algorithms selected all terms present in the systems during 30 Monte Carlo simulations.	51
Table 5.2 – Hydraulic pump models. The first two are from (BARBOSA et al., 2011) obtained via OLS method. PE is the one-step-ahead MSE (mlc^2); SE is the free-run MSE (mlc^2); N_p the number of parameters present in the model; and $max. N_p$ is the number of all possible regressors for given (l, n_y, n_u)	65
Table 5.3 – Memory usage for building one full candidate regressor matrix for the real problem training data (3200 samples)	67
Table 5.4 – Piezoelectric actuator models parameters (obtained via LS)	72
Table 5.5 – Piezoelectric models’ free-run simulation errors	72
Table 5.6 – Piezoelectric actuator gray-box model’s parameters (obtained via CLS)	75

LIST OF SCRIPTS

5.1	Influence of the noise level experiment.	50
5.2	Influence of OLS probability and PEM and SEM techniques experiment	54
5.3	Hydraulic pump identification experiment	63
5.4	Piezoelectric actuator identification experiment	69

LIST OF ACRONYMS

AIC Akaike's Information Criterion

AR Autoregressive

CLS Constrained Least Squares

EA Evolutionary Algorithms

EE Equation Error

ELS Extended Least Squares

ERR Error Ratio Reduction

FBNN Filter Based Neural Networks

FROE Forward-Regression Orthogonal Estimator

GA Genetic Algorithms

GP Genetic Programming

LS Least Squares

MA Moving Average

MAPE Mean Absolute Percentage Error

MBF-GP Multi Basis Function Genetic Programming

MDL Minimum Description Length

MGGP Multi-Gene Genetic Programming

MISO Multiple Input Single Output

MOEA Multi-Objective Evolutionary Algorithms

MSE Mean Squared Error

MSSE Mean Square Simulation Error

NARMAX Non-linear Autoregressive Moving Average with Exogenous input

NARX Non-linear Autoregressive with Exogenous input

OE Output Error

OLS Orthogonal Least Squares

PE Prediction Error

PEM Prediction Error Minimization

RBF Radial Basis Function

RLS Recursive Least Squares

RMSE Root Mean Squared Error

SE Simulation Error

SEM Simulation Error Minimization

SGA Simple Genetic Algorithm

SISO Single Input Single Output

SRR Simulation Error Reduction Ratio

STLF Short Term Load Forecasting

WGN White Gaussian Noise

LIST OF SYMBOLS

u, y - input and output signals

ξ - residual

n_y - maximum output term lag

n_u - maximum input lag

n_ξ - maximum residual lag

l - degree of nonlinearity

v - equation noise

e - output noise

\mathcal{M} - model structure

θ - model parameters

Ψ - regressor matrix

v - white noise

\bar{u}, \bar{y} - steady-state values of $u[k]$ and $y[k]$

CONTENTS

1	Introduction	14
1.1	Objectives	15
1.2	Contributions of this Dissertation	16
1.3	Organization	17
2	System Identification	18
2.1	Dynamic Tests and Data Acquisition	18
2.2	Choice of Mathematical Representation	19
2.2.1	A NARMAX model for a White Noise Output Error Problem	20
2.3	Model Structure Selection	20
2.4	Parameter Estimation	22
2.5	Model Validation	23
2.5.1	Simulation Methods	23
2.5.1.1	One-step-ahead Prediction	24
2.5.1.2	Free-run Simulation	24
2.5.1.3	Multistep-ahead Prediction	24
2.5.1.4	Multiple Shooting	25
3	Evolutionary Algorithms	27
3.1	Genetic Algorithms	28
3.1.1	GA Representation	29
3.1.2	GA Selection Operators	30
3.1.3	GA Recombination Operators	31
3.1.4	GA Mutation Operators	32
3.1.5	Selecting new generation	33
3.1.6	Genetic Algorithms in System Identification	33
3.2	Genetic Programming	36
3.2.1	GP Representation	37
3.2.2	GP Selection Operators	37
3.2.3	GP Recombination and Mutation Operators	37
3.2.4	Genetic Programming in System Identification	39

3.3	Multi-Gene Genetic Programming	41
3.3.1	Modification of the Genetic Operators	41
3.4	Multi-Gene Genetic Programming in System Identification	42
4	Materials and Methods	44
4.1	Proposed Algorithm	44
4.2	Individuals Evaluation	47
5	Experiments	48
5.1	Influence of the Noise Level	48
5.1.1	Data set	49
5.1.2	Algorithm parameters and toolbox guide	49
5.1.3	Results and discussion	51
5.2	Influence of OLS Probability and PEM and SEM Techniques	52
5.2.1	Data set	52
5.2.2	Algorithm parameters and toolbox guide	53
5.2.3	Results and discussion	54
5.2.3.1	System S_2	55
5.2.3.2	System S_3	57
5.3	Hydraulic Pump	60
5.3.1	Data set	61
5.3.2	Algorithm setup	61
5.3.3	Results and discussion	64
5.4	Piezoelectric Actuator	67
5.4.1	Data set	68
5.4.2	Algorithm setup	70
5.4.3	Results and Discussion	71
6	Conclusion	76
A	The MGGP toolbox	81
A.1	mggpElement Class	81
A.1.1	Create SISO Model	81
A.1.2	Create MISO Model with Constant Term	82

A.1.3	Simulate a System	84
A.1.3.1	Simulate a System with White Noise Equation Error	84
A.1.3.2	Simulate a System with Colored Noise Equation Error	86
A.1.4	The Least Squares Functions	86
A.1.5	Predictors	87
A.1.6	MSE built-in scores	88
A.1.7	Moving Average Models	89
A.1.8	Get Regressor Matrix	89
A.1.9	Orthogonal Least Squares	90
A.1.10	Include New Functions	91
A.1.11	Handling constraints with built-in functions	92
A.2	Save and Load functions	94
A.3	The mggpEvolver Class	96
A.3.1	Simple Example	97

1 INTRODUCTION

Dynamic models can be built directly from input and output data through a process known as *system identification*. A model allows the comprehension and prediction of a system's behavior. To complete an identification problem, one must perform the following steps: *i) dynamic tests, ii) choice of mathematical representation, iii) model structure determination, iv) parameter estimation, and v) model validation* (AGUIRRE, 2015). Generally, most of the real systems of interest are nonlinear. In this sense, *Nonlinear Autoregressive Moving Average with Exogenous Variables* (NARMAX) models (LEONTARITIS; BILLINGS, 1985) are of great interest in this area due to their flexibility and representation capacity. The main problem encountered in working with NARMAX models is the selection of the appropriate model structure, that is, the determination of the regressors that together best represent the system. The key point of structure selection is to choose a model structure as simple as possible but sufficiently complex to capture the dynamics underlying the data (AGUIRRE; LETELLIER, 2009).

A widely used criterion for NARMAX model structure selection is the *Error Reduction Ratio* (ERR) (BILLINGS; CHEN, 1989), which evaluates how good each single model term is at explaining the output data variance. It is considered a one-step-ahead *Prediction Error Minimization* (PEM) technique. Some algorithms were built based on the ERR criterion for structure selection, such as the *Forward Regression Orthogonal Estimator* (FROE) (BILLINGS; CHEN, 1989) and other *Orthogonal Least Squares* (OLS)-based methods (CHEN; BILLINGS; LUO, 1989). Piroddi and Spinelli (2003) discuss the limitations of ERR-based algorithms, mainly regarding training data with the presence of certain input characteristics (soft input) and the use of *Simulation Error Minimization* (SEM) techniques. Another issue is that such techniques suffer from the *curse of dimensionality* with the increment of the degree of nonlinearity and higher long-term dependencies.

Alternative methods for solving the structure selection problem can be derived from *Evolutionary Algorithms* (EAs), such as *Genetic Algorithms* (GAs) (GOLDBERG; HOLLAND, 1988; HOLLAND, 1975) and *Genetic Programming* (GP) (KOZA, 1992). Some examples of these methods can be seen in Chen et al. (2007), Li and Jeon (1993), Madar, Abonyi and Szeifert (2005). However, these methods depend on the assembly of a full regressor matrix for the given maximum delays and nonlinearity degrees, which may become computationally impracticable. One very flexible algorithm for use in system identification is the *Multigene Genetic Programming*

(MGGP) (HINCHLIFFE et al., 1996; HINCHLIFFE, 2001; HINCHLIFFE; WILLIS, 2003). In the NARMAX context, each *locus* of an MGGP individual is a model term represented by a genetic program. Its features allow the population size to fluctuate and facilitate automatic time lag determination, which eliminates the need for a regressor matrix with all possible terms.

Recently, several modeling and forecasting works have been developed with the use of MGGP (such as Ghareeb and Saadany (2013), Mehr and Kahya (2017), Riahi-Madvar et al. (2019), Safari and Mehr (2018)). The algorithm has been shown to be very flexible and to present good performance in building *rational* models. In the algorithm, the user is able to define the set of functions to be used in the modeling (such as multiplication and exponentiation) and it is possible to change this set without any modification in the algorithm itself. This flexibility of MGGP individual representation is the main feature explored in this work.

Some nonlinearities are known to be challenging for system identification techniques. One of them is hysteresis behavior, which presents a memory effect. Several works have investigated which features must be present in a model to reproduce hysteresis (ABREU et al., 2020; DENG; TAN, 2009; MARTINS; AGUIRRE, 2016; MORRIS, 2011). However, finding algorithms that can capture all this features automatically and deal with very large search spaces is challenging due to curse of dimensionality.

1.1 Objectives

The main objective of this work is to develop a system identification approach able to deal with large search spaces, automatic time lag determination and easy inclusion of specific functions to model nonlinear systems.

Specific objectives consist of:

1. analyse the algorithm performance under Output Error and Equation Error problems;
2. analyse the algorithm performance under different cost functions for distinct input characteristics;
3. implement and analyse a hybridization approach between MGGP and OLS/ERR in order to find parsimonious models;

4. analyse the inclusion of specific functions into the algorithm's primitive set that allow the algorithm to model hysteresis.
5. develop a toolbox in *python* that implements the proposed MGGP/ERR algorithm. We sought to develop a friendly interface that allows the users to build and test their own fitness functions.

1.2 Contributions of this Dissertation

This dissertation proposes the use of a hybrid MGGP/ERR algorithm to solve the structure selection problem of NARMAX models. Both algorithms (MGGP and OLS/ERR) work in a *symbiotic* way. On the one hand, the MGGP algorithm selects groups of terms from the candidate regressor space. Over those groups the OLS/ERR structure selection algorithm is applied, and the resultant models are assessed by a cost function. This interaction is supposed to facilitate the OLS/ERR task in a "divide and conquer" manner when the search space is very large, it avoids the curse of dimensionality. On the other hand, the OLS/ERR algorithm works as a pruning method over the MGGP individuals. It leaves only relevant terms in all models. Therefore, this interaction is supposed to guide the population evolution towards a promising region in the search space.

First, the algorithm is applied in the identification of some test systems. We address the *Equation Error* (EE) problem, and analyse the influence of OLS/ERR probability over model errors for systems excited by soft input. Then, we use a hydraulic pumping data benchmark and finally perform hysteretic system identification. Some specific functions are included into the primitive set to allow models to reproduce hysteresis. The results show that: the algorithm explores a wide search space in which traditional OLS algorithms would require high computational power and memory; the hybridization of MGGP with OLS/ERR is beneficial for the identification of systems excited by soft input; and that with the inclusion of specific functions in the primitive set of the MGGP individuals, the algorithm is able to model hysteresis.

Most of these results can be found in the following published papers:

- CASTRO, H. C.; BARBOSA, B. H. G.. Multi-gene Genetic Programming for Structure Selection of Polynomial NARMAX models. *Anais da Sociedade Brasileira de Automática*, v. 2, n. 1, 2020.

Other papers related to system identification published during the development of this dissertation are:

- CASTRO, H. C.; BARBOSA, B. H. G.. Algoritmos Multi-Objetivos para Detecção de Estruturas em Modelos NARX utilizando técnicas PEM e SEM. Em: Anais do 14º Simpósio Brasileiro de Automação Inteligente. Campinas : Galoá. 2019.
- MOTA, F. L. O., CARVALHO, G. S., CASTRO, H. C., BARBOSA, B. H. G. Identificação de um Sistema de Bombeamento Hidráulico com Algoritmo Evolucionário Multi-objetivo. Anais da Sociedade Brasileira de Automática, v. 2, n. 1, 2020.

1.3 Organization

This dissertation is organized as follows: Chapter 2 presents an introductory material on system identification; Chapter 3 introduces basic concepts of evolutionary algorithms and some works on its application in system identification, Chapter 4 presents the proposed algorithm, Chapter 5 describes and discusses the performed experiments, and Chapter 6 concludes the work with final considerations. The Appendix A consists of a tutorial on *how to use* the toolbox developed in this dissertation.

2 SYSTEM IDENTIFICATION

Systems modeling is the research area that aims at developing mathematical models that represent real dynamic systems. There are several techniques that can be used to obtain those models and they are classified as follows (AGUIRRE, 2015; SJÖBERG et al., 1995):

- *white-box modeling*: models are obtained from mathematical relations that describe the phenomena involved in the process. Thus, it is necessary a deep knowledge of the system;
- *gray-box modeling (or identification)*: models are obtained from input-output data in addition to prior information about the process;
- *black-box modeling (or identification)*: models are obtained from dynamic input-output data with no prior information.

Due to difficulties faced to accomplish white-box modeling, *system identification* offers alternative techniques that only need dynamic input-output data. The main steps of an identification problem are (AGUIRRE, 2015):

1. *dynamic tests and data acquisition*;
2. *choice of mathematical representation*;
3. *model structure selection*;
4. *parameter estimation*; and
5. *model validation*.

2.1 Dynamic Tests and Data Acquisition

In this step, the experimental data are acquired. The input signal must be designed in order to extract representative dynamical information. This signal should be able to excite all dynamical and static characteristics of the system. For nonlinear system identification, random number generators are normally used to build high order persistently exciting input signals (AGUIRRE, 2015).

The sampling period must be chosen carefully. Under-sampled data, in which the sampling time is excessively large, may cause a bad representation of the real system. Whereas over-sampled data, in which the sampling time is very short, may cause numerical instability and high computational cost, which hinders the model structure determination (BILLINGS; AGUIRRE, 1995).

2.2 Choice of Mathematical Representation

There are several mathematical representations that can be chosen. It is worth to highlight the *artificial neural networks* (BRAGA; CARVALHO; LUDERMIR, 2000; HAYKIN, 2001), the Volterra series (RUGH, 1983), the interconnected block models (COELHO, 2002; PEARSON; POTTMANN, 2000; WIENER, 1966; WIGREN, 1993) and the polynomial and rational models (CORRÊA, 2001; BILLINGS; CHEN, 1989; BILLINGS; TAO, 1991; BILLINGS; ZHU, 1994).

The NARMAX models (LEONTARITIS; BILLINGS, 1985) are of great interest in the area of system identification due to its flexibility and representation capabilities. These models are extensions of NARX models, in which residual terms are included to remove parameters' bias. In NARMAX models, the current output is obtained from past input-output and residual signals, as follows:

$$y[k] = F^l(y[k-1], \dots, y[k-n_y], u[k-1], \dots, u[k-n_u], \xi[k-1] \dots \xi[k-n_\xi]) + \xi[k], \quad (2.1)$$

where $F[\cdot]$ is a nonlinear function; $y[k]$, $u[k]$ and $\xi[k]$ are the output signal, input signal and residual vector, respectively; and n_y , n_u and n_ξ are their respective maximum lags. In the case of *polynomial* models, the nonlinear function is a polynomial function of degree l (F^l).

Note that the number of terms, or regressors, of the model increases exponentially with the augmentation of the nonlinearity degree and input and output maximum lags. Moreover, an over-parameterized model can lead to numerical instability in parameter estimation, unnecessary computational cost and the representation of dynamics that do not exist in the real system (AGUIRRE; BILLINGS, 1995). The aforementioned issues justify the importance of the structure selection step in system identification.

2.2.1 A NARMAX model for a White Noise Output Error Problem

According to Aguirre (2015), a *white noise* is a signal characterized by an autocorrelation function that satisfies $r_{\xi\xi}(k) = 0, \forall k \neq 0$. Its power spectrum contains energy at all frequencies, which are all equally important. On the other hand, in *colored noise*, the frequencies are not all equally important. It can be modeled, for instance, by an *Autoregressive* (AR) process excited by white noise. The power spectrum of *colored noise* does not have energy at all frequencies, and its power density is concentrated in a relatively narrow range of frequencies.

Consider a general system of the form:

$$\begin{cases} \tilde{y}[k] = F(\tilde{y}[k-1], \dots, \tilde{y}[k-n_y], u[k-1], \dots, u[k-n_u]) + v[k] \\ y[k] = \tilde{y}[k] + e[k], \end{cases} \quad (2.2)$$

where $v[k]$ represents the EE and $e[k]$ represents the OE.

A white noise output error problem consists of (2.2) with $v[k] = 0$ and $e[k]$ as *white noise*. The white noise OE becomes a colored EE noise. To exemplify this statement, consider the system S_{ex} :

$$S_{ex} : \begin{cases} \tilde{y}[k] = \theta_1 \tilde{y}[k-2] + \theta_2 u[k-1] + \theta_3 \tilde{y}[k-2]u[k-1] + v[k] \\ y[k] = \tilde{y}[k] + e[k]. \end{cases} \quad (2.3)$$

By isolating $\tilde{y}[k]$ from the second equation of (2.3) and replacing it in the first equation, the following NARMAX model is obtained:

$$\begin{aligned} y[k] &= \theta_1 y[k-2] + \theta_2 u[k-1] + \theta_3 y[k-2]u[k-1] \\ &\quad - \theta_1 e[k-2] - \theta_3 e[k-2]u[k-1] + e[k]. \end{aligned} \quad (2.4)$$

2.3 Model Structure Selection

One of the most used criteria for structure selection of NARMAX models is the ERR (BILLINGS; CHEN; KORENBERG, 1989). It evaluates the relevance of a regressor candidate regarding its capacity to explain the output variance. ERR-based algorithms are considered as PEM algorithms. For instance, in the FROE algorithm (BILLINGS; CHEN; KORENBERG, 1989) the model structure is incremented iteratively until a certain precision of one-step-ahead prediction

is achieved. In FROE, the model parameters are estimated via the OLS method. These orthogonalization techniques are designed in such a way that, at each step, the relevance of each regressor candidate can be assessed separately via the ERR:

$$[ERR]_i = \frac{\hat{\sigma}_i^2 \sum_{k=1}^N w_i^2[k]}{\sum_{k=1}^N y^2[k]}, \quad (2.5)$$

where w_i is the i^{th} auxiliary orthogonal regressor and $\hat{\sigma}_i$ is its corresponding estimated parameter. Regressors with higher ERR are included in the model. Some similar techniques are proposed in the literature to accomplish structure selection and parameter estimation at the same time (CHEN; BILLINGS; LUO, 1989), and they are referred to as OLS/ERR in this work.

However, Piroddi and Spinelli (2003) presented some limitations of the FROE algorithm . It is shown that, in the presence of certain characteristics of noise or of input signals, FROE may find incorrect or redundant models. In such cases, FROE is considered a local search technique with great probability of finding sub-optimal solutions (FALSONE; PIRODDI; PRANDINI, 2015), and models can be extremely imprecise and even unstable. Thus, to circumvent such limitations, Piroddi and Spinelli (2003) proposed to replace the ERR criterion by the *Simulation Error Reduction Ratio* (SRR) criterion, which is defined by the reduction of the *Mean Squared Simulation Error* (MSSE) normalized by the variance of the system output signal:

$$[SRR]_j = \frac{MSSE(\mathcal{M}_i) - MSSE(\mathcal{M}_{i+1})}{\frac{1}{N} \sum_{k=1}^N y^2[k]} \quad (2.6)$$

where \mathcal{M}_i is the model structure in the i^{th} iteration and \mathcal{M}_{i+1} is the candidate model for the subsequent iteration with the inclusion of the j^{th} regressor. Hence the model structure selection operation is performed based on the minimization of simulation error, and the algorithm is considered to be a SEM technique.

Another criterion for structure selection is the *Akaike's Information Criterion* (AIC) (AKAIKE, 1974), defined as:

$$AIC(\hat{\theta}) = -2\ln(f(y|\hat{\theta})) + 2p, \quad (2.7)$$

where y is the system output vector, $f(y|\hat{\theta})$ is a probability function, and p is the number of independently adjusted parameters to get $\hat{\theta}$. AIC can be used to compare a set of candidate models and select the number of terms. Models with the lowest AIC are optima.

2.4 Parameter Estimation

As NARX models are linear-in-parameters, the parameter estimation can be done by the *least squares* (LS) estimator as follows:

$$\hat{\theta}_{LS} = [\Psi^T \Psi]^{-1} \Psi^T y \quad (2.8)$$

where Ψ is the regressor matrix, y is the output data vector and $\hat{\theta}_{LS}$ represents the parameters estimated via LS.

For problems with output error, LS yields a biased parameter for NARX models. In these cases, it is recommended to identify a noise model. The NARX model becomes a NARMAX model, whose parameters cannot be estimated via the traditional LS estimator. A method that can be used to estimate these parameters is the *Extended Least Squares* (ELS) (AGUIRRE, 2015; YOUNG, 1968) method. For this case, consider that the prediction residuals ($\xi = y - \Psi \hat{\theta}_{LS}$) can be modeled as:

$$\xi[k] = c_i v[k-i] + v[k], \quad (2.9)$$

where $v[k]$ is white noise and c_i is the parameter of the corresponding noise model term. The $v[k-1]$ term is included in the regressor matrix:

$$\Psi^* = \begin{bmatrix} & v[k-1] \\ & v[k] \\ \Psi & v[k+1] \\ & \dots \\ & v[k+N-2] \end{bmatrix}, \quad (2.10)$$

which is called an *extended matrix*. A new vector of parameters is defined as $\theta^* = \begin{bmatrix} \theta & c_i \end{bmatrix}$, and its values are to be estimated via LS. As $v[k]$ is unknown, the process must be iterative, and the residual vector must be calculated at each iteration.

2.5 Model Validation

In this step, the model's capability to generalize is assessed. Aguirre (2015) highlights the importance of using different data sets to identify and validate the model. There are many criteria that can be used to assess the model's adequacy. The most commonly used measure is the *Mean Squared Error* (MSE), that is defined by:

$$MSE = \frac{1}{N} \sum_{k=1}^N (y[k] - \hat{y}[k])^2, \quad (2.11)$$

where $y(k)$ is the measured data, $\hat{y}(k)$ is the model's estimated values. Other relevant measures are: the *Root Mean Squared Error* (RMSE), defined by:

$$RMSE = \frac{\sqrt{\sum_{k=1}^N (y[k] - \hat{y}[k])^2}}{\sqrt{\sum_{k=1}^N (y[k] - \bar{y})^2}}, \quad (2.12)$$

where \bar{y} is the mean value of the signal $y(k)$; and the *Mean Absolute Percentage Error* (MAPE) measure:

$$MAPE = \frac{100}{N} \sum_{k=1}^N \left| \frac{y[k] - \hat{y}[k]}{y[k]} \right|. \quad (2.13)$$

2.5.1 Simulation Methods

Note that NARX/NARMAX models can be simulated in different ways. They yield prediction error estimations that have different properties and capabilities that can be used to assess model generalization. Figure 2.1 presents a graphical representation of these methods.

2.5.1.1 One-step-ahead Prediction

Predictions are obtained using the available information up to and including step $[k - 1]$. Therefore, the model's estimate is a function of lagged input and system output, such that

$$\hat{\mathbf{y}}[k] = F(\mathbf{y}[k - 1], \dots, \mathbf{y}[k - n_y], \mathbf{u}[k - 1], \dots, \mathbf{u}[k - n_u]) = \Psi_{yu}^T \hat{\boldsymbol{\theta}}, \quad (2.14)$$

where \mathbf{y} is the system output data vector, Ψ_{yu}^T the regressor matrix and $\hat{\boldsymbol{\theta}}$ the estimated parameters. The subscript yu indicates that the matrix is composed of output and input regressors. As a general rule, one-step-ahead predictions are not good indicators of a model's capability to explain system dynamics (AGUIRRE, 2015). The error that this estimator yields is called the *Prediction Error* (PE).

2.5.1.2 Free-run Simulation

The free-run simulation is a recursive estimator that predicts infinite steps ahead using the input signal and the model's previous estimated values. Thus, from a certain initial condition ($\hat{\mathbf{y}}_0 = \mathbf{y}_0$), the estimation is defined by:

$$\hat{\mathbf{y}}[k] = F(\hat{\mathbf{y}}[k - 1], \dots, \hat{\mathbf{y}}[k - n_y], \mathbf{u}[k - 1], \dots, \mathbf{u}[k - n_u]) = \Psi_{\hat{y}u}^T[k - 1] \hat{\boldsymbol{\theta}}, \quad (2.15)$$

for $k = \max(n_y, n_u), \dots, N$, where $\Psi_{\hat{y}u}^T[k - 1]$ is a regressor vector and the subscript $\hat{y}u$ indicates that it is composed of output estimation and input regressors, $\max(n_y, n_u)$ refers to the initial condition vector size. This estimator is considered a good method for assessing the model's capability to explain observations (AGUIRRE, 2015). The error that this estimator yields is called the *Simulation Error* (SE).

2.5.1.3 Multistep-ahead Prediction

In this case, the free-run simulation is truncated to K steps ahead and rebooted soon after with a one-step moving horizon (more details in (AGUIRRE, 2015; RIBEIRO et al., 2020)). Thus,

the estimation is defined by:

$$\begin{cases} \tilde{y}[i] = \Psi_{\tilde{y}_{ku}}^T [i-1] \hat{\theta}, & \text{for } i = k-K, \dots, k \text{ and } \tilde{y}_0 = y_{k-K} . \\ \hat{y}[k] = \tilde{y}[k], & \text{for } k = \max(n_y, n_u) + K, \dots, N , \end{cases} \quad (2.16)$$

Figure 2.1(c) presents a clear graphical representations of this method. Note that there is a K -steps free-run simulation and only the K -th value is used as output prediction. Then, a one-step-ahead moving horizon is applied and the K -steps free-run simulation is performed again. This process is repeated while there are available original data (system's output and input).

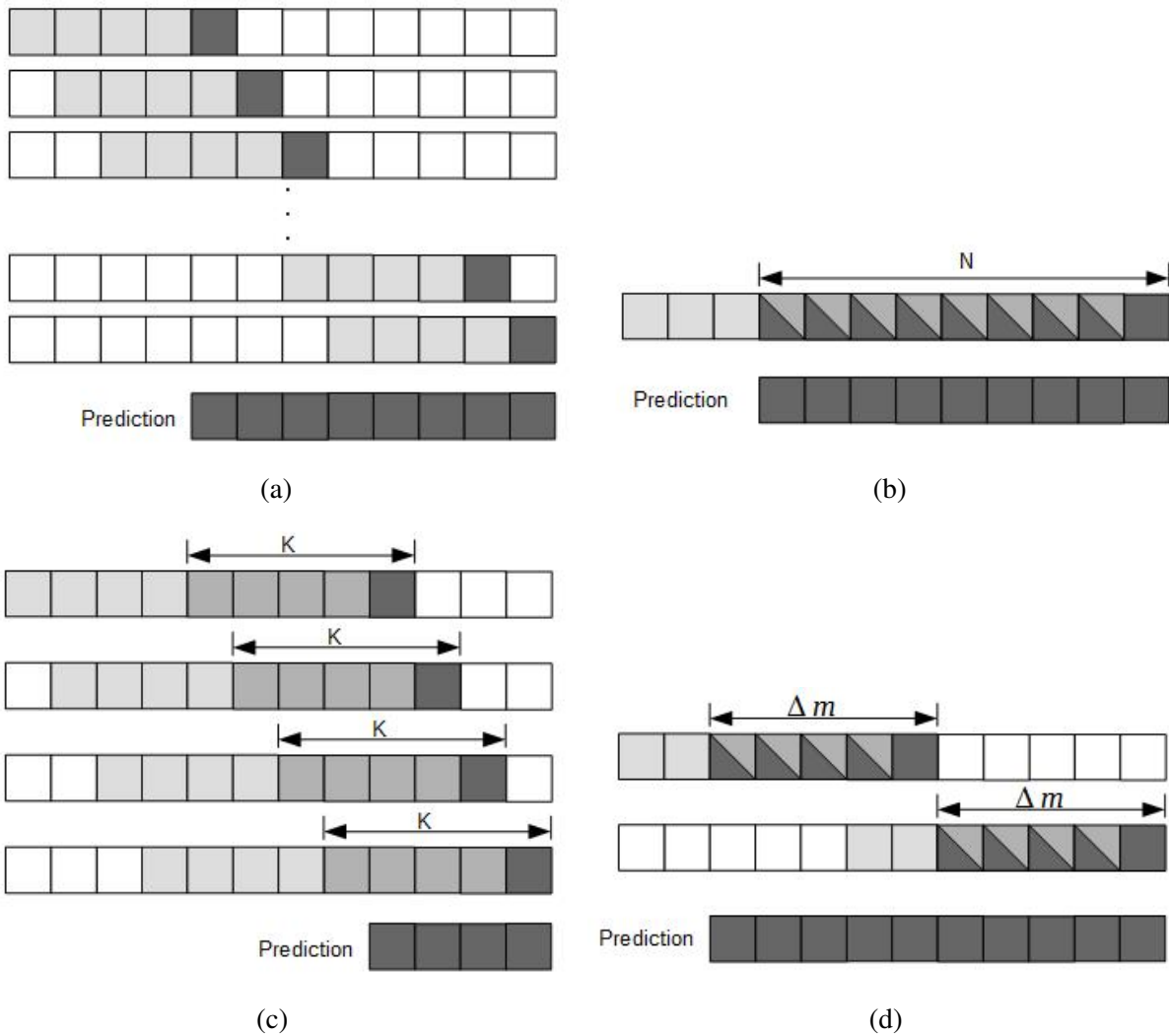
2.5.1.4 Multiple Shooting

For *multiple shooting*, the data are split into M intervals of size Δm +initial conditions, each one with its own initial condition y_0^i , for $i = 1, \dots, M$. The estimation $\hat{y}^i[k]$ is computed over the interval Δm :

$$\begin{cases} \hat{y}^i[m] = \Psi_{\hat{y}_{iu}}^T [m-1] \hat{\theta}, & \text{for } m = 1, \dots, \Delta m \text{ and } i = 1, \dots, M \\ \hat{y}[k] = \bigcup_{i=1}^M \hat{y}^i \end{cases} \quad (2.17)$$

This differs from the multistep-ahead predictor due to the moving horizon of Δm steps. Figure 2.1(d) presents a clear graphical representation of this method. Note that the data is split into two intervals of size Δm +initial conditions. A free-run simulation is performed for each of them and the predictions are concatenated. Ribeiro et al. (2020) compares the smoothness of the error space yielded by free-run simulation, multistep-ahead prediction and multiple shooting during the parameter optimization process. Multiple shooting allows optimization problems to be solved that would be infeasible in a free-run simulation setting.

Figure 2.1 – Simulation Methods. (a) one-step-ahead prediction, (b) free-run simulation, (c) multi-step-ahead prediction and (d) multiple shooting. Light gray represents original data used as initial conditions, medium gray represents predicted data used as initial conditions and dark gray represents the model output.



(Source: Author)

3 EVOLUTIONARY ALGORITHMS

The EAs, based on natural processes, build computational models capable of solving problems. There are a wide variety of algorithms which simulates the evolution of species through the natural phenomena of selection, mutation and reproduction that occur in a given population of individuals. The computational representation of these phenomena is called *genetic operator* (EIBEN; SMITH et al., 2003; LINDEN, 2008).

There are two points of extreme significance in EA: the representation, or codification, of individuals and the evaluation function (or cost function). The codification can be understood as the individual genotype, also called chromosome, and the evaluation function yields the individual phenotype, also called fitness. The EAs are optimization algorithms, in which an objective function is minimized or maximized. Hence the evaluation function must be chosen carefully. A well chosen function must represent all the knowledge one has about the problem, including its constraints. An optimization problem may be defined as:

$$\begin{aligned} \min J(\theta) \\ \text{subject to: } \lambda_{1i} \leq \theta_i \leq \lambda_{2i}, \quad i = 1, 2, \dots, m, \end{aligned} \tag{3.1}$$

where m is the number of parameters θ to be adjusted by the optimization algorithm and λ constrains the search space to the feasible set. The evaluation function (or cost function) translates the information the chromosome encodes into a numeric value that is the measure of the chromosome quality.

Generally, an EA exhibits standard behaviors: it begins with an initial population of random individuals (chromosomes), and at each generation (main loop), the best solutions are sought (*selection*), and then combined through *recombination/reproduction/crossover* and changed through *mutation* to potentially generate better individuals. The Algorithm 1 summarizes this process.

In many problems, the quality of the solution depends on several and possibly conflicting objectives. A multi-objective optimization problem can be defined as:

$$\begin{aligned} \min J_1(\theta), J_2(\theta), \dots, J_n(\theta) \\ \text{subject to: } \theta \in \Theta, \end{aligned} \tag{3.2}$$

Algorithm 1: EVOLUTIONARY ALGORITHMS

```

1  $P :=$  initial population
2 evaluate individuals ( $P$ )
3 while it doesn't satisfy stop condition do
4    $P' =$  parents selection ( $P$ )
5   apply recombination and mutation ( $P'$ )
6   evaluate individuals ( $P'$ )
7    $P =$  select new generation ( $P, P'$ )
8 end

```

where n is the number of objective functions and Θ is the feasible set. In these cases, *Multi-Objective Evolutionary Algorithms* (MOEA) can be used. An MOEA algorithm works with the concepts of *dominance* and of *Pareto-optimal set*.

Consider the optimization problem defined in (3.2). The solution $J(A) = \{J_1(A), J_2(A), \dots, J_n(A)\}$ is said to be dominant over $J(B)$ if each $J_i(A)$, for $i = 1, \dots, n$, is lesser than or equal to the respective $J_i(B)$, and there is at least one $J_i(A)$ lesser than the respective $J_i(B)$ (EIBEN; SMITH et al., 2003):

$$A \succ B \Rightarrow \forall i \in \{1, 2, \dots, n\}, J_i(A) \leq J_i(B), \text{ and } \exists j \in \{1, 2, \dots, n\}, J_j(A) < J_j(B). \quad (3.3)$$

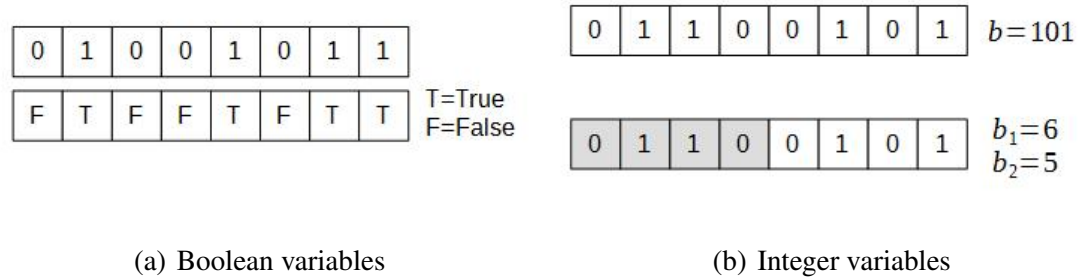
The *Pareto-optimal set* consists of all non-dominated solutions. These are solutions which quality cannot be improved in any objective without affecting negatively one of the others.

Note that EAs use stochastic factors to initialize and evolve the population. Therefore, they are heuristics that do not guarantee the best solution. EAs are search techniques classified as "Random-Guided Techniques", which means, despite the random components, they use the current state to guide the search (LINDEN, 2008). Among the EAs, we highlight the GA and the GP. They are introduced in the next sections.

3.1 Genetic Algorithms

GAs were introduced by Holland (1975) as a way to study the adaptation and evolution of species. However, they have become a powerful tool as a method of function optimization. De (1975) together with Goldberg and Holland (1988) define the classic genetic algorithm, or *Simple*

Figure 3.1 – Binary Representation of Boolean variables (a) and numerical variables with a chromosome representing one and two variables (b).



(Source: Author)

Genetic Algorithm (SGA). It uses *binary* representation, *roulette wheel* selection and *one-point crossover* reproduction. SGA focuses on crossover as a means to generate new solution candidates for a *maximization* problem. We use SGA as basis to exemplify GAs representation and operators.

3.1.1 GA Representation

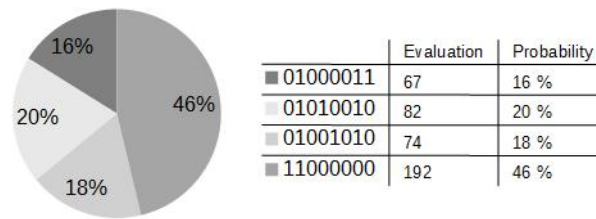
In the SGA, the individuals consists of a sequel of binary numbers. Thus the representation is named *binary representation*. As an analogy to the biological chromosome, each indivisible part of the representation is called a *gene*. Figure 3.1 presents some examples of binary individuals.

When the binary number represents Boolean variables, the genotype-phenotype mapping is intuitive: the chromosome length is defined as the number of items to be classified as *true* or *false*. On the other hand, when the binary number represents integers or real numbers, the chromosome length depends on the search space and the desired precision. To represent more than one integer or real number in a single chromosome, one must select which genes are responsible for code each variable. The conversion of a k -bit binary number coded in the chromosome to a real number is possible after setting the lower and the upper limits of the representation – $[lower, upper]$. The precision is given by $\frac{upper - lower}{2^k - 1}$ and the conversion operation is given by (EIBEN; SMITH et al., 2003):

$$real = lower + \frac{upper - lower}{2^k - 1} \cdot b, \quad (3.4)$$

where b is the integer corresponding to the binary number.

Figure 3.2 – Virtual roulette wheel in a maximization problem: each individual gets a selection probability equals to its percentage of the evaluations sum.



(Source: Author)

The binary representation is inadequate for some problems due to certain limitations. For example, it is difficult to handle multiple dimensions of continuous variables, mainly when great precision is required. In this case, the most appropriate representation is the *real representation* (HERRERA; LOZANO; VERDEGAY, 1998), in which each gene represents exactly one variable to be optimized. The real representation uses the maximum precision the computer is capable to offer. Depending on the specificity of each problem, more adequate and possibly more powerful representations can be used (see more in Linden (2008)).

3.1.2 GA Selection Operators

The genetic operator responsible for select the fittest individuals is based on the process of natural selection. Therein, the fitter the individual the greater the chance of survival. Note that individuals with poor evaluations should not be completely discarded from population, since they may carry important genetic information to generate fitter children.

In SGA, the *roulette wheel* method is used as selection operator Holland (1975). In this method, the chance of an individual to be selected equals its own percentage of a virtual roulette. The SGA maximizes the objective function, hence the individual selection probability corresponds to its fitness parcel in the sum of all individuals evaluations. Figure 3.2 presents an example of this method in a hypothetical population: it consists of a virtual roulette that represents the sum of the fitnesses of four individuals (67, 82, 74 and 192). The probabilities of each individual to be selected are 16%, 20%, 18% and 46%. Some disadvantages of the *roulette wheel* method are highlighted by Eiben, Smith et al. (2003):

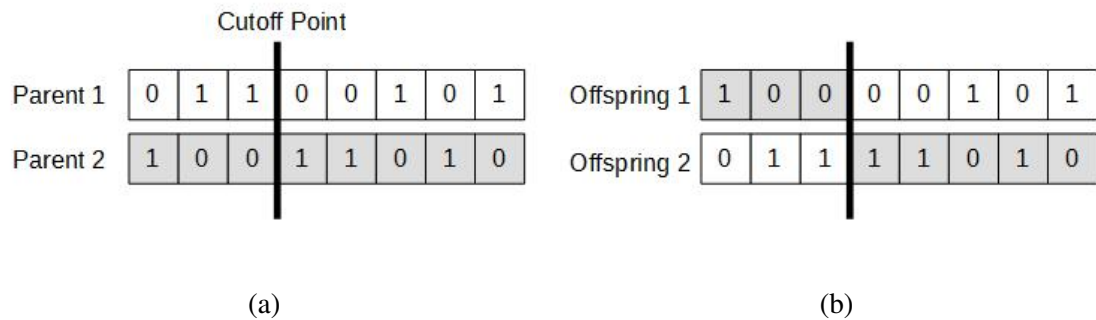
- individuals with a much higher fitness than the remainder of the population tend to direct the search process to themselves. As consequence, they are the only ones to be reproduced in most of the generations. Hence the population is composed of increasingly similar individuals throughout time. In this case, there is not enough genetic diversity for proper evolution of the population.
- if individuals fitnesses are close to each other, there is almost no *selection pressure*, which is the greater probability of survival of the individuals with the best fitness. In this case, the selection is almost uniformly random and the population's average fitness grows very slowly.
- the roulette wheel behaves differently when the evaluation function is shifted. For example, consider that f is the evaluation function. The probability of selecting individuals with evaluation $f + x$, where x is a real number much higher than the average fitness of the population, are basically the same.

One way to circumvent these disadvantages is to use selection operators which does not take the absolute best fitness into consideration, but the best relative fitness between individuals. Such as the *Tournament* operator, in which a number k of individuals is chosen at random from the population. They are compared to each other and the best of them is chosen to be a parent. Since two parents are required for reproduction, this procedure is repeated. The magnitude of the fitnesses difference is not relevant. Therefore, the selection pressure on the population remains constant.

3.1.3 GA Recombination Operators

After parenting selection, their genotypes are recombined to generate two offspring. It is expected that, at each generation, some children will have their characteristics improved. SGA uses the simplest recombination operator: the *one-point crossover* operator. A cutoff point is selected at random among the genes of a chromosome. It divides the parent individuals into two parts: one on the left and one on the right side of the cutoff point. The offspring are produced by concatenating the left part of one parent with the right of the other and vice-versa. Figure 3.3 presents an example of this genetic operator.

Figure 3.3 – One Point Cross-over: a cutoff point is randomly selected and the parents exchange genetic material to generate the offspring.



(Source: Author)

Regarding real representations, different *crossover* operators can be used. Such as: the *flat crossover* operator (RADCLIFFE, 1991), in which the value of the gene i in the offspring chromosome h is chosen randomly from the interval $[c_i^1, c_i^2]$, where c_i^j is the gene i of the parent chromosome j ; the *arithmetical crossover* operator, in which the offspring chromosomes h^1 and h^2 are generated according to the relationships $h_i^1 = \lambda c_i^1 + (1 - \lambda)c_i^2$ and $h_i^2 = \lambda c_i^2 + (1 - \lambda)c_i^1$, where λ is chosen from the interval $[0, 1]$ and may vary over generations (EIBEN; SMITH et al., 2003; LINDEN, 2008); and the *BLX - α crossover* operator, also called *blend crossover*, where h_i is chosen randomly in the interval $[c_{min} - I \cdot \alpha, c_{max} + I \cdot \alpha]$, where $c_{max} = \max(c_i^1, c_i^2)$, $c_{min} = \min(c_i^1, c_i^2)$ and $I = c_{max} - c_{min}$. The *BLX-0.0 crossover* is the same as *flat crossover* (see more at Herrera, Lozano and Verdegay (1998)).

3.1.4 GA Mutation Operators

The offspring may change through mutation given a mutation probability. The mutation process is always stochastic and the probability of applying the mutation operator should be low. If it is too high, the algorithm behaves as a random search for solutions (*random walk*).

SGA uses *uniform mutation* operator, the most common mutation operator for binary chromosomes. In this operator, each gene can flip its value (from 0 to 1 and from 1 to 0) depending on an independent mutation probability. Note that, there are two mutation probabilities: one to determine whether the operator is applied or not, and other to determine whether each gene is flipped or not. Figure 3.4 presents an example of *uniform mutation*.

Figure 3.4 – Uniform mutation: a random number is generated for each gene. If it is smaller than the independent mutation probability (5% in this example), that specific attribute is flipped.

Individual	0	1	1	0	0	1	0	1
	0.73	0.04	0.91	0.01	0.28	0.56	0.03	0.17
Mutated Individual	0	0	1	1	0	1	1	1

(Source: Author)

The mutation operator for real representations consists of replacing the value of a gene, or genes, by a value chosen randomly within some range $[L, U]$, where L is the lower limit and U is the upper limit. Eiben, Smith et al. (2003) discriminates two types of mutation operators for real representations according to the probability distribution from which the new values are taken: the *uniform mutation*, in which the new values are chosen uniformly at random within the range $[L_i, U_i]$, where L_i and U_i are fixed values for each gene i ; and the *non-uniform mutation*, in which the new values are taken from a Gaussian distribution with mean equals the current state of the gene and with standard deviation specified by the user.

3.1.5 Selecting new generation

Assuming that the population evolves in an environment of limited resources, not all individuals among the population of parents and offspring will be present in the next generation. The simplest way to select survivors of a generation is to replace the entire population (parents) with the generated offspring. This "new generation selection operator" is named *population module* (LINDEN, 2008).

The population module can also use the *elitism* scheme, in which the individuals with the best fitness are maintained in the population of the next generation.

3.1.6 Genetic Algorithms in System Identification

Li and Jeon (1993) use GA to select the most statistically significant terms of the polynomial NARX model to avoid over-parameterization. Once the values of n_y , n_u and l (maximum delay in y , in u and non-linearity degree, respectively) are chosen, the number of candidate terms (n) for the

model can be determined as:

$$n = M + 1, \quad (3.5)$$

where

$$M = \sum_{i=1}^l n_i$$

$$n_i = \frac{n_{i-1}(n_y + n_u + i - 1)}{i}, n_0 = 1.$$

The binary representation is used. The variables are Boolean, thus each gene represents the inclusion or exclusion of one possible model term. The value 1 means the inclusion of that term in the model and the value 0 means its exclusion. For example, if $n_y = n_u = 3$ and $l = 3$, there are 84 candidate terms: $\theta_0, y[k-1], y[k-2], y[k-3], u[k-1], u[k-2], u[k-3], y[k-1]^2, y[k-2]^2, \dots$. Therefore, the system $y[k] = \theta_0 + \theta_1 u[k-1] + \theta_2 y[k-1]^2$ is represented by $A = 100010010\dots 0$. The parameters are estimated via the LS method. Li and Jeon (1993) determine the fitness of individuals in the population using the one-step-ahead MSE, so that

$$f = \frac{1}{1 + \text{MSE}}. \quad (3.6)$$

Chen et al. (2007) introduce an enhanced evaluation function to identify NARX dynamic models by means of GA. The binary representation is used. Each gene is a *sub-string* of the chromosome corresponding to a model parameter. The maximum lags (n_y and n_u) and the number of non-linear terms in their simplest form (s) are determined during the search process. Thus the exact number of parameters is not known. Therefore, there are empty genes (buffers) to be used depending on the values of n_y, n_u and s , which are limited to not exceed the amount of genes reserved for the corresponding parameters in the model representation.

The chromosome has the following structure:

$$\begin{aligned} & \theta_1^y \quad \theta_2^y \quad \dots \quad \theta_{n_y}^y \quad \text{buffer} \quad \text{buffer} \quad \theta_1^u \quad \theta_2^u \quad \dots \quad \theta_{n_u}^u \quad \text{buffer} \quad \text{buffer} \quad \Rightarrow \\ \Rightarrow & \theta_1^2 \quad \theta_2^2 \quad \dots \quad \theta_s^2 \quad \text{buffer} \quad \text{buffer} \quad n_y \quad n_u \quad s \end{aligned}$$

The genes corresponding to the model parameters are floating point numbers that follow the encoding:

1	0101	1011	1110	1101	0101	0	0000	0011
coefficient sign 0 → + 1 → -	a_4	a_3	a_2	a_1	a_0	exponent sign 0 → + 1 → -	k exponent Digits	
	coefficient digits after decimal point							

where θ_i^x is calculated by:

$$\theta_i^x = \pm \left(\frac{1}{16} \cdot a_4 + \frac{1}{16^2} \cdot a_3 + \frac{1}{16^3} \cdot a_2 + \frac{1}{16^4} \cdot a_1 + \frac{1}{16^5} \cdot a_0 \right) \cdot 2^{\pm k}. \quad (3.7)$$

Regarding the genes corresponding to the values of n_y , n_u and s , they are integers that follow the encoding:

000	000	000	011
b_3	b_2	b_1	b_0

in which the integer value is evaluated as

$$n_x = 8^0 \cdot b_0 + 8^1 \cdot b_1 + 8^2 \cdot b_2 + 8^3 \cdot b_3. \quad (3.8)$$

The evaluation function proposed by Chen et al. (2007) seeks to better describe the main properties of the model for the best performance of the GA process. Based on the general rule, the normalized one-step-ahead prediction error is used:

$$e = \sqrt{\frac{\sum_{k=1}^n (y[k] - \hat{y}[k])^2}{\sum_{i=k}^n y[k]^2}} \quad (3.9)$$

where y is the desired output and \hat{y} is the model response. However, an evaluation function based only on this error is unable to justify the model structure (CHEN et al., 2007).

If there are non-modeled system dynamics, these dynamics have their influence on the prediction residues $\xi(i) = y(i) - \hat{y}(i)$. Let the correlation function between two generic data vectors a

and b be given by:

$$R_{a,b}(\tau) = \frac{1}{n-\tau} \sum_{i=0}^{n-\tau} a[i]b[i+\tau]. \quad (3.10)$$

Chen et al. (2007) define the evaluation function:

$$f(y, u, \xi) = \frac{1}{e + \rho_{y,\xi} + \rho_{u,\xi} + \rho_{\xi,\xi y} + \rho_{\xi,y^2} + \rho_{\xi^2,y^2}}, \quad (3.11)$$

where

$$\rho_{a,b} = \sqrt{\frac{\sum_{\tau=1}^{n-1} R_{a,b}^2[\tau]}{\sum_{i=1}^n y^2[i]}}.$$

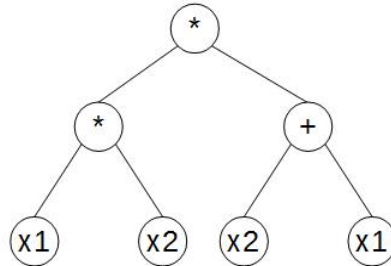
Aguirre, Barbosa and Braga (2010) implemented the GA to estimate parameters in NARX/-NARMAX models. They use mono and multi-objective approaches with real representation, in which each single *locus* is a gene that represents a parameter to be estimated. The individuals are selected via *stochastic universal sampling* (BAKER, 1987), a *heuristic crossover* is implemented (it returns a offspring closer to the best parent) and *Gaussian mutation*, in addition to the use of an elitist strategy. Barbosa et al. (2011) also implemented the GA to estimate parameters for hydraulic pumping models. They use a bi-objective approach with cost function composed of the acsSE in dynamical data and the error in the system static curve (*gray-box identification*).

3.2 Genetic Programming

Unlike the GA, the GP is an evolutionary algorithm where the structure of the solution is not specified in advance. The individuals are computer programs that randomly evolve into new programs. In this sense, the GP may be considered as a *machine learning* technique and not as an optimization technique (EIBEN; SMITH et al., 2003).

The problem to be solved by the GP consists of building a function or a program that makes the mapping between input and output data. The chromosome is assessed by executing the program and by determining its error relating to the desired output.

Figure 3.5 – Tree representation. The function (or program) $f(x_1, x_2) = x_1 \cdot x_2 \cdot (x_2 + x_1)$ is represented as a tree.



(Source: Author)

3.2.1 GP Representation

The most common representation used in the GP is the *tree* representation. In this representation, from a root node, the tree is divided into several branches in which the internal nodes have arithmetic functions (+, -, *, /, max, ...) and the terminals, also called leaves, have variables and constants. As a result, the tree representation hierarchically synthesizes a mathematical function. Figure 3.5 presents the tree representation of a function of two variables $f(x_1, x_2) = x_1 \cdot x_2 \cdot (x_1 + x_2)$, and the root node multiplies two branches: $(x_1 \cdot x_2)$ and $(x_1 + x_2)$.

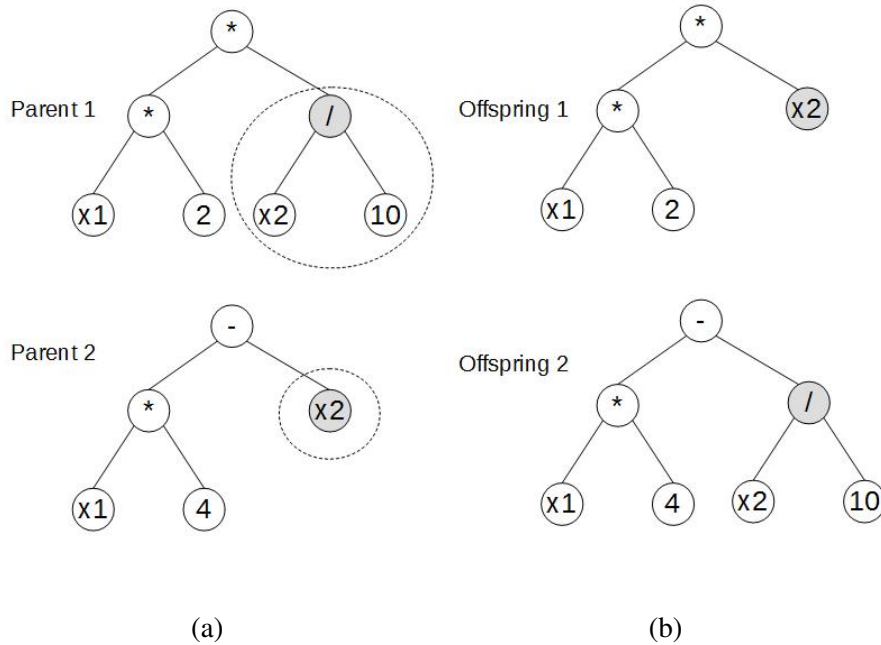
3.2.2 GP Selection Operators

According to Poli et al. (2008), the most common selection operator used in the GP is the *tournament selection* operator, in which a number of individuals are chosen randomly from the population. They are compared to each other and the best of them is chosen to be a parent.

3.2.3 GP Recombination and Mutation Operators

In the GP, the offspring are usually generated by recombination *or* mutation, rather than recombination followed by mutation. In this sense, given a crossover probability p_r and a mutation probability p_m , if there is recombination, there is no mutation and vice-versa. Koza (1992) advises the use of the GP with no mutation at all. Other studies point out that approaches with only crossover may have inferior performance (LUKE; SPECTOR, 1997).

Figure 3.6 – GP *subtree crossover* operator. The parent individuals $2 \cdot x_1 \cdot \frac{x_2}{10}$ and $4 \cdot x_1 - x_2$ exchange subtrees as genetic materials and generate two offspring: $2 \cdot x_1 \cdot x_2$ and $4 \cdot x_1 - \frac{x_2}{10}$.



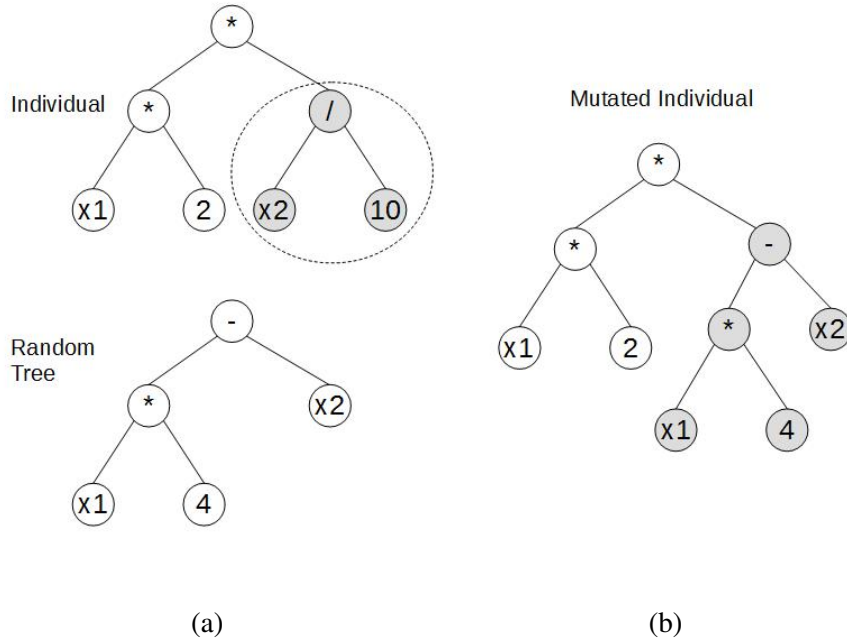
(Source: Author)

The most common operators of reproduction and mutation are the *subtree crossover* and the *subtree mutation*, respectively. In the former, two nodes are chosen randomly in the parent individuals and the sub-trees formed from these points are interchanged between the chromosomes. Figure 3.6 presents an example of such operation. In this case, two parents exchange subtrees as genetic materials and generate two offspring:

$$\begin{aligned} \text{parent1}(x_1, x_2) &= 2 \cdot x_1 \cdot \frac{x_2}{10} \quad \text{and} \quad \text{parent2}(x_1, x_2) = 4 \cdot x_1 - x_2 \\ \text{offspring1}(x_1, x_2) &= 2 \cdot x_1 \cdot x_2 \quad \text{and} \quad \text{offspring2}(x_1, x_2) = 4 \cdot x_1 - \frac{x_2}{10}. \end{aligned}$$

In the latter, a mutation point is chosen at random. The existing subtree from that point is removed and a new randomly generated tree is inserted in its place. A subtree mutation example is presented in Figure 3.7. The individual $f(x_1, x_2) = 2 \cdot x_1 \cdot \frac{x_2}{10}$ has the branch composed of $\frac{x_2}{10}$ replaced by the random tree $r(x_1, x_2) = 4 \cdot x_1 - x_2$. As result the mutated individual $f_m(x_1, x_2) = 2 \cdot x_1 \cdot (4 \cdot x_1 - x_2)$ is generated.

Figure 3.7 – GP *subtree mutation* operator. The individual $2 \cdot x_1 \cdot \frac{x_2}{10}$ has the branch composed of $\frac{x_2}{10}$ replaced by the random tree $4 \cdot x_1 - x_2$.



(Source: Author)

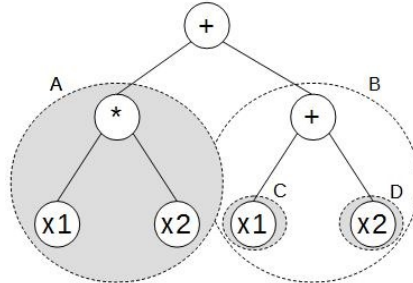
3.2.4 Genetic Programming in System Identification

The use of GP in System Identification has the advantage of promoting automatic structure selection. There is no need to define the degree of non-linearity, the maximum lags and the functional form in advance.

Madar, Abonyi and Szeifert (2005) address the structure selection problem of polynomial NARX models via the GP. To represent polynomial models, internal nodes are limited to the set of operators $F = \{+, *\}$ and terminals to the set of variables $T = \{x_1, x_2, \dots, x_m\}$, where $x \in \{y[k-1], y[k-2], \dots, y[k-n_y], u[k-1], u[k-2], \dots, u[k-n_u]\}$. In addition, a syntactic rule is established so that the internal nodes below a node with operator $*$ are switched to $*$, thus there is no addition operator after a multiplication operator, i.e., consider the individual in Figure 3.8: if the root was a multiplication operator ($*$) the root of the subtree B would be switched from $(+)$ to $(*)$.

A decomposition method extracts the regression terms from the individual. The subtrees are selected by starting from the root node, following the branches until reaching a nonlinear node (multiplication node). For example, see the tree represented in Figure 3.8. The root node is the

Figure 3.8 – Decomposition of a tree in regression terms. Starting from the root node, follow the branches until reach a nonlinear node or a terminal. The individual $(x_1 \cdot x_2) + (x_1 + x_2)$ yields the regressors $x_1 \cdot x_2$, x_1 and x_2 .



(Source: Author)

operator $+$, thus it is possible to decompose this tree into two sub-trees A and B . The root of the subtree A is a nonlinear function (operator $*$), thus this entire subtree is a regression term ($F_1 = x_1 \cdot x_2$). The root of the subtree B is the $+$ operator, therefore it is decomposed into two subtrees C and D that are terminals, hence they are the regression terms ($F_2 = x_2$ and $F_3 = x_3$). The selected regressors yield a linear-in-the-parameters model: $y = \theta_0 + \theta_1 x_1 x_2 + \theta_2 x_2 + \theta_3 x_3$. Thus the parameters are estimated via the LS method.

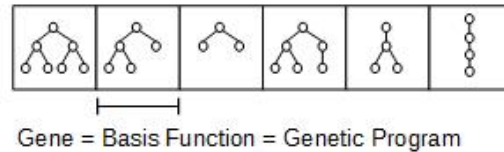
The authors use the following evaluation function, that is maximized:

$$f_i = \frac{r_i}{1 + \exp(a_1(L_i - a_2))} \quad (3.12)$$

where f_i is the fitness of the individual i , r_i is the correlation coefficient between the model output and the desired output, L_i is the tree length (number of nodes), and a_1 and a_2 are parameters of a penalty function design to avoid overparameterized models.

The resultant model may contain terms that contribute very little to its accuracy, therefore MADAR; ABONYI; SZEIFERT suggest the use of OLS/ERR to improve the algorithm performance. The regressor that are less significant (low ERR) are removed from the model. This "pruning" method is applied before the evaluation step. For example, suppose that $ERR_{F_1} \lll ERR_{F_2}$, $ERR_{F_1} \lll ERR_{F_3}$ and $ERR_{F_1} + ERR_{F_2} + ERR_{F_3} = 1$, where F_1 , F_2 and F_3 are the regressors generated by the subtrees A , C and D from Figure 3.8, respectively. Hence the subtree A is eliminated and the final model is $y = \theta_0 + \theta_1 x_2 + \theta_2 x_3$.

Figure 3.9 – MGGP individual. It is a sequel of genetic programs as basis functions. The linear combination of these programs represents the mathematical model.



(Source: Author)

3.3 Multi-Gene Genetic Programming

The MGGP was introduced by Hinchliffe et al. (1996), under the name of *Multi Basis Function Genetic Programming* (MBF-GP). It follows the methods already established in Systems Identification, in which the models are constructed by combining a number of functions. The MGGP can be represented as the combination of separated basis functions:

$$g(\varphi, \Theta) = \sum_{i=1}^m \theta_i g_i(\varphi) \quad (3.13)$$

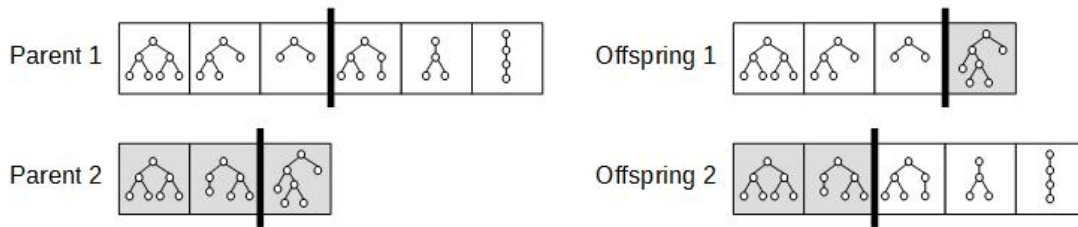
where m is the number of basis functions, g_i represents individual functions and θ_i represents the model parameters. Ljung (1999) shows how this expansion can be used to represent and analyze virtually any nonlinear modeling technique. For example, a *feedforward* neural network is the combination of a number of *log-sigmoid* or *hyperbolic tangent* functions. Figure 3.9 presents the representation of a generic MGGP individual as an example.

Hinchliffe (2001) points out possible weaknesses of these traditional modeling techniques: they are restricted to the use of a particular type of basis function and the general model structure (for instance neural network architecture) is fixed before starting the stage of parameter optimization. In this sense, the MGGP is very attractive because it does not have a fixed structure and can vary the number of basis functions during the evolutionary process.

3.3.1 Modification of the Genetic Operators

The main difference between the MGGP genetic operators and those of the standard GP is the recombination operators. In MGGP, they are referred to as *high-level crossover* (Figure 3.10) and *low-level crossover* (Figure 3.11). In the former, the genetic materials are exchanged as entire

Figure 3.10 – MGGP High-Level Crossover. A cutoff point is determined for each parent individual, that exchanges a set of basis functions as genetic materials.



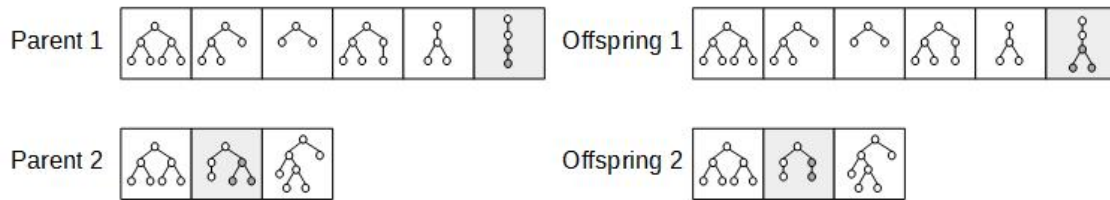
(Source: Author)

basis functions; that is, the MGGP parents exchange their GP individual in a way similar to GA one-point crossover. Note that in *high-level crossover*, the resultant offspring can have sizes different from those of each other and even from those of their parents. This occurs because the cutoff point is chosen for each parent. In this sense, the MGGP algorithm works with fluctuating individual sizes. In the latter, the genetic materials are exchanged as subtrees of the basis functions; that is, only one gene is randomly chosen from each parent individual, and their GPs (or basis functions) exchange genetic materials as in the GP *subtree crossover operator*.

3.4 Multi-Gene Genetic Programming in System Identification

Hinchliffe and Willis (2003) use the MBF-GP as a means to develop an automatic function selection procedure for dynamic models. What differs their work from the previous papers presented until this point is that HINCHLIFFE; WILLIS propose an algorithm that automatically discovers the appropriate lag terms required to build an accurate model without predetermining the set of terminal variables $T = \{x_1, x_2, \dots, x_n\}$. That is, instead of the GP construct dynamic basis functions by providing it with a terminal set containing shifted process inputs and outputs, it is provided a back-shift operator, q^{-1} , to the function set (or node set). Thus the terminal set consists solely of the process input and output signals shifted by a single time sample. The single shifting is justified for the terminal alone must be a valid model term. As an example, the term $\hat{y}[k] = u[k-6] - y[k-2]$ is represented in the GP as $f = q3(q2(u1)) - q1(y1)$, where $q3$ stands for q^{-3} that shifts $u(k-1)$ by three samples, and $u1$ is the single sample shifted input terminal. The authors worked with rational models and the function set is composed of $\{+, -, /, *, \text{POW}, \text{SQRT}$,

Figure 3.11 – MGGP Low-Level Crossover. Parent individuals exchange basis function's subtrees as genetic materials.



(Source: Author)

SQR, EXP, LOG, Back-shift operators: q_0, q_1, q_2, q_3 }. The model parameters are optimized using the *Recursive Least Squares* (RLS) method (AGUIRRE, 2015). The results of the MGGP algorithm is compared to the results of the *Filter Based Neural Network* (FBNN) (Willis et al., 1991), which had the benefit of time delay removal via correlation analysis to identify the time delay before network training. There was little difference between the two techniques in terms of model accuracy.

Ghareeb and Saadany (2013) implemented the MGGP in a *Short Term Load Forecasting* (STLF) problem in power system operation. A data set of the Egyptian electrical network is used. This data set includes the daily maximum temperature, the daily minimum temperature and the corresponding actual peak load. The MGGP has successfully forecasted the future load with high accuracy compared to that of a *Radial Basis Function* (RBF) network and of a standard single-gene GP.

Mehr and Kahya (2017) proposed a Pareto-optimal *Moving Average* (MA) MGGP for daily streamflow prediction. The data is pre-processed by a MA filter, that diminishes the lagged prediction effect of stand-alone MGGP models. A multi-objective framework is used to select a parsimonious model through an interactive complexity-efficiency trade-off. Thus MA filtered data is entered into the black-box MGGP system and then, Pareto front plot of the best generated population is depicted to choose a parsimonious model. It is considered up to 7-day lag for the autoregressive model from which the GP selects the most effective inputs. The model was superior to the stand-alone GP, MGGP and conventional *Multivariate Linear Regression* (MLR) prediction models in terms of both prediction accuracy and simplicity.

4 MATERIALS AND METHODS

4.1 Proposed Algorithm

This dissertation proposes a modified MGGP algorithm for nonlinear system identification. We suggest a hybridization with the traditional OLS/ERR method, which is applied to the MGGP individuals depending on some probability. This hybridization is named MGGP/ERR. Moreover, we suggest the use of "two levels" of mutation: a *high-level mutation* and a *low-level mutation* (named after the high- and low-level crossovers). As any evolutionary algorithm, the MGGP/ERR starts with an initial population which evolves through the generations by applying *genetic operators*.

Regarding the *genetic operators*, MGGP works with two levels of crossover: *high-level crossover* and *low-level crossover* (see figures 3.10 and 3.11). In the proposed algorithm, for a given crossover probability $CXPB$, there is 50% chance of applying one or another of the crossover operators. As mentioned before, we also propose two kinds of mutation:

- *high-level* mutation, which swaps one randomly selected gene for a new one; and
- *low-level* mutation, which occurs as a GP *subtree mutation* operator (see 3.2.3), that is, one single gene is selected and its basis function has a subtree replaced by a new random subtree.

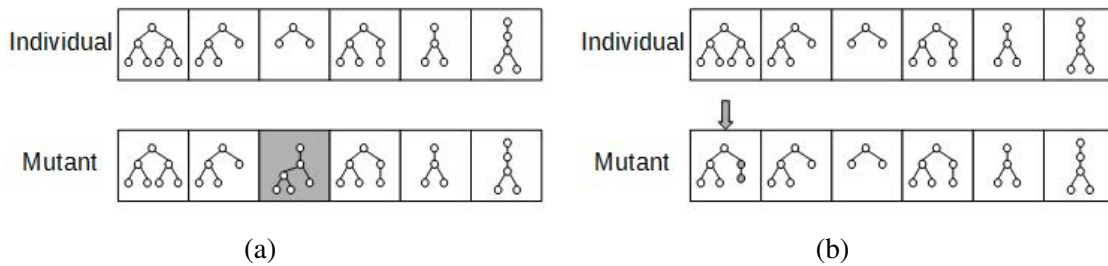
Figure 4.1 exemplify these operators. Note that in the *high-level* mutation (Figure 4.1(a)), the basis function of one randomly selected gene (dark gray) is replaced by an entirely new basis function. On the other hand, in the *low-level* mutation (Figure 4.1(b)), the basis function of one randomly selected gene (pointed by a dark gray arrow) has a subtree replaced by a new one. For a given mutation probability $MTPB$, it has also a 50% chance of applying one or another of the mutation operators. As recommended for GP evolution (POLI et al., 2008), the individual experiences either crossover or mutation, *i. e.* both operators are not applied to the same individual.

The elitism operator is applied over the generations and it selects the best individuals from both offspring population and previous elite individuals.

A set of parameters must be configured in the algorithm:

- *population size*: defines the number of individuals present in the population;

Figure 4.1 – Two level MGGP mutation. (a) *High-level* mutation. (b) *low-level* mutation.

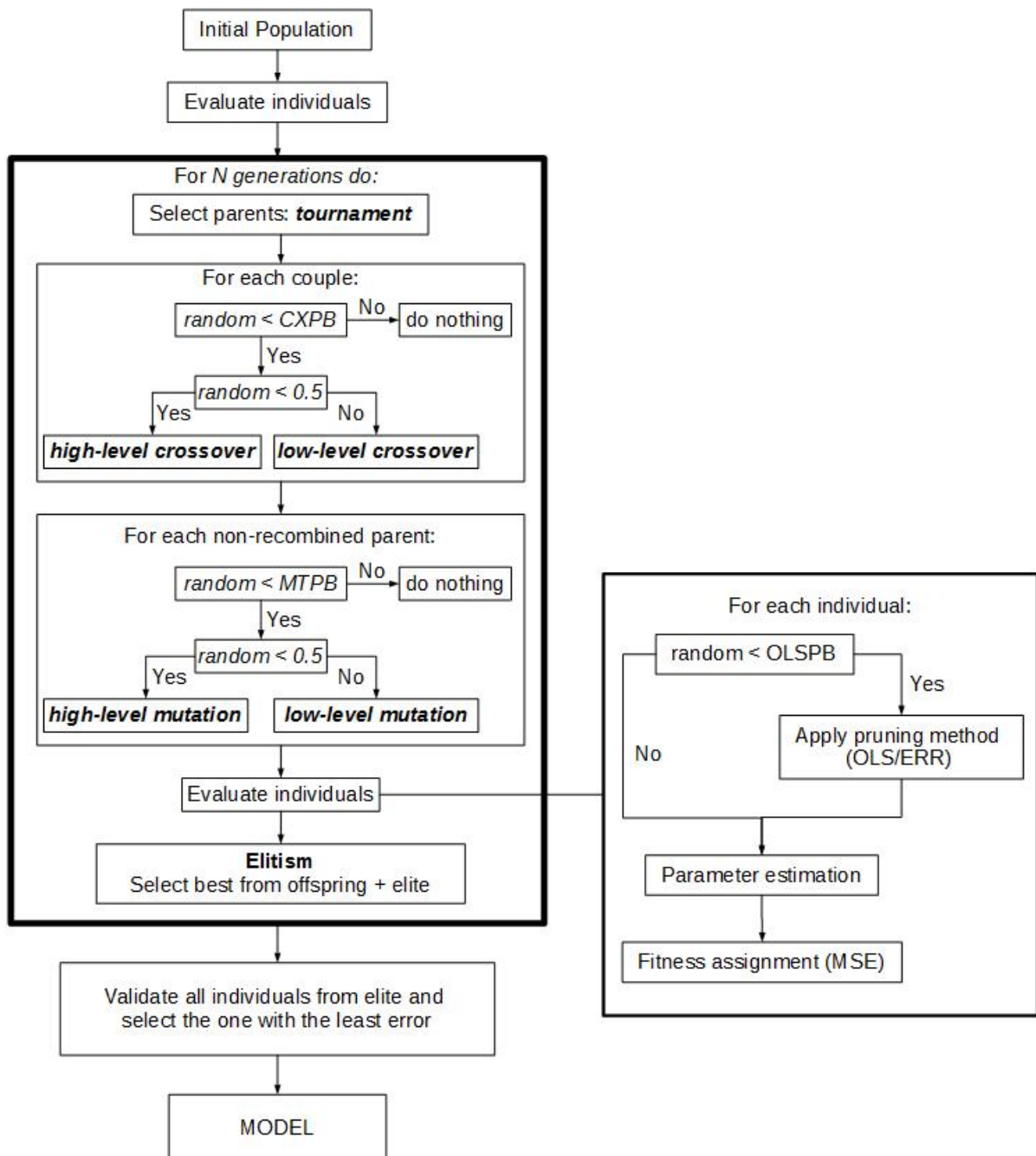


(Source: Author)

- *crossover probability (CXPB)*: defines the probability of a pair of individuals being combined together through a recombination operator;
- *mutation probability (MTPB)*: defines the probability of a single individual being mutated through a mutation operator if it has not experienced recombination;
- *maximum GP height*: limits the size of a GP tree regarding its height;
- *maximum number of MGGP terms*: limits the model size regarding the number of terms that the model can possess;
- *elite size*: defines the percentage of individuals from the population that can remain in the next generation;
- *primitive functions*: the set of functions (multiplication, division, exponentiation, trigonometric operations, etc.) used as *nodes* in GP individuals. Additionally, the set of *back-shift operators* (q^{-1}, q^{-2}, \dots) responsible for the automatic lag determination.
- *pruning probability (OLSPB)*: defines the probability of applying the pruning strategy (OLS/ERR) over an individual during evaluation step.
- *pruning tolerance*: defines the minimum ERR a term must yield to not be discarded.

Figure 4.2 exhibits the algorithm flowchart. It begins with an initial population that is evaluated. Then, the generation loop starts: i) the parent individuals are selected via tournament, ii) each parent couple has a chance to be recombined (CXPB), iii) each individual that has not been recombined has a chance to be mutated (MTPB), iv) the individuals are evaluated, and v) the elitism

Figure 4.2 – Algorithm flowchart



(Source: Author)

operator is applied. After the generation loop, the most significant individuals are validated and the one with the least validation error is selected as the system model. Details from the evaluation function are discussed next.

4.2 Individuals Evaluation

In this step, for any EA, a fitness value is assigned to every individual in the current population. Regarding system identification EAs, in which each individual is a possible model, the model parameters would be estimated and a statistic discriminant value would be assigned to every individual. Traditionally, in this step there is no modification in the individual structure. The MGGP/ERR algorithm is disruptive in this common sense. It uses a pruning strategy that reduces the individuals to only relevant terms during the evaluation step. Thus the parameters estimation, pruning and fitness assignment are performed.

The *cost function* is generically defined by:

```
Evaluation(individual) :
    if rand() < OLSPB:
        apply_pruning_method    // OLS/ERR
    parameter_estimation        // LS or ELS
    fitness_assignment          // MSE
```

Note that there are two design parameters to be set (as mentioned in the previous section), the probability (OLSPB) of applying the OLS/ERR algorithm and its tolerance. The tolerance regards the minimum ERR value a regressor must yield to be selected. The parameters θ estimated by OLS/ERR is equivalent to that estimated via the standard LS method. Hence, if the OLS/ERR is not triggered, the LS method is used. The fitness of an individual is the MSE.

Remark 1 *The MGGP toolbox developed in this dissertation allows the user to work with Single Input Single Output (SISO) and Multiple Input Single Output (MISO) models.*

Remark 2 *The automatic lag determination allows the user not to worry about the assembly of a regressor matrix with all possible terms. The traditional parameters n_y , n_u and l are replaced by the maximum GP height parameter.*

Three different fitness functions are used during individuals evaluation (see definitions in 2.5.1): i) one-step-ahead prediction error PE, ii) free-run simulation error SE, and iii) multiple-shooting simulation error. The next chapter describes the experiments and discuss the results.

5 EXPERIMENTS

In this chapter, we assess the hybrid MGGP/ERR algorithm performance in solving some important identification problems, which are: influence of the noise level, PEM *versus* SEM techniques, and *soft input* excitement. The last two are interconnected, since SEM techniques are usually used to solve *soft input* problems. It is not hard to understand that the noise level may interfere in the algorithm performance. We focused in the equation error problem, which does not yield parameters biases. PEM and SEM techniques differ from resultant model generalization capabilities. And *soft inputs* excite poorly the systems in which are applied. They yield outputs that do not carry all information about the system and hinder the identification process. In total, we perform four experiments. The first two analyse noise level and soft input problems using stochastic test systems to generate data. The third identify a hydraulic pumping system benchmark. And the last one identify a piezoelectric actuator, which is characterized by the hysteretic behavior. In the next sections we state the problems, present from where the data are acquired, present the algorithm configuration used in the process together with scripts on how to perform the experiment using the toolbox, and discuss the results.

5.1 Influence of the Noise Level

The objective of this experiment is to investigate the influence of noise level in the MGGP/ERR algorithm performance. The assessment is yielded for *equation error problem*: the EE noise level ($v[k]$) varies and OE noise level is fixed to zero ($e[k] = 0$) over 30 Monte Carlo simulations for each noise level. The validation errors yielded by the models selected via MGGP/ERR are compared to the ones yielded by the models selected via standalone OLS/ERR, over 30 other Monte Carlo simulations, using the parameters ($l = 3, n_y = 3, n_u = 3$) to build the whole candidate regressor matrix. Only the PE fitness function (2.14) is used in this test.

5.1.1 Data set

The following test systems are used to generate data (MADAR; ABONYI; SZEIFERT, 2005; PIRODDI; SPINELLI, 2003):

$$\begin{aligned}
 S_1 : \quad & \tilde{y}[k] = 0.8u[k-1]^2 + 1.2y[k-1] - 0.9y[k-2] - 0.2 + v[k] \\
 & y[k] = \tilde{y}[k] + e[k] \\
 & \\
 S_2 : \quad & \tilde{y}[k] = 0.75y[k-2] + 0.25u[k-1] - 0.2y[k-2]u[k-1] + v[k] \\
 & y[k] = \tilde{y}[k] + e[k],
 \end{aligned}
 \tag{5.1}$$

where the input signal $u[k]$ is a white Gaussian noise with zero mean and variance of one ($u[k] \sim WGN(0, 1)$), $v[k]$ is white noise signal ($v[k] \sim WGN(0, x)$) that forms the EE problem, with $x \in \{0.02, 0.04, 0.06, 0.08\}$, and $e[k] = 0$.

5.1.2 Algorithm parameters and toolbox guide

The following set up is used for training:

```

population size = 300; generations = 100; elitism = 10%;
maximum GP height = 3; maximum number of MGGP terms = 10;
CXPB = 0.8; MTPB = 0.2 (if not crossover); OLSPB = 1.0
OLS tolerance = 1e-3; delay functions = q1, q2, q3;
primitive functions = multiplication;
parameter estimation = LS;
fitness function = PE.

```

These design parameters are intuitive, some of them from trial and error, *i.e.*, the *maximum GP height* is chosen small to form short basis functions, which yield low degree of nonlinearity. Note that the primitive functions are composed of multiplication operators only. This allows the representation of *polynomial* NARX/NARMAX models.

Script 5.1 – Influence of the noise level experiment.

```

1  import numpy as np
2  from mggp import mggpElement, mggpEvolver
3
4  def evaluate(ind):
5      try:
6          element.compile_model(ind)
7          element.ols(ind, 1e-3, y, u)
8          theta = element.ls(ind, y, u)
9          return element.score_osa(ind, theta, y, u),
10     except np.linalg.LinAlgError as e:
11         return np.inf,
12
13     element = mggpElement()
14     element.setPset(maxDelay=3, numberOfVariables=2,
15                    constant=True)
16     element.renameArguments()
17     evolver = mggpEvolver(popSize=100, CXPB=0.8, MTPB=0.2,
18                           n_gen=100, maxHeight=3, maxTerms=30,
19                           verbose=False, elite=10,
20                           element=element)
21
22     for std in [0.02, 0.04, 0.06, 0.08]:
23         for i in range(30):
24             y, u = generateData(std)
25             hof, log = evolver.run(evaluate=evaluate)
26             bestModel = element.model2List(hof[0])
27             element.save("fileName.pkl", bestModel)

```

The Script 5.1 shows how to perform the *influence of noise level* experiment. The MGGP toolbox is composed of two classes: *mggpElement* and *mggpEvolver*. An *mggpElement* object carries information about individual's characteristics and an *mggpEvolver* object is responsible for population evolution. Note that the toolbox allows the user to perform an experimentation with very few lines of code. We can summarize the coding process in four steps: *i*) define an element object

Table 5.1 – Number of times the algorithms selected all terms present in the systems during 30 Monte Carlo simulations.

	MGGP/ERR				OLS/ERR			
	0.02	0.04	0.06	0.08	0.02	0.04	0.06	0.08
S_1 (4 terms)	26/30	26/30	27/30	29/30	23/30	20/30	21/30	22/30
average <i>SNR</i>	88.37	76.75	69.82	64.21	88.37	76.75	69.82	64.21
S_2 (3 terms)	22/30	27/30	25/30	27/30	30/30	30/30	30/30	30/30
average <i>SNR</i>	51.47	40.05	32.34	27.15	51.47	40.05	32.34	27.15

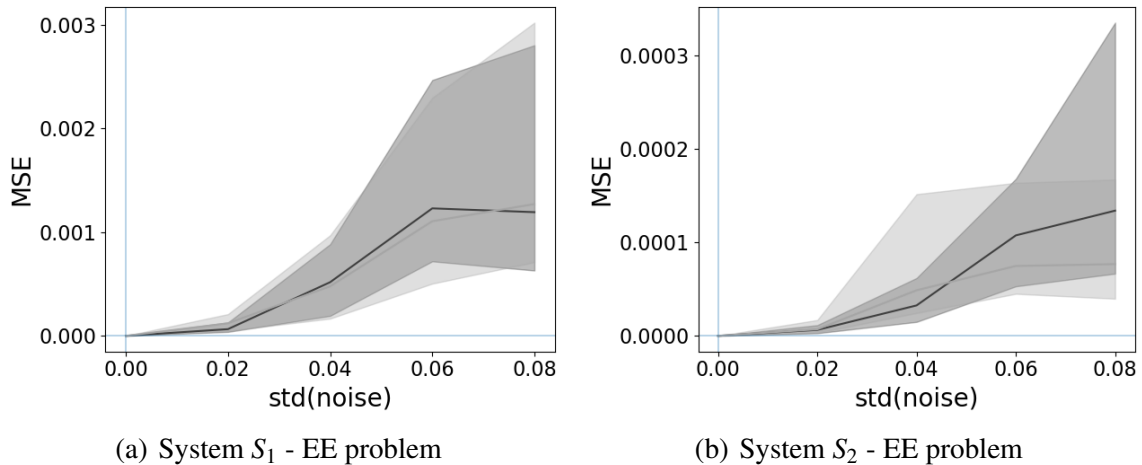
that carries the primitive set used to form nodes and terminals, *ii*) define the evaluation function, *iii*) define the evolver element that runs the evolution process, and *iv*) run the evolution process. Details about the functionalities of each method can be seen in Appendix A.

5.1.3 Results and discussion

Table 5.1 presents the performance of the proposed MGGP/ERR algorithm and of the standalone OLS/ERR algorithm regarding the number of times the algorithms are able to select *every* term present in the model (in 30 Monte Carlo executions). Note that the EE noise level has no effect on the structure selection for both MGGP/ERR and OLS algorithms since the number of times they select all correct terms does not correlate with the noise level. In addition, the MGGP/ERR algorithm yields better performance than the OLS for system S_1 , and the opposite is true for system S_2 . Regarding the first, the means of correctly selected models are 27.00 for MGGP/ERR and 21.50 for OLS (out of 30). For the second, the values are 25.50 for MGGP/ERR against 30.00 for OLS. These results indicate that the best algorithm to be used depends on the system dynamics.

Figure 5.1 presents the free-run simulation errors of the selected models for validation data. Note that the algorithms yield similar performance despite the number of times they select all terms present in the model. It indicates that the terms which are not selected are not so relevant to the system dynamics. We highlight Figure 5.1(b), in which MGGP/ERR models perform slightly better than OLS/ERR algorithm for the highest noise level, even though the latter selected models correctly more times than the former.

Figure 5.1 – The free-run mean squared errors (or simulation errors) for the selected models via OLS (dark gray) and MGGP (light gray) algorithms for the case increasing equation error (EE) noise level. The shaded plots represent the inter-quartile range and the inner lines represent the median value.



5.2 Influence of OLS Probability and PEM and SEM Techniques

In this experiment, the influence of OLS/ERR pruning in the MGGP structure selection algorithm is analyzed. The cost functions (individuals evaluation) are implemented using PEM or SEM techniques for comparison. In the identification process, the systems are excited by *soft input*, and the validation data are generated using a persistently exciting signal (ideal input). This is a challenging situation where the identification data do not carry all information about the system. In addition, we use a polynomial framework to model a rational test system.

5.2.1 Data set

Two OE test systems are used to generate data for the identification process. The first is the system S_2 , defined in (5.1), and the second is the following system taken from (HINCHLIFFE; WILLIS, 2003):

$$S_3 : \quad \tilde{y}[k] = \tilde{y}[k-1] \cdot \tilde{y}[k-2] \cdot \frac{2.5 + \tilde{y}[k-1]}{1 + \tilde{y}[k-1]^2 + \tilde{y}[k-2]^2} + u[k-11], \quad (5.2)$$

$$y[k] = \tilde{y}[k] + e[k],$$

where $e[k]$ is a WGN with standard deviation $std(e) = 0.02$ for system S_2 ($SNR \approx 51.47$) and $var(e) = 0.04 var(\tilde{y})$ for system S_3 ($SNR \approx 28.05$) and, in both cases, the input:

- for the identification data, $u[k] \sim WGN(0;0.20)$ that is processed by a low-pass filter with poles in 0.95 and 0.9;
- for the validation data, $u[k] \sim WGN(0;1)$.

Training and validation data are composed of 500 samples and the validation data are simulated without noise.

5.2.2 Algorithm parameters and toolbox guide

The algorithm set up for the identification of both systems differ only in the individual size constraints as follows:

```

population size = 200; generations = 50; elitism = 10%;
CXPB = 0.8; MTPB = 0.2 (if not crossover); OLSPB = 0-100%;
OLS tolerance = 1e-3; delay functions = q1, q2, q3;
primitive functions: multiplication;
parameter estimation: LS
fitness function = PE / SE
for system S2:
    maximum GP height = 3; maximum MGGP terms = 10;
for system S3:
    maximum GP height = 5; maximum MGGP terms = 30.
```

The Script 5.2 presents how to perform the *influence of OLS probability and PEM and SEM techniques* experiment. In addition to the Script 5.1, an OLS probability is included into the evaluation function. PEM and SEM techniques are defined by the return method in the evaluation function.

Script 5.2 – Influence of OLS probability and PEM and SEM techniques experiment

```

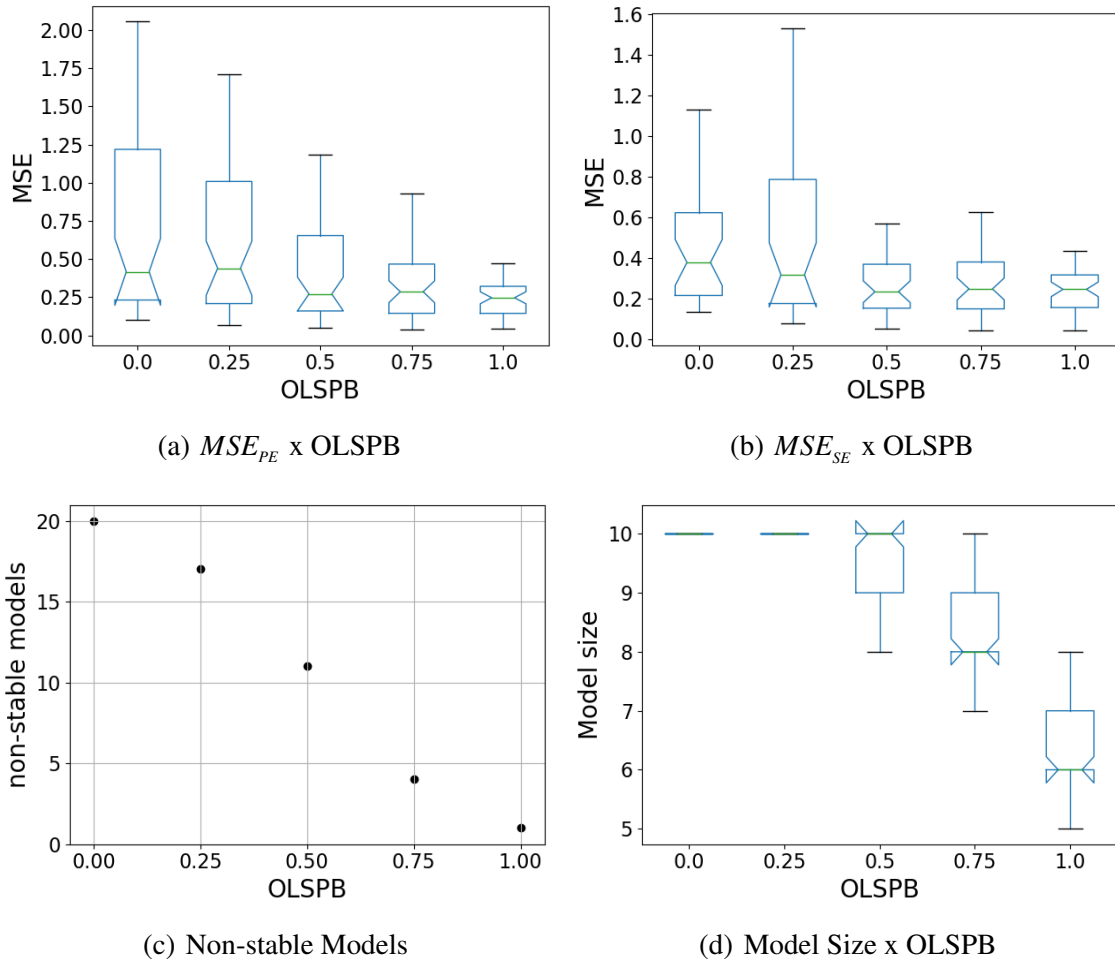
1  import numpy as np
2  from mggp import mggpElement, mggpEvolver
3  import random
4
5  def evaluate(ind):
6      try:
7          element.compile_model(ind)
8          if random.random() < OLSPB:
9              element.ols(ind, 1e-3, y, u)
10             theta = element.ls(ind, y, u)
11             ind.theta = theta
12             return element.score_osa(ind, theta, y, u),
13             # return element.score_freeRun(ind, theta, y, u),
14         except np.linalg.LinAlgError as e:
15             return np.inf,
16     element = mggpElement()
17     mggp = mggpEvolver(popSize=200, CXPB=0.8, MTPB=0.2,
18                       n_gen=50, maxHeight=5, maxTerms=30,
19                       verbose=False, elite=10, element=element)
20     for OLSPB in [0.0, 0.25, 0.5, 0.75, 1.0]:
21         for i in range(50):
22             y, u = generateData()
23             hof, log = evolver.run(evaluate=evaluate)
24             bestModel = element.model2List(hof[0])
25             element.save("fileName.pkl", bestModel)

```

5.2.3 Results and discussion

The results of this experiment consist of the PE and SE performances obtained over the validation data with *ideal input*, the number of unstable models in free-run simulation (out of 50) and the model sizes over 50 Monte Carlo simulations for each *OLSPB* value.

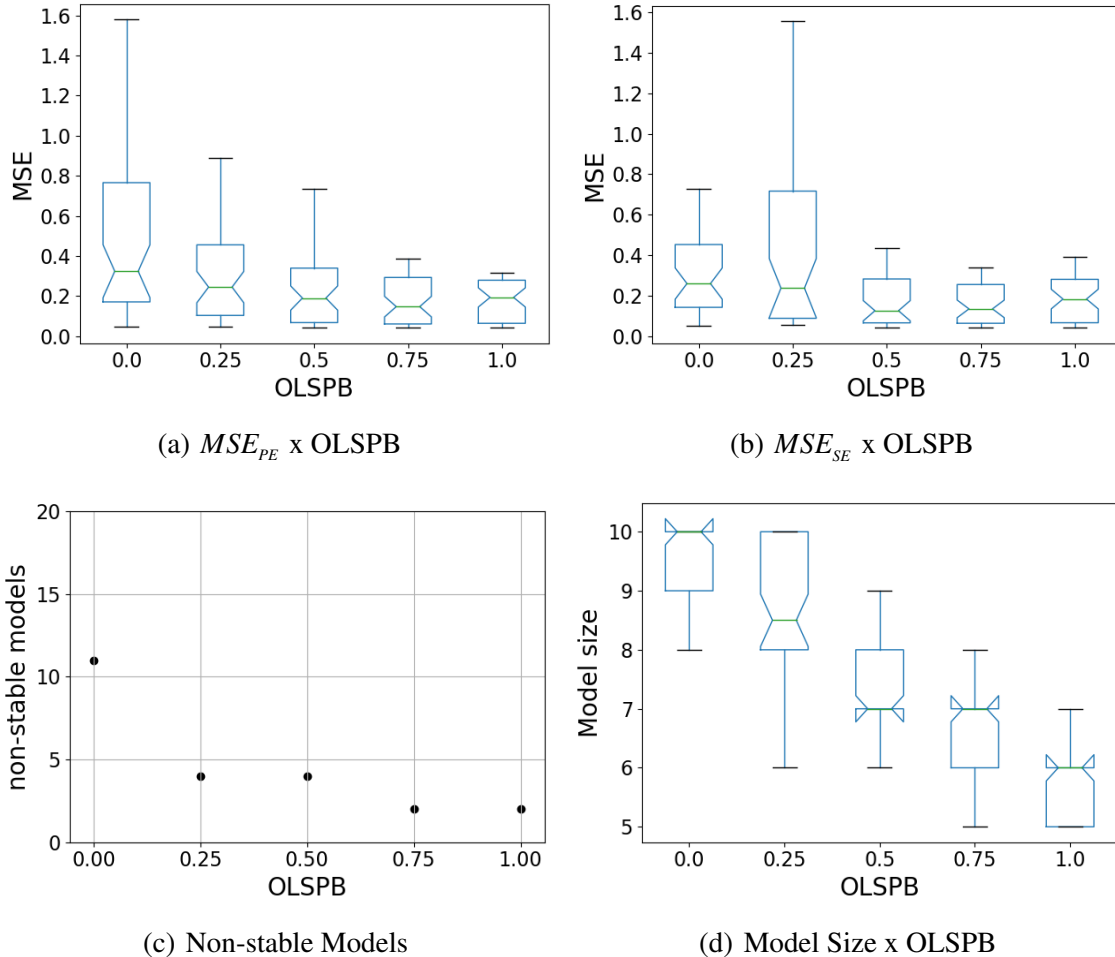
Figure 5.2 – Selected models results for system S_2 using PEM technique and *soft input* in 50 Monte Carlo simulations. It consists of the graphics (a) $PE \times OLSPB$, (b) $SE \times OLSPB$, (c) *non-stable models* $\times OLSPB$ and (d) *model sizes* $\times OLSPB$.



5.2.3.1 System S_2

Figures 5.2 and 5.3 present the results for system S_2 using PEM and SEM techniques, respectively. In each figure, there are three box-plots relating PE, SE and *model size* to the probability $OLSPB$ and one graphic relating the number of models (out of 50) that are unstable in free-run simulation over validation data to the $OLSPB$ values. Note that there is an improvement in terms variance of the PE and SE scores for both PEM and SEM techniques with increment of $OLSPB$. However, the boxes notches overlap hence nothing can be said about their medians. The graphics $SE \times OLSPB$ (figures 5.3(b) and 5.4(b)) are complementary to the graphics *non-stable models* $\times OLSPB$ (figures 5.3(c) and 5.4(c)). For example, regarding the PEM technique and $OLSPB = 0.0$,

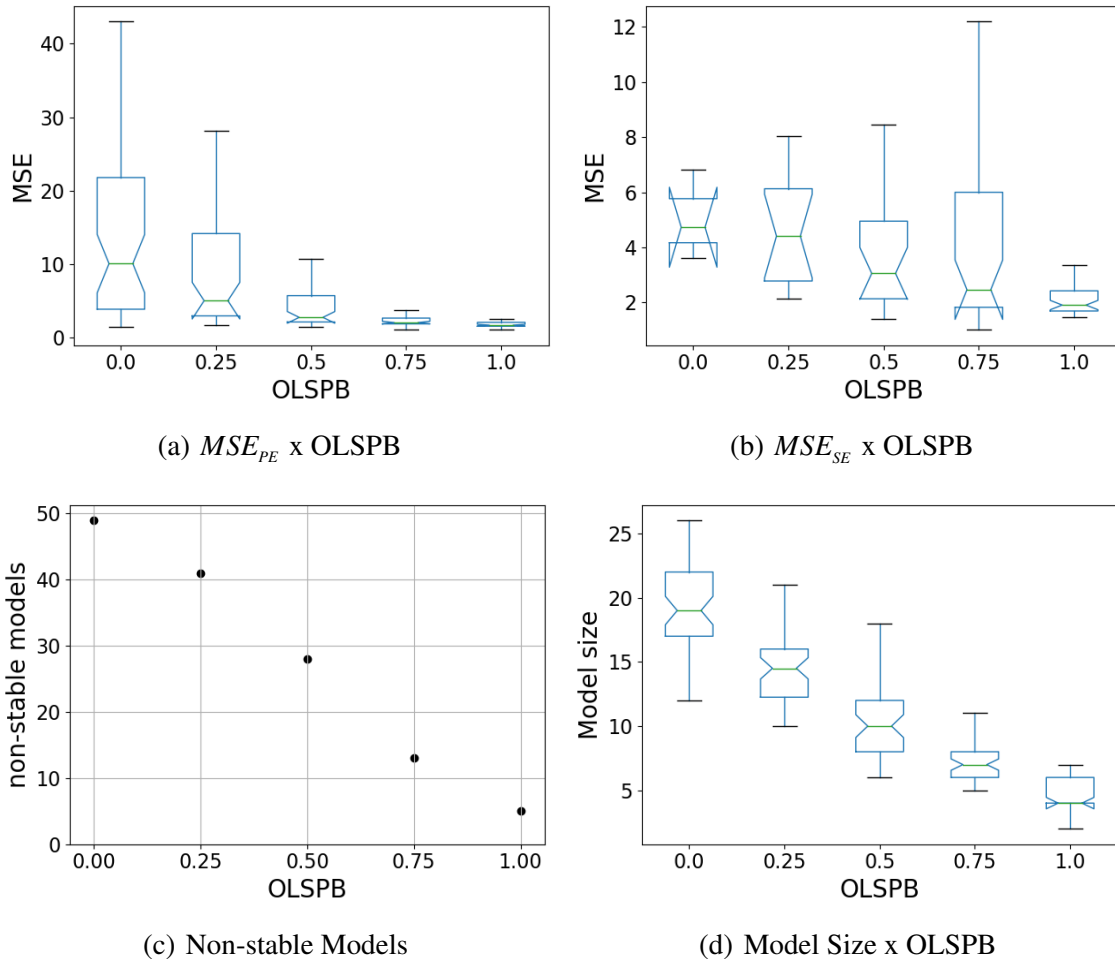
Figure 5.3 – Selected models results for system S_2 using SEM technique and *soft input* in 50 Monte Carlo simulations. It consists of the graphics (a) $PE \times OLSPB$, (b) $SE \times OLSPB$, (c) *non-stable models* $\times OLSPB$ and (d) *model sizes* $\times OLSPB$.



there are 20 models (out of 50) that are BIBO unstable on validation data, and these models are not represented in the box-plot $SE \times OLSPB$. Thus, for $OLSPB = 1.0$ (in which there is only one unstable model), the median and variance of SE are much smaller than the ones for $OLSPB = 0.0$ (in which there are 20 unstable models). It can be inferred that there is a significant improvement in the SE over the validation data with the increment of the $OLSPB$. Moreover, regarding the model size $\times OLSPB$ boxplot, the boxes notches do not overlap for $OLSPB$ bigger than 0.5, which means that the median values of the models size decrease with the increment of $OLSPB$.

For PEM technique, $OLSPB = 0$ and $OLSPB = 0.25$, all 50 models are of size 10. It goes down to between 5 and 8 terms, with median of 6, for $OLSPB = 1.0$. It is clear that the removal of spurious terms influences the number of unstable models. It is worth mentioning the difference

Figure 5.4 – Selected models results for system S_3 using PEM technique and *soft input* in 50 Monte Carlo simulations. It consists of the graphics (a) $PE \times OLSPB$, (b) $SE \times OLSPB$, (c) *non-stable models* $\times OLSPB$ and (d) *model sizes* $\times OLSPB$.

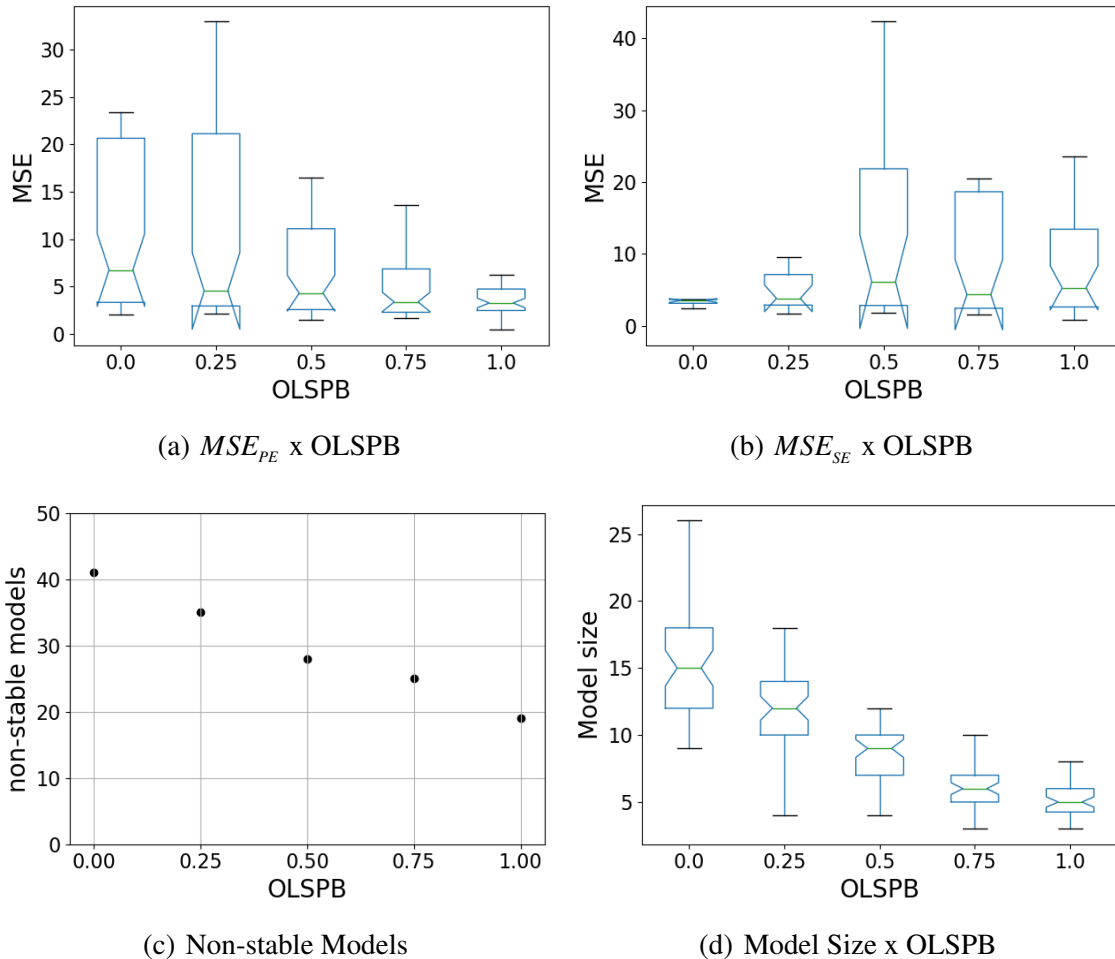


between the *non-stable models* graphics for PEM and SEM techniques. Note that for $OLSPB = 0.0$, the number of unstable models is smaller for SEM technique (11 models for SEM and 20 models for PEM technique). This behavior is expected since SEM based algorithms yield better models when soft inputs are used (PIRODDI; SPINELLI, 2003).

5.2.3.2 System S_3

Figures 5.4 and 5.5 present the results for system S_3 using PEM and SEM techniques, respectively. Note that, for $OLSPB = 0.0$, all the models obtained by PEM technique and 41 models (out of 50) obtained by SEM technique are BIBO unstable in free-run simulation. The

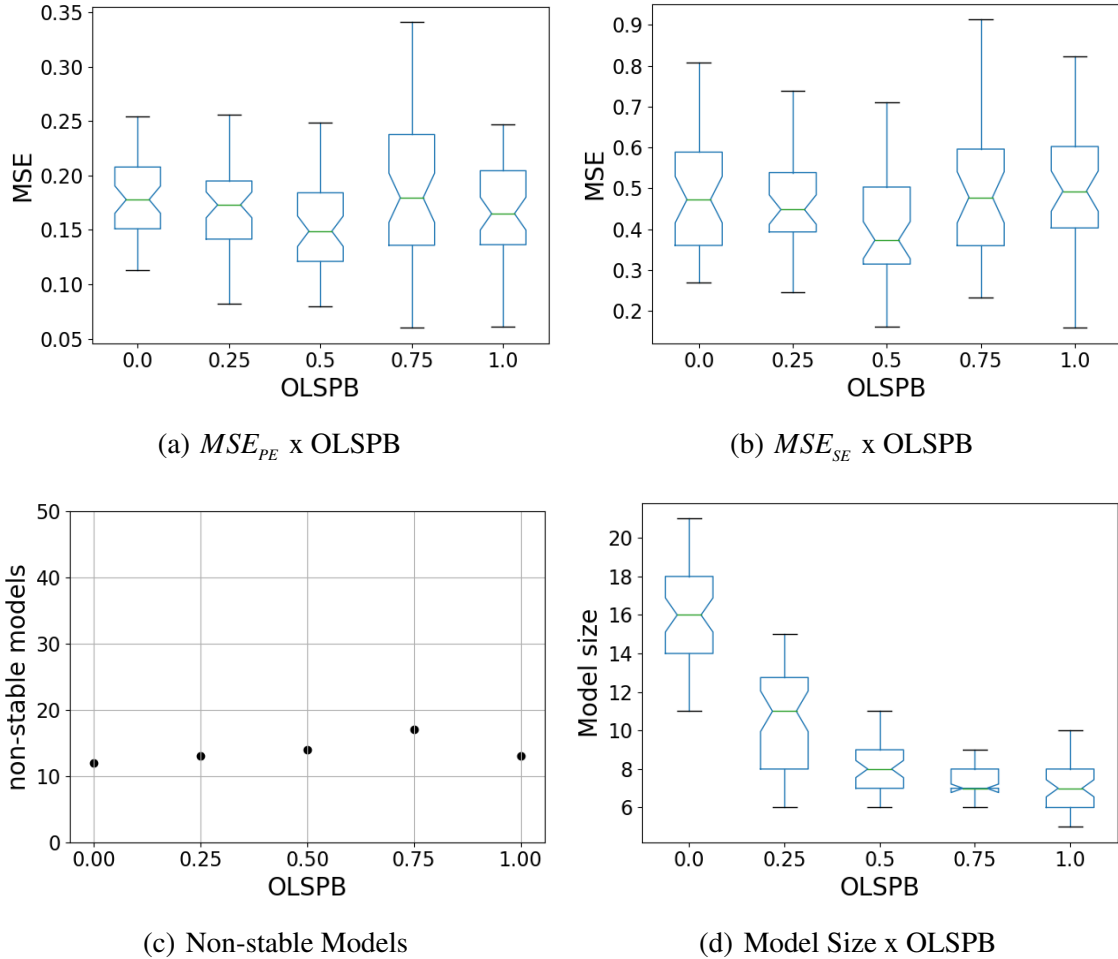
Figure 5.5 – Selected models results for system S_3 using SEM technique and *soft input* in 50 Monte Carlo simulations. It consists of the graphics (a) $PE \times OLSPB$, (b) $SE \times OLSPB$, (c) *non-stable models* $\times OLSPB$ and (d) *model sizes* $\times OLSPB$.



improvement in terms of stable models was greater for PEM technique (from 50 unstable models in $OLSPB = 0.0$ to 5 unstable models in $OLSPB = 1.0$) than the one for SEM technique (from 41 unstable models in $OLSPB = 0.0$ to 19 unstable models in $OLSPB = 1.0$). The improvement in the PE for this system is clearer than the one for system S_2 . In Figure 5.5(a), the variance and median of the PE decreases systematically with the increment of the $OLSPB$. Note that the notches of $OLSPB = 0.0$ and $OLSPB = 0.5$ do not overlap. It is worth mentioning that, for system S_3 , the use of PEM technique together with 100% chance of applying OLS/ERR algorithm during evaluation step achieved better performance than the use of SEM technique with the same $OLSPB$.

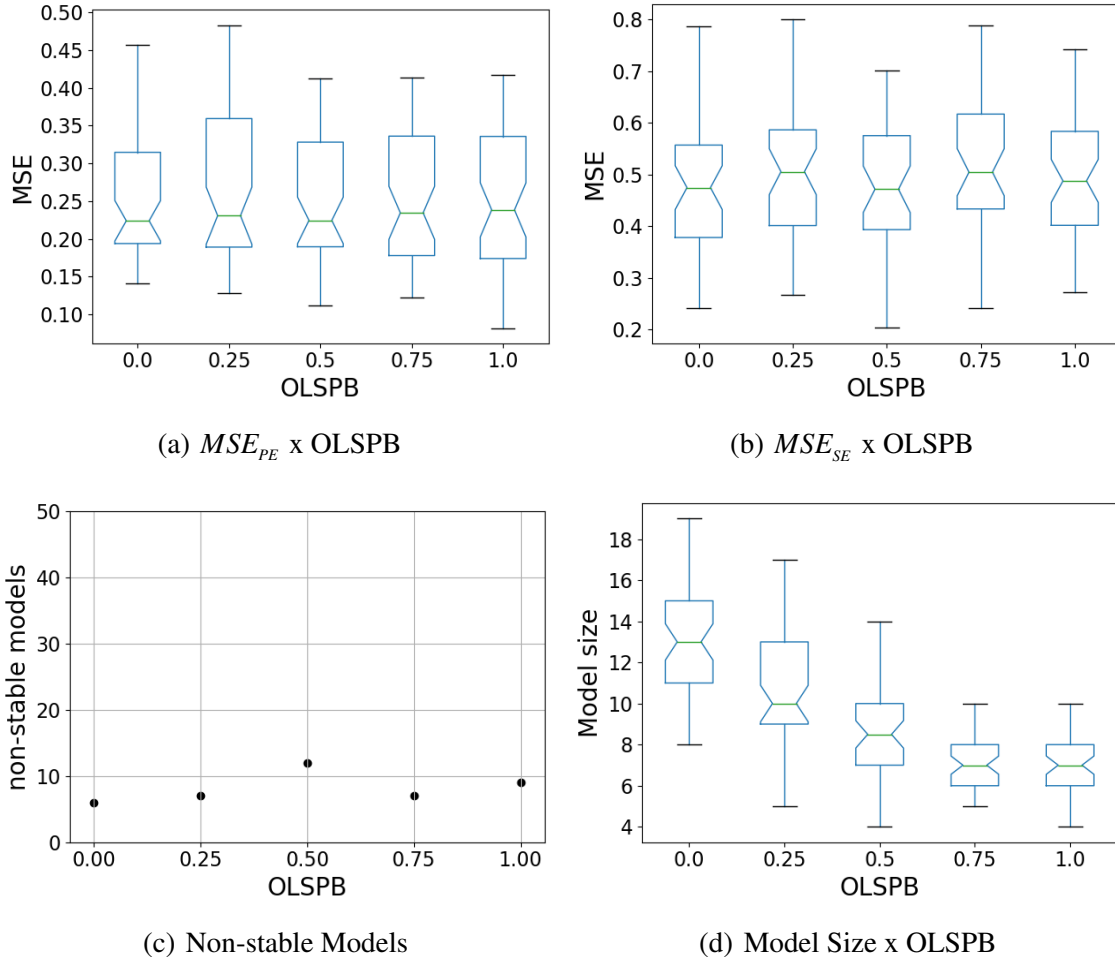
The system S_3 has high degree of non-linearity (with rational structure) and long input lag term dependency ($u[k - 11]$). The degree of nonlinearity necessary to represent this system in

Figure 5.6 – Selected models results for system S_3 using PEM technique in 50 Monte Carlo simulations with *ideal input*. It consists of the graphics (a) $PE \times OLSPB$, (b) $SE \times OLSPB$, (c) *non-stable models* $\times OLSPB$ and (d) *model sizes* $\times OLSPB$.



a *polynomial* structure is not known in advance. It should be emphasized that if the OLS/ERR algorithm was applied in this example for structure selection using the parameters ($l = 5, n_y = 11, n_u = 11$), the RAM memory would burst out for low performance computers (> 2 Gb for a single matrix). Moreover, the OLS/ERR algorithm usually fails to select the correct regressor among candidates highly correlated, *i.e.*, the correct term $y[k - 1]$ and the term $y[k - 2]$, which is highly correlated to the former if the system is excited by *soft input*. In this way, the MGGP algorithm facilitates the OLS/ERR algorithm task, as the former selects groups of terms from the candidate regressor space over which the latter is applied. Eventually, the highly correlated regressors do not appear in the group of regressor candidates selected by the MGGP and the OLS/ERR selects the correct terms.

Figure 5.7 – Selected models results for system S_3 using SEM technique in 50 Monte Carlo simulations with *ideal input*. It consists of the graphics (a) $PE \times OLSPB$, (b) $SE \times OLSPB$, (c) *non-stable models* $\times OLSPB$ and (d) *model sizes* $\times OLSPB$.

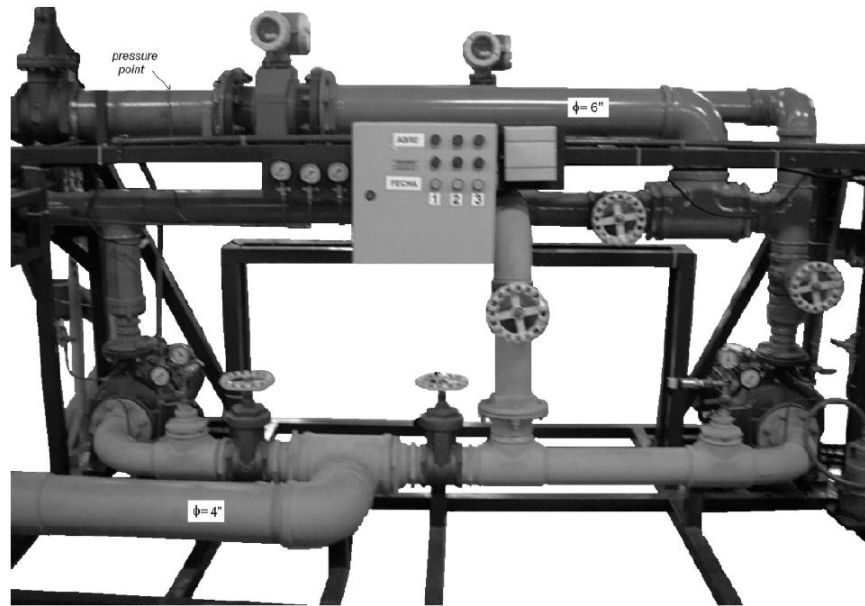


One last analysis must be made to conclude this experiment. Until this point, only *soft inputs* were used in the training step. Figures 5.6 and 5.7 show the results for system S_3 in an ideal situation of training data, that excites the system in all possible frequencies (*ideal input*). There is no improvement due to the increment of the *OLSPB* in terms of performance for PEM nor SEM techniques. Although, the use of OLS/ERR yields more parsimonious models.

5.3 Hydraulic Pump

The hydraulic pumping system (Figure 5.8) here studied simulates the behavior of the hydraulic part of a hydroelectric power plant, and the output pressure must be controlled over a wide

Figure 5.8 – Hydraulic pump system.



(Source: Barbosa et al. (2011))

range of operating conditions. Mathematical models are desired to simulate and to design a closed-loop controller for the real pumping system, in which the model output is the system pressure and the model input is the pumps' reference speed (BARBOSA et al., 2011).

5.3.1 Data set

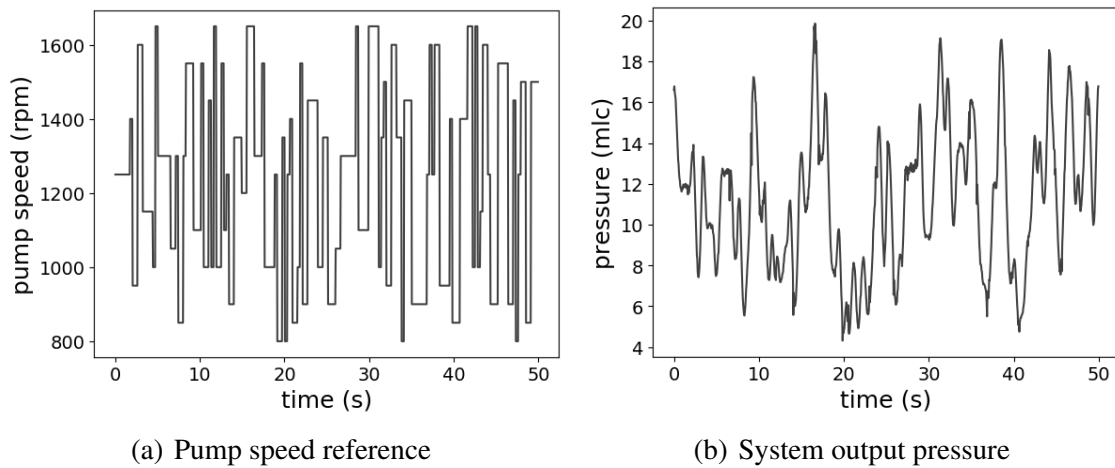
The data set, also used in (BARBOSA et al., 2011), is taken from the Artificial Intelligence and Automation group webpage (<http://www.aia.ufla.br/home/filesdatasets/>). The dynamical data are composed of $N = 3200$ samples for model identification and $N = 800$ samples for validation, with a sampling time of $T_s = 50$ ms. An example of input-output data is shown in Figure 5.9.

5.3.2 Algorithm setup

To model the hydraulic pump system, two cost functions are tested: the PEM and SEM techniques. None extra primitive function is included in the algorithm, so it looks for polynomial models. The following setup is used:

```
population size = 500; generations = *; elitism = 10%;
maximum GP height = 5; maximum number of MGGP terms = 50
```


Figure 5.9 – Hydraulic pump dynamical data.



```

CXPB = 0.8; MTPB = 0.2 (if not crossover); OLSPB = 0.2;
OLS tolerance = 1e-7; delay functions = q1, q2, q3, q4, q5;
primitive functions: multiplication;
parameter estimation: ELS;
fitness function = PE and SE.

```

The algorithm is run until the minimum error stops improving. The parameter estimation method used in this experiment is the *extended least squares* (see 2.4). In this case, the identification is performed considering the problem as a white noise output error problem (see 2.2.1).

The Script 5.3 presents a code to perform this experiment using the MGGP toolbox. Note the use of extended least squares in line 10. In this experiment, after every 100th generation the best individual from the population is saved and only the elite is maintained for the next one.

Script 5.3 – Hydraulic pump identification experiment

```

1  from mggp import mggpElement, mggpEvolver
2  import numpy as np
3  import random
4
5  def evaluate(ind):
6      try:
7          element.compile_model(ind)
8          if random.random() < OLSPB:
9              element.ols(ind, 1e-7, y, u)
10             theta = element.ls_extended(ind, y, u)
11             return element.score_osa(ind, theta, y, u),
12             # return element.score_freeRun(ind, theta, y, u),
13         except np.linalg.LinAlgError as e:
14             return np.inf,
15
16 element = mggpElement()
17 element.setPset(maxDelay=5)
18 element.renameArguments({'ARG0': 'y1', 'ARG1': 'u1'})
19 mggp = mggpEvolver(popSize=200, CXPB=0.8, MTPB=0.2,
20                    n_gen=100, maxHeight=5, maxTerms=30,
21                    elite=10, element=element)
22 y, u = trainingData()
23 OLSPB = 0.2
24 k = 500
25 seed = None
26 for i in range(100):
27     hof, log = mggp.run(evaluate=evaluate, seed=seed)
28     best = element.model2List(hof[0])
29     element.save("fileName.pkl", best)
30     seed = hof.items

```

5.3.3 Results and discussion

In this experiment, two models were found by the MGGP algorithm, one from the PEM technique (model $MGGP_{PEM}(5.3)$) and another from the SEM technique (model $MGGP_{SEM}(5.4)$):

$$\begin{aligned}
y_{PEM}[k] = & \theta_1 y[k-1] + \theta_2 y[k-4] + \theta_3 y[k-12] + \theta_4 u[k-4] + \theta_5 u[k-13] \\
& + \theta_6 y[k-1]y[k-2] + \theta_7 y[k-1]y[k-3] + \theta_8 y[k-1]u[k-4] \\
& + \theta_9 y[k-1]u[k-6] + \theta_{10} y[k-5]u[k-2] + \theta_{11} y[k-5]u[k-7] \\
& + \theta_{12} y[k-8]u[k-7] + \theta_{13} u[k-1]u[k-4] + \theta_{14} u[k-1]u[k-6] \\
& + \theta_{15} u[k-4]u[k-6] + \theta_{16} y[k-1]^2 y[k-2] + \theta_{17} y[k-1]y[k-3]^2 \\
& + \theta_{18} y[k-1]y[k-5]^2 + \theta_{19} y[k-3]^2 u[k-3] + \theta_{20} y[k-1]u[k-2]^2 \\
& + \theta_{21} y[k-1]y[k-3]u[k-4] + \theta_{22} y[k-1]u[k-3]u[k-5] \\
& + \theta_{23} y[k-7]^2 u[k-2] + \theta_{24} y[k-1]^3 u[k-6] \\
& + \theta_{25} y[k-1]y[k-8]^2 u[k-1] \\
& + \theta_{26} y[k-1]y[k-3]^2 u[k-3] + \theta_{27} y[k-2]y[k-5]^2 u[k-5] \\
& + \theta_{28} y[k-3]^2 u[k-4]^2 + \theta_{29} y[k-3]^2 y[k-5]u[k-5] \\
& + \theta_{30} y[k-8]^2 u[k-1]u[k-6] + \theta_{31} y[k-2]y[k-3]y[k-5]u[k-4] \\
& + \theta_{32} y[k-1]^3 y[k-7]u[k-7] + \xi_{MA}[k],
\end{aligned} \tag{5.3}$$

$$\begin{aligned}
y_{SEM}[k] = & \theta_1 y[k-7] + \theta_2 u[k-4] + \theta_3 u[k-10] + \theta_4 u[k-11] + \theta_5 y[k-1]y[k-3] \\
& + \theta_6 y[k-1]y[k-2] + \theta_7 y[k-1]u[k-4] + \theta_8 y[k-5]u[k-7] \\
& + \theta_9 y[k-6]u[k-4] + \theta_{10} y[k-6]u[k-10] + \theta_{11} y[k-3]^3 \\
& + \theta_{12} y[k-1]^2 y[k-2] + \theta_{13} y[k-3]^2 u[k-4] + \theta_{14} y[k-2]y[k-4]y[k-7] \\
& + \theta_{15} y[k-1]y[k-4]u[k-6] + \theta_{16} y[k-6]y[k-9]u[k-12] \\
& + \theta_{17} y[k-1]u[k-5]u[k-12] + \theta_{18} u[k-4]^3 + \theta_{19} u[k-5]^2 u[k-10] \\
& + \theta_{20} y[k-3]^2 u[k-4]^2 + \theta_{21} y[k-3]y[k-4]y[k-7]u[k-10] \\
& + \theta_{22} y[k-2]y[k-3]y[k-5]u[k-4] + \theta_{23} y[k-2]y[k-3]^2 u[k-6] \\
& + \theta_{24} y[k-5]^2 u[k-5]u[k-6] + \theta_{25} y[k-1]^3 y[k-5]u[k-4] + \xi_{MA}[k],
\end{aligned} \tag{5.4}$$

where the term ξ_{MA} represents the moving average part of the model. Model terms are not sorted by ERR value.

Table 5.2 – Hydraulic pump models. The first two are from (BARBOSA et al., 2011) obtained via OLS method. PE is the one-step-ahead MSE (mlc^2); SE is the free-run MSE (mlc^2); N_p the number of parameters present in the model; and $max. N_p$ is the number of all possible regressors for given (l, n_y, n_u) .

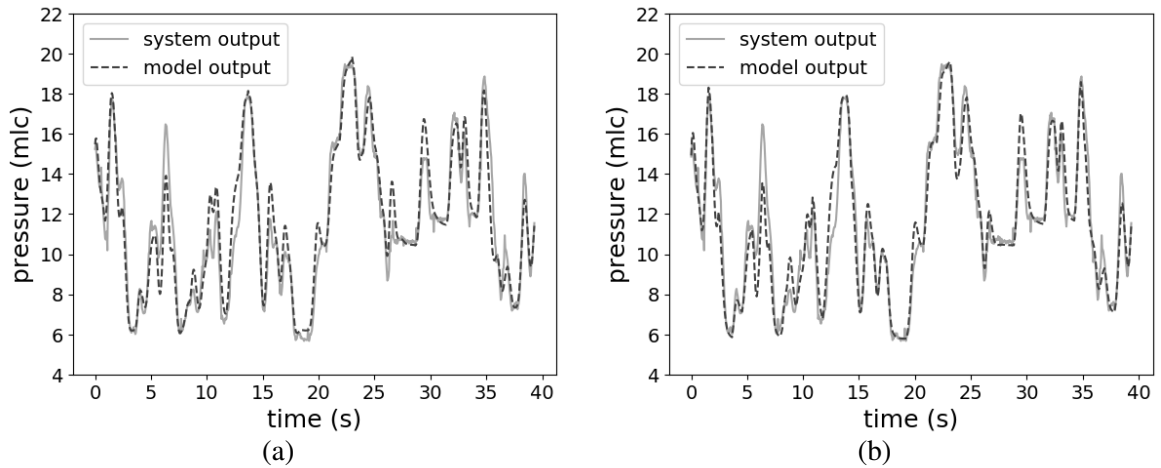
Model	PE(Ident.)	PE(Val.)	SE(Ident.)	SE(Val.)	N_p	(l, n_y, n_u)	$max. N_p$
Barbosa (15)	0.100	0.082	2.600	2.243	17	(2, 6, 6)	90
Barbosa (17)	0.086	0.070	1.447	1.120	23	(3, 6, 6)	454
$MGGP_{PEM}(5.3)$	0.067	0.058	1.910	1.150	32	(5, 12, 13)	142505
$MGGP_{SEM}(5.4)$	0.088	0.068	1.221	0.997	25	(5, 9, 12)	65779

Barbosa et al. (2011) worked with two models in which parameters were estimated via the ELS method and structures were obtained by OLS/ERR algorithm. The model *Barbosa (15)* is obtained using $l = 2$, $n_y = 6$ and $n_u = 6$; and the model *Barbosa (17)* with $l = 3$, $n_y = 6$ and $n_u = 6$. Their performances are described in Table 5.2, where PE and SE of each model over the identification and validation data are presented, as well as the number of terms a polynomial model would contain considering all possible regressors for a given nonlinearity degree (l), maximum output lag (n_y) and maximum input lag (n_u). The performances of the obtained MGGP models are also described in the table.

Model $MGGP_{PEM}$ (5.3) achieved the best one-step-ahead prediction results. Model $MGGP_{PEM}$ (5.3) and model *Barbosa (17)* obtained $PE = 0.067 \text{ mlc}^2$ and $PE = 0.086 \text{ mlc}^2$, respectively, for the training data (model $MGGP_{PEM}$ (5.3) performs 22% better), and $PE = 0.058 \text{ mlc}^2$ and $PE = 0.070 \text{ mlc}^2$ for the validation data (model $MGGP_{PEM}$ (5.3) performs 17% better), respectively. However, the free-run simulation errors are $SE = 1.910 \text{ mlc}^2$ and $SE = 1.447 \text{ mlc}^2$ for the training data (model $MGGP_{PEM}$ (5.3) performs 32% worse) and $SE = 1.150 \text{ mlc}^2$ and $SE = 1.120 \text{ mlc}^2$ for the validation data (model $MGGP_{PEM}$ (5.3) performs 3% worse), respectively. Note that for the validation data, the simulation error of model $MGGP_{PEM}$ (5.3) is very close to the error of model *Barbosa (17)*. Considering only the validation data as the criterion, model $MGGP_{PEM}$ (5.3) is considered better, with the best prediction error and similar simulation error. Moreover, the algorithm was trained with the PE cost function, which means that it accomplished its objective finding better one-step-ahead model for the hydraulic pump under analysis without critically worsening its free-run prediction in comparison with the previous work's results.

Model $MGGP_{SEM}$ (5.4) achieved better PE and SE than *Barbosa (17)* for the validation data. The errors are $PE = 0.068 \text{ mlc}^2$ and $PE = 0.070 \text{ mlc}^2$ (model $MGGP_{SEM}$ (5.4) performs 3%

Figure 5.10 – Free-run simulation of models obtained by MGGP algorithm for the hydraulic pump validation data: (a) Model (5.3) - PEM technique and (b) Model (5.4) - SEM technique



better) and $SE = 0.997 \text{ mlc}^2$ and $SE = 1.120 \text{ mlc}^2$ (model $MGGP_{SEM}$ (5.4) performs 11% better), respectively. The PE is not as good as that for model $MGGP_{PEM}$ (5.3). Nevertheless, the algorithm was trained with the SE cost function and found a model with the best free-run prediction. The choice of which model to use depends on the application, i.e., whether better PE or better SE is desirable.

Note that the proposed algorithm built a model with nonlinearity degree of 5 and maximum input and output delays of 13 and 12, respectively ($l = 5$, $n_u = 13$ and $n_y = 12$). This means a total of 142505 possible regressors for a traditional OLS/ERR analysis (not considering the constant term). Table 5.3 presents the memory usage for building the full candidate regressor matrix for different sets of degrees of nonlinearity and maximum output and input lags (l, n_y, n_u). The memory usage increases exponentially, and for the last set (5, 12, 12), a memory error is raised in the attempt to build the matrix of shape (3188 x 118754). Therefore, the proposed MGGP algorithm is able to explore a considerably larger search space than the traditional OLS method without the need for high-performance computers. Although the interesting feature of explore a wide range in search space, the MGGP/ERR have a considerable disadvantage compared to stand alone ERR-based algorithms: processing time. While stand alone ERR-based algorithms take a few seconds to perform the hydraulic pump identification presented in this section, the MGGP/ERR takes several hours.

Table 5.3 – Memory usage for building one full candidate regressor matrix for the real problem training data (3200 samples)

	(l, n_y, n_u)			
	(3, 3, 3)	(5, 5, 5)	(5, 10, 10)	(5, 12, 12)
Memory (mb)	2.02	73.18	1293.04	≈ 2820.00

IntelCore™ i7 - 4500U 1.8GHz 8Gb Memory.

5.4 Piezoelectric Actuator

Piezoelectric actuators (PEAs) are used in micro- and nano-positioning applications. Their resolutions, fast responses, and large actuating forces are their main characteristics. However, the existence of nonlinearities such as hysteresis makes modeling and controlling PEAs challenging (PENG; CHEN, 2013). Abreu et al. (2020) present a property that a model must attain in order to represent the main aspects of a hysteretic system and use it to obtain constraints on the structure and parameters of the model:

Property 1 *An identified model of hysteresis, under a constant input, has two or more real nondiverging equilibria.*

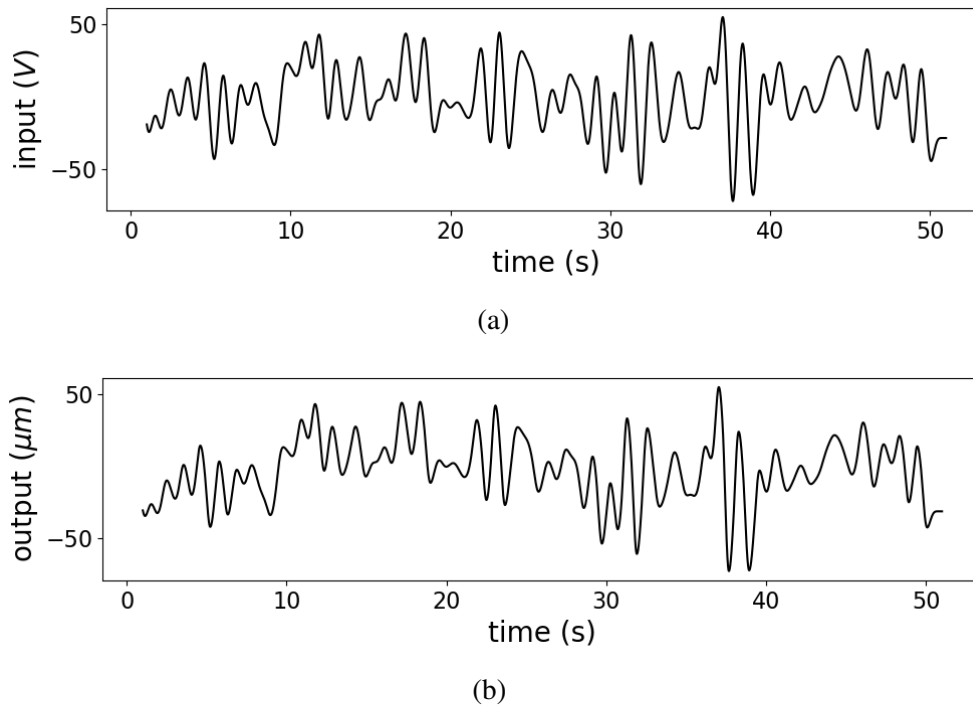
Martins and Aguirre (2016) ensure that the identified model has at least one equilibrium point under loading-unloading inputs so that Property 1 is attained. To do so, the following general type model is used:

$$y_k = F^l(y_{k-1}, \dots, y_{k-n_y}, \dots, u_{k-1}, \dots, u_{k-n_u}, \phi_{1,k-1}, \phi_{2,k-1}) \quad (5.5)$$

where $\phi_{1,k} = u_k - u_{k-1}$, $\phi_{2,k} = \text{sign}(\phi_{1,k})$ and F^l is a polynomial function of nonlinearity degree l , with $\phi_{2,k} = 1$ for loading and $\phi_{2,k} = -1$ for unloading.

According to Abreu et al. (2020), Property 1 is satisfied if the model has a *continuum of equilibrium points* (MORRIS, 2011), which can be verified by taking $y[k] = \bar{y}$, $\forall k$, $u[k] = \bar{u}$, $\forall k$ and get $\bar{y} = \Sigma_y \bar{y}$, where Σ_y is the sum of all parameters of all linear output regressors and equals 1 ($\Sigma_y = 1$).

Figure 5.11 – Training Data: piezoelectric actuator



5.4.1 Data set

Consider the piezoelectric actuator with hysteretic nonlinearity modeled by the Bouc-Wen model (WEN, 1976) and whose mathematical model is given by (RAKOTONDRABE, 2011):

$$\begin{cases} \dot{h}(t) = A\dot{u}(t) - \beta|\dot{u}(t)|h(t) - \gamma\dot{u}(t)|h(t)|, \\ y(t) = d_p u(t) - h(t) \end{cases} \quad (5.6)$$

where $y(t)$ is the displacement, $u(t)$ is the voltage applied to the actuator, $d_p = 1.6 \frac{\mu m}{V}$ is the piezoelectric coefficient, $h(t)$ is the hysteretic nonlinear term and $A = 0.9 \frac{\mu m}{V}$, $\beta = 0.008 V^{-1}$ and $\gamma = 0.008 V^{-1}$ are parameters that determine the shape and scale of the hysteresis loop.

The excitation signal is generated by low-pass filtering white Gaussian noise. A fifth-order low-pass Butterworth filter with a cutoff frequency of 1 Hz is used. The sampling time is set to $T_s = 0.001$ s, and the data set is 50 s long (50000 samples), as shown in Figure 5.11. This signal is used as training data. For the validation data, the system is excited by $u(t) = 40 \sin(2\pi t)$ V. These parameters used to build the data set are the same as the ones used in Abreu et al. (2020).

Script 5.4 – Piezoelectric actuator identification experiment

```

1  def evaluate(ind):
2      try:
3          element.constraint_funcs(ind,['sgn','phi'],
4                                     ['mul','sgn','phi'],terminals)
5          element.compile_model(ind)
6          if random.random()<0.2:
7              element.ols(ind,random.random()*1e-7,y,u)
8              theta = element.ls(ind,y,u)
9              return element.score_ksa(ind,theta,k=500,y,u),
10         except np.linalg.LinAlgError as e:
11             return np.inf,
12
13 def sgn(x1,x2):
14     return np.sign(x1-x2)
15 def phi(x1,x2):
16     return np.subtract(x1,x2)
17 element = mggpElement()
18 element.addPrimitive(sgn,2)
19 element.addPrimitive(phi,2)
20
21 mggp = mggpEvolver(popSize=500,CXPB=0.8,MTPB=0.2,
22                   n_gen=20,maxHeight=5,maxTerms=30,
23                   elite=10,element=element)
24 y,u = trainingData()
25 terminals = element.getTerminalsObjects()
26 seed = None
27 for i in range(100):
28     hof = mggp.run(evaluate=evaluate,seed=seed)
29     best = element.model2List(hof[0])
30     element.save("fileName.pkl",best)
31     seed = hof.items

```


5.4.2 Algorithm setup

Since the dataset is composed of a very large number of samples, the multiple shooting simulation error is used in the cost function. In this case, the dataset is divided into batches of size $500 + \max(n_u + n_y)$, which means that each individual has different batch size since individuals can have different maximum lags. Additionally, based on Martins and Aguirre (2016), two extra functions are added to the algorithm: $\text{subtract}(\phi_1)$ and $\text{sign}(\phi_2)$, both of which receive lagged variables as arguments. Besides, two different constraints are tested: i) the functions' arguments are *any* of the lagged variables (u_k and y_k), this is considered here as a *black-box* identification approach and ii) the arguments are just input terms (u_k), this is considered a *gray-box* identification approach since it constraints the model structure as suggested in Martins and Aguirre (2016), Abreu et al. (2020).

The parameters of the algorithm are set as follows:

```

population size = 500; generations = *; elitism = 10%;
maximum GP height = 3; maximum number of MGGP terms = 20;
CXPB = 0.8; MTPB = 0.2 (if not crossover); OLSPB = 0.2;
OLS tolerance = random()*1e-7; delay functions = q1, q2, q3;
primitive functions = multiplication, sign, subtraction;
parameter estimation = LS;
fitness function = multiple shooting
                    simulation interval of 500 steps.

```

The algorithm is run until the minimum fitness value stops improving (manual command, which is possible after every 20 generations). The sign function is modified to receive as arguments two variables that are subtracted from each other: $\text{sign}(\text{subtract}(x1, x2))$, for instance, $\text{sign}(u[k - 2] - u[k - 4])$, $\text{sign}(u[k - 6] - u[k - 3])$. Martins and Aguirre (2016) claim that the sign of the first difference of the input in addition to polynomial terms is a sufficient condition to reproduce hysteresis. In this experiment, the algorithm is free to assemble sign functions of any difference of arguments, or even to define whether the sign function is used or not.

The Script 5.4 present a code to perform this experiment using the MGGP toolbox. The function `constraint_funcs()` present in the evaluation function is responsible to restrain the argu-

ments of $sgn()$ and $phi()$ functions, which must receive as arguments only terminals. The multiple shooting cost function is performed by the function $score_ksa()$. For more details see Appendix A.

5.4.3 Results and Discussion

In this experiment, two models were identified by the MGGP algorithm, one from black-box identification (model $MGGP_{black}$ (5.7)) and another from gray-box identification (model $MGGP_{gray}$ (5.8)) (in which model structure is constrained):

- Black-box approach

$$\begin{aligned}
y_{black}[k] = & \theta_1 sign(y[k-3] - y[k-2]) + \theta_2 y[k-3]u[k-4]u[k-5] + \theta_3 u[k-1] + \theta_4 u[k-3] + \theta_5 y[k-1] \\
& + \theta_6 y[k-3]y[k-5]sign(y[k-2] - y[k-3]) + \theta_7 u[k-5]u[k-6]sign(y[k-2] - y[k-3]) \\
& + \theta_8 y[k-2]u[k-3]u[k-4] + \theta_9 u[k-4]u[k-5]sign(y[k-2] - y[k-3]) + \theta_{10} u[k-2]^3 \\
& + \theta_{11} y[k-4]^2 u[k-3]sign(y[k-2] - y[k-3]) + \theta_{12} u[k-3]u[k-4]sign(y[k-2] - y[k-3]) \\
& + \theta_{13} u[k-1]sign(y[k-1] - y[k-2]) + \theta_{14} y[k-4]u[k-4]sign(y[k-2] - y[k-3]) \\
& + \theta_{15} y[k-2](y[k-2] - u[k-4])sign(y[k-2] - y[k-3]) + \theta_{16} u[k-6] \\
& + \theta_{17} y[k-2]u[k-5]sign(y[k-2] - y[k-3]) \\
& + \theta_{18} u[k-2]u[k-3]u[k-5] + \theta_{19} y[k-1]y[k-3]u[k-2]
\end{aligned} \tag{5.7}$$

- Gray-box approach

$$\begin{aligned}
y_{gray}[k] = & \theta_1 y[k-5]^2(u[k-1] - u[k-2]) + \theta_2 y[k-4]u[k-4](u[k-1] - u[k-2]) + \theta_3 y[k-1] \\
& + \theta_4 sign(u[k-3] - u[k-7]) + \theta_5 u[k-1]u[k-2](u[k-4] - u[k-7]) \\
& + \theta_6 y[k-1]u[k-6] + \theta_7 y[k-1](u[k-6] - u[k-4])sign(u[k-4] - u[k-5]) \\
& + \theta_8 y[k-2]u[k-2] + \theta_9 u[k-1]u[k-3](u[k-6] - u[k-4]) \\
& + \theta_{10} sign(u[k-1] - u[k-6]) + \theta_{11} (u[k-3] - u[k-1]) \\
& + \theta_{12} u[k-2] + \theta_{13} u[k-4] + \theta_{14} sign(u[k-5] - u[k-4]) \\
& + \theta_{15} u[k-1](u[k-6] - u[k-4])sign(u[k-4] - u[k-1])
\end{aligned} \tag{5.8}$$

Figures 5.12 and 5.13 present the free-run validation for quasi-static input and for the cases in which the input becomes constant during *loading* and *unloading*. Table 5.5 presents the MSE measures for the training and test data of both models.

Table 5.4 – Piezoelectric actuator models parameters (obtained via LS)

Model	Values		
$MGGP_{black}$	$\theta_1 = 7.18e-4$	$\theta_2 = 1.11e-4$	$\theta_3 = 0.6633$
	$\theta_4 = -0.8556$	$\theta_5 = 1.0000$	$\theta_6 = 2.31e-3$
	$\theta_7 = 2.24e-2$	$\theta_8 = -1.10e-4$	$\theta_9 = -4.87e-2$
	$\theta_{10} = 1.01e-4$	$\theta_{11} = 7.78e-09$	$\theta_{12} = 2.63e-2$
	$\theta_{13} = -9.42e-06$	$\theta_{14} = -4.49e-3$	$\theta_{15} = -2.30e-3$
	$\theta_{16} = 0.1923$	$\theta_{17} = 2.16e-3$	$\theta_{18} = -1.01e-4$
	$\theta_{19} = -2.43e-07$		
$MGGP_{gray}$	$\theta_1 = 2.35e-4$	$\theta_2 = -7.70e-4$	$\theta_3 = 1.0000$
	$\theta_4 = 4.24e-4$	$\theta_5 = -1.75e-3$	$\theta_6 = 7.15e-06$
	$\theta_7 = 4.17e-3$	$\theta_8 = -7.13e-06$	$\theta_9 = -2.93e-3$
	$\theta_{10} = -6.04e-4$	$\theta_{11} = -0.9500$	$\theta_{12} = -0.5749$
	$\theta_{13} = 0.5749$	$\theta_{14} = -6.27e-05$	$\theta_{15} = 6.84e-3$

The models yielded low SE for quasi-static validation as we can see in the Table 5.5 and in figures 5.13(a) and 5.14(a). However, for the cases where the input u becomes constant during *loading* and *unloading*, their free-run simulation present a decreasing value (figures 5.13(d), 5.13(e), 5.14(d) and 5.14(e)), which means that the models do not have a continuum of equilibrium points and do not satisfy Property 1.

Model $MGGP_{gray}$ (5.8) performs better than model $MGGP_{black}$ (5.7) for both training (MSE of $0.0458 \mu\text{m}$ and of $0.0588 \mu\text{m}$) and validation data (MSE of $0.0201 \mu\text{m}$ and of $0.0395 \mu\text{m}$, respectively). In order to rectify the absence of continuum of equilibrium points, the parameters of the gray-box model were fine tuned. Consider the case of constant input and constant output in which $u[k] = \bar{u}$ and $y[k] = \bar{y}$. Model $MGGP_{gray}$ (5.8) becomes:

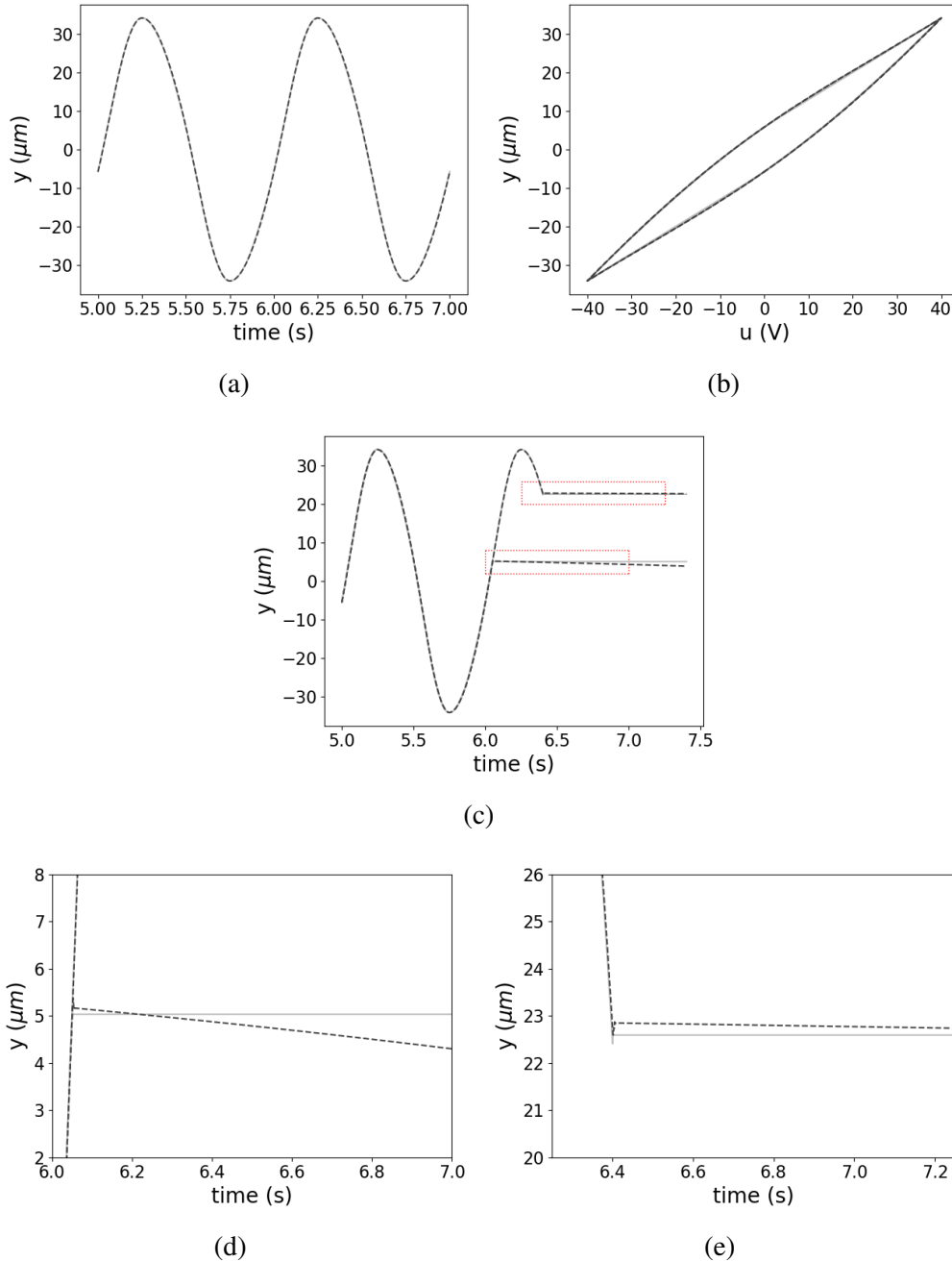
$$y[k] = \theta_3 \bar{y} + \theta_6 \bar{y} \bar{u} + \theta_8 \bar{y} \bar{u} + \theta_{12} \bar{u} + \theta_{13} \bar{u}. \quad (5.9)$$

The *continuum of equilibrium points* is present if $y[k] = 1 \cdot y[k-1]$. Considering (5.9), the following restrictions must be made: $\Sigma_y = \theta_3 = 1$, $\Sigma_{yu} = \theta_6 + \theta_8 = 0$ and $\Sigma_u = \theta_{12} + \theta_{13} = 0$. A

Table 5.5 – Piezoelectric models' free-run simulation errors

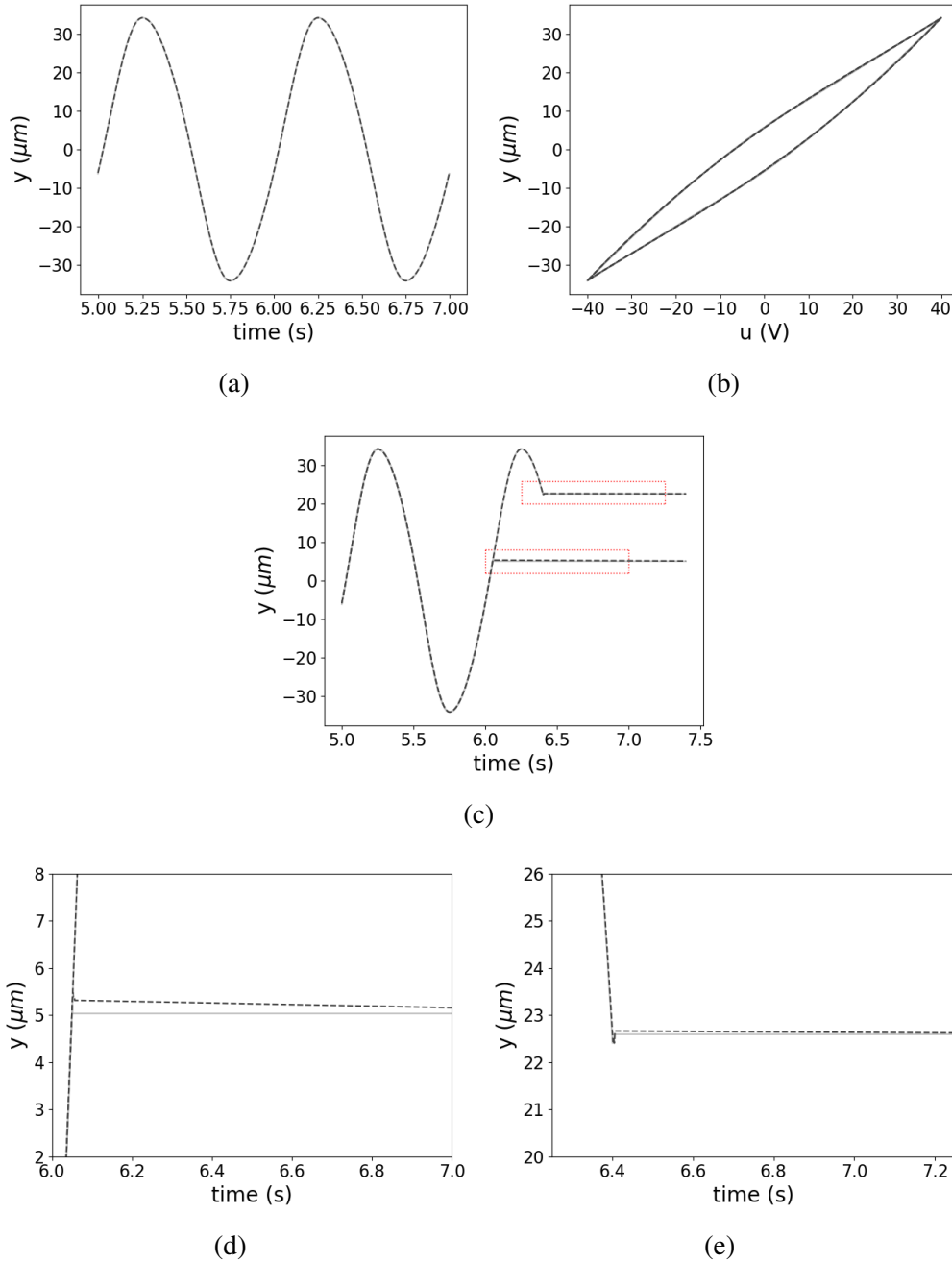
Model	MSE (train)	MSE (test)
$MGGP_{black}$ (5.7)	0.0588	0.0395
$MGGP_{gray}$ (5.8)	0.0458	0.0201
$MGGP_{gray}$ (5.8) - <i>CLS</i>	0.0473	0.0205

Figure 5.12 – Black-box identified model’s free-run validation for quasi-static input data. (a) $y \times t$, (b) $y \times u$ and (c) $y \times t$ for the cases in which input u becomes constant during *loading* (d) and *unloading* (e). Dashed lines represent MGGP model simulation.



new set of parameters is obtained by re-estimating them via CLS (Table 5.6). The errors for the new set of parameters are shown in Table 5.5. The free-run simulation for quasi-static input, which becomes constant during loading and unloading, is shown in Figure 5.14. As one can note, it has a

Figure 5.13 – Gray-box identified model’s free-run validation for quasi-static input data. (a) $y \times t$, (b) $y \times u$ and (c) $y \times t$ for the cases in which input u becomes constant during loading (d) and unloading (e). Dashed lines represent MGGP model simulation.



constant output. Now, Model $MGGP_{gray}$ (5.8) has a continuum of equilibrium points and satisfies Property 1.

Figure 5.14 – Gray-box identified model’s free-run simulation for quasi-static input which becomes constant during loading (a) and unloading (b) after constraining parameter estimation. Dashed lines represent MGGP model simulation.

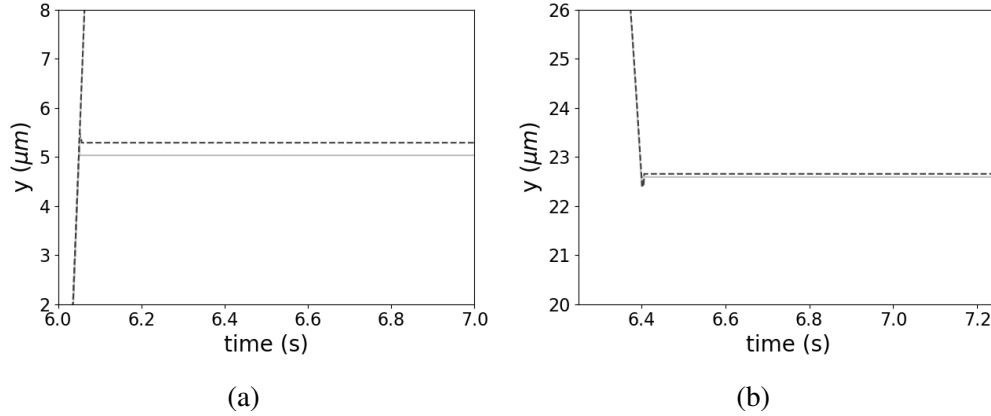


Table 5.6 – Piezoelectric actuator gray-box model’s parameters (obtained via CLS)

Model	Values		
$MGGP_{gray-CLS}$	$\theta_1 = 2.41e-4$	$\theta_2 = -7.82e-4$	$\theta_3 = 1.0000$
	$\theta_4 = 3.99e-4$	$\theta_5 = -1.76e-3$	$\theta_6 = 7.98e-06$
	$\theta_7 = 4.08e-3$	$\theta_8 = -7.98e-06$	$\theta_9 = -2.94e-3$
	$\theta_{10} = -5.86e-4$	$\theta_{11} = -1.0154$	$\theta_{12} = -0.6402$
	$\theta_{13} = 0.6402$	$\theta_{14} = -5.63e-05$	$\theta_{15} = 6.74e-3$

Remark 3 We highlight that the MGGP/ERR algorithm selected regressors in such a way that it is possible to cancel the nonlinear clusters Σ_{yu} and Σ_u .

6 CONCLUSION

This work addressed the problem of nonlinear system identification using evolutionary algorithms with focus on the model structure selection topic. An introductory material on system identification and evolutionary algorithms is presented, including some of the main works on this interdisciplinary topic. The chosen algorithm to work with is the MGGP. It uses a very flexible individual representation, which allows fluctuating chromosome sizes in the same population and automatic time lag determination.

It is proposed a hybrid MGGP/ERR algorithm to solve NARMAX models structure selection problem. The algorithm is first applied in a real life hydraulic pump identification problem. The MGGP/ERR presented good performance in pursuing the least prediction error (PE) and free-run simulation error (SE). Also, the algorithm showed the capacity to explore regions in the regressors space that is highly costly in terms of memory usage for the traditional OLS/ERR algorithm. The use of genetic programming (GP) in NARMAX terms representation granted the possibility to include functions in the nodes set that allowed the algorithm to identify a simulated hysteretic system. Multiple-shooting simulation and gray-box framework were used in this experiment. Finally, in order to better understand the MGGP/ERR behavior, some stochastic test systems identification were performed. The hybridization showed itself beneficial in the soft input identification problem. Results show that models obtained by MGGP/ERR yield better PE, are more stable in free-run simulation and more parsimonious than the ones obtained by standalone MGGP. A disadvantage of MGGP/ERR compared to standalone OLS/ERR is that the former consumes considerably more processing time. Nonetheless, it is extremely useful for offline applications.

A very relevant contribution of this work is the development of a toolbox for research purpose named *mggp*. It is a python package that includes the framework used in the aforementioned experiments. A tutorial on *how to use* is presented in Appendix A.

Future works include the toolbox enhancement, *i.e.*, implement an adaptive mutation probability, other types of mutation and crossover operators and a multi-objective framework. Moreover, identify real life hysteretic systems for control purposes.

BIBLIOGRAPHY

- ABREU, P. E. et al. Identification and nonlinearity compensation of hysteresis using narx models. **Nonlinear Dynamics**, Springer, v. 102, n. 1, p. 285–301, 2020.
- AGUIRRE, L. A. **Introdução à Identificação de Sistemas - Técnicas Lineares e Não-Lineares: Teoria e Aplicação**. 4^a. Belo Horizonte: Editora UFMG, 2015.
- AGUIRRE, L. A.; BARBOSA, B. H.; BRAGA, A. P. Prediction and simulation errors in parameter estimation for nonlinear systems. **Mechanical Systems and Signal Processing**, v. 24, n. 8, p. 2855 – 2867, 2010.
- AGUIRRE, L. A.; BILLINGS, S. A. Dynamical effects of overparametrization in nonlinear models. **Physica D: Nonlinear Phenomena**, Elsevier, v. 80, n. 1-2, p. 26–40, 1995.
- AGUIRRE, L. A.; LETELLIER, C. Modeling nonlinear dynamics and chaos: a review. **Mathematical Problems in Engineering**, Hindawi, v. 2009, 2009.
- AKAIKE, H. A new look at the statistical model identification. **IEEE Transactions on Automatic Control**, v. 19, n. 6, p. 716–723, 1974.
- BAKER, J. E. Reducing bias and inefficiency in the selection algorithm. In: **Proceedings of the second international conference on genetic algorithms**. United States: Taylor & Francis, 1987. v. 206, p. 14–21.
- BARBOSA, B. H. G. et al. Black and gray-box identification of a hydraulic pumping system. **Control Systems Technology, IEEE Transactions on**, v. 19, n. 2, p. 398 –406, march 2011.
- BILLINGS, S.; AGUIRRE, L. Effects of the sampling time on the dynamics and identification of nonlinear models. **International Journal of Bifurcation and Chaos**, v. 5, p. 1541–1556, 12 1995.
- BILLINGS, S. A.; CHEN, S. Identification of non-linear rational systems using a prediction-error estimation algorithm. **International Journal of Systems Science**, Taylor & Francis, v. 20, n. 3, p. 467–494, 1989.
- BILLINGS, S. A.; CHEN, S.; KORENBERG, M. J. Identification of mimo non-linear systems using a forward-regression orthogonal estimator. **International Journal of Control**, Taylor & Francis, v. 49, n. 6, p. 2157–2189, 1989.
- BILLINGS, S. A.; TAO, Q. H. Model validity tests for non-linear signal processing applications. **International Journal of Control**, Taylor & Francis, v. 54, n. 1, p. 157–194, 1991.
- BILLINGS, S. A.; ZHU, Q. M. Nonlinear model validation using correlation tests. **International Journal of Control**, Taylor & Francis, v. 60, n. 6, p. 1107–1120, 1994.
- BRAGA, A. d. P.; CARVALHO, A.; LUDERMIR, T. **Redes neurais artificiais: teoria e prática**. Editora LTC, 2000.
- CHEN, Q. et al. Genetic algorithm with an improved fitness function for (n) arx modelling. **Mechanical Systems and Signal Processing**, Elsevier, v. 21, n. 2, p. 994–1007, 2007.

CHEN, S.; BILLINGS, S. A.; LUO, W. Orthogonal least squares methods and their application to non-linear system identification. **International Journal of control**, Taylor & Francis, v. 50, n. 5, p. 1873–1896, 1989.

COELHO, M. d. S. **Modelos de Hammerstein e de Wiener: conexões com modelos narx e sua aplicação em identificação de sistemas não-lineares**. Phd Thesis (PhD Thesis) — Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica, 2002.

CORREIA, M. **Identificação caixa-cinza de sistemas não-lineares utilizando representações NARMAX polinomiais e racionais**. Phd Thesis (PhD Thesis) — PhD thesis, Universidade Federal de Minas Gerais.(PPGEE), 2001.

DE, J. Ka an analysis of the behavior of a class of genetic adaptative systems. **Ann Arbor, USA, Ph. D Thesis-Department of Computer and Communication Sciences, University of Michigan**, 1975.

DENG, L.; TAN, Y. Modeling hysteresis in piezoelectric actuators using narmax models. **Sensors and Actuators A: Physical**, Elsevier, v. 149, n. 1, p. 106–112, 2009.

EIBEN, A. E.; SMITH, J. E. et al. **Introduction to evolutionary computing**. Bristol, UK: Springer, 2003.

FALSONE, A.; PIRODDI, L.; PRANDINI, M. A randomized algorithm for nonlinear model structure selection. **Automatica**, v. 60, p. 227 – 238, 2015.

GHAREEB, W. T.; SAADANY, E. F. E. Multi-Gene Genetic Programming for Short Term Load Forecasting. In: **2013 3rd International Conference on Electric Power and Energy Conversion Systems (EPECS)**. Istanbul, Turkey: IEEE, 2013. (International Conference on Electric Power and Energy Conversion Systems).

GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. **Machine learning**, Springer, v. 3, n. 2, p. 95–99, 1988.

HAYKIN, S. **Redes neurais: princípios e prática**, 2ª edição, tradução: Paulo martins engel. **Editora: Bookman, Porto Alegre, Cap**, v. 1, n. 2, p. 3, 2001.

HERRERA, F.; LOZANO, M.; VERDEGAY, J. L. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. **Artificial intelligence review**, Springer, v. 12, n. 4, p. 265–319, 1998.

HINCHLIFFE, M. et al. **Modelling chemical process systems using a multi-gene genetic programming algorithm**. 1996. 56–65 p.

HINCHLIFFE, M. P. **Dynamic modelling using genetic programming**. Phd Thesis (PhD Thesis), University of Newcastle upon Tyne, UK, 2001.

HINCHLIFFE, M. P.; WILLIS, M. J. Dynamic systems modelling using genetic programming. **Computers & Chemical Engineering**, v. 27, n. 12, p. 1841 – 1854, 2003.

HOLLAND, J. H. Adaptation in natural and artificial systems. **The University of Michigan Press**, 1975.

KOZA, J. R. **Genetic programming: on the programming of computers by means of natural selection**. London: MIT press, 1992.

LEONTARITIS, I.; BILLINGS, S. A. Input-output parametric models for non-linear systems part i: deterministic non-linear systems. **International journal of control**, Taylor & Francis, v. 41, n. 2, p. 303–328, 1985.

LI, C. J.; JEON, Y. Genetic algorithm in identifying non linear auto regressive with exogenous input models for non linear systems. In: **IEEE. 1993 American Control Conference**. San Francisco, CA, USA, 1993. p. 2305–2309.

LINDEN, R. **Algoritmos genéticos**. Rio de Janeiro: Brasport, 2008.

LJUNG, L. **System Identification: Theory for the User**. 2nd edition. United States: Prentice Hall, 1999.

LUKE, S.; SPECTOR, L. A comparison of crossover and mutation in genetic programming. **Genetic Programming**, v. 97, p. 240–248, 1997.

MADAR, J.; ABONYI, J.; SZEIFERT, F. Genetic programming for the identification of nonlinear input - Output models. **Industrial & Engineering Chemistry Research**, 44, n. 9, p. 3178–3186, 2005.

MARTINS, S. A. M.; AGUIRRE, L. A. Sufficient conditions for rate-independent hysteresis in autoregressive identified models. **Mechanical Systems and Signal Processing**, Elsevier, v. 75, p. 607–617, 2016.

MEHR, A. D.; KAHYA, E. A pareto-optimal moving average multigene genetic programming model for daily streamflow prediction. **Journal of hydrology**, Elsevier, v. 549, p. 603–615, 2017.

MORRIS, K. What is hysteresis? **Applied Mechanics Reviews**, American Society of Mechanical Engineers Digital Collection, v. 64, n. 5, 2011.

PEARSON, R. K.; POTTMANN, M. Gray-box identification of block-oriented nonlinear models. **Journal of Process Control**, Elsevier, v. 10, n. 4, p. 301–315, 2000.

PENG, J.; CHEN, X. A survey of modeling and control of piezoelectric actuators. **Modern Mechanical Engineering**, Scientific Research Publishing, v. 3, n. 1, 2013.

PIRODDI, L.; SPINELLI, W. An identification algorithm for polynomial narx models based on simulation error minimization. **International Journal of Control**, Taylor & Francis, v. 76, n. 17, p. 1767–1781, 2003.

POLI, R. et al. **A field guide to genetic programming**. Lulu.com, 2008. Available at: <<http://www.gp-field-guide.org.uk>>.

RADCLIFFE, N. J. Equivalence class analysis of genetic algorithms. **Complex systems**, v. 5, n. 2, p. 183–205, 1991.

RAKOTONDRABE, M. Bouc–wen modeling and inverse multiplicative structure to compensate hysteresis nonlinearity in piezoelectric actuators. **IEEE Transactions on Automation Science and Engineering**, v. 8, n. 2, p. 428–431, 2011.

RIAHI-MADVAR, H. et al. Pareto Optimal Multigene Genetic Programming for Prediction of Longitudinal Dispersion Coefficient. **Water Resources Management**, 33, n. 3, p. 905–921, 2019.

RIBEIRO, A. H. et al. On the smoothness of nonlinear system identification. **Automatica**, v. 121, p. 109158, 2020.

RUGH, W. J. **Nonlinear system theory : the Volterra/Wiener approach**. United States: Johns Hopkins University Press, 1983.

SAFARI, M. J. S.; MEHR, A. D. Multigene genetic programming for sediment transport modeling in sewers for conditions of non-deposition with a bed deposit. **International Journal of Sediment Research**, 33, n. 3, p. 262–270, 2018.

SJÖBERG, J. et al. Nonlinear black-box modeling in system identification: a unified overview. **Automatica**, Elsevier, v. 31, n. 12, p. 1691–1724, 1995.

WEN, Y.-K. Method for random vibration of hysteretic systems. **Journal of the engineering mechanics division**, American Society of Civil Engineers, v. 102, n. 2, p. 249–263, 1976.

WIENER, N. Nonlinear problems in random theory. **Nonlinear Problems in Random Theory**, by Norbert Wiener, pp. 142. ISBN 0-262-73012-X. Cambridge, Massachusetts, USA: The MIT Press, August 1966.(Paper), p. 142, 1966.

WIGREN, T. Recursive prediction error identification using the nonlinear wiener model. **Automatica**, Elsevier, v. 29, n. 4, p. 1011–1025, 1993.

Willis, M. J. et al. Artificial neural networks in process engineering. **IEE Proceedings D - Control Theory and Applications**, v. 138, n. 3, p. 256–266, 1991.

YOUNG, P. C. The use of linear regression and related procedures for the identification of dynamic processes. In: IEEE. **Seventh Symposium on Adaptive Processes**. Los Angeles, CA, USA, 1968. p. 53–53.

A THE MGGP TOOLBOX

A.1 `mggpElement` Class

A *mggpElement* object is responsible to carry the attributes and functions used to create and evaluate individuals from a MGGP population.

It is able to build MISO models. Its default configuration creates an element object capable of building SISO models which variables are 'y1', 'u1' and (optional) 'e1'. The number '1' in the variable name indicates that it is a one-step lagged variable ($y1 = y[k-1]$). Also, the only function present in the primitive set is 'mul' with arity equals 2, that is, it receives two arguments – `mul(x1,x2)`.

Three parameters should be set in a `mggpElement` object:

- `maxDelay` = corresponding to the maximum number of back-shift operator to be included into the primitive set. For example:
 $maxDelay = 3 \rightarrow \{q1, q2, q3\}$ (default)
 $maxDelay = 5 \rightarrow \{q1, q2, q3, q4, q5\}$ and so on.
- `MA` = this parameter enables the use of the variable 'e1' that represents residual terms. That means, when it is set `True`, the functions related to extended least squares and (extended) one-step-ahead prediction will depend on the terms of 'e1'. If it is set `False` (default), those functions will work as a white noise output error problem.
- `constant` = if it is set `'True'` it enables the terminal '1' and the individuals are allowed to have constant term. Otherwise, if it is set `'False'` (default), the terminal '1' is not included into the primitive set.

Those parameters can be changed using the function

```
element.setPset(maxDelay, numberOfVariables, MA, constant)
```

A.1.1 Create SISO Model

The *mggpElement Class* possesses the function *createModel(listStrings)* that receives as argument a list of string in which each string is a term of the model

Consider the following NARX system (Piroddi, 2003):

$$y(k) = 0.75y(k-2) + 0.25u(k-1) - 0.20y(k-2)u(k-1)$$

Lets create an element object with the default parameters. Then create a model object with the variables of the aforementioned system.

```
from mggp import mggpElement

element = mggpElement()
listStrings = ['q1(y1)', 'u1', 'mul(q1(y1), u1)']
model = element.createModel(listStrings)
element.compile_model(model)
```

If the user wants to print the model terms in the console, just do:

```
for term in model:
    print(str(term))
```

Note: If it is needed, the *createModel()* function sets an attribute 'lagMax' in the model object which contains the maximum lag applied by back-shift operators. That means, the real model maximum lag is *maximumLag* = model.lagMax+1. The example model has a model.lagMax = 1, and the maximum lag of the model is 2.

A.1.2 Create MISO Model with Constant Term

Consider the following NARX MISO system, that has two inputs *u* and *h*.

$$y(k) = 0.75y(k-2) + 0.25u(k-1) - 0.20y(k-2)u(k-1) - 0.5h(k-2) + 0.1$$

Now, the number of variables is 3 and it has a constant term. The user can change the name of the arguments using the function

```
element.renameArguments(dictionary)
```

The default names are 'ARG0', 'ARG1', 'ARG2', etc. **The first argument must always be the output.** The dictionary maps the default names to the new ones.

```

element = mggpElement()
element.setPset(maxDelay=3,numberOfVariables=3,MA=False,
               constant=True)
element.renameArguments({'ARG0':'y1','ARG1':'u1','ARG2':'h1'})
listStrings = ['q1(y1)', 'u1', 'mul(q1(y1),u1)', 'q1(h1)', '1']
model = element.createModel(listStrings)
element.compile_model(model)

```

Note¹: It is advised to name the arguments with the suffix '1'. It helps the user to remember that the variable is a one-step lagged variable.

Note²: The constant term is always represented as '1' in the *listStrings* argument. Its value will be determined by the parameter θ value.

A.1.3 Simulate a System

To simulate the aforementioned SISO model as a system use the free-run simulation function

```
element.predict_freeRun(model,theta,y0,*inputs)
```

No matter the size of the initial conditions y_0 you use, the function will work only with the size of the maximum lag in your model. For example, our model has a maximum lag equals 2, y_0 must have at least size 2.

Lets say our input is 100 Gaussian distributed random values with zero mean and standard deviation 1 and use the previous SISO model already built.

```
u = np.random.normal(loc=0,scale=1,size=(100))
y0 = np.zeros((2))
theta = np.array([0.75,0.25,-0.20])
y = element.predict_freeRun(model,theta,y0,u[:-1])
```

Note¹: The model must be compiled in order to identify the terms functions.

Note²: $u[:-1]$ is used to neglect the last sample of u in the prediction so that y and u have the same sizes.

To plot the output just do:

```
import matplotlib.pyplot as plt
plt.plot(y)
plt.title('Simulation of the example model')
```

A.1.3.1 Simulate a System with White Noise Equation Error

$$y(k) = 0.75y(k-2) + 0.25u(k-1) - 0.20y(k-2)u(k-1) + v(k)$$

```
element = mggpElement()
element.setPset(maxDelay=3,numberOfVariables=3,MA=False,
```

```
        constant=False)
element.renameArguments({'ARG0':'y1','ARG1':'u1','ARG2':'v'})
listStrings = ['q1(y1)', 'u1', 'mul(q1(y1),u1)', 'v']
model = element.createModel(listStrings)
element.compile_model(model)
y0 = np.zeros((2))
u = np.random.normal(loc=0, scale=1, size=(100))
v = np.random.normal(loc=0, scale=0.02, size=(100))
theta = np.array([0.75, 0.25, -0.20, 1])
y = element.predict_freeRun(model, theta, y0, u[:-1], v[:-1])
```


A.1.3.2 Simulate a System with Colored Noise Equation Error

$$y(k) = 0.75y(k-2) + 0.25u(k-1) - 0.20y(k-2)u(k-1) + 0.8v(k-1) + v(k)$$

```

element = mggpElement()
element.setPset(maxDelay=3,numberOfVariables=3,MA=False,
                constant=False)
element.renameArguments({'ARG0':'y1','ARG1':'u1','ARG2':'v'})
listStrings = ['q1(y1)','u1','mul(q1(y1),u1)','q1(v)','v']
model = element.createModel(listStrings)
element.compile_model(model)
y0 = np.zeros((2))
u = np.random.normal(loc=0,scale=1,size=(100))
v = np.random.normal(loc=0,scale=0.02,size=(100))
theta = np.array([0.75,0.25,-0.20,0.8,1])
y = element.predict_freeRun(model,y0,theta,u[:-1],v[:-1])

```

Note: Although in the example the noise variable v do not have the suffix '1', this delay is applied on the 'regressor'. However, the noise variable is temporary. Experimental models do not have it as argument. So, it can be interpreted as a non-lagged variable.

A.1.4 The Least Squares Functions

- `element.ls(model,y,*inputs)`

Example:

For the SISO model:

```
theta = element.ls(model,y,u)
```

For the MISO model:

```
theta = element.ls(model,y,u,v)
```

- `element.ls_extended(model, y, *inputs)`

Example:

For the SISO model:

```
theta = element.ls_extended(model, y, u)
```

For the MISO model:

```
theta = element.ls_extended(model, y, u, v)
```

Note: If the MA parameter is set 'True', the ELS will extend the regressor matrix with the MA part of the model. On the other hand, if it is set 'False', the extension is made as it was a white noise output error problem.

A.1.5 Predictors

There are four predictors in the toolbox:

- `element.predict_freeRun(model, theta, y0, *inputs)`

Returns the free-run simulation with the initial conditions in the beginning.

- `element.predict(model, theta, y, *inputs)`

Returns the one-step-ahead prediction (without initial conditions in the beginning)

- `element.predict_extended(model, theta, y, *inputs)`

Returns the one-step-ahead prediction (without initial conditions in the beginning) from the extended regressor matrix. If the MA parameter is set 'False', the extension is made as it was a white noise output error problem.

- `element.predict_ksa(model, theta, k, y, *inputs)`

Returns a tuple with the k-steps-ahead prediction (with initial conditions) and the 3-d batched array of y. They are 3-d arrays with the form (*batches, data, 1*). The number of batches are calculated dividing the data-set in several windows of (*k*+ model maximum lag) size. The remaining data is discarded.

It can be confusing how to compare the predicted value with the desired one. There are two ways for the user to remove the initial conditions from the *y* vector:

```
yd = y[model.lagMax+1:]
```

and the built-in function

```
yd = element.getDesiredY(model, y)
```

This function also works in the 3-d array from *ksa* analysis.

A.1.6 MSE built-in scores

- `element.score_osa(model, theta, y, *inputs)`
- `element.score_osa_ext(model, theta, y, *inputs)`
- `element.score_freeRun(model, theta, y, *inputs)`
- `element.score_ksa(model, theta, k, y, *inputs)`

A.1.7 Moving Average Models

The Moving Average configuration is activated by the MA argument in the *setPset* function. If it is set 'True' the *mggpElement* object automatically includes an extra variable named 'e1'. That means, the *numberOfVariables* argument should not take into account the residual term. Consider the following NARMAX model:

$$y(k) = \theta_1 y(k-2) + \theta_2 u(k-1) - \theta_3 y(k-2)u(k-1) + \theta_1^e e(k-1)$$

```

element = mggpElement ()
element.setPset (maxDelay=3, numberOfVariables=2, MA=True,
                constant=False)
element.renameArguments ({'ARG0' : 'y1', 'ARG1' : 'u1' })
listStrings = ['q1 (y1)', 'u1', 'mul (q1 (y1), u1)', 'e1']
model = element.createModel (listStrings)
element.compile_model (model)

theta = element.ls_extended (model, y, u)

ypred = element.predict_extended (model, theta, y, u)

```

Note: The free-run predictor for NARMAX models neglects the MA part.

A.1.8 Get Regressor Matrix

The user can get the regressor Matrix through the function:

```
element.makeRegressors (model, y, *inputs, **options)
```

There are two arguments in options that can be set. They are

- mode: {'default', 'extended'}
- theta: if mode is set to 'extended', the theta values must be sent.

Examples:

```
p = element.makeRegressors(model, y, u, mode='default')
p = element.makeRegressors(model, y, u, mode='extended', theta=theta)
```

Note: If MA is set 'True', the 'extended' mode must be used. If it is set 'False' the 'extended' mode will return a regressor matrix as it was a white noise output error problem.

A.1.9 Orthogonal Least Squares

The built-in *ols* function apply a classical Gram-Smith Orthogonal Least Squares structure selection method.

```
element.ols(model, tol, y, *inputs, **kwargs)
```

Example:

```
element = mggpElement()
element.setPset(maxDelay=3, numberOfVariables=2, MA=False, constant=False)
element.renameArguments({'ARG0': 'y1', 'ARG1': 'u1'})
listStrings = ['q1(y1)', 'u1', 'mul(q1(y1), u1)', 'y1', 'q1(u1)']
model = element.createModel(listStrings)
element.compile_model(model)
element.ols(model, 1e-3, y, u)
for term in model:
    print(str(term))
```

Note¹: If the MA mode is set 'True', the key argument 'theta' must be set. The algorithm will create a extended regressor matrix and apply ols pruning over it. Probably, all residual terms will be removed.

```
element.ols(model, tol, y, *inputs, theta=theta)
```

Note²: The *ols* function perform an in-place operation, that means, it modifies the object sent as argument.

Note³: The resultant model do not have its terms sorted by ERR coefficient.

A.1.10 Include New Functions

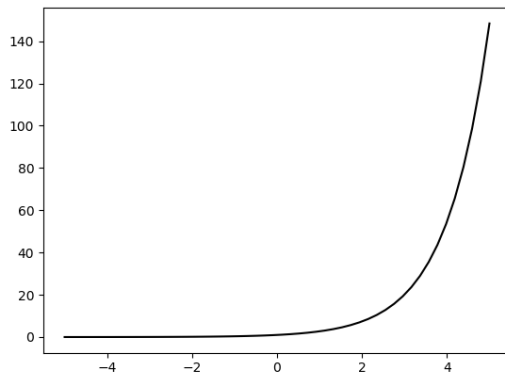
Other functions can be included into the primitive set through the function:

```
element.addPrimitive(function, arity)
```

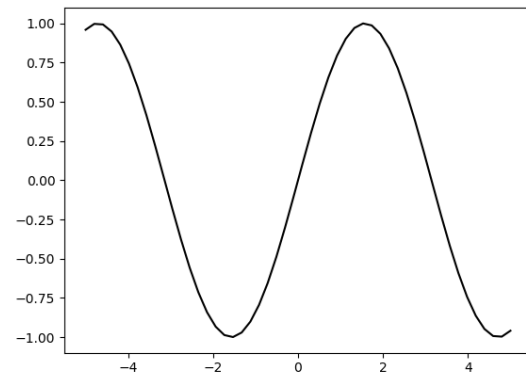
Note: Arity is the number of arguments the function takes.

Example 1: Exponential function

```
element = mggpElement()
element.setPset(maxDelay=1, numberOfVariables=2)
element.renameArguments({'ARG0': 'y1', 'ARG1': 'u1'})
element.addPrimitive(np.exp, 1)
listStrings = ['exp(u1)']
model = element.createModel(listStrings)
element.compile_model(model)
u = np.linspace(-5, 5)
y0 = np.zeros(1)
theta = np.array([1])
y = element.predict_freeRun(model, theta, y0, u)
plt.figure()
plt.plot(u, y[1:])
```



(a) Example 1



(b) Example 2

Example 2: Sinusoidal function

```

element = mggpElement()
element.setPset(maxDelay=1,numberOfVariables=2)
element.renameArguments({'ARG0':'y1','ARG1':'u1'})
element.addPrimitive(np.sin,1)
listStrings = ['sin(u1)']
model = element.createModel(listStrings)
element.compile_model(model)
u = np.linspace(-5,5)
y0 = np.zeros(1)
theta = np.array([1])
y = element.predict_freeRun(model, theta, y0, u)
plt.figure()
plt.plot(u,y[1:])

```

A.1.11 Handling constraints with built-in functions

There are cases in which the structure of a model has some restrictions. It is possible to create constraints in functions arguments through the function:

```

element.constraint_funcs(model,funcs,consts,values)

```

The arguments present in the list of string 'consts' will be removed from the functions present in the list of strings 'funcs' and replaced by values.

- model: model object to be constraint
- funcs: list of strings with functions names
- consts: list of strings with functions or arguments names. The latter must be the default names - 'ARG0', 'ARG1', etc
- values: list of terminals objects to be used as replacement

The terminals list can be gotten by the function:

```
terminals = element.getTerminalsObjects()
```

For example, if the user wants to limit the 'mul' function to be used only with 'u1' variables:

```
element.constraint_funcs(model, 'mul', 'ARG0', terminals[1])
```

Note: to check terminals names use

```
terminals[index].name
```

Example 1: include the *sign* function and restrain it to not have 'mul' nor 'sign' as arguments, and replace them by any terminal

```
def sgn(x1):
    return np.sign(x1)

element = mggpElement()
element.setPset(maxDelay=1, numberOfVariables=2, constant=True)
element.renameArguments({'ARG0': 'y1', 'ARG1': 'u1'})
element.addPrimitive(sgn, 1)
listStrings = ['sgn(y1)', 'sgn(mul(u1, u1))', 'sgn(sgn(u1))']
model = element.createModel(listStrings)
element.compile_model(model)
```



```

terminals = element.getTerminalsObjects()
element.constraint_1arityFuncs(model, ['sgn'],
                                ['mul', 'sgn'], terminals)

```

Example 2: include a *sign* function of arity 2 that return $sign(x1-x2)$ and restrain it to not have 'mul', 'sign' nor 'y1' as arguments, and replace them by 'u1'

```

def sgn(x1,x2):
    return np.sign(x1-x2)

element = mggpElement()
element.setPset(maxDelay=1,numberOfVariables=2,constant=True)
element.renameArguments({'ARG0':'y1','ARG1':'u1'})
element.addPrimitive(sgn,2)
listStrings = ['sgn(u1,y1)', 'sgn(u1,mul(u1,u1))',
               'sgn(sgn(y1,u1),y1)']
model = element.createModel(listStrings)
element.compile_model(model)
terminals = element.getTerminalsObjects()
element.constraint_2arityFuncs(model, ['sgn'],
                                ['mul', 'sgn', 'ARG0'], terminals[1])

```

A.2 Save and Load functions

To make it easier to the user, *mggpElement class* implements a save and a load functions (using *pickle* package).

```

element.save(filename,dictionary)

element.load(filename)

```

It can be used with dictionary objects. For example, if the user wants to save a model, a theta value, and training data:

```
modelListString = element.model2List(model)
dictionary = {'model':modelListString,
             'theta':theta,
             'y_train':y,
             'u_train':u}
element.save('modelInfo.pkl',dictionary)
```

Note: It is advised to save models as list of strings as the *Individual* instances can generate conflicts with another *base.creator* modules. The user can get it from the function:

```
element.model2List(model)
```

And to retrieve the saved objects:

```
dictionary = element.load('modelInfo.pkl')
modelListString = dictionary['model']
model = element.createModel(modelListString)
theta = dictionary['theta']
y_train = dictionary['y_train']
u_train = dictionary['u_train']
```

A.3 The *mggpEvolver* Class

The *mggpEvolver* class is responsible to execute the evolution of a population. The individuals from this population are created according to the primitive set defined in a *mggpElement* object. The following parameters can be set:

- *popSize*: population size (default = 100)
- *CXPB*: crossover probability (default = 0.9)
- *MTPB*: mutation probability (default 0.1)
- *n_gen*: number of generations (default = 50)
- *maxHeight*: maximum height of GP elements (default = 3)
- *maxTerms*: maximum number of model terms (default = 5)
- *elite*: percentage of the population to be included into the *hall of fame* object and be kept in the population (default = 5)
- *verbose*: print statistics at each generation (default = True)
- *element*: *mggpElement* object with the information needed to create individuals.

The *run* function have two arguments:

- *evaluate*: function which returns the individual fitness to be **minimized**. It must posses one single argument that is the individual to be evaluated. The function must return a tuple (value,) – with the comma after value.
- *seed*: list of valid models (created by *element.createModel* function). If 'None' (default), no seed is included into the population.

The *run* function return a *hall of fame* object.

A.3.1 Simple Example

Consider the system:

$$y(k) = 0.75y(k-2) + 0.25u(k-1) - 0.20y(k-2)u(k-1)$$

where $u = WGN(0, 1)$ with an output Gaussian noise of zero mean and standard deviation $std = 0.08$.

```

from mggp import mggpElement, mggpEvolver
import numpy as np

# Simulate the System
element = mggpElement()
element.setPset(maxDelay=1, numberOfVariables=2)
element.renameArguments({'ARG0': 'y1', 'ARG1': 'u1'})
listStrings = ['q1(y1)', 'u1', 'mul(q1(y1), u1)']
model = element.createModel(listStrings)
element.compile_model(model)
u = np.random.normal(loc=0, scale=1, size=(500))
y0 = np.zeros((2))
theta = np.array([0.75, 0.25, -0.20])
y = element.predict_freeRun(model, theta, y0, u[:-1])
y += np.random.normal(loc=0, scale=0.08, size=(500, 1))

# Create the element object to be used in MGGP
# in this case, it is the same used to create training data.

mggp = mggpEvolver(popSize=500, CXPB=0.9, MTPB=0.1, n_gen=50, maxHeight=3,
                   maxTerms=5, verbose=True, elite=5, element=element)

def evaluate(ind):

```

```
try:
    element.compile_model(ind)
    theta = element.ls(ind,y,u)
    ind.theta = theta
    SE = element.score_osa(ind, theta, y, u)
    return SE,
# exception treatment for cases of Singular Matrix
except np.linalg.LinAlgError:
    return np.inf,
hof = mggp.run(evaluate=evaluate,seed=None)
model = hof[0]

for term in model:
    print(str(term))
```