**CZECH TECHNICAL UNIVERSITY IN PRAGUE**

**FACULTY OF MECHANICAL ENGINEERING**

ČVUT

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

**DIPLOMA THESIS**

# RELAY FEEDBACK IDENTIFICATION USING GUIDING EVOLUTIONARY ALGORITHM

**2020**

**ADRIAN SALDANHA**

# MASTER‘S THESIS ASSIGNMENT

## I. Personal and study details

| | | | |
|---|---|---|---|
| Student's name: | **Saldanha Adrian** | Personal ID number: | **484357** |
| Faculty / Institute: | **Faculty of Mechanical Engineering** | | |
| Department / Institute: | **Department of Instrumentation and Control Engineering** | | |
| Study program: | **Mechanical Engineering** | | |
| Branch of study: | **Instrumentation and Control Engineering** | | |

## II. Master’s thesis details

Master’s thesis title in English:

**Relay feedback identification using GEA for PID control**

Master’s thesis title in Czech:

**Reléová zpětnovazební identifikace využitím GEA pro PID řízení**

Guidelines:

1. Acquaint with methods of relay feedback identification and with methods of global optimisation using swarm intelligence.
2. Program and compare GEA (Guiding Evolutionary Algorithm) with other swarm intelligence algorithms with respect to suitability for relay identification of the second-order model parameters with time delay.
3. On the simulation models, verify the programmed identification module and PID control setting using the identified model.
4. On the real laboratory apparatuses, verify the programmed identification module and PID control.

Bibliography / sources:

[1] CAO L., XU L. and GOODMAN E. (2016). A Guiding Evolutionary Algorithm with Greedy Strategy for Global Optimisation Problems, 10 pages, Article ID 2565809, Hindawi Publishing Corporation, USA
[2] YANG, Dr. (2014) Nature-Inspired Optimisation Algorithms, First Edition, pp. 1-20, 141-150, Elsevier Insights, London, UK
[3] CHIDAMBARAM M. and SATHE V. (2014). Relay Autotuning for Identification and Control. Cambridge University Press, Cambridge

Name and workplace of master’s thesis supervisor:

**prof. Ing. Milan Hofreiter, CSc., U12110.3**

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **30.04.2020** Deadline for master's thesis submission: **27.08.2020**

Assignment valid until: ______________

| | | |
|---|---|---|
| prof. Ing. Milan Hofreiter, CSc. | | prof. Ing. Michael Valášek, DrSc. |
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student’s signature

# Abstract

The Guiding Evolutionary Algorithm (GEA) was proposed by Cao, Xu and Goodman in 2016. The original algorithm was designed to apply the advantages of the previously published Particle Swarm Optimization (PSO), Genetic Algorithm (GA) and the Bat Algorithm (BA). As part of this thesis, we explore the working of this algorithm on standard functions and then compare the results with those achieved by PSO and BA to verify whether GEA is really superior.

Once we verify the effectiveness of GEA, we shall then proceed to apply the same for System Identification of pre-defined systems in MATLAB for parametric identification of process models of several processes. The identification results are checked for consistency, after which we apply PID Control methods for controlling the same.

Finally, the complete solution which includes optimization, identification and control is tested on a real process namely the two tanks system from the Automatic Control Laboratory and thereby, we can verify whether our proposed method is effective in real process identification. If we can arrive at a positive conclusion, we can take this concept to the next level by applying the same technique to Model Predictive Control wherein, an accurate and efficient algorithm can help to correctly predict future outcomes many steps ahead and can be extremely beneficial in terms of time and cost.

# Statement

# Acknowledgement

This work was supported by the Grant Agency of the Czech Technical University in Prague.
I would like to express my sincere gratitude to my thesis supervisor Ing. Milan Hofreiter, CSc. for his constant guidance and teaching throughout the course of this thesis. This thesis was made possible all thanks to him for which I am truly grateful.
I would like to also thank in a special way, my ex-colleagues and especially my managers from the company ABO Valve s.r.o. for being supportive towards me during these two years of my studies.
Last, but not the least, I would like to thank my family, who have always been my pillar of support.

**CONTENTS**

# List Of Figures

# List of Tables

# 1 Introduction

In this thesis, we shall delve into the details of different optimization algorithms, mainly probability-based algorithms involving swarm intelligence. These algorithms come under the general category of meta-heuristic algorithms wherein, we randomly search within a target sample space looking for a global minimum. The central idea of meta-heuristic algorithms is that, in general these algorithms do not guarantee that we will find the global optimum solution, but rather, that we find a reasonable solution in the given search time (simulation time) with an acceptable error and which satisfies the constraints of the given search function. By appropriate methods as will be described in the later chapters, we may efficiently search a sample space using different meta-heuristic algorithms and compare the results with each other, thereby, finding the most appropriate for our given problem.

The subject of meta-heuristics is quite a wide topic and though, there are countless algorithms in use, we shall look into the sub-branch of algorithms using swarm intelligence and genetic-based approaches, both of which are nature-based. Our guiding material for this research will be the work of Professor Xin-She Yang, titled 'Nature-Inspired Optimization Algorithms (Yang, Nature-Inspired Optimization Algorithms, 2014). In this book, Yang explores a wide range of nature-inspired algorithms which are modelled upon elements in Nature such as Bats (Bat Algorithm) or Eagles (Eagle Strategy). The idea behind using these is to mirror the evolutionary advantages of different species over the course of being and to apply the concepts to solving real-world problems in optimization. As these algorithms have been laid out already, we shall not explain the in-depth working of the same, but rather apply it to our problem at hand.

The main algorithm of focus as part of this research will be the Guiding Evolutionary Algorithm (Cao, Xu, & Goodman, 2016) which is rather a novel approach and has, until now, not yet been applied in practice. The results of the paper *A Guiding Evolutionary Algorithm with Greedy Strategy for Global Optimization Problems* (Cao, Xu, & Goodman, 2016) claims that the results of the novel approach for solving multi-modal problems is quite significant and for functions with higher dimensions, can converge to a global optimum far sooner as compared to some of the former approaches such as Bat algorithm which, in turn has a far better convergence rate when compared to a few other strategies such as Particle Swarm Optimization (Kennedy & Eberhart, 1995) and the Genetic Algorithm (Holland, 1975).

Lastly, in order to assert the superiority of an algorithm over another, we need to apply it to a practical problem, which, in this case shall be “Relay-based Identification of a non-linear system”. By means of the ‘No-Free Lunch Theorem’ (Wolpert & Macready, 1997), we know that there is no universal algorithm for all problems, but rather, for a given problem, we can say that a certain algorithm might be most efficient as compared to another, judged by various criteria. Therefore, as part of this work, we shall apply our algorithms to the specific problem of system identification and judge which of these are effective for our purpose.

In general, optimization problems occur in each and every sector of industry from waste disposal to VLSI design and thus are relevant in several tasks. Every small bit of improvement can have great impacts in terms of cost, energy and time and therefore this topic can be applied universally.

The structure of the thesis is organized as follows: Chapter 1 gives and introduction to the thesis as a holistic approach. Chapters 2 describes the various algorithms that will be covered in this topic of thesis and Chapter 3 gives a summary involving comparison of each of these algorithms in terms of time (convergence rate) and efficiency (finding global optimum) for a number of available functions. In Chapter 4, we shall move to the concept of relay-based feedback identification and PID control, which is the central application for which we shall apply our approach. In this chapter, we shall also explore the identification and control results of the algorithm on pre-defined systems. Chapter 5 is similar to Chapter 4, except that in this chapter, we apply the methods on a real system rather than on some theoretical processes. We shall summarize the approaches and results of all topics in Chapter 6 which will be our conclusion.

# 2 ALGORITHMS

## 2.1. Optimization

To begin with, we shall first define the a few important simple terms which are going to be important for us to understand some concepts in the later chapters. Since we are dealing with an optimization task, we will first define the term '***optimization***', which can be defined as the process of finding the best design in the given design space in terms of certain criteria which we shall refer to as objective functions taking into consideration the list of constraints. The first step in each and every optimization problem is to create or define the model of the system to be optimized, also known as ***quantitative model*** (Parkinson, Balling, & Hedengren, 2018). The model is designed based on physical properties and rules which govern the process or system. This is by far the most important step in the process as the optimization would be rather useless if the physical model itself is incorrect. The next step is to determine the design variables or in terms of optimization these are called ***decision variables*** or ***degrees of freedom*** of the computational model, which are basically the parameters to be optimized. The decision variables can be physical or virtual parameters whose value must be within a certain space also known as ***design space***. The number of decision variables determine the number of dimensions of the design space. Next, once we have our decision variables, we need to determine and state the criteria of the optimization. The criteria consist of the Objectives of the optimization (***Objective Function***) and the ***constraints***. The ***objective function or cost function*** is the physical parameter dependent on the decision variables, which is to be optimized for the given model, while the ***constraints*** are the rules governing the system which are based on physical or user-defined limits.

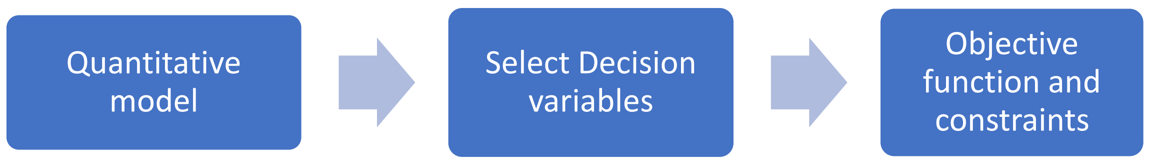


*Figure 1 - Steps for optimization problem formulation*

The above flow-chart represents the steps necessary for the formulation of the optimization problem. Only once the problem has been formulated correctly can we then proceed to apply any sort of algorithm. To express the above steps in mathematical form, we can formulate our function as follows:

1. Quantitative Model / Function: $f_i(x)$ where $x \in \Re$, $(i = 1, 2 \dots M)$
2. Decision variables: $\boldsymbol{x} = (x_1, x_2, \dots x_d)^T$

where $d$ = number of dimensions / decision var.

3. Objective function: $f_i(x)$ $(i = 1, 2 \dots M)$
4. Constraints: $h_j(x)$, $g_k(x)$

where $h_j(x)$, $g_k(x)$ represent physical constraints

The functions $f_i(x)$ where $i = 1, 2, \dots M$ are the ***objective functions***. In case M = 1, this represents a single objective (Yang, Nature-Inspired Optimization Algorithms, 2014). The vector **x** represents the ***decision variables*** and the space spanned by decision variables is the ***search space*** or design space i.e. $\Re^d$. For d = 1, it represents a 1-dimensional problem which can be plotted on a number line. However, complex problems which involve multiple optimization parameters can be much greater and thus the computational power required to solve such problems increases based on the algorithm in use. If the functions $f_i, h_i$ and $g_i$ are all linear, then the problem becomes a ***linear programming problem*** which is a different topic altogether and one which we shall not visit since our functions are all linear.

In the chapters 3, 4 and 5, we shall see the step-by-step formulation of specific optimization problems wherein the above steps will be made clearer.

## 2.2. Optimization Algorithms

An ***algorithm*** can be defined as a procedure to be followed for obtaining a valid result or outcome. For our case, we shall focus on algorithms especially useful for the purpose of optimization. Generally, we use prior experience and feedback from iteratively repeated processes in order to make decisions over time so as to optimize designs to save cost, time and space. However, by using the computational capabilities of modern computers, we can use similar processes to iteratively calculate the outcomes of a much wider range of values (or '***guesses'***) and thereby select the values with the lowest cost functions. The manner of performing guesses or predicting the lowest values is governed by how efficient our algorithm is and the complexity increases exponentially as the number of decision variables increases. As an example, brute force technique is the one common way of finding all available values within a given design space but the computational time and complexity for larger dimensions is a task difficult for even modern computers to solve (except quantum computers). For the purpose of this thesis, we shall look at two prominent types of optimization algorithms namely ***Gradient Based*** and ***Metaheuristic Algorithms***.

### *2.2.1* Deterministic Algorithms

Deterministic algorithms are those which tend to follow a definite procedure for arriving at a solution. It means that the algorithm typically follows the same path each time it is executed. In general, simplex methods used for solving linear programming problems and gradient-based algorithms come under this category of deterministic algorithms. As mentioned before, linear programming problems are not in our field of interest as we are only dealing with non-linear problems and systems and hence we shall only look at gradient-based algorithms as these set the basis of our methodology of applying metaheuristic-based optimization algorithms.

Typically, Gradient-based algorithms can be used to solve ***uni-modal*** or ***multi-modal*** problems in iterative fashion by means of Newton's gradient-based approach. The details of this approach are described in greater detail in the book "Nature-inspired metaheuristic algorithms" (Yang, Nature-Inspired Optimization Algorithms, 2014) and hence we shall not delve into this topic. The important aspect of gradient-based (or descent-based) algorithms is that Newton's Method provides a technique to find the minima or the maxima of a function within a very finite search space by finding the gradient or slope of the curve and then finding the point at which the derivative is zero which provides the maxima or the minima depending on the points near the peak or valley. The speed of convergence can be set based on the step size within the iteration process. The main problem with this approach is that it can be used to find only the local peak or valley within a certain search space which limits the application of this method. The gradient-based algorithm can also be called as ***Hill-climbing algorithm*** (see figure 2). In general, we can see that the Hill-climbing algorithm follows a definite approach to finding the peaks which is why it belongs to a category of deterministic algorithms.

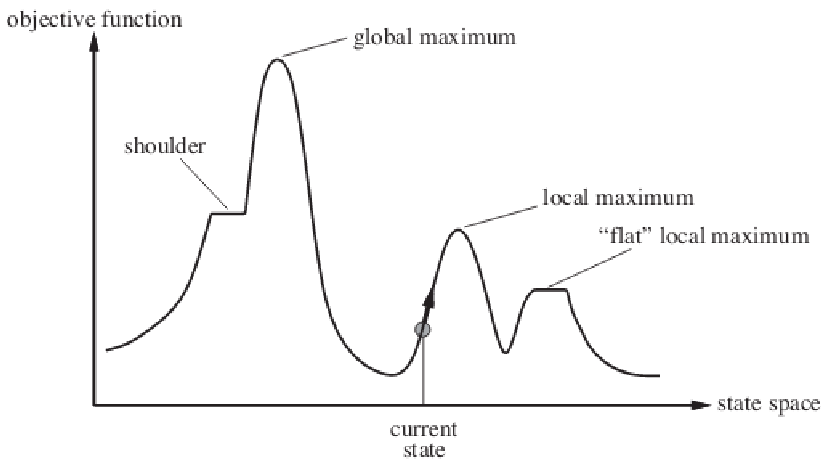


*Figure 2 - Hill-Climbing Algorithm [https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence]*

In practice however, the gradient based approached can be improved by instilling some degree of randomness to the above procedure. A good example is the ***Hill-Climbing Algorithm with Random restart*** wherein we randomize the positions around which we are to find the peak and then proceed to find the absolute values of the peaks within that region. The maximum value of all the peaks would then suggest a global maximum which is far better than the hill-climbing algorithm by itself. In spite of the positives, this method can get way too complicated for multi-modal problems since the number of random restarts would then have to be greater than the total numbers of peaks within the search space which limits the application of this algorithm. The Hill-Climbing with Random restart comes under a third category of hybrid algorithms which is a mixture of deterministic and stochastic algorithms.

In general, we tend not to use deterministic algorithms for such cases of optimization problems as it tends to follow a path similar to brute force approach which is computationally demanding for even modern computers (except quantum computers). The convergence time with deterministic algorithms increases drastically as the number of dimensions and the design spaces gets larger. Instead, we try to use randomization techniques which are part of stochastic algorithms for this purpose since our goal is not always to find the absolute minimum (which can take infinite time to determine), but rather, our main aim is to find the best outcome which satisfies our given criteria within a reasonable amount of time.

### *2.2.2* Stochastic Algorithms

The basic principle of stochastic algorithms relies on the principle of probability wherein, we use randomization techniques in a well-defined and progressive manner so as to search for optima in various different sections of the design space. The result that we obtain, may not be the global optima as mentioned before. That is, while we look for a viable solution, we cannot guarantee that we shall arrive at the precise solution, but rather that we will look for the best solution which meets our pre-stated criteria. This is called ***Heuristics***, that is we use stochastics as a method of arriving at a solution but there is no guarantee that we will find a solution. A sub-branch of ***Heuristics*** can be termed as ***Meta-heuristics*** which will be the topic which we will be looking at as part of this thesis. The difference is basically that ***Heuristics*** are mostly problem-dependent whereas ***Metaheuristics*** are inherently problem independent (meta = beyond ordinary!).

Historically, the heuristic approach to problem solving has existed from the 1940s when the Mathematician Alan Turing used the term '***heuristic search***' for the method he used to decode the Enigma machines. Later research into heuristics led to the development of a wide range of algorithms namely ***Genetic Algorithm*** (Holland, 1975), ***Simulated Annealing*** (Kirkpatrick et all, 1983), ***Tabu Search*** (Glover, 1986), ***Ant Colony Optimization*** (Dorigo, 1992), ***Particle Swarm Optimization*** (Kennedy & Eberhart, 1995), ***Bat Algorithm*** (Yang, A New Metaheuristic Bat-Inspired Algorithm, 2010). A common denominator of most of the afore-mentioned optimization algorithms, excluding Tabu Search is the fact that these come under the category of ***Nature-Inspired Metaheuristics*** and have first been modelled around nature. We shall be exploring some of these methods as part of this thesis.

To summarize the general topic of nature-inspired metaheuristics, we shall first describe the three main operators for most of these algorithms:

1. ***Crossover Operator:*** The role of this operator is to provide a good mixing within the solution space.
2. ***Mutation Operator:*** This method helps us to search within the design space looking for solutions different from what we have already obtained previously. This helps avoid us getting stuck in a local optimum. The mutation operator provides the necessary exploratory framework for ***diversifying*** our solution.
3. ***Selection:*** Selection is the mechanism by which we choose the best individual from a list of individuals so that this individual can progress to the next generation from which we can develop even more suitable solutions. The selection mechanism provides the essential exploitation necessary for ***intensifying*** our search.

Almost all of the algorithms in our discussion will include the above operators using different techniques. The efficacy of each method depends not only on the manner in which it is conducted, but also on the problem at hand.

Furthermore, to assist with understanding of the next chapters, it is crucial to first define a few important terms which will be relevant to our algorithms:

Our approach can differ depending on the algorithm we choose, and we will see later that some algorithms are better than some others depending on the type of problem. As concluded in the "No Free-Lunch Theorem", we see that there is no universal algorithm which works best for each and every problem or, in other words, or, as put forth by Wolpert and Macready, "For both static and time-dependent optimization problems, the average performance of any pair of algorithms across all possible problems is identical" (Wolpert & Macready, 1997). Hence, we will inspect a few available algorithms and evaluate the performance for our specific functions.

a. ***Dimensions (d):*** The dimensions it the number of variables which influence the objective functions. This is same as the number of parameters to be optimized.

b. ***Random variables ($x_i$):*** This is common to all stochastic processes. It is a variable whole value is randomly determined by different set of random distribution rules such as *Gaussian distributions, Uniform Distributions, Levy distributions.* For our purpose, we shall mainly follow gaussian distributed and uniformly distributed random variables.

c. ***Random Walk:*** A random walk is step taken randomly from a fixed point. The final value depends on a number of randomly taken steps from the initial position.

Mathematically, a random walk can be described as $S_N$ where $S_N$ is a series of consecutive random steps $X_i$ (Yang, 2010).

$$S_N = \sum_{i=1}^{N} X_i = X_1 + X_2 + \cdots + X_N \tag{1}$$

$$S_N = \sum_{i=1}^{N-1} X_i + X_N = S_{N-1} + X_N \tag{2}$$

Where $X_i$ = random step drawn from a random distribution.

d. ***Solutions (x):*** For each and every algorithm, we use solutions which is basically a vector containing a number of search points in the design space.

Mathematically, we can represent this as:

$$\boldsymbol{x_N} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{x_i} = (x_1, x_2, \dots x_N)^T \tag{3}$$

Where

$x_1 = (x_{1_1}, x_{1_2} \dots x_{1_d}),$

$x_2 = (x_{2_1}, x_{2_2} \dots x_{2_d}),$

$x_i = (x_{i_1}, x_{i_2} \dots x_{i_d})$

We shall be using all of the above methods in our understanding of the algorithms which follow. With these definitions and descriptions, we can finally take a deep dive into the algorithms which we shall be using.

## 2.3. Particle Swarm Optimization

Particle Swarm Optimization is a meta-heuristic based optimization procedure which was developed by modelling swarm behaviour observed in nature such as bird flocking, fish schooling and swarming theory (Kennedy & Eberhart, 1995). The advantage of this procedure is its relative simplicity in implementing and it’s also being computationally less demanding in terms of memory and speed. Additionally, unlike other genetic algorithms, there is no encoding / decoding necessary for Particle Swarm Optimization. Although the PSO is not the major algorithm for this thesis work, it is an important pre-requisite for obtaining a deeper conceptual understand of the next algorithms.

In general, the optimization technique is based on the simple concept that each individual in subsequent iterations (generations) can benefit from the observations by the other individuals from the previous generations, consequently, improving the subsequent results. The individuals are considered as particles, while the entire group of particles are considered as a swarm, thus the name *Particle Swarm*. The trajectories of the individual particles are adjusted in a quasi-stochastic manner which includes:

a) Stochastic component and a

b) Deterministic component

The underlying concept behind the algorithm is deep and ingenious and can be found in the cited paper. For the purpose of this thesis however, we shall describe how the algorithm works from the point of view of the mechanics rather than the principle behind it.

Let us consider a number of particles (individuals) **N** in a sample space. The position and velocity of each particle in the sample space is influenced by the particle's previous position as well as the best position of all particles until that point. The vectors **x** and **v** describe the position and velocities of the particles at any given time (generation). As mentioned earlier, each individual particle is attracted toward the position of the current global best $\mathbf{g^*}$ and its own best location $\mathbf{x_i^t}$ in history. The vectors $\mathbf{x}_i$ and $\mathbf{v}_i$ are updated as follows:

$$\boldsymbol{v}_i^{t+1} = \theta \boldsymbol{v}_i^t + \alpha \boldsymbol{\epsilon_1}[\boldsymbol{g}^* - \boldsymbol{x}_i^t] + \beta \boldsymbol{\epsilon}_2[x_i^* - \boldsymbol{x_i^t}] \qquad (4)$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \qquad (5)$$

Where:

$\boldsymbol{v}_i^t$ and $\boldsymbol{v}_i^{t+1}$: Velocities of the 'i'th particle at generations t and t+1 respectively

$\boldsymbol{x}_i^t$ and $\boldsymbol{x}_i^{t+1}$: Positions at generations (time) t and t+1.

$\boldsymbol{\epsilon_1}$ and $\boldsymbol{\epsilon}_2$: Random vectors with values $\in[0, 1]$

$\alpha, \beta$: Learning parameters of value $\approx 2$.

$\boldsymbol{g}^*$: Global best position of all particles

$\boldsymbol{x}_i^*$: Individual Best position of particle **i**.

$\theta$: Inertia Value of the particle (optional term but necessary for better convergence

From equation 4, we see that the updated velocity is a linear sum of the deviations of the current position from the current global best $\boldsymbol{g}^*$ as well as the deviation of the position from the individual best $\boldsymbol{x}_i^*$. The global best position directly the other particles towards the best one, while the historical best position of each particle accounts for an element of mutation wherein, while the particle is drawn to the global best, it is also influenced by its own historical best which provides an element of mutation. The weights of each can be easily controlled by tweaking the values of $\alpha$ and $\beta$. With this, with every consecutive iteration, the individual particles move nearer towards the global best. Additionally, the global best $\boldsymbol{g}^*$ is not fixed and it must be updated with every generation using the cost function as follows:

$$\boldsymbol{g}^* = \min\{f(\boldsymbol{x_i})\}\ for\ (i = 1, 2 \dots n) \tag{6}$$

This is done at the end of every iteration and understandably, the value of the global best is changed only if the new minimum value of the cost $f(\boldsymbol{x_i})$ is less than the previous global best. Thus, as the program updates, the positions are regularly updated with each generation until, at some point, the error from the minima / maxima of the cost function arrives at an acceptable limit.

The pseudo-code for the algorithm is shown in the below figure [Yang, 2010].

```
Particle Swarm Optimization

1. Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \dots, x_d)^T$
2. Initialize locations $x_i$ and velocity $v_i$ of n particles
3. Find $\boldsymbol{g}^*$ from $\min\{f(\boldsymbol{x}_1), f(x_2) \dots f(x_n)\}$ (at t = 0)
4. While (criterion)
   For (loop over n-particles and d-dimensions
   Generate new velocity $\boldsymbol{v}_i^{t+1}$
   Calculate new locations $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^{\boldsymbol{t}} + \boldsymbol{v}_i^{t+1}$
   Evaluate objective function at locations $x_i^{t+1}$
   Find the current best for each particle $\boldsymbol{x}_i^*$
   End for
   Find the current global best $\boldsymbol{g}^*$
   Update $t = t + 1$ (iteration counter)
5. End while
6. Output final results $\boldsymbol{x}_i^*$ and $\boldsymbol{g}^*$.
```

*Figure 3 - Pseudo-code - Particle Swarm Optimization (Kennedy & Eberhart, 1995)*

## 2.4. Bat Algorithm

The Bat Algorithm (BA) was developed by Xin-She Yang in 2010 and it is based on the feature of echolocation of Bats, which bats use to find their prey. While the inherent principle of BA is quite similar to PSO, the core of this search method is that it is based on frequency tuning unlike PSO. By experience, this method is found to be extremely efficient in practice and therefore, has been put to use in a wide variety of applications since (Yang, Nature-Inspired Optimization Algorithms, 2014).

Biologically, microbats use the principle of echolocation to detect their prey and avoid obstacles in the dark. The method by which this takes place is similar to the principle of sonar, wherein the bats emit a very loud sound pulse (bursts) and listen for the echo that bounces back from the objects (Yang, 2010). Once the wave is reflected back from the object to the bat, the bat is able to detect the echo, the time difference between the two ears and the loudness variations of the echoes to build up a three-dimensional scenario of the surrounding, by which, they can detect the distance, orientation and the moving speed of the target object. Typically, the microbats emit pulses of about 8-10 ms at a constant frequency in the range of 25 kHz to 150 kHz with around 10 to 20 bursts every second. The loudness of the emitted pulse varies as it is usually the loudest when searching for its prey and turns quieter when heading toward it.

With these basic biological principles in mind, we can proceed to formulate a mathematical model of the same operation. Let us state the three idealized rules in the echolocation of bats (Yang, A New Metaheuristic Bat-Inspired Algorithm, 2010):

1. All Bats use echolocation to sense distance to its prey (food/prey and physical barrier can be thought of to mean the same.
2. Bats fly randomly with velocity $v_i$ and position $x_i$. They automatically adjust the frequency or wavelength of their emitted pulses and adjust the rate of pulse emission $r \in [0,1]$ depending on the proximity of the target.
3. We assume that the loudness can vary from a large (positive) $A_0$ to a minimum value $A_{min}$.

Additionally, we shall also consider the following assumptions that the frequency f is in the range $[f_{min}, f_{max}]$ corresponding to the wavelength $[\lambda_{min}, \lambda_{max}]$. Since $f.\lambda = constant$, we can vary the value of only f, instead of changing $\lambda$ as well and for simplicity, we use $f \in [0, f_{max}]$. The rate of pulse emission can range from [0, 1].

Using the above rules, we can construct our pseudo code as shown in the below figure.

```
Bat Algorithm
    1. Objective function $J(\boldsymbol{x}), \boldsymbol{x} = (x_1, \dots, x_d)^T$
    2. Initialize locations $x_i$ and velocity $v_i$ of n particles, $(i = 1,2 \dots n)$
    3. Initialize frequencies $f_i$, pulse rates $r_i$ and loudness $A_i$
    4. Find $\boldsymbol{g}^*$ from min{$J(\boldsymbol{x}_1), J(\boldsymbol{x}_2) \dots J(\boldsymbol{x}_n)$} (at t = 0)
    5. While (t<max number of iterations)
Generate new solutions by adjusting frequency,
Update velocities and locations as per Bat Algorithm
If (rand > $r_i$)
        Select a solution among the best
        Generate a local solution around the best
End if
Generate new solution by flying randomly
If (rand < $A_i$ & f($\mathbf{x}_i$) < f($\mathbf{x}^*$))
        Accept new solutions
        Increase $r_i$ and reduce $A_i$
End if
Rank the bats and find current best $\mathbf{x}^*$
    6. End while
```

*Figure 4 - Pseudo Code - Bat Algorithm (BA) (Yang, A New Metaheuristic Bat-Inspired Algorithm, 2010)*

Similar to the PSO, let us construct a set of Bats (*solutions*), similar to particles in PSO.

We shall consider the solutions as $\mathbf{x}_i^t$ and velocities as $\boldsymbol{v}_i^t$ in a d-dimensional space. The new positions (solutions), velocities and frequencies at time t can be given by:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{7}$$

$$v_i^{t+1} = v_i^t + f_i.(x_i^t - x^*) \tag{8}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{9}$$

The above part of the algorithm serves to randomly search for new solutions in the sample space and to move to the new solutions. This working somewhat like the crossover and mutation operator using *random walks* wherein, new solutions are found using the previous solution as well as the frequency update provides a fair deal of randomness. Similarly, the search for the new global best x* is like the selection operator and it updated at the end of each iteration by finding the minimum cost function for all bats as follows:

$$x^* = \min\big(J(x_1), J(x_2), \dots J(x_n)\big) \tag{10}$$

In addition to the above, we need to provide a way for the solution to slowly converge to the best solution and hence we add a local search operator which directs all the bats towards the ultimate prey. The local search provides the necessary mechanism for *exploitation* whereas, the global search from the first part provides th mechanism for *exploration*. The local search can be achieved as follows:

$$x_{new} = x_{old} + \sigma \epsilon_t A^t \tag{11}$$

Note that the above search does not need to take place at every iteration but must function increasingly as the program advances. This can be regulating by increasing the ***pulse emission rates*** as the number of generations increase. Additionally, as the bat moves closer to the prey, the ***loudness*** decreases, and this can be represented using simple probability related functions as follow:

$$A_i^{t+1} = \alpha A_i^t \tag{12}$$

$$r_i^{t+1} = r_i^0 [1 - e^{-\gamma t}] \tag{13}$$

Where:

| | |
|---|---|
| $f_i$: | frequency of the i'th bat (solution) |
| $f_{min}, f_{max}$ : | min, max frequency |
| $x^*$ : | global best position |
| $x_i$: | $[x_1, x_2, \dots x_d]$, position vector (d-dimensions) of i'th bat (solution) |
| $v_i$ : | $[v_1, v_2, \dots v_d]$, velocity vector (d-dimensions) of i'th bat (solution) |
| $A_i$: | Loudness of pulse |
| $r_i$: | Pulse emission rate |
| $\beta$: | Uniformly distributed random vector |
| $\epsilon_t$: | Normally distributed random vector [0,1] |
| $\sigma$: | Scaling factor (depends on search area) |

| | |
|---|---|
| $\alpha$: | Cooling factor [0 <$\alpha$<1] |
| $\gamma$: | Exponential Factor [>0] |

Typically, the above parameters are tuned based on experimentation and also on the sample space. In the initial step, each bat will have different values of loudness and pulse emission rate. The initial loudness of each bat is taken closer to 1, while the pulse emission rate is closer to 0. With every generation, the pulse emission rate increases reaching 1 towards the end, while the loudness conversely decreases as the number of generations increase. To improve the performance and to increase the diversity of search, we choose the parameters $\alpha$ and $\gamma$ in such a way that that exploitation stage is not too quick or else the solution may lead to stagnation while the exploration is not too slow either or else, the time for attaining the optimum solution will thereby increase. Thus, we can see the advantages of the algorithm in that, the dynamics can be adjusted based on our objective (convergence rate or optimum solution) as well as the type of optimization problem which makes the program extremely *versatile*. Furthermore, due to the nature of the pulse emission rate $r_i$, it automatically facilitates *automatic zooming* into the region where promising results are found thereby switching from exploration to exploitation which, thereby enables *quick convergence* rate in comparison to other available algorithms.

## 2.5. Guiding Evolutionary Algorithm (GEA)

The Guiding Evolutionary Algorithm with Greedy Strategy was proposed by Cao et all in the year 2016 for solving problems on global optimization (Cao, Xu, & Goodman, 2016). The underlying principle behind this approach was primarily inspired by previously developed similar swarm-based intelligence methods PSO, BA and Genetic Algorithm so as to obtain the advantages of each, while also improving the convergence to global optima instead of getting stuck at local optima. Since this method is a relatively novel approach compared to the other methods, we shall first describe this method, while explaining the dynamics and later try to evaluate the functioning for different functions – unimodal as well as multi-modal problems[1]. This algorithm shall be the main work of this thesis and therefore we will look at this one in much greater depth.

[1] Uni-modal = function having one global optimum only, Multi-modal = function having one or more global optimum values and a few local optimum values

Firstly, to understand the mechanics of GEA, it is necessary to understand the working of BA as described in section 2.4. The author of the GEA claims that while the BA is highly efficient and can converge quickly, it often tends to converge to the local minima instead of trying to arrive at the global minima in a multi-modal problem. While being inherently similar to PSO, it rejects historical experience of its own position, but tries to accept a better individual solution with some probability. The GEA algorithm was proposed to improvise this method and therefore aim to achieve better results at arriving at the global best.

Similar to most other optimization algorithms, the GEA consists of three basic operators, namely – Crossover, Mutation and Local search. Let us describe the function of these three operators as originally described in the paper (Cao, Xu, & Goodman, 2016).

1. Crossover:

As described in section 2.2 (b), the crossover operator is used to ensure good mixing within the solutions. In case of the GEA, the crossover is achieved in the following manner, which is similar to PSO:

$$x_i^t = x_i^{t-1} + (x_*^{t-1} - x_i^{t-1}) * \beta \tag{14}$$

Where:

| | |
|---|---|
| $x_i^t$: | new position of solution $x_i$ |
| $x_i^{t-1}$: | position of solution $x_i$ at previous iteration |
| $x_*^{t-1}$: | current global best individual |
| $\beta$: | step length of position increment, uniformly distributed random variable [0, 2] |

We see in the above expression that the new position of the individual is driven by the best individual with a step length of $\beta$, or in other words, the current best individual combines with the current individual to generate a new offspring. According to the paper, the effectiveness of this algorithm is in the fact that with each iteration, the individuals move closer to the best individual, while together moving toward the global best individual. Later, in section 3, we shall verify to what extent this statement is true.

2. Mutation:

Once again, as described in section 2.2 (b), mutation is applied in order to increase the diversity of the solution by searching through the unexplored solution space of the problem (*exploration*). This helps in avoiding being trapped in local optima. By means of the GEA, we shall achieve mutation using the formula:

$$x_i^t = x_i^t + \epsilon M \tag{15}$$

Where:

$\epsilon$: Uniform Random Vector [-1, 1]

$M$: Mutation Vector,

$M_j = \max(x_{ij}^t - a, b - x_{ij}^t)$ and [a,b] = range of $j^{th}$ dimension

In order to make the search more efficient, the mutation operation is carried out with a probability $p$ such that the probability of mutation increases as the number of generations progress. That is, the probability is generally low in the beginning stages and exponentially increases with the number of iterations.

According to GEA, the probability function p is defined as:

$$p = c * \ln\left(\frac{T_{max}}{T_{max} - t}\right) \tag{16}$$

Where:

| | |
|---|---|
| $p$: | probability of mutation |
| $T_{max}$ : | maximum number of generations |
| $t$: | current generation |
| $c$: | 0.2 (constant) |

Thus, the equation (15) is executed only if the probability of mutation exceeds the value given in equation (16). From equation (15), we see that the mutation operation is a simple, linear function of the current position $x_i^t$ and the Mutation vector M. Later, in section 3, we shall evaluate this function and we shall see why this mechanism does not really function precisely as it is supposed to and hence, we shall modify this same function in a way so as to obtain maximum diversification while still remaining within the acceptable range.

3. Local Search:

The local search function is used to achieve maximum exploitation around the best solution of the search problem. With this facility, we shall proceed to locate the global best quickly once the best neighbourhood has been found. Once again, it is not advantageous to do a local search during the beginning stages of the algorithm or else, the solution might stay at the local optima. Hence we try to achieve the local search in a similar way as we achieved the mutation by following the probability function $p$. The formula for local search using GEA can be expressed as follows:

$$x_i^t = x_*^{t-1} + \epsilon L \tag{17}$$

Where:

$\epsilon$: Uniform random number

$L$: Local search vector, $L_j = 0.1 * (b - a)$, where j denotes the dimension.

Once again, we see from equation (17) that the exploitation takes place around the best position, which enables quicker convergence, but could lead to the solution getting stuck at a local minimum in case of multi-modal problems. For example, if, instead, we use the vector $x_i^{t-1}$ instead of $x_*^{t-1}$, then the search would effectively include the surroundings of the previous position but, as a downside, the convergence time would thereby increase consequently.

```
GEA Algorithm
1. Objective function $J(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \dots, x_d)^T$
2. Initialize locations $x_i$, define parameters c, M, L
3. Evaluate the initialized positions
4. Select the best individual $x_*^{t-1}$
5. While (t<max number of iterations)
   For each individual:
Make crossover to generate a new individual $x_i^t$
If (rand <p)
Make mutation for $x_i^t$
End if
Generate new solution by flying randomly
If (rand < $A_i$ & f($\mathbf{x}_i$) < f($\mathbf{x}^*$))
Accept new solutions
Increase $r_i$ and reduce $A_i$
End if
Rank the bats and find current best $\mathbf{x}^*$
6. End while
```

*Figure 5 - Pseudocode - GEA Algorithm (Cao, Xu, & Goodman, 2016)*

The advantages of GEA can be seen in that, it is relatively simple to execute, and it has fairly good results in evaluating multiple functions, superior in some cases to BA as well as PSO. Additionally, since the mutation as well as the local search are both based on probability, the efficiency of the algorithm is greatly increased in terms of optimization time. We shall examine the advantages / disadvantages once again in section 3 using real functions.

# 3 Simulation Results for Pre-defined functions

We have defined the working of each of the algorithms in use. To test the effectiveness of the above described algorithms, we shall use a number of pre-determined functions and evaluate the efficiency of each algorithm in terms of *convergence rate* as well as *optimum result*. The test functions shall be unimodal as well as multi-modal functions. With uni-modal functions, our goal is to compare the convergence rate of each algorithm as, most often, all algorithms are able to accurately predict the optimum value correctly whereas, in case of multi-modal problems, we shall test for the convergence rate as well as the optimum value attained at the end of the iterations.

## 3.1. Test Functions:

The test-functions are as per the Table 1. Each function defines a 3-dimensional surface and consists of a single or a number of valleys (local minima):

| Functions | Function Name | Expression | Domain |
|---|---|---|---|
| F1 | De-Jong's Sphere function | $f(x) = \sum_{i=1}^{D} x_i^2$ | [-100, 100] |
| F2 | Schwefel 2.2 function | $f(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$ | [-15, 15] |
| F3 | Griewangk's function | $f(x) = -\prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + \sum_{i=1}^{D} \frac{x_i^2}{4000} + 1$ | [-15, 15] |
| F4 | Rosenbrock's function | $f(x) = \sum_{i=1}^{D-1} 100 * (x_{i+1} - x_i^2)^2 + (x_i - 1)^2$ | [-15, 15] |
| F5 | Rastrigin's function | $f(x) = D * 10 + \sum_{i=1}^{D}(x_i^2 - 10 * \cos(2\pi x_i))$ | [-5, 5] |

| | | | |
|---|---|---|---|
| F6 | Michalewicz function | $f(x) = -\{\sin(x)\left[\sin\left(\frac{x^2}{\pi}\right)\right]^{2m} + \sin(y)\left[\sin\left(\frac{2y^2}{\pi}\right)\right]^{2m}\}$ | [0, 4] |

*Table 1 - Predefined functions for simulation*

For all of the above functions, the goal is to evaluate the point or points at which the value of the function is minimum. In case of unimodal functions, there exists only a single minimum, whereas for multi-modal functions, there can be a single or multiple global minimum and several local minima. Since we would like to visualize these functions shall evaluate all of the above algorithms for D = 3 only. In case needed, the number of dimensions can be increased as necessary. The above functions can be displayed graphically as below:

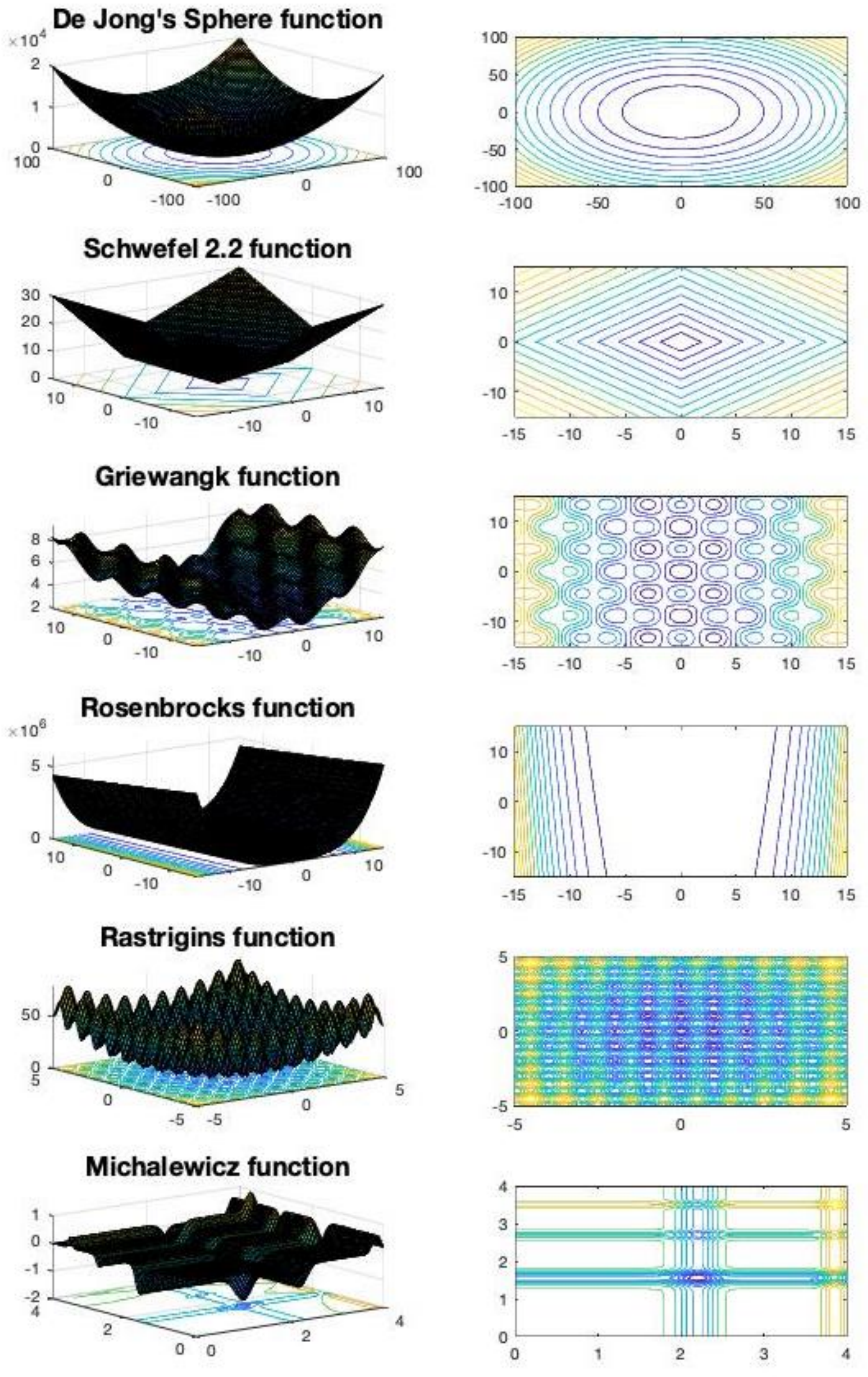


*Figure 6 - Functions F1-F6*

By looking at the above functions we can clearly see the nature of each function in terms of unimodal vs multi-modal. For example, the De-Jong’s function, Schwefel 2.22 function and the Rosenbrock’s function are clearly unimodal, whereas the Griewangk’s function, Rastrigin’s function and Ackley's function are multi-modal functions with a single global minimum and many local minima.

In evaluating the above functions, we will make use of different number of solutions (***N***) for example 20, 40, 50 and 100 solutions and then we shall compare the convergence time as well as the final value attained. The results of the evaluation can be summarized in the below charts.

## 3.2. Parameterization

In order to obtain the best results for each algorithm, we need to set the parameters using some empirical methods or by experimentation. The following parameters were used for each of the three algorithms:

1. PSO: The parameters of PSO are set based on Cao's paper (Cao, Xu, & Goodman, 2016).

| Parameter | Value |
|---|---|
| $\beta$ | 1.5 |
| $\gamma$ | 0.9 |
| $\theta$ | 0.7 |
| $\alpha_0$ | 1.5 |
| $\alpha$ | $\alpha_0 * \gamma^t$ |

2. BAT: The parameters of the BAT algorithm were taken from a previous research paper (Yang, A New Metaheuristic Bat-Inspired Algorithm, 2010) and, after repeated testing on the above functions, were found to give good results.

| Parameter | Value |
|---|---|
| $Q_{min}$ | 0 |
| $Q_{max}$ | 2 |
| $\alpha$ | 0.97 |
| $\gamma$ | 0.9 |
| $\theta$ | 0.7 |

| | |
|---|---|
| $\sigma$ | $0.1 * (ub - lb)$ |
| $A$ | 0.95 |
| $\rho$ | 0.6 |

3. GEA: The parameters for the GEA algorithm are taken directly as per the GEA algorithm itself. (Cao, Xu, & Goodman, 2016).

| Parameter | Value |
|---|---|
| $c$ | 0.97 |
| $\beta$ | [0, 2] |

Besides the above, as a thumb rule, we shall set the number of individuals $\boldsymbol{n}$ as 10 x (no of dimensions).

## 3.3. Results

The results of the above functions are obtained by simulating these functions using MATLAB / Simulink. We can summarize these as follows:

### *3.3.1* F1 - De-Jong's Sphere function:

Figure 7 shows the results of evaluation of the De Jong's Sphere Function, from which, we can see that the three algorithms are equally able to arrive at the minimum value is around 20 iterations, each with 30 individuals only, whereas, the BAT Algorithm also arrives close to the absolute minimum in about the same time. To improve the results, it would be okay to increase the number of individuals (bats or particles) thereby, reducing the convergence time.

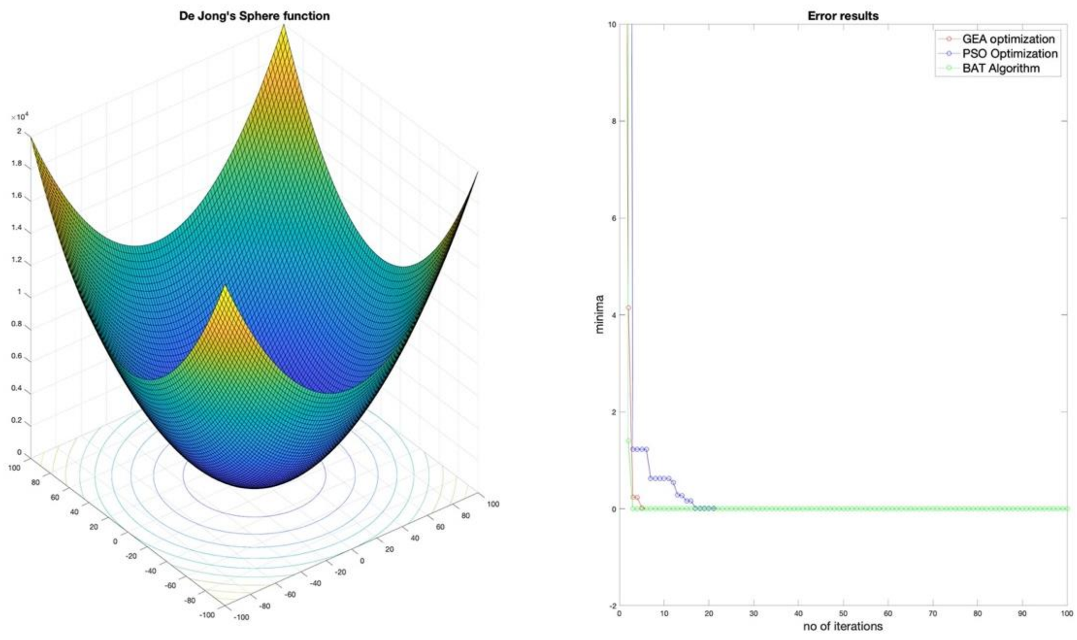


*Figure 7 - Error Results - De Jong's Function*

### *3.3.2* F2 - Schwefel 2.2 function:

Once again, similar to the curve for F1, we see that all algorithms have arrived at a solution in similar amount of time and the same when replicated again yields similar results.

The reason for the quick convergence is the fact that both these functions F1 and F2, are unimodal and hence the solutions are all influenced by the correct minimum values. For the next few functions, however, we would be able to see a considerably difference in the time it takes to arrive at a minimum value.

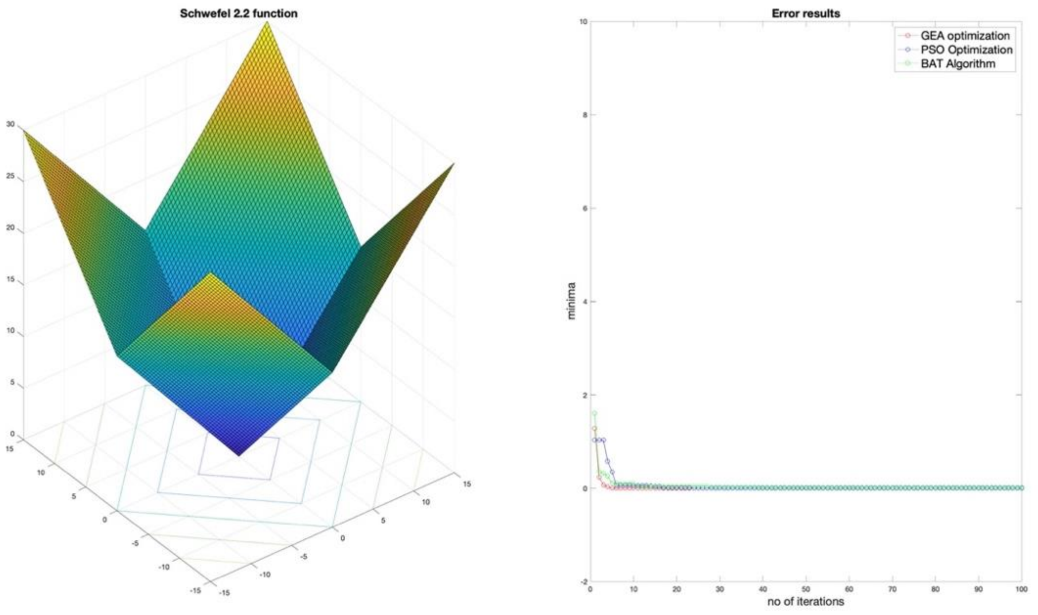


*Figure 8 - Error Curve for Schwefel 2.2 function*

### *3.3.3* F3 - Griewangk's function:

Griewangk’s function is by far the most complicated of all the functions from the figure 6. This is because it firstly consists of multiple global minima and additionally, several local minima, all of which differ by a minuscule amount. Therefore, should the heuristic algorithm be unable to locate a precise position around the global minima, there is a high likelihood of the solution getting stuck at one of the local minima.

To illustrate this, let us look at one of the results of simulation:

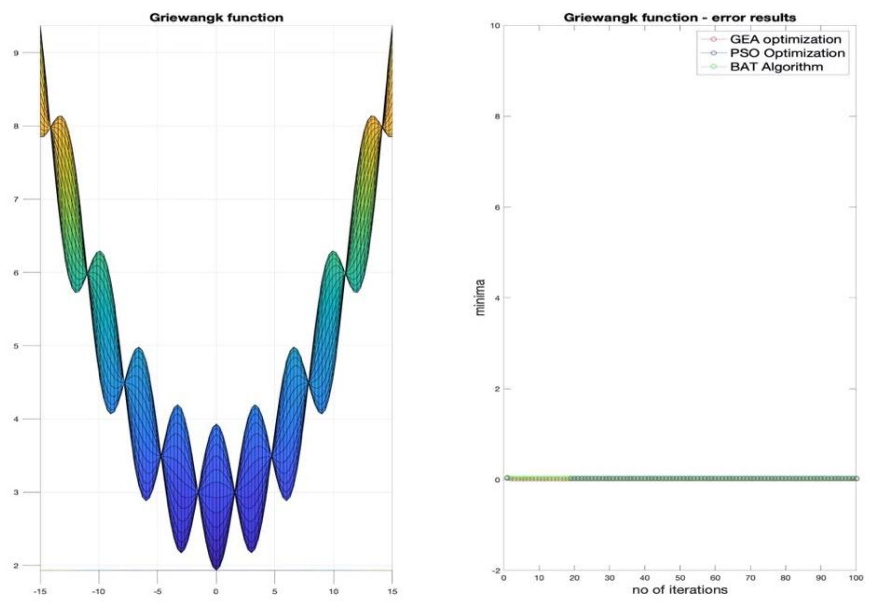


*Figure 9 - Error Curve - Griewangk's Function*

From the above error-curve, it appears as if all algorithms successfully find the minima. However, when we plot the points on the contour plot, we obtain the following:

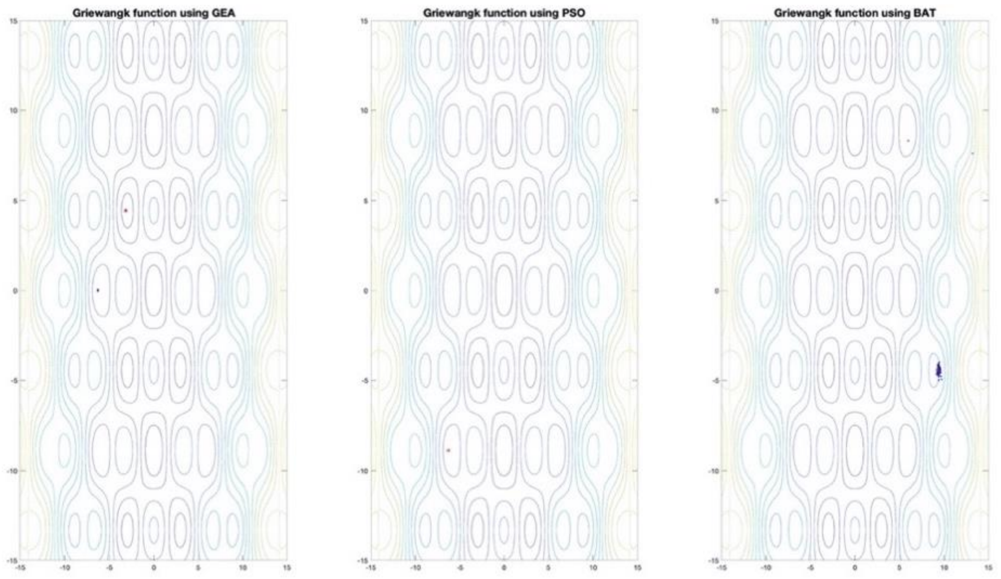


*Figure 10 - Contour Plot - GEA, PSO, BA*

If we look at the actual points and best solutions (denoted by the red Asterix), we find that none of the algorithms are successful in locating the real optima which is located at the point (0,0). This is because, due to the heuristic nature of the algorithms, all points converge toward the best solution within the given search-space, which, in some cases may not be the real optimum value. Therefore, to circumvent this problem and to obtain a better estimate, we need to either increase the number of iterations or increase the numbers of individuals which would help to widen the search space.

To mitigate this, one method is to increase the number of individuals to 50 instead of 30 and wherein, we can see that there’s a higher probability of finding the minimum value.

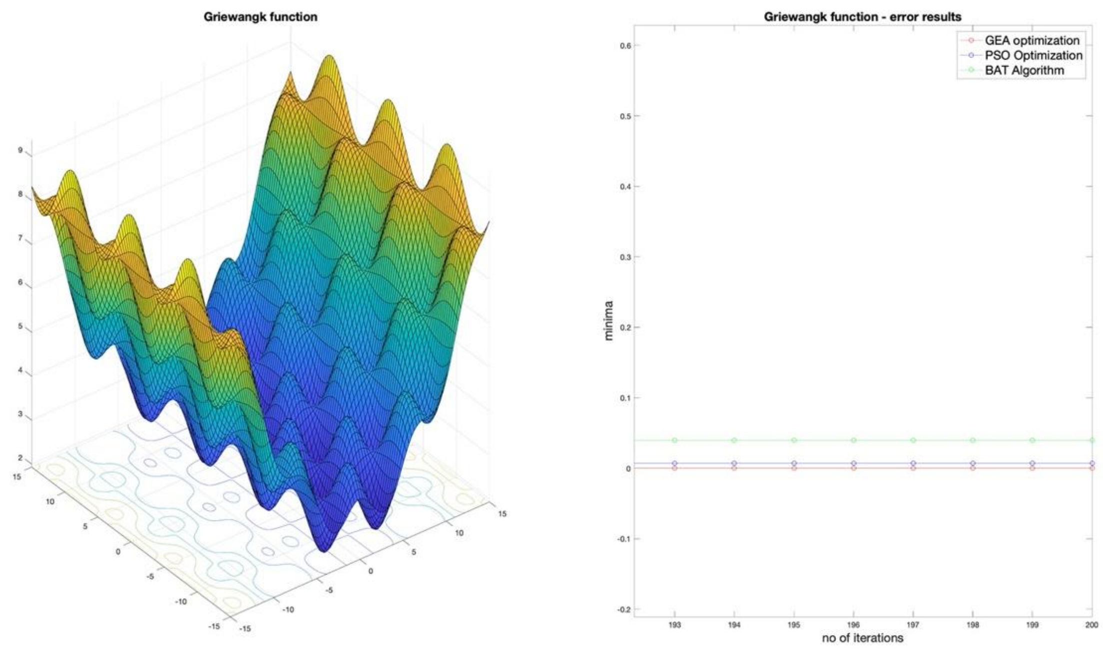


*Figure 11 - Error Curve - Griewangk's Function (2)*

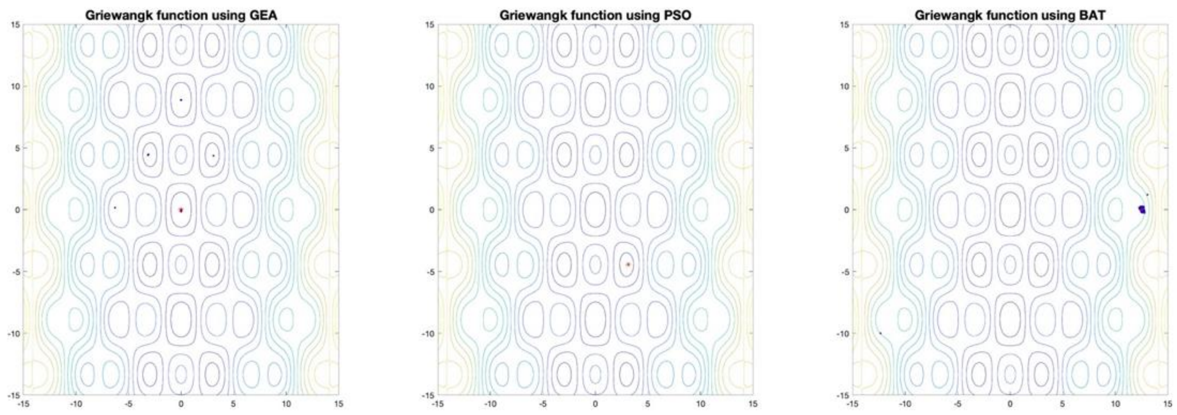


*Figure 12 - Contour Plot - Griewangk's function (2)*

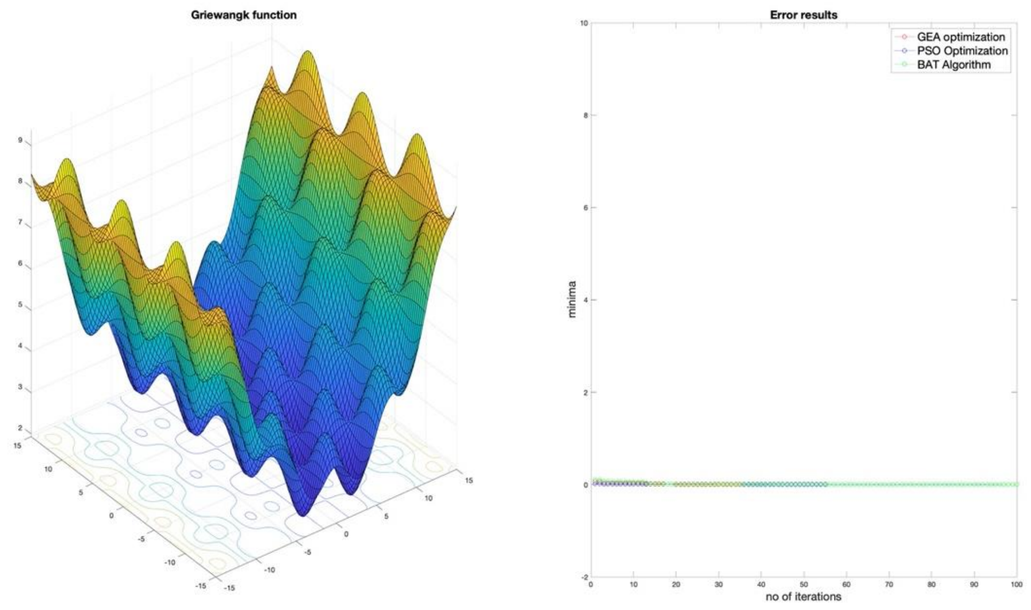


*Figure 13 - Error Curve - Griewangk's Function (3)*

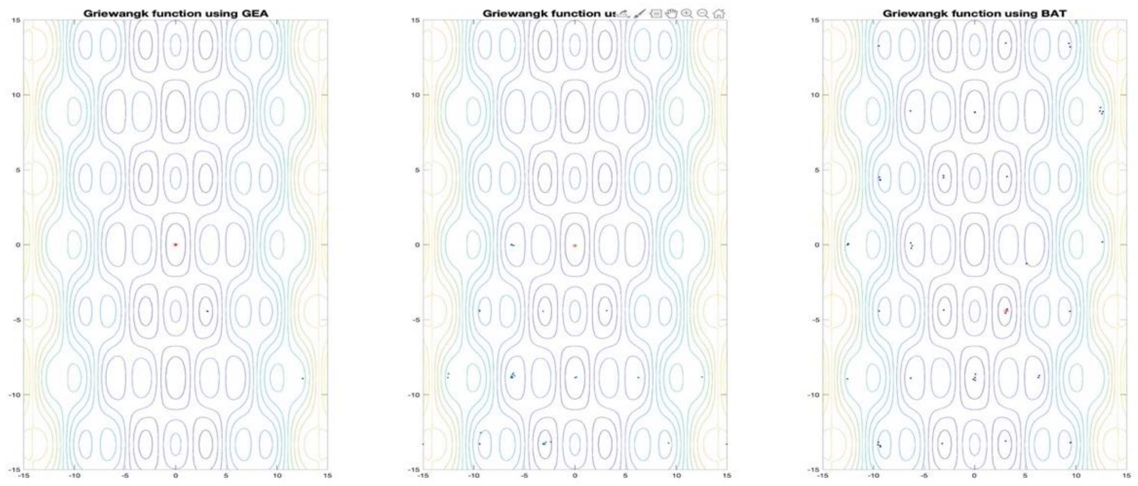


*Figure 14 - Contour Plot - Griewangk's Function (3)*

If we look at the actual points and best solutions (denoted by the red Asterix), we find that none of the algorithms are successful in locating the real optima which is located at the point (0,0). Figures (14) and (15) show the simulation results with N = 200 and n = 50. We can conclude from the above figures that, the algorithms GEA and PSO are able to find the true optima in majority of the tries, BA and sometimes PSO tends to settle at the local optima. This does not necessarily mean that GEA is superior to PSO and BA, but that BA will have to be tuned to a greater precision in order to obtain more accurate results. This is one of the drawbacks of BA since, due to the presence of multiple parameters, the tuning is relatively difficult, which makes GEA a viable alternative. Secondly, after repeated experimentation with the original GEA algorithm, it was found that the mutation function using the original method would not work efficiently for this problem in which there are multiple minima. The reason for this is because the original GEA mutation operator (equation (15)) works only if the condition (rand<p) is satisfied. Practically, this does not work efficiently since the mutation either operates on all dimensions or does not operate at all. To fix this, we introduce the same mutation operation, but applied to each dimension individually. This can be done by modifying equation (15) as follows:

$$x_i^t = x_i^t + \epsilon M_j .* (rand(j) < p) \tag{18}$$

This means that the once some parameters are fixed, the others can mutate easily, thus giving better results especially when some of the dimensions have been able to find a good solution.

For all future purposes, we shall consider the equations (18) as our new replacement to equation (15) from the GEA algorithm.

### *3.3.4* F4 – Rosenbrock’s function:

The results of the algorithm with Rosenbrock’s function (banana function) can be summarized as follows:

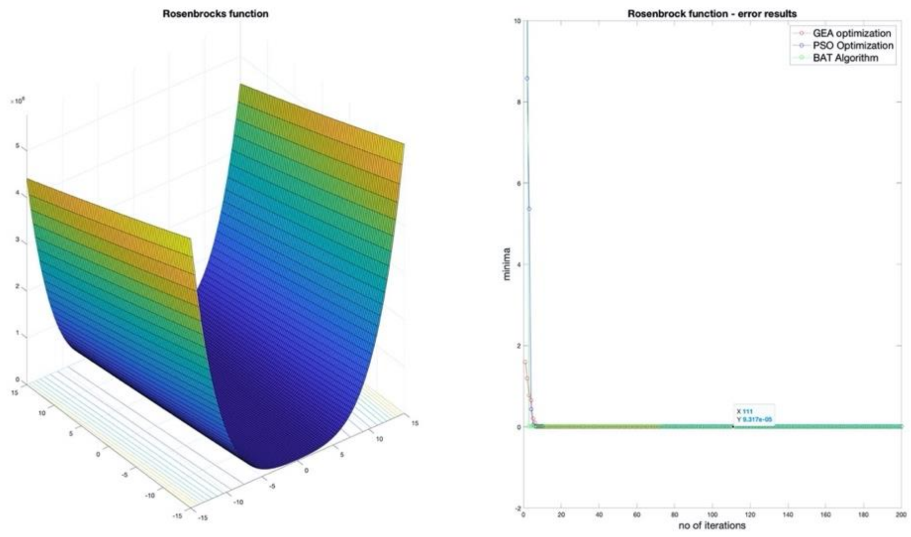


*Figure 15 - Error Curve - Rosenbrock's Function*

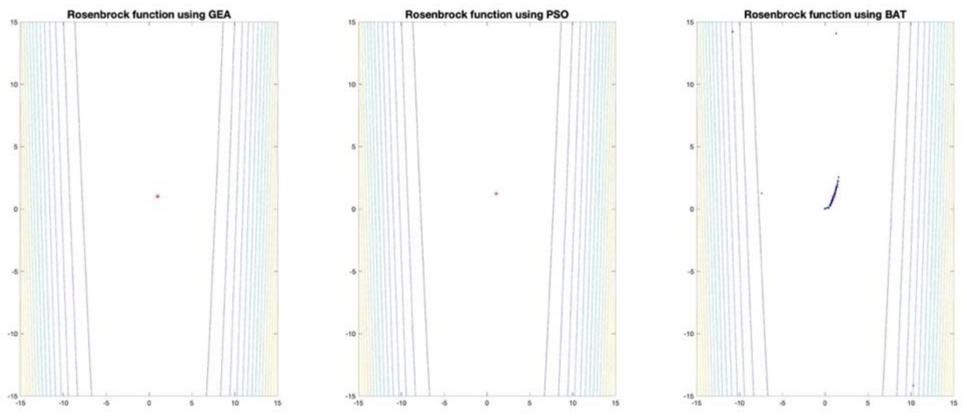
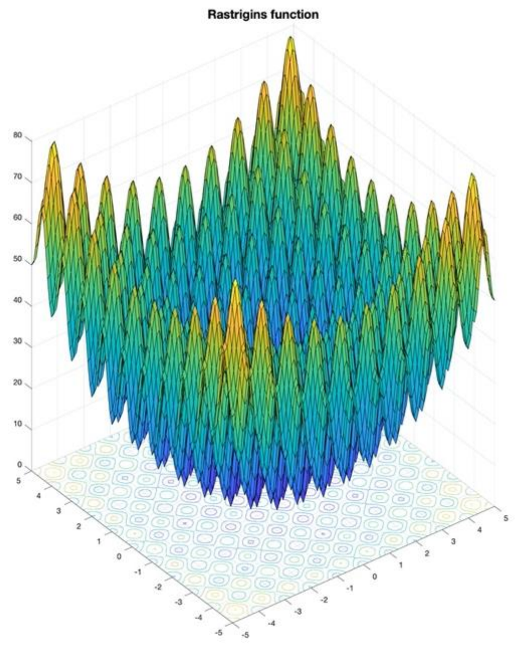


*Figure 16 - Contour Plot - Rosenbrock's function*

As we can see from the above figures, the results for Rosenbrock’s function are very similar for all of the above algorithms. This is because there is only one minimum here and also since there is a gradient which directs the individuals to the global minimum.

### *3.3.5* F5 – Rastrigin’s function:



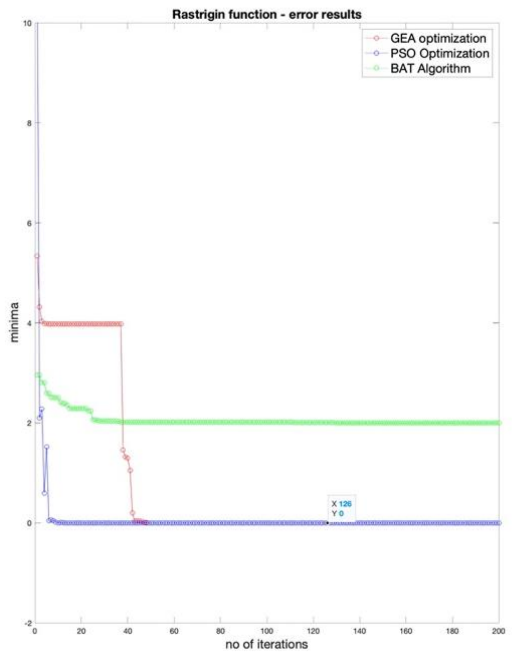


*Figure 17 - Error Curve - Rastrigin's function*

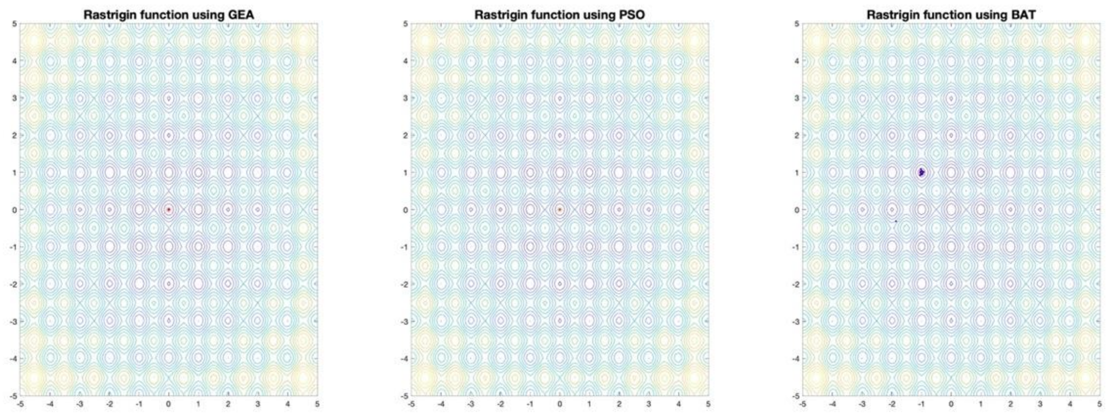


*Figure 18 - Contour Plot - Rastrigin's function*

Once again, similar to the function F3, we see that the Rastrigin's function also consists of multiple global optima and several other local optima. This makes it a lot cumbersome to solve and this is precisely where our heuristic algorithms come in handy. As we see in figure 20, both PSO and GEA are able to locate the precise optima, whereas the performance of BA is lacking by a big margin. Again, the tuning can vastly improve the performance of BA, especially in this case, but the tuning, by itself becomes a hyper-optimization problem, which, we shall not get into greater detail.

### *3.3.6* F6 – Michalewicz function:

The Michalewicz function by itself is a tricky function because it consists of two minima, one global and the other local. There is a high chance that if the solution is found at the local minima, it might just stay there and so also the other way, if the solution has been found at the global minima, there is a good chance of the other individuals converging to this point very quickly. We can see the results from the below figure (22).

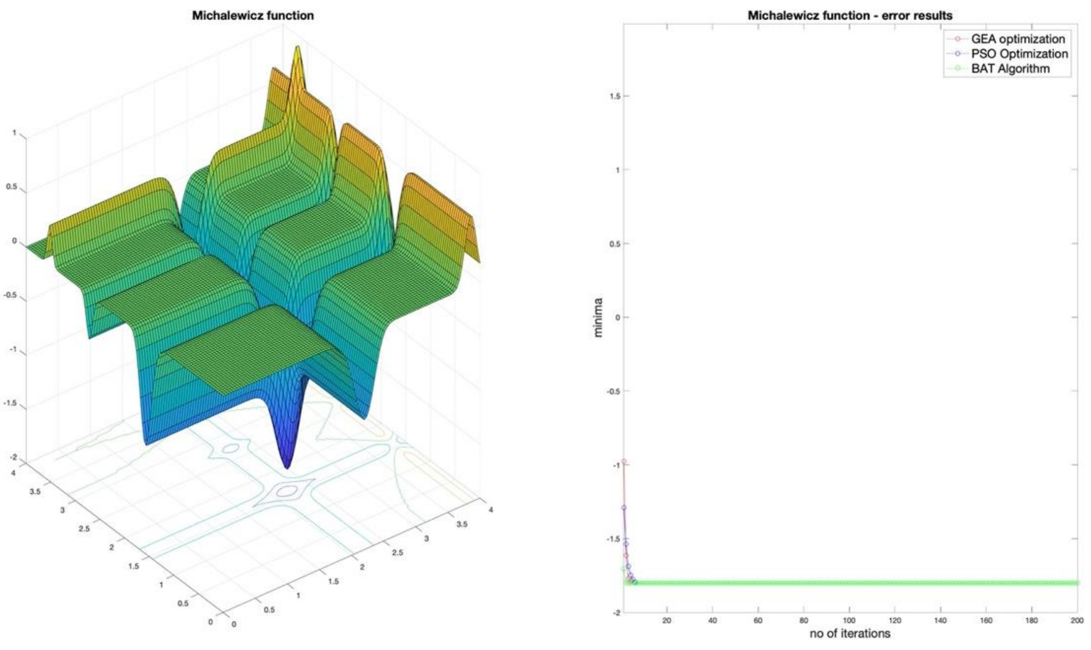


*Figure 19 - Error Curve – Michalewicz Function*

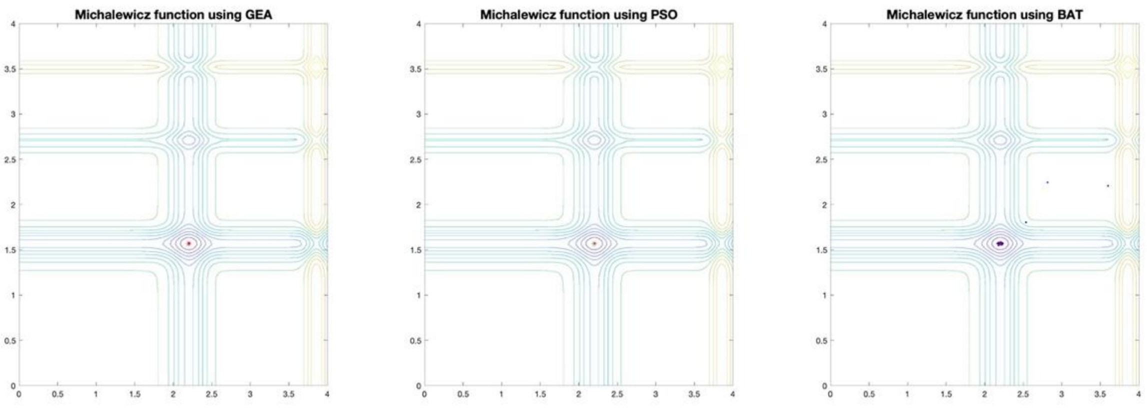


*Figure 20 - Contour Plot - Michalewicz Function*

In this simulation, we see that all three algorithms have converged correctly to the desired point.

## 3.4. Summary:

We can summarize all of the above results in the below table as follows:

| **Function** | **No. of Iterations (N)** | **No. of individuals (n)** | **Best - GEA** | **Best - PSO** | **Best – BA** |
|---|---|---|---|---|---|
| F1 | 100 | 30 | (0,0,0) | (0,0,0) | (0.0764, 0.0109, 0.0059) |
| F2 | 100 | 30 | (0,0,0) | (0,0,0) | (-0.0057, -0.0071, 0.0128) |
| F3 | 100 | 30 | (-0.0072, -0.0801, 0.0016) | (-3,14, 4.438, 0.0074) | (3.14, -4.4530, 0.0074) |
| F4 | 200 | 30 | (1,1, 0) | (1, 1, 0) | (1.0055, 1.0112, 0) |
| F5 | 100 | 30 | (0, 0, 0) | (0, 0, 0) | (-0.99, 0.99, 2) |
| F6 | 100 | 30 | (2.2029, 1.5708, -1.8013) | (2.2029, 1.5708, -1.8013) | (2.2031, 1.5710, -1.8013) |

*Table 2 - Simulation results for pre-defined functions*

## 3.5. Conclusion:

From the above results, we can say that the GEA algorithm is on par with PSO and BA for accurately predicting global minima. While the convergence for unimodal functions are more or less the same, the dynamics are seen to be less consistent where multi-modal functions are concerned. One of the advantages of GEA compared with the other two is that it is much simpler to implement in comparison with PSO and especially BAT due to the fact that there are much fewer parameters for tuning in GEA. With PSO, the above tuning is relatively good as the algorithm functions satisfactorily, whereas, in case of BA, it requires precise tuning of multiple parameters.

With the above results, we shall move to a practical implementation of the above algorithms in identification of multiple order linear systems.

# 4 Relay-Feedback Identification

In this section, we shall move ahead with the previously explained optimization algorithms and try to understand the concept of system identification and then estimate the approximate model for different pre-defined processes. In section 4.1, we shall describe the theory behind System Identification and describe a few different ways to achieve the same. Next, in section 4.2 we shall look at some theoretical models defined by varied types of processes and how to estimate equivalent models using a lower order process. Finally, in section 4.3, we shall apply the above algorithms to these processes and analyse the results of the same.

## 4.1. System Identification

### *4.1.1* Theory:

We can define the term System Identification as the method or process used to estimate the model of a process which could efficiently and accurately predict the dynamics of the given process. By identifying a model of the system, we are then able to predict the outputs of the process for a wide range of inputs within the given domain. Furthermore, the advantage of a well-defined mathematical model is that we can experiment with the system and tweak the system itself so as to optimize it or to test it in advance to analyse the possibility of extreme conditions or failures without having to deal with the physical system or process directly. This gives us a lot more flexibility in design of the process and can help to provide a deeper insight into its functioning.

To begin with, we shall describe the three methods of obtaining a model of a process / system, namely:

1. White Box identification
2. Grey box identification
3. Black box identification

*White box identification* is used when the system dynamics are fully known and can be derived from first principles (Stoev & Schoukens, 2016) and the models derived via White Box identification can be considered as *Theoretical Models*. In case of *grey box modelling*, we need to have some knowledge of the dynamics and combine it with some experimental measurements of the same to obtain our model, the models derived from this method can be called *Semi-empirical Models*. *Black box identification* necessitates no knowledge about the dynamics and is obtained solely by experimentation by fitting experimental data, the models are known as empirical models of the system. In practice, we seldom use white box models as practical systems are far more complicated to be sufficiently well described from first principles. Nevertheless, we do need to have a general idea of the system dynamics which can be useful in helping us decide what kind of model we should use.

For the purpose of this study, we shall look into some of the methods used for *Black box identification* only since our algorithm must work accurately on any system regardless of the internal dynamics. Future uses of the term '*system identification'* will apply to black box identification. To begin with, we shall first define a few important technical terms related to this topic which we will be using repeatedly in the following sections. A lot of the below ideas and methods have been derived from the work of Lennart Ljung's book "System Identification: Theory for the User".

### *4.1.2* Definitions

1. System: This can be described as an object in which variables of different kinds interact and produce observable signals (Ljung, 1987). For example, a simple water-tank with and inlet and an outlet can be a good example of a system.

2. Process Output: Observable signals of interest to the user for the purpose of identification. In case of a water tank, the flow output at the exit can be considered as the process output.

3. Control Input: The signal which is fed into the system from the controller is called the control input. For example, if the water tank is controlled by a PID controller, then the output from the controller can be considered as the control input to the process. In absence of a controller, the control input is the same as the input to the system.

4. Disturbance: Any unwanted input signal which affects the output of the process is called a disturbance.

5. Process Model: A process model is a mathematical representation of the system in terms of variables which can describe the system to a sufficiently acceptable level. A process model for a dynamic system can be in terms of differential equations or it can be a Laplace model of the system.

6. Linear System: A system is said to be linear if it obeys the superposition principle i.e the linear combination of inputs is the same linear combination of the output responses of the individual inputs. Almost all systems are inherently non-linear, but for the purpose of analysis, we most often reduce the system to a linear one so that we can perform various operations on the system which is relatively difficult for non-linear systems.

7. Time-invariant: A system is time invariant is its response to a certain input signal does not depend on absolute time.

### *4.1.3* **Methods in Practice:**

According to (Ljung, 1987), the construction of a model from data involves three basic entities namely:

1. Data: Input/ Output data
2. Set of Candidate Models: This is obtained by specifying what kind of model we are searching for. Prior knowledge from physics and engineering principles as well as general engineering and insight are important in selecting the appropriate model.
3. Rule: A Rule is used to assess the candidate models using the data, i.e. this is the identification method used to select the best model.

The process of identification involves the following basic steps as described in by Ljung (Ljung, 1987):

- Collect the data
- Choose a model set
- Pick the best model in this set based on the above Rule
- Validate the selected model to assess how the model relates to the observed data, to prior knowledge and to its intended use

The process of identification can be described by the below flowchart (figure 21).

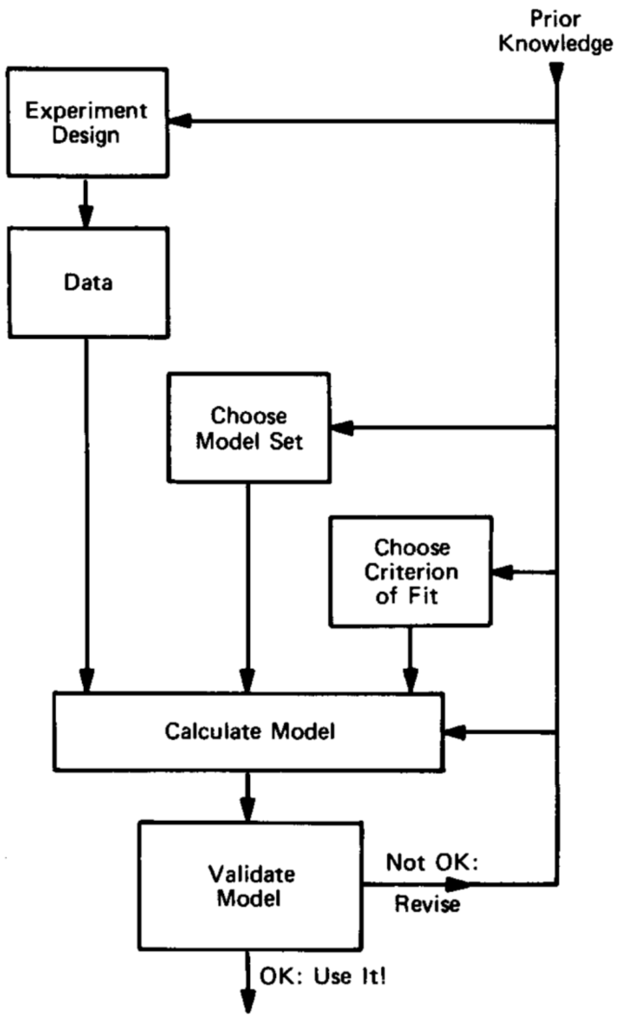


*Figure 21 - System Identification Loop (Llung, 1975)*

## 4.2. Why Identification?

The most basic requirement for industrial processes is 'controlling' the process so as to obtain the desired output for the supplied input signal. In order for us to be able to control the process correctly, we need to select an appropriate controller, which, in turn entails having a fairly good idea of the process which we are controlling first. Thus, the identification of dynamic transfer function models is essential for model-based controller design (Ramakrishnan & Chidambaram, 2003). Due to the complex nature of most chemical processes, it is often difficult to derive the specific model of the entire system, in which cases, system identification provides a tool to identify lower-order models based on input-output data, which is, in most cases, sufficient for us to understand the concrete dynamics of our system. Thus, the idea of identification is to reduce a complex, higher order process to a simple lower order system which can approximately describe the dynamics of the system. In the below sections, we shall try to fit a higher-order system to a second-order model and examine the performance of the same.

As described in the paper "Asymmetic relay autotuning" (Berner, Hägglund, & Åström, 2016), the identification process consists of four sub-tasks, namely, the *Experiment*, the *Model*, the *Controller* and *Evaluation* of the results.

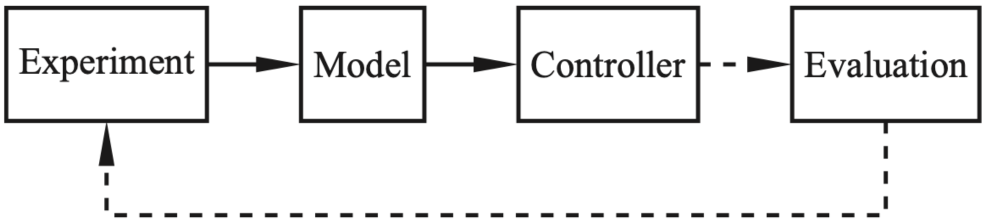


*Figure 22 - Tuning Procedure (Berner, Hägglund, & Åström, 2016)*

1. Experiment: What type of experiment needs to be done and how the experiment is to be designed. Sections 4.3 and 4.4 describe how the experiment is to be conducted.
2. Model: What model structure is to be used? What are the parameters for design? This shall be looked into in section 4.2.
3. Controller: The type of controller is selected based on the model found from the above step. In our case, we shall use a *Proportional – Integral – Derivative* (PID) controller. More details about PID tuning in this concrete case is given in section 4.7.
4. Evaluation: Finally, the results of control must be evaluated to check if the performance of the controller is satisfactory or if something needs to be changed in the previous step. This is to be handled by the user. In this experiment, we shall review the results in section 4.6 and check if our identified model fits our criteria.

According to our concrete experiment, we shall use a Second-Order Plus Time-Delay Model to estimate our lower order model. The reason for using a Second-Order model will be laid out in section 4.4.

## 4.3. Model

As described earlier, a process model is the mathematical representation of a physical system. One way to represent a system is in terms of differential equations. However, in this section and also in the next ones, we shall be representing our processes only in terms of a Laplace model (or Transfer Function Model) i.e. in terms of ‘s’ where **s** is a complex variable of the form: $s = \sigma + j\omega$. More details about Laplace Transforms can be found in (Seborg, Edgar, Mellichamp, & III, 2017). Note that this representation follows if and only if the system is linear, which, for the following purposes, we will use models of *linear time-invariant* systems only. Laplace models are particularly useful in linear systems since it reduces a differential equation into a simple sequence from which, we can easily apply algebraic and mathematical operations

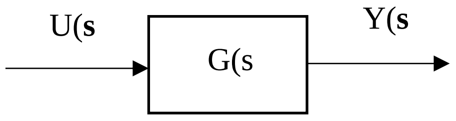


*Figure 23 - System Representation*

Let us consider a simple linear, time invariant system of the form (see fig. 22):

$Y(s) = G(s)U(s)$

Where:

Y(s) = system output, $Y(s) = L\{y(t)\}$

U(s) = system input, $U(s) = L\{u(t)\}$

Then we can describe the system in terms of its transfer function as G(s) in its basic form as:

$$G(s) = \frac{Y(s)}{U(s)} \tag{19}$$

In general, the degree of a linear system can be defined by the highest degree of the ‘s’ term in the denominator of the transfer function. The most basic representation of a higher order dynamic system can be represented as follows:

a) Differential equation:

$$a_n \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_1 \frac{dy}{dt} + a_0 y = b_m \frac{d^m u}{dt^m} + b_{m-1} \frac{d^{m-1} u}{dt^{m-1}} + \cdots + b_1 \frac{du}{dt} + b_0 u \tag{20}$$

b) Transfer function Model: The above differential equation model can be represented in terms of a laplace transform model as follows

$$G(s) = \frac{Y(s)}{U(s)} = \frac{\sum_{i=0}^{m} b_i s^i}{\sum_{i=0}^{n} a_i s^i} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_0} \tag{21}$$

In cases where the process consists of a Time Delay ($T_d$), there can be an extra term in the numerator which accounts for the delay time. This can be represented as:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{e^{-(sT_d)} \sum_{i=0}^{m} b_i s^i}{\sum_{i=0}^{n} a_i s^i} = \frac{(b_m s^m + b_{m-1} s^{m-1} + \cdots + b_0). e^{-sT_d}}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_0} \tag{22}$$

For the purpose of identification, we shall concentrate on the following six types of models:

1. Non-oscillatory Process (Lag-dominated):

$$P_1(s) = \frac{1}{(s+1)(0.1s+1)(0.01s+1)(0.001s+1)} \tag{23}$$

A non-oscillatory lag-dominated process is one in which, the poles (or eigenvalues) of the system do not lie on the imaginary axes as well as these models are with relatively small time-delays ($\frac{\theta}{\tau} \ll 1$).

2. Balanced Process:

$$P_2(s) = 1/(s+4)^4 \tag{24}$$

3. Delay-dominated Process:

$$P_3(s) = \frac{1}{(0.05s+1)^2} e^{-s} \tag{25}$$

4. Oscillatory Process:

$$P_4(s) = \frac{1}{(0.5s^2+s+1)^2} \tag{26}$$

5. Non-oscillatory Process with Time Delay:

$$P_5(s) = \frac{1}{(s+1)(0.3s+1)^2} e^{-sT_d} \tag{27}$$

6. Fifth Order Process with Time Delay:

$$P_6(s) = \frac{e^{-s}}{(5s+1)^5} \tag{28}$$

## 4.4. Relay-Feedback Identification

The use of an on-off relay to generate a sustained oscillation in the control loop was proposed by Astrom and Hagglund (1984). This is a closed-loop method of identification of transfer function models which is based on the observation that when an open loop output lags the input by $\pi$ radians, then the closed loop system may oscillate with a period $P_u$ ( (Chidambaram & Sathe, 2014). The ultimate gain $K_u$ and ultimate frequency $\omega_u$ can be calculated from the response as (Astrom & Hagglund, 1984):

$$K_u = \frac{4h}{\pi a} \tag{29}$$

$$\omega_u = \frac{2\pi}{P_u} \tag{30}$$

Where $h$ = magnitude of the relay, a = process output, $P_u$ = ultimate period, $\omega_u$ = ultimate frequency

For the purpose of identification, we shall use the concepts based on the Article “Alternative Identification Method using Biased Relay Feedback” (Hofreiter, Alternative Identification Method using Biased Relay Feedback, 2018). The paper proposes a method of System Identification of Dynamical Systems using a relay. By means of this method, the asymmetric relay is used in a closed loop circuit forcing the system into self-sustained oscillations. This can be illustrated by means of a closed-loop system as illustrated in figure 22.

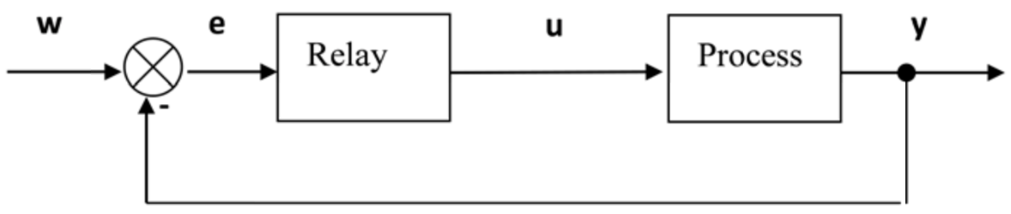


*Figure 24 - Block Diagram of a Process under Relay Feedback*

As seen in the figure (23), the set-point ‘w’ supplies the input to the closed-loop system, the error ‘e’ from the set-point is fed to the relay from which we get the manipulated variable ‘u’ which is supplied into the process to produce the controlled variable ‘y’. Relay controllers or on-off controllers are simple feedback controllers which are commonly used in simple systems such as heating systems. Typical on-off controllers consist of two possible output values and a dead-zone:

1. $u(t) = u_A$ if $e \geq \epsilon_A$
2. $u(t) = u_B$ if $e < \epsilon_B$
3. Dead-band if $\epsilon_A < e \leq \epsilon_B$

The above can be demonstrated in the below figure (24)

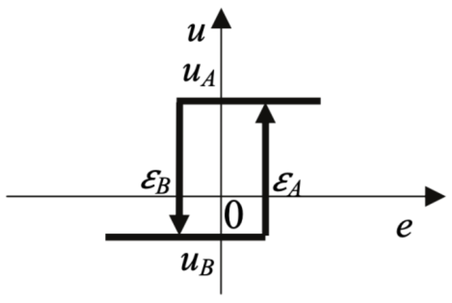


*Figure 25 - Asymmetric Relay (Hofreiter, Alternative Identification Method using Biased Relay Feedback, 2018)*

The advantage of the use of an asymmetric relay instead of a symmetric one has been described by the paper “Asymmetric Relay Autotuning” (Berner, Hägglund, & Åström, 2016). As shown by Berner et al., asymmetric relay provides a better excitation of the process at lower frequencies than its symmetric counterpart without complicating the experiment in general. We shall further confirm this during identification of real processes in section 5.

Now, that we have described the working of the relay, we shall further examine our experiment. The block diagram of our setup is according to the below figure (25).

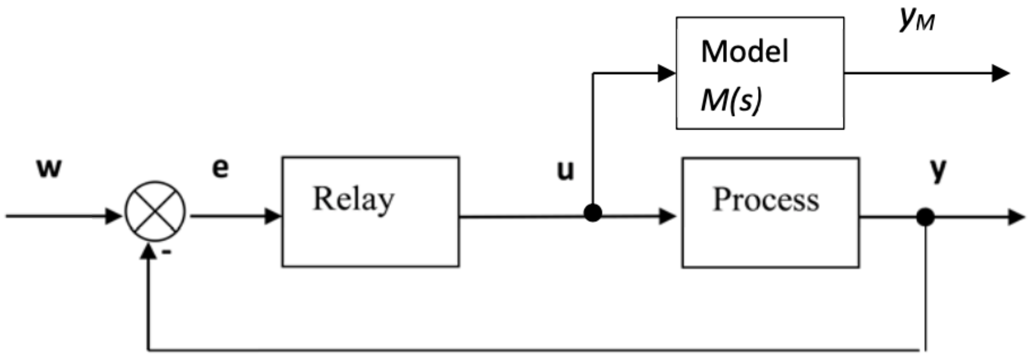


*Figure 26 - Experimental Block Diagram*

As seen in the above block diagram (figure (25)), the output of the relay controller is fed into the process G(s) as well as to our model M(s) from which we obtain the outputs y and $y_M$ respectively. From the values of y and $y_M$, we try to evaluate the error which gives us the difference which we square to obtain the 'squared error'. If we integrate the error over time, we attain the ITAE criterion for our cost function. This can be seen from figure (26).

## 4.5. Problem Statement

In section 4.2, we have defined the six different types of system and also described the procedure used for identification. For the purpose of approximation, we shall use a simple dynamic SOPDT model which can replicate the dynamics of these systems to a sufficiently accurate degree as explained in section 4.3. The advantage of this is that the SOPDT model can describe almost any linear system. As explained by Ramakrishnan and Chidambaram, the SOPTD model can incorporate various processes such as under-damped and higher order processes in which case, an FOPTD model is not sufficient (Ramakrishnan & Chidambaram, 2003). Furthermore, SOPTD models can also be used for unstable processes in which case, an FOPTD model is not sufficient. Additionally, the optimization techniques from section 3, for estimating the parameters of our model can work on non-linear systems as well.

A simple SOPDT model can be described as below:

$$G(s) = \frac{K}{a_1 s^2 + a_2 s + 1} e^{-sT_d} \tag{31}$$

Where, 'K' = Process gain, '$\theta$' = Time Delay, '$a_1$' and '$a_2$' are dynamic constants of the transfer function. These parameters $\{K, a_1, a_2, T_d\}$ are unknown parameters which we are to estimate using the GEA algorithm described in section [2]. We can describe the unknown parameters using the below representation:

$$x_j = \{a_1, a_2, K, T_d\} \tag{32}$$

Where:

$j =$ Dimension index (1:d)

$d =$ Number of dimensions (= 4 in this case)

The cost function for optimization of the above parameters can be summarized as:

$$J = \int_0^{T_k} \left(y_m(t) - y(t)\right)^2 dt \tag{33}$$

$y_m =$ model output

$y_t =$ system output

$T_k =$ simulation time

Therefore, we can construct our problem statement as follows:

Find the parameters $\boldsymbol{x_i}$ (i = 1,2 … d) to minimize the cost function J,

Subject to constraints:

$\{\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{K}, \boldsymbol{T_d}\} > \boldsymbol{0}$

For the purpose of identifying the below systems, we shall consider $T_d$ in the range (0,5).

## 4.6. Simulink Scheme

Now that we have our identification model, we can try to set up our system to construct our optimization criteria. In the below Simulink model (figure 23), the transfer function G represents the real system (in this case it is given by equations 23-28). The signal input to the system is provided by means of a relay controller, also known as an 'On-off controller'. Finally, the cost function J is found out by using the ITAE criteria (Integral Time Absolute Error).

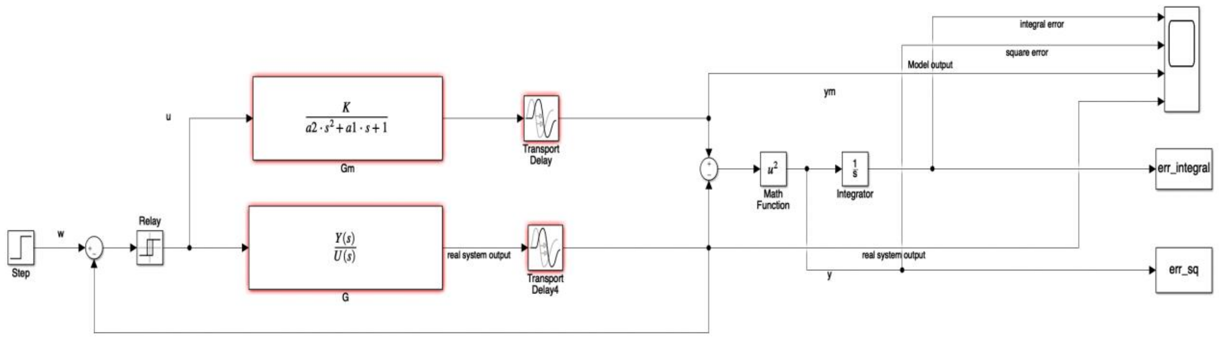


*Figure 27 - Relay Identification Schematic*

For simulating the functions described in section 4.2 (equations 23-28), we have used the following parameters for our closed-loop system:

1. $u_A = 2$
2. $u_B = -1$
3. $\epsilon_A = 0.1$
4. $\epsilon_B = -0.1$
5. $w = 0$
6. Simulation Time, Tsim = 200s

The results of the simulation are logged into the MATLAB Workspace as *err_integral*, which, in turn is our cost function *J* (or *fitness*) for optimization. For each solution, we obtain one value of *err_integral and w*ith every iteration the parameters $\boldsymbol{x_i}$ are routinely updated to minimize the cost function.

The flow-chart of the above process can be shown as below:

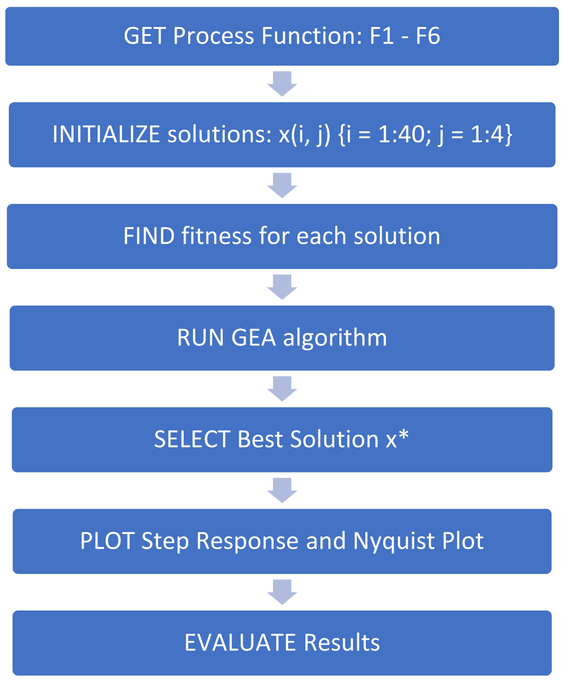

GET Process Function: F1 - F6
INITIALIZE solutions: x(i, j) {i = 1:40; j = 1:4}
FIND fitness for each solution
RUN GEA algorithm
SELECT Best Solution x*
PLOT Step Response and Nyquist Plot
EVALUATE Results

*Figure 28 – Flowchart of Identification Process*

## 4.7. PID Control

We explained in section 4.2, that the most important purpose of system identification is to understand the process so that we can then *control* the system. We have already described the identification procedure in section 4.6, hence we can now move to controlling the above processes. Most of the systems described in section 4.2 are inherently stable by default. Hence our main goal for control would be to minimize the control error as well as to improve the response time of the system, while keeping the overshoot to the minimum.

The most commonly used controller in industrial applications is the *Proportional-Integral-Derivative Controller* (PID controller). A PID controller uses closed-loop control in an Industrial Control System or Chemical Process by calculating the error value $e(t)$ as the difference between the setpoint (SP) and a measured process output, applying corrective action based on Proportional, Integral and Derivative terms (P, I and D) respectively.

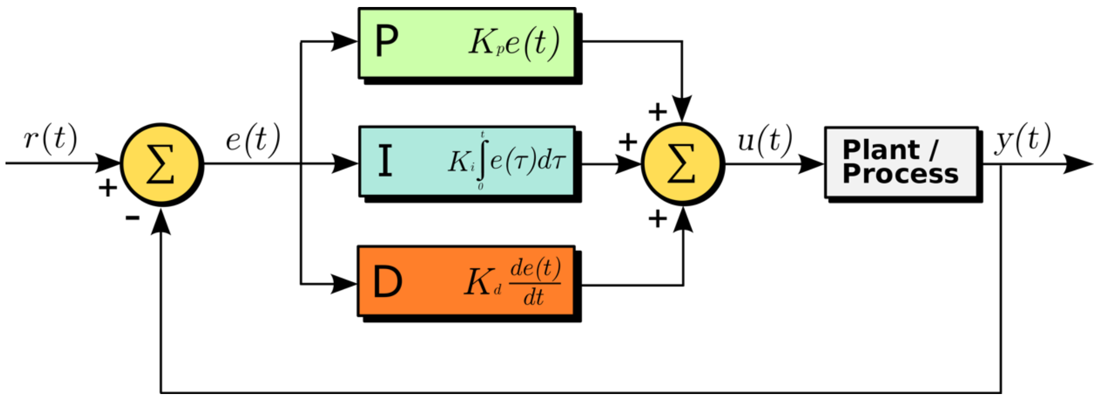


*Figure 29 - PID Controller Block Diagram (PID Controller, 2020)*

The general equation of the PID controller in Laplace-Domain can be given as:

$$R(s) = r_P + \frac{r_I}{s} + r_D s \tag{34}$$

Here,

$r_P$ = Proportional Gain – P-term $(K_p)$

$r_I$ = Integral Gain – I-term $(K_i)$

$r_D$ = Derivative Gain – D-term $(K_d)$

The ***P-term*** is proportional to the error $e(t)$, whereby, the controller produces a response proportional to the error value. The ***I-term*** provides control action by integrating the error over time and produces the response depending on the magnitude of the error. The larger the response, the larger the integral part and once the error is eliminated, the integral term ceases to grow. The ***D-term*** estimates the future trend of the error based on the current rate of change of error, thus, providing ‘anticipatory control’ action or dampening effect. Using the control error $e(t)$ as input, the controller attempts to adjust the control variable $u(t)$, thus, minimizing the error over time.

The major challenge of using a PID Controller lies in tuning the parameters $r_p$, $r_i$ and $r_d$ for appropriate control action. The parameters can be tuned depending on the control action we desire and depending on our control criteria such as: disturbance rejection, setpoint tracking, peak overshoot, etc. There exist several methods for tuning PID controllers of which, the most popular ones are Ziegler-Nichols, Cohen-Coon, Astrom-Hagglund methods of PID Controller tuning. With most of these methods, we obtain the tuning parameters by experimentation and evaluating the process for its instantaneous response. However, instead of using experimental methods, we are going to focus on PID tuning methods based on system identification techniques which, in this case is based on Relay-based Identification approach.

As part of this thesis work, we shall look into the below four methods for tuning our PID Controller namely:

a) Direct Synthesis Method (DS) or *Lambda Tuning Method*

b) Phase Margin Criterion (PMC) based PID Controller Tuning (Hofreiter, Zaklady Automatickeho Rizeni, 2016)

c) Simple Control (SIMC Tuning Method)

Before proceeding with the tuning rules, we shall reduce our model $G_M(s) = K.\frac{e^{-T_d s}}{a_2 s^2 + a_1 s + 1}$ to one of the below forms:

A) Oscillatory Processes:

$$G(s) = \frac{(K\omega_0^2)e^{-sT_d}}{s^2 + 2\zeta\omega_0 s + \omega_0^2} \tag{35}$$

Where $\omega_0 =$ oscillation frequency of system and $\zeta =$ damping ratio

B) Non-Oscillatory Processes:

$$G(s) = \frac{Ke^{-sT_d}}{(T_1 s + 1)(T_2 s + 1)} \tag{36}$$

Where $T_1, T_2 =$ Time constants of modelled system

### *4.7.1* Direct Synthesis Method (DS):

The Direct Synthesis (DS) Method was proposed by Seborg and Chen for set-point tracking as well as for disturbance rejection in Closed-Loop Circuits. According to this method, the controller design is based on a process model and a desired closed-loop transfer function (Chen & Seborg, 2002).

Consider the block diagram of a closed loop system as shown in the figure (30).

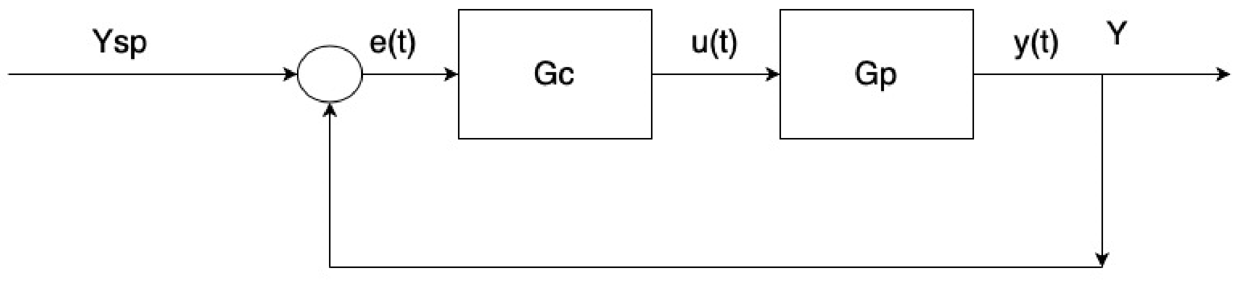


*Figure 30 - Block Diagram - Closed Loop System*

From the above figure, we can obtain the closed-loop transfer function of the system as follows:

$$\frac{Y}{Y_{sp}} = \frac{G_C G}{1 + G_c G} \tag{37}$$

Thus we obtain the expression for $G_c$ as:

$$G_C = \frac{1}{G} \cdot \frac{\frac{Y}{Y_{sp}}}{1 - \frac{Y}{Y_{sp}}} \tag{38}$$

Since we do not know the value of $Y/Y_{sp}$ a priori, we therefore cannot directly use the above equation. However, we can perform a few substitutions in the above equations to achieve the same.

$$G_c = \frac{1}{\bar{G}} \cdot \frac{\left(\frac{Y}{Y_{sp}}\right)_d}{1 - \left(\frac{Y}{Y_{sp}}\right)_d} \tag{39}$$

Where,

$\bar{G}$ = Model of actual process

$\left(\frac{Y}{Y_{sp}}\right)_d$ = desired closed-loop transfer function

To obtain the closed-loop transfer function, we would need to approximate the closed-loop response to a First order system with or without delay from which, we can tune our controller to achieve the desired objective.

Since our Process model already incorporates a time delay, we thus approximate our closed-loop transfer function in terms of a First Order Plus Time Delay model (FOPDT):

$$\left(\frac{Y}{Y_{sp}}\right)_d = \frac{e^{-(sT_d)}}{\tau_C s + 1} \tag{40}$$

Where, $\tau_C$ = closed loop time constant

From equations (40) and (41), we thus obtain the following:

$$G_c = \frac{1}{\bar{\bar{G}}}.\frac{e^{-(sT_d)}}{\tau_c s + 1 - e^{-(sT_d)}} \tag{41}$$

Where, $\tau_c$ = desired closed loop time constant

Using Taylor's Series approximation for $e^{-(sT_d)} = 1 - T_d s$, we obtain:

$$G_c = \frac{1}{\bar{\bar{G}}}.\frac{e^{-(sT_d)}}{(\tau_c + T_d)s} \tag{42}$$

The above controller action that we see above contains also the integral control action.
It must be noted that choosing the value of the time constant $\tau_c$ is key in achieving the desired functionality of the closed-loop system. We can choose the value arbitrarily or based on experience. A low value of $\tau_c$ makes the controller tuning more aggressive, whereas a high value reduces the effectiveness of the controller. A general approach to selecting the value of $\tau_c$ is by using the below equation (44):

$$\tau_{dom} > \tau_c > T_d \tag{43}$$

Where,

$\tau_{dom}$ = dominant time constant of the process

As a thumb rule, for the three tuning methods, we shall choose the value of $\tau_c$ using the relation:

$$\tau_c = \frac{\tau_{dom}}{3} \tag{44}$$

With the above background, the tuning relations for our PID Controller using the Direct Synthesis Method for a SOPTD system can be given by:
For non-oscillatory systems,

$$r_p = \frac{1}{K}.\frac{T_1 + T_2}{\tau_C + T_d}; \tag{45}$$

$$\tau_I = T_1 + T_2;\ r_i = r_p/\tau_i$$

$$\tau_d = \frac{T_1 T_2}{T_1 + T_2};\ r_d = r_p.\tau_d$$

For oscillatory systems,

$$r_p = \frac{2\zeta T_0}{\tau_c + T_d}; \tag{46}$$

$$\tau_i = 2\zeta T_0;\ r_i = \frac{r_p}{\tau_i}$$

$$\tau_d = \frac{T_0}{2\zeta};\ r_d = r_p.\tau_d$$

Note that this method can be applied to oscillatory as well as non-oscillatory systems.

### *4.7.2* PMC Method

The PMC Method of tuning sets the PID parameters by requiring that the controlled closed loop phase margin is $\frac{\pi}{4}$, hence the name 'Phase Margin Tuning' (Hofreiter, Zaklady Automatickeho Rizeni, 2016).

According to this method, the tuning rules for a process model $G(s) = \frac{Ke^{-(sT_d)}}{a_2s^2+a_1s+1}$ can be given by the following equations:

a) For $T_d > 0$,

For non-oscillatory processes, (47)

$$r_i = \frac{\pi}{K.4.T_d}$$

$$r_p = a_1.r_i$$

$$r_d = a_2.r_i$$

For oscillatory processes,

$m_a = \frac{\pi}{2T_d m_a K}$, where $m_a$ = gain margin <2:5>

$$r_p = a_1 r_i$$

$$r_d = a_2 r_i$$

b) For $T_d > 0$,

For non-oscillatory processes, (48)

$$r_i = \frac{1}{K\tau_c}$$

$$r_p = a_1 r_i$$

$$r_d = a_2 r_i$$

Once again, similar to the DS tuning method, we need to choose a suitable closed-loop time constant $\tau_c$ for our closed loop circuit $G_{wy}(s) = \frac{1}{\tau_c s+1}$. We can use the same equation (44) for choosing a suitable $\tau_c$.

### *4.7.3* SIMC Method

The SIMC Tuning Method (Skogestad's IMC Tuning Method) is based on a non-oscillatory model of the controlled process. The tuning relations according to this method can be given by the following equations (Skogestad, 2004):

a) For $T_1 \leq 8T_d$ and $T_1 > T_2$,

$$r_p = \frac{0.5}{K} \cdot \frac{T_1 + T_2}{T_d} \tag{49}$$

$$\tau_i = T_1 + T_2 = a_1; r_i = \frac{r_p}{\tau_i}$$

$$\tau_d = \frac{T_2}{1+\frac{T_2}{T_1}}; r_d = r_P . \tau_d$$

b) For $T_1 = 8T_d$ and $T_1 > T_2$,

$$r_p = \frac{0.5}{K} \cdot \frac{T_1}{T_d}\left(1 + \frac{T_2}{8T_d}\right) \tag{50}$$

$$\tau_i = 8T_d + T_2, r_i = \frac{r_p}{\tau_i}$$

$$\tau_d = \frac{T_2}{1+\frac{T_2}{8T_d}}, r_d = r_P . \tau_d$$

c) For $T_d = 0$, approximate the Second Order model as a first order process with delay, i.e. $G(s) = \frac{Ke^{-(\theta s)}}{\tau_1 s+1}$ using the following relations:

$$\tau_1 = T_1 + \frac{T_2}{2}; \text{ where } T_1 > T_2 \tag{51}$$

$$\theta = \frac{T_2}{2};$$

Using the above, we can use the tuning relations as follows:

$$r_p = \frac{1}{K} \cdot \frac{\tau_1}{\tau_c + \theta} = \frac{1}{k'} \cdot \frac{1}{\tau_c + \theta} \tag{52}$$

$$\tau_i = \min\{\tau_1, 4(\tau_c + \theta)\}, r_i = r_p/\tau_i$$

$$\tau_d = T_2, r_d = r_p . \tau_d$$

### *4.7.4* Simulink Scheme – PID Control

With the help of the tuning relations specified in section 4.7.1, 4.7.2 and 4.7.3, we can now proceed to tune our controller.

For PID control of the above processes, we use the following Simulink scheme as described in figure (31). As can be seen in the below figure, we have a switch sw1 which is used to switch between the relay and the PID controller. In the first part of the program for identification, the manual switch is connected to the relay to create stable oscillation in the circuit. Once the identification parameters are found, the switch sw1 is then connected to the PID controller, which in turn controls the process Gp(s). To see the controller in action, we will look at the performance for Set-point tracking by means of step input S1 as well as for disturbance rejection D1.

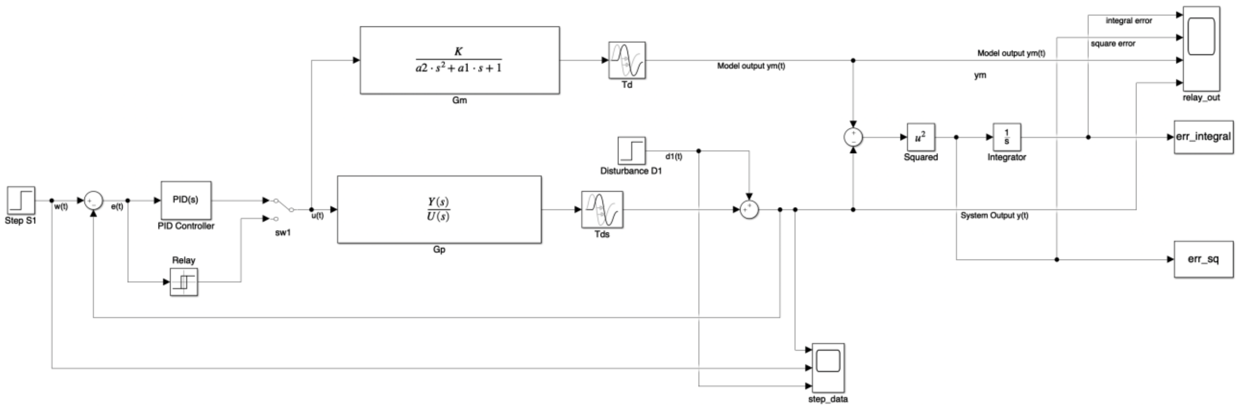


*Figure 31 - Simulink Scheme - PID Control*

The parameters of the step input and disturbance are set as follows:

| Input | Step time | Initial Value | Final Value |
|---|---|---|---|
| S1 | 10 | 0 | 1 |

*Table 3 - Step Input Parameters*

With reference to the above table, we shall observe the system output y(t) only for setpoint tracking (servo problem) with respect to input w(t), since we would need to use different relations for disturbance rejection (regulator problem). For setpoint tracking (servo problem), the output of the system will have to change to the setpoint value, whereas, for disturbance rejection (regulator problem), the output of the system will have to revert back to its value prior the occurrence of the disturbance.

## 4.8. Identification and Control Results for pre-defined systems

Based on the above system settings, the functions F1-F6 were simulated using the Simulink Scheme described in figure (31). Each function was evaluated ten times from which we obtained the following results.

| No. | Function | K | a2 | a1 | Td | fmin | Elapsed Time (s) |
|---|---|---|---|---|---|---|---|
| 1 | $F1 = \frac{1}{(s+1)(0.1s+1)(0.01s+1)(0.001s+1)}$ | 0.998 $\pm$ 0.096 | 0.0674 $\pm$ 0.049 | 1.172 $\pm$ 0.235 | 0.033 $\pm$ 0.039 | 0.0207 $\pm$ 0.023 | 92.37 $\pm$ 33.76 |
| 2 | $F2 = \frac{1}{(s+1)^4}$ | 0.966 $\pm$ 0.083 | 3.32 $\pm$ 0.369 | 3.034 $\pm$ 0.26 | 0.891 $\pm$ 0.099 | 0.122 $\pm$ 0.049 | 93,9 $\pm$ 45,56 |
| 3 | $F3 = \frac{e^{-s}}{(0.05s+1)^2}$ | 1.002 $\pm$ 0.009 | 0.00038 $\pm$ 0.001 | 0.0823 $\pm$ 0.008 | 1.029 $\pm$ 0.029 | 0.324 $\pm$ 0.139 | 135.1 $\pm$ 50.1 |
| 4 | $F4 = \frac{1}{0.5s^2 + s + 1}$ | 0.943 $\pm$ 0.065 | 1.017 $\pm$ 0.122 | 1.147 $\pm$ 0.107 | 0.364 $\pm$ 0.105 | 0.364 $\pm$ 0.105 | 107.4 $\pm$ 39.65 |
| 5 | $F5 = \frac{e^{-s}}{(s+1)(0.3s+1)^2}$ | 0.99 $\pm$ 0.10 | 0.515 $\pm$ 0.153 | 1.437 $\pm$ 0.114 | 1.122 $\pm$ 0.105 | 0.155 $\pm$ 0.107 | 107,83 $\pm$ 57,30 |
| 6 | $F6 = \frac{e^{-s}}{(5s+1)^5}$ | 1.022 $\pm$ 0.217 | 109.9 $\pm$ 13.17 | 17.17 $\pm$ 1.33 | 8.20 $\pm$ 0.66 | 0.222 $\pm$ 0.058 | 83.65 $\pm$ 69.09 |

*Table 4 - Identification Results for Pre-defined Functions*

The correctness of the identified functions can be evaluated by plotting the Nyquist and Step response plots of the real system and the identified system. While the step-response plot helps us evaluate the transient characteristics of the system, the Nyquist plot helps evaluate the frequency characteristics of the same.

### *4.8.1* P1 – Non-Oscillatory, Lag-Dominated Process

Figure (29) shows the step response and nyquist plots of the process P1 and of the SOPDT model of the same. We see that the output of the model is consistent as it matches the output of the system with a standard deviation of $\pm$ 10%.

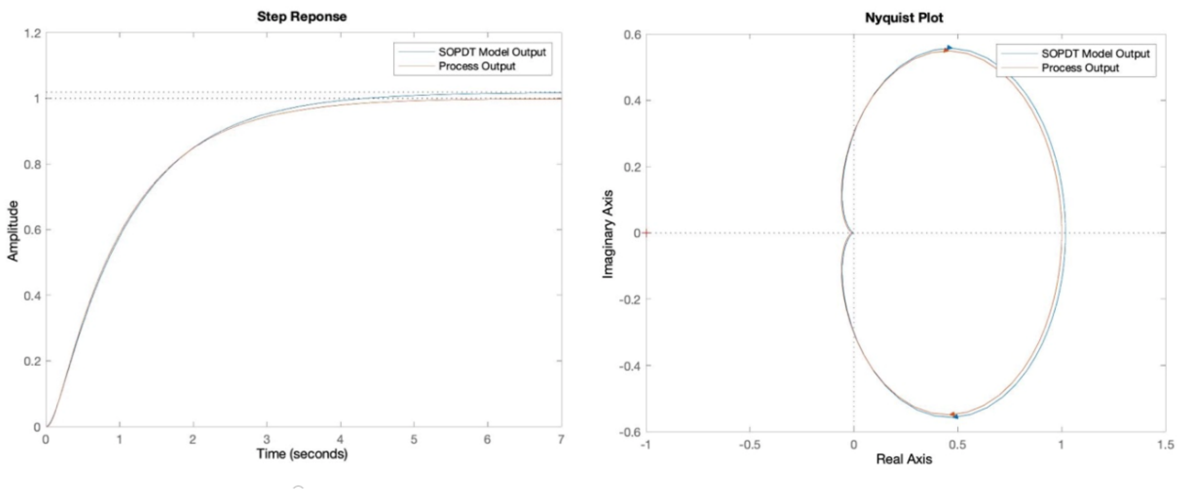


*Figure 32 - Step Response vs Nyquist Plot - Process P1*

Using the Relay Identification Procedure on the Process $P_1(s) = \frac{1}{(s+1)(0.1s+1)(0.01s+1)(0.001s+1)}$, we identify the SOPTD model as (using result no. 10):

$$G_{P1}(s) = \frac{0.9843}{0.1095s^2 + 1.038s + 1} \tag{53}$$

Where,

$K = 0.9843, a_2 = 0.1095, a_1 = 1.038, T_d = 0$

From the above equation (53), we obtain the following:

$$T_1 = 0.9188\text{s}, \;\; T_2 = 0.1192\text{s}, T_0 = 0.3309, \zeta_0 = 1.5684 \tag{54}$$

We can see that the described process is a non-oscillatory system and hence we can use the relations from equations (45) for the DS Method, (48) from the PMC method and (51), (52) from the SIMC Methods to obtain the following:

| | | **Tuning Method** | | |
|---|---|---|---|---|
| **No.** | **Parameter** | **DS** | **PMC** | **SIMC** |
| 1 | **Proportional Gain, rp** | 3.44 | 10.82 | 2.72 |
| 2 | **Integral Gain, ri** | 3.32 | 10.42 | 2.96 |
| 3 | **Derivative Gain, rd** | 0.36 | 1.14 | 0.32 |

*Table 5 - PID Tuning Parameters - Process P1*

Using the above tuning parameters, we obtain the PID Control response as per figure (33).

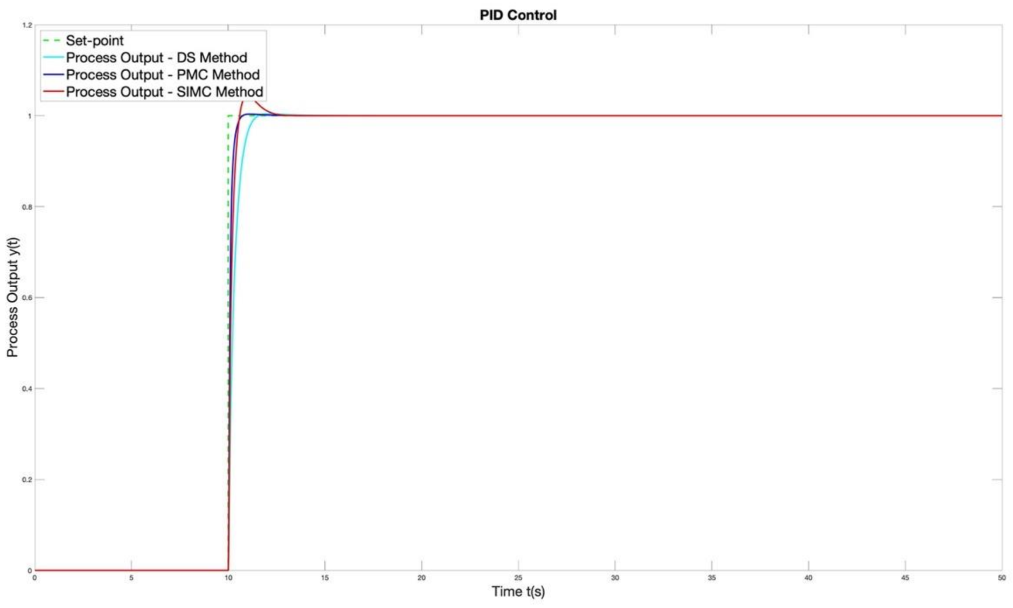


*Figure 33 - PID Control - Process P1*

From the above curves, we see that the DS Method and the PMC Methods both have good tuning characteristics, while the SIMC Tuning method has a slight overshoot with a longer settling time. The results obtained by DS and PMC are acceptable whereas, for SIMC tuning, the overshoot can be a problem.

### *4.8.2* P2 – Balanced Process

Figure (34) shows the step response and the Nyquist plot of the identified process P2, which, as we can see, closely resembles the real process output.

Again, we use our optimization algorithm to identify the process $P_2(s) = 1/(s+1)^4$, from which, we obtain the following SOPTD model (from result no. 7):

$$G_{P2}(s) = \frac{0.9446e^{-0.8837s}}{3.3514s^2 + 2.9958s + 1} \tag{55}$$

Where, $K = 0.9446$, $a_2 = 3.3514$, $a_1 = 2.9958$, $T_d = 0.8837$s

Using this model, we can obtain the parameters $T_0$ and $\zeta_0$ as:

$$T_0 = 1.8307, \zeta_0 = 0.8182 \tag{56}$$

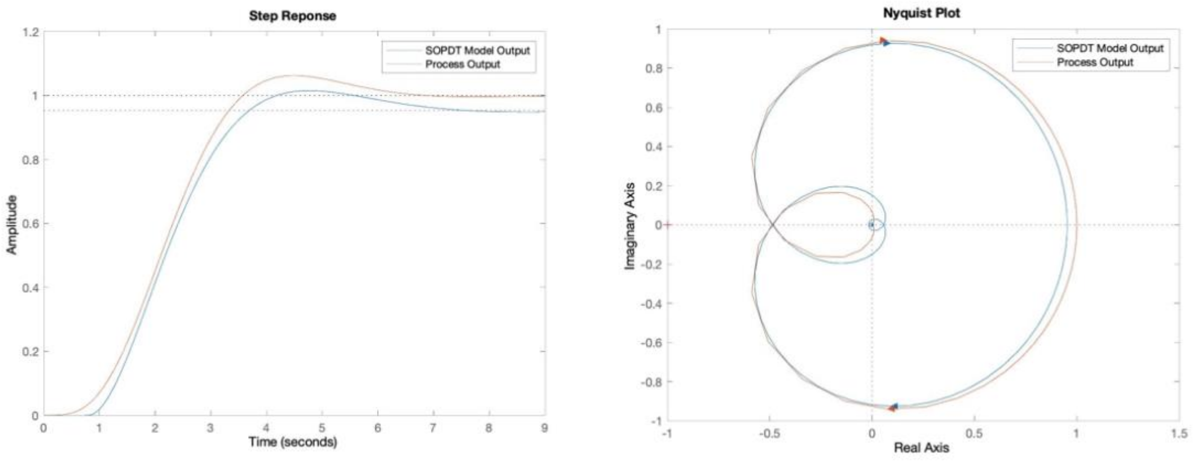


*Figure 34 - Step Response vs Nyquist Plot - Process P2*

Since the above process is oscillatory with time delay, we use the relations from equations (46) and (47) to obtain the PID tuning parameters as can be seen in table (6).

| | | Tuning Method | | |
|---|---|---|---|---|
| **No.** | **Parameter** | **DS** | **PMC** | **SIMC** |
| 1 | **Proportional Gain, rp** | 2.01 | 1.88 | - |
| 2 | **Integral Gain, ri** | 0.67 | 0.63 | - |
| 3 | **Derivative Gain, rd** | 2.24 | 2.10 | - |

*Table 6 – PID Tuning Parameters - Process P2*

The results of the PID tuning can be shown as below:

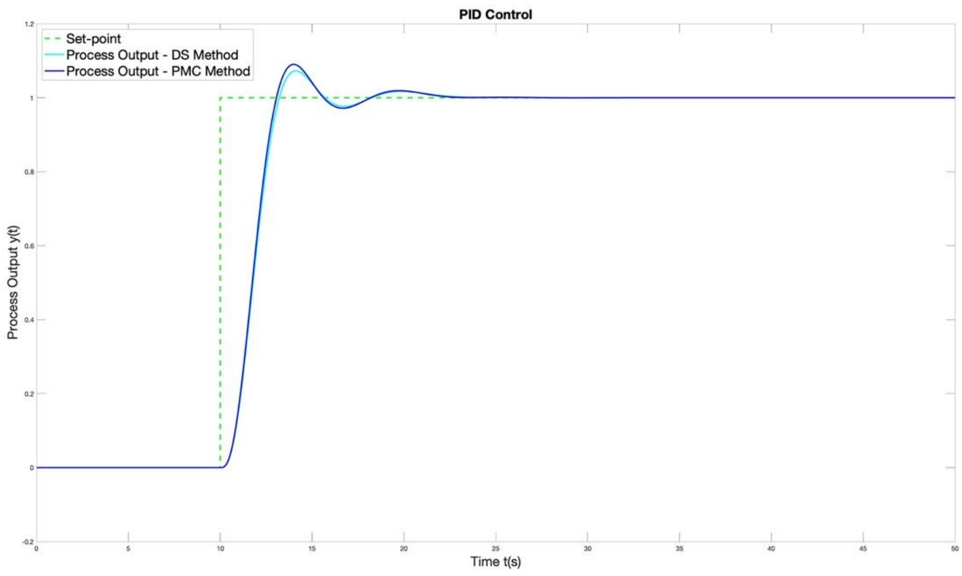


*Figure 35 - PID Control - Process P2*

From the above figure, we see oscillations in the process output in spite of the control action. Therefore, both the above tuning methods are ineffective in this case. One of the reasons for the incorrect tuning is in our selection of the time constant $\tau_c$.

### *4.8.3* P3 – Delay-dominated Process

Figure (36) shows the step plot and the Nyquist plot of the process P3 respectively. As can be seen from the Nyquist plot, the plot of the process output is consistent with the modelled output at low frequencies but tends to deviate from the process output as the frequency increases. This is due to the fact that the system is a non-minimum phase system due to the time-delay.

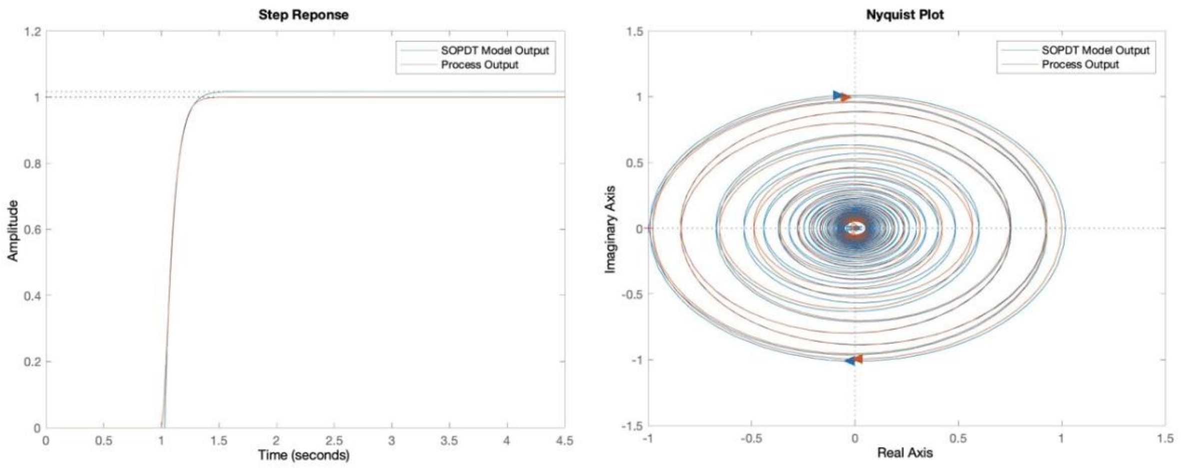


*Figure 36 - Step Response vs Nyquist Plot – Process P3*

The SOPTD model identified for the above process $P_3(s) = \frac{1}{(0.05s+1)^2} e^{-s}$ can be given by:

$$G_{P3}(s) = 1.0175.\frac{e^{-1.0285s}}{0.0786s + 1} \tag{57}$$

Where,

$K = 1.0175, T_d = 1.0285, a_2 = 0, a_1 = 0.0786$

Using the above model, we obtain the dynamic parameters as follows:

$$T_1 = 0.0786 \tag{58}$$

With this, we obtain the tuning results as per table (7),

| | | Tuning Method | | |
|---|---|---|---|---|
| **No.** | **Parameter** | **DS** | **PMC** | **SIMC** |
| 1 | **Proportional Gain, rp** | 0.38 | 0.06 | 0.04 |
| 2 | **Integral Gain, ri** | 0.64 | 0.75 | 0.48 |
| 3 | **Derivative Gain, rd** | 0.03 | 0.00 | 0.00 |

*Table 7 - PID Tuning Parameters - Process P3*

The results with the tuning can be seen in the figure (37).

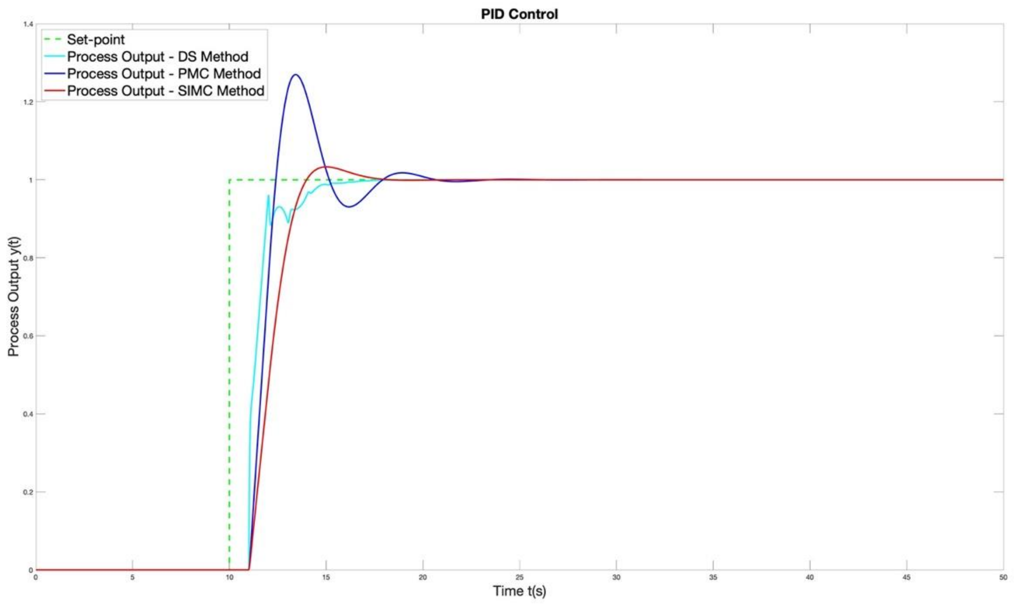


*Figure 37 - PID Control - Process P3*

We see from the above that the DS and the SIMC tuning results provide satisfactory results for controlling the process P3, but the PMC tuning produces oscillations and also a peak overshoot when tuned for this process.

### *4.8.4* P4 – Oscillatory Process

We see from the below figure the step response and the Nyquist plots of the Process P4. The results are nearly similar for the real and the modelled systems.

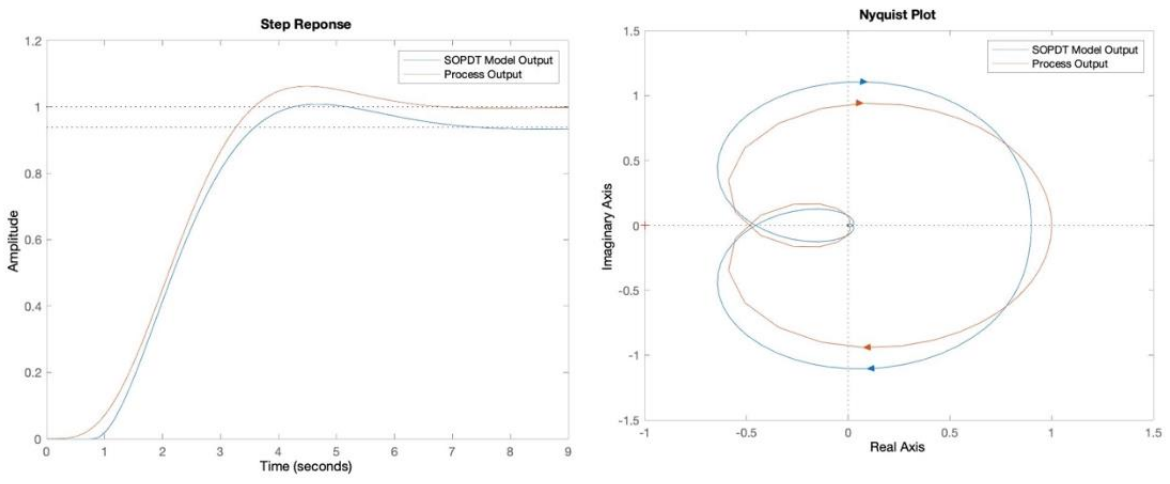


*Figure 38 - Step Response vs Nyquist Plot – Process P4*

The SOPTD model identified for the above process $P_4(s) = \frac{1}{(0.5s^2+s+1)^2}$ can be given by:

$$G_{P3}(s) = 0.9469.\frac{e^{-0.8172s}}{0.8799s^2 + 1.1987s + 1} \tag{59}$$

Where,

$K = 0.9469, T_d = 0.8172, a_2 = 0.8799, a_1 = 1.1987$

Using the above model, we obtain the dynamic parameters as follows:

$$T_0 = 0.9380, \zeta_0 = 0.6389 \tag{60}$$

With this, we obtain the tuning results as per table (8),

| | | Tuning Method | | |
|---|---|---|---|---|
| **No.** | **Parameter** | **DS** | **PMC** | **SIMC** |
| 1 | **Proportional Gain, rp** | 0.98 | 0.81 | - |
| 2 | **Integral Gain, ri** | 0.82 | 0.68 | - |
| 3 | **Derivative Gain, rd** | 0.72 | 0.60 | - |

*Table 8 - PID Tuning Parameters - Process P4*

The results with the tuning can be seen in figure (39).

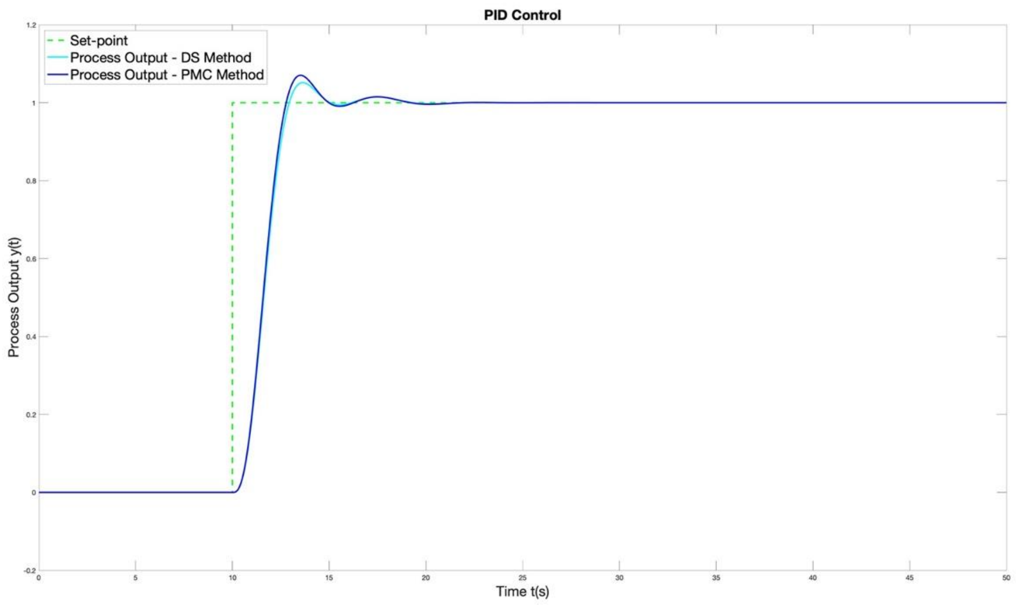


*Figure 39 - PID Control - Process P4*

From the Output results, we see that both tuning methods are unsatisfactory and are not reliable for control. Better tuning methods must be investigated for tuning this process.

### *4.8.5* P5 – Non-oscillatory Process with Time-Delay

Figures (40) show the step response and the Nyquist plots of the system P5, which, as we can see are almost identical.

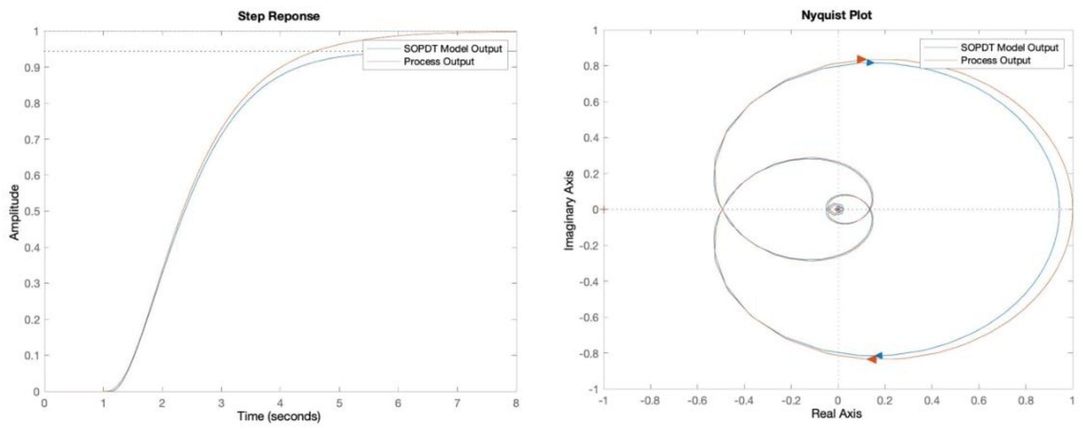


*Figure 40 - Step Response vs Nyquist Plot – Process P5*

On running the program, we identify the SOPTD model of the process $P_5(s) = \frac{1}{(s+1)(0.3s+1)^2}e^{-s}$ as:

$$G_{P5}(s) = \frac{0.9767.e^{-1.1509s}}{0.4658s^2 + 1.4062s + 1} \quad (61)$$

Where,

K = 0.4658, $a_2 = 0.4658$, $a_1 = 1.4062$, $T_d = 1.1509$

Using the above model, we obtain the dynamic parameters as follows:

$$T_0 = 0.7954, \zeta_0 = 0.9167 \quad (62)$$

With this, we obtain the tuning results as per table (9),

| | | **Tuning Method** | | |
|---|---|---|---|---|
| **No.** | **Parameter** | **DS** | **PMC** | **SIMC** |
| 1 | **Proportional Gain, rp** | 0.63 | 0.98 | 0.63 |
| 2 | **Integral Gain, ri** | 0.44 | 0.70 | 0.44 |
| 3 | **Derivative Gain, rd** | 0.21 | 0.33 | 0.21 |

*Table 9 – PID Tuning Parameters - Process P5*

We can see from the above table that the tuning parameters are the same for DS and SIMC tuning methods. The output of the PID controller can be seen in the figure (41).

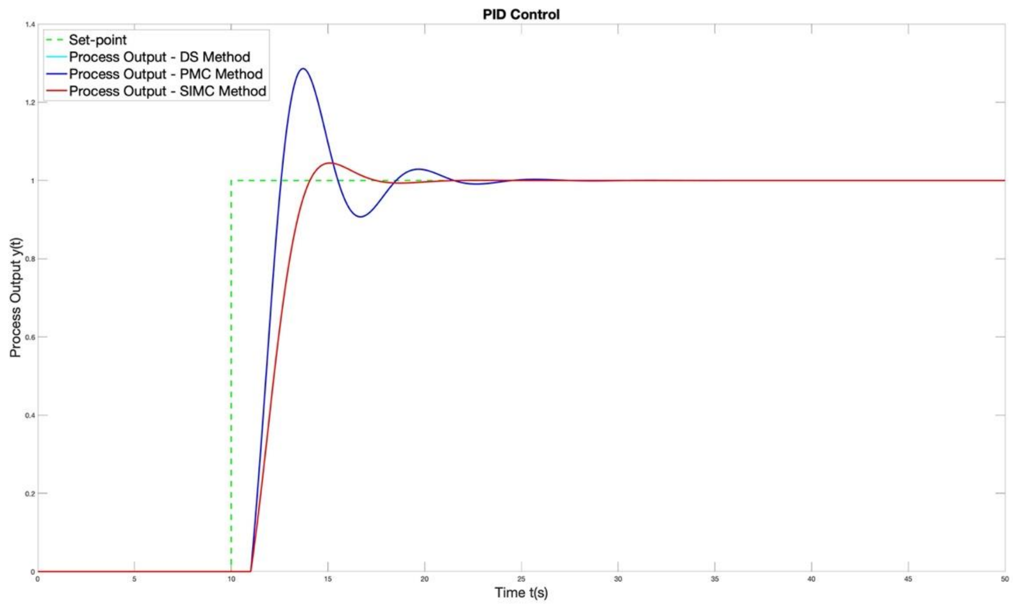


*Figure 41 - PID Control - Process P5*

From the above figure, we see that the DS Method and the SIMC Method provide fairly output characteristics, whereas the PMC tuning method lags behind these two due to its large overshoot.

### *4.8.6* P6 – Fifth-Order Process with Time-Delay

In comparison with the previous systems, the process P6 is by far the hardest one for identification and hence we can see a large standard deviation of $\pm 20\%$.

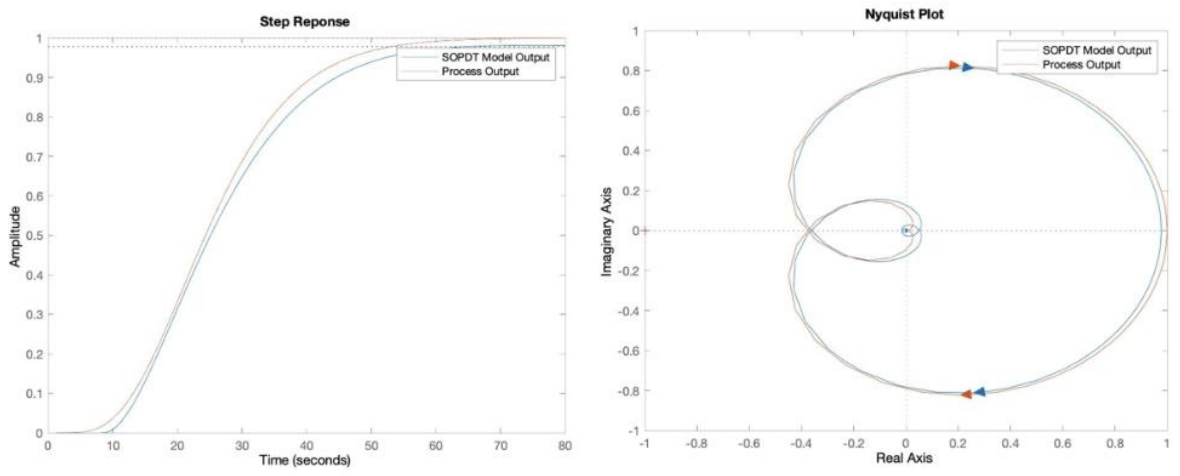


*Figure 42 - Step Response vs Nyquist Plot – Process P6*

Upon identification, we obtain the following SOPTD model of the process $P_6(s) = \frac{e^{-s}}{(5s+1)^5}$:

$$G_{P6}(s) = \frac{0.9773.\,e^{-8.61s}}{107.21s^2 + 18.07s + 1} \tag{63}$$

Where,

$K = 0.9773,\ a_2 = 107.21,\ a_1 = 18.07,\ T_d = 8.61$

We obtain the dynamic parameters as:

$$T_0 = 10.3542,\ \zeta_0 = 0.8726 \tag{64}$$

On tuning the controller using the DS and PMC method, we obtain the following parameters:

| | | **Tuning Method** | | |
|---|---|---|---|---|
| **No.** | **Parameter** | **DS** | **PMC** | **SIMC** |
| 1 | **Proportional Gain, rp** | 1.07 | 1.12 | - |
| 2 | **Integral Gain, ri** | 0.06 | 0.06 | - |
| 3 | **Derivative Gain, rd** | 6.33 | 6.62 | - |

*Table 10 – PID Tuning Parameters - Process P6*

Plugging the above parameters to our PID controller, we obtain the following output:

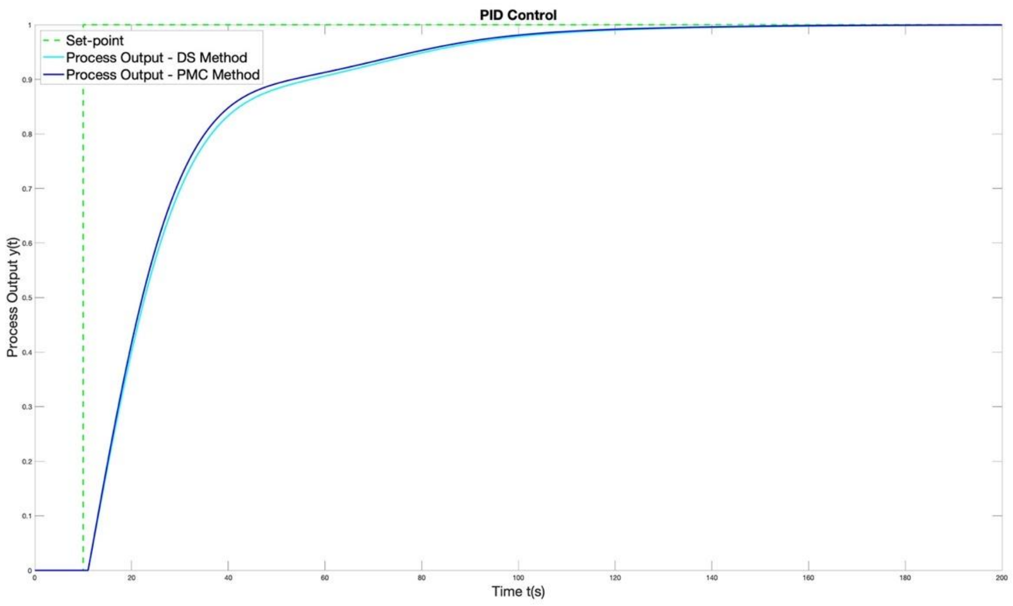


*Figure 43 - PID Control - Process P6*

As we see in the above figure, the PID tuning results in no overshoot, but with quite a long settling time. The Tuning results are, however, acceptable.

## 4.9. Conclusion

### *4.9.1* Identification

In conclusion, all the systems described previously were identified to a sufficiently acceptable degree using the GEA method of optimization. An interesting point regarding the identification program is that, when the original GEA algorithm was initially applied using the method described in section 2.5, the results obtained were highly inaccurate, which was mostly due to the nature of the local search and the crossover operator in the original code. To improve this, the algorithm was slightly modified to apply some of the benefits of BA by using frequency-based search as well as by improving the local search by way of mutating individual parameters rather than all parameters at once.

As per the original GEA Algorithm, the local search is achieved by the following equation:

$$\text{If rand} > \text{p, Make Local Searc} x_i^t = x_*^{t-1} + \epsilon L \qquad (65)$$

The problem with the above equation is that the local search is made if and only if the condition (rand>p) is fulfilled. This means that as the number of iterations increase, the probability of local search is higher. We reach a certain point when the solution reaches a point around the valley, but due to the lower limit condition that the values cannot be less than zero, we face the situation wherein, there is a higher chance of the solution crossing below zero instead of arriving at the optimum value which is around zero and thus ends up getting reset than arriving at the precise minima.

To circumvent this problem, we randomize the dimensions individually such that there is a chance of the individual parameters converging to the optimum instead of all parameters requiring to converge. This can be demonstrated using the below equation.

$$x_{ij}^{t} = \left(x_{*j}^{t-1} + \epsilon L\right).* \left(rand(1,d)\right), \tag{66}$$

Where j = 1:d

Using the above equation, we see that although all parameters do not need to converge together, the individual parameters can, thus leading to a better solution especially when the optimum values are near to zero. This has also be verified experimentally by simulating the above using the equation (34) and alternately using the equation (35).

With the help of this small modification, the time for identification was roughly around 105 seconds on average with some functions converging more quickly than others, which is in fact extremely quick due to the use of the GEA algorithm. Additionally, to improve the convergence rate, we can always decrease the accuracy of the solution. For the purpose of control and especially considering the fact that the above method will be mainly in use for non-linear systems, it is not absolutely required for us to have identified an accurate model. Instead, a reasonable model like the ones identified above will be sufficient for all practical control purposes since we will have to be using a controller anyway. In the next section, we shall explore the control of the above processes.

### *4.9.2* Control

We can conclude from the above results that all the three identification methods work well for controlling non-oscillatory processes with short time delays, like in processes P1, P3 and even P6. However, when it comes to oscillatory processes such as in processes P2 and P4, we still see an overshoot in the process output, which is undesirable in most case and thus, we would need a more appropriate method for this purpose. Process P5 is a bit more complicated when compared to the others, which is because of the delay to time constant ratio, which, thus makes the control using the above methods not very effective. We would need to look into alternate methods of tuning such as AMIGO tuning method to verify its consistency.

It must be noted carefully that the identification and control of the above processes makes use of the process model only and does not relate directly to the exact system. This is the advantage of relay-based identification is that we can control the entire system using a simple process model only instead of knowing the precise dynamics of the system. Furthermore, as we can see, we can use the same methods on a non-linear system as well, which makes our process extremely versatile.

# 5 Physical System

To verify the theory and the algorithms presented in the previous sections, we need to implement the above and experiment with a real system. For the purpose of this thesis, we have looked at the identification behaviour of two systems from the automatic control laboratory. The first system was the water-levitation system where we supply an input signal from the PC to a hydraulic pump which helps to pump water through a nozzle jet on top of which, rests a light-weight ball. The position of the ball is dependent on the flow-rate of water at the nozzle exit, which, in turn, is dependent on the input to the pump. On obtaining the results from the lab, it was found that the identification works correctly to a certain extent, but it is difficult to verify the results in practice since the static conditions of the experiment changes at different points of the experiment, which makes the observations unreliable. Being a non-linear system, this would have been the perfect model for identification, but due to the unreliable nature of this experiment, all results of this first system were discarded and instead, we made use of the second system – Two tanks system.

Section 5.1 describes the physical setup of our system, including the components used in the process while section 5.2 shows the static characteristics of the same system for finding the operating point of our setup. Sections 5.3 and 5.4 describe the problem statement and the Simulink scheme and finally, sections 5.5 and 5.6 explores the results of the identification and PID control.

## 5.1. Experimental Setup

The given system is a combination of two chambers arranged vertically in a tube with a system of interconnecting valves. Figure (44) shows the overall schematic of our two-tanks system. The pump C1 supplies water to the upper chamber. Since the system is hermetically sealed, it develops not only a water head, but an additional pressure head, which causes the accumulated water in the upper chamber to trickle into the lower chamber and eventually back into the tank. A pressure differential sensor is used to map the height of the water in the lower column. Since the upper and lower tanks are connected to each other in series, it is understandable that the system is of second order.

The setup consists of the following basic components:

- Hydraulic Pump ($q_{max} = 10\ lpm$)
- Power Supply (24 VDC, $i_{max} = 1\ A$)

- Pressure Sensor S1
- PC Connected to the pump via MATLAB/Simulink and appropriate hardware
- Qmax (10 l/min)
- Water Chambers – A1 and B1
- Water Tank

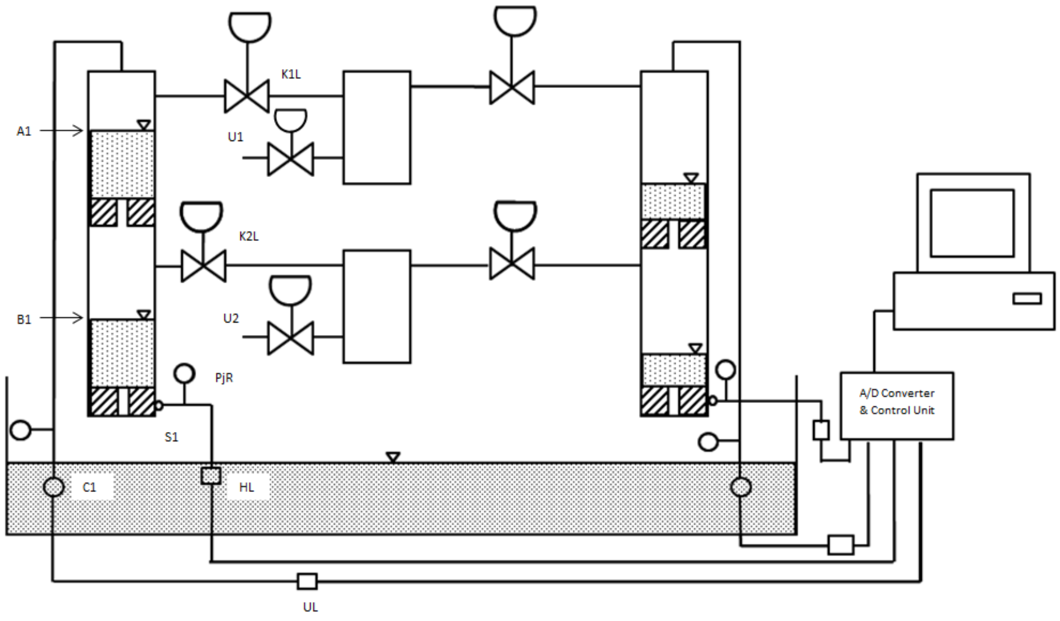


*Figure 44 – Schematic Diagram - Two-tanks system*

We can see a more detailed view of the process from figure (45). We perform the experiment by supplying an input signal to the process via the MATLAB/SIMULINK software from the PC which is transmitted to the pump, which in turn pumps up water to the upper chamber A1. From the bottom hole of the upper chamber, the water trickles down to the lower tank B1, thereby creating a second-order system.

Our goal is to find out the relationship between the height of water column in B1 and the input to the pump. As can be seen from the figure (45), the pump is fed with input 0-10 V from the PC and it thus pumps up water to the chamber A1 depending on the magnitude of the supply voltage. From the chamber A1, the water flows down directly into the chamber B1 whose height we are to measure. We can see that the height of the water column in B1 is dependent on the flow rate of water from the chamber A1, which in turn depends on the height of water in the chamber A1. The final height in B1 is measured by means of the pressure sensor S1. One of the benefits of using a pressure sensor for measurement instead of a level sensor is that, due to prolonged usage, the tanks end up with a lot of air bubbles within, which may result in measurement errors if we do use a level sensor.

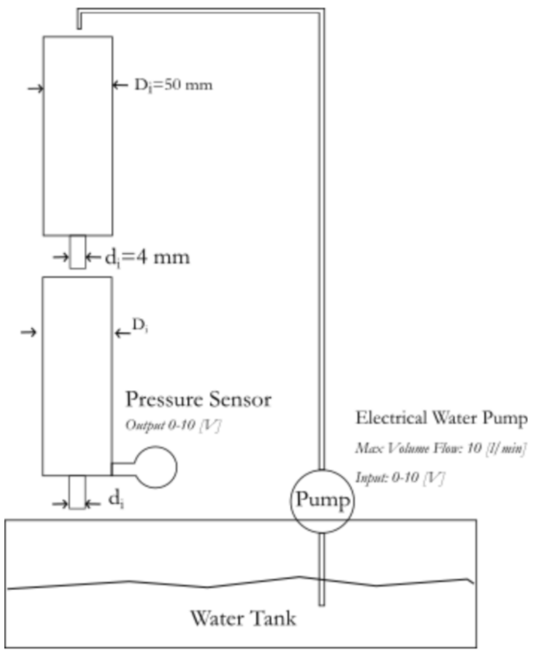


*Figure 45 – Functional Diagram - Two Tanks System*

For experimentation, we assume that the pressure sensor behaves in a linear manner, meaning that the sensor output behaves linearly with respect to the sensed pressure.

## 5.2. Static Characteristics

Our goal with identification is to verify the output for different values of the input signal and derive a dynamic model of the same. Firstly, to start with, we need to plot the static characteristics of the system to ensure that our inputs and outputs are within the linear range of the system. To obtain the static characteristics, we would need to plot the output height (y) for different constant values of input (u). We thus obtain the figure 46.

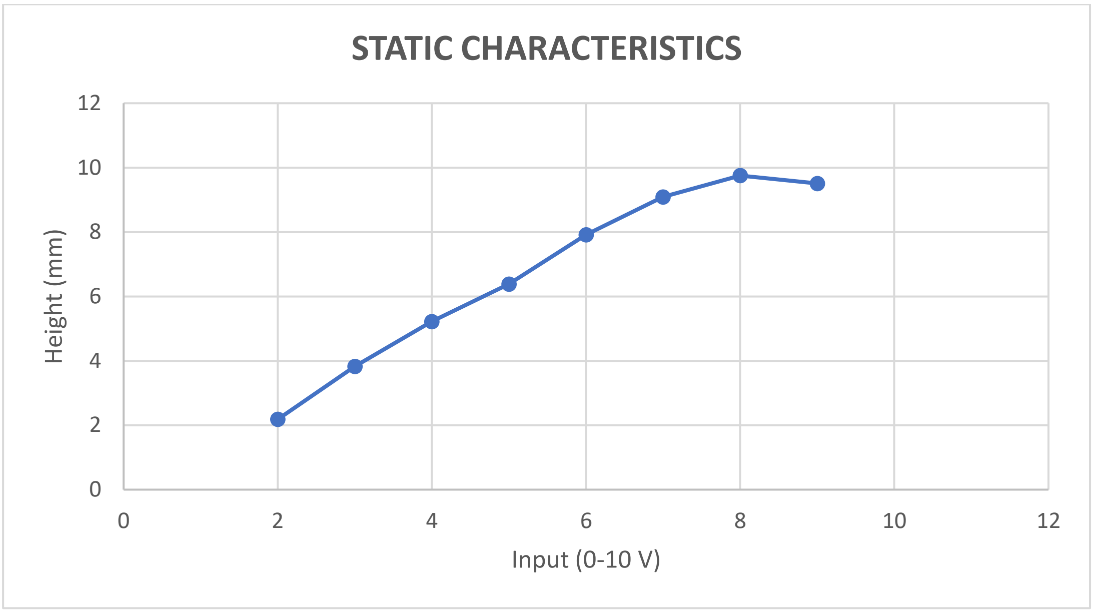


*Figure 46 - Static Characteristics - Two tank system*

We see from the above Static Characteristics that the system is inherently linear in the range 2 – 8 V. Therefore, we shall set our operating point at the point (4, 5.22).

## 5.3. Problem Statement

Similar to section 4.4, our problem here is to identify or estimate a dynamic model of the 'Two-tanks System' from figure (44), (45). As before, we shall use the SOPTD dynamic model from equation (23) and estimate the parameters $\{K, a_2, a_1, T_d\}$ using the GEA optimization algorithm.

$$x_j = \{a_1, a_2, K, T_d\} \tag{67}$$

Where:

$j =$ Dimension index (1:d)

$d =$ Number of dimensions (= 4 in this case)

The cost function for optimization of the above parameters can be summarized as:

$$J = \int_0^{T_k} \left(y_m(t) - y(t)\right)^2 dt \tag{68}$$

$y_m =$ model output

$y_t =$ system output

$T_k =$ simulation time

Therefore, we can construct our problem statement as follows:

1. Find the parameters $\boldsymbol{x_i}$ (i = 1,2 … d) to minimize the cost function J,

Subject to constraints:

$\{\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{K}, \boldsymbol{T_d}\} > \boldsymbol{0}$

2. Compare the results of the SOPTD model with those of the real system in terms of the step response and Nyquist plots.

## 5.4. Experiment

The experimental setup consists of the following components in the Simulink system:

1. Relay: A relay controller is used to create oscillations in the system by supplying non-linear input to the process.
2. Constant Input: The input source is constant and is set at the working point, which we have considered as at input = 4 V.
3. HPS Block: This block contains the Internal COM logic to convert the internal input signal (uL) from the relay to the physical system, then obtain the readings from the pressure sensor (yL) and then feed it back into the input sum block.

The parameters of the real System are set as follows:

- $u_A = 7$
- $u_B = 2$
- $\epsilon_A = eps$
- $\epsilon_B = -eps$
- $w = 5.22$ (at operating point 4)
- Simulation Time, Tsim = inf

Note that, unlike the previous Simulink scheme, the simulation for the physical system will consists of two parts:

A) Physical Experiment: The program is started as per the figure (47) with the input ($uL$) and relay parameters as above. The input to the system and the output is logged to the workspace for further processing.

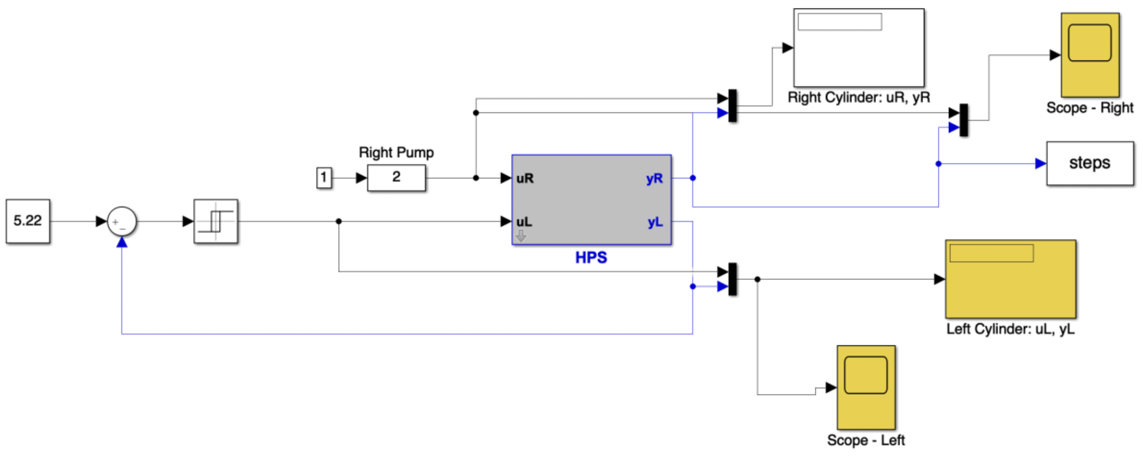


*Figure 47 – Simulink scheme of Physical System*

B) Identification Scheme: Once we obtain the results from the real system, we then feed the input $uL$ to our SOPTD model and measure the model output $y_m$. The error between the model output and the real system output is squared and integrated over time from which we obtain the ITAE of the system which we are to minimize as part of our cost function. Figure (48) demonstrates the schematic of our identification problem.

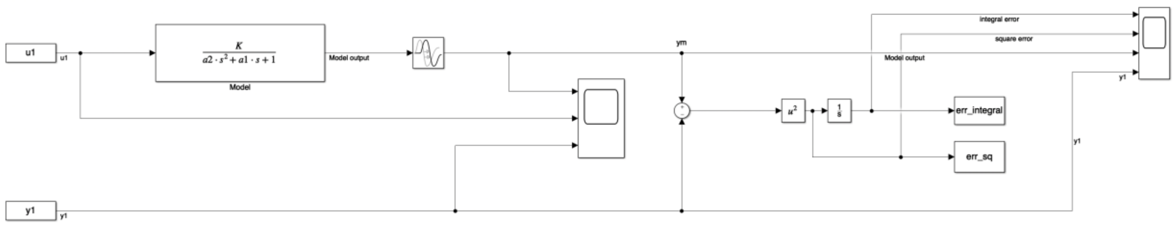


*Figure 48 - Simulink Scheme – Identification*

## 5.5. Results

Using the above methods, the above experiment was conducted on the two tanks system. The input was fed into the system using the parameters described in section 5.4, from which, we obtained the input / output matrices as u1 and y1 respectively. This gives us the experimental results from the setup.

To verify our identification method, we need to use the input u1 and feed this to our SOPTD model to obtain the process output of our model ($y_m$) as seen in figure (49). Similar to what we did in the section 4.4, we consider our cost function $J$ as the difference between the model output $y_m$ and our process output $y1$. The optimization algorithm is then run using the MATLAB script and with each iteration, the Simulink model is run within the script to calculate the cost function $J$ or in other words, the fitness of each solution. The MATLAB algorithm tries to minimize this cost function by finding optimum values of $\{K, a_2, a_1, T_d\}$ to finally arrive at the best solution.

The results from multiple simulations can be summarized in the below table 5.

| No | Op. Pt | Input, w | No of Individuals | Sim_time | No or Iterations | K | a2 | a1 | Td | Error | Elapsed time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 3.83 | 40 | 300 | 20 | 1.14 | 273.31 | 31.23 | 0 | 18.70 | 135.91 |
| 2 | 3 | 3.83 | 40 | 300 | 20 | 1.13 | 277.67 | 40.03 | 0 | 18.50 | 141.60 |
| 3 | 3 | 3.83 | 40 | 300 | 20 | 1.13 | 287.31 | 40.03 | 0 | 18.59 | 135.46 |
| 4 | 3 | 3.83 | 40 | 300 | 20 | 1.13 | 265.57 | 40.11 | 0 | 18.66 | 129.24 |
| 5 | 3 | 3.83 | 40 | 300 | 20 | 1.13 | 276.36 | 40.10 | 0 | 18.51 | 133.18 |
| 6 | 3 | 3.83 | 40 | 300 | 20 | 1.13 | 301.36 | 38.77 | 0 | 18.87 | 141.61 |
| 7 | 4 | 5.22 | 40 | 300 | 20 | 1.10 | 376.28 | 49.61 | 0 | 19.21 | 76.58 |
| 8 | 4 | 5.22 | 40 | 300 | 20 | 1.12 | 281.51 | 41.07 | 0 | 19.80 | 103.41 |
| 9 | 4 | 5.22 | 40 | 300 | 20 | 1.06 | 439.81 | 42.48 | 0 | 17.48 | 69.32 |
| 10 | 4 | 5.22 | 40 | 300 | 20 | 1.10 | 376.03 | 47.53 | 0 | 17.16 | 95.81 |
| 11 | 4 | 5.22 | 40 | 300 | 20 | 1.09 | 387.61 | 49.66 | 0.0803 | 19.74 | 98.62 |
| 12 | 4 | 5.22 | 40 | 300 | 20 | 1.09 | 290.22 | 46.74 | 0 | 17.65 | 203.20 |
| | **Mean:** | | | | | 1.11 | 322.34 | 42.28 | 0 | 18.57 | 122.00 |
| | **Standard Deviation:** | | | | | 0.02 | 60.60 | 5.32 | 0 | 0.82 | 36.09 |

*Table 11 - Identification Results*

As can be seen from results in the above table, the the experiment was carried out at two different operating points of the system, i.e. at (3, 3.83) and at (4, 5.22). We can easily see that the results obtained using the second operating point converge more quickly as compared to the first and from the obtained step responses, we also see that the results of the step response are more accurate in the first case as compared to the second. The reason for this is because the asymmetricity of the relay in the first case is much larger than in the second which generates unbalanced oscillations resulting in less accurate results.

To verify the correctness of the above identification, we shall compare the step responses of the real system with the modelled system and examine the results visually. The ITAE error from the above table is already a good indicator of the correctness, but nevertheless, it is important to use a secondary method so as to validate our results. We shall thus use the step response and the Nyquist diagram to verify our results.

1. Step Response:

Using the simulation results for $\{K, a_2, a_1, T_d\}$ as = $\{1.10, 376.28, 49.62, 0\}$ and using the step input of 3.5, we obtain the following figure (50).

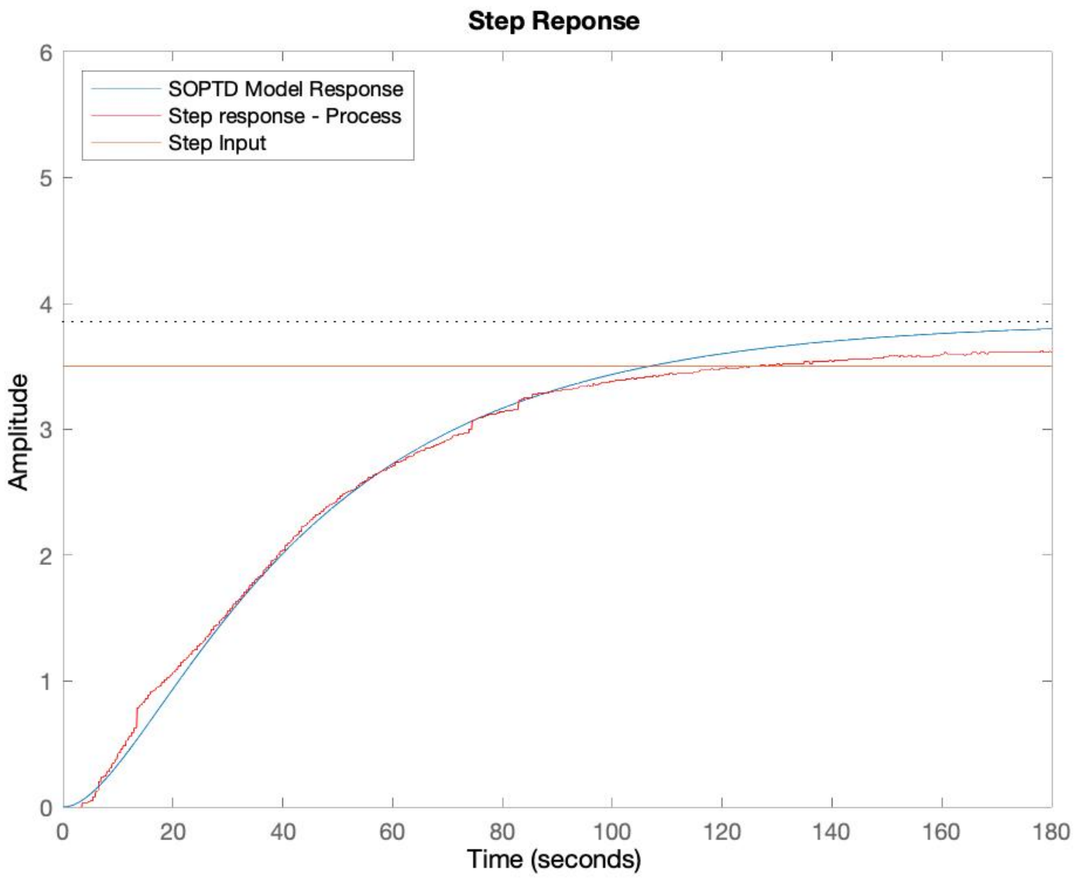


*Figure 49 - Step Response*

We see from the above figure (49) that the output from our real process (in red) closely matches the output from our SOPTD model. We can therefore say that our identification procedure is fairly accurate.

2. Nyquist Plot:

While the step response gives us a fair idea about the accuracy of our estimation, it does not tell us anything about the response of the system based on the frequency. For this, we need to use a bode diagram or a Nyquist plot to verify the same. In our case, we will use the Nyquist plot to see whether or not our system holds true for frequencies other than zero. In essence, the Nyquist plot is the magnitude-phase plot of the system for different frequencies of the input.

First, we shall apply a sine input to our experimental setup using frequencies 0.05 rad/s, 0.1 rad/s and 0.2 rad/s with which, we shall obtain three different points on the magnitude-phase plot. The relations for plotting the points on the curve can be found from the following:

Amplitude Relation:

$$|G(j\omega)| = \frac{y_A}{u_A} \tag{69}$$

Where $y_A$ and $u_A$ can be found from the scope output y and u as from the figure (20).

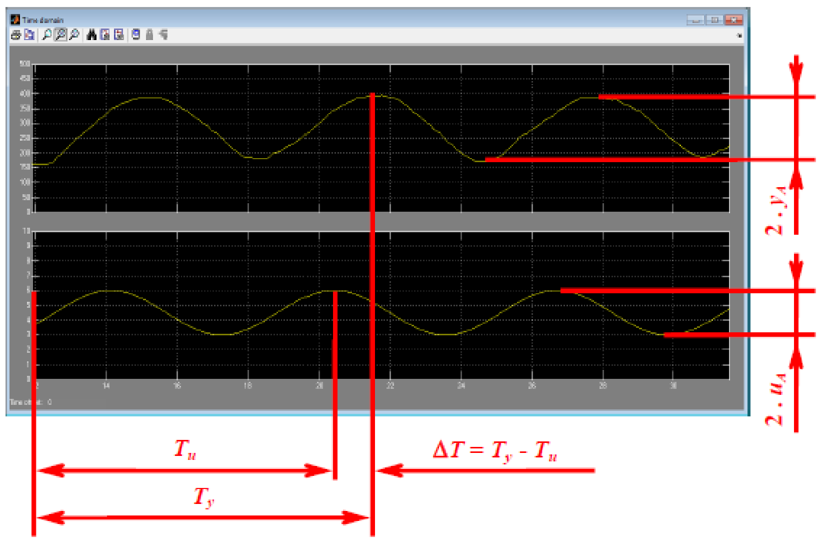


*Figure 50 - Frequency Response Characteristics - Time Plot*

Phase-Shift Relation:

$$\Delta T = T_y - T_u \tag{70}$$

$$\phi = -\frac{\Delta \mathrm{T}}{\mathrm{T}} 360°$$

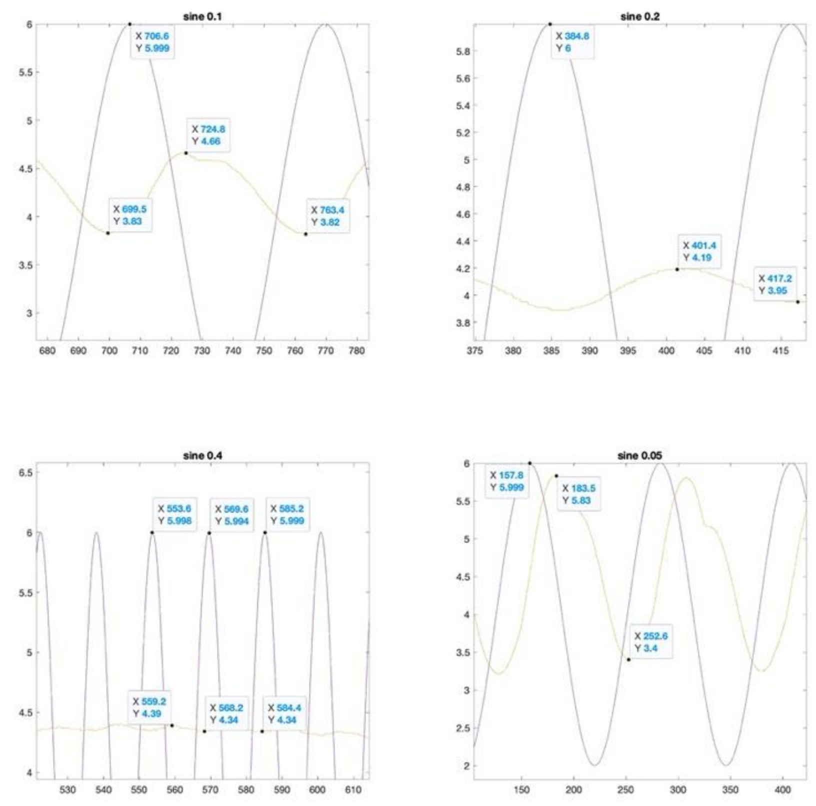


*Figure 51 - Frequency Response - Time plot from Laboratory*

Figure (51) shows the time-plot of the output $y_a$ vs the input $u_a$ which we obtained directly from the experiment. Using the above relations – equations (40) and (41), we can plot the calculated points on the Nyquist plot with x and y co-ordinates as:

$$x_{cood} = |G(j\omega)|.\cos\phi \text{ and } y_{cood} = |G(j\omega)|\sin\phi \tag{71}$$

Thus, we get the three points (-0.0518, -0.2035), (-0.0590, -0.0106) and (-0.0078, -0.0098).

With the above, relations, we have our three points for the three readings on the Nyquist plot. We need to verify if these three points lie on the Nyquist plot of the modelled transfer function G(s). We thus obtain the following figure (52) as follows:

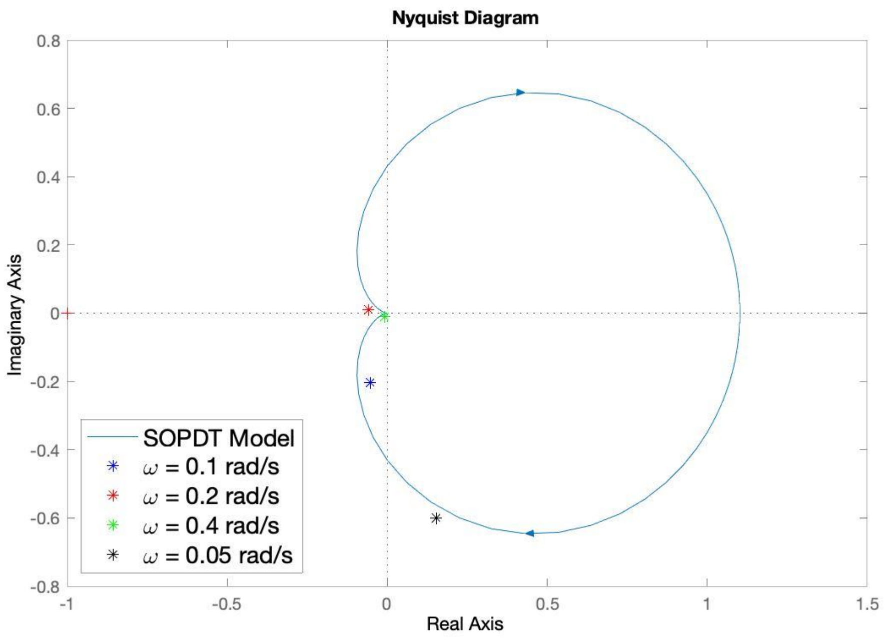


*Figure 52 - Nyquist Plot – Two Tanks System*

As can be seen from the figure, the points almost complete coincide with the Nyquist curve from the SOPTD model estimated from the above algorithm. Hence we can conclude that the identified system is an acceptable model of the real process.

## 5.6. PID Control

Now that we have identified our model correctly, we can proceed to get into the 'real' deal, which is controlling the above process. For this we need to tune our PID method using the methods described in the above sections (4.7.1), (4.7.2) and (4.7.3) evaluate the results for the same.

First, let us consider one of the results obtained from the Identification process as:

$$G_P(s) = \frac{1.09}{290.22s^2 + 46.74s + 1} \tag{72}$$

Where,

$K = 1.09,\ a_2 = 290.22,\ a_1 = 46.74$

Using the relations for DS Method and PMC Method of tuning, we obtain the following:

| | Tuning Method | | | |
|---|---|---|---|---|
| No. | Parameter | DS | PMC | SIMC |
| 1 | Proportional Gain, rp | 3.36 | 3.36 | - |
| 2 | Integral Gain, ri | 0.07 | 0.07 | - |
| 3 | Derivative Gain, rd | 25.48 | 25.48 | - |

*Table 12 - Tuning Parameters - Two Tank System*

We can summarize the results of the above tuning in the below figure (53).

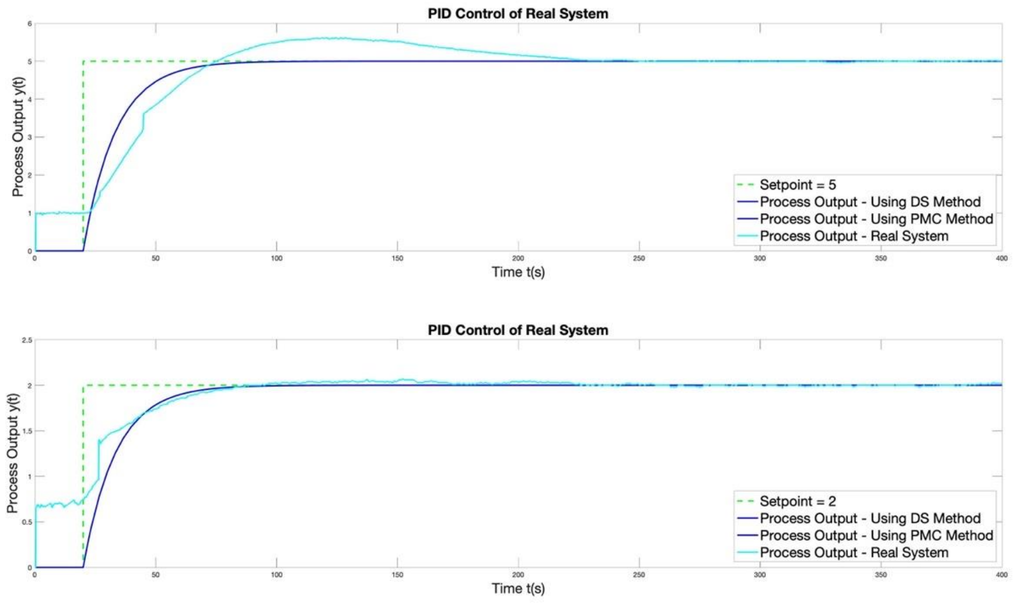


*Figure 53 - PID Control - Two Tanks System, SP = 5 and SP = 2*

We see from the above two results that the response of the real system is quite different from the results we obtained on the real model. The reason for this can be attributed to the fact that when we tune our controller in Simulink, we consider values of the Proportional, Integral and Derivative Gain which may or may not be achievable in real. However, the physical system limits these values causing saturation of the pump and due to non-linearities within the system. While the process output is far lower than the real output, the error keeps building up, thus increasing the reset wind-up of the integral part. Thus, as the setpoint increases, the overshoot of the Process output increases consequently. To reduce the overshoot, we would need to add additional compensation or use a different tuning method for minimum overshoot of the controller.

# 6 Conclusion

With the above experiment, we have fulfilled the three main parts of this thesis work namely:

1. Optimization
2. Identification
3. Control

In the Optimization part, we have seen how effective and efficient metaheuristic algorithms can be in peak finding. While all the three methods, PSO, BA and GEA are extremely efficient, the GEA tuning procedure is relatively simple to execute for a wide range of applications as the tuning procedure is not as hard as compared to PSO and BA. Furthermore, one of the advantages of GEA is that it also includes the mutation operator which, as we have seen is a key factor in arriving at the optimum value. Other optimization techniques which can be looked into are the SOMA optimization and TABU search (Glover, 1986). Both these techniques cannot accurately be called as 'Metaheuristic Algorithms' per se, nevertheless, given the alternate approach at peak finding, it would be interesting to research into these as well, especially the TABU search approach which is not probabilistic based but rather, it uses the concept of a TABU list thus, making sure that incorrect predictions are not repeated unlike the heuristic approach.

In the second part, we have developed a deeper insight into alternate identification procedures using Relay-Based Identification. Due to the nature of the optimization algorithms which we are using the convergence rate is extremely quick with almost all of the solutions converging in less than five minutes which is highly efficient for large systems. Furthermore, from the simulation of pre-defined functions, we saw that the algorithm was able to predict most models fairly accurately using which we were able to achieve control our processes. While this procedure is relatively simple compared to some of the other available system identification approaches, it is nonetheless, extremely efficient. The other major advantage is that the same technique for identification can also be used for modelling non-linear systems and this makes the method promising for further research. One of the drawbacks of this algorithm method is the fact that it uses a lot of memory and computing power for searching. With the help of MATLAB and Simulink software, we are able to achieve this in relatively less time. However, the above programs cannot be directly on plant sites using PLCs without the appropriate SCADA systems for data acquisition, processing and manipulation.

With optimization and identification, we moved to PID control of the identified processes. For all of the PID control algorithms used in this thesis, we made use of the identified SOPTD process model only to derive the PID tuning parameters. This means that, once the system has been identified correctly, we can then tune the controller purely by means of the process model which we have identified. This is extremely advantageous and efficient for control in practical plants or processes. The PID tuning methods which we have looked at in the previous sections are not wholly efficient for the processes which we have selected in this thesis. Nevertheless, they do a good job with non-oscillatory processes without time-delay. The PID tuning can be improved to achieve better control for which, we can look into more efficient methods such as *Pole Placement Method* or *PMC Tuning method* for oscillatory systems which makes use of the gain margin approach.

All in all, we have explored a novel approach to System Identification and Control and although the final control was not as successful as we hoped it to be, the identification approach is acceptable and functional. The concept can be further expanded in the field of ***Model Predictive Control*** wherein, the control characteristics can be greatly improved by using more efficient algorithms for control such as the ones we looked into here. We can also expand the concept of identification to include much larger and more complicated processes since the underlying concept of optimization still holds true regardless.

# 7 Bibliography

Cao, L., Xu, L., & Goodman, E. (2016). A Guiding Evolutionary Algorithm with Greedy Strategy for Global Optimization Problems. *Computational Intelligence and Neuroscience*, 1-10.

Yang, X.-S. (2014). *Nature-Inspired Optimization Algorithms.* Elsevier.

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks* (pp. 1942-1948). Perth: IEEE.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, Michigan, USA: University of Michigan Press.

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, vol. 1, no. 1*, 67-82.

Parkinson, A., Balling, R., & Hedengren, J. (2018). *Optimization Methods for Engineering Design, Second Edition.* Provo, Utah: Brigham Young University.

Yang, X.-S. (2010). A New Metaheuristic Bat-Inspired Algorithm. In J. G. Eds, *Nature Inspired Cooperative Strategies for Optimization* (pp. 65-74). Berlin: Springer.

Stoev, J., & Schoukens, J. (2016). Nonlinear system identification - Application for industrial hydro-static drive-line. *Control Engineering Practice, Volume 54*, 1-2.

Ljung, L. (1987). *System Identfication: Theory for the User.* Linkoping: Prentice Hall.

Seborg, D. E., Edgar, T. F., Mellichamp, D. A., & III, F. J. (2017). *Process Dynamics and Control.* Dellaware: Wiley.

Hofreiter, M. (2018). Alternative Identification Method using Biased Relay Feedback. *IFAC Papers Online* (pp. 891-896). Prague: Elsevier.

Berner, J., Hägglund, T., & Åström, K. (2016). Asymmetric Relay Autotuning - Practical features for industrial use. *Control Engineering Practice, Volume 54*, 231-245.

Ramakrishnan, V., & Chidambaram, M. (2003). Estimation of a SOPTD transfer function model using a single asymmetrical relay feedback test. *Computers & Chemical Engineering, Volume 27, Issue 12*, 1779-1784.

Chidambaram, M., & Sathe, V. (2014). *Relay Autotuning for Identification andControl.* Cambridge: Cambridge University Press.

Astrom, K., & Hagglund, T. (1984). Automatic Tuning f Simple Regulators with Specifications on Phase and Amplitude Margins. *Automatica, Volume 20, Issue 5*, 645 - 651.

*PID Controller*. (2020, 08 09). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/PID_controller

Hofreiter, M. (2016). *Zaklady Automatickeho Rizeni.* Prague: CVUT.

Chen, D., & Seborg, D. (2002). PI/PID Controller Design Based on Direct Synthesis and Disturbance Rejection. *Industrial & Engineering Chemistry Research*, 4807-4822.

Skogestad, S. (2004). *Modelling, Identification and Control.*

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 533-549.

# Appendices

The accompanying materials are made available in a CD under the below folders:

**/MATLAB:** This folder contains the MATLAB/SIMULINK files and the related sub-folders for execution

**/Identification:** This folder consists of the accompanying results of identification

**/Control:** This folder contains the results of the PID Control

**/Two Tanks:** This folder contains the results of the physical system

**Adrian_master_thesis.pdf:** Electronic copy of this Diploma Thesis