CHALMERS
UNIVERSITY OF TECHNOLOGY

# Development of optimization methods for Volvo fuel economy simulation.

Master's thesis in

Signal and Systems

RAFAEL KLÜPPEL SMIJTINK

Development of optimization methods for Volvo fuel economy simulation.

Master thesis done in partnership with Volvo AB.

RAFAEL KLÜPPEL SMIJTINK

Supervisor: Thierry Bondat, Volvo AB

Examiner: Jonas Fredriksson, Department of Signals and Systems.

Technical report no EX008/2017
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Göteborg Sweden
Telephone + 46 (0) 31-772 1000

Cover: On the bottom, the cover shows two example vehicles produced by Volvo AB, a bus and a truck. On the top, the representation of fuel consumption and performance, which are the two main features that could be optimized with the tool created in this report.

Göteborg, Sweden 2017

CHALMERS
UNIVERSITY OF TECHNOLOGY

## Abstract

*Optimization methods are being used progressively in the automotive industry. New legislation on emissions have pushed the development of more efficient vehicles, thus vehicles become more complex, the task of finding the optimal design only by engineering expertise becomes almost impossible, or at least deemed to result in sub-optimal designs. To deal with the increased complexity of the task, this thesis aims to develop a tool to perform optimization to solve several problems, e.g. model calibration, vehicle feature optimization. The tool developed is able to perform mixed-integer and multi-objective optimization, and handle various types of constraints, as many of the problems require these abilities. The tool has been tested on benchmark functions and on test cases provided by Volvo AB. On the first test case, it has managed to perform a multi-objective problem with both discrete and continuous variables and successfully reduced the calibration error of a Gipps traffic model. On the second test case, it optimizes a gearshift map for an automated gearbox. The aim was to reduce fuel consumption and improve the performance. By the end, it was obtained a 1.4% improvement in fuel consumption with the same performance as the reference calibration.*

**Keywords: Optimization, Multi-objective, Mixed-integer, Calibration, ZF gearbox, Gipps traffic model.**

# Nomenclature

## Abbreviations

| ***Variable*** | ***Description*** |
|---|---|
| GSP | *Global Simulation Platform* |
| DOE | *Design of Experiments* |
| SQP | *Sequential Quadratic Programming* |
| MADS | *Mesh Adaptive Direct Search* |
| NSGA II | *Second version of the Non-dominating Sorting Genetic Algorithm.* |
| ZDT | *Zitzler-Deb-Thiele's multi-objective benchmark function* |
| RMS | *Root Mean Square error* |
| RSM | *Response Surface Method* |
| ZF | *Gearbox manufacturer* |

## Symbols

| ***Variable*** | ***Description*** | ***Units*** |
|---|---|---|
| $f$ | *Function to be minimized* | -- |
| $g$ | *Inequality constraint function* | -- |
| $h$ | *Equality constraint function* | -- |
| $M_{eq}$ | *Number of equality constraints* | -- |
| $M_{ineq}$ | *Number of inequality constraints* | -- |
| $x$ | *Vector of design variables* | -- |
| $v$ | *Vector with mutated variables* | -- |
| $u$ | *Vector with the design variable of the child* | -- |
| $F$ | *Mutation factor* | -- |
| $CR$ | *Crossover probability* | -- |

| | | |
|---|---|---|
| *randn* | *Random number generated by normal distribution* | -- |
| *rand* | *Random number generated by uniform distribution* | -- |
| $S_{CR}$ | *Vector of successful crossover probability values* | -- |
| $S_F$ | *Vector of successful mutation factor values* | -- |
| *c* | *Adaptation control parameter of JADE* | -- |
| *p* | *Percentile of best individuals parameter of JADE* | -- |
| $\sigma$ | *Tolerance* | -- |
| *N* | *Number of allowed values for discrete variable* | -- |
| *d* | *Euclidean distance* | -- |
| *NP* | *Number of individuals in population* | -- |
| $\tau$ | *Response time* | *s* |
| $S_v$ | *Safety margin* | *m* |
| $b_l$ | *Deceleration of the leader* | $m/s^2$ |
| $b_f$ | *Deceleration of the follower* | $m/s^2$ |

## Subscripts

| | |
|---|---|
| *m* | *Mean values* |
| *i* | *Individual index* |
| *j* | *Variable index* |
| *G* | *Generation index* |
| *Best* | *Best values in population* |
| *R* | *Random picked individual* |
| *Max* | *Maximum value* |

## Superscripts

| | |
|---|---|
| *P* | *p% individuals of population* |

## Contents

# 1. Introduction

Automotive industries have progressively increased the usage of CAE (Computer Aided Engineering) methods to improve lead-time and reducing costs in product development projects. Simulation is one of these tools that have the greatest potential in contributing with early development analysis, prototype expenses and improving overall project results. Instead of building prototypes on each development loop, it is possible to access many vehicle features by simulation before a defined concept even exists. By doing early concept analysis, errors that would be found only during testing phase, can be identified and corrected.

GSP (Global Simulation Platform) is a tool developed at Volvo AB. It is based on Simulink models that simulate the complete vehicle, the interactions of its subsystems and its external interactions with the driver and the environment (road). The models simulate the physical and control behaviour of components through 0D/1D dynamic models [1]. This tool is essentially used to evaluate vehicle features such as fuel consumption and performance and to generate useful insight of the vehicle behaviour.

A single truck can be adapted to a wide range of applications by modifying engine and gearbox control calibrations, tyres, final gear ratio and many other parameters. To find the configuration that meets all requirements and that has the lowest fuel consumption, one way is by simulating all possible combinations of parameters, tough it can be impossible given the number of possibilities to be tested. This approach is often referred to as the brute force method.

For many years, engineers used a variation of the brute force method that relies on their expertise of knowing, from all the possible configurations, which were the most promising settings and testing only those, drastically reducing the number of simulations to be made. This approach sometimes does not consider configurations that could be even better than the ones believed feasible by the engineer.

New legislation on emissions have pushed the automotive industries to develop more efficient vehicles, thus vehicles become more complex, the task of finding the optimal design only by engineering expertise becomes almost impossible, or at least deemed to result in suboptimal designs. To deal with the increased complexity of the task, optimization methods have been applied to solve some of the problems that are common in automotive industry.

## 1.1. Purpose

Optimization can be used to solve several problems in automotive industries. Problems like optimal component sizing, controller calibration, optimization of vehicle features, calibration of models, simply anything that is not easily achieved by trial and error methods or is considered too complex, could be solved by optimization techniques.

The purpose of this thesis was to develop a platform that applies optimization algorithms to solve many of the problems highlighted earlier. Currently at Volvo AB, there are several optimization methods being used in different departments, but they are developed for specific needs and not reused. To avoid that, one of the purposes of building an optimization tool was to have common methods that could be reused for different problems in all the company.

There are multiple tools and software that perform optimization in the market, but to use them in the specific problems at Volvo AB would require a long time to set up. The interface between the applications and the optimization tool would take too long to be developed. With an in-house developed tool, the algorithms available can be tuned for the target tasks, giving faster and better results with increased compatibility.

Section 1.1.1 and 1.1.2 explain two situations where optimization could be used at Volvo.

### 1.1.1. Vehicle Feature Optimization

The optimization of a vehicle using simulation models can be seen as a multi-level optimization problem [2]. Figure 1, shows the three different possible levels of optimization of a vehicle. The first is the topology, which represent the number of components and the order that they are connected in the driveline. The second is the sizing of the components and technology optimization, known also as outer loop or offline optimization. Finally, the third is the optimal control of each component, known as inner loop or online optimization.

If the topology is considered constant and the goal is to optimize the sizing and control of components, this constitutes a bi-level optimization and it is inherently a mixed-integer multi-objective problem [3]. This is mainly because there is a finite number of components to be chosen from, and then the choice is over a discrete finite domain. The control and vehicle parameters can be either discrete or continuous; a problem, which can have both types, is called mixed-integer. It is multi-objective because even fuel consumption being the main objective of an optimization in the automotive industry, this cannot be done without taking into account the performance and driveability aspects, the optimization should try to minimize fuel consumption and maximize the performance of the vehicle. Other objectives could be taken into account, like the minimization of the number of gearshifts per kilometre.

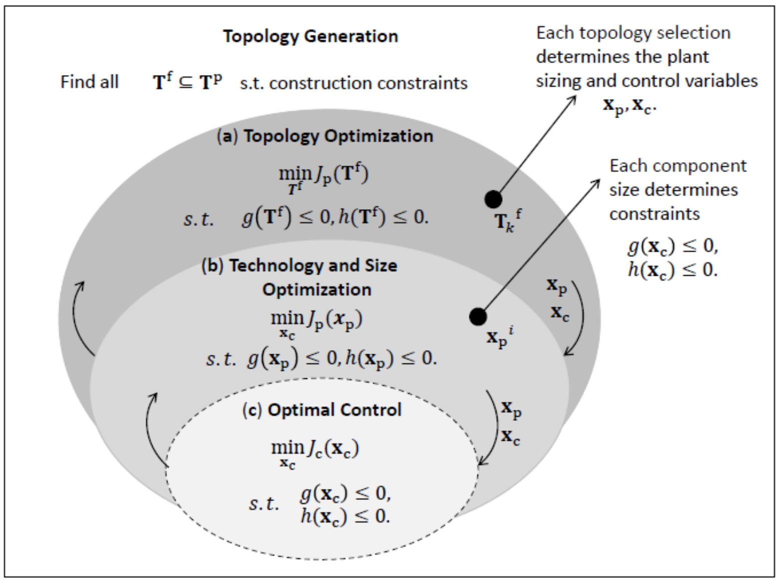

Topology Generation
Find all $\mathbf{T}^f \subseteq \mathbf{T}^p$ s.t. construction constraints
Each topology selection determines the plant sizing and control variables $\mathbf{x}_p, \mathbf{x}_c$.
(a) Topology Optimization
$\min_{T^f} J_p(\mathbf{T}^f)$
$s.t.\ \ g(\mathbf{T}^f) \leq 0, h(\mathbf{T}^f) \leq 0.$
$\mathbf{T}_k{}^f$
Each component size determines constraints
$g(\mathbf{x}_c) \leq 0,$
$h(\mathbf{x}_c) \leq 0.$
(b) Technology and Size Optimization
$\min_{\mathbf{x}_c} J_p(x_p)$
$s.t.\ \ g(\mathbf{x}_p) \leq 0, h(\mathbf{x}_p) \leq 0.$
$\mathbf{x}_p{}^i$
$\mathbf{x}_p$
$\mathbf{x}_c$
(c) Optimal Control
$\min_{\mathbf{x}_c} J_c(\mathbf{x}_c)$
$s.t.\ \ g(\mathbf{x}_c) \leq 0,$
$h(\mathbf{x}_c) \leq 0.$
$\mathbf{x}_p$
$\mathbf{x}_c$

*Figure 1- Different optimization levels [2].*

One important aspect that must be taken into account, if one have limited computational power, is to prefer the usage of light models when performing complex optimization. If this is not possible, the usage of algorithms that need the lowest number of simulations per step, not to occur in a long optimization process should be prioritized. The GSP models, used in this thesis, are considered complex and time consuming, with an average running time of 2 minutes.

### 1.1.2. Model calibration by optimization

Optimization is a very general method that could be used to solve infinity of problems, and then it is natural that this technique could be used in model calibration problems. These problems often have high dimensionality and are hard to tune given the wide range of values that the calibration variables can assume. Often calibration can be a tedious trial and error process, and in this case, through optimization it is possible to find good solutions with less effort while searching the domain in a smart way.

The objective in this case is to find the parameters that minimize the error between the outputs of the model and a measured or reference output that is being modelled.

## 1.2. Objective

The following objectives should be met by the end of the thesis:

- This master thesis aims to develop an optimization platform to work with GSP and other applications such as Matlab scripts and Simulink models. The platform must apply different optimization methods to solve usual problems found at Volvo AB. The most common problems are optimal component sizing, controller calibration, optimization of vehicle features, and calibration of models.
- The resultant tool should be able to perform mixed-integer multi-objective optimization, since many of the problems at Volvo AB require this ability.
- The tool must be easy to use and to be easily customizable by the user, so it is possible to set up complex problems without difficulty.
- The platform should be tested on real situations that reflect the main possible usages of the tool at Volvo AB.

## 1.3. Delimitations

Only a few optimization algorithms were tested, since the main idea is not to perform a full benchmark between them. The focus was on the development of the tool and on the handling of the various types of problems rather than finding the best algorithms for each problem.

Only offline (outer loop) optimization was be contemplated by the tool. The module could be extended to perform online optimization also, but given time constraints, this was not done.

# 2. Optimization

This section gives a background about optimization for the better understanding of the thesis.

## 2.1. Single Objective Optimization

The objective of an optimization method is to solve the following system of equations [4]:

$$\begin{array}{lll} \text{Minimize} & f(x) & (1) \\ \text{Subject to} & g_j(x) \leq 0, \quad j=1, 2, \ldots, \text{Mineq}, & \\ & h_i(x) = 0, \quad i=1, 2, \ldots, \text{Meq}. & \end{array}$$

where *f* is the function to be minimized (often referred to as objective or cost function), and in the case of this thesis can be the result of a Simulink® model or of a calculation made in a Matlab® script. *h* and *g* are the equality and inequality constraint functions respectively, appearing in any number and being calculated from the results of the calculations or from the design variables *x*.

## 2.2. Optimization Methods

There are different classes of optimization methods, [4] defined them to be gradient dependent and gradient-free methods.

### 2.2.1. Gradient based methods

Gradient dependent methods use the information of the gradient to guide the search to the next point of evaluation, since in this thesis the cost function is a model in Simulink or calculation in a Matlab script, hence non-differentiable, gradient methods become less efficient since they require estimating the gradient by finite difference method. This approach increases the number of function evaluations at each step, therefore, gradient-free methods become an interesting choice.

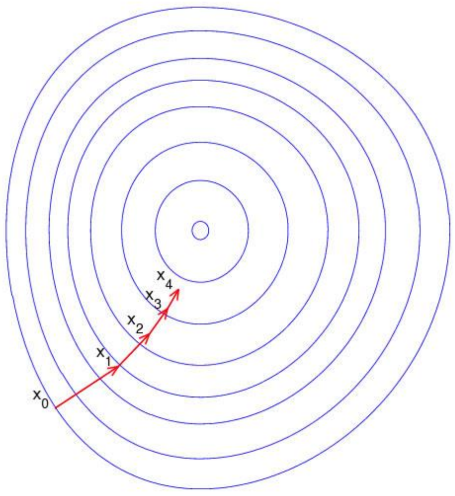


*Figure 2 – Gradient descent search method.*

Figure 2 illustrates the gradient descent method. In this method, the point is moved in the space in the direction of the gradient (perpendicular to the lines that represent constant cost function value), the size of the step is proportional to the gradient, more spaced lines means lower values of gradient, therefore smaller steps.

Gradient methods usually are trapped to the first minimum found, since for most of them the stopping criterion is to achieve zero gradient. Figure 3 illustrates how the starting point of the search affects the convergence towards a local minimum, if the starting point were to the left of the "hill" the solver would have found the global minimum. This fact makes the gradient-based methods less reliable to solve multi-modal (more than one minimum) problems.

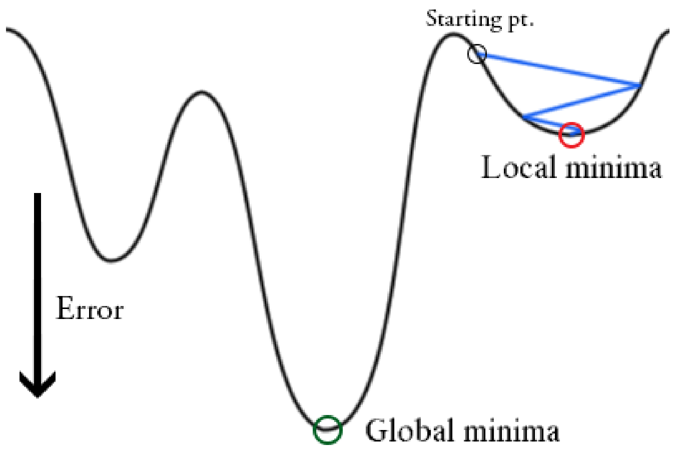


*Figure 3 – Solver being trapped in local minima depends on the starting point.*

### 2.2.2. Gradient-free methods

Gradient-free methods, as the name states, do not calculate the gradient of the function. These methods are highly desired in this thesis since they guide the search without "wasting" function evaluations to estimate the gradient locally. Instead, they search the domain in an efficient way.

The three main families of gradient-free methods are direct search, surrogate based and stochastic methods [4]. Only the most relevant algorithm of each family are discussed in this section, since there are a large number of different methods in these families.

#### 2.2.2.1. Direct Search methods

There are several types of direct search methods and each one search the space in a distinct way. The main idea behind them is to evaluate a certain quantity of points at each step (the number of points can change) and use their cost functions and constraint values to move them around the search space until the optimum is found. The most known methods in this category are the *Nelder-Mead Simplex* Algorithm [5] and the *Mesh Adaptive Direct Search* (*MADS*) [6] that is available in Matlab® under the name of *Pattern Search*.

Both of these methods evaluate a mesh in the search space at each iteration, the best of the points that form the mesh does define where the mesh shall move for the next iteration. The mesh converges to a region where the candidate to a minimum is located, and then it shrinks until it finds it. If the cost function value does not improve when shrinking, the algorithm expands the mesh again in an attempt to find another area that could be better.

Figure 4 shows some examples of meshes that could be used in the *MADS* algorithm, each of the points represent an evaluation of the cost function.

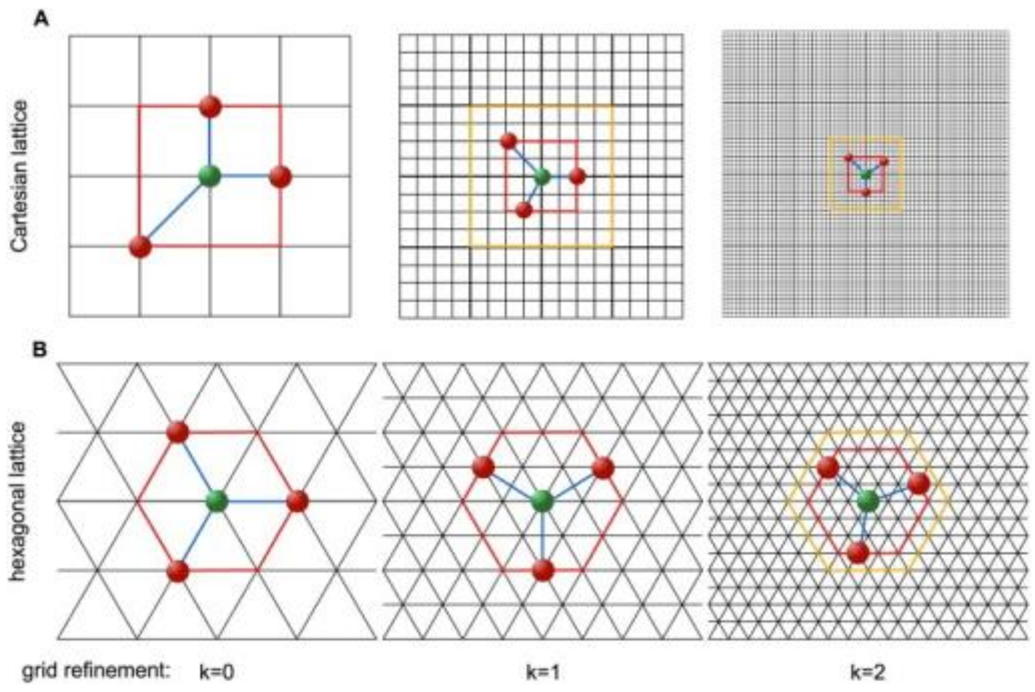


*Figure 4 – Examples of meshes that are used in MADS algorithm.*

### 2.2.2.2. Surrogate model methods

Surrogate model optimization methods use an approximation of the cost function at the range of the evaluation points of the step. This model can be a simple linear/quadratic model or even a response surface generated by the evaluation points [4].

Then, by optimizing this surrogate model, a new direction to move the evaluation points is determined. They are moved in the search space and evaluated until the error between the surrogate model and the cost function values is minimized and the solution converges. Models that are more complex require more evaluation points for a given number of dimensions, increasing the computational cost, but providing a better representation of the function.

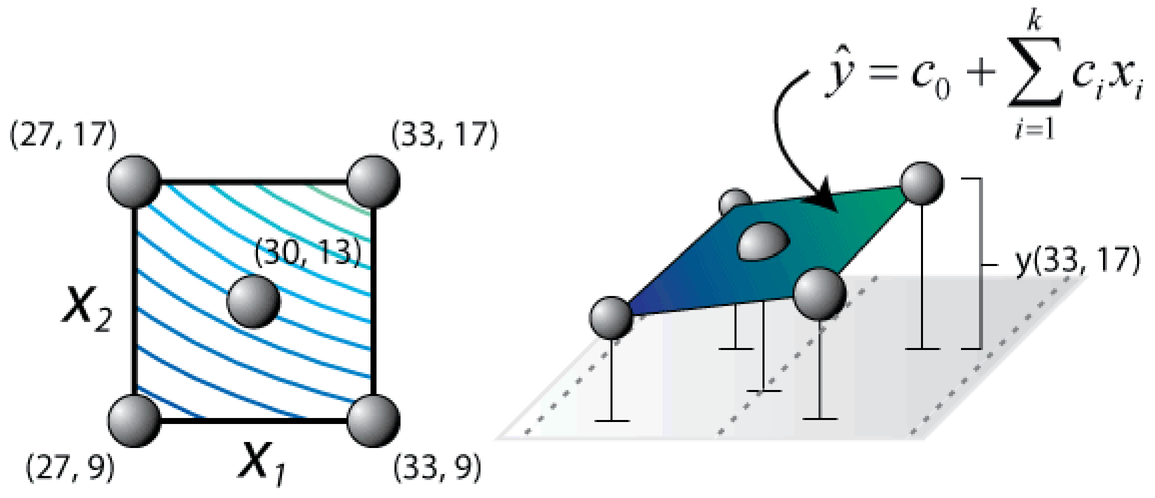


*Figure 5 – Surface fitting on the points used in the response surface method.*

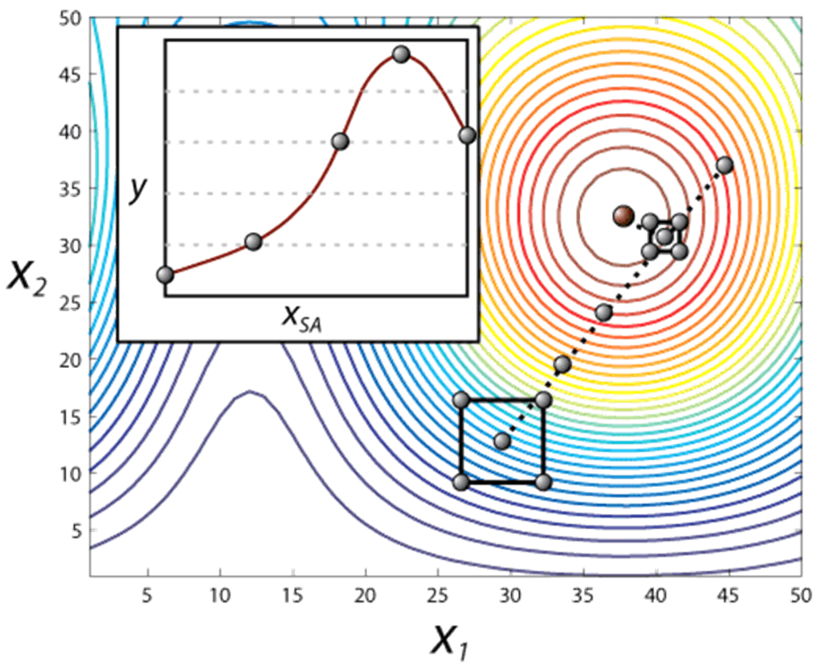


*Figure 6 – Illustration of the response surface method.*

Figure 5 and Figure 6 explains the methodology behind the *response surface method* (RSM), which is similar to many surrogate methods. In this example, a surface is fitted to the points as shown in Figure 5. Then, a direction of search is derived from it and a step size is calculated being proportional to the gradient (similar to the gradient descent method, but in this case applied to maximization). The search continues in the same direction until the value decreases if compared to the previous one. Then, the search goes back one step and evaluates more points to derive a new response surface and a new search direction, as shown in Figure 6.

#### 2.2.2.3. Stochastic methods

A different approach of derivative-free algorithms are the stochastic methods, these methods can perform better than deterministic ones in cases where the cost function has multiple local minima and discontinuities. Since the topology of the cost function in this thesis is unknown and can change depending on the parameters being optimized these methods quickly become an attractive option.

The most common types of stochastic methods are the meta-heuristic methods that aim to improve an answer rather than finding the global minimum. They are proven useful when there is little knowledge of the problem and brute force search is not a viable possibility given the search space is large and the computational cost high. Additionally most of them are applicable to both continuous and discrete optimization (including mixed integer) and can be extensible to multi-objective optimization. Some disadvantages are the lack of convergence proof, the high computing time (but still much lower than brute force) and the sensitivity of the control parameters, which are problem dependent [7].

##### 2.2.2.3.1. Evolutionary Algorithms

The most known family of meta-heuristic methods are the Evolutionary Algorithms (EA). Given a population of individuals in an environment with limited resources, the competition between them causes natural selection, where the fittest individual survives. In the context of optimization, the individuals are the candidate solutions, randomly generated. They compete with other candidate solutions, and at the end of each optimization step, the fittest of them is selected [8].

At each step, the parent individuals create new individuals called children (or offspring) with new parameter combinations. This is called recombination (crossover). Then the new individuals evolve through the mutation operation that aims to increase the diversity and to generate new variable values. By the end of the step, there is a population two times bigger than in the start. Since there are only resources from half of them, the algorithm uses the selection operator to select the individuals that are the fittest in the population (lower cost function value) until the original population size is achieved [8].

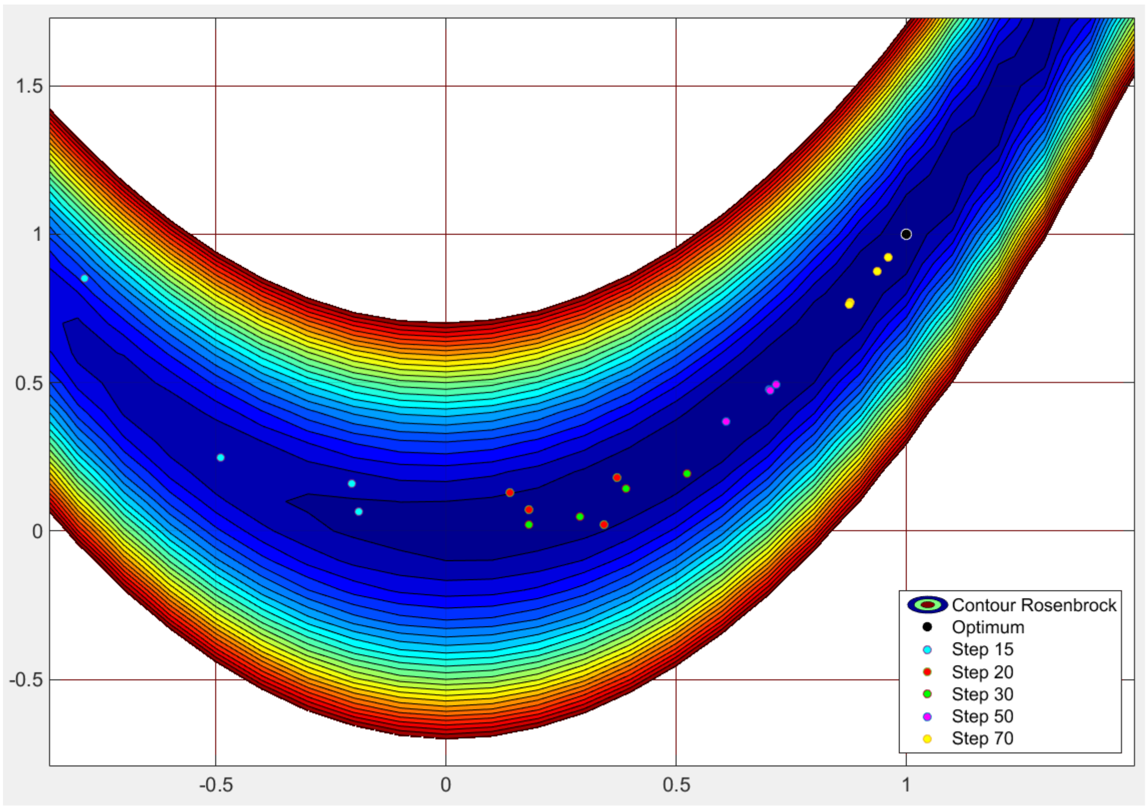


*Figure 7 – Solutions converging towards the optimum region of Rosenbrock function. Different evolutions stages are shown in different colours.*

Figure 7, shows the convergence of an evolutionary algorithm towards the optimum. The example function is the Rosenbrock function (see Section 3.5 for more details). In the figure, individuals at multiple optimization steps are shown, the points converge towards the optimum, shown in black.

Important concepts that define the search in evolutionary algorithms are exploration and exploitation. Exploration is the initial phase of the search where new individuals are being generated through mutation and recombination (crossover), therefore populating the search space. Exploitation is the next phase of the search where the individuals start to converge to regions of good fitness. Too much of one of these characteristics can make the search inefficient. Too much exploration can lead to bad convergence speed and too much exploitation leads to premature convergence, not giving enough time for the individuals to search properly the space before converging to a possible local minimum [8].

#### 2.2.2.3.2. Differential Evolution algorithms

Differential evolution (DE) algorithms uses in the mutation operator information about other individuals of the population. This could be compared to social behaviour and collective intelligence. This intelligence is incorporated to the algorithm by the use of differences between individuals, done by the differentiation operator, which is simple and fast [7]. Therefore, DE is strict mathematical formulated.

It has all the operators as normal evolutionary algorithms, which are mutation, crossover and selection. The differential operator is applied in the mutation phase, where the mutated value is the result from a difference of two individuals of the population times the mutation factor. In the mutation strategy DE/rand/1 [9], three individuals are randomly chosen from the population, the mutated vector is written according to Equation 2:

$$\mathbf{v}_{i,g} = \boldsymbol{x}_{r0,g} + \mathrm{F}(\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g}) \tag{2}$$

where $x_{r0,g}$, $x_{r1,g}$, $x_{r2,g}$ the randomly chosen individuals at generation g, F is the mutation factor and $\mathbf{v}_{i,g}$ is the mutated vector at generation g. There are several other mutation strategies for DE algorithms, some of them use the best individual of the population to guide the others towards the optimum, and these are called, *greedy strategies* [9].

The main advantages of differential evolution over common evolutionary algorithms is the higher convergence speed resultant from the collective intelligence of the population, through the differential operator new successful individuals can be generated more easily.

#### 2.2.2.3.3. Adaptive Evolution algorithms

Most of the evolutionary algorithms require two main parameters to control the evolution, the crossover (recombination) probability and the mutation factor. These parameters are highly dependent on the problem being optimized and they affect directly the convergence in a way not yet understood. Therefore, it is impossible to determine a parameter combination that is suitable for various problems or performs well at different evolution stages of a single problem [9].

To avoid tedious trial and error calibration of the parameters by the user each time, adaptive parameter control methods were developed, their main goal is to change dynamically the parameters during the optimization. Using feedback from the evolution process the parameters are changed to comply with the algorithms needs at each step [9].

The implementation is done by storing a set of control parameters for each individual, and using the successful individual's control parameters to update the population's control parameters.

The result is an algorithm that is robust and flexible on a wide range of problems. In the case of this thesis, this is a highly desirable quality, given that the target problem is most of the times unknown. Tough, one must be aware that a non-adaptive algorithm can outperform an adaptive one if correctly calibrated.

## 2.3. Possible variable types on optimization problems

Many problems in industry require the usage of variables that can only assume specific values. Problems that use only discrete values are often referred to as combinatorial problems, and there are many algorithms specialized to solve them.

In the platform developed in this thesis, the interest lies in a different type of problem, where there could be variables that are continuous and discrete. A problem with both types of variables is called a mixed-integer problem.

Some algorithms can use binary coding for the variables instead of the common known real coded variables (real numbers). The binary coding is often used in Genetic Algorithms. It simplifies the recombination (crossover) operator, since it splits the binary number that represents a variable value in a number of parts and recombines those parts with the ones of a variable from another individual. Since this coding is not so easily understandable by users, in the platform real coded variables are preferred.

## 2.4. Multi-Objective Optimization

Often the problems that are solved by engineers in the industries consist of more than one objective. In automotive companies, a good example is to both minimize fuel consumption and maximize performance. In this case, the goal is to find the best compromise between the objectives, given that they are conflicting.

A multi-objective problem can be formulated as follows:

$$\min \boldsymbol{f}(x) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_M(\boldsymbol{x}))^T \tag{3}$$

where **x** is optimized over a *D*-dimensional search space and $f_i(\boldsymbol{x})$ is one of the M objectives to be minimized [9].

In multi-objective optimization the concept of optimum changes, since now the goal is to find the best compromise (trade-off) between the objectives. For this, the concept of Pareto optimality is used. A solution $\boldsymbol{x}^*$ is Pareto optimal if there is no feasible vector **x** which would decrease some objective without causing a simultaneous increase in at least one other objective (assuming minimization) [10]. In mathematical notation vector $\boldsymbol{x}^*$ dominates another vector **x** if:

$$\forall i \in \{1,2,\dots,M\} : f_i(\boldsymbol{x}^*) \le f_i(\boldsymbol{x}) \; and \; \exists i \in \{1,2,\dots,M\} : f_i(\boldsymbol{x}^*) < f_i(\boldsymbol{x}). \tag{4}$$

A *Pareto front* is the hyper-surface in the objective space that is defined by all Pareto-optimal solutions. The result is a set of best compromise solutions that cannot be dominated by others [9].

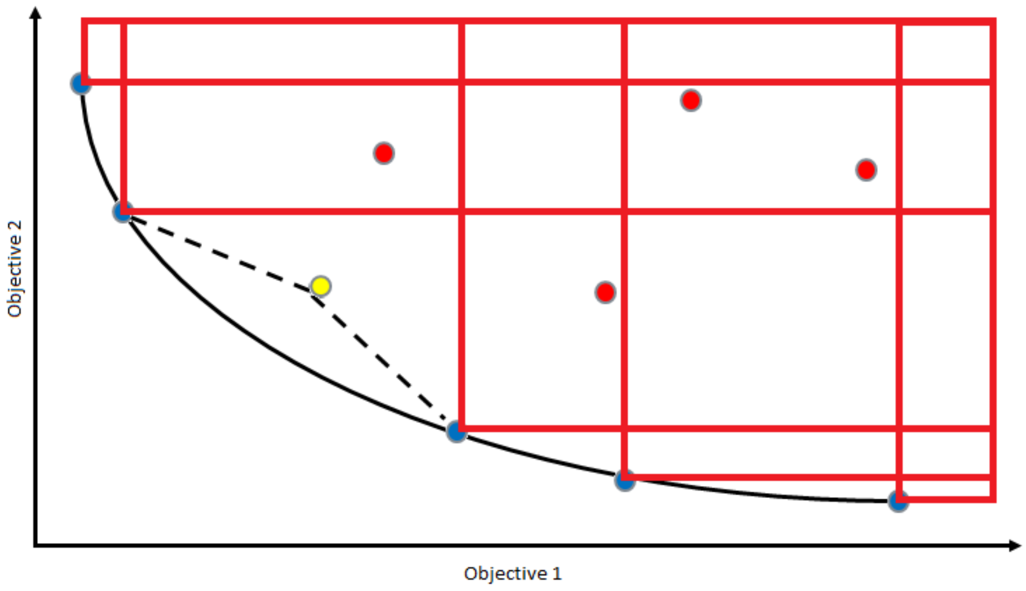


*Figure 8- A Pareto front example. The objective is to minimize both objectives.*

Figure 8 shows an example of a Pareto front. The blue points dominate the red ones, this is shown by the red rectangles drawn starting at the blue points, the area formed by the red rectangles is the area of the objective space that is dominated by correspondent blue point.

It is evident that the yellow point is not inside any rectangles meaning that is not dominated by any other solution and should be integrated to the Pareto front. Section 3.6.2 shows more examples of possible Pareto fronts.

Evolutionary algorithms, that are population based, find several Pareto-optimal solutions in a single run, different from other algorithms that require multiple runs to achieve the same approximation, which is the case of some stochastic processes and most of the gradient based methods [11]. The crossover and mutation operators naturally make the solutions converge to a Pareto position in the front, and its resolution depends on the number of individuals in the population.

There are methods to transform a multi-objective problem into a single objective one, without disregarding the other objectives, the most common technique is *scalarization* [12], where the objective becomes a weighted sum of the other objectives, the downside of this method is that it depends on assigning the right weight value by the user. The result using *scalarization* does not provide the Pareto front, and to achieve that, the optimization must be run several times with different weights, then all of these results could approximate the Pareto front [3].

A more effective approach is the *hypervolume* indicator [13], it concatenates the multiple objectives into a single objective taking into account the dominance of the solutions, then the objective becomes only minimizing the *hypervolume* instead of minimizing each of the objectives. Since it takes into account the dominance of the solutions, the result of the optimization becomes automatically a Pareto front. Therefore, the calculation of the hypervolume becomes highly computationally expensive given the number of objectives and the number of elements in the Pareto front [13]. Figure 9 shows an example of a hypervolume defined by eight points in a three objective space.

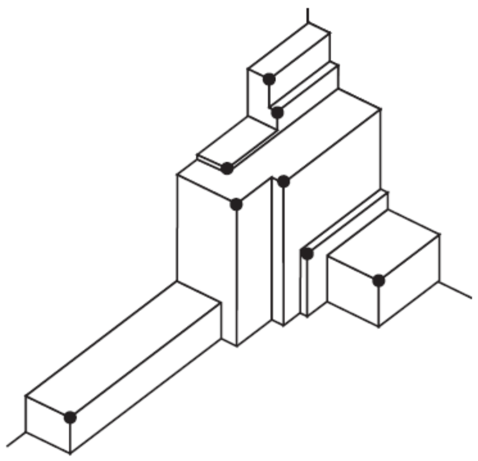


*Figure 9 – Hypervolume defined by a set of eight solutions for three objective functions [13].*

Instead of dealing with a multi-objective problem, an alternative method is to set a constraint for the other objectives, forcing the results to be within a margin of a target value. Then it is possible to ensure that the other objective is not be denigrated.

# 3. Methods

This section describes the methodology used, the methodology did not follow any standards and it was particular to this thesis. Each of the steps is described in the next sub-sessions.

## 3.1. Platform building

Initially the building of the platform was addressed. It was developed in Matlab® software since the optimization platform used at Volvo AB is fully based on Simulink®, what gave a higher compatibility, also there was a high number of optimization algorithms available as Matlab® code in the internet free to use.

The platform followed a modular configuration, with the more reusable scripts as possible to lower maintenance cost in the future. The topology of the tool is shown in Figure 10. The tool can performs tasks like optimization and calibration, for each of those tasks an algorithm should be selected, then the proper connector, a script that creates the interface between the tool and the application. The objective value is computed from the outputs of the application. This can be the result of a simulation model, a simple Matlab® script or a generic output from any commercial software.

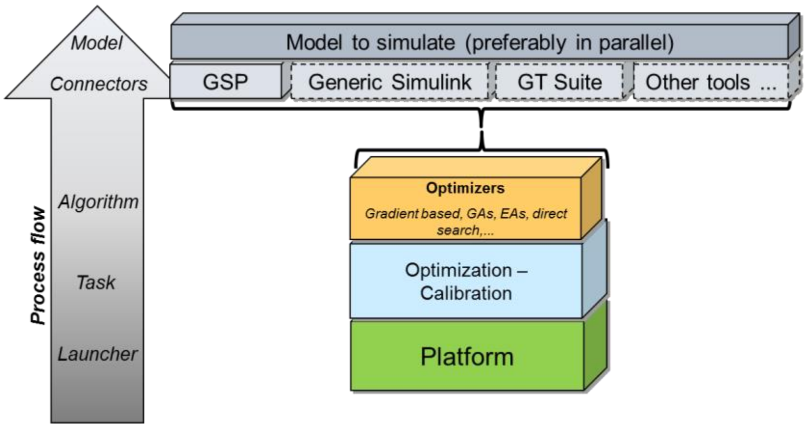

Model
Connectors
Algorithm
Task
Launcher
Process flow
Model to simulate (preferably in parallel)
GSP
Generic Simulink
GT Suite
Other tools ...
Optimizers
Gradient based, GAs, EAs, direct search,...
Optimization – Calibration
Platform

*Figure 10 - Overview of the platform and its subparts.*

The platform was built on top of a previous development started in 2015, but several aspects of it were changed. The first of these changes was to enable user defined cost functions and constraints functions, in a way that the user could choose as many as wanted of each, they are very generic which gives the user full freedom of implementation. The tool needed a resume functionality, in case the user needs more steps in the evaluation or if the optimization crashes by any reason (leaving the laptop without the power cord attached is the most common occurrence).

The tool needed to handle multi-objective as well as mixed-integer optimization, the implementation of both functionalities is described later in this chapter.

The simulations and calculations were parallelized in Matlab® to save time, in this way, the whole batch of simulations can be split into different *workers*, and each of those *workers* runs its own batch and sends the data back to the main session of Matlab®. In this way, optimization in applications like GSP can be run in a reasonable time. This can be done by launching several Matlab® sessions that are closed once their batch is complete, or by using the distributed computing toolbox from Matlab®.

Afterwards it was identified the need of implementing the possibility to load calibration files to be used in the calibration task, then the results of simulation could be compared with a calibration measurement so an error can be calculated.

To set up a problem the user must code the cost and the constraint functions. Then, indicate the model or calculation file that is used to evaluate a design. If the GSP application is being used, the cycles should also be defined. If the goal is to perform a calibration, then a measurement file must be indicated. The variables must be defined with information and ranges. Finally, the user must input the number of steps that are run and the number of processor cores to be used if the goal is to run the evaluations in parallel.

## 3.2. Algorithms selected

Five algorithms were selected to be evaluated in this thesis. Derivative-free algorithms were preferred over gradient-based methods given the nature of the target problems of this thesis (as discussed in Section 2.2).

Many algorithms have been created in the past years, selecting an algorithm for a task can be complicated given that each is made to a specific kind of problem, constraint and variables type. In the case of this thesis, an extra complication had to be taken into account, which was the fact that the algorithms could be used for any problem, any kind of constraints and variables from discrete to continuous set by the user.

Candidate algorithms were selected based on their effectiveness to solve similar problems as the ones of this thesis, this was identified by analysing papers where the authors use multiple techniques and benchmark them [3] [14]. However, the models used in most papers were simple models and less computational expensive than the ones used in this thesis, hence faster algorithms are needed. Additionally, the algorithms benchmarked were out-dated and better versions of them were available for usage.

Table 1 shows the information of each of the selected algorithms. It is important to state that the algorithms did not need to comply with all the requirements of the tool, and that some methods were applied so these algorithms, so they did comply with those requirements. In the case of the mixed integer optimization, a method was applied that enables the usage of both discrete and continuous variables for all algorithms (see Section 2.3 for more details), but none of the algorithms is a proper mixed-integer optimizer. Also for multi-objective optimization, the only algorithm that can properly perform this task is *JADE* (see Section 3.6), but the others can be used if a weighting factor is applied to the cost functions and the optimization is run several times for different sets of weights. Evolutionary algorithms are by nature unconstrained, it was necessary to implement a constraint handling method for *JADE*.

*Table 1 – Selected algorithms properties and their classification.*

| ***Algorithm*** | ***Family*** | ***Applicable on*** | ***Constraint*** | ***Multi-objective method*** | ***Variables type*** |
|---|---|---|---|---|---|
| Lean gradient (in-house) | Direct search surrogate model based | Smooth convex problems | Non-linear | *scalarization* | Continuous |
| Linear DoE (in-house) | Direct search surrogate model based | Smooth convex problems | Non-linear | *scalarization* | Continuous |
| fmincon – SQP (Matlab®) | Gradient-based line search | Smooth convex problems | Non-linear | *scalarization* | Continuous |
| Pattern search (Matlab®) | Direct search global optimizer | Smooth non-convex problems | Non-linear | *scalarization* | Continuous |
| JADE | Adaptive Differential Evolutionary | Non-smooth non-convex problems | Unconstrained | Non Dominating Sorting by crowding density | Continuous |

The next sections describe briefly the mechanisms behind each of the selected algorithms.

### 3.2.1. Lean Gradient

The *Lean Gradient* algorithm is an in-house optimizer developed at Volvo AB and it is classified as a surrogate model optimizer. The main idea behind it is to evaluate a DoE (Design of Experiments) design [15]. This design is a number of parameter combinations that represents the data with a certain resolution. To achieve a high resolution on a design, the size of it (number of different parameter sets) must be large. To avoid large number of tests at each optimization step a lean design was adopted [16]. This design can represent the data well enough to guide the optimization towards the optimum.

At each step, the lean design is evaluated and the cost function values are used to solve an undetermined system of equations. The solution is used to determine what the direction that the optimizer should go is. In the next step, the search space is updated or by moving the ranges or by shrinking it (zooming into and specific area in the domain).

Since there is no condition that makes the solver increase the range (zoom out) of the search it can be trapped in local minima, but for cases where the function is smooth and convex it converges fast and with a low number of evaluations of the cost function.

### 3.2.2. Linear DoE

*Linear DoE* was also developed at Volvo AB. The method consists of evaluating a saturated DoE design [16]. Then, using the values of the cost function and constraints at those points it creates a linear surrogate model. The linear model is optimized within the range taking into account the values of the constraints. As a result, the range can move or shrink depending on the best value of the step. The solver stops when the error between the linear model and the actual cost function is low, or when there is no improvement of the cost function values.

*Linear DoE* requires more function evaluations at each step than *Lean Gradient*, since it uses a saturated design instead of a lean design, but it is still less than most algorithms. It is especially good for constrained optimizations, since the constraints are also modelled in the range.

### 3.2.3. Fmincon

*Fmincon* is the Matlab® algorithm for constrained minimization, and it is a part of the basic optimization toolbox [17]. This function uses different algorithms inside and each of them can be used in certain situations, the documentation web page [18] shows recommendations to choose the correct algorithm for the task. The recommendation is to use the *interior-point* algorithm first and then *sqp* (sequential quadratic programming) if the result is not as expected. The problem with the *interior-point* algorithm is that, due to the barrier function calculated inside the algorithm, the solution does not approach the boundaries of the inequalities constraints [18]. Since in the platform, all constraints are implemented as inequality constraints (see Section 3.3 for more information), this algorithm is not recommended given that it could provide sub-optimal solutions too often. Finally, the algorithm selected in the platform was the *sqp*.

The *sqp* is a line search method that calculates the derivative by finite differences method. It applies a small increment on each search dimension to estimate the gradient. Then, the algorithm determines the step size that should be given towards the optimum, and it repeats the process until the minimum is found.

*Fmincon* is a local optimizer, that means that it is not made to find the optimal of multi-modal functions, but for simple functions, it can be fast converging.

### 3.2.4. Pattern Search

*Pattern Search* is a Matlab® function that is a part of the Global Optimization Toolbox. It gathers multiple grid search methods, the one being used in the platform is the MADS (mesh adaptive direct search). The main idea of MADS is to evaluate a mesh of points in the search space, and moving and shrinking it around the search domain until the minimum is found, if the cost function values do not improve when it shrinks, the solver expands the mesh again to search for a new region. *Pattern Search* is a global optimizer that can find the global optimum of multi-modal functions.

### 3.2.5. JADE

*JADE* [9] [19] is an adaptive differential algorithm, which means that it is a differential evolution algorithm and has adaptive control parameters. Therefore, it aggregates the social intelligence of differential evolution with the flexibility and robustness of the adaptive evolution.

It is known that the parameters *F* and *CR* (mutation factor and crossover probability) are problem-dependent and that it is necessary to calibrate them for each problem. An advantage of *JADE* is that it eliminates the need of this calibration by implementing the parameters *c* and *p*, which can be chosen according to their role in the algorithm given that they are insensitive to the problem [9]. The parameter *c* rules the adaptation of the crossover probability and mutation factor, while *p* determines the percentage of best values that can be randomly chosen to compute the mutation as it is be shown in Equation 9. The adaptation of the control parameters is done according to the following equations:

$$CR_m = (1-c)\ .\ CR_m + c\ .\ mean(S_{CR}) \tag{5}$$

$$F_m = (1-c)\ .\ F_m + c\ .\ \frac{\sum (S_F)^2}{\sum S_F} \tag{6}$$

$$CR_i = CR_m + 0.1\ .\ randn \tag{7}$$

$$F_i = F_m + 0.1\ .\ \tan\big(\pi\ .\ (rand - 0.5)\big) \tag{8}$$

where $CR_m$ and $F_m$ are the mean values of the distribution that generate $CR_i$ and $F_i$ for the individual *i* at each generation. $S_{CR}$ and $S_F$ are vectors that contain the values of *CR* and *F* that successfully generated better individuals in the last generation, in other words, generated children (offspring) that replaced their parents. The value of $CR_i$ is generated by a normal distribution with mean $CR_m$ and standard deviation of 0.1 as Equation 7 shows, while the value of $F_i$ is calculated by a Cauchy distribution, as Equation 8 shows. The Cauchy distribution is more helpful to diversify the values of mutation factor when compared to a normal distribution, decreasing the probability of early convergence at local minima [9]. The Lehmer mean $\left(\frac{\sum (S_F)^2}{\sum S_F}\right)$ used in Equation 6 also helps propagating larger values of mutation factors, improving the progress [9].

The successful individuals are responsible to update the control parameters and adapt them to the needs of the algorithm at a certain evolutionary stage. The higher the value of the parameter *c* the fastest this adaptation takes place. Too high values can lead to fast convergence, but low reliability in the results. *JADE* performs best with c ∈ [1/5, 1/20] and p ∈ [5%, 20%] [19].

*JADE* has two different versions, with and without the archive. The archive stores members of the population of past generations that can be used in the evolution process. The archive is set to a certain size, and once it becomes full, individuals are randomly taken out of it in order to maintain its original size. The archive stores the inferior solutions throughout the process, and it is used in the mutation process.

The mutation strategy that *JADE* uses is DE/current-to-*p*best/1 developed especially for this algorithm. It is a greedy strategy in order to improve convergence speed, but the algorithm does not lose its reliability since the adaptive control of the mutation factor determines how greedy the search is [9]. Equation 9 shows how the calculation of the mutated vector is done.

$$v_{i,g} = x_{i,g} + F_i\left(x_{best,g}^{p} - x_{i,g}\right) + F_i\left(x_{r1,g} - \tilde{x}_{r2,g}\right) \tag{9}$$

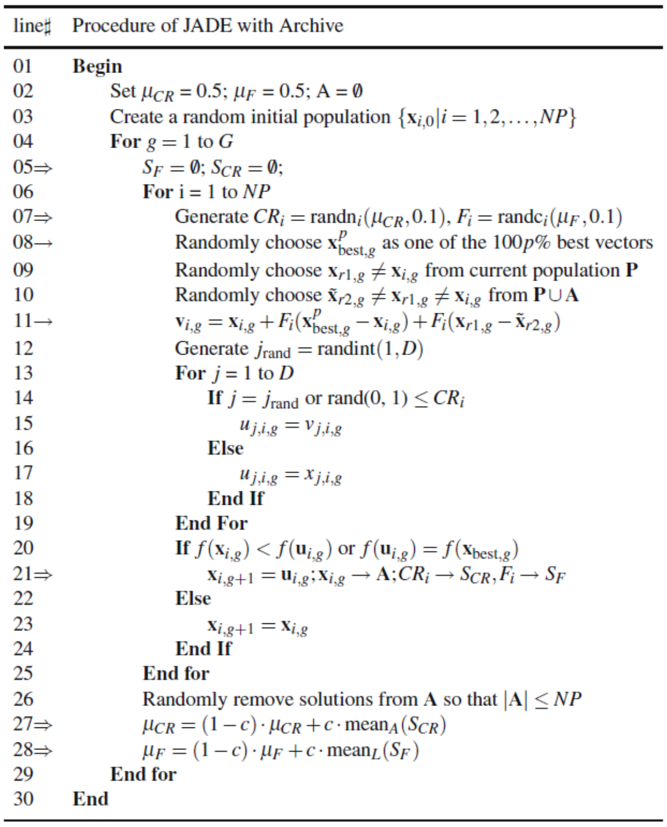
where $v_{i,g}$ is the mutated vector, $x_{i,g}$ is the parent individual, $x_{best,g}^{p}$ is a vector randomly chosen from the $p$% best individuals of the population, and $x_{r1,g}$, $\tilde{x}_{r2,g}$ are the random individuals chosen from the population, but in the case of *JADE* with archive, the individual $\tilde{x}_{r2,g}$ is chosen from the union of the archive and the population. In this way, the expression $x_{r1,g} - \tilde{x}_{r2,g}$ is a difference between a good solution (from the current population) and a possible worst solution (from the union of the archive and the population) and helps guiding the individuals towards the optimum.

The $v_{i,g}$ vector goes into the crossover operator to generate the $u_{i,g}$ trial vector. The $u_{i,g}$ is the child individual generated at generation g and has values from the parent $x_{i,g}$ and the mutated vector $v_{i,g}$.

A mutated value of a variable is transmitted to a child under two conditions. The first is if a randomly generated number in the range of 0 to 1 is higher than the crossover probability of that individual, $CR_i$. The second is particular to JADE, and for each individual, a variable j is randomly chosen in the range of 1 to D, where D is the number of variables being optimized. Then, the mutation occurs for that variable (does not matter the result from the first condition). This ensures that always the child has one coordinate mutated even if the $CR_m$ value is really low (low probability of generating high $CR_i$ values for the individuals).

At the end the selection process choses between the children and the parents, which ones are the best.

Figure 11 shows the algorithm of *JADE* with archive for single objective optimization.



```
line♯  Procedure of JADE with Archive
01     Begin
02         Set $\mu_{CR} = 0.5$; $\mu_F = 0.5$; $A = \emptyset$
03         Create a random initial population $\{\mathbf{x}_{i,0} | i = 1, 2, \ldots, NP\}$
04         For $g = 1$ to $G$
05⇒            $S_F = \emptyset$; $S_{CR} = \emptyset$;
06             For i = 1 to $NP$
07⇒                Generate $CR_i = \text{randn}_i(\mu_{CR}, 0.1)$, $F_i = \text{randc}_i(\mu_F, 0.1)$
08→                Randomly choose $\mathbf{x}^p_{\text{best},g}$ as one of the $100p\%$ best vectors
09                 Randomly choose $\mathbf{x}_{r1,g} \neq \mathbf{x}_{i,g}$ from current population $\mathbf{P}$
10                 Randomly choose $\tilde{\mathbf{x}}_{r2,g} \neq \mathbf{x}_{r1,g} \neq \mathbf{x}_{i,g}$ from $\mathbf{P} \cup \mathbf{A}$
11→                $\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F_i(\mathbf{x}^p_{\text{best},g} - \mathbf{x}_{i,g}) + F_i(\mathbf{x}_{r1,g} - \tilde{\mathbf{x}}_{r2,g})$
12                 Generate $j_{\text{rand}} = \text{randint}(1, D)$
13                 For $j = 1$ to $D$
14                     If $j = j_{\text{rand}}$ or rand(0, 1) $\leq CR_i$
15                         $u_{j,i,g} = v_{j,i,g}$
16                     Else
17                         $u_{j,i,g} = x_{j,i,g}$
18                     End If
19                 End For
20                 If $f(\mathbf{x}_{i,g}) < f(\mathbf{u}_{i,g})$ or $f(\mathbf{u}_{i,g}) = f(\mathbf{x}_{\text{best},g})$
21⇒                    $\mathbf{x}_{i,g+1} = \mathbf{u}_{i,g}; \mathbf{x}_{i,g} \to \mathbf{A}; CR_i \to S_{CR}, F_i \to S_F$
22                 Else
23                     $\mathbf{x}_{i,g+1} = \mathbf{x}_{i,g}$
24                 End If
25             End for
26             Randomly remove solutions from $\mathbf{A}$ so that $|\mathbf{A}| \leq NP$
27⇒            $\mu_{CR} = (1-c) \cdot \mu_{CR} + c \cdot \text{mean}_A(S_{CR})$
28⇒            $\mu_F = (1-c) \cdot \mu_F + c \cdot \text{mean}_L(S_F)$
29         End for
30     End
```

*Figure 11 – JADE with archive single objective algorithm.*

## 3.3. Constraint Handling

The platform has three types of constraints: design, performance and standard constraints.

The design constraints are calculated using the variables values only, this can be used to verify if a set of values can be evaluated or not. If the candidate solution is unfeasible by design, it is not evaluated and it is assigned a dummy value.

Performance constraints are used only for the GSP application, where there is a need to evaluate the vehicle in a standard performance cycle e.g. an acceleration cycle. By running this additional simulation, it is possible to determine if the vehicle complies with basic performance criteria, if the vehicle does not comply with the criteria it is not be evaluated in the other cycles and a dummy value is assigned to its cost function.

The dummy values assigned to the individuals unfeasible by design or by performance are later on corrected. This correction is done following the method shown in Section 3.3.3.

Standard constraints, refers to the constraints that are calculated based on the results of the calculation or simulation. They take the form of inequality constraints always, if the user wants to use an equality constraint then it is needed to set the absolute value of the constraint breaching minus a tolerance to be lower than zero according to the Equation 10 [20]. This was necessary to avoid having dedicated constraint-handling techniques for each algorithm.

$$|h(x)| - \sigma \leq 0 \tag{10}$$

### 3.3.1. Built-in Matlab optimizer constraint handling

The Matlab® optimization algorithms evaluated (*fmincon* and *pattern search*) are treated as black boxes and it was impossible to change their constraint handling approach. Then, to use them in the platform, all constraints were considered non-linear and were all set to be inequality constraints. Again, if the user wants to use an equality constraint, it must use the methodology of Equation 10.

### 3.3.2. In-house optimizers constraint handling

*Linear DoE* algorithm uses both the cost function and the constraint values to create a linear surrogate model, and then it solves the resultant constrained linear problem. In this way, the optimum found complies with the constraints at each iteration. The algorithm only stops when the value of the constraint breaching is lower than the tolerance set by the user.

*Lean gradient* uses the information of the best feasible cost function value of each step to guide the search. This method is not robust, and then it is recommended using *Linear DoE* on constrained problems.

### 3.3.3. JADE constraint handling

Evolutionary algorithms are naturally unconstrained, to constrain an evolutionary algorithm, such as JADE, a common practice is to multiply the constraint breaching by a penalty value and add this value to the cost function of the individual that is considered unfeasible. In this way, the search is not be guided to the unfeasible direction, since the cost function value is higher in the unfeasible region.

In general, the penalty values should be kept as low as possible, just above the limit that unfeasible solutions are optimal. This is called the *minimum penalty rule* [21]. If the penalty value is too low, then an unfeasible individual can be selected, since the value of the penalty is seen as negligible. On the other hand, a high penalty value can discourage the algorithms to search the region close to the feasible boundary, where often the optimum lies.

According to [21], there are six classes of penalty functions: static penalty; dynamic penalty; annealing penalty; adaptive penalty; co-evolutionary penalty and death penalty. Each one of them has their pros and cons.

Static penalties are problem dependant, the value of the penalty must be carefully chosen so they are not too high or low than the cost function values at the unfeasible points, therefore previous knowledge about the cost function values is necessary. Additionally, it is difficult to determine a single value of penalty for a constraint that works well throughout the whole domain.

In dynamic penalty methods, it is hard to derive a good penalty function to change the values of the penalties dynamically, and the derived expression is still problem dependent.

Co-evolutionary, adaptive penalty and annealing penalty are all very advanced methods that would take long time to implement and tune, increasing the complexity of the problem considerably, still not guaranteeing a robust implementation of the cost functions.

The chosen method is a variation of the death penalty method described by [22]. In this method, the individual that is considered unfeasible receives the value of the worst feasible individual of the whole process added the sum of the normalized constraint breaching. In this way, during the selection process if (i) two feasible solutions are compared, the best objective value is chosen, (ii) one feasible and one unfeasible solution are compared, the feasible solution is chosen, (iii) two unfeasible solutions are compared, the one that has the lowest constraint breaching is chosen. It is vital to stress out that constraint-breaching values must be normalized to avoid any bias from a certain constraint before adding to the objective value [22].

Figure 12 illustrates how this method was implemented. The dashed line is the cost function values, when approaching from the right, the values of the cost function become lower, and then the search mechanism would continue the search in this direction.

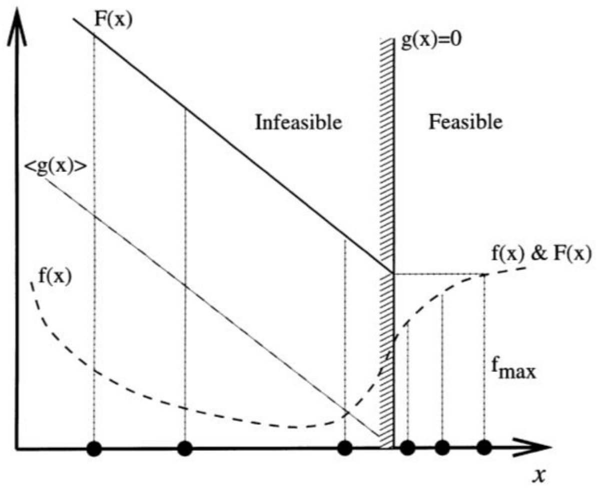


*Figure 12 – Illustration of the constraint handling method implemented for JADE in the platform [22].*

It is needed to discourage further search for better individuals in the unfeasible area (region on the left of g(x) = 0), and this is done by giving to the first point of the unfeasible area (just after the constraint become activated) the value of the worst feasible cost function indicated by the figure by $f_{max}$. In order to guide the individuals back to the feasible area, a slope is created by giving to the unfeasible individuals the worst feasible cost function value added to the sum of the normalized constraint breaching. In this way an individual further inside the unfeasible region would be guided towards the border and then back to feasibility.

For all the algorithms, if a design constraint or a performance constraint is breached, the simulations or calculations are not run, and then a dummy value is assigned for the cost function. Later, in the post processing, it is assigned to the unfeasible solution the value of the worst feasible solution added to the sum of the normalized constraint breaching.

## 3.4. Mixed-integer implementation

The following method was applied to allow both discrete variables and continuous variables to be used in the optimization without any algorithm compatibility.

In this method, the user inputs the different levels that the variable can assume, then the values are coded as 1, 2 … N, where N is the number of variables levels (allowed values), these coded variables are sent to the optimization algorithm as continuous variables. Each time that the algorithms sends to the cost function a combination of values to be evaluated, the platform rounds the continuous value to the closest integer. The result is the code of the variable, which is used to retrieve the correct value of the variable.

This method is not a proper mixed-integer optimization, but it is a very simple way to make any algorithm compatible with this feature. In Section 5.1, it was verified to be a successful method.

## 3.5. Algorithms benchmark

Each algorithm was tested on a set of eight benchmark computationally expensive cost functions from CEC 2014 (Congress on Evolutionary Computation) [23]. This helped to identify the strengths of each algorithm on different problems, each of these functions has a peculiar characteristic that makes it complex to optimize, and several times real problems present some of those characteristics (see Section 5.1.1 for more details). By benchmarking them, it was possible to determine if an algorithm could be successful in real test case usage or not.

The benchmark functions were coded for any number of dimensions, the higher the number of dimensions the harder it is to solve them. In this thesis, the number of dimensions was fixed to ten. The range of evaluation for each of these functions was changed from what is recommended by CEC 2014 since many optimizers have initial evaluations exactly on the middle of the range, where the optimum is located. As a result, they achieved their goal at the first step, to avoid that, the range used is non-symmetrical. Table 2 shows the information about these functions. Plots for the two dimensional version of these functions are found in Appendix A.

The first four functions are uni-modal, which means that they have only one local minimum that is  the global minimum, while the last four functions are multi-modal, having many local minima and one global minimum.

The tests compared the number of evaluations needed in order to find the global minimum of these functions and if the correct value was found in a single run. The number of evaluations is directly proportional to the execution time. Different methods can be compared by number of evaluations, since there is a negligible computational overhead for each method.

Table 2 – CEC 2014 test functions information.

| ID | Function Name | Range | Minimum | Formula |
|---|---|---|---|---|
| 1 | Sphere | [-20,22] | $x_j = 0, j \in [1, \dots, 10]$. | $f_1 = \sum_{j=1}^{N} x_j^2$ |
| 2 | Ellipsoid | [-20,22] | $x_j = 0, j \in [1, \dots, 10]$. | $f_2 = \sum_{j=1}^{N} j.x_j^2$ |
| 3 | Rotated Ellipsoid | [-20,22] | $x_j = 0, j \in [1, \dots, 10]$. | $f_3 = \sum_{j=1}^{N} \left( \sum_{k=1}^{j} x_k^2 \right)^2$ |
| 4 | Step | [-20,22] | $x_j = 0, j \in [1, \dots, 10]$. | $f_4 = \sum_{j=1}^{N} floor\left(x_j + 0.5\right)^2$ |
| 5 | Ackley | [-32,33] | $x_j = 0, j \in [1, \dots, 10]$. | $f_5 = -20.e^{\left(-0{,}2.\sqrt{\sum_{j=1}^{N} \frac{x_j^2}{j}}\right)} - e^{\left(\sum_{j=1}^{N} \frac{\cos(2.\pi.x_j)}{j}\right)} + 20 + e$ |
| 6 | Griewank | [-50,52] | $x_j = 0, j \in [1, \dots, 10]$. | $f_6 = \sum_{j=1}^{N} \frac{x_j^2}{4000} - \prod_{j=1}^{N} \cos\left(\frac{x_j}{\sqrt{j}}\right) + 1$ |
| 7 | Rosenbrock | [-20,22] | $x_j = 1, j \in [1, \dots, 10]$. | $f_7 = \sum_{j=1}^{N-1} (100.(x_j^2 + x_{j+1})^2 + \left(x_j - 1\right)^2)$ |
| 8 | Rastringin | [-20,22] | $x_j = 0, j \in [1, \dots, 10]$. | $f_8 = \sum_{j=1}^{N} (x_j^2 - 10.cos\left(2.\pi.x_j\right) + 10)$ |

A complete benchmark would require multiple runs of each algorithm, and to analyse the mean and standard deviations of these results, but given time constraints, only a single run of each algorithm was made. This was not entirely wrong, given that users do not usually run the problems more than once. In a certain way, these tests assessed the capacity of the algorithm to deliver good results in a single run, tough no clear conclusions could be drawn in terms of convergence speed or accuracy. The process was stopped once the algorithms found a value lower than 0.001, since for all these functions the minimum is equal to zero.

The results are shown Table 3. The first value is the cost function and the value in parenthesis is the number of evaluations that were required to achieve this result. The results that are in bold are the best results for each problem and the ones that are written in red are optimizations that were stopped for taking too long, this was identified when the number of evaluations was about twice the second highest number of evaluations for that problem. Cells with red shading did not find the minimum, and green shading shows the successful ones.

*Table 3 – Results of the optimization of the CEC 2014 test functions. The first value of each cell is the final cost function value, the value in the parenthesis is the number of cost function evaluations needed to achieve the result. Text in bold highlights the best method that solved the problem, while the red text shows simulation that were stopped for taking too long.*

| ***Function*** | ***lean_gradient*** | ***linear_DoE*** | ***fmincon*** | ***patternsearch*** | ***JADE*** |
|---|---|---|---|---|---|
| *f1* | *6,19E-4 (342)* | ***6,4E-4 (96)*** | *0,1054 (242)* | *0.0874 (11089)* | *9,74E-4 (5850)* |
| *f2* | *9,87E-4 (498)* | ***4,90E-5 (300)*** | *2,34E-4 (352)* | *0.9241 (11658)* | *4,35E-4 (6200)* |
| *f3* | ***9,97E-4 (1332)*** | *2,1715 (2399)* | *1,5962 (242)* | *29,6956 (3311)* | *8,26E-4 (6850)* |
| *f4* | *2 (192)* | *10 (70)* | *974 (33)* | *0 (3706)* | ***0 (2350)*** |
| *f5* | *2,0133 (552)* | *3,0274 (792)* | *19,9668 (44)* | *1,14E-2 (10331)* | ***1,12E-2 (7650)*** |
| *f6* | *2,21E-2 (768)* | *0.6442 (6307)* | *0,3983 (231)* | *5,66E-2 (5193)* | *1,77E-4 (22300)* |
| *f7* | *171,5582 (2760)* | *7,2996 (14098)* | *83,864 (407)* | *18,896 (9794)* | ***8,77E-4 (602)*** |
| *f8* | *10.9445 (7842)* | *0,1269 (715)* | *134,1259 (418)* | *1,0005 (5998)* | ***8,74E-4 (17450)*** |

The built-in optimization algorithms from Matlab® proved to be unsuccessful in the tests. *Fmincon* only found the optimum of a single function, and was still slower than *Linear DoE* and *Lean Gradient* at it. *Pattern Search* only found the optimum once and was slower than *JADE*. *Lean Gradient* was better than *Linear DoE*, but not by much. *JADE* was the only algorithm that successfully solved all the problems.

These results represent the behaviour of a single run of each method, if more executions were done, the results could be different. Naturally, a different starting point for *fmincon* or *Pattern Search* could have improved the answer, but most of the times there was no indication of where this point should be placed.

Nevertheless, some recommendations can be derived from these results. The first was that JADE is the right choice for most of the problems, since it was clearly more robust than the others were. *Linear DoE* and *Lean Gradient* started to be better when the cost function was not so complex, but each evaluation of it was costly time wise, e.g. a complex simulation, since they use less function evaluations to converge. *Fmincon* could be used when there was certainty that the problem was convex and smooth. *Pattern Search* could be taken out of the tool since it was worse than JADE in all tests.

## 3.6. Multi-Objective implementation

The only algorithm that performs proper multi-objective optimization is JADE. Other algorithms evaluated in this thesis can also perform it by using the scalarization method, as explained in Section 2.4, but to get a Pareto front it is necessary to re-run the optimization with different set of weights. This section focus in the implementation and modification of JADE to perform multi-objective optimization.

### 3.6.1. Multi-Objective implementation in JADE

In population based algorithms the individuals quickly become non-dominated, then it becomes impossible to select the best when the dominance criteria ties. Therefore, an alternative measure must be adopted [9].

Figure 13, shows the Pareto front at different evolution stages of a multiobjective optimization. In the initial stages, the left figure shows that there is a clear gradient towards the Pareto front and that the best individuals are used to guide the others towards it, but as the process continues, the individuals become more scattered and the difference between them cannot be used anymore to guide the search as efficiently.

To overcome the challenges of multiobjective optimization the authors of [9] suggest that modifications in *JADE* are made in the selection and mutation operations.

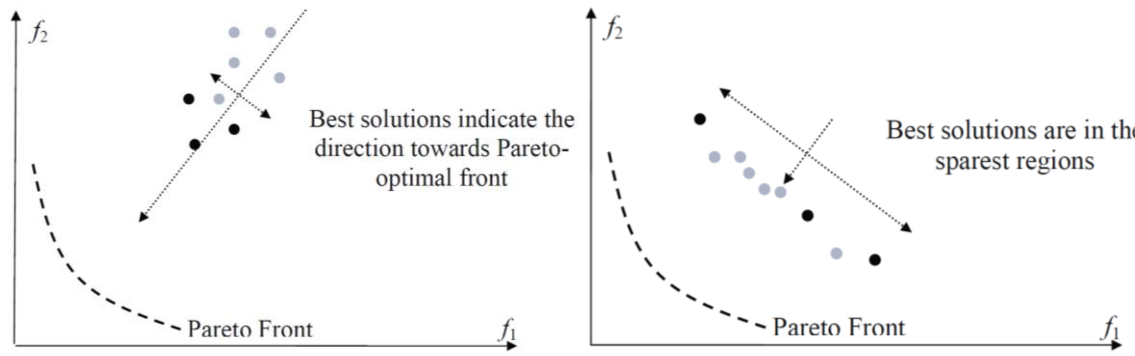

$f_2$
Best solutions indicate the direction towards Pareto-optimal front
Pareto Front
$f_1$
$f_2$
Best solutions are in the sparest regions
Pareto Front
$f_1$

*Figure 13 – Pareto front at different evolution stages, left figure shows individuals in early stages and right figure shows at more advanced stages. [9]*

The selection process suggested by [9] was not successfully implemented, there were some inconsistancies when runnnig the ZDT test functions (see Section 3.6.2), consequently, only the mutation process was modified following the guidelines of the authors of [9]. Finally, the implemented selection process was the non-dominating sorting method from *NSGA II* [24].

The new selection process implementation has the intention to reduce the number of individuals from 2NP to NP, where NP is the number of individuals in the population. This was done in three steps.

The first step is to sort the population in groups of neutrally dominant individuals. Each group represent a different Pareto front, formed by individuals that do not dominate each other. The first front is formed by the most dominant individuals of the population and they receive a rank value of one. Then a comparison is run among all individuals, and the rank number of them is increased by a unit every time another solution dominates it. By the end of this step there is a clear distinction of how dominant an individual is and to which front it belongs. The rank value determines how dominated an individual is, the lower the value the best it is. Individuals with the same rank number belong to the same front.

The second step is to calculate the crowding density to untie individuals with the same rank number. The basic principle is to measure how crowded the Pareto front is, and remove the individuals that are too close to their neighbours and belong to the same Pareto front (have the same rank value). By doing that, the solutions that are more distanced and separated from the others are preferred, resulting in a better approximation of the Pareto front and increasing the diversity of the individuals that form it.

The crowding density is calculated by Equation 11:

$$d_i = \begin{cases} \sum_{i=1}^{k} \log d_{ij}, & if\ \alpha = 1 \\ (1-\alpha)^{-1} \sum_{i=1}^{k} d_{ij}^{1-\alpha}, & otherwise, \end{cases} \tag{11}$$

where $d_{ij}$ is the Euclidean distance between individual $i$ and the $j$-th neighbour. As recommended in [9] if $\alpha$ is set to two, the crowding density becomes the harmonic distance calculated by the Equation 12:

$$d_i = \frac{1}{\frac{1}{d_{i,1}} + \frac{1}{d_{i,2}} + \cdots + \frac{1}{d_{i,k}}} \tag{12}$$

In *NSGA II* [24], the crowding density is calculated as the sum of the Euclidean distances, instead of the harmonic distance (using $\alpha$ equals to zero instead of two). The sum of distances was used because it was the best performing on the tests ran on the ZDT functions as shown in Section 3.6.2.

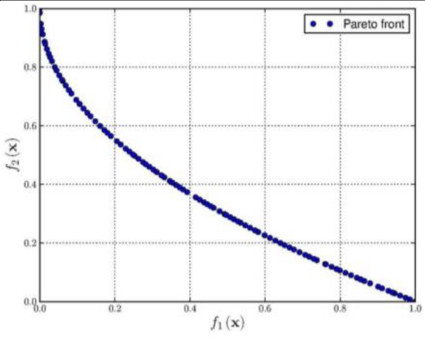
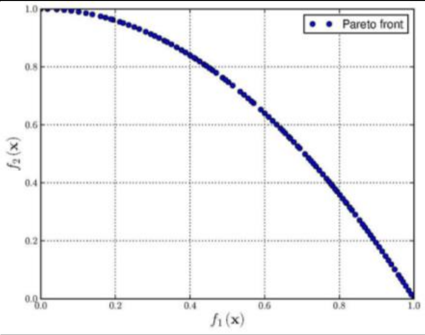
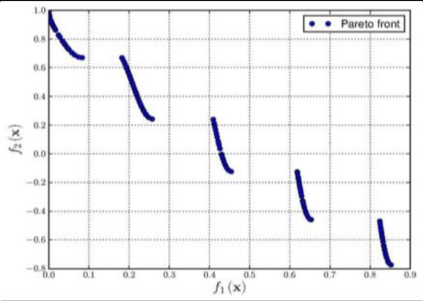
The third step consists of sorting the individuals first by rank, at this point, the NP worst individuals are removes and the ones that are parents are added to the archive to be used later in the mutation process. The individuals that have the same rank number are sorted by crowding density and the final selected individuals are the best NP individuals of the population.

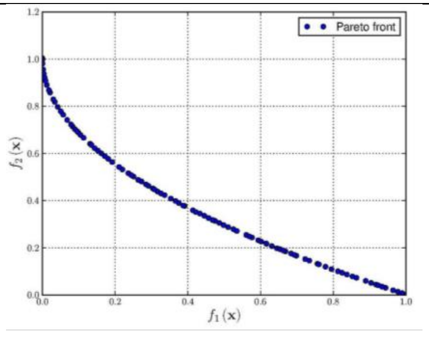
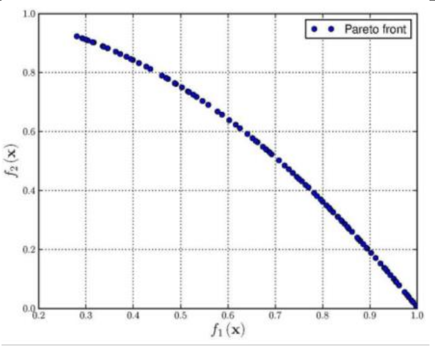
The mutation operation occurs in a very similar way in both mono and multi-objective cases. Since the recent inferior solutions are still capable of providing guidance towards the optimum, the archive stores the parents recently removed in the selection process [9]. Then the vector $\tilde{x}_{r2}$ is selected only from the archive and not from the union of the archive and the current population, given that the latter is unable to guide the process.

### 3.6.2. ZDT functions testing

The ZDT test functions [25] were used to verify the implementation of the multi-objective modification of *JADE*, since it was done by following the suggestions of [9] and afterwards modified to comply with these tests using the method from the *NSGA II* algorithm. The objective was to assess how good the Pareto front generated by *JADE* was. Table 4 shows information about each test functions [26]. Each of the functions was optimized for 30 variables.

*Table 4 – ZDT test functions information, the plots can be found at [27].*

| ***Test function*** | ***Formula*** | ***Pareto front*** |
|---|---|---|
| ZDT 1 | $f_1 = x_1,$ $f_2 = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}}\right],$ $g(x) = 1 + 9.\frac{\left(\sum_{j=2}^{N} x_j\right)}{N-1}$ |  |
| ZDT 2 | $f_1 = x_1,$ $f_2 = g(x)\left[1 - \left(\frac{x_1}{g(x)}\right)^2\right],$ $g(x) = 1 + 9.\frac{\left(\sum_{j=2}^{N} x_j\right)}{N-1}$ |  |
| ZDT 3 | $f_1 = x_1,$ $f_2 = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}} - \frac{x_1}{g(x)} sin(10\pi x_1)\right],$ $g(x) = 1 + 9.\frac{\left(\sum_{j=2}^{N} x_j\right)}{N-1}$ |  |

| | | |
|---|---|---|
| ZDT 4 | $f_1 = x_1,$ $f_2 = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}}\right],$ $g(x) = 1 + 10(N-1) + \sum_{j=2}^{N}[x_j^2 - 10\cos(4\pi x_j)]$ |  |
| ZDT 6 | $f_1 = 1 - e^{(-4x_1)}\,sin^6(6\pi x_1),$ $f_2 = g(x)\left[1 - \left(\frac{x_1}{g(x)}\right)^2\right],$ $g(x) = 1 + 9.\left[\frac{\left(\sum_{j=2}^{N} x_j\right)}{N-1}\right]^{\frac{1}{4}}$ |  |

The goal of the authors of [25] was to create a set of functions that would test problems like multimodality, deception, isolated optima, convexity or non-convexity, discreteness and non-uniformity. Each of these functions has some of these features in it [25]. The ZDT 5 was not optimized in this thesis since the deception, which is the main feature modelled in it, is not easily found in real problems and is not often used in benchmark papers.

Normally the tests are run several times and the final Pareto front is the result of all those runs. Since in this thesis the objective was only to verify the implementation and not to benchmark the algorithm, the optimizations were run only once, what does occur most of the times in real life usage of the platform, where the user does not want to run the process multiple times.

For ZDT 4, some modifications had to be done. The mean initial values of F and CR (mutation factor and crossover probability) were changed to 0.7 and 0.3 respectively. This was done to avoid being trapped in a local minimum. It is natural that for complex problems the control parameters of JADE need to be changed, the adaptation feature eliminates this need for most of the problems, but sometimes it is still needed. Since ZDT 4 is a multimodal problem, in several runs, the algorithm was trapped in a local minimum when using the default values of CR and F. The resultant Pareto front was of the same shape, but translated to higher values of the second objective.

The resultant Pareto fronts can be seen in Appendix B and they prove that the adaptation of the multi-objective capability on *JADE* was successful.

# 4. Test cases used

This section explains the selected test cases used to test the tool. They use different features of the tool and their goal is to test these features on real life usage.

## 4.1. Gipps traffic model calibration

Optimization can be useful to calibrate complex models. In this test case, the goal was to calibrate the parameters of a Gipps traffic model in order to minimize the speed and position errors of the simulated vehicle against measurement data.

The goal of Gibbs model is to predict the response of vehicles in a stream of traffic, the response of a vehicle is dependent on the vehicle in front of it in the stream. The variables modelled are the response time (time that it takes to take an action, $\tau$), the safety margin (clearance distance to the vehicle in front, $S_v$), and the maximum deceleration rates of the leader (the vehicle in front, $b_l$) and of the follower (the target vehicle, $b_f$) [28]. Table 5 shows the ranges used in the optimization for each of these variables.

*Table 5 – Ranges of the variables used in the optimization.*

| Variable | Range |
|---|---|
| Response time, $\boldsymbol{\tau}$ | [0.3, 15] s |
| Safety margin, $\mathbf{S_v}$ | [3, 50] m |
| Deceleration of leader, $\mathbf{b_l}$ | [-15, -0.4] m/s$^2$ |
| Deceleration of follower, $\mathbf{b_f}$ | [-10, -0.1] m/s$^2$ |

The model was coded as differential equations in Matlab® and its output are the position, the speed and the acceleration of a vehicle following a traffic stream. Four different methods to solve this differential equation were coded in different files. The methods used were, Euler Implicit, Euler Explicit, Midpoint and Treiber (Euler explicit for speed and midpoint for position) [29].

A design constraint was used to make sure that the results would be strictly real. For that, the argument of the square roots of the model was set not to be lower than zero.

Additionally, calibration files storing calibration data of the follower and leader vehicles were initialized for different time steps in the beginning of each process. Different time steps affect the outcome of the calibration.

To set up the problem to be run in the platform, the user must adapt the script that runs the simulation (if it already exist). Then, code the design constraint and the cost functions in separate files. After, input the variables that shall be optimized with their information and ranges. Finally indicate the calibration file that has the measurement data in different time steps. All this process was done quite fast and in an easy way.

This optimization was done in four steps. In all those, the goal was to minimize the RMS (root mean square) error in position and speed of the simulated vehicle and the measurement.

In the first step, only the four basic variables described in Table 5 were optimized. All optimizers were compared, and to be fair, the scalarization method (see Section 2.4) was applied to deal with the two objectives. The solver used was Euler implicit and the time step was set to be 1 second.

The second step performs the same optimization as the first step, but using JADE multi-objective implementation.

Then for the third step the different measurement files recorded with different time step were used and a time step variable was implemented that allows to choose which measurement file should be used in the calibration, this variable is discrete and can assume the values of 0.01, 0.1, 0.5 and 1 second. This was done as an alternative of running an optimization for each time step value. The solver used was the Euler implicit.

In the fourth step, it was added another discrete variable that describes which solver should be used to solve the differential equations. The final problem had four continuous variables and two discrete. This was done to avoid running an optimization for each combination of time step and solver type, since the interaction between these variables is unknown.

This optimization tested the multi-objective, the calibration and the mixed-integer capabilities of the tool. The results are presented in Section 5.1.

## 4.2. ZF gear-shifting optimization

This test case was an attempt to test the limits of the tool, instead of fast running Matlab® scripts or simple Simulink® models, GSP complete vehicle simulations were used for the evaluation, since with them, it was possible to capture the correct behaviour of gear shifting. The complexity also increases considerably from the previous cases and many features of the tool were explored.

The objective was to optimize fuel consumption and performance of a bus in a cycle by optimizing the gearshift maps of its automated gearbox. The target vehicle was an urban bus and has a six-speed gearbox manufactured by ZF.

GSP models were used to evaluate the target vehicle regarding fuel consumption and performance. Each simulation using a GSP model takes about 2 minutes to complete, then it was vital to minimize the number of evaluation to make the process feasible time wise.

Ten maps were selected to be optimized, each of them representing a possible gearshift e.g. from first to second, from second to third, all the way to sixth gear and downshifts also, from sixth to fifth all the way back to first gear. These maps have 18 coordinates (6x3) that contains the threshold speed to shift (which was the lowest speed required) in total 180 variables were optimized.

The cycle evaluated was a city cycle, which was believed to be the most representative cycle of the target vehicles application. The representativeness of the cycle is of extreme importance, given that the result of the optimization are only be optimal for cycles similar to this.

The optimization needed to be constrained to make sure that the results are feasible. This was done by implementing performance constraints and a standard constraint. The performance constraints certified that the target vehicle had the same gradeability and acceleration as the reference vehicle. All these constraints were calculated based on the performance cycle, if the candidate calibration does not comply with these performance constraints, the main city cycle is not simulated, the unfeasible individual in this case goes through the process described in Section 3.3.3 and receives a penalty value not to be selected in the process as feasible.

A standard constraint was applied to the number of gearshifts per 100 kilometres to ensure the durability of the gearbox.

To set up the problem the process was similar to the previous case, but with some peculiarities regarding the GSP application. The user must indicate which model to be used to evaluate the calibrations alongside the performance cycle and the target cycle for the calibration (the cycle that measures the fuel consumption and the performance). Then, the constraint and cost function files had to be coded, in each of them the constraints had to be derived from the results of the simulations. The 180 variables had to be defined and set with proper ranges.

Since the problem consists of two objectives, *JADE* algorithm was used to generate a Pareto front, in this way the result is a set of feasible calibrations, each of them representing a compromise between fuel consumption and performance. The performance was measured by the average speed on the cycle and the goal was to maximize it. The fuel consumption was calculated as litres per 100 kilometres and the goal was to minimize it.

# 5. Results

This section shows the results of the test cases and discusses them in detail.

## 5.1. Traffic model calibration results

This section shows the results of the traffic simulation calibration. The methods used for this optimization are described in Section 4.1. The set-up of the problem was much faster and easier that the traditional method of coding the whole problem and connecting it to the optimization algorithm.

### 5.1.1. Step1: Basic continuous variables and weighted objective

In the first step, the four basic variables were optimized using all the optimizers available in the platform. The objectives were averaged with equal weights to simplify the problem (*scalarization*), but their values are shown separately for comparison. Table 6 shows the results for each of those optimizers and the number of evaluations that was required to achieve this result.

*Table 6 – First case results of the traffic model calibration.*

| ***Algorithm*** | ***Response time (s)*** | ***Safety margin (m)*** | ***b*** | ***$\hat{b}$*** | ***Position error*** | ***Speed error*** | ***Number of evaluations*** |
|---|---|---|---|---|---|---|---|
| ***Lean gradient*** | 0,97 | 3,92 | -10,47 | -3,47 | 3,17 | 0,9 | 270 |
| ***Linear DoE*** | 2,11 | 3,01 | -2,81 | -3,63 | 4,08 | 1,08 | 350 |
| ***Fmincon*** | 7,65 | 3 | -0,64 | -10 | 14,44 | 2,52 | 44 |
| ***Pattern Search*** | 7,65 | 3 | -0,4 | -1,74 | 10,18 | 1,82 | 539 |
| ***JADE*** | 1,04 | 3 | -2,73 | -1,75 | **2,09** | **0,54** | **1100** |

Both *fmincon* and *pattern search* performed very poorly, the calibration error of the model was very high for both of them indicating that they could have been possibly stuck in a local minimum. *JADE* shows the best result, but it comes with a high price, it takes 1100 evaluations to achieve it. *Lean gradient* shows good results, better cost function value than *Linear DoE* with less number of function evaluations. Since the evaluations were not costly (less than a second) the trade-off tends in the direction of *JADE*.

If a Pareto is wanted as a result, the best-recommended algorithm would be *JADE*, since it achieves better results and can generate a Pareto front in a single run (since it is a proper multi-objective method and it is a population-based algorithm). *Lean gradient* would require several runs to generate a Pareto front and the result would not be as good as for *JADE* given that *Lean gradient* does not find cost function values as low as *JADE*, then the result would be a Pareto front higher in objective space.

To support the statement that *fmincon* and *pattern search* were stuck in a local minima, plots of the cost function were generated close to the optimum point found by those algorithms. Since the representation of the cost function is five-dimensional, a simplification was made. The function was plotted for each variable being varied one at each time, keeping the others constant. The result is similar as if the function was “sliced” in a certain dimension. This gives a picture of how does the function look like in each dimension. The plots for all the optimizers can be found in Appendix C1-C5.

Figure 14 shows the plot for the response time being varied for the fmincon optimizer, where the first plot is the position error and the second is the speed error. The function during the optimization only receives the averaged value and does not have the knowledge of each of the objectives (since the scalarization method is being used), and then the optimum point found should be the one that minimizes both objectives the most.

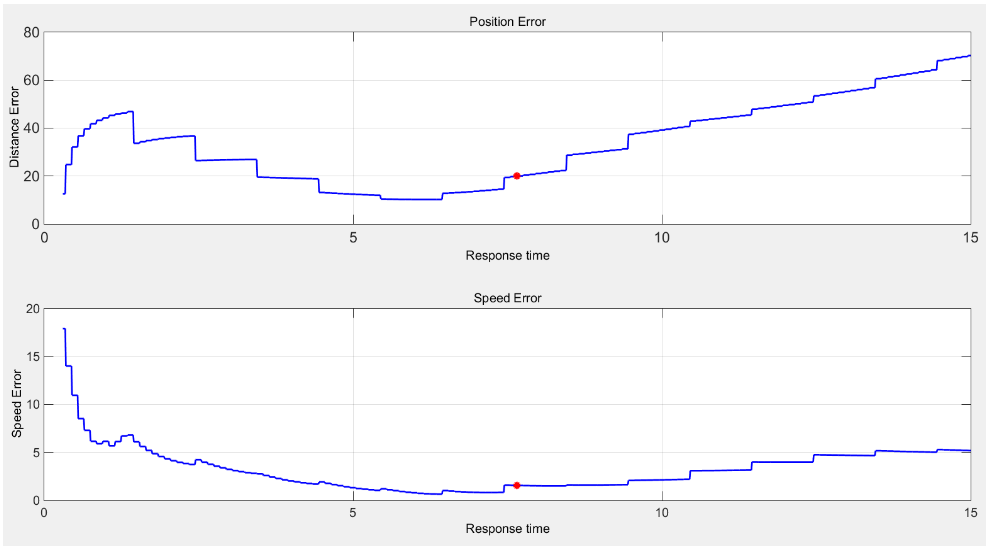


*Figure 14 - Plot of fmincon results, response time being varied and the other variables kept constant.*

Figure 15 shows the same plot as in Figure 14, but using the values found by *JADE* optimizer, this picture shows how these functions (distance and speed error) changed depending on the location, it also shows that for this variable the best point found was truly optimal for both objectives. The landscape for the response time suggests a stair-shaped function that resembles a step function.

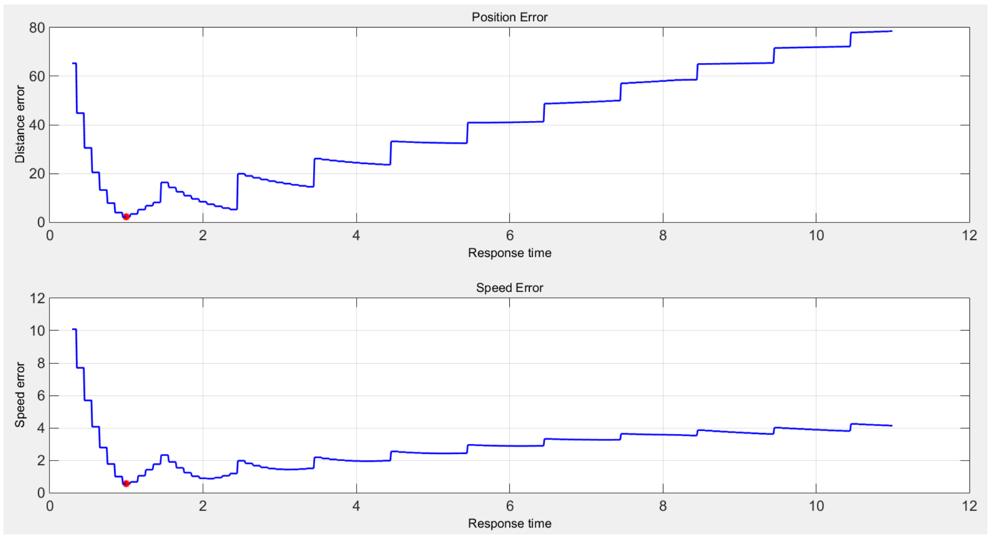


*Figure 15 - Plot of JADE results, response time being varied and the other variables kept constant.*

The topography of the cost function changes according to where the function was "sliced", each of the optimum points found shows a different topography of the function and gives an idea of why some solvers became stuck in local minima.

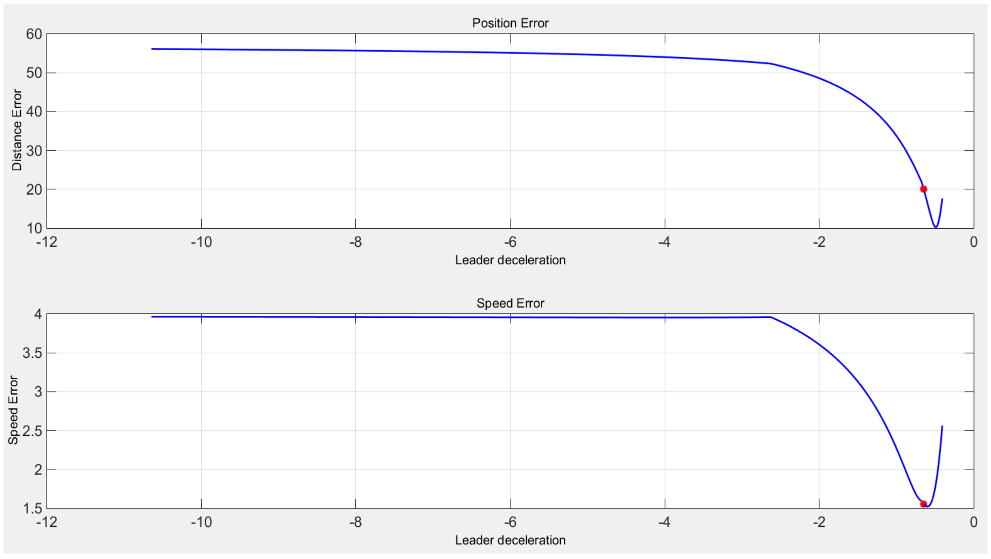


*Figure 16 - Plot of fmincon results, leader deceleration being varied and the other variables kept constant.*

Figure 16 shows the landscape of the function for the leader deceleration in the optimum found by *fmincon*, while Figure 17 shows the same graph but in the location of the optimum found by *JADE*. The levels of the error were much lower for *JADE*, and the landscape was very different.

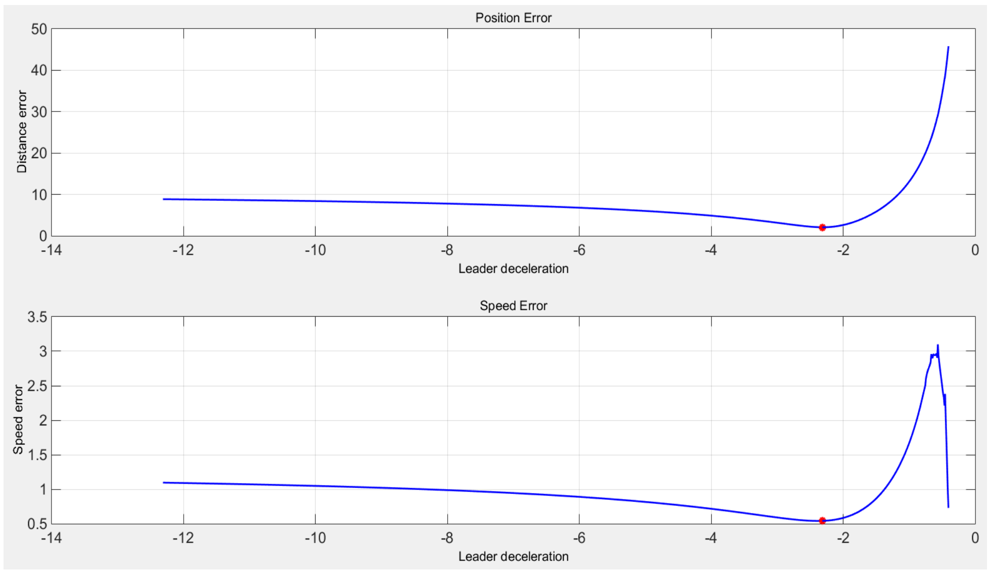


*Figure 17 - Plot of JADE results, leader deceleration being varied and the other variables kept constant.*

For the follower deceleration, it looks like a valley with very low gradient, which is close to the behaviour of Rosenbrock test function (see Appendix A6), as shown in Figure 18.

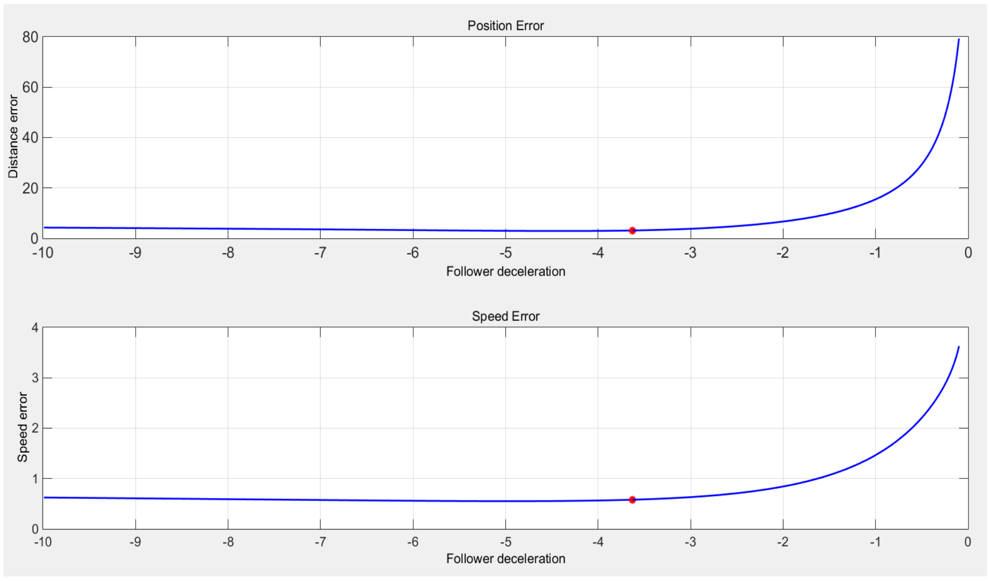


*Figure 18 - Plot of* Linear DoE *results, follower deceleration being varied and the other variables kept constant.*

By analysing all these graphs, it was possible to conclude that the problem is very complex and multi-modal, given that has many local minima. This explains why *Fmincon* did not achieve good results, but does not explain why *Pattern Search* performed so poorly. Additionally the behaviour of the function for each variable resembles the functions used on the benchmark in Section 3.5, justifying their choice.

### 5.1.2. Step 2: Basic continuous variables with multi-objective approach

For the second step of this optimization, *JADE* algorithm was used as proper multi-objective method. The goal was to generate a Pareto front for the case optimized in Section 5.1.1. The benefit of having a Pareto front as a result is that the user can choose the compromise between the objectives.

The resultant Pareto front can be seen in Figure 19. The plot on the left is the full Pareto front and the plot on the right is the part of the Pareto that has the best trade-off objective values.

The left graph shows clearly a stair shaped Pareto front, and this can be explained by the graphs in Appendix C5, in those graphs it is possible to see that the function is very stair shaped and because the minimum for one of the objectives is not always located in the same place as for the other objective.

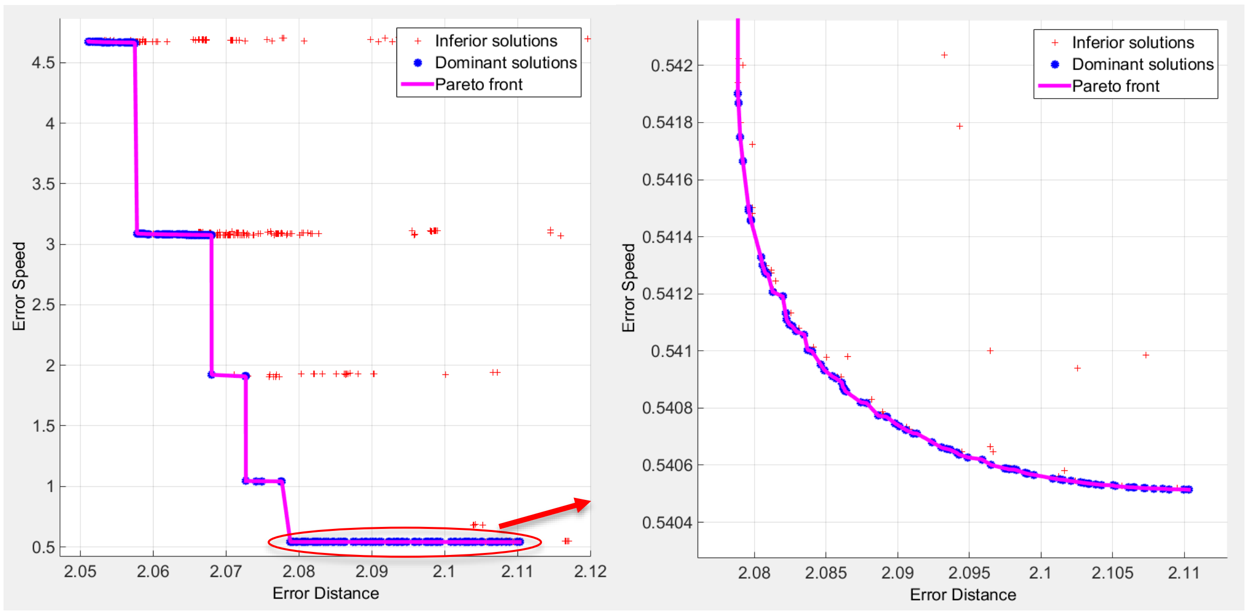


*Figure 19 – Pareto front for the second step of the optimization.*

Each of the smaller fronts in Figure 19 corresponds to a different level of the function's steps, not necessarily being worse than the main front, but representing a different set of compromise (minimizing the error in distance more than the error in speed). The zoomed region of the best compromise front (the one circled in red) is shown in the right figure. The best compromise in this case was when both objectives were minimized the most (with equal preference).

The following results of the next steps are only show the interest region, since it was of interest to minimize both errors as much as possible, but all of them showed the same stair-shaped Pareto fronts.

### 5.1.3. Step 3: Time step as discrete variable

In the third step, the time step variable was added to the problem, depending on its value the process used a different calibration file recorded with the respective time step for the calibration.

The resultant Pareto front of this step is shown in Figure 20. It is possible to see that in the interest region the fronts are the same as in the previous step, given that all the members of the Pareto used time step of 1-second.

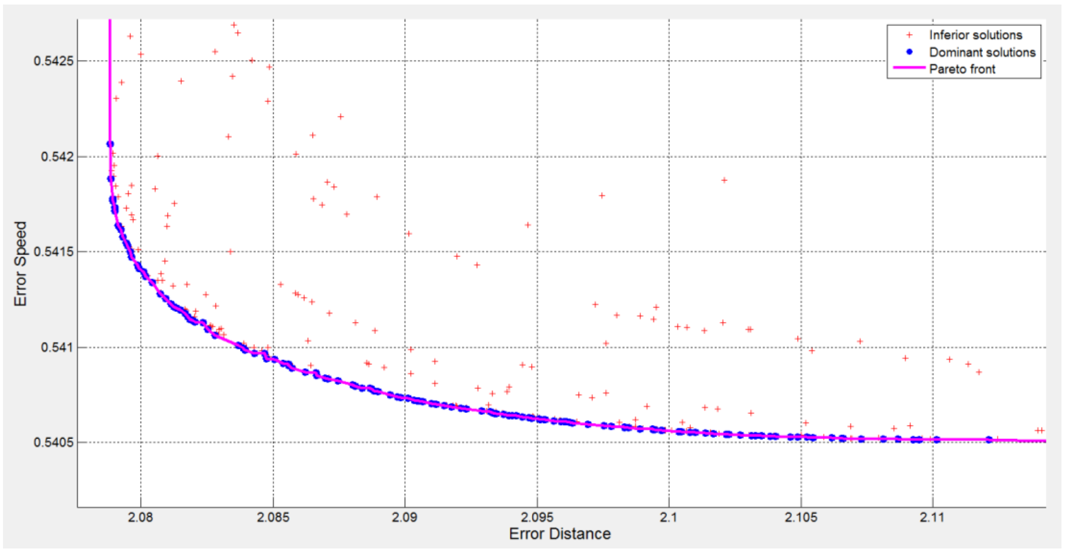


*Figure 20 - Pareto front for Step 3 of the traffic model calibration, where the time step was treated as a variable.*

If comparing the time of running all the time steps separately, coding the time step as a discrete variable resulted in less optimization time overall. The lower time step the higher the calculation time is. Then comparing to the optimization fully run on low time step, considering also that it takes more time to converge, it was still faster to run with the variable coded as discrete, since not all the evaluations were run with the low time step and the convergence occurs at high time step which was faster to be evaluated.

In Appendix D, the Pareto fronts are shown for each time step separately. Figure 21 shows all these graphs in a single picture for comparison purposes. It is possible to see again that the best time step for calibration was the 1-second, reinforcing the results found in the third step of this optimization process.

The results concur to literature. According to [30], larger time steps are better for the calibration, and the same trend was shown in the results, tough this relationship was not linear as shown in Figure 21. The time step of 0.5 second presented worse calibration results than the step of 0.01 and 0.1 second.

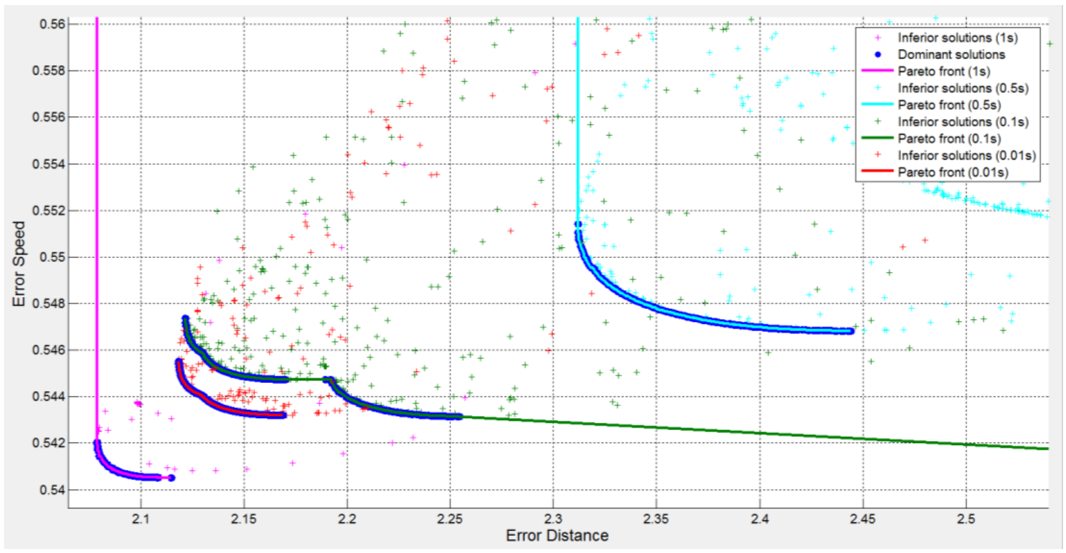


*Figure 21 – Pareto front for each of the available time steps, the solver user was Euler implicit.*

### 5.1.4. Step 4: Time step and solver type as discrete variables

For the last step, the solver type was added as discrete variable. The solver type indicates which of the available solvers should be used to solve the differential equations of the Gipps model. Again, *JADE* was used, since the objective was to generate a Pareto front for the case. The result of the main front can be seen in Figure 22.

As seen in the previous section, all the points that form the main front used the time step of 1-second and in this case, the majority of the front used the Euler implicit solver type.

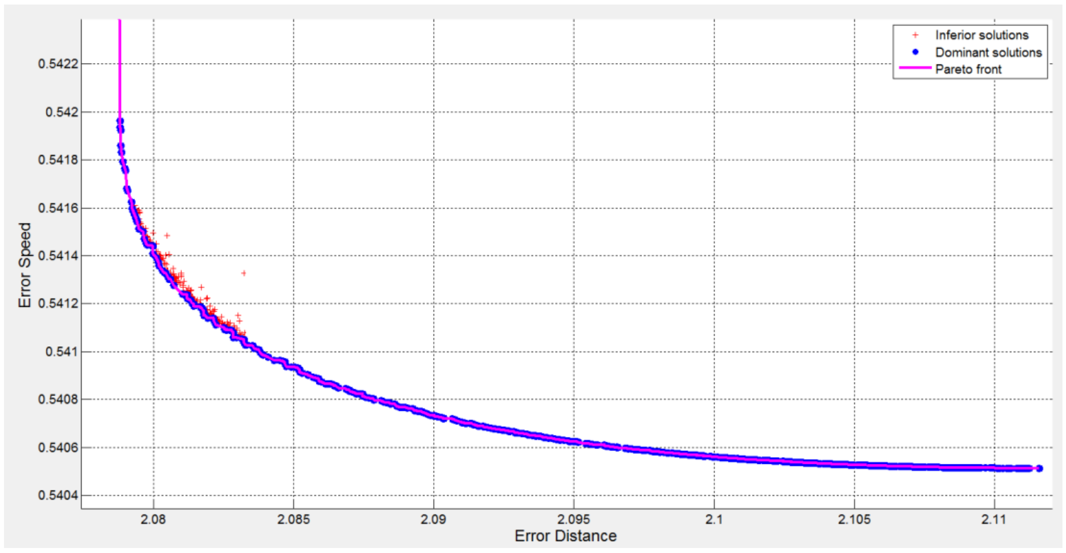


*Figure 22 - Pareto front for Step 4 of the traffic model calibration, where the solver type and the time step where treated as variables.*

To understand the effect of each solver in the problem, four different optimizations each with a different solver type were run. The results are shown in Figure 23.

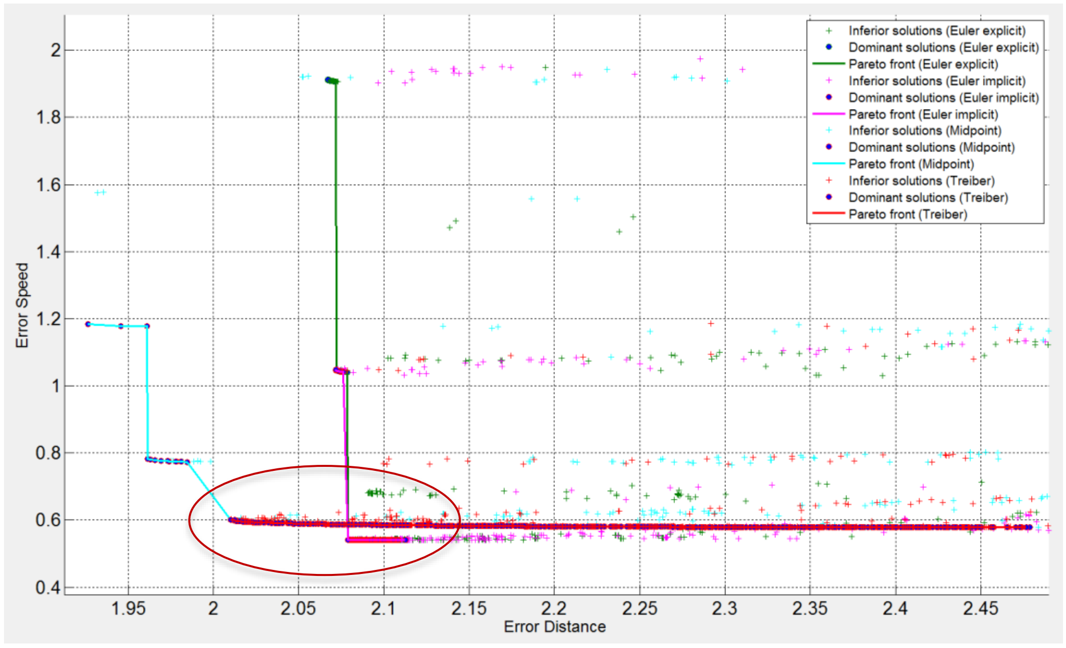


*Figure 23 – Pareto fronts for each solver types.*

Each of the four different curves is a Pareto front of a solver. It is evident that some of the solvers improved one objective more than the other, e.g. the Midpoint solver was the best to improve the error in distance while it stays behind Euler implicit and explicit when it comes to minimize the speed error. Since the focus was on minimizing both objectives at the same time, the region of interest is the one highlighted with a red circle in Figure 23, and it is showed in Figure 24 in a better way.

Figure 24 shows that the methods Treiber and Midpoint have similar results in the interest region, the same happens for Euler implicit and explicit method. The Euler implicit and explicit methods were more successful in minimizing the speed error than Midpoint or Treiber, but the latter were better to minimize the distance error.

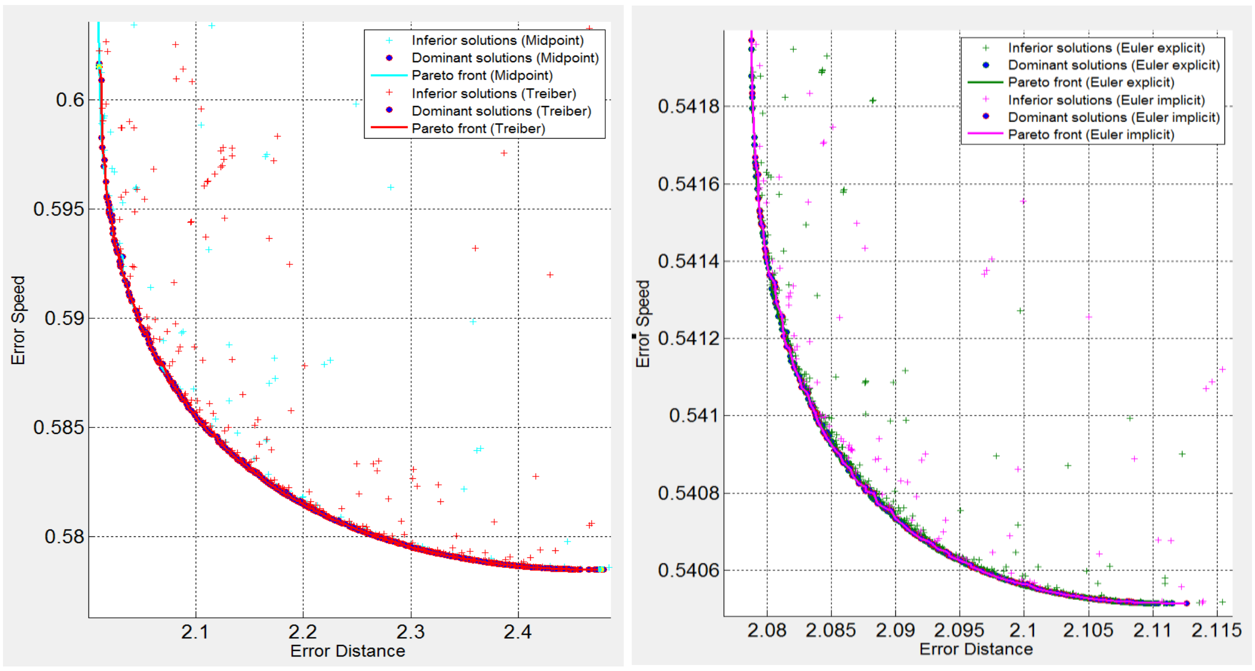


*Figure 24 – Pareto fronts of the interest regions for the traffic model calibration for different solvers.*

Running the optimization with both the time step and the solver type as discrete variables gave the best results in a single run discarding the need of running multiple times with different combinations of these variables.

## 5.2. ZF gearbox optimization

This section provides the results from the ZF gearbox optimization test case. For confidentiality reasons, no absolute values were shown in this section.

The method used to set-up the problem was successful and intuitive. It was much simpler than coding the problem and interfaces between GSP and the optimization algorithm.

As described in Section 4.2, *JADE* algorithm was used to generate a Pareto front for this case. This is shown in Figure 25 where each point represents the result of fuel consumption and performance of a calibration generated by the optimizer. On the right part of the graph are located the calibrations that resulted in higher average speed, and in the bottom, the ones resulting in lower fuel consumption. The black point represents the reference calibration from ZF and the blue points are the final solutions that form the Pareto front.

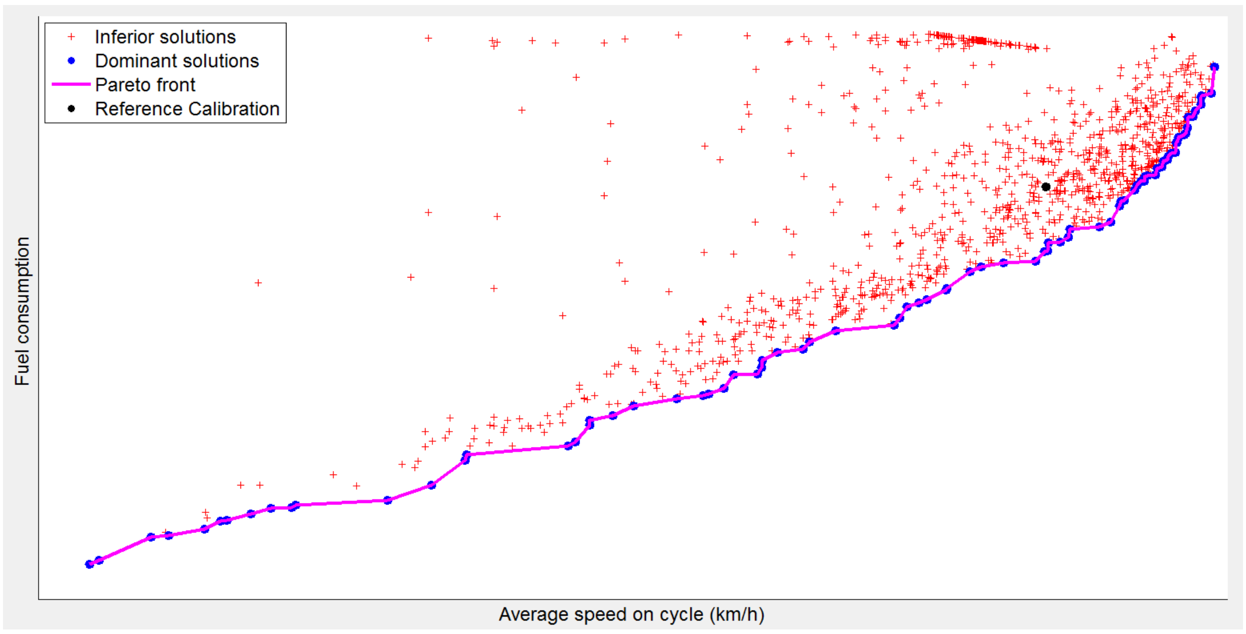


*Figure 25 – Pareto front of fuel consumption and performance for the ZF gearshift optimization.*

It is possible to see that the front advanced passed the values of the reference calibration, which means that the result was improved both in performance and in fuel consumption. The Pareto front was generated in 104 steps. By the end, the process generated a calibration that achieved 1.4% fuel consumption reduction with the same performance if compared to the reference calibration. Another calibration improved the performance by 0.4% with same fuel consumption.

All the individuals in the front comply with the constraints, which mean that they have at least the same gradeability and acceleration as the reference. In addition, the calibrations did show less than 10% increase in gearshift number in the city cycle.

Since *JADE* is a meta-heuristic method that seeks an improvement rather than finding the true optimum, then it is up to the user to decide when the process should be terminated. A great way to know if no more improvements are possible is to see if the variables values have converged. By the end of the process, looking at the values of each variable, it was realised that the process did not yet converge and that greater improvements could be expected if the process was continued.

It is important to point out that GSP models do not evaluate several criteria that are necessary to be taken into account to determine a successful calibration. Some of these criteria are noise, vibration and comfort. Therefore, the process described in this thesis does not eliminate the need of vehicle testing, but it certainly makes the process much faster by suggesting a calibration that is close to be optimal.

# 6. Conclusions

The goals set for the platform were accomplished. The platform can handle mixed-integer multi-objective optimization and can use applications like Matlab® scripts, simple Simulink® models and GSP models to evaluate the design variables.

The test cases used to test and develop the multiple functionalities of the tool proved the versatility of the platform to fit to multiple needs and perform the most various analyses.

For the traffic model calibration, both the multi-objective features of the tool, the ability to perform mixed-integer optimization and the usage of design constraints, where proven to work and to generate high quality results in a reasonable time. Additionally, the coding of the time step and the solver type as discrete variables proved to be more time saving than the traditional brute force method (running all combinations of discrete variables values).

For the ZF gearbox optimization, the ability to handle very high dimensionality problems, performance constraints and multi-objective optimization were proven to work well and to improve the current calibration in a reasonable time given the complexity of the problem.

The benchmark of the algorithms showed that *JADE* was the most robust method, since it solved all problems and it is considered the main algorithm of the tool. *Linear DoE* and *Lean Gradient* should be used when the cost function is not complex (smooth and convex), but when each evaluation of it is time demanding, since these solvers would use less function evaluations than JADE.

The CEC 2014 test functions [23] proved to test very relevant features that occur on real test cases situations as shown in the traffic model calibration. An algorithm that performs well in all of them has higher chance of performing well in real test case usage. A future user of this platform should rely on Table 1 to choose the correct algorithm to be used in a problem, if not enough information about the problem exists.

The platform simplified considerably the optimization process, since it requires only a couple of scripts from the user in order to run. In the other hand, the results of the optimization still have to be critically analysed. The optimization algorithm has no information about the problem except the cost function and constraints. If those are not properly defined the results shall not be optimal or feasible, then it is vital that the user must understand the problem that is being optimized.

# 7. Future work

Several improvements could be done in the platform. The implementation of more algorithms could be one, especially to cover some features that are not being covered by the selected algorithms. A good suggestion is to add a proper mixed integer algorithm to improve convergence speed in problems that have both types of variables. Another would be to add algorithms specific to solve the linear programming problems that sometimes appear as requests at Volvo AB.

The tool does not provide means to perform inner loop (online) optimization, the topology generation is also not performed by this tool, and both of these features could be added to this platform to make it more complete.

For very complex problems like the ZF gearbox optimization, an alternative approach could be used, which consists of starting the process with JADE algorithm and after the potentially good calibrations are found, stop the process and use one the points of the Pareto front as a starting point for *fmincon*. JADE does most of the hard work of finding the good region in the search space, but to converge to the minimum itself takes time, then by using *fmincon* this would be faster, since it does use the gradient information and guarantee that the solution is in the minimum and not around it.

## 8. Bibliography

[1] L. a. A. S. Guzzella, Vehicle propulsion systems, Berlin Heidelberg: Springer-Verlag, 2013.

[2] E. H. T. M. N. E. P. &. S. M. Silvas, "Review of Optimization Strategies for System-Level Design in Hybrid Electric Vehicles.," 2016.

[3] E. B. E. H. T. &. S. M. Silvas, "Comparison of bi-level optimization frameworks for sizing and control of a hybrid electric vehicle.," *IEEE Vehicle Power and Propulsion Conference (VPPC),* pp. 1-6, October 2014.

[4] K. A. Diest, Numerical Methods for Metamaterial Design, Netherlands: Springer, 2013.

[5] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal,* vol. 7, no. 4, pp. 308-313, 1965.

[6] C. Audet and D. J. E. John, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on optimization,* vol. 17, no. 1, pp. 188-201, 2006.

[7] F. Vitaliy, Differential evolution–in search of solutions, New York: Springer, 2006.

[8] A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing. Vol. 53, Heidelberg: Springer, 2003.

[9] J. Zhang and A. C. Sanderson, Adaptive differential evolution: a robust approach to multimodal problem optimization, vol. Vol. 1, Springer Science & Business Media, 2009.

[10] C. A. Coello, D. A. Van Veldhuizen and G. B. Lamont, Evolutionary algorithms for solving multi-objective problems. Vol 242., New York: Kluwer Academic, 2002.

[11] A. Abraham and J. Lakhmi, Evolutionay Multiobjective Optimization, London: Springer, 2005.

[12] P. P. a. D. Wilde, Principles of Optimal Design: Modeling, Cambridge University Press; 2 edition, 2000.

[13] J. M. Bader, "Hypervolume-based search for multiobjective optimization: theory and methods.," *No. 112. Johannes Bader,* 2010.

[14] G. Wenzhong and S. H. Porandla, "Design Optimization of a Parallel Hybrid Electric Powertrain," in

*2005 IEEE Vehicle Power and Propulsion Conference*, 2005.

[15] D. C. Montgomery, Design of analysis of experiments, John Wiley & Sons, Inc., 2013.

[16] S. Ahlinder, "Personal Communication," 2016.

[17] "fmincon," The MathWorks Inc., [Online]. Available: https://se.mathworks.com/help/optim/ug/fmincon.html. [Accessed 03 December 2016].

[18] "Choosing the Algorithm," The MathWorks Inc, [Online]. Available: https://se.mathworks.com/help/optim/ug/choosing-the-algorithm.html. [Accessed 03 December 2016].

[19] J. Zhang and A. C. Sanderson, "JADE: adaptive differential evolution with optional external archive.," *IEEE transactions on evolutionary computation.,* vol. 13, no. 5, pp. 945-958, 2009.

[20] K. Li, J. Li, Y. Liu and A. Catiglione, "A Novel Differential Evolution Algorithm Based on JADE for Constrained Optimization," *Computational Intelligence and Intelligent Systems: 7th Internation Symposium,* vol. 575, pp. 84-94, 2016.

[21] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art.," *Computer methods in applied engineering,* vol. 191, no. 11, pp. 1245-1287, 2002.

[22] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering,* vol. 186, no. 2, pp. 311-338, 2000.

[23] P. N. Suganthan, "CEC 2014," [Online]. Available: http://web.mysites.ntu.edu.sg/epnsugan/PublicSite/Shared%20Documents/Forms/AllItems.aspx. [Accessed 2016].

[24] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation 6.2,* pp. 182-197, 2002.

[25] E. Zitzler, K. Deb and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results.," *Evolutionary Computation,* vol. 8, no. 2, pp. 173-195, 2000.

[26] N. Chase, M. Rademacher and R. Goodman, "A benchmark study of multi-objective optimization methods," *BMK-3021,* vol. 6, no. 09, 2009.

[27] "Wikipedia: Test functions for optimization," Wikimedia Foundation Inc., 7 October 2016. [Online]. Available: https://en.wikipedia.org/wiki/Test_functions_for_optimization. [Accessed 27 November 2016].

[28] P. G. Gipps, "A behavioural car-following model for computer simulation," *Transportation Research Part B: Methodological,* vol. 15, no. 2, pp. 105-111, 1981.

[29] A. Kesting and M. Treiber, Traffic Flow Dynamics: Data, Models and Simulation., Berlin: Springer, 2013.

[30] M. Treiber and A. Kesting, "Microscopic calibration and validation of car-following models - a systematic approach," *Procedia-Social and Behavioral Sciences,* vol. 80, pp. 922-939, 2013.

[31] S. Surjanovic and D. Bingham, "Virtual Library of Simulation Experiments: Test Functions and Datasets," Simon Fraser University, January 2015. [Online]. Available: https://www.sfu.ca/~ssurjano/optimization.html. [Accessed 27th November 2016].

[32] S. G. Johnson, "NLopt," 2014. [Online]. Available: http://ab-initio.mit.edu/wiki/index.php/NLopt#Download_and_installation. [Accessed 2016].

[33] M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in optimization and numerical analysis (pp. 51-67)*, Oaxaca, Mexico, 1994.

[34] L. Sean, Essentials of metaheuristics., Lulu: Department of Computer Science, George Manson Univerity, 2009.

[35] "Electrical Power System Analysis & Nature inspired Optimization Algorithms.," 13 January 2015. [Online]. Available: http://al-roomi.org/benchmarks/unconstrained/n-dimensions/177-hyper-elipsoid-function. [Accessed 27th Novermber 2016].

[36] J. Cattin, "Personal Communication," 2016.

# 9. Appendix A: Plots of the CEC 2014 test functions for two dimensions.

A1 : Plot of Griewank test function.

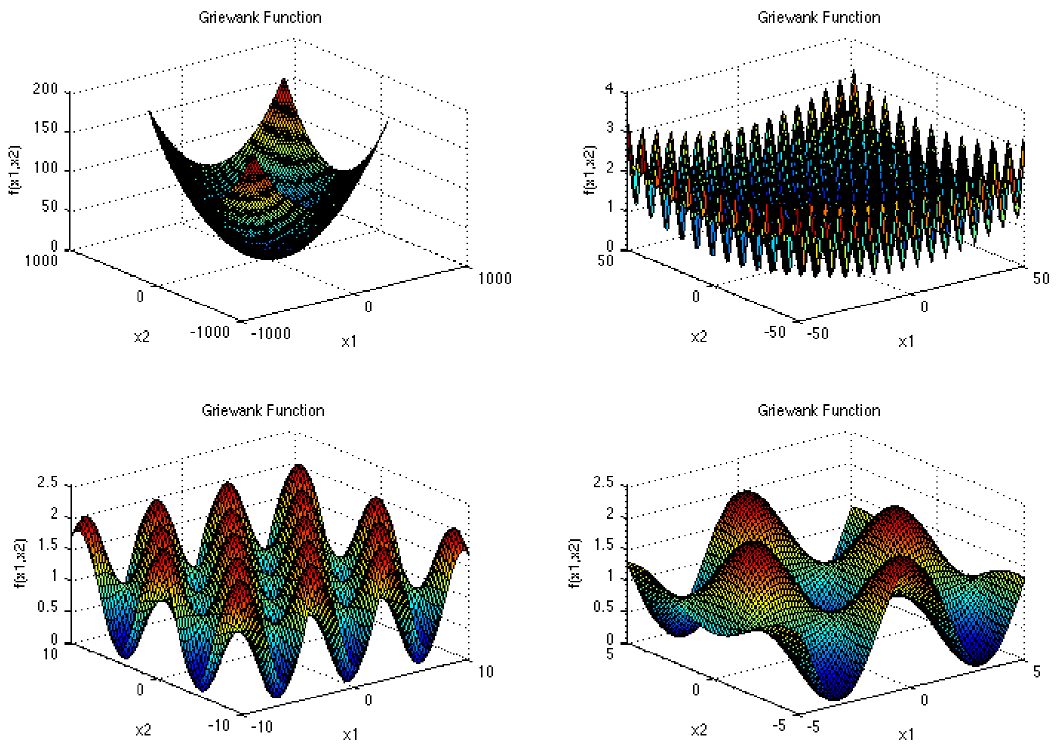


*Figure 26 – Plots of Griewank test function in different ranges. From the top left graph to bottom right, the range is zoomed in to show the function close to the optimum [31].*

A2: Plot of Ackley test function.

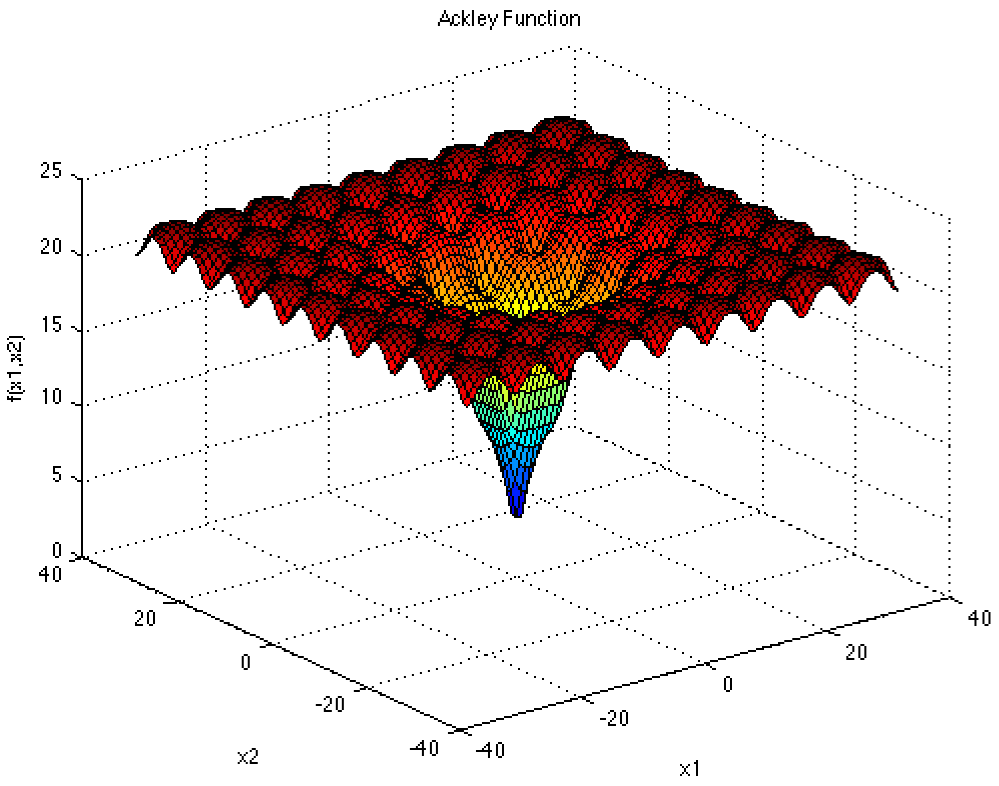


*Figure 27 – Plot of Ackley test function [31].*

A3: Plot of Rastrigin test function

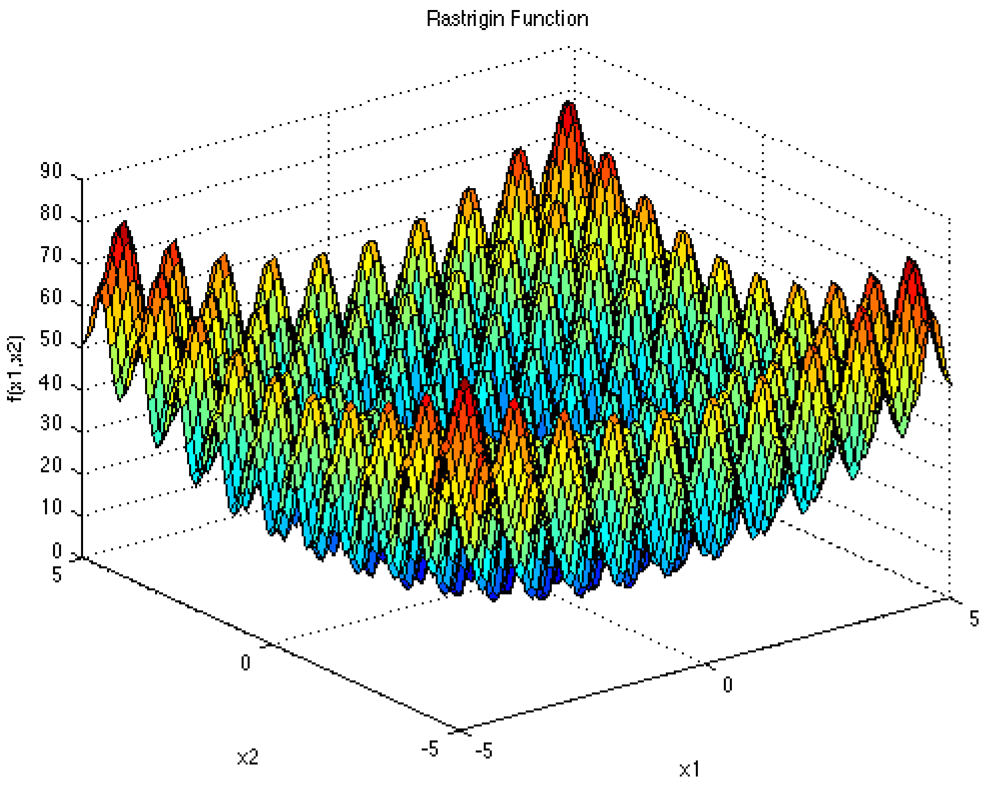


*Figure 28 – Plot of Rastrigin test function [31].*

A4: Plot of Rotated Hyper-Ellipsoid test function.

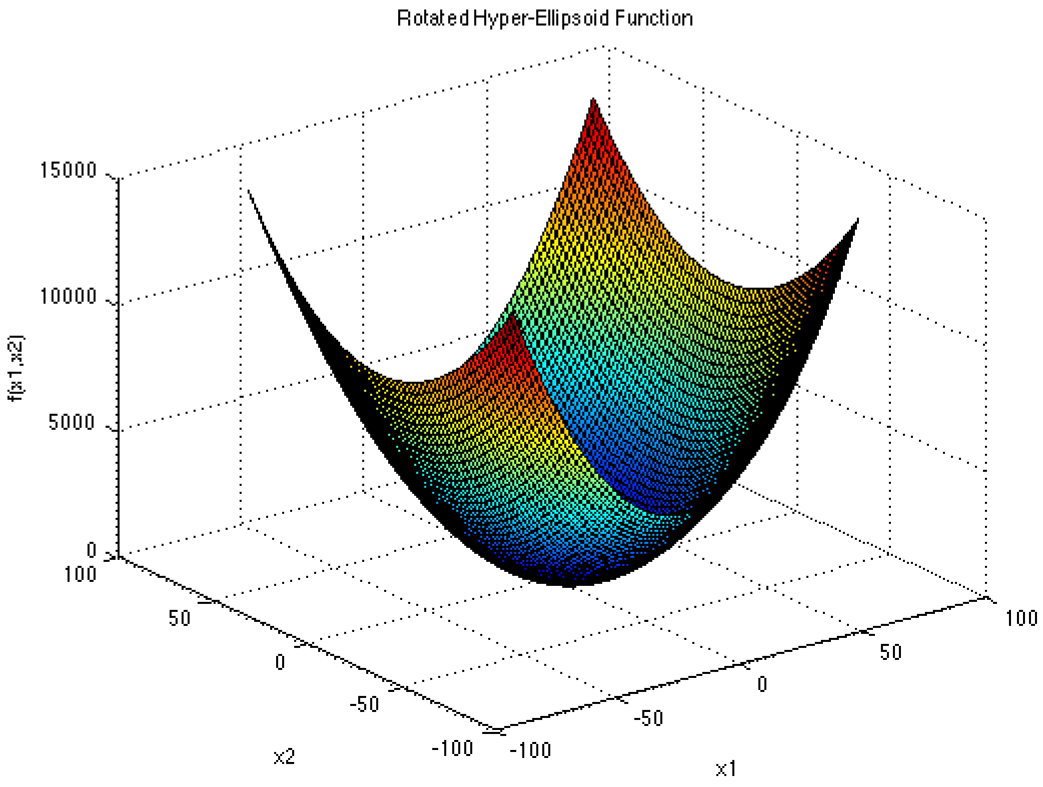


*Figure 29 – Plot of Rotated Hyper-Ellipsoid test function [31].*

A5: Plot of Sphere test function.

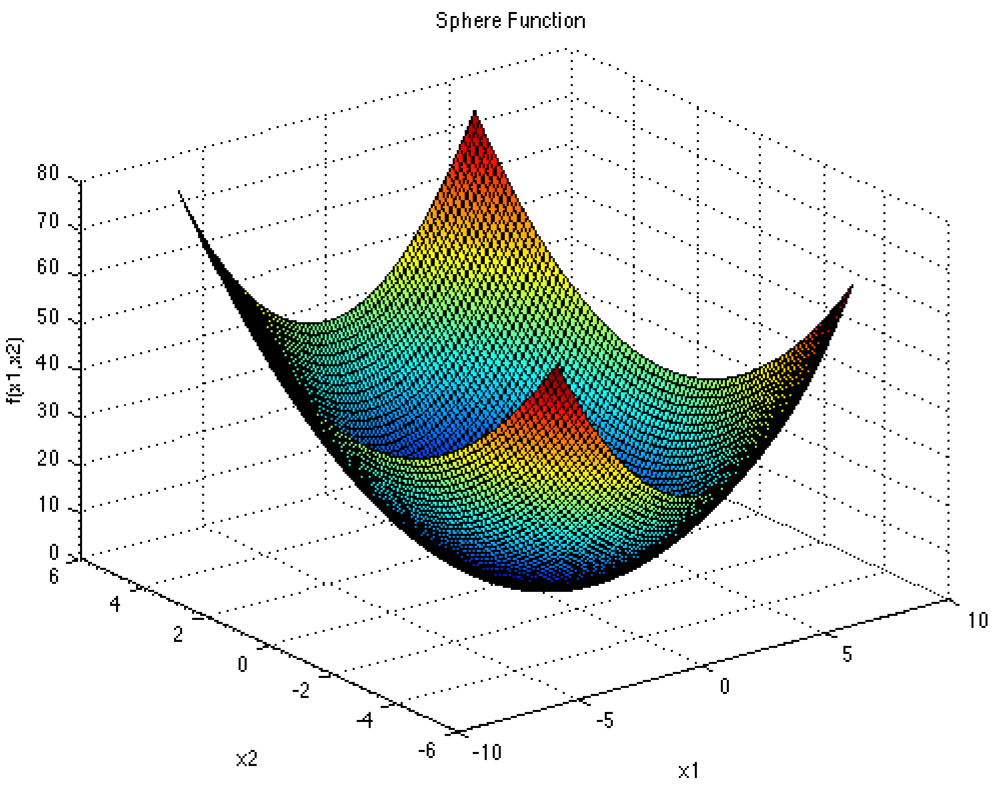


*Figure 30 - Plot of Sphere test function [31].*

A6: Plot of Rosenbrock test function.

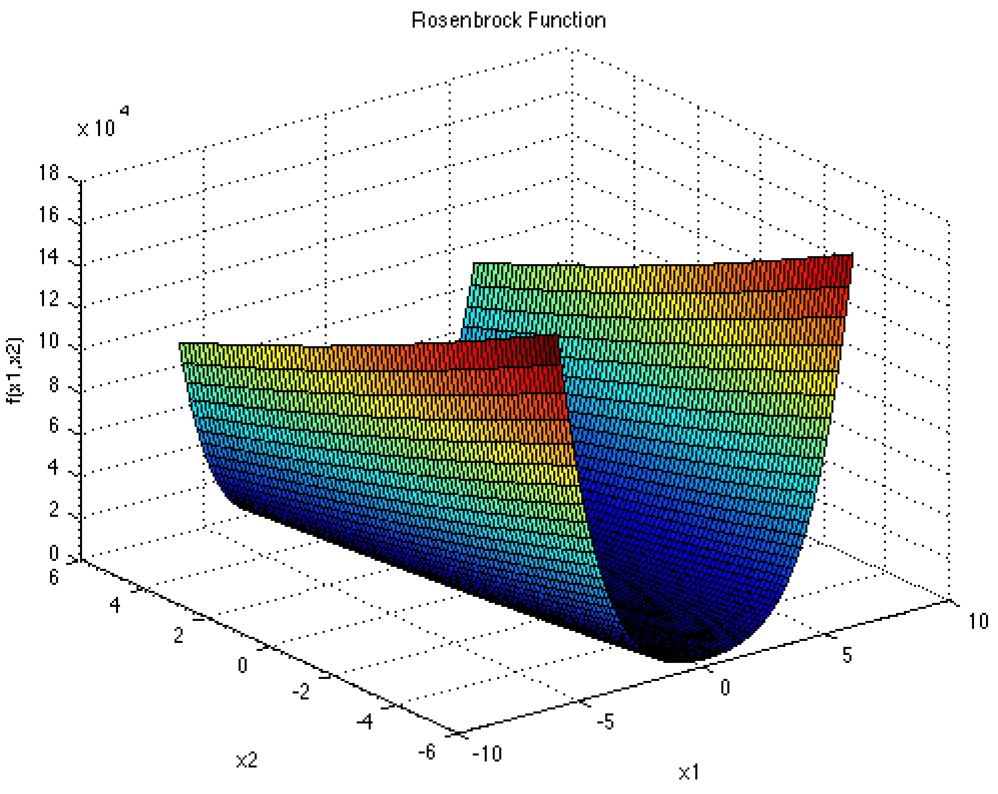


*Figure 31 – Plot of Rosenbrock test function [31].*

A7: Plot of Ellipsoid test function.

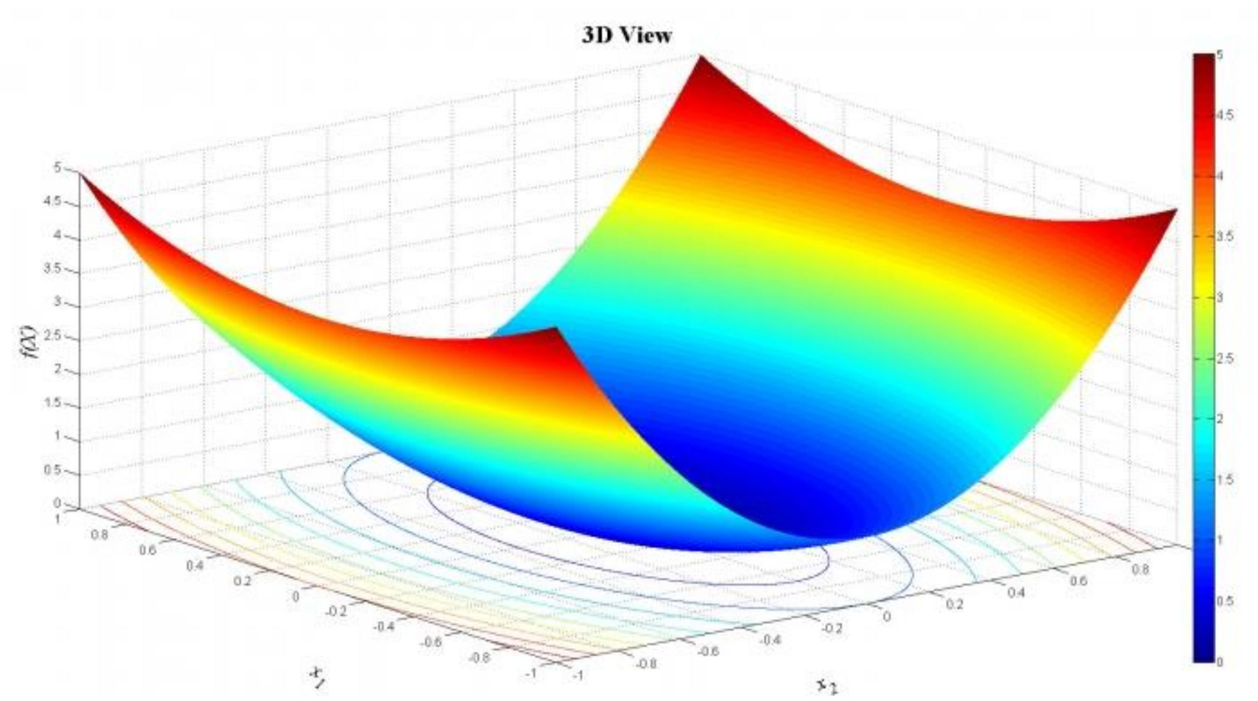


*Figure 32 – Plot of Ellipsoid test function [35].*

## 11. Appendix B: ZDT Pareto fronts obtained by *JADE*

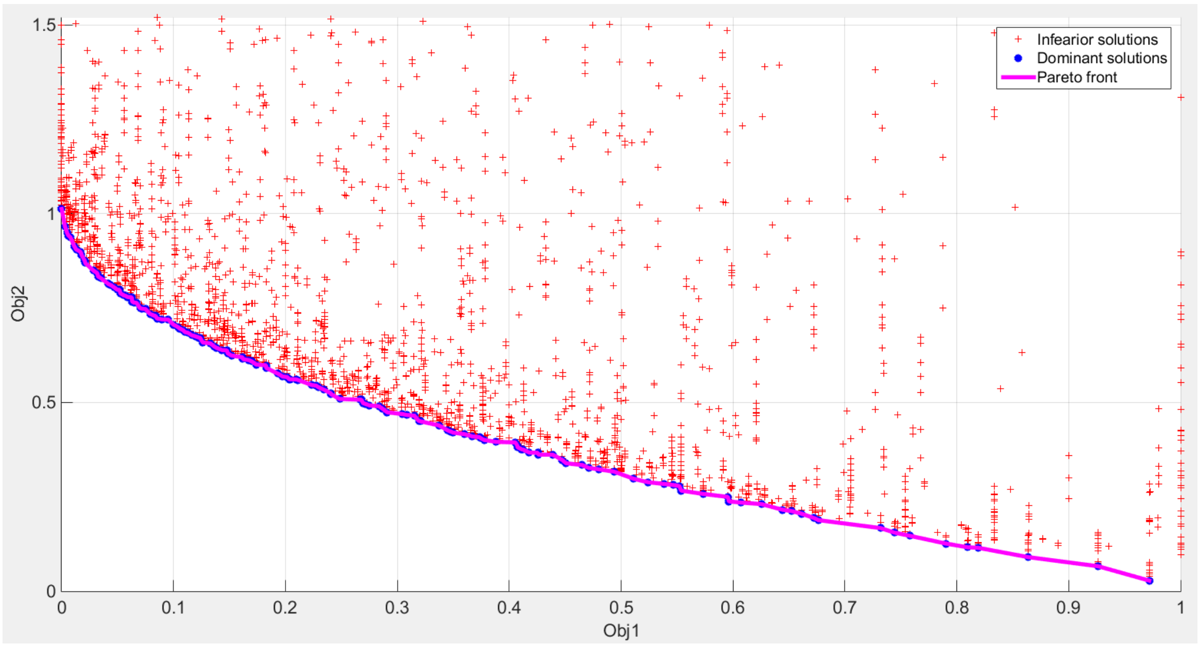


*Figure 33 – Pareto front of ZDT 1 function generated by JADE.*

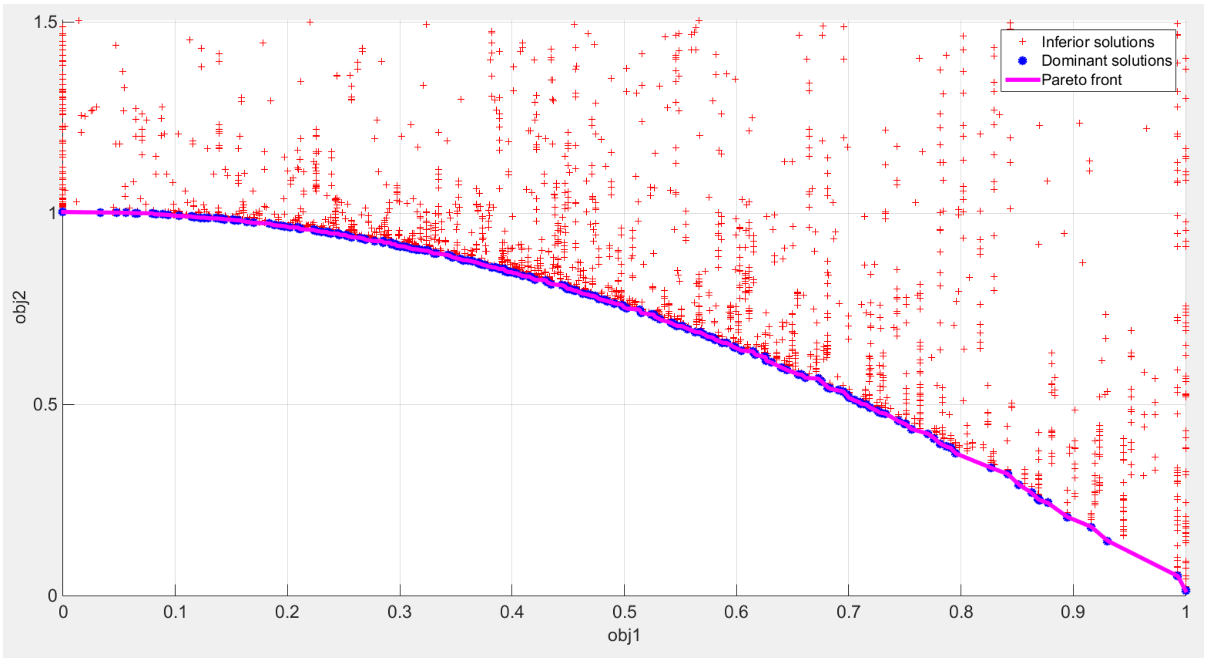


*Figure 34 – Pareto front of ZDT 2 function generated by JADE.*

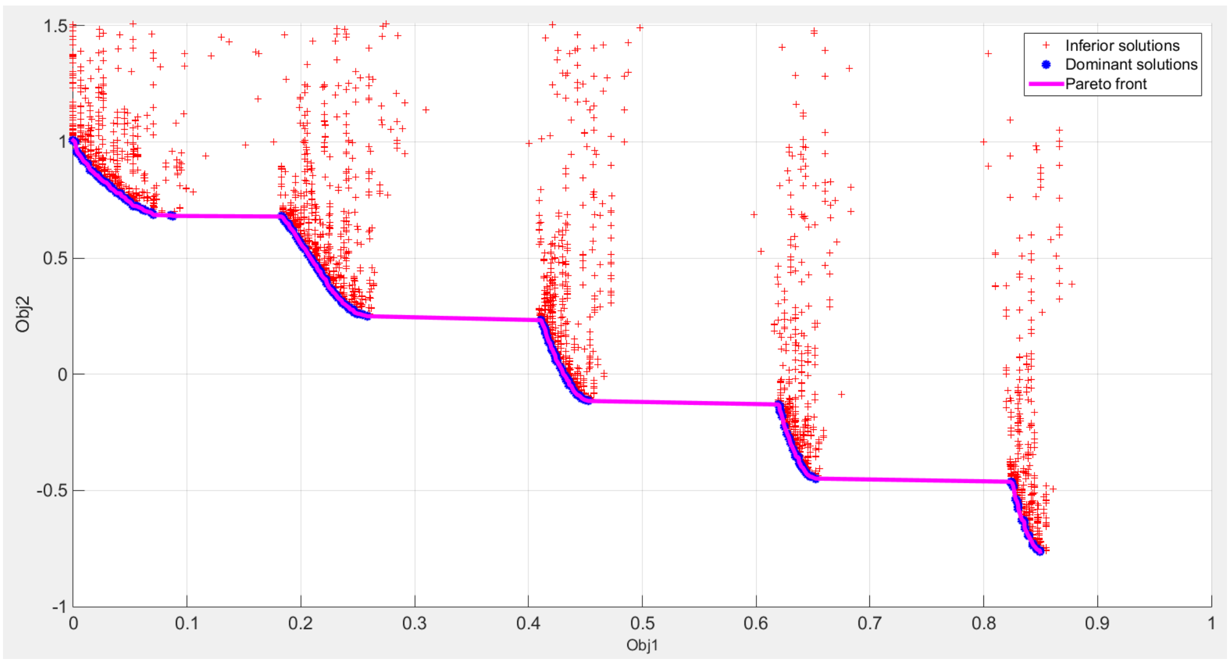


*Figure 35 - Pareto front of ZDT 3 function generated by JADE.*

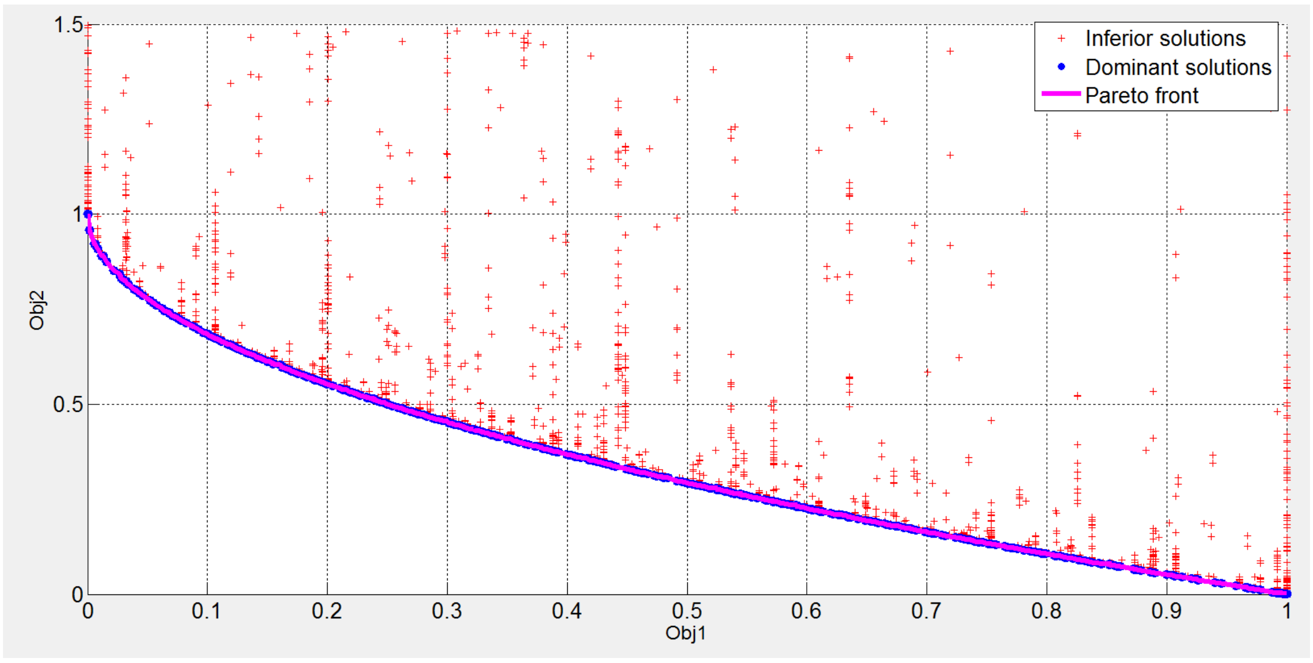


*Figure 36 – Pareto front of ZDT 4 function generated by JADE.*

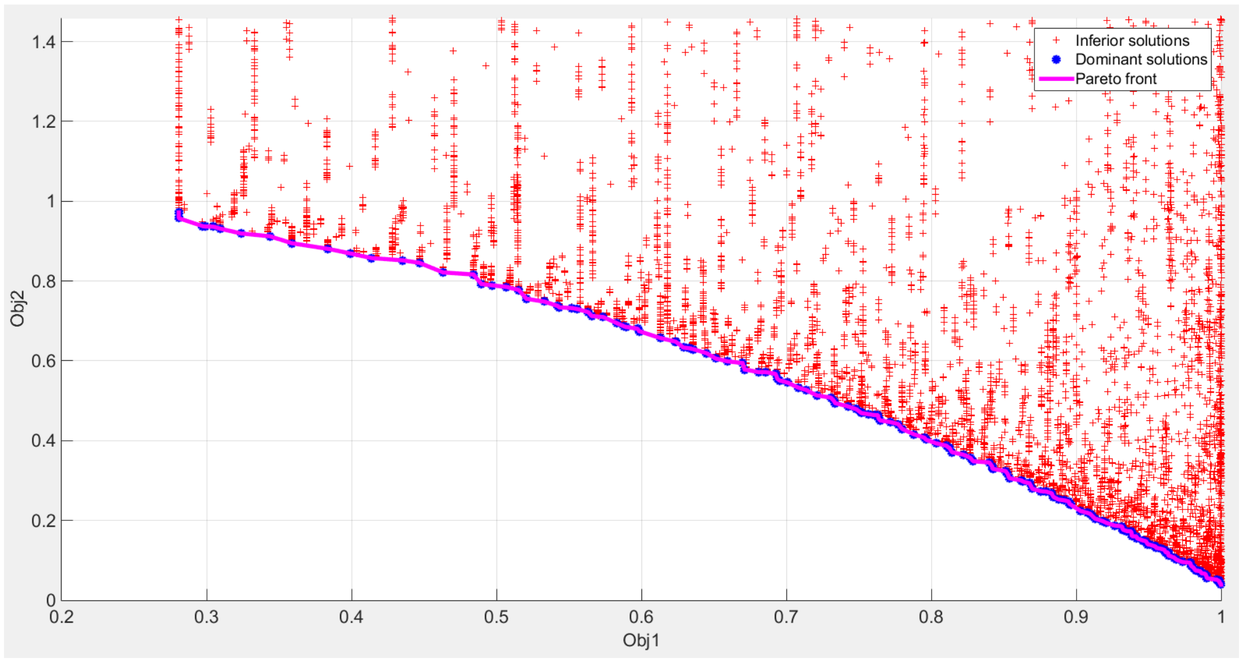


*Figure 37 – Pareto front of ZDT 6 generated by JADE.*

## 12. Appendix C: Plots around the optimum points found by the algorithms in the traffic model calibration.

C1: *Lean Gradient* plot for response time being varied and the other variables kept constant.

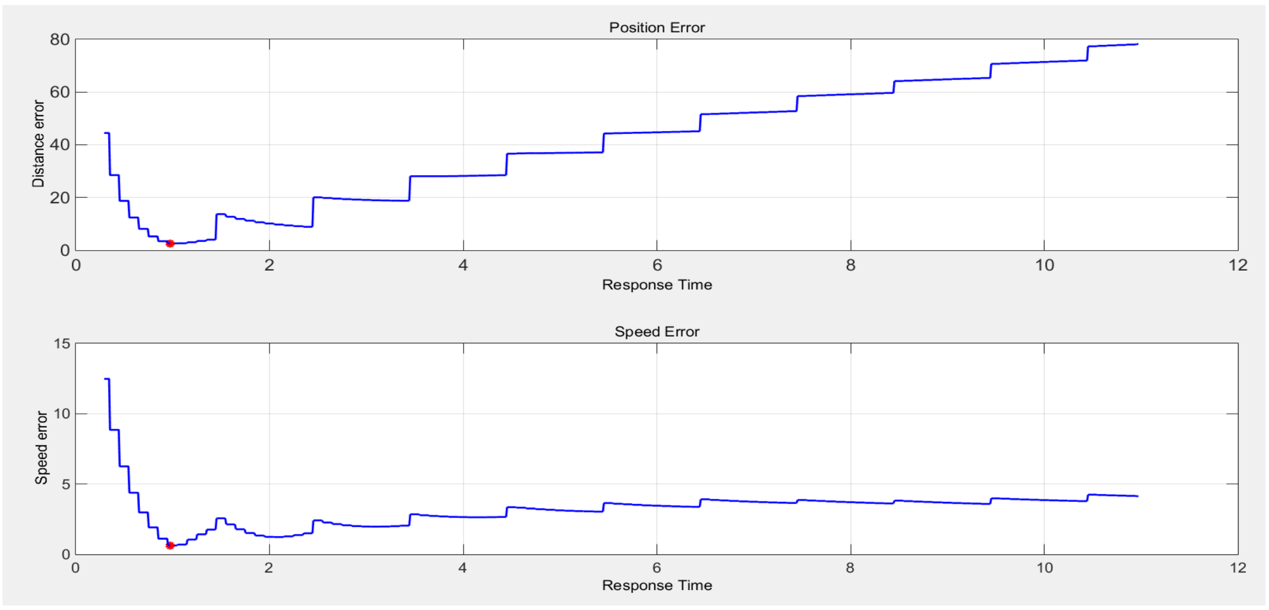


*Figure 38 – Plot of Lean Gradient results, response time being varied and the other variables kept constant.*

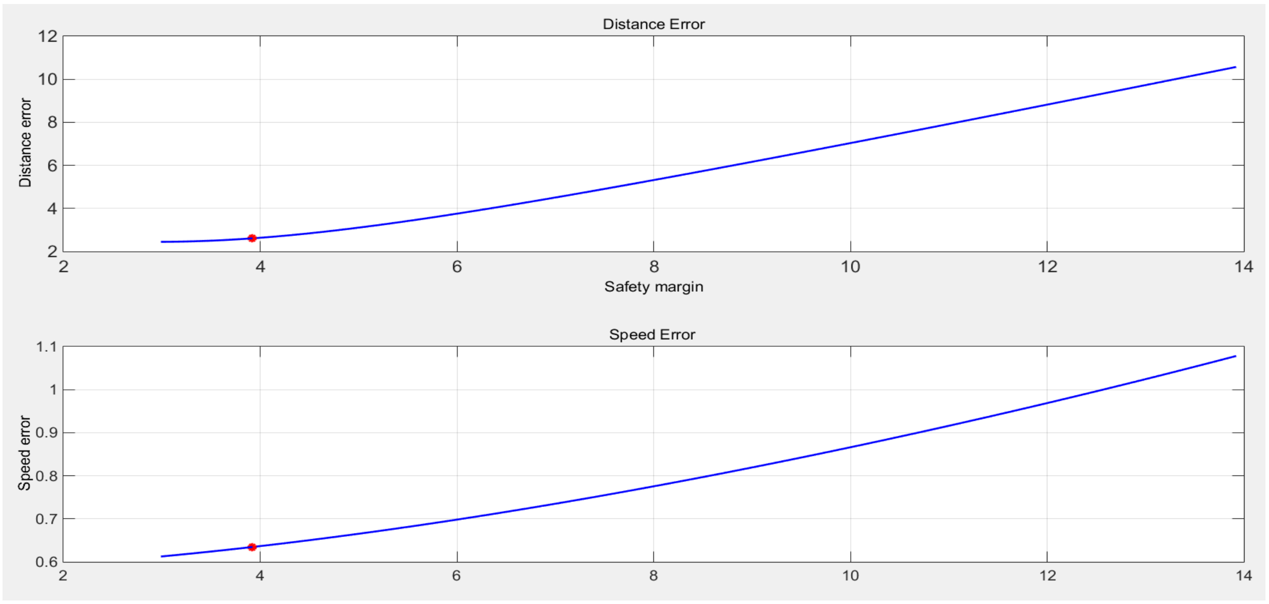


*Figure 39- Plot of Lean Gradient results, safety margin being varied and the other variables kept constant.*

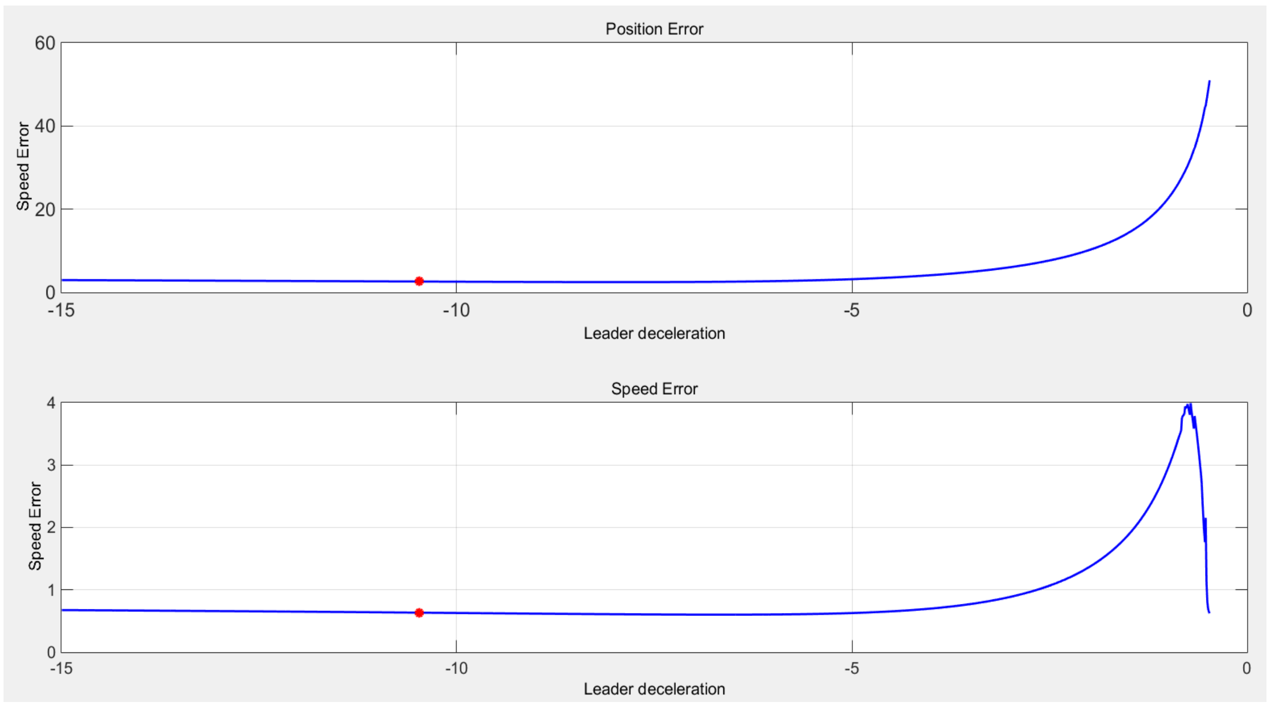


*Figure 40 - Plot of Lean Gradient results, leader deceleration being varied and the other variables kept constant.*

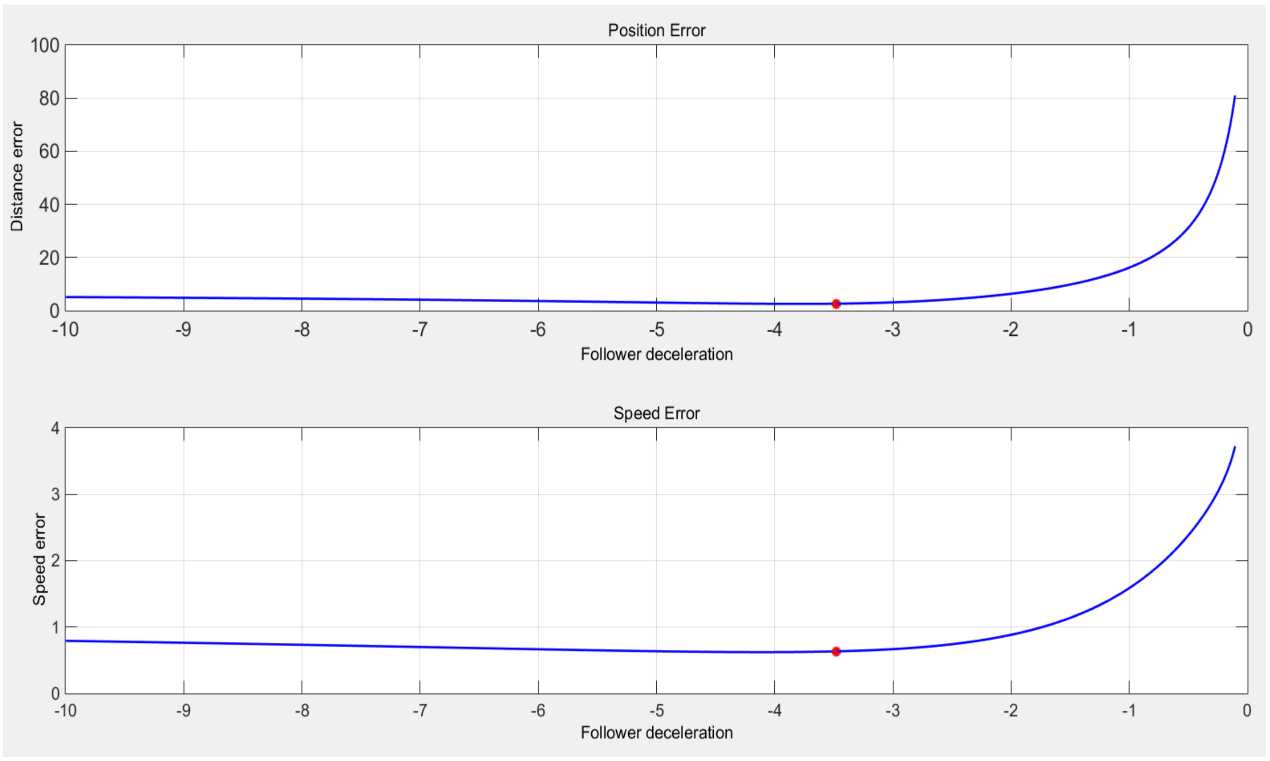


*Figure 41 - Plot of Lean Gradient results, follower deceleration being varied and the other variables kept constant.*

C2: *Linear DoE* plot for response time being varied and the others kept constant.

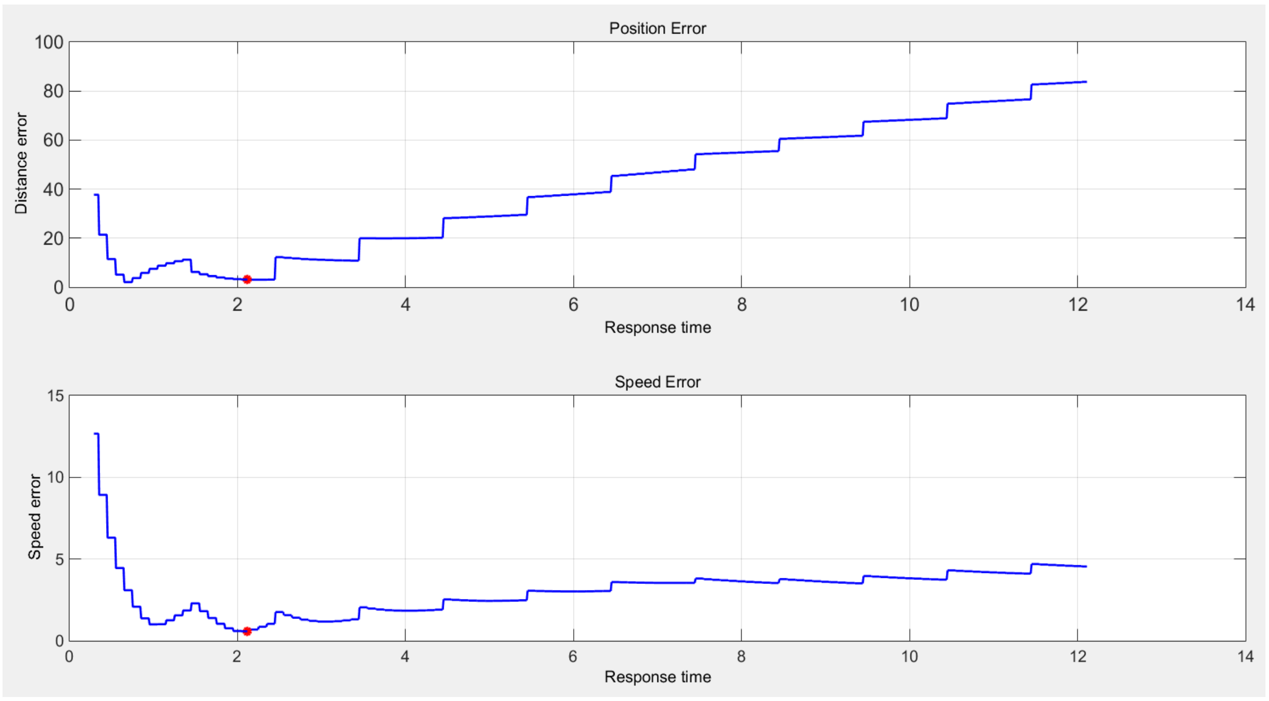


*Figure 42 - Plot of Linear DoE results, response time being varied and the other variables kept constant.*

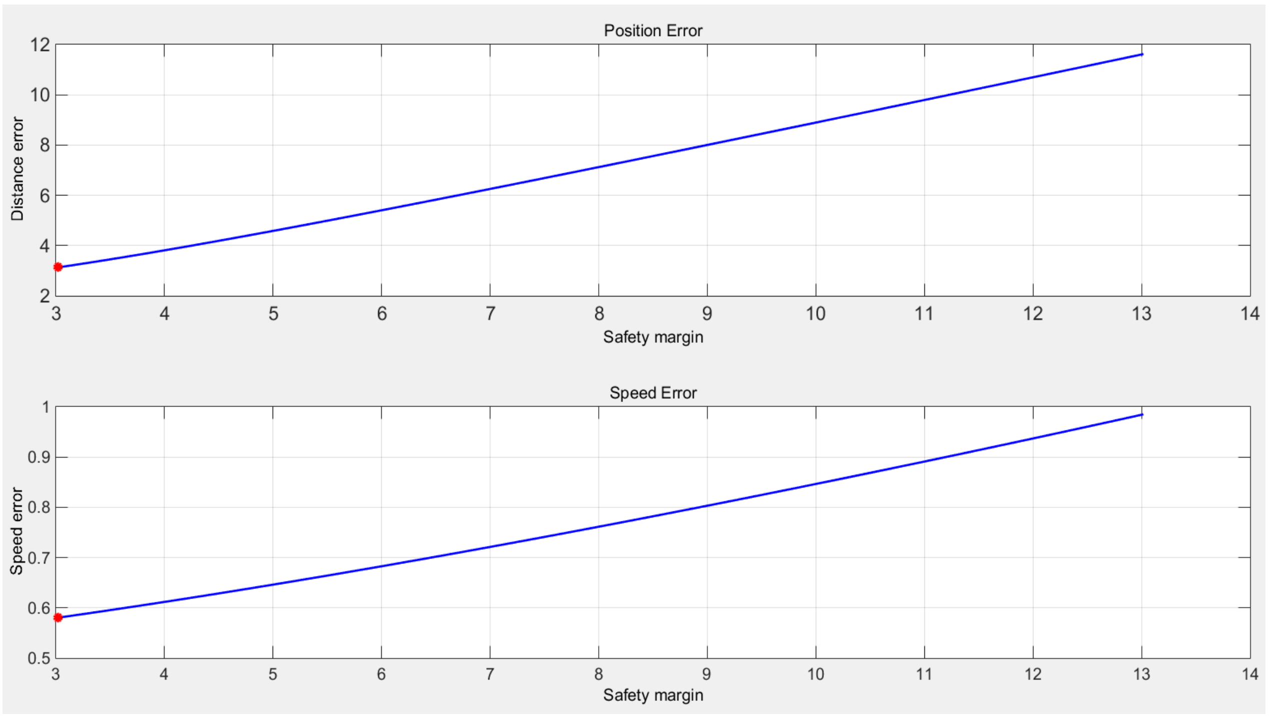


*Figure 43- Plot of Linear DoE results, safety margin being varied and the other variables kept constant.*

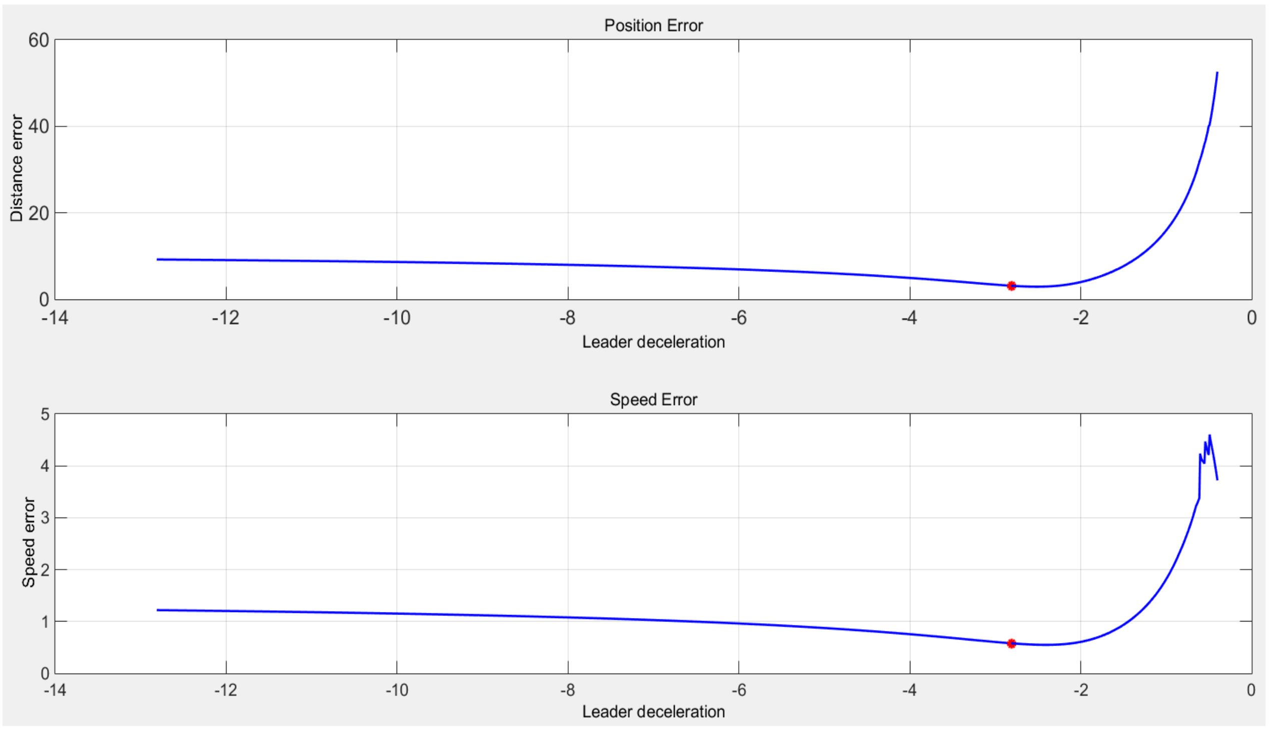


*Figure 44 - Plot of Linear DoE results, leader deceleration being varied and the other variables kept constant.*

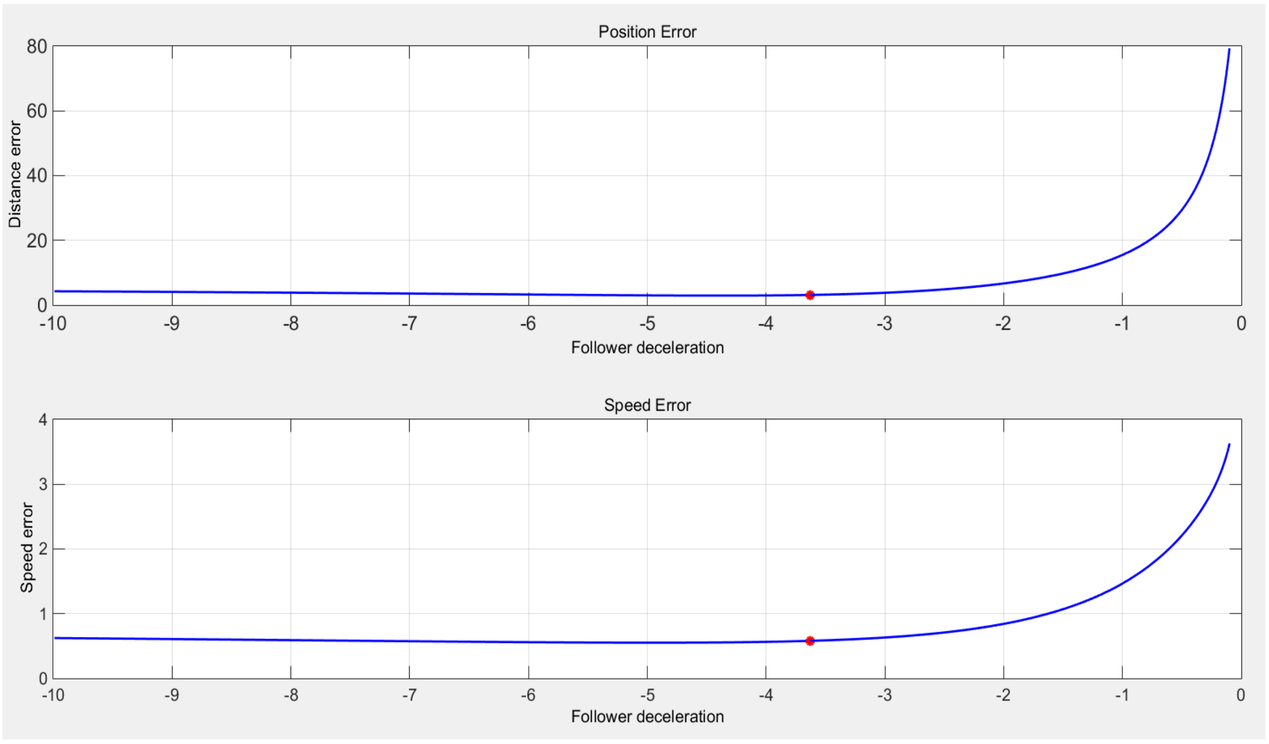


*Figure 45 - Plot of Linear DoE results, follower deceleration being varied and the other variables kept constant.*

C3: *fmincon* plot for response time being varied and the others kept constant.

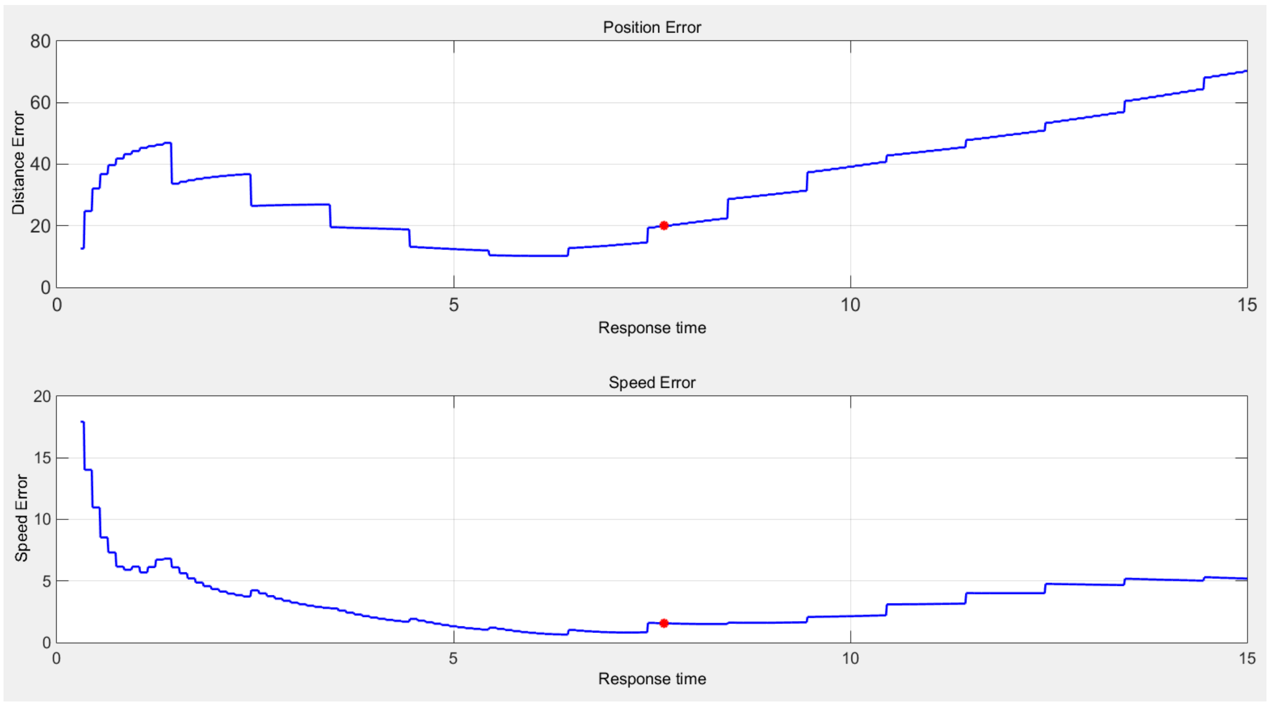


*Figure 46 - Plot of fmincon results, response time being varied and the other variables kept constant.*

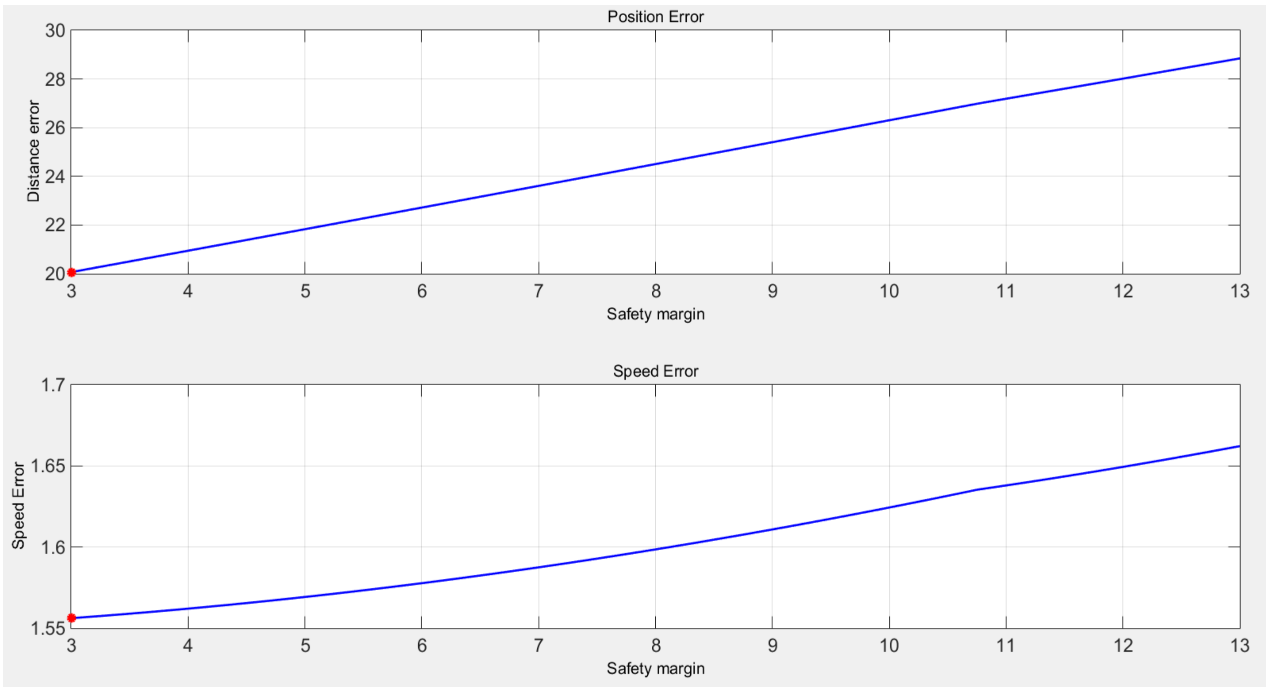


*Figure 47- Plot of fmincon results, safety margin being varied and the other variables kept constant.*

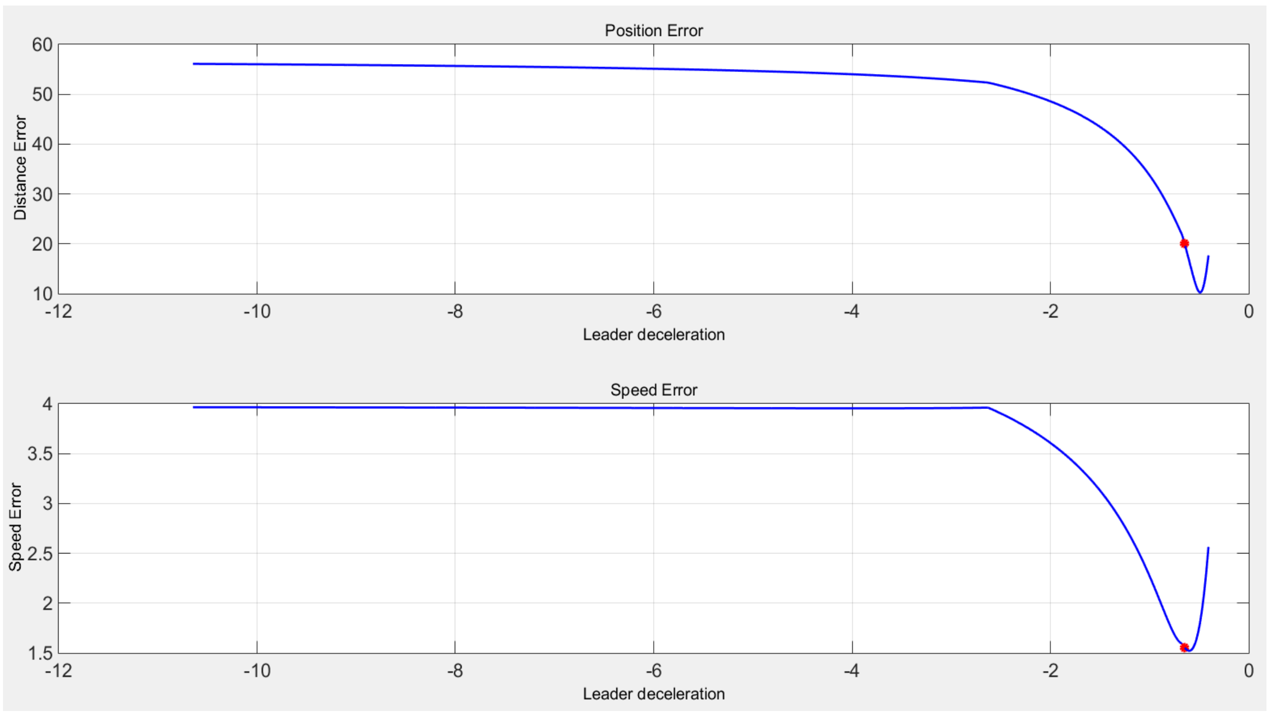


*Figure 48 - Plot of fmincon results, leader deceleration being varied and the other variables kept constant.*

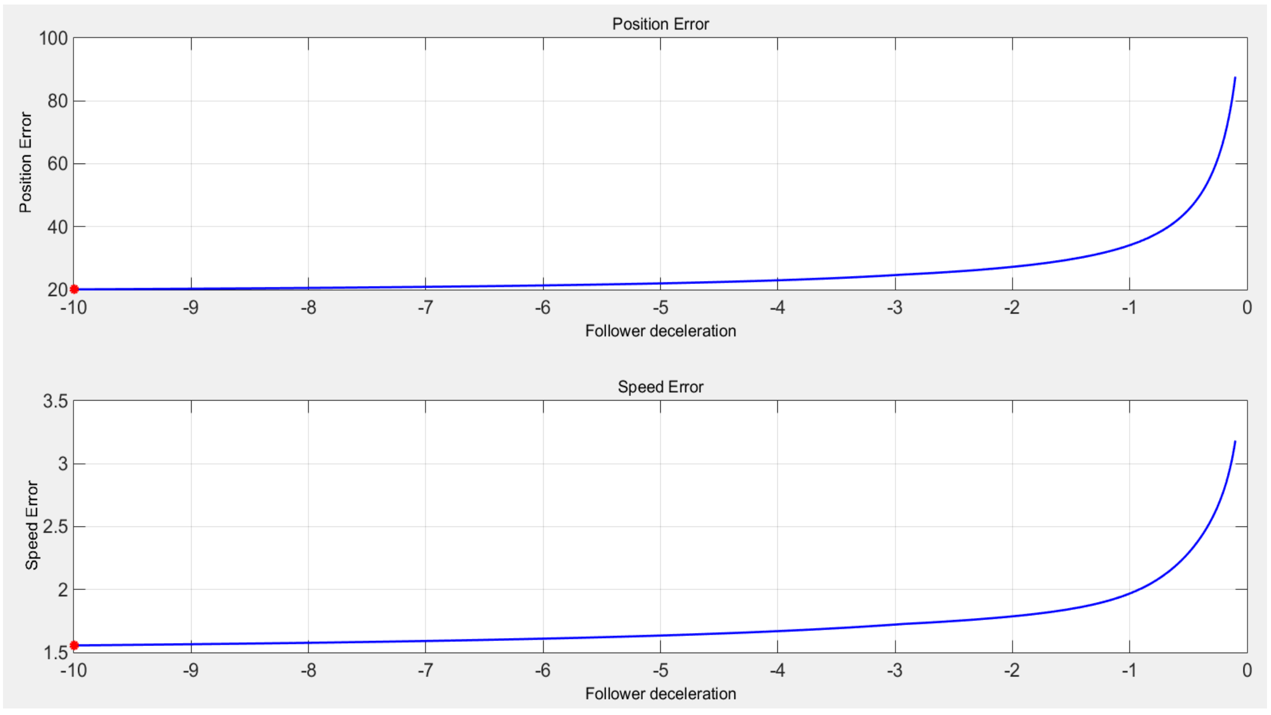


*Figure 49 - Plot of fmincon results, follower deceleration being varied and the other variables kept constant.*

C4: *Pattern search* plot for response time being varied and the others kept constant.

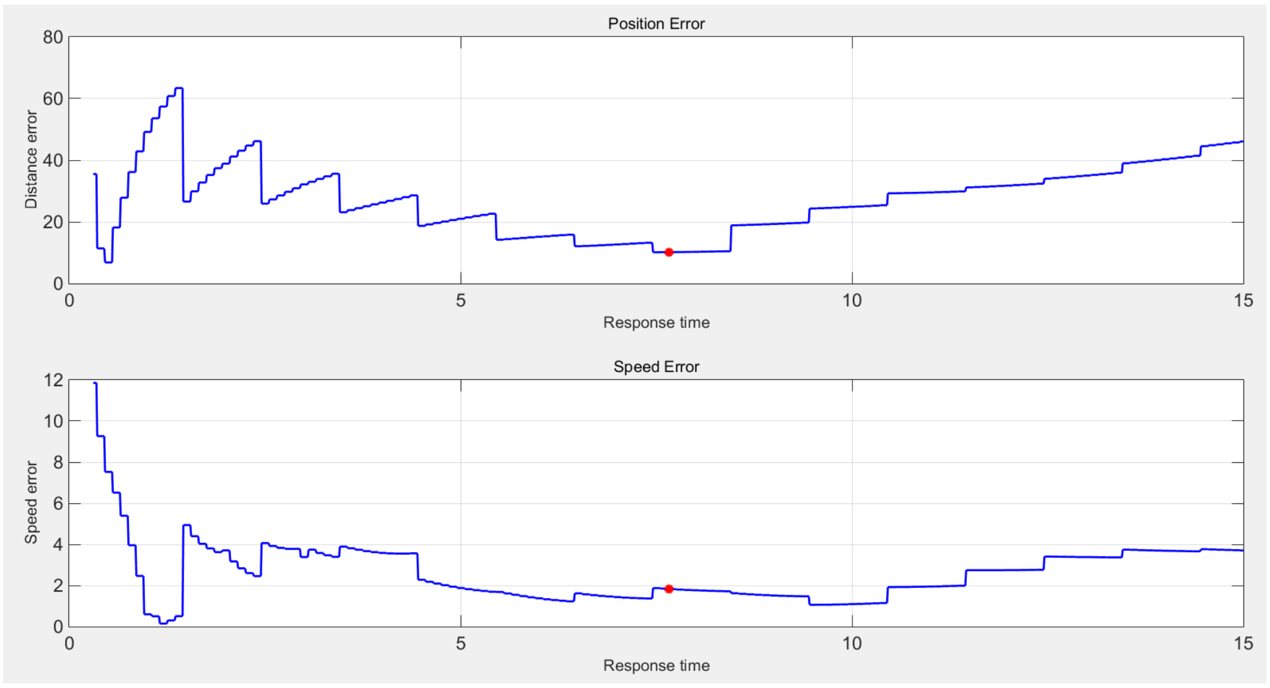


Figure 50 - Plot of pattern search results, response time being varied and the other variables kept constant.

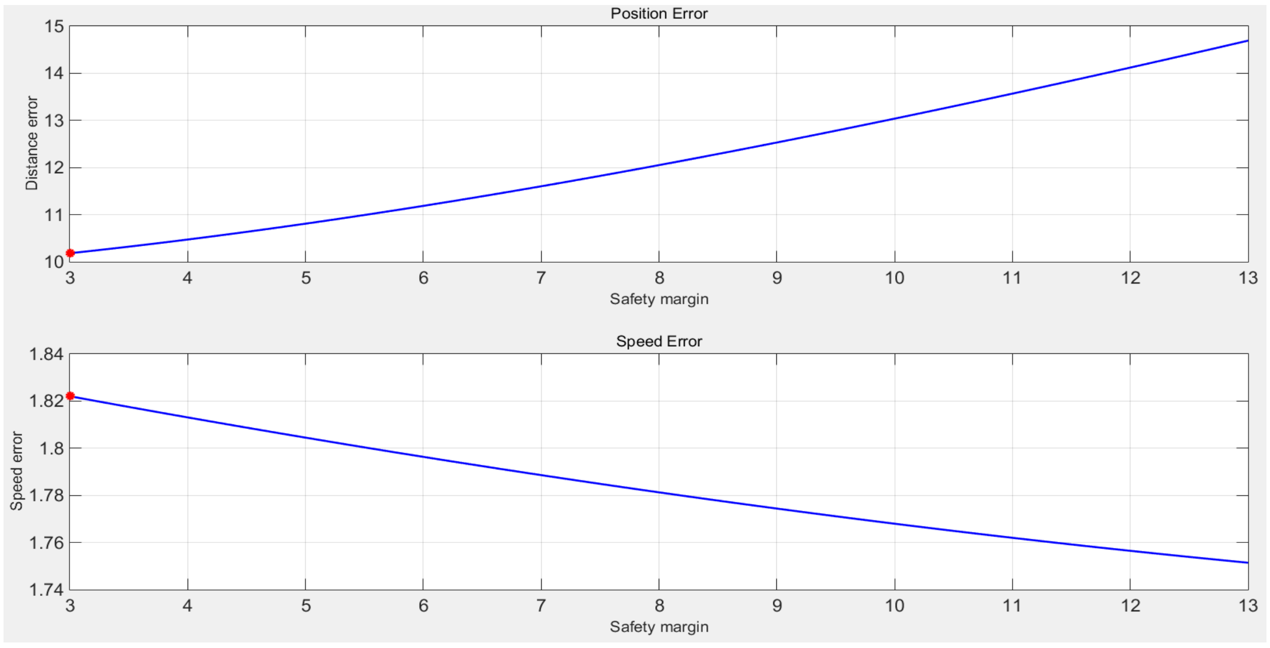


Figure 51 - Plot of pattern search results, safety margin being varied and the other variables kept constant.

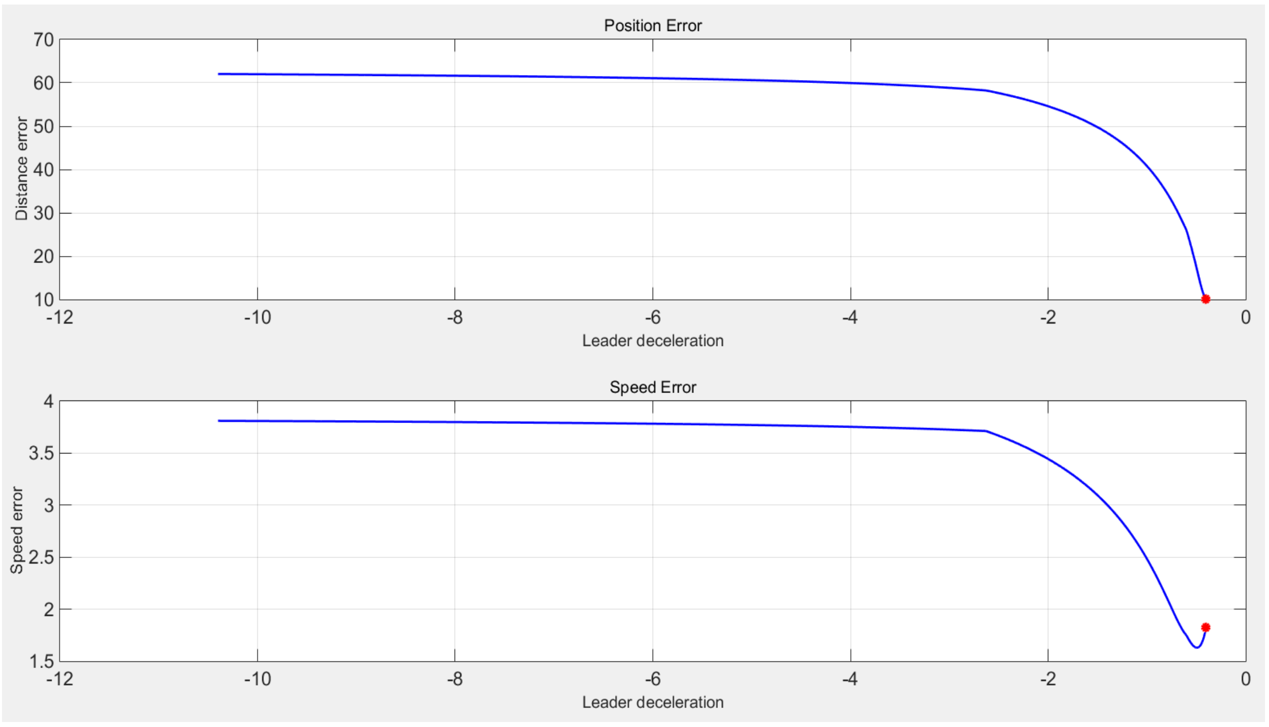


*Figure 52- Plot of pattern search results, leader deceleration being varied and the other variables kept constant.*

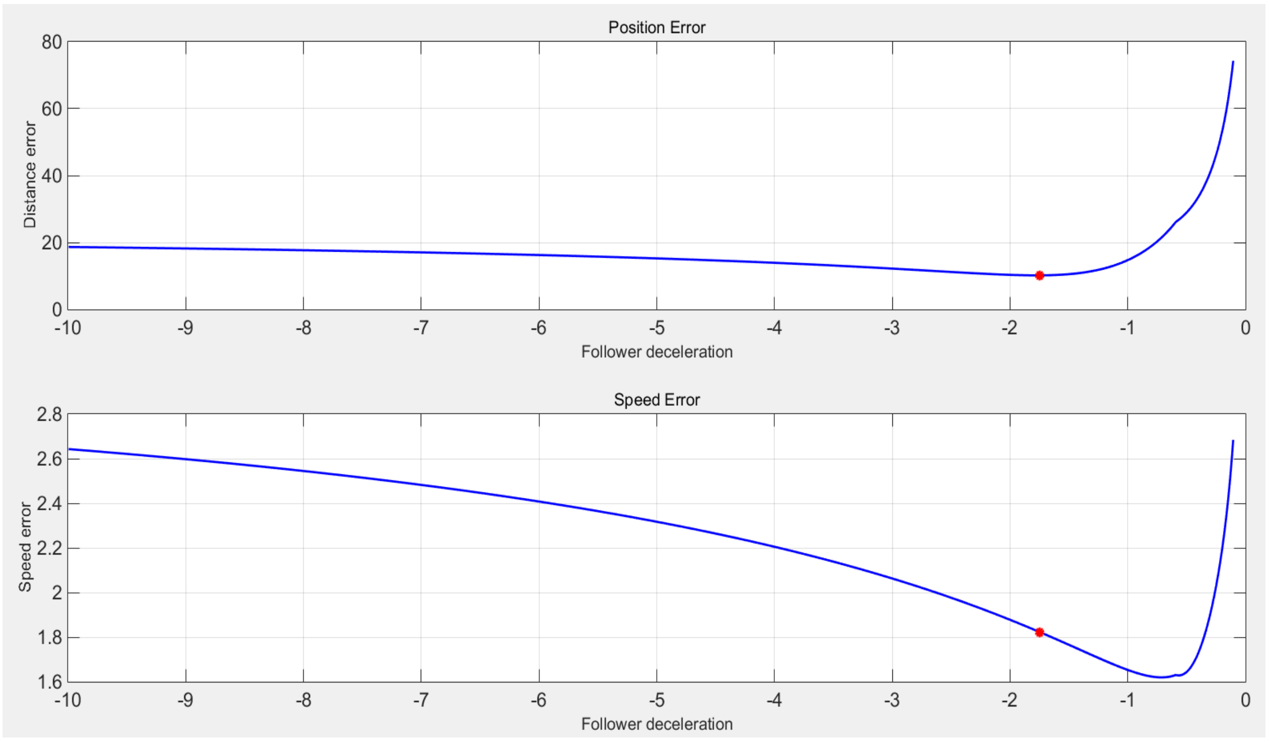


*Figure 53 - Plot of pattern search results, follower deceleration being varied and the other variables kept constant.*

C5: *JADE* plot for response time being varied and the others kept constant.

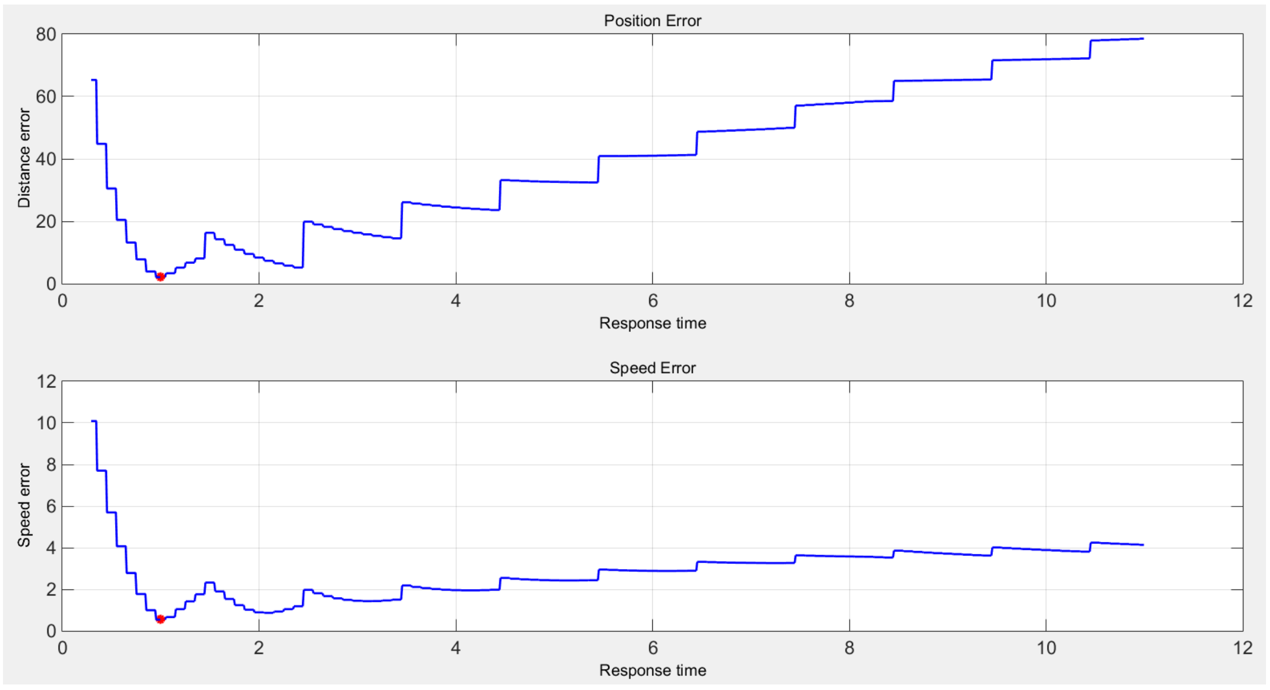


*Figure 54 - Plot of JADE results, response time being varied and the other variables kept constant.*

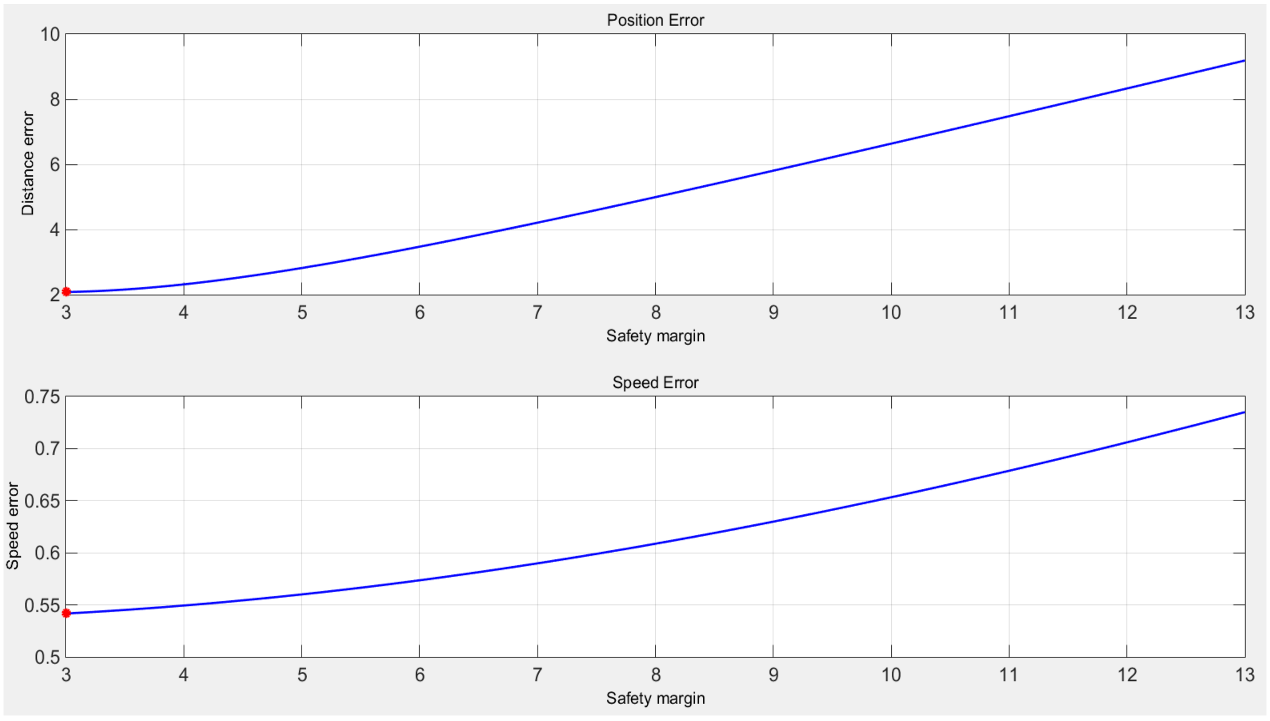


*Figure 55- Plot of JADE results, safety margin being varied and the other variables kept constant.*

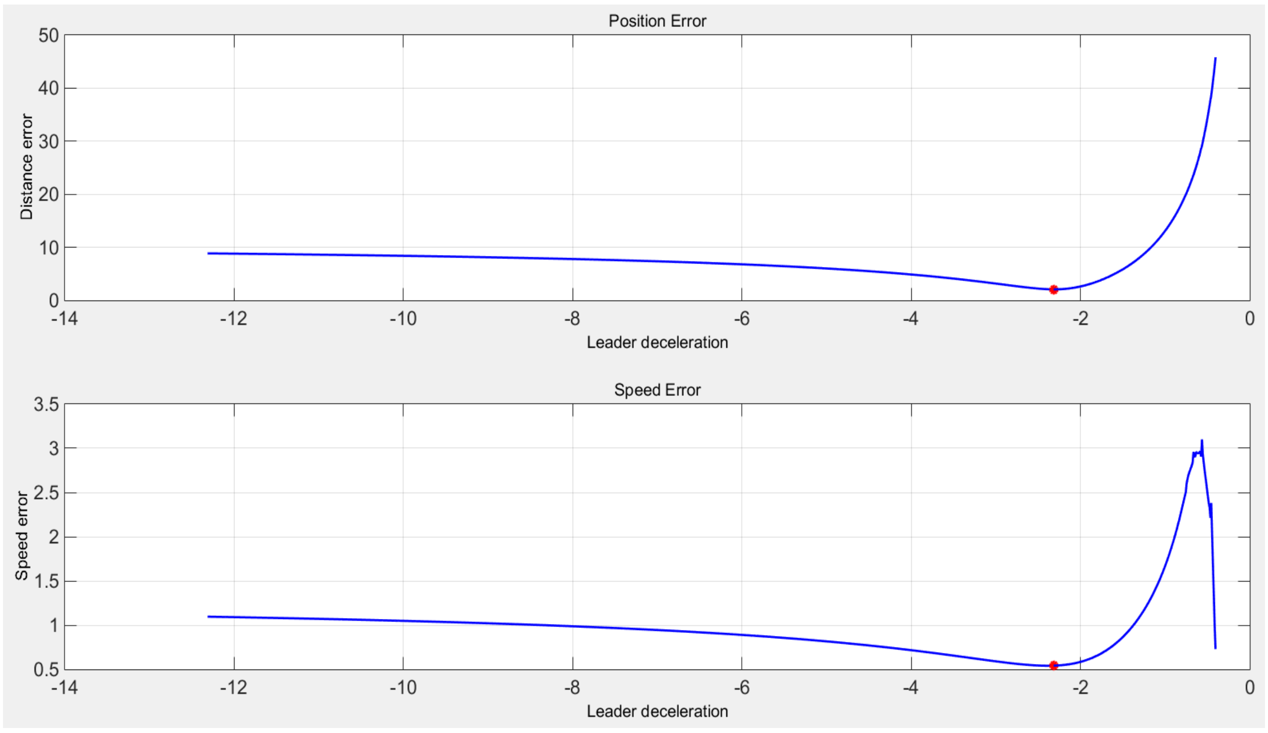


*Figure 56 - Plot of JADE results, leader deceleration being varied and the other variables kept constant.*

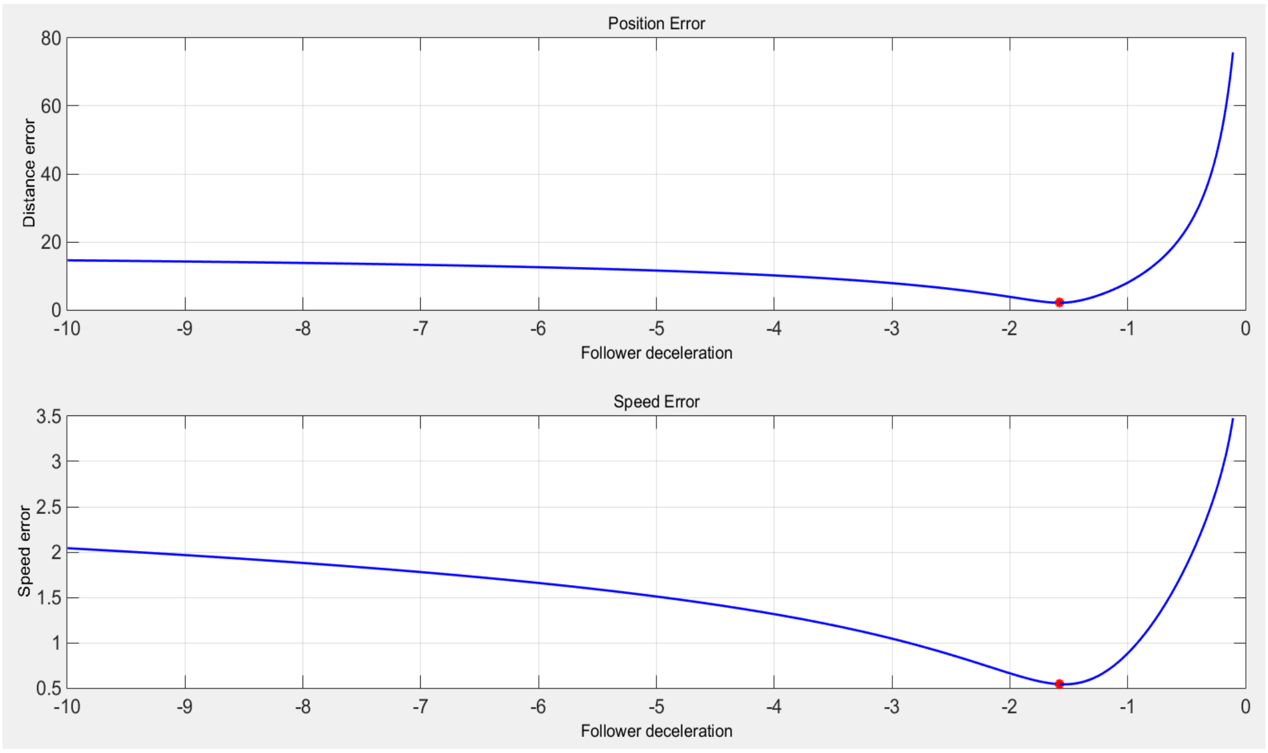


*Figure 57 - Plot of JADE results, follower deceleration being varied and the other variables kept constant.*

## 13. Appendix D: Pareto front for different time steps of the traffic model calibration.

D1: Time step of 1 second using Euler implicit method.

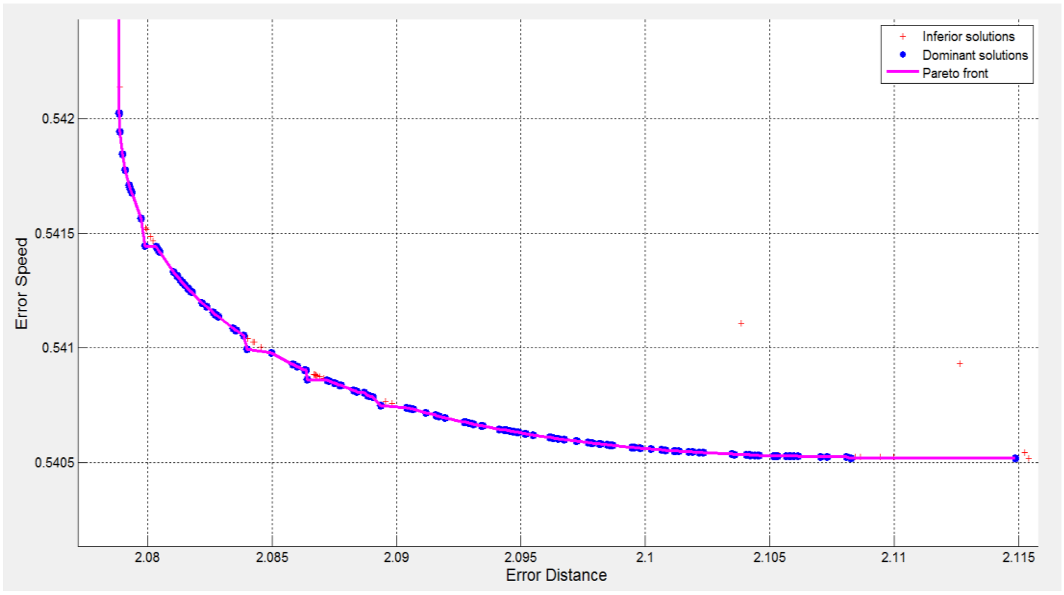


*Figure 58- Pareto front generated for the 1s time step case of traffic model calibration.*

D2: Time step of 0.5 second using Euler implicit method.

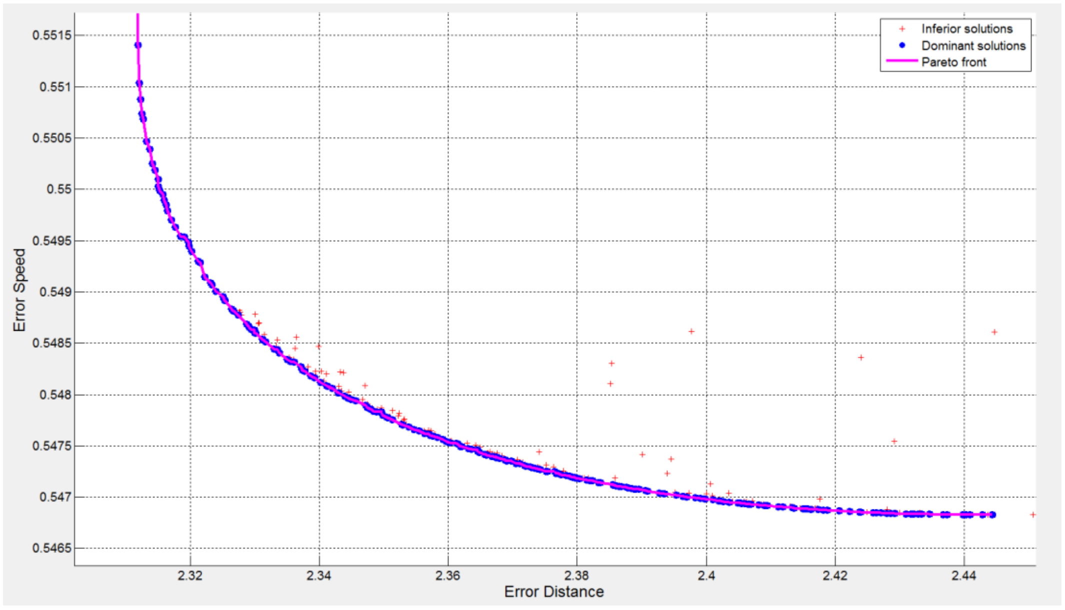


*Figure 59 - Pareto front generated for the 0.5s time step case of traffic model calibration.*

D3: Time step of 0.1 second using Euler implicit method.

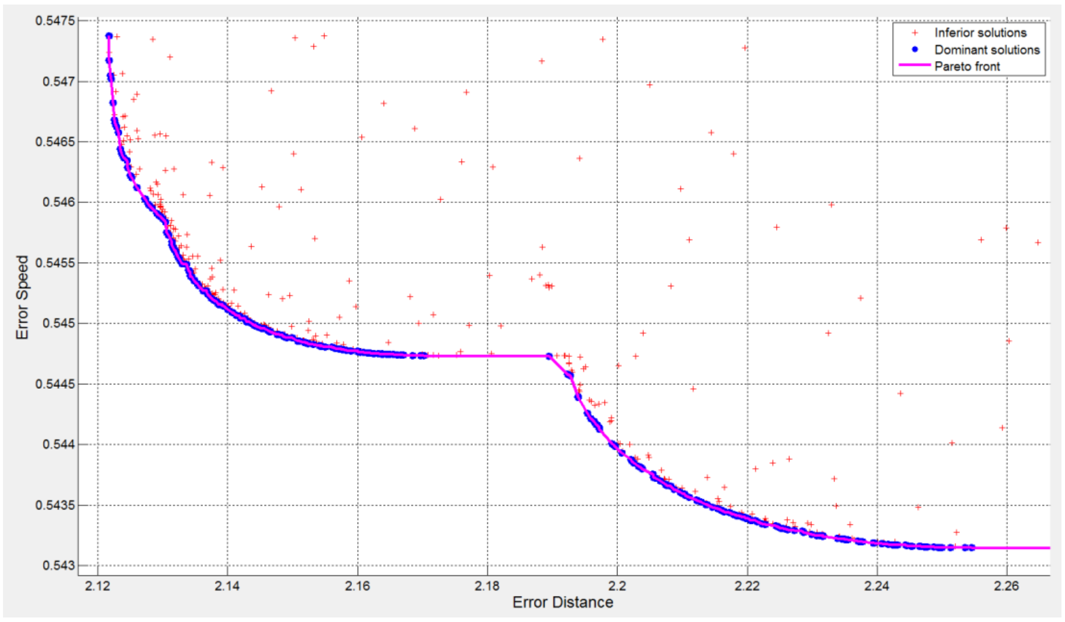


*Figure 60 – Pareto front generated for the 0.1s time step case of traffic model calibration.*

D4: Time step of 0.01 second using Euler implicit method.

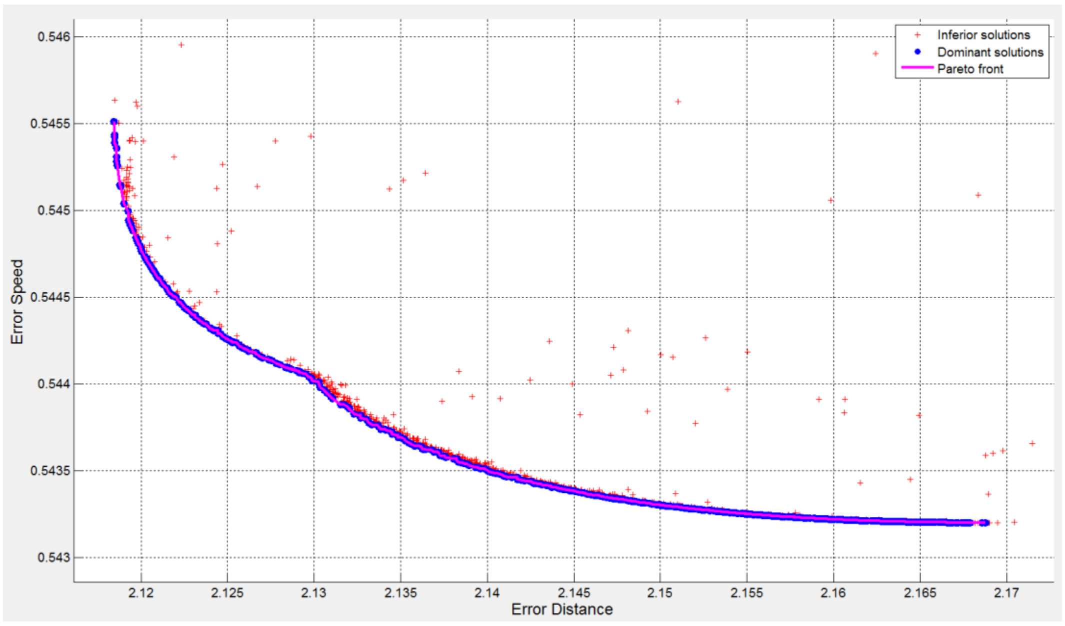


*Figure 61 - Pareto front generated for the 0.01s time step case of traffic model calibration.*

# 14. Appendix E: Pareto front for different solver types of the traffic model calibration.

E1: Euler explicit solver using time step of 1 second.

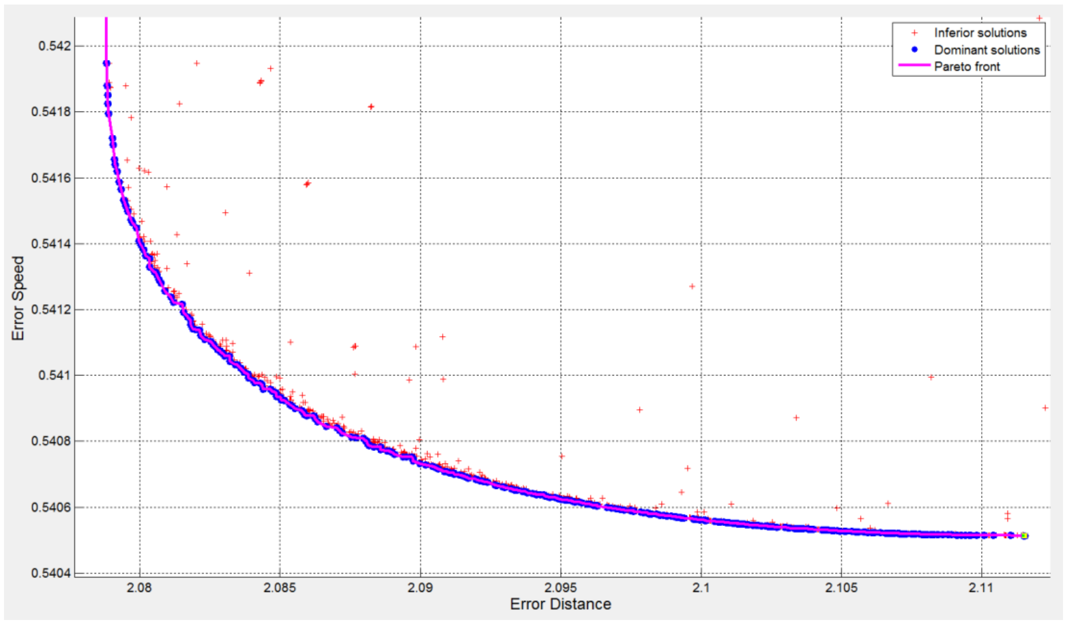


*Figure 62 - Pareto front generated for the Euler explicit solver case of traffic model calibration.*

E2: Euler implicit solver using time step of 1 second.

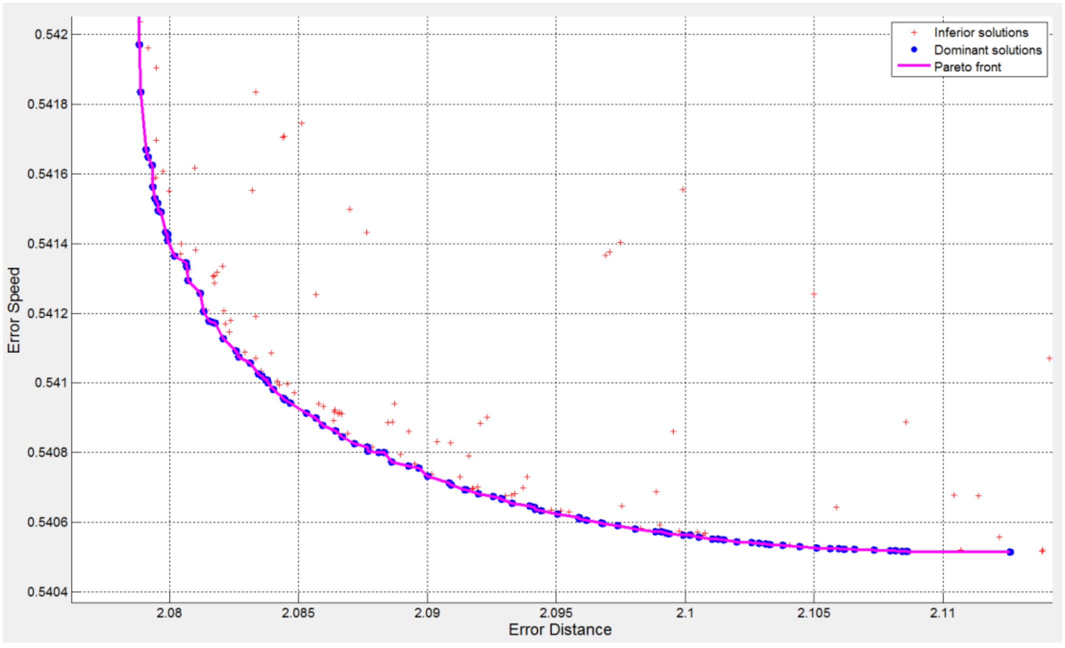


*Figure 63 - Pareto front generated for the Euler implicit solver case of traffic model calibration.*

E3: Midpoint solver using time step of 1 second.

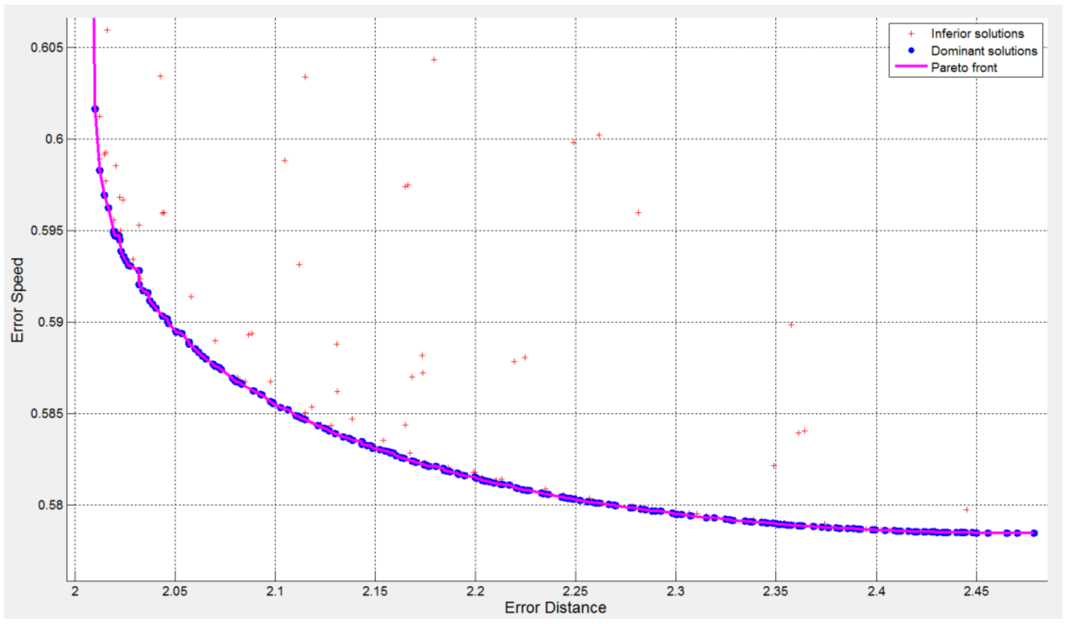


*Figure 64 - Pareto front generated for the Midpoint solver case of traffic model calibration.*

E4: Treiber solver using time step of 1 second.

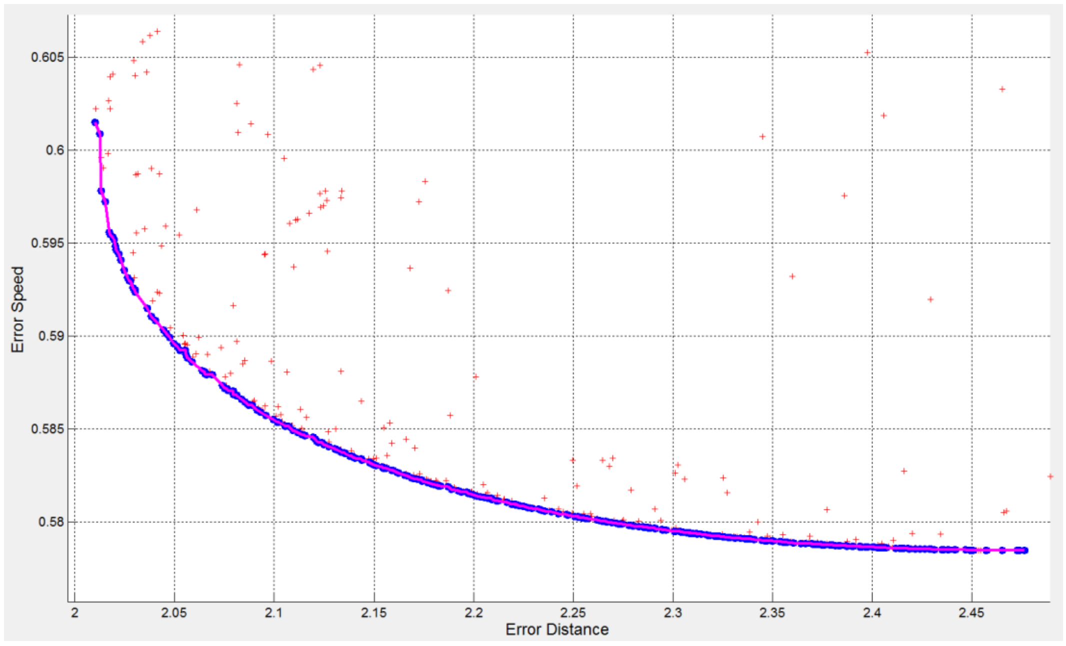


*Figure 65 - Pareto front generated for the Treiber solver case of traffic model calibration.*