SMART PROJECTILE PARAMETER ESTIMATION USING META-OPTIMIZATION

A Dissertation
Presented to
The Academic Faculty

By

Matthew S. Gross

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology

December 2017

SMART PROJECTILE PARAMETER ESTIMATION USING META-OPTIMIZATION

Approved by:

Dr. Mark Costello, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Eric Johnson
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Brian German
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Graeme Kennedy
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Aldo Ferri
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: July 31, 2017

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Mark Costello for his advice and support over my five years at Georgia Tech. Through his guidance, I have grown as an engineer, a researcher, and a leader. I would also like to thank my committee for taking the time and effort to review my thesis and provide valuable feedback to make sure this work is the best it can be. Additionally, I would like to thank Dr. Frank Fresconi and the U.S. Army Research Laboratory for funding much of my research which gave me the experience necessary to complete this thesis.

Surviving five years here at Georgia Tech would not have been possible without the help and support of my fellow graduate students in Dr. Costello's research group. Since my first day on campus, the Center for Advanced Machine Mobility has been a welcoming environment for research, collaboration, and most importantly, friendship. I would like to thank all current and past CAMM members who helped me become the engineer I am today.

Lastly, I would like to thank my friends and family who have supported me throughout my entire journey. I will be forever grateful for the love and support from my wonderful parents who have always placed a high emphasis on education and have pushed me to be who I want to be. I also want to thank the Cochran family who brought me into their lives from the moment I met them. Most importantly, I want to thank my amazing wife Katy who has been there with me through every delay and hardship, keeping me on target and making sure I made it all the way to my PhD. These past few months finish this document have been the hardest of my life, and I could have never made it through without her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

System identification and parameter estimation are valuable tools in the analysis and design of smart projectile systems. Given the complexity of these systems, it is convenient to work with mathematical models in place of the actual system. Parameter estimation uses time history data of the system to determine a model that accurately matches the data. Many techniques have been developed to perform parameter estimation, including regression methods, maximum likelihood estimators, and Kalman filters.

Maximum likelihood methods, in particular the output error method (OEM), pose the estimation problem in terms of an optimization problem. OEM has seen extensive use on projectile systems, utilizing a numerical optimizer such as a Newton style algorithm to solve for unknown parameters. These algorithms are prone to converging on local minima present in the projectile dynamics, requiring reasonable initial guesses of the parameters to ensure convergence. However, for new smart projectile systems, prior estimates of the control parameters may not be available. Thus, there is a need for reliable and robust parameter estimation methods that are not dependent *a priori* knowledge of the parameters.

This thesis proposes a new method for smart projectile parameter estimation based on OEM. To achieve robust and reliable parameter estimates, a new underlying optimization algorithm is formed, dubbed meta-optimization. Meta-optimization employs a diverse set of individual optimization algorithms with both local and global search capabilities. The meta-optimizer operates by iteratively selecting a single algorithm to deploy in a stochastic manner, giving preference to algorithms which have performed well on the problem. This approach allows synergies to develop between the individual optimizers, boosting performance beyond what each optimizer is capable of individually.

A suite of benchmark functions commonly used in the optimization field are used to analyze the meta-optimization framework and compare it to other existing algorithms. These functions have the benefit of known structure and solutions and are less computationally intensive than the parameter estimation problem. A series of trade studies are performed to evaluate each component of meta-optimization and determine a robust configuration for use on general optimization problems. Meta-optimization is also compared to the individual algorithms it employs, showing superior performance and reliability over this benchmark suite. Finally, meta-optimization is compared to other state of the art algorithms, showing comparable performance.

The new parameter estimation method is applied to an example smart projectile system equipped with a new microspoiler control mechanism. Both synthetic and experimental trajectory data is used to evaluate the effectiveness of the proposed method. From the synthetic data, the parameter estimation algorithm accurately estimates the aerodynamic coefficients of a standard projectile as well as parameters for a smart projectile executing a maneuver. This synthetic data is also used to conduct trade studies investigating how the data itself impacts the accuracy of the parameter estimates. Lastly, the method is applied to flight test data collected at the U.S. Army Research Laboratory spark range with good results in the presence of large measurement errors.

CHAPTER 1
INTRODUCTION AND BACKGROUND

System identification and parameter estimation are common tools within the aerospace discipline. When working with complex systems, it is often difficult or impractical to use the actual system for tasks such as designing new control strategies [1, 2] or characterizing new aircraft configurations [3, 4]. Therefore, it is useful to develop mathematical models which approximate the behavior of the actual system. System identification is used to determine appropriate models by observing the response of the system to various inputs. To simplify this process, a single model can be chosen which contains a number of parameters that govern the behavior of the model. This type of system identification is known as parameter estimation where the model is known *a priori*. A general nonlinear model for parameter estimation is given by Equation 1.1 [5]

$$\mathbf{z} = h(\mathbf{x}) + \mathbf{v} \tag{1.1}$$

where $\mathbf{z}$ is the measurement vector, h is a function with known form, $\mathbf{v}$ is the measurement noise vector, and $\mathbf{x}$ is the set of parameters which govern the model. The model is often based on the underlying physics of the system with varying levels of complexity. For example, a dynamic model of a smart projectile system contains the aerodynamic coefficients for the projectile body and any additional parameters associated with the control mechanism [6, 7]. The quality of a set of parameters is determined by defining a function which compares the output of the model with trajectory data. This parameter estimation problem can then be posed as an optimization problem, using a numerical optimization algorithm to seek the set of parameters which best fit the data [8]. Given the complexity of many models, solving this problem can

be difficult for existing numerical optimization methods.

Numerous numerical optimization strategies have been employed to perform parameter estimation. These algorithms are iterative approaches that gradually progress towards a solution which may or may not be the global solution to the problem. Typically, gradient based optimizers have been used to solve parameter estimation problems [8]. Other optimizers such as particle swarm optimization [9, 10, 11] and genetic algorithms [12, 13, 14] have also been used. Each of these algorithms are suited to different types of problems. For example, gradient based algorithms are strong local search methods, but are only capable of searching within a neighborhood around a solution. This is seen in Figure 1.1a where a gradient based optimizer is used to solve the Griewank function. This function has a global minimum at (0,0) and alternating minimums and maximums along the x and y directions. Within this space, there are four local minima at $(\pm\pi, \pm\sqrt{2}\pi)$. As a demonstration, steepest descent, a simple optimization method that follows the gradients of the function, is started from a point near (2,-2). As it progresses, the contours of the function steer the algorithm towards the local minimum at $(\pi, -\sqrt{2}\pi)$. Global methods like particle swarm, on the other hand, are able to freely explore the entire parameter space, but often require significant computation time to determine solutions and are also not guaranteed to find the global best solution. Figure 1.1b shows a population of particles, each with a velocity vector which describes the motion of each particle as it searches the space.

It is important to note that there is no single optimizer that can successfully solve all optimization problems, let alone solve the problems in a computationally efficient manner. The proper selection of an optimizer for a given problem is a critical decision by an engineer as the selection of the wrong algorithm can have serious consequences. More and more, optimization is being used in an embedded manner where the optimizer is part of a completely automated process, such as within an

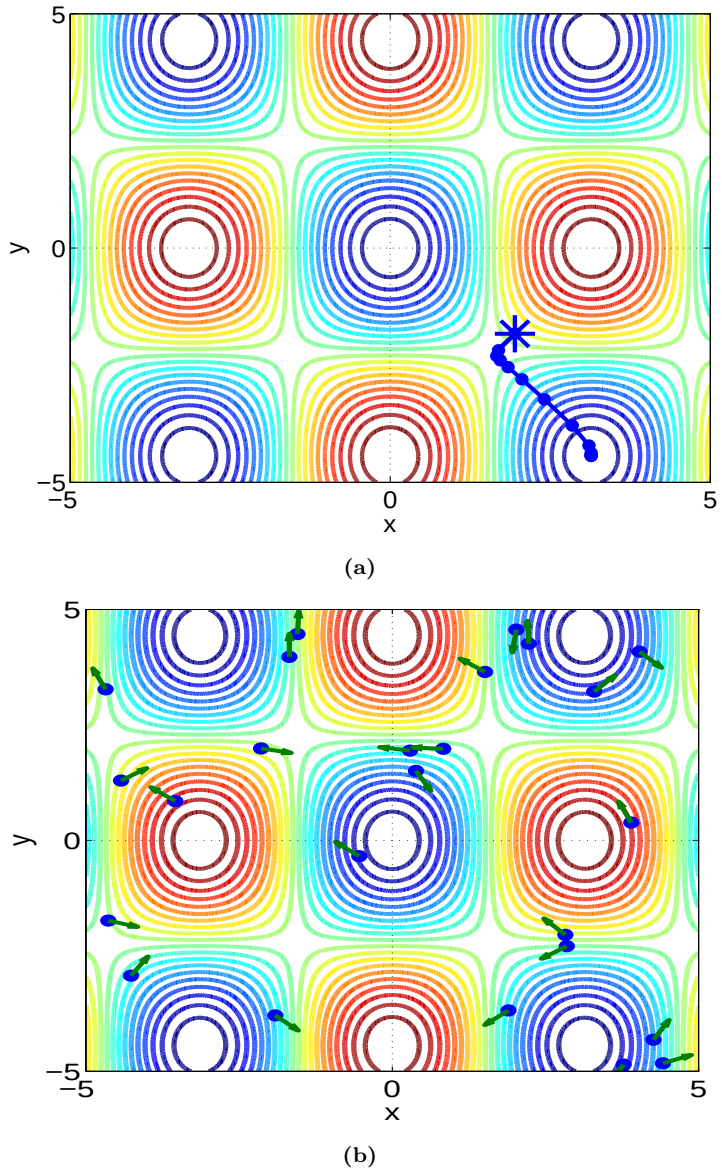


Figure 1.1: Example Optimizer Performance on 2-D Griewank Function (a). Steepest Descent; (b). Particle Swarm Optimization

adaptive flight control algorithm [15]. In these cases, the requirements placed on the optimizer become shifted from traditional practice. It is paramount that the optimizer work every time and achieve an acceptable solution within a certain amount of time. Due to the nature of a given problem, numerical optimizers can be prone to getting stuck during the iterative process, yielding unsatisfactory results. This can occur in a few ways. On multi-modal problems, optimizers can converge on a

local solution instead of the global solution. Optimizers can also diverge on complex problems, resulting in non-physical solutions. Some optimizers can wander aimlessly in a limited part of the parameter space far from the global optimum. Any of these scenarios would be problematic in an embedded system and would be an unacceptable result for the optimizer. Thus, there is a need for optimizers that are easy to use and highly reliable at solving a wide range of optimization problems. The concept of algorithm portfolios and hybrid optimizers provide a potential avenue for improving reliability by combining a diverse set of optimizers in a common framework that maximizes performance and adaptability.

1.1 Related Work

Many techniques exist for performing parameter estimation on projectile systems with the most common utilizing numerical optimizers to obtain accurate estimates. Existing approaches typically employ gradient based methods which require reasonable estimates of the parameters to guarantee convergence. Metaheuristic optimizers, on the other hand, are able to search globally and can avoid local optima. The wide range of optimizers to choose from presents its own challenge, with the selection of appropriate optimizers requiring *a priori* knowledge of the problem. Multiple methods have been developed to solve this selection problem by allocating resources between various optimizers based on performance. More reliable optimizers have also been developed which combine multiple algorithms to reduce the limitations of the individual optimizers.

1.1.1 Aerospace System Parameter Estimation

Significant literature exists on the topic of system identification and parameter estimation including entire books on general system identification methods [16, 17, 18], as well as books specifically covering applications to aerospace systems [5, 19, 20].

Parameter Estimation problems are often posed as optimization problems through the use of maximum likelihood estimators (MLE). The MLE seeks the set of system parameters which maximize the likelihood of the system model matching a set of experimental observations. This method is most effective for stochastic dynamic systems with both process and measurement noise. The output error method is a simplified MLE which neglects the process noise, allowing the system dynamics to be directly integrated. An objective function, also known as a cost function, is defined to minimize the difference between given data and a simulated model at a discreet number of points on a set of trajectories. Figure 1.2 gives a general block diagram for the output error method [5]. Here, observed measurements of the actual system are compared to outputs from the model using estimated parameters. The parameter estimates are updated based on the objective function output. It is typical for this method to be coupled to a Newton style numerical optimizer to solve for unknown parameters.

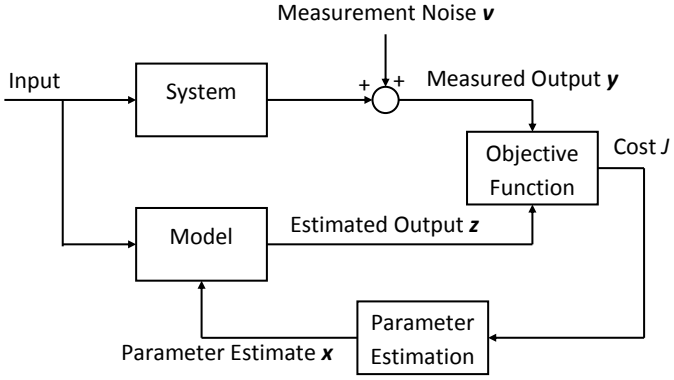


Figure 1.2: Block Diagram of Output Error Method [5]

In the field of projectile parameter estimation, a popular MLE based algorithm is the Aeroballistics Research Facility Data Analysis System (ARFDAS) [21, 22, 23]. ARFDAS uses projectile linear theory to produce parameter estimates used as the starting point for the MLE. An iterative approach is used to match simulated six degree-of-freedom (6DOF) trajectories with experimental data typically obtained

from spark range testing [6]. A similar approach was developed by Montalvo and Costello using the output error method. Like ARFDAS, linear theory is employed to determine initial values for the unknown parameters before the Levenberg-Marquardt process is used to estimate the parameters based on coupled computational fluid dynamics (CFD)/rigid body dynamics (RBD) virtual flyouts [7]. Burchett used a gradient based approach based on a linear model of the projectile dynamics. Simulated yaw card data with no measurement noise was used to demonstrate the algorithm. The gradient based method was also compared to a genetic algorithm based approach [24, 25]. Condaminet et. al. investigated four different problem configurations for estimating the parameters of a reduced order ballistic model using partial flight data. In all cases, a Newton-Raphson technique is used to solve the optimization problems [26].

To perform parameter estimation on projectile systems, data is collected from test firing projectiles under a range of conditions, often in a spark range [6, 27, 28]. A spark range consists of a set of orthogonal spark shadowgraph stations which allow for the measurement of the position and orientation of the projectile in flight. Each spark station consists of a spark box and a camera. The spark box generates an extremely short duration flash, allowing the camera to capture an instantaneous image of the projectile in flight. The US Army Research Laboratory (ARL) Transonic Experimental Facility spark range is pictured in Figure 1.3. Here, the orthogonal spark stations can be seen with one set in the trench on the ground and the other along the adjacent wall. The white screens provide a backdrop for the projectile images, producing a clear image of the projectile. An example shadowgraph image is shown in Figure 1.4. The position and orientation of the projectile at each station can then be measured using these images. While spark ranges are known to provide accurate measurements, the limited number of measurements can cause issues when performing parameter estimation. Alternatively, the projectile can be outfitted with

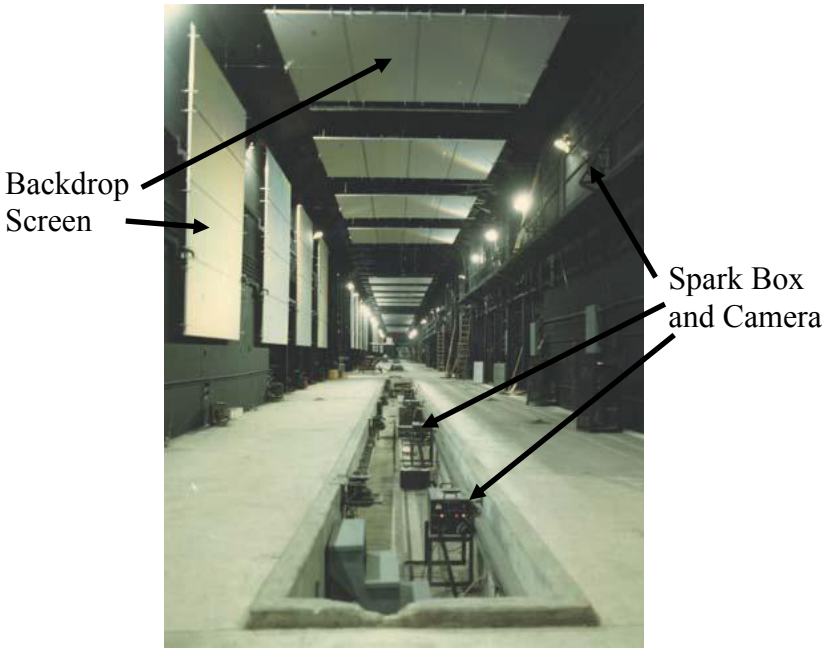


Figure 1.3: Picture of ARL Spark Range [6]

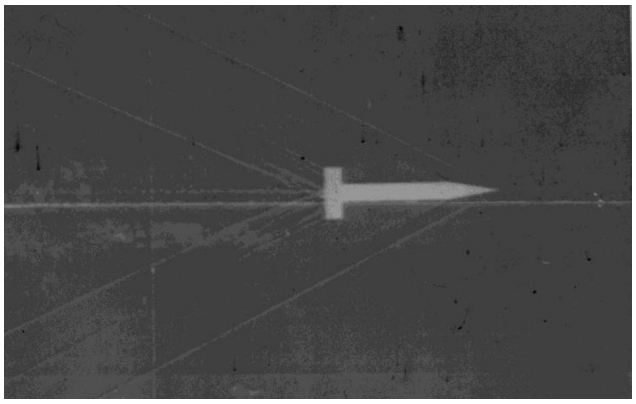


Figure 1.4: Example Shadowgraph Image of Projectile From ARL Spark Range [29]

on board sensors which record detailed telemetry throughout the flight [30]. More recently, computational fluid dynamics (CFD) simulations have been employed to determine the aerodynamics of new projectiles [31, 32]. In addition, CFD techniques have been combined with flight dynamics simulations to create virtual flyout data that is used in place of flight testing [33, 34, 7].

One major drawback of existing projectile parameter estimation algorithms is that while they are powerful and have been successfully deployed to date, they work best

when given initial parameter values close to the solution. Newton style numerical algorithms are local search techniques and are prone to converging on local minima which are very common in projectile parameter estimation problems. Projectile linear theory can provide reasonable estimates for some parameters, but does not capture the fully nonlinear behavior of the projectile and may not be able to provide reasonable estimates for some parameters. This issue greatly decreases the performance of these techniques on new projectile systems that are more complex and contain parameters that are completely unknown and cannot be easily estimated through other methods. These tools also are difficult to modify to account for the addition of control mechanisms and control logic and may not be the most efficient under all circumstances.

1.1.2 Numerical Optimization

An enormous amount of information exists on the development and formulation of optimizers of various styles. Numerous books have been written on the topic including books on convex optimization [35], numerical optimization [36], linear and nonlinear optimization [37], and metaheuristics [38, 39, 40, 41] in addition to entire journals devoted to optimization such as the *Journal of Optimization Theory and Applications*, *Journal of Global Optimization*, *IEEE Transactions on Evolutionary Computation*, and many more. Some specialized books have also been written covering genetic and evolutionary algorithms [42, 43], particle swarm optimization [44] and simulated annealing [45]. Entire courses are taught on optimization at the university level. The optimization literature includes theoretical mathematical research, numerical approaches, and numerous applications. Despite the diverse nature of these algorithms, every optimizer follows a similar framework. First, the algorithm begins from an initial solution or set of solutions and then iteratively improves this solution until it reaches some stopping criteria.

The basic structure of an optimization problem consists of an objective or cost function which is minimized or maximized by proper selection of problem parameters. The problem parameters to be selected may be constrained by equality or inequality equations. The objective function is defined to describe the goals of the engineering problem in mathematical terms. Mathematically, an optimization problem can be expressed as Eq. 1.2.

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0,\ i = 1, \ldots, p \end{aligned} \tag{1.2}$$

where $f(\mathbf{x})$ is the cost function, $g_i(\mathbf{x})$ are the inequality constraints, and $h_i(\mathbf{x})$ are the equality constraints. An optimal solution occurs when the gradient of the cost function equals zero and the Hessian is positive definite for minimization problems or negative definite for maximization problems. Some optimization problems may contain multiple solutions which satisfy these requirements. These problems are considered multi-modal with a single global optimum and numerous local optima.

Due to the complexity of most optimization problems, numerical methods are often employed to obtain solutions. Each optimizer is suited for a certain type of problem based on the desired topology. Most optimizers can be broken into two classes: gradient based methods and metaheuristics. Gradient based methods, also known as hill climbers, are iterative approaches which seek a lower cost value along a search direction in the parameter vector space. The search direction is determined using local information on the gradient and curvature of the cost function. These optimization algorithms are extremely efficient at solving uni-modal and convex problems, but are prone to converging to local optima on multi-modal problems. Due to this fact, gradient based methods are generally considered local search methods and are highly dependent on initial guesses of the solution.

Gradient based algorithms are comprised of two main components; a search direction and a line search algorithm. The primary difference between these algorithms are the methods for determining an appropriate search direction. The most obvious direction is the negative of the gradient of the cost function where it is guaranteed a lower cost can be found. A search in this direction is known as steepest descent. While steepest descent provides the fastest decrease in cost, this search direction can be extremely inefficient for complex problems [36]. The most important search direction is the Newton direction. Newton's method is an iterative approach for finding the solution of a nonlinear function by taking the first order Taylor series expansion of the function. For the case of optimization problems, Newton's method models the cost function as locally quadratic with a closed form solution that is easily determined. For quadratic problems, Newton's method can obtain the solution with a single step and is very efficient on other convex problems. However, this model requires both the gradient and the Hessian matrix at the current search point making it a second order method. Computation of the Hessian, in particular, can be computationally expensive for complex problems [37].

The other component of gradient based methods is the line search algorithm. The goal of the line search is to find the lowest cost value along the line defined by the search direction in the parameter vector space. The simplest line search method is a backtracking search. Backtracking begins with a Newton step and gradually reduces the step length until a nontrivial decrease in the cost is observed. A Newton step is chosen as the first step as it would be expected to produce fast convergence near the solution. By slowly decreasing the step length, the algorithm ensures that the step is not too small and will eventually lower cost [37]. Another common line search technique is the golden section method. This approach is based on the concept of the golden ratio of the distances between three points on the line. An iterative procedure is used to gradually reduce the bounds around a minimum value until the minimum is

found [46]. On highly nonlinear problems, line search algorithms can struggle where the local model used to generate the search direction is a poor approximation of the cost function.

Metaheuristics provide an alternative numerical optimization approach with both local and global search capabilities. A heuristic is a procedure or process for solving a problem based on intuition or experience. A metaheuristic is, therefore, a higher level procedure which selects an appropriate heuristic to apply to a given problem. The term is also generally used to classify all optimizers that do not use gradient information. Global metaheuristic optimizers are able to search the entire function space and often converge on the global optimum, however this is not guaranteed. These global methods often require a large amount of computation time to determine the solution, or they may be unable to solve certain classes of problems entirely [38, 39, 40, 41].

Many metaheuristic algorithms are based on phenomenon observed in nature such as genetic evolution and insect and bird behavior and combine stochastic and local searches. The stochastic nature of metaheuristic algorithms allows for the exploration of the search space without getting caught in local optima, guiding the algorithm towards the global solution. These optimizers are best suited for problems where little information on the problem topology is known. They also have the advantage of not requiring gradient information, greatly simplifying the algorithms and allowing for more general applicability [41]. Some of the most popular metaheuristic approaches are genetic algorithms and similar evolutionary algorithms. Genetic algorithms (GA) take a population of candidate solutions and express them as a string of bits, or genes, to form a chromosome. New populations are generated using various genetics inspired operators, such as gene mutation and crossover, along with a form of natural selection [42]. Storn and Price developed a simplified evolutionary algorithm called differential evolution (DE) which uses real numbers to express the variables with requisite modi-

fications to the mutation and crossover operators [47]. Another popular population based optimizer is particle swarm (PSO) developed by Kennedy and Eberhart. The algorithm is based on swarm behavior observed in nature, specifically the sharing of information between agents in the swarm [48, 49].

Another category of numerical optimization algorithms for global optimization are Bayesian optimizers. The basis of Bayesian global optimization (BGO) is to approximate the objective function using a probabilistic model that is computationally cheaper than the objective function to solve [50]. These optimizers are especially effective when the objective function is computationally expensive to evaluate. BGO begins with an *a priori* distribution of the objective function which represents the initial guess of the function landscape. A new point is sampled by seeking the point with greatest expected improvement according to the *a priori* distribution. This metric allows for a balance between exploration of uncertain regions of the parameter space and exploitation of the predicted distribution. The *a priori* distribution is then updated based on observing the objective function at this new point [51]. The efficient global optimization algorithm (EGO) is a popular type of BGO, using a kriging based model to predict the objective function and a branch and bound algorithm to maximize the expected improvement to determine the next sample point [52, 53]. The exploration and exploitation capabilities of BGO can be controlled through the selection of different methods for obtaining future sampling points [54].

1.1.3 Algorithm Selection

For an engineer, the selection of an appropriate algorithm to use for a given problem is a critical decision which may be difficult to make *a priori*. The process for selecting a method is essentially the algorithm selection problem outlined by Rice [55]. To select the best algorithm, Rice proposed a method which seeks to find a mapping from a set of problems to a set of algorithms based on a specified performance measure.

This approach is known as meta-learning and typically uses machine learning to develop this mapping [56, 57, 58, 59, 60]. Another common approach is the algorithm portfolio which combines a number of different algorithms into a portfolio and allocates computational resources intelligently to reduce computation time and improve performance for a given problem. The expected performance of each algorithm is determined by running each algorithm on a set of training problems and applying machine learning techniques to help predict future performance [61, 62, 63]. A major drawback of this approach is that it requires training of the mapping before it is used on a new problem, requiring a large amount of computation time for training and limiting applicability to only similar problems. Also, the optimal algorithm may vary throughout the solution process, with a sequence of algorithms being more efficient than a single algorithm.

Alternatively, a dynamic algorithm portfolio can be used which develops a mapping while solving the problem using feedback from the solution process. Given a set of algorithms, each algorithm is allocated a discrete amount of time to work on solving the problem. The amount of time given to each algorithm is based on the relative performance of each algorithm, favoring those which perform best [64, 65, 66]. This approach can be improved by treating the time allocation problem as a multi-armed bandit problem [67, 68]. This type of problem assumes that the reward from each action is governed by an unknown probability distribution. By sampling from the set of actions, estimates for each one can be found with the goal of maximizing the cumulative reward. This method is also used on continuous optimization problems to select optimizers from a portfolio [69, 70]. Peng et. al. use a suite of population based optimizers in a portfolio framework to minimize the risk of solving a given problem. Each optimizer is allocated a portion of the total computation time to run with periodic migration of solutions between the algorithms [71]. Yuen et. al. propose a portfolio with multiple evolutionary algorithms, where selection is based

on the predicted performance of each optimizer at some future point, extrapolated from the cost curve for each optimizer [72, 73]. The selection of optimizers for use in a portfolio can also be automated to further improve performance and generality [74, 75]. Carchrae and Beck take a sequential approach where each optimizer is run for an allocated amount of time using the current best solution as a starting point. The order of the optimizers is random and the computation time allocated is updated after all optimizers have run based on the relative performance of each optimizer [76].

A similar approach has also been applied to the selection and combination of surrogate models for solving optimization problems. Surrogate models are mathematical models used to approximate an objective function that is relatively expensive to evaluate [77]. Common surrogate models used in optimization are polynomial regression [78, 79, 80], radial basis functions [81, 82], and kriging [52, 83]. Like numerical optimizers, no single surrogate model works well on all problems and the best model is often unknown *a priori*. Goel et. al. proposed an ensemble approach, combining multiple surrogate models using a weighted average of the model predictions based on different measures of the goodness of each model [84]. Dempster-Shafer theory can also be used to select and mix models according to various performance metrics [85, 86]. An ensemble approach was also applied to the EGO algorithm, where multiple surrogate models were used simultaneously to determine new points to sample in the parameter space, favoring models that more accurately fit the data [87].

1.1.4 Hybrid and Memetic Algorithms

One common feature of many portfolio approaches is that the individual algorithms are run independently, with little or no sharing of information. This means that many times, computation time is being used on algorithms which are not advancing towards the solution. One alternative is to combine multiple optimizers together to improve performance. These hybrid optimizers, often called Memetic algorithms, are

a class of optimizer which combine global search methods, such as genetic algorithms and particle swarm, with a local search method to improve the exploitation capability of the global method. Memetic algorithms follow a general framework with alternating phases of exploration using the global search method and exploitation with local search [88]. The term memetic algorithm originated with Moscato who proposed a new method for solving combinatorial problems which combined simulated annealing for local search and a genetic algorithm like crossover feature to introduce cooperation between population members [89]. These hybrid methods are extremely useful on continuous, multimodal problems where existing metaheuristics are inefficient at obtaining accurate solutions.

The most common optimizers used in memetic algorithms are particle swarm, differential evolution, and genetic algorithms. These optimizers have been paired with a number of local search techniques such as Nelder-Mead simplex, the Hooke-Jeeves method [90], multiple trajectory search [91], stochastic local search [92], iterated local search [39], and gradient based methods [93, 94]. Within the memetic algorithm framework, there exist numerous methods for interweaving local search into the global search methods. One simple approach is to alternate between optimizers sequentially, such as the method proposed by Wang, et al. which combines PSO with a gradient based optimizer in a two phase algorithm. The first phase consists of a gradient descent to search for a local minimum. With a local minimum found, the PSO phase commences and runs until a lower cost is found [95]. These sequential hybrids often contain reseeding or other strategies for maintaining diversity and preventing premature convergence [96, 97, 94, 98]. Local search can also be applied in parallel to a subset of the population to work together with the global search [99, 100, 101]. Alternatively, the local search can be embedded in the update rules for the population at every iteration. Rafajlowicz and Subudhi, Jena, and Gupta add a local search step to DE after a trial vector is generated [102, 103] while Noel replaces the cognitive

component of the particle velocity update in PSO with a gradient descent operation [104].

To better handle a diverse set of optimization problems, adaptive memetic algorithms have been proposed which select from multiple local search strategies. These adaptive methods use the diversity of the population to determine which local search algorithm to use at the given time. This approach helps balance exploration and exploitation of the function space and controls the diversity to prevent premature convergence [105, 106, 107, 108]. Other hybrid methods have been proposed which combine multiple metaheuristic optimizers. Shi, et al. developed a hybrid method with GA and PSO where each optimizer runs separately and the populations are shuffled periodically [109]. Shelokar, et al. combined PSO and ant colony optimization (ACO), using ACO as a local search method[110]. Another popular combination is PSO and DE. One approach invokes DE when PSO has stalled to prevent premature convergence of PSO [111]. A second approach incorporates the memory structure of PSO into the mutation and crossover operations in DE [112]. Different local search methods can also be combined to create a global search method [113].

1.2 Thesis Contributions

The objective of this thesis is to develop a new method for performing parameter estimation on smart projectile systems that is highly accurate and reliable for a wide range of problems. The core of parameter estimation is the numerical optimization strategy, with existing approaches utilizing optimizers which require reasonable initial estimates of the parameters to guarantee convergence. On new projectile configurations where initial estimates are unavailable, these methods would be unreliable when presented with complex estimation problems. This thesis proposes a new method for robust numerical optimization dubbed meta-optimization, an optimizer of optimizers. The goal of meta-optimization is to iteratively deploy a diverse set of optimizers in

an intelligent manner, improving accuracy and reliability across a wide set of optimization problems. The meta-optimizer must be able to select appropriate optimizers, ensure smooth transitions between different optimizers, prevent premature convergence in local minima, and improve optimizer parameters which are poorly tuned. The contributions of this work are expressed through the following three aims:

1. Analysis of the smart projectile parameter estimation problem to understand potential challenges for conventional numerical optimization algorithms.

2. Development of the individual components of the meta-optimizer with trade studies to determine the most effective approaches.

3. Evaluation of meta-optimization on performing parameter estimation on a new projectile configuration using simulated and experimental flight test data.

1.3 Thesis Outline

This dissertation is composed of 7 chapters which are briefly detailed below.

- **Chapter 1: Introduction and Background.** A review of parameter estimation and numerical optimization is presented. Also related work on existing projectile parameter estimation techniques, algorithm selection methods, and hybrid optimizers and memetic algorithms is discussed.

- **Chapter 2: Methodology for Smart Projectile Parameter Estimation.** An overview of the 6 degree of freedom smart projectile flight dynamics model and accompanying aerodynamics model is given. Example smart projectile system and spark range testing procedure are also described.

- **Chapter 3: Topology Analysis of Smart Projectile Parameter Estimation Problem.** An analysis of the topology of the smart projectile parameter

estimation problem is performed. Specific projectile dynamics and their effects on the problem topology are investigated. Local minima in parameter estimation problem are characterized and corroborated through simulation of projectile dynamics.

- **Chapter 4: Description of Meta-Optimization Framework.** The meta-optimization framework is developed with detailed descriptions of individual components. The individual optimizers used by meta-optimization are described.

- **Chapter 5: Benchmark Function Testing of Meta-Optimization.** In this chapter, the general behavior of meta-optimization on solving optimization problems is evaluated using benchmark optimization functions. The performance of meta-optimization is compared to the individual optimizers on these benchmark functions. Trade studies are conducted to understand the behavior of meta-optimization components and a robust configuration is determined. Also, meta-optimization is compared to state of the art optimizers on a suite of benchmark functions used in optimization competitions.

- **Chapter 6: Smart Projectile Parameter Estimation Results.** Robust meta-optimization configuration is employed to perform parameter estimation for a smart projectile system. Simulated trajectory data is used to evaluate capabilities of meta-optimization to accurately estimate parameters of a projectile system. The effects of measurement noise and number of measurements on the accuracy of the parameter estimates are explored through trade studies. Parameter estimation is performed using experimentally obtained spark range data.

- **Chapter 7: Conclusions and Future Work.** Conclusions are drawn based on the results from the previous chapters. In addition, recommendations for

future work are proposed.

CHAPTER 2
METHODOLOGY FOR SMART PROJECTILE PARAMETER ESTIMATION

Parameter estimation is the process of characterizing a model for a real world system based on data collected from the system. In the case of smart projectile systems, data is typically collected from flight testing in a spark range which provides measurements of the position and orientation of the projectile at discrete points along the flight. Trajectory data can also be generated using CFD/RBD virtual flyouts which utilize a CFD model to compute the aerodynamic forces and moment on the projectile, creating realistic trajectories without the need for expensive flight testing [7, 34]. Figure 2.1 shows an example of the type of data obtained from a spark range with a small number of measurements and some error in each measurement.

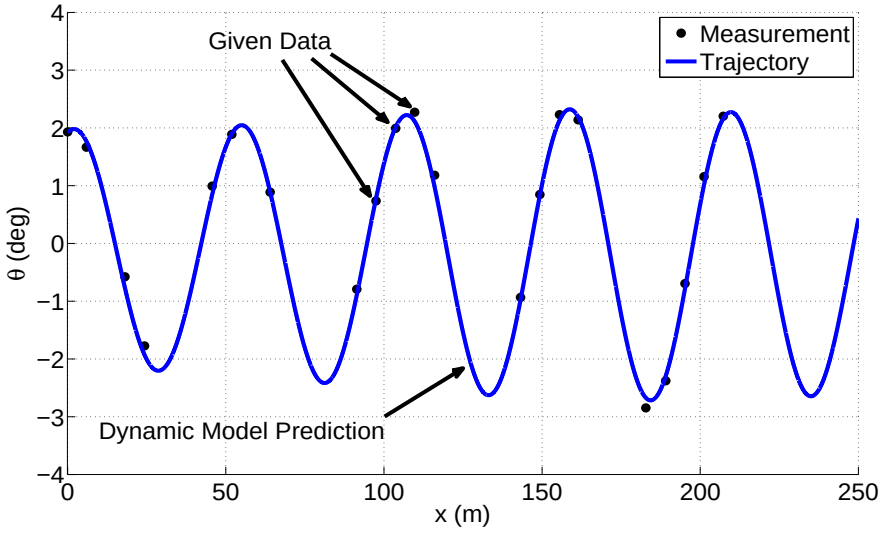


Figure 2.1: Example Spark Range Measurements of Projectile Pitch Angle

The parameter estimation problem is formulated using the Output Error Method (OEM) which defines a cost function as a function of the difference between given

trajectory data and a dynamic model prediction of the same trajectory. Trajectory data comes in the form of measured data from flight testing or CFD/RBD virtual flyouts. Predictions of the projectile trajectory are obtained from a six degree-of-freedom (6DOF) projectile flight dynamics representation which simulates the flight of the projectile given estimates of unknown parameters.

2.1 Output Error Method

The projectile parameter estimation problem is cast in the output error format. Under this formulation, a cost function is defined based on comparing estimated measurements to known data points along a trajectory. In this case, the estimated measurements are generated from a simulated trajectory generated using estimates for any unknown parameters. Figure 2.2 shows a schematic of how OEM computes the cost. Here, there are three measurements of θ at specific range locations. A trajectory is

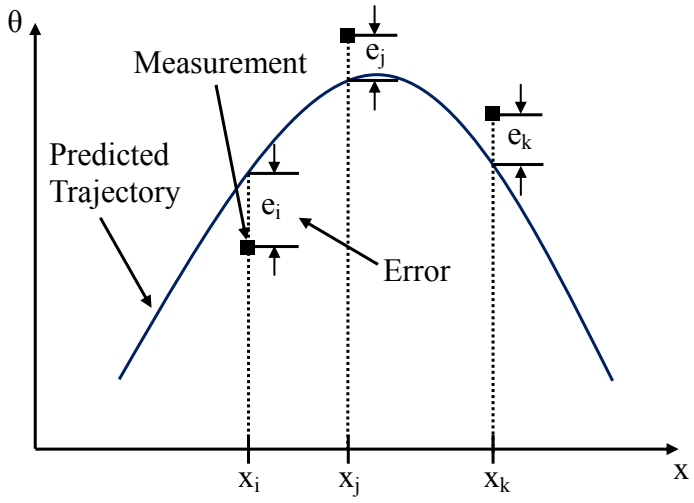


Figure 2.2: Example Schematic of OEM Cost Function Trajectory Errors

simulated using an estimate of the parameters of the system. The error between the predicted trajectory and the measurements is found by taking the difference between the measurement and the predicted trajectory at the same range or time value. These errors are computed for every measurement of every state and combined into a single

cost function. The typical cost function for the OEM is the sum squared error as defined in Eq. 2.1 [5, 114].

$$J(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{N}[\mathbf{z}_i - \mathbf{y}_i]^T G_i[\mathbf{z}_i - \mathbf{y}_i] + \sum_{i=1}^{n} r_i |\max(0, g(\mathbf{x}))|^3 \tag{2.1}$$

Here, $\mathbf{x}$ is the vector of unknown parameters, $\mathbf{y}$ is the measured state vector, $\mathbf{z}$ is the estimated measurement, N is the number of measurements, n is the number of parameters, G_i is the weighting and scaling matrix, r_i is the penalty coefficient, and $g(\mathbf{x})$ is the inequality constraint function which is positive when the inequality is not satisfied. The weighting and scaling matrix G_i is generated from three components. First, the state errors are scaled by the standard deviation of the measurement noise, ensuring all states contribute to the cost equally. Second, the states can be weighted against each other to limit the impact of certain states or to highlight others. Finally, weighting can be applied to each individual measurement along the trajectory in order to highlight certain behavior and improve the accuracy of the parameter estimates.

Each parameter is constrained to its search range by an exterior penalty function. In Eq. 2.1, this is represented by the function $g(\mathbf{x})$ which treats each boundary condition as an inequality constraint. This penalty function adds to the cost when a parameter exceeds its bounds. A cubic penalty function is chosen because the second derivative is zero on the boundary, providing sufficient smoothness for hill climbing based optimization algorithms. The magnitude of the penalty function is set to allow some exploration beyond the boundary while the overall cost remains high. As the cost is reduced, the boundary grows steeper relative to the current cost.

In addition to the cost function, the quality of fit for a given set of parameters is evaluated using the metric χ^2. For the jth state, χ^2 is computed using Eq. 2.2 [115]:

$$(\chi^2)^j = \sum_{i=1}^{n} \frac{(z_i^j - y_i^j)^2}{\sigma_j^2} \tag{2.2}$$

where σ_j is the standard deviation of the measurement noise for this state. Lower values indicate a better fit with the expected value of χ^2 for an optimal trajectory on the order of the number of measurements.

2.2 Projectile Flight Dynamics Model

The six degree of freedom rigid body flight dynamics simulation is the heart of the cost function calculation. This model predicts the position, orientation, velocity, and angular velocity of the projectile as a function of time over the trajectory. It includes gravity, aerodynamic, and control forces and moments. The orientation of the projectile body is given by the aerodynamic standard 3-2-1 Euler angle sequence as seen in Figure 2.3. This sequence defines a set of rotations from the Inertial (I) reference frame to the Body (B) reference frame.

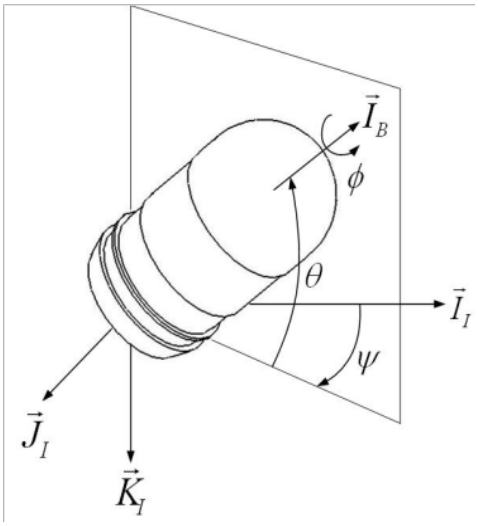


Figure 2.3: Projectile Orientation Definition

The motion of projectile is governed by a set of kinematic and dynamic equations. The kinematic equations of motion describe the changes in inertial position (x, y, z) and orientation (ϕ, θ, ψ) in terms of the body frame velocity (u, v, w) and angular

velocity (p, q, r). These equations are given in Eqs. 2.3 and 2.4 [116].

$$\begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{Bmatrix} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta s_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\theta c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{2.3}$$

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{2.4}$$

In these equations, the shorthand notation $s_\alpha = sin(\alpha)$, $c_\alpha = cos(\alpha)$, and $t_\alpha = tan(\alpha)$ is used. The dynamic equations of motion govern the motion of the body as a result of the forces and moments that act on the body. These equations are formed by equating the time derivative of the linear and angular momentum with the summation of the forces and moments about the center of mass of the projectile. The translational and rotational dynamics are given by Eqs. 2.5 and 2.6.

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = \frac{1}{m} \begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} - \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{2.5}$$

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = [I_B]^{-1} \left(\begin{Bmatrix} L \\ M \\ N \end{Bmatrix} - \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} [I_B] \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \right) \tag{2.6}$$

Here, m is the mass of the projectile, $[I_B]$ is the moment of inertia matrix, X, Y, Z are the total forces on the body expressed in the B frame, and L, M, N are the total moments about the center of mass expressed in the B frame.

The forces and moments depicted in Eqs. 2.5 and 2.6 are total forces and moments acting on the projectile which include aerodynamic (A), gravity (G) and control forces

(C). The total forces and moments are given in Eqs. 2.7 and 2.8.

$$\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} = \begin{Bmatrix} X_A \\ Y_A \\ Z_A \end{Bmatrix} + \begin{Bmatrix} X_G \\ Y_G \\ Z_G \end{Bmatrix} + \begin{Bmatrix} X_C \\ Y_C \\ Z_C \end{Bmatrix} \tag{2.7}$$

$$\begin{Bmatrix} L \\ M \\ N \end{Bmatrix} = \begin{Bmatrix} L_A \\ M_A \\ N_A \end{Bmatrix} + \begin{Bmatrix} L_C \\ M_C \\ N_C \end{Bmatrix} \tag{2.8}$$

The aerodynamic forces on the projectile are modeled using existing ballistic expansions with known coefficients given in Eq. (2.9).

$$\begin{Bmatrix} X_A \\ Y_A \\ Z_A \end{Bmatrix} = -Qd \begin{Bmatrix} C_{X_0} + C_{X_2}\frac{v^2+w^2}{V^2} \\ C_{Y_0} + C_{N_\alpha}\frac{v}{V} - C_{Y_{p\alpha}}\frac{w}{V}\frac{pd}{2V} \\ C_{Z_0} + C_{N_\alpha}\frac{W}{V} + C_{Y_{p\alpha}}\frac{v}{V}\frac{pd}{2V} \end{Bmatrix} \tag{2.9}$$

Here, C_{X0} is the zero yaw drag coefficient, C_{X2} is the dynamic drag coefficient, C_{Y0} and C_{Z0} are the zero yaw normal force coefficients, $C_{N\alpha}$ is the normal force coefficient, $C_{Yp\alpha}$ is the Magnus force coefficient, d is the aerodynamic diameter, V is the total velocity, and Q is the dynamic pressure given by $Q = \frac{1}{8}\pi\rho d^2V^2$. The air density ρ is a function of the altitude h of the projectile as specified by:

$$\rho = 1.22566578494891(1.0 - 0.0000225696709h)^{4.258} \tag{2.10}$$

The weight of the projectile expressed in the projectile body frame is given by:

$$\begin{Bmatrix} X_G \\ Y_G \\ Z_G \end{Bmatrix} = \begin{Bmatrix} -s_\theta \\ s_\phi c_\theta \\ c_\phi c_\theta \end{Bmatrix} mg \tag{2.11}$$

The total aerodynamic moments are given in Eq. 2.12:

$$\begin{Bmatrix} L_A \\ M_A \\ N_A \end{Bmatrix} = Qd^2 \begin{Bmatrix} C_{l_0} + C_{l_p}\frac{pd}{2V} \\ C_{m_0} + C_{m_\alpha}\frac{w}{V} + C_{n_{p\alpha}}\frac{v}{V}\frac{pd}{2V} + C_{m_q}\frac{qd}{2V} \\ C_{n_0} - C_{m_\alpha}\frac{v}{V} + C_{n_{p\alpha}}\frac{w}{V}\frac{pd}{2V} + C_{m_q}\frac{rd}{2V} \end{Bmatrix} \tag{2.12}$$

where C_{l0} is the roll moment coefficient, C_{lp} is the roll damping moment coefficient, C_{m0} and C_{n0} are the zero yaw pitch and yaw moment coefficients, $C_{m\alpha}$ is the pitching moment coefficient, $C_{np\alpha}$ is the Magnus moment coefficient, and C_{mq} is the pitch damping coefficient. These aerodynamic coefficients are typically functions of Mach number. The control forces X_C, Y_C, Z_C and control moments L_C, M_C, N_C from Eqs. 2.7 and 2.8 are application dependent, with each control method having unique effects on the projectile dynamics. With all of the applied forces and moments computed, Eqs. 2.3-2.6 are numerically integrated forward in time using a 4th-order Runge-Kutta method to generate a trajectory for a projectile configuration.

For most projectile parameter estimation problems, only the aerodynamic coefficients are estimated, but any parameter used in the model could be estimated as well. This could include parameters such as the projectile mass m, the moments of inertia $[I_B]$, and the projectile diameter d. The projectile aerodynamic model presented here has 12 coefficients that need to be estimated. However, for symmetric projectiles, C_{Y_0}, C_{Z_0}, C_{m_0}, and C_{n_0} are zero. In addition, for finned projectiles, it is assumed that the Magnus coefficients, $C_{Yp\alpha}$ and $C_{Np\alpha}$, are zero as roll rates tend to be small, resulting

in negligible Magnus effects. This reduces the number of aerodynamic coefficients for finned projectiles to seven: C_{X0}, C_{X2}, $C_{N\alpha}$, C_{l0}, C_{lp}, $C_{m\alpha}$, and C_{mq}.

2.3 Spark Range Flight Testing

The data used in this work to perform parameter estimation was collected in the spark range at the U.S. Army Research Laboratory Transonic Experimental Facility. The spark range consists of twenty five orthogonal spark shadowgraph stations arranged in groups of five along an approximately 200 m range. From each set of images gathered from the spark stations, position and angular measurements are taken. The time of each image is also recorded. Due to issues in the collection and processing of the images, about 5-10% of measurements in a given shot cannot be made. To generate synthetic spark range data, a trajectory is simulated from launch with the position, orientation, and time recorded at the range location corresponding to each spark station. Noise is then added to the measurements to mimic real world data. Typical measurement errors have a standard deviation of 3 mm for position measurements and 0.1° for angle measurements. To reduce the impact of measurement noise and improve the accuracy of the parameter estimates, multiple trajectories can be combined together within a single cost function.

The estimated measurements generated by the simulation within the cost function are obtained by noting when the simulated trajectory passes the time corresponding to a measurement and interpolating each state back to this time. It is assumed that the time measurements are known exactly as any measurement errors are significantly smaller than errors in the state data. Three techniques are used to handle the initial conditions for each predicted trajectory. In the first method, the simulated trajectories are started from the first spark station, using the first set of measurements as initial conditions. This simplifies the estimation process as only the initial velocities and angular velocities must be estimated. While a small amount of error may be

added due to noise in the initial measurements, the effect on the quality of the estimates is minimal as long as the measurement error is low. Alternatively, the initial position and orientation can be estimated. However, this adds additional parameters that must be estimated, making the problem more difficult for the optimizers. In this case, a small range based on the typical measurement error is defined around each of the first measurements, restricting the potential values of the initial conditions.

The final approach instead begins the simulations from launch. Typically, position and orientation of the gun are fixed, but slight variations in velocity and angular velocity may occur as the projectile leaves the barrel. Also, the initial roll angle is unknown and must be estimated. The initial time must also be estimated as the time measurements are not calibrated to launch. This approach is necessary when the projectile initial conditions must be constrained to prevent coupling with other parameters. The challenge of this approach is a lack of observability of the projectile behavior prior to the first measurement. The computation time for each trajectory is also increased by 25% due to the longer simulated flight time.

Two models are used for the aerodynamic coefficients. For individual trajectories, the coefficients can be assumed to be constant with respect to Mach number as the coefficients do not vary significantly over a single trajectory. When working with multiple trajectories at different Mach numbers, it is necessary to use a Mach varying aerodynamic model to best approximate the real world aerodynamics. A linear model provides a reasonable approximation as the coefficients are roughly linear over small ranges in Mach number, particularly in the supersonic regime. To characterize this model, an upper and lower Mach number limit are defined based on the range of expected Mach numbers for the trajectory data. The aerodynamic coefficients used in the simulations are computed using:

$$C(M) = C_{lo} + (C_{hi} - C_{lo})\frac{M - M_{lo}}{M_{hi} - M_{lo}} \tag{2.13}$$

where C_{lo} and C_{hi} are estimates of the coefficient at the Mach number limits.

2.4 Example Smart Projectile System

The smart projectile configuration considered in this work is a finned projectile equipped with a single set of microspoilers. The base projectile is a 30 mm Army-Navy Finner (ANF). This round is a popular testbed for new control mechanisms as it has been studied extensively by the community with well documented aerodynamics. This projectile configuration is shown in Figure 2.4. The round is axi-symmetric with four fins at the rear of the body. The mass properties of the standard 30 mm ANF are given in Table 2.1 and the nominal aerodynamic coefficients obtained from the Projectile Design/Analysis System (PRODAS) aeroprediction tool [117] are given in Table 2.2.

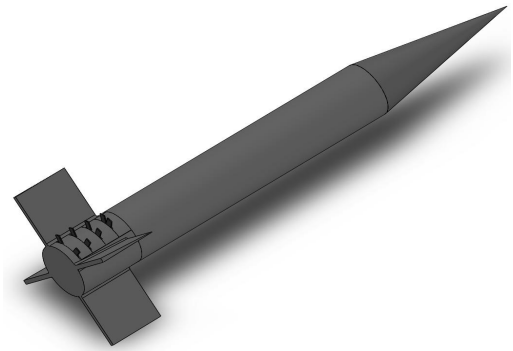


Figure 2.4: 30 mm Army-Navy Finner with Microspoilers

Table 2.1: Physical Properties of 30 mm Army-Navy Finner

Physical Property	Value
mass (kg)	1.5887
diameter (m)	0.03
center of gravity - I_B (m)	0.135
I_{XX} (kg-m^2)	0.000192388
$I_{YY} = I_{ZZ}$ (kg-m^2)	0.00987337

Table 2.2: Standard PRODAS ANF Aerodynamic Coefficients

Mach	C_{X0}	C_{X2}	$C_{N\alpha}$	C_{l0}	C_{lp}	$C_{m\alpha}$	C_{mq}
0.01	0.4698	3.32	12.549	0.086875	-8.0214	-32.771	-396.7
0.4	0.4723	3.32	13.716	0.097	-8.9549	-36.366	-417.1
0.6	0.4735	3.32	14.315	0.10225	-9.4336	-38.136	-427.6
0.7	0.5166	3.56	15.103	0.109	-10.056	-40.228	-437.4
0.75	0.5382	3.68	15.497	0.11237	-10.366	-41.238	-442.2
0.8	0.5597	3.8	15.892	0.11588	-10.677	-42.224	-447.1
0.85	0.6265	4.04	16.524	0.12125	-11.176	-43.865	-456.9
0.875	0.6598	4.16	16.84	0.124	-11.425	-44.666	-461.7
0.9	0.6932	4.28	17.156	0.12675	-11.675	-45.456	-466.6
0.925	0.7303	4.52	17.476	0.12937	-11.925	-46.218	-471.2
0.95	0.7673	4.76	17.795	0.13212	-12.174	-46.968	-475.9
0.975	0.8366	5.15	18.117	0.13487	-12.423	-47.708	-481
1	0.9059	5.53	18.439	0.13763	-12.671	-48.436	-486.2
1.025	0.9299	5.93	18.716	0.14013	-12.903	-50.85	-514
1.05	0.9539	6.34	18.993	0.14275	-13.136	-53.317	-541.8
1.1	0.9086	7.11	19.599	0.14788	-13.607	-58.399	-608.7
1.2	0.8308	8.12	18.522	0.138	-12.704	-54.511	-592.2
1.35	0.7769	7.54	16.661	0.12113	-11.152	-47.185	-553.7
1.5	0.7307	6.94	14.751	0.1035	-9.5432	-39.4	-515.5
1.75	0.6641	6.34	14.337	0.099	-9.1227	-36.855	-507.8
2	0.6174	5.74	10.762	0.066875	-6.1794	-22.225	-406
2.25	0.5765	5.43	10.133	0.061	-5.6344	-19.143	-388.9
2.5	0.5356	5.13	9.503	0.055	-5.0894	-16.048	-371.8
3	0.4742	4.4	8.161	0.04375	-4.0529	-10.366	-331.7
3.5	0.4458	4.06	7.121	0.035125	-3.2611	-6.535	-296.4
4	0.4174	3.71	6.081	0.0265	-2.4692	-2.698	-261

2.4.1 Microspoiler Control Mechanism

The microspoiler mechanism consists of four sets of small protrusions which extend and retract from the projectile body with a prescribed motion and a set frequency. As seen in Figure 2.4, the microspoilers are placed between the rear fins of the projectile and are oriented such that they are on the top of the projectile body. Microspoilers add additional forces and moments acting on the projectile and are incorporated into the equations of motion of the projectile through the control forces and moments in Eqs. 2.5 and 2.6. The magnitude of the forces and moments at a given time are a function of how much of the microspoilers are exposed. The coefficients are functions of Mach number with values obtained from CFD analysis of the microspoilers given in Table 2.3 [118].

Table 2.3: Microspoiler Coefficients Obtained from CFD Analysis

Mach	δ_A	δ_N	δ_m
1.1	-7.7	13.7	1.51
1.5	-13.1	34.4	4.13
2.0	-18.5	47.8	5.92
2.5	-23.7	60.6	7.56
3.0	-29.0	73.7	9.25
4.0	-43.1	103.9	13.05
5.0	-59.2	135.2	17.00

The mechanism is designed to spin at a set rate with a spin up period at launch. A first order model based off of bench testing of the mechanism is used to approximate the spin rate as it reaches steady state. The expansion for the microspoiler forces and

moments is given by:

$$\begin{Bmatrix} X_C \\ Y_C \\ Z_C \end{Bmatrix} = \lambda \left(\omega_0 + \Omega_0 \left(t + \tau_{ms} e^{-\frac{t}{\tau_{ms}}} - \tau_{ms} \right) \right) \begin{Bmatrix} \delta_A \\ 0 \\ \delta_N \end{Bmatrix} \tag{2.14}$$

$$\begin{Bmatrix} L_C \\ M_C \\ N_C \end{Bmatrix} = \lambda \left(\omega_0 + \Omega_0 \left(t + \tau_{ms} e^{-\frac{t}{\tau_{ms}}} - \tau_{ms} \right) \right) \begin{Bmatrix} 0 \\ \delta_m - \delta_N \Delta S L_{CG} \\ 0 \end{Bmatrix} \tag{2.15}$$

The effect of the microspoilers is parameterized by six parameters: the axial force coefficient δ_a, the normal force coefficient δ_N, the pitching moment coefficient δ_m, the initial microspoiler phase ω_0, the microspoiler spin rate Ω_0, and the microspoiler time constant τ_{ms}. Note that by assuming the microspoiler forces and moments are acting about the projectile center of mass, the δ_N term in Eq. 2.15 goes to zero. The magnitude function λ is determined by the design of the microspoiler mechanism and is given in Figure 2.5.

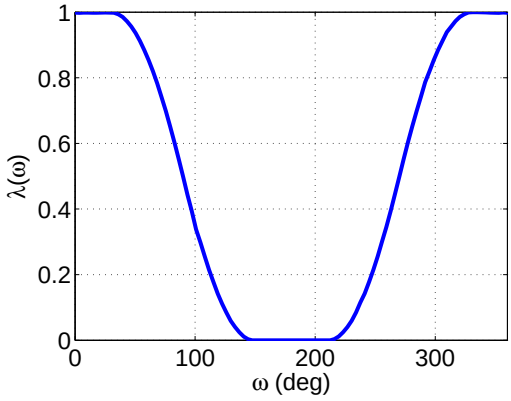


Figure 2.5: Microspoiler Actuation Profile

2.4.2 Example Trajectory Results

To understand the dynamics of this projectile and the effects of the microspoilers, two trajectories were simulated using the model described in this chapter, one with only the base ANF and the other with the ANF equipped with microspoilers. Both trajectories were fired with an initial velocity of 1023 m/s or Mach 3. The first trajectory was given an initial pitch rate of 20 rad/s in order to fully excite the dynamics of the projectile. The state trajectories for both cases are shown in Figures 2.6-2.18. For reference, the total angle of attack of the projectile $\bar{\alpha}$ is given by:

$$\bar{\alpha} = \tan^{-1}\left(\frac{\sqrt{v^2 + w^2}}{u}\right) \tag{2.16}$$

The projectile dynamics can be broken up into three different groups that are mostly decoupled from one another: the axial dynamics, the roll dynamics, and the epicyclic dynamics. The axial dynamics govern the change in axial velocity u and are primarily driven by the axial force X and has little influence from the other states. In Figure 2.12, the overall trend in u is dictated by C_{X0} while C_{X2} causes the small oscillations in velocity. However, these small changes in velocity are nearly imperceptible in the range trajectory in Figure 2.6. When working with spark range data that only has measurements of x position, changes in the predicted trajectories due to C_{X0} will be more pronounced than C_{X2}.

Like the axial dynamics, the roll dynamics are largely decoupled from the other dynamics as the impact of other states on the roll rate is minimal. This is due to the roll moment L being significantly larger than the contributions from q and r in Eq. 2.6. The roll rate profile is controlled by the roll generation coefficient C_{l0} and the roll damping coefficient C_{lp}. These parameters dictate the rate of increase in roll rate as well as the steady state roll rate which is controlled by the ratio C_{l0}/C_{lp}. For this trajectory, the roll rate shown in Figure 2.15 indicates that this round has not

yet reached steady state by the end of the flight.

Finally, the epicyclic dynamics cover the remaining projectile body velocities and angular velocities. These dynamics govern the periodic motion of the projectile and play an important role in the stability of the projectile. The coefficients associated with the epicyclic dynamics are $C_{N\alpha}$, $C_{m\alpha}$ and C_{mq}. The effects of the epicyclic dynamics can be clearly seen in Figures 2.7, 2.8, 2.10, and 2.11 where these states oscillate at a consistent frequency. The large initial pitch rate generates large angles of attack, exciting the dynamics and producing large oscillations. For a fin stabilized projectile, the projectile is statically stable such that angular rates will naturally decay over time. This is seen in Figure 2.18 where the total angle of attack decays from 9° after launch to only 2° at the end of the flight. The angular motion will also decay as the roll rate increases due to coupling between the roll rate and the pitch and yaw rates seen in Eq. 2.6.

The microspoilers add an additional axial force, normal force, and pitching moment to the projectile. The general effect on the axial dynamics is a slightly steeper decrease in u with the active microspoiler trajectory losing about 10 m/s more by the end of the flight. The most pronounced effects from the microspoilers are seen in Figures 2.7 and 2.8. While the base round had almost no motion in y, the micospoilers steer the round 0.5 m to the right. The microspoilers also push the round towards the ground with a change of 0.5 m from the launch height. The microspoilers also generate large perturbations in angle of attack, achieving a maximum of about 7.5°. Because the microspoilers spin independently of the projectile roll rate, the forces and moments are not acting in a consistent direction and are generally out of phase with the epicyclic dynamics. This produces interesting interactions with the projectile dynamics. For example, looking at Figure 2.10, the oscillations in θ decay from 100 m to about 200 m before the oscillations increase again. Another example of these interactions is seen in Figure 2.18 where the angle of attack increase after

275 m while the base round continues to decrease. This behavior demonstrates how the microspoilers are able to influence the projectile trajectory throughout the entire flight.

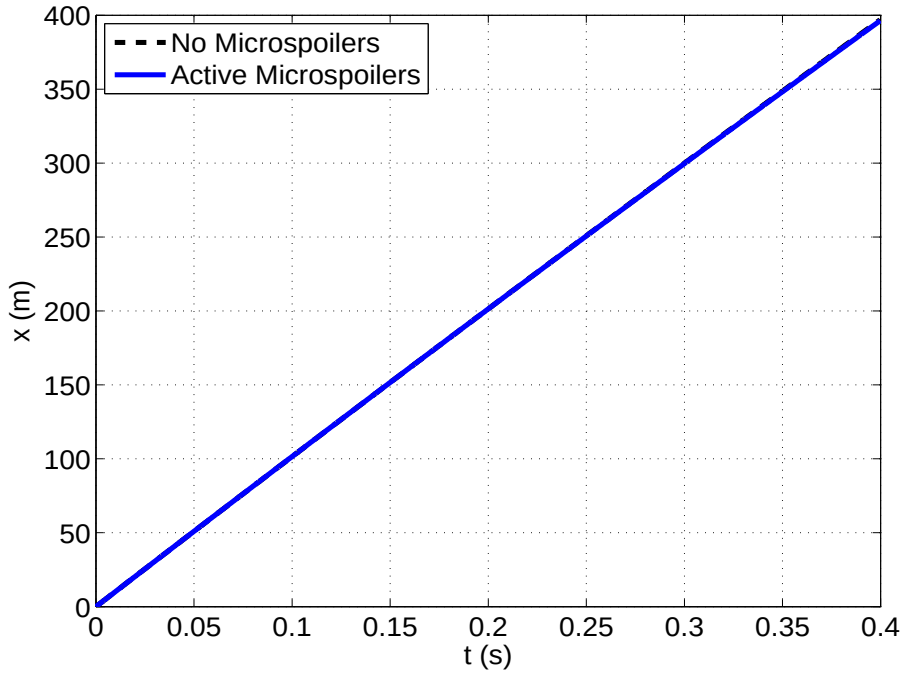


Figure 2.6: Example Trajectory Inertial-X Position vs. Time

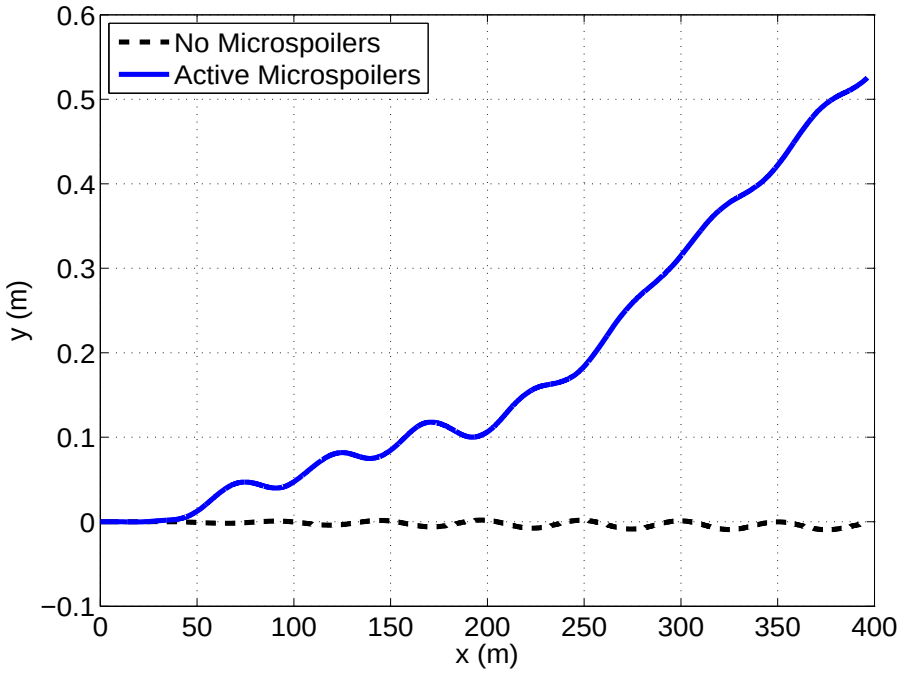


Figure 2.7: Example Trajectory Inertial-Y Position vs. Range

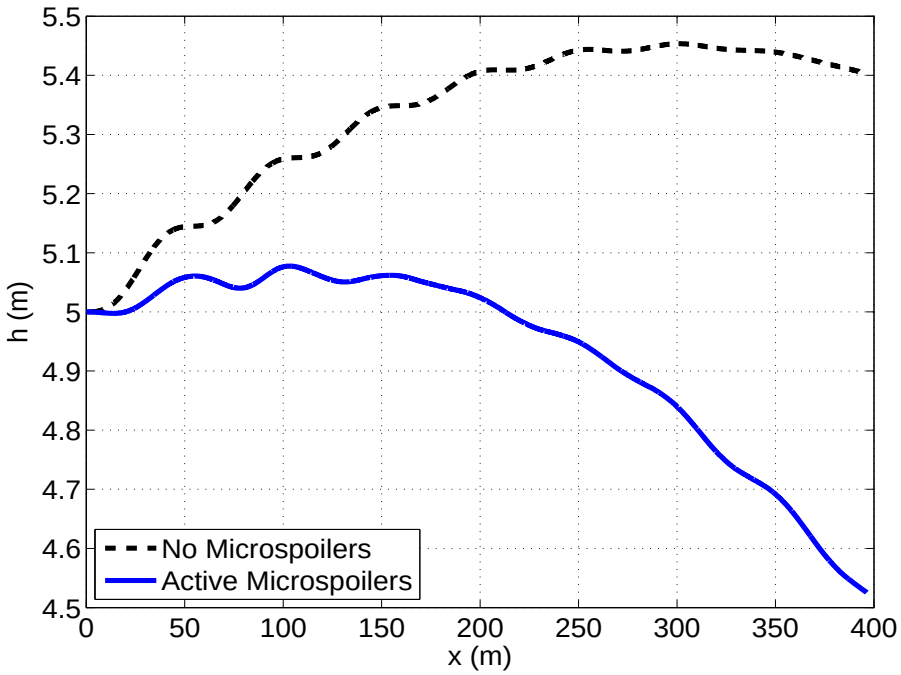


Figure 2.8: Example Trajectory Altitude vs. Range

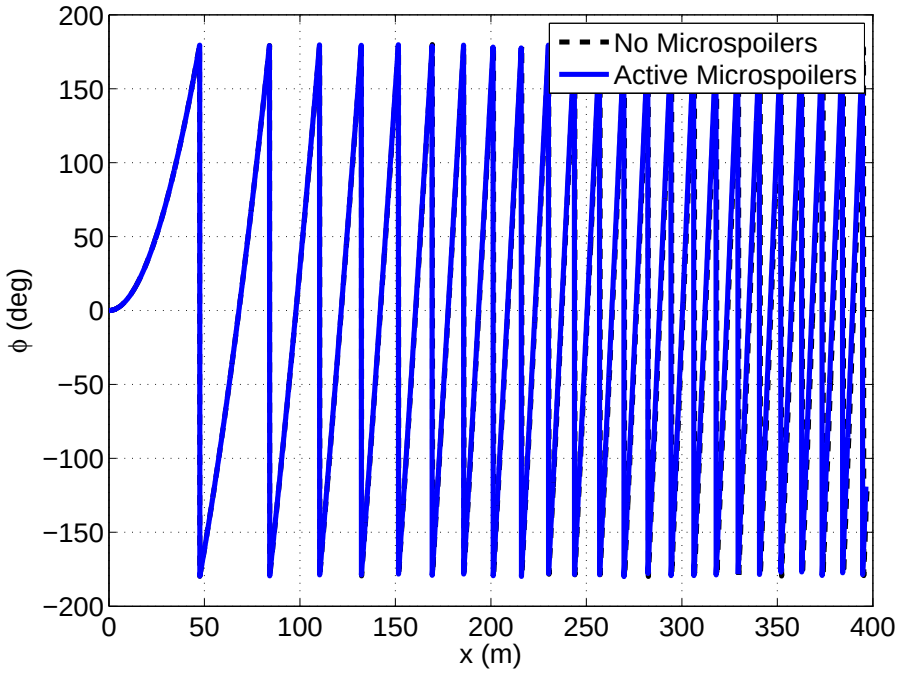


Figure 2.9: Example Trajectory Roll Angle vs. Range

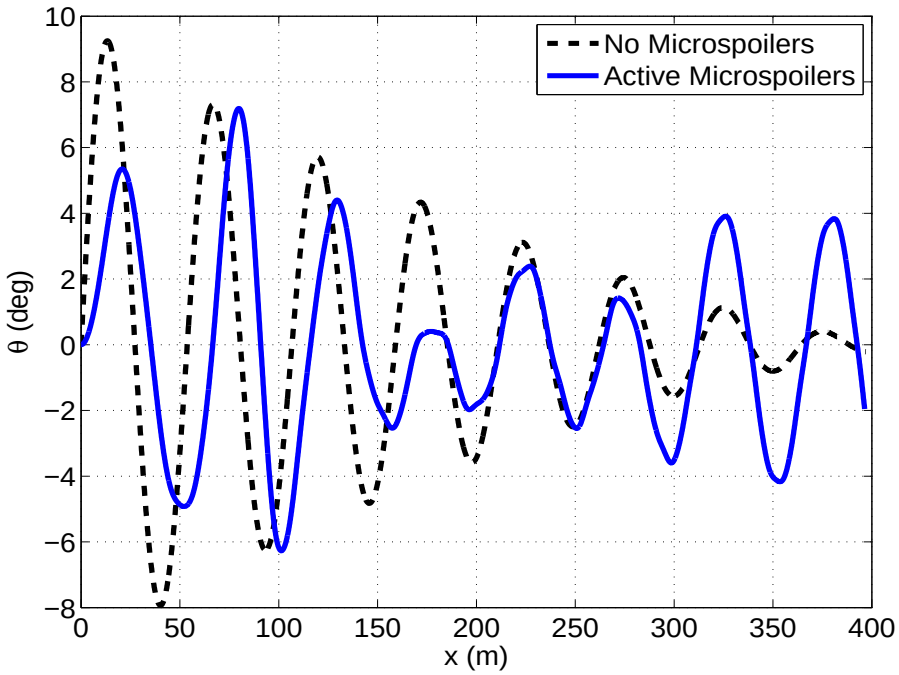


Figure 2.10: Example Trajectory Pitch Angle vs. Range

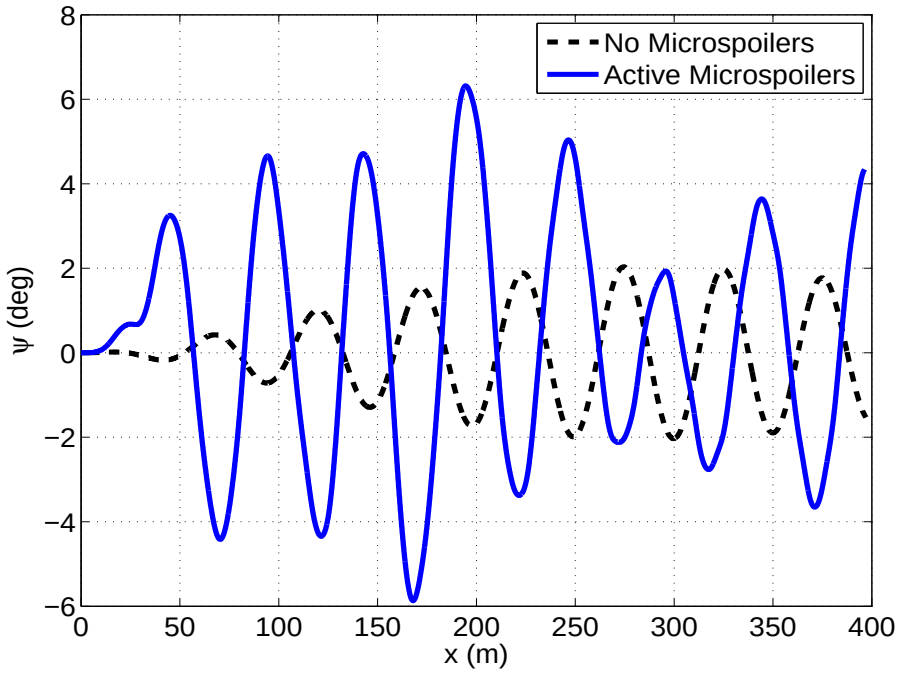


Figure 2.11: Example Trajectory Yaw Angle vs. Range

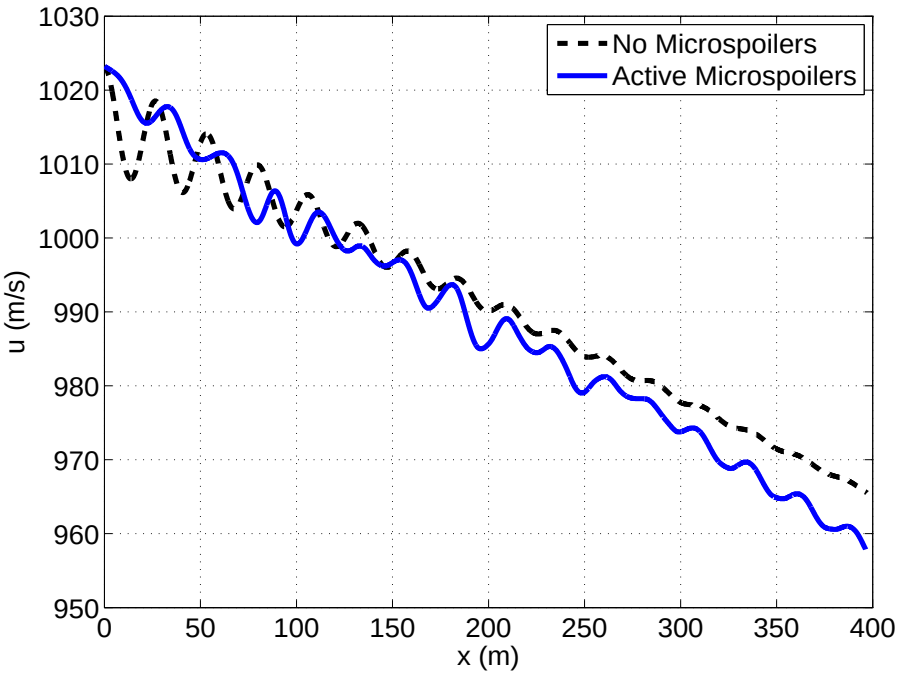


Figure 2.12: Example Trajectory Body X Velocity vs. Range

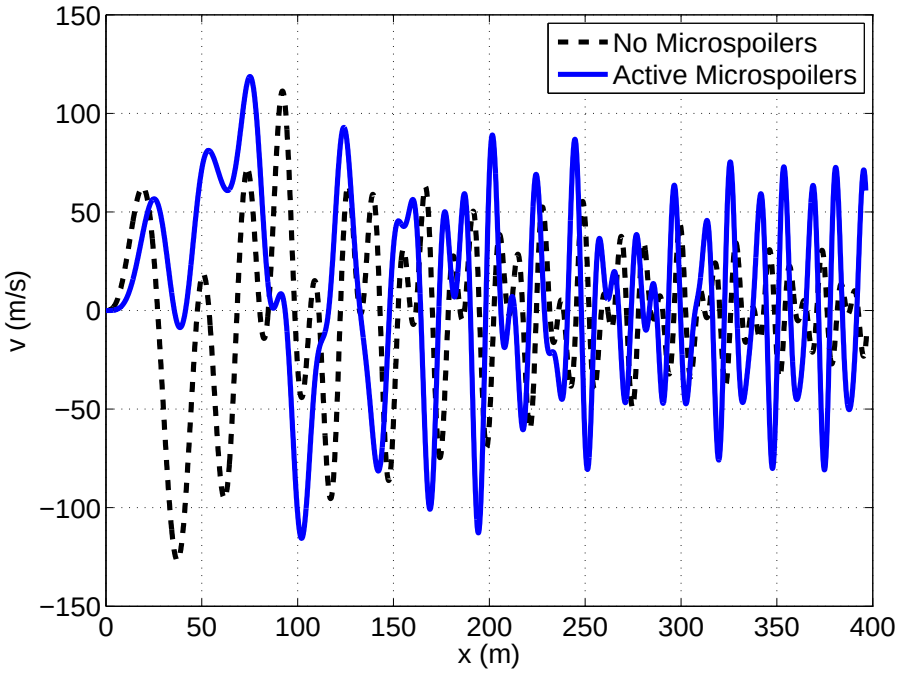


Figure 2.13: Example Trajectory Body Y Velocity vs. Range

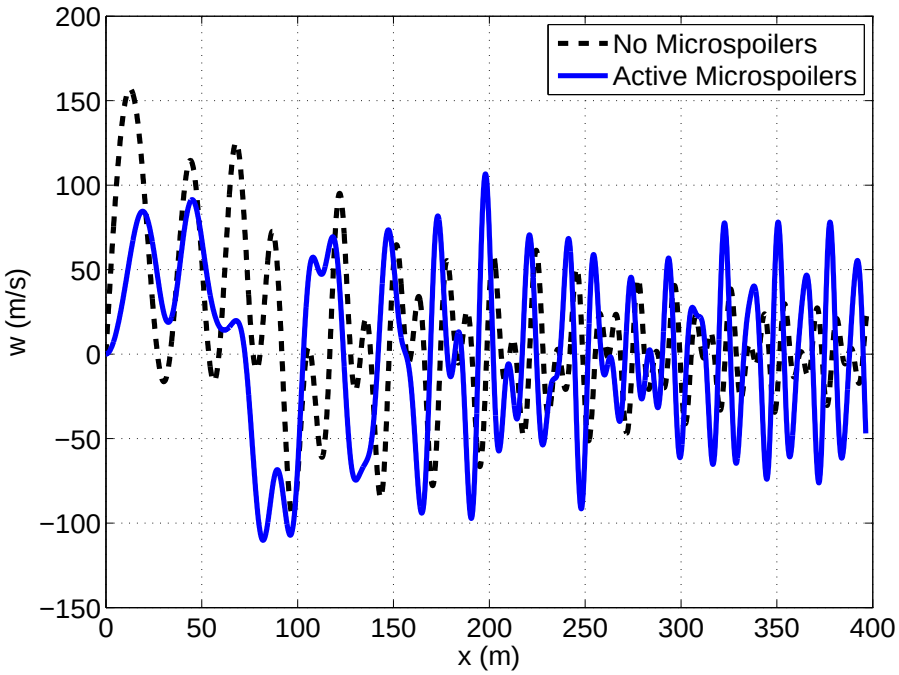


Figure 2.14: Example Trajectory Body Z Velocity vs. Range

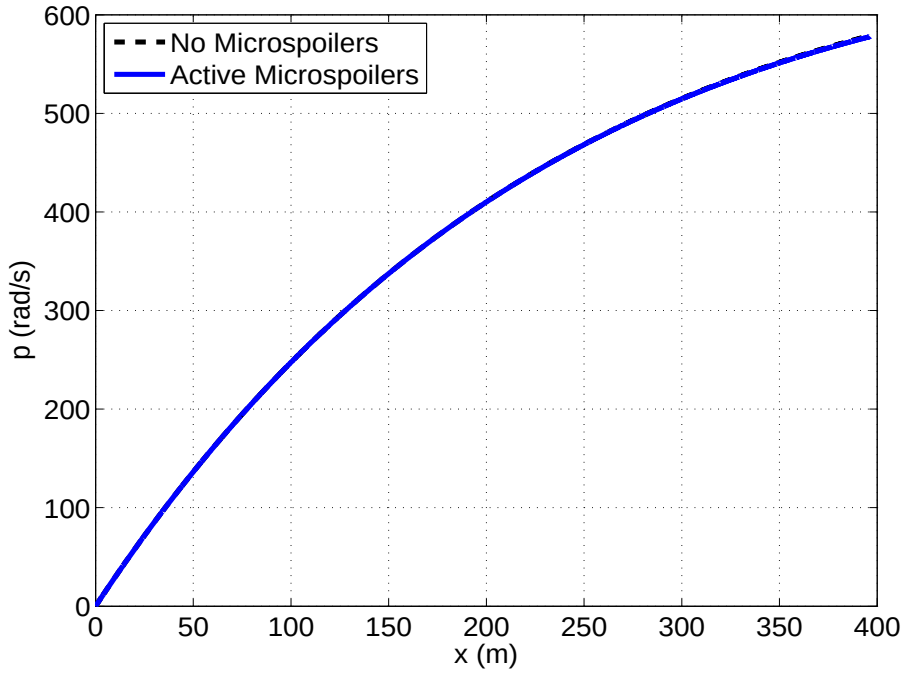


Figure 2.15: Example Trajectory Roll Rate vs. Range

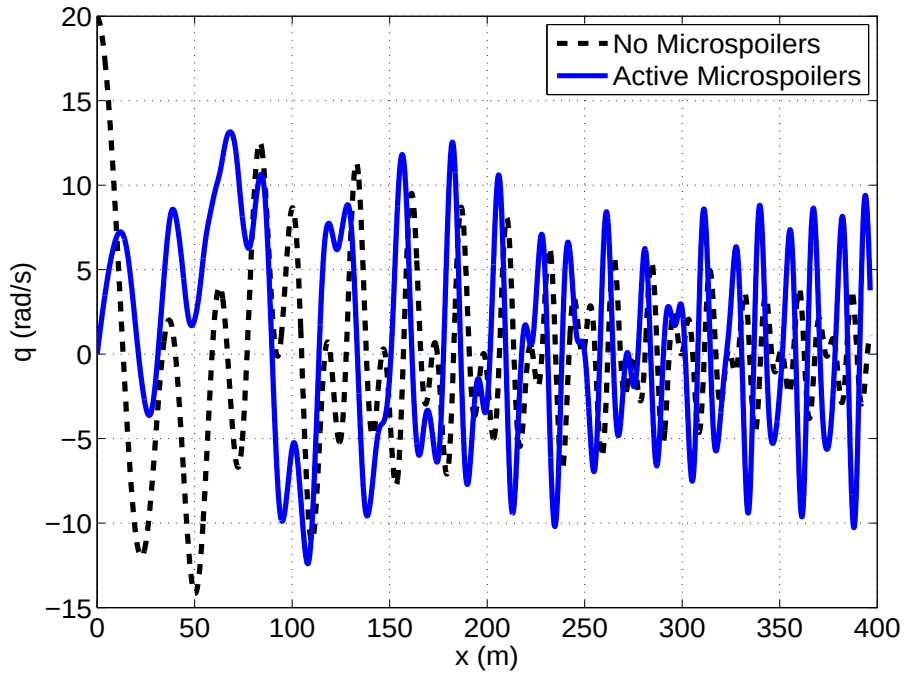


Figure 2.16: Example Trajectory Pitch Rate vs. Range

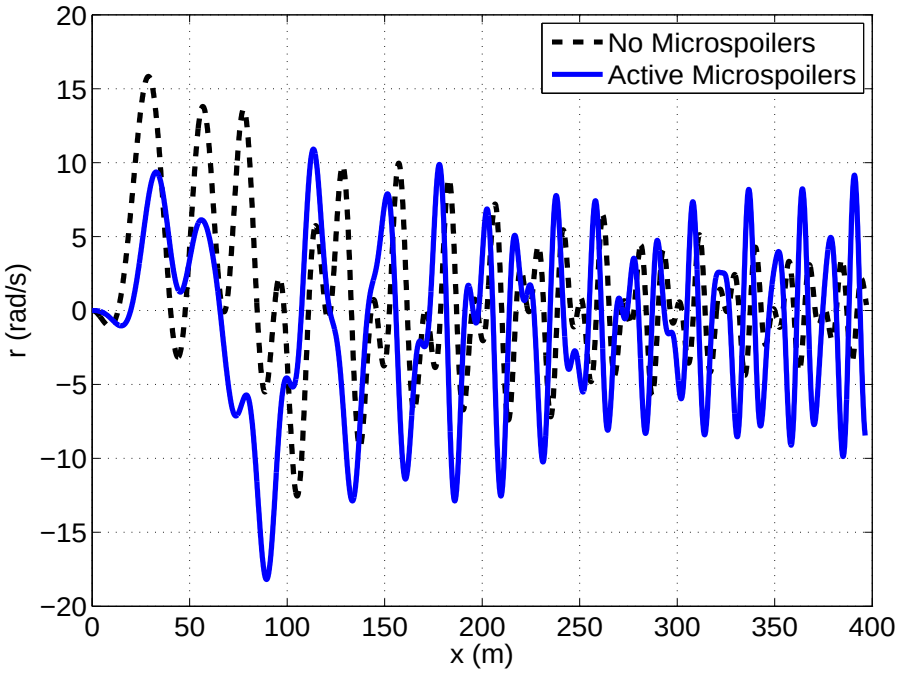


Figure 2.17: Example Trajectory Yaw Rate vs. Range

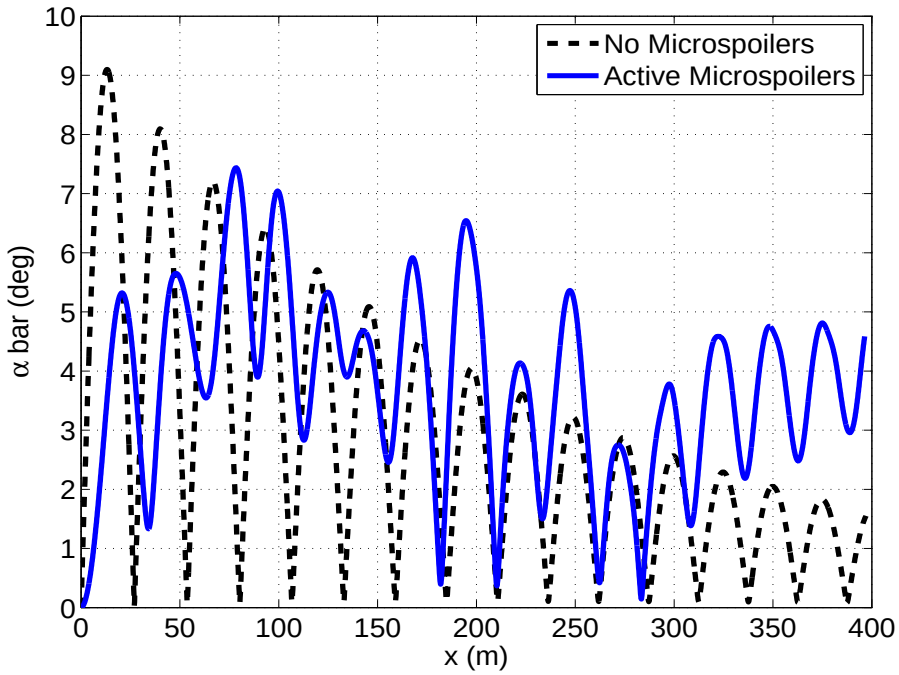


Figure 2.18: Example Trajectory Total Angle of Attack vs. Range

CHAPTER 3
TOPOLOGY ANALYSIS OF SMART PROJECTILE PARAMETER ESTIMATION PROBLEM

The structure and topology of an optimization problem has a significant impact on which optimizers will succeed and which will struggle to solve the problem. Unlike benchmark functions which have a defined mathematical form that can be easily visualized and analyzed, more practical optimization problems provide very little intuition on which optimizers are most appropriate. In the case of the smart projectile parameter estimation problem, the high dimensionality and complexity of the objective function makes it a difficult problem to analyze. However, valuable insight can be obtained by observing the nature of this problem in a range of situations to understand the underlying topology of the associated optimization problem.

The analysis performed in this chapter is based on synthetic spark range data generated for the ANF with microspoilers described in Section 2.4. The synthetic measurements are obtained using the procedure in Section 2.3. Individual trajectories are considered with aerodynamic coefficients held constant to the values from Table 2.2 at Mach 3. For any single trajectory, the aerodynamic coefficients are generally considered to be constant as Mach number does not vary significantly over the duration of the flight. The cost function in Eq. 2.1 was used with the trajectory prediction simulations starting from the first measurement. The states x, y, z, ϕ, θ, and ψ can be included in the cost function with all states and measurements weighted equally. Unless otherwise noted, no noise is added to the synthetic measurements and no bounds are placed on the parameters. Typical search ranges for each parameter are given in Table 3.1. Note, the nominal values for the initial conditions are representative values for these parameters as the initial conditions vary between

trajectories. Two methods are used to analyze the topology of the smart projectile parameter estimation problem. The first method generates cost landscapes at different cross sections of the parameter space. The second method uses a local optimizer to search the parameter space to detect the location of local minima.

Table 3.1: Typical Search Range for Estimated Projectile Parameters

Parameter	Nominal Value	Lower Bound	Upper Bound
C_{X0}	0.4742	0.3	0.6
C_{X2}	4.4	3.0	6.0
$C_{N\alpha}$	8.161	7.0	9.5
C_{l0}	0.04375	0.03	0.06
C_{lp}	-4.0529	-6.0	-3.0
$C_{m\alpha}$	-10.366	-15.0	-5.0
C_{mq}	-331.7	-500.0	-200.0
u_0 (m/s)	1000.0	800.0	1100.0
v_0 (m/s)	50.0	-200.0	200.0
w_0 (m/s)	50.0	-200.0	200.0
p_0 (rad/s)	100.0	50.0	150.0
q_0 (rad/s)	10.0	-20.0	20.0
r_0 (rad/s)	10.0	-20.0	20.0

3.1 Parameter Cross Section Landscape Analysis

While it is impossible to visualize the topology of the smart projectile parameter estimation problem over the entire parameter space, it is insightful to observe the behavior of the cost function on cross sectional slices of regions of interest. To generate these slices, two parameters are varied over a 1000x1000 grid with the remaining parameters held fixed to known values. The cost function is evaluated at every grid point, generating a set of contours that are visualized. The ranges for each investigated parameter extend 10% beyond the standard search ranges in each direction. In

addition, the gradient of the cost function is computed along the search boundaries to indicate the slope of the cost function along the boundary. The behaviors of interest are the roll, epicyclic (v, w, q, r), and microspoiler dynamics.

3.1.1 Roll Dynamics Analysis

The parameters associated with roll are the roll generation coefficient (C_{l0}), roll damping coefficient (C_{lp}), and initial roll rate (p_0). As discussed in Section 2.4, the roll dynamics are decoupled from the other states. This allows for easy investigation of the roll dynamics. Roll data is typically wrapped, meaning all angles are restricted to -180° to 180°. Sometimes roll data is unwrapped to an absolute angle. With unwrapped roll measurements, fitting the roll parameters is very simple and the associated optimization problem is unimodal. However, when the roll measurements are wrapped, the topology becomes multimodal with complex character and a number of local minima.

To observe this phenomenon, a case is constructed with only the roll measurements included in the cost function, restricting the optimization problem to only the roll dynamics. The first cross section is taken about C_{l0} and C_{lp} with the remaining parameters held fixed. The search range for C_{l0} is from 0.03 to 0.06 and the search range for C_{lp} is from -6 to -3. Figure 3.1 shows the contours of the cost function over these two parameters. In the figure, the box represents the typical search bounds on these parameters. The boundary is color coded to indicate if the gradient is pointing into or out of the search space and the arrows represent the direction of the gradient on the boundary. The black lines denote the nominal values of the parameters with the global minimum at their intersection. Overall, this landscape is highly multimodal with a few local minima within the search bounds as well as some outside of the search space that would attract optimizers out of the search space. A deep, narrow valley occurs along the nominal ratio of C_{l0}/C_{lp} which contains the global minimum.

Without a reasonable estimate for C_{l0}/C_{lp}, an optimizer may struggle to reach the valley, instead drawn away into a local minima. However, these local minima are very shallow compared to the global minimum.

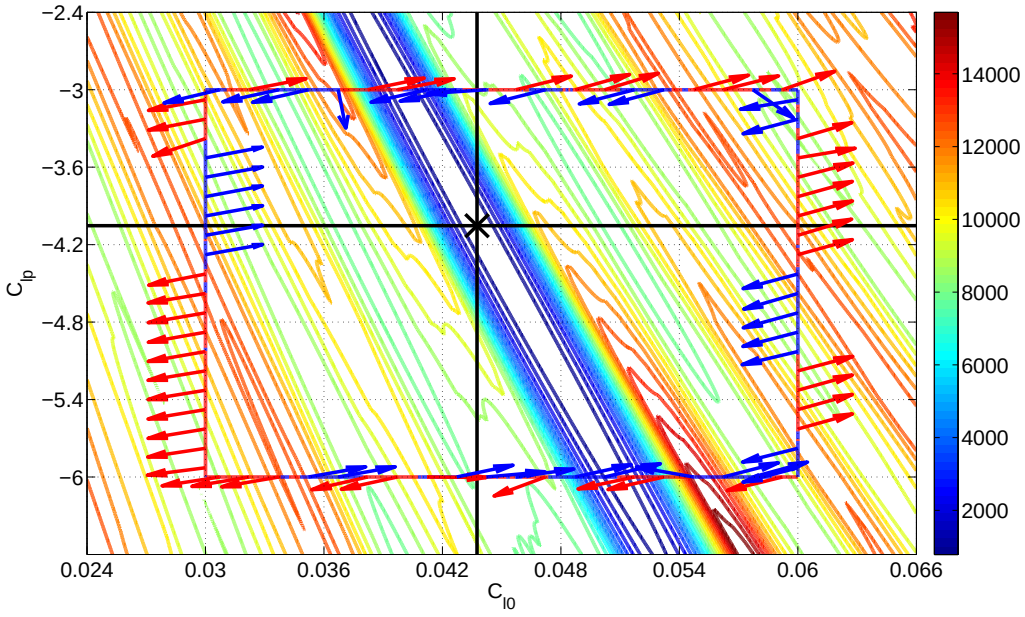


Figure 3.1: Contour of Cost Function over Roll Generation and Roll Damping, Nominal Parameters C_{l0} = 0.04375 and C_{lp} = -4.0529

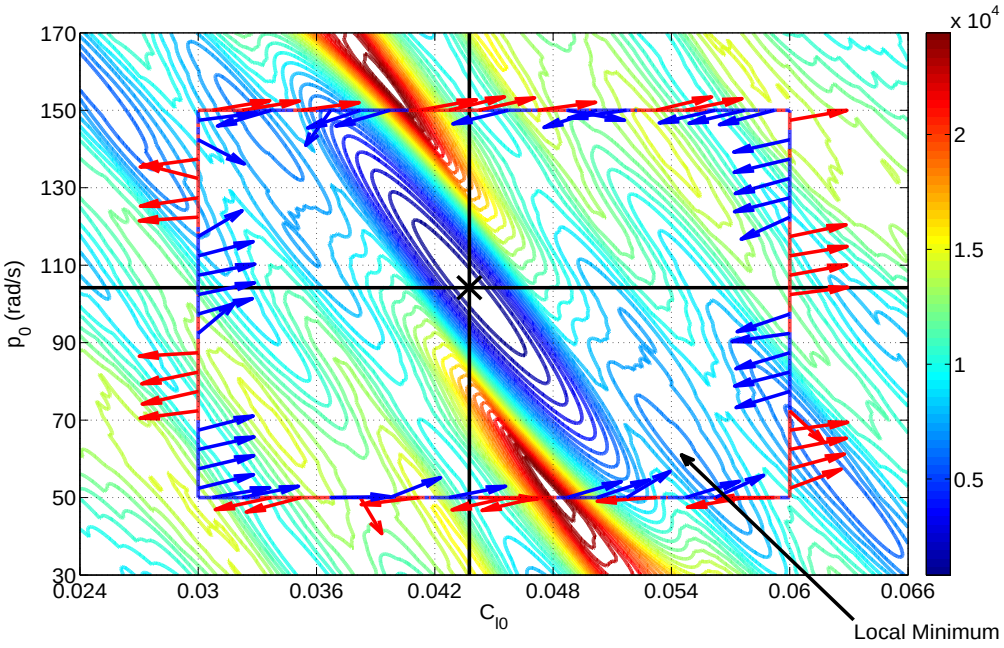


Figure 3.2: Contour of Cost Function over Roll Generation and Initial Roll Rate, Nominal Parameters C_{l0} = 0.04375 and p_0 = 104.2 rad/s

Another interesting combination of the roll parameters is C_{l0} and p_0. This time, C_{lp} is held fixed and p_0 is varied with a search range of 50 rad/s to 150 rad/s. The landscape shown in Figure 3.2 indicates another highly multimodal topology. However, the local minima in this case are more pronounced and the basin of attraction around the global minimum is a small oval. The local minima are also deeper, posing a greater hazard to optimizers solving this problem.

To better understand the causes of these local minima, a trajectory was simulated using the parameters of the local minimum in the lower right corner corresponding to $C_{l0} = 0.0544$ and $p_0 = 62.62$ rad/s. Figure 3.3 shows a comparison between the simulated roll angle trajectory using the nominal parameters and the parameters of the local minimum. For this set of parameters, the roll angle comes into phase with the measurements at around 100 m and again briefly at 200 m. While it remains out of phase for most of the flight, these brief periods where the trajectories align yield lower cost values. Any change in either parameter will cause the trajectory to shift

out of phase which creates the local minimum. As this phenomenon is driven by the roll dynamics themselves, these local minima would occur regardless of the number of roll angle measurements used.

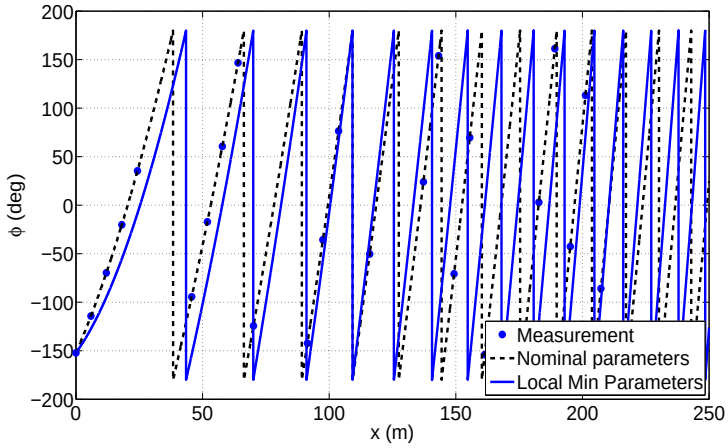


Figure 3.3: Roll Angle Comparison, Local Minimum $C_{l0} = 0.0544$, $p_0 = 62.62$

Interactions between the roll parameters are not the only cause of local minima in the roll dynamics. To investigate the effects of the roll parameters C_{l0} and p_0 individually, each parameter is varied along with the initial velocity u_0. x position is added to the cost function along with ϕ to observe changes in the cost due to u_0. For both cases, u_0 is varied across a search range of 800 m/s to 1100 m/s. First, C_{l0} is considered with the cost landscape shown in Figure 3.4. Overall, the cost function is more sensitive to changes in u_0 than C_{l0}. On this zoomed in view, a number of clearly defined local minima are present at varying C_{l0} values near the nominal u_0. These local minima are symmetric about the nominal C_{l0} with four occurring within the search space and two less than 20% from the nominal value. The local minimum at $C_{l0} = 0.036963$ is simulated and plotted in Figure 3.5. As with the previous local minimum, the roll angle comes into phase with the measurements towards the end of the flight, creating the local minimum.

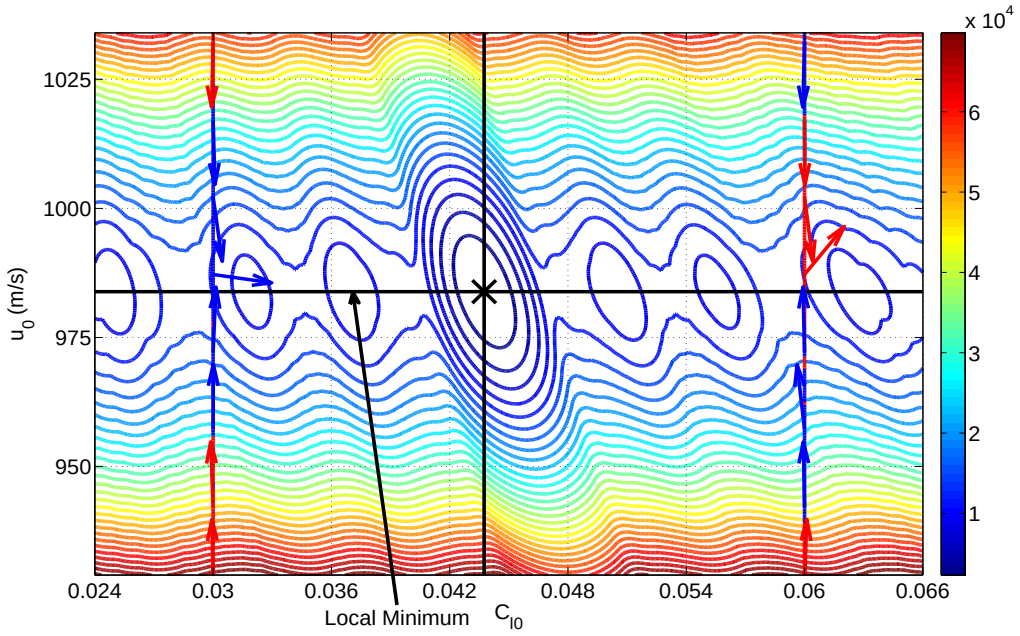


Figure 3.4: Contour of Cost Function over Roll Generation and Initial Velocity, Nominal Parameters $C_{l0} = 0.04375$ and $u_0 = 984$ m/s

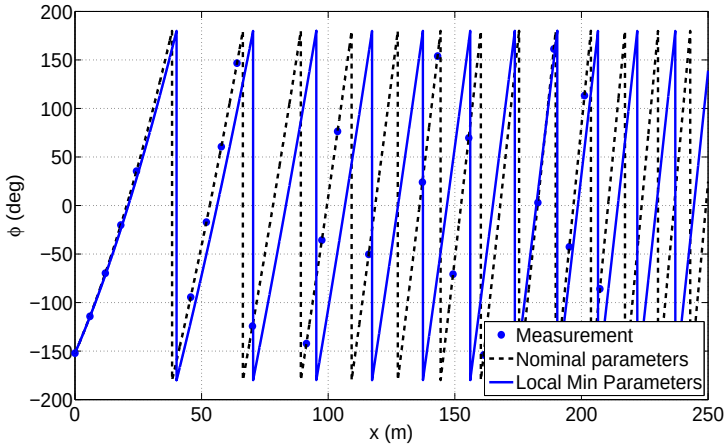


Figure 3.5: Roll Angle Comparison, Local Minimum $C_{l0} = 0.036963$

The final parameter of interest is p_0. Figure 3.6 shows the cross section of u_0 vs p_0. Again, the cost function is more sensitive to u_0 than p_0 with local minima occurring near the nominal value of u_0. The local minima in and near the search range are about 50 rad/s from the nominal value, but the basins of attraction for the

local minima extend about 25 rad/s. Looking at the roll trajectory in Figure 3.7, the trajectories align from about 150 m to the last measurement.

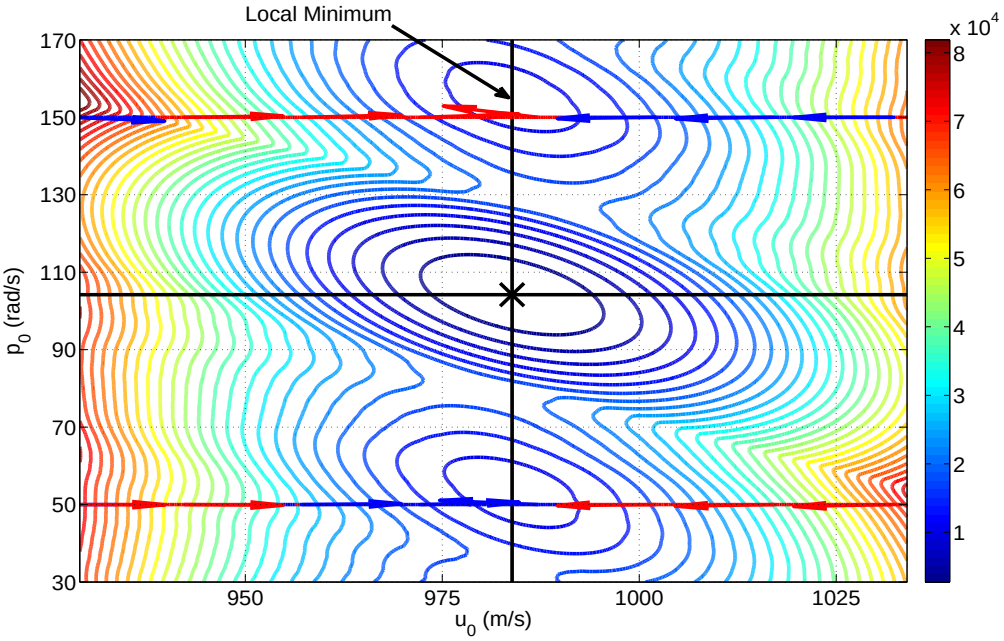
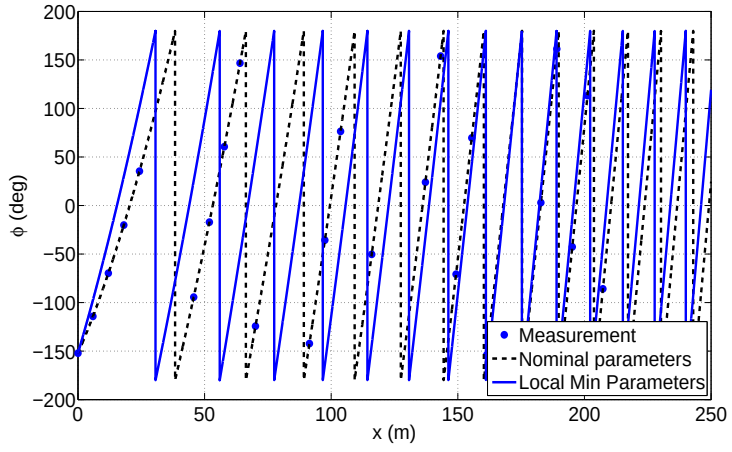


Figure 3.6: Contour of Cost Function over Initial Velocity and Initial Roll Rate, Nominal Parameters $u_0 = 984$ m/s and $p_0 = 104.2$ rad/s



2

Figure 3.7: Roll Angle Comparison, Local Minimum $p_0 = 154$ rad/s

Overall, the roll dynamics with wrapped roll measurements present numerous challenges for an optimizer tasked with estimating these parameters. Both C_{l0} and

p_0 exhibit multiple local minima within the search space, and when combined, create a complex topology that is difficult to navigate. These local minima are caused by specific parameter combinations that allow the roll angle trajectory to come into phase with a subset of the measurements. This phenomenon is caused by the roll dynamics and thus is independent of the level of noise and the number of roll measurements.

3.1.2 Epicyclic Dynamics Analysis

The epicyclic dynamics of a projectile govern the angular motion of the projectile about the pitch and yaw axes as well as the angle of attack and angle of sideslip of the projectile. This corresponds to the states v, w, q, and r. The aerodynamic coefficients which drive the epicyclic dynamics are the normal force ($C_{N\alpha}$), pitching moment ($C_{m\alpha}$), and pitch damping (C_{mq}). To evaluate these dynamics, a trajectory with high angle of attack is constructed with an initial q at launch of about 20 rad/s. As seen in Section 2.4, an initial pitch rate of this magnitude will sufficiently excite the epicyclic dynamics. All states except for ϕ are included in the cost function as the epicyclic dynamics do not influence the roll angle.

Interesting behavior in the epicyclic dynamics is observed using a cross section over $C_{N\alpha}$ and $C_{m\alpha}$. Here, $C_{N\alpha}$ has a search range of 7 to 9.5 and $C_{m\alpha}$ has a search range of -15 to -5. The landscape shown in Figure 3.8 indicates a strong sensitivity in cost to $C_{m\alpha}$ relative to $C_{N\alpha}$. In general, $C_{N\alpha}$ is only marginally observable without high angle of attack while $C_{m\alpha}$ significantly alters the angular motion of the projectile. The primary feature of this landscape is a ridge which occurs around $C_{m\alpha} =$ -15. This feature is represented by the direction of the gradients along the entire $C_{m\alpha}$ = -15 boundary pointing away from the search space. At this coefficient value, the cost begins to decrease as the magnitude of $C_{m\alpha}$ increases. A valley occurs around $C_{m\alpha}$ = -17, not far from the boundary of the search space.

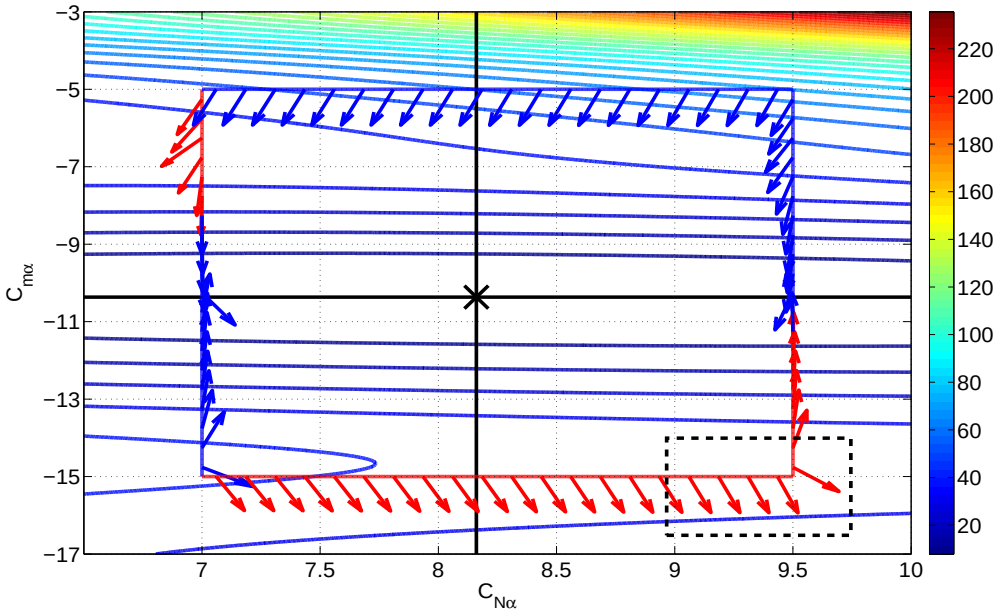


Figure 3.8: Contour of Cost Function over Normal Force and Pitching Moment, Nominal Parameters $C_{N\alpha}$ = 8.161 and $C_{m\alpha}$ = -10.366

To understand why this behavior occurs, multiple trajectories are simulated with varying $C_{m\alpha}$ around -17. Figures 3.9-3.11 show comparisons of h, θ, and ψ between the nominal $C_{m\alpha}$ and each of these cases. Similar to the roll dynamics, this local minimum occurs when the oscillations of each of these states come into phase with the measurements for the last set of spark stations. As $C_{m\alpha}$ is moved away from -17, the trajectories shift out of phase, resulting in an increase in the cost.

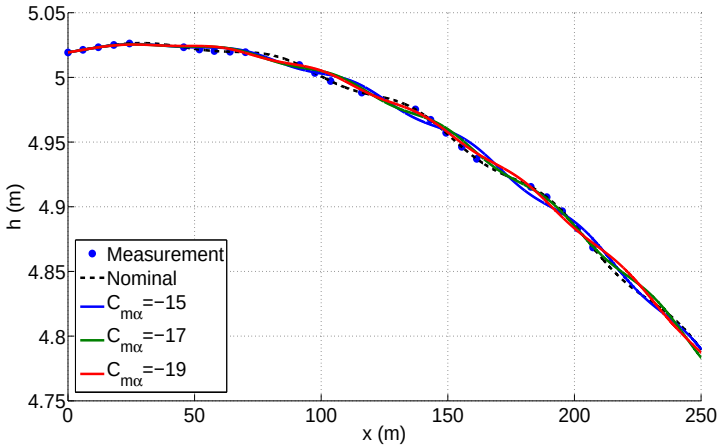


Figure 3.9: Altitude Trajectory Comparison, $C_{m\alpha}$ Local Minimum

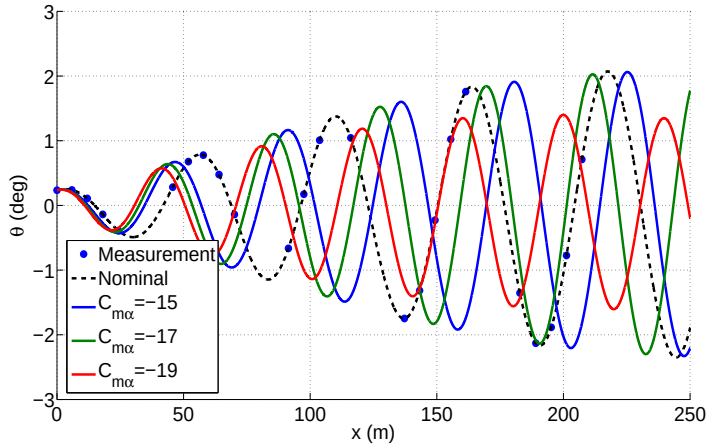


Figure 3.10: Pitch Angle Trajectory Comparison, $C_{m\alpha}$ Local Minimum

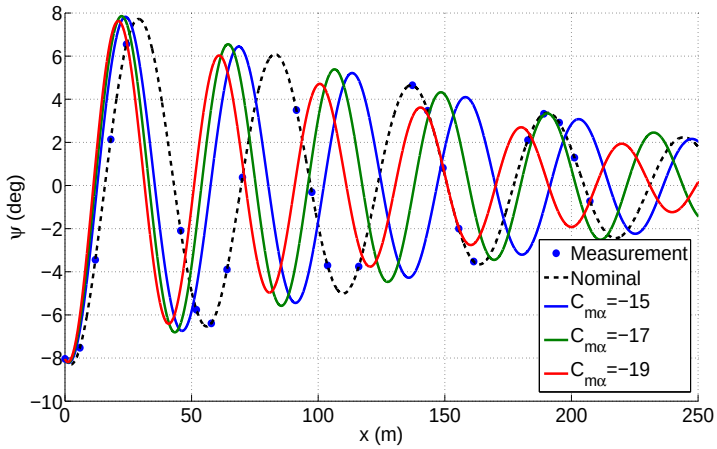


Figure 3.11: Yaw Angle Trajectory Comparison, $C_{m\alpha}$ Local Minimum

This structure becomes an issue when the penalty function is added to the cost function to constrain the parameters to the desired search space. Figure 3.12 shows a close-up of the region around the lower right corner of the landscape when the penalty function is included. Due to the local minimum in the unbounded landscape caused by the epicyclic dynamics, cost decreases as $C_{m\alpha}$ moves away from the search space towards $C_{m\alpha}$ = -17. However, the value of the penalty function increases rapidly the further $C_{m\alpha}$ gets from the boundary, eventually overtaking the pull of the dynamics and creating a new local minimum not far outside the boundary. A local minimum also forms on the search boundary when $C_{m\alpha}$ is varied along with C_{mq}, q_0, and r_0.

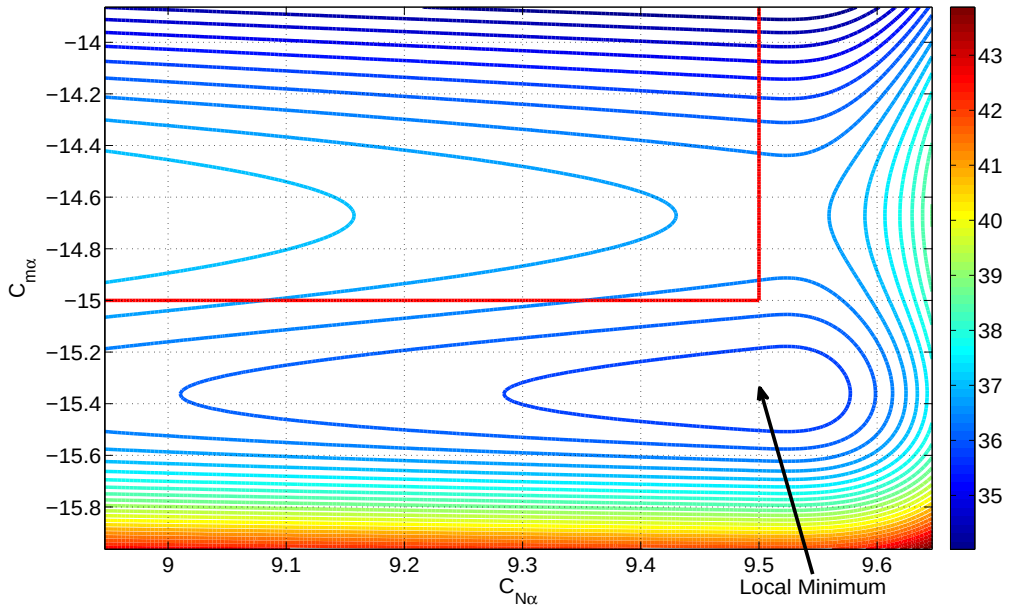


Figure 3.12: Contour of Cost Function over Normal Force and Pitching Moment With Penalty Function, Nominal Parameters $C_{N\alpha}$ = 8.161 and $C_{m\alpha}$ = -10.366

The epicyclic dynamics are also highly coupled with errors in some parameters having a large impact on the landscape of the other parameters. One method to observe this behavior is to examine the landscape of the cost function when noise is added to the synthetic measurements. The position measurement noise has a standard deviation of 3 mm and the angle measurement noise has a standard deviation of 0.1°.

For noisy measurements, the optimal set of parameters may not necessarily be same the as the nominal parameters. Keeping the parameters fixed to their nominal values, however, has the effect of adding small errors to each parameter. Landscapes observed under these conditions would characterize the cost function at a set of parameters typically encountered during the optimization process.

The landscape of $C_{N\alpha}$ vs $C_{m\alpha}$ with measurement noise and boundaries is given in Figure 3.13. The primary impact on the cost landscape is a shift of the global minimum from the middle of the search space to slightly past the boundary on $C_{N\alpha}$. In addition, a local minimum occurs in the lower left corner that was not present in either Figure 3.8 or 3.12. This demonstrates how the coupling between the parameters can influence the landscape that the optimizers must traverse, creating numerous local minima which fluctuate due to changing parameters and measurement noise.

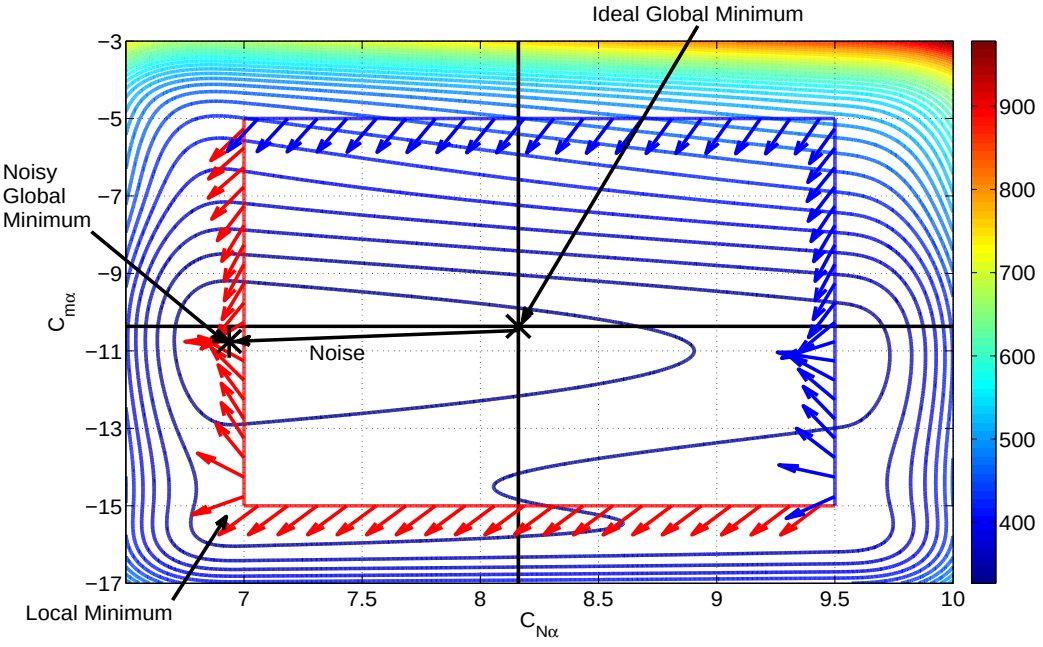


Figure 3.13: Contour of Cost Function over Normal Force and Pitching Moment With Noise and Penalty Function, Nominal Parameters $C_{N\alpha}$ = 8.161 and $C_{m\alpha}$ = -10.366

For the epicyclic dynamics, $C_{m\alpha}$ has a significant impact on the dynamics, creating a local minimum beyond the typical search range. This tendency in the underlying

dynamics causes numerous local minima to form near the search boundary when the penalty function is enforced. Small errors in some of the parameters can also cause dramatic shifts in the cost landscape, often pushing the cost contours into the boundary. This indicates that as an optimizer is progressing on this problem, some of the parameters may be drawn into the search boundary until accurate estimates of other parameters are obtained. However, if a local minimum forms along this boundary, the optimizer may become trapped, unable to improve any parameter.

3.1.3 Microspoiler Dynamics Analysis

The final component of the projectile parameter estimation problem is the microspoiler dynamics. The model for the microspoilers is given in Section 2.4. Of particular interest is the effect of the microspoiler spin rate (Ω_0) on the cost function. For this case, the spin rate is assumed to be constant for the entire flight, isolating the effects of the spin rate from the time constant τ_{ms}. ϕ is again excluded from the cost function as the microspoilers to not affect the roll dynamics. A landscape is constructed over the axial force δ_A and Ω_0. Typically, δ_A varies from -45 N to -15 N while Ω_0 varies from 350 rad/s to 500 rad/s. The cost contours in Figure 3.14 show multiple local minima in terms of Ω_0. While all of the gradients point into the search space, there are two local minima which occur in this space. Given the large basins of attraction for these local minima, only a narrow band of about 25-30 rad/s around the global minimum stays within its basin. These basins are also relatively deep with costs not much higher than the optimal solution. Additional local minima appear as Ω_0 increases outside of the search space which may be reached if the boundaries are not enforced.

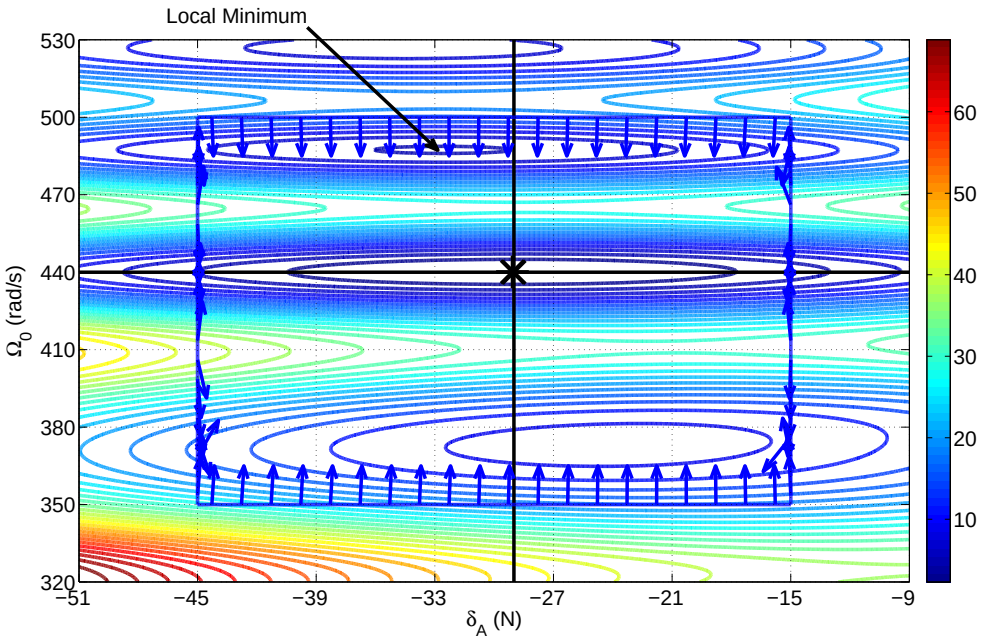


Figure 3.14: Contour of Cost Function over Microspoiler Axial Force and Spin Rate, Nominal Parameters δ_A = -29 N and Ω_0 = 440 rad/s

The local minimum at $\Omega_0 = 488$ rad/s is used to understand the nature of these local minima in the microspoiler dynamics. Figures 3.15 and 3.16 show a comparison of the θ and ψ trajectories simulated at Ω_0 values around this local minimum. As Ω_0 approaches the local minimum, the trajectories begin to shift into phase with the measurements and start to match the amplitude of the oscillations. Increasing or decreasing Ω_0 causes significant changes in the trajectories, reducing the quality of the fit and thus increasing the cost. This makes escaping these local minima even more difficult for the optimizers and presents a significant challenge for obtaining accurate estimates. When taken together with the roll dynamics and the epicyclic dynamics, the smart projectile parameter estimation problem presents a number of pitfalls for any optimizer. Importantly, these local minima are all based on interactions between certain parameters and the dynamics of the projectile system which manifest as valleys and basins within the cost landscape.

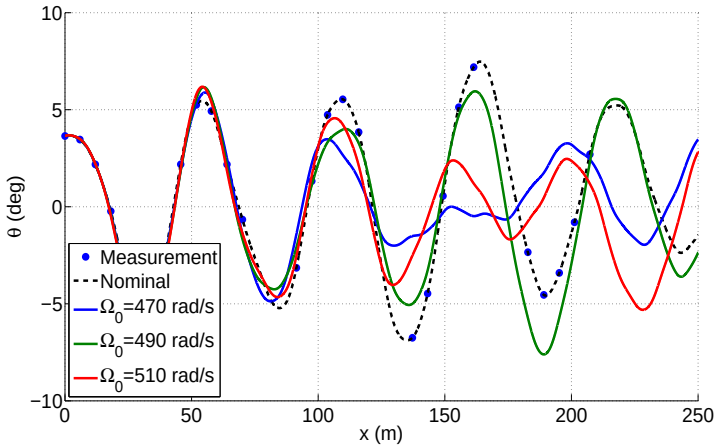


Figure 3.15: Pitch Angle Trajectory Comparison, Ω_0 Local Minimum

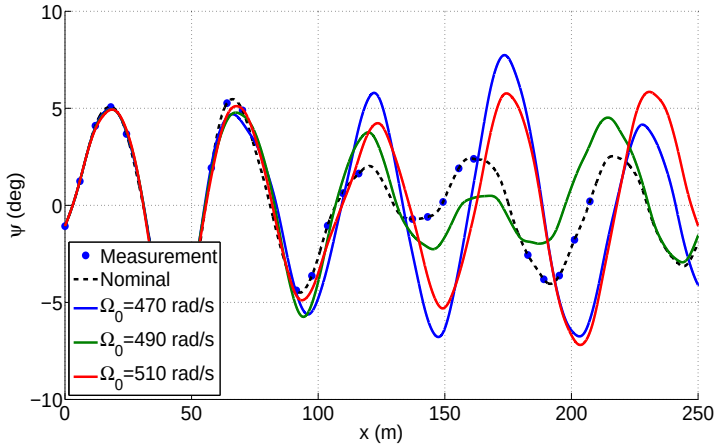


Figure 3.16: Yaw Angle Trajectory Comparison, Ω_0 Local Minimum

3.2 Local Search Analysis

Another technique for understanding the topology of a cost function is to observe the behavior of a local search algorithm such as a hill climber operating on the optimization problem over a number of trials [119]. Since hill climbers follow the gradients of the cost towards a local minimum, they can be used to identify the location and basin of attraction of local minima in the search space [120]. The basin of attraction for a local minimum is defined as the subset of the parameter space in which a local search started within this region will converge on the local minimum

[60]. The number of local minima as well as the size and shape of their basins of attraction provide valuable insight into the overall topology of the problem.

The projectile parameter estimation problem considered here is based on fitting a single synthetic trajectory with constant aerodynamic coefficients and no measurement noise. Three different parameter estimation cases are considered: low angle of attack, high angle of attack, and active microspoilers. A different synthetic trajectory is generated for each case based on the requirements for each estimation problem. To evaluate the topology of the projectile parameter estimation problem, BFGS is run 1000 times from random initial parameters inside the search range until either the cost crosses a threshold of 10^{-6}, the rate of reduction in cost falls below a set threshold, or the optimizer is unable to find a point which reduces the cost. The starting point, ending point, and gradient at the ending point are recorded as well as the amount of time the optimizer spends searching outside of the search bounds. To determine the location of any local minima the hill climber encountered, the ending locations for each trial were categorized based on distance from neighboring points. A large clustering of points in one area indicates the presence of a local minimum [119].

3.2.1 Low Angle of Attack Analysis

The simplest parameter estimation case is estimating parameters using a single, low angle of attack trajectory. This type of trajectory allows for easy estimation of the base drag coefficient C_{X0} and the roll parameters C_{l0} and C_{lp} as the epicyclic dynamics are not excited, leaving only the base drag and roll moments acting on the projectile. In addition, the initial velocity u_0 and the initial roll rate p_0 must also be estimated for a total of five parameters when fitting a single trajectory. A synthetic low angle of attack trajectory is generated using the baseline projectile model from Section 2.4 with no initial angular velocity. Only x and ϕ are needed in the cost function to estimate these parameters. As seen in Section 3.1.1, the roll dynamics are multimodal

when the roll measurements are wrapped which causes issues for the hill climber.

First, the low angle of attack case is evaluated without bounds on the parameters, allowing the optimizer to explore beyond the target search space. A summary of results for this case is presented in Table 3.2. Out of 1000 trials, only 8.4% cross the cost threshold indicating convergence. In total, there were 205 unique stopping points for the optimizer. 19 clusters contained 10 or more points with 6 containing 30 or more. Of the six most common local minima, 5 fell outside of the search space. In total, 62.4% of trials ended outside of the search range of at least one parameter. These results confirm the observation from Section 3.1.1 that the wrapped roll dynamics contain a large number of local minima. The addition of the penalty function to constrain the search space has little impact on the overall performance of the hill climber with only 7.5% reaching the solution. In this case, there were 27 clusters of 10 or more and 4 of 30 or more. 18 of the 27 clusters were on or slightly beyond at least one boundary with 60.6% of all ending points on or beyond a boundary. This indicates that interactions between the projectile dynamics and the boundaries are creating local minima which are not strictly caused by the projectile dynamics.

Table 3.2: Low Angle of Attack Local Search Analysis Results

Case	Percent Converged On Solution	Cluster Count>10	Cluster Count>30	Percent On or Outside Bounds
Unbounded	8.4	19	6	62.4
Bounded	7.5	27	4	60.6

3.2.2 High Angle of Attack Analysis

Trajectories with high angle of attack are used to estimate the remaining body aerodynamic coefficients which are only observable with sufficient angle of attack. This

typically means angle of attack values greater than 5° for most of the flight. These coefficients are the nonlinear drag coefficient C_{X2}, the normal force coefficient $C_{N\alpha}$, the pitching moment coefficient $C_{m\alpha}$, and the pitch damping coefficient C_{mq}. To help simplify the estimation process, C_{X0}, C_{l0}, and C_{lp} are fixed to their nominal values as these are typically estimated separately. A single synthetic trajectory using the baseline projectile with a large initial q at launch, like the trajectory discussed in Section 2.4.2, is used to estimate these parameters. The initial velocities and angular velocities at the first measurement are also estimated, resulting in 10 parameters for this estimation problem. All six states are included in the cost function, including ϕ, as inaccuracies in p can lead to large errors in the predicted trajectories of θ and ψ through coupling of the angular velocity dynamics. Table 3.3 provides a summary of the results for this case.

Beginning with the unbounded case, 65.8% converged on the solution with 2 local minima with 10 or more points. One local minimum was not far beyond the boundary in C_{X2} and C_{mq} while the other was extremely far from the search space. In total, 34.1 % exceeded at least one boundary with 19% having a final C_{mq} value less than -1000 (nominal -331.7), well beyond the boundary. Pitch damping values this large are not realistic and demonstrate the need to constrain the search space to prevent the optimizers from reaching non-physical parameters. The cost function was also not sensitive to C_{X2} and $C_{N\alpha}$ which sometimes varied significantly between points which were otherwise close for the remaining parameters. Adding bounds, 62.5% converged on the global minimum, however, 2 very large local minima occurred on the boundary. In total, 92.5% of all trials ended at one of these three points. The two local minima were both on the C_{X2}, $C_{m\alpha}$, and C_{mq} boundaries. A total of 35.1% of trials finished on at least one boundary. Escaping a local minimum in three dimensions like this is a challenge for most optimizers and is a potential pitfall for meta-optimization.

Table 3.3: High Angle of Attack Local Search Analysis Results

Case	Percent Converged On Solution	Cluster Count>10	Cluster Count>30	Percent On or Outside Bounds
Unbounded	65.8	2	0	34.1
Bounded	62.5	2	2	35.1

To better understand why such unrealistic parameter combinations create local minima, one local minimum from the unbounded, high angle of attack landscape was simulated to observe the projectile behavior under these conditions. The parameters for this local minimum are given in Table 3.4 with the nominal parameters given as a reference. This local minimum is very far away from the search space in both C_{X2} and C_{mq}. $C_{N\alpha}$ and $C_{m\alpha}$ are also very far from their nominal values. Figures 3.17-3.20 show the simulated trajectory for these parameters compared to the nominal trajectory. Given this extremely high value of C_{mq}, all angular oscillation is quickly damped out with θ and ψ going to zero. The trajectory also essentially cuts a straight line through the y and h measurements. With no angle of attack, the cost function is no longer sensitive to the other aerodynamics coefficients, allowing them to vary significantly without penalty. Such a trajectory would never be experienced in practice, but these and similar parameters were reached by the hill climber on a number of trials.

Table 3.4: High α Local Minimum Parameters

Parameter	Local Minimum	Nominal
C_{X2}	222.84	4.4
$C_{N\alpha}$	0.026	8.161
$C_{m\alpha}$	-1.03	-10.366
C_{mq}	-8230	-331.7
u_0 (m/s)	988.8	983.9
v_0 (m/s)	-127	-128
w_0 (m/s)	62.9	63.3
p_0 (rad/s)	104.2	104.2
q_0 (rad/s)	-8.06	1.87
r_0 (rad/s)	-15.7	3.9

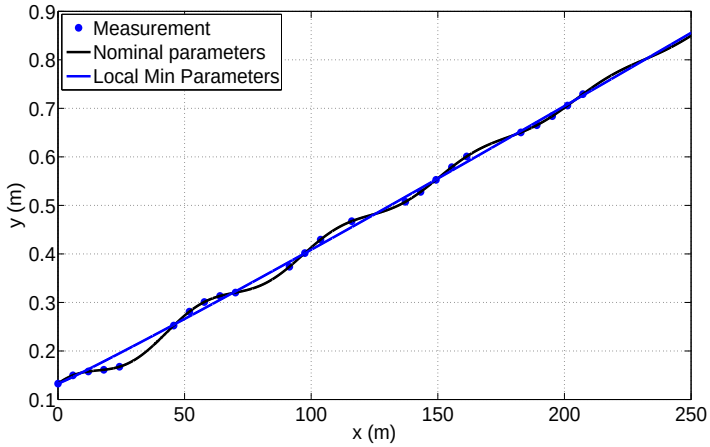


Figure 3.17: High Angle of Attack Local Minimum Inertial-Y Position Trajectory Comparison

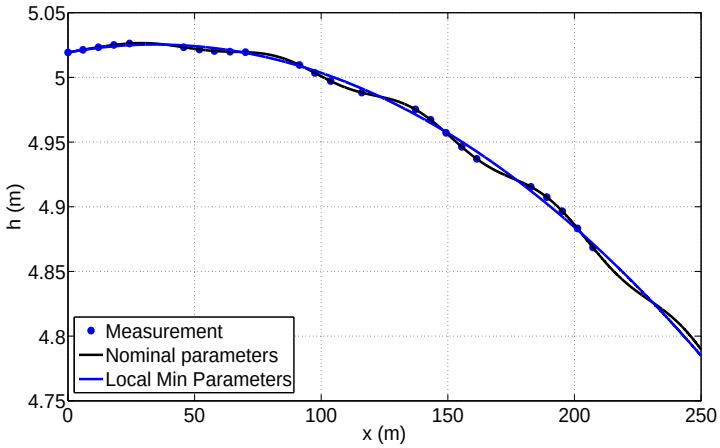


Figure 3.18: High Angle of Attack Local Minimum Altitude Trajectory Comparison

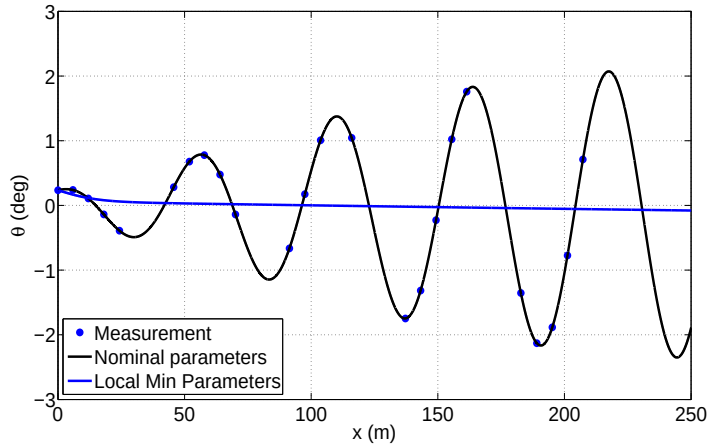


Figure 3.19: High Angle of Attack Local Minimum Pitch Angle Trajectory Comparison

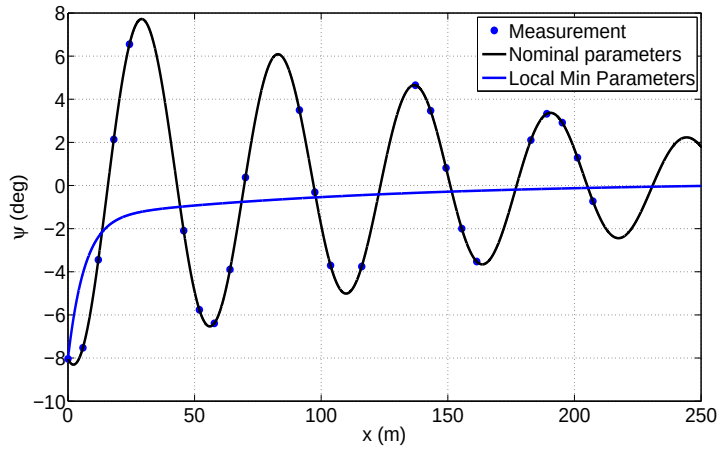


Figure 3.20: High Angle of Attack Local Minimum Yaw Angle Trajectory Comparison

3.2.3 Active Microspoiler Analysis

The final case considered is the estimation of the microspoiler parameters. For this analysis, the synthetic trajectory is generated using a constant spin rate for the entire flight. To isolate the effects of the microspoilers on the projectile motion, the body aerodynamic coefficients are held fixed, leaving only the microspoiler force and moment coefficients δ_A, δ_N, and δ_m, initial phase ω_0, and spin rate Ω_0 to be estimated in addition to the initial velocities and angular velocities. The results for this case are summarized in Table 3.5.

Without bounds, 41.5% of trials reached the global minimum with another 21.5% falling into 4 different local minima. All four of these local minima were outside of the bounds of at least one parameter. Overall, the optimizer spent 60% of the computation time searching outside of the bounds of δ_A, δ_N, and δ_m with 52.5% ending outside of the search space. With the addition of bounds on the search space, the number of trials converging on the global minimum increased to 48.5%. One large local minimum with 17.7% of trials occurred on the boundary for δ_N, δ_m, and Ω_0. One local minimum with a cluster of 10 points had $\Omega_0 = 379$ rad/s, close to the local minima in Ω_0 identified in Section 3.1.3. In total, 39.7% of trials stalled on the boundary, less than the number escaping the search space previously. This large reduction in trials leaving the search space, combined with the increase in the number of successful trials, indicates that the boundaries are reshaping the landscape, steering the hill climber back into the search space in some places.

Table 3.5: Active Microspoiler Local Search Analysis Results

Case	Percent Converged On Solution	Cluster Count>10	Cluster Count>30	Percent On or Outside Bounds
Unbounded	41.5	4	2	52.5
Bounded	48.5	2	1	39.7

Two observations can be made based on the local search analysis of the parameter estimation problem. First, as was observed in Section 3.1, the projectile dynamics create a number of local minima in the parameter estimation cost function. The roll parameters in particular are difficult for a hill climber to navigate due to numerous local minima scattered about the search space. Second, some local minima exist outside of the search space with a basin of attraction that reaches inside the parameter boundaries. For hill climbers beginning within one of these basins, it will be drawn outside of the search space towards that local minimum. In some instances, these local minima correspond to non-physical parameter combinations which produce unrealistic trajectories. Even on runs where the hill climber converged on the solution, it may have taken a path far outside of the boundaries to get there. When bounds are placed on the parameters, the attraction of these local minima still shape the cost landscape, pulling the hill climber into the boundary where new local minima are formed. For the low and high angle of attack cases, there was little change in performance of the hill climber when the boundaries were enforced. However, performance improved for the active microspoiler case with boundaries, indicating a possible benefit for the optimizers from the boundaries.

CHAPTER 4
DESCRIPTION OF META-OPTIMIZATION FRAMEWORK

For any one optimizer, there are a set of optimization problems that the optimizer is well suited to solve and other problems that pose significant challenges. The effectiveness of an optimizer is based on the nature of the specific algorithm and the topology of the problem. For example, hill climbers are extremely efficient on convex problems, but are attracted towards the nearest local minimum. Metaheuristic methods such as particle swarm optimization (PSO) can freely search the parameter space, but lack the refinement capabilities near a minimum that hill climbers posses. On certain problems, different metaheuristics may also struggle with clustering where the entire population remains near a local minima, unable to escape and resume searching. Since these issues vary from optimizer to optimizer and problem to problem, selecting an appropriate optimizer for a given problem is a difficult task for an engineer.

The goal of meta-optimization is to intelligently deploy a diverse set of optimizers, leveraging the strengths of each optimizer and minimizing their weaknesses in order to reliably solve challenging optimization problems with minimal user intervention. A good metaphor for this process is the delegation of tasks between various workers. A project manager assigns one worker to solve a certain problem and requests an update after a period of time. The worker is given all of the necessary resources to work on the problem. After the allotted time, the worker reports back with his best results and is graded on performance and efficiency. If the worker is making acceptable progress, the worker continues until the worker gets stuck or slows down. The manager then decides if they will let the worker continue on the problem or assign it to another worker. This process continues until the problem is solved. In

this way, if one worker is not able to provide results, another worker is assigned the task to ensure continual progress towards the solution.

Meta-optimization bears some similarities to algorithm portfolios and hybrid optimizers. While meta-optimization uses a bank of optimizers and seeks to predict performance, there are some key differences from existing methods. Unlike algorithm portfolio approaches, meta-optimization shares information between optimizers with all optimizers using a common population of solutions. This allows for synergies to develop between the individual optimizers and ensures maximum use of resources towards improving the solution. Also, a wide range of optimizers are used with a combination of local and global search methods, similar to hybrid optimizers. However, meta-optimization does not have a fixed structure or rules for switching between global and local search. In this way, meta-optimization is able to solve a wide range of problems with various complexities and structures.

This chapter provides an in depth description of the meta-optimization framework which consists of five main parts: the bank of optimizers, the performance metric, the optimizer selection routine, the optimizer manager, and the auto-tuning algorithm. An overview of this framework is given by Figure 4.1, which shows the general flow of the meta-optimizer. The basic flow of the meta-optimizer proceeds as follows: an optimizer is chosen, resources are allocated to the optimizer, the optimizer runs for a period of time, and then the performance of the optimizer is evaluated. This process is then repeated until a solution is found or the meta-optimizer has exhausted its resources. The objective function, also known as the cost function, represents the optimization problem that is to be solved. In this work, the objective function $f(\mathbf{x})$ takes a common form given by Eq. 4.1. Here, $\mathbf{x}$ is the parameter vector, and $\mathbf{L}$ and

$\mathbf{U}$ are the lower and upper bounds on the parameter space respectively.

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{L} \leq \mathbf{x} \leq \mathbf{U} \end{aligned} \tag{4.1}$$

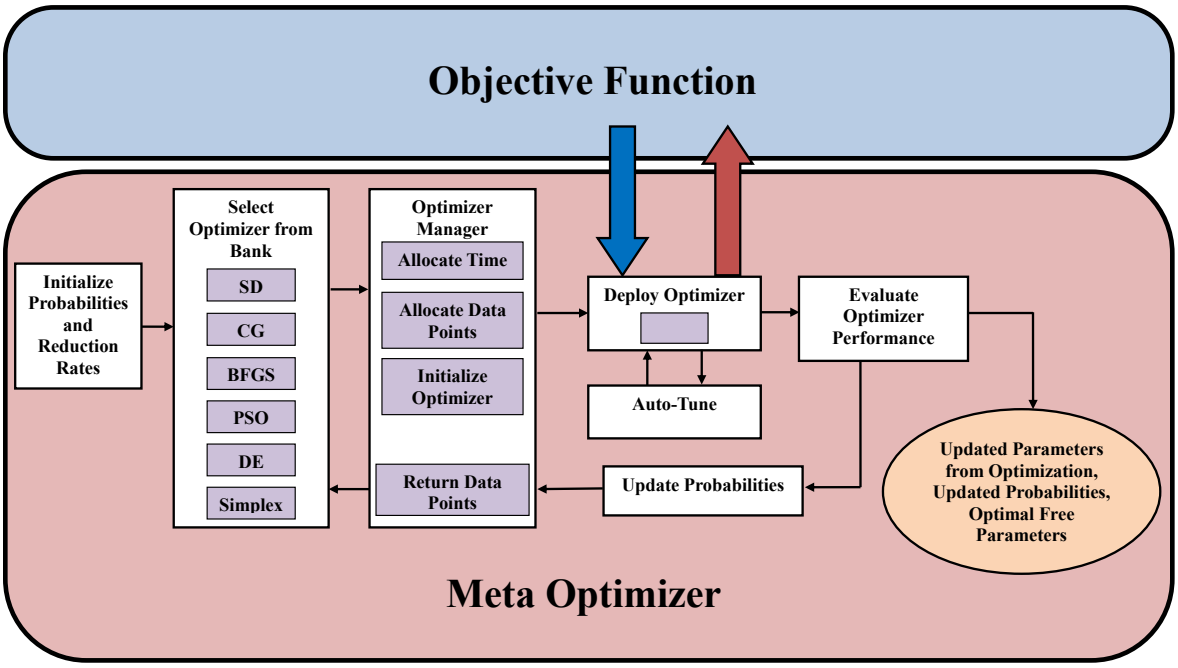


Figure 4.1: Flow Chart Representation of Meta-Optimization Framework

4.1 Bank of Optimizers

A key part of meta-optimization is the resident bank of numerical optimizers that can be used. Many different optimizers, each with numerous variants, could be selected for inclusion in the bank of optimizers. Each optimizer is individually capable of obtaining a solution to the given optimization problem. To demonstrate the capabilities of meta-optimization, nine common optimization algorithms were chosen based on a mixture of local and global search methods, forming the basic group of optimizers deployed by meta-optimization. Each optimizer uses the basic implementation

of the algorithm, except in cases where modifications were necessary to adapt the algorithms to the meta-optimization framework.

The primary category of local search methods are considered hill climbers, where the algorithm searches for a better solution by incrementally varying the current best solution, gradually moving towards a local optimum. Included hill climbers are steepest descent (SD) [46], conjugate gradient (CG) [121], and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [37]. It is known that the primary drawback with hill climbers is that they are highly prone to converging to local minima. This is due to the nature of these algorithms in which they are only able to search in a direction that lowers the cost function with little ability to explore the search space. Once the algorithm reaches a local minima, there is no way for the algorithm to leave this point as the gradient is zero and thus the algorithm cannot produce a new search direction. For this reason, these methods are generally considered local optimization methods.

To balance these methods, it is necessary to include global optimizers in the bank of optimizers. These methods search the entire parameter space by operating on a large set of points, continually seeking the global minimum. The global optimizers employed are particle swarm optimization (PSO) [48, 49], differential evolution (DE) [47], invasive weed optimization (IWO) [122, 123, 124, 125], and ant colony optimization (ACO) [126]. The Nelder-Mead (Simplex) Method [127, 128] and tabu search [129, 130] are also included in the bank of optimizers. These optimizers form a diverse set of algorithms, each with different techniques for solving optimization problems. Each optimizer is suited to different types of problems, providing meta-optimization with a reliable set of methods to solve a wide range of problems. An overview of the included optimizers is given in Table 4.1.

Table 4.1: Overview of Included Optimization Algorithms

Type	Optimization Algorithm	Advantages	Disadvantages	Citation
Hill Climber	Steepest Descent (SD)	Simple to implement, no memory	Converges to local minima, inefficient for poorly conditioned problems	[46]
	Conjugate Gradient (CG)	Faster than SD	Converges to local minima, not all search directions reduce cost function	[121]
	BFGS	Fast convergence	Converges to local minima, larger memory storage	[37]
Global Metaheuristic	Particle Swarm (PSO)	Global search capacity, simple implementation	No guaranteed convergence, long run times, poor local search	[48, 49]
	Differential Evolution (DE)	Global search capacity, simple implementation	No guaranteed convergence, long run times, poor local search	[47]
	Invasive Weed (IWO)	Global search capacity, simple implementation	No guaranteed convergence, long run times, poor local search	[122, 123, 124]
	Tabu Search (TS)	Global search capacity, able to search uphill	No guaranteed convergence, long run times	[129, 130]
	Ant Colony (ACO)	Global search capacity, simple implementation	No guaranteed convergence, long run times, poor local search	[126]
Local Metaheuristic	Nelder-Mead Simplex (SIM)	Local search without gradient information, fast convergence	No guaranteed convergence	[127, 128]

4.1.1 Steepest Descent (SD)

The simplest of the hill climbers, SD uses the negative of the gradient as its search direction [46]. The search direction $\mathbf{p}_k$ is defined as:

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k) \tag{4.2}$$

A first order central finite difference is used to compute the gradient at the current position as shown in Eq. 4.3:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{4.3}$$

where h is the step length. An adaptive step length algorithm adjusts the step lengths online to ensure accurate gradient estimates [131]. This algorithm is detailed in Appendix A. Due to the large variations in magnitude of the parameters in some optimization problems, the gradient can become poorly conditioned, requiring scaling to improve performance. Scaling is performed by transforming the parameters by a matrix D such that $\mathbf{x} = \mathbf{D}\mathbf{y}$. The scaled gradient then becomes $\nabla F(\mathbf{y}) = \mathbf{D}\nabla f(\mathbf{x})$. The Hessian matrix is used to scale variables and is given by [131]:

$$\mathbf{D} = \begin{bmatrix} \frac{1}{\sqrt{H_{11}}} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sqrt{H_{nn}}} \end{bmatrix} \tag{4.4}$$

The diagonal values of the Hessian are approximated using a first order central finite difference given by Eq. 4.5. This scaling matrix is adjusted at every step, allowing it to adapt to changes in the parameters as the algorithms progresses.

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{4.5}$$

A new point is generated by performing a step along the search direction using Eq. 4.6:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{4.6}$$

where α_k is the search step length. In this implementation, a backtracking search is used to determine the step length for the current iteration beginning with a step length of 1. A step is only accepted if it produces a nontrivial reduction in cost specified by the Armijo condition [37]:

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + \mu \alpha_k \mathbf{p}_k^T \nabla f(\mathbf{x}_k) \tag{4.7}$$

A typical value for the constant μ is 10^{-4}. If a step is not accepted, then the step length is reduced by:

$$\alpha_k(i+1) = r\alpha_k(i) \tag{4.8}$$

where r is the reduction coefficient, usually equal to 0.5. As SD approaches a solution, the relative change in the cost between iterations is used to detect convergence. This condition is specified as:

$$\frac{|f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})|}{\max\left[|f(\mathbf{x}_{k-1})|, 10^{-5}\right]} \leq \epsilon \tag{4.9}$$

where ϵ is a set threshold [46]. SD will also stop when no cost reduction can be found using the backtracking search.

4.1.2 Conjugate Gradient (CG)

Also known as the Fletcher-Reeves method, this hill climber is based on the concept of conjugacy of each search direction [121]. This implies that each new search direction is linearly independent to all previous search directions with respect to a matrix A.

Mathematically, this means that:

$$\mathbf{p}_k^T A \mathbf{p}_{k-1} = 0 \tag{4.10}$$

In the CG method, the search directions are conjugate with respect to the Hessian matrix H. The search direction is given by:

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k) + \beta_k \mathbf{p}_{k-1} \tag{4.11}$$

where β_k is defined as:

$$\beta_k = \frac{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)}{\nabla f(\mathbf{x}_{k-1})^T \nabla f(\mathbf{x}_{k-1})} \tag{4.12}$$

This formula will ensure conjugacy for convex problems where the step length α_k is chosen to minimize the function along the search direction [121]. For general nonlinear functions and inexact line searches, an additional check must be performed to ensure the search direction is always a descent direction. This requires that $\mathbf{p}_{k+1}^T \nabla f(\mathbf{x}_k) < 0$. If this inequality does not hold, the new search direction is pointing uphill and the search direction is reset to the negative gradient [131, 46]. This implementation of CG uses the same backtracking line search and stopping criteria as SD.

4.1.3 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

BFGS is a quasi-Newton method which iteratively updates an approximation of the Hessian matrix to improve performance and reduce computational expense. For this method, the search direction is computed by:

$$\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k) \tag{4.13}$$

where $\mathbf{B}$ is the approximation of the Hessian. A rank-two update is used to ensure both symmetry and positive definiteness of the Hessian approximation and is given

by:

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{(\mathbf{B}_k\mathbf{s}_k)(\mathbf{B}_k\mathbf{s}_k)^T}{\mathbf{s}_k^T\mathbf{B}_k\mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k} \tag{4.14}$$

where $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$ and $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$. To simplify the implementation of the algorithm, the inverse of the Hessian approximation is instead computed by:

$$\mathbf{B}_{k+1}^{-1} = \left(\mathbf{I} - \frac{\mathbf{s}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}\right)\mathbf{B}_k^{-1}\left(\mathbf{I} - \frac{\mathbf{y}_k\mathbf{s}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}\right) + \frac{\mathbf{s}_k\mathbf{s}_k^T}{\mathbf{y}_k^T\mathbf{s}_k} \tag{4.15}$$

When BFGS is first called, the matrix $\mathbf{B}_k^{-1}$ is set to identity. The slope of the search direction is also checked to ensure a downhill direction with $\mathbf{B}_k^{-1}$ set to identity if this is violated [37, 36]. BFGS also uses a backtracking search in this implementation with the same stopping criteria as SD and CG.

4.1.4 Particle Swarm Optimization (PSO)

PSO is a metaheuristic first developed by Kennedy and Eberhart after observing swarm behavior in nature [48]. The core principle of the algorithm is the sharing of information between each particle in the swarm. Each particle is given an initial position and a random velocity with maximum magnitude equal to 1/20th of the total search range for each parameter. This value is not considered a tuning parameter for PSO as it is only used when the population is initialized. Population sizes typically range from 25 to 50 particles. At each iteration, the position of the ith particle in the jth dimension is updated using the following equation:

$$x_{k+1}^i(j) = x_k^i(j) + V_{k+1}^i(j) \tag{4.16}$$

where V is the velocity of the particle. The velocity update is given by:

$$V_{k+1}^i(j) = wV_k^i(j) + c_p r_p(P^i(j) - x_k^i(j)) + c_g r_g(P_g(j) - x_k^i(j)) \tag{4.17}$$

where w is the inertia weight, c_p is the cognitive learning factor, c_g is the social learning factor, r_p and r_g are uniform random numbers in the range [0,1], P^i is the personal best of particle i, and P_g is the global best solution found so far. A bound is placed on the velocity to prevent particles from quickly exiting the search space. The parameters w, c_p, and c_g, as well as the number of particles are used as tuning parameters. Nominal values for these parameters are 0.729 for w and 1.42 for c_p and c_g with population sizes from 25 to 50 particles [48, 49].

The diversity in the population is used to determine when the swarm has collapsed and will no longer improve. Diversity measures how spread out or clustered the points are in the population and is given by:

$$\sigma_d = \sqrt{\frac{\sum_{i=1}^{N}(x_r^i - \bar{x}_r)^2}{N-1}} \tag{4.18}$$

$$x_r^i = \sqrt{\sum_{j=1}^{n} \frac{(x^i(j) - \bar{x}_0(j))^2}{\Delta x_0(j)^2}} \tag{4.19}$$

where $\bar{x}_0(j)$ is the mean of the search range of parameter j, $\Delta x_0(j)$ is the size of the search range, and $\bar{x}_r = \frac{1}{N}\sum_{i=1}^{N} x_r^i$ [132]. Two diversity based checks are employed. First, PSO will stop when the diversity falls below a specified threshold. Second, PSO will stop if there is no significant change in diversity for 100 iterations. The first criteria detects when the population has collapsed on a single point while the second criteria detects when the optimizer is unable to update any of the points in the population.

4.1.5 Differential Evolution (DE)

Like other evolutionary algorithms, DE is a population based optimizer modeled after various mechanisms from biological evolution such as mutation, reproduction, and competition. DE was designed by Storn and Price as a simple and effective evolu-

tionary algorithm for use on real valued optimization problems [47]. The algorithm contains three operations; mutation, crossover, and selection. New points are generated according to a greedy condition such that:

$$\mathbf{x}_{k+1}^{i} = \underset{\mathbf{y}}{\operatorname{argmin}}\{f(\mathbf{y})|\mathbf{y} \in \{\mathbf{x}_{k}^{i}, \mathbf{u}_{k+1}^{i}\}\} \tag{4.20}$$

where $\mathbf{u}_{k+1}^{i}$ is the trial vector which is generated using a two step process. The mutation step consists of generating a mutant vector based on three members of the population as given by:

$$\mathbf{v}_{k+1}^{i} = \mathbf{x}_{k}^{r_1} + F(\mathbf{x}_{k}^{r_2} - \mathbf{x}_{k}^{r_3}) \tag{4.21}$$

where F is the mutation amplitude and r_1, r_2, and r_3 are mutually exclusive random integers in the range [1,NP] that are also different from the current index i. The trial vector is then generated one dimension at a time using the crossover operation:

$$u_{k+1}^{i}(j) = \begin{cases} v_{k+1}^{i}(j) & \text{if } r \leq CR \text{ or } j = r_i \\ x_{k}^{i}(j) & \text{if } r > CR \text{ and } j \neq r_i \end{cases} \tag{4.22}$$

where r is a random number on the range [0,1], CR is the crossover rate, and r_i is a randomly chosen index from 1 to the number of dimensions, n. The tuning parameters for DE are the population size and the parameters F and CR. Common tuning parameter values for DE are a population size of 25 with $F = 0.8$ and $CR = 0.9$ [47]. DE also uses the same diversity based stopping criteria as PSO.

4.1.6 Simplex (SIM)

Also known as the Nelder-Mead method, SIM is a direct search method based on the concept of creating a simplex within the search space and then adapting the simplex to the local landscape of the search space [127]. A simplex is a polytope with $n + 1$

vertices in n dimensions such as a triangle on a plane. The algorithm arranges a set of test points as a simplex and iteratively replaces the worst point in the set through various operations including reflection, expansion, contraction, and shrinking. At each iteration, the vertices are sorted based on cost where $i = 1$ denotes the lowest cost and $i = n + 1$ denotes the highest value. Next, the average of the vertices $\bar{\mathbf{x}}$ is taken, excluding the worst point. The first operation is reflection of the worst point with the reflected point defined by:

$$\mathbf{x}_r = (1 + \alpha)\bar{\mathbf{x}} - \alpha\mathbf{x}^{n+1} \tag{4.23}$$

where α is the reflection coefficient which is typically equal to 1. If $f_1 \leq f_r \leq f_n$, then $\mathbf{x}^{n+1}$ is replaced with $\mathbf{x}_r$. If $f_r < f_1$, then expansion is performed to further reduce the cost. The expanded point is generated using:

$$\mathbf{x}_e = \gamma\mathbf{x}_r + (1 - \gamma)\bar{\mathbf{x}} \tag{4.24}$$

where γ is the expansion coefficient typically equal to 2. If $f_e < f_r$, then $\mathbf{x}^{n+1}$ is replaced with $\mathbf{x}_e$, otherwise it is replaced with $\mathbf{x}_r$. If $f_r \geq f_n$, a contraction step is performed. There are two different types of contraction step. The first is if $f_n \leq f_r < f_{n+1}$ where the contracted point is given by:

$$\mathbf{x}_c = \rho\mathbf{x}_r + (1 - \rho\bar{\mathbf{x}}) \tag{4.25}$$

where ρ is the contraction coefficient typically equal to $\frac{1}{2}$. Alternatively, if $f_r \geq f_{n+1}$, the contraction point is computed using:

$$\mathbf{x}_{cc} = (1 + \rho)\bar{\mathbf{x}} - \rho\mathbf{x}^{n+1} \tag{4.26}$$

If $f_c < f_r$ or $f_{cc} < f_{n+1}$, the contracted point is accepted, otherwise a shrink is triggered. The shrink consists of replacing every point except the best using the following equation:

$$\mathbf{v}^i = \mathbf{x}^1 + \sigma(\mathbf{x}^i - \mathbf{x}^1) \tag{4.27}$$

where σ is the shrink coefficient typically equal to $\frac{1}{2}$. The four coefficients α, γ, ρ, and σ are the tuning parameters for the method [127, 128]. SIM uses a metric similar to diversity which uses the standard deviation of the cost of the simplex to determine when the size of the simplex is too small to continue. This criterion is given by [127]:

$$\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}(f(\mathbf{x}_i) - \bar{f})^2} \leq 10^{-10} \tag{4.28}$$

where $\bar{f}$ is the average cost of all points in the simplex.

4.1.7 Invasive Weed Optimization (IWO)

IWO is based on the reproduction and dispersion of weeds in nature. The general principle behind the algorithm is that stronger weeds generate more seeds which are dispersed throughout the search space and grow into plants. Starting from an initial population of plants, usually 5 to 10, each plant produces a number of seeds s_i based on its cost. A linear relation is used to determine the number of seeds given by:

$$s_i = s_{min} + (s_{max} - s_{min})\frac{f_i - f_{max}}{f_{min} - f_{max}} \tag{4.29}$$

s_{max} is typically set from 3 to 5 while s_{min} is set from 0-1. Seeds are dispersed according to a normal distribution $\mathbf{x}^s \sim N(\mathbf{x}^i, \Sigma^i)$ centered on the parent plant with a covariance matrix Σ^i. An adaptive approach is used to compute each σ_i^j in Σ_i based

on the relative cost of the parent plant specified by:

$$\log \sigma^i(j) = \log \sigma_{lo}(j) + (\log \sigma_{hi}(j) - \log \sigma_{lo}(j)) \frac{f_i - f_{min}}{f_{max} - f_{min}} \tag{4.30}$$

The log of the standard deviation is used to allow for a better range of σ values when there is a large difference between f_{min} and f_{max}. The bounds σ_{hi}^j and σ_{lo}^j are specified as a percentage of the total search range for each variable, ranging from 10% of the search range to 0.001%. After all plants have reproduced, the best p_{max} plants out of both of the parents and children are retained for the next generation. It is common to retain 10 to 50 plants depending on the problem. The tuning parameters for IWO are the initial population, p_{max}, s_{min}, s_{max}, σ_{lo}, and σ_{hi} [122, 123, 124, 125]. The diversity based stopping criteria are also used for IWO.

4.1.8 Tabu Search (TS)

While tabu search has primarily been used for combinatorial optimization problems, there are some variants for continuous problems. The continuous tabu search (CTS) algorithm developed by Siarry and Bethiau shows good performance on many multimodal functions [133]. The primary concept in TS is the tabu list which contains points in the search space that have been recently evaluated. A point is considered tabu if it falls within a ball of radius r_t around any point in the tabu list. The distance from a point to the center of the tabu ball is scaled based on a representative value for each variable. This prevents the method from quickly returning to previously explored areas. A limited number of points, n_t, are retained in the tabu list with the oldest point removed on every iteration, allowing TS to return to promising areas after a period of time.

The first step of TS is the generation of n neighbors around the current point that are not within a tabu ball. To uniformly search the parameter vector space,

the neighborhood is broken into m concentric hyper-rectangles [130]. A geometric partitioning is used such that the side lengths are given by:

$$h^{m-i+1}(j) = \frac{h^m(j)}{2^{i-1}} \quad (i = 1, 2, ..., m) \tag{4.31}$$

The maximum side length h^m is set as a fraction of the total search range for each variable. This partitioning strategy was found to be the most effective across multiple functions [133]. The neighbors are generated by randomly sampling a single point within each partition. A point sampled in the ith partition is given by:

$$x_{k+1}^i(j) = \begin{cases} x_k(j) + h^{i-1}(j) + r(h^i(j) - h^{i-1}(j)) & \text{if } j = r_i \text{ and } r_d > 0.5 \\ x_k(j) - h^{i-1}(j) - r(h^i(j) - h^{i-1}(j)) & \text{if } j = r_i \text{ and } r_d < 0.5 \\ x_k(j) + h^i(j)(1 - 2r) & \text{if } j \neq r_i \end{cases} \tag{4.32}$$

where r is a uniform random number, r_i is a random integer from 1 to the number of dimensions, and r_d is a uniform random number used to determine the direction of the sampling. After all neighbors have been generated, the algorithm moves to the neighbor with the lowest cost that is not tabu, even if the value is higher. After moving to the best neighbor, the previous point is added to the tabu list. Only the N most recent points are stored in the tabu list. If a lower cost cannot be found after 20 iterations, the neighborhood size and tabu radius are halved and the tabu list is cleared, resuming the search from the best point found so far. Typical tuning parameters values for TS are $n_t = 10$, h^m equal to 25% of the search range, and r_t equal to 10% of the smallest side length of the hyper-rectangle. TS stops when there has been a specified number of reductions in neighborhood size without sufficient cost reduction [130].

4.1.9 Ant Colony Optimization (ACO)

Like TS, ACO was also originally developed for combinatorial optimization with multiple methods for applying the general approach to solve continuous problems. One of the simplest and most effective approaches was developed by Socha and Dorigo which adapts the concept of pheromone information for continuous applications [126]. ACO begins with a solution archive of size K, typically made up of 50 potential solutions. At each iteration, m ants are constructed, one dimension in the parameter space at a time, using a Gaussian kernel probability density function (PDF) based on the solution archive. The number of ants can vary from 2 to 25 depending on the problem. The PDF is a weighted sum of Gaussian kernels centered at each solution $\mathbf{x}_i$ in the archive. Instead of sampling from the entire PDF, a single kernel is chosen such that the new ant is generated according to:

$$x(j) \sim N(x^l(j), \sigma^l(j)) \tag{4.33}$$

where l denotes the index of the chosen kernel. The selection process begins by sorting the solutions in the archive based on quality, with the best solutions ranked first. These ranks are used to weight each solution according to:

$$w_i = \frac{1}{qK\sqrt{2\pi}} e^{-\frac{(i-1)^2}{2q^2K^2}} \tag{4.34}$$

where q controls the behavior of the weights. Smaller values of q favor the best solutions while larger values distribute the probabilities more evenly. A value of 10^{-4} gives the highest weights to the best few solutions. Each solution is then assigned a probability to be selected p_i given by:

$$p_i = \frac{w_i}{\sum_{r=1}^{K} w_r} \tag{4.35}$$

Note, the choice of kernel is only performed once per ant. Before the chosen kernel can be sampled, the standard deviation must be determined. The standard deviation is based on the average distance between the chosen solution and the rest of the archive such that for dimension j:

$$\sigma^l(j) = \xi \sum_{i=1}^{K} \frac{|x^i(j) - x^l(j)|}{K-1} \tag{4.36}$$

where ξ controls the convergence rate of the optimizer with $\xi = 0.85$ commonly used. After all of the ants are constructed, the K best solutions are retained for the next generation. The ACO tuning parameters are K, m, q, and ξ [126]. As with the other population based optimizers, ACO stops based on the diversity criteria.

4.2 Optimizer Performance Evaluation

A key aspect of meta-optimization is an assessment of an optimizer's performance when employed on a particular problem. The effectiveness of each optimizer is used to determine when it is appropriate to change optimizers and which optimizer should be deployed next. Various measures of online optimizer performance, such as computation time, cost reduction, predicted performance, and optimizer risk, can be used as a single performance metric or combined in a composite metric [60, 72, 71]. For the purposes of meta-optimization, a good optimizer is one which has low computation time and high cost reduction. In order to codify these characteristics, a metric is defined that is a function of the percent objective function reduction and the number of function calls. The percent reduction in the objective function is used such that the metric is independent of the current objective function value while total function calls is a convenient surrogate for computation time. A metric combining these two measures provides a characterization of the efficiency of the optimizer, evaluating the cost reduction per unit of computation time. Slightly greater weight is placed on the

magnitude of the cost reductions than computation time as optimizers which achieve greater cost reductions should be rated well. Finally, the metric must provide reasonable evaluation of optimizer performance throughout the optimization process and across all problems.

Based on these considerations, the function shown in Figure 4.2 was chosen as a performance metric for this implementation of meta-optimization. This function represents an example efficiency function intended to provide good evaluation of optimizer performance over a wide range of problems. The overall shape of the function was modeled off of a two dimensional sigmoid function with the output η ranging from 0 for poor performance to 1 for good performance. The inputs to the function are the normalized percent cost reduction J^* and the normalized computation time t^*. To obtain these measures, the percent cost reduction J is divided by a reference cost reduction J_{ref} while the computation time t is divided by a reference computation time t_{ref}. These reference values are set to represent satisfactory performance for each optimizer as defined by the user. Since the amount of cost reduction and computation time can vary greatly between optimizers, the function must cover a large range in J^* and t^*, with values up to 20 for both measures. The function is designed such that the point $J^* = 1$ and $t^* = 1$ has a value of 0.5, indicating neutral performance. Also, the function places a larger weight on higher cost reductions, benefiting functions that run longer while achieving good cost reductions.

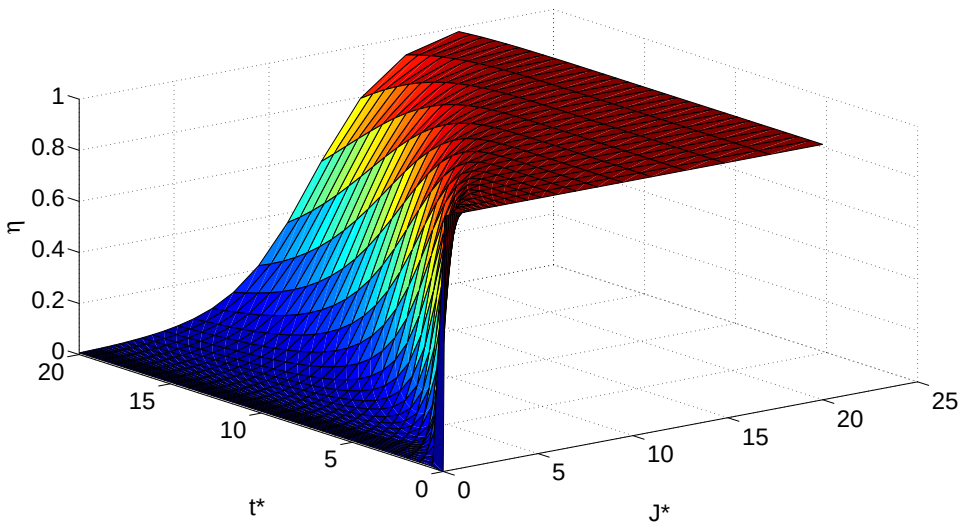


Figure 4.2: Optimizer Performance Metric Based on Cost Reduction and Computation Time

While running, every optimizer is monitored for adequate performance to determine if the optimizer is progressing well or has slowed or stalled and should be stopped. Efficiency is evaluated over a moving window of function calls N_w. The reference computation time t_{ref} is also set to N_w as the window is the typical range over which performance is evaluated. The first efficiency check is performed after a minimum number of 125% of the window (N_w), with a delay of half of the window before efficiency is evaluated again. This ensures the optimizer is given a reasonable chance to reduce the cost and allows the optimizer to continue running for some time as long as it is performing well. If the efficiency is above a threshold of 0.95, the optimizer will continue to run, otherwise it will continue with probability equal to the efficiency value.

4.3 Optimizer Selection

The role of the optimizer selection process is to iteratively deploy a single optimizer with preference towards optimizers which perform well on the current problem.

As the meta-optimizer progresses towards the solution, different optimizers will be more effective on the problem than others. The optimizer selector learns from the performance of each optimizer to ensure appropriate optimizers are selected. The optimizer selection takes the form of a variable structure learning automaton where an optimizer is selected based on a probability distribution. This work considers various potential methods for updating the probabilities, namely:

Linear Reward-Penalty: A simple update rule is the linear reward-penalty (L_{R-P}) scheme where the update rate of the probabilities are the same regardless of the output from the problem. For a continuous output, such as the efficiency η_i of the current optimizer, the probability updates are given by the following equations [134]:

$$p_i(k+1) = p_i(k) - \eta(k)a_{LA}p_i(k) + [1-\eta(k)][\frac{b_{LA}}{m-1} - b_{LA}p_i(k)] \text{ if } \alpha(k) \neq \alpha_i \quad (4.37)$$

$$p_i(k+1) = p_i(k) + \eta(k)a_{LA}[1-p_i(k)] - [1-\eta(k)]b_{LA}p_i(k) \text{ if } \alpha(k) = \alpha_i \quad (4.38)$$

The value a_{LA} is the reward parameter, the value b_{LA} is the penalty parameter, and m is the total number of optimizers. These equations operate by modifying the probability of the optimizer which just ran based on its performance and distributing that change to the other optimizers. As a ground rule, a minimum probability of 3% is maintained for each optimizer to prevent the probabilities of some optimizers being driven towards zero. This also sets a maximum probability for any one optimizer at 76%.

Relative Performance: An alternate probability update rule allocates probabilities based on the relative performance of each optimizer. Each optimizer is assigned a weight based on the efficiency of that optimizer from the last time it was used. The probabilities for each optimizer are distributed based on the weights according to:

$$p_i = \frac{w_i}{\sum_{i=1}^{m} w_i} \quad (4.39)$$

The weights w_i are assigned in a discrete or continuous manner. For the discrete case, a set of bins are defined with each bin corresponding to a range of efficiency values. Each bin is assigned a weight which is given to all optimizers within that bin. An example weighting scheme is given in Table 4.2 below. Here, η_i corresponds to the upper limit of each bin.

Table 4.2: Example Discrete Weighting Scheme

η_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
w_i	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5

For the continuous case, a function is defined to relate efficiency to weight given by $w_i = W_0 + W_1 h(\eta_i)$. Example weighting functions $h(\eta)$ include linear, quadratic, and square root relations. The efficiency values for each optimizer are scaled using an exponential decay based on the time since that optimizer was last called.

Learning Free: Two additional methods of selecting algorithms are reported in this work which do not use learning. The first method randomly selects optimizers with equal and constant probabilities. This would be equivalent to setting a_{LA} and b_{LA} in Eq. 4.37 to zero. The second method selects optimizers sequentially based on the order of the optimizers in Section 4.1. At the beginning of each run, a random optimizer is selected to be deployed first with the sequence repeating until the meta-optimizer stops.

4.4 Optimizer Manager

The optimizer manager is in charge of handling the hand-off between optimizers and ensures each optimizer is provided the information it needs to operate on the problem. Given the diverse range of optimizers employed by the meta-optimizer, the exchange between optimizers is critical to the smooth operation of the algorithm. Each optimizer has various inputs and outputs which may not necessarily match

with the next optimizer that will be employed. The responsibility of the optimizer manager is to provide a centralized system for starting each optimizer when it is selected, including a set of initial points and any information needed to initialize the optimizer. This also includes reseeding portions of the population and restarting the optimization process when necessary.

Initialization: At the beginning of the meta-optimization process, the global population is seeded with a large number of points generated based on a uniform random distribution over the search ranges for each parameter. This population is shared between all of the optimizers, allowing for the exchange of information between optimizers. The population must also be sufficiently large to provide enough points for any optimizer.

Transition to Single Point Optimizer: For SD, CG, BFGS, and TS, only a single point is operated on at a time. When deployed, the best solution from the global population is used as the starting point for these optimizers. When the optimizer has completed, the previous best solution is replaced with the new value obtained from the optimizer.

Transition to Population Based Optimizer: PSO, DE, SIM, IWO, and ACO all require a number of points drawn from the common population. Based on the population sizes for each optimizer, the best points are taken from the population, guaranteeing preservation of the global best solution found so far. Before these points are provided to the optimizer, the diversity in the set is checked using Eq. 4.18. For these optimizers, the diversity plays a critical role in the ability of the optimizer to explore the parameter space. If the diversity falls below a threshold $\sigma_{d,min}$, a portion of the set is reseeded. After running, the set of points are returned to the global population to be used by the next optimizer.

Reseeding of the Current Population: When reseeding is triggered, the manager reseeds a percentage of the population (RS). The goal of reseeding is to increase the

diversity in the population used by the current optimizer, allowing the optimizer to resume searching and potentially escape a local minimum. This is necessary to prevent different optimizers from conflicting where the population returned from one optimizer does not allow the next optimizer to appropriately operate. Two methods are employed to generate new points, balancing exploitation of a region of interest with exploration of the parameter space. The first process is exploitation which is performed with probability r_{RS}. New points are seeded based on a kernel density function (KDF) built from all previous solutions evaluated by the optimizers. As the meta-optimizer runs, optimizers will tend towards certain regions of the parameter space with low cost. Exploitation seeks to place more points within these regions, aiding the optimizers in refining the cost. The remaining points are seeded through exploration by randomly sampling from the full search range. This approach provides the optimizers with a highly diverse population, greatly increasing their exploration capacity. Most importantly, exploration places points far from any local minima, giving the optimizers an opportunity to search a new region of the parameter space and potentially escape a local minimum.

Restarting of Meta-Optimization Process: In the event that the meta-optimizer remains locked in a local minimum for a long period of time, the entire meta-optimization process is restarted. The frequency of restarts is governed by the re-starting threshold (N_R) which sets a limit on the number of function calls meta-optimization can expend while stalled in a local minimum. Unlike reseeding which only updates a small portion of the local population used by an optimizer, the re-starting process generates an entirely new global population shared by all optimizers. When the new population is generated, an exclusion zone is placed around all previously detected local minima to prevent any optimizer from starting too close to the local minima and quickly returning. Exclusion zone types considered in this implementation are a ball of set radius, similar to the tabu ball from TS, or a range in

each dimension. Like in TS, distance from the center of each ball is scaled using a representative value for each parameter. The radius of the exclusion zone r_R is specified as a percentage of the search range for each parameter. After a certain number of restarts, the meta-optimizer stops, returning the best local minima found as the solution.

4.5 Auto-Tuning

The performance of any optimizer is dependent on the various parameters which control the optimizer's behavior. For example, PSO has four tuning parameters: population size, inertial weight, and two learning factor coefficients. Tuning parameters are unique to every optimizer and can vary greatly between applications. Typically, the selection of algorithm tuning parameters is determined by the user, whether from good values that have been used previously or through manual tuning of the parameters. For new optimization problems, the best parameters may not be known initially and require a significant amount of user effort to properly tune. Instead, auto-tuning is performed where the tuning procedure is conducted online within the meta-optimization procedure. This improves reliability by adapting the optimizers to the current problem, allowing for hands-off operation of the individual optimizers.

Auto-tuning is performed using a wandering search which gradually explores the parameter space in a random manner. This type of search was chosen for this implementation as it is very simple to implement with basic logic to intelligently guide the search process. It is generally impractical to utilize traditional numerical optimization algorithms to perform auto-tuning within meta-optimization as this would add a significant amount of computation time that is not spent working on the optimization problem itself. When auto-tuning is activated, the optimizer is run twice for 1000 function calls, once with the current parameters and once with new parameters. This provides a side by side comparison of the optimizer with the two parameter sets over

a reasonable number of function calls. Only one parameter is tuned at a time to limit interactions between tuning parameters, with each parameter having an equal probability of being selected.

Each parameter has a search magnitude and a set of positive and negative increment probabilities. The new parameter value is generated by perturbing the current parameter with a random length step based on the search magnitude in a random direction selected based on the positive or negative probabilities. After both runs are complete, the optimizer is checked for reduction in objective function, average objective function value, and diversity. The new parameter is accepted if it has lower average objective function value and higher diversity, achieving Pareto dominance. If no new parameters are selected after both directions have been explored, the search magnitude is reduced by a specified factor. The search direction probabilities are updated using a P-model learning automaton given in Eq. 4.40 which uses the acceptance or rejection of the new parameters as its input [134].

$$\begin{aligned} \text{if accepted} &\begin{cases} p_i(n+1) = p_i(n) + a[1 - p_i(n)] \\ p_j(n+1) = (1-a)p_j(n) \end{cases} \\ \text{if not accepted} &\begin{cases} p_i(n+1) = (1-b)p_i(n) \\ p_j(n+1) = b + (1-b)p_j(n) \end{cases} \end{aligned} \tag{4.40}$$

Here, i is the index of the tested direction, j is the index of the opposite direction, and the reward and penalty parameters a and b are set to 0.5. At the conclusion of the auto-tuning run, the results with the lowest objective function value are retained, independent of whether or not the new parameter is accepted. If any additional points are needed in the population due to a change in population size, they are added using the reseeding procedure from Section 4.4.

CHAPTER 5
BENCHMARK FUNCTION TESTING OF META-OPTIMIZATION

Mathematical benchmark functions are a common tool for the design and testing of new optimization strategies. These functions are simple mathematical relations with known structure and solutions. The advantage of using benchmark functions is that they provide a common set of functions for researchers to use for side by side comparison of different algorithms. The varying complexity and occurrence of local minima in these functions allows for the evaluation of optimizers on a wide range of problem types. Knowledge of the function solution also helps guide development of new optimizers as performance can be easily graded.

Benchmark functions are also useful as a surrogate for practical optimization problems. For engineering optimization problems such as smart projectile parameter estimation, the computation time required to solve the problem makes it impractical to use the actual problem to test and develop new optimizers. For example, the projectile parameter estimation problem runs an entire six degree-of-freedom simulation for each trajectory in the cost function. Even on a high performance computer system, only a few function calls may be executed every second, making it difficult to use for extensive trade studies and analysis. Real world problems may also lack a known optimal solution if little information about the system is known *a priori*.

In this chapter, a suite of benchmark functions are used to analyze the behavior of the meta-optimization framework on solving various problems. To establish the efficacy of meta-optimization, the framework is compared to the individual optimizers on this suite of functions. A series of trade studies are then conducted, investigating the performance and effects of the individual components of the framework. These trade studies are used to inform the selection of a configuration for the meta-optimizer

that is robust for use in system identification in Chapter 6. Finally, the tuned meta-optimizer is compared to other state of the art optimizers on a suite of benchmark functions used in optimization competitions.

5.1 Benchmark Function Suite

Ten well known benchmark functions often used by other researchers were chosen to evaluate meta-optimization. Each function has a different topology and dimensionality with a wide range of difficulty. The functions used in this chapter are given in Table 5.1 with equations following. Of these functions, only the elliptic function f_8 is unimodal while the rest are multimodal. The Rosenbrock function f_1 has a large banana shaped valley with a local minimum at (-1,1,...1). Functions f_2 and f_3 are periodic with a huge number of local minima. The Levy function f_4 is a valley with a mostly flat bottom with ripples creating local minima. The Ackley function f_5 has a deep funnel with numerous local minima along the sides. The Schaffer F6 function f_6 resembles a multi-dimensional wave with repeating crests and valleys. Finally, f_7 is a combination of f_1 and f_3, producing a very complex, highly multimodal structure. All of these functions are designed to have a global minimum value of 0.0 and are evaluated over a range of $x_0 = [-10, 10]$.

5.1.1 Rosenbrock Function

$$f_1(\mathbf{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{5.1}$$

5.1.2 Rastrigin Function

$$f_2(\mathbf{x}) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)] \tag{5.2}$$

Table 5.1: Benchmark Suite Function Descriptions

Function Number	Function Name	Function	Dimension	Global Min	Ref
1	Rosenbrock	f_1	30	(1,...1)	[135]
2	Rosenbrock	f_1	100	(1,...1)	[135]
3	Rastrigin	f_2	30	(0,...0)	[135]
4	Griewank	f_3	10	(0,...0)	[135]
5	Levy	f_4	30	(1,...1)	[136]
6	Ackley	f_5	10	(0,...0)	[135]
7	Schaffer F6	f_6	10	(0,...0)	[135]
8	Expanded Griewank Plus Rosenbrock	f_7	10	(1,...1)	[135]
9	High Conditioned Elliptic	f_8	30	(0,...0)	[135]
10	High Conditioned Elliptic	f_8	100	(0,...0)	[135]

5.1.3 Griewank Function

$$f_3(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{5.3}$$

5.1.4 Levy Function

$$f_4(\mathbf{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{n}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_n - 1)^2[1 + \sin^2(2\pi w_n)] \tag{5.4}$$

Where $w_i = 1 + \frac{x_i - 1}{4}$.

5.1.5 Ackley's Function

$$f_5(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{5.5}$$

5.1.6 Expanded Schaffer F6 Function

$$g(x,y) = 0.5 + \frac{\sin^2\left(\sqrt{x^2+y^2}\right) - 0.5}{(1+0.001(x^2+y^2))^2} \tag{5.6}$$

$$f_6(\mathbf{x}) = g(x_1,x_2) + g(x_2,x_3) + ... + g(x_{n-1},x_n) + g(x_n,x_1)$$

5.1.7 Expanded Griewank Plus Rosenbrock Function

$$f_7(\mathbf{x}) = f_3(f_1(x_1,x_2)) + f_3(f_1(x_2,x_3)) + ... + f_3(f_1(x_{n-1},x_n))f_3(f_1(x_n,x_1)) \tag{5.7}$$

5.1.8 High Conditioned Elliptic Function

$$f_8(\mathbf{x}) = \sum_{i=1}^{n} (10^6)^{\frac{i-1}{n-1}} x_i^2 \tag{5.8}$$

5.2 Nominal Configuration Results

A nominal configuration of the meta-optimizer was chosen to demonstrate the general performance of the framework on these benchmark functions. The parameters for this nominal configuration are shown in Table 5.2. It should be noted that this meta-optimization configuration is not specialized to perform well on these functions, but is intended to provide reasonable performance on most problems.

Table 5.2: Nominal Meta-Optimization Parameters

Parameter	Name	Value
	Algorithm Selection Method	Learning Automaton
a_{LA}, b_{LA}	Learning Automaton Parameters	0.3
N_w	Efficiency Check Window	4000
J_{ref}	Reference % Cost Reduction	4.0%
RS	Reseeding Rate	0.8
r_{RS}	Reseeding Refinement Probability	0.5
$\sigma_{d,min}$	Reseeding Diversity Limit	$5e^{-6}$
N_R	Restarting Threshold	500,000
	Restarting Exclusion Zone	Ball
r_R	Exclusion Zone Radius	1.0

5.2.1 Single Function Results

In order to highlight the manner in which the meta-optimization algorithm proceeds during execution, detailed results are provided using the Ackley function (f_5). The 2 dimensional Ackley function is shown in Figure 5.1. This function forms a funnel around the global minimum at $(0, 0)$ with many local minima along the funnel, making it a very difficult function to solve for many optimizers. Figure 5.2 shows the change in cost as the meta-optimizer progresses as well as which optimizer is running at a given time. Note, the vertical lines are used to align significant cost reductions with the corresponding optimizer. For this particular execution of meta-optimization, during the initial phase of optimization (0 to 0.5×10^5 function calls), notable reductions in the cost are achieved by BFGS and DE. This is followed by a long period (0.5×10^5 to 1.5×10^5 function calls) where numerous optimizers were deployed with only a small number reducing the cost. Towards the end of the optimization process (after 1.5×10^5 function calls), a large reduction in the objection function is obtained using PSO. Even though PSO was performing well, it stopped due to the diversity in the population dropping below the set threshold. Next, TS was deployed, running until

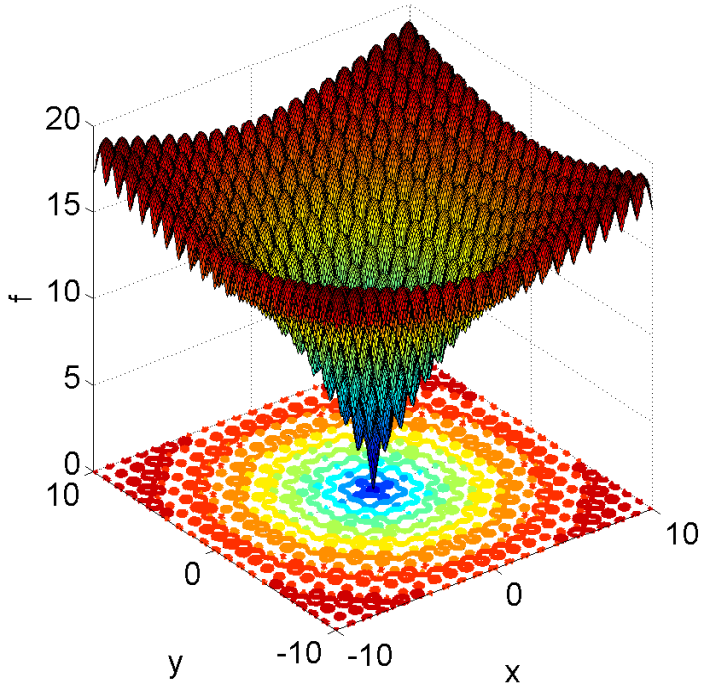


Figure 5.1: 2-Dimensional Ackley Function

reaching its stopping criteria of excessive reductions in neighborhood size. Finally, TS provided additional small reductions before SD finally reached the cost threshold. The overall behavior follows the general structure of this function which is very flat with many local minima and a very deep and narrow funnel with the global minimum at the bottom. While the stochastic nature of the meta-optimizer produces different results for a single instance, the overall trends are similar over a large number of trials.

The switching behavior of the meta-optimizer can also be seen in Figure 5.2. When stalled in a local minimum where no optimizer is doing well, the meta-optimizer will quickly switch between optimizers until it finds one that can continue progress. When an optimizer is performing well, it will run for a long period before reaching its stopping criteria. On occasion, the meta-optimizer may try another optimizer which may also perform well or switch back to the original optimizer. It may also continue to choose the current optimizer, stringing together multiple calls.

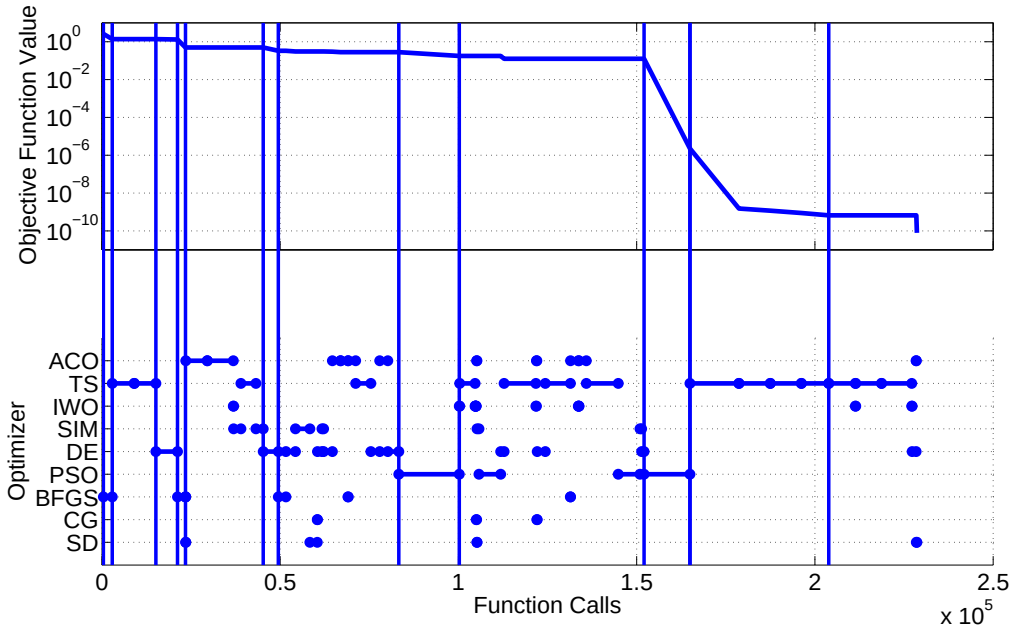


Figure 5.2: Ackley Function Cost Profile with Optimizer Deployment History

The total percent cost reduction for each optimizer is shown in Figure 5.3. This is computed by summing the percent reduction for each optimizer on every iteration. Only six optimizers were able to reduce the cost on this run with TS and PSO providing the most. Given the nature of an individual run, not every optimizer is given a good opportunity to work on the problem. As seen in Figure 5.4, the more effective optimizers like TS and DE were given a few more opportunities than the other optimizers while CG had the fewest with 3 due to its inability to make progress when deployed. These differences are also due to the fact that the optimizers continue to be used as long as they are performing well, allowing them to achieve large cost reductions for a single deployment. This can be seen clearly in Figure 5.5 where PSO, DE, and TS ran for the most amount of time. PSO in particular was only called four times, but each time ran for a long time. The hill climbers also tend to have short run times due to quickly hitting their stopping criteria when finding a local minimum.

The probabilities of being selected for each optimizer over the course of the run are

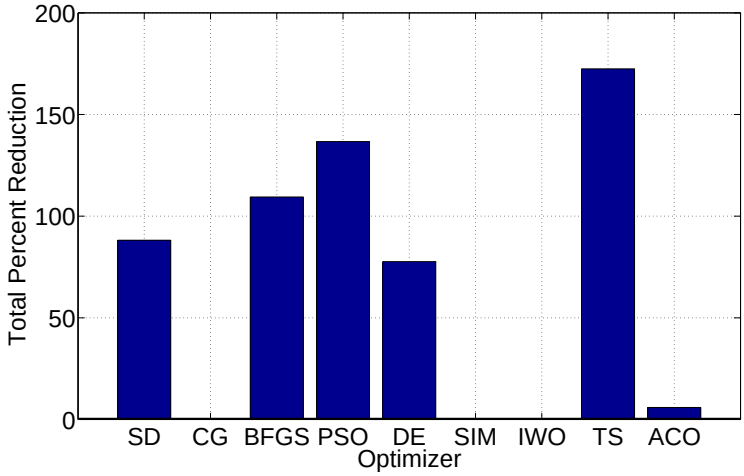


Figure 5.3: Ackley Function Total Percent Cost Reduction

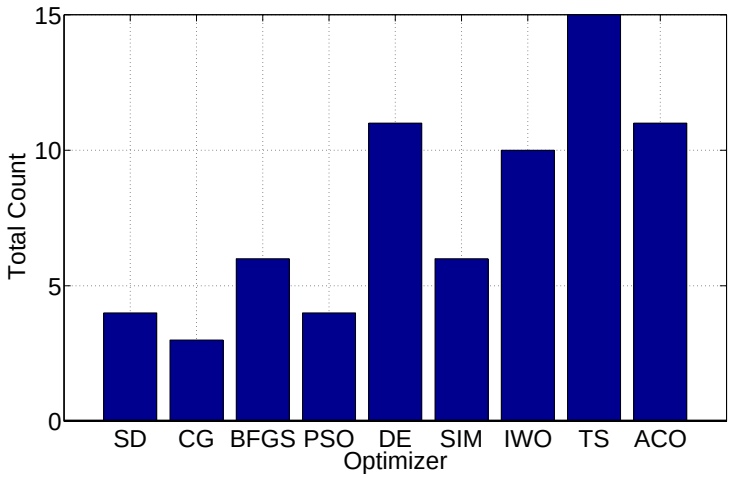


Figure 5.4: Ackley Function Total Number of Calls of Each Optimizer

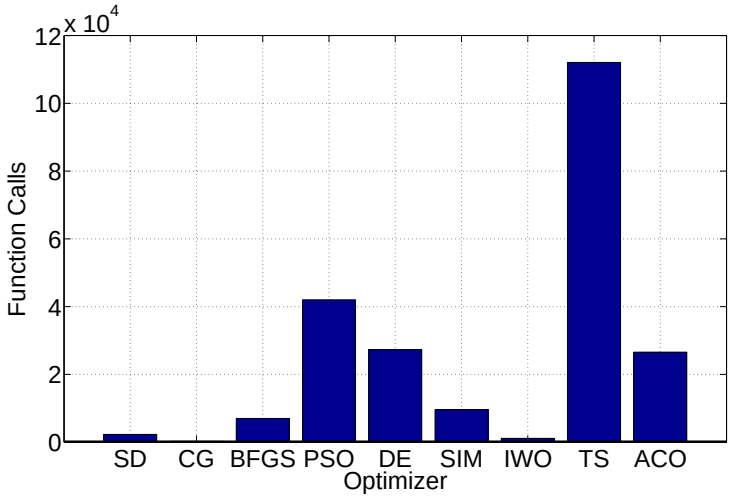


Figure 5.5: Ackley Function Total Number of Function Calls Used by Each Optimizer

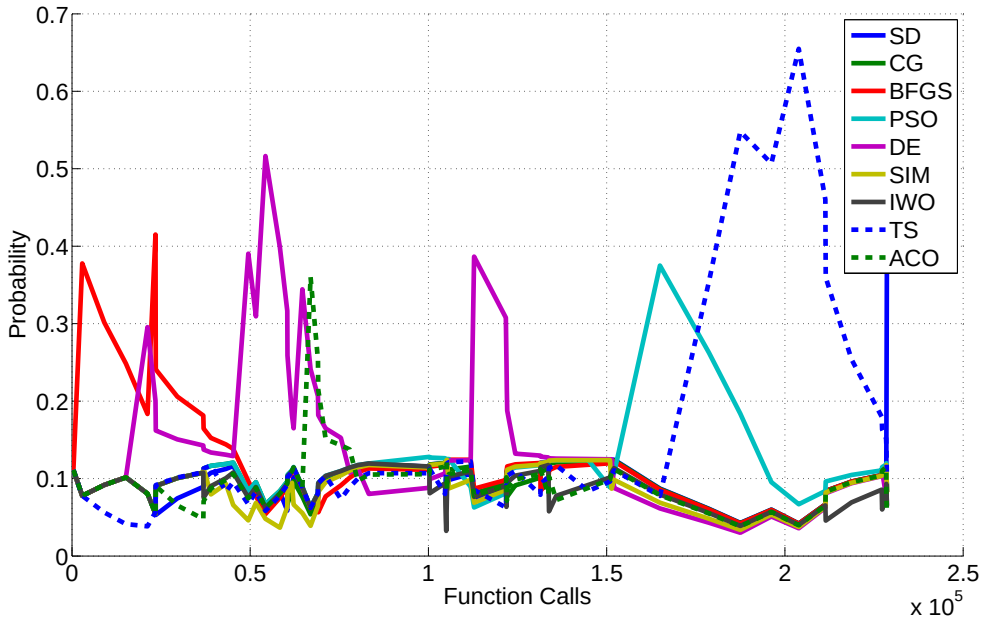


Figure 5.6: Ackley Function Optimizer Probability vs Function Calls

shown in Figure 5.6. When an optimizer does well, the learning automaton increases its probability, making it more likely to be called. As soon as an optimizer does poorly, its probability is reduced, giving the other optimizers an opportunity to work on the problem. Generally, only one or two optimizers have a high probability at a given time with the other optimizers at roughly equal probability.

5.2.2 Benchmark Suite Results

Each of the individual optimizers used by the meta-optimizer were run on the set of benchmark functions to use as a comparison to the meta-optimizer. For each function, the individual optimizers were run 500 times with the results averaged for each function. Each optimizer was given a budget of 10,000,000 function calls to solve the problem. A cost threshold of 10^{-10} was used to stop the optimizers and denote a successful solution. In addition, optimizers were stopped when their stopping criteria from Section 4.2 were met. It should be noted that the same set of parameters were

used for each optimizer across all tests and the parameters were not tuned to any particular function. The meta-optimizer was also run 500 times on each function with a budget of 10,000,000 function calls or 10 restarts. The success rate, mean final cost, and mean functions calls for each optimizer are given in Tables 5.3-5.5.

Table 5.3: Individual Optimizer Success Rate (%) on Benchmark Suite

Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
SD	70.6	72.4	0	32.2	0	0	0	0	100	100
CG	54.8	50.2	0	15.6	0	0	0	0	100	100
BFGS	61.2	54.4	0	20.2	0	0	0	0	100	100
PSO	63	49.6	0	0.8	0.2	0	0	0	100	100
DE	76.6	0	0	3.6	38.2	1	0	0	100	99.4
SIM	0	0	0	38.4	0	0	0	0	0	0
IWO	0	0	0	0	0	0	0	0	0	0
TS	80.2	3	0	1.2	0	0	0	0	0	0
ACO	59.4	0	0	0.2	0	0	0	0	100	2
MO	100	100	100	65.4	100	100	2	92.4	100	100

Examining the performance of the individual optimizers in Table 5.3, only f_8 (Elliptic Function) could be solved every time by at least one optimizer. A number of optimizers also have a high success rate on f_1 (Rosenbrock Function). Function f_8 is easy to solve for hill climbers because it is unimodal while f_1 has only a single local minimum and a very large basin of attraction around the global minimum. In general, this group of functions provide a challenge for the individual optimizers with a wide variation in performance between functions. The meta-optimizer, on the other hand, matches or exceeds the performance of the individual optimizers on every function and is able to solve every problem at least some of the time. It also produces a lower cost than any optimizer as seen in Table 5.4, in particular on the problems meta-optimization could not solve every time. This demonstrates how meta-optimization is able to combine the various optimizers together to achieve better performance than

Table 5.4: Mean Final Cost for Individual Optimizers on Benchmark Suite

Function	1	2	3	4	5	6	7	8	9	10
SD	1.2e+00	1.1e+00	9.9e+02	5.9e-02	7.5e+01	2.2e+00	6.4e+03	6.6e-01	2.7e-12	8.4e-12
CG	3.0e+02	2.2e+04	9.6e+02	8.0e-02	7.5e+01	2.2e+00	2.1e+04	6.8e-01	9.1e-12	1.2e-11
BFGS	4.5e+03	2.3e+05	9.8e+02	9.3e-02	7.8e+01	2.2e+00	1.6e+05	6.7e-01	7.4e-12	3.7e-12
PSO	1.4e+00	1.8e+00	7.9e+01	4.9e-02	4.5e+00	3.5e-01	6.6e-01	1.7e-01	9.4e-11	8.7e-11
DE	3.2e+05	1.5e+07	4.5e+01	6.1e-02	5.3e-01	3.3e-01	1.4e+00	4.1e-01	9.1e-11	2.6e-07
SIM	8.7e+01	1.4e+02	3.6e+01	2.4e-02	9.9e-01	6.1e-01	9.7e+01	3.7e-01	2.9e+02	5.2e+02
IWO	8.6e-01	4.5e+01	3.8e+02	7.5e-02	4.8e+01	1.6e+00	2.5e+00	5.6e-01	1.9e+02	3.9e+03
TS	9.4e-01	9.2e-01	2.0e+02	7.6e-02	6.3e+01	1.2e+00	1.5e+00	6.2e-01	1.9e+01	8.4e+02
ACO	1.5e+00	1.4e+07	1.5e+02	5.6e-02	1.8e+01	7.2e-01	1.3e+00	3.9e-01	9.1e-11	5.7e+03
MO	6.6e-11	5.3e-11	3.8e-11	3.5e-03	3.2e-11	7.3e-11	1.4e-01	4.1e-03	3.5e-12	4.0e-12

Table 5.5: Mean Successful Function Calls for Individual Optimizers on Benchmark Suite

Function	1	2	3	4	5	6	7	8	9	10
SD	4.8e+05	1.6e+06	0.0e+00	1.7e+02	0.0e+00	0.0e+00	0.0e+00	0.0e+00	8.3e+02	2.9e+03
CG	7.9e+04	3.1e+05	0.0e+00	6.5e+03	0.0e+00	0.0e+00	0.0e+00	0.0e+00	9.1e+02	3.1e+03
BFGS	8.0e+03	4.6e+04	0.0e+00	3.1e+02	0.0e+00	0.0e+00	0.0e+00	0.0e+00	9.5e+02	3.2e+03
PSO	1.5e+06	2.9e+06	0.0e+00	2.9e+04	2.4e+04	0.0e+00	0.0e+00	0.0e+00	3.4e+04	3.8e+05
DE	3.5e+05	0.0e+00	0.0e+00	2.2e+04	6.9e+04	4.0e+04	0.0e+00	0.0e+00	9.5e+04	4.9e+05
SIM	0.0e+00	0.0e+00	0.0e+00	1.5e+03	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
IWO	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
TS	3.0e+06	8.7e+06	0.0e+00	3.2e+03	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00
ACO	3.1e+05	0.0e+00	0.0e+00	2.0e+03	0.0e+00	0.0e+00	0.0e+00	0.0e+00	1.4e+04	1.0e+05
MO	3.1e+05	4.6e+05	1.3e+06	3.3e+06	1.2e+05	1.3e+05	3.9e+06	2.5e+06	6.3e+04	1.4e+05

what any one optimizer can achieve alone. While meta-optimization is able to reliably solve these problems, it is not necessarily faster than all of the individual optimizers at solving each problem as shown in Table 5.4. This is expected as meta-optimization will give many different optimizers opportunities to work on a problem and may not always choose the fastest optimizer. Given the significant increase in performance over the individual optimizers, this increase in computation time is for many scenarios an acceptable trade off.

Additional metrics for the performance of the meta-optimizer are given in Table 5.6. For most functions, the meta-optimizer required at least one restart on a number of runs with functions 4, 7, and 8 requiring restarts on over 60%. On average, about 7 restarts were needed for function 4, 9 restarts were needed for function 7, and 4 for function 8. Functions 1, 2 and 3 also required some restarts but converged without a restart on the majority of runs and usually only required a single restart. The meta-optimizer also spent a majority of the time stalled on functions 4, 7, and 8 indicating the difficulty of these problems and the propensity of these optimizers to find local minima. In this case, the meta-optimizer was considered stalled when there was little or no cost reduction for 50,000 function calls. The final metrics, optimizer count deviation and average probability deviation, measure the deviation from the case where each optimizer is called an equal number of times with equal probability. For reference, if every optimizer had an average probability $\pm 10\%$ from equal, it would produce a deviation of 10^{-3}. This shows that the algorithm selection process is most effective on functions 5, 6, 9 and 10 with deviations over 10%. Functions 4, 7, and 8 also have the smallest deviations. When the meta-optimization has stalled, the probabilities of all of the optimizers remains roughly the same as no optimizer is able to make progress and have its probability increased.

Table 5.6: Meta-Optimization Performance Metrics on Benchmark Suite

Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Restart Rate (%)	16.6	31.4	20.4	90.8	0	0	99.8	61.4	0	0
Average # of Restarts	1.15663	1.35032	1.13725	6.82379	0	0	9.34068	4.04886	0	0
Time Stalled (%)	34.92	41.9	35.31	78.56	0.7059	1.026	66.4	66.73	0	0
Optimizer Count Deviation	6.80e-05	4.89e-05	5.10e-04	2.83e-06	5.20e-03	2.42e-03	1.32e-05	1.76e-05	7.99e-03	6.13e-03
Average Probability Deviation	2.26e-04	4.12e-04	7.14e-04	2.45e-05	5.40e-03	2.92e-03	1.46e-05	4.46e-05	2.42e-03	1.71e-03

Table 5.7: Ratio of Individual Optimizer Contributions to Total Meta-Optimization Cost Reduction (%) on Benchmark Suite

Function	1	2	3	4	5	6	7	8	9	10
SD	14.3	11.4	4.61	8.78	6.71	6.28	7.49	5.95	14.2	14.0
CG	19.9	18.3	4.26	8.75	7.72	6.41	7.49	5.57	15.3	15.3
BFGS	23.4	36.6	4.77	8.27	6.00	7.34	7.94	5.18	14.4	15.8
PSO	12.5	10.5	41.3	27.7	43.3	35.8	26.1	41.3	11.4	12.2
DE	8.23	1.74	29.6	18.6	11.2	25.0	18.4	25.6	11.6	12.8
SIM	7.29	8.85	7.21	8.71	10.2	7.88	11.2	5.98	12.8	12.3
IWO	0.385	0.133	0.331	0.668	0.638	0.495	1.19	0.582	0.762	0.549
TS	7.37	10.5	4.21	8.70	7.12	6.15	10.5	6.00	7.31	6.20
ACO	6.49	1.95	3.67	9.81	7.17	4.67	9.67	3.86	12.2	11.1

The relative performance of each optimizer can also be seen by examining the performance of each optimizer deployed by the meta-optimizer on each problem. Table 5.7 shows the contribution of each optimizer to the total cost reduction for each function. PSO is the most effective on functions 3-8, with DE also performing well on these problems. The hill climbers have the best performance on functions 1, 2, 9, and 10, but struggle on the harder multi-modal problems. SIM is the most effective local search method on functions 3-7 with TS most effective on function 8 and equally effective as SIM on function 4. IWO generally struggles but is able to provide a small amount of cost reduction on every problem. ACO provides some cost reduction on all functions, but lags PSO and DE significantly.

5.3 Meta-Optimization Trade Studies

A series of trade studies were conducted to explore the behavior of the various components of meta-optimization and their effects on performance. The list of parameters investigated are given in Table 5.2. These parameters are broken up into four groups: algorithm selection parameters, efficiency evaluation parameters, reseeding parame-

ters, and restarting parameters. For each case, a nominal configuration was used with only the investigated parameters varied. For the algorithm selection methods, the configuration in Table 5.2 was used as a baseline. For all other cases, the probability update was turned off with each optimizer having equal probability. This was done to help isolate the effects of each parameter from the other components of the framework. Every configuration was tested on the set of functions from Table 5.1 and evaluated on the metrics used in Section 5.2.2. To simplify the results and remove differences between each function, all results except the deviations are normalized based on the performance of the nominal configuration. Full plots of all of the following trade studies are found in Appendix B.

5.3.1 Probability Update Rules

Three different update rules were considered: learning automaton, discrete weighting, and continuous weighting. For the learning automaton, five different values of the reward and penalty parameters a_{LA} and b_{LA} were considered as well as two cases with differing values. The nominal configuration with constant probabilities ($a = b = 0$) and the nominal configuration with sequential selection of optimizers are included as comparisons for the algorithm selection methods. Figure B.1 shows the full results for the learning automaton parameters. Looking first at success rate in Figure B.1a, there is a slightly positive trend of increasing success rate with increasing a_{LA} and b_{LA} on functions 4 and 8 while function 7 shows a slightly negative trend. The constant probabilities case performs the best on function 7, and generally does well on functions 4 and 8. On the other hand, the sequential case performs poorly on function 4, but does achieve better success rate on function 7. It should be noted that due to the overall low success rates on function 7 and the stochastic nature of the meta-optimizer, performance on this function can regularly vary by a few percent between cases, resulting in large differences in relative success rate. In general,

there is little correlation between computation time and these parameters. From Figure 5.7, $a_{LA} = b_{LA} = 0.2, 0.3$ are the most consistent and overall require the fewest restarts. However, Figure 5.8 shows that the higher parameter values remain stalled for less time, especially on functions 5 and 6. On both of these metrics, the constant probabilities case surpasses the nominal on functions 1 and 2, but performs worse on the rest of the functions while the sequential case stalls very little on functions 5 and 6. Looking at the optimizer count and average probability deviations in Figures 5.9 and 5.10, increasing a_{LA} and b_{LA} increases the deviation, indicating more variation in choosing optimizers. When $a_{LA} < b_{LA}$, there is very little deviation while $a_{LA} > b_{LA}$ produces extremely large deviations. No improvement in the other metrics is seen for these configurations.

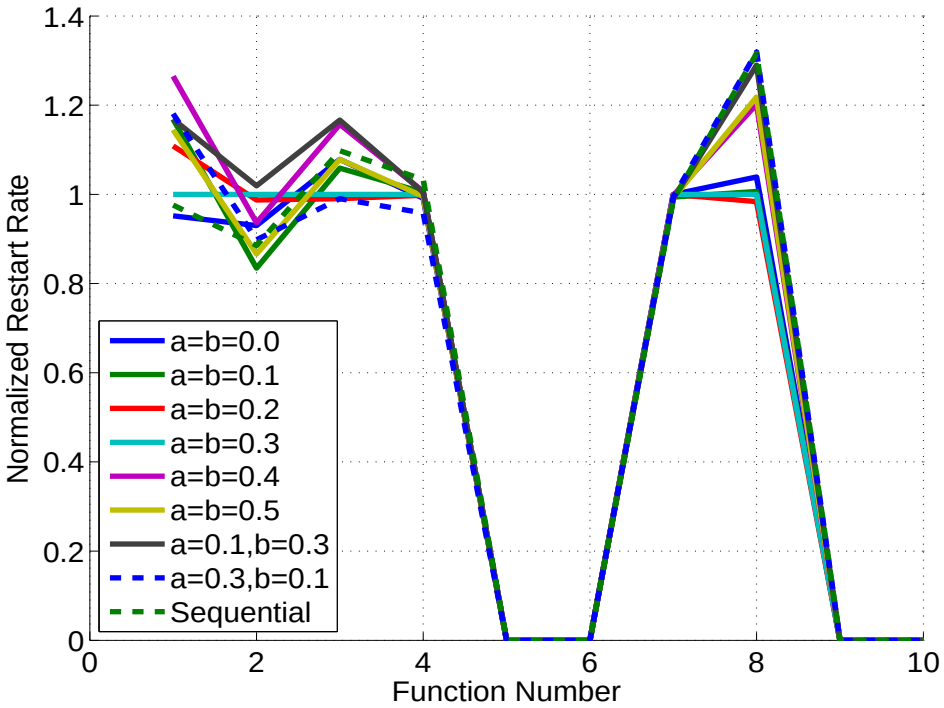


Figure 5.7: Learning Automaton Reward and Penalty Parameter Normalized Restart Rate

For both the discrete and continuous weighting schemes, linear, quadratic, and square root relations were tested with $w_0 = 1$ and $w_1 = 5$. Results for these configurations are given in Figure B.2. On success rate shown in Figure B.2a, the continuous

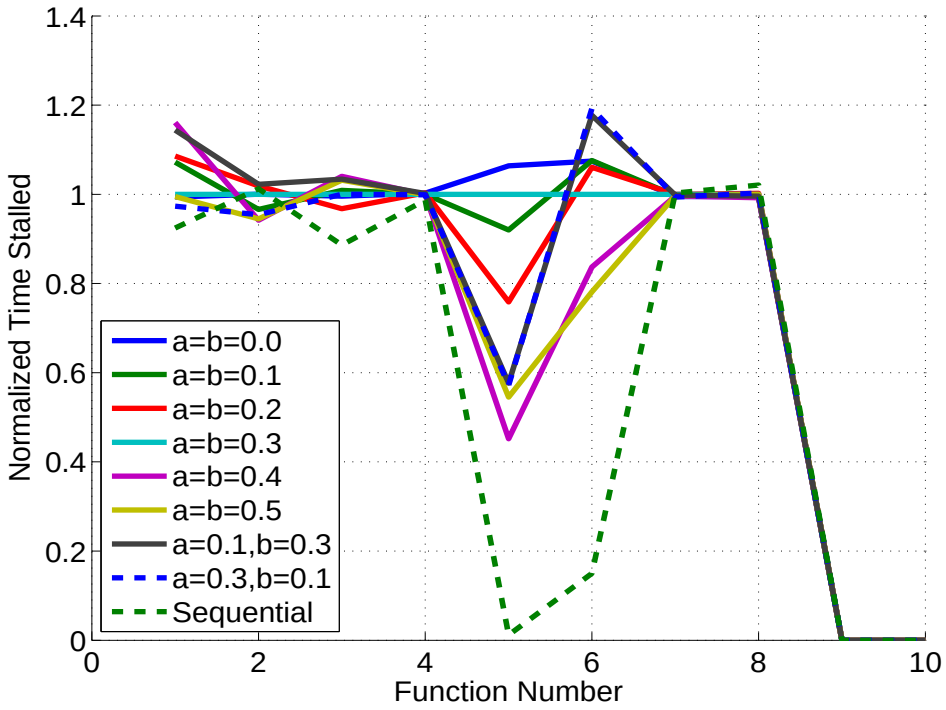


Figure 5.8: Learning Automaton Reward and Penalty Parameter Normalized Time Stalled

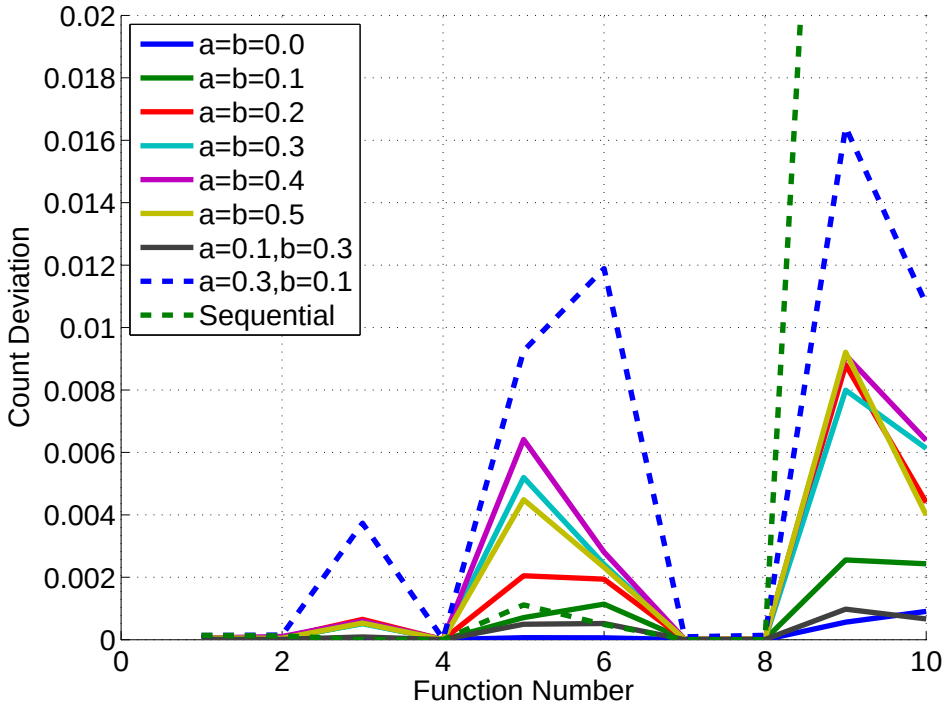


Figure 5.9: Learning Automaton Reward and Penalty Parameter Optimizer Count Deviation

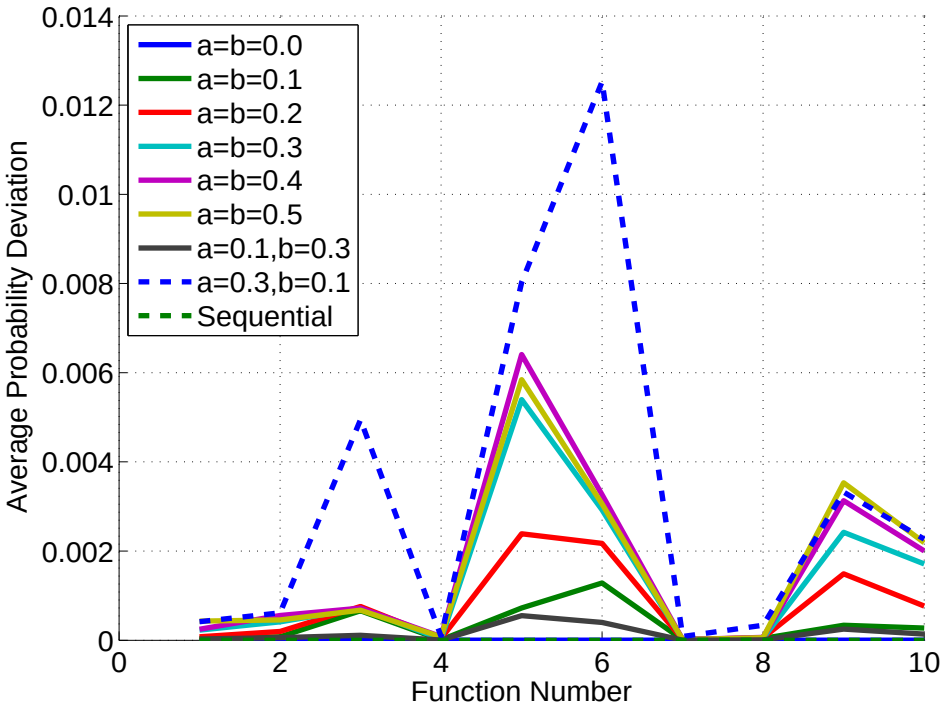


Figure 5.10: Learning Automaton Reward and Penalty Parameter Optimizer Average Probability Deviation

methods performed best on function 4 while the discrete methods performed best on functions 7 and 8. From Figure 5.11, the continuous weighting ran the fastest overall with the square root weighting the fastest, achieving significantly lower computation time on functions 9 and 10. With constant probabilities, computation time was consistent, but overall worse than the other cases. The sequential selection case in particular was very fast on functions 2, 6, and 10, but significantly slower on 4 and 5. On functions such as 2 and 10, the hill climbers are very efficient and will often converge on the solution before another optimizer is used. However, on multimodal functions like functions 4 and 5, the meta-optimizer must cycle through all of the optimizers, even if only a few are able to escape a local minima, while the hill climbers continue moving towards the local minima every time they are used. With sequential selection, meta-optimization is guaranteed to select a hill climber within a few iterations. Figure B.2c shows that the continuous weighting was also best on restarts on all but function 8, while the discrete weighting was better on time stalled

as seen in Figure 5.12. For the deviations shown in Figures 5.13 and 5.14, the continuous, square root weighting is the most consistent with significantly more deviation on functions 1-4, 7, and 8. On the other functions, it is on par with the deviation from the learning automaton with $a_{LA} = b_{LA} \geq 0.3$.

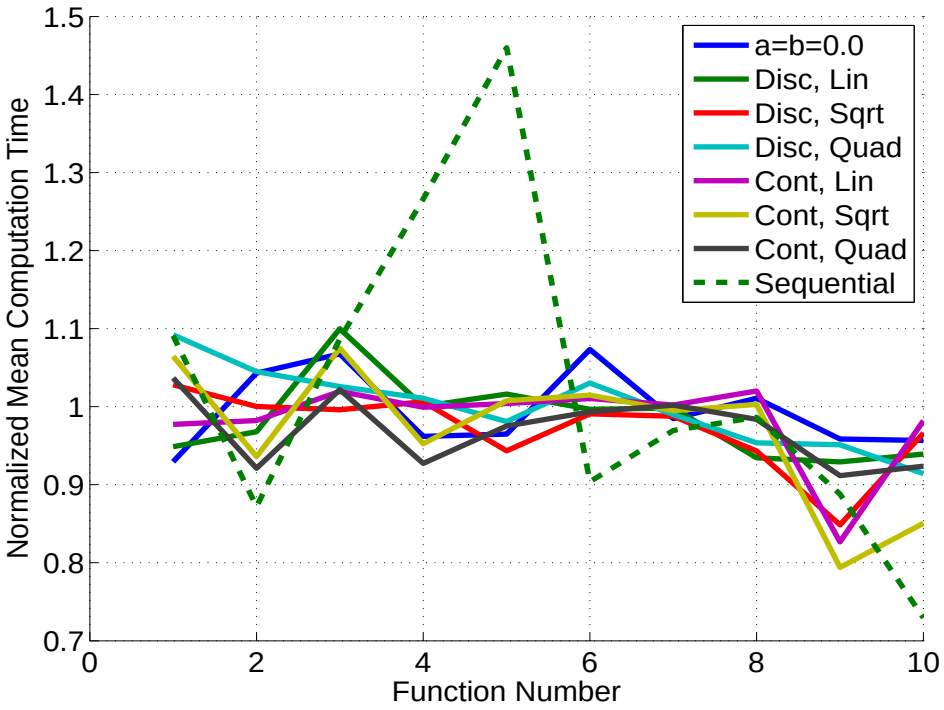


Figure 5.11: Probability Weighting Methods Normalized Mean Computation Time

Compared to the case with constant probabilities, the algorithm selection methods did not significantly alter the success rate on any of the functions. However, the most effective selection configurations achieved significant computation time savings on some functions. In terms of restart rate and time stalled, the probability updates may reduce performance on functions 1 and 2. This could be caused by two factors. First, on some trials, the meta-optimizer may select optimizers such as PSO and DE which are effective on this function, but not the most efficient. In this case, the probabilities for these optimizers may be boosted, lowering the chance of selecting the significantly more efficient hill climbers. In the second scenario, if a hill climber begins outside of the basin of attraction of the global minimum, it will be attracted

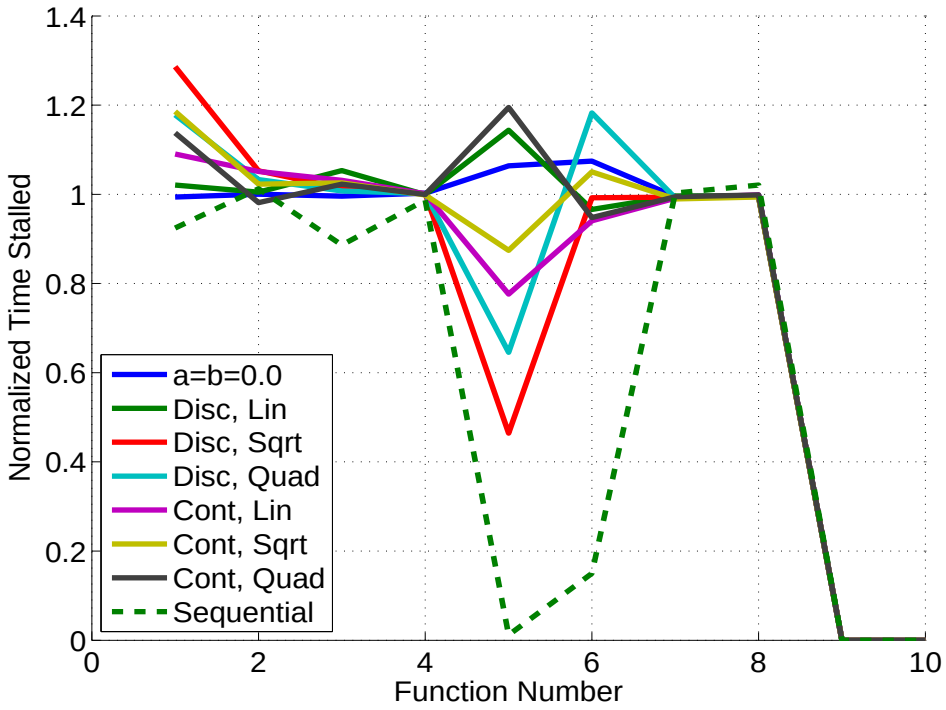


Figure 5.12: Probability Weighting Methods Normalized Time Stalled

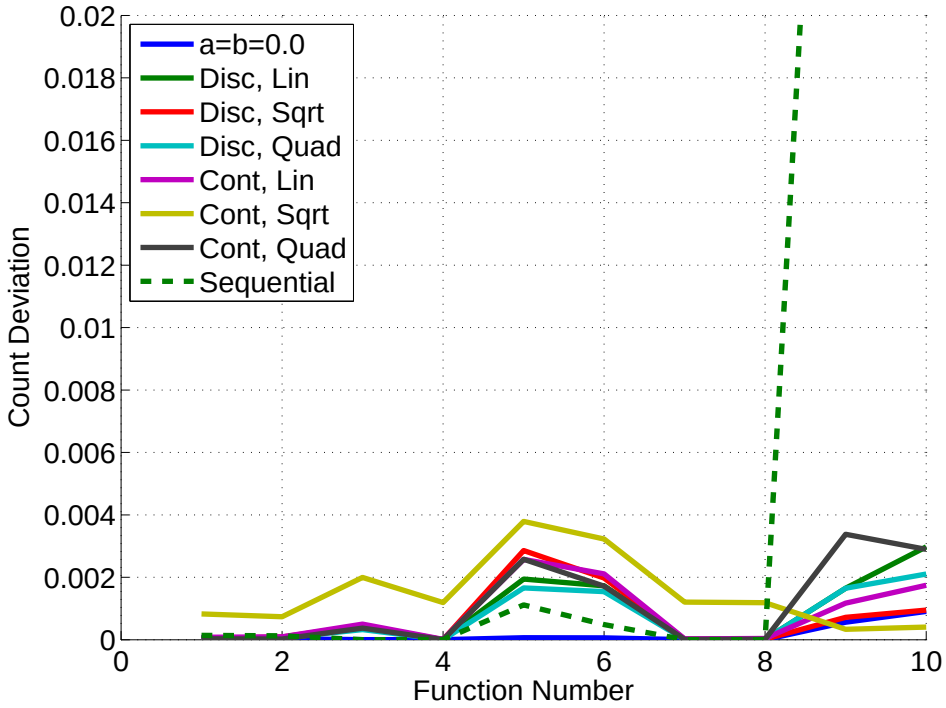


Figure 5.13: Probability Weighting Methods Optimizer Count Deviation

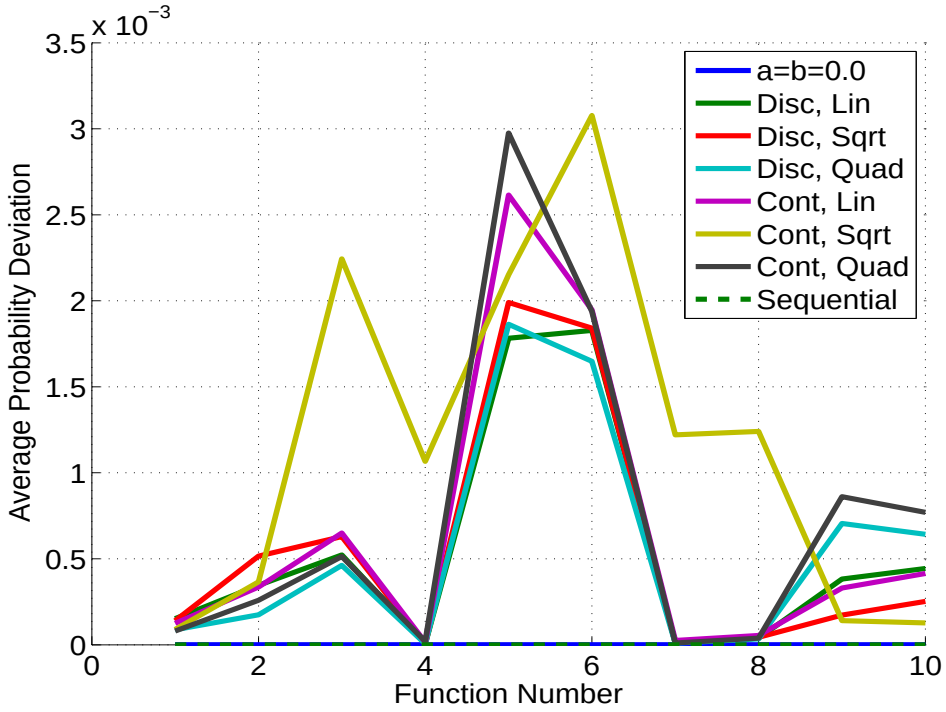


Figure 5.14: Probability Weighting Methods Optimizer Average Probability Deviation

towards the local minimum of this function. Since the hill climbers are efficient on this function, they will achieve high efficiencies and have their probabilities increased. This lowers the likelihood of selecting a global method that could potentially move away from the local minimum before the hill climber reaches the local minimum and the meta-optimizer stalls. On the other functions where there are more local minima for the hill climbers to find, the opposite trend occurs where the probability update rules lower the probabilities of the hill climbers, reducing the likelihood of reaching a local minimum. Overall, these algorithm selection methods show superior performance over random selection with constant probability and sequential selection.

The final trade study on the probability update rules is an investigation on the effect of the weighting function coefficients W_0 and W_1. To test the impact of these parameters, the continuous, square root weighting function was used as the baseline. Instead of varying W_0 and W_1 independently, a test matrix was constructed by varying W_0 and $W_1 - W_0$. W_0 has two primary effects on the probabilities for each

optimizer. Increasing W_0 will decrease the maximum possible probability and increase the minimum probability, reducing the potential range of probabilities. On the other hand, increasing $W_1 - W_0$ increases the maximum probability and decreases the minimum probability, increasing the potential range of probabilities. The net effect is an increase in the range of probabilities for all W_0 values. The values investigated are $W_0 = \{0.5, 1.0, 2.0, 3.0\}$ and $W_1 - W_0 = \{3.0, 4.0, 5.0, 6.0\}$.

Figures B.3-B.6 show some of the results for this trade study. Not shown are success rate and computation time which did not show any discernible trends in either parameter. However, the combination of $W_0 = 1.0$ and $W_1 - W_0 = 5.0$ performed slightly better on both success rate and computation time. In terms of restart rate shown in Figure B.3, lower values of W_0 required slightly fewer restarts, especially on function 3. Medium values of $W_1 - W_0$ also required slightly fewer restarts overall. Figure B.4 indicated that lower W_0 values also stalled less overall with better performance on functions 5 and 6. The largest trends occur in the deviations as seen in Figures 5.15 and B.6. For both metrics, the deviation increases with increasing $W_1 - W_0$ and decreases with increasing W_0. This conforms with the nature of the weighting functions where a larger range in probabilities will cause meta-optimization to favor certain optimizers throughout a run, increasing the deviation from equal usage.

The primary effect of the algorithm selection methods is on the frequency of selecting each optimizer. The large deviations in both the number of times each optimizer is called and the average probability for each optimizer demonstrate how the probability update rules favor certain optimizers over others on a given run. However, large deviations are not necessarily beneficial on all functions. For example, cases with higher deviations tended to perform worse on functions 1, 2, 9, and 10 and slightly better on functions 5 and 6. To preserve reliability over a wide range of problems, algorithm selection methods which produce moderate levels of deviation should be preferred.

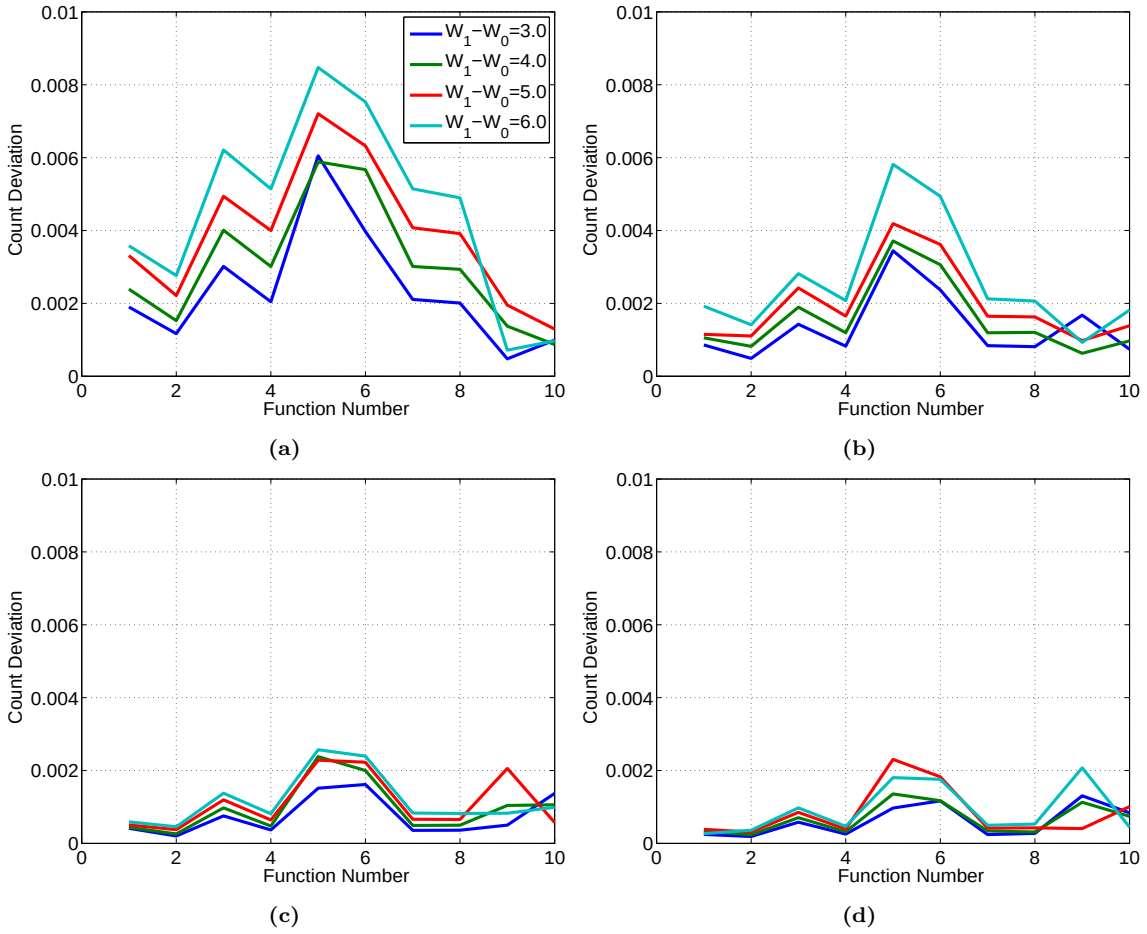


Figure 5.15: Weighting Coefficient Optimizer Count Deviation (a). $W_0 = 0.5$**; (b).** $W_0 = 1.0$**; (c).** $W_0 = 2.0$**; (d).** $W_0 = 3.0$

5.3.2 Optimizer Efficiency Evaluation

The efficiency check is characterized by two primary parameters: the check window N_w and the reference cost reduction J_{ref}. To observe the effect of these parameters, a test matrix was constructed with check window sizes $N_w = \{1000, 2000, 3000, 4000, 5000, 6000\}$ function calls and reference cost reduction per 1000 function calls $J_{ref} = \{0.5, 1.0, 1.5, 2.0\}$. The check window not only controls the range over which the optimizers are evaluated, but also the minimum number of function calls allotted each optimizer and the frequency of efficiency checks. A longer window allows for a more accurate evaluation of optimizer performance, but it also slows down the frequency of exchanges between optimizers. The reference cost reduction determines what is considered sa-

tisfactory performance for the optimizers with higher values grading each optimizer more harshly. J_{ref} is specified here in terms of cost reduction per 1000 functions calls such that the reference cost reduction per function call is the same across all window sizes.

The results for this trade study are shown in Figures B.7-B.10. Figure 5.16 shows a serious degradation in performance with a shorter window at $J_{ref} = 0.5$ on function 4. This is primarily due to the meta-optimizer switching off of optimizers too quickly and not examining a long enough window to properly evaluate performance. There is no discernible trend in success rate for J_{ref} seen in Figure B.7. The longer window also does better on function 7, but performs slightly worse on function 8. Clear trends in window size in terms of computation time can be seen at J_{ref} in Figure 5.17. For functions 1, 2, 5, 6, 8, 9, and 10, computation time generally increases as the window increases while the opposite is true on function 3 and 4. Figure 5.18 shows computation time generally decreasing with increasing J_{ref}. With a higher reference performance, meta-optimization will switch off of poorly performing optimizers faster, reducing the amount of computation time used on these optimizers. Restarting rate as seen in Figure B.9 also generally decreases with increasing window with no trend in J_{ref}. Figure B.10 shows that longer windows stall the least on all but functions 5 and 6 where medium size windows perform best. On functions 1, 5, and 6, there is a slightly positive trend of increasing time stalled with increasing J_{ref}, however the trend does not hold for other functions. Overall, shorter and longer windows may provide benefits on certain functions, but overall have a larger variance in performance. Intermediate sized windows provide the most consistent performance. A higher reference cost reduction is also beneficial, particularly on reducing computation time.

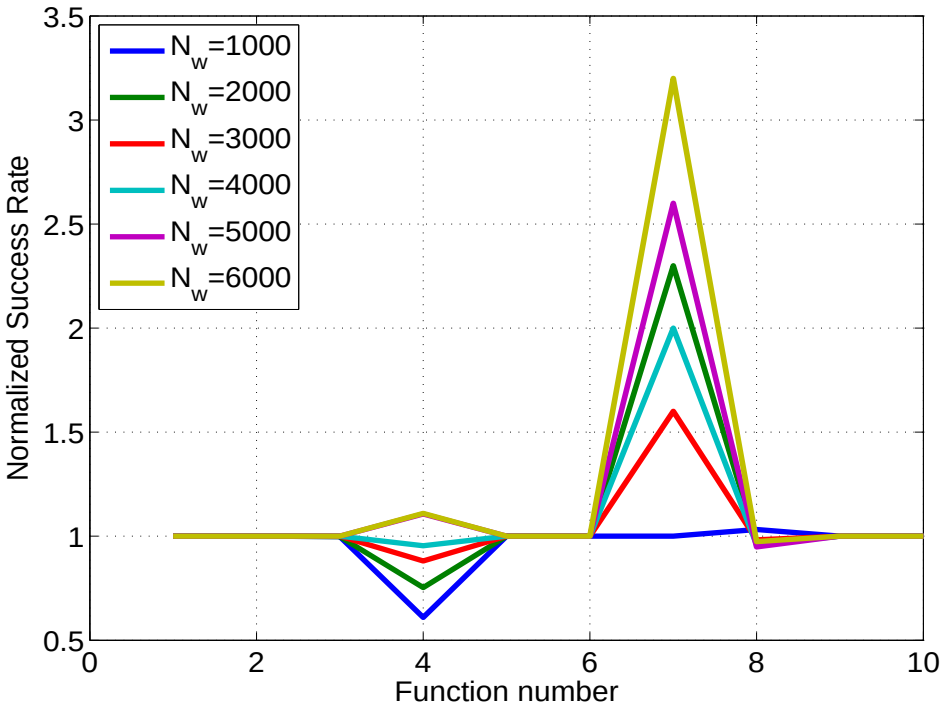


Figure 5.16: Efficiency Check Window at $J_{ref} = 0.5$ Normalized Success Rate

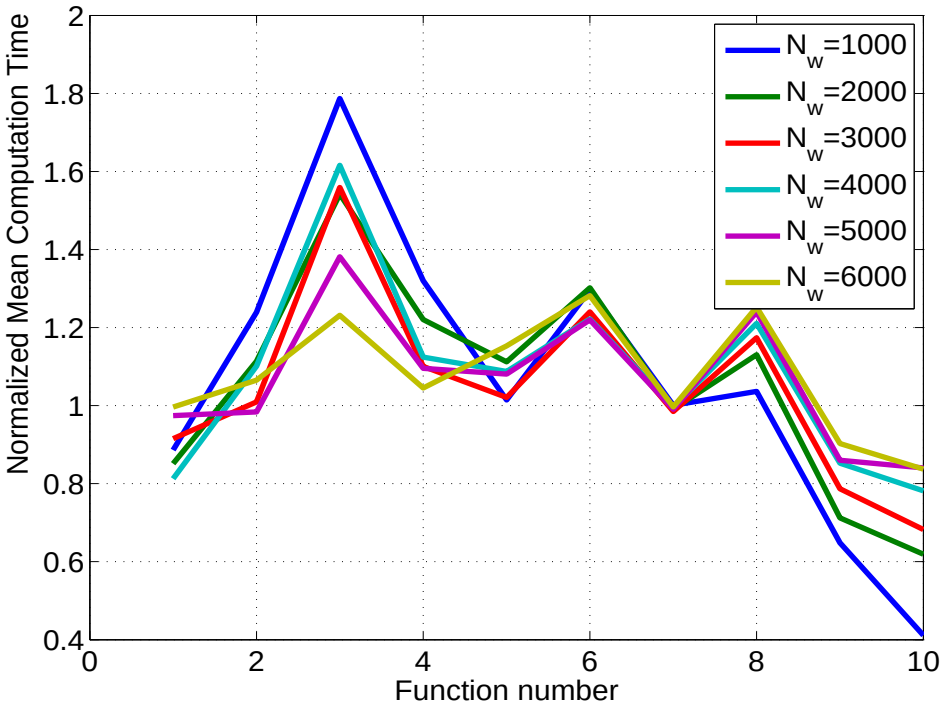


Figure 5.17: Efficiency Check Window at $J_{ref} = 1.5$ Normalized Mean Computation Time

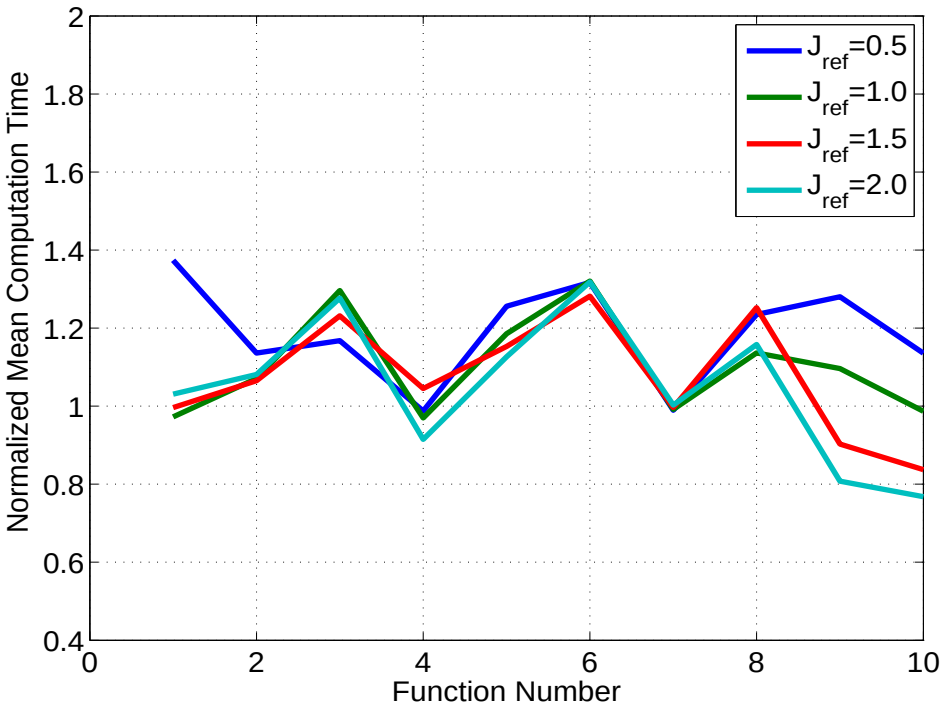


Figure 5.18: Reference Cost Reduction at $N_w = 6000$ Normalized Mean Computation Time

5.3.3 Reseeding Parameters

The three parameters that govern reseeding are the reseeding rate RS, the reseeding refinement rate r_{RS}, and the diversity threshold $\sigma_{d,min}$. The first parameter investigated is the reseeding rate RS which controls the percentage of the population for a given optimizer that will be reseeded. Ten values were considered ranging from 0.0 to 0.9. Figure B.11 shows the results for this trade study. Looking at Figure 5.19, there is a clear and significant reduction in success rate for smaller values of RS, especially on functions 3, 4, and 8. The opposite trend exists on function 7, but there is significant variation in performance on this function. From Figure 5.20, computation time also generally decreases with increasing RS until $RS = 0.8$ after which computation time increases again. Similar trends also exist for restart rate and time stalled as seen in Figures B.11c and B.11d respectively, with $RS = 0.8$ performing best on every measure except success rate. This behavior indicates some diminishing returns

in increasing RS to the point that too high of RS actually degrades performance. In these cases, very little of the population is retained, effectively restarting the optimizer from an entirely new population. This population may be so diverse that the optimizers expend most of their computation time on exploration with little ability for exploitation. Computation time and restart rate on function 3 and time stalled on function 6 are also very sensitive to RS, with very large increases for smaller RS. This demonstrates the importance of reseeding in allowing the global optimizers to free themselves from local minima in a timely manner. Without reseeding, there is a significant degradation in performance on every function with the meta-optimizer unable to solve functions 3 and 8 at all, demonstrating how critical reseeding is to the capabilities of meta-optimization.

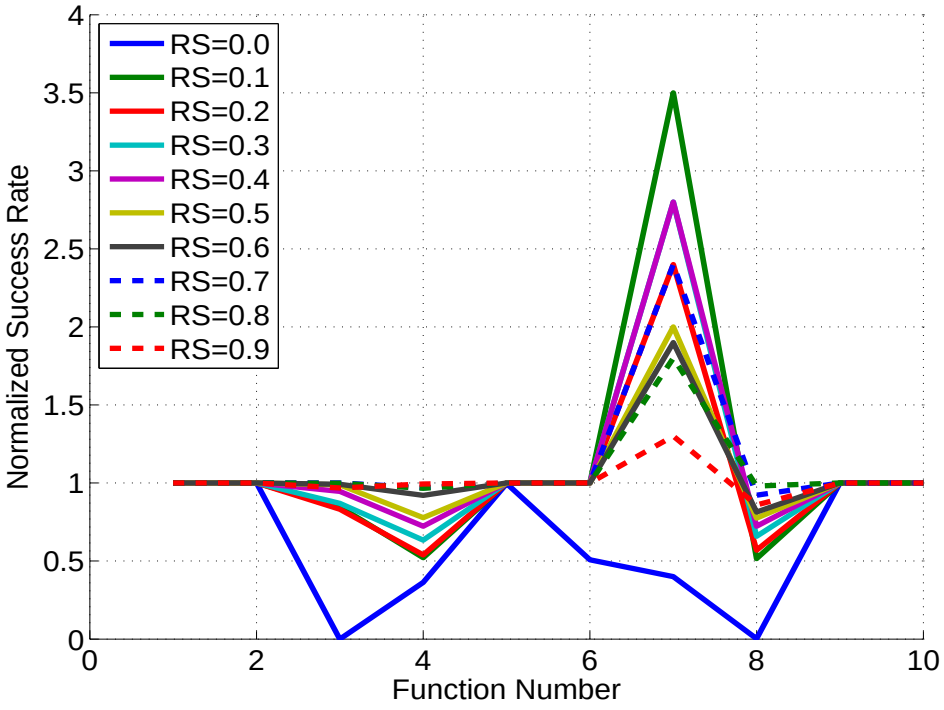


Figure 5.19: Reseeding Rate Normalized Success Rate

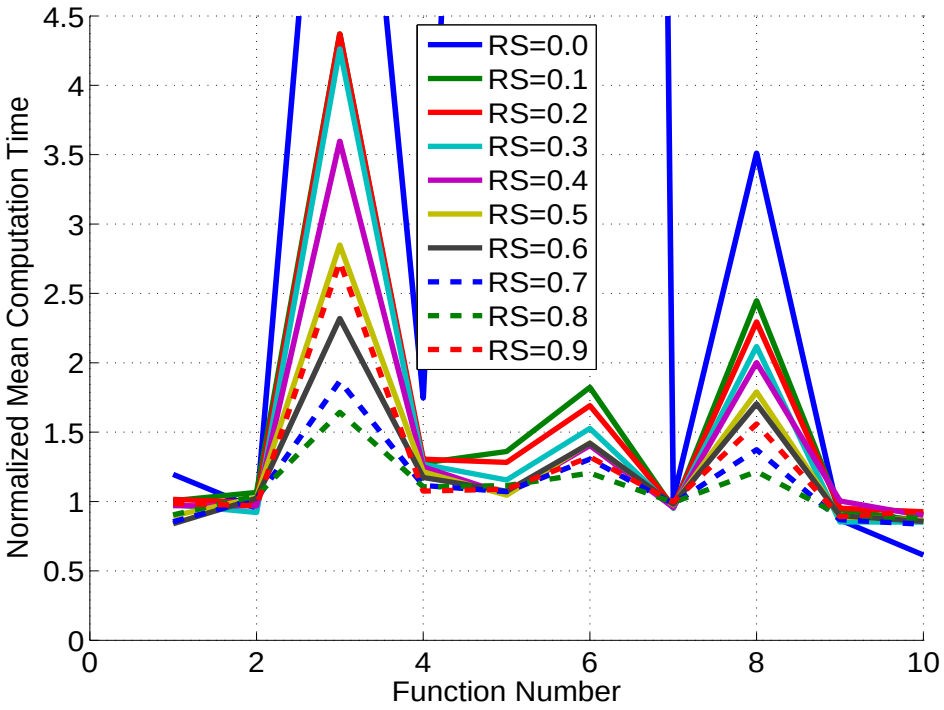


Figure 5.20: Reseeding Rate Normalized Mean Computation Time

The reseeding refinement rate r_{RS} controls the frequency of refinement in the reseeding process as explained in Section 4.4. This parameter controls the balance between exploration and exploitation when new points are generated, with a lower value corresponding to greater exploration. Nine values from 0.1 to 0.9 were considered for this trade study with the results shown in Figure B.12. Figure 5.21 shows a strong trend of decreasing performance with increasing r_{RS} on function 4, while medium values tend to do best on function 7. Medium values as seen in Figure B.12b also run the fastest overall with lower values performing best on all but functions 1 and 2. There is little trend in restarting rate seen in Figure B.12c, however medium values tend to restart slightly less often. There is also a slightly positive trend in time stalled in Figure B.12d with $r_{RS} = 0.2$ doing well on all but function 6. Based on these results, meta-optimization performs best with more exploration than exploitation. By seeding more points throughout the parameter space, the optimizers are provided a more diverse population, fueling their exploration capabilities. Also, the

longer the meta-optimizer remains near a local minimum, the more the KDF for exploitation will shift towards the local minimum, causing more reseeded points to fall in that region. If too much of the population is in the vicinity of a local minimum, the optimizers will quickly collapse on the local minimum and the meta-optimizer will remain stalled.

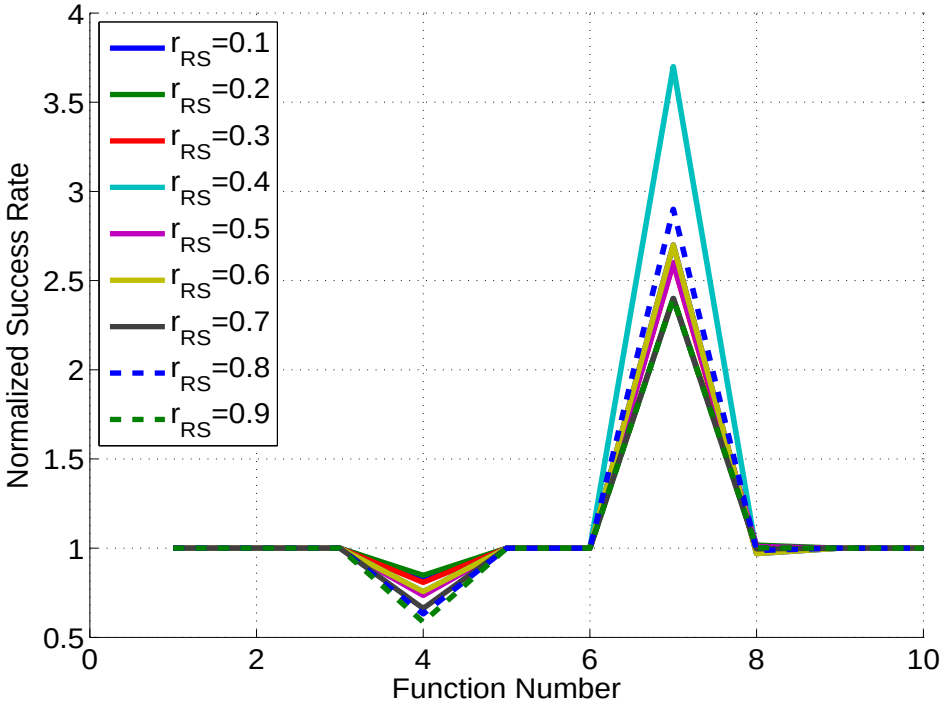


Figure 5.21: Reseeding Refinement Rate Normalized Success Rate

The final parameter that controls reseeding is the diversity limit $\sigma_{d,min}$ used to trigger reseeding. A higher threshold will cause reseeding to be performed more often while a lower threshold will make reseeding less likely. The diversity threshold is also used to control when the population based optimizers have collapsed and should stop running. The threshold was varied from 5×10^{-4} to 5×10^{-8} with results shown in Figure B.13. From Figure 5.22, there is a noticeable decrease in performance on function 7 and a small decrease on function 4 for lower diversity limits. In Figure 5.23, computation time increased significantly for the lower diversity limits. In addition, Figures B.13c and B.13d show similar trends for restarting rate and time stalled

respectively. With the lower threshold to trigger reseeding, the meta-optimizer is not reseeding often enough for the global optimizers to be able to escape local minima, leading to a reduction in performance. On the other hand, the highest diversity threshold saw a significant increase in computation time on function 3. This is likely due to the meta-optimizer stopping the global methods too soon, not allowing them to run long enough to search the parameter space and avoid local minima. The frequent reseeding can also reduce performance by resetting the population before the optimizers have had a chance to cooperate on exploring the search space. Overall, medium values of the diversity threshold performed best with a value of 5×10^{-6} showing good performance across all functions.

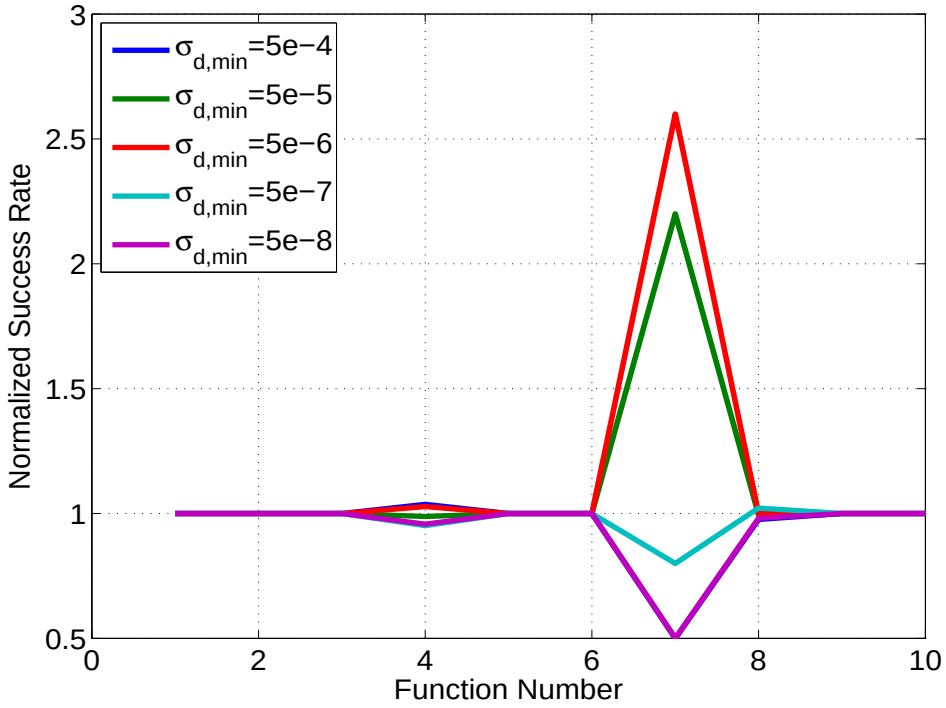


Figure 5.22: Reseeding Diversity Threshold Normalized Success Rate

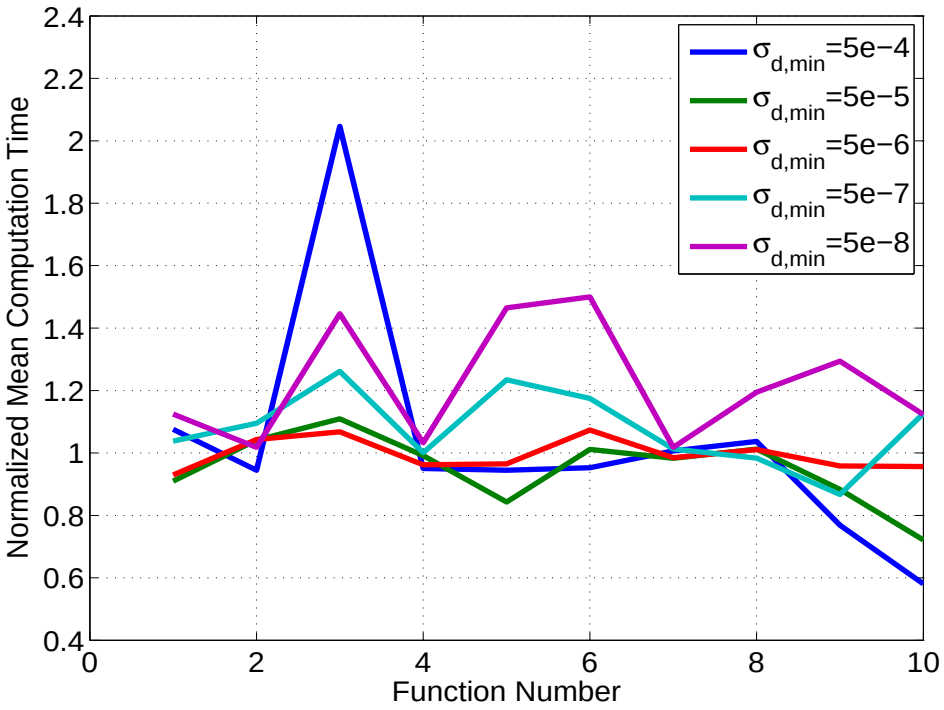


Figure 5.23: Reseeding Diversity Threshold Mean Computation Time

5.3.4 Restarting Parameters

The frequency of restarts can play a significant role in the performance of meta-optimization. The restarting threshold N_R controls the number of function calls the meta-optimizer is allowed to expend while stalled in a local minimum. While the global optimizers may be able to get out of a local minimum if given enough time, it may be more efficient to start over, maintaining knowledge of the location of previously reached local minima. Six threshold values were evaluated ranging from 50,000 to 500,000 function calls as well as a case with no restarts. The limit on total number of restarts is removed to allow reasonable comparisons between each of these cases as a lower restarting threshold will lead to many more restarts within the computation time limit.

Figures 5.24-5.26 show the results for this trade study. Note that time stalled is not evaluated for this trade study as the amount of time stalled is directly related to the restarting threshold with a larger threshold allowing meta-optimization to remain

stalled for a longer portion of a run. Looking at Figure 5.24, lowering the threshold can both help and hinder meta-optimization. While the lower restarting threshold improved performance on function 4, there were significant reductions in performance on functions 3 and 8, functions meta-optimization does not typically struggle with. Figure 5.25 shows a very large increase in computation time on these two functions as well. A lower threshold did achieve some computation time reductions on functions 1 and 2, but performed worse overall. As expected, the rate of restarts seen in Figure 5.26 increased significantly as the threshold was reduced. Without restarts, the meta-optimizer was still able to solve functions 1, 3, 5, 6, 9, and 10 every time, but struggled on functions 2, 4, and 8. This indicates that on some problems, the meta-optimizer is able to free itself from a local minima if given enough time. However, as Figure 5.25 shows, the meta-optimizer without restarts required twice as many function calls to solve function 1 as the nominal configuration, and on no function did the absence of restarts show improved performance.

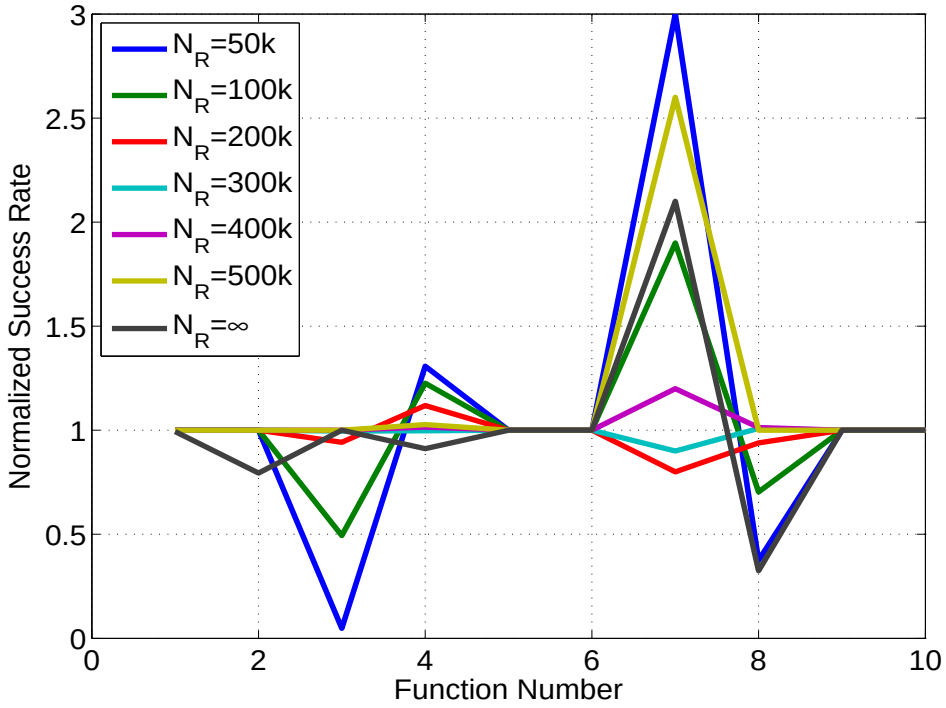


Figure 5.24: Restarting Threshold Normalized Success Rate

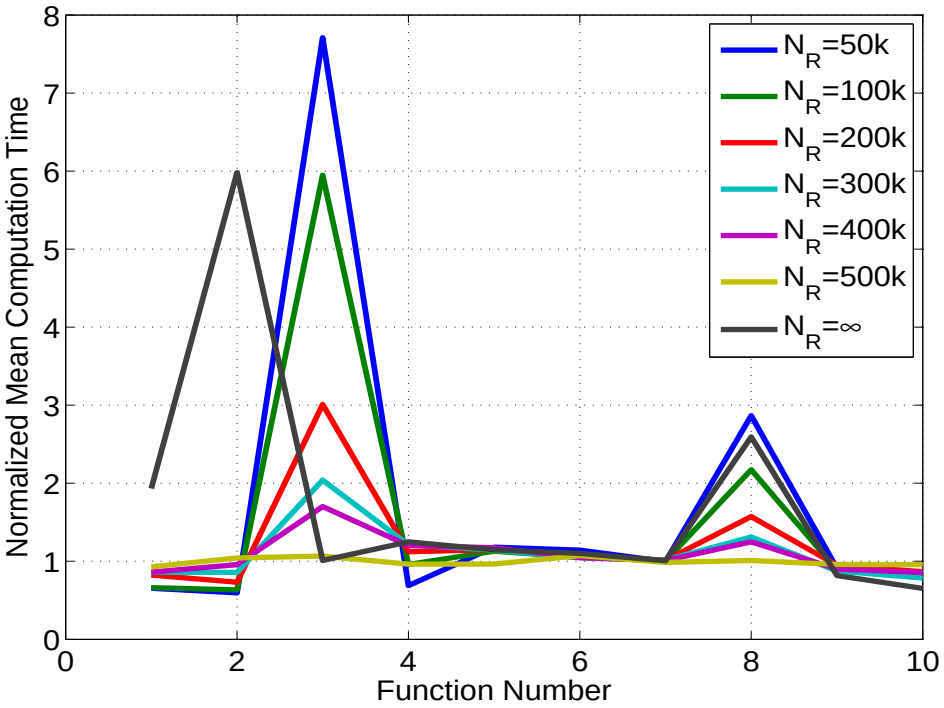


Figure 5.25: Restarting Threshold Mean Computation Time

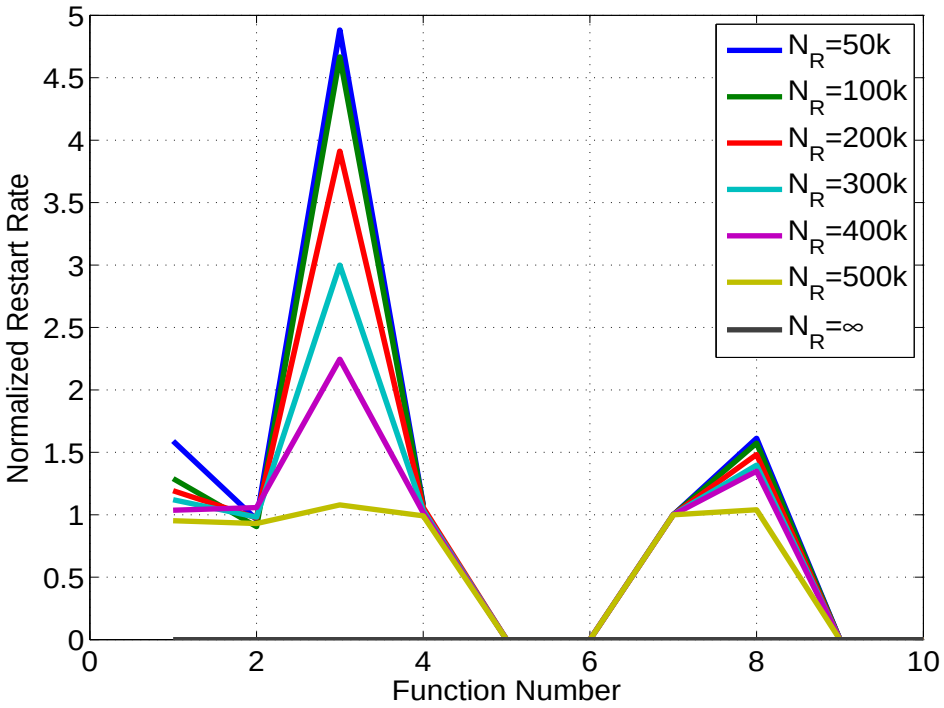


Figure 5.26: Restarting Threshold Mean Restart Rate

The second component of the restarting mechanism is the exclusion zone. When a restart is performed, no new points are generated within an area around all previous local minima. This zone can be either a ball around the local minimum or a range of values in each parameter. Both exclusion zone types were considered with radius $r_R = \{1, 2, 3\}$ with results in Figure B.15. Figure 5.27 shows a large increase in success rate on function 4 for the range type zone with slightly worse performance on function 7. As seen in Figure 5.28, computation time is greatly reduced on function 4 using the range type zone, but the opposite holds true on function 8. Both of these effects are more pronounced as the radius of the exclusion zone increases. No discernible trend in either exclusion zone type or radius is observed for restart rate and time stalled in Figures B.15c and B.15d respectively. While the exclusion zone radius performed well overall, it should be noted that in some cases, a large radius exclusion zone can cover all or most of the search space after a number of restarts, making it difficult to generate new points. Therefore, it is preferable to limit the size of the exclusion zone. For the range type zone, a lower radius provides a good balance of performance while maintaining the large improvements on function 4.

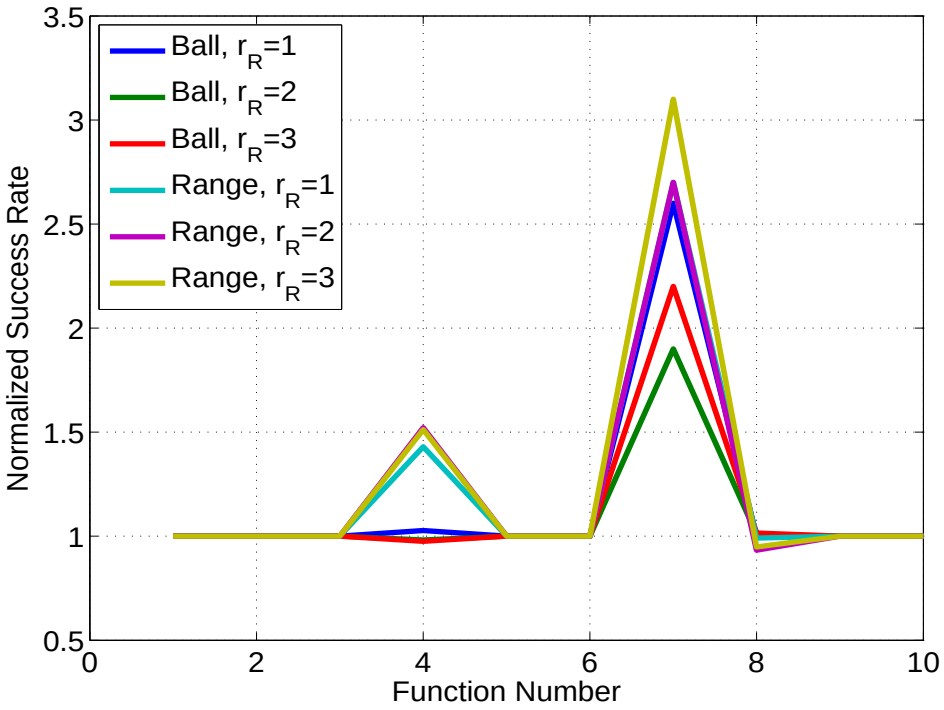


Figure 5.27: Restarting Exclusion Zone Normalized Success Rate

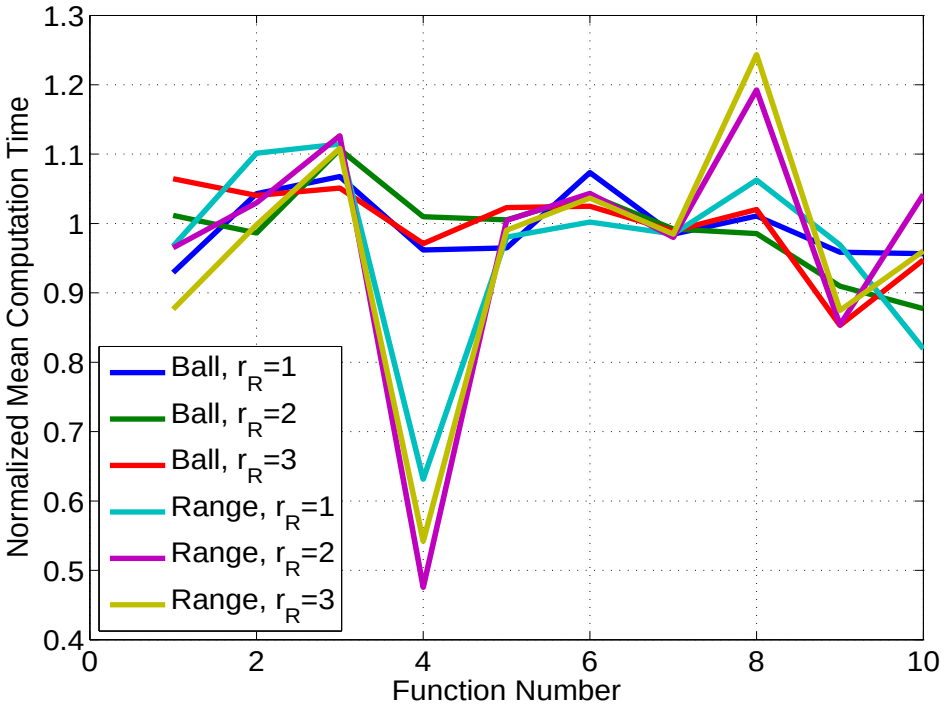


Figure 5.28: Restarting Exclusion Zone Mean Computation Time

5.3.5 Final Meta-Optimization Configuration

Based on the results from each of these trade studies, the meta-optimization configuration in Table 5.8 was constructed to maximize the performance of meta-optimization as a general use tool on wide range of problems. First, for the probability update rules for the algorithm selection, a continuous, square root weighting function was chosen given its good computation time performance and high deviations. Analysis of the weighting coefficients W_0 and $W_1 - W_0$ indicated the best results for medium values of both, with the combination $W_0 = 1$ and $W_1 = 5$ performing well. For the efficiency window and reference cost reduction, the combination of a medium window of 4000 function calls and a high J_{ref} of 2% balances the benefits of a longer window on some functions and a shorter window on others. The reseeding rate of 0.8 was the most effective in almost all cases with some reduction in performance when the reseeding rate was increased. Lower reseeding refinement rates, on the other hand, showed promising results, indicating a preference for exploration over exploitation in reseeding. A medium value of the diversity threshold also performed well overall with both lower and higher thresholds seeing worse results on some problems. The long restarting threshold of 500,000 function calls was maintained to allow the optimizers time to escape local minima without restarting too quickly. Finally, the exclusion range provided a significant improvement in success rate on function 4 with only a minimal reduction on function 8.

Table 5.8: Final Meta-Optimization Configuration

Parameter	Name	Value
	Algorithm Selection Method	Continuous
	Weighting Function	Square Root
W_0	Weighting Function Constant	1.0
W_1	Weighting Function Slope	5.0
N_w	Efficiency Check Window	4000
J_{ref}	Reference % Cost Reduction	8.0%
RS	Reseeding Rate	0.8
r_{RS}	Reseeding Refinement Probability	0.2
$\sigma_{d,min}$	Reseeding Diversity Threshold	$5e^{-6}$
N_R	Restarting Threshold	500,000
	Restarting Exclusion Zone	Range
r_R	Exclusion Zone Radius	1.0

The performance of this configuration is given in Table 5.9 with the nominal results included as a comparison. On function 4, the final configuration is able to solve the problem almost 50% more often than the nominal configuration. The small increase on function 6 and small decrease on function 8 are minor differences and are effectively equal in performance. Across all of the functions, computation time is similar for both configurations with variations of about 5-10%. The largest changes are on function 7 where the final configuration is 25% slower than the nominal, but is 30-40% faster on functions 9 and 10. Restarting rate and time stalled are also similar across all functions with the final configuration restarting more often on function 8 and stalling less often on function 5. Overall, the final configuration matches the performance of the nominal configuration with large improvements on a few functions.

Table 5.9: Comparison of Nominal and Final Configurations on Benchmark Suite

	Success Rate (%)									
Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nominal	100	100	100	65.4	100	100	2	92.4	100	100
Final	100	100	100	91	100	100	6	91.4	100	100
	Mean Computation Time									
Nonimal	3.1e+05	4.6e+05	1.3e+06	3.3e+06	1.2e+05	1.3e+05	3.9e+06	2.5e+06	6.3e+04	1.4e+05
Final	2.8e+05	4.8e+05	1.5e+06	3.5e+06	1.3e+05	1.4e+05	5.1e+06	2.6e+06	4.4e+04	7.4e+04
	Restart Rate (%)									
Nominal	16.6	31.4	20.4	90.8	0	0	99.8	61.4	0	0
Final	19.6	29.2	25	90.8	0	0	99.2	79	0	0
	Time Stalled (%)									
Nominal	34.92	41.9	35.31	78.56	0.7059	1.026	66.4	66.73	0	0
Final	44.1	41	37.69	77.96	0.2362	1.701	65.38	65.41	0	0

5.4 Competition Suite Testing

In addition to the ten benchmark functions in Section 5.1, meta-optimization was evaluated on the CEC2014 benchmark function competition suite [135]. Using this set of functions, meta-optimization can be directly compared with other state of the art optimizers which participated in this competition. The competition suite consists of 30 different functions with 10, 30, 50, and 100 dimensions. For each function, competition rules permit the optimizer to run 51 times with a budget of $n \times 10,000$ function calls. Three optimizers from the competition were chosen as a comparison for meta-optimization: linear success-history based adaptive differential evolution (L-SHADE) [137], replacement strategy differential evolution (RSDE) [138], and united multi-operator evolutionary algorithms (UMOEAs) [139]. These optimizers are briefly described below.

Linear Success-History based Adaptive Differential Evolution (L-SHADE): Developed by Tanabe and Fukunaga, L-SHADE builds off of the existing SHADE algorithm [140], adding adaption of the population size to the method. SHADE adjusts the DE tuning parameters F and CR using a memory of previous values of the parameters which were successful at producing better solutions. The addition of a linearly decreasing population allows the algorithm to shift from exploration with a large initial population to more efficient exploitation when the population is small [137].

Replacement Strategy Differential Evolution (RSDE): For many optimization problems, DE can expend a large number of functions calls freeing individuals in the population from a local minima. To overcome this issue, Xu, Huang, and Ye add two replacement strategies to DE to improve efficiency and reliability. The first strategy replaces individuals which fail to show improvement with a new position close to the best point found so far, improving the exploitative ability of the algorithm. The

second strategy redistributes the entire population when it has collapsed on a local minima [138].

United Multi-Operator Evolutionary Algorithms (UMOEAs): UMOEAs is an adaptive approach for combining multiple evolutionary algorithms into a single method. Three different multi-operator evolutionary algorithms are used to evolve a subpopulation for a number of generations. Each algorithms contains additional logic for allocating resources between each operator. Next, the most effective algorithm is allowed to evolve its subpopulation for a subsequent number of populations. At the completion of each cycle, the worst subpopulation is updated using information from the best subpopulation [139].

The final configuration of meta-optimization given in Table 5.8 was used for this test. It should be noted that this comparison is being made even though meta-optimization was designed with the goal of general reliability and robustness while the other optimizers were tuned specifically for maximum speed and performance on this set of problems. Detailed results for meta-optimization and these optimizers are given in Tables 5.10-5.13. The results presented here for the three optimizers L-SHADE, RSDE, and UMOEAs were originally reported by the authors of each algorithm and were not obtained from independently implementing and evaluating these algorithms.

Overall, meta-optimization was comparable to these state of the art optimizers on many of the competition suite functions. Meta-optimization had some difficulty on the unimodal functions (1, 2, and 3) for $n = 10$. At this low dimension, there is a limited computational budget which does not allow meta-optimization sufficient time to select the hill climbers which are most efficient on these types of problems. The other optimizers all incorporate local search strategies in the algorithm, allowing them to perform well on these problems. On the multimodal functions, the meta-optimizer achieved similar performance to the other optimizers on many functions.

Table 5.10: Optimizer Final Cost Error Mean and Standard Deviation n = 10

	Meta-Opt		LSHADE		RSDE		UMOEAs	
Fn	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
1	3.37e+03	1.24e+04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
2	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
3	1.27e+02	6.86e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
4	6.28e+00	1.52e+00	2.90e+01	1.30e+01	2.81e+00	8.24e+00	0.00e+00	0.00e+00
5	7.03e-01	2.86e-01	1.40e+01	8.80e+00	1.92e+01	3.92e+00	1.69e+01	7.36e+00
6	9.50e-01	7.22e-01	1.80e-02	1.30e-01	5.29e-02	2.13e-01	0.00e+00	0.00e+00
7	1.14e-01	7.64e-02	3.00e-03	6.50e-03	3.55e-02	3.12e-02	0.00e+00	0.00e+00
8	1.48e+00	1.09e+00	0.00e+00	0.00e+00	6.61e-01	9.31e-01	0.00e+00	0.00e+00
9	1.31e+01	5.75e+00	2.30e+00	8.40e-01	8.52e+00	3.71e+00	4.65e+00	1.94e+00
10	7.04e+01	8.01e+01	8.60e-03	2.20e-02	6.84e+01	6.65e+01	6.34e-01	1.14e+00
11	4.95e+02	1.98e+02	3.20e+01	3.80e+01	2.91e+02	1.93e+02	1.59e+02	1.64e+02
12	9.92e+00	3.19e+01	6.80e-02	1.90e-02	2.21e-01	1.37e-01	8.89e-04	3.59e-03
13	1.56e-01	9.66e-02	5.20e-02	1.50e-02	1.28e-01	3.18e-02	9.46e-03	5.05e-03
14	2.26e-01	9.71e-02	8.10e-02	2.60e-02	1.36e-01	4.36e-02	8.34e-02	3.33e-02
15	1.12e+00	5.79e-01	3.70e-01	6.90e-02	9.83e-01	3.70e-01	6.56e-01	2.00e-01
16	2.51e+00	4.96e-01	1.20e+00	3.00e-01	2.23e+00	4.32e-01	1.55e+00	6.47e-01
17	4.99e+02	1.81e+03	9.80e-01	1.10e+00	4.77e+01	5.52e+01	9.90e+00	1.66e+01
18	1.58e+01	2.27e+01	2.40e-01	3.10e-01	2.00e+00	1.10e+00	9.95e-01	9.54e-01
19	8.25e-01	5.08e-01	7.70e-02	6.40e-02	1.03e+00	3.55e-01	1.57e-01	2.56e-01
20	1.48e+01	2.05e+01	1.80e-01	1.80e-01	7.21e-01	6.22e-01	2.98e-01	2.85e-01
21	7.95e+01	1.25e+02	4.10e-01	3.10e-01	1.21e+00	3.33e+00	5.57e-01	1.13e+00
22	4.61e+01	6.03e+01	4.40e-02	2.80e-02	1.17e+01	9.74e+00	2.35e-01	1.97e-01
23	3.23e+02	4.44e+01	3.30e+02	0.00e+00	3.29e+02	0.00e+00	2.00e+02	0.00e+00
24	1.24e+02	8.77e+00	1.10e+02	2.30e+00	1.19e+02	6.59e+00	1.13e+02	3.61e+00
25	1.77e+02	2.88e+01	1.30e+02	4.00e+01	1.30e+02	1.93e+01	1.32e+02	2.43e+01
26	1.00e+02	9.75e-02	1.00e+02	1.60e-02	1.00e+02	3.65e-02	1.00e+02	1.48e-02
27	1.90e+02	1.75e+02	5.80e+01	1.30e+02	9.12e+01	1.40e+02	1.73e+01	5.38e+01
28	3.94e+02	6.47e+01	3.80e+02	3.20e+01	3.87e+02	4.88e+01	2.00e+02	0.00e+00
29	1.24e+03	1.66e+03	2.20e+02	4.60e-01	2.13e+02	2.59e+01	2.03e+02	2.16e+01
30	6.47e+02	2.13e+02	4.60e+02	1.30e+01	5.05e+02	1.06e+02	2.00e+02	0.00e+00

Table 5.11: Optimizer Final Cost Error Mean and Standard Deviation n = 30

	Meta-Opt		LSHADE		RSDE		UMOEAs	
Fn	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
1	7.87e-02	5.56e-01	0.00e+00	0.00e+00	1.50e+03	1.70e+03	0.00e+00	0.00e+00
2	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
3	4.32e-04	2.58e-03	0.00e+00	0.00e+00	4.74e-02	1.16e-01	0.00e+00	0.00e+00
4	1.33e+01	1.79e+00	0.00e+00	0.00e+00	3.05e+00	1.34e+01	0.00e+00	0.00e+00
5	1.62e+00	4.53e-01	2.00e+01	3.70e-02	2.03e+01	9.88e-02	2.01e+01	1.26e-01
6	8.97e+00	1.78e+00	0.00e+00	0.00e+00	5.16e+00	2.01e+00	0.00e+00	0.00e+00
7	1.76e-02	3.57e-02	0.00e+00	0.00e+00	8.46e-04	1.59e-03	0.00e+00	0.00e+00
8	1.20e+01	4.38e+00	0.00e+00	0.00e+00	2.04e+01	7.04e+00	1.35e+00	1.16e+00
9	1.14e+02	4.70e+01	6.80e+00	1.50e+00	5.80e+01	1.65e+01	8.84e+00	2.78e+00
10	3.57e+02	2.11e+02	1.60e-02	1.60e-02	3.29e+02	2.47e+02	8.93e+00	1.65e+01
11	3.02e+03	6.68e+02	1.20e+03	1.80e+02	2.74e+03	6.44e+02	1.46e+03	7.91e+02
12	1.52e+03	2.42e+03	1.60e-01	2.30e-02	4.44e-01	1.66e-01	2.56e-03	2.35e-03
13	4.36e-01	1.31e-01	1.20e-01	1.70e-02	3.05e-01	5.50e-02	5.46e-02	1.55e-02
14	3.95e-01	2.00e-01	2.40e-01	3.00e-02	2.36e-01	3.37e-02	2.04e-01	4.02e-02
15	8.62e+00	4.06e+00	2.10e+00	2.50e-01	5.92e+00	2.59e+00	3.25e+00	5.21e-01
16	1.16e+01	8.17e-01	8.50e+00	4.60e-01	1.06e+01	7.70e-01	9.93e+00	7.41e-01
17	2.05e+03	1.40e+03	1.90e+02	7.50e+01	1.24e+03	3.79e+02	9.77e+02	3.61e+02
18	8.66e+02	5.37e+03	5.90e+00	2.90e+00	9.54e+01	4.34e+01	2.12e+01	1.04e+01
19	9.82e+00	1.64e+00	3.70e+00	6.80e-01	5.65e+00	1.46e+00	3.56e+00	6.90e-01
20	6.12e+01	4.86e+01	3.10e+00	1.50e+00	3.73e+01	2.55e+01	1.10e+01	4.45e+00
21	1.17e+03	9.99e+02	8.70e+01	9.00e+01	4.71e+02	2.34e+02	3.38e+02	2.19e+02
22	6.07e+02	2.39e+02	2.80e+01	1.80e+01	1.91e+02	1.19e+02	9.54e+01	8.05e+01
23	3.17e+02	0.00e+00	3.20e+02	0.00e+00	3.15e+02	1.62e-06	2.00e+02	0.00e+00
24	2.23e+02	8.77e+00	2.20e+02	1.10e+00	2.24e+02	1.65e+00	2.00e+02	0.00e+00
25	2.05e+02	2.57e+00	2.00e+02	5.00e-02	2.03e+02	1.17e-01	2.00e+02	0.00e+00
26	1.01e+02	1.31e-01	1.00e+02	1.60e-02	1.00e+02	4.14e-02	1.00e+02	2.83e-02
27	5.62e+02	5.02e+01	3.00e+02	0.00e+00	4.69e+02	9.46e+01	2.00e+02	0.00e+00
28	9.52e+02	2.25e+02	8.40e+02	1.40e+01	9.05e+02	1.21e+02	2.00e+02	0.00e+00
29	5.80e+03	4.36e+03	7.20e+02	5.10e+00	6.52e+05	2.66e+06	2.05e+02	2.98e+00
30	4.04e+03	1.37e+03	1.20e+03	6.20e+02	1.70e+03	8.67e+02	2.00e+02	0.00e+00

Table 5.12: Optimizer Final Cost Error Mean and Standard Deviation n = 50

	Meta-Opt		LSHADE		RSDE		UMOEAs	
Fn	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
1	0.00e+00	0.00e+00	1.20e+03	1.50e+03	2.25e+04	1.21e+04	0.00e+00	0.00e+00
2	1.63e-01	8.11e-01	0.00e+00	0.00e+00	3.58e+03	6.56e+03	0.00e+00	0.00e+00
3	7.36e-04	5.20e-03	0.00e+00	0.00e+00	4.10e-01	1.38e+00	0.00e+00	0.00e+00
4	2.77e+01	6.81e+00	5.90e+01	4.60e+01	6.41e+01	3.62e+01	7.82e-02	5.58e-01
5	2.48e+00	6.33e-01	2.00e+01	4.60e-02	2.05e+01	9.61e-02	2.01e+01	1.80e-01
6	1.81e+01	2.44e+00	2.60e-01	5.20e-01	1.72e+01	4.33e+00	6.03e-02	3.02e-01
7	1.38e-02	1.68e-02	0.00e+00	0.00e+00	2.40e-03	4.40e-03	0.00e+00	0.00e+00
8	2.58e+01	1.02e+01	0.00e+00	0.00e+00	5.04e+01	1.24e+01	4.29e+00	3.04e+00
9	2.45e+02	9.88e+01	1.10e+01	2.10e+00	1.42e+02	3.51e+01	1.94e+01	3.61e+00
10	6.80e+02	3.12e+02	1.20e-01	4.10e-02	1.52e+03	9.46e+02	7.55e+01	9.45e+01
11	6.37e+03	9.23e+02	3.20e+03	3.30e+02	6.15e+03	1.05e+03	3.98e+03	1.99e+03
12	3.09e+03	5.07e+03	2.20e-01	2.80e-02	5.38e-01	2.02e-01	1.11e-03	9.06e-04
13	5.08e-01	2.08e-01	1.60e-01	1.80e-02	4.23e-01	6.05e-02	9.85e-02	2.02e-02
14	4.86e-01	2.70e-01	3.00e-01	2.50e-02	2.78e-01	2.25e-02	2.24e-01	3.38e-02
15	2.08e+01	7.06e+00	5.20e+00	5.10e-01	9.96e+00	6.53e+00	5.46e+00	9.66e-01
16	2.08e+01	7.11e-01	1.70e+01	4.80e-01	1.93e+01	8.42e-01	1.92e+01	7.28e-01
17	5.16e+03	5.85e+03	1.40e+03	5.10e+02	4.10e+03	1.68e+03	2.45e+03	4.60e+02
18	2.49e+02	1.00e+02	9.70e+01	1.40e+01	3.40e+02	2.38e+02	9.00e+01	6.12e+01
19	1.37e+01	5.59e+00	8.30e+00	1.80e+00	1.46e+01	2.03e+00	1.17e+01	2.07e+00
20	2.05e+02	1.48e+02	1.40e+01	4.60e+00	1.60e+02	7.30e+01	7.08e+01	3.00e+01
21	3.73e+03	2.51e+03	5.20e+02	1.50e+02	1.58e+03	6.55e+02	1.47e+03	3.92e+02
22	1.39e+03	4.51e+02	1.10e+02	7.50e+01	4.61e+02	2.34e+02	3.64e+02	1.76e+02
23	3.39e+02	0.00e+00	3.40e+02	0.00e+00	3.44e+02	1.19e-05	2.00e+02	0.00e+00
24	2.73e+02	5.92e+00	2.80e+02	6.60e-01	2.76e+02	2.38e+00	2.00e+02	0.00e+00
25	2.09e+02	6.24e+00	2.10e+02	3.60e-01	2.06e+02	7.81e-01	2.00e+02	0.00e+00
26	1.01e+02	1.88e-01	1.00e+02	1.40e+01	1.12e+02	4.97e+01	1.04e+02	1.96e+01
27	8.22e+02	7.94e+01	3.30e+02	3.00e+01	8.04e+02	1.00e+02	2.00e+02	0.00e+00
28	1.47e+03	3.79e+02	1.10e+03	2.90e+01	1.61e+03	3.90e+02	2.00e+02	0.00e+00
29	1.52e+04	3.14e+03	7.90e+02	2.40e+01	5.28e+06	1.64e+07	2.16e+02	2.52e+00
30	7.09e+03	2.36e+03	8.70e+03	4.10e+02	1.12e+04	1.75e+03	2.00e+02	0.00e+00

Table 5.13: Optimizer Final Cost Error Mean and Standard Deviation n = 100

	Meta-Opt		LSHADE		RSDE		UMOEAs	
Fn	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
1	5.61e+03	6.87e+03	1.70e+05	5.70e+04	8.33e+05	2.89e+05	0.00e+00	0.00e+00
2	0.00e+00	0.00e+00	0.00e+00	0.00e+00	7.39e+03	9.84e+03	0.00e+00	0.00e+00
3	0.00e+00	0.00e+00	0.00e+00	0.00e+00	9.77e-01	2.21e+00	0.00e+00	0.00e+00
4	1.03e+03	3.65e+03	1.70e+02	3.10e+01	1.86e+02	4.06e+01	2.31e+01	1.46e+00
5	4.26e+00	1.16e+00	2.10e+01	3.10e-02	2.08e+01	7.85e-02	2.00e+01	1.02e-02
6	4.78e+01	4.18e+00	8.70e+00	2.30e+00	6.02e+01	7.48e+00	8.52e-01	8.71e-01
7	2.03e-03	6.90e-03	0.00e+00	0.00e+00	1.27e-03	2.71e-03	0.00e+00	0.00e+00
8	9.09e+01	2.42e+01	1.10e-02	7.40e-03	1.94e+02	3.02e+01	2.45e+01	1.05e+01
9	7.47e+02	3.96e+02	3.40e+01	5.00e+00	3.20e+02	5.41e+01	5.38e+01	6.68e+00
10	2.19e+03	6.59e+02	2.60e+01	5.80e+00	9.31e+03	2.67e+03	2.99e+03	1.57e+03
11	1.42e+04	1.60e+03	1.10e+04	5.60e+02	1.55e+04	1.54e+03	7.88e+03	1.44e+03
12	1.01e+04	1.17e+04	4.40e-01	4.70e-02	7.42e-01	1.97e-01	6.77e-04	4.18e-04
13	6.37e-01	2.35e-01	2.40e-01	2.10e-02	5.44e-01	4.09e-02	2.05e-01	3.36e-02
14	2.58e+00	1.27e+01	1.20e-01	7.30e-03	2.09e-01	1.23e-02	2.27e-01	2.15e-02
15	7.88e+01	2.88e+01	1.60e+01	1.20e+00	5.24e+01	1.82e+01	1.17e+01	1.41e+00
16	4.46e+01	1.08e+00	3.90e+01	4.80e-01	4.24e+01	1.21e+00	4.26e+01	7.72e-01
17	1.99e+04	1.19e+04	4.40e+03	7.10e+02	9.86e+04	4.60e+04	5.30e+03	7.94e+02
18	4.79e+08	3.36e+09	2.20e+02	1.70e+01	1.26e+03	1.08e+03	4.10e+02	1.03e+02
19	3.25e+01	2.34e+00	9.60e+01	2.30e+00	8.16e+01	2.58e+01	5.84e+01	8.74e+00
20	6.44e+02	1.96e+02	1.50e+02	5.20e+01	5.50e+02	1.76e+02	3.12e+02	6.68e+01
21	1.67e+04	1.10e+04	2.30e+03	5.30e+02	3.49e+04	1.90e+04	4.42e+03	1.60e+03
22	3.78e+03	1.82e+03	1.10e+03	1.90e+02	1.51e+03	4.63e+02	9.27e+02	3.22e+02
23	3.47e+02	4.38e+01	3.50e+02	0.00e+00	3.48e+02	2.12e-03	2.00e+02	0.00e+00
24	3.43e+02	1.74e+01	3.90e+02	2.90e+00	4.06e+02	5.67e+00	2.00e+02	1.51e-03
25	2.40e+02	3.58e+01	2.00e+02	0.00e+00	2.42e+02	7.50e+00	2.00e+02	0.00e+00
26	2.35e+02	1.87e+02	2.00e+02	0.00e+00	1.98e+02	4.97e+01	1.98e+02	1.40e+01
27	1.71e+03	1.40e+02	3.80e+02	3.30e+01	2.01e+03	1.65e+02	2.00e+02	0.00e+00
28	3.12e+03	8.16e+02	2.30e+03	4.60e+01	4.11e+03	7.20e+02	2.00e+02	0.00e+00
29	2.12e+04	4.10e+03	8.00e+02	7.60e+01	8.29e+07	8.20e+07	2.55e+02	1.18e+01
30	1.37e+04	3.41e+03	8.30e+03	9.60e+02	1.31e+04	2.31e+03	2.00e+02	0.00e+00

On function 5, meta-optimization was the most effective across all dimensions and was the best on function 19 for $n = 100$. However, meta-optimization struggled on functions 7, 9, 12, 18, and 22 in most dimensions. Function 7 is a variation of the Griewank function (f_5) which meta-optimization was not able to solve consistently and used a large number of function calls to solve when it was able to. In the case of function 9, a variation of the Rastrigin function (f_8), meta-optimization was able to solve it every time, but required a large number of function calls and some restarts to achieve this result. For both of these functions, the computation time limit played a significant role in degrading the performance of the meta-optimizer.

Table 5.14: Optimizer Mean Rank

Dimension	Meta-Opt	LSHADE	RSDE	UMOEAs
10	3.67	1.53	2.67	1.57
30	3.60	1.47	3.00	1.43
50	3.17	1.80	3.40	1.50
100	3.03	1.87	3.33	1.47

Table 5.15: Number of Times With Top Rank

Dimension	Meta-Opt	LSHADE	RSDE	UMOEAs
10	2	19	5	15
30	2	19	2	18
50	2	15	0	17
100	4	13	1	19

Tables 5.14 and 5.15 provide aggregate results for each optimizer. In Table 5.14, each optimizer was given a rank from 1 to 4 based on the mean final error for each function. The ranks for each optimizer were then averaged over the 30 functions in each dimension. Table 5.15 counts the number of times each optimizer was ranked first on a given problem for each dimension. At $n = 10$, meta-optimization performed the worst of the four methods and struggled on a number of functions. As the dimension

increased, meta-optimization performed better, surpassing RSDE at $n = 50$ and $n = 100$. It should again be noted that the meta-optimization configuration was not tuned to handle the restrictions for the competition suite. Since the lower dimensional cases have smaller computational budgets, meta-optimization does not have sufficient time to determine effective optimizers and allow them to solve the problem. If given sufficient time to run, meta-optimization would achieve excellent results on many of these functions. In addition, meta-optimization is limited by the capabilities of the included optimizers which individually struggle on these functions. Since meta-optimization considers each optimizer as an individual kernel function that it can deploy, these state of the art optimizers could potentially be included in the meta-optimization bank, taking advantage of their superior performance on these problems.

CHAPTER 6
SMART PROJECTILE PARAMETER ESTIMATION RESULTS

In this chapter, the parameter estimation algorithm using the output error method previously described in Chapter 2 will be applied to parameter estimation of a smart projectile system. Since one of the primary features of the method is the robust optimization algorithm, results will be focused on the performance of the numerical optimization scheme. The Army-Navy Finner (ANF) described in Section 2.4 will be used as the example round. The projectile is equipped with a new microspoiler control mechanism. Spark range data is used to estimate unknown parameters for this system. As shown in Chapter 3, the topology of the parameter space for this parameter estimation problem contains numerous local minima caused by the interactions of the estimated parameters with the projectile dynamics.

To exercise the parameter estimation algorithm, cases will be examined where synthetic trajectory data is generated via simulation of the projectile system and cases where data is obtained from flight experiments in a spark range. Cases using simulated flight data are advantageous as the underlying parameters to be identified are known, allowing for accurate validation of the parameter estimation procedure. The simulated data is also used to explore the nature of the projectile parameter estimation problem through various trade studies. Finally, parameter estimation is performed on data collected from experimental flight tests at the U.S. Army Research Laboratory spark range.

6.1 Simulated Trajectory Results

All synthetic measurement data used in this chapter is generated from simulation of the smart projectile system described in Section 2.4. The mass properties and

standard aerodynamic coefficients for this projectile are given in Tables 2.1 and 2.2 respectively with the microspoiler parameters given in Table 2.3. For this test, the projectile and microspoiler aerodynamic coefficients were assumed to vary linearly with Mach number. In general practice, the aerodynamic coefficients can be assumed to be linear over small ranges of Mach number. The linear coefficients were defined using the coefficient values at Mach 2 and Mach 3 in Table 2.2 or the projectile aerodynamics and the values in Table 2.3 for the microspoiler coefficients. The microspoilers are modeled using Eqs. 2.14 and 2.15 with a spin rate Ω_0 of 440 rad/s and a time constant τ_{ms} of 0.025 s^{-1}.

A build up procedure is employed to accurately estimate the projectile aerodynamic coefficients and control parameters. This process is designed to mimic a typical test procedure for conducting spark range tests. First, the uncontrolled round is fired with low initial angular velocity in order to maintain a low angle of attack for the entire flight. This low angle of attack trajectory allows for the accurate estimation of C_{X0}, C_{l0}, and C_{lp} as these parameters do not depend on angle of attack. With these parameters estimated, the uncontrolled round is next fired with an initial angular velocity perturbation, such as the perturbation due to a yaw inducer, to generate trajectories with significant angle of attack. The remaining projectile body aerodynamic coefficients can then be estimated from these trajectories. The final case considers the round fired with the microspoilers deployed for the entire flight. By fixing the projectile body aerodynamic coefficients to their estimated values, the effects of the microspoilers can be isolated from the body aerodynamic behavior allowing for more accurate estimates.

The initial conditions for each simulated trajectory were randomly generated based on a nominal value of $z_0 = -5$ m and $u_0 = 1023$ m/s and standard deviations given in Table 6.1. The standard deviation on u_0 was selected to produce a sufficient distribution of Mach numbers to properly estimate the linear aerodynamic coefficients.

The exception is the high angle of attack case which adds an additional $q_0 = 20$ rad/s. During spark range testing, this can be achieved by using a yaw inducer to perturb the round as it leaves the barrel of the gun. ϕ_0 was initialized to a random value between $-180°$ and $180°$. For the active microspoiler trajectories, ω_0 was set to a random value between $0°$ and $360°$.

Table 6.1: Standard Deviations of Initial Conditions Used To Generate Synthetic Data

θ (rad)	ψ (rad)	u (m/s)	p (rad/s)	q (rad/s)	r (rad/s)
0.001	0.001	35.0	2.0	1.0	1.0

Simulated measurements are recorded at the range locations of the spark stations at the U.S. Army Research Laboratory Transonic Experimental Facility spark range. Measurement noise is added with standard deviation of 3 mm and $0.1°$ for position and angle measurements respectively. The probability of excluding a measurement was set at 10%. The ϕ measurements are wrapped to $-180°$ and $180°$. Eq. 2.13 is used to characterize the aerodynamic coefficients within the cost function with a Mach range from 2.75 to 3.25. This range covers the potential distribution of Mach numbers encountered in the simulated trajectories.

This parameter estimation technique uses the meta-optimization configuration given in Table 5.8 with two modifications. Compared to the simple mathematical benchmark functions, the parameter estimation cost function takes significantly longer to compute, greatly increasing the run time for even a small number of iterations. Due to this issue, the restarting threshold was reduced to 200,000 function calls. Since the optimal cost is unknown *a priori* for these parameter estimation problems, a χ^2 threshold of 30.0 for each state is set to indicate an acceptable fit. The meta-optimizer is stopped when there has been no cost reduction for 25,000 functions calls and the cost is below the threshold. The second modification to the meta-optimization configuration adds a second reference cost reduction rate which applies below the χ^2 threshold.

Under normal operation on this parameter estimation problem, the meta-optimizer will slow down considerably as it approaches the solution, making only small improvements compared to earlier in the process. To account for this, a lower reference reduction rate is used to prevent optimizers from being improperly penalized during this phase.

6.1.1 Simulated Low Angle of Attack Results

For the low angle of attack case, three trajectories were used to estimate C_{X0}, C_{l0}, and C_{lp}. Since these were the only parameters estimated, only x position and roll angle were included in the cost function. All of the states and measurements were weighted equally in the cost function. The trajectory prediction simulations within the cost function were initialized from launch, requiring estimates for ϕ_0, u_0, p_0, and t_0. In total, this case required estimates for 18 different parameters. The results for this case are given in Table 6.2 and the final χ^2 values in Table 6.3. The values of the linear aerodynamic model at $M = 2.75$ and $M = 3.25$ are used as the truth values for evaluating the accuracy of the parameter estimates. As seen in Table 6.2, the parameter estimation algorithm accurately estimated all three coefficients and obtained good fits for both x and ϕ. χ^2 values in this range indicate that the errors in the optimal fits are on the order of the magnitude of the noise added to the measurements.

Table 6.2: Simulated Low Angle of Attack Parameter Estimation Results

	M = 2.75			**M = 3.25**		
Parameter	**Actual**	**Estimate**	**% Error**	**Actual**	**Estimate**	**% Error**
C_{X0}	0.51	0.5086	0.278	0.4384	0.4400	0.3675
C_{l0}	0.0495	0.0496	0.2455	0.038	0.0378	0.4558
C_{lp}	-4.5845	-4.5982	0.2999	-3.5213	-3.4996	0.6161

Table 6.3: χ^2 Values for Simulated Low Angle of Attack Trajectories

	x	ϕ
Average	16.19	18.51
Standard Deviation	2.538	2.786

State histories for one of the three trajectories used in the cost function is shown in Figures 6.1-6.4. These trajectories show how accurately this parameter estimation method fits the measurement data, even in the presence of noise. Figure 6.1 shows the x error at every measurement for the optimal trajectory in place of the full x trajectory. This is due to the fact that the measurement errors are extremely small relative to the position. For this trajectory, the estimated x measurements are all within 6 mm from the data. In Figure 6.2, the final estimated trajectory is shown alongside the measurements and the initial trajectory obtained during the initialization phase of the parameter estimation process. The initial trajectory provides an indication of how poor the initial parameter estimates were and how much of an improvement this method makes. In the case of ϕ, the initial trajectory is completely out of phase with the measurements while the final trajectory fits the data perfectly. Finally, Figures 6.3 and 6.4 show state histories of u and p respectively. Another advantage of analyzing simulated data is that states that are not directly measured can still be plotted as a comparison to the estimated trajectory. For both u and p, the final trajectory matches the states exactly which would only be possible with highly accurate parameter estimates.

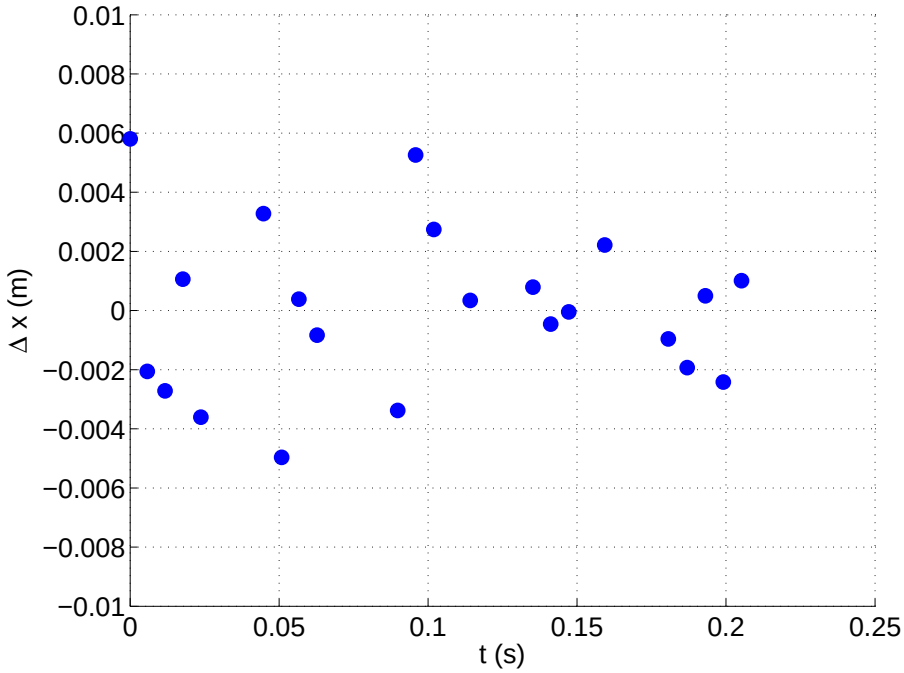


Figure 6.1: Simulated Low Angle of Attack X Error vs. Time

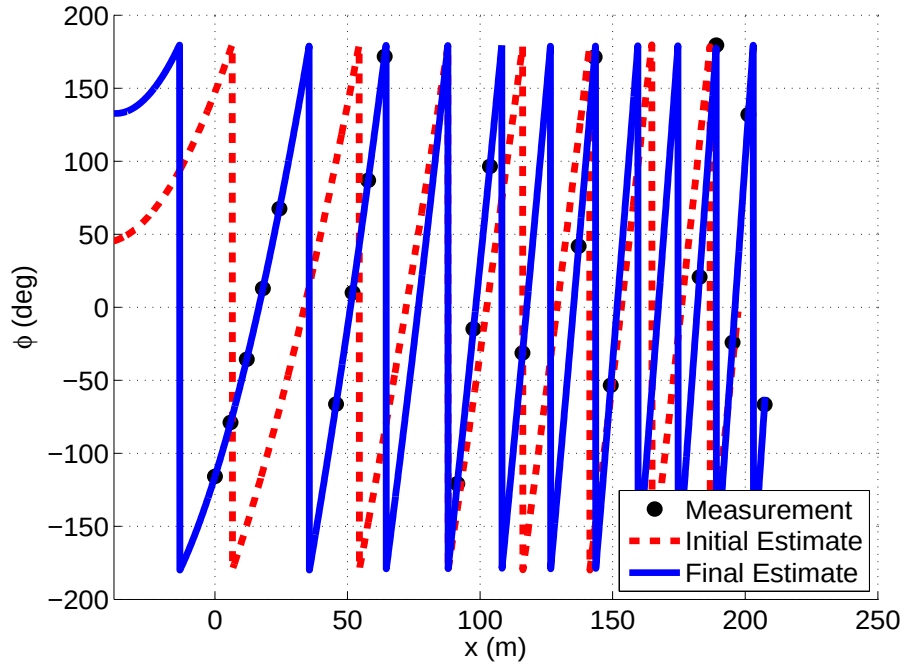


Figure 6.2: Simulated Low Angle of Attack Roll Angle vs. Range

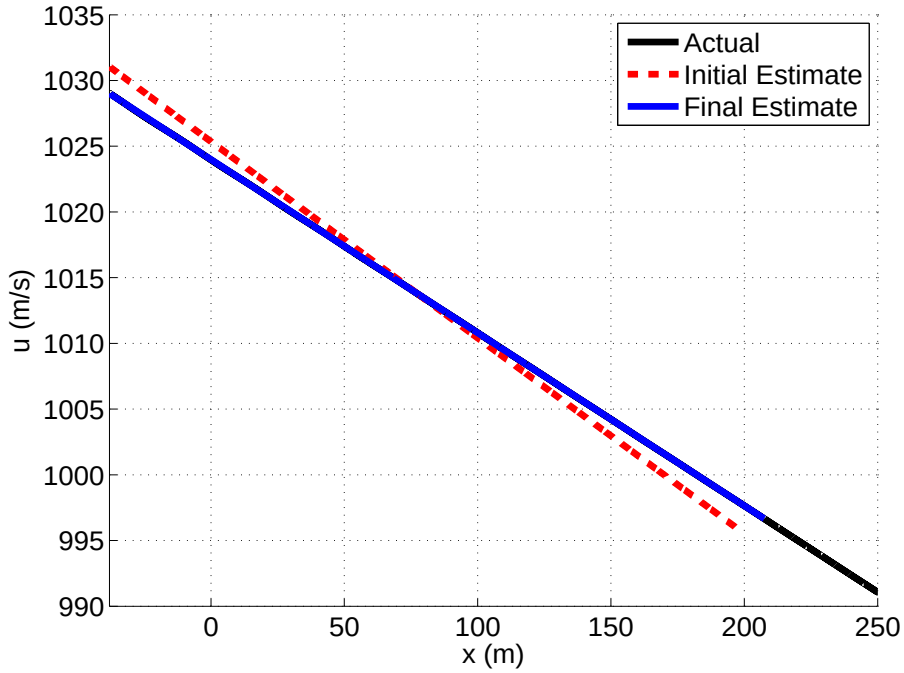


Figure 6.3: Simulated Low Angle of Attack Body X Velocity vs. Range

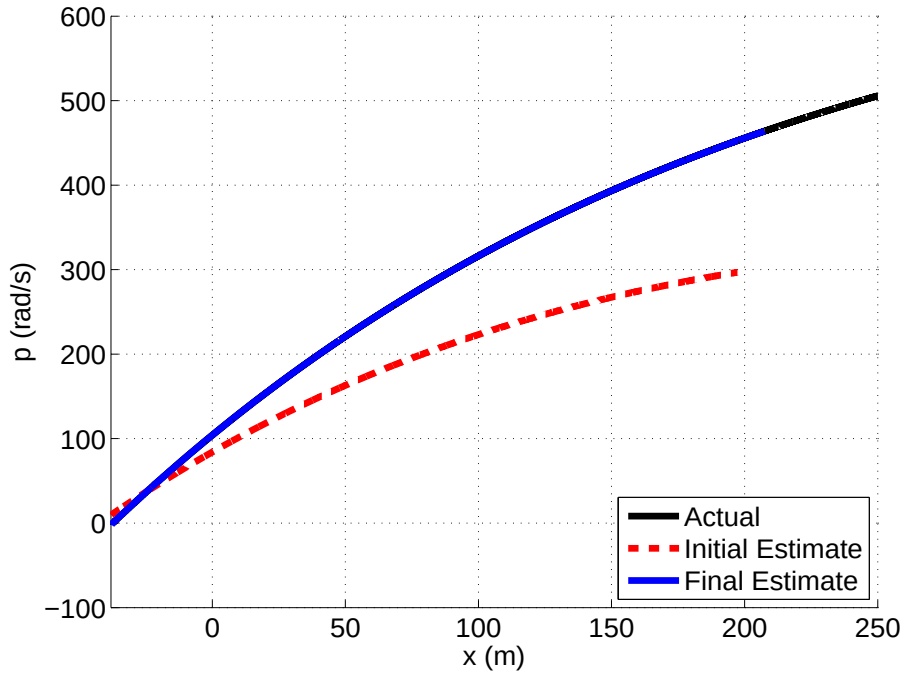


Figure 6.4: Simulated Low Angle of Attack Roll Rate vs. Range

The cost reduction profile is seen in Figure 6.5 below. For this case, about 300,000 function calls were needed for the meta-optimizer to reduce the cost below the threshold. After a large reduction in cost over the first 50,000 function calls, progress slows as the meta-optimizer tries all of the optimizers, looking for one to perform well. Finally, SIM takes over and reduces the cost below the threshold. Figure 6.6 shows the total percent cost reduction for each optimizer. This is a cumulative metric which evaluates the total contributions of each optimizer towards reducing the cost. On this run, CG, DE, and SIM provide almost all of the cost reduction with small amounts from SD, BFGS, and IWO. The total number of times each optimizer was deployed over this run is shown in Figure 6.7. Overall, the optimizers were all given roughly equal opportunities to work on the problem with CG being used the most. As CG was the most effective optimizer, it was favored over the other optimizers and received 4 more calls than any other optimizer. The total number of function calls used by each optimizer in Figure 6.8 provides additional insight into the distribution of resources between the optimizers. Here, CG used the most function calls due to being the most effective and most used optimizer. Combined with the total cost reduction, the function calls used by each optimizer provide insight into the efficiency of each method. In particular, DE and SIM used less than 40% of the function calls as CG, but achieved over 50% as much cost reduction. PSO, TS, and ACO, on the other hand, used a similar number of function calls, but were unable to reduce the cost.

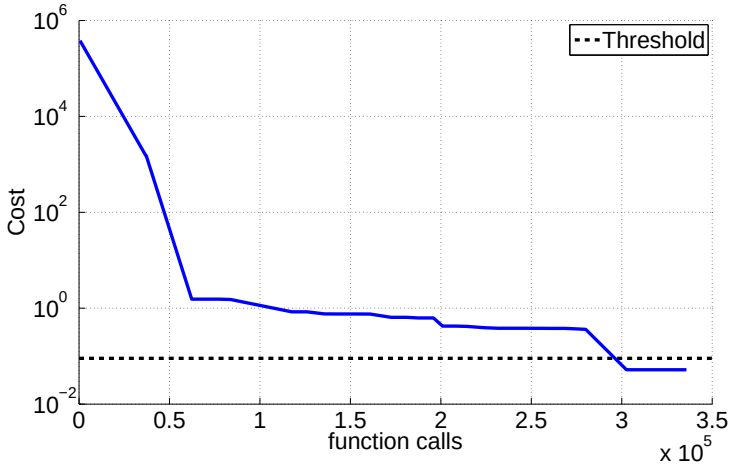


Figure 6.5: Simulated Low Angle of Attack Cost Profile

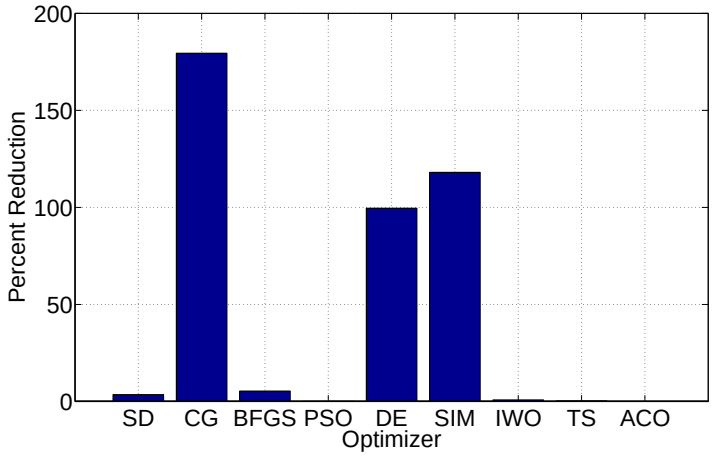


Figure 6.6: Simulated Low Angle of Attack Total Percent Cost Reduction

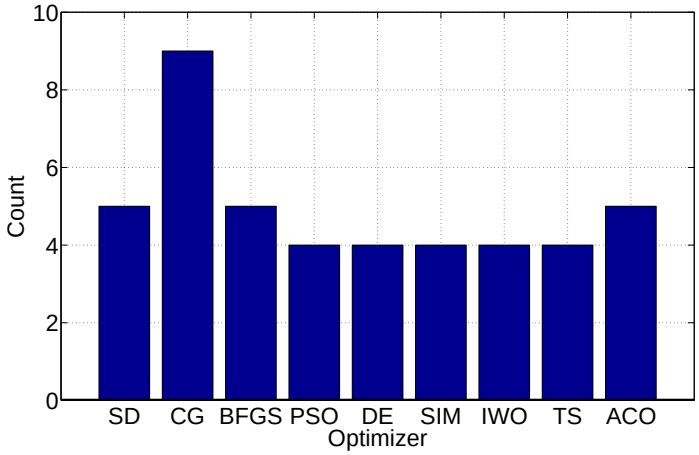


Figure 6.7: Simulated Low Angle of Attack Total Number of Calls of Each Optimizer

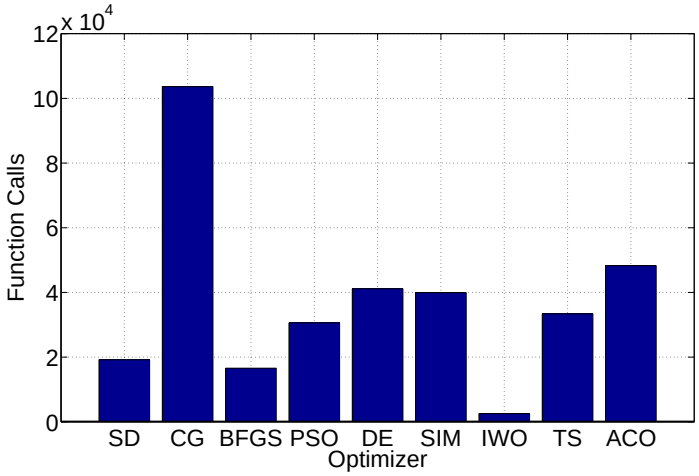


Figure 6.8: Simulated Low Angle of Attack Total Function Calls Used by Each Optimizer

Figures 6.9-6.11 show the profiles for the estimated parameters. After the large initial reductions in cost, both C_{X0} values come close to the actual values while the other parameters still maintain some error. The estimate of C_{X0} at $M = 2.75$ comes very close to the boundary after this initial reduction before moving back into the search space. C_{l0} and C_{lp} take longer to converge, slowly moving towards the solution before jumping to their final values after the last cost reduction around 300,000 function calls. While C_{lp} at $M = 2.75$ started near its boundary, none of the optimizers were attracted in that direction.

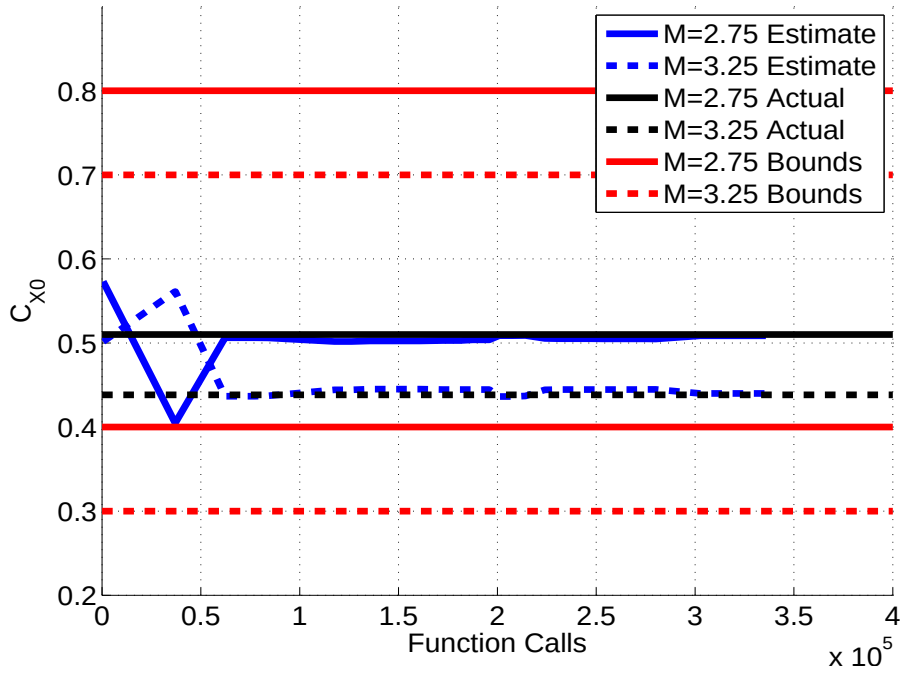


Figure 6.9: Simulated Low Angle of Attack Base Drag Profile

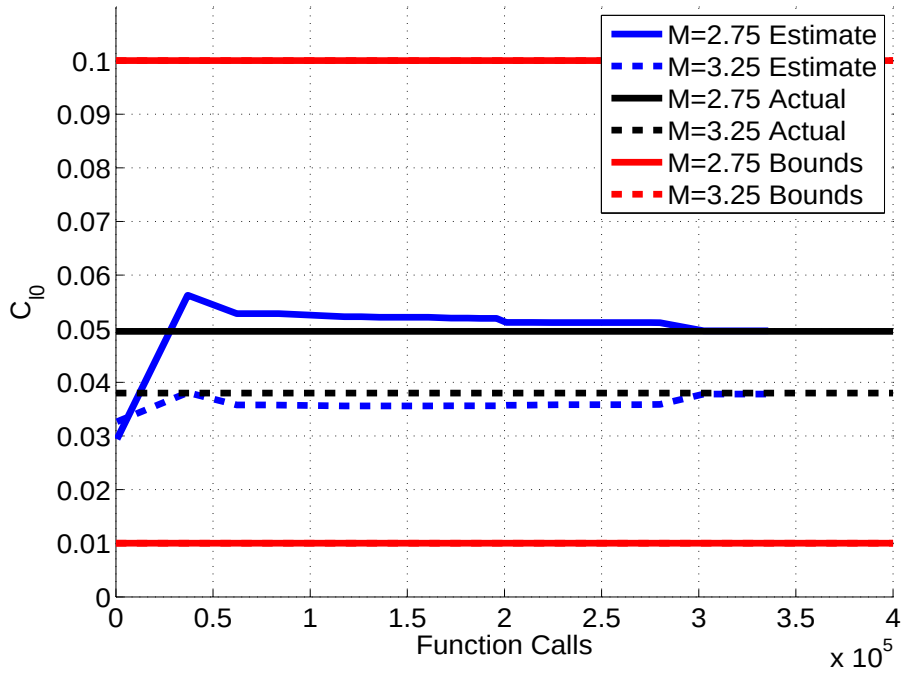


Figure 6.10: Simulated Low Angle of Attack Roll Generation Profile

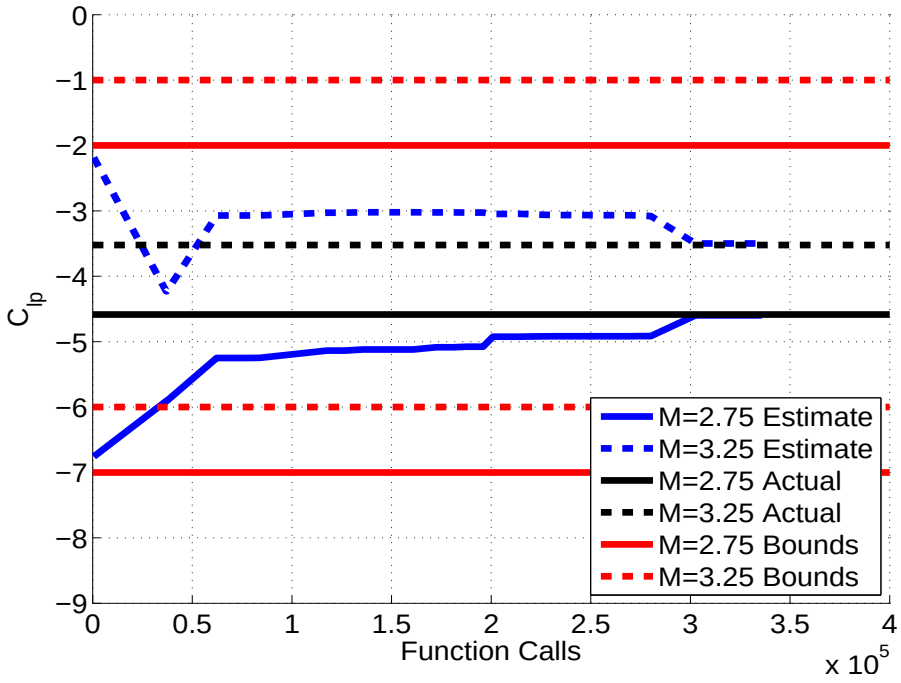


Figure 6.11: Simulated Low Angle of Attack Roll Damping Profile

6.1.2 Simulated High Angle of Attack Results

With the first three projectile body aerodynamic parameters estimated, the remaining four parameters C_{X2}, $C_{N\alpha}$, $C_{m\alpha}$ and C_{mq} can be estimated using five high angle of attack trajectories. To simplify the estimation process, the previously estimated body aerodynamic coefficients were held fixed to the estimated values from Table 6.2. All six states were included in the cost function. As with the low angle of attack case, the states and measurements were all equally weighted and the trajectory prediction simulations were started at launch. For five trajectories, this resulted in a total of 48 parameters to estimate. Table 6.4 gives the results for the remaining body coefficients with the χ^2 values for each state in Table 6.5

Table 6.4: Simulated High Angle of Attack Parameter Estimation Results

	M = 2.75			**M = 3.25**		
Parameter	**Actual**	**Estimate**	**% Error**	**Actual**	**Estimate**	**% Error**
C_{X2}	4.735	4.734	0.0139	4.065	3.3866	16.668
$C_{N\alpha}$	8.8112	9.0298	2.481	7.5107	7.8045	3.911
$C_{m\alpha}$	-13.3308	-13.3186	0.0912	-7.4012	-7.4168	0.2107
C_{mq}	-350.275	-348.025	0.6424	-313.125	-316.95	1.2216

Table 6.5: χ^2 Values for Simulated High Angle of Attack Trajectories

	x	y	z	ϕ	θ	ψ
Average	19.85	19.09	18.55	20.84	26.25	14.41
Standard Deviation	7.221	2.324	6.59	3.214	11.39	3.855

For this case, the parameter estimation algorithm accurately estimated most of the parameters with some small error in the estimates of $C_{N\alpha}$ and a large error in C_{X2} at $M = 3.25$. Even with these errors, the χ^2 values for all of the states indicate very good fits of the data. This implies that the errors in the estimated parameters are likely due to poor observability in these parameters compared to the others. C_{X2} and $C_{N\alpha}$ are known to be difficult to estimate as they have smaller impacts on the flight of the projectile. These observability issues are exacerbated when measurement noise is added to the data. Without any errors in the measurements, all of the parameters can be estimated exactly. In addition, the parameters at $M = 3.25$ may be less observable in general as the trajectories are almost always below this Mach number. In the case of C_{X2}, given the linear model for the coefficient and a very accurate estimate at $M = 2.75$, the actual error in the coefficient in the Mach number ranges experienced in flight would be much less.

Figures 6.12-6.24 show the full state histories for one trajectory used in this case. Note that altitude h is plotted in place of z and is given by $h = -z$. Figures 6.12-6.17

correspond to the six states in the cost function. These trajectories help corroborate the excellent fits indicated by χ^2. The measurement errors can be seen clearly in Figures 6.13 and 6.14 with some measurements slightly off of the fit trajectory. In terms of the large error in the C_{X2} measurement, Figure 6.12 shows a good fit in terms of the x measurements and there is no perceptible difference between the u trajectories in Figure 6.18. For this trajectory in particular, this demonstrates how the trajectories are not sensitive to errors in C_{X2}. The final trajectory fits the remaining states extremely well, reinforcing the accuracy of the parameter estimates.

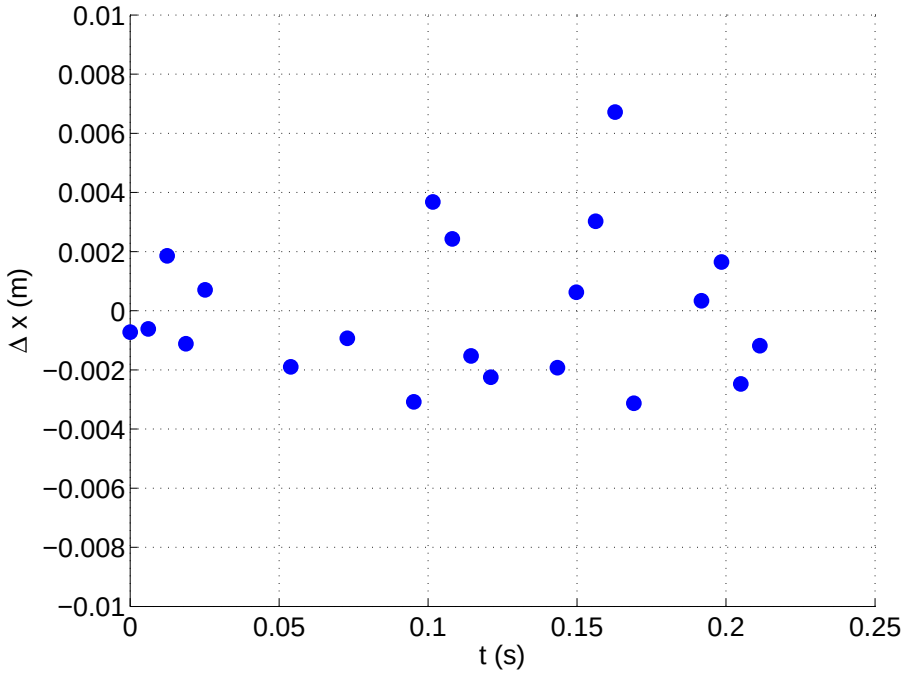


Figure 6.12: Simulated High Angle of Attack X Error vs. Time

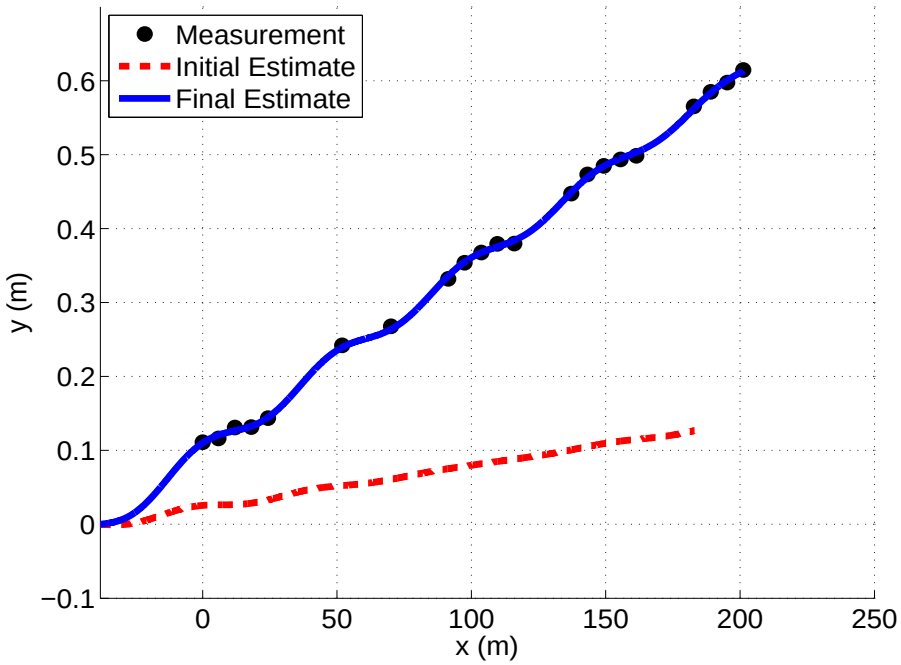


Figure 6.13: Simulated High Angle of Attack Inertial-Y Position vs. Range

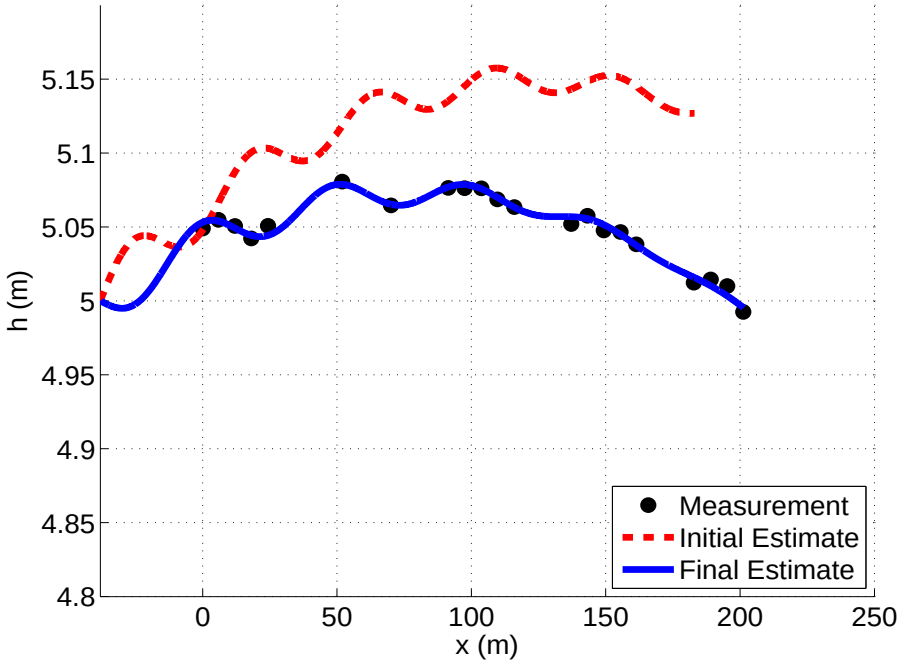


Figure 6.14: Simulated High Angle of Attack Altitude vs. Range

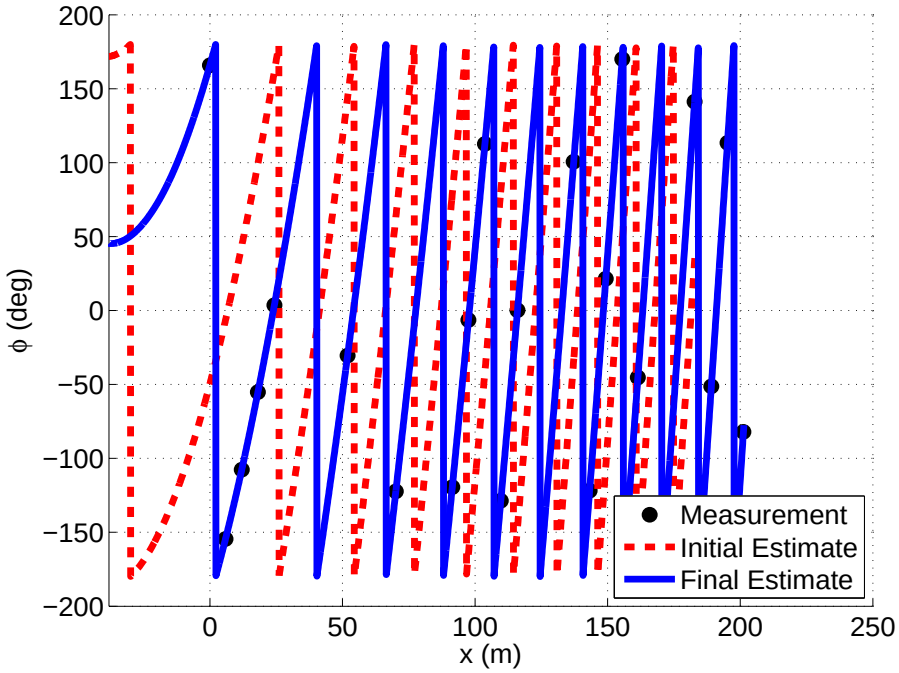


Figure 6.15: Simulated High Angle of Attack Roll Angle vs. Range

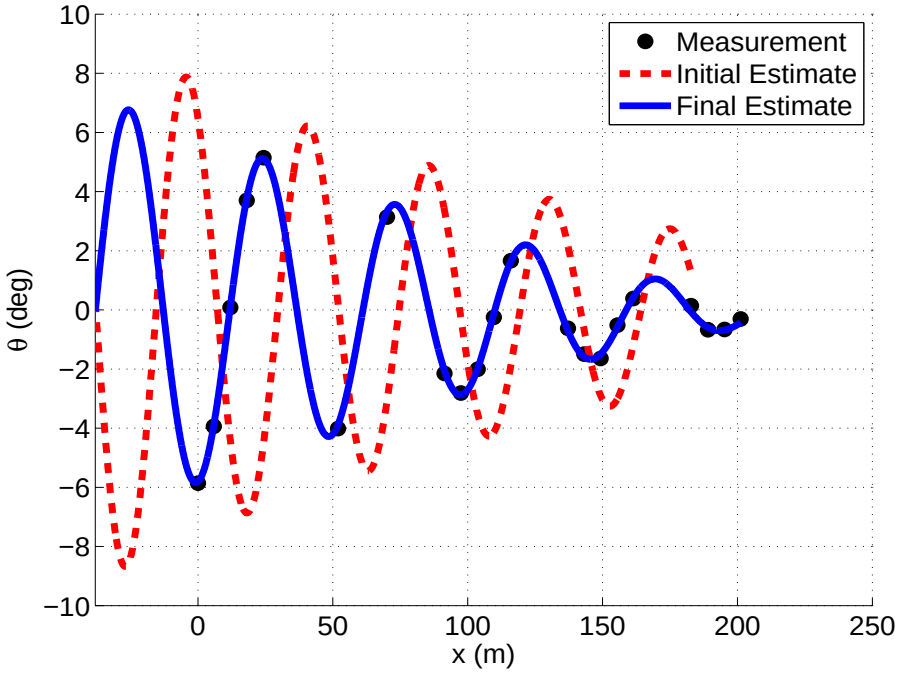


Figure 6.16: Simulated High Angle of Attack Pitch Angle vs. Range

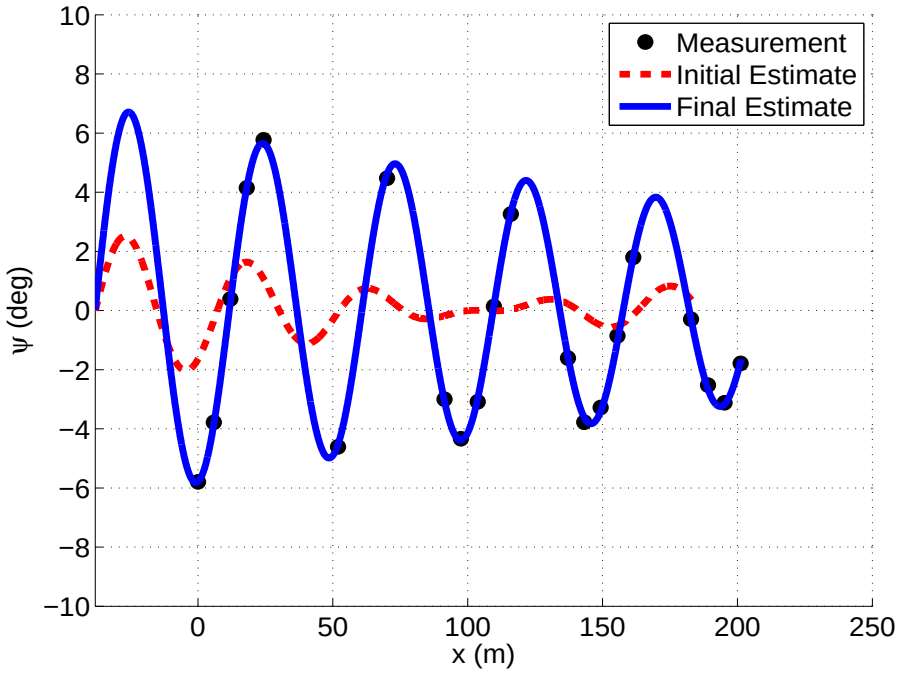


Figure 6.17: Simulated High Angle of Attack Yaw Angle vs. Range

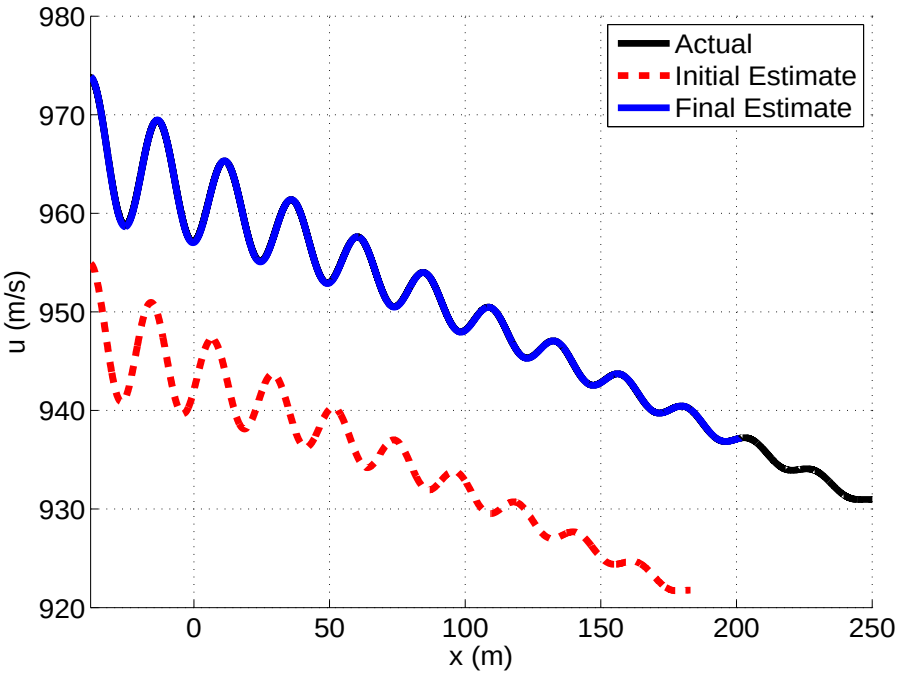


Figure 6.18: Simulated High Angle of Attack Body X Velocity vs. Range

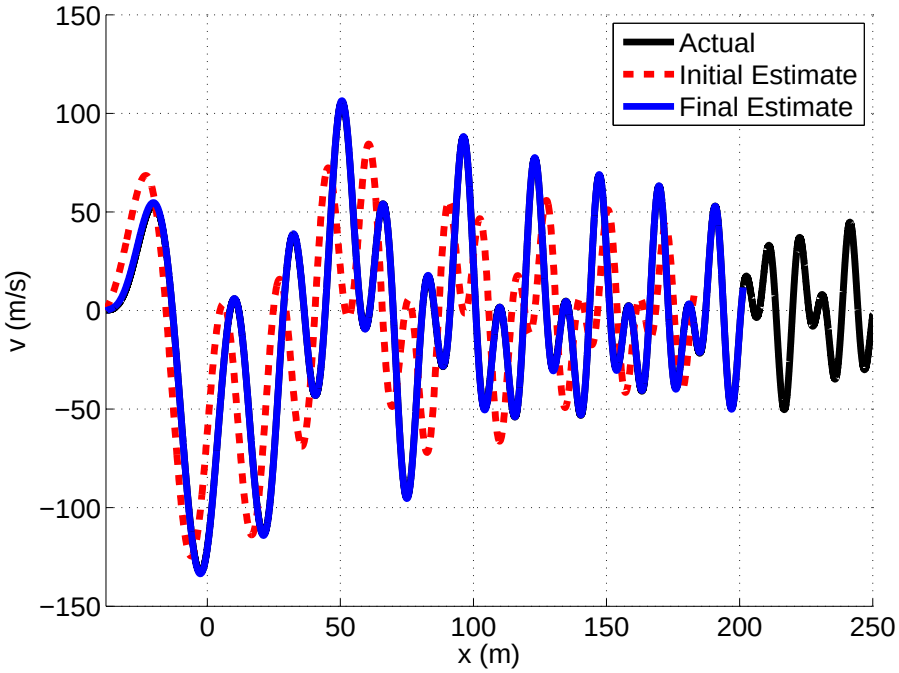


Figure 6.19: High Angle of Attack Body Y Velocity vs. Range

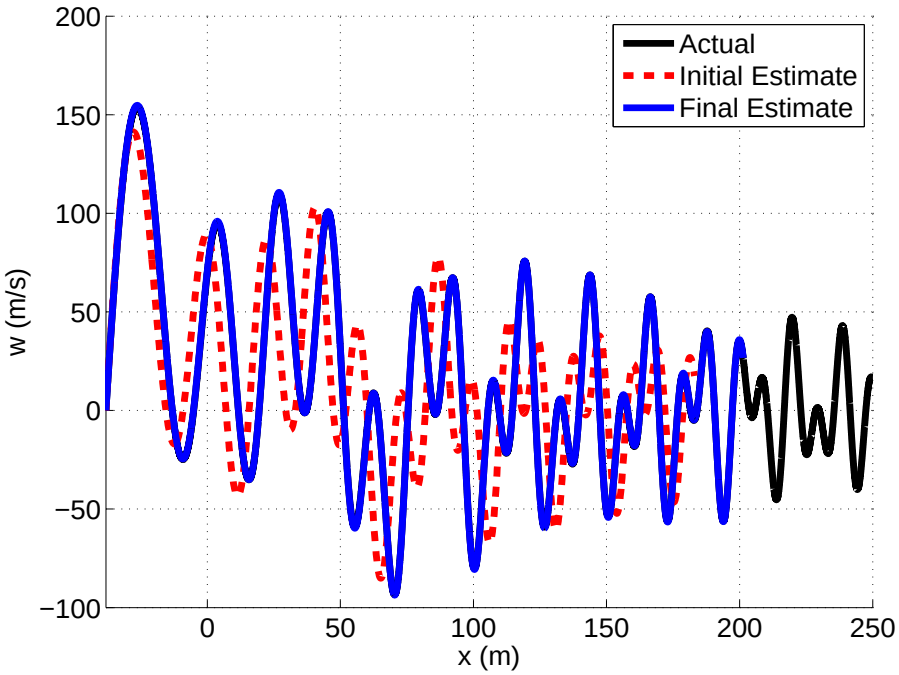


Figure 6.20: Simulated High Angle of Attack Body Z Velocity vs. Range

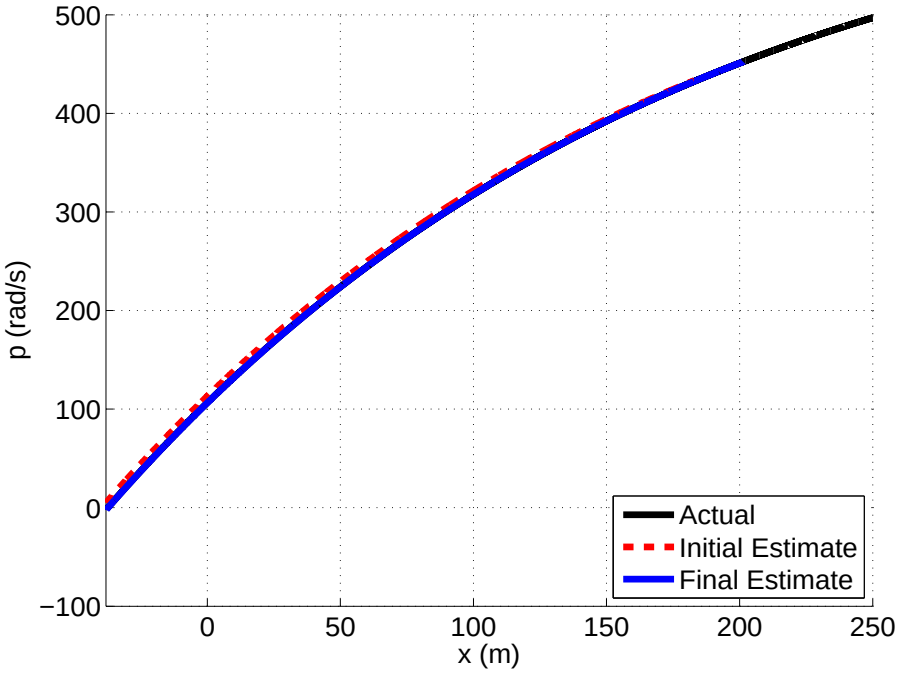


Figure 6.21: Simulated High Angle of Attack Roll Rate vs. Range

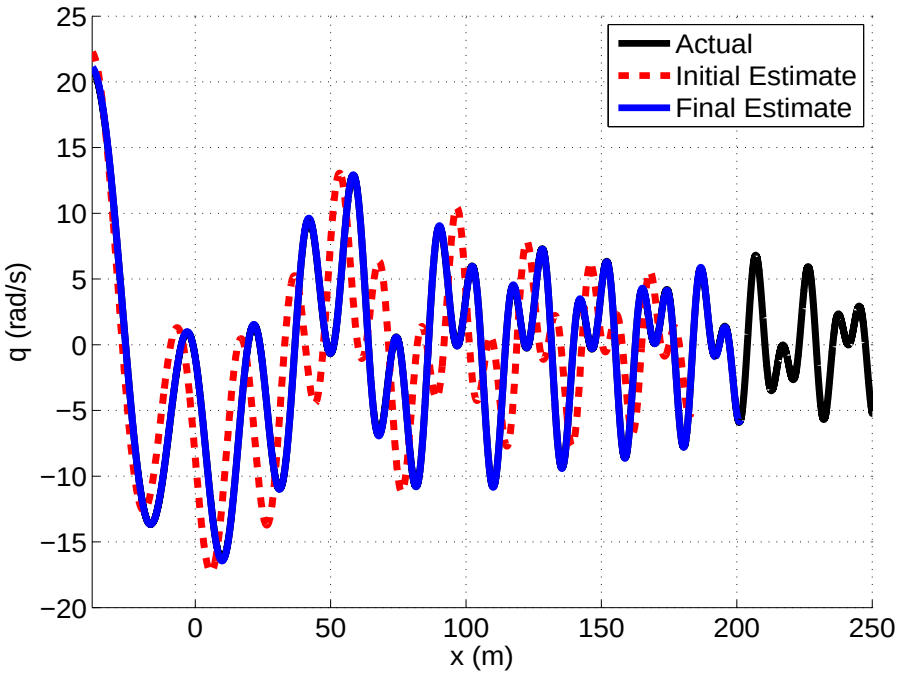


Figure 6.22: Simulated High Angle of Attack Pitch Rate vs. Range

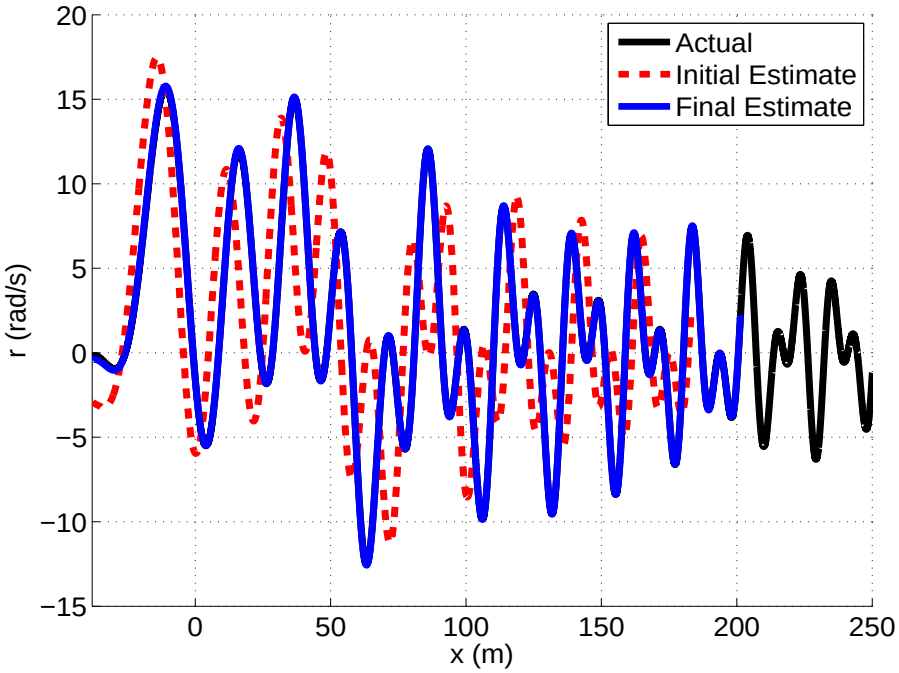


Figure 6.23: Simulated High Angle of Attack Yaw Rate vs. Range

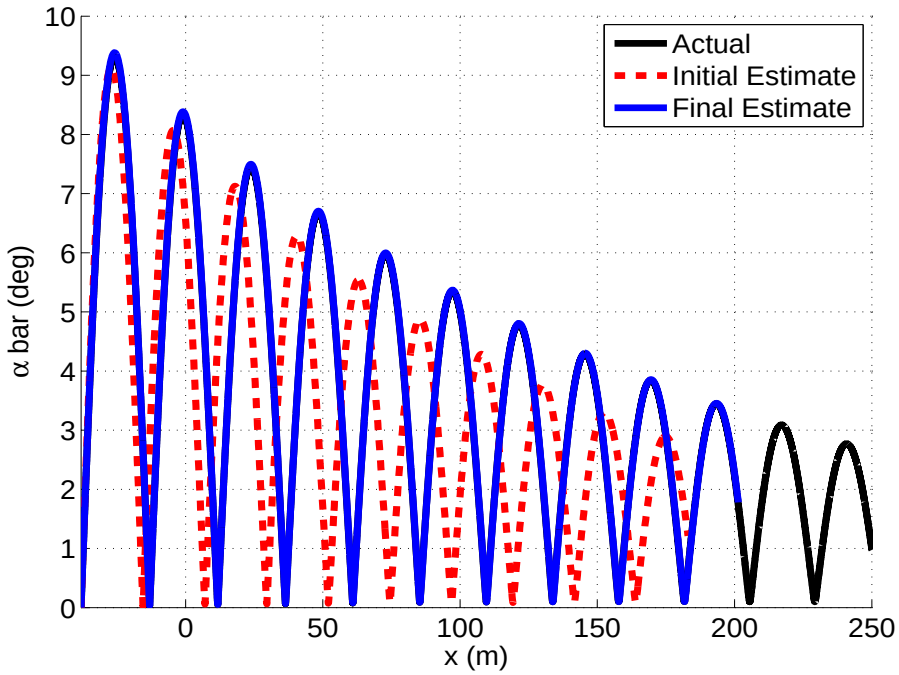


Figure 6.24: Simulated High Angle of Attack Total Angle of Attack vs. Range

The cost reduction profile for this case as seen in Figure 6.25 follows a similar path as the low angle of attack case. The meta-optimizer achieves large cost reductions over the first 200,000 function calls and crosses the cost threshold at around 300,000 function calls. After this point, the meta-optimizer continues to gradually improve the solution for another 1,000,000 function calls. Figure 6.26 shows a similar trend as the low angle of attack case with a few optimizers providing the majority of the cost reductions. On this trial, SD, CG, PSO, and DE all contribute equally with small contributions from BFGS, SIM, and TS. The number of times each optimizer was deployed (in Figure 6.27) is more distributed between the optimizers with BFGS called the most and CG called the least. One reason BFGS was called the most is that it was very efficient near the solution where the cost reductions are much lower. CG on the other hand was called on the first two iterations, but was not selected again for another 30 iterations. Finally, the total number of function calls for each optimizer is shown in Figure 6.28. Six of the optimizer all used roughly equal numbers of function calls with the other three using many fewer. Like BFGS, SIM was efficient close to the solution, allowing it to run for longer than other optimizers. TS, on the other hand, was generally ineffective in this range, but required a number of function calls to reach its stopping criteria.

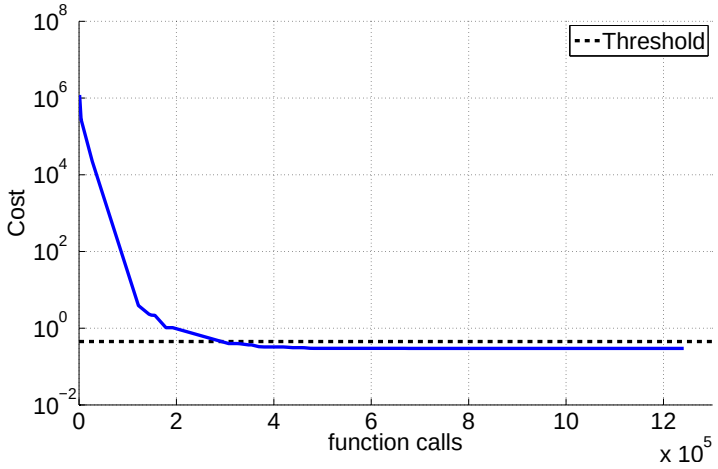


Figure 6.25: Simulated High Angle of Attack Cost Profile

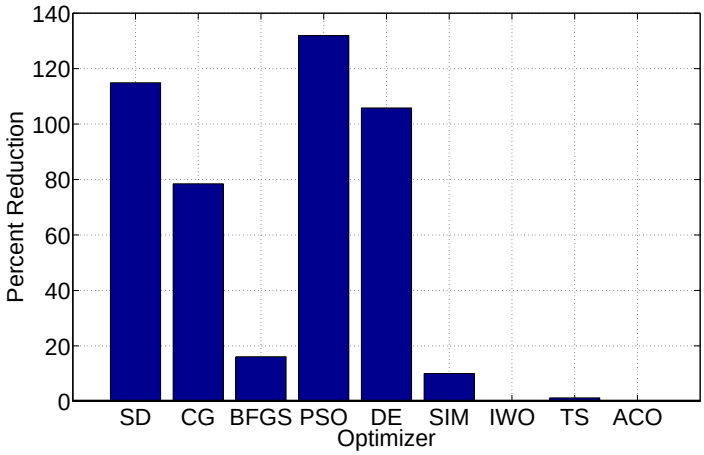


Figure 6.26: Simulated High Angle of Attack Total Percent Cost Reduction

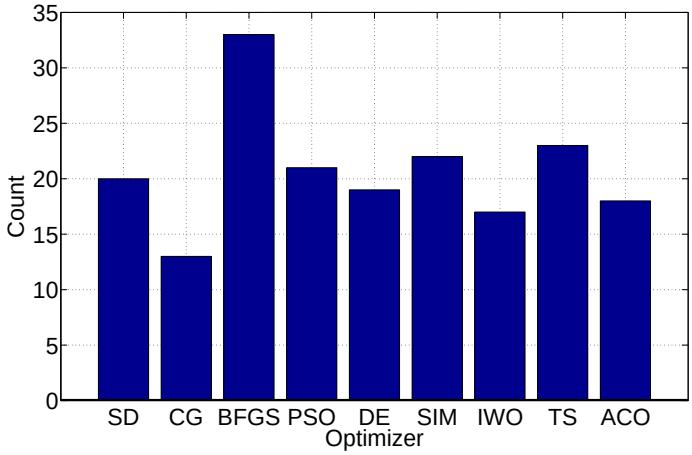


Figure 6.27: Simulated High Angle of Attack Total Number of Calls of Each Optimizer

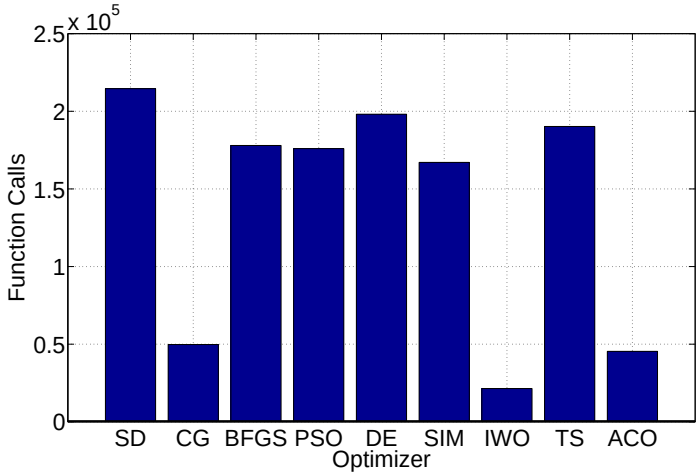


Figure 6.28: Simulated High Angle of Attack Total Function Calls Used by Each Optimizer

The convergence profile for each of the parameters is given in Figures 6.29-6.32. The first parameter to converge was $C_{m\alpha}$ which reached its actual value in about 150,000 function calls. $C_{m\alpha}$ is one of the most important aerodynamic coefficients as it controls the angular behavior and angle of attack of the projectile, making the cost highly sensitive to variations in this coefficient. C_{mq} requires about 600,000 function calls to converge while C_{X2} and $C_{N\alpha}$ continue to move until the meta-optimizer stops. For all four coefficients, at least one of the estimates exceeded the search bounds at some point. C_{X2}, $C_{m\alpha}$, and C_{mq} all quickly leave the search space, but return after a few iterations. Only $C_{N\alpha}$ at $M = 2.75$ remains near its boundary for more than a few thousand function calls, finally jumping towards the actual value after about 400,000 function calls. This jump corresponds to similar corrections in the other parameters which occurs around the time the cost threshold is crossed in Figure 6.25.

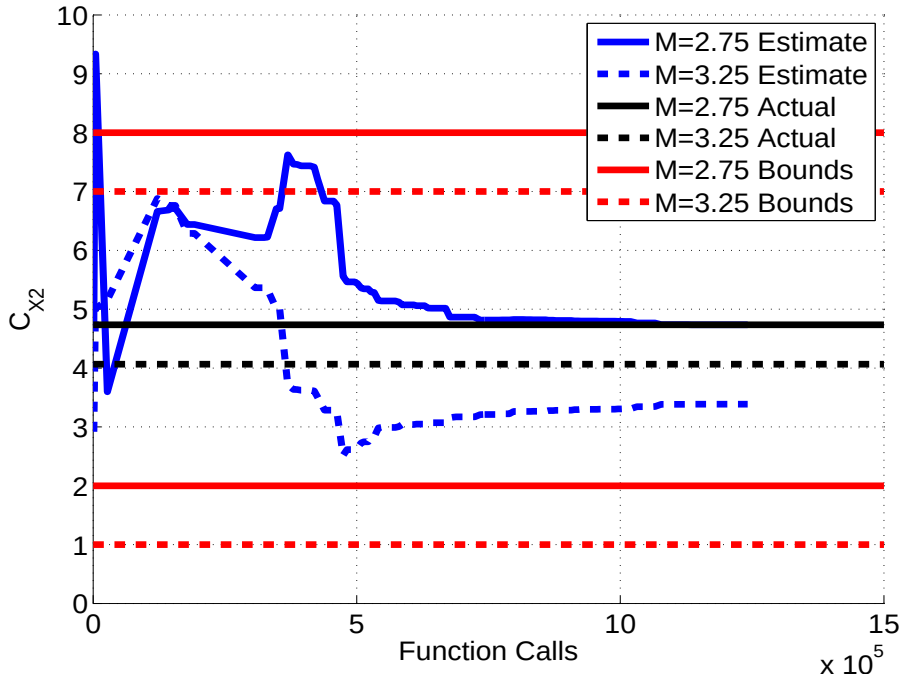


Figure 6.29: Simulated High Angle of Attack Nonlinear Drag Profile

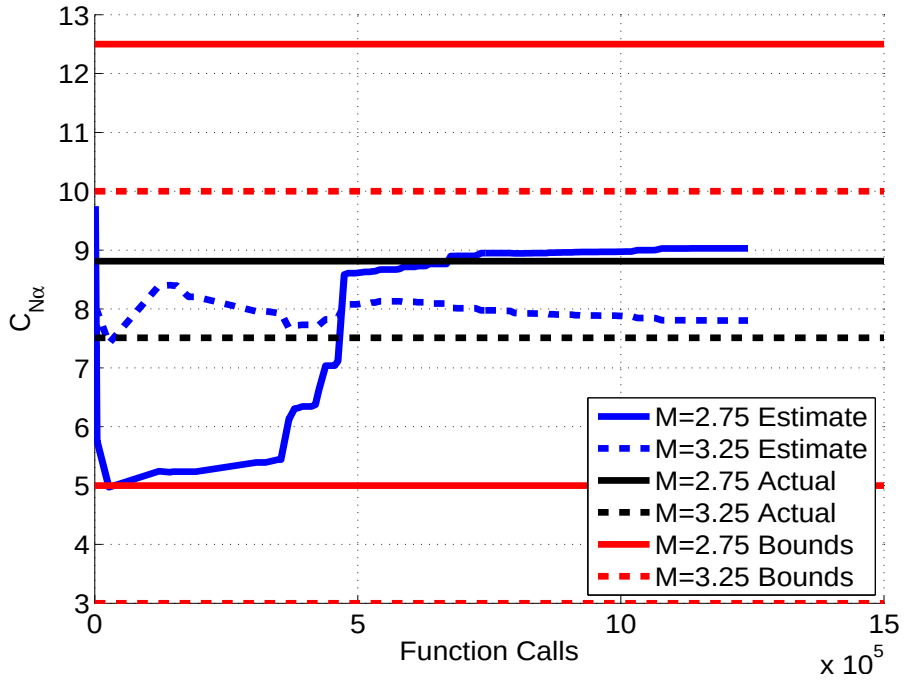


Figure 6.30: Simulated High Angle of Attack Normal Force Profile

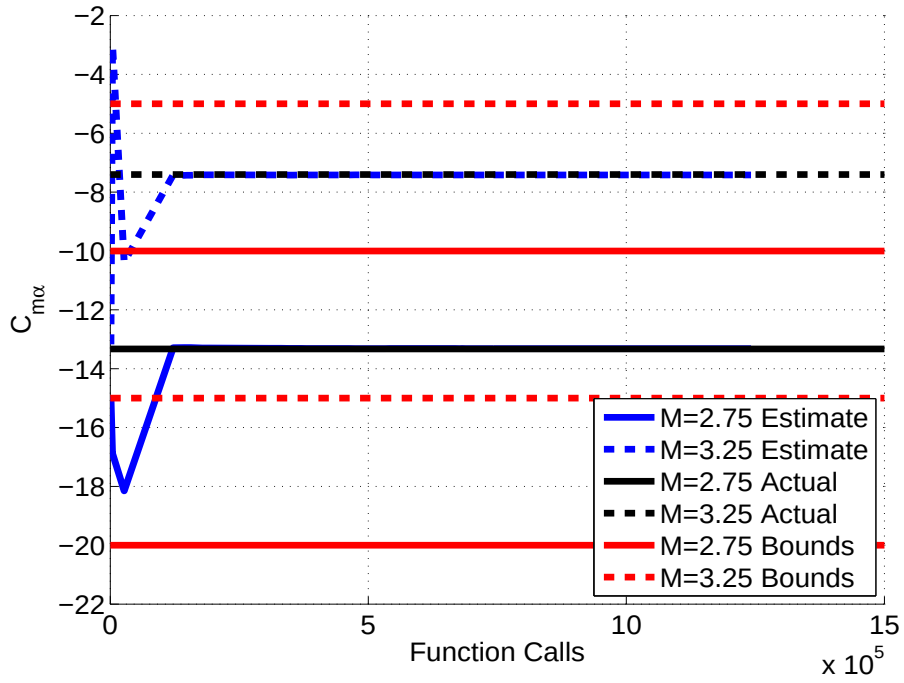


Figure 6.31: Simulated High Angle of Attack Pitching Moment Profile

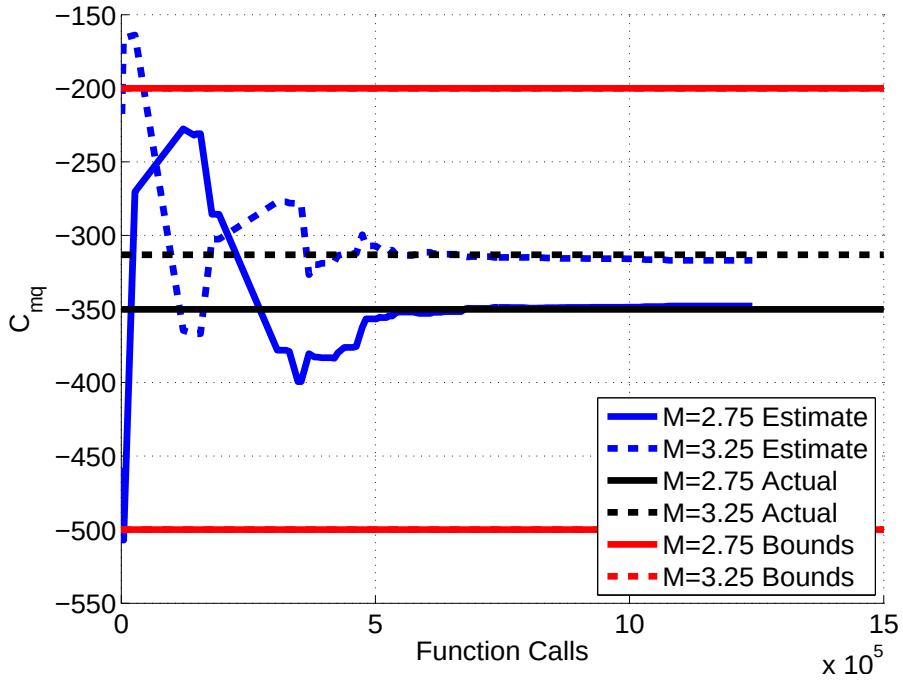


Figure 6.32: Simulated High Angle of Attack Pitch Damping Profile

6.1.3 Simulated Active Microspoiler Results

Lastly, the microspoiler parameters are estimated with the the projectile body aerodynamic coefficients fixed to their estimated values. These parameters include the the axial force coefficient δ_a, the normal force coefficient δ_N, and the pitching moment coefficient δ_m as well as the microspoiler mechanism initial phase ω_0, spin rate Ω_0, and time constant τ_{ms}. The initial phase, spin rate, and time constant are estimated for every trajectory. Unlike the previous cases, the trajectory prediction simulations in the cost function were started from the first measurement. This was done to limit the interactions between the initial pitch rate at launch and the microspoiler pitching moment. As seen in Section 2.4.2, the microspoilers produce a similar effect at launch as a large angular velocity perturbation. Under some conditions, changes in the estimates of the microspoiler parameters may be matched by changes in the estimated pitch rate, yielding minimal changes in the overall trajectory. To handle fitting the data in this manner, a modification is needed to the microspoiler model.

When starting the simulation at the first measurement, the microspoiler mechanism will already be spinning at some rate based on the time since launch. The time of launch can therefore be added to the model such that the spin rate is given by:

$$\Omega(t) = \Omega_0(1 - e^{-\frac{(t-t_{initial})}{\tau_{ms}}}) \tag{6.1}$$

and the phase is given by:

$$\omega(t) = \omega_0 + \Omega_0((t - t_{initial}) + \tau_{ms}e^{-\frac{(t-t_{initial})}{\tau_{ms}}} - \tau_{ms}) \tag{6.2}$$

with the initial time estimated based on a least squares fit of the x and t data.

As with the high angle of attack case, five trajectories were used to estimate the microspoiler parameters. Again, all six states were included in the cost function with equal weighting. By beginning the trajectory prediction simulations at the first measurement, estimates are needed for the initial conditions of every state resulting in 81 parameters to estimate. The test results for this case are given in Tables 6.6 and 6.7 and the final χ^2 values in Table 6.8

Table 6.6: Simulated Microspoiler Coefficient Parameter Estimation Results

	M = 2.75			**M = 3.25**		
Parameter	**Actual**	**Estimate**	**% Error**	**Actual**	**Estimate**	**% Error**
δ_A (N)	-26.325	-27.046	2.544	-31.625	-33.495	5.914
δ_N (N)	67.225	84.99	26.44	80.175	73.78	7.979
δ_m (Nm)	8.4175	8.341	0.9096	10.0825	10.249	1.648

Table 6.7: Simulated Microspoiler Mechanism Parameter Estimation Results

Parameter	**Actual**	**Mean Estimate**	**STD Estimate**	**% Error**
Ω_0 (rad/s)	440.0	440.44	0.5569	0.101
τ_{ms} (1/s)	0.025	0.026	0.0023	4.051

Table 6.8: χ^2 Values for Simulated Active Microspoiler Trajectories

	x	y	z	ϕ	θ	ψ
Average	23.87	18.12	22.88	21.87	24.91	23.43
Standard Deviation	11.19	3.575	8.074	6.973	2.546	9.551

Overall, the parameter estimation algorithm is able to accurately estimate the microspoiler parameters with only small errors in δ_A and larger errors δ_N. These errors may be due to the small errors in C_{X2} and $C_{N\alpha}$ previously estimated. Also, like their projectile body counterparts, these parameters are also difficult to estimate as the cost function is not sensitive to errors in these parameters. There may also be some coupling between the initial conditions and the microspoiler parameters which induce additional errors in the estimates. The estimation algorithm is also successful in estimating the spin rate and time constant for each trajectory which have more indirect effects on the projectile dynamics. The χ^2 values indicate excellent fits of all states.

The trajectory results for one of the trajectories used by the parameter estimation algorithm are given in Figures 6.33-6.45. Looking at the trajectories for the measured states in Figures 6.33-6.38, the estimation algorithm does an excellent job at fitting a trajectory to the available data as indicated by χ^2. Especially for y and h, there is little disagreement between the fit trajectory and the measurements. Considering how well the estimation algorithm fit these states, the large errors in δ_N could not have played a major role in obtaining this fit. Figures 6.37 and 6.38 show excellent fitting of θ and ψ which allows for accurate estimation of δ_m. Looking at v and w in Figures 6.40 and 6.41 respectively, there are some slight differences between the final trajectory and the actual trajectory. On v, there is an error in the initial estimate of about 2 m/s with a similar error on w_0. These errors are about 10% of the actual values and may be an indication of trade-offs between the initial velocities and δ_N.

Figures 6.43 and 6.44 also show small differences between the q and r trajectories. While the estimate for δ_m is very accurate, these differences may be due to the errors in τ_{ms} or slight errors in Ω_0 and ω_0. However, these effects do not appear to influence the θ and ψ fits, meaning the errors may be primarily due to the measurement noise.

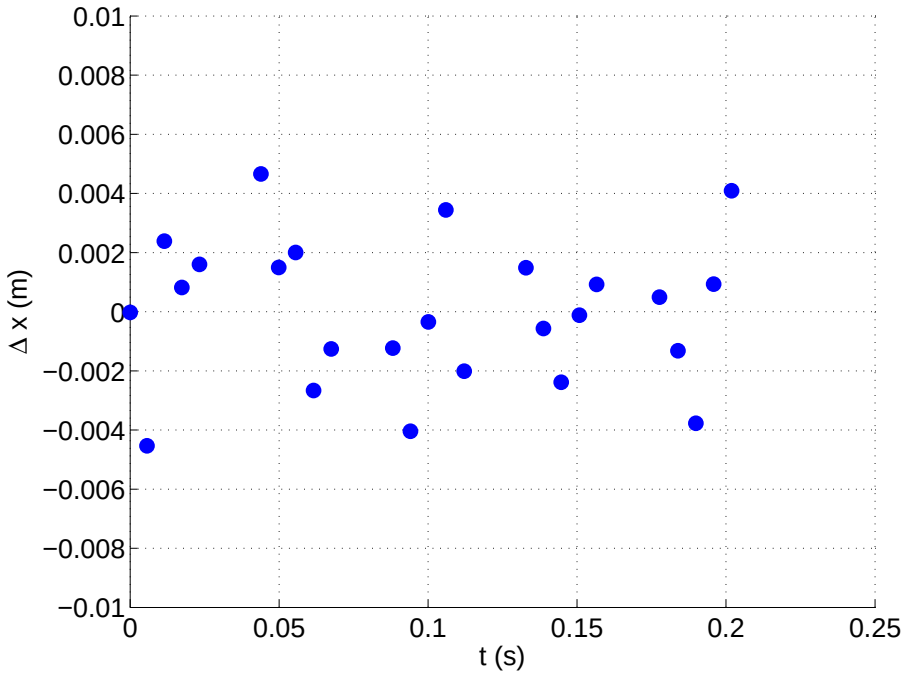


Figure 6.33: Simulated Active Microspoilers X Error vs. Time

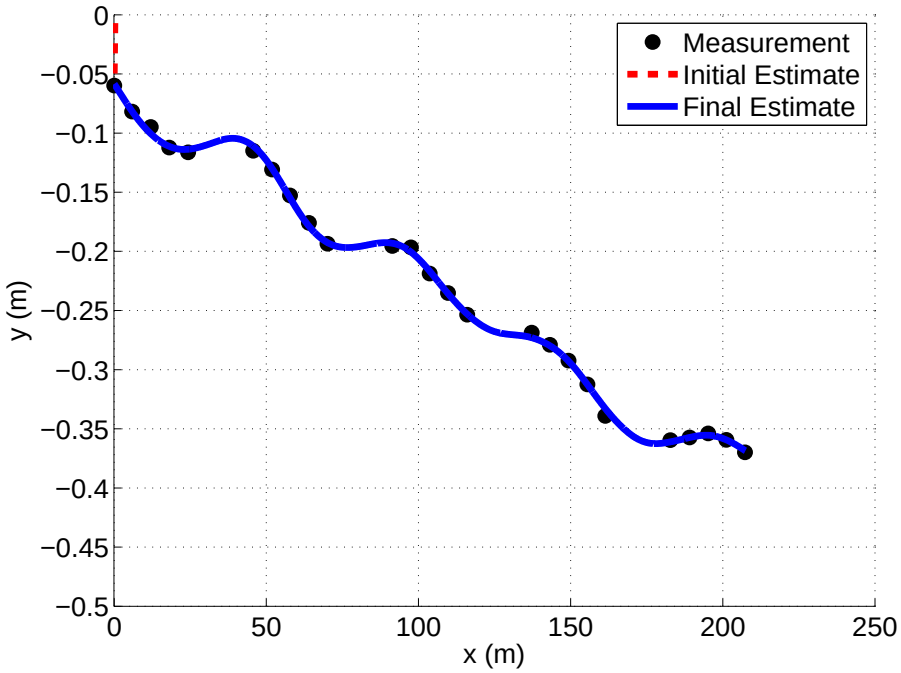


Figure 6.34: Simulated Active Microspoilers Inertial-Y Position vs. Range

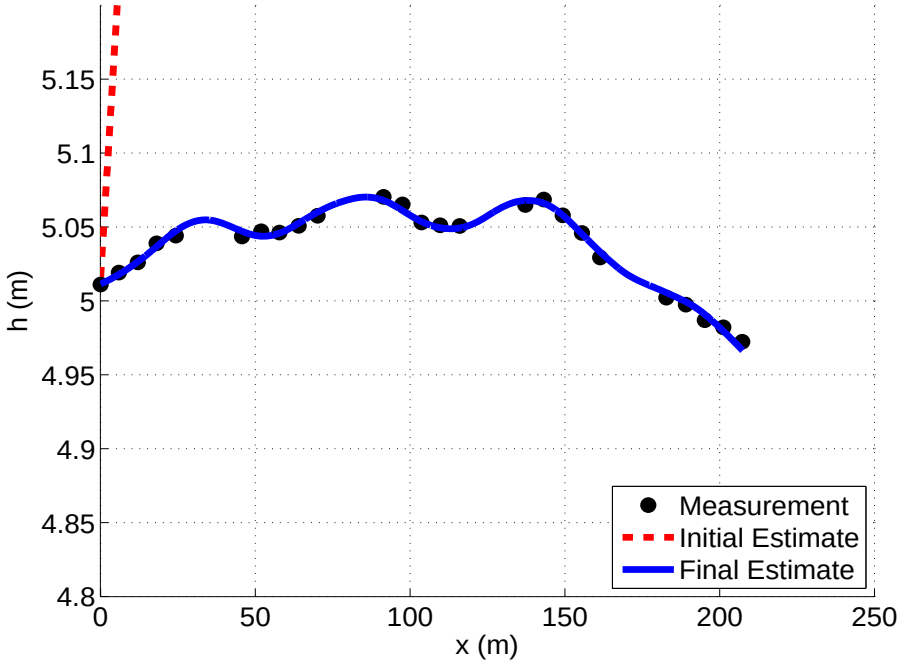


Figure 6.35: Simulated Active Microspoilers Altitude vs. Range

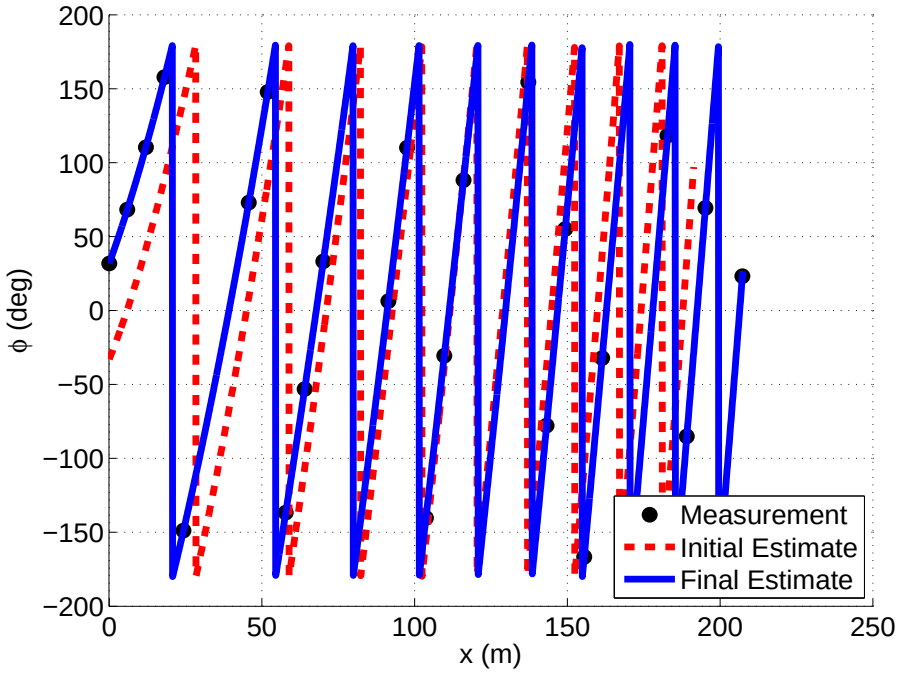


Figure 6.36: Simulated Active Microspoilers Roll Angle vs. Range

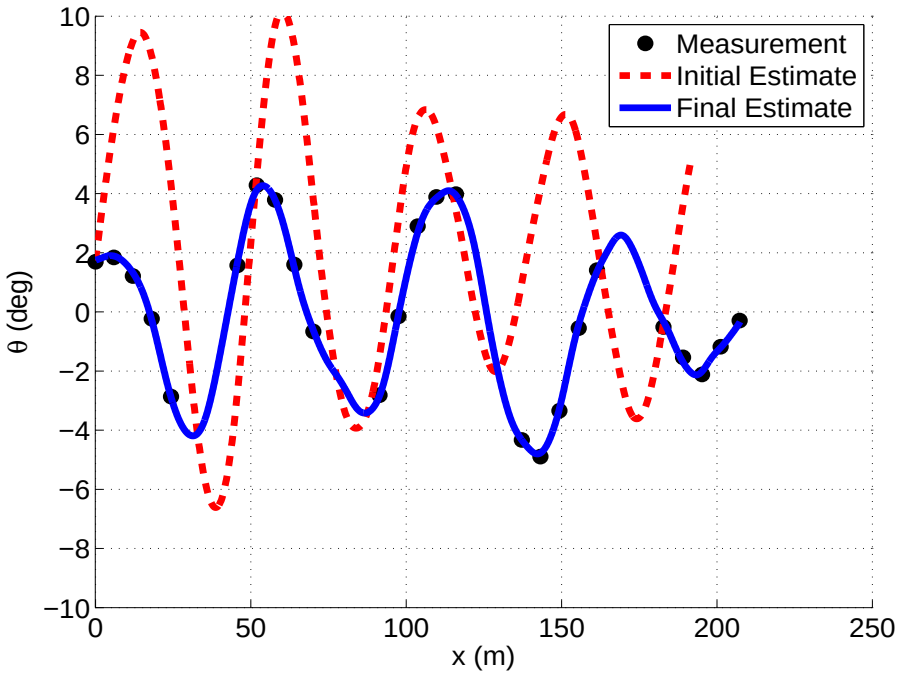


Figure 6.37: Simulated Active Microspoilers Pitch Angle vs. Range

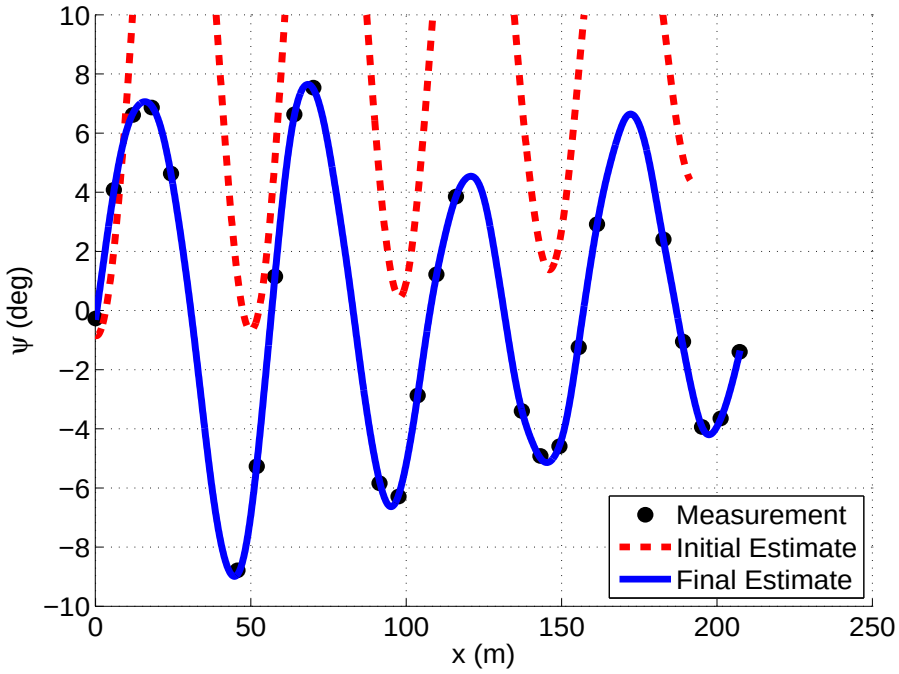


Figure 6.38: Simulated Active Microspoilers Yaw Angle vs. Range

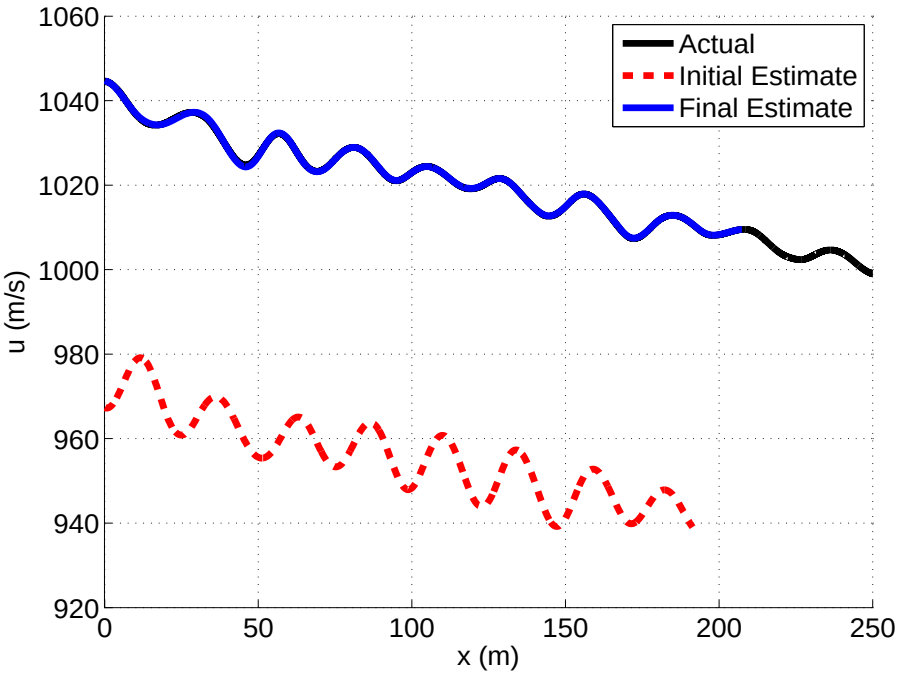


Figure 6.39: Simulated Active Microspoilers Body X Velocity vs. Range

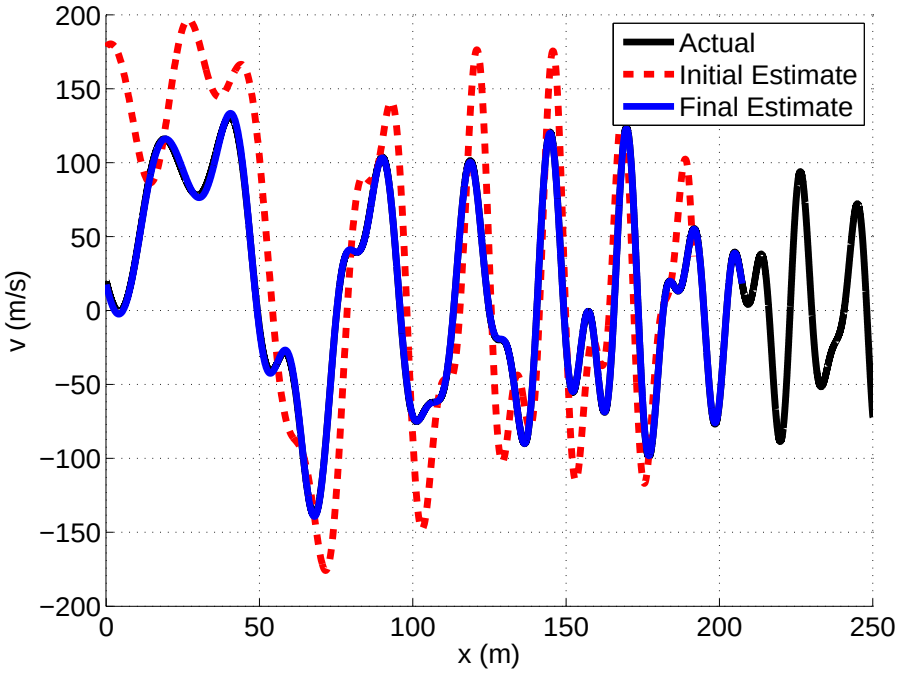


Figure 6.40: Simulated Active Microspoilers Body Y Velocity vs. Range

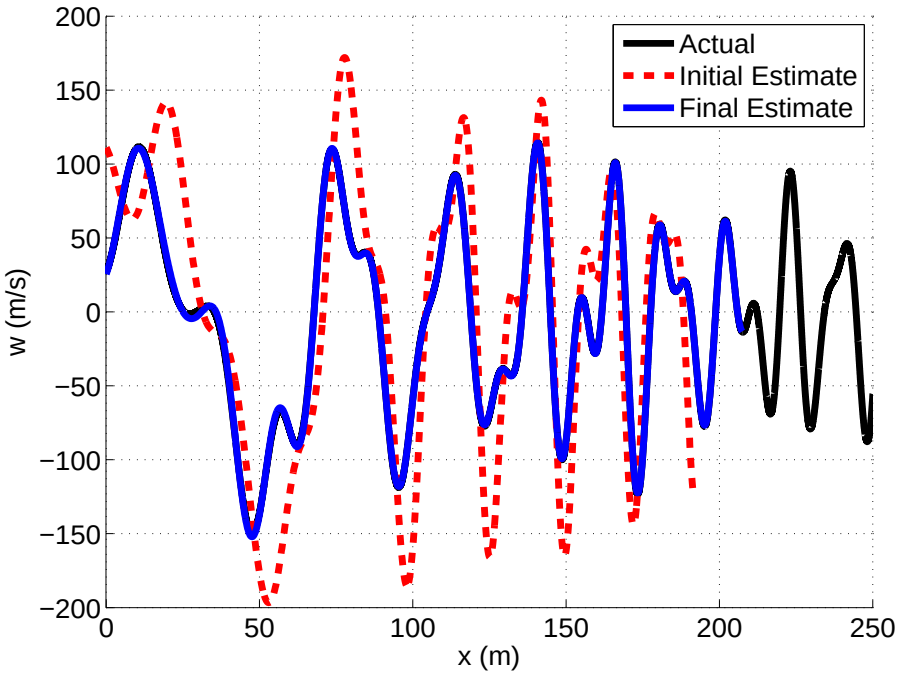


Figure 6.41: Simulated Active Microspoilers Body Z Velocity vs. Range

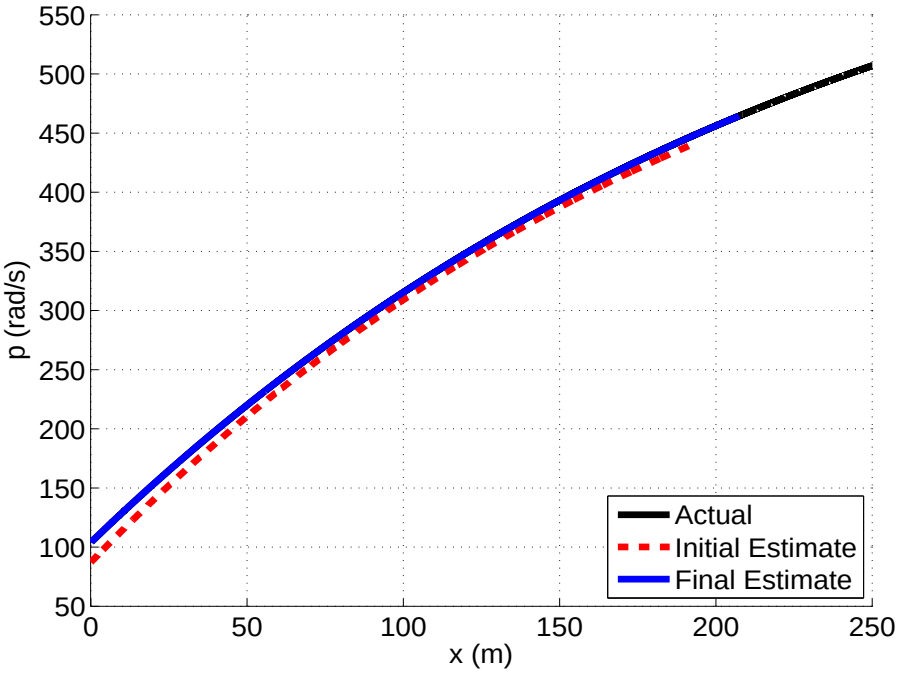


Figure 6.42: Simulated Active Microspoilers Roll Rate vs. Range

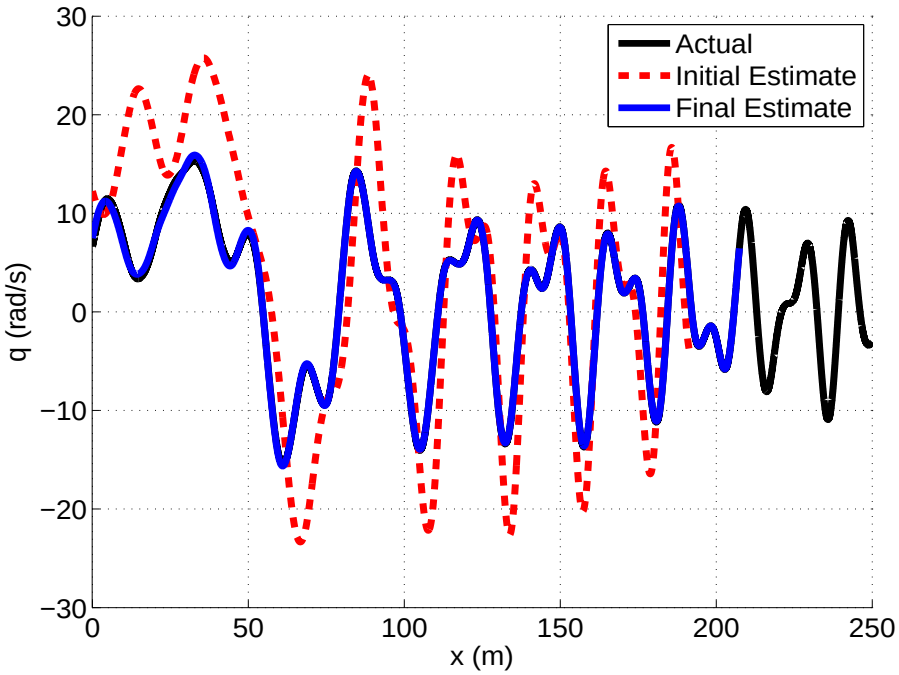


Figure 6.43: Simulated Active Microspoilers Pitch Rate vs. Range

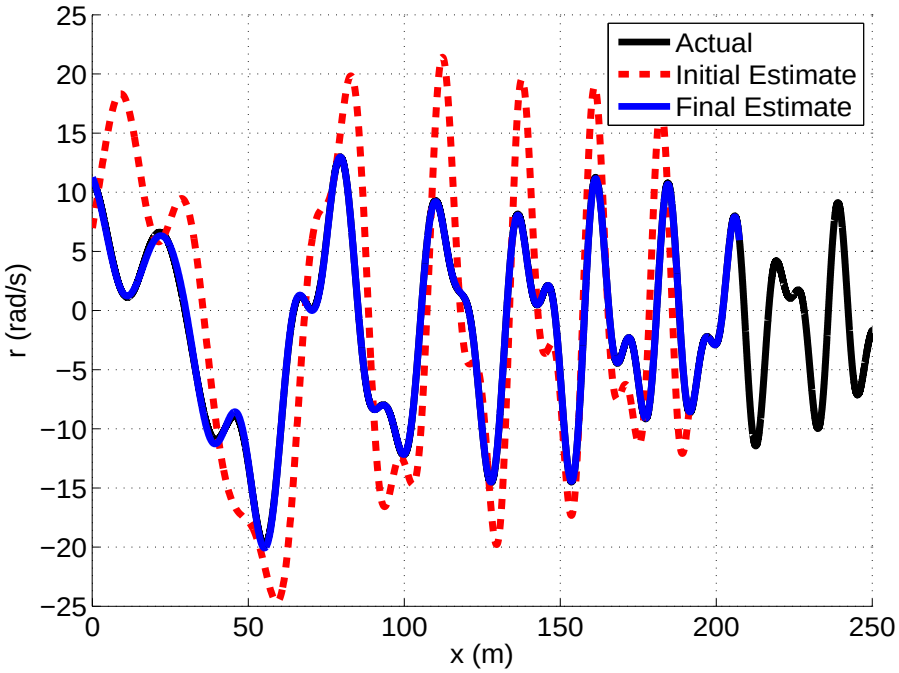


Figure 6.44: Simulated Active Microspoilers Yaw Rate vs. Range

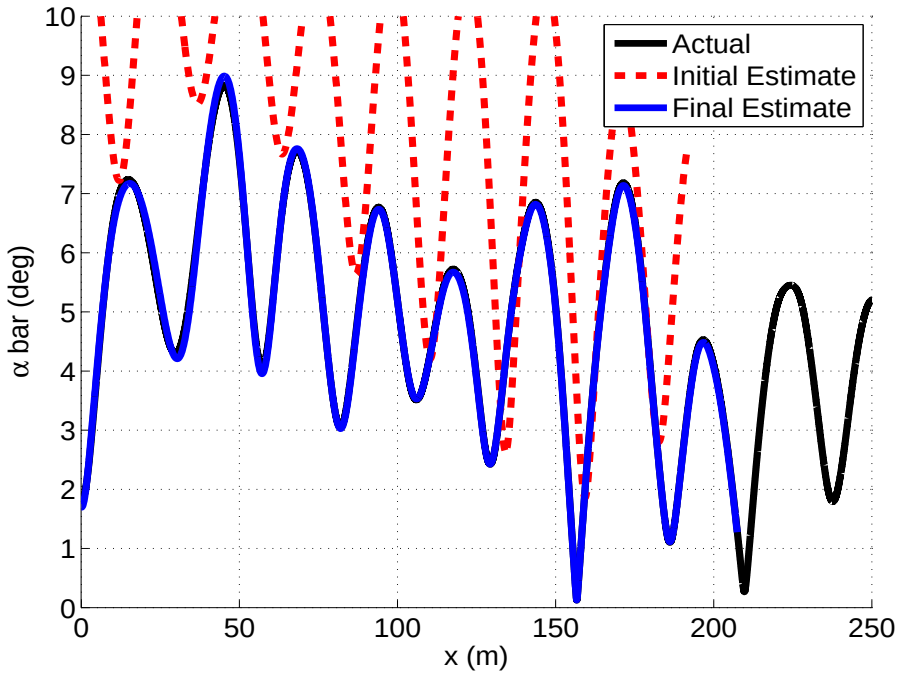


Figure 6.45: Simulated Active Microspoilers Total Angle of Attack vs. Range

Unlike the previous cases, the cost profile in Figure 6.46 shows the meta-optimizer stalling for a long period of time. After a large initial reduction over the first 200,000 function calls, and some smaller reductions over the next 500,000 function calls, progress stalls almost completely after about 800,000 function calls. It takes the meta-optimizer over 800,000 more function calls to get going again when SIM is able to make significant progress. After crossing the threshold at this time, the meta-optimizer continues making small improvements over another 1.5 million function calls. From Figure 6.47, CG, BFGS, and SIM dominate the cost reductions with small amounts from SD, DE, and TS. While effective on the high angle of attack case, PSO and DE were not given many good opportunities to make progress on this trial. The total counts for each optimizer in Figure 6.48 shows a mostly equal distribution in counts with some preference towards CG and BFGS which performed well. SIM used the most function calls as seem in Figure 6.49 with all but DE, IWO, and ACO using more than 200,000.

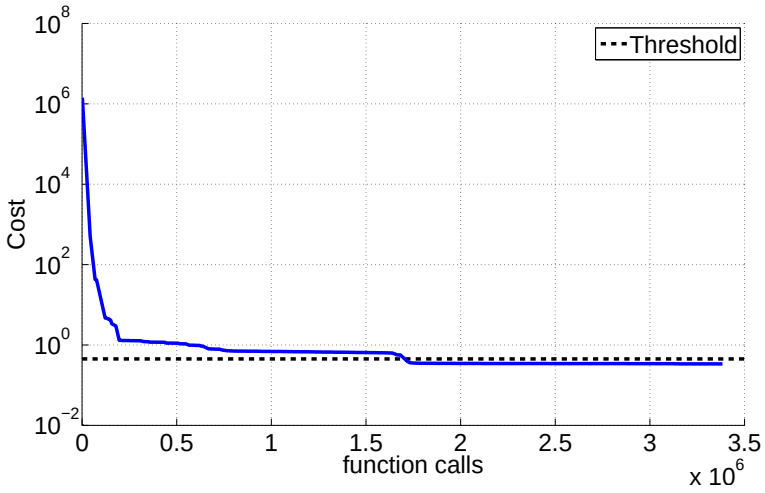


Figure 6.46: Simulated Active Microspoilers Cost Profile

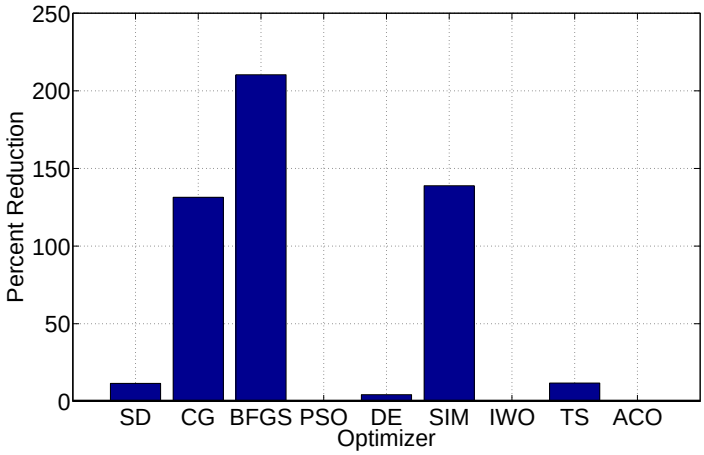


Figure 6.47: Simulated Active Microspoilers Total Percent Cost Reduction

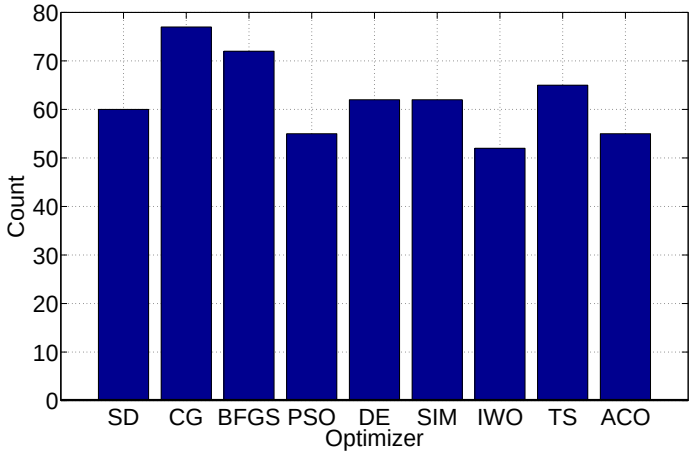


Figure 6.48: Simulated Active Microspoilers Total Number of Calls of Each Optimizer

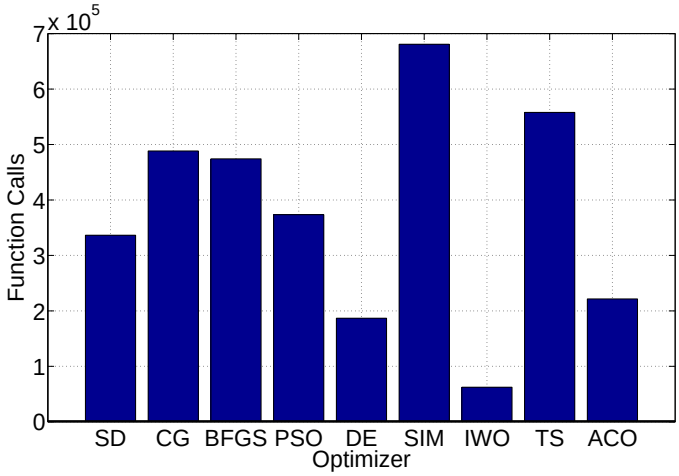


Figure 6.49: Simulated Active Microspoilers Total Function Calls Used by Each Optimizer

Profiles for each of the microspoiler parameters are given in Figures 6.50-6.54. The first parameters to converge are the spin rates for each trajectory with trajectories 2 and 3 the first to settle, followed by 1 and 4 after 800,000 function calls, and finally trajectory 5 at around 1.6 million function calls, corresponding to the final jump in cost in Figure 6.46. All of the parameters make large changes at this time, with δ_A and δ_m moving very close to their actual values. The $M = 2.75$ estimate of δ_N remains far from its actual value for the entire process, remaining on the search boundary of 90 for a long time. Ω_0 for trajectory 1 exceeds the boundary early on, but quickly returns to the search space. At least one parameter for each of the coefficients also briefly reach the boundary. Finally, τ_{ms} varies for each trajectory with all of the final values within 15% of the actual values. When the simulation is started from the first measurement, the microspoilers are already spinning at about 75% of the max speed by the time the simulation starts, making the impact of τ_{ms} on the trajectory less than if the simulation began at launch.

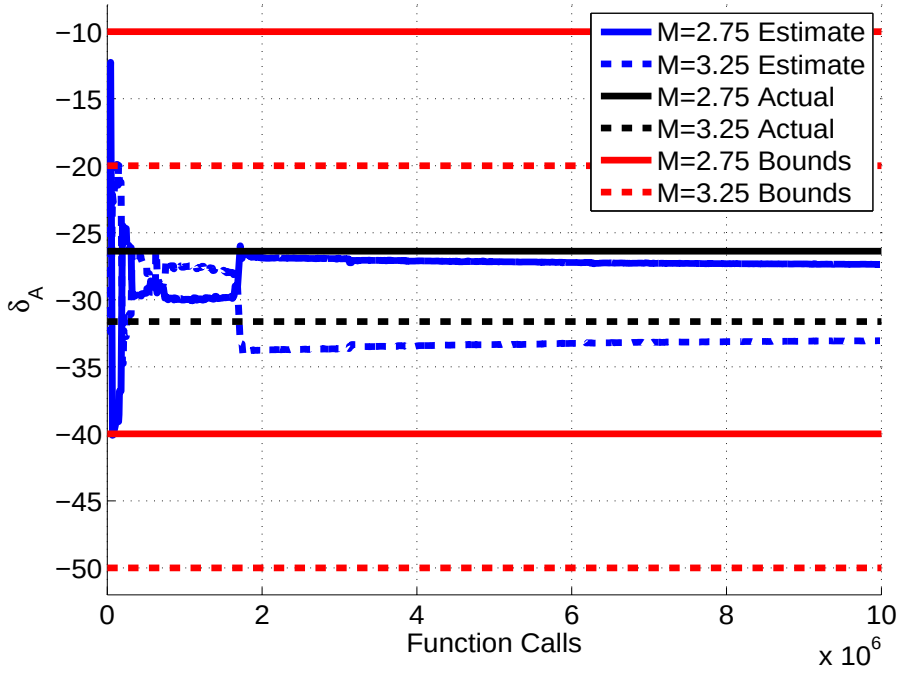


Figure 6.50: Simulated Active Microspoilers Axial Force Profile

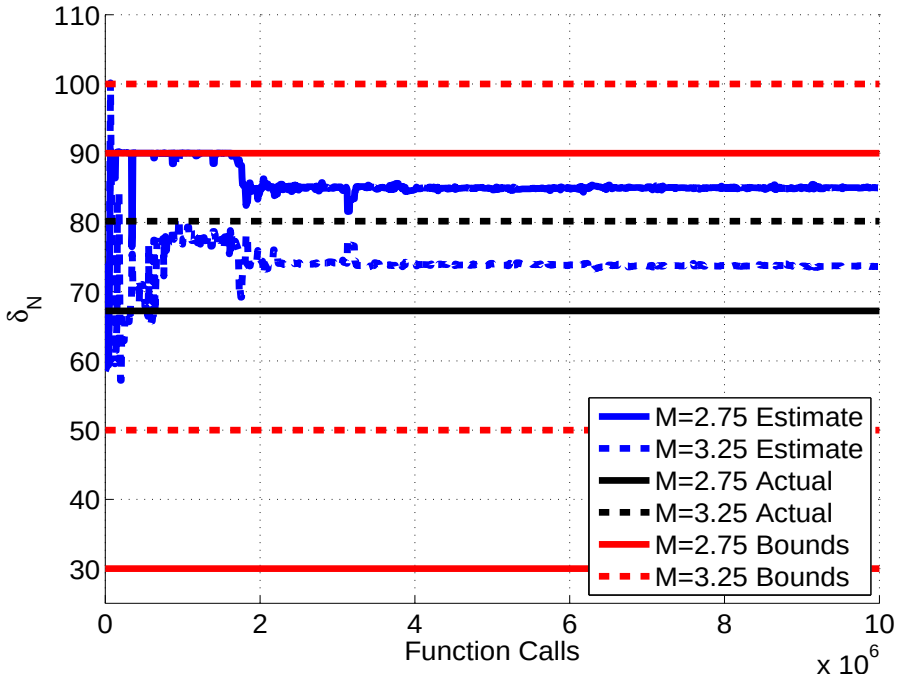


Figure 6.51: Simulated Active Microspoilers Normal Force Profile

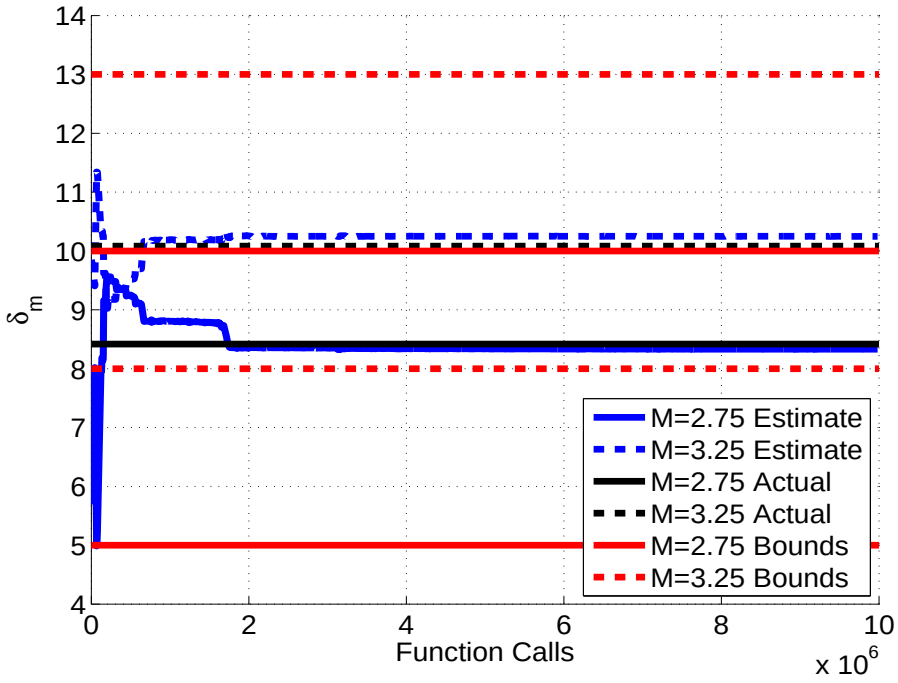


Figure 6.52: Simulated Active Microspoilers Pitching Moment Profile

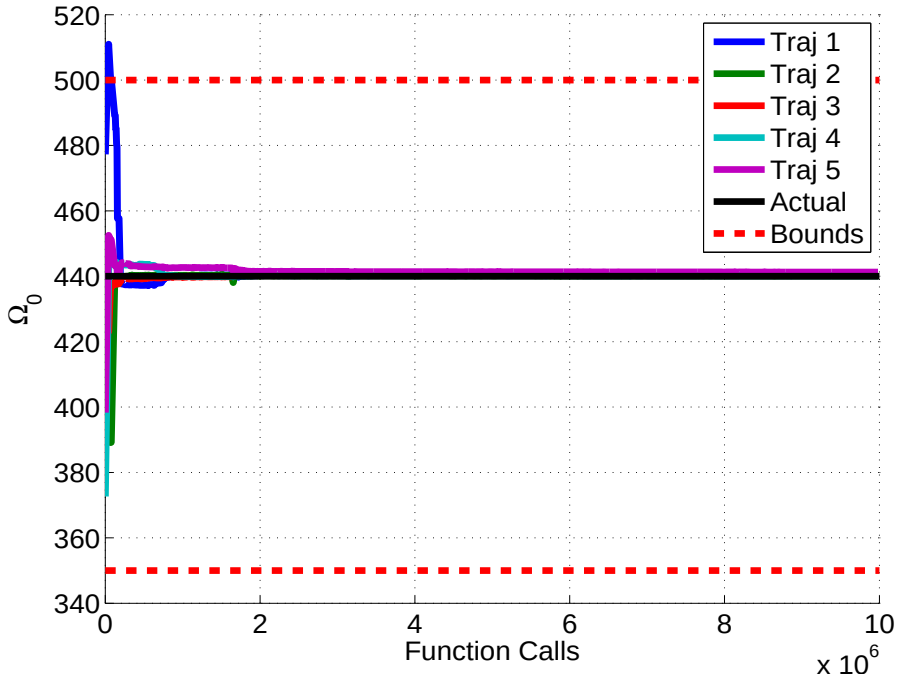


Figure 6.53: Simulated Active Microspoilers Nominal Spin Rate Profile

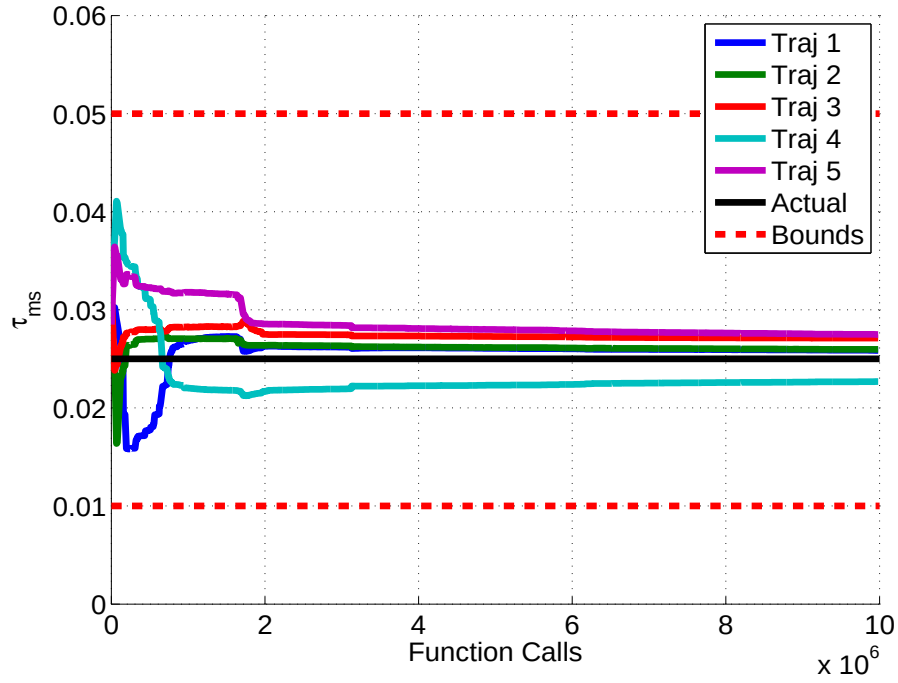


Figure 6.54: Simulated Active Microspoilers Time Constant Profile

6.1.4 Individual Optimizer Comparison

It is also useful to examine a comparison between the performance of the parameter estimation algorithm using both the standard individual optimizers and the new method meta-optimization. The simulated high angle of attack case is used for this test. Here, each optimizer was started from the same initial point and with the same set of points. The optimizers then run for 1,000,000 function calls or satisfy their stopping criteria. The final cost for each of the optimizers is given in Table 6.9. The cost threshold for this problem was 0.45 which DE, ACO, and meta-optimization reach. PSO performs very poorly, indicating poor tuning of its parameters for this problem. All of the hill climbers reach a local minima far from the solution with SIM stalling as well.

Table 6.9: Final Cost for Individual Optimizers

SD	CG	BFGS	PSO	DE	SIM	IWO	TS	ACO	MO
135.66	24.021	119.43	2538.4	0.3987	970.21	236.39	83.839	0.4281	0.2975

The cost reduction profile for each optimizer is shown in Figure 6.55. In this case, meta-optimization is the first optimizer to cross the cost threshold and the only optimizer to reach the solution in the alloted time. However, almost all of the other optimizers are faster at initially reducing the cost with SIM particularly efficient. However, it quickly plateaus after only 9000 function calls. The hill climbers also quickly reach local minima and stop there. PSO, IWO, and TS all gradually reduce the cost, but do not come close to the solution. Of the two remaining optimizers, ACO is very efficient over the first 50,000 function calls, but then slows considerably as it approaches the solution. DE takes a more gradual path, also slowing down as the cost decreases. The meta-optimizer, on the other hand, is slightly less efficient over the initial phase, but rapidly reaches the solution in under 200,000 function calls.

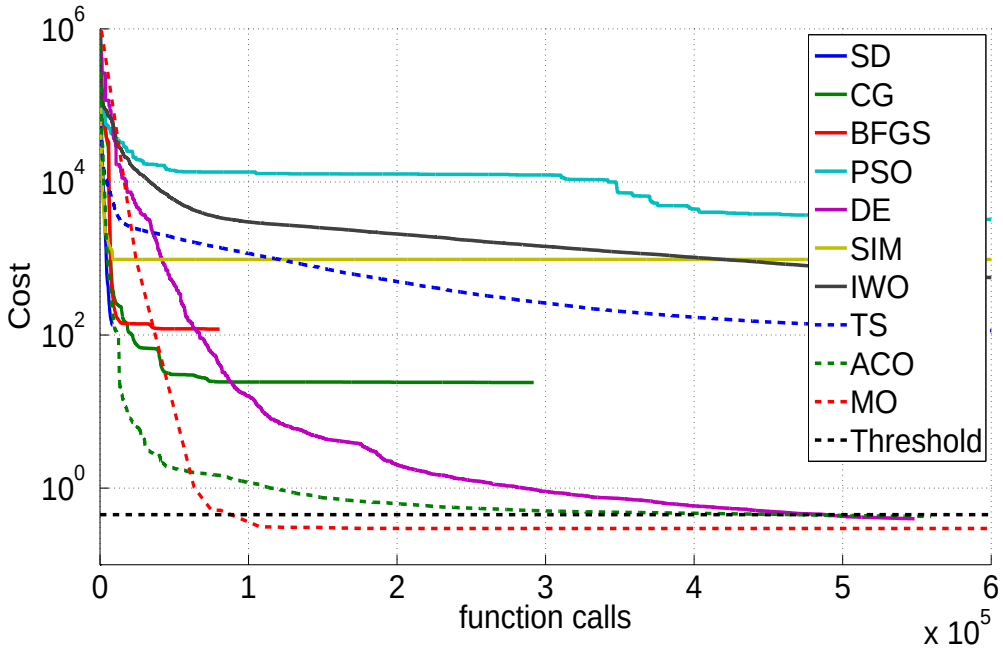


Figure 6.55: Comparison of Optimizer Performance on Simulated High Angle of Attack Problem

6.2 Projectile Parameter Estimation Trade Studies

Various trade studies were performed to explore the nature of the projectile parameter estimation problem using different data sets and estimation approaches. All of these trade studies use simulated measurement data including some of the data used in Section 6.1. The first trade study investigates how the number of measurements and the noise on each measurement impacts the accuracy of the estimates obtained by the parameter estimation algorithm. The second trade study considers how the number of trajectories used by the estimation algorithm impact the accuracy of the parameter estimates. The final trade study evaluates the feasibility of combining different types of trajectories to estimate parameters that may have observability issues when estimated together.

6.2.1 Number of Measurements and Amount of Noise

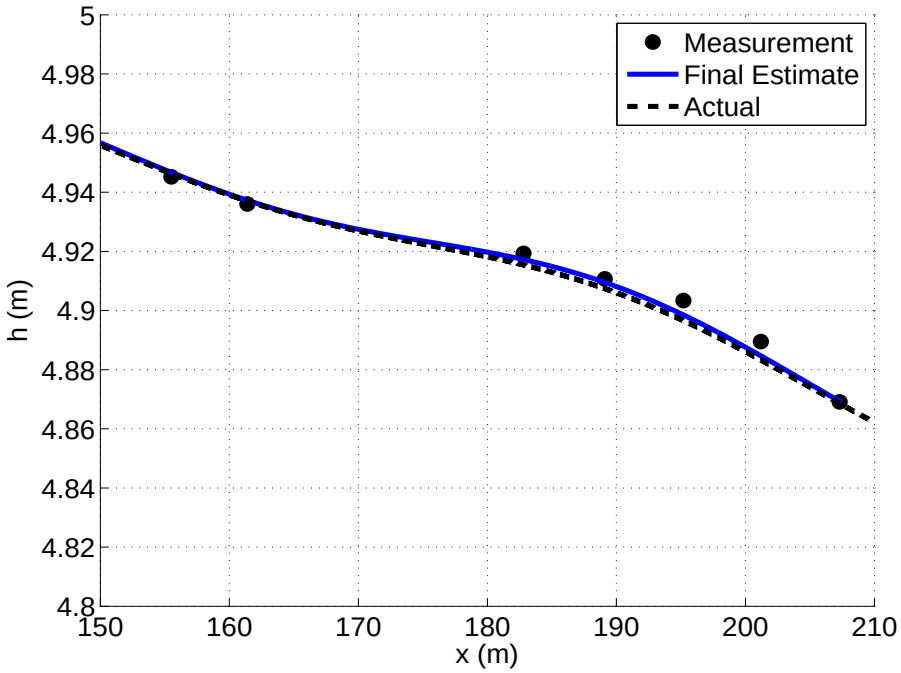
One downside of spark range measurements is the limited number of measurements obtained. Other methods such as onboard sensors and radar systems can greatly increase the number of measurements for each trajectory. Increasing the number of measurements acts similarly to using multiple trajectories by giving the parameter estimation algorithm more data points to use. However, this does not affect the prevalence of local minima that are due to interactions between the parameters and the projectile dynamics. The amount of noise on the data also plays a significant role in the potential accuracy of the parameter estimates. As the noise level increases, the errors in the measurements can lead the estimation algorithm away from the actual trajectory. Also, since the standard deviation of the noise is used to scale the states in the cost function, higher noise leads to the estimation algorithm accepting lower quality fits.

For this trade study, 25, 50, 100, and 200 simulated spark range measurements were considered at varying levels of noise. For the 25 measurement case, measurements were taken at the spark station locations while the measurements for the higher cases were evenly distributed along the spark range. Seven standard deviations were used equal to 0.2, 0.6, 1.0, 2.0, 3.0, 5.0, and 10.0 times the nominal standard deviation of 3 mm for position and 0.1° for angle. The simulated data sets were all based on a single high angle of attack trajectory with constant aerodynamic coefficients. The parameter estimation algorithm was tasked with estimating all seven body aerodynamic coefficients in addition to the unknown initial conditions. Tables 6.10 and 6.11 give the normalized cost per measurement and average normalized percent error for each case. Both the cost and percent error are normalized by the results from the case with 25 measurements and 1.0σ which correspond to the standard spark range conditions.

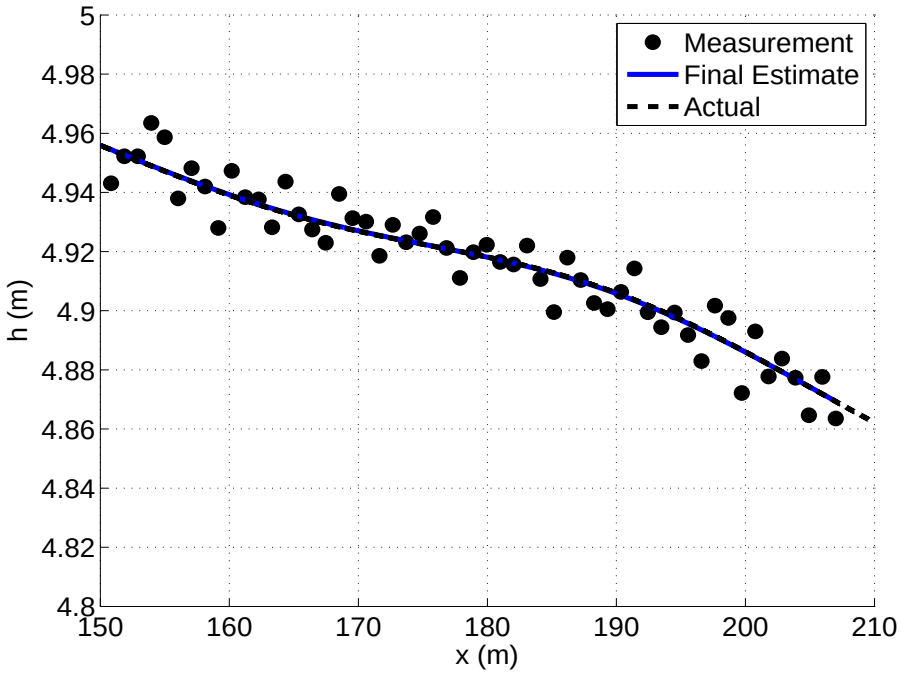
Table 6.10: Normalized Final Cost Per Measurement over Number of Measurements and Amount of Noise

N	**0.2σ**	**0.6σ**	**1.0σ**	**2.0σ**	**3.0σ**	**5.0σ**	**10.0σ**
25	1.4339	1.0945	1.0000	1.0429	1.2311	1.2935	1.3537
50	1.4264	1.1961	1.3426	1.0179	1.2925	1.4413	1.3405
100	1.5811	1.2538	1.5325	1.1985	1.5446	1.4739	1.3800
200	1.5603	1.3207	1.6089	1.1367	1.4153	1.4709	1.4869

Table 6.10 shows a general trend of increasing cost per measurement with increasing number of measurements. One possible cause of this phenomenon is that as the number of measurements increases, the variance in the nominal cost between trajectories decreases. Here, nominal cost refers to the cost corresponding to the actual trajectory compared to the noisy measurements. When there are fewer measurements, it is possible to have a set of measurements with very low or very high noise that is not balanced along the trajectory. With more measurements, the sampled noise will begin to approach a normal distribution which produces a more consistent nominal cost on every trajectory. In addition, the cases with fewer measurements may be prone to overfitting of the data, allowing the parameter estimation algorithm to reach cost values below the nominal cost. This issue is seen in Figures 6.56a and 6.56b which show a comparison between the h fits for the 25 and 200 measurement cases with noise of 2.0σ. For the 25 measurement case, four measurements in the last set all have errors in the same direction, causing the the estimated trajectory to bend towards these measurements. This reduces the cost compared to the actual trajectory, but at the cost of some accuracy in the parameter estimates. On the other hand, the errors in the 200 measurement case are evenly distributed around the trajectory, allowing a fit that is nearly identical to the actual trajectory.



(a)



(b)

Figure 6.56: 2.0σ Altitude Fitting Comparison: (a). 25 Measurements; (b). 200 Measurements

Table 6.11: Measurement Trade Study Average Normalized Percent Error

N	**0.2σ**	**0.6σ**	**1.0σ**	**2.0σ**	**3.0σ**	**5.0σ**	**10.0σ**
25	1.5793	3.1737	1.0000	13.3064	15.3160	26.3583	23.6234
50	0.8114	3.9695	2.3904	11.8376	10.0355	19.0497	32.5358
100	0.4710	1.1573	2.6499	1.3344	11.6892	6.0559	50.2513
200	1.1541	0.6099	2.0343	4.6609	7.2717	9.1068	19.8644

Two trends can be seen in the average percent error in Table 6.11. While the results show large variation in accuracy between cases, N=200 is generally better than $N = 25$ across noise levels. This behavior is expected as more measurements help the parameter estimation algorithm reduce the impact of the noise on the parameter estimates. The second trend is a decrease in accuracy with increasing noise. While the estimation algorithm is still able to obtain accurate estimates at 1.0σ, higher levels of noise show severe degradations in accuracy, with numerous runs struggling to estimate C_{X2} and $C_{N\alpha}$. In these cases, the parameters reach the search boundaries, bounding the percent error. For data sets with large measurement errors, a large number of trajectories or more measurements for each trajectory are needed to obtain accurate parameter estimates.

6.2.2 Number of Trajectories

As seen in Section 6.2.1, measurement noise can play a significant role in the ability of the estimation algorithm to obtain accurate parameter estimates, especially given the limited data from a single spark range test. These estimates can be greatly improved by using multiple trajectories within the cost function. However, every additional trajectory adds computation time in addition to requiring more parameters to estimate. This is because the initial conditions vary between trajectories and must be estimated for each individual trajectory. To test the effect of the number of trajectories on the performance of the estimation algorithm, five cases were considered

with 1, 3, 5, 7, and 10 trajectories. The high angle of attack case from Section 6.1 was used for this trade study with the five trajectory case used as a baseline and normalizing factor.

Table 6.12 shows the normalized number of function calls for each case which are normalized to the case with 5 trajectories. While the computation time can vary significantly between individual runs, there is a significant trend with the computation time increasing with more trajectories. Fewer trajectories to fit means fewer parameters to estimate and fewer data points to fit to simultaneously. This reduces the complexity of the problem, making it easier for the meta-optimizer to find the solution. The final cost per trajectory is given in Table 6.13. The cost per trajectory is computed by dividing the final cost by the number of trajectories. Even though the costs for each case are within about 10% of the baseline with 5 trajectories, there is a clear trend showing a increase in cost with more trajectories. As seen in Section 6.2.1, the parameter estimation algorithm may overfit the data when there are few data points on a single trajectory. When more trajectories are used, the estimation algorithm must balance the cost between all of the trajectories, limiting the likelihood of overfitting any one trajectory. With too many trajectories, the estimation algorithm may sacrifice the accuracy of some trajectories in order to fit others with larger measurement errors.

Table 6.12: Normalized Total Function Calls for Varying Number of Trajectories

1 Traj	3 Traj	5 Traj	7 Traj	10 Traj
0.3078	0.3087	1.0000	1.8259	2.0914

Table 6.13: Final Cost Per Trajectory for Varying Number of Trajectories

1 Traj	3 Traj	5 Traj	7 Traj	10 Traj
0.0543	0.0590	0.0595	0.0636	0.0660

Finally, Table 6.14 gives the percent error in each parameter estimate for each trajectory case. With only one trajectory, the parameter estimation algorithm struggles to fit every parameter except $C_{m\alpha}$. Since Mach number decreases only a small amount over a single trajectory, it is almost impossible to estimate aerodynamic coefficients that vary over a large Mach range. For almost every parameter, increasing the number of trajectories improves the accuracy of the parameter estimates with the largest improvement in $C_{N\alpha}$. The only exception is C_{mq} which has a slightly worse fit with ten trajectories compared to five and seven. This may be due to the inclusion of one or more trajectories with larger errors in the θ and ψ measurements, adding error to the C_{mq} estimate.

Table 6.14: Percent Error in Final Parameter Estimates for Varying Number of Trajectories

Parameter	1 Traj	3 Traj	5 Traj	7 Traj	10 Traj
$C_{X2,lo}$	51.9984	24.8259	0.0139	7.8901	7.8262
$C_{X2,hi}$	72.2102	5.8239	16.6882	22.4277	8.7278
$C_{N\alpha,lo}$	6.1296	19.1535	2.4813	0.4234	0.4553
$C_{N\alpha,hi}$	26.4669	18.3829	3.9111	4.9383	0.0434
$C_{m\alpha,lo}$	0.0164	0.1121	0.0912	0.0412	0.0127
$C_{m\alpha,hi}$	0.2588	0.1508	0.2107	0.1470	0.0602
$C_{mq,lo}$	28.3656	10.5345	0.6424	0.6056	1.4074
$C_{mq,hi}$	59.6908	9.5145	1.2216	0.2940	1.7128

6.2.3 Combining Trajectories

For a projectile system such as the microspoiler projectile considered in this work, the addition of control forces and moments to the estimation problem creates observability issues for both body and control parameters. For example, C_{X0} and δ_A are inherently coupled as errors in one parameter can be offset by changes in the other parameter without a meaningful change in cost. Similar issues can also exist between C_{X0} and C_{X2} when just the projectile body aerodynamics are considered. It is for

these reasons that the build up procedure from Section 6.1 is used to perform parameter estimation where each parameter can be estimated without concerns for these interactions. However, the flexibility of the parameter estimation algorithm may allow for a different approach where trajectories of differing type are combined together to estimate all parameters simultaneously. The trajectories used for this trade study are the same trajectories used in Section 6.1.

The first test investigates the combination of trajectories to estimate the projectile body aerodynamic coefficients. Two combinations are considered, one with five high angle of attack trajectories (H) and one with two low (L) and three high angle of attack trajectories. This combination was chosen to provide a balance between the trajectories while maintaining observability in all of the parameters. All of the projectile body coefficients were estimated except C_{l0} and C_{lp} which were fixed to their estimated values from Table 6.2. This was done because the roll parameters are easily estimated independently and estimates of these parameters are not dependent on the type of trajectory used.

Table 6.15 shows the percent error in the parameter estimates from the actual values for these two combinations. In this table, C_{lo} represents the coefficient estimate at $M - 2.75$ and C_{hi} represents the estimate at $M = 3.25$. The combination with only high angle of attack trajectories had small errors $C_{X0,hi}$ and $C_{N\alpha}$ with larger errors in C_{X2}. With the addition of low angle of attack trajectories, the $C_{X0,hi}$ estimate improved significantly with large improvements in C_{X2} as well. Since the drag force of the low angle of attack trajectories is almost entirely due to C_{X0}, an accurate estimate of C_{X0} is required to fit to these trajectories. This allows the parameter estimation algorithm to more accurately estimate C_{X2} as any interaction between the parameters would increase the cost on only the low angle of attack trajectories. However, the estimates of $C_{m\alpha}$ and C_{mq} get slightly worse because these parameters have low observability on the low angle of attack trajectories. As a

result, the estimation algorithm has fewer measurements to utilized to estimate these parameters.

Table 6.15: Percent Error in Final Parameter Estimate for Two Uncontrolled Trajectory Combinations

Parameter	5H % Error	2L, 3H % Error
$C_{X0,lo}$	0.179	0.1857
$C_{X0,hi}$	3.249	0.06836
$C_{X2,lo}$	13.08	7.699
$C_{X2,hi}$	75.41	10.4
$C_{N\alpha,lo}$	2.714	1.403
$C_{N\alpha,hi}$	3.406	5.023
$C_{m\alpha,lo}$	0.09261	0.1293
$C_{m\alpha,hi}$	0.1708	0.2471
$C_{mq,lo}$	0.7596	3.407
$C_{mq,hi}$	1.902	1.164

In the second test, active microspoiler trajectories (M) are combined with no microspoiler trajectories to estimate both the body and microspoiler aerodynamic coefficients. As with the first test, C_{l0} and C_{lp} are held fixed to their estimated values. Four combinations were considered: five active microspoiler, two low angle of attack and three active microspoiler, two high angle of attack and three active microspoiler, and one low angle of attack, two high angle of attack, and two active microspoiler. The percent error in each parameter is given in Table 6.16. With only active microspoiler trajectories, there are large errors in C_{X2}, δ_A, and δ_N with smaller errors in C_{X0} and $C_{N\alpha}$. The inclusion of low angle of attack trajectories improves the estimates for C_{X0} and C_{X2}, but has little effect on δ_A. There is also a small improvement in $C_{m\alpha}$, C_{mq} and δ_m. Including high angle of attack trajectories, on the other hand, sees an almost across the board reduction in performance. Using all three types of trajectories does well on C_{X0} and δ_m, but struggles on the other parameters. In all of these cases, there is some benefit to combining trajectories, especially for

estimating C_{X0}. However, without enough trajectories of each type, it is difficult for the estimation algorithm to differentiate between some of the parameters, ultimately leading to a loss of accuracy and minimal benefit to combining trajectories. Using a larger number of each type of trajectory may allow for more accurate estimation, but at the cost of increased problem difficulty and computation time.

Table 6.16: Percent Error in Final Parameter Estimate for Four Active Microspoiler Trajectory Combinations

Parameter	**5M** % **Error**	**2L, 3M** % **Error**	**2H, 3M** % **Error**	**1L, 2H, 2M** % **Error**
$C_{X0,lo}$	0.3621	0.2082	2.823	0.2884
$C_{X0,hi}$	1.77	0.1878	7.307	0.4103
$C_{X2,lo}$	20.96	11.23	8.917	23.47
$C_{X2,hi}$	18.02	8.942	23.42	30.46
$C_{N\alpha,lo}$	5.579	1.261	3.411	0.6076
$C_{N\alpha,hi}$	2.513	5.599	6.316	5.372
$C_{m\alpha,lo}$	0.1601	0.02905	0.1666	0.2927
$C_{m\alpha,hi}$	0.2484	0.1095	0.2422	0.1054
$C_{mq,lo}$	4.649	2.286	2.452	2.616
$C_{mq,hi}$	0.8577	1.267	3.75	4.353
$\delta_{A,lo}$	29.48	29.47	27.45	48.26
$\delta_{A,hi}$	32.55	34.93	36.76	58.11
$\delta_{N,lo}$	5.786	33.88	33.89	26.36
$\delta_{N,hi}$	29.67	35.99	31.09	37.64
$\delta_{m,lo}$	0.8421	0.7042	0.3059	0.2562
$\delta_{m,hi}$	1.857	0.7471	0.9813	0.4364

6.3 Experimental Data Analysis

To conclude the analysis of the smart projectile parameter estimation algorithm, the method is applied to experimental data collected from the ARL Transonic Experimental Facility spark range. A total of fourteen shots were obtained; four baseline rounds and ten equipped with active microspoilers. These shots ranged in Mach

number from 1.8 to 2.5. The constructed rounds were also about 30% lighter than the standard ANF described in Table 2.1. Similar to the procedure in Section 6.1, the four unactuated trajectories are used to determine the projectile body aerodynamic parameters in order to simplify the estimation of the microspoiler parameters. A selection of the actuated trajectories are then used to estimate the microspoiler parameters.

One major issue with the data obtained from the spark range is the amount of error present in the roll angle measurements. Each trajectory was missing numerous roll measurements and the measurements that were obtained had errors with a standard deviation of about 20-30°. This large error, coupled with gaps in the measurements, makes it difficult for the parameter estimation algorithm to estimate the roll parameters. Since the roll behavior of the projectile is generally decoupled from the other states, the roll coefficients C_{l0} and C_{lp} as well as the initial roll angle and roll rate are estimated independently. The amount of error on the measurements for the remaining states were approximated by observing the error between the data and the optimal trajectory determined by the estimation algorithm for each individual trajectory. The approximated standard deviations for the measurement errors are 5 mm for position, 0.2° for θ and ψ, and 30° for ϕ.

6.3.1 Roll Parameter Estimation

The first step in processing the experimental data is estimating the roll parameters. While the roll dynamics are decoupled from the other dynamics, the roll rate plays a significant role in the motion of the projectile. However, the large errors in the roll measurements make it extremely difficult for the estimation algorithm to determine the roll parameters and other parameters simultaneously. Three of the unactuated trajectories were used to estimate the roll parameters. Only x and ϕ were included in the cost function with C_{X0} and u_0 estimated in addition to the roll parameters.

For this case, p_0 was bounded at ± 20 rad/s as the rounds were fired from a smooth-bore barrel which does not produce large initial roll rates. In addition, weights were added to the ϕ measurements with larger weights on the earlier measurements. These weights varied linearly from a weight of 1.0 at the first station to 0.1 at the last station. The weights are then scaled such that the sum of the weights is equal to the number of measurements. On these tests, the projectile reaches its steady state roll rate about halfway through the flight. Larger weights on the early measurements help to better estimate the transient dynamics of the projectile and puts a larger emphasis on the behavior of the projectile before it reaches the first measurement. For this case, the weights decrease linearly with range.

The estimates for C_{l0} and C_{lp} at $M = 1.8$ and $M = 2.5$ are given in Table 6.17. The nominal values of these parameters for the standard ANF from Table 2.2 are included as reference. The first observation from these results is that the estimated coefficients are significantly higher than the nominal values. However, the mean χ^2 value for ϕ is 24.46, indicating a good fit given the large amount of noise in the data. The ratio $\frac{C_{l0}}{C_{lp}}$, on the other hand, is almost exactly the same as for the nominal round. This ratio characterizes the steady state roll rate of the projectile which is more observable than the transient behavior. Even with the weighting along the trajectory, the lack of observability of the projectile motion before the first measurement makes accurately estimating these parameters extremely difficult. Figures 6.57-6.59 show how well this set of parameters fits the data with very few noticeable errors.

Table 6.17: Experimental Roll Parameter Results

	M = 1.8		M = 2.5	
Parameter	**Nominal ANF**	**Estimate**	**Nominal ANF**	**Estimate**
C_{l0}	0.092575	0.19481	0.055	0.13903
C_{lp}	-8.53404	-18.3912	-5.0894	-13.0521
$\frac{C_{l0}}{C_{lp}}$	-0.0108	-0.0106	-0.0108	-0.0107

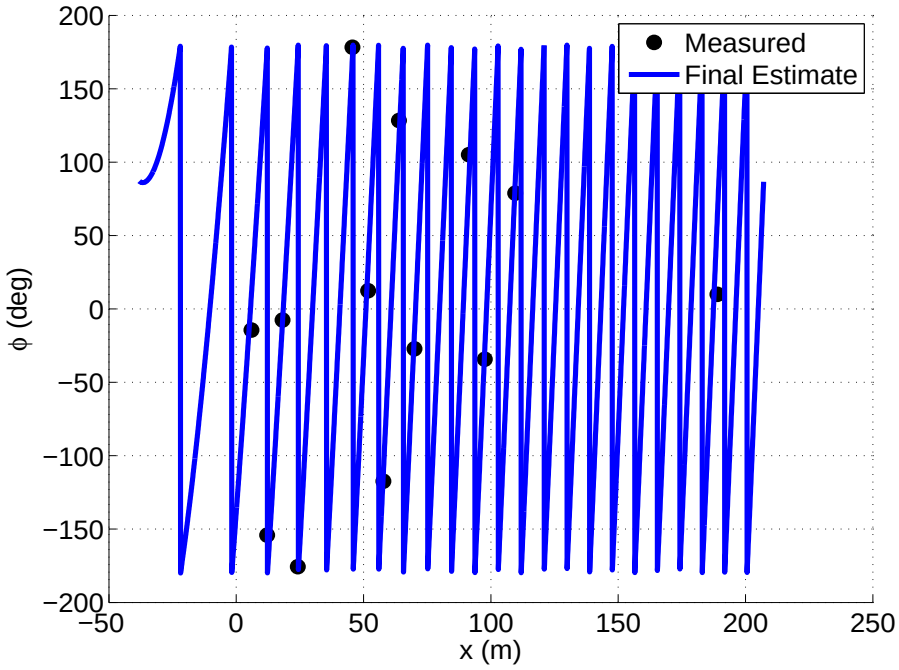


Figure 6.57: Unactuated Trajectory 1 Roll Angle Fit

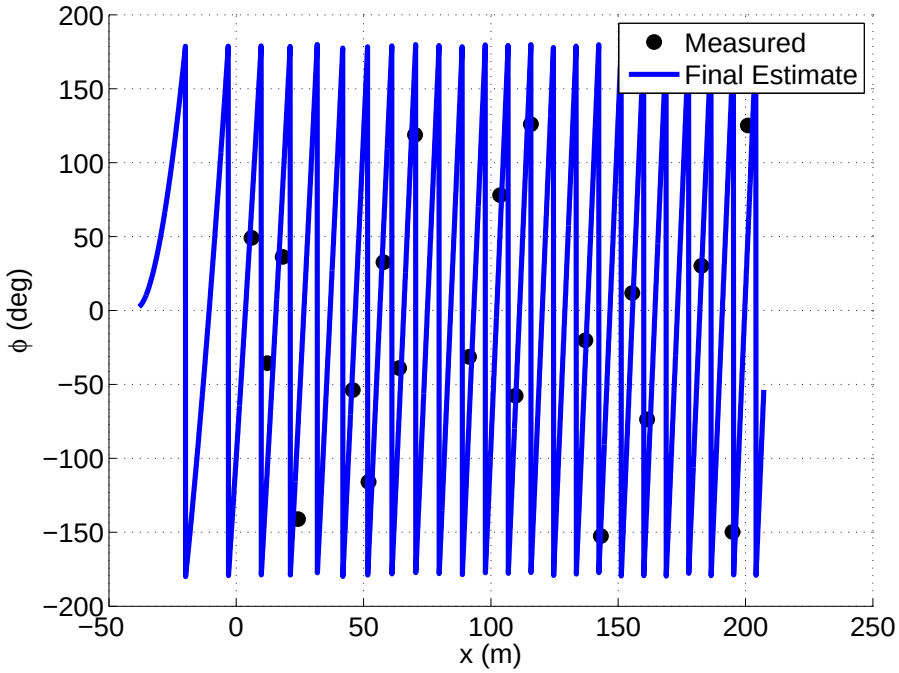


Figure 6.58: Unactuated Trajectory 2 Roll Angle Fit

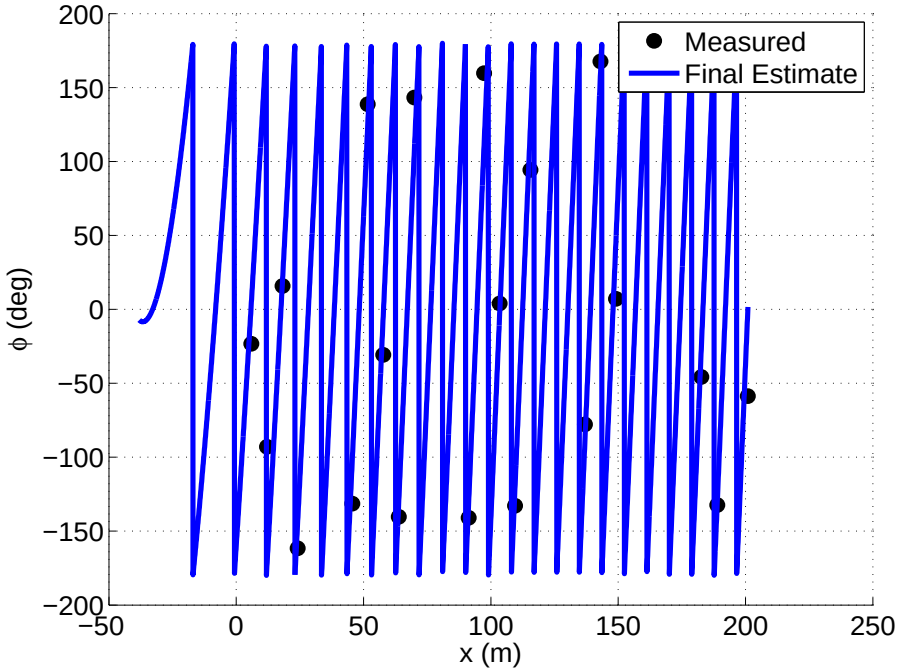


Figure 6.59: Unactuated Trajectory 3 Roll Angle Fit

6.3.2 No Microspoiler Results

With estimates for the roll parameters from Section 6.3.1, the remaining body aerodynamic coefficients are estimated. For this case, the simulations in the cost function were started from the first spark station measurement. Since the roll parameter estimates only captured the steady state behavior of the roll rate, there may be larger errors in the roll profile prior to the first measurement, causing additional errors in the trajectories of the other states. A small weight was placed on the roll measurements, reducing the impact of these errors on the cost function. The four unactuated shots were all within the range of $M = 2.1$ to $M = 2.5$.

Table 6.18: Experimental Body Aerodynamic Parameter Results

	M = 2.1		**M = 2.5**	
Parameter	**Nominal ANF**	**Estimate**	**Nominal ANF**	**Estimate**
C_{X0}	0.601	0.5239	0.5356	0.4583
$C_{m\alpha}$	-20.9922	-21.9045	-16.048	-14.3902
C_{mq}	-399.16	-398.01	-371.8	-379.68

Table 6.19: χ^2 Values for No Microspoiler Trajectories

	x	y	z	θ	ψ
Average	32.59	30.62	11.17	28.26	34.12
Standard Deviation	15.13	10.94	3.18	6.248	18.11

The coefficient estimates for the body aerodynamic parameters at $M = 2.1$ and $M = 2.5$ are given in Table 6.18. Note, estimates for C_{X2} and $C_{N\alpha}$ could not be found due to a lack of observability of these parameters at the low angles of attack of these shots. Overall, the parameter estimation algorithm provided accurate fits to the experimental data. The estimates for $C_{m\alpha}$ and C_{mq} were very close to the nominal values at both Mach numbers and the estimate for C_{x0} was about 15% less

than the expected value. This discrepancy may be due to small errors in the preflight weight measurements or small timing errors in the data. Table 6.19 shows the χ^2 values for the trajectories which indicate good fits of all of the trajectories. This is demonstrated in Figures 6.61-6.65 which show the fit for one of the data sets. From these trajectories, it is clear how accurately the estimation algorithm is fitting the spark range data, especially the θ and ψ measurements. The angle of attack on this trajectory remains below 4° and decays to below 2° about halfway downrange, too low to obtain accurate estimates of C_{X2} and $C_{N\alpha}$.

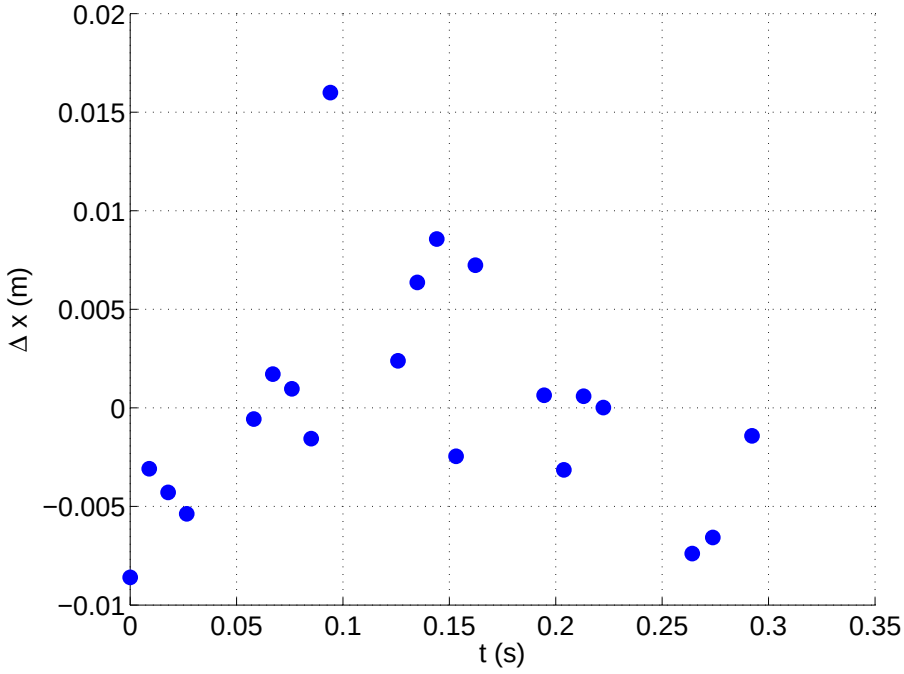


Figure 6.60: Unactuated Inertial-X Position Error

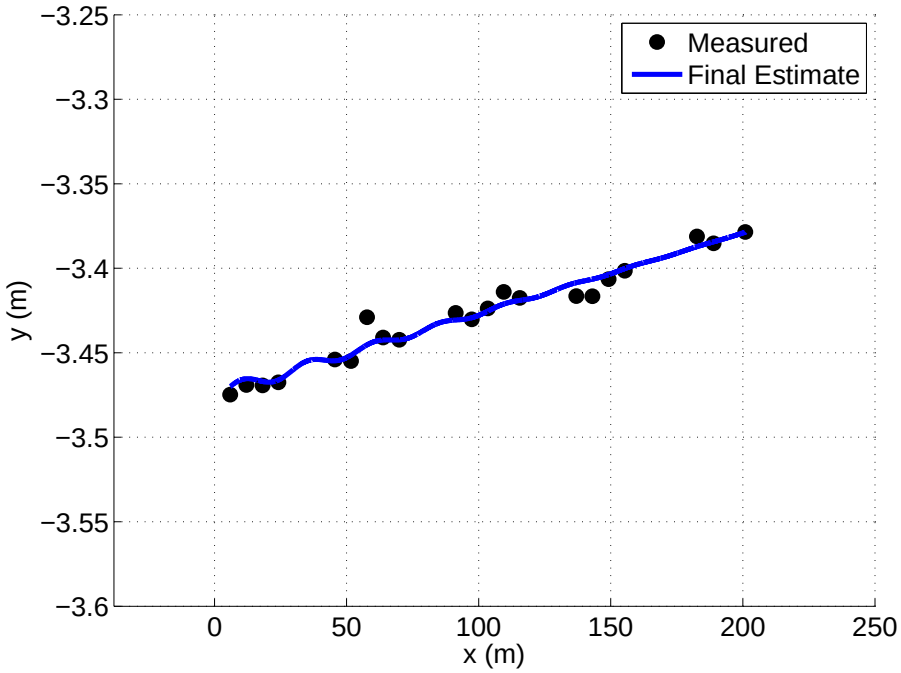


Figure 6.61: Unactuated Inertial-Y Position vs. Range

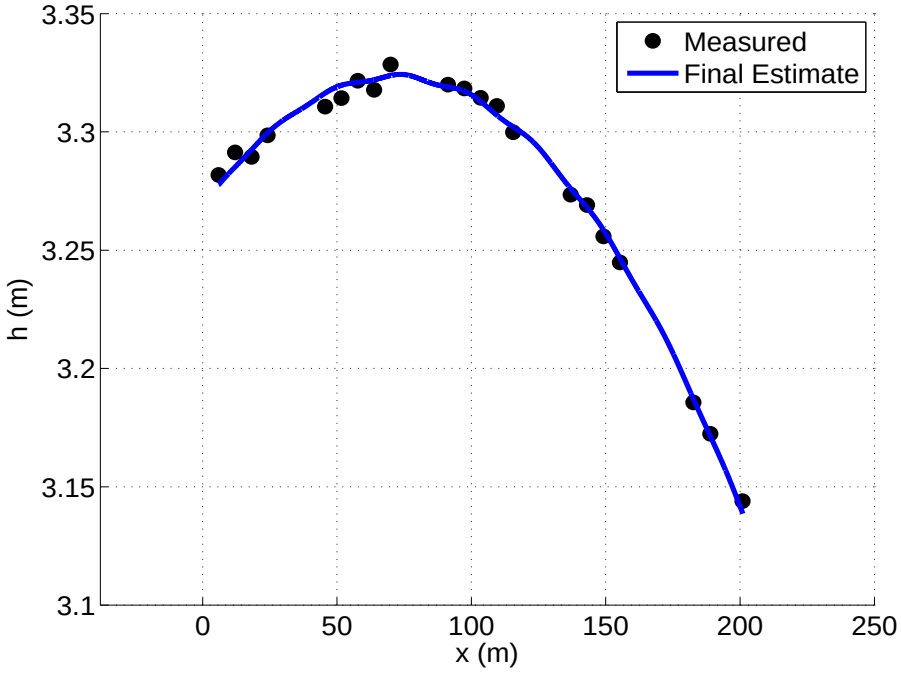


Figure 6.62: Unactuated Altitude vs. Range

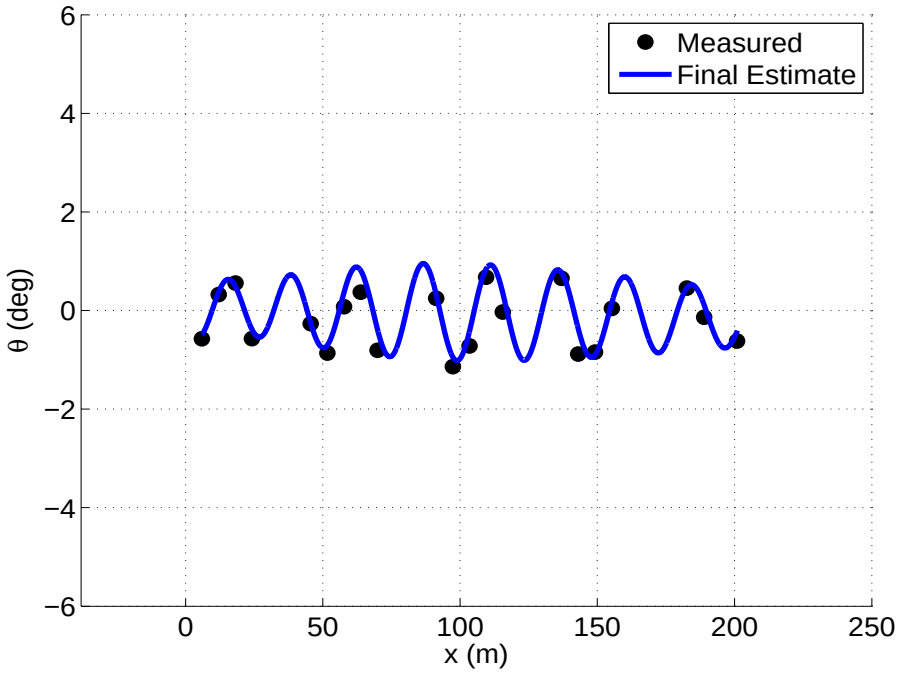


Figure 6.63: Unactuated Pitch Angle vs. Range

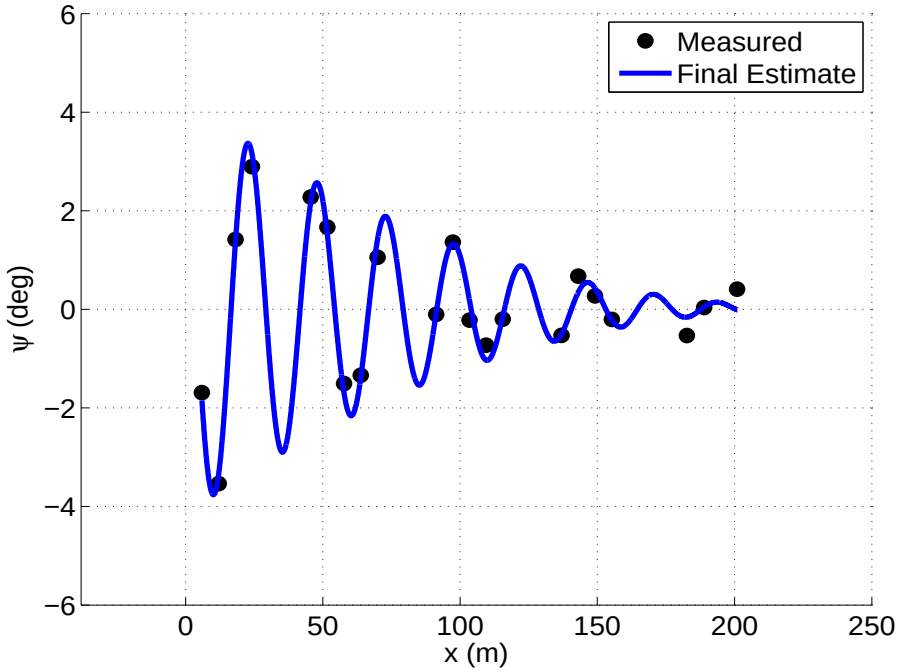


Figure 6.64: Unactuated Yaw Angle vs. Range

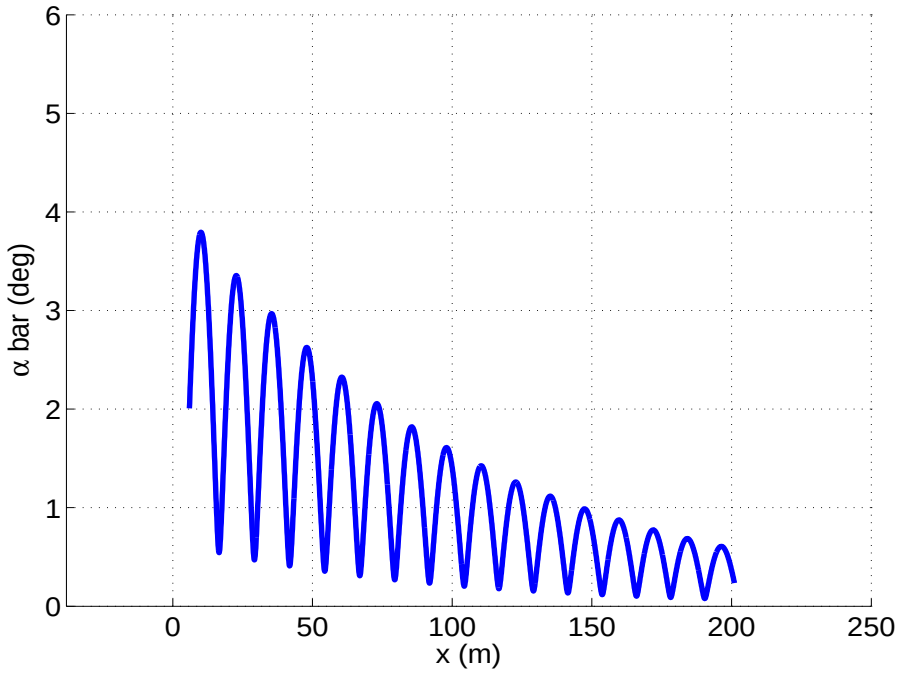


Figure 6.65: Unactuated Total Angle of Attack vs. Range

6.3.3 Active Microspoiler Results

To determine the microspoiler parameters, four of the ten trajectories were selected based on the performance of the parameter estimation algorithm in fitting these data sets individually. The projectile body aerodynamic coefficients C_{X0}, C_{l0}, C_{lp}, $C_{m\alpha}$, and C_{mq} were fixed to the estimated values above with C_{X2} and $C_{N\alpha}$ set to the nominal ANF values. For this case, the simulations in the cost function were started from launch. Even though there were concerns about the accuracy of the roll rate from launch to the first measurement, this was necessary to isolate the impact of the microspoilers on the projectile by restricting the initial angular velocity at launch. From observing the no microspoiler results, initial pitch rate q was constrained to a range of ± 2.5 rad/s. The effect of the microspoilers is also most pronounced at launch where they produce a large perturbation in pitching moment.

It was also necessary to estimate the initial roll angle as the orientation of the microspoilers at launch has a large impact on the resulting motion. A small weight

was placed on the ϕ measurements to help improve the accuracy of the ϕ_0 estimate without the errors in the ϕ measurements significantly adding to the cost. Only initial ϕ, u, v, w, q, r, and t were estimated with the remaining initial states fixed to known values and p_0 set to zero. Linearly decreasing weights were also used on every state trajectory. Because the microspoilers were actuated off of the projectile roll cycle, the angular motion induced by the microspoilers dissipates once the projectile reaches its steady state roll rate. Thus, the data from the first few spark stations are more important for determining the microspoiler parameters. The four trajectories evaluated cover a Mach range of 1.8 to 2.1.

Table 6.20: Experimental Microspoiler Parameter Results

	M = 1.8		**M = 2.1**	
Parameter	**CFD**	**Estimate**	**CFD**	**Estimate**
δ_A (N)	-16.4	-44.098	-19.5	-55.769
δ_N (N)	42.7	66.344	50.36	102.329
δ_m (Nm)	5.26	5.243	6.25	6.615

Table 6.21: Microspoiler Mechanism Parameter Estimation Results

Parameter	**Ideal**	**Mean Estimate**	**STD Estimate**
Ω_0 (rad/s)	440.0	440.57	33.02
τ_{ms} (1/s)	0.025	0.0298	0.0145

Table 6.22: χ^2 Values for Active Microspoiler Trajectories

	x	y	z	θ	ψ
Average	179.19	53.26	25.48	170.59	184.20
Standard Deviation	166.15	17.99	6.98	29.94	47.31

Tables 6.20 and 6.21 show the microspoiler coefficient estimates with the values of the microspoiler coefficients previously determined from CFD are also given as a reference [118]. Of the three microspoiler force and moment coefficients, the parameter

estimation algorithm does an excellent job estimating δ_m with large errors in δ_A and δ_N. The error in δ_A can be attributed to the value of C_{X0} of the body being about 10% below the nominal ANF value. It should be noted that when the nominal value of C_{X0} was used in place of the estimated value, the estimates for δ_A were significantly improved. δ_N, much like $C_{N\alpha}$, has very low observability making accurate estimates difficult. While the estimate of δ_m at $M = 1.8$ almost exactly matches the CFD value, there is some error in the $M = 2.1$ estimate. Since all of the trajectories are below $M = 2.1$, there is less observability in the parameter at the higher end of the Mach range. The average estimated microspoiler spin rate was nearly identical to the design value, however not every trajectory matched this performance. The average estimated time constant for the microspoiler mechanism was also close to the design value of 0.025 s^{-1} determined from the bench testing of the mechanism.

Looking at the χ^2 values in Table 6.22, the estimation algorithm fits y and z very well but has some issues with the remaining states. The large variation in χ^2 between trajectories indicates that the estimation algorithm may have difficulty fitting one or more of the states for a given trajectory. There are a few possible explanations for these errors in the trajectory fitting. First, as was discussed in Section 6.3.1, the estimates of C_{l0} and C_{lp} were only accurate in terms of the steady state roll rate and showed good agreement with the data starting at the first measurement. Errors in the roll rate profile early in the flight can lead to errors in all of the states throughout the flight. Second, the estimates of the body aerodynamic coefficients may not be accurate in the Mach range of these trajectories. All of the unactuated trajectories were fired at Mach numbers above 2.1, requiring extrapolation of the parameter estimates to the range of the actuated data. Given the large range in Mach number and lack of trajectory data at the lower Mach number, the assumption of the aerodynamic coefficients varying linearly may not hold. Finally, the model of the microspoiler aerodynamics and mechanism may not be accounting for all of the

effects of the system in practice, adding errors to the parameter estimates.

The quality of the fits can be seen by looking at the final trajectory for one of the data sets shown in Figures 6.66-6.71. Figures 6.67 and 6.68 demonstrate the excellent fits of these states. Looking at Figures 6.69 and 6.70, much of the error in the final estimate is due to errors towards the end of the trajectory and a few data points where the magnitude of the estimated trajectory does not quite match the measurement. In general, the parameter estimation algorithm is able to obtain the general behavior in θ and ψ, allowing it to accurately estimate δ_m.

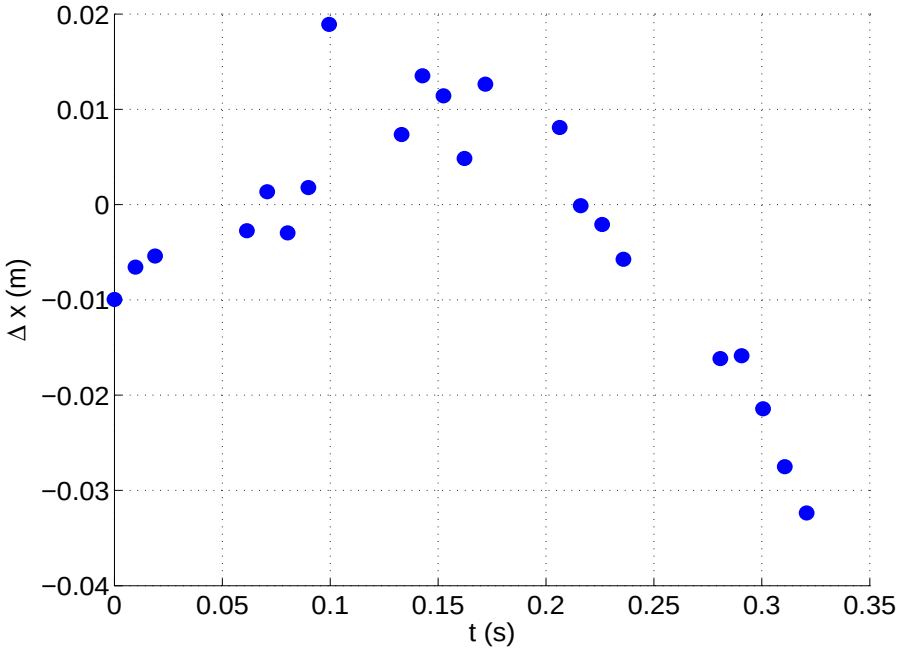


Figure 6.66: Active Microspoiler Inertial-X Position Error

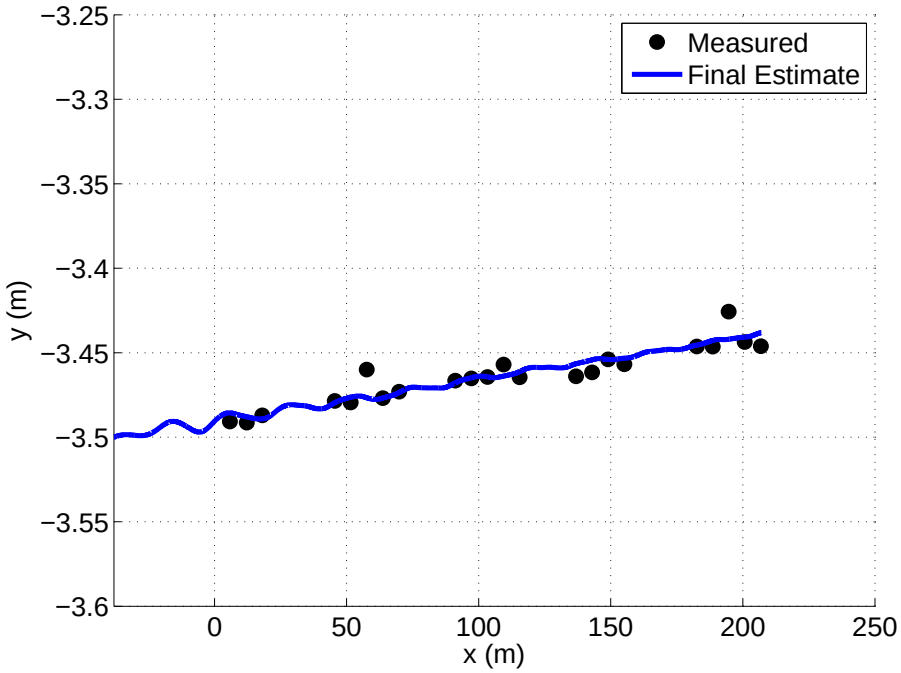


Figure 6.67: Active Microspoiler Inertial-Y Position vs. Range

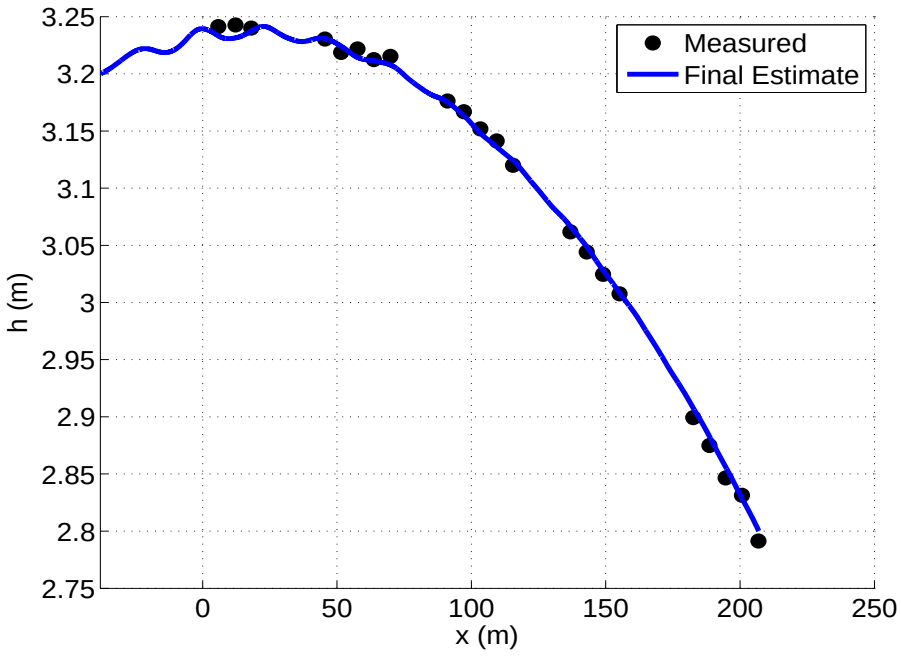


Figure 6.68: Active Microspoiler Altitude vs. Range

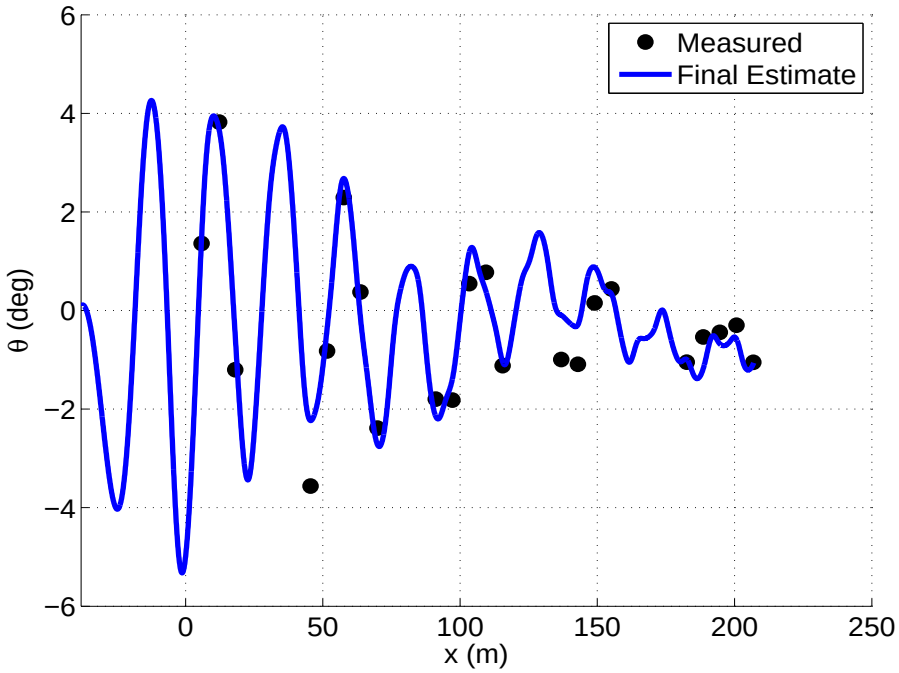


Figure 6.69: Active Microspoiler Pitch Angle vs. Range

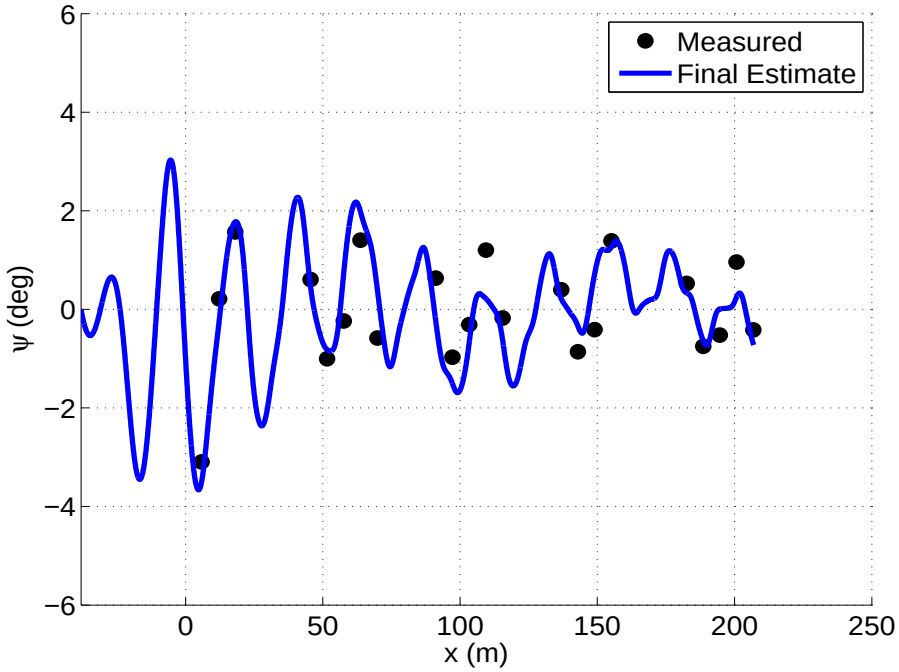


Figure 6.70: Active Microspoiler Yaw Angle vs. Range

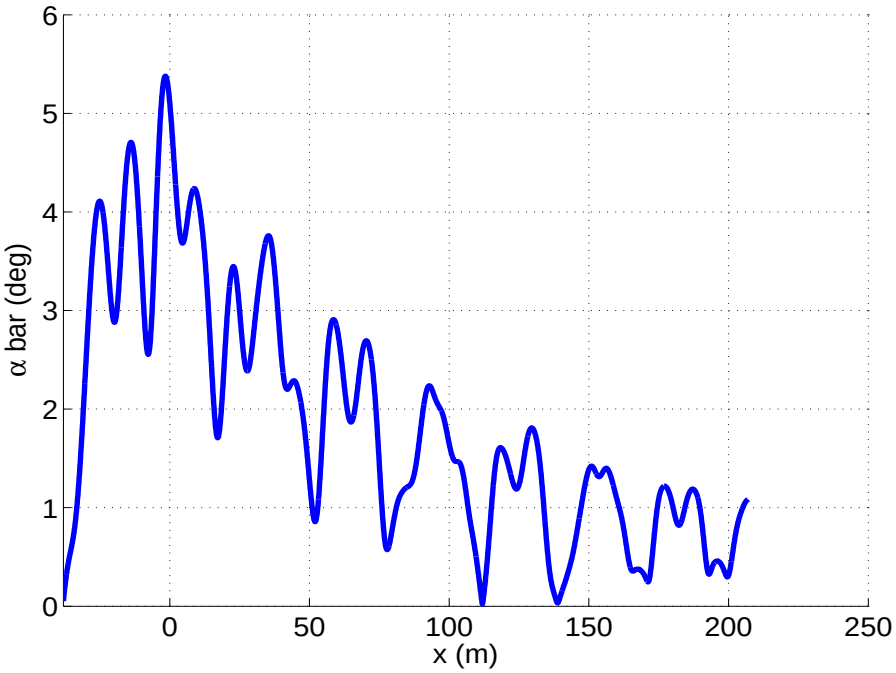


Figure 6.71: Active Microspoiler Total Angle of Attack vs. Range

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The process of performing parameter estimation for new projectile configurations is a key component of the design process and is necessary for creating accurate models that are used for developing control systems and running simulations for the projectile. Typically, flight test data is used in addition to CFD and wind tunnel data when available. While many techniques currently exist for estimating the aerodynamic coefficients for projectiles based on flight test data, these methods rely on accurate initial estimates for the coefficients to ensure convergence. For new projectile systems with complicated control mechanisms and generally unknown parameters, a new method is needed to perform parameter estimation. This thesis explored a new approach to smart projectile parameter estimation, combining an output error based algorithm with a new, robust optimization method dubbed meta-optimization. Meta-optimization was designed to obtain accurate and reliable parameter estimates for new and complex smart projectile configurations.

To better understand the nature of the smart projectile parameter estimation problem, a detailed analysis was performed to characterize the landscape of this problem. The smart projectile considered in this work was a standard finned projectile equipped with a microspoiler actuator system. Flight data was collected from the U.S. Army Research Laboratory spark range which collects discrete measurements of the position and orientation of the projectile. The first analysis used slices of the cost function at different parameter combinations to visualize the topology of the cost function and identify local minima. When the roll measurements were bounded, the

roll parameters showed numerous local minima, particularly in C_{l0} and p_0, caused by the dynamics of the projectile. Local minima also occurred in $C_{m\alpha}$ and Ω_0 and were confirmed from the projectile dynamics. Local search analysis also revealed numerous local minima, some far outside the search space corresponding to non-physical parameter sets. However, when bounds were applied to constrain the parameters to the search space, local minima appeared along the boundary that were not present before. The presence of these local minima make it difficult to reliably use existing local search based methods as reasonable estimates of the parameters may not exist *a priori*.

The meta-optimization framework addressed these concerns by incorporating a mixture of local and global optimizers, each suited for different types of problems. Meta-optimization operates by selecting a single algorithm and deploying it on the problem. Periodically, the performance of the optimizer is evaluated using an efficiency metric based on cost reduction and computation time. Selection of the optimizers is performed using a variable structure learning automaton which assigns probabilities to each optimizer that are weighted towards the most effective optimizers. The transition between optimizers is handled by a common manager that provides the optimizers with the resources to operate on the problem. Finally, auto-tuning adjusts the optimizers themselves to improve performance online. This framework allows meta-optimization to reliably solve a wide range of optimization problems.

A suite of optimization benchmark functions were used to analyze and develop the meta-optimization framework. These functions contained a mixture of unimodal and multimodal functions with varying topology and dimensionality. A single optimization run was used to demonstrate the behavior of meta-optimization where each optimizer is given opportunities to work on the problem, collectively working towards the solution. Meta-optimization was then compared to each of the individual optimizers on this suite of functions. On every function, meta-optimization

proved to be more effective than any of the optimizers, reliably solving most of the problems. On the few problems meta-optimization could not solve every time, the functions also were challenging for the individual optimizers, indicating a limitation in the performance of meta-optimization due to the capabilities of the included optimizers. The benchmark suite was also used to conduct a series of trade studies on each component of meta-optimization. A final configuration was determined from these trade studies maximizing performance and robustness over a wide range of problems. Finally, meta-optimization was evaluated on the CEC 2014 competition benchmark suite and compared to three state of the art optimizers entered in the competition. Even without tuning to handle the conditions of the competition, meta-optimization achieved comparable performance on many of the functions, with better performance on the higher dimensional problems.

Lastly, the parameter estimation algorithm was applied to an example smart projectile system. Fist, the estimation algorithm was evaluated on synthetic trajectory data designed to replicate data typically obtained from flight testing. Using a build up procedure designed to replicate spark range testing, the proposed algorithm was able to accurately estimate all of the aerodynamic coefficients for the projectile body and microspoilers. The accuracy of estimates obtained by estimation algorithm was shown through multiple trade studies to be highly dependent on the amount of noise on the measurements. These errors are mitigated when more measurements are available, either by increasing the number of measurements in each trajectory or using multiple trajectories simultaneously. Lastly, the parameter estimation procedure was repeated using flight data recorded from experimental testing at the ARL Transonic Spark Range. The estimation algorithm successfully obtained estimates for the microspoiler parameters with sufficient accuracy to previous CFD results. Some of the errors in the estimates can be attributed to the high levels of noise in the data and a lack of observability of some parameters due to the designed test set up. Overall,

this new parameter estimation algorithm utilizing meta-optimization has proven to be an effective tool for analyzing new smart projectile systems.

7.2 Recommended Future Work

The results of this dissertation have demonstrated the capability of the proposed parameter estimation algorithm to reliably obtain parameter estimates under a wide range of conditions. The meta-optimization framework developed for this method offers numerous avenues for continued development of reliable optimization algorithms and applications for these methods. Below is a list of potential opportunities for continued improvement of the smart projectile parameter estimation algorithm and meta-optimization framework.

1. For estimating parameters of projectiles based on experimental data, a more detailed aerodynamic model should be considered which accounts for nonlinearities in the projectile aerodynamics. One core assumption in the projectile aerodynamic model is that the coefficients are linear over small ranges in Mach number. However, in practice the coefficients can change rapidly at certain Mach numbers, making estimates across these ranges inaccurate. An aerodynamic model that can handle these factors would significantly improve the estimation capabilities of the parameter estimation algorithm for future projectile systems.

2. Further analysis should be conducted into combining different types of trajectories in order to estimate more parameters simultaneously. By grouping these trajectories together, the total number trajectories necessary to fully estimate the parameters of the projectile system could be reduced, saving time and cost when conducting flight test experiments.

3. Given the flexibility of this parameter estimation algorithm, the ability of the

method to estimate other parameters of the projectile should be explored. These parameters include the mass properties of the projectile and the atmospheric conditions which are typically measured prior to flight testing. Potentially any parameter within the projectile model could be estimated using this parameter estimation algorithm.

4. The primary limitation of meta-optimization is the effectiveness of the individual optimizers included in the framework. More advanced variants of each algorithms should be considered which are capable of readily freeing themselves from local minima. In addition, state of the art optimizers such as L-SHADE [137] should be evaluated for inclusion as these optimizers have demonstrated superior performance on extremely difficult optimization problems.

5. Additional methods for auto-tuning of the included optimizer tuning parameters should be investigated. Methods such as online parameter control have recently shown promise on controlling evolutionary algorithm parameters. Additional research should also be conducted to develop a database of known effective tuning parameter sets for intelligent initialization of these optimizers on new problems.

6. The penalty function used to constrain the parameter search space was shown to create difficulties for numerous optimizers. New techniques for handling search bounds and general constraints should be considered which do not create additional local minima. Optimizer specific strategies should also be investigated to improve flexibility for the framework. Further research is needed into the behavior of the optimizers along these boundaries.

7. Many additional methods exist for evaluating optimizer performance beyond cost reduction and computation time. Composite metrics which incorporate

metrics such as predicted performance and optimizer risk should be developed and evaluated in the meta-optimization framework.

8. Finally, future work should apply meta-optimization to other challenging engineering optimization problems. Each new application will necessitate further development of meta-optimization, improving the reliability and robustness of the method. Ultimately, meta-optimization provides a common framework that could be applied to any situation where automated selection of diverse algorithms is needed.

Appendices

APPENDIX A

ADAPTIVE FINITE DIFFERENT STEP LENGTH ALGORITHM

The accuracy of a finite difference approximation is highly dependent on the choice of step size. The basis of a finite difference is the assumption that the function in generally linear in the neighborhood of the current point. If too large of a step is used, the finite difference can pick up nonlinearities in the function, reducing the accuracy of the derivative estimate. If the step is too small, there may be no appreciable change in cost, resulting in a very small or zero derivative estimate. Instead, Gill, Murray, and Wright use an adaptive approach which adjusts the step length in an online manner [131]. This is also useful when the ideal step length is unknown a priori. The algorithm uses the relative condition error of the finite difference to determine if the current step length is acceptable. Before proceeding, a few terms must be defined. The method is presented for a single parameter but can be easily applied to multiple dimensions. First, the forward and backwards finite differences are given by:

$$\varphi_F(x,h) = \frac{f(x+h) - f(x)}{h} \text{ and } \varphi_B(x,h) = \frac{f(x) - f(x-h)}{h} \tag{A.1}$$

Next, the second order difference is given by:

$$\Phi(x,h) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{A.2}$$

The bounds on the relative condition error of Φ, φ_F, and φ_B are defined as:

$$\hat{C}(\Phi) = \frac{4\epsilon_A}{h^2|\Phi|} \text{ and } \hat{C}(\varphi_F) = \frac{2\epsilon_A}{h|\varphi_F|} \text{ and } \hat{C}(\varphi_B) = \frac{2\epsilon_A}{h|\varphi_B|} \tag{A.3}$$

where ϵ_A is the estimated error and is a function of the cost specified by:

$$\epsilon_A = (1 + |f(x)|) \max\left(\epsilon_M, \frac{\epsilon_A(x_0)}{1 + |f(x_0)|}\right) \tag{A.4}$$

where ϵ_M is machine precision and $\epsilon_A(x_0)$ is determined through the use of a difference table. The table begins with a set of values of the cost f evaluated at a set of points. Each point is given by $x_i = x + ih$ where h is a small number. It is assumed that the computed value $\bar{f}_i$ takes the form:

$$\bar{f}_i = f(x_i) + \delta_i = f(x_i) + \theta_i \epsilon_A \tag{A.5}$$

The first column of the table contains the values of $\bar{f}_i$ and each subsequent column is generated using the difference operator. After k differences, $\Delta^k \bar{f}_i = \Delta^k f_i + \Delta^k \delta_i$. Since $\Delta^k f = h^k f^{(k)}$, the value $|h^k f^{(k)}$ will become small as k increases, leaving only the differences in the errors δ_i. A pattern of behavior emerges after a few columns where the differences are all of similar magnitude but alternating sign. An estimate of ϵ_A can be obtained from this column using:

$$\epsilon_A^{(k)} \sim \frac{\max_i |\Delta^k \bar{f}_i|}{\beta_k} \tag{A.6}$$

where $\beta_k = \sqrt{\frac{(2k)!}{(k!)^2}}$. The table begins with f evaluated at 20 points and continues until $k = 10$. The estimated value of ϵ_A is the average of the values computed for columns 4 through 10. With ϵ_A known, the bounds on the relative condition error can be evaluated and checked. If $\hat{C}(\Phi) \leq 0.001$, h is reduced by a factor of 5 and $\hat{C}(\Phi)$ is computed again. If $\hat{C}(\Phi) > 0.1$, then the original h is retained, otherwise, h is reduced until $\hat{C}(\Phi) \geq 0.001$. If the original $\hat{C}(\Phi) \geq 0.1$, then h is increased by a factor of 5 until $\hat{C}(\Phi) \leq 0.1$. If $0.001 \leq \hat{C}(\Phi) \leq 0.1$, then no adjustment is needed.

APPENDIX B

META-OPTIMIZATION TRADE STUDY RESULTS

B.1 Probability Update Rules

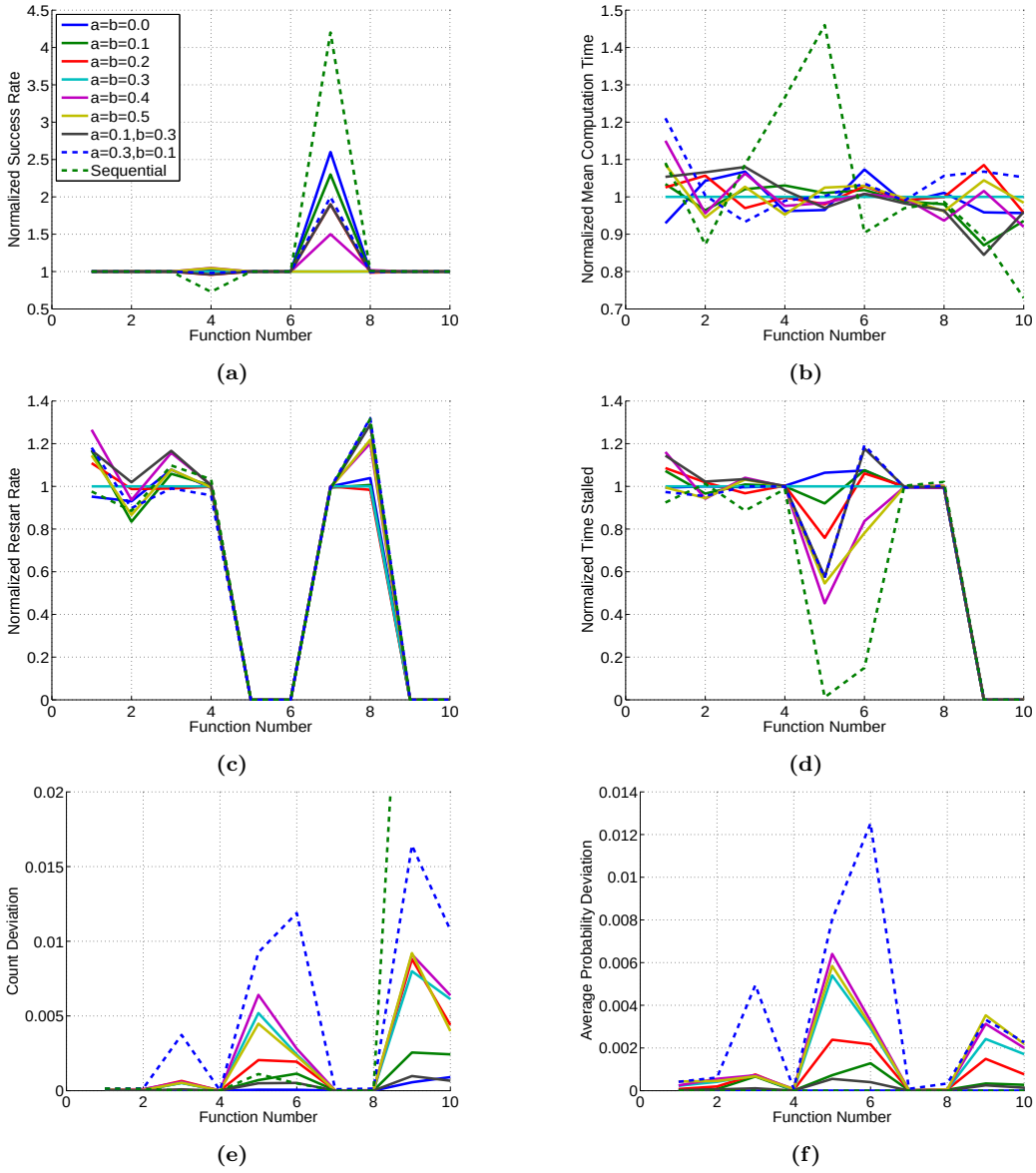


Figure B.1: Learning Automaton Reward and Penalty Factors (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate; (d). Time Stalled; (e). Optimizer Count Deviation; (f). Average Probability Deviation. Overall, lower values of the parameters perform better while higher values have larger deviations. Also includes a comparison to constant probability and sequential selection cases with the probability updates showing superior performance.

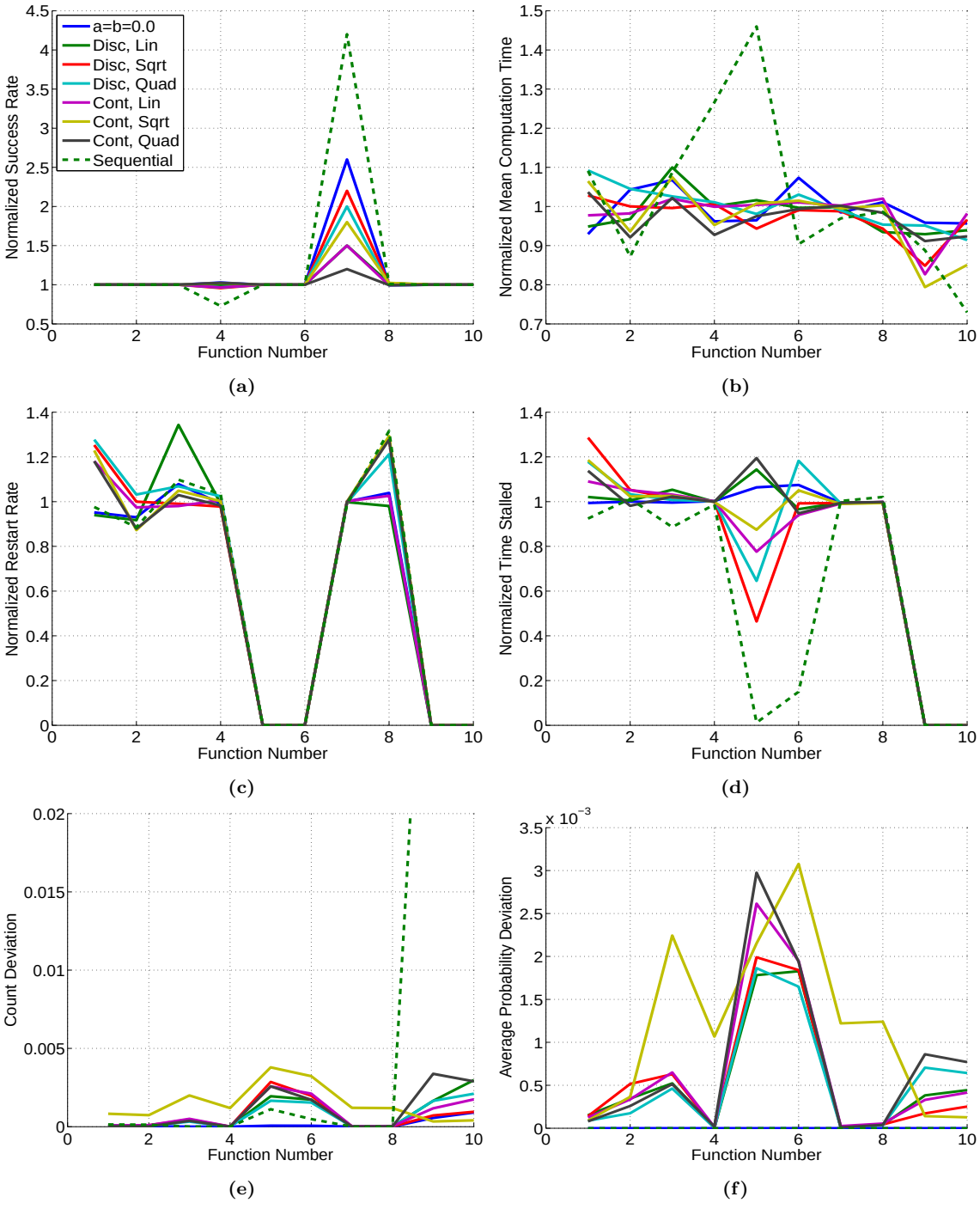


Figure B.2: Probability Weighting Method (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate; (d). Time Stalled; (e). Optimizer Count Deviation; (f). Average Probability Deviation. Discrete and continuous weighting schemes are considered with three different function shapes. Continuous weighting performed best on computation time and restart rate while the discrete weighting was best on time stalled. Square root weighting produced the largest deviations.

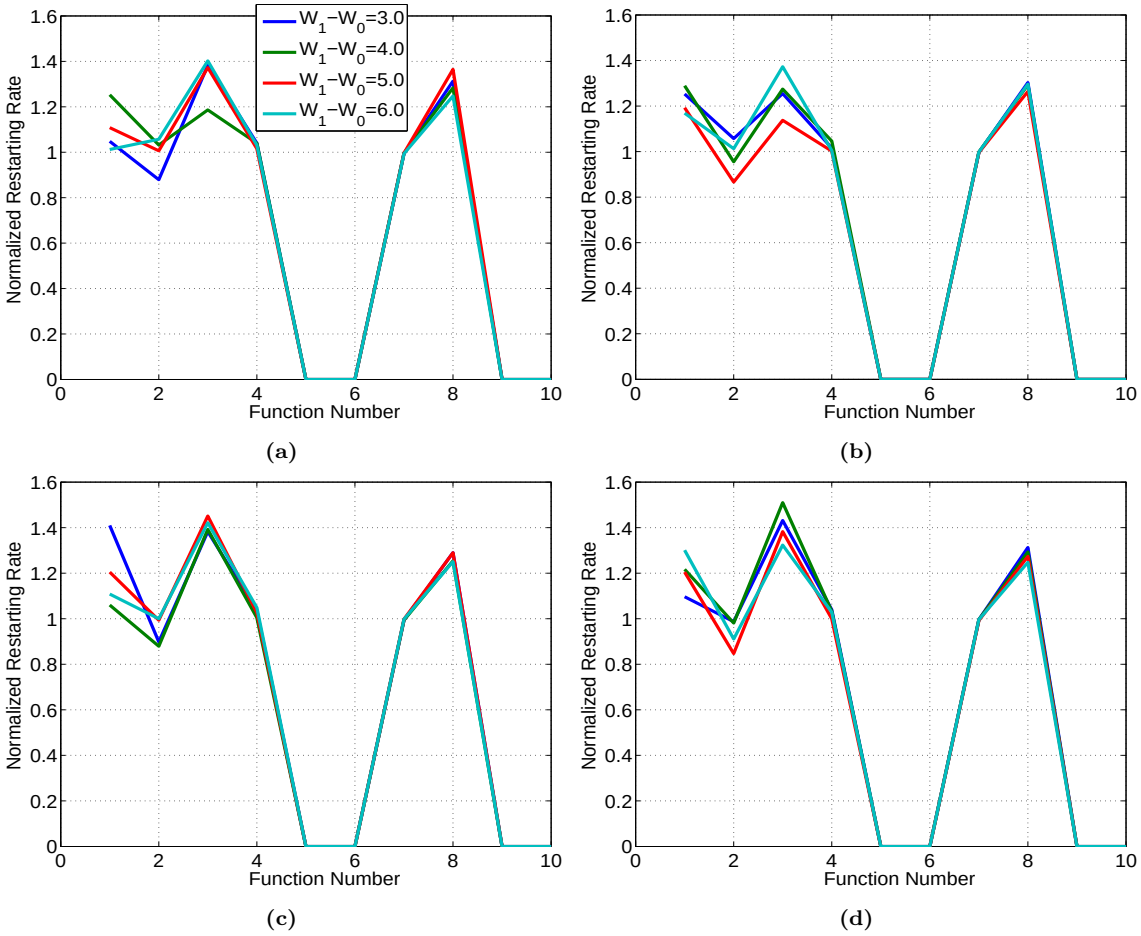


Figure B.3: Weighting Coefficient Restarting Rate (a). $W_0 = 0.5$**; (b).** $W_0 = 1.0$**; (c).** $W_0 = 2.0$**; (d).** $W_0 = 3.0$**. Lower values of** W_0 **and medium values of** $W_1 - W_0$ **required fewer restarts than other combinations.**

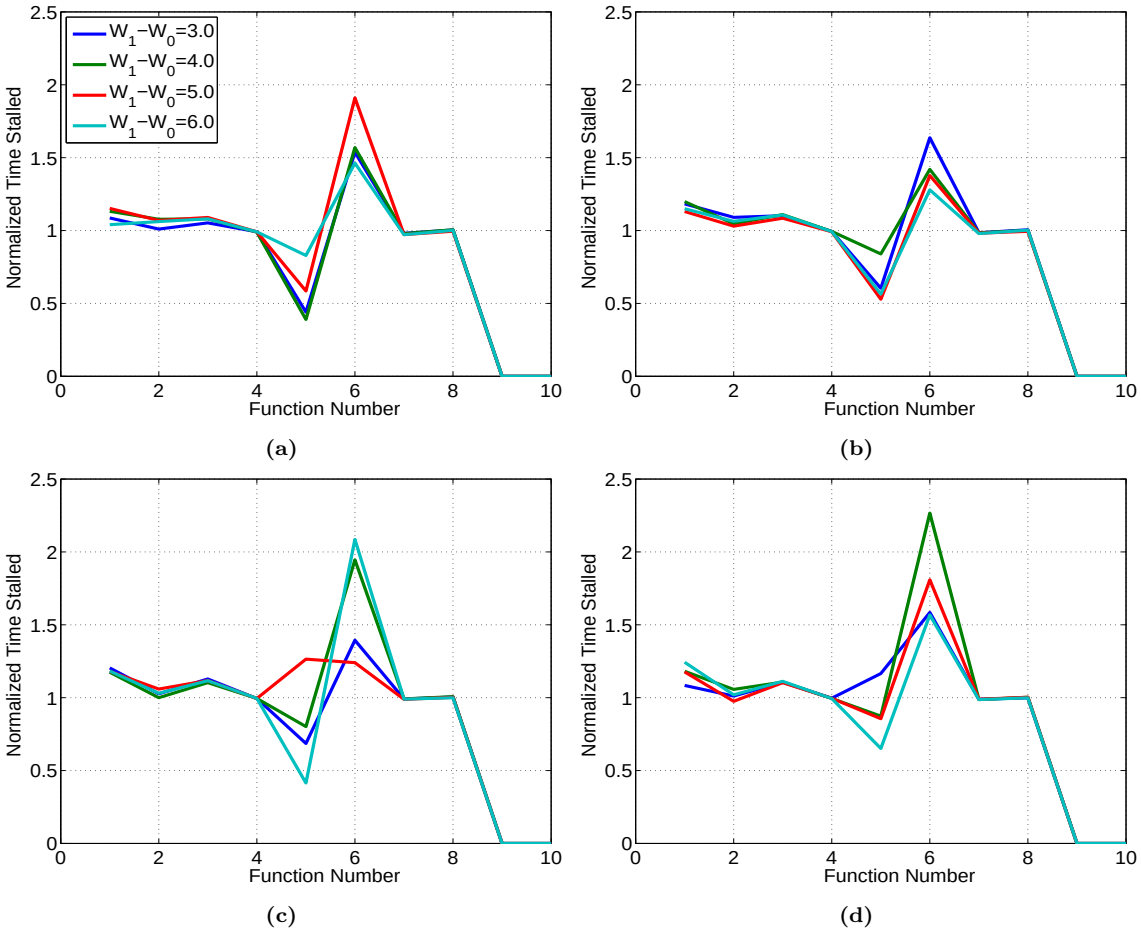


Figure B.4: Weighting Coefficient Time Stalled (a). $W_0 = 0.5$; (b). $W_0 = 1.0$; (c). $W_0 = 2.0$; (d). $W_0 = 3.0$. Lower values of W_0 stall less, especially on functions 5 and 6.

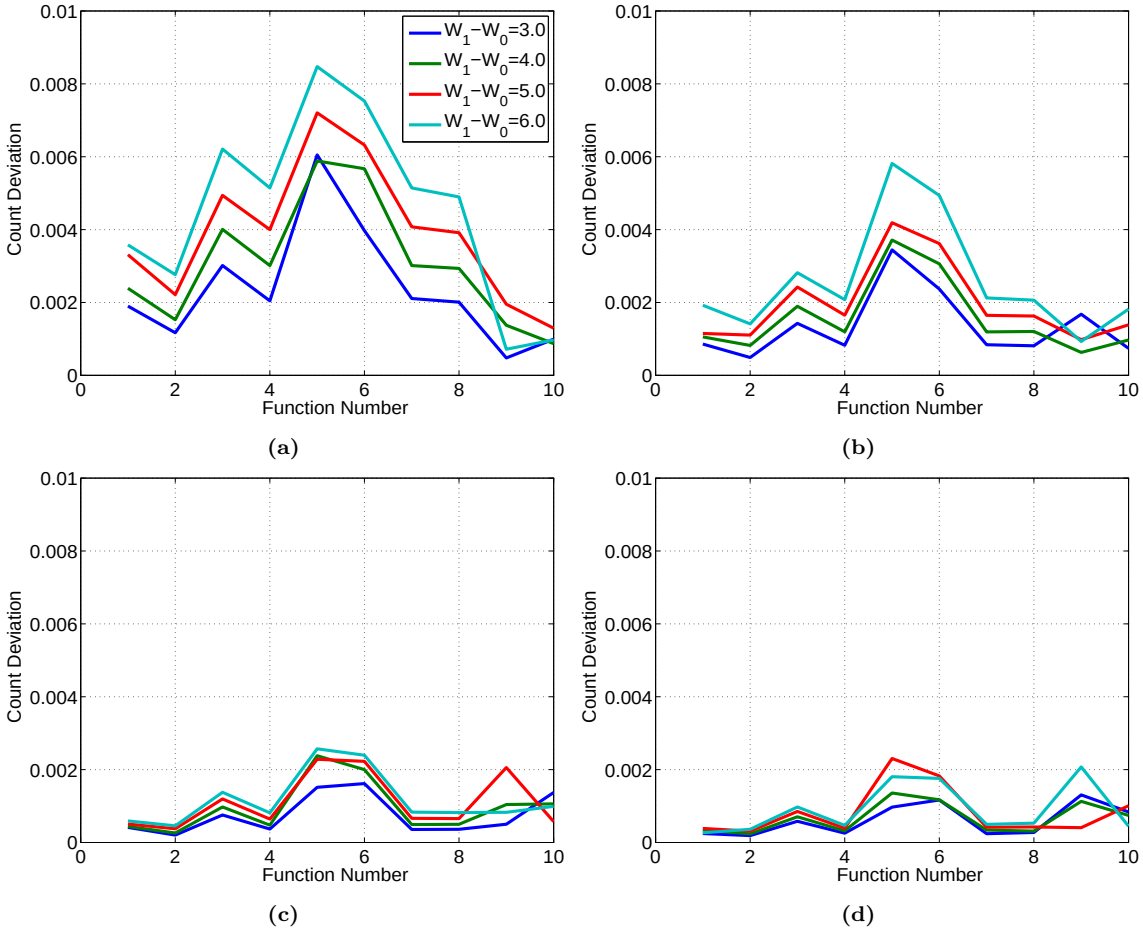


Figure B.5: Weighting Coefficient Optimizer Count Deviation (a). $W_0 = 0.5$**; (b).** $W_0 = 1.0$**; (c).** $W_0 = 2.0$**; (d).** $W_0 = 3.0$**. Clear trend of increasing deviation with decreasing** W_0 **and increasing** $W_1 - W_0$**.**

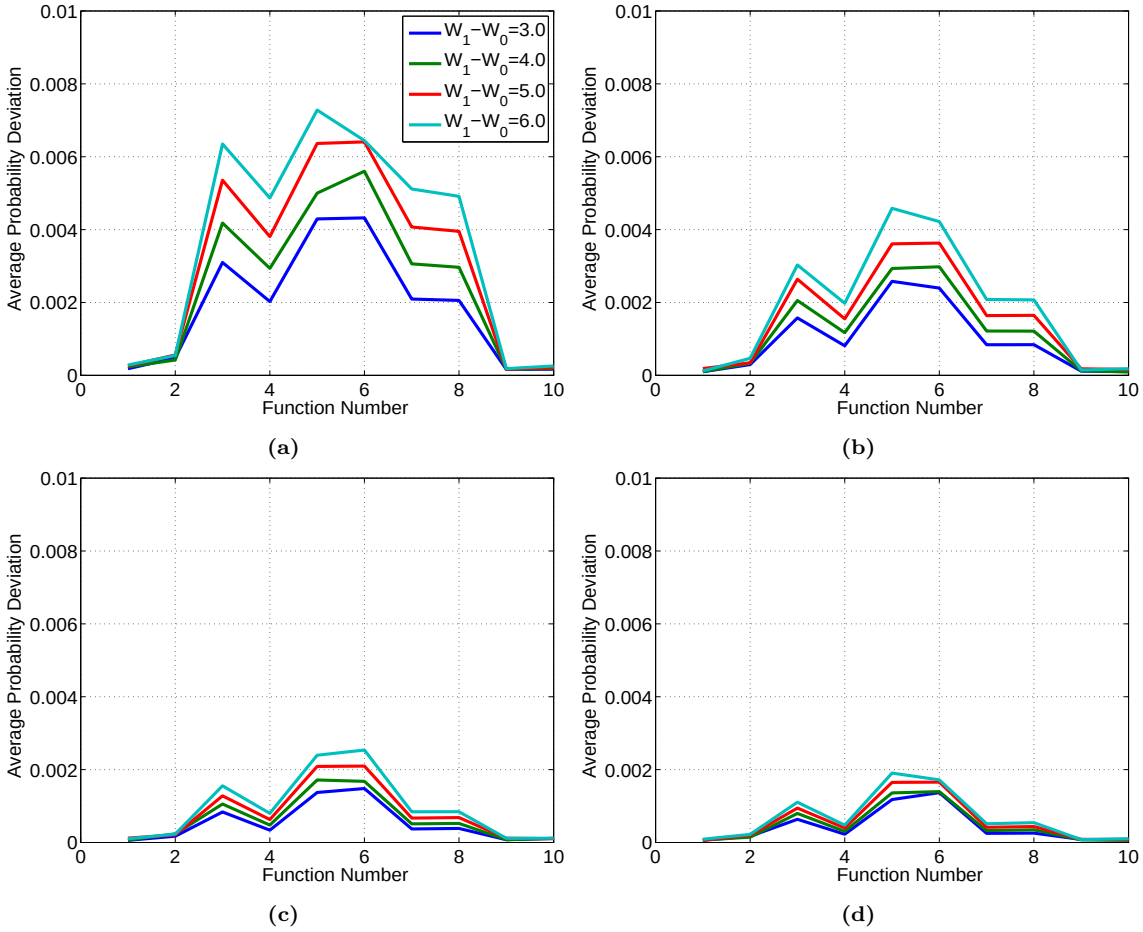


Figure B.6: Weighting Coefficient Optimizer Probability Deviation (a). $W_0 = 0.5$**; (b).** $W_0 = 1.0$**; (c).** $W_0 = 2.0$**; (d).** $W_0 = 3.0$**. Similar trend of increasing deviation with decreasing** W_0 **and increasing** $W_1 - W_0$**. No difference on functions 9 and 10.**

B.2 Optimizer Efficiency Evaluation

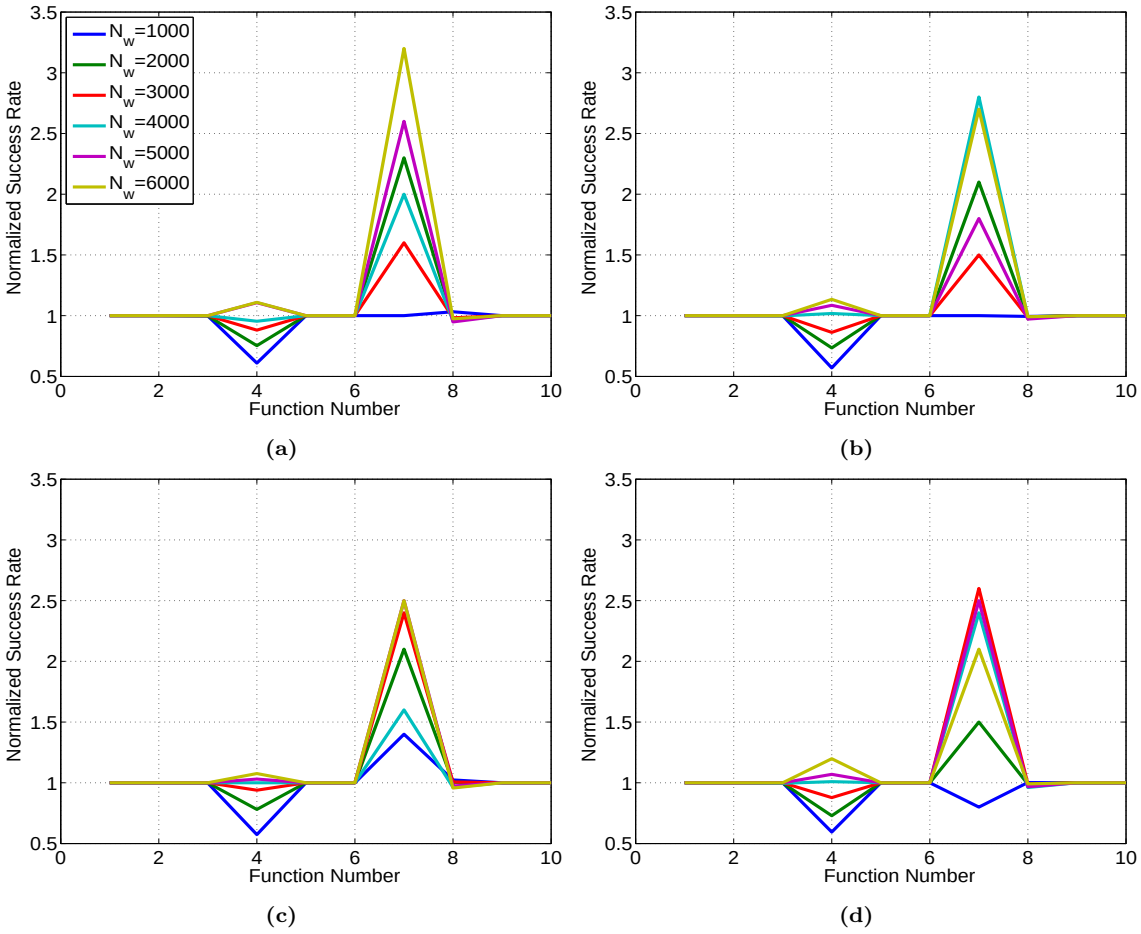


Figure B.7: Optimizer Efficiency Evaluation Parameters Success Rate (a). $J_{ref} = 0.5$; (b). $J_{ref} = 1.0$; (c). $J_{ref} = 1.5$; (d). $J_{ref} = 2.0$. Shorter efficiency windows experience significant loss in success rate. Little trend seen in terms of J_{ref}

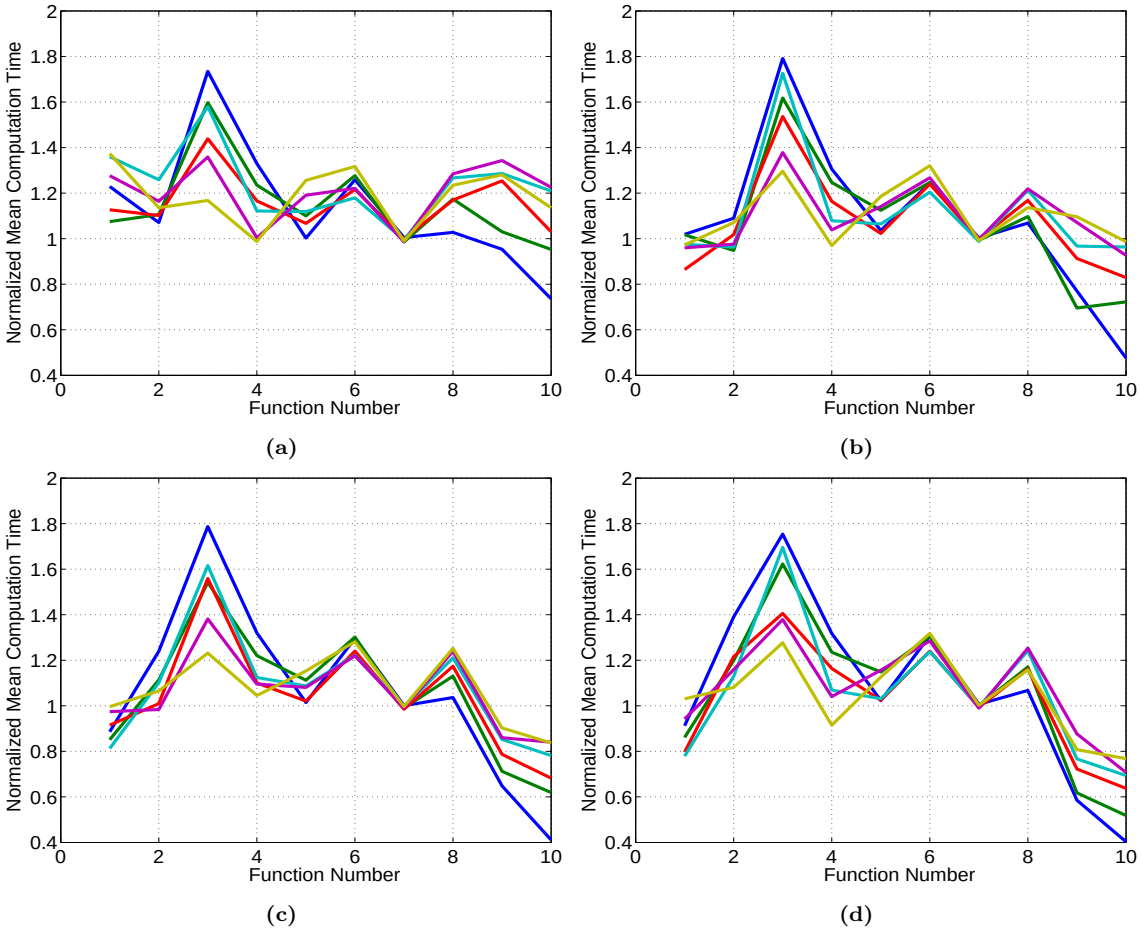


Figure B.8: Optimizer Efficiency Evaluation Parameters Mean Computation Time (a). $J_{ref} = 0.5$; (b). $J_{ref} = 1.0$; (c). $J_{ref} = 1.5$; (d). $J_{ref} = 2.0$. Computation time generally increases as the window increase and decreases with increasing J_{ref}.

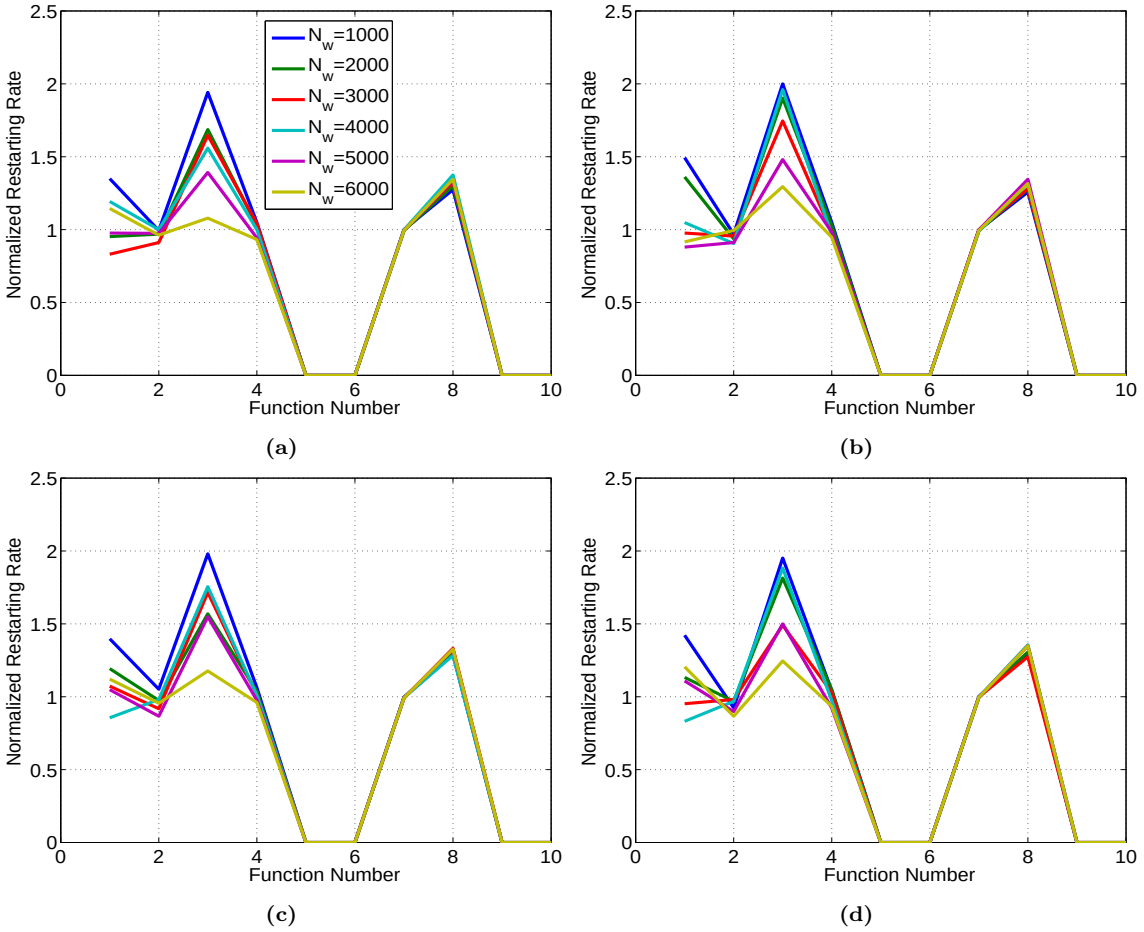


Figure B.9: Optimizer Efficiency Evaluation Parameters Restart Rate (a). $J_{ref} = 0.5$**; (b).** $J_{ref} = 1.0$**; (c).** $J_{ref} = 1.5$**; (d).** $J_{ref} = 2.0$**. General trend of decreasing restarting rate with increasing window. No trend in** J_{ref}**.**

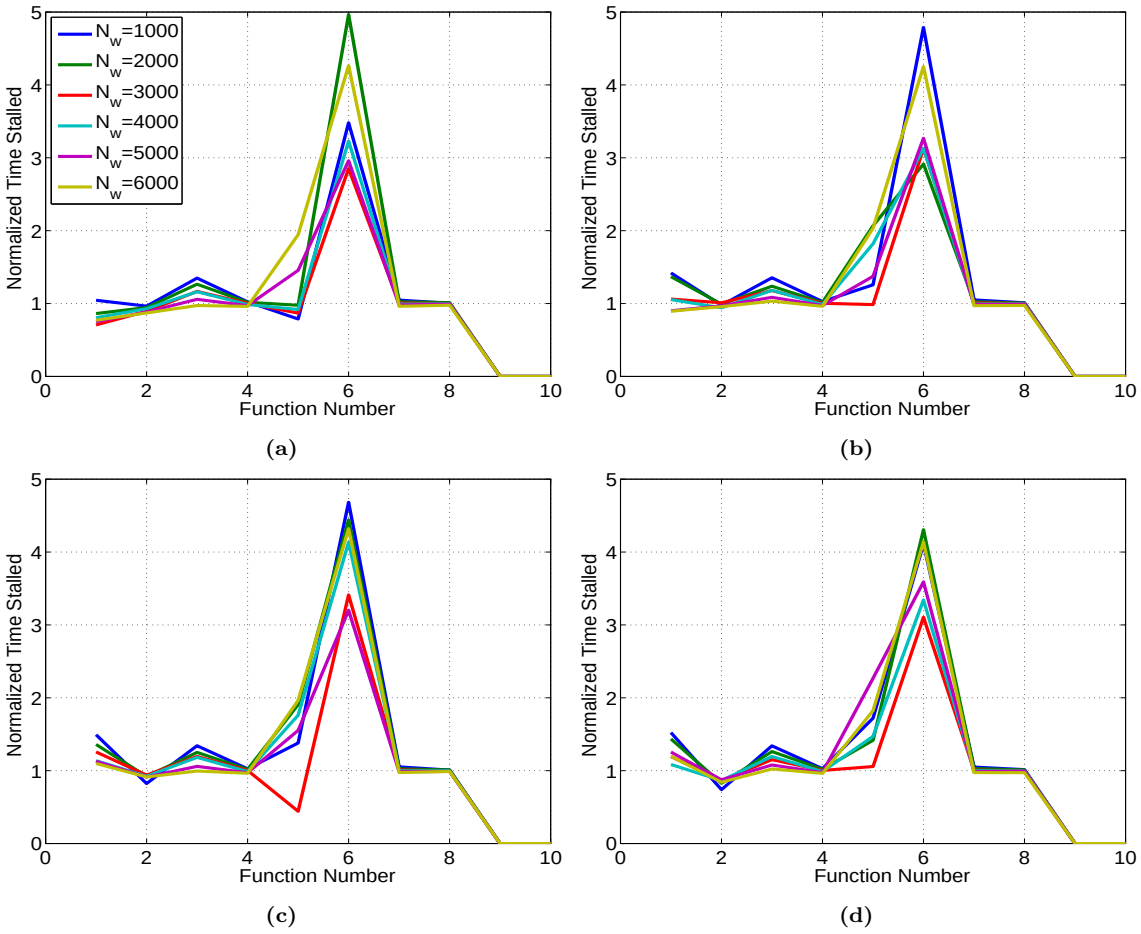


Figure B.10: Optimizer Efficiency Evaluation Parameters Time Stalled (a). $J_{ref} = 0.5$; (b). $J_{ref} = 1.0$; (c). $J_{ref} = 1.5$; (d). $J_{ref} = 2.0$. For most functions, longer windows stall less with medium windows stalling the least on functions 5 and 6. Time stalled increases with increasing J_{ref} for functions 1, 5, and 6.

B.3 Reseeding

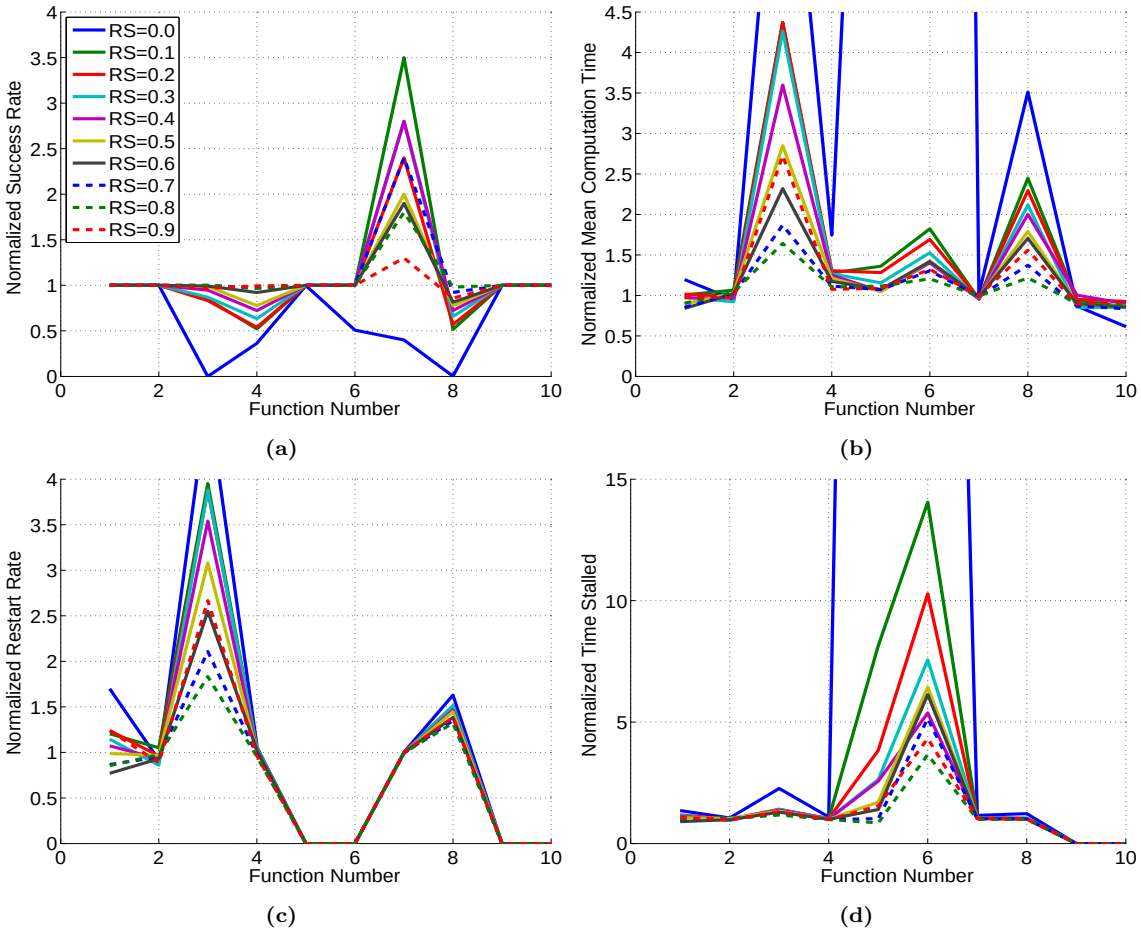


Figure B.11: Reseeding Rate Trade Study (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate; (d). Time Stalled. Significant reduction in performance as reseeding rate decreases. Some loss in performance is also present if reseeding rate is too high. Meta-optimization largely ineffective without any reseeding.

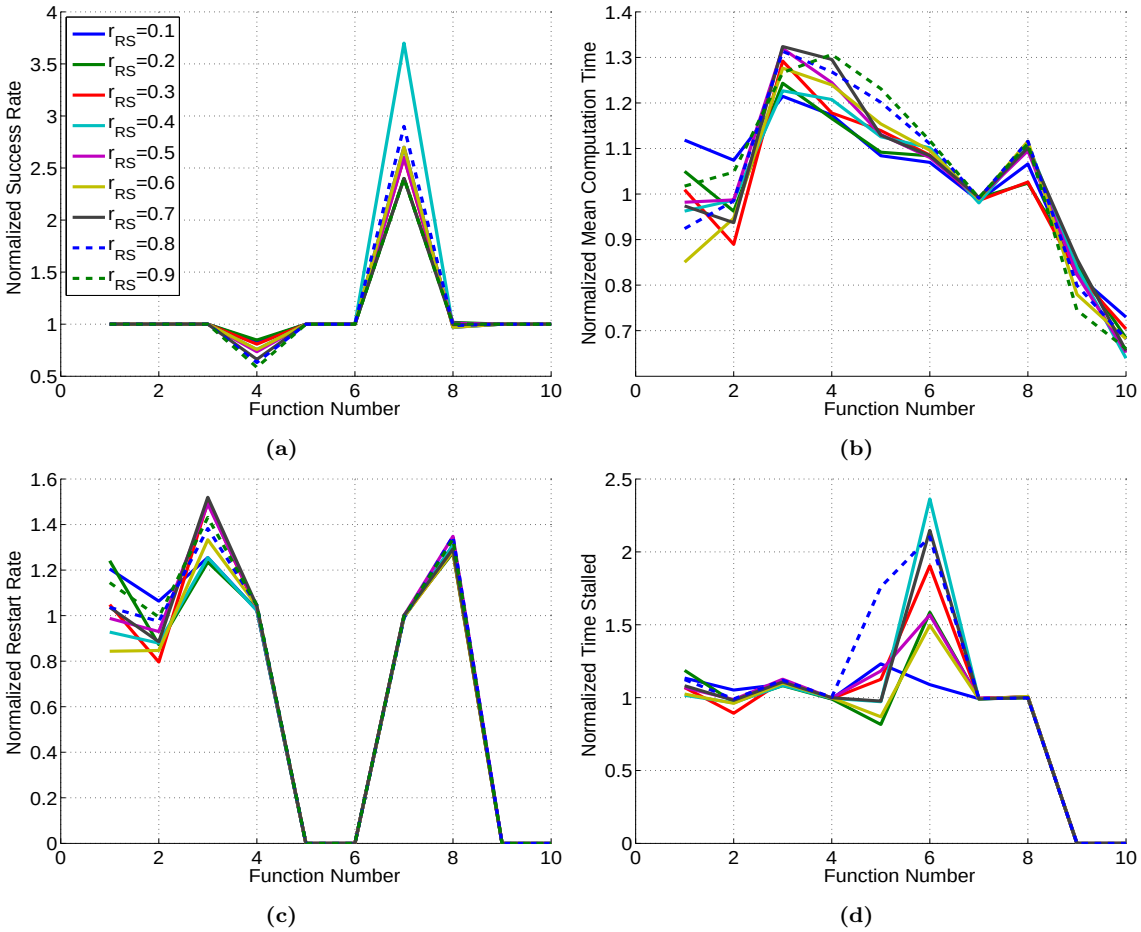


Figure B.12: Reseeding Refinement Rate Trade Study (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate; (d). Time Stalled. Lower reseeding rates are generally better on success rate while medium rates are fastest. Lower rates are also best on time stalled.

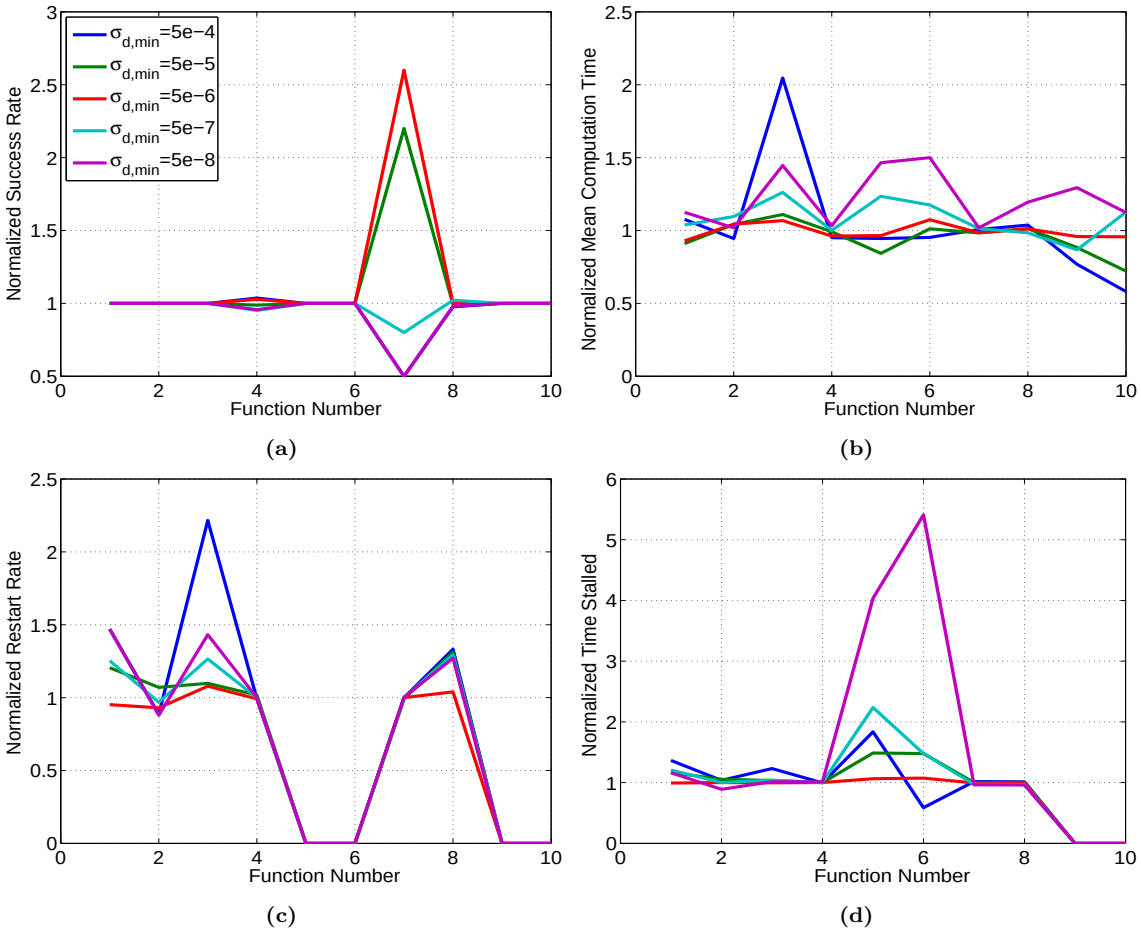


Figure B.13: Reseeding Diversity Threshold Trade Study (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate; (d). Time Stalled. Significant loss in performance on all metrics when the diversity threshold is too low or too high. Intermediate values perform best overall.

B.4 Restarting

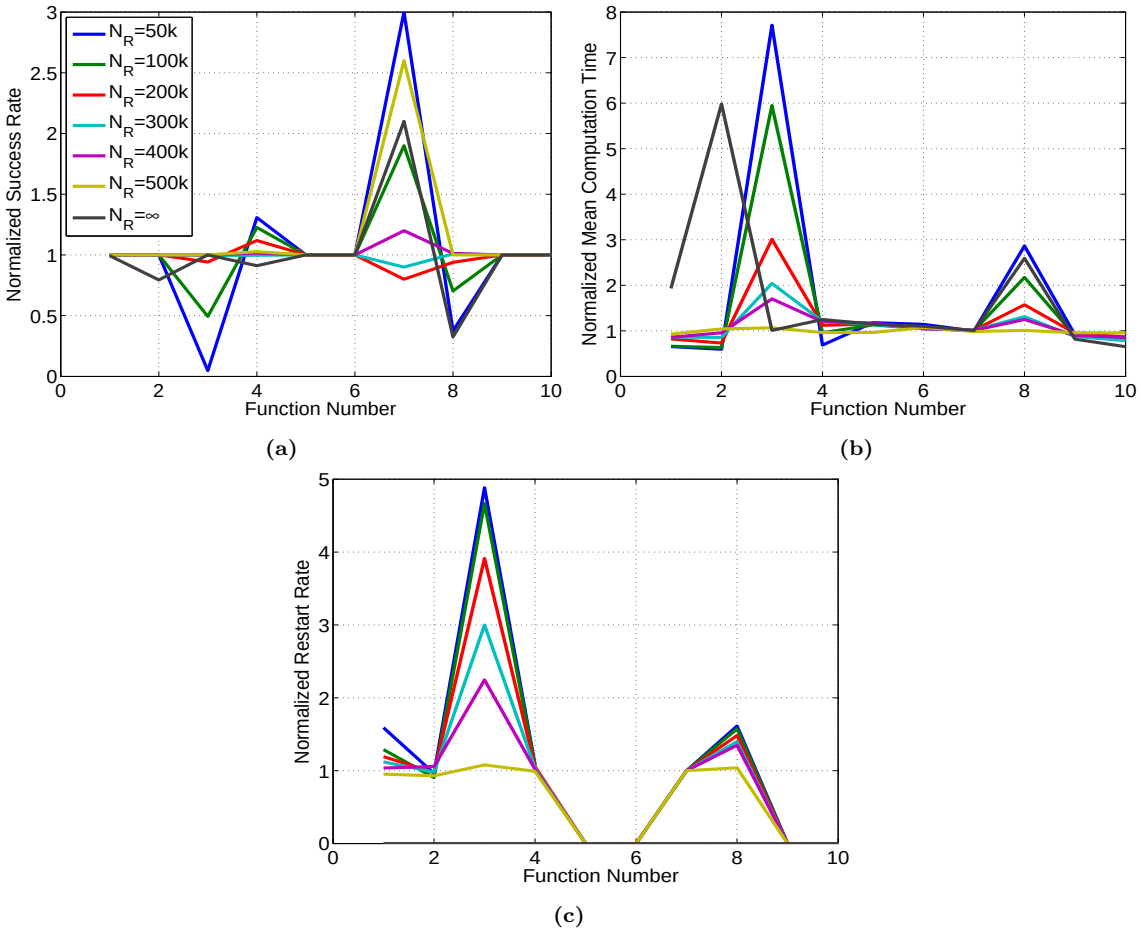


Figure B.14: Restarting Threshold Trade Study (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate. A lower restarting threshold was better on some functions, but significantly worse overall. Without restarts, meta-optimization can still solve some problems reliably, but at the cost of increased computation time.

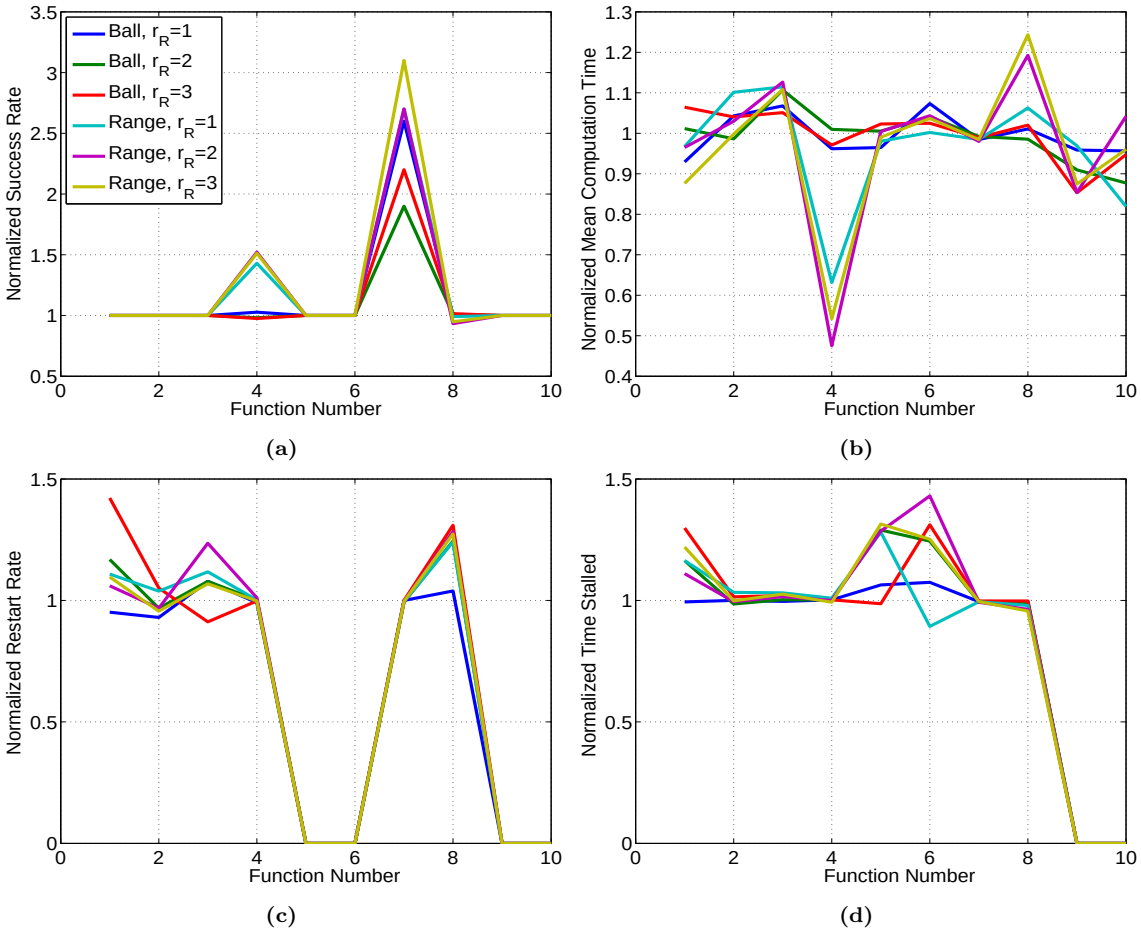


Figure B.15: Restarting Exclusion Zone Trade Study (a). Success Rate; (b). Mean Computation Time; (c). Restart Rate; (d). Time Stalled. Ball and range types of exclusion zone considered at three different radii. Significant improvement on function 4 with the range type exclusion zone. Larger radius generally performs better but may be too restrictive on the parameter space.

REFERENCES

[1] M. B. Tischler, "System identification requirements for high-bandwidth rotorcraft flight control system design," *Journal of Guidance, Control, and Dynamics*, vol. 13, no. 5, pp. 835–841, 1990.

[2] M. B. Tischler, "System identification methods for aircraft flight control development and validation," *Advances in Aircraft Flight Control*, pp. 35–69, 1996.

[3] P. C. Murphy and V. Klein, "Transport aircraft system identification from wind tunnel data," in *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Honolulu, Hawaii, 2008.

[4] A. Dorobantu, A. Murch, B. Mettler, and G. Balas, "System identification for small, low-cost, fixed-wing unmanned aircraft," *Journal of Aircraft*, vol. 50, no. 4, pp. 1117–1130, 2013.

[5] V. Klein and E. A. Morelli, *Aircraft System Identification: Theory and Practice*. Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2006.

[6] F. Fresconi, I. Celmins, and B. Howell, "Obtaining the aerodynamic and flight dynamic characteristics of an asymmetric projectile through experimental spark range firings," in *AIAA Atmospheric Flight Mechanics Conference*, Portland, Oregon, 2011.

[7] C. Montalvo and M. Costello, "Estimation of projectile aerodynamic coefficients using coupled cfd/rbd simulation results," in *AIAA Atmospheric Flight Mechanics Conference*, Toronto, Ontario, 2010.

[8] K. W. Iliff, "Parameter estimation for flight vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 12, no. 5, pp. 609–622, 1989.

[9] X. Deng, "System identification based on particle swarm optimization algorithm," in *International Conference on Computational Intelligence and Security*, Wuhan, China, 2009.

[10] A. Alfi and H. Modares, "System identification and control using adaptive particle swarm optimization," *Applied Mathematical Modelling*, vol. 35, pp. 1210–1221, 2011.

[11] M. Schwaab, E. C. J. Biscaia, J. L. Monteiro, and J. C. Pinto, "Nonlinear parameter estimation through particle swarm optimization," *Chemical Engineering Science*, vol. 63, pp. 1542–1552, 2008.

[12] K. Kristinsson and G. A. Dumont, "System identification and control using genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 5, pp. 1033–1046, 1992.

[13] S. Mondoloni and L. S. Litwin, "Parameter estimation using genetic algorithms for air traffic management simulation scenario generation," in *AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Forth Worth, Texas, 2010.

[14] M. S. Whorton, "Closed-loop system identification with genetic algorithms," *Journal of Aerospace Computing, Information, and Communication*, vol. 5, pp. 161–173, 2008.

[15] S. F. Campbell, N. T. Nguyen, J. Kaneshige, and K. Krishnakumar, "Parameter estimation for a hybrid adaptive flight controller," in *AIAA Infotech@Aerospace Conference*, Seattle, Washington, 2009.

[16] L. Ljung, *System Identification: Theory for the User*, Second. Upper Saddle River, New Jersey: Prentice Hall, 1999.

[17] K. J. Keesman, *System Identification: An Introduction*. London, England: Springer-Verlang London Limited, 2011.

[18] R. Doraiswami, C. Diduch, and M. Stevenson, *Identification of Physical Systems: Applications to Condition Monitoring, Fault Diagnosis, Soft Sensor and Controller Design*. Chichester, United Kingdom: John Wiley & Sons Ltd, 2014.

[19] R. V. Jategaonkar, *Flight Vehicle System Identification: A Time Domain Methodology*. Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2006.

[20] M. B. Tischler and R. K. Remple, *Aircraft and Rotorcraft System Identification: Engineering Methods with Flight Test Examples*. Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2006.

[21] W. Hathaway, M. Steinhoff, R. Whyte, D. Brown, J. Choate, and R. Aldergren, "Expert systems and ballistic range data analysis," in *AIAA Aerospace Ground Testing Conference*, Nashville, Tennessee, 1992.

[22] M. Fischer and W. Hathaway, "Aeroballistic research facility data analysis system (arfdas)," U.S. Air Force Armament Laboratory, Tech. Rep. AFATL-TR-88-48, 1988.

[23] J. Sahu, "Time-accurate numerical prediction of free flight aerodynamics of a finned projectile," in *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, San Francisco, California, 2005.

[24] B. T. Burchett, "Aerodynamic parameter identification for symmetric projectiles: Comparing gradient based and evolutionary algorithms," in *AIAA Atmospheric Flight Mechanics Conference*, Portland, Oregon, 2011.

[25] ——, "Aerodynamic parameter identification for symmetric projectiles: An improved gradient based method," *Aerospace Science and Technology*, vol. 30, pp. 119–127, 2013.

[26] V. Condaminet, F. Delvare, D. Choi, H. Demailly, C. Grignon, and S. Heddadj, "Identification of aerodynamic coefficients of a projectile and reconstruction of its trajectory from partial flight data," *Computer Assisted Method in Engineering and Science*, vol. 21, pp. 177–186, 2014.

[27] K. C. Massey and S. Silton, "Testing the maneuvering performance of a mach 4 projectile," in *AIAA Applied Aerodynamics Conference*, San Francisco, California, 2006.

[28] E. Scheuermann, M. Costello, S. Silton, and J. Sahu, "Aerodynamic characterization of a microspoiler system for supersonic finned projectiles," *Journal of Spacecraft and Rockets*, vol. 52, no. 1, pp. 253–263, 2015.

[29] D. Kim, L. Strickland, M. Gross, J. Rogers, M. Costello, F. Fresconi, and I. Celmins, "Actuator design and flight testing of an active microspoiler-equipped projectile," *Journal of Dynamic Systems, Measurement, and Control*, vol. 139, no. 10, pp. 111 002–1–15, 2017.

[30] F. Fresconi and T. Harkins, "Experimental flight characterization of asymmetric and maneuvering projectiles from elevated gun firings," *Journal of Spacecraft and Rockets*, vol. 49, no. 6, pp. 1120–1130, 2012.

[31] S. Silton, "Numerical experiments on finned bodies," in *AIAA Applied Aerodynamics Conference*, Atlanta, Georgia, 2014.

[32] S. Silton and F. Fresconi, "The effect of canard interactions on aerodynamic performance of a fin-stabilized projectile," in *AIAA Aerospace Sciences Meeting*, Kissimmee, Florida, 2015.

[33] J. Dykes, C. Montalvo, M. Costello, and J. Sahu, "Use of microspoilers for control of finned projectiles," *Journal of Spacecraft and Rockets*, vol. 49, no. 6, pp. 1131–1140, 2012.

[34] J. Stahl, M. Costello, and J. Sahu, "Projectile aerodynamic coefficient estimation using integrated cfd/rbd and flight control system modeling," in *AIAA Atmospheric Flight Mechanics Conference*, Chicago, Illinois, 2009.

[35] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[36] J. Nocedal and S. J. Wright, *Numerical Optimization*, Second. New York, NY: Springer Science+Business Media, LLC, 2006.

[37] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, Second. Society for Industrial and Applied Mathematics, 2009.

[38] E.-G. Talbi, *Metaheuristics: From Deisgn to Implementation*. Hoboken, NJ: John Wiley & Sons, Inc., 2009.

[39] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, Second. New York, NY: Springer Science+Business Media, LLC, 2010.

[40] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Second. Frome, UK: Luniver Press, 2010.

[41] ——, *Engineering Optimization: An Introduction with Metaheuristic Applications*. Hoboken, NJ: John Wiley & Sons, Inc., 2010.

[42] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: Massachusetts Institute of Technology, 1996.

[43] K. Deb, *Multiobjective Optimization Using Evolutionary Algorithms*. Chichester, UK: John Wiley & Sons, Inc., 2001.

[44] S. Kiranyaz, T. Ince, and M. Gabbouj, *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Heidelberg, Germany: Springer-Verlag Berlin Heidelberg, 2014.

[45] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, New York: John Wiley & Sons, Inc, 1989.

[46] G. N. Vanderplaats, *Multidiscipline Design Optimization*. Vanderplaats Research and Development, 2007.

[47] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[48] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE Conference on Neural Networks*, Perth, Washington, 1995.

[49] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, 1998.

[50] J. Mockus, *Bayesian Approach to Global Optimization: Theory and Applications*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1989.

[51] ——, "Application of bayesian approach to numerical methods of global and stochastic optimization," *Journal of Global Optimization*, vol. 4, pp. 347–365, 1994.

[52] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998.

[53] D. Huang, T. T. Allen, W. I. Notz, and R. A. Miller, "Sequential kriging optimization using multiple-fidelity evaluations," *Structural and Multidisciplinary Optimization*, vol. 32, pp. 369–382, 2006.

[54] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," in *Proceedings of the IEEE*, vol. 104, 2016.

[55] J. R. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, pp. 65–118, 1976.

[56] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, pp. 77–95, 2002.

[57] M. G. Lagoudakis and M. L. Littman, "Algorithm selection using reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, San Francisco, CA, USA, 2000.

[58] W. Armstrong, P. Christen, E. McCreath, and A. P. Rendell, "Dynamic algorithm selection using reinforcement learning," in *International Workshop on Integrating AI and Data Mining*, Hobart, Tasmania, 2006.

[59] M. A. Munoz, M. Kirley, and S. K. Halgamuge, "The algorithm selection problem on the continuous optimization domain," *Computational Intelligence*

in Intelligent Data Analysis, Studies in Computational Intelligence, vol. 445, pp. 75–89, 2013.

[60] M. A. Munoz, Y. Sun, M. Kirley, and S. K. Halgamuge, "Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges," *Information Sciences*, vol. 317, pp. 224–245, 2015.

[61] C. Gomes and B. Selman, "Algorithm portfolios," *Artificial Intelligence*, vol. 126, pp. 43–62, 2001.

[62] M. Streeter and S. F. Smith, "New techniques for algorithm portfolio design," in *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, Helsinki, Finland, 2008.

[63] K. Leyton-Brown, E. Nudelman, G. Andrew, J. Mcfadden, and Y. Shoham, "A portfolio approach to algorithm selection," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[64] M. Gagliolo, V. Zhumatiy, and J. Schmidhuber, "Adaptive online time allocation to search algorithms," *Machine Learning: ECML 2004*, vol. 3201, pp. 134–143, 2004.

[65] M. Gagliolo and J. Schmidhuber, "Dynamic algorithm portfolios," in *Ninth International Symposium on Artificial Intelligence and Mathematics*, Ft. Lauderdale, Florida, 2006.

[66] ——, "Learning dynamic algorithm portfolios," *Annals of Mathematics and Artificial Intelligence*, vol. 47, no. 3, pp. 295–328, 2006.

[67] M. Gagliolo, "Online dynamic algorithm portfolios," Ph.D. Dissertation, Universit della Svizzera Italiana, 2010.

[68] M. Gagliolo and J. Schmidhuber, "Algorithm portfolio selection as a bandit problem with unbounded losses," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 2, pp. 49–86, 2011.

[69] P. Baudis and P. Posik, "Online black-box algorithm portfolios for continuous optimization," *Parallel Problem Solving from Nature-PPSN XIII*, pp. 40–49, 2014.

[70] P. Baudis, "Cocopf: An algorithm portfolio framework," in *Poster 2014 - the 18th International Student Conference on Electrical Engineering*, Prague, Czech Republic, 2014.

[71] F. Peng, K. Tang, G. Chen, and X. Yao, "Population-based algorithm portfolios for numerical optimization," *IEEE Transations on Evolutionary Computation*, vol. 14, no. 5, pp. 782–800, 2010.

[72] S. Y. Yuen, C. K. Chow, and X. Zhang, "Which algorithm should i choose at any point of the search: An evolutionary portfolio approach," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, Amsterdam, The Netherlands, 2013.

[73] S. Y. Yuen, C. K. Chow, X. Zhang, and Y. Lou, "Which algorithm should i choose: An evolutionary algorithm portfolio approach," *Applied Soft Computing*, vol. 40, pp. 654–673, 2016.

[74] K. Tang, F. Peng, G. Chen, and X. Yao, "Population-based algorithm portfolios with automated constituent algorithms selection," *Information Sciences*, vol. 279, pp. 94–104, 2014.

[75] S. Y. Yuen and X. Zhang, "On composing an algorithm portfolio," *Memetic Computing*, vol. 7, no. 3, pp. 203–214, 2015.

[76] T. Carchrae and J. C. Beck, "Applying machine learning to low-knowledge control of optimization algorithms," *Computational Intelligence*, vol. 21, no. 4, pp. 372–387, 2005.

[77] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker, "Surrogate-based analysis and optimization," *Progress in Aerospace Sciences*, vol. 41, pp. 1–28, 2005.

[78] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd Edition. New York, New York: John Wiley & Sons, Inc., 1998.

[79] G. E. P. Box and N. R. Draper, *Empirical Model-Building and Response Surfaces*, 3rd Edition. New York, New York: John Wiley & Sons, Inc., 1987.

[80] R. H. Myers, D. C. Montgomery, and y. e. Anderson-Cook Christine M., *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*.

[81] M. Bjorkman and K. Holmstrom, "Global optimization of costly nonconvex functions using radial basis functions," *Optimization and Engineering*, vol. 1, pp. 373–397, 2000.

[82] H.-M. Gutmann, "A radial basis function method for global optimization," *Journal of Global Optimization*, vol. 19, pp. 201–227, 2001.

[83] S. Jeong, M. Murayama, and K. Yamamoto, "Efficient optimization design method using kriging model," *Journal of Aircraft*, vol. 42, no. 2, pp. 413–420, 2005.

[84] T. Goel, R. T. Haftka, W. Shyy, and N. V. Queipo, "Ensemble of surrogates," *Structural and Multidisciplinary Optimization*, vol. 33, pp. 199–216, 2007.

[85] J. Muller and R. Piche, "Mixture surrogate models based on dempster-shafer theory for global optimization problems," *Journal of Global Optimization*, vol. 51, pp. 79–104, 2011.

[86] J. Muller and C. A. Shoemaker, "Influence of ensemble surrogate models and sampling strategy on the solution quality of algorithms for computationally expensive black-box global optimization problems," *Journal of Global Optimization*, vol. 60, pp. 123–144, 2014.

[87] F. A. C. Viana, R. T. Haftka, and L. T. Watson, "Efficient global optimization algorithm assisted by multiple surrogate techniques," *Journal of Global Optimization*, vol. 56, pp. 669–689, 2013.

[88] F. Neri, C. Cotta, and P. Moscato, *Handbook of Memetic Algorithms*. Berlin, Germany: Springer-Verlag, 2012.

[89] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Toward memetic algorithms," California Institute of Technology, Tech. Rep. Caltech Concurrent Computation Program 826, 1989.

[90] R. Hooke and T. A. Jeeves, "Direct search solution of numerical and statistical problems," *Journal of the ACM*, vol. 8, no. 2, pp. 212–229, 1961.

[91] L.-Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," in *IEEE Congress on Evolutionary Computation*, Hong Kong, China, 2008.

[92] H. H. Hoos and T. Stutzle, *Stochastic Local Search: Foundations and Applications*. San Francisco, California: Morgan Kaufmann Publishers, 2005.

[93] Q. Yuan, Z. He, and H. Leng, "A hybrid genetic algorithm for a class of global optimization problems with box constraints," *Applied Mathematics and Computation*, vol. 197, pp. 924–929, 2008.

[94] A. R. Yildiz, "An effective hybrid immune-hill climbing optimization approach for solving design and manufacturing optimization problems in industry," *Journal of Materials Processing Technology*, vol. 209, pp. 2773–2780, 2009.

[95] Y.-J. Wang, J.-S. Zhang, and Y.-F. Zhang, "A fast hybrid algorithm for global optimization," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Guangzhou, China, 2005.

[96] S. Muelas, A. LaTorre, and J.-M. Pena, "A memetic differential evolution algorithm for continuous optimization," in *International Conference on Intelligent Systems Design and Applications*, Pisa, Italy, 2009.

[97] Y. Gao and Y.-J. Wang, "A memetic differential evolution algorithm for high dimensional functions' optimization," in *Third International Conference on Natural Computation*, Haikou, China, 2007.

[98] D. Molina, M. Lozano, C. Garcia-Martinez, and F. Herrera, "Memetic algorithms for continuous optimisation based on local search chains," *Evolutionary Computation*, vol. 18, no. 1, pp. 27–63, 2010.

[99] S.-K. Fan, Y.-C. Liang, and E. Zahara, "Hybrid simplex search and particle swarm optimization for the global optimization of multimodal functions," *Engineering Optimization*, vol. 36, no. 4, pp. 401–418, 2004.

[100] ——, "A genetic algorithm and a particle swarm optimizer hybridized with nelder-mead simplex search," *Computers and Industrial Engineering*, vol. 50, pp. 401–425, 2006.

[101] Y. G. Petalas, K. E. Parsopoulos, and M. N. Vrahatis, "Memetic particle swarm optimization," *Annals of Operations Research*, vol. 156, pp. 99–127, 2007.

[102] W. Rafajlowicz, "A hybrid differential evolution-gradient optimization method," in *International Conference on Artificial Intelligence and Soft Computing*, Zakopane, Poland, 2015.

[103] B. Subudhi, D. Jena, and M. M. Gupta, "Memetic differential evolution trained neural networks for nonlinear system identification," in *IEEE Region 10 Colloquium and the Third International Conference on Industrial and Information Systems*, Kharagpur, India, 2008.

[104] M. M. Noel and T. C. Jannett, "Simulation of a new hybrid particle swarm optimization algorithm," in *Southeastern Symposium on Systems Theory*, Atlanta, Georgia, 2004.

[105] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "A fast adaptive memetic algorithm for online and offline control design of pmsm drives," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 37, no. 1, pp. 28–41, 2007.

[106] ——, "Application of memetic differential evolution frameworks to pmsm drive design," in *IEEE Congress on Evolutionary Computation*, Hong Kong, China, 2008.

[107] A. Caponio, F. Neri, and V. Tirronen, "Super-fit control adaptation in memetic differential evolution frameworks," *Soft Computing*, vol. 13, pp. 811–831, 2009.

[108] V. Tirronen, F. Neri, and T. Karkkainen, "A memetic differential evolution in filter design for defect detection in paper production," in *Workshops on Applications of Evolutionary Computation*, Valencia, Spain, 2007.

[109] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and L. M. Wang, "An improved ga and a novel pso-ga-based hybrid algorithm," *Information Processing Letters*, vol. 93, pp. 255–261, 2005.

[110] P. S. Shelokar, P. Siarry, V. K. Jayaraman, and B. D. Kulkarni, "Particle swarm and ant colony algorithms hybridized for improved continuous optimization," *Applied Mathematics and Computation*, vol. 188, pp. 129–142, 2007.

[111] K. Muranaka and E. Aiyochi, "Computational properties of hybrid methods with pso and de," *Electronics and Communications in Japan*, vol. 97, no. 4, pp. 1128–1135, 2014.

[112] R. P. Parouha and K. N. Das, "A robust memory based hybrid differential evolution for continuous optimization problem," *Knowledge-Based Systems*, vol. 103, pp. 118–131, 2016.

[113] M. A. Ahandani, M.-T. Vakil-Baghmisheh, and M. Talebi, "Hybridizing local search algorithms for global optimization," *Computational Optimization and Applications*, vol. 59, pp. 725–748, 2014.

[114] J. R. Raol, J. Singh, G. Girija, and E. Institution of Electrical, *Modelling and Parameter Estimation of Dynamic Systems.* London, United Kingdom: The Institution of Engineering and Technology, 2004.

[115] I. Hughes and T. Hase, *Measurements and Their Uncertainties : A Practical Guide to Modern Error Analysis.* New York, New York: Oxford University Press, 2010.

[116] B. Etkin, *Dynamics of Atmospheric Flight.* Mineola, New York: Dover Publications, 2000.

[117] *Prodas version 3 technical manual*, Arrow Tech Associates, Burlington, Vermont, 2000.

[118] A. Leonard, J. Rogers, and J. Sahu, "Aerodynamic optimization of microspoiler actuators for guided projectiles," *Journal of Spacecraft and Rockets*, 2017.

[119] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory landscape analysis," in *Genetic and Evolutionary Computation Conference*, Dublin, Ireland, 2011.

[120] J. Garnier and L. Kallel, "Efficiency of local search with multiple local optima," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 1, pp. 122–141, 2002.

[121] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, no. 2, pp. 149–154, 1964.

[122] A. R. Mehrabian and C. Lucas, "A novel numerical optimization algorithm inspired from weed colonization," *Ecological Informatics*, vol. 1, pp. 355–366, 2006.

[123] H. Bolandi, M. H. Ashtari Larki, S. H. Sedighy, M. S. Zeighami, and M. Esmailzadeh, "Estimation of simplified general perturbations model for orbital elements from global positioning system data by invasive weed optimization algorithm," *Journal of Aerospace Engineering*, vol. 229, no. 8, pp. 1384–1394, 2014.

[124] S. Karimkashi and A. A. Kishk, "Invasive weed optimization and its features in electromagnetics," *IEEE Transactions on Antennas and Propagation*, vol. 58, no. 4, pp. 1269–1278, 2010.

[125] Z. D. Zaharis, C. Skeberis, T. D. Xenos, P. I. Lazaridis, and J. Cosmas, "Design of a novel antenna array beamformer using neural networks trained by modified adaptive dispersion invasive weed optimization based data," *IEEE Transactions on Broadcasting*, vol. 59, no. 3, pp. 455–460, 2013.

[126] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, pp. 1155–1173, 2008.

[127] J. A. Nelder and R Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[128] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the nelder-mead simplex method in low dimensions," *SIAM Journal of Optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[129] O. Hajji, S. Brisset, and P. Brochet, "A new tabu search method for optimization with continuous parameters," *IEEE Transactions on Magnetics*, vol. 40, no. 2, pp. 1184–1187, 2004.

[130] R. Chelouah and P. Siarry, "Tabu search applied to global optimization," *European Journal of Operational Research*, vol. 123, pp. 256–270, 2000.

[131] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Academic Press, 1981.

[132] K. Zielinksi, D. Peters, and R. Laur, "Stopping criteria for single-objective optimization," in *Proceedings of the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore, 2005.

[133] P. Siarry and G. Berthiau, "Fitting of tabu search to optimize functions of continuous variables," *International Journal For Numerical Methods in Engineering*, vol. 40, pp. 2449–2457, 1997.

[134] C. Unsal, "Intelligent navigation of autonomous vehicles in an automated highway system: Learning methods and interacting vehicles approach," Ph.D. Dissertation, Virginia Polytechnic Institute and State University, 1998, ch. 3.

[135] J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, Tech. Rep. Technical Report 201311, 2013.

[136] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems," *Journal of Global Optimization*, vol. 31, pp. 635–672, 2005.

[137] R. Tanabe and A. S. Fukunaga, "Improving the search performance of shade using linear population size reduction," in *IEEE Congress on Evolutionary Computation*, Beijing, China, 2014.

[138] C. Xu, H. Huang, and S. Ye, "A differential evolution with replacement strategy for real-parameter numerical optimization," in *IEEE Congress on Evolutionary Computation*, Beijing, China, 2014.

[139] S. M. Elsayed, R. A. Sarker, D. L. Essam, and N. M. Hamza, "Testing united multi-operator evolutionary algorithms on the cec2014 real-parameter numerical optimization," in *IEEE Congress on Evolutionary Computation*, Beijing, China, 2014.

[140] R. Tanabe and A. S. Fukunaga, "Success-history based parameter adaption for differential evolution," in *IEEE Congress on Evolutionary Computation*, Cancun, Mexico, 2013.

VITA

Matthew Gross was born and raised in Fort Washington, PA in the suburbs of Philadelphia, PA. In 2012, he received a Bachelor of Science degree in Aerospace Engineering from the University of Maryland, College Park. After graduating, Matthew moved to Atlanta, GA to begin his studies at the Georgia Institute of Technology. He received a Masters of Science degree in Aerospace Engineering in 2014 and continued his studies to pursue a Doctor of Philosophy. His research interests include flight dynamic modeling, control system design, system identification, and engineering optimization.