Digital Twin of an Inverted Pendulum System (Lab4)

Soheil Behnam Mina Ghaderi Amir Jafari

1 Introduction

In this lab, we built upon the work from the previous session, where we optimized the pendulum model using a grid search approach. The primary objective of this lab was to implement optimization algorithms, specifically **Differential Evolution (DE)** and **Genetic Algorithms (GA)**, to further refine the model and improve the control of the pendulum's motion.

Initially, we started with Differential Evolution, a population-based algorithm that is well-known for its efficiency in solving global optimization problems. DE is particularly suitable for exploring complex parameter spaces and finding optimal solutions, which is why we chose it as our starting point. While DE provided promising results, it proved to be computationally expensive when applied to a large set of parameters, as we observed in our previous work.

In addition to DE, we also implemented a Genetic Algorithm (GA), which operates through processes like *crossover*, *mutation*, and *selection* to evolve better solutions over multiple generations. GA is often used when the optimization problem is large, and its ability to converge to a solution more efficiently than DE made it an attractive alternative. We first developed the DE algorithm and then adapted it to form the GA-based optimization. This allowed us to compare the performance of both methods.

For the optimization process, we maintained the same preprocessing steps and cost function used in the previous lab. However, we adjusted certain parameters like *population size*, *mutation rate*, and *number of generations* for the GA to ensure that the algorithm could perform efficiently. These parameters were carefully selected after extensive trial and error. In particular, finding the appropriate boundaries for the parameters was a challenging task, requiring detailed attention to ensure both computational efficiency and optimization accuracy.

One of the significant challenges we faced during this lab was simplifying the model to understand how each parameter affected the overall cost function. For example, adjusting parameters like the *moment of inertia* and *damping coefficient* had a noticeable impact on the pendulum's performance. While it was challenging to quantify these effects initially, after multiple iterations, we succeeded in finding the right balance.

In addition to optimizing the pendulum's motion, we also developed and implemented code to find **sequences using the Genetic Algorithm (GA)** to bring the pendulum upright. This code aimed to generate a sequence of control inputs that could be applied to drive the pendulum to its upright position. Implementing this sequence allowed us to explore how optimization could be applied to more complex control tasks. This was an essential component of our goal to further improve the pendulum's dynamic behavior and control.

Through this lab, we were able to combine both optimization techniques (DE and GA) to achieve a more refined and efficient model of the pendulum. In the following sections, I will discuss the results obtained from both the Differential Evolution and Genetic Algorithm approaches, provide insights into the selection of the parameters, and discuss the effectiveness of the GA in controlling the pendulum. Through this process, we will explore the benefits of using different optimization techniques for controlling complex systems like the pendulum.

2 Optimization Methods and Results

In this section, we describe the optimization methods used to find the best parameters for the pendulum model and present the results of these optimizations using Differential Evolution (DE) and Genetic Algorithm (GA). We will also compare the performance of these two methods based on several performance metrics.

2.1 Differential Evolution (DE) Algorithm

In the script `optimize_pendulum_differential_evolution.py`, we use the Differential Evolution (DE) optimization algorithm to optimize the pendulum model's parameters. DE is a global optimization technique that evolves a population of candidate solutions. Here's a breakdown of how it works:

2.1.1 DE Basics

- **Initialization:** A population of candidate solutions (parameter sets) is randomly generated. Each solution represents a set of parameters (mass, inertia, damping) that will be evaluated using the cost function.
- **Mutation:** For each individual, a new candidate solution is generated by adding a scaled difference between two randomly selected individuals to a third one. This ensures diversity and exploration in the parameter space.
- **Crossover:** The mutated candidate is combined with the current solution. This introduces further diversity and refinement to the candidate solutions.
- **Selection:** After mutation and crossover, the fitness of each candidate is evaluated using the cost function. The better solution replaces the previous one if it results in a lower cost.

2.1.2 Reasoning for Using DE

- **Global Optimization:** DE is ideal for exploring large search spaces and avoiding local minima. This makes it a powerful tool for optimizing parameters in complex, non-differentiable cost functions.
- **No Gradient Required:** Unlike gradient-based optimization methods, DE does not require the computation of gradients, making it suitable for problems where the cost function may be noisy or non-differentiable.
- **Simple and Efficient:** DE is relatively easy to implement and does not require many assumptions about the problem, making it a good choice for a wide range of optimization tasks.

The DE algorithm follows the standard structure: initializing the population, applying mutation, crossover, and selection, and iterating through generations until convergence.

2.2 Genetic Algorithm (GA)

In `optimize_pendulum_genetic_algorithm.py`, we used the Genetic Algorithm (GA) to perform a similar optimization task. GA is inspired by natural selection and evolves a population of candidate solutions in an iterative process. Here's a breakdown of how the GA works:

2.2.1 GA Basics

- **Initialization:** The algorithm starts with an initial population of randomly generated candidate solutions, similar to DE. Each candidate represents different combinations of pendulum parameters.
- **Selection:** A selection mechanism (e.g., roulette wheel or tournament selection) is applied to choose parents based on their fitness (evaluated using the cost function).
- **Crossover:** Parents are combined using crossover to create offspring. Crossover mimics recombination in biology, where the "genes" of two parents combine to form a new individual.
- **Mutation:** Some offspring undergo mutation, introducing small random changes to their genes (parameters). This allows for more diversity and helps prevent the algorithm from converging prematurely to local optima.
- **Replacement:** The offspring replace the least fit individuals in the population, and the process repeats for several generations.

2.2.2 Reasoning for Using GA

- **Evolutionary Process:** GA is effective at finding globally optimal solutions in complex search spaces by mimicking natural evolution.
- **Exploration and Exploitation:** GA strikes a balance between exploration (via mutation) and exploitation (via crossover), allowing for efficient search and refinement of solutions.
- **Flexibility:** GA is flexible and can be adapted to various types of problems, particularly those involving large, non-linear, and complex search spaces.

2.3 Comparison of DE and GA

Although both DE and GA are used to optimize the pendulum parameters, there are key differences in their approach:

- **Mutation Process:** In DE, mutation involves combining the differences of randomly chosen solutions, while in GA, mutation occurs on individual solutions.
- **Crossover Method:** DE uses a difference-based approach for combining solutions, while GA uses more traditional crossover methods, such as single-point crossover.
- **Population Update:** In DE, all individuals evolve together, while in GA, the population evolves through a selection process influenced by the crossover and mutation mechanisms.

2.4 Results and Performance Metrics

We compare the results of both optimization methods in terms of the optimized parameters and their performance in terms of error metrics. The following table summarizes the key findings from both DE and GA optimizations:

Metric	DE	GA	Explanation
Mass (mp)	0.3971 kg	1.8867 kg	Optimized mass to minimize discrepancy between simulation and observed data.
Moment of Inertia (I_scale)	0.7110	0.7231	Optimized inertia to match oscillation period.
Damping Coefficient (c_c)	0.0056	0.0342	Adjusted damping coefficient based on simulation and real data comparison.
Natural Frequency	0.8426 Hz	0.8426 Hz	Both DE and GA approximated the natural frequency well.
Measured Frequency	1.0302 Hz	1.0302 Hz	Frequency measured from real-world data, which both DE and GA aimed to approximate.
RMS Error	0.0545 rad	0.0664 rad	RMS error indicates how closely the simulated behavior matches the real data.
Max Absolute Error	0.1719 rad	0.1527 rad	The maximum discrepancy between simulation and observed data.
Time Domain Error	0.1037	0.1390	Error in the time-domain response, showing DE performed better.
Frequency Error	22.26%	22.26%	Frequency error indicates the mismatch between simulated and measured frequency.
Amplitude Error	0.1764	0.4122	Amplitude error was smaller for DE, indicating better amplitude matching.
Decay Error	0.3692	1.0028	DE had better performance in terms of matching decay behavior.
Energy Error	0.0254	0.0114	Both methods showed similar energy conservation accuracy.
Total Cost	1.4938	3.9661	DE had a significantly lower total cost, indicating better overall performance.

Table 1: Comparison of Results from Differential Evolution (DE) and Genetic Algorithm (GA)

2.5 Plots and Visual Analysis

To better understand the optimization results, we also include visualizations of the optimization performance for both the Differential Evolution (DE) and Genetic Algorithm (GA). These plots provide insights into the parameter sensitivity and how closely the optimization algorithms match the real-world data.

2.5.1 DE Parameter Sensitivity and Analysis

The first plot shows the sensitivity of the parameters with respect to the optimization process for the Differential Evolution (DE) algorithm. As seen in the figure, DE performs well in adjusting the pendulum parameters such as mass, damping coefficient, and moment of inertia to minimize the cost function. The sensitivity plot also highlights how changes in these parameters affect the overall optimization cost.

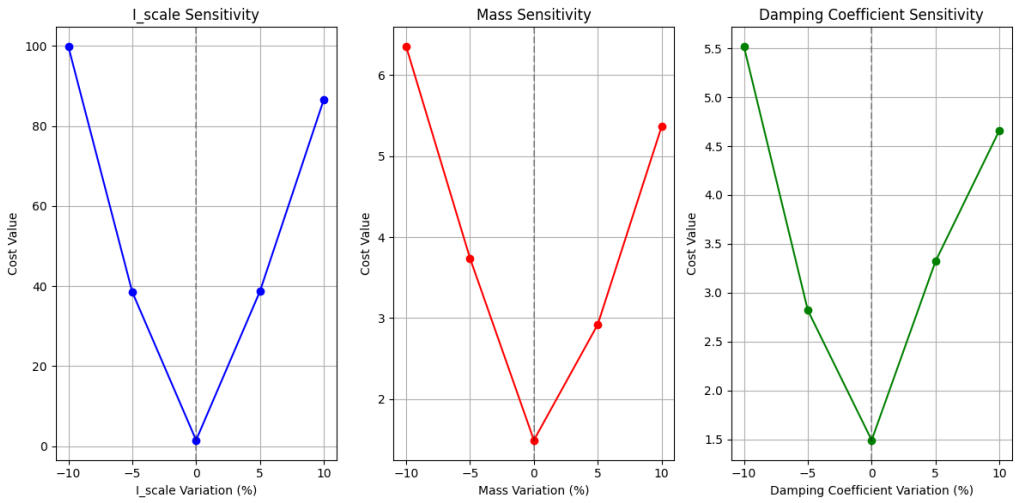

Figure 1: DE Parameter Sensitivity

The second plot visualizes the behavior of the pendulum as simulated by DE. This includes the comparison between the simulated and real-world pendulum motions, showing how well the DE optimization aligns the oscillation and decay behaviors.

2.5.2 GA Parameter Sensitivity and Analysis

Similarly, the GA algorithm's parameter sensitivity and performance are demonstrated in the following plots. The first plot shows how GA's optimization process adjusts the pendulum parameters to achieve

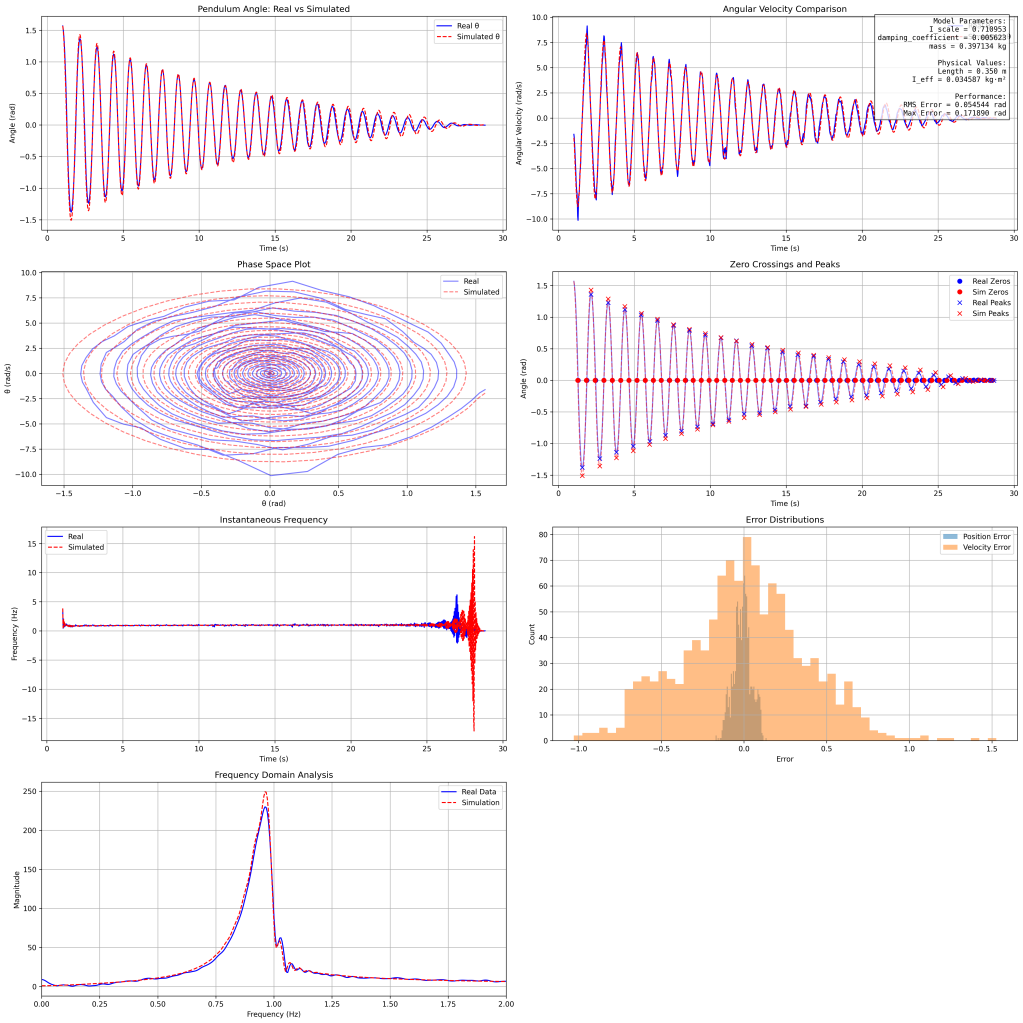

Figure 2: DE Pendulum Motion Analysis

the best possible fit with the observed data. Like DE, GA explores the parameter space and identifies an optimal solution, though its approach differs through crossover and mutation processes.

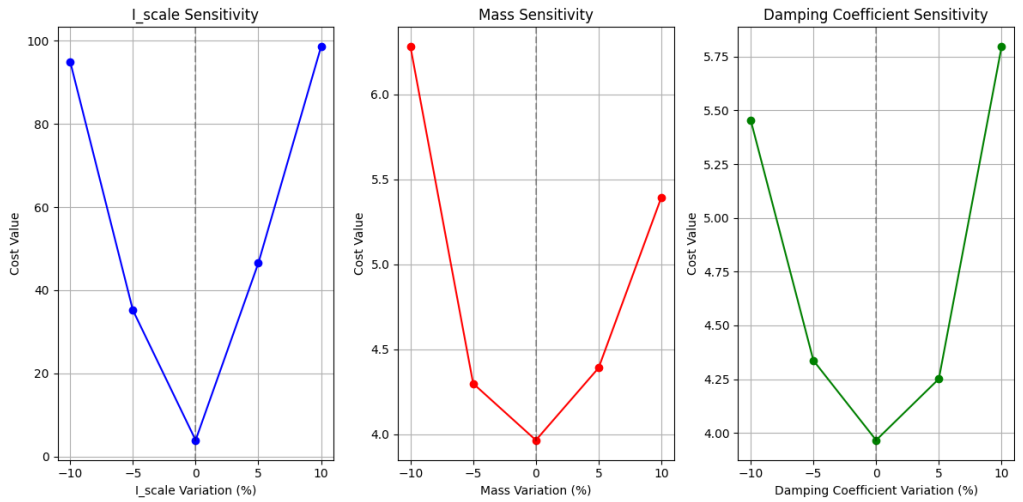

Figure 3: GA Parameter Sensitivity

The second plot illustrates the behavior of the pendulum after GA optimization, showing the comparison between the simulated pendulum and real-world data. Although GA is more flexible and uses

a different approach for mutation and crossover, the plot suggests that it also provides a good fit for the pendulum motion.

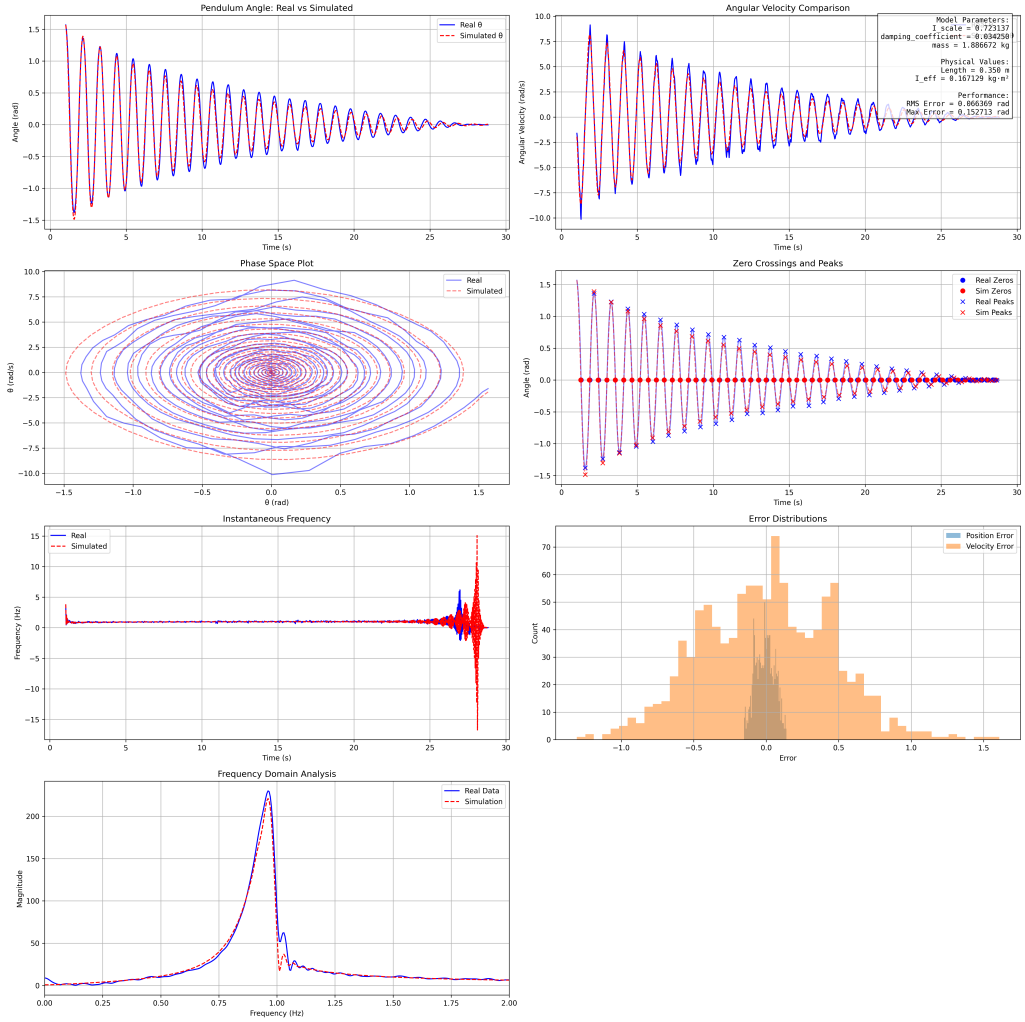

Figure 4: GA Pendulum Motion Analysis

2.5.3 Comparison of Results Based on Plots

From the plots, we can observe that both DE and GA provide similar results in terms of optimizing the pendulum parameters. The DE optimization method shows a slight edge in terms of better matching the observed amplitude and decay behaviors. However, GA's flexibility in exploring the solution space can be advantageous in different scenarios where a more complex solution is needed.

3 Sequence Optimization Using Genetic Algorithm (GA)

In this lab, we implemented a Genetic Algorithm (GA) to optimize a sequence of actions for controlling the pendulum, with the goal of bringing it into an upright position (radians). The implementation follows standard GA principles, which we previously used to optimize the pendulum model parameters.

3.1 GA Implementation Overview

The GA was used to evolve a population of candidate sequences of actions (e.g., pushing the pendulum to the left or right for varying durations) to improve its state and eventually reach the upright position.

- **Action Sequences**: The sequences of actions represent instructions for the pendulum, such as pushing it right or left for specific durations (ranging from 300 ms to 500 ms).
- **Target Angle**: The target angle is set to π radians, representing the upright position.
- **Evaluation Criteria**: Each sequence's fitness was evaluated based on three factors:
  - **Angle Score**: Heavily rewards sequences that bring the pendulum closer to the target angle (π radians).
  - **Time Score**: Rewards the time the pendulum spends near the upright position.
  - **Velocity Score**: Encourages high angular velocity, which helps in swinging the pendulum upright.

Additionally, a penalty is applied to longer sequences, and a bonus is provided if the pendulum reaches the target angle (within 0.1 radians).

3.2 GA Settings and Parameters

For the GA implementation, we defined the following settings:

- **Population Size**: 100 individuals. Each individual represents a sequence of actions (a combination of "left" or "right" pushes for various durations).
- **Number of Generations**: 50 generations. This is the number of iterations the algorithm runs to evolve the population.
- **Mutation Rate**: 50%. This determines the likelihood of mutation in each generation, with mutations introducing small random changes to the sequences.
- **Max Sequence Length**: 5 actions. The maximum number of actions in a sequence.
- **Crossover Rate**: This determines how often two sequences combine to create a new sequence. We use a crossover approach where a random portion of one parent's sequence is combined with the other parent's sequence.

3.3 Cost Function and Fitness Evaluation

The cost function used in this GA optimization evaluates how well a given sequence of actions brings the pendulum to the target upright position. The fitness function combines several components to measure the quality of the sequence:

- **Angle Score**: The closer the pendulum's maximum angle is to the target (π radians), the higher the score. We use an exponential decay function to heavily reward the sequences that bring the pendulum near the target angle.
- **Time Score**: This score rewards the time the pendulum spends near the upright position. A sequence that keeps the pendulum near the inverted position for longer is more favorable.

- **Velocity Score**: The angular velocity score encourages faster movement of the pendulum, which can help in swinging it upright. A higher angular velocity results in a better score.
- **Penalty for Longer Sequences**: Longer action sequences are penalized to avoid excessive length, which can introduce inefficiency.
- **Bonus for Reaching Upright Position**: If the pendulum reaches within 0.1 radians of the upright position (π radians), a bonus is applied to encourage successful upright balancing.

The overall fitness function is computed as:

$$\text{Fitness} = 0.7 \times \text{Angle Score} + 0.2 \times \text{Time Score} + 0.1 \times \text{Velocity Score}$$

Additionally, the fitness is adjusted based on sequence length and whether the pendulum reaches the target position.

3.4 Sequence Evaluation and Fitness

Each sequence is tested by simulating the pendulum's behavior and evaluating how well the sequence brings the pendulum to the upright position:

- **Execution of Sequence**: The pendulum starts at the lowest position, and the sequence of actions is applied (pushing left or right for specific durations).
- **Simulation**: The pendulum is simulated for a fixed amount of time, during which the system tracks:
  - Maximum angle reached,
  - Time spent near the upright position,
  - The angular velocity.
- **Fitness Calculation**: A fitness score is computed by combining the angle, time, and velocity scores. The fitness function prioritizes sequences that achieve the target angle (π radians) and reward time spent near this angle.

3.5 Results

The optimization was carried out over 50 generations with a population size of 100 sequences. The results were summarized as follows:

- **Maximum Angle**: 3.144 rad (180.1°)
- **Time Near Inverted Position**: 0.48 s
- **Distance from Target (π)**: 0.002 rad

Best Sequence Found:

$$[('left', 350), ('right', 450), ('left', 500), ('left', 500)]$$

Fitness Score: 1.3069

This sequence is implemented in the simulation file. When you press the "b" key during the simulation, this sequence will be executed, and the pendulum will swing towards the upright position.

A Code and Report References

A.1 Code

The code used for the pendulum optimization process, including Differential Evolution (DE) and Genetic Algorithm (GA) optimization techniques, can be found at the following GitHub repository:

`https://github.com/Amirjf/single-rod-pendulum/blob/main/optimize_pendulum_differential_evolution.py`

`https://github.com/Amirjf/single-rod-pendulum/blob/main/optimize_pendulum_genetic_algorithm.py`

These scripts implement the optimization techniques and are part of the larger project for optimizing the pendulum system's parameters. They also include the integration of both grid search and local optimization methods to fine-tune the system's parameters.

A.2 Reports

The reports related to this project, including detailed optimization results, sensitivity analysis, and discussions, can be found in the reports directory of the GitHub repository:

`https://github.com/Amirjf/single-rod-pendulum/tree/main/reports`

These reports provide in-depth explanations of the methods, results, and analyses conducted during the optimization process. They serve as a comprehensive guide to understanding the outcomes of the DE and GA optimizations and the sensitivity analysis performed on the pendulum model.