The University of Sheffield

ACS6124 Decision Systems Assignment

Leander Stephen Desouza

Registration Number: 230118120

Department of Automatic Control Systems Engineering

Contents

List of Figures

List of Tables

Executive Summary

Overview and Tuning Process Findings

This report highlights the procedure to optimize the gains of the Proportional-Integral Controller of a proposed vehicle propulsion system. The tuning parameters of the PI controller need to follow the specific priorities and goals set by the client. Subsequent tradeoffs and suggested controller architecture changes are also provided in situations where all of the expected outcomes fall short.

Two variations of the client's requirements are tested on the optimization process of the controller. These specific values are mentioned in the goals section of the Appendix. When the goals are not initially modified, all metrics of the performance evaluations are met except the rise time of the system. Here, we observe a rise time error of about 20%. When the control effort energy is reduced as per the second requirement, we find that the transient response worsens further. Here, the rise time fails with an error percentage of about 400%, whereas the peak time error increases to about 180%. This can lead the system to have more inertia and respond with delayed behaviour. We can incorporate the recommendations into the controller for the system to promote a sustainable design.

Recommendations for Future Action

1. Derivative Component Addition: To improve the transient response, we incorporate a derivative component into the PI controller to dampen the system's response. This reduces the overshoot and settling time of the system. This can be in the form of serial resistors and parallel capacitors connected to an op-Amp.

2. Feedforward Controller: If the vehicle propulsion has predictable levels of noise, a feedforward controller can be added in conjunction with the PI controller to dampen the system response to reduce the prolonged spikes.

3. Upgrading actuator quality: Using actuators with less starting torque and lower moment of inertia can reduce the rise time and improve the transient response.

1 Multi-objective optimization for Engineering Design

1.1 Introduction to Decision Systems for Engineering Design

Decision systems for Engineering Design is a field that includes multiple disciplines, from computer science to management and general engineering. They aid in the creation of various models for computation to assist in the decision-making process during the engineering design phase. The ultimate target of these systems is to devise a method to improve efficiency and reduce overall costs for the design phase. As a result, these systems are heavily incorporated in a plethora of engineering fields, such as industrial, mechanical and civil lines of work.

1.2 MOO compared with other decision-making methods

1. **Multiple Objectives and Visualization**: As present in real-world scenarios, MOO can handle multiple objectives simultaneously. This is very useful when considering tradeoffs between decisions due to conflicting objectives. In addition, MOO can display a Pareto front to help clients grasp the relationships between the corresponding objectives.

2. **Incorporation of client preferences**: In MOO, the decision-makers priorities and goals are included in the pursuit of the solution, whereas other decision-making methods fail to grasp the client's needs.

3. **Pareto optimality**: Since MOO deals with various metrics, it does not provide a single solution but a collection of Pareto-optimal ones. This means that the solution can no longer improve without violating any other of the client's needs.

1.3 Literature Review of MOO being incorporated into EVs

This paper [2] focuses on the optimal allocation of the size of various energy resources to challenge issues such as power losses and voltage levels. In this literature [3], a selfoptimizing power matching strategy is proposed, evaluating the efficiency of energy and the degradation of the fuel cell. This literature [4] uses the resizing of the motor, battery and longitudinal vehicles as reference vehicle characteristics as the target EV objectives. This literature [5] mentions various topologies for batteries of EVs using MOO. Various topologies are considered, taking motors and axles into account. This paper [6] compares and contrasts various MOO algorithms in terms of their suitability, efficiency, strengths and weaknesses and provides recommendations for each use case.

1.4 Comparison of the classes in solving MOO problems

1. **Pareto-based**: These systems are based on Pareto optimality, which indicates that if a solution is Pareto non-dominated, then no other solution is present that achieves another objective without violating the former. The Pareto-optimal set represents solutions with various tradeoffs between their goals. To find a **candidate approximation** set, Many-objective reverse mapping is used to develop a framework using the initial Pareto front set as its training data to reverse map the objectives.

2. **Decomposition-based**: These classes use the MOEA/D (Multi-Objective Evolutionary Algorithms) framework. This involves joining different objective functions into a scalar value using weighted vectors. To find a **candidate approximation** set, each weight vector is taken as a directional search to define a scalar function. The resulting solution of a single objective optimization problem gives exactly one Pareto optimal solution.

3. **Set-based**: These systems are a subset of MCDM (Multi-Criteria Decision Making) methods. Their overall objective is to find the non-preferred from a discrete set of alternatives. The best solutions are chosen that provide the desired trade-off information. To find a **candidate approximation** set, they use the concept of rough sets for many features and criteria decision-making. The DRSA (Dominance-based Rough Set Approach) converts this data into lower and upper approximation bounds with prior knowledge from the stated problem.

2 Problem Formulation

2.1 Design Variables and Parameters

The control system **design variables** include the tuning parameters of the PI controller, which are the proportional K_p and the integral K_i gains. The **parameters** in the MOP represent factors that are not under the design engineer's control. These include environmental conditions, linking variables, and physical constants. However, for this design problem, we ignore all the unknown parameters.

2.2 Objectives, Constraints and Preferences

The **objectives** represent the criteria for the **desired direction** of the goals set by the client, but not on the absolute level. Therefore, this MOP's objectives must either be maximized or minimized. The **constraints** represent the defined or **specific level** of the performance criteria. This is usually set on the desired performance criteria that the client is unwilling to compromise on and set on the highest corresponding priority. We don't have an equality constraint, so the following equation describes the inequality. Here, k represents the kth performance criterion. $\boxed{z_k < g_k(x)}$ The **preferences** relate to the desires of the client for acquiring a particular performance level. These include constraints, which are a form of hard preference. Goals indicate preferred performance levels against criteria, and a ranking or ordering system over the criteria known as priorities.

2.3 Combination of all criterion

Combining all the specific subfeatures, we intend to find the specific design x, from a suitable design space D, where the client's performance criterion z is met. Including the inequality constraint g, we can formally express the constrained MOO problem:

$$\text{minimize } x \rightarrow \boxed{z = f(x)}$$

$$\text{subject to} \rightarrow g(x) < 0, x \in D$$

3 Sampling Plan

In the context of a designing space, sampling plans are a technique for creating models that can precisely predict outcomes based on a design variables set. The goal of a sampling plan is to select a part of the design space that can be used to build a surrogate model that can accurately predict the behaviour of the whole design space. During this setup, we have experimented with the following sampling plans with a 10x10 grid space:

3.1 Full Factorial

Here, each axis on a grid represents a factor, and each grid point is a unique permutation of levels for all such factors. This design evaluates each single grid point. This captures all possible interactions between the factors and is essential for understanding how each factor correlates with another. `P = fullfactorial(q, Edges)` Here, q represents a k-vector containing the number of points in each dimension and is set to $[10, 10]$, and $Edges$ represents if the points would be equally spaced from edge-to-edge or in the centres of the bins filling in the unity cube.

3.2 Latin Hypercube

This method focuses on uniformly covering each region. The levels of each factor are randomly distributed across a singular interval; then, these are projected on the actual levels to create the final points of design. `P = lhsdesign(q(1)*q(2), length(q))`. Here, LHS sampling plan takes the first and second arguments as the length of the number of rows and columns of the resulting sampling plan matrix, respectively.

3.3 Sobol set

This method is a subset of LHS, but uses a special sequence called a low-discrepancy sequence to generate points. This ensures greater spread of points within the parametric space, hence performing well in the space-filling aspect.

P = net(sobolset(length(q)), q(1)*q(2)). First, we create a new Sobol sequence object P, with the dimensions matching $[10, 10]$. Next, we generate $q(1) * q(2)$ points from the Sobol sequence.

3.4 Analysis of Space-Filling properties

We now proceed to analyze each of the three sampling plans with the ϕ metric. This metric is based on the concept of discrepancy. This is an indicator of how evenly spaced the design points are across the design space. A **low ϕ value** indicates that the points are more evenly distributed and, hence, can capture the behaviour of the design space more accurately. `phi_metric = mmphi(P, 2, 2)`. Here, the first parameter of the *mmphi* function describes the order of the distance taken as reference, and the second parameter represents the Euclidean distance.

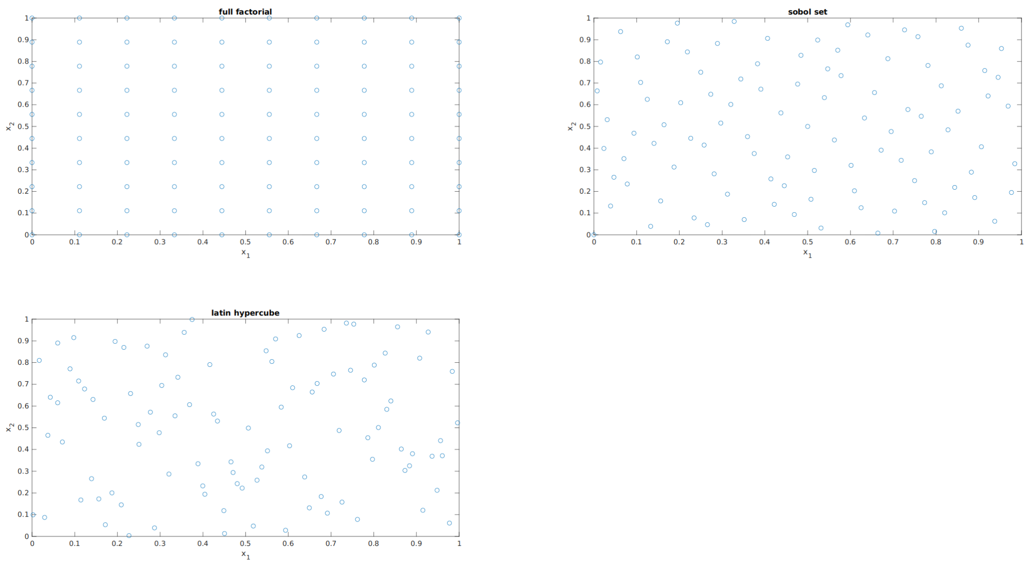

Figure 1: Sampling plans of Full factorial, Sobol set, and Latin Hypercube respectively

As seen in figure 1, the full factorial plan seems to be the most evenly spaced out of the rest of the sampling plans, which is also reflected in the ϕ metric. Another inference is that the LHS design does not effectively fill the space like the others.

$$\boxed{\phi_{full\_factorial} = 207.819922} \qquad \phi_{sobol\_set} = 240.189783 \quad \phi_{latin\_hypercube} = 284.077514$$

4 Knowledge Discovery

4.1 Evaluation Function

After the above analysis, the full factorial sampling plan is chosen. Now, we pass the sampling plan into an evaluation function that analyzes our plan using the client's required metrics. `Z = evaluateControlSystem(P)`. Here, P is the sampling plan, whereas Z is the performance evaluation matrix, consisting of each design as its individual row and each performance criterion as its columns. The performance criterion is as follows:

```
labels = {'max_pole', 'gain_margin', 'phase_margin', 'rise_time',
   'peak_time', 'overshoot', 'undershoot', 'settling_time',
   'steady-state_error', 'control_input'}
```

4.2 Visualization

We proceed to plot each metric of the design evaluation individually on a parallel plot. This is implemented using the inbuilt *parallelplot*() function. As seen from figure 2,

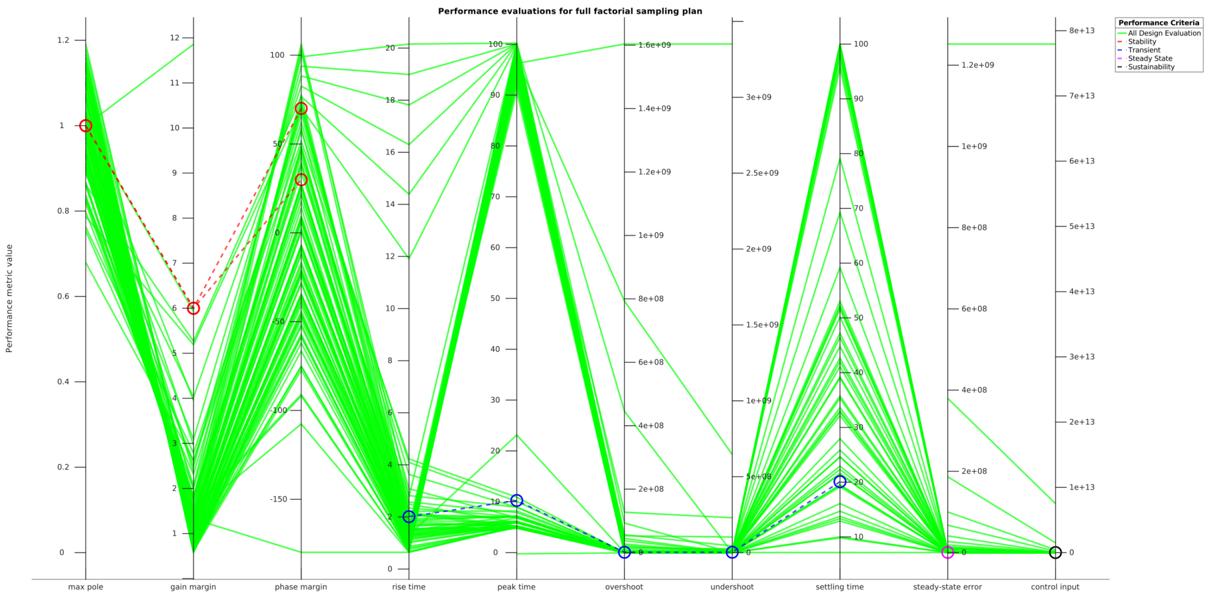

Figure 2: Parallelplot of the performance evaluations matrix with limits for stability, transience, steady-state error, and sustainability criterion.

the **stability region** of the system can be found by the bounds specified in the red dotted line. For the max pole, the dotted line is the upper bound, the gain margin is the lower bound, and a range bounds the phase margin. This seems not ideal, as one would expect a clear boundary to determine stability. However, the demarcation is clear for the **transient region**, which is upper bounded by the blue dotted line.

4.3 Data mining and lower dimensional data relationships

A good way to analyze correlation in multi-variate data is to employ Principal Component Analysis to reduce dimensions. First, we analyze the variance contributed by the 10 classes, then the correlation between the most dominated components. In figure 3, most

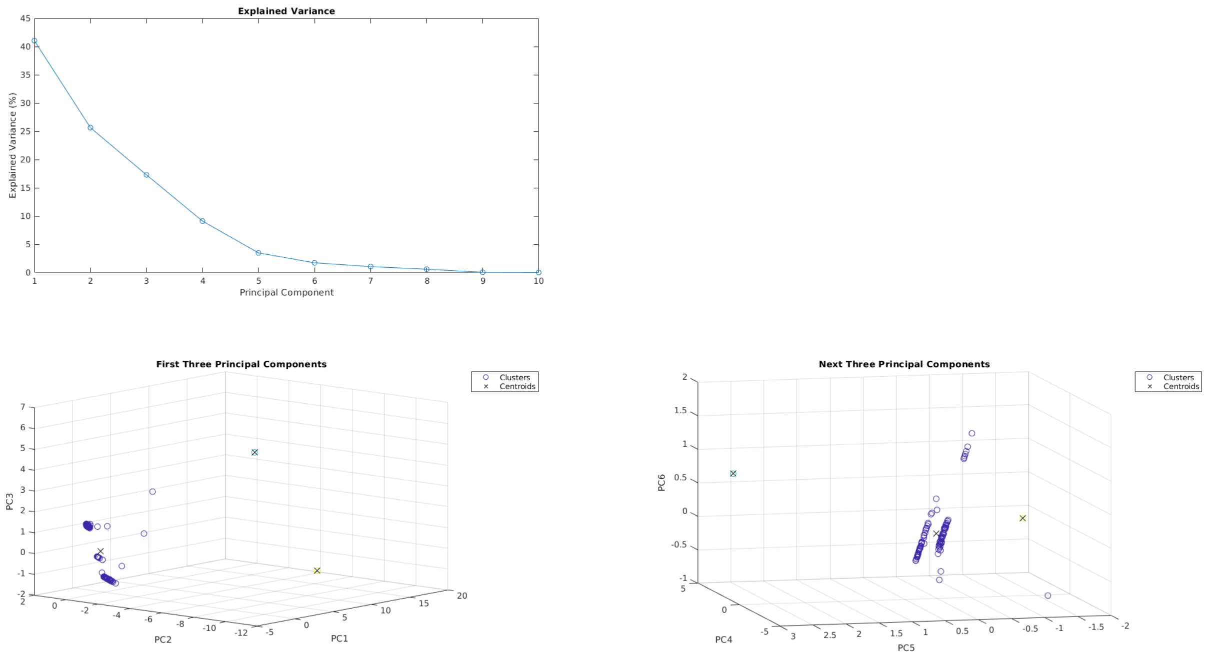

Figure 3: Subplot of the contributed variance and K-means clustering of the first six most dominant principal components.

of the contributed variance comes from the first six principal components. Additionally, further correlation through clustering can be found by grouping triplets together in a scattered plot. There seem to be only a few outliers and out-of-place points after an initial K-means clustering run. Therefore, these changes could be applied if specific performance criterion goals were not required.

5 Optimization Process

5.1 Post processing and creation of an Optimizing Engine

We convert the maximising and the range problem of the gain margin and phase margin to a singular minimizing problem. We convert the performance evaluation into decibels and invert the sign for the gain margin. Hence, the resulting target goal would be -6dB. Hence, `Z(:,2) = -20*log10(Z(:,2))`.

To convert the range of the phase margin, we take the absolute difference between the midpoint of the lower and the upper bounds of the range. Hence, the minimizing limit would be 20. Therefore, `Z(:,3) = abs(Z(:,3) - (30+70)/2)`. Finally, we remove the inf values generated by the evaluation function and set it to a high limit to avoid extreme skewing of the parallel plot. So evaluate as shown: `Z(isinf(Z)) = 1e3`

The engine focuses on morphing the population generated by the chosen sampling plan, into a new population every iteration while incrementally improving its fit against the chosen goals and priorities. Here, priorities indexed at 0, 1, 2, 3 show a low, moderate, high, and hard preference level. Hence, `goals=[1,-6,20,2,10,10,8,20,1,0.67]`, and `priority=[3,2,2,1,0,1,0,0,1,2]`. The following subsections are iterated as chosen by the design engineer.

5.1.1 Population initialization and Fitness Calculation

As described, we pass our population from our initial evaluation function to the post-processing function aliased as mentioned: `Z = optimizeControlSystem(P)`.

1. **Preferred Ranking**: Here, we use the rank_prf() function to return preferred rankings from 1-100 based on the goals and priorities set. Additionally, we flip the rankings, as the required ranking for our use case is the greatest.
   `ranking = rank_prf(Z, goals, priority)`; `ranking = max(ranking)-ranking`

2. **Crowding**: We use the NSGA-II (Non-dominated Sorting Genetic Algorithm) to return crowding distances based on each population ranking. The crowding distance

measures how near an individual point is to its neighbours in the object space. This is also used to maintain varied points in the population, preventing it from collapsing prematurely in a specific region. `distances = crowding(Z,ranking)`

5.1.2 Choosing selection-for-variation and simulating variation

The NSGA-II algorithm uses BTWR(Binary tournament with replacement) to determine if two randomly selected individuals when compared based on their crowding distances, the one with the larger value, is the tournament winner and passed on to the next step with its corresponding index. `selectThese = btwr(distances, length(distances))`

We use the two variation operators from NSGA-II to create a new children population.

1. **Simulated Binary Crossover**: This algorithm takes in two parents and returns two children. The distribution of its offspring is controlled by the SBX parameter nc. The key idea is to promote search space exploration by creating offspring that is very different from the parents. Here, the second parameter represents the problem's bounds. `offspring = sbx(P(selectThese, :), [0, 0; 1, 1], nc)`

2. **Polynomial Mutation**: First, a point is selected randomly for mutation; then, a small perturbation is added from the mutation parameter to introduce diversity in the children C, using the mutation parameter nm.
`C = polymut(offspring, bounds, nm, probability); unifiedPop = [P; C]`

5.1.3 Performing selection-for-survival

In this step, the reducer NSGA-II algorithm does the clustering step to find the best new parent population based on crowding and ranking of the combined population. We intend to use the default behaviour, which divides the provided population into half.

```
new_indices= reducerNSGA_II(unifiedPop,rank_prf(Z_unified,goals,priority),
                crowding(Z_unified, rank_prf(Z_unified, goals, priority)))
P = unifiedPop(new_indices, :)
```

5.1.4 Convergence Monitoring

As each iteration proceeds, we monitor the convergence of the population by comparing it with the hypervolume indicator. This measures the space volume dominated by a set of solutions in the objective space. This volume is bounded by the reference point and is taken to be the maximum of the initial performance evaluation. If the hypervolume stabilizes or slightly increases, this indicates that the solution has converged.

```
Res = [Res, Hypervolume_MEX(Z_unified, reference_point)]
```

6 Optimization Results

6.0.1 Visualization and Convergence Analysis

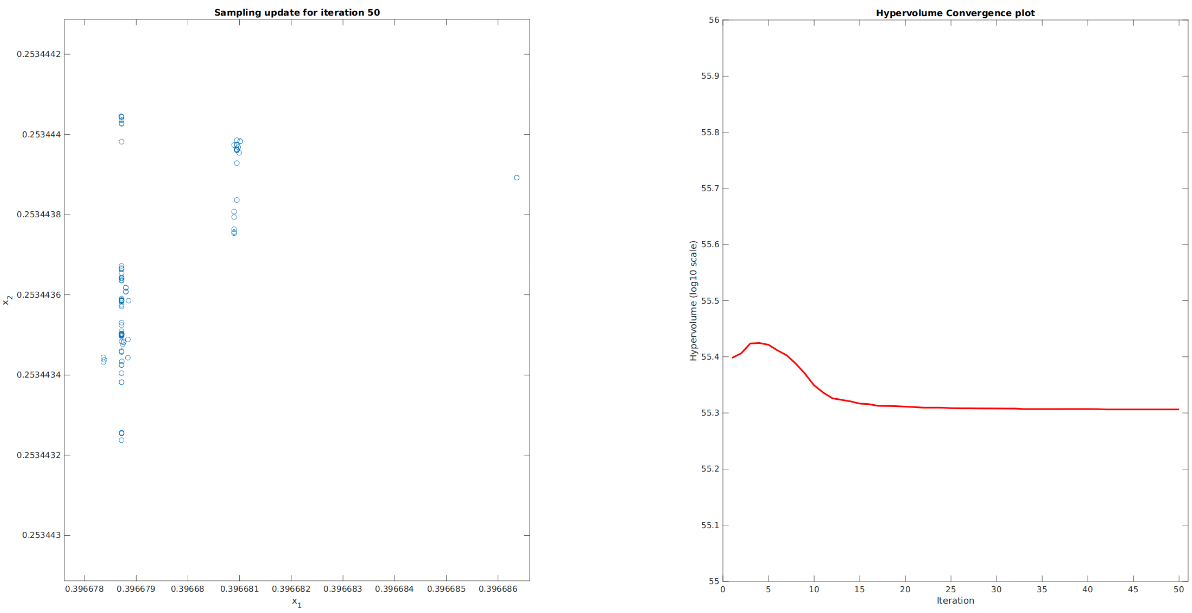

Figure 4: Final position of the population (zoomed), and Hypervolume Convergence

As seen from the figure 4, the hypervolume has stabilized and hence reached convergence. Here, the logarithm of the hypervolume is taken as reference, to save precious computational time. The knowledge discovery plot indicates all of the client's performance criteria has met, except the **rise time** of the system.

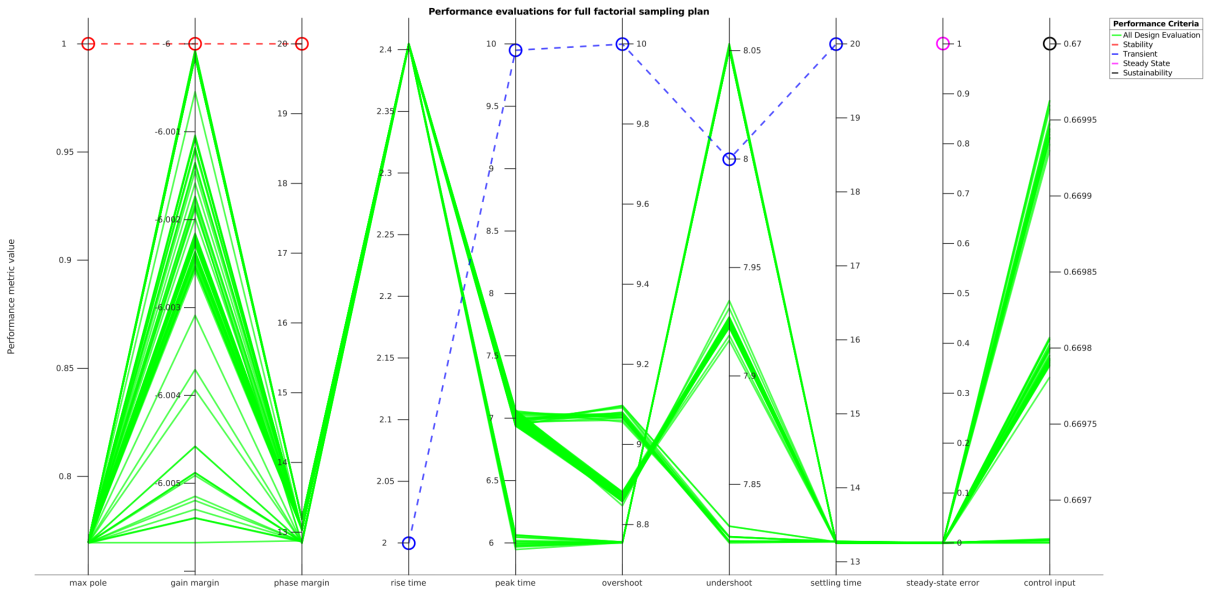

Figure 5: Parallelplot of the performance criterion tuned to match client's requirements.

6.0.2 Finding the best fit

This iteration loop runs 50 times, and the final morphed population is analyzed. The best fit is found by taking the sum of the weighted difference between the performance evaluations and the goal values. The weighted priority vector is chosen to suit the client's priorities. `weighted_priority = [0.8,0.6,0.6,0.4,0.2,0.4,0.2,0.2,0.6,0.6]`

Here, the target indices are found by a binary piecewise intersection method comparing the best amount of common values between the hard, high, moderate, and low indices. After the suitable indices are found, the index of the minimum deviation is stored.

```
deviation = sum((abs(Z(indices(idx),:) - goals)/goals).* weighted_priority)
```

The best performance evaluation Z_{best}, and the corresponding gains are as follows:

$$\boxed{Z_{best} = [0.76, -6.004, 13.08, 2.40, 7.00, 8.80, 7.94, 13.26, 1.58e^{-9}, 0.669]}$$

$$\boxed{[Kp, Ki] = [0.397, 0.252]}$$

6.0.3 Levels of Success and Tradeoffs

The deviation measured from the goals is minimal; overall, the best solution found is a good fit for the client's preferences except for the moderately prioritised rise time. Since the solutions provided by the optimization process are Pareto-optimal, hence no solution exists that can satisfy the rise time constraint without violating any other constraint.

```
success_level% = [23.1, 0.1, 34.6, -20.3, 30, 12, 0.7, 33.7, 99.9, 0.001]
```

```
rise_time_violation = Z_best(4) - goals(4) = 2.40 - 2 = 0.4s
```

6.0.4 Mitigation

The direct implication of the additional rise time is the delayed reaction of the system reaching the required state of the system. This can usually be corrected by adding a **derivative component** of the system. This dampens the system response, reduces the rise time, and improves the transient state of the system.

7 Sustainability Analysis

7.1 Goal Modification and Visualization

We intend to conserve the energy for control efforts by reducing the goal to `0.63MJ`. The entire iteration step is again run for 50 steps. As we can observe in the figure, the **rise time** and **peak time** are clearly violated, while the rest of the criterion is met.

$\boxed{\mathrm{Z}_{best} = [0.7, -13.3, 18.2, 9.8, 28, 0.05, 3.3e^{-5}, 17.7, 1.8e^{-9}, 0.629]}$ $\boxed{[\mathrm{Kp},\ \mathrm{Ki}] = [6.52\mathrm{e}^{-5}, 0.12]}$

7.2 Implications on Transient Performance

The rise and peak time errors are increased up to 390% and 180%, respectively. Even though this criterion has moderate and low priorities, the transient system is now prone to **more inertia** and **delayed response** due to the rise time changes and a decrease in the system's natural frequency due to the peak time errors.

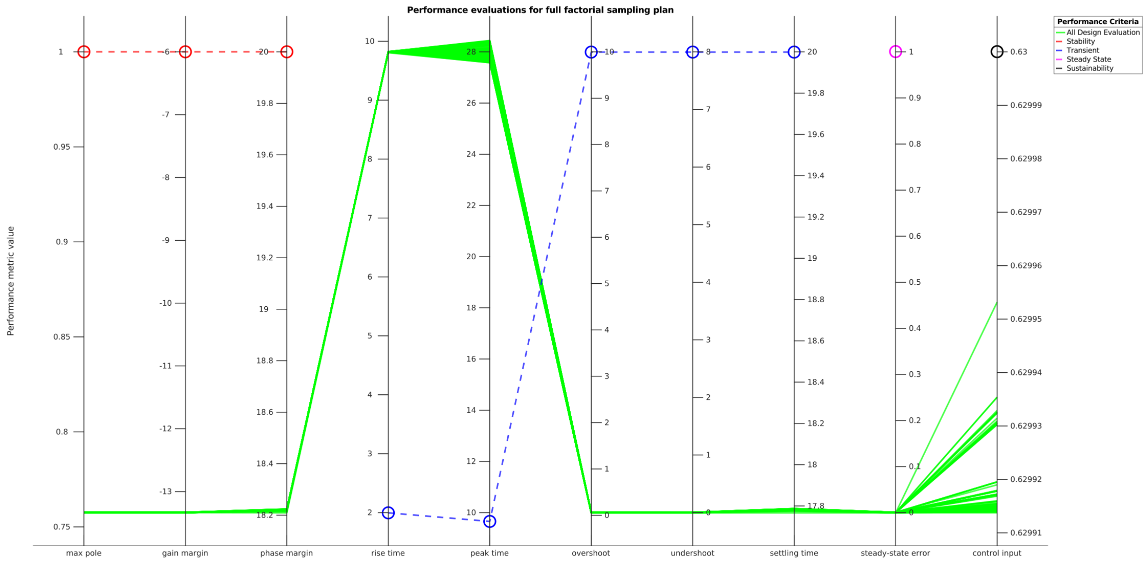

Figure 6: Parallelplot of the performance evaluations with reduced control effort energy.

7.3 Controller Architecture Changes to Achieve Sustainability

Achieving more sustainability is important with energy conservation of about `0.04MJ`; hence we can convert the controller into a PID system. Hardware changes include adding a **differentiator circuit** coupled with an Op-Amp. Additional resistors can be added as input to the opamp, and capacitors can be added to the feedback loop. A potentiometer can be used to tune the derivative gain.

8 Recommendations

Based on the two variations for the control effort and the data observed from the knowledge discovery and optimisation results, we need to reduce the system's rise and peak time to improve transient performance by using **less control effort** and promoting **sustainability**. The following steps can be followed to achieve the same:

1. **Adding a Derivative Component and Tuning**: The use of a PI with a derivative controller (PID) can dampen the system response and can be responsible for predicting future errors based on the value at hand. This helps reduce the settling time and overshoot of the system. Since a digital system represents the problem,

this can be added by taking the difference between current and previous errors and dividing by the time step.

Next, the controller algorithm would need updating and tuning all three components to achieve the desired system response. Since adding such a term in a noisy system would cause stability issues, this is usually accompanied by a low-pass filter with a dedicated loop.

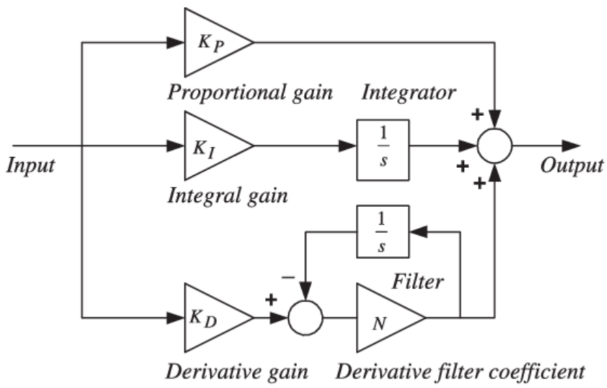

Figure 7: PID Controller Schematic [1]

2. **Feedforward Controller**: If the control system has predictable input or noise levels, a feedforward controller can used to improve the system response if connected in conjunction with the PI controller.

3. **Improving Performance of Actuators**: The rise time of the system is usually affected by the motors, valves or any form of actuators in the system. Using actuators with less moment of inertia and less starting torque can significantly improve the transient state of the system.

9 Conclusion

9.1 Study results and Linking with the vehicle propulsion system

The report explores the tradeoffs between the best solution that the algorithm finds and indicates the possibility of an increased rise time in the final controller. Additionally, the minimization of the control effort does lead to a worsening in the transient performance of the system, thus increasing the limit for peak time as well.

As the client's propulsion system is fitted with a PI controller with tunable gain control, the same problem statement **could be extended**. The interdependency of the vehicle's energy efficiency, wheel friction, braking and many more would make the approach similar to that of a **Multi-Objective Optimization Problem**. The optimization section of the report highlights the incorporation of the client's goals with priorities, and the system also **preserves flexibility** in changes when tuning for the sustainability criterion.

9.2 Applying Problem Methodology

In the subsequent nature of the propulsion problem, for future modifications, we would incorporate them onto the `goals` array as seen in the Appendix. Next, the ranges need to be converted into a purely minimizing problem by adjusting the goals. Finally, the priorities can be added to the performance evaluation by modifying the `priorities` vector. Ultimately, the best solution would indicate the success level% for each variation and the corresponding violation, with insights on improving the system's hardware.

9.3 Additional Decision-making tools for multiple objectives

1. **Model Predictive Control (MPC)**: It uses a prediction model to formulate a MOO to find the best control action that follows a minimized cost function. This process is reiterated at each time step, considering all the system constraints. This makes it flexible, handling multiple input and output systems.

2. **Multi-Attribute Utility Theory (MAUT)**: This method assigns weights to each criterion based on the client's priority and evaluates a score for each alternative. The alternative with the best score is chosen and, hence, can be used for a well-detailed comparison of multiple criteria.

References

[1] E. Şahin. (2019) Design of a pid controller with fractional order derivative filter for automatic voltage regulation in power systems. Accessed: May 19, 2024. [Online]. Available: https://www.researchgate.net/figure/Structure-of-PID-controller-with-derivative-filter$_{f}ig14_{2}$61429619

[2] R. S. F. Ferraz, R. S. F. Ferraz, A. C. R. Medina, and J. F. Fardin, "Multi-objective approach for optimized planning of electric vehicle charging stations and distributed energy resources," *Electrical Engineering*, vol. 105, pp. 4105–4117, 2023. [Online]. Available: https://doi.org/10.1007/s00202-023-01942-z

[3] J. Zhou, C. Feng, Q. Su, S. Jiang, Z. Fan, J. Ruan, S. Sun, and L. Hu, "The multi-objective optimization of powertrain design and energy management strategy for fuel cell–battery electric vehicle," *Sustainability*, vol. 14, no. 10, p. 6320, 2022. [Online]. Available: https://doi.org/10.3390/su14106320

[4] A. Karandikar, A. K. Ravi, E. Dharumaseelan, and E. Tamilselvam, "Model-based design and multi-objective robust optimization of electric vehicle for performance, range and top speed," in *Advances in Multidisciplinary Analysis and Optimization*. Springer, 2020, pp. 173–187. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-15-5432-2$_{1}$5

[5] P. Othaganont, F. Assadian, and D. J. Auger, "Multi-objective optimisation for battery electric vehicle powertrain topologies," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 2016. [Online]. Available: https://doi.org/10.1177/0954407016671275

[6] N. F. Alshammari, M. M. Samy, and S. Barakat, "Comprehensive analysis of multi-objective optimization algorithms for sustainable hybrid electric vehicle charging systems," *Mathematics*, vol. 11, no. 7, p. 1741, 2023. [Online]. Available: https://doi.org/10.3390/math11071741

Appendix

Performance criterion

Stability	Transient	Steady State	Sustainability
Largest closed-loop pole Gain Margin Phase Margin	Rise time Peak time Maximum overshoot Maximum undershoot Settling time	Steady-state error	Control Effort

Table 1: Important characteristics of system responses

Criterion	Direction	Goal	Priority
Largest closed-loop pole	Minimise	< 1	Hard constraint
Gain margin	Maximise	6 dB	High
Phase margin	Range	$\geq 30° \& < 70°$	High
Rise time	Minimise	2 s	Moderate
Peak time	Minimise	10 s	Low
Maximum overshoot	Minimise	10%	Moderate
Maximum undershoot	Minimise	8%	Low
Settling time	Minimise	20 s	Low
Steady-state error	Minimise	1%	Moderate
Control effort	Minimise	0.67 MJ	High

Table 2: Raw performance criterion for initial case

Criterion	Direction	Goal	Priority
Largest closed-loop pole	Minimise	< 1	3
Gain margin	Minimise	< -6	2
Phase margin	Minimise	< 20	2
Rise time	Minimise	< 2	1
Peak time	Minimise	< 10	0
Maximum overshoot	Minimise	< 10	1
Maximum undershoot	Minimise	< 8	0
Settling time	Minimise	< 20	0
Steady-state error	Minimise	< 1	1
Control effort	Minimise	< 0.67 or < 0.63	2

Table 3: Modified performance criterion for ease of optimization

MATLAB Script

```
% Copyright (c) 2024 Leander Stephen D'Souza

% Program to analyze different sampling plans and build an optimizing engine

clc; clear; close all;

% Build all the mex files
mex(fullfile(pwd, '/EA_Toolbox/rank_nds.c'));
mex(fullfile(pwd, '/EA_Toolbox/crowdingNSGA_II.c'));
mex(fullfile(pwd, '/EA_Toolbox/btwr.c'));
mex(fullfile(pwd, '/EA_Toolbox/sbx.c'));
mex(fullfile(pwd, '/EA_Toolbox/polymut.c'));
mex('-DVARIANT=4', ...
    fullfile(pwd, '/Hypervolume/Hypervolume_MEX.c'), ...
    fullfile(pwd, '/Hypervolume/hv.c'), ...
    fullfile(pwd, '/Hypervolume/avl.c'));
mex(fullfile(pwd, '/EA_Toolbox/rank_prf.c'));

% Add sampling and evaluation functions to the path
addpath(fullfile(pwd, '/sampling/'));
addpath(fullfile(pwd, '/evaluation/'));
addpath(fullfile(pwd, '/EA_Toolbox/'));

% Labels for the design constraints
design_constraints = {'max pole', 'gain margin', 'phase margin', 'rise time', 'peak time
    ', 'overshoot', ...
    'undershoot', 'settling time', 'steady-state error', 'control input'};

% Sampling plans to analyze
sampling_plans_list = {'full factorial', 'sobol set', 'latin hypercube', 'random Latin
    hypercube'};

% Analyze different sampling plans
[P, best_sampling_plan] = mmphi_analysis(sampling_plans_list);
fprintf('The best sampling plan is: %s\n', best_sampling_plan);

% % Find the lower dimensional representation of the data
data_mining(evaluateControlSystem(P));

% Building the optimizing engine
% Initial Client Assignment
iterations = 50;
```

```
priority = [3, 2, 2, 1, 0, 1, 0, 0, 1, 2];
weighted_priority = [0.8, 0.6, 0.6, 0.4, 0.2, 0.4, 0.2, 0.2, 0.6, 0.6];
goals = [1, -6, 20, 2, 10, 10, 8, 20, 1, 0.67];

buildOptimizingEngine(true, P, iterations, goals, priority, weighted_priority,
    best_sampling_plan, design_constraints);

% Reduce control input
goals(10) = 0.63;
buildOptimizingEngine(true, P, iterations, goals, priority, weighted_priority,
    best_sampling_plan, design_constraints);



% Function to analyze the best fit from the matrix of design evaluations, priority, and
    goals
function analyze_best_fit(P, Z, priority, weighted_priority, goals, design_constraints)
    % declare the indices
    high_priority_indices = [];
    moderate_priority_indices = [];
    low_priority_indices = [];

    % get the hard priority index
    hard_priority_index = find(priority == max(priority));

    if max(priority) - 1 >= 0
        % get the high priority index
        high_priority_indices = find(priority == max(priority) - 1);
    end

    if max(priority) - 2 >= 0
        % get the moderate priority index
        moderate_priority_indices = find(priority == max(priority) - 2);
    end

    if max(priority) - 3 >= 0
        % get the low priority index
        low_priority_indices = find(priority == max(priority) - 3);
    end

    % Get the indices that satisfy the hard priority
    hard_indices = find(Z(:, hard_priority_index) < goals(hard_priority_index));
    high_indices = [];
    moderate_indices = [];
```

```
    low_indices = [];

    % get the high priority index
    for i = 1:length(high_priority_indices)
        high_indices = [high_indices; find(Z(:, high_priority_indices(i)) < goals(high_priority_indices(i)))];
    end

    % get the moderate priority index
    for i = 1:length(moderate_priority_indices)
        moderate_indices = [moderate_indices; find(Z(:, moderate_priority_indices(i)) < goals(moderate_priority_indices(i)))];
    end

    % get the low priority index
    for i = 1:length(low_priority_indices)
        low_indices = [low_indices; find(Z(:, low_priority_indices(i)) < goals(low_priority_indices(i)))];
    end

    % get the intersection of the hard, high, moderate, and low indices
    hard_high_indices = intersect(hard_indices, high_indices);
    hard_high_moderate_indices = intersect(hard_high_indices, moderate_indices);
    hard_high_moderate_low_indices = intersect(hard_high_moderate_indices, low_indices);

    if isempty(hard_indices)
        fprintf('No indices satisfy the hard constraints, Bad design\n');
        indices = [];
    elseif isempty(hard_high_indices)
        fprintf('No indices satisfy the hard and high constraints\n');
        fprintf('The number of indices that satisfy the hard constraints is: %d\n', length(hard_indices));
        indices = hard_indices;
    elseif isempty(hard_high_moderate_indices)
        fprintf('No indices satisfy the hard, high, and moderate constraints\n');
        fprintf('The number of indices that satisfy the hard and high constraints is: %d\n', length(hard_high_indices));
        indices = hard_high_indices;
    elseif isempty(hard_high_moderate_low_indices)
        fprintf('No indices satisfy the hard, high, moderate, and low constraints\n');
        fprintf('The number of indices that satisfy the hard, high, and moderate constraints is: %d\n', length(hard_high_moderate_indices));
        indices = hard_high_moderate_indices;
    else
```

```
        fprintf('The number of indices that satisfy the hard, high, moderate, and low constraints is: %d\n', length(hard_high_moderate_low_indices));
        indices = hard_high_moderate_low_indices;
    end

    % check the difference between the goals and the Z values to find the best fit
    for idx = 1:length(indices)
        diff = abs(Z(indices(idx), :) - goals) / goals;
        diff = diff .* weighted_priority;
        % remove the NaN values
        diff(isnan(diff)) = 0;
        best_fit(idx) = sum(diff);
    end
    % get the index of the minimum best fit
    min_best_fit_index = find(best_fit == min(best_fit));

    % print the row of Z that satisfies the minimum best fit
    best_solution = Z(indices(min_best_fit_index), :);

    % print the violation of the constraints
    for i = 1:length(best_solution)
        if best_solution(i) > goals(i)
            % print the label of the feature
            fprintf('The %s constraint is violated\n', design_constraints{i});
        end
    end

    % print success rate deviation from goal
    success_rate = (abs(goals) - abs(best_solution)) ./ goals * 100;
    % round the success rate to 1 decimal place
    success_rate = round(success_rate, 1);

    % print the success rate
    fprintf('The success rate deviation from the goal is: %s\n', mat2str(success_rate));

    % print the best solution
    fprintf('The best solution is: %s\n', mat2str(best_solution));

    % get the values of Kp and Ki that satisfy the minimum best fit
    fprintf('The values of Kp and Ki that satisfy the minimum best fit are: %s\n', mat2str(P(indices(min_best_fit_index), :)));
end


```

```
% Function to build the optimizing engine with preferability
function buildOptimizingEngine(enable_preference, P, iterations, goals, priority,
    weighted_priority, best_sampling_plan, design_constraints)
    reference_point = max(optimizeControlSystem(P));
    convergence = zeros(1, iterations, 'double');
    bounds = [0, 0; 1, 1];
    figure;
    set(gcf, 'Position', get(0, 'Screensize'));

    for i = 1:iterations
        % Step 1: Initializing the population
        Z = optimizeControlSystem(P);

        % Step 2: Calculating fitness
        % Step 2.1: Non-dominated sorting with preferability (flipped ranking)
        if enable_preference
            ranking = rank_prf(Z, goals, priority);
        else
            ranking = rank_nds(Z);
        end
        % inverse the ranking
        ranking = max(ranking) - ranking;

        % Step 2.2: Crowding distance assignment
        % NSGA-II density estimator
        distances = crowding(Z, ranking);

        % Step 3: Performing selection-for-selection
        % Binary tournament selection with replacement.
        % Returns the indices of the selected individuals.
        selectThese = btwr(distances, length(distances));

        % Step 4: Performing variation

        % Step 4.1: Simulated binary crossover
        % Simulated Binary Crossover operator
        % for real number representations.

        % Z -> objectives
        % P - > decision variables

        parents  = P(selectThese, :);
        offspring = sbx(parents, bounds);

```

```
        % Step 4.2: Polynomial mutation
        % Polynomial mutation operator
        % for real number representations.
        %
        C = polymut(offspring, bounds);

        % Step 5: Performing selection-for-survival

        % Step 5.1: Combine the parent and offspring populations
        unifiedPop = [P; C];

        % Step 5.2: Reducing the population
        % NSGA II clustering procedure.
        % Selects the new parent population from the unified population
        % of previous parents and offspring.

        Z_unified = optimizeControlSystem(unifiedPop);
        if enable_preference
            new_indices = reducerNSGA_II(unifiedPop, rank_prf(Z_unified, goals, priority
    ), crowding(Z_unified, rank_prf(Z_unified, goals, priority)));
        else
            new_indices = reducerNSGA_II(unifiedPop, rank_nds(Z_unified), crowding(
    Z_unified, rank_nds(Z_unified)));
        end

        % Step 5.3: Select the new population
        P = unifiedPop(new_indices, :);

        % Step 6: Check for convergence
        convergence(i) = log10(Hypervolume_MEX(Z, reference_point));
        % Hypervolume is a measure of the volume of the objective space, dominated by
    the Pareto front.

        % Step 7: Plot the new population and convergence

        % Plot the progress using drawnow
        subplot(1, 2, 1);
        plot(P(:,1), P(:,2), 'o');
        title(sprintf('Sampling update for iteration %d', i));
        xlabel('x_1');
        ylabel('x_2');
        drawnow;
    end

```

```
    % Plot the convergence using subplot and lines
    subplot(1, 2, 2);
    plot(convergence, 'LineWidth', 2, 'Color', 'r');
    title('Hypervolume Convergence plot');
    xlabel('Iteration');
    ylabel('Hypervolume (log10 scale)');
    xlim([0, iterations + 1]);
    ylim([55, 56]);

    % Get the design evaluations
    Z = optimizeControlSystem(P);

    % Implement knowledge discovery
    knowledge_discovery(Z, best_sampling_plan, design_constraints, goals);

    % Analyze best fit from the matrix of design evaluations
    analyze_best_fit(P, Z, priority, weighted_priority, goals, design_constraints);
end

% Function to optimize the sampling plan
function Z_optimized = optimizeControlSystem(P)
    Z = evaluateControlSystem(P);

    % Step 1: Convert gain margin to decibels
    Z(:,2) = 20*log10(Z(:,2));

    % Step 2: Minimize the gain margin
    Z(:,2) = -Z(:,2);

    % Step 3: Minimize the absolute deviation of the gain margin within 30 and 70 dB
    Z(:,3) = abs(Z(:,3) - 50);

    % Step 4: Remove all inf values in Z, set to a high value
    Z(isinf(Z)) = 1e3;

    % Return the optimized sampling plan
    Z_optimized = Z;
end

% Function to analyze different sampling plans using mmphi
function [P_best, best_sampling_plan] = mmphi_analysis(sampling_plans_list)
    scale = 1;
    q = [10, 10];
    Edges = 1;
```

```
    min_phi_metric = Inf;
    n_rows = ceil(length(sampling_plans_list) / 2);

    figure;
    set(gcf, 'Position', get(0, 'Screensize'));

    for i = 1:length(sampling_plans_list)
        sampling_plan = sampling_plans_list{i};
        if strcmp(sampling_plan, 'full factorial')
            P = fullfactorial(q, Edges);
        elseif strcmp(sampling_plan, 'sobol set')
            P = sobolset(length(q));
            P = net(P, q(1)*q(2));
        elseif strcmp(sampling_plan, 'latin hypercube')
            P = lhsdesign(q(1)*q(2), length(q));
        elseif strcmp(sampling_plan, 'random Latin hypercube')
            P = rlh(q(1)*q(2), length(q), Edges);
        else
            error('Invalid sampling plan specified.');
        end

        phi_metric = mmphi(P * scale, 5, 1);
        fprintf('The MMPhi metric for %s sampling plan is: %f\n', sampling_plan, phi_metric);

        if phi_metric < min_phi_metric
            min_phi_metric = phi_metric;
            best_sampling_plan = sampling_plan;
            P_best = P;
        end

        % Plot the sampling plan using subplot
        subplot(n_rows, 2, i);
        plot(P(:,1), P(:,2), 'o');
        title(sprintf('%s', sampling_plan));
        xlabel('x_1');
        ylabel('x_2');
    end
    pause(1);
end

% Function to implement knowledge discovery
function knowledge_discovery(Z, best_sampling_plan, design_constraints, goals)
    figure;
```

```
    set(gcf, 'Position', get(0, 'Screensize'));

    stability = nan * ones(size(Z, 2), 1);
    stability(1:3) = goals(1:3);

    transient = nan * ones(size(Z, 2), 1);
    transient(4:8) = goals(4:8);

    steady_state = nan * ones(size(Z, 2), 1);
    steady_state(9) = goals(9);

    sustainability = nan * ones(size(Z, 2), 1);
    sustainability(10) = goals(10);

    % Append extra line to A
    Z = [Z; stability'; transient'; steady_state'; sustainability'];

    % create a vector of strings
    groupDataVector = cell(1, size(Z, 1) - 4);
    for i = 1:length(groupDataVector)
        groupDataVector{i} = 'All Design Evaluation';
    end

    % Grouping vector
    groupDataVector = [groupDataVector, 'Stability', 'Transient', 'Steady State', 'Sustainability'];

    p = parallelplot(Z, 'groupData', groupDataVector, 'Color', {'green','red', 'blue', 'magenta', 'black'}, 'LineWidth', 2);
    p.MarkerStyle = {'none','o', 'o', 'o', 'o'};
    p.MarkerSize(end) = 15;
    p.LineStyle = {'-', '--', '--', '--', '--'};
    p.LegendTitle = 'Performance Criteria';


    p.CoordinateTickLabels = design_constraints;
    p.YLabel = 'Performance metric value';
    p.Title = sprintf('Performance evaluations for %s sampling plan', best_sampling_plan);
end

% Function to find the lower dimensional representation of the data
function data_mining(Z)
    figure;
```

```
set(gcf, 'Position', get(0, 'Screensize'));

% remove inf values
Z(isinf(Z)) = 1e6;

% normalize the data
Z = normalize(Z);

% apply pca
[coeff, score, latent, ~, explained] = pca(Z);

% plot the explained variance
subplot(2, 2, 1);
plot(explained, 'o-');
title('Explained Variance');
xlabel('Principal Component');
ylabel('Explained Variance (%)');

% plot the first three principal components

% perform kmeans clustering on the first three principal components
[idx, C] = kmeans(score(:,1:3), 3);

% plot the first three principal components
subplot(2, 2, 3);
scatter3(score(:,1), score(:,2), score(:,3), 50, idx, 'o');
% change the color of the centroids
hold on;
scatter3(C(:,1), C(:,2), C(:,3), 100, 'k', 'x');
hold off;
title('First Three Principal Components');
xlabel('PC1');
ylabel('PC2');
zlabel('PC3');
% label the legend
legend('Clusters', 'Centroids');

[idx, C] = kmeans(score(:,4:6), 3);

% plot the next three principal components
subplot(2, 2, 4);
scatter3(score(:,4), score(:,5), score(:,6), 50, idx, 'o');
% change the color of the centroids
hold on;
```

```
    scatter3(C(:,1), C(:,2), C(:,3), 100, 'k', 'x');
    hold off;
    title('Next Three Principal Components');
    xlabel('PC4');
    ylabel('PC5');
    zlabel('PC6');
    % label the legend
    legend('Clusters', 'Centroids');
    pause(0.5);
end
```