ACS6124-002 Multisensor and Decision Systems
Part II Decision System for Engineering Design

Saba Firdaus Ansaria
Reg No: 210110201

May 2022

Executive Summary

Designing the propulsion system of a nuclear-powered powertrain is a complex engineering endeavour. A controller will manage the propulsion system, and building the controller system without testing all the use-cases and boundary conditions is an expensive risk. Therefore the system conditions need to be checked beforehand via a Proportional-Integral (PI) controller. In this report, the pilot study for tuning the gains of the PI system is described with a set of optimization tools. As several codependent sensitive objectives need to be tuned, the problem is formulated as a multi-objective optimization problem and solved with a fast elitist genetic algorithm. Furthermore, after exploring the design variables and the objectives, it is found that there is a robust negative correlation among some of the objectives, which can be daunting for simple optimization tools to handle. To reduce the chance of halting into suboptimal solution space, NSGA-II is used to cover more solution search space. The testing criterion for the efficiency of the study is the chief engineer's preferences and target values of PI gains. This study optimizes the PI gains and analyzes the relationships among the testing objectives and how they can influence the outcome. It is found that the preference priorities need to be changed based on the strong negative correlations among the objective, which was previously unknown. The pilot study suggests some critical recommendations and revisions related to the PI controller design decision and objective decision boundaries. Furthermore, we demonstrate the genetic multi-objective optimization and data mining analysis tools and frameworks and the superior abilities to optimize vehicle design components for efficient and robust performance. The outcomes of this pilot study are presented using efficient data visualization tools, diagrams and tables to interpret the breadth and depth of the study.

1 Multi-objective optimization for Engineering Design

1.1 Decision systems for engineering design

Designing a complex engineering system is a multidisciplinary and multifidelity task. The whole system design is divided into ontological sub designs and assigned to different teams from very different backgrounds. Not only do the teams have to work coherently, but they must meet specific customer and regulatory requirements. The teams need to balance tradeoffs between the objectives and take hard decisions for optimal outcome [1]. This decision-making depends on the variables that the teams can control, also known as design variables. To bridge the interdisciplinary gap, these design decisions are treated like objectives and the higher level main design problem is represented with a mathematical cost function. This strategy is advantageous for searching for tuned design variables in the experimental space, and it bridges multidisciplinary hierarchies.

However, the design problem can be resolved using various optimisation techniques. It can be bayesian, dynamic programming related or multiobjective optimisation. Each of these approaches has its advantages and disadvantages [1, 2, 3]. However, in this report, multiobjective optimisation is favoured as these optimisation techniques can mitigate the tradeoffs among objectives and goals while managing the priority goals and thrive on continuous change in objectives and increasing complexity. Also, multiobjective optimisation techniques are highly scalable. The point of interest in this report is a genetic algorithm based multiobjective optimisation NSGA-II [4]. It uses the resources effectively with less computational complexity in identifying promising candidate designs while understanding the relationships between design facets and performance requirements.

1.2 Approaches for decision systems

As mentioned in Section 1.1, there are various optimization approaches for decision making for complex engineering designs. Some of the strategies used in vehicle design are discussed in this section. [2] use topological optimization (ANSYS) to optimize the distribution of 17 design variables in a battery thermal management system (spacing size, air manifold size, mass flow rate etc.), by fulfilling given constraints and minimizing a predefined cost function. They followed a multi-fidelity model optimization and three thermal management systems and concluded that variable fidelity models achieve balance with topological optimization more accurately and efficiently. [5] use dynamic programming to optimize power distribution between the vehicle drive train and the battery heating system for electric vehicles. Their work showed an effective reduction in the overall energy loss in the battery. However, the work formulated a cost function assuming individual

power drain sources as weighted coefficients. The problem with this approach is that this approach would not be optimal with higher uncertainty, and a significant increase in the parameter space can seriously damage dynamic programming performance. [1] propose a similar dynamic programming approach for minimizing fuel consumption and tailpipe emissions in vehicles, and they suggested that although the dynamic programming (DP) based approach optimizes air to fuel ratio, they are complex to implement, and reinforcement algorithm-based solutions may be better options. Multiobjective genetic algorithm (MOGA) [6] based optimization algorithms became very popular for vehicle designs. Unlike the DP based methods, the MOGAs are easy to scale, and the solutions can be interpreted with higher perceptibility. [6, 7, 8] use MOGA based NSGA-II [4] for the energy management system for the vehicle powertrain and the vehicle suspension system. The use of NSGA-II is because it is an elitist algorithm that explicitly preserves diversity (crowding mechanism) and chooses a non-dominated pareto-optimal solution. As a result, it is less likely to be stuck in a local sub-optimal solution. All these optimization processes can be summarised into two categories, i.e., single-solution and population-based algorithms.

1.3 Population based optimizers

Some of the popular population-based optimizers that can be used in multiobjective optimization are particle swarm optimization (PSO), ants colony optimization (ACO) and genetic algorithm (GA). All these three optimizers (PSO, ACO, GA) uses randomly or pseudo-randomly generated population space to start the optimization process. Although sometimes the initialization can affect the final solution, more or less, all these three algorithms start from a random search space [9, 10].

Every particle in PSO computes its position and the best fitness value and the global best position, which has the best fitness value depending on the objectives. Slowly the particles start to move towards the optimal fitness space at that iteration, and the velocity is fine-tuned during the process. ACO particles mimic ant behaviours as agents who find the shortest routes to the optimal space, and other agents follow that. These agent particles communicate among themselves in terms of routes and minimum paths on the sample space as graphs. On the contrary, genetic algorithms try to calculate the fitness values of their current population and select the non-dominated solution fronts based on the objective functions. Those regions are selected, and more population is generated using crossover and mutation to cover the search space. The goal of GAs is basically to preserve the fittest population and generate more along in that area. On the contrary, PSO does not apply selection, and the whole population continues in the optimization duration. It is the only evolutionary algorithm that does not apply survival of the fittest.

In GAs, the elitist solutions are used to produce new offspring using crossover and

mutation. However, ACO and PSO do not involve directly in exchanging materials with other particles. However, these particles communicate with each other (topological neighbours) to influence their trajectory, and velocity [11]. ACO agents leave a *pheromone* path for the other agents.

The computational efficacy between PSO and GA is much debated. However, PSOs generally take less computational resources than GAs to reach the same solution space [12]. However, the solution quality was similar to the research [12]. Between ACO and GA algorithms, some work shows that ACO takes less computational resource [13] and some shows that ACO takes more resource [14]. It will be pretty safe to assume that the computational resource can be dependent on the implementation approach, problem space and objectives. In all the studies, it has been prompted that the solution quality in genetic algorithms is similar, if not superior, compared to PSO and ACO. Assessing all the pros and cons among the best population-based evolutionary optimization algorithms, it is quite clear that genetic algorithms (NGSA-II in this study) are an efficient choice, if not the best choice for our problem.

Finally, genetic algorithms can be modified and adapted to constraint satisfaction problems, where multiple objectives have different priorities. This pilot study is based on such a problem where the objective criterion is priority based. On the contrary, PSO and ACO are less accurate with multiobjective with constraint satisfaction problems. Hence, the genetic algorithm is the perfect tool for our optimization problem in the current pilot study for PI gain optimization.

2 Problem Formulation

The main goal is to optimise the Proportional-Integral (PI) controller so that the feedback control system satisfies the chief engineer's requirements. The function *evaluateControlSystem* takes the pair of control gains K_p and K_I and returns the performance. However, there is no Simulink model available at present. Therefore a set of design variables need to be generated and optimised to the possible optimal values satisfying the chief engineer's preferences. Let us take $S \leftarrow \{(K_{P1}, K_{I1}), \ldots, (K_{Pn}, K_{In})$ are the design sample points and the parameters in the optimizer are p. The problem has nine objectives, and there are denoted by the following functions such as maximum closed-loop pole magnitude (f1), gain margin (f2), phase margin (f3), 10-90% rise time (f4), peak time (f5), overshoot (f6), undershoot (f7), 2% settling time (f7), steady-state error (f8). Hence, according to the chief engineer's criterion, the problem can formally be formulated as a constrained multi-objective optimisation problem such as

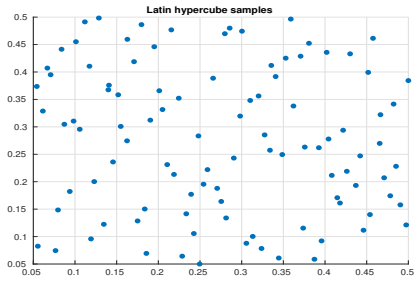
$$
\begin{aligned}
\underset{s}{\text{minimize}} \quad & GA(s,p) \\
\text{subject to} \quad & f1(s,p) < z1 \qquad f2(s,p) = z2 \quad f3(s,p) = z3 \\
& f4(s,p) = z4 \qquad f5(s,p) = z5 \quad f6(s,p) = z6 \\
& f7(s,p) = z7 \qquad f8(s,p) = z8 \quad f9(s,p) = z9 \\
& s \leftarrow S \qquad\qquad\qquad p \leftarrow P
\end{aligned} \tag{1}
$$

Where $z1 = 1, z2 = \frac{1}{1+6}, z3 = 45, z4 = 2, z5 = 10, z6 = 10, z7 = 8, z8 = 20, z9 = 1$ and $z \leftarrow Z$. The goals are set to the given criterion. Objective f2 and f3 have been modified to be able to use them as a minimisation term. Finally, a set of prefarability and priority needs to be considered for the optimization process. If the prefarability operator is $Pr*$ then the optimisation can be written as

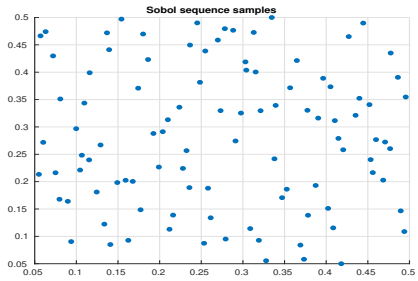
$$
\begin{aligned}
\underset{s}{\text{minimize}} \quad & Pr^*(GA(s,p), F) \\
\text{subject to} \quad & F \leftarrow \{f1, f2, ..., f9\} \qquad s \leftarrow S \quad p \leftarrow P
\end{aligned} \tag{2}
$$

The chief engineer has put the priorities in the following order f1 is a hard constant, f2 & f3 have high priority, f4 & f6 & f9 has moderate priority, and the rest of the objectives are low priority.

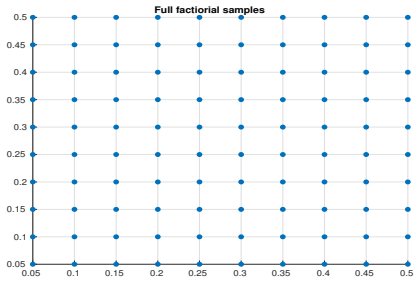
3 Sampling Plan



(a) Latin hypercube samples



(b) Sobol sequence samples



(c) Full factorial sampling samples

Figure 1: Pictorial scatter representation of different sampling techniques

To carry out the optimization formulation from Section 2, an efficient sampling plan is necessary. These samples would be the starting points for the optimization process. An ideal sampling plan should follow some characteristics, such as it should be properly distributed on a given hypercuboid region, and the samples should cover the whole sample space range. Here, three sampling plans have been explored, and their merits are tested.

Latin hypercube sampling

Latin hypercube sampling partitions (LHS) samples in the non-overlapping sampling regions while dividing the range of each component on the basis of equal probability interval. These mutually exclusive regions independently generate the design variable, and these variables from different regions interact and form hypercubes.

Sobol sequences

Sobol sequences are quasi-random numbers generated with lower discrepancy (better space-filling property). These samples generally produce faster convergence, and stable optimization estimators [15].

Full factorial sampling

The full factorial sampling strategy divides the sample space into a set of non-overlapping hypercubes and defines samples at the corner or the centre of the hypercube. As a result, this approach relies less on the assumption about relative sparsity among the important input design variables.

3.1 Space filling properties

Space-filling properties express if a sampling technique has sample points everywhere in the design variable space and covers the experiment region with few gaps as possible. In this report, the Latin hypercubes based [16] criterion is used to indicate the quality of the space-filling by the sampling mentioned above. The metric is denoted as ϕ and defined as

$$\phi(X) = (\sum_{k=1}^{n} \frac{s_j}{d_k^q})^{\frac{1}{q}} \tag{3}$$

Where X is the sampling plan, $d \leftarrow \{d_1, d_2, ..., d_n\}$ are unique distances between pairs $s \leftarrow \{s_1, s_2, ..s_n\}$, and q is tunable hyperparameter. Here $q = 2$ and the measurements for the corresponding sampling plans are shown in Table 1. Here, for each sampling plan, 100 samples have been derived, and they are used to compute ϕ. The space-filling representations are shown in Figure 1. To calculate ϕ without range bias, all the sample populations have been rescaled into the same range.

According to Figure 1, the sampling population spread for LHS and Sobel are similar, and they have primarily uniform spread among the hypercuboids. However, it is visible

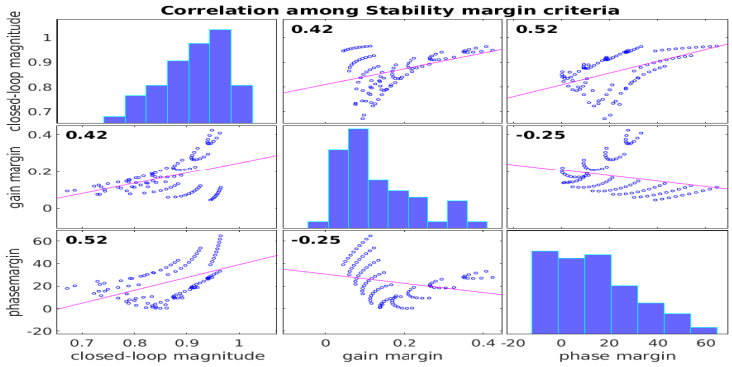
Sampling Plan	Space Filling Metric (ϕ)
Sobol sequence	507.0749
Full factorial	403.7873
Latin Hypercube	490.0260

Table 1: Comparison of different sampling techniques with space filling metric (ϕ)

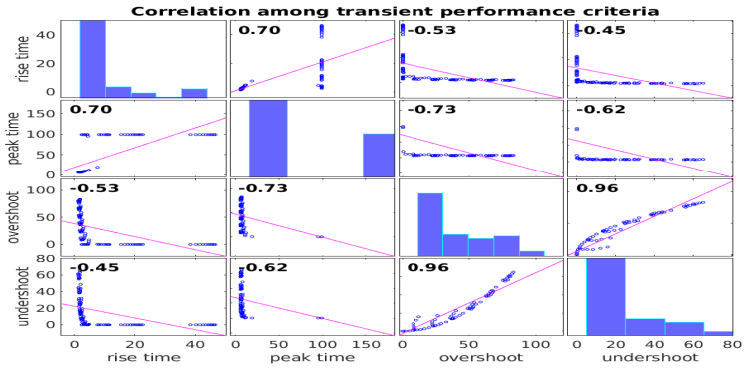
that they did not cover the whole design space. However, full factorial samples cover all the hypercuboids evenly. The same conclusion can be drawn from Table 1 with a population size of 100 because the full factorial technique got the lowest phi value (403.78) by a significant margin.

The same experiment has been repeated multiple times and with multiple populations ranging from 100 to 300, and similar conclusions have been drawn. Hence, for the following stages of the report, full factorial sampling will work as the primary sampling plan.

4 Knowledge Discovery



(a) Correlation among stability margin criteria.



(b) Correlation among transient performance criteria.

Figure 2: Correlation among the objectives with distribution.

According to Section 2, the PI control system function is modified to *optimizeControlSystem* for generating the design objectives with design variable population to accommo-

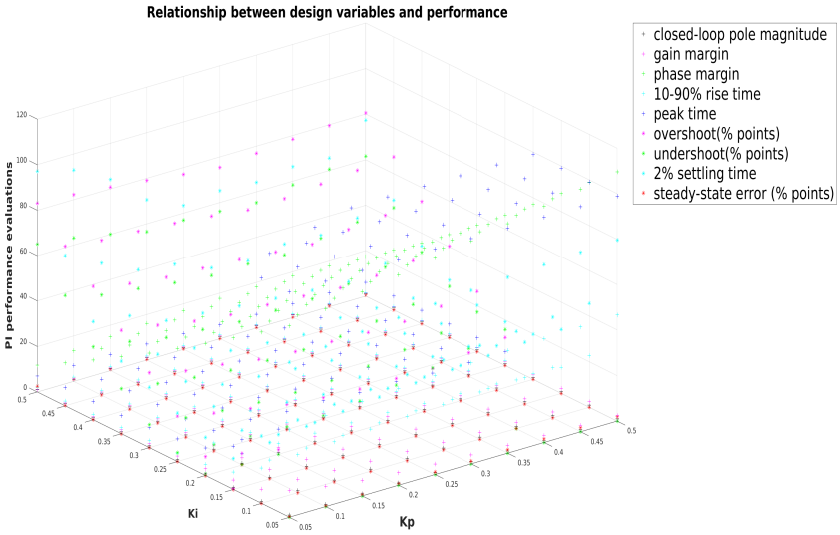


Figure 3: Scatter plot representation among design variables and PI outcomes.

date all the objectives with optimizer minimisation goal . In this section, the generated design variables using *full factorial sampling* are passed through *evaluateControlSystem*, and the relationships among the design variables and objectives are analysed and visualised. Three approaches are used to perform these tasks.

1. **Correlation among Objectives:** The correlation among objectives have been observed with MATLAB function *corrplot* for analysing the tradeoffs and dependencies among the objectives itselves within mainly two groups ,i.e, stability criteria and transient performance criteria. The plots are shown in Figure 2.

2. **Correlation between design variables and objectives:** The correlation is computed with MATLAb function *corrcoef* and the results are shown in Table 3.

3. **Datapoint Scatter plot:** The spread between the KP, KI and individual objectives are shown in Figure 3.

Var1	Var2	Correlation	p value
KP	closed-loop magnitude	0.0932	0.3562
KP	gain margin	0.1605	0.1105
KP	phase margin	0.1668	0.0970
KP	rise time	0.0267	0.7918
KP	peak time	0.0911	0.3673
KP	overshoot	-0.0430	0.6709
KP	undershoot	0.0229	0.8206
KP	settling time	-0.0330	0.7444
KP	steady state error	0.0053	0.9578

Table 2: Correlation K_P vs Objectives

Var1	Var2	Correlation	p value
KI	closed-loop magnitude	0.1592	0.1134
KI	gain margin	0.9171	0.0000
KI	phase margin	-0.5803	0.0000
KI	rise time	-0.7483	0.0000
KI	peak time	-0.8298	0.0000
KI	overshoot	0.9528	0.0000
KI	undershoot	0.8941	0.0000
KI	settling time	0.1242	0.2179
KI	steady state error	-0.1902	0.0579

Table 3: Correlation K_I vs objectives

4.1 Analysis

1. In both of Figure 2 a&b , the pairwise subplots contain the scatterplot of the pair of objectives with least square reference line whose slope is the displayed correlation coefficient. The diagonals are the histogram. It can be clearly observed that there is a strong positive correlation between *closed-loop magnitude-gain margin/ phase margin*, *overshoot-undershoot*, *peak time-rise time* and strong negative correlation between *gain margin-phase margin* (2a),*peak time-undershoot*, *peak time-overshoot*. The negative correlations has to be keep in consideration specially while judging preference and priority based tradeoffs.

2. Table 3 strongly suggest the objectives are strongly correlated (positive/negative) with design variable KI. The highest positive correlation of *KI* is with *gain margin*, *overshoot* and *undershoot*. However, *KI* is strongly negatively correlated with both *phase margin* and *peak time* but lightly positively correlated with *closed-loop magnitude*. This will be a tradeoff because *closed-loop magnitude, gain margin , phase margin* are higher priority in chief engineer's criterion preferability.

3. The design variables along with each objective are plotted in Figure 3 to see the spread of the data.

5 Optimization Process

The optimization process is described in Algorithm 1. Multi-objective Genetic algorithm NSGA-II [4] has been used as the main backbone concept of the optimization process. NSGA-II is an elitist algorithm which finds the best sample population through sample distance ranking and non-dominated sorting. As described in Section 2, the optimization process here is an objective cost minimization task. In this work, the optimization process has gone through several scenarios to explore the best possible scenarios for reaching the desired objective goals while understanding mutual trade-offs between the objective priorities. The main backbones of the process are mentioned below.

5.1 Number of Iterations

One of the most critical hyperparameters of the optimization process is the total number of iterations for the process. Generally, up to a certain limit, the optimization becomes more optimal as the number of iterations increases. However, after a saturation point, the performance might decrease, and it is explored in this report in Section 6. In another approach, rather than fixing a number of iterations, the optimization process can run up to its convergence. We stop the loop in the convergence-based regime when the loss

Algorithm 1 Optimization process

```
1: procedure OPTIMIZATION(samples)
2:     Initialise the population with sampling function                ▷ Initialise and re-scale.
3:     Set number of iterations for the optimizer.
4:     while reach the number of iterations do
5:         Get the designs from the sampling plan.
6:         z ← optimizeControlSystem(samples)        ▷ output of PI simulation with objectives
7:         Calculate the fitness of the population.
8:         if objective priorities is True then
9:             fitness_rank ← rank(z)
10:        else
11:            Set Goals goals ← criterion_goals according to criterion objective.
12:            Set priorities of the objectives priorities ← criterion_objectives.
13:            fitness_rank ← rank(z, goals, priorities)
14:        End of If/Else
15:        Get the sample neighbour distance according to the ranks.
16:        crowding_distance ← crowding(z, fitness_rank)
17:        Select the samples for mutation.
18:        Get mutated children of the selected population.
19:        Concatenate the children with parent population samples.
20:        Calculate fitness of the whole population.
21:        Select n number of best designs and population from the pool.
22:        Calculate hypervolume to track the training and learning progress.
23:    End Loop
24:    Visualise the hypervolume                                       ▷ Final Visualisation
```

does not change over certain iterations. However, this strategy sometimes may be led to sub-optimal or local-optimal regions.

5.2 Accommodating the objective designs for minimization

Some of the PI module objectives (gain margin, phase margin) are modified to use them in the objective minimization optimization process. For example, the gain margin(dB) in PI output is converted to $\frac{1}{(1+gainmargin)}$ for using it in minimization objectives.

5.3 Ranking and priorities

One of the major aspects of NSGA-II is the fast non dominated sorted approach with $\mathcal{O}(MN^2)$, where N is the number of population and M is the number of objectives [4]. The algorithm finds the solutions it dominates for each design solution and derives all the non-dominated fronts. In this report, *rank_nds* function has been used to perform non dominated sorting. However, it treats all the objectives with the same priority. To introduce preferability and priority of the objectives given by the chief engineer, *rank_prf* function has been used to rank the designs. The main focus of this function is objective pair comparison with their weighted priorities and goals. The functions *rank_nds* and *rank_prf* has been explored with the possible implications on optimization on Section 6.

5.4 Crowding Distance

The crowding distance is a measure of density estimation of solutions surrounding a particular design solution [17]. Each objective function measures the average distance of the two neighbours, and the final distance is computed using all the distances of the objectives for a given sample. It is calculated in this work using the function *crowding*. Furthermore, the density can be used to explore the uncharted regions in the solution space.

5.5 Crossover and Mutation

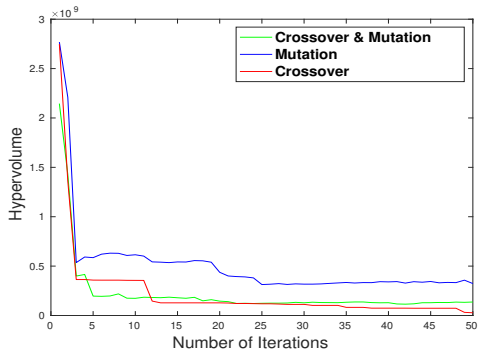
Binary tournament operator performs the selection of the population (*btwr* function). The selected population produces the new children population using three different scenarios to explore crossover and mutation to provide the best analysis of this optimization. For the first scenario, crossover and mutation have been used simultaneously in two halfs of the selected population. In the last two scenarios, crossover and mutation (*sbx* and *polymut*) have been used individually to the whole populations. The results have been discussed in Section 6.

5.6 Preferability Training

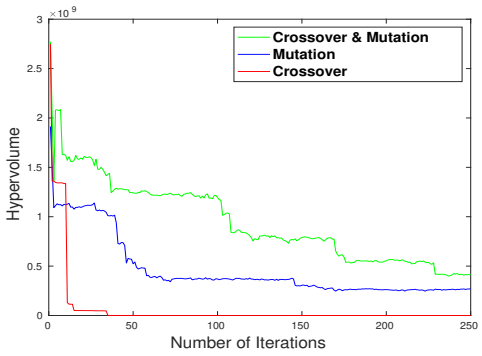
The chief engineer's preferences need to be optimized. Hence, as a final experiment, the optimization process is run in different stages using separate priority goals on different stages. The reason to run the optimization process in this way is to find the possible trade-offs among the objectives and constraints. Furthermore, generated population samples from previous stages have been used to explore if the transfer of knowledge from other stages is helpful for optimization. The optimizer is trained with the criterion preferences given by the chief engineer. Firstly, the largest closed-loop pole magnitude has been given the highest priority, the rest of the objectives are set to 0, and the low priority objectives are set to -inf (*preferability choice 1*). In the second scenario *Largest closed-loop pole* is set to 2 and the gain and loss margin set to 1, and the rest of the objective priorities remain the same as before (*preferability choice 2*). In the final scenario, all the priorities are set according to the chief engineer's choice (*preferability choice 3*). *preferability choice 3* is run for both 50 and 250 iterations.

6 Optimization Results

Same initial design variables are used throughout all the experiments to keep the initial population space consistent. This gives a better chance to compare the outcomes. The optimisation results are presented in Figure 4, 5 and Table 4. Figure 4 demonstrates the



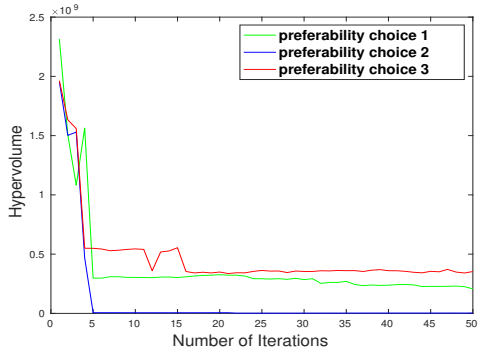
(a) Optimization scenarios for 50 iteration.



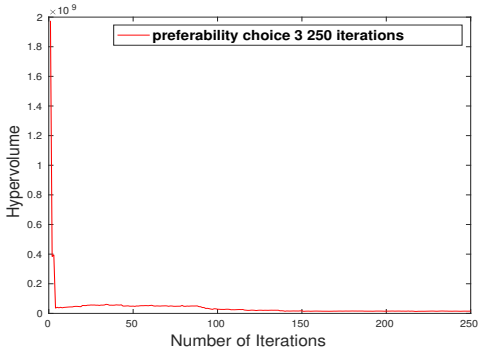
(b) Optimisation scenarios for 250 iteration.

Figure 4: Comparing child population sampling scenarios with crossover, mutation.

results from Section 5.5 experiments. The experiments are run for 50 and 250 epochs. It can be seen that population generation with only a crossover operator results in better convergence after both 50 and 250 iterations. However, generation with both crossover and mutation results in a balanced overall performance according to the MSE loss. The MSE loss is calculated on the final optimised design variables' PI output and the desired criterion values.



(a) Preference scenarios iteration=50.



(b) Chief engineers preference scenario for 250 iteration.

Figure 5: Comparing criterion preference scenarios over iterations.

Figure 5 shows the step by step execution of the desired criterion priorities. As discussed in Section 5.6, there are three scenarios in which the experiments are conducted. *Preferability choice 1* produces a slower convergence. However, things started to get interesting with *Preferability choice 2* where the hypervolume metric reduces quickly. In this scenario, gain margin and phase margin have been given equally high priority compared to the rest of the objectives. Overall it produces significantly better solutions compare to the non-priority optimization cases (Table 4). *Preferability choice 3* produces the best solutions after 250 iterations (Figure 5b and Table 4).

Mean square error (MSE) loss does not always represent the ideal scenario in optimization for the cost function. Nevertheless, it can indicate the quality of the solution

Optimisation Scenario	epochs	Mean square error
preferability choice 3	250	375.1392
preferability choice 3	50	545.6264
preferability choice 2	50	382.3713
preferability choice 1	50	459.2004
No-preferability crossover+mutation	50	519.3919
No-preferability crossover	50	535.8358
No-preferability Mutation	50	651.3058

Table 4: Final MSE loss for each optimization scenario compare to the desired goals

space up to a certain degree. Therefore, MSE loss has been used here to compute the final loss at the end of each optimization experiment case. Table 4 signifies the distance of the outcome to the desired outcomes. Also, in comparison to Figure 4 & 5, MSE demonstrates the fact that the convergence rate or the overall hypervolume does not always describe the final solution space. Sample population Ranking with preferability criterion results in better convergence and significantly better solution space.

Furthermore, the preferability operator should be used with caution. As we know from Section 4, the objectives might be highly correlated with each other, either positively or negatively. In a negative correlation, the criterion priority or preference might be a tradeoff on each other. It can be already seen with gain and phase margin.

After the 250th iteration of *Preferability choice 3* (chief engineer's preference, 250 iteration), the nearest solution with the desired goal according to MSE is given below. KP = 0.4155, KI = 0.3566 and the objectives largest closed-loop pole = 0.7506, gain margin = 6.84dB, phase margin = 35.0562, rise time = 2.7132, peak time = 8.000, maximum overshoot = 14.49, Maximum undershoot = 2.6655, settling time = 15.0143, steady state error = 0.00. Some of the the final objective does not meet the requirement but mostly all of them are close to the desired criterion.

7 Recommendations

The recommendations based on the knowledge discovery in Section 4, experiments in Section 5, and results in Section 6, the following recommendations are suggested

1. The correlation among objectives is highly impactful, as well as the correlation between the design variables and the objectives. As we recall from Section 4, gain margin and phase margin is negatively correlated as well as peak time and rise time with overshoot and undershoot. The effect can be clearly seen in the results based on the preference criterion. The priority in *rise time* is moderate but *maximum undershoot* priority is low. As a result, *rise time* stays near the desired output as 2.7132 (desired is 2) but *maximum undershoot* output becomes 2.66 (desired is 8). However, *gain margin* and *phase margin* as negatively correlated but their final outputs gain margin = 6.84dB, phase margin = 35.0562 remain near the desired

region (gain margin 6db and phase margin between 30 and 60). Similar behaviour can be observed with the other objectives and their correlations. Therefore it is recommended to use similar priority (preference) levels for the objectives if they are negatively correlated. For example, priority among *maximum undershoot* and *rise time* should be the same for achieving better solution design variables.

2. The number of iterations is a crucial factor in stimulating the outcomes of design variables and achieving optimal design variables. It is recommended that before choosing an optimal design variable, the optimization process should be executed on a different set of iterations and initial sample population to find the optimal solution.

3. After experimenting with different children solution population strategies from selected parent samples, it is recommended that both crossover and crossover & mutation may be used for the optimization process. They may vary in convergence speed, but the final outcome should be similar. However, design sample generation with only mutation is discouraged.

4. Finally, after experimenting with the optimization process, it is highly recommended to change the desired objective goal values given the high correlation among objectives and the tradeoff in the criterion priority. The values need to be changed, or the preferences should be strongly directed by the relationship between objectives and design variables.

8 Conclusion

This pilot study for optimizing gains for PI controller not only optimizes the PI gains with given objectives but also demonstrates a fully functional framework for vehicle design decision optimization with the given objective. Sections 5 and 6 demonstrated that the framework could be adapted to any design decision-based optimization problem due to the scalability of the tools and the functions. This type of framework can benefit your company by simulating and optimizing design decision boundary conditions and trade-offs. Furthermore, this can be adapted to other optimization algorithms such as SPGA, SPGA-II, and hybrid genetic swarm optimization algorithms. Because the problem formulation and the framework design are highly flexible and scalable, as demonstrated in this pilot study by using various scenarios with various adaptive modules.

Finally, with the advent of deep neural networks, this pilot study can be done with deep reinforcement learning and graph-based discriminative networks. This would be a viable future work further to enhance the design efficiencies of your company's vehicle powertrain.

References

[1] I. Kolmanovsky, I. Siverguina, and B. Lygoe, "Optimization of powertrain operating policy for feasibility assessment and calibration: stochastic dynamic programming approach," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 2, 2002, pp. 1425–1430 vol.2.

[2] X. Wang, Y. Liu, W. Sun, X. Song, and J. Zhang, "Multidisciplinary and multifidelity design optimization of electric vehicle battery thermal management system," *Journal of Mechanical Design*, vol. 140, no. 9, p. 094501, 2018.

[3] W. ElMaraghy, H. ElMaraghy, T. Tomiyama, and L. Monostori, "Complexity in engineering design and manufacturing," *CIRP Annals*, vol. 61, no. 2, pp. 793–814, 2012.

[4] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 182–197, 2002.

[5] B. Sakhdari and N. Azad, "An optimal energy management system for battery electric vehicles," *IFAC-PapersOnLine*, vol. 48, no. 15, pp. 86–92, 2015, 4th IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling E-COSM 2015.

[6] W. S. Vaz, A. K. Nandi, and U. O. Koylu, "A multiobjective approach to find optimal electric-vehicle acceleration: Simultaneous minimization of acceleration duration and energy consumption," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 4633–4644, 2015.

[7] B. Gadhvi, V. Savsani, and V. Patel, "Multi-objective optimization of vehicle passive suspension system using nsga-ii, spea2 and pesa-ii," *Procedia Technology*, vol. 23, pp. 361–368, 2016, 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016 05-06 February, 2016.

[8] T. Deng, C. Lin, J. Luo, and B. Chen, "Nsga-ii multi-objectives optimization algorithm for energy management control of hybrid electric vehicle," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 233, no. 4, pp. 1023–1034, 2019.

[9] A. E. Babalola, B. A. Ojokoh, and J. B. Odili, "A review of population-based optimization algorithms," in *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*, 2020, pp. 1–7.

[10] S. Cheng, B. Liu, T. Ting, Q. Qin, Y. Shi, and K. Huang, "Survey on data science with population-based algorithms," *Big Data Analytics*, vol. 1, no. 1, pp. 1–20, 2016.

[11] R. C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *International conference on evolutionary programming*. Springer, 1998, pp. 611–616.

[12] R. Hassan, B. Cohanim, O. De Weck, and G. Venter, "A comparison of particle swarm optimization and the genetic algorithm," in *46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference*, 2005, p. 1897.

[13] N. B. Sariff and N. Buniyamin, "Genetic algorithm versus ant colony optimization algorithm."

[14] I. A. Ashari, M. A. Muslim, and A. Alamsyah, "Comparison performance of genetic algorithm and ant colony optimization in course scheduling optimizing," *Scientific Journal of Informatics*, vol. 3, no. 2, pp. 149–158, 2016.

[15] S. Kucherenko, D. Albrecht, and A. Saltelli, "Exploring multi-dimensional spaces: A comparison of latin hypercube and quasi monte carlo sampling techniques," *arXiv preprint arXiv:1505.02350*, 2015.

[16] M. D. Morris and T. J. Mitchell, "Exploratory designs for computational experiments," *Journal of Statistical Planning and Inference*, vol. 43, no. 3, pp. 381–402, 1995.

[17] C. R. Raquel and P. C. Naval Jr, "An effective use of crowding distance in multiobjective particle swarm optimization," in *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation*, 2005, pp. 257–264.

9 Appendix

MATLAB codes used : Including own and lab provided.

```
% main function sampling plan generation and knowledge discovery

clear all;

% derive sampling plan
% samplingPlan = fullfactorial([10 10],2);

% samplingPlan = rescale(samplingPlan, 0.05,0.5);
% derive PI output conditions based on sampling plan
% PIoutput = optimizeControlSystem(samplingPlan);

load("saved/sampling_plan.mat","samplingPlan");
```

```
priority = [1,0,0,0,-inf,0,-inf,-inf,0];
goal = [0.99,0.142,7.5,2,10,10,8,20,1];

[samplingPlanPriority50_1,hypervolumePriority50_1] = optimisation(samplingPlan,goal,priority,50,1);
save("saved/hypervolumePriority50_1.mat","hypervolumePriority50_1");
save("saved/sampling_planPriority50_1.mat","samplingPlanPriority50_1");


priority = [1,2,2,0,-inf,0,-inf,-inf,0];
goal = [0.99,0.142,7.5,2,10,10,8,20,1];

[samplingPlanPriority50_2,hypervolumePriority50_2] = optimisation(samplingPlan,goal,priority,50,1);
save("saved/hypervolumePriority50_2.mat","hypervolumePriority50_2");
save("saved/sampling_planPriority50_2.mat","samplingPlanPriority50_2");

priority = [2,1,1,0,-inf,0,-inf,-inf,0];
goal = [0.99,0.142,7.5,2,10,10,8,20,1];

[samplingPlanPriority50_3,hypervolumePriority50_3] = optimisation(samplingPlan,goal,priority,50,1);
save("saved/hypervolumePriority50_3.mat","hypervolumePriority50_3");
save("saved/sampling_planPriority50_3.mat","samplingPlanPriority50_3");

priority = [2,1,1,0,-inf,0,-inf,-inf,0];
goal = [0.99,0.142,7.5,2,10,10,8,20,1];

[samplingPlanPriority250_3,hypervolumePriority250_3] = optimisation(samplingPlan,goal,priority,250,1
save("saved/hypervolumePriority250_3.mat","hypervolumePriority250_3");
save("saved/sampling_planPriority250_3.mat","samplingPlanPriority250_3");



load("saved/sampling_plan.mat","samplingPlan");

[samplingPlan50_1,hypervolume_nextPhase50_1] = optimisationNopreference(samplingPlan,250,1);
save("saved/hypervolume50_1.mat","hypervolume_nextPhase50_1");
save("saved/samplingPlan50_1.mat","samplingPlan50_1");

[samplingPlan50_2,hypervolume_nextPhase50_2] = optimisationNopreference(samplingPlan,250,2);
save("saved/hypervolume50_2.mat","hypervolume_nextPhase50_2");
save("saved/samplingPlan50_2.mat","samplingPlan50_2");

[samplingPlan50_3,hypervolume_nextPhase50_3] = optimisationNopreference(samplingPlan,250,3);
save("saved/hypervolume50_3.mat","hypervolume_nextPhase50_3");
save("saved/samplingPlan50_3.mat","samplingPlan50_3");

[samplingPlan50_4,hypervolume_nextPhase50_4] = optimisationNopreference(samplingPlan,250,4);
save("saved/hypervolume50_4.mat","hypervolume_nextPhase50_4");
```

```
save("saved/samplingPlan50_4.mat","samplingPlan50_4");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% optimisation function

function [samplingPlan, hypervolume]=optimisation(samplingPlan,goal,priority,iterations,choice)

PIoutput = optimizeControlSystem(samplingPlan);

for loop=1:iterations

dominance_ranks = rank_prf(PIoutput,goal, priority);
fitness_dominance = dominance_ranks;
% for i=1:length(fitness_dominance)
%     fitness_dominance(i) = max(dominance_ranks)-fitness_dominance(i);
% end
crowding_distance = crowding(PIoutput,fitness_dominance);
selectThese = btwr(crowding_distance);

selectProcessed = unique(selectThese);

validPopulation = samplingPlan(selectProcessed,:);

minKP = min(validPopulation(:,1));
maxKP = max(validPopulation(:,1));
minKI = min(validPopulation(:,2));
maxKI = max(validPopulation(:,2));


if choice==1

    for i=1:length(validPopulation)/2
        childPopulation(i,1) = polymut(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = polymut(validPopulation(i,2),[minKI;maxKI]);
    end

    for i=floor(length(validPopulation)/2):length(validPopulation)
        childPopulation(i,1) = sbx(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = sbx(validPopulation(i,2),[minKI;maxKI]);
    end

elseif choice ==2

    for i=1:length(validPopulation)
        childPopulation(i,1) = polymut(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = polymut(validPopulation(i,2),[minKI;maxKI]);
    end
```

```
elseif choice ==3

    for i=1:length(validPopulation)
        childPopulation(i,1) = sbx(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = sbx(validPopulation(i,2),[minKI;maxKI]);
    end

elseif choice ==4

    for i=length(validPopulation)
        childPopulation(i,1) = sbx(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = sbx(validPopulation(i,2),[minKI;maxKI]);
    end

    for i=1:length(validPopulation)
        childPopulation(i,1) = polymut(childPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = polymut(childPopulation(i,2),[minKI;maxKI]);
    end
end


concat_population = [samplingPlan;childPopulation];

PIoutput = optimizeControlSystem(concat_population);
dominance_ranks = rank_nds(PIoutput);
fitness_dominance = dominance_ranks;
% for i=1:length(fitness_dominance)
%     fitness_dominance(i) = max(dominance_ranks)-fitness_dominance(i);
% end
crowding_distance = crowding(PIoutput,fitness_dominance);


selected_population = reducerNSGA_II(concat_population,fitness_dominance,crowding_distance,100);
samplingPlan = concat_population(selected_population,:);
PIoutput = optimizeControlSystem(samplingPlan);
hypervolume(loop,1) = Hypervolume_MEX(PIoutput);

end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [samplingPlan, hypervolume]=optimisationNopreference(samplingPlan,iterations,choice)
```

```
PIoutput = optimizeControlSystem(samplingPlan);

for loop=1:iterations

dominance_ranks = rank_nds(PIoutput);
fitness_dominance = dominance_ranks;
% for i=1:length(fitness_dominance)
%     fitness_dominance(i) = max(dominance_ranks)-fitness_dominance(i);
% end
crowding_distance = crowding(PIoutput,fitness_dominance);
selectThese = btwr(crowding_distance);

selectProcessed = unique(selectThese);

validPopulation = samplingPlan(selectProcessed,:);

minKP = min(validPopulation(:,1));
maxKP = max(validPopulation(:,1));
minKI = min(validPopulation(:,2));
maxKI = max(validPopulation(:,2));

if choice==1

    for i=1:length(validPopulation)/2
        childPopulation(i,1) = polymut(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = polymut(validPopulation(i,2),[minKI;maxKI]);
    end

    for i=floor(length(validPopulation)/2):length(validPopulation)
        childPopulation(i,1) = sbx(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = sbx(validPopulation(i,2),[minKI;maxKI]);
    end

elseif choice ==2

    for i=1:length(validPopulation)
        childPopulation(i,1) = polymut(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = polymut(validPopulation(i,2),[minKI;maxKI]);
    end

elseif choice ==3

    for i=1:length(validPopulation)
        childPopulation(i,1) = sbx(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = sbx(validPopulation(i,2),[minKI;maxKI]);
    end
```

```
elseif choice ==4

    for i=length(validPopulation)
        childPopulation(i,1) = sbx(validPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = sbx(validPopulation(i,2),[minKI;maxKI]);
    end

    for i=1:length(validPopulation)
        childPopulation(i,1) = polymut(childPopulation(i,1),[minKP;maxKP]);
        childPopulation(i,2) = polymut(childPopulation(i,2),[minKI;maxKI]);
    end
end


concat_population = [samplingPlan;childPopulation];

PIoutput = optimizeControlSystem(concat_population);
dominance_ranks = rank_nds(PIoutput);
fitness_dominance = dominance_ranks;
% for i=1:length(fitness_dominance)
%     fitness_dominance(i) = max(dominance_ranks)-fitness_dominance(i);
% end
crowding_distance = crowding(PIoutput,fitness_dominance);


selected_population = reducerNSGA_II(concat_population,fitness_dominance,crowding_distance,100);
samplingPlan = concat_population(selected_population,:);
PIoutput = optimizeControlSystem(samplingPlan);
hypervolume(loop,1) = Hypervolume_MEX(PIoutput);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% plotting graphs


load("saved/hypervolumePriority50_1.mat","hypervolumePriority50_1");
load("saved/hypervolumePriority50_2.mat","hypervolumePriority50_2");
load("saved/hypervolumePriority50_3.mat","hypervolumePriority50_3");
load("saved/hypervolumePriority250_3.mat","hypervolumePriority250_3");


plot(hypervolume_nextPhase50_1,"Color","0 1 0","DisplayName","Crossover & Mutation");
hold on
```

```
plot(hypervolume_nextPhase50_2,"Color","0 0 1","DisplayName","Mutation");
hold on
plot(hypervolume_nextPhase50_3,"Color","1 0 0","DisplayName","Crossover");
hold on
legend("FontSize",14,FontWeight="bold");
legend
xlabel("Number of Iterations","FontSize",16);
ylabel("Hypervolume","FontSize",16);
% title("Comparison among scenarios with crossover and mutation");

plot(hypervolumePriority50_1,"Color","0 1 0","DisplayName","preferability choice 1");
hold on
plot(hypervolumePriority50_2,"Color","0 0 1","DisplayName","preferability choice 2");
hold on
plot(hypervolumePriority50_3,"Color","1 0 0","DisplayName","preferability choice 3");
hold on
legend("FontSize",14,FontWeight="bold");
legend
xlabel("Number of Iterations","FontSize",16);
ylabel("Hypervolume","FontSize",16);


plot(hypervolumePriority250_3,"Color","1 0 0","DisplayName","preferability choice 3 250 iterations")
hold on
legend("FontSize",14,FontWeight="bold");
legend
%set(gca, 'yLim', [0, 1.5]);
xlabel("Number of Iterations","FontSize",16);
ylabel("Hypervolume","FontSize",16);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% Simulation model for a PI controller.
%
% Z = evaluateControlSystem(X);
%
% Input:  X - a sampling plan
%             (one design per row, one design variable per column)
% Output: Z - performance evaluations
%             (one design per row, one criterion per column;
%              criteria are...
%              1: maximum closed-loop pole magnitude
%              2: gain margin
%              3: phase margin
%              4: 10-90% rise time
%              5. peak time
```

```
%              6. overshoot (% points)
%              7. undershoot (% points)
%              8. 2% settling time
%              9. steady-state error (% points))
%
function Z = optimizeControlSystem(X);

[noInds, noVar] = size(X);

Z = NaN * ones(noInds,9);

warning off

for ind = 1:noInds

    % Extract the candidate controller gains.
  kp=X(ind,1);
  ki=X(ind,2);

  % Form the open-loop transfer function.
  olNum = [(kp+ki)*(1-exp(-1)) ...
           -kp*(1-exp(-1))];

  olDen = [1 ...
           -1-exp(-1) ...
           exp(-1) ...
           0 ...
           0];

  % Form the closed-loop transfer function.
  clNum=[(1-exp(-1))*(kp+ki) ...
         -(kp/(kp+ki))*(1-exp(-1))*(kp+ki)];
  clDen=[1 ...
         -(1+exp(-1)) ...
         exp(-1) ...
         (1-exp(-1))*(kp+ki) ...
 -(kp/(kp+ki))*(1-exp(-1))*(kp+ki)];

  % Get stability measure.
  sPoles = roots(clDen);
  clStable = max(abs(sPoles));

  olTF = tf(olNum, olDen, 1.0);

  [gainMargin, phaseMargin, wcGP, wcPM] = margin(olTF);

  % modify gain margin for optimizer
```

```
  modifiedGainMargin = 1./(1+(20*log10(gainMargin)));

  % Do a unit step response.
  timeData = 0:1:100;
  outputData = dstep(clNum, clDen, timeData);

  % Collect results where possible (stable).
  if clStable < 1
    riseTime = getRiseTime(timeData, outputData, outputData(end));
    settleTime = getSettlingTime(timeData, outputData, 0.02, ...
outputData(end));
  else
    riseTime = getRiseTime(timeData, outputData, 1.0);
    settleTime = getSettlingTime(timeData, outputData, 0.02, 1.0);
  end
  [overshoot, overTime, undershoot, underTime] = getShoots(timeData, outputData);
  ssError = getSSError(outputData);
  phaseMargin = abs(45-phaseMargin);
%   % Assign to output variable.
  Z(ind,1) = clStable;
  Z(ind,2) = modifiedGainMargin;   %-1* for minimisation
  Z(ind,3) = phaseMargin;  %-1* for minimisation
  Z(ind,4) = riseTime;
  Z(ind,5) = overTime;
  Z(ind,6) = 100*overshoot;
  Z(ind,7) = 100*undershoot;
  Z(ind,8) = settleTime;
  Z(ind,9) = 100*ssError;

  warning on

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function correlationPlot(X,label,plot_name,fig)
figure(fig);
corrplot(X,VarNames=label);
title(plot_name);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function correlatioVariables(var1,var2,var1names,var2names)

for i=1:length(var2)
```

```
    if isinf(var2(i,8))
        var2(i,8) = nan;
    end
end
for i=1:length(var1(1,:))
        for j=1:length(var2(1,:))
            [R,P] = corrcoef(var1(:,i),var2(:,j),"Rows","complete");
            fprintf("Correlation between %s and %s is %f and p value is %s . \n",var1names(i),var2nar
        end
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% mse calculation


load("saved/sampling_planPriority50_1.mat","samplingPlanPriority50_1");
load("saved/sampling_planPriority50_2.mat","samplingPlanPriority50_2");
load("saved/sampling_planPriority50_3.mat","samplingPlanPriority50_3");
load("saved/sampling_planPriority250_3.mat","samplingPlanPriority250_3");

load("saved/no_priority_250/samplingPlan50_2.mat","samplingPlan50_2")

goal = [0.99,0.142,7.5,2,10,10,8,20,1];

mse_loss = 0;

objective = optimizeControlSystem(samplingPlanPriority250_3);
for i=1:length(objective)
    mse_loss = mse_loss + immse(objective(i,:),goal);
end
disp(mse_loss/100);

temp_loss = 9999999999999999;
index = 0;
for i=1:length(objective)
     if temp_loss > immse(objective(i,:),goal)
         temp_loss = immse(objective(i,:),goal);
         index = i;
     end
end
disp(objective(index,:));
disp(samplingPlanPriority50_3(index,:));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% main function sampling plan generation and knowledge discovery

clear all;


sobol_object = sobolset(2,'Skip',100,'Leap',100);
samplingPlan_sobol = net(sobol_object,100);
samplingPlan_sobol = rescale(samplingPlan_sobol,0.05,0.5);
figure(1);
scatter(samplingPlan_sobol(:,1),samplingPlan_sobol(:,2),"filled");
title("Sobol sequence samples");
grid on;

samplingPlan_rlh = rlh(100,2);
samplingPlan_rlh = rescale(samplingPlan_rlh, 0.05,0.5);
figure(2);
scatter(samplingPlan_rlh(:,1),samplingPlan_rlh(:,2),"filled");
title("Random Latin hypercube samples");
grid on;


samplingPlan_factorial = fullfactorial([10 10],2);
samplingPlan_factorial = rescale(samplingPlan_factorial, 0.05,0.5);
figure(3);
scatter(samplingPlan_factorial(:,1),samplingPlan_factorial(:,2),"filled");
title("Full factiorial samples");
grid on;

samplingPlan_randor = lhsdesign(100,2);
samplingPlan_randor = rescale(samplingPlan_randor, 0.05,0.5);
figure(4);
scatter(samplingPlan_randor(:,1),samplingPlan_randor(:,2),"filled");
title("Latin hypercube samples");
grid on;

best1 = mm(samplingPlan_factorial,samplingPlan_rlh);
best2 = mm(samplingPlan_sobol,samplingPlan_rlh);
best3 = mm(samplingPlan_sobol,samplingPlan_factorial);
best4 = mm(samplingPlan_factorial,samplingPlan_randor);

phi_sobol = mmphi(samplingPlan_sobol);
phi_rlh = mmphi(samplingPlan_rlh);
phi_fac = mmphi(samplingPlan_factorial);
phi_lhs = mmphi(samplingPlan_randor);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% used function %%%%
%
% reducerNSGA_II.M
%
% newPop = reducerNSGA_II(unifiedPop, ranks, crowdings, noToSelect)
%
% NSGA II clustering procedure.
% Selects the new parent population from the unified population
% of previous parents and offspring.
%
% This is an M-File for MATLAB.
% Written by Robin Purshouse, 23-Oct-2002
%
% Inputs: unifiedPop - the combined parent and offspring populations
%         ranks      - ranking of the combined population
%         crowdings  - density estimates (large values = more remote)
%         noToSelect - size of newPop, defaults to 1/2 unifiedPop
%
% Output: selected   - indices for the new parent population
%
% Reference: Deb, K., 2001, 'Multi-Objective Optimization using
%            Evolutionary Algorithms', Chichester: Wiley, pp233-241.

function selected = reducerNSGA_II(unifiedPop, ranks, crowdings, noToSelect)

% Check for correct number of inputs.
if nargin < 3
  error('At least three inputs are required.');
end

% Gather input information, and check that it's OK.
popSize = size(unifiedPop, 1);
rankSize = size(ranks, 1);
crowdingSize = size(crowdings, 1);

if popSize ~= rankSize
  error('ranks vector is not of correct size');
end

if popSize ~= crowdingSize
  error('crowdings vector is not of correct size');
end

% Generate noToSelect if not provided.
if nargin < 4
  noToSelect = floor(popSize / 2);
```

```
end

% Handle quick returns.
selected = [];
if noToSelect >= popSize
  selected = [1:popSize]';
  return;
elseif noToSelect <= 0
  return;
end

% Go through the ranks and add to the new population until we spill over.
full = 0;
currentRank = -1;
noNewPop = 0;
while(~full)
  currentRank = currentRank + 1;
  thisRank = find(ranks == currentRank);
  noThisRank = size(thisRank, 1);
  if( (noNewPop + noThisRank) < noToSelect )
    selected = [selected; thisRank];
    noNewPop = noNewPop + noThisRank;
  else
    full = 1;
  end
end

% Shuffle the front that has spilled over, and put in as many as possible
% in terms of crowding distance.
thisRank = thisRank(randperm(noThisRank)');
[reRank, reRankI] = sort(-1*crowdings(thisRank));
selected = [selected; thisRank(reRankI(1:noToSelect-noNewPop))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X = fullfactorial(q,Edges)
% Generates a full factorial sampling plan in the unit cube
%
% Inputs:
%       q - k-vector containing the number of points along each dimension
%       Edges - if Edges=1 the points will be equally spaced from edge to
%               edge (default), otherwise they will be in the centres of
%               n = q(1)*q(2)*...q(k) bins filling the unit cube.
%
% X - full factorial sampling plan
%
% Copyright 2007 A Sobester
```

```
%
% This program is free software: you can redistribute it and/or modify  it
% under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or any
% later version.
%
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser
% General Public License for more details.
%
% You should have received a copy of the GNU General Public License and GNU
% Lesser General Public License along with this program. If not, see
% <http://www.gnu.org/licenses/>.

if nargin < 2, Edges=1; end

if min(q) < 2
error('You must have at least two points per dimension.');
end

% Total number of points in the sampling plan
n = prod(q);

% Number of dimensions
k = length(q);

%Pre-allocate memory for the sampling plan
X = zeros(n,k);

%Additional phantom element
q(k+1)=1;

for j=1:k
if Edges==1
one_d_slice = (0:1/(q(j)-1):1);
else
one_d_slice = (1/q(j)/2:1/q(j):1);
end

column = [];

while length(column) < n
for l=1:q(j)
column = [column; ones(prod(q(j+1:k)),1)*one_d_slice(l)];
end
end
```

```
X(:,j) = column;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CROWDING.M
%
% NSGA-II density estimator
%
% distances = crowding(popData, ranking)
%
function distances = crowding(popData, ranking);

% Get dimensions.
[noSols, noObjs] = size(popData);
[rankRows, rankCols] = size(ranking);

% Error handling.
if (rankRows ~= noSols) | (rankCols ~= 1)
   error('Inconsistent input data.');
end

% Set up the output vector.
distances = zeros(noSols, 1);

% Get maximum rank.
maxRank = max(ranking);

% Get the bounds on the objectives.
minObj = min(popData);
maxObj = max(popData);

% Perform the density estimation in a rank-wise fashion.
for rank = 0:maxRank
   % Get indices to solutions of this rank.
   solIndices = find(ranking == rank);

   % Only do the estimation if we have some data.
   if ~isempty(solIndices)

      % Grab the subset of data.
      subsetPopData = popData(solIndices, :);

      % Get the bounds on the objectives.
      %minObj = min(subsetPopData);
      %maxObj = max(subsetPopData);
```

```
      % Do the NSGA-II crowding.
      distances(solIndices) = crowdingNSGA_II(subsetPopData, minObj, maxObj);
   end
end

%infIndices = find(distances == -1);
%if ~isempty(infIndices)
%   distances(infIndices) = Inf;
%end

% Deb's method biases in favour of boundary solutions at equilibrium - not correct.
maxDistance = max(distances);
infIndices = find(distances == -1);
if ~isempty(infIndices)
   distances(infIndices) = maxDistance;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Phiq = mmphi(X, q, p)
% Calculates the sampling plan quality criterion of Morris and Mitchell.
%
% Inputs:
%       X - sampling plan
%       q - exponent used in the calculation of the metric
%       p - the distance metric to be used (p=1 rectangular - default, p=2
%           Euclidean)
%
% Output:
%       Phiq - sampling plan 'space-fillingness' metric
%
% Copyright 2007 A Sobester
%
% This program is free software: you can redistribute it and/or modify  it
% under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or any
% later version.
%
% This program is distributed in the hope that it will be useful, but
% WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser
% General Public License for more details.
%
% You should have received a copy of the GNU General Public License and GNU
% Lesser General Public License along with this program. If not, see
% <http://www.gnu.org/licenses/>.

% Assume defaults if arguments list incomplete
```

```
if ~exist('p','var')
    p = 1;
end

if ~exist('q','var')
    q = 2;
end

% Calculate the distances between all pairs of
% points (using the p-norm) and build multipli-
% city array J
[J,d] = jd(X,p);

% The sampling plan quality criterion
Phiq = sum(J.*(d.^(-q)))^(1/q);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```