# Part II – Decision Systems for Engineering Design
# Laboratory B: Optimization and Decision Support

*Robin Purshouse*
University of Sheffield

Spring Semester 2023/24

## 1 Introduction to the laboratories

### 1.1 Overview

In the laboratories for Part II of the module, you will use decision system technologies to design a digital control system for a plant. In your assignment, you will report on your lab activities and findings (see the separate Assignment Briefing for more details about the assessment requirements).

### 1.2 Aim of Laboratory B

In Laboratory B, your aim is to identify a set of controller gains that will meet the preferences of the Chief Engineer or—in case the preferences cannot be satisfied—communicate to the Chief Engineer the trade-offs that will need to be made in order to resolve the design problem.

### 1.3 Materials

In preparation for this laboratory, go to the ACS6124 Blackboard site and download the following files from the *Decision Systems* Laboratories area, and place them in your working directory (or other convenient location on the Matlab path):

- `EA_Toolbox`—this is a set of components that can be used to assemble multi-objective optimizers. Some of the components were written by Carlos Fonseca from the University of Coimbra, during his time as a researcher in ACSE.
- `Hypervolume.zip`—this set of files enable the hypervolume metric to be computed in Matlab. It was written by a collaborative group of researchers: Carlos Fonseca, Manuel López-Ibánez, Luís Paquete and Andreia Guerreiro

For computational performance reasons, both the hypervolume metric and many of the components in the toolbox are written in C with a Matlab wrapper. To enable these functions to be used in Matlab, you will need to compile them for your system using the `mex` command. For example, to compile the simulated binary crossover operator `sbx.c` you need to use the command:

```
mex sbx.c
```

The hypervolume metric requires a more involved compilation. In this case (only), use the command:

```
mex -DVARIANT=4 Hypervolume_MEX.c hv.c avl.c
```

You may need to install or configure a compiler for your operating system to undertake the required compilation. For guidance, see the associated Matlab documentation: `https://www.mathworks.com/help/matlab/build-cc-mex-files-1.html`.

Note that you will also need the files from Laboratory A to undertake this second laboratory.

# 2 The decision problem

## 2.1 Performance criteria

In the previous lab, 10 performance criteria were identified for the digital control system. The Chief Engineer has now expressed preferences for levels of attainment against these criteria. These preferences are given in Table 1.

It is evident that the Chief Engineer has different priorities for different criteria. For example, largest closed-loop pole magnitude is a hard constraint (the value must be less than unity), whilst the undershoot is a less important criterion to be satisfied. The Chief Engineer has also provided target levels of performance to be met across all criteria.

# 3 Task B.0 Preliminaries

In Laboratory A, you used the function `evaluateControlSystem` to evaluate a design. This function is a little 'raw' for the purposes of optimization and requires some post-processing to be added:

1. The gain margin, $G_M$, is evaluated as a ratio of gains and needs converting to decibels using the conversion factor $20 \log_{10} G_M$;
2. The EA Toolkit assumes that all objectives must be minimised; however the gain margin needs to be maximised. This conversion can be achieved by multiplying the gain margin by -1 or switching it to the denominator using $1/(1+20 \log_{10} G_M)$.

| Criterion | Direction | Goal | Priority |
|---|---|---|---|
| Largest closed-loop pole | Minimise | $<1$ | Hard constraint |
| Gain margin | Maximise | 6 dB | High |
| Phase margin | Range | $\geq 30^o$ & $\leq 70^o$ | 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 1: Chief Engineer preferences for the digital control system

3. Rather than be maximised or minimised, the phase margin should lie within a range. This can be achieved by either setting up two separate objectives (one for maximisation and one for minimisation) or by minimising the absolute deviation from the mid-point of the range, subject to a satisficing goal of half the range.

4. As you will have already experienced through your sampling of the system in Laboratory A, if the closed-loop system is unstable then it may not be possible to compute sensible values for the transient or steady-state performance criteria. For the purposes of optimization, it would be prudent to place limits on the minimum and maximum values achievable for each of the criteria (otherwise operators that rely on aggregating objectives—e.g. crowding distance—may produce garbage output).

**You need to write the post-processing routines yourself.** In what follows, the post-processed evaluation function is known as `optimizeControlSystem`.

# 4 Task B.1: Build the optimization engine

In this part of the lab, you will use the EA toolbox to construct a basic multi-objective optimizer: NSGA-II. These notes will take you through the process step-by-step, but will require you to think a little for yourself in order to code the overall algorithm. A description of the NSGA-II optimizer can be found in the notes for Lecture 3. You are also free to implement an alternative multi-objective optimizer if you wish.

## 4.1 Initialising the population

You can use your favoured sampling plan from Lab A as your initial population, here denoted as `P`. The first thing to do is re-evaluate the designs in the sampling plan, according to the post-processed evaluation function:

```
Z = optimizeControlSystem(P)
```

## 4.2 Calculating fitness

We are now going to enter the loop of the optimizer and you will need to write some code to represent the loop (e.g. by using the `while` operator).

Inside the loop, the first thing to do is calculate the fitness of the candidate designs. Since we are using the NSGA-II optimizer, there are two parts to this: non-dominated sorting and crowding distance.

### 4.2.1 Non-dominated sorting

Non-dominated sorting has been implemented in the mex-file `rank_nds`. The function takes the performance data as its only argument and returns the associated dominance ranks of each design. The locally non-dominated designs in the population are assigned rank zero.

From the perspective of fitness, higher fitnesses are conventionally regarded as better—so the dominance rank will need inverting to be expressed in terms of fitness.

### 4.2.2 Crowding distance

Crowding distance has been implemented in the m-file `crowding`. The function takes the performance data as its first argument and the dominance ranks as its second argument (recall that distances are only calculated between solutions of equivalent rank).

## 4.3 Performing selection-for-variation

NSGA-II uses binary tournament selection to choose the designs which will be subjected to variation operators (sometimes known as the "mating pool"). This selection operator has been implemented using the `btwr` mex-file. The function assumes that higher fitnesses are better. Use the command `help btwr` for more details about the operator.

## 4.4 Performing variation

The canonical NSGA-II algorithm includes two variation operators: simulated binary crossover (which takes two parents and returns two children) and polynomial mutation (which takes one parent and returns one child). These functions have been implemented as the mex-files `sbx` and `polymut` respectively. Refer to the help files for information on how to use and configure these operators.

## 4.5 Performing selection-for-survival

By this stage in the optimization loop, you should be dealing with two populations of designs: the original parents `P` and the post-variation children `C`. NSGA-II is an elitist

algorithm, which will take forward the best designs from across the two populations to form the next iteration of the optimizer.

Firstly, the child population of designs will need to be evaluated. Next, fitnesses for all designs (including the parents) will need to be calculated using the concatenated set of parent and child evaluations. The fitnesses can then be passed to NSGA-II's selection-for-survival operator `reducerNSGA_II`. The function will return the indices from the concatenated set that should be preserved going forward.

### 4.6 Monitoring convergence

It can be useful to check the convergence of the optimizer—either for diagnostic purposes or to act as a termination criterion for the optimization loop. The hypervolume indicator (which we encountered in Lecture 4 in the context of set-based optimizers) is a suitable choice of convergence metric. It has been implemented in the function `Hypervolume_MEX`. You will need to inspect the first few lines of the `Hypervolume_MEX.c` file to understand its usage.

### 4.7 Analysing results

Run the NSGA-II optimizer for 250 iterations using a population size of 100. What does the resulting trade-off surface look like? Have you managed to achieve a design that satisfies the Chief Engineer's preferences?

## 5 Task B.2: Apply preference information

In this next part of the lab, you will incorporate preferences into the multi-objective optimizer. Here, we will use the preferability operator introduced in Lecture 5, but you are free to use any other scheme of your choosing.

### 5.1 Incorporating preferability

Recalling that preferability can act as a direct replacement for the weak dominance operator, it is simply a case of replacing the non-dominated sorting procedure, `rank_nds`, in the NSGA-II algorithm with the preferability operator, `rank_prf`.

The mex-file `rank_prf` takes three arguments: a matrix of design evaluations, a vector of goals, and a vector of priorities. Note that the lowest priority level is denoted as zero in this implementation (rather than one, as given in the lecture notes).

### 5.2 Progressive use of preferences

In the lab, the optimization process is not strictly interactive, since the Chief Engineer is not on hand to be involved 'in the loop'. However, it might still be wise to consider a progressive incorporation of the Chief's preferences. The following guidance should be treated as general advice—it is not guaranteed to give the best results:

1. Run the optimizer for 50 iterations with the largest closed-loop pole criterion at priority level 1 and all other criteria at priority level 0. Do not set goals for the low priority criteria (i.e. set the goals to `-inf`). Are you starting to see a population of designs that is predominantly closed-loop stable?

2. Run the optimizer for a further 50 iterations with the largest closed-loop pole at priority level 2, the other stability criteria and the control effort at priority level 1, and all remaining criteria at priority level 0. Again, do not set goals for the lowest priority criteria. You can either re-start the optimizer from the sampling plan, or use the optimizer's intermediate population (the latter may be more successful if you have a reasonably high mutation rate, since the optimizer may need fresh material in order to redirect its search). Are you starting to see stable designs that satisfy the gain margin, phase margin and control effort criteria?

3. Run the optimizer for a further 50 iterations with priority levels and goals that reflect the full range of Chief Engineer's preferences in Table 1. Again, you may wish to re-start the optimizer or continue from the present iteration. Can all the preferences be satisfied, and do you see trade-offs emerging between the different performance criteria?

4. Run the optimizer for a final 50 iterations relaxing the priority level of control effort to be equal to that of rise time, overshoot and steady-state error. What does this tell you about the relationship between control effort and those transient criteria?

# 6 Task B.3: Perform decision support

Once you have completed the optimization process, your objectives are to:

- Identify whether or not a design has been found that satisfies all of the Chief Engineer's preferences for the control system;
- Identify key trade-offs between objectives, harmonious relationships between objectives, and relationships between the design variables and performance criteria for the problem, focusing on the region of optimality;
- Identify what trades with other criteria are necessary to satisfy more demanding sustainability performance (i.e. tightening the goal on control effort);
- Identify whether or not the optimizer has converged;
- Present these findings to a Chief Engineer in a compelling way.

As in the previous laboratory, this part of the lab is open-ended and you are free to explore any options that you feel are the most promising, including methods not covered in the lectures.