NTNU
Norwegian University of
Science and Technology

Optimising KBE Generated Mechanisms

Arnt Underhaug Lima

Master of Science in Engineering and ICT

Submission date:	June 2017
Supervisor:	Bjørn Haugen, MTP
Co-supervisor:	Ole Ivar Sivertsen, MTP
	Ivar Marthinusen, MTP

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

ABSTRACT

Mathematical optimisation provides tools for creating designs that provably performs well. When such tools are combined with a knowledge based engineering (KBE) application for mechanism design, the process of optimising the designs of mechanisms can be fully automated. A prototype system, that assembles existing optimisation tools, a KBE system, and a tool for analysing mechanisms, has been created as part of this research. This prototype is evaluated by performing case studies. The results demonstrate that optimisation can be conducted, and indicates that the prototype can be used in many design problems. From the results, a technique utilising surrogate models is suggested as the preferred optimisation technique. This technique is suggested as it can accommodate an interactive and iterative process for formulating optimisation problems. The results also exemplify the need for a bug-free KBE system, when such a system is used in optimisation. In the aggregate, the prototype demonstrates that mechanism design problems can be formulated as optimisation problems and that the resulting optimisation problems can be reliably solved.

SAMMENDRAG

Matematisk optimalisering gir verktøy for å lage design som beviselig har god ytelse. Når slike verktøy kombineres med en KBE-applikasjon for mekanismedesign, kan prosessen med å optimere mekanismens konstruksjon automatiseres. Et prototypesystem, som samler eksisterende optimaliseringsverktøy, et KBE-system og et verktøy for å analysere mekanismer, er laget som en del av denne oppgaven. Denne prototypen er evaluert ved å utføre casestudier. Resultatene viser at optimalisering kan utføres, og indikerer at prototypen kan brukes i mange designproblemer. Fra resultatene foreslås en teknikk som benytter *surrogate models* som den foretrukne optimaliseringsteknikken. Denne teknikken foreslås, da den kan imøtekomme en interaktiv og iterativ prosess for å formulere optimaliseringsproblemer. Resultatene viser også behovet for et KBE-system uten kritiske feil, når et slikt system brukes i optimering. Samlet sett demonstrerer prototypen at mekanismedesignproblemer kan formuleres som optimaliseringsproblemer, og at de resulterende optimeringsproblemene kan løses pålitelig.

PREFACE

This is a master's thesis combining the fields of mathematical optimisation, knowledge based engineering, mechanical simulation, and software development.

The thesis is submitted to the Department of Mechanical and Industrial Engineering at the Norwegian University of Science and Technology in Trondheim. It is a continuation of my final year project. Also, it leverages the works of former students, including Eivind Kristoffersen, Anders Kristiansen, and Rasmus Korvald Skaare.

I would like to thank Bjørn Haugen who accepted the formal responsibility of supervising my thesis, despite being on sabbatical. Further, I would like to extend my sincere gratitude to Ole Ivar Sivertsen and Ivar Marthinusen. Your insights, feedback and ideas has been greatly appreciated. I am also grateful for the help I have received from Runar Heggelien Refsnæs of Fedem AS and Trapper Schuler of Technosoft Inc.

Trondheim, June 2017
Arnt Underhaug Lima

TABLE OF CONTENTS

LIST OF FIGURES

ACRONYMS

AML Adaptive Modelling Language. 3, 19–23, 26, 33, 60, 62, 65

API application programming interface. 19–22, 25–27

DoE design of experiments. 11, 23, 33, 38, 40, 42, 46, 52, 61, 65

FEDEM Finite Element Dynamics in Elastic Mechanisms. 2, 19–23, 26, 27, 33, 34, 46, 47, 51–54, 60, 61, 63

I/O input/output. 19

IPOPT Interior Point OPTimizer. 18, 24, 25, 40, 41

IP interior point. 10, 11, 18, 41

KBE knowledge based engineering. i, iii, v, 2, 3, 11, 15, 17, 29, 32, 33, 46, 52, 59, 60, 65, 79

KKT conditions Karush-Kuhn-Tucker conditions. 8, 34

MDAO multidisciplinary design analysis and optimisation. 18, 21, 23, 24, 33, 34, 61, 63

NFEM nonlinear finite element method. 11

REPL read eval print loop. 19, 20

SLSQP Sequential Least Squares Quadratic Programming. 18, 31, 40, 41

SM surrogate model. i, iii, 11, 22–24, 31, 33, 39–41, 51–54, 61–63, 65

SNOPT Sparse Nonlinear OPTimizer. 18, 40, 41

SQP sequential quadratic programming. 9, 10, 18, 34, 41

CHAPTER

ONE

INTRODUCTION

In a world with ever-increasing demands on productivity and cost effectiveness, the possibility of using optimisation in product design is an appealing prospect. Using optimisation provides a way of mathematically proving that a given design is the best possible. However there are, as is common when applying mathematics to the real world, some challenges.

When employing mathematical optimisation to engineering problems, there is a strong requirement on the accuracy of the model used in the optimisation. This requirement is because it is the model itself that is optimised, and any behavioural difference in the model and the real world can cause the optimisation process to diverge from reality. Even with an accurate model, there is a requirement for the process to capture the actual design goals; otherwise there is the possibility of maximising the wrong performance in the design.

Still, with a perfect model to optimise, there is some challenges left. These include the time to evaluate the model, which can be tremendous for engineering simulations. The optimisation systems themselves can also prove hard to select and configure.

Despite these challenges, there is a significant amount of tools available for using optimisation in engineering problems. While a wide selection of optimisation tools is a benefit, it does mean that there is a need to select the appropriate tool, before optimisation can be used in a new engineering domain [17, 30]. The process of setting up an optimisation system is also reasonably complicated.

It is these two tasks, both evaluating optimisation techniques, and connecting the engineering models to an optimisation system, which form the basis of this thesis. [1] The particular

[1]A note on some terminology used throughout this work: *Optimisation algorithm* refers to the numerical algorithm in charge of changing parameters to reduce an objective function. *Optimisation system* denotes the

domain is the field of mechanism design.

The effort in this thesis is supported by the use of a KBE environment for mechanism design. KBE can be thought of as a system for automating the engineering process. This is a requirement, as the model that used in the optimisation process should be able to run without human interaction. In other words, the design tool has to be fully automated.

KBE can also be thought of as a way of making the engineering process more human-friendly, as the menial tasks of engineering design is automated, the user is freed to do more creative and fulfilling tasks[29]. A similar humanisation of optimisation tools would allow design engineers to focus on the design, as opposed to the configuring of the optimisation process itself. Thus automating the configuration of optimisation jobs is also of interest.

The tasks outlined in the task description are in direct correspondence with the issues outlined here. In the abstract, the thesis looks at the optimisation of elastic linkages, specifically the effectiveness of various optimisation techniques and model formulations are investigated. In the concrete, the thesis looks at the development effort required to connect a specific KBE system to both a specific simulation tool and a selection of optimisation tools.

Even more concretely, the thesis looks at the specific challenges of using the KBE system, as developed by Kristiansen and Kristoffersen [15]. The model is generated by this system and evaluated by the use of the simulation tool FEDEM [4]. To a large extent both the KBE system and the simulation tool are treated as black boxes, and the details on the internals are available elsewhere [15, 4, 6, 5, 32]. That is not to say the connections to the optimisation system is not treated.

The connections to the optimisation system are core to the goals of this work, and they are thoroughly described. Closing the loop that connects the geometric model, the simulation software, and the optimisation algorithms that modifies the geometry, as seen in Figure 1.1, is the main goal of this work. Also there is a focus on investigating if such a connection allows for a generic system that is applicable to many design problems. [2]

whole computer system that optimises an engineering model. *Optimisation technique* refers to the algorithmic approach used in an optimisation system, but includes more than only the optimisation algorithm, such as the method used for obtaining the sensitivity of the design.

[2]This is in contrast to the use of FEDEM supported optimisation, in Kristiansen and Kristoffersen[15], which is done in a more limited fashion

The task list, from the task description, is restated below. Chapter 3 outlines the method used to answer these tasks, and also include the rationale for the reminding chapters.

Task List

1. Evaluate the possibility of optimising models defined in AML, by using state-of-the-art optimisation technologies available in Python
2. Evaluate the suitability of some optimisation techniques for use in mechanism optimisation. The techniques may include:
   (a) sensitivities from finite difference estimation
   (b) surrogate models
   (c) various optimisation algorithms
3. Specify the interaction between the optimisation tools and the user interface of the KBE tool, including:
   (a) selection of design variables
   (b) specification of constraints and objective function
4. As time allows, perform case studies

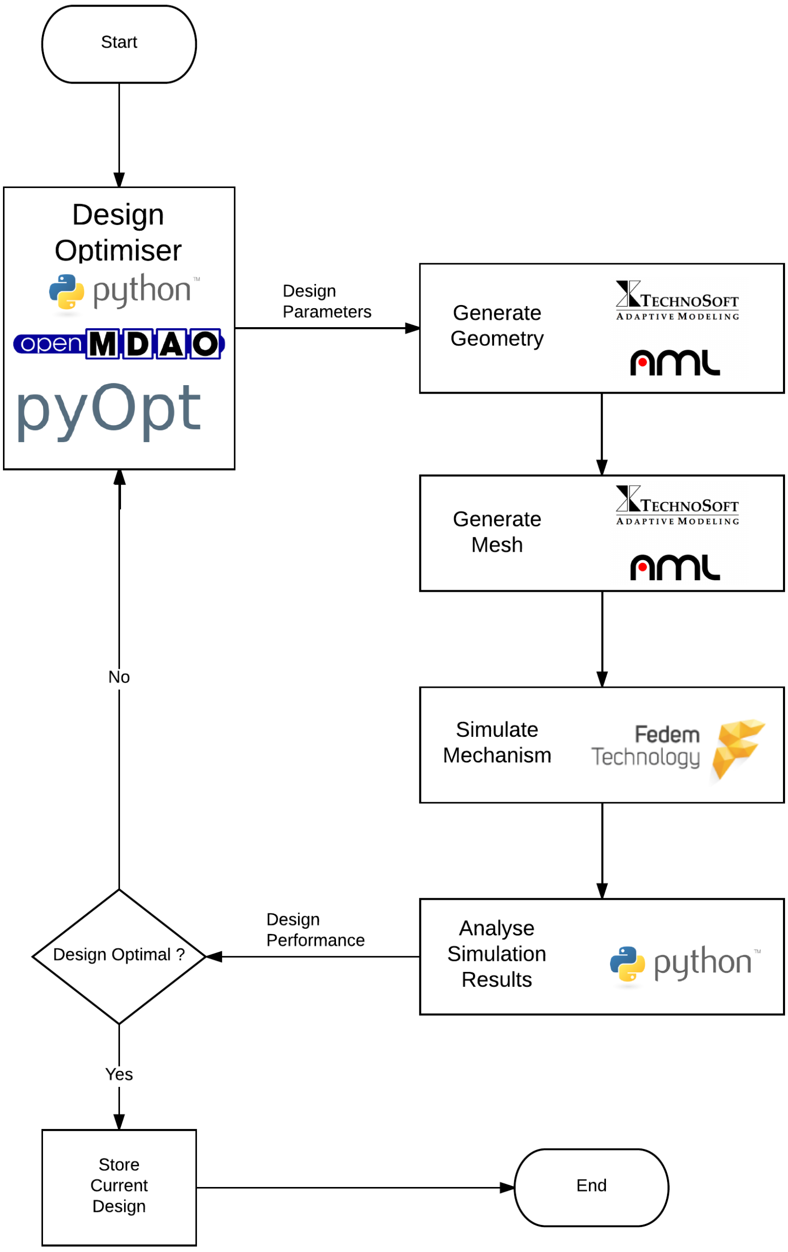


Figure 1.1: The stages of an iterative design optimisation system, with the software responsible on some of the stages represented by logos.

CHAPTER

TWO

THEORY

This chapter contains the theoretical underpinnings for the optimisation techniques that form the foundation of the system developed in this work. The vast field of mathematical optimisation is complex, and since the reader might be unfamiliar with the field, this chapter seeks to provide a basic understanding of the terms and mathematics that form the foundation of mathematical optimisation.

In addition to providing the basic theoretical background, this chapter also has a more in-depth treatment of some topics. The reason for this is that these topics support the analysis and evaluation of the optimisation techniques used later in the thesis.

2.1 Optimisation

2.1.1 Optimisation Fundamentals

Primarily this section defines the terms used to describe optimisation problems. These definitions are explained with the help of an example, in an effort to make the terms more tangible. The work here is based on introduction to optimisation in Sobieszczanski-Sobieski, Morris, and Tooren[33, p. 10-44] , and the remaining text of Section 2.1.1 is used verbatim from the author's project report [16] (spelling and formatting have changed slightly).

Consider a simple cantilever beam with a quadratic thin-walled cross section and a load applied at the free end. Such a system might be described in terms of a few parameters: t – the wall thickness, h – the height of the cross section, L – the length of the beam, and F – the magnitude of the applied load. Figure 2.1 depicts the beam and the relevant parameters.

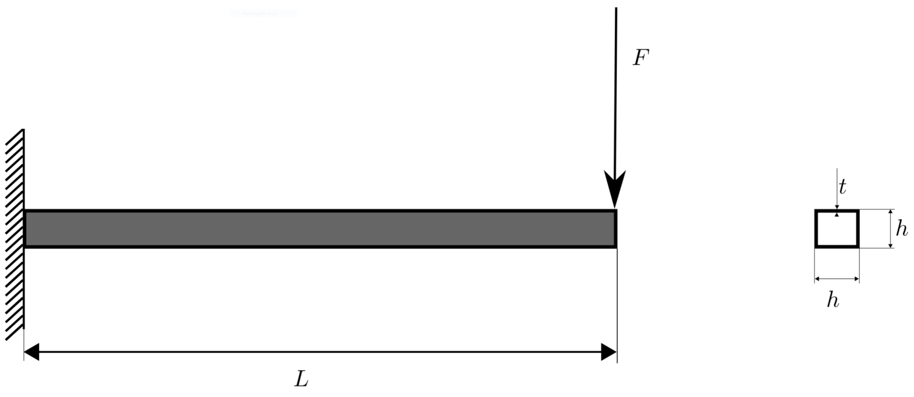


Figure 2.1: The Cantilever Beam and its Parameters

From this, it is possible to calculate various properties of the system such as σ_{max} – the maximum stress in the configuration, and V – the volume of the material used. Quantities such as these are termed *state variables* in the context of optimisation. [1] All the state variables are customarily combined in a vector denoted by $\boldsymbol{y}$.

Due to material cost considerations, it can be beneficial to *minimise* the material used in the construction. For this example, that means minimising $V = 4thL$. The function being minimised is called the *objective function*, and the symbol f is generally used in the mathematics [2].

A straight forward minimisation of the volume is of course not attractive. Firstly the stress would not be tolerable. In engineering problems such as these, it is common to check for the required condition $\sigma_{\text{max}} \leq \sigma_{\text{allow}}$, i.e. the structure's maximum stress is less than some allowed stress, typically the yield stress scaled by some safety factor. Secondly, there might be geometrical limitations, such as limitations on the cross-section parameters, i.e. $t^l \leq t \leq t^u$.

Both of the two previous limitations are called *constraints*. The former is an *inequality constraint*, and would normally be written $\sigma_{\text{max}} - \sigma_{\text{allow}} \leq 0$. It is possible to have any number of such constraints. In mathematical notation, the j-th inequality constraint is written $g_j \leq 0$. The latter limitation mentioned before is called a *move limit*. A move limit is a more direct constraint when compared with an inequality constraint. It provides direct lower and upper bounds on variables in the design. For a variable x the upper bound is customarily denoted x^u, the expression x^l denotes the lower bound.

Thus far it has not been specified what the parameters of the functions f, $\boldsymbol{y}$ and g_j are.

[1] The terms *behaviour variables* and *dependent variables* are also used.

[2] Any minimisation problem can be trivially recast as a maximisation problem, by multiplying the objective function with -1.

The variables that can be changed to modify a design is called *design variables*. All the design variables can be collected in a vector $\boldsymbol{x}$, and it is these variables that are changed to minimise the objective function. In the example, the vector $\boldsymbol{x} = [t, h]^\intercal$ is an appealing set of design variables. Similarly to the objective function, the constraint satisfaction is evaluated as a function of the design variables. Therefore $f(\boldsymbol{x})$ and $g_j(\boldsymbol{x})$ are both functions of $\boldsymbol{x}$. [3] Further the move limits for all design variables can be expressed as the double vector inequality $\boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u$.

Each design variable described as being either *discrete* or *continuous*. If all mathematical functions in an optimisation problem are differentiable, with respect to a design variable, the variable is defined to be a continuous design variable. Otherwise, it is defined to be discrete. In the running example, the variable h is continuous. If the beam is constructed out of sheet metal that only comes in a selection of gauges, the variable t is a discrete variable. However, if the selection of gauges is dense, t might be treated as continuous in the optimisation problem and, once the optimisation is done, the nearest gauge value selected. If this method is used, t is said to be a *quasidiscrete*.

The set of all the values of $\boldsymbol{x}$ that satisfies all the constraints are called the problem's *feasible region* or *domain*. In the domain, there is a value of $\boldsymbol{x}$ that minimises the objective function. In the example this is the value of $\boldsymbol{x} = [t, h]^\intercal$ which yield the least volume $V(\boldsymbol{x})$, while satisfying $g_1(\boldsymbol{x}) = \sigma_{\text{max}}(\boldsymbol{x}) - \sigma_{\text{allow}} \leq 0$ and the move limits. This value for $\boldsymbol{x}$ is called the *optimal solution*, and is denoted $\boldsymbol{x}_{\text{opt}}$. If only part of the domain is considered in the previous definition, the optimum is said to be a *local optimum*. The term *global optimum* is sometimes used to clarify if an optimal solution is not only a local optimum. As local optima are often all that can be found for engineering problems, the distinction is not necessarily useful.

Engineering problems typically have multiple design variables and constraints. Thus a common numbering system is useful. It can be defined as follows. There are n design variables x_i, $i = 1 \dots n$ and m inequality constraints g_j, $j = 1 \dots m$. Obviously there are also n move limits $x_i^l \leq x_i \leq x_i^u$. In addition there can be defined p *equality constraints* on the form $h_j(\boldsymbol{x}) = 0, j = 1 \dots p$.

The General Problem Summarised

Optimisation problems are instances of a general problem, which is summarised in Equation 2.1, for later reference. The notation is the same as used in the text of Section 2.1.1

[3] Although both f and g_j both are described as functions of $\boldsymbol{x}$ here, computational considerations will prove it useful to think of them as $f(\boldsymbol{y}(\boldsymbol{x}))$ and $g_j(\boldsymbol{y}(\boldsymbol{x}))$. If $\boldsymbol{y}(\boldsymbol{x})$ is the identity function, the two viewpoints are exactly the same, and in any case the mathematics can describe exactly the same problems.

$$\begin{array}{lll} \min\limits_{\boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u} & f(\boldsymbol{x}) & \\ \text{subject to} & g_j(\boldsymbol{x}) & \leq 0, \quad j = 1 \dots m \\ & h_j(\boldsymbol{x}) & = 0, \quad j = 1 \dots p. \\ \text{to obtain} & \boldsymbol{x}_{\text{opt}} & \end{array} \tag{2.1}$$

2.1.2 Karush–Kuhn–Tucker Conditions

In this section the Karush-Kuhn-Tucker conditions (KKT conditions), alternatively called the Kuhn–Tucker conditions are described. The description is based on Nocedal and Wright[20, p. 304-329]. Further, it should be noted that the entirety of the remaining text in Section 2.1.2 has been lifted verbatim from Lima [16], only some minor formatting related alterations has been performed.

The Kuhn–Tucker Conditions are a set of first order necessary conditions for when a point $\boldsymbol{x}$ is an optimising point. [4] These conditions, for the general problem given by Equation (2.1), are given in Equation (2.2). They transform the general problem in Equation (2.1) to a set of equations and therefore represent some form of simplification of the problem. They also provide a way of detecting an optimum and can thus provide a halting criterion for optimisation algorithms.

$$\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}) = 0 \tag{2.2a}$$

$$\lambda_j g_j(\boldsymbol{x}) = 0 \qquad \text{for all } j = 1 \dots m \tag{2.2b}$$

$$\lambda_j \geq 0 \qquad \text{for all } j = 1 \dots m \tag{2.2c}$$

$$g_j(\boldsymbol{x}) \leq 0 \qquad \text{for all } j = 1 \dots m \tag{2.2d}$$

A thorough treatment of the Kuhn–Tucker Conditions are considered out of scope for this text. That said a few brief, superficial observations on Equation (2.2) should provide the reader with an understanding of the nature of constrained optimisation. [5] First note that the conditions from Equation (2.2d) are inherited from the general problem. This part of Equation (2.2) is called the *primal feasibility*. The parts Equation (2.2b) and Equation (2.2c) are called *complementary slackness* and *dual feasibility* respectively.

Equation (2.2b) can be interpreted as a condition which states that either $g_j(\boldsymbol{x})$ or λ_j is 0. Further it can be noted that if $g_j(\boldsymbol{x}) < 0 \iff \lambda_j = 0$ then the constraint g_j has no effect in Equation (2.2a). Inequality constraints with a value less than 0, for a specified $\boldsymbol{x}$,

[4] A local optimising point is characterised by the fact that any movement of $\boldsymbol{x}$ will either increase the value of f or violate a constraint.

[5] Much of the mathematical rigour is sacrificed in this section, the reader is again referred to Nocedal and Wright[20, p. 304-329] for a formal proof, and Sobieszczanski-Sobieski, Morris, and Tooren [33, p. 27-46] for a more thorough informal explanation.

are called *passive* constraints. If the value is 0, the constraint limits the directions $\boldsymbol{x}$ can be moved, and the constraint is called *active*. The set of all active constraints is called the *active set*.

The passive constraints does not affect Equation (2.2a). The active constraints are equal to 0 and behave similarly to equality constraints. Because of this, it is interesting to ignore the passive constraints and treat the active set in Equation (2.2a) as part of the equality constraints. That is to take $m = 0$. The optimality conditions are in this case called the *Lagrange conditions* and Equation (2.2a) can be rewritten as:

$$\nabla f(\boldsymbol{x}) = -\sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}) \tag{2.3}$$

Moving along the vector $\nabla h_j(\boldsymbol{x})$ will violate the constraint, as it changes the value. Thus the linear combination on the RHS of Equation (2.3) can be considered the space of all moves that will violate a constraint $\boldsymbol{x}$. The LHS of Equation (2.3) denotes the direction of largest increase in f. Taken together this means that if Equation (2.3) can not be solved for the variables μ_j, there must exist a legal move, which will improve f. In other words, the reason for not following ∇f, the best possible move, can not fully be explained by the fact that it would be an illegal move.

2.1.3 Sequential Quadratic Programming

Sequential quadratic programming (SQP) is a class of optimisation algorithms that can be used to solve non linear optimisation problems. A thorough introduction to SQP can be found in Nocedal and Wright[20, p. 529-561]. This section contains a summary of the aforementioned work, with a perspective aimed at supporting the discussions and comparisons undertaken later in the thesis.

SQP starts off by transforming the general problem in equation (2.1). Conceptually this transformation can be thought of as a quadratic approximation of the objective, with a linearisation of the constraints. Equation (2.4) defines this transforamtion.

$$\begin{aligned} \min_{p} \quad & f_k + \nabla f_k^{\mathsf{T}} p + \frac{1}{2} p^{\mathsf{T}} \nabla_{xx}^{\mathsf{T}} \mathcal{L}_k p & & \\ \text{subject to} \quad & \nabla c_i(x_k)^{\mathsf{T}} p + c_i(x_k) & = 0 \quad & i \in \mathcal{E} \\ & \nabla c_i(x_k)^{\mathsf{T}} p + c_i(x_k) & \geq 0 \quad & i \in \mathcal{I} \end{aligned} \tag{2.4}$$

In this equation c denotes a vector of both equality and inequality constraints, with the indexes for the equality constrains in the set $\mathcal{E}$ and the the indexes for the inequality constraints in the set $\mathcal{I}$. $\mathcal{L}_k$ indicates the Lagrangian of the problem, and is defined by $\mathcal{L}(x, \lambda) = f(x) - \lambda^{\mathsf{T}} c(x)$, which is equation (2.2a) in a different notation.

Equation (2.4) approximates the original problem, centred around the k-th iterate, as indicated by the various subscripts. This means that, for each iteration, the algorithm creates a quadratic approximation of the problem, solves the approximation, and does the update $x_{k+1} := x_k + p$. It can be proven that such iterations will solve the original problem, and that the simplified problem can be reliably solved [20, p. 530-535].

Nocedal and Wright notes that the algorithms used for solving the approximation take advantage of the fact that the inequality constraints can be active or passive, as discussed in Section 2.1.2. With this notion, the inequality disappears from equation (2.4), and the simplification is generally successfully solved. Further the convergence properties are considered surprisingly good [20, p. 533].

Readers familiar with iterative numerical methods, for instance from finite element methods, might notice that the iterations on the approximation of the objective are closely related to Newton's method. In fact Nocedal notes that SQP "will act like a Newton method for equality constrained optimisation" ([20, p. 533]). The term $\nabla^{\intercal}_{xx}\mathcal{L}_k$ corresponds to the Hessian used in Newton methods, and Quasi Newton methods, e.g. the BFGS approximation, can be employed in SQP methods [20, p. 536-540]. This means no evaluation of the problems Hessian is required, which is beneficial as even an evaluation of the gradient can be expensive, especially if a finite difference approximation is used.

2.1.4 Interior Point Optimisation

Interior point (IP) methods for optimisation is the main alternative to SQP, for nonlinear optimisation [20, p. 563] . This section contains a description of the interior point methods, and as with Section 2.1.3, the perspective is aimed at supporting the upcoming discussions and evaluations. The section is built on the description of the interior point method in Nocedal and Wright [20, p. 563-593], and the reader is referred to this chapter for a more in-depth description.

Interior point methods starts by introducing an additional parameter, μ, in the problem defined by equation (2.1). Additionally, the inequality constraints are rewritten as equality constraints, by introducing the additional parameters s_i [6].

$$\begin{aligned} \min_{x,s} \quad & f(x) + \mu \sum_{i=0}^{m} \log s_i \\ \text{subject to} \quad & g_i(x) - s_i = 0, \quad i = 1 \dots m \\ & h_i(x) = 0, \quad i = 1 \dots p \end{aligned} \tag{2.5}$$

Equation (2.5) defines this transformation of the problem mathematically. In this transformation on can think of the parameter s_i as the distance to the border the inequality constraints enforce on the problem. The signs of the s_i parameters are enforced by the term $\log s_i$, which blows up as x approaches the infeasible region.

During the iterations of an interior point method, the parameter μ goes from a comparatively large value towards 0, as equation (2.5) is solved multiple times. This means that, at

[6]These parameters are called slack variables

the start, the term $\mu \sum_{i=0}^{m} \log s_i$ forces the point to be far from any inequality constraint. As μ decreases, the minimisation starts to trade reduction in the objective for closeness to the infeasible region, while the logarithmic term continues to act as a barrier, should the solution come to near the border [7].

The path traced out by the values of x as the value of μ decreases, will always be feasible, as long as the previous value of x was feasible, i.e. they remain interior to the feasible region [20, p. 566]. For engineering problems, this means that is if the initial design is feasible, all successive iterates of the design, will also be feasible. This can be beneficial, as discussed in Chapter 5 and Chapter 6. Moreover, the interior point methods can be constructed such that they converge even if the initial design is infeasible.

Readers familiar with nonlinear finite element method (NFEM) might also note that the introduction of the parameter μ bears some similarity to the introduction of pseudo-time, and the control parameter λ, in nonlinear statics. The pseudo-time is used because it is hard to define how the system responds to a complex load case. Instead, the load is added incrementally, as a small addition in the loads is comparatively easy to handle [7]. Informally one might think of the issues with the complex load case to be caused by the fact that the algorithm has too much freedom, i.e. there are too many solutions. Similarly to how the effect of the loads is incrementally introduced to ease the job of the NFEM solver, the incremental introduction of the freedom associated with the constraints ease the job of an interior point optimisation algorithm.

2.2 Design of Experiments and Surrogate Models

This section presents an interesting optimisation technique, which can be selected independently of the optimisation algorithm that is used to solve a problem. The technique can be separated into two stages, the first one being *design of experiments (*DoE*)* followed by the generation of *surrogate model (*SM*)*. Both of these stages are briefly described, with the goal of enabling the reader to understand the use of these techniques.

Design of experiments refer to the process of selecting the values of the variables in an experiment, with the goal of obtaining as much information as possible. In the context of the model used in an optimisation system, this would be the process of selecting points in the design space that give as much information as possible about the model. A procedure that is demonstrated to provide a set of samples that has multiple favourable properties is the Orthogonal-Maximin Latin Hypercube Designs [13]. DoE techniques have also been shown to be useful in optimisation systems utilising KBE [1].

After a set of function evaluations has been obtained, curve fitting methods can be used to get so-called surrogate models [8] [33, p. 298-300]. The curve fitting method can be as simple as an approximation of a (multi-dimensional) Taylor series. In general, the surface

[7] For this reason, interior point methods are also called barrier methods [20, p. 565-566].

[8] The terms response surface and meta model are also used as synonyms by some authors; other authors haves subtle differences in the definition of these terms.

is obtained by manipulating the constants of an analytic expression. For an optimisation problem the obvious functions to approximate are the objective and the constraints. This has several benefits. Amongst them is the fact that the approximations can be many orders of magnitude faster to evaluate. Further, analytic sensitivities are typically also available. The consequence of this is that when an optimisation algorithm is used on the approximations of the objective and constraints, as opposed to being used directly on the model, the algorithm can use function values and sensitivities without doing costly computations.

CHAPTER

THREE

METHOD

3.1 Rationale

In the introduction the challenges of using optimisation in a new engineering discipline were discussed. The challenges described included the issue of selecting optimisation algorithms, and strategies; and the challenge of actually employing them on a design problem. Both the investigation into the severity of the challenges, and knowledge about solutions that can be applied are investigated. In this chapter, the strategy for this investigation is outlined.

The view that primarily informs the method selected is that empirical results, and practical know-how, are better ways to investigate the problem at hand, as opposed to using a mathematical description to a priori determining the best optimisation strategy. With such a prototyping approach, it is possible to validate the system as a whole [34, p. 111], and thus prove that an actual working system can be implemented. The mathematics defining optimisation can describe parts of a potential optimisation system, but a running prototype demonstrates that the system can be implemented, that all the computational requirements can be met.

That is not to say, that the mathematical description is not useful. A mathematical viewpoint will be used to analyse and inform the selection of optimisation tools. However, there exist several arguments against using the mathematical descriptions, without an actual working implementation.

Firstly, there is the challenge of verifying the mathematical behaviour of the function being optimised, i.e. the behaviour of the mechanism model. This behaviour is fundamental to the mathematical analysis of the optimisation algorithms. The actual behaviour is somewhat obscured by the layers of engineering software that is used in the model, and thus it is hard to know what assumptions can be tolerated directly.

Secondly, there is the challenge of translating a mathematical description into a running piece of software. Even though the mathematical description of all the tools use linear algebra, and are meant to be implemented as software, there are potential numerical challenges, e.g. the scaling of functions are known to be problematic [20, p. 26-27].

Taken together the method used in this work is to identify challenges, and prove feasibility, by implementing actual working solutions. The approach is twofold, starting with case studies, in which results from optimising simple designs are collected. Meanwhile, a system for running the cases is developed.

The case studies inform which optimisation strategies work on the mechanism model. Information about convergence, execution time, design improvement, and more can be used both to demonstrate the feasibility of optimising mechanisms and to determine which optimisation strategies have the best performance. Should a given strategy fail, the aforementioned quantitative results can be used, together with the theory, to offer an explanation as to why the failure occurred. Similarly, the quantitative results, together with the practical experience of working with the system, can be used to give a holistic evaluation of which optimisation strategy is best suited to optimise the model of the mechanism.

Development of a system that can be used to optimise mechanisms is necessary to perform the mentioned case studies, as the various components have to be connected somehow. In addition to providing the connections between various components, the optimisation system can aid the evaluation of cases, since a well-structured system allows code reuse between the different cases. Further, the amount of customisation, that is required for the system to handle a new case, is an indication of how much configuration would be needed in a future version of the system, that is meant to be used by design engineer. [1]

[1] Only a *prototype* system is developed, the state of the code reflects this purpose, meaning the code currently is in need of refactoring. Also, the code has only a limited separation of results, system, and cases; this again reflects the prototyping approach. For these reasons, as well as the extent of the code base, the code defining the system is largely excluded from this document.

3.2 Summary

By developing a prototype optimisation system and performing case studies with the help of this prototype, the possibility of optimising mechanisms modelled in a KBE application is evaluated. The cases demonstrate the possibility of solving the particular problem that constitutes the case, but they are also selected with the intent of showing specific aspects of the system, e.g. the importance of selecting the right optimisation algorithm. Furthermore, the cases collectively, together with the experiences associated with the development of the optimisation system, inform the discussion of the applicability of using optimisation to solve new design problems.

CHAPTER

FOUR

SYSTEM OVERVIEW

4.1 Purpose

This chapter describes some of the details of the optimisation system, which form the infrastructure required to prototype the use of optimisation in mechanism design. Primarily it serves as a description of the system that is used to execute the cases in Chapter 5, by outlining the components (Section 4.3), and their interactions (Section 4.5). These outlines, together with the preparations required to define a case (Section 4.4), form the foundations of the system used to evaluate the cases. Thus this chapter characterises the context in which the cases are executed.

In addition to support the understanding of the cases, this chapter also describes the experiences gained by developing the system. As outlined in the methodology, these experiences serve as the basis for discussing the prospect of using optimisation, together with KBE systems, in more general terms. While the whole chapter supports this general discussion, the sections on challenges (Section 4.6), architecture (Section 4.7), and usability (Section 4.8) are included specifically to support the discussion of how optimisation can be used to solve new design problems.

4.2 Third Party Components

In this section, the variety of tools that are used to build the system is described. There is little context as to what the specific tools are used for in this section; the primary purpose is to provide the reader with an understanding of what a tool can do. All the tools are directly utilised by the system, and the specific usage of each tool are given later.

4.2.1 Scientific Tools for Python

These tools include NumPy [21] and SciPy [12, 22] which provide linear algebra tools and science related methods, respectively. [1] In addition there is Matplotlib [18, 10], Seaborn [37, 38] and Bokeh [39] which provide plotting capabilities. Bokeh in particularly useful, as it provides convenient tools for creating interactive plots which can be embedded in Jupyter notebooks. Jupyter notebooks are interactive documents which allow the user to mix code, notes, LaTeX-style mathematical expressions and plots [27, 25].

The final component that is classified as a scientific tool for Python is the Pandas package. This package is also built atop NumPy and introduces a convenient way of working with tabular data. The benefits include e.g. the possibility of defining a table that contains a time series of x and y coordinates. In this table, it is possible to specify the index to be points in time, as opposed to integer indexes, which are the normal indexes for arrays and matrices. With such an index it is possible to query the table for all points in a given time range, or to sum two tables together, without the risk of summing values that did not occur at the same time.

4.2.2 Optimisation Related Tools

Development of the optimisation system is leverages a framework called OpenMDAO [23, 11]. This framework has model decomposition capabilities, interfaces to a variety of optimisation algorithms, includes multiple surrogate model algorithms and incorporate multiple design of experiment strategies. The benefits of using this framework include a modular approach both for the optimisation strategy and modelling of the problem, e.g. changing the optimisation algorithm is a simple one-line change. The model decomposition also makes it possible to define the flow of parameters across model components, without having to specify the execution order.

One of the main ways OpenMDAO interfaces with optimisation algorithms is through the PyOptSparse [14] package (which is a reimplementation of PyOpt [26]). Through this package OpenMDAO can use the optimisation algorithms SLSQP, SNOPT [9], IPOPT [36]. The former two of these are SQP algorithms (see Section 2.1.3) the latter is an IP algorithm (see Section 2.1.4). SNOPT is a commercially available algorithm, whereas the other two are released under open-source licences.

The reason for selecting these algorithms are more thoroughly described in Lima [16], but the main reasons are as follows. IPOPT and SNOPT has been claimed to be the state-of-the-art algorithms for optimising nonlinear problems [20, p. 592]. The same algorithms have been shown to have superior performance for engineering problems, especially over genetic algorithms[30, 17], for this reason, any use of genetic algorithms has not been considered in this work. Finally they provide, as mentioned, samples of both SQP and IP methods, which means the two algorithms are somewhat different. SLSQP was included as the SciPy package includes this algorithm; thus it is easy to obtain a working and tested copy of the algorithm.

[1] Stricktly speaking SciPy is built atop NumPy

4.2.3 Tools for Connecting Components

For communication the system utilises a COM-API that FEDEM exposes to the operating system. This communication is made possible by the PyWin32 package [28]. The package Rpyc [31] is used for calling Python functions across machines, with the aid of a network connection between them.

The Docker toolchain [2] allows developers to define a Linux environment with the help of a text file, in which the commands for e.g. installing software and setting environment variables are defined. Further Docker includes tools for starting a virtual machine running Linux and loading configuration files. [2]

4.3 System Components

4.3.1 Components as Abstractions

When software systems are developed, the process of breaking the system up into layers or components form one of the primary tools in software engineering [34, p. 157-158]. The components described in this section constitutes such a foundational layer for the optimisation system. Thus any understanding of the system as a whole starts off by understanding what the foundational components bring to the uppermost layer, which is the purpose of the following sections.

4.3.2 Aml Connection

Currently AML does not support batch-mode execution. This is a requirement for using applications written in AML as part of a larger optimisation system. Since AML comes with a REPL, it is possible to make a connection to a running AML program with the help of I/O redirecting, as is commonly done on Unix platforms. A proof-of-concept implementation of this was developed as part of the author's final year project [16], and the implementation was further developed as part of the work associated with this thesis.

Loosely speaking the I/O redirection works by having a Python program type into the AML development terminal, and then reading the output, as pure text. Among the main complications, with the implementation, was the fact that the AML REPL returns immediately after a command has been entered. This is in contrast to waiting until the entered command has finished execution. The consequence of this is that it is impossible for Python to know when the execution of a command is finished. This problem was solved by always printing a specific string after each command, and then waiting for this string to appear in the output.

[2]Docker technically defines Linux containers, but this has no consequence for the optimisation system.

In addition to providinga simple connection between programs written in AML and Python, the AML connection offers an abstraction layer which means it is possible to call AML functions in a fashion that looks native to Python. E.g. the AML `create-model` command is executed from Python as `aml.run(CreateModel( class_name='main-mechanism-class'))`.

It is also possible to execute native AML code in a more direct fashion. As demonstrated by the following example:

```
aml.run(BlockingCommand(
     '''(loop for link in !final-link-ref-list do
          (change-value (the analysis export-surface?
               (:from link)) nil)
          )
          (write-nastran-bdf-files (the))'''))
```

4.3.3 Fedem Wrapper

As previously mentioned the optimisation system leverages the FEDEM COM-API. This API is rather closely related to the FEDEM user interface; and can be used for specifying a simulation, starting it, and extracting the results. To have an API that is better suited for use in the optimisation system, a wrapper is created.

A more convenient API makes the evaluation of cases easier. E.g. the simulation results are returned, from the API wrapper, as Pandas data-frames, leveraging the capability of time-based indexes. Further, the wrapped API can reduce the number of arguments required for the various calls to FEDEM.

The wrapper does not only provide a more convenient API, it also adds functionality. When a simulation is started with the COM-API, the function immediately returns (this is similar to the issue with the AML REPL immediately returning). To detect when a simulation is complete, the wrapper monitors the timestamp of the last completed time step. When the last result is available, the wrapper returns the results to the optimisation system.

This added capability is further extended by checking for convergence issues, which is not reported in the COM-API. The detection of these issues is again done by monitoring the time stamp of the last available time step. If this timestamp is not increasing, the simulation is assumed to have failed, and the wrapper throws an exception.

4.3.4 Object Hierarchy Factory

As described in 4.4.1, the optimisation system takes Yaml descriptions of the design space as input. The purpose of the object hierarchy factory is to create set of objects that represent the model of the mechanism. This is done with the model decomposition capabilities in OpenMDAO, and the decomposition itself is relatively close to the object hierarchy used in the AML application. [3]

The benefits of doing this decomposition are mainly for simplifying later work with the model. This includes good support for scaling parameters and outputs. Furthermore, the decomposition capabilities also enable simple ways of changing which variables are design variables.

Another benefit of doing the decomposition step is the possibility of visualising the model. This visualisation is useful for debugging. In addition, it serves as a description of the design and shows how the various parts of the design influence one another.

4.3.5 Aml Input Writer

The AML program takes input as a set of text files. These files are generated by a component of the optimisation system. For the generation, a template is filled out, and the files are written to the file system. The input files are then loaded by calling the appropriate functions with the AML connection component.

4.3.6 Fedem File Editor

Some functionality of the FEDEM COM-API is either not implemented or inordinately complicated to use. For these situations, there is a component that reads in a FEDEM input file, and inserts lines at the appropriate locations.

4.4 Case Configuring

In this section the various tasks required as preparations for running a case are summarised. Only preparatory tasks are described, the main execution flow is found in Section 4.5. Most of the implications of the preparations, as relating to the end user, are discussed in Section 4.8 and Section 6.3.

4.4.1 Defining the Design Space

The design space is defined in Yaml [3], a simple serialisation format. In this file the topology, cross sections, joint positions, etc. are defined. Thus this Yaml file is analogous to the set of text files that constitutes the input files for the AML program.

However, there is a difference in the level of abstractions in the two input types. The Yaml file defines the design space, as opposed to a single point in the design space. Concretely

[3]This technique is closely related to the factory method pattern [8, p. 107-116]

this means the Yaml file has built in support for expressing ranges, and default values, for the numerical values that describe a mechanism.

4.4.2 Defining the Analysis

The analysis part of the case preparation involves defining a method on a class. This method is responsible for taking the results from FEDEM and coming up with a scalar value for the optimisation objective. Any constraints must also be calculated.

Since the wrapped FEDEM API returns Pandas data frames, and these objects are designed to do calculations on time series, the API for performing this analysis, is quite helpful.

4.5 Execution Flow

With all the components of the system, as described in Section 4.3 and the preparations as described in Section 4.4, the execution of a case is quite straight forward. For a case that does direct optimisation, without the use of surrogate models, the flow might be described by the following steps:

1. Load the design space description and construct the object hierarchy
2. Create an instance of the class responsible for analysing the results and connect it to the object hierarchy
3. Connect the optimisation algorithm to the object hierarchy, including automatic connection of design variables and specification of the move limits
4. Specify the name of the objective and constraints in the object responsible for the analysis
5. Optionally specify a recorder, which is responsible for storing the results and parameters from each iteration of the optimisation algorithm
6. Run the optimisation algorithm, for each iteration the following happens:
   (a) Construct AML input by evaluating the object hierarchy
   (b) Open the AML input, using the AML connection
   (c) Cause the FEDEM input to be generated, using the AML connection
   (d) Perform changes to the FEDEM input, if required
   (e) Open FEDEM input
   (f) Using the FEDEM API, further specify the simulation to be performed, e.g. specify the simulation duration, using the wrapped FEDEM API
   (g) Start the simulation, using the wrapped FEDEM API

(h) Analyse the results from FEDEM, to obtain values for the objective, and any constraints

If a design of experiments algorithm and surrogate models are used instead of the direct optimisation, as described above, the flow for the design-of-experiments phase is very similar, due to the flexibility of OpenMDAO. A design of experiments method is specified instead of an optimisation algorithm, but otherwise, the flow is exactly the same. In each iteration, normally performed by the optimisation algorithm, corresponds to the executing of an experiment. The results are still stored by the recorder and can be used later, to construct surrogate models.

4.6 Challenges and Their Consequences

As alluded to in the introduction of this chapter, this project has seen a set of challenges, that to a degree has limited the progress. These challenges are described in this section and are all related software provided by various third parties. Thus including descriptions of the challenges does not only offer explanations for the limitations on the progress of the project, but they are also important observations on assembling the specific tools into an optimisation system. Moreover, these experiences are also of general interest, as any new optimisation system can encounter similar issues during development.

4.6.1 Aml Error Messages

As previously mentioned AML does not have support for batch mode execution, and this capability has been added as part of the work associated with this thesis. Since AML assumes there is a user present, there is at times message boxes that pop up, and requires user interaction before the execution continues.

The most obvious example of such a message box is a box with licensing information that appears when AML is started. This issue is resolved by starting AML once, and using the same process for all iterations of the optimisation algorithm. Reusing the process is far from ideal, and it would be advantageous to start AML anew for each iteration, since this guarantees that no state is preserved between iterations. The current solution does, for example, require manual deletion of the geometry, to avoid a situation where the computer to runs out of memory.

Another message box that appears is if the meshing fails. This means that, for the majority of the project, constant monitoring of the computer is required as a case is executing. The reason for this is that the message box requires a mouse click before the execution can continue. At a late stage in the project it was discovered that the mesh failure was due to a bug relating to the blending of edges, in the AML application, and when this feature was turned off, pretty much all mesh failures disappeared.

4.6.2 Multivariable Surrogate Models

The surrogate models was created by leveraging the capabilities of OpenMDAO. When a surrogate model is defined in OpenMDAO using multiple scalars as parameters, or when the model is defined using a single vector as a parameter, the surrogate model seems to be working as expected.

However, when multiple vectors are used as parameters, the surrogate models does not appear to approximate the sample data used for training the model. Inspections of the OpenMDAO source code, during execution, with the help of a debugger, suggests that only the first vector is used as a parameter. Further, the internal state of OpenMDAO seems to indicate that there is a bug in the code that calculates the dimensionality of the inputs. This issue was circumvented by simply concatenating the vectors into a single large vector parameter. While multiple vectors are a suitable way to model e.g. a set of points, this issue, once known, is not that serious.

4.6.3 PyOptSparse Compilation

PyOptSparse is a library that allows for many optimisation algorithms to be used from Python. The installation process includes compiling these algorithms and also compiling the appropriate Python bindings. Since the optimisation algorithms are written in Fortran and C++, and Python is written in C, a wide selection of compilers are required. The installation instructions emphasise that it is important that there is a correspondence between the compiler used to create Python, and the compilers used when installing PyOptSparse. The installation instructions also mention that installation on 64-bit Windows is untested.

After several attempts at installing this package on the 64-bit Windows operating system that has the other software installed, the decision to do the installation on a virtual machine, running Linux, was made. The installation instructions were found to be sufficiently detailed so that the installation was easily completed.

Since the project utilises Windows only software, and PyOptSparse was only successfully installed on Linux, a communication channel is required between the two operating systems. This was achieved with the help of the Rpyc, which was used to execute the analysis of the model on Windows while running the main optimisation algorithm on the Linux virtual machine.

4.6.4 IPOPT and Python3

While the installation of PyOptSparse compiled the IPOPT algorithm, any attempt at using this algorithm caused PyOptSparse to throw a error. The only information in this exception was that something was wrong with IPOPT.

By editing the Python code of the PyOptSparse project, it was possible to check that IPOPT itself had not experienced any errors. Further editing, this time of the C code in the Py-

OptSparse project, involved ensuring that every error message was printed, as opposed to catching it and displaying a more helpful error message.

The issue turned out to be the Python bindings for IPOPT, which used some functions in the Python/C API. These functions were only available in Python2, and by changing the C code, the issue went away. The changes needed were conveniently available as part of the pyIPOPT project [40].

4.7 Runtime Overview

Due to the workarounds introduced for the reasons described in Section 4.6, especially with the workaround associated with Section 4.6.3, an overview of the logical structuring of the system is included here. In Figure 4.1 all the actual running parts are depicted. The figure emphasise the system's contact with the 3rd party components, this is in contrast to the internal architecture, which is described in sections 4.3 and 4.5.

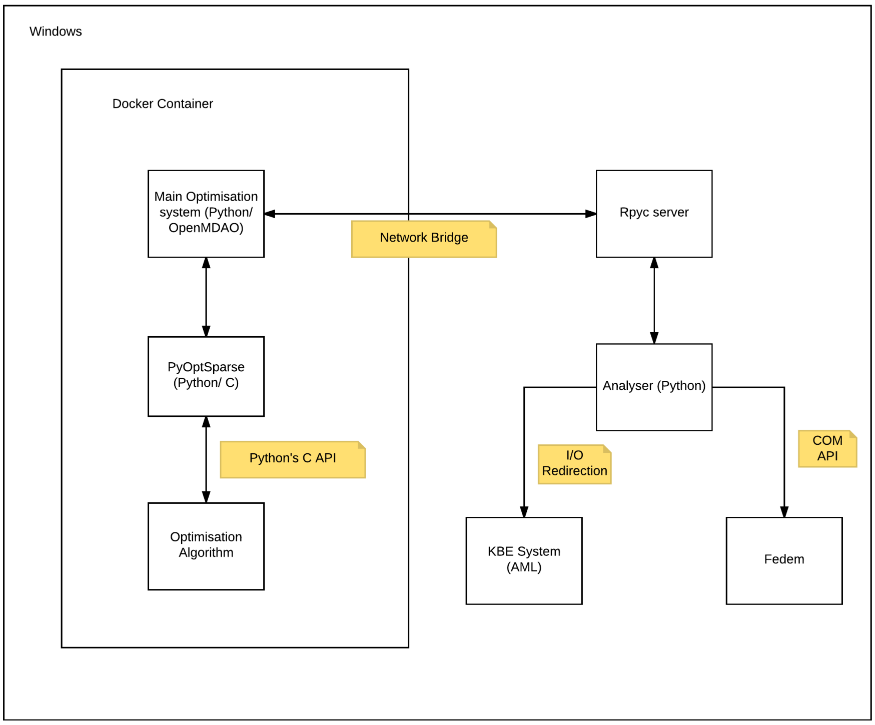


Figure 4.1: Here the various components of the system can be seen. Notes describe the communication type for the more unusual communication types. The parts that are vitalised using Docker are also shown. If a programming language or framework were actively used in a component, the relevant language or framework is shown in parenthesis.

4.8 User Interface Considerations

As discussed in the introduction and method, the user interface of the optimisation system is not the prime concern of this thesis. It is considered more important to have actual evidence of the feasibility and effectiveness of optimising models of mechanisms in the AML program. Still, as mentioned in the introduction, the usability of the optimisation system is essential if the system is to see actual usage. Thus the purpose of this section is to have a look at how a user interface can be included in the optimisation system, at a later stage.

Section 4.4 outlined the preparations that are currently needed before a case can be started. These preparations are the definition of the design space and preparation of the analysis. To make the system usable, without writing code, these two activities have to be wrapped in use interfaces.

Currently, the system uses a Yaml file to define the design space. Thus a user interface capable of defining such files is all that is required, for this activity to become user-friendly. Given that the structure of the Yaml file is closely related to the way a mechanism is defined in the AML program, the creation of such a graphical user interface is not expected to be insurmountable. Exporting a Yaml file from the user interface is also perceived to be a simple task, as this serialisation format is commonly used in a variety of programming languages.

During the author's final year project [16], the beginnings of a user-friendly way of defining the analysis step were developed. This concept starts with making the observation that the responses that are obtained from FEDEM are functions of time. A possible representation of these functions is a time series of numerical values as provided, by the wrapped FEDEM API, in the form of a Pandas data frame. When the analysis is defined as a set of operations on the data frames, the code can be thought of as composing functions. FEDEM already includes a tool for composing functions, namely the control system editor in FEDEM. Thus it was postulated that graphically composing functions is a familiar way of working, for the intended end user of the optimisation system. See Figure 4.2 for an example of how the control system editor is used.

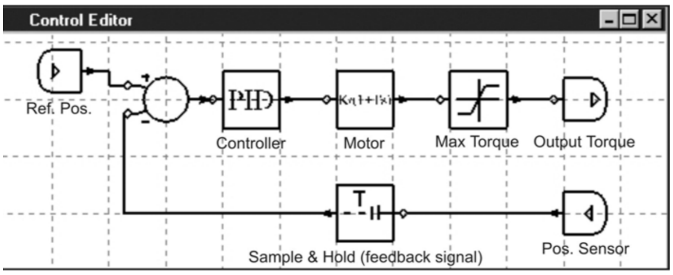


Figure 4.2: Combining and transforming signals in FEDEM's control system editor (Figure reprinted from [6, p. 183])

Example

To make the suggestion more concrete, consider a situation where the design requirement is for a node to be in given position at a given time. In the graphical syntax, this might look like Figure 4.3. The code, using the wrapped FEDEM API and Pandas, is shown below. Note that there is a substantial correspondence between each box in the figure and each line in the code.

In addition to the similarity of the code and Figure 4.3, there is also a strong connections with the example in Figure 4.3 and Figure 4.2. The syntax in Figure 4.3 is more verbose, as it displays all the options. Such options are a part of the control system in Figure 4.2, e.g. for the PID node must have some scaling factors. In Figure 4.3, these options are not hidden, mainly to provide clarity to the computation it describes.

Example Code

```
result = fedem.get_res_on_bid(triad01_bid,
                              'Position␣matrix:TMAT34:Position␣X',
                              'x')
result.columns = ['x']
not_of_interest = result.loc[lambda df: df.x < 0, :]
settle_time = not_of_interest.index[0]
obj = abs(settle_time -2.0)
unknowns['res'] = obj
```

4.9 Summary

In this chapter, the internals of the prototype optimisation system was described. Background information on the wide variety of third-party tools that is used by the system was provided. These tools are assembled into the foundational layer of the system's architecture. The foundational tools come together and enable an understanding of how a case is defined and how the execution of a case progresses. However no demonstration of the system was given, and all descriptions are abstract, with no concrete designs. Providing more concrete information is the purpose of Chapter 5, which, by evaluating cases, demonstrate the capabilities and effectiveness of the optimisation system. With more concrete cases the system can be characterised in a less abstract way, by actually showing the usage. The more concrete cases also enable a discussion of the overall architecture, including the complexities associated with the bugs described in this chapter.

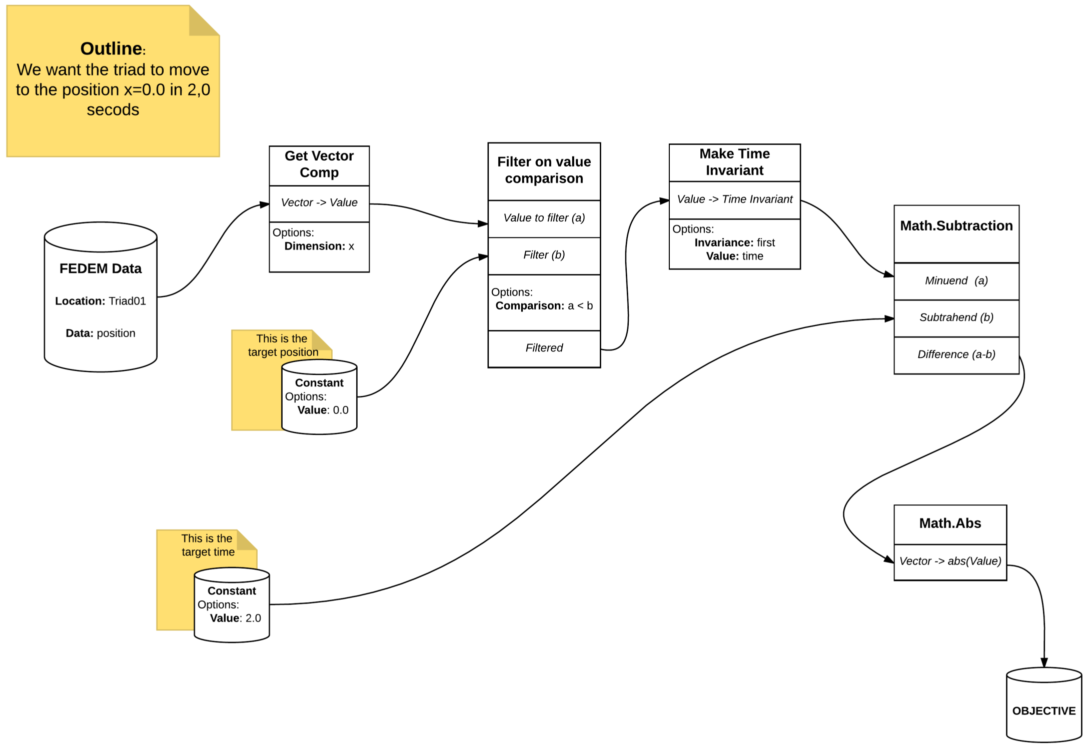


Figure 4.3: The visual syntax for the computation required in the example.

CHAPTER

FIVE

CASES

This chapter presents the cases that have been evaluated as part of the effort to assess the effectiveness of the system described in Chapter 4. Furthermore, the cases provide more insight into how various descriptions of mechanisms behave during optimisation processes. Thus the cases offer observations on the effectiveness of optimisation techniques, for optimisation of mechanisms, and observations on the feasibility of adding optimisation capabilities to a KBE system.

Three cases are presented in this chapter, but due to the iterative and exploratory nature of the work, the separation into three cases is somewhat arbitrary. Roughly speaking the cases can be understood as a demonstration of the system capabilities (Section 5.1), a comparison of optimisation algorithms (Section 5.2), and an exploration of the issues associated with a more complex case (Section 5.3).

5.1 Case 1: Capabilities Demonstration

In this case, a relatively simple mechanism, with relatively simple analysis and parameterisation, is optimised. Specifically, the location of a joint is varied, with the desired path of a point as the objective. The main goal, with this case, is to prove that the system functions as a whole, with all of the components, and all of the complexity these components bring.

5.1.1 Model Definition

Since this case uses the same design space, and post processing, multiple times, the definition of these are described in a separate section, and constitute the first part of the method used in the execution of this case. Because the text is meant to describe high-level concerns, the reader is also referred to Appendix A, which contains the source code level of details necessary for a complete understanding of the design space definition and design analysis.

Design Space Definition

The geometry used in this case is a four bar mechanism, which has the property that the tip of the coupler (Point 4 in Figure 5.1) travels in a nearly straight line, for an interval of the path it traces, as the input crank performs a revolution. This is a well-known configuration (see e.g. [32, p. 159-173]), subsequently the design position of the mechanism is the position that is known to give the straight line behaviour. The reference design is then modified by parametrising the position where the rocker is connected to ground, i.e. parametrising the position of Point 2 in Figure 5.1.

Design Point Analysis

For each point in the design space, a procedure for analysing the performance of the design is required, see Section 4.4.2. The objective is to find the mechanism that has the straight line behaviour. As the path traced by the tip of the coupler is the chief concern, the path of this point constitutes the primary input of the analysis. In addition to the path, the analysis requires two points that define a line segment which forms the reference path the mechanism should follow.

The first step, in the analysis, is to find the start and end time. These values define the time interval that should be used when comparing the mechanism path to the reference path. Since the inertia and stiffness of the mechanism are different for differing design points, the start and end time can not be hard-coded; they need to be detected. Hence the analysis detects when the mechanism is closest to the first point and selects this time as the start time. Similarly, the end time is defined by when the mechanism is closest to the second point.

The path of the coupler is defined with time as a parameter, whereas the reference path, and the goal of creating a straight line, are both time invariant. Consequently the coupler path should be made time invariant by using a new parameterisation, similarly to how the substitution $ds = v(t)dt$ is used to solve line integrals in calculus. For convenience the path is defined to be $r(t(\tau))$ with the properties $t(\tau = 0) = t_{\text{start}}$, $t(\tau = 1) = t_{\text{end}}$ and $dr/d\tau = Const.$

By creating a path for the reference line, also parameterised with τ, the distance between the paths is easily calculated as $D = r(\tau) - r_{\text{ref}}(\tau)$. This distance may be described as a function $D(\tau) : \mathbb{R} \rightarrow \mathbb{R}^3$, and by integrating the L_2 norm of this function, a measure of how much the coupler follows the reference path is obtained. For the actual implementation of this, all operations are written as vectorised expressions, and the integral $\int_0^1 |D(\tau)|^2 \, d\tau$ is solved numerically with the trapeze method. Further the property $dr/d\tau = Const.$ ensures that the formulation is not affected if e.g. the mechanism has a large distance to the reference path, for a short time at significant speed.

For the potential use as a constraint, a measure of how much of the path of the coupler is above the reference path is needed. In other words, the constraint is violated if the y component of $D(\tau)$ is positive, and a measure of the degree of violation is required (the y direction is directed up in Figure 5.1). The measure of the constraint is obtained with an integral similar to the previous integral, only a slight alteration of the function $D(\tau)$ is required. The aforementioned alteration is obtained by taking the value of the new function to be 0 if the y component of $D(\tau)$ is negative, elsewise the original value of $D(\tau)$ is used.

5.1.2 Case Executions

During the work associated with this case, various sub-cases has been executed. These include the following:

1. Optimisation using the model and finite difference sensitivities, with the constraint ignored
2. Optimisation using the model and finite difference sensitivities, with the constraint
3. Optimisation on a surrogate model, with the constraint

The SLSQP algorithm was used for all the executions mentioned above.

5.1.3 Results

Optimisation Without Constraint

The first execution, with no constraint, converged to a point a few millimetres from the known solution, with a slightly better performance. As the difference in both the design space point, and performance, were very small; the point that the algorithm converged to can be thought of as the same point as the known, working solution. Nevertheless, an actual improvement to the analytic design was found.

Figure 5.2 shows the designs that were evaluated by the optimisation algorithm, for the execution without a constraint. A circle indicates that the two values corresponding to this point were used to generate a design, in an iteration of the optimisation algorithm. Since the two design parameters happened to be the planar coordinates of a physical point, each circle has a physical interpretation as the location of the Point 2 in Figure 5.1, and the two figures uses the same coordinate system.

Scatter plots are used throughout this chapter. They are used to visualise the points evaluated by the optimisation system. Hence some observations on how these plots are used are in place. From the optimisation algorithm's perspective, every point in the design space is a dimensionless vector of real numbers. This vector might correspond to spatial coordinates, and thus it might have a physical interpretation, but this is not a necessity. The scatter plots are used to illustrate the behaviour of algorithms; consequently, the physical interpretation is of less importance e.g. the clustering in Figure 5.2 is more interesting than the coordinates of the convergence point. Since any proper naming of the axes would be rooted in the spatial interpretation, and the spatial interpretation is of lesser interest, all naming is omitted.

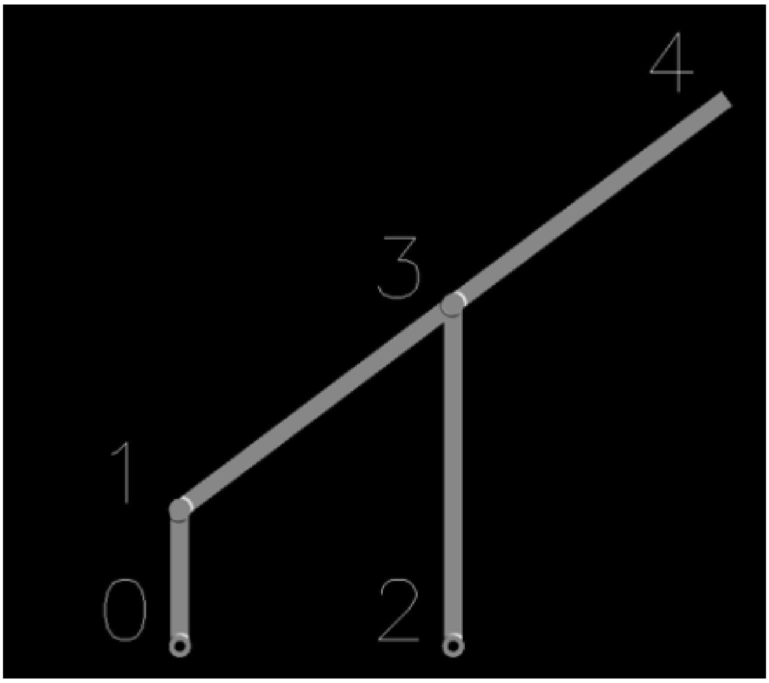


Figure 5.1: The geometry of the straight line generator, as generated by the KBE system, in the form that solves the original case

From Figure 5.2 it is possible to see that the algorithm fairly quickly goes from the lower left corner, where it started, to a location near the convergence point. In some later iteration the algorithm jumps to the upper left corner, and again quickly moves to the convergence point. This last step repeats, multiple times, and the quick movement appears to be happening along a line.

Optimisation With Constraint

When the constraint was introduced the problem converged to a markedly different point, most notably the design variable of corresponding to the y value became negative, and the value of the objective increased slightly. In order to confirm that the problem formulation was correct, both the known reference design, and the new solution were analysed directly in FEDEM, without relying on the optimisation system. The output of this analysis is available in Figure 5.4, and per the constraint, the path never obtains a negative y value.

The behaviour of the optimisation algorithm is shown in Figure 5.3. In this figure, one can note that the algorithm rapidly decreases the objective function, but then is blocked by the constraint. Soon the algorithm does a jump and again rapidly decrease the objective, this behaviour similar to the behaviour for the unconstrained case.

Optimisation on Surrogate Model

For the DoE 40 cases were evaluated. The results of these experiments are visualised in Figure 5.5, which has the same logic and interpretation for the placement of the circles corresponding to Figure 5.2. From these observations two surrogate models were constructed using OpenMDAO. One SM for modelling the objective function, and one for modelling the constraint function. When the optimisation algorithm was used on these functions, the algorithm converged to a design practically equal to the design found when the algorithm worked on the KBE model directly.

In order to confirm the accuracy of the approximations, the surrogate model was used to create approximations for the value of the objective function in specific points. Such approximations were set up for every point evaluated when the constrained case was solved by direct optimisation. Figure 5.6 visualises the deviation between the approximation and the real value of the points mentioned above, and indicate that the deviation is spread fairly evenly thought out the design space.

5.1.4 Discussion of Case Results

Given the complexity, as described in Section 4, especially the complexity associated with the connection to AML (Section 4.3.2), the fact that the system can reliably complete iterations, as described in Section 4.5, can be considered to be promising evidence of the feasibility of connecting AML to Python based optimisation tools. Further, the effectiveness of the somewhat abstract formulations of the objective, and the constraint, also support an optimistic view of what optimisation can achieve in the design of mechanisms. Specifically, the results show that when optimisation is used, designs that already are known can be rediscovered. When compared with traditional synthesis methods, the rediscovery process might even involve less work, if the optimisation system already has been created, since only a declaration of the objective is required.

The solution to the constrained problem is obtained with a simple extension of the formulation for the unconstrained case. The ease of extending the unconstrained case offers some insight into the generality and abstractness that naturally comes when specifying design problems as optimisation problems. Thus it shows that, in a workflow using optimisation, the requirements can be iteratively refined.

In addition to simplifying the workflow, the formulation of design problems as optimisation problems may provide better designs, when compared with traditional analytic methods. The unconstrained case demonstrated this phenomenon. The performance, as measured, was worse for the reference design than for the design found by the optimisation algorithm. This can be explained by the fact that the optimisation process incorporates more physical phenomenon, such as inertia and flexibility. Thus one may consider the unconstrained optimisation problem to be a better description of the design problem, as the analytic specification is not able to provide a design that has the same performance.

Increases in the objective function, as seen in both the constrained and unconstrained case, can be attributed to the line search procedure used in SQP algorithms [20, p. 545]. However a thorough treatment of this topic is not included, as this not deemed necessary. In comparison the practical considerations associated with the line search is important. Primarily the issue is that, since the objective is not strictly decreasing, the last iteration is not always going to be anywhere near the optimum. It has been claimed that it is important that the latest iteration has the best performance [35, p. 53]. However, with the logging capabilities included in OpenMDAO it is trivial to find the best design over all the iterations. Still, if the optimisation algorithm exits with a message, that indicates normal convergence was not achieved, this step should not be omitted.

Since finite difference was used on both the constrained and unconstrained case, the issue of setting the step size had to be met. Since the design parameters are real parameters, with a physical meaning, any engineer with an intuition for the amount of change required to obtain a meaningful, but small, change in the design, can come up with a sensible step size. Using this method the step size was set to 10.0mm, which was used when both cases converged.

In the results, it was noted that the final value of the objective function was smaller for the unconstrained case when compared with the corresponding value for the constrained case. This is in full agreement with the KKT conditions, described in Section 2.1.2. From the FEDEM analysis, it is possible to confirm that the constraint would be violated in the unconstrained optimum. Further one can then see, from Equation (2.2), that when the constraint is nonzero, the gradient of the objective is nonzero. Thus the constrained optimum does not coincide with the unconstrained optimum, and since the unconstrained optimum is the minimum of the objective function, the constrained optimum must have a larger value. From a designer's perspective, the increase in the objective function can be intuitively understood, by thinking of the constraint as an additional restraint to the design process, hence it is impossible to create the best possible design.

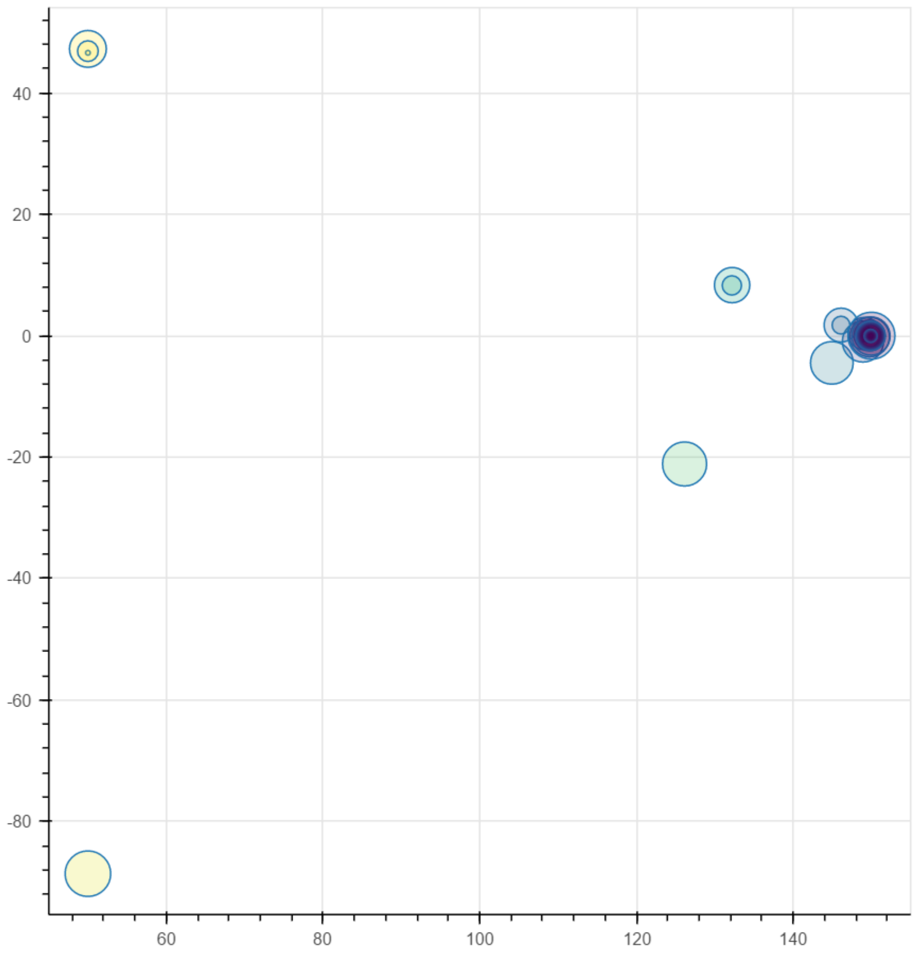


Figure 5.2: Designs evaluated when optimising without constraint (colour indicates value of cost function, with yellow for high, and dark purple for low cost; size of circle indicates iteration, with diameter decreasing for each iteration)

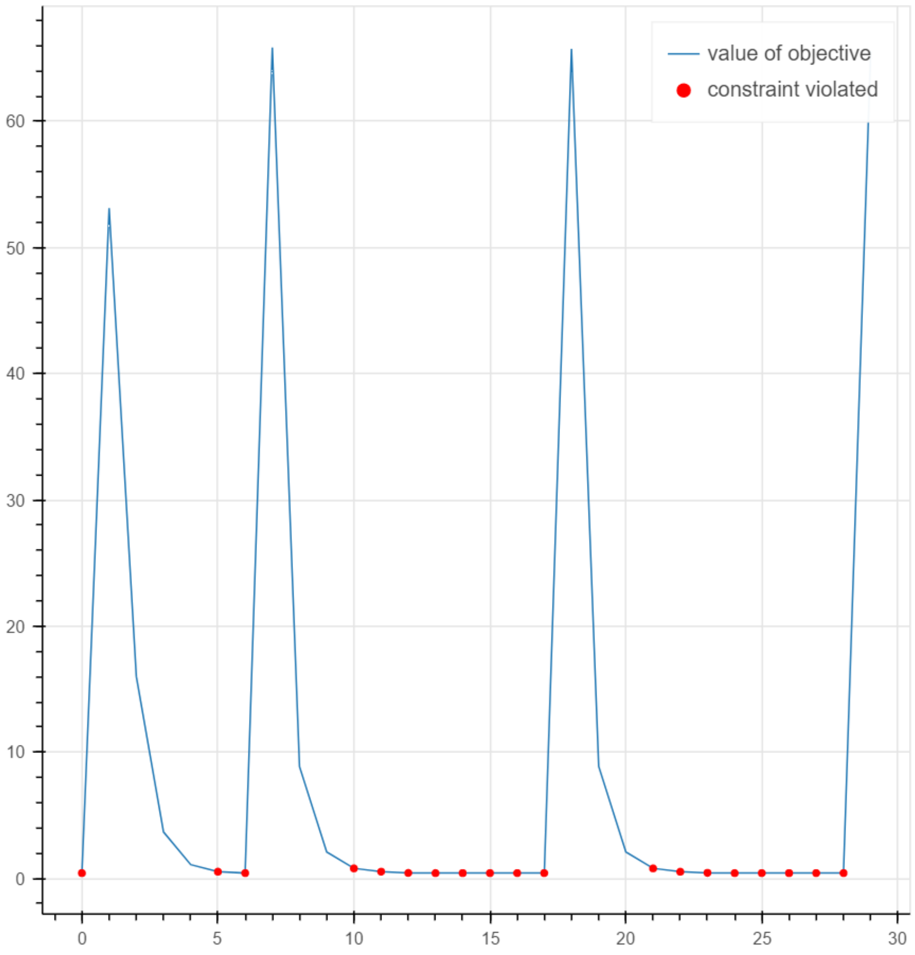
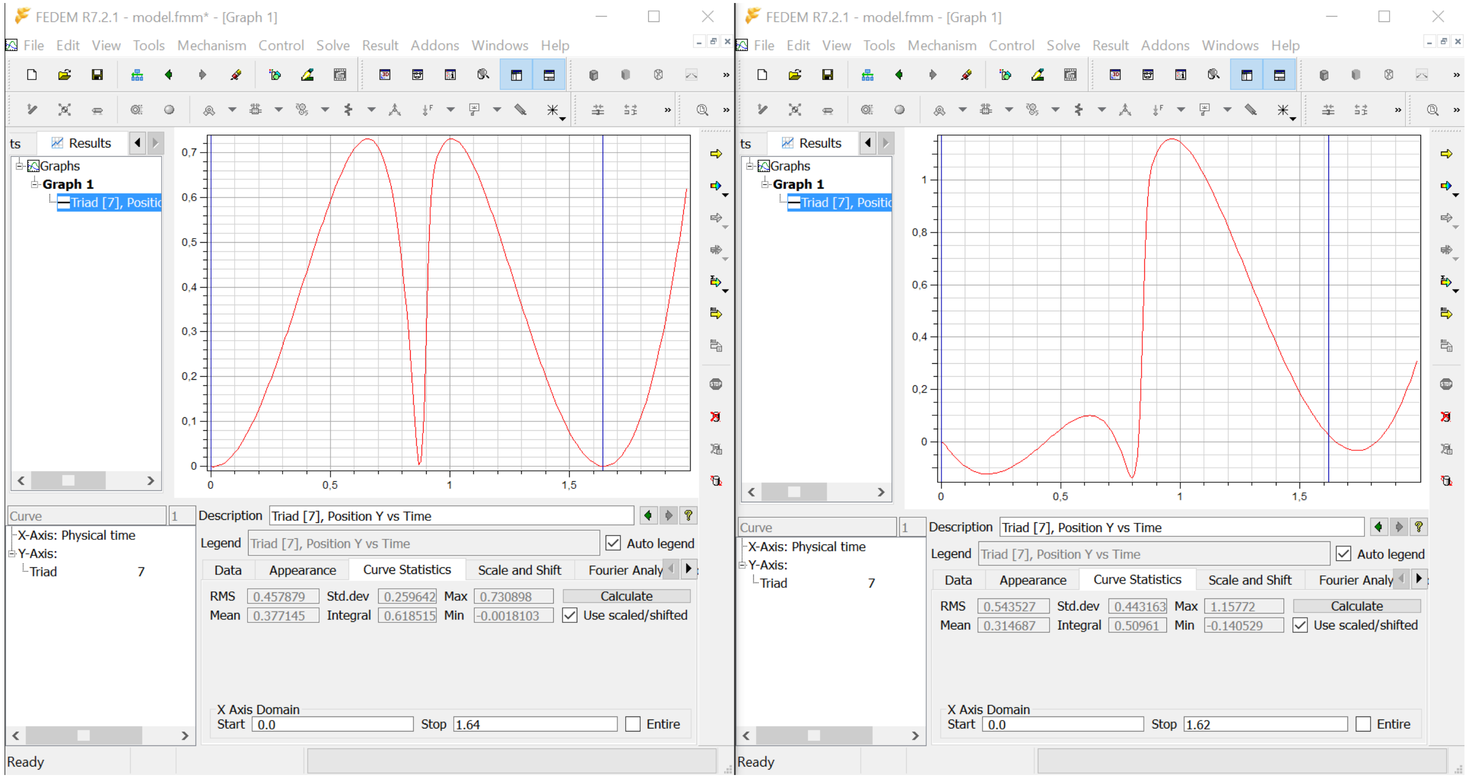


Figure 5.3: Value of cost function, for each iteration



X-Axis: Physical time
Y-Axis:
Triad 7

Description: Triad [7], Position Y vs Time
Legend: Triad [7], Position Y vs Time [✓ Auto legend]

```
RMS   0.457879   Std.dev   0.259642   Max   0.730898
Mean  0.377145   Integral  0.618515   Min   -0.0018103
[✓ Use scaled/shifted]

X Axis Domain
Start 0.0        Stop 1.64        [ ] Entire
```

X-Axis: Physical time
Y-Axis:
Triad 7

Description: Triad [7], Position Y vs Time
Legend: Triad [7], Position Y vs Time [✓ Auto legend]

```
RMS   0.543527   Std.dev   0.443163   Max   1.15772
Mean  0.314687   Integral  0.50961    Min   -0.140529
[✓ Use scaled/shifted]

X Axis Domain
Start 0.0        Stop 1.62        [ ] Entire
```

Figure 5.4: Deviation, from reference path, in the y direction, for two designs. Right is the reference design, left is the optimum found when the constraint is enforced

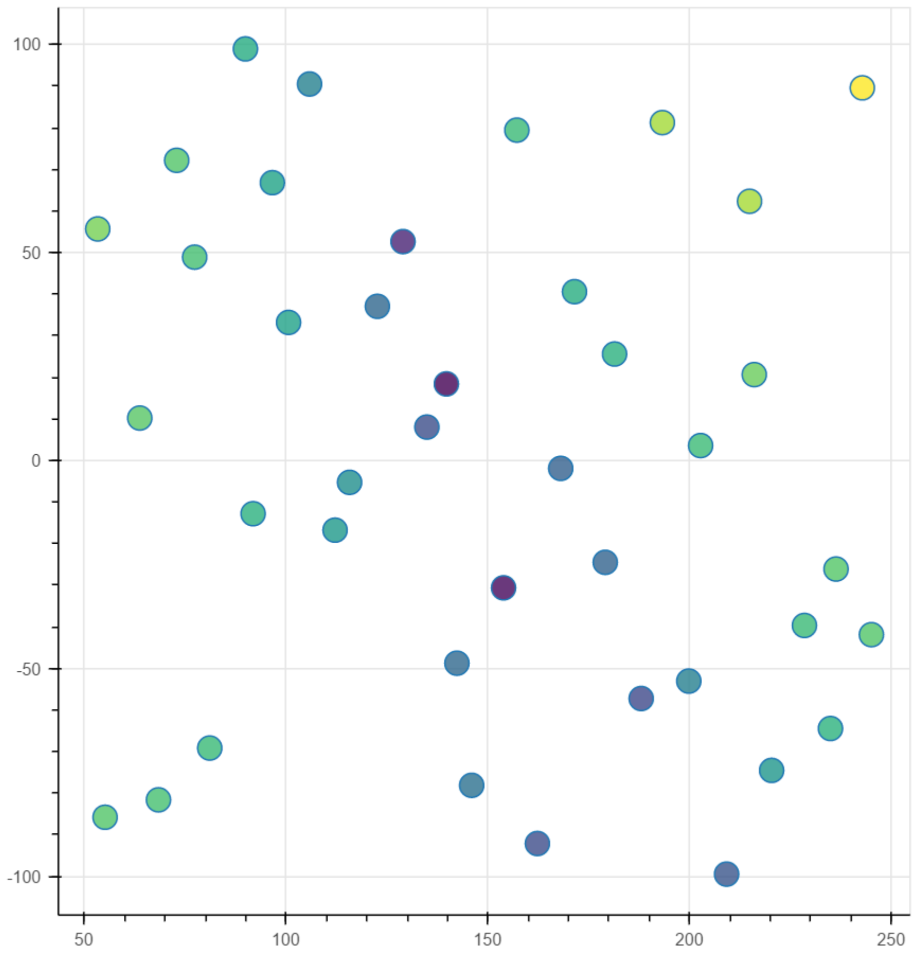


Figure 5.5: Samples in the design of experiments (colour indicates value of cost function, with yellow for high, and dark purple for low cost)

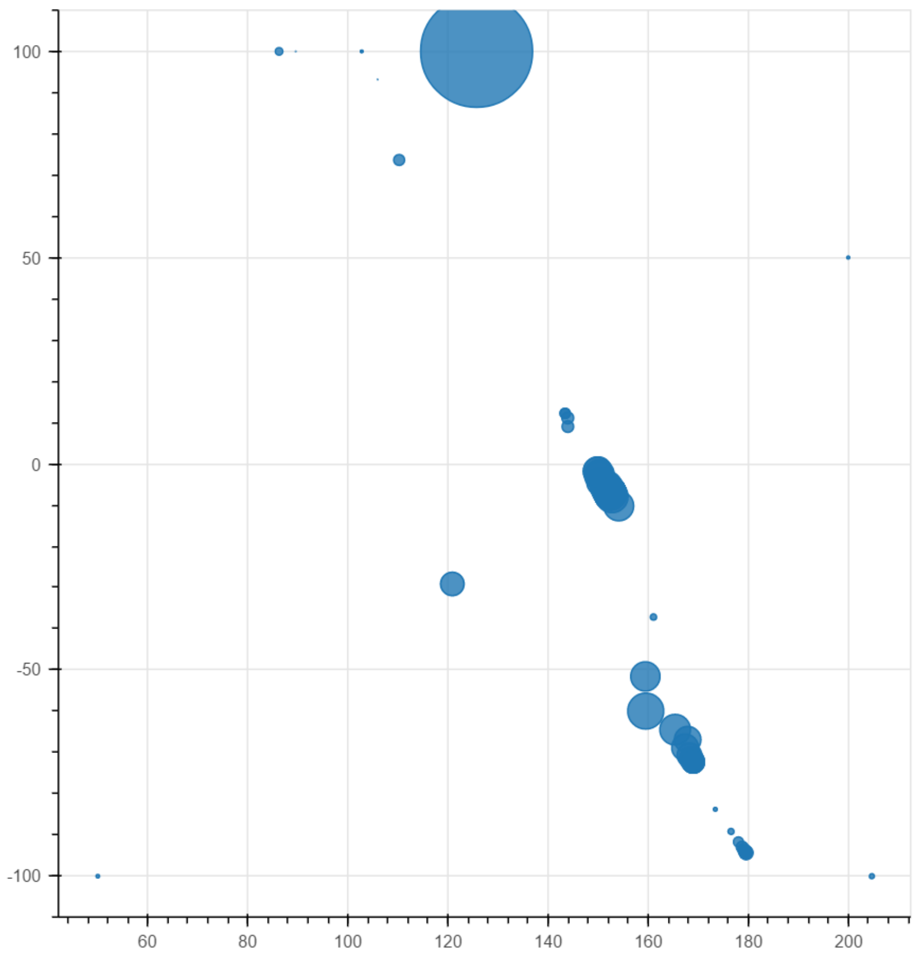


Figure 5.6: Deviation in value of the objective, between the predictions of the surrogate model and evaluations in the corresponding direct optimisation (larger circle indicates larger deviation)

5.2 Case 2: Comparison of Optimisation Algorithms

This case solves a mechanism design problem multiple times, using different optimisation algorithms. The goal is to obtain information on the effectiveness of the various optimisation algorithms. Aspects of interest include execution time and any additional considerations that affect the algorithm's suitability as part of an optimisation system.

5.2.1 Model Definition

For this case, the exact model and design space, as used in Section 5.1, were reused, with the constraint active.

5.2.2 Case Executions

The three optimisation algorithms described in Section 4.2.2, namely SLSQP, SNOPT and IPOPT were used. Hence three separate executions were performed as part of this case. As described in Section 4.6.3 and Section 4.6.4, there were some issues related to the compilation required for this algorithms, and the virtual Linux machine was used to circumvent these issues.

To reduce the execution time, and thus make experimentation easier, a design of experiments and surrogate model were used. The points used to construct the surrogate model are shown in Figure 5.7. Since a large number of points, specifically 128, were used, it is assumed that the generated approximations are sufficiently accurate for evaluating the performance of the algorithms. Such a high number of observations are probably excessive if the goal was to solve a single optimisation problem, however since the goal is to evaluate the algorithms, no effort was made to try a smaller sample set.

5.2.3 Results

All three algorithms converged to the same point, as illustrated by Figure 5.8. Also evident from that figure is the fact that the SNOPT algorithm takes a more indirect path, whereas the other two algorithms take a more direct path (all paths starts in the upper right corner). Figure 5.9 also supports the view that SNOPT is less direct, this algorithm uses a significantly higher number of iterations to reduce the objective to the value for which the algorithms converge.

All function evaluations performed outside the feasible region are marked in Figure 5.9, but only SNOPT had this behaviour. This observation can also be made from the visualisation of the constraints in Figure 5.10, which illustrates how IPOPT and SLSQP steadily approaches the value of 0, but never reaches it. SNOPT, on the other hand, actually violates the constraint at multiple iterations.

5.2.4 Discussion of Case Results

In the aggregate, one might say that the behaviour of IPOPT and SLSQP are very similar, and SNOPT contrasts their behaviour by having worse convergence rate, and evaluating the design outside the feasible region. This is to some extent surprising, since the SLSQP and SNOPT are both SQP algorithms, but IPOPT is interior point based. SNOPT has also been demonstrated to be faster than the two alternatives used in this work, when optimising a mechanism [30].

The speed difference can probably be attributed to the low number of design variables and constraint used in this case, as SNOPT is meant to work with problems of higher dimensionality. Still, the difference in convergence rate is still disappointing, especially since SNOPT not only reduce the objective the slowest, but also approaches the constraint the slowest, and then proceeds to violate it. Further, the lacking performance of SNOPT is hampered by the monetary cost of obtaining a license, as the other two algorithms are open source projects, and thus freely available.

IPOPT does not evaluate the model in the infeasible region, this can be a benefit in engineering problems, since the models, being physics based, might not be sensible outside the feasible region. Thus it is not necessary to formulate models that are valid in points of the design space which are infeasible. Although, if the initial design is infeasible, the model must allow the algorithm to move to a feasible point. Also, due to the numerical nature, the design might be evaluated slightly outside, but still near, the border of the feasible region

Finally one should observe that the performance of the optimisation algorithm is less important if a surrogate model is used. The construction of these models means that the evaluation of both the values and sensitivities are fast. Therefore the execution time of the optimisation algorithm is orders of magnitude smaller than the gathering of sample data.

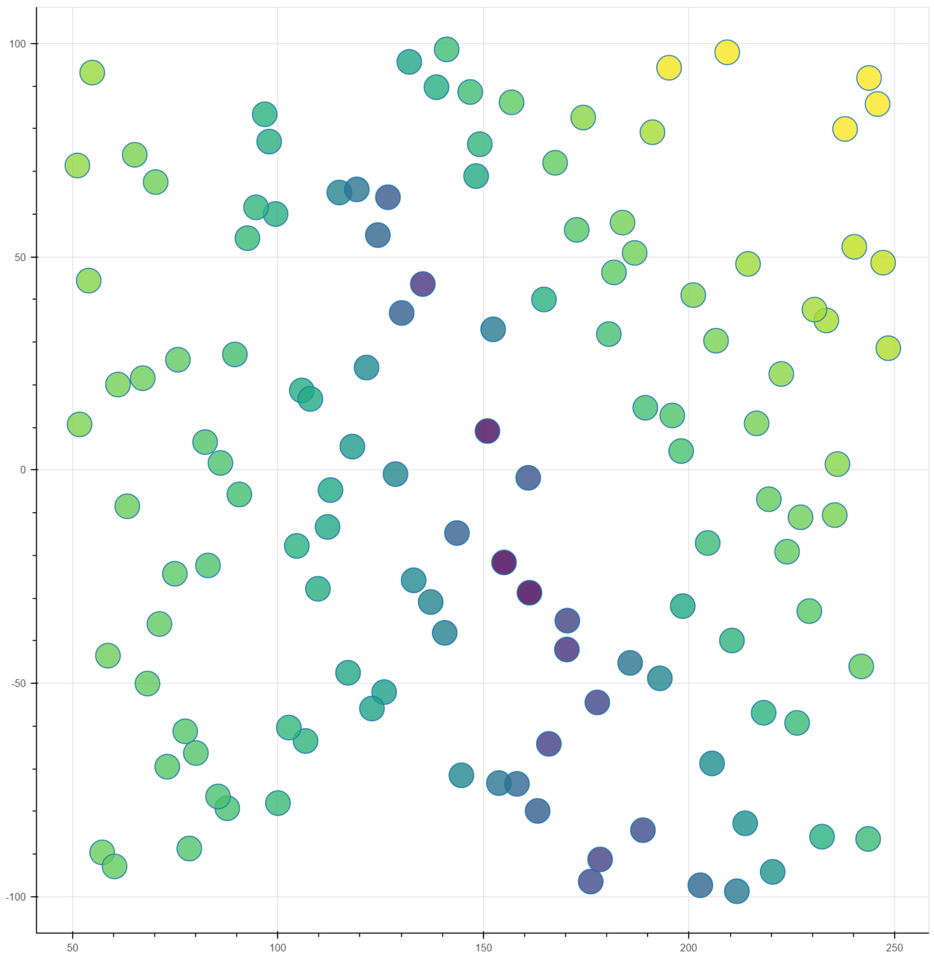


Figure 5.7: Samples in the design of experiments (colour indicates value of cost function, with yellow for high, and dark purple for low cost)

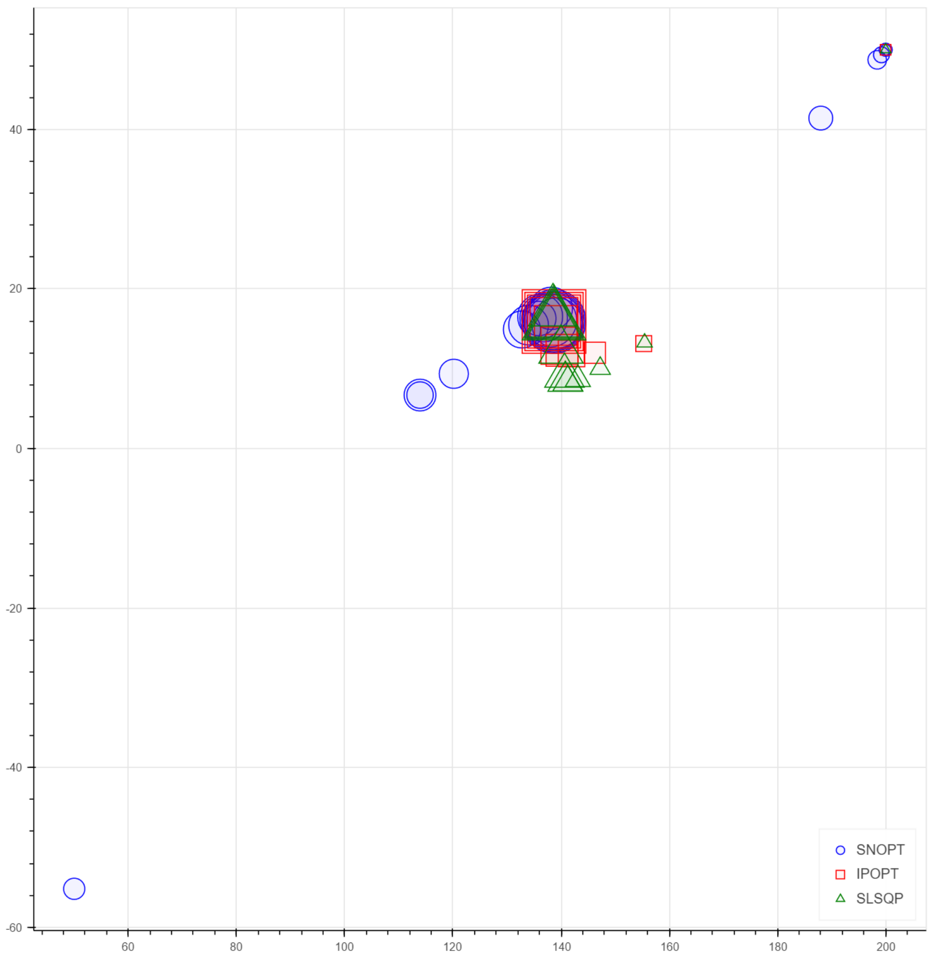


Figure 5.8: Designs evaluated when optimising (size of symbol indicates iteration, with magnitude increasing for each iteration)

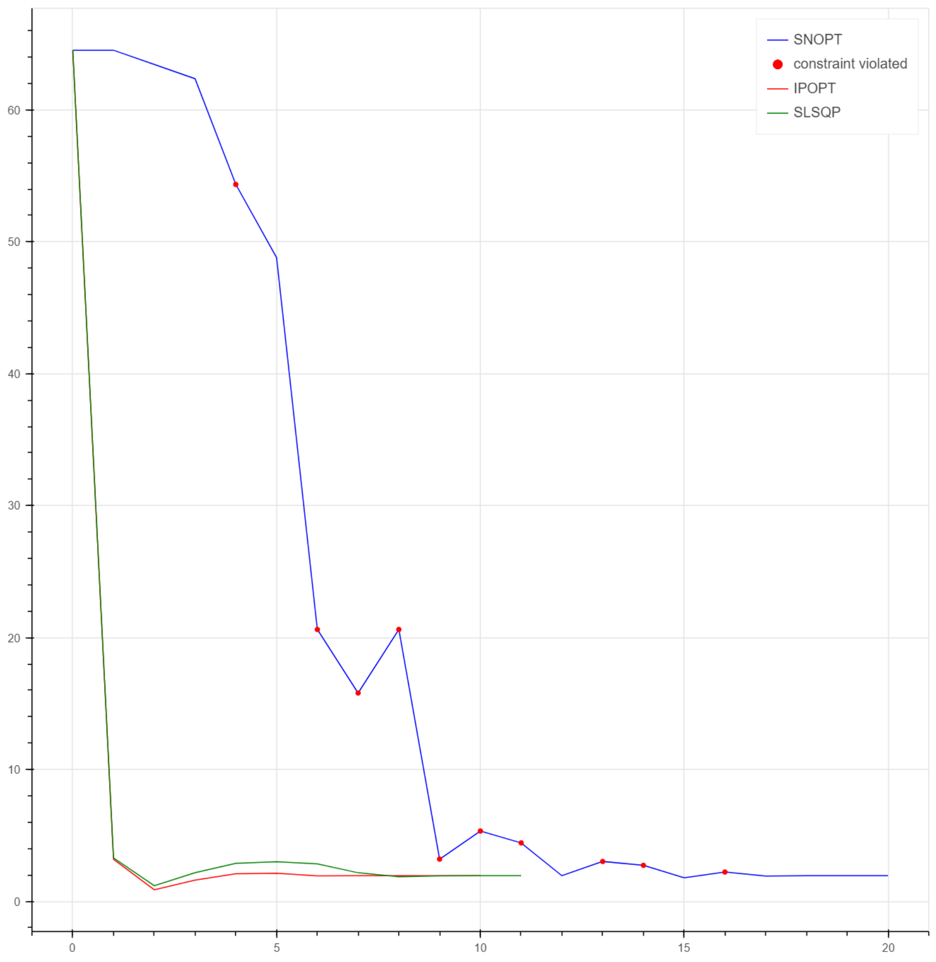


Figure 5.9: Value of objective, for each iteration

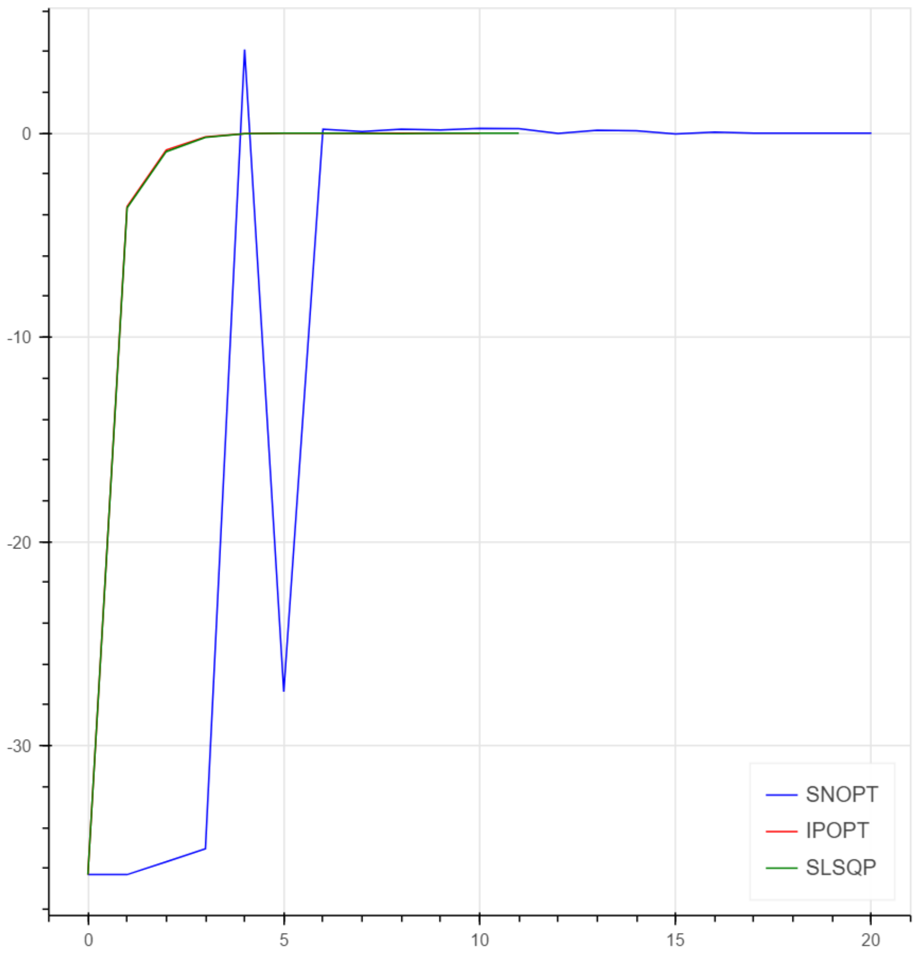


Figure 5.10: Value of the constraint, for each iteration

5.3 Case 3: Concerns Relating to Complexity

In the final case, a more complex optimisation problem is explored. When the additional complexity is introduced, the case becomes more informative, and can better express the suitability of optimisation in the mechanism design process. The effectiveness of the current system can also be more deeply investigated. Due to the more exploratory nature of this case, this section uses a narrative to illustrate the appropriate ideas, as opposed to Section 5.1 and Section 5.2, which has a more rigid outline. Since the text is meant to describe high-level concerns, the reader is also referred to Appendix B, which contains the source code level of details necessary for a complete understanding of the design space definition and design analysis.

5.3.1 The Original Case

In Myklebust [19, p. 82-91] there is a description of a case, and the same design problem has been repurposed for this case. The design is of a mechanism that has the purpose of taking parcels of a conveyor belt and placing them on a rotating turret. A Stephenson 3 mechanism is used to solve the problem. Figure 5.12 shows the parameters Myklebust used to define the problem, many of the parameters can be considered state variables when the problem is reformulated as an optimisation problem. Some of the variables, such as the positions, will be driven by the optimisation algorithm since they are required to instantiate the KBE model; thus they form the basis for the design variables. Variables such as the velocities can only be obtained from FEDEM, and thus they should be considered state variables. They must, therefore, be used when formulating the objective and constraints.

Figure 5.11 shows the path of the coupler, in in the mechanism Myklebust found to solve the design problem. This path is available in FEDEM when a design has been evaluated. Myklebust uses numerical values only for the pickup and deposit points, as the system used requires the specifications to be given for specific positions of the mechanism. Since the complete path is available in FEDEM, a more complex case could be developed, where e.g. the maximum acceleration for the whole path is a constraint. Such adaptations of the case have not been attempted, and the specifications of the original case are used directly.

5.3.2 Results Extraction

The original case solves the design problem exactly, as a consequence of the design tooling used. When the same case is to be reformulated as an optimisation problem, it is not necessarily known what the most appropriate formulations of the objective and constraints are. For this reason, it was decided to do a design of experiments and storing the state variables that were used in the original case as the results of each experiment. With such a setup, many formulations of the objective can be created after all the expensive simulations have been completed.

As stated, most of the design parameters, in the original case, are associated with either the pickup point or the deposit point. These responses are easily obtained as a time series from FEDEM. However, the design problem requires the responses for specific positions of the coupler point. Consequently, the times for the pickup and deposit action has to be calculated, and this is done by taking the two time steps for which the coupler is closest to the pickup point and deposit point respectively.

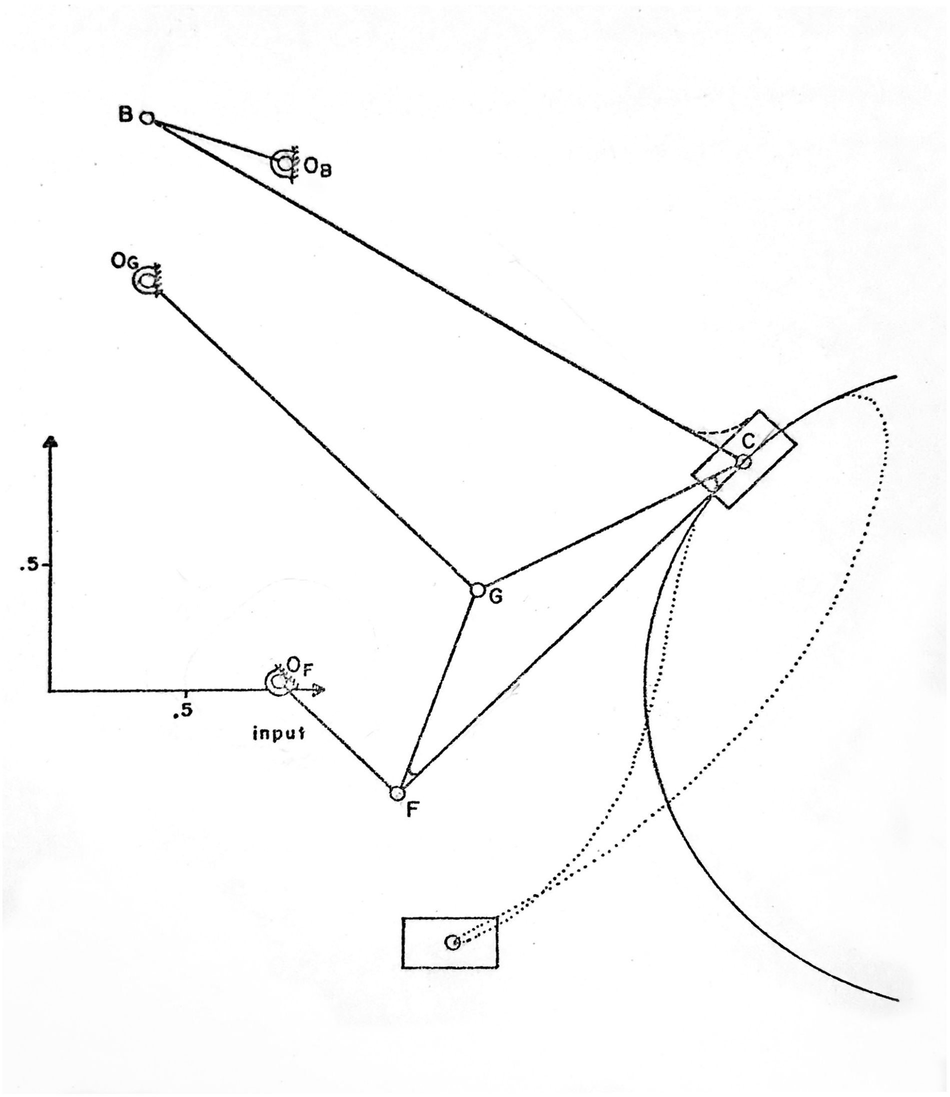
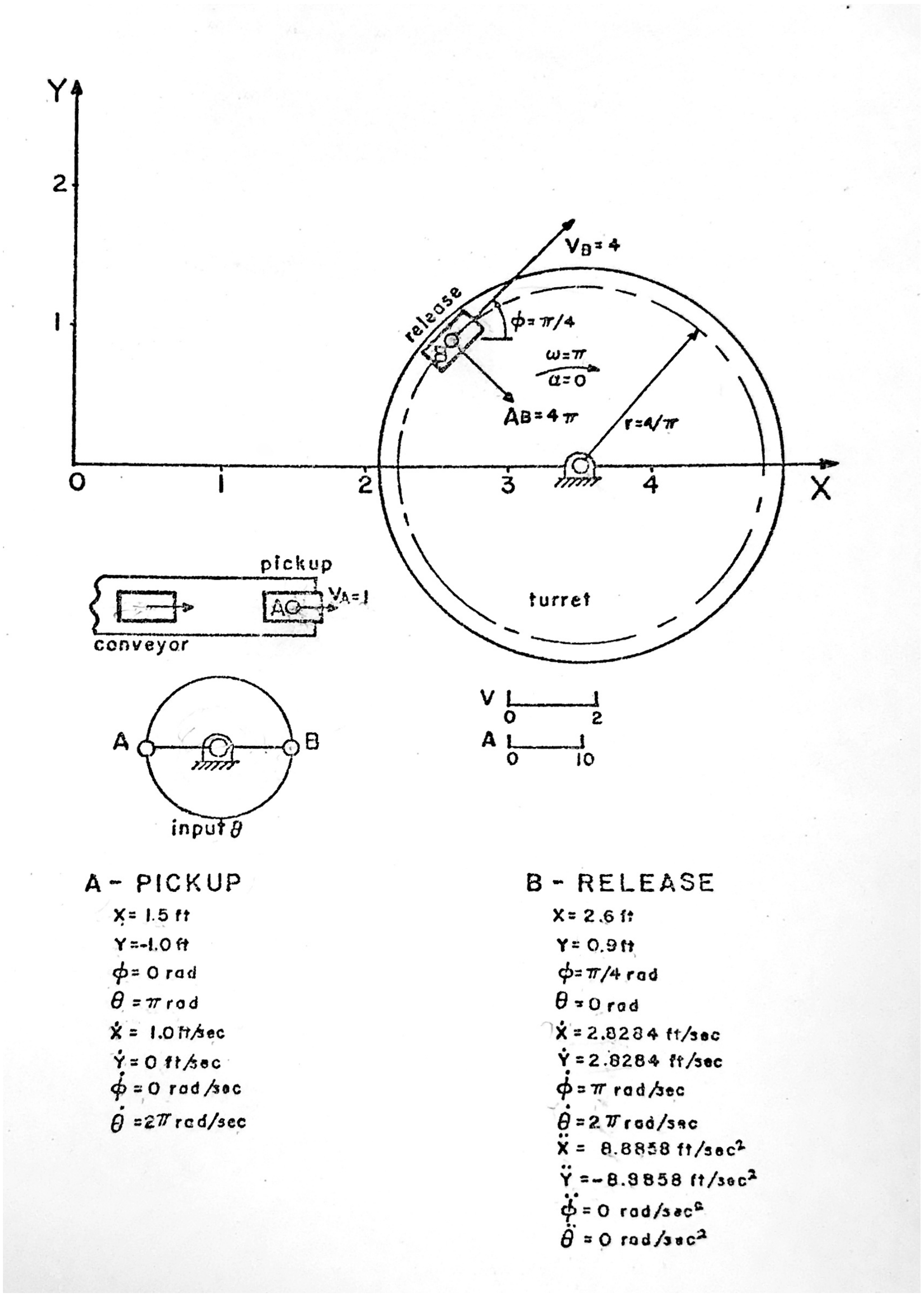


Figure 5.11: The geometry of the Stephenson 3 mechanism, as given in the original case, with the couple path given. The geometry is in the deposit position with the coupler at the turret. (Figure reprinted from Myklebust [19, p. 85])



A - PICKUP

$X = 1.5$ ft
$Y = -1.0$ ft
$\phi = 0$ rad
$\theta = \pi$ rad
$\dot{X} = 1.0$ ft/sec
$\dot{Y} = 0$ ft/sec
$\dot{\phi} = 0$ rad/sec
$\dot{\theta} = 2\pi$ rad/sec

B - RELEASE

$X = 2.6$ ft
$Y = 0.9$ ft
$\phi = \pi/4$ rad
$\theta = 0$ rad
$\dot{X} = 2.8284$ ft/sec
$\dot{Y} = 2.8284$ ft/sec
$\dot{\phi} = \pi$ rad/sec
$\dot{\theta} = 2\pi$ rad/sec
$\ddot{X} = 8.8858$ ft/sec^2
$\ddot{Y} = -8.8858$ ft/sec^2
$\ddot{\phi} = 0$ rad/sec^2
$\ddot{\theta} = 0$ rad/sec^2

Figure 5.12: The input parameters used in the original case, including the graphical interpretation of the mathematical values. (Figure reprinted from Myklebust [19, p. 84])

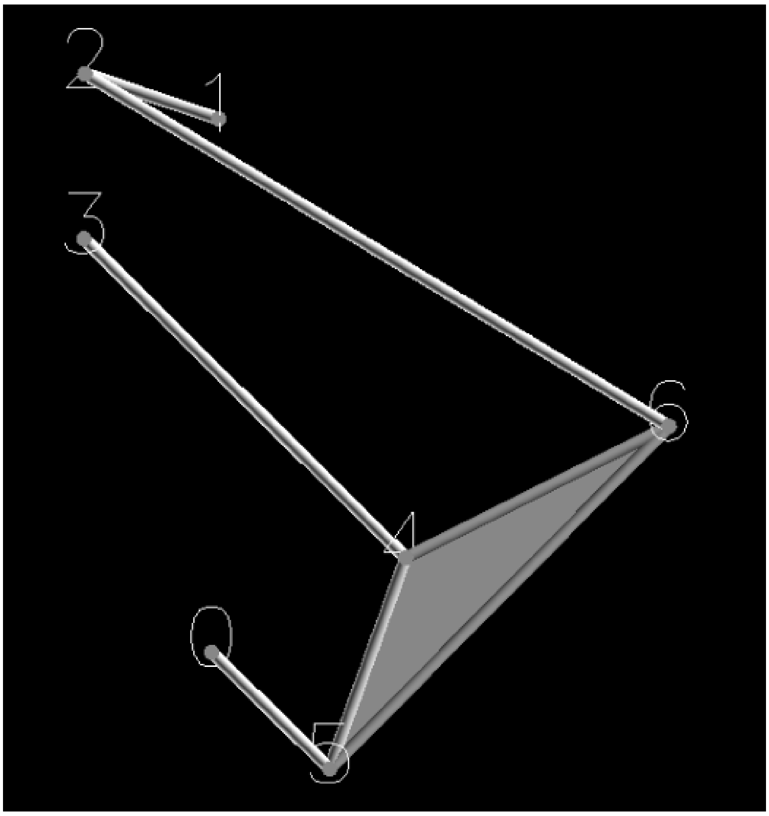


Figure 5.13: The geometry of the Stephenson 3 mechanism, generated by the KBE system, in the form that solves the original case

5.3.3 Initial Attempt

For the initial attempt, four design variables were used, these variables correspond to varying the points 3 and 4 in Figure 5.13. All the evaluated designs are visualised in Figure 5.14, and from this figure, it is evident that many designs failed. By observing the designs in FEDEM, it is possible to see that the main reason for a design failing is that the mechanism jams; the mechanism is not able to do a full rotation, it becomes a rocker-rocker configuration.

Even with the high number of failures, an attempt at constructing a surrogate model was made, by ignoring the failed points. In the beginning, there was a weak correspondence between the surrogate model and the values in the observation. The reason for this turned out to be the issue relating to surrogate models that use multiple vectors, i.e. the two points, as parameters. This was discussed in Section 4.6.2, and resolved by combining the points into a single vector.

With a sensible surrogate model constructed, an attempt at solving the design problem was made, by minimising the distance between the vector of responses and the vector of desired responses. This problem converged, but when the design suggested by the algorithm was opened in FEDEM, it became evident that the design was not able to make a complete revolution. Since the surrogate model had no observations of the jamming behaviour, the fact that the algorithm was unable to bypass this issue can not be considered a fault.

5.3.4 Changing the Parameterisation

By making further observations on the jamming behaviour mentioned above, the reason for this behaviour can be recognised. Taking all point numbers from Figure 5.13, one might note that, by only considering the link segments $(3, 4)$ and $(4, 5)$, and considering the joint in Point 3 to be fixed, all possible locations of Point 5 is easily defined. This location forms a disc, represented by the white area in Figure 5.15 , and the area has to contain the circle defining the path of the crank (i.e. segment $(0, 5)$). The d_i values can be thought of as some sort of clearance size values. They have to be positive for the path of the Point 5 on the crank to be in the area Point 5 can reach as part of the segments $(3, 4)$ and $(4, 5)$.

It is possible to obtain additional insight from Figure 5.15, by noting that the relation below must necessarily hold.

$$L_0 + L_1 = (L_0 - L_1) + d_1 + 2L_2 + d_0 \tag{5.1}$$

$$L_1 = \frac{d_1 + d_0}{2} + L_2 \tag{5.2}$$

By solving for L_1, it can be shown that this variable is a function of d_0, d_1 and L_2. Thus the L_1 can be removed as a parameter of the mechanism; it can be derived from other values. Further, as the values d_i does not affect the geometry directly, the sum can be rewritten as $d_0 + d_1 = D$, with the move limit $D > 0$.

$$L_1 = \frac{D}{2} + L_2 \tag{5.3}$$

By using the relation in (5.3) the mechanism can be parameterised such that the jamming behaviour can be eliminated. First, the mechanism is defined with a set of lengths and the angles of the joints (in the design position). Then Equation (5.3) is susbtituted for L_1. From such a description, the points that serve as input to the KBE system can be calculated in a fashion similar to the transformation from polar to Cartesian coordinates.

With this change of parameters, a DoE was executed, the results are shown in Figure 5.16. All errors relating to the jamming behaviour was removed. Still, as illustrated, a large number of failed experiments are present. These were due to the meshing procedure failing.

5.3.5 Removing Bugs

In an attempt to remove the meshing related failures, the creation of blends was removed. After the automatic creation of blends was turned off, all errors vanished when the experiments in Figure 5.16 where executed a second time. With a robust model, a new DoE was computed, in an attempt to reduce any inaccuracies, a large sample size of 256 was selected. All of these samples are visualised in Figure 5.17. The nature of the few errors that still occur are unknown, but because of the low number and distribution, the missing values are assumed to be insignificant.

These new samples were first used to create various surrogate models. The performance of the surrogate models were compared by extracting the predicted responses for the reference design. Only the nearest neighbour surrogate model type was found to give sensible results. Table 5.1 has some of the predicted values listed, together with the requirements and actual results from FEDEM.

Table 5.1: Case 3 Results

	Reference design		Optimisation on: Model of responses		Optimisation on: Model of objective		Target
	From SM	From FEDEM	From SM	From FEDEM	From SM	From FEDEM	
a_x	2.367	2.424	2.466	2.545	–	2.692	2.708
a_y	-2.260	-2.627	-2.465	-2.163	–	-2.805	-2.708
v_x	-0.852	-0.873	-0.854	-0.854	–	-0.855	-0.862
v_y	-0.881	-0.860	-0.880	-0.879	–	-0.881	-0.862
t_p	3.522	3.530	3.492	3.493	–	3.503	3.500

In addition to evaluating SM types, two attempts at discovering a design that meets the requirements, with the help of optimisation, were made. The first attempt used models of the responses, and the objective was calculated for each iteration, as the Cartesian distance to the requirements. The second attempt calculated the value of the objective, using the same Cartesian distance. Then a model of was constructed, using the values of the objective as the samples. Both of the attempts converged, and the values for the responses are given in Table 5.1. Only the velocities and acceleration at the pickup point, and the time at which the mechanism was in the pickup position, were used in the optimisation, mainly as an initial simplification.

From Table 5.1 two key observations can be made. Firstly note that when the problem is solved by optimising on the model of the objective, the results are closer to the target requirements, when compared with the reference design. This can be interpreted in two ways. If results from FEDEM are taken as the real behaviour of the mechanism, the optimisation algorithm can obtain results that are better than the analytic method used for the reference design. Since FEDEM incorporates more physics, this view has some appeal. If on the other hand, the analytic results are taken to be the true results, the optimisation algorithm can get as close to the target requirements as can be expected, since the FEDEM based model is not able to give an accurate value of the true results. For both of the interpretations, the results can be said to be reasonably close, especially since the values of the responses has quite an extensive range over the design space.

The second thing to note is that when the optimisation algorithm uses a surrogate model of the objective, the predicted value is negative at the optimum. This has no physical meaning; it should be impossible to get a negative value. However, the degenerate model seems to give better results when compared with the optimisation on the model of the responses. Optimisation does not necessarily need accurate absolute predictions from the model, the relative differences of two designs are sufficient to compare them. Thus, the results seem to suggest that when the model, using is using an approximation of the objective, can capture the relative differences, in such a way that the final solution has a better performance.

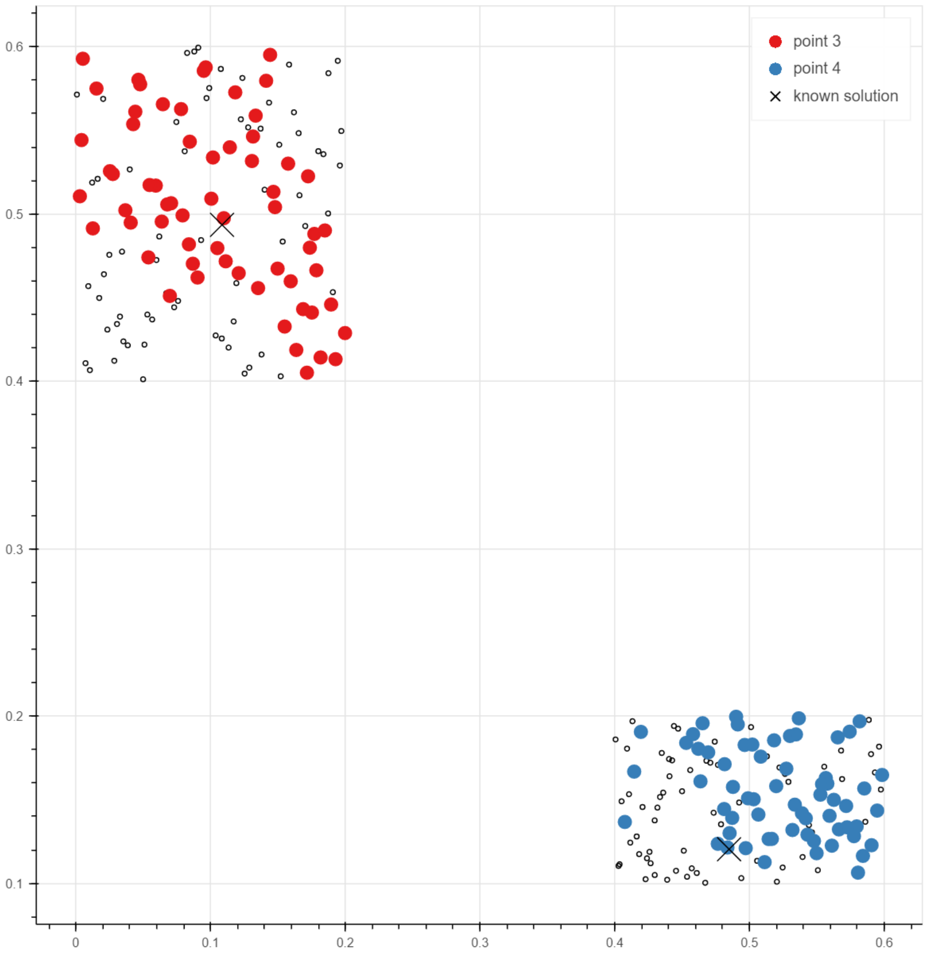


Figure 5.14: Designs evaluated for the Stephenson 3 mechanism (two points are plotted for each design evaluated, with the position of each point corresponding to the point's position in the global reference system; circle without colour indicated failed simulation)

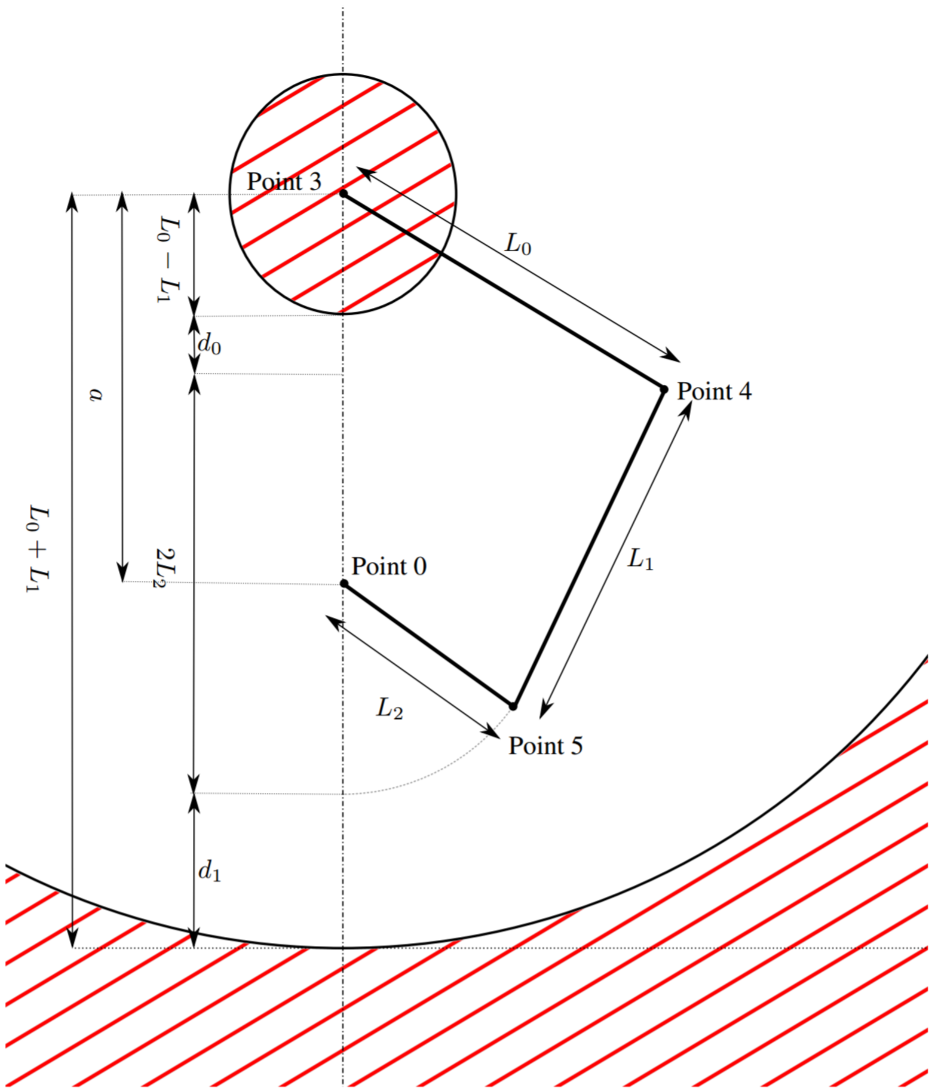


Figure 5.15: Illustration of the area point 5 can not reach, with the relevant lengths shown

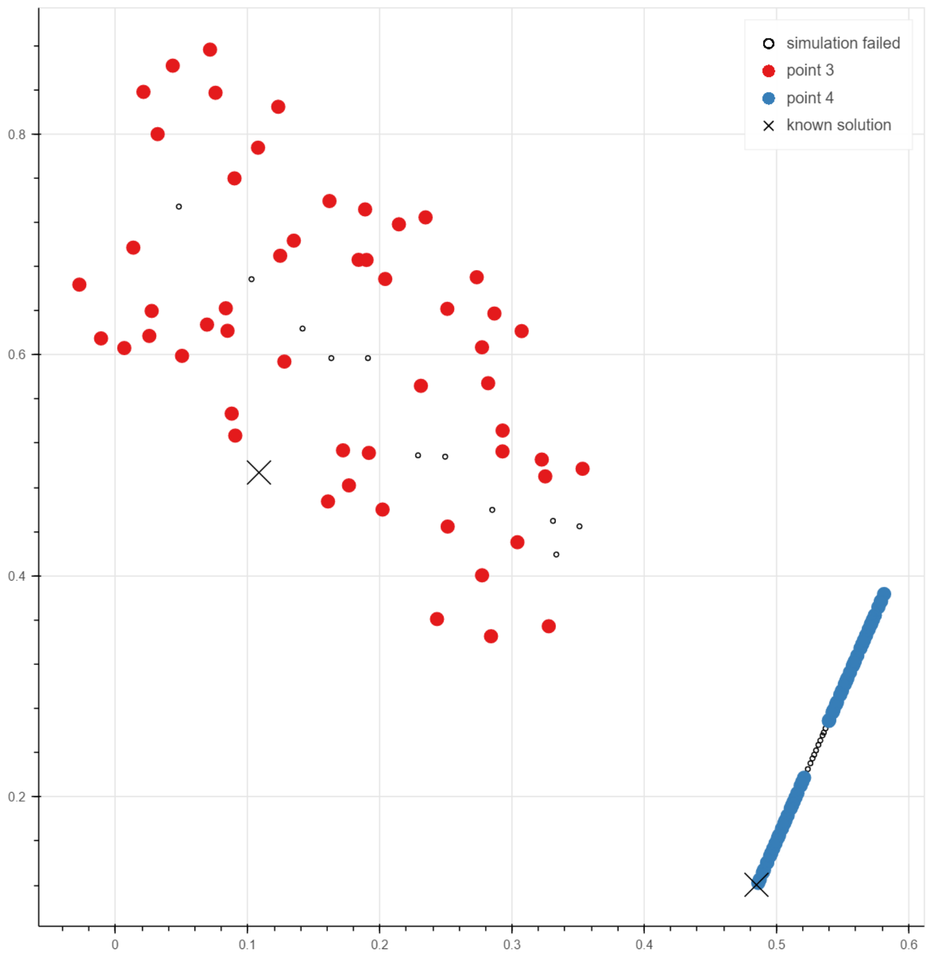


Figure 5.16: Designs evaluated for the Stephenson 3 mechanism, with new parameterisation (two points are plotted for each design evaluated, with the position of each point corresponding to the point's position in the global reference system)

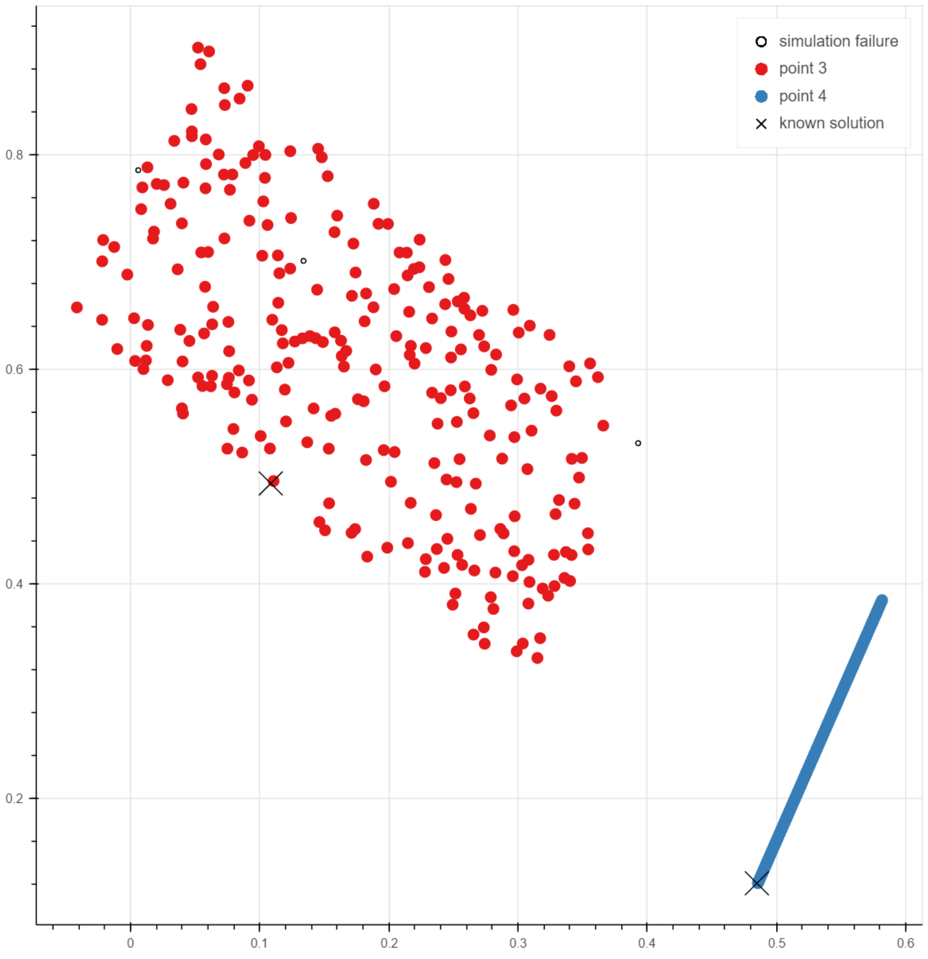


Figure 5.17: Designs evaluated for the Stephenson 3 mechanism, with new parameterisation, and no blends (two points are plotted for each design evaluated, with the position of each point corresponding to the point's position in the global reference system)

CHAPTER

SIX

DISCUSSION

This chapter contains the overarching concerns relating to the performance and architecture of the optimisation system that is described in the previous chapters. In the subsequent sections, the concerns discussed take three main perspectives; these relate to the use of optimisation with a KBE system, the use of optimisation in mechanism design, and the process of extending the specific KBE system used in this work. While three such perspectives are used, they are interconnected and treating them individually would cause the text to be extraneously hard to read. Instead, the discussion is structured by grouping the concerns into three categories: feasibility, applicability and usability.

Such a structuring provides a narrative that is compelling, mainly because it follows the steps that would be required, should a real system be developed, as opposed to the prototype system that has been developed. The feasibility section looks at the possibility of developing a real optimisation system, and it mainly takes a software development viewpoint. In the applicability section, the focus is on the use of the optimisation system in the design process; it has the design process as the primary viewpoint. Finally, the usability section addresses concerns relating to the end user of a real system, and thus it has the end user as the basis for the viewpoint.

6.1 Feasibility

For the feasibility of the system the results from the cases, which demonstrates working optimisation, is the primary outcome. Actual optimisation has been accomplished, even with all the moving parts, including the use of third-party optimisation tools, KBE generated geometry, and a finite element tool for analysis. With such results, it can be claimed that the feasibility of developing the required optimisation system is proven. On the other hand, the extent to which the complexity of design problems solved by the system can be increased, especially in its current state, can not be said to be sufficiently proven. However, this concern relates more to the applicability and usability of the system. The feasibility of the software is proven by the fact that all the systems have been successfully connected.

Furthermore, the breadth, of the design problems solved in the results, suggest that both the tools selected in the implementation of the system and the system itself are somewhat general. Essentially the fact that the system can solve multiple problems indicate that it is feasible to create a general system for optimisation. This is especially interesting as on the software side of things, as AML does not currently have any support for batch mode operation. With the success of the batch mode connection used in the optimisation system, the results demonstrate that the power of Python based optimisation tools can be used with existing KBE applications.

Section 4.6 described some of the more challenging aspects of the development of the system. The solutions should mainly be considered workarounds, and thus they are not desired as the system is transitioned from the prototype implementation. Prototyping is itself the reason such workarounds were used, for instance running a virtual machine should be unnecessary, but for prototyping purposes, the benefit of being able to evaluate the performance of various algorithms efficiently is perceived to more advantageous than a clean runtime environment.

All the issues in Section 4.6 can be resolved without using workarounds, in fact, every issue, except the AML error messages, is related to open source projects, and the specific code relating to the issue has been pinpointed. Consequently, these issues are not insurmountable. Similarly, the issue relating to the AML error messages is expected to be easily solvable, and the reason for lacking a feature for turning off the messages can be traced to the fact that batch mode operation is not officially supported.

6.2 Applicability

With the limited number of cases that has been evaluated, it is hard to use these as an argument for how the system generalises to new design problems. Still, given the reliance on proven KBE and simulation tools, it can be expected that some of the generality is inherited. Since it is not commonly done, the primary new concept, which poses a challenge to the generality of the optimisation system, is the process of describing a design problem as an optimisation problem. The power and generality of optimisation suggest that this is possible, and indeed it was for the cases explored in this work, but the finesse and level of abstraction required to formulate such problems may prove to be a demanding task.

Evaluation of the results from FEDEM may be the hardest part of defining an optimisation problem for a given design problem. The evaluation of the results is required to work on all points in the design space, and in addition, the evaluation has a temporal component. Hence the level of abstraction required is higher than the normal design process, and thus the formulation of the evaluation procedure is more demanding than traditional analysis of numerical results.

Many, or even most, of the significant responses from FEDEM can be plotted. By thinking of these values as signals, and processing them in a fashion similar to the way control systems are defined, as described in Section 4.8, some of the complexity may be circumvented, as this way of thinking is already used in the design process. For all of the cases in Chapter 5, the evaluation procedures were defined with code, but the code still used the processing of time series as the primitives for expressing the procedure. Even though a textual representation of the evaluation procedure was used, a graphical representation of the evaluation procedure can be thought of as a different syntax for the same semantics. Thus the flexibility demonstrated by the use of this technique indicates that it may be applicable to new design problems.

The change of design parameters used in Section 5.3 can also be considered part of the redefinition of the design problem as an optimisation problem. When the change was introduced in Section 5.3, the initial purpose was to understand the reason for the jamming behaviour, with the intent of describing it as a constraint. Instead, it turned out that, once the behaviour was understood, the change of parameters with the resulting removal of the issue, was simple to implement. This change in design variables illustrates that the variables used to define a model are not necessary the best-suited design space. Further, the simplicity with which the change can be introduced, exemplify the benefits of using OpenMDAO for decomposing the problem.

Moreover the change of the driving parameters of the model bears some resemblance to how models are defined in parametric CAD systems, which means this way of thinking is used in the traditional design process. As a result, the introduction of a new set of design variables is a known technique, and the decoupling of the parameters used to define the model, and the parameters used to define the design space, can potentially be used to simplify the definition of optimisation problems. Admittedly the change of parameters in Section 5.3 is not very simple, due to the fact that the geometrical considerations have to hold for both all the points in the design space, and for all joint positions.

Section 5.2 compared some optimisation algorithms, but it should be noted that, through experience with all the cases, the results of the comparison of the algorithms does not fully form the basis for selecting the wider optimisation technique. Most of the optimisation examples in this work leverages the use of design of experiments and surrogate models. The rationale behind this is that all the time-consuming simulations can be done once, and after that, multiple formulations of the objective and constraints can be evaluated. Overall this approach allows for a more experimental and exploratory approach to formulating the optimisation problem, and as noted, it is this redefinition of the design problem that constitutes the main new task that needs to be accomplished to use optimisation in the design process.

In addition to enable a more iterative approach to formulating the optimisation problem, the use of surrogate models removes the need to specify suitable scaling factors, suitable numerical tolerances and suitable step size for the finite difference method. Such suitable values were found for the case in Section 5.1, but the amount of experimentation required, with the associated waiting time, indicates that it probably is difficult to apply this approach to new problems. With surrogate models, the selection of suitable values is bypassed, as the slower convergence rate associated with using high tolerances, and no scaling, does not cause excessive wall time costs. Sensitivities are also available, eliminating the need for finite difference approximations and the associated step size specification.

Other than the challenge related to defining the optimisation problem, the need for batch execution of the model strongly affects the applicability of the optimisation system. To explore the design space, either with an optimisation algorithm or through sampling, the system must be able to reliably execute and analyse the model. Through the use of exceptions in Python, the software can be made to handle crashes, and with the removal of the AML error messages, the software would be able to execute the model without requiring any manual intervention. Even without the software crashing, special care should be taken to ensure that the model does not fail unnecessarily, e.g. as seen in Section 5.3 before the blends were removed. When the model fails due to bugs, the view the optimisation algorithm has of the model is influenced by some effect that does not have any physical interpretation, and the results can not be expected to be acceptable. This issue also occurs if the formulation of the result evaluation procedure causes an exception for parts of the design space that should be valid. Thus the challenge of defining the optimisation problem is once again critical to the applicability of the optimisation system.

6.3 Usability

Ease of use concerns, for design engineers that would use the optimisation system, dictates that as much as possible of the optimisation problem definition process be wrapped in a graphical user interface. An important part of this interface is a tool similar to the control system editor, as described in Section 4.8. As discussed above, the treatment of the responses as signals has been verified by implementing the evaluation process using these primitives. Also mentioned was the fact that the current implementation uses textual code for the syntax, and it remains to be shown that the graphical syntax is usable. Specifically, the graphical view might prove to be too complex to have any decent usability. Similarly, it has not been shown that the graphical syntax can be used by design engineers, without requiring significant amounts of training.

One way these usability concerns could be reduced, is by the use of templates for use in the definition of the simulation evaluation procedures. However, this requires that the applicability of the templates be demonstrated, as there is a risk of conceiving templates that are specific to a very limited set of design problems. In the same way, templates for some parameter changes might increase the usability of the system. This would include e.g. the parameterisation that ensures the crank-rocker configuration, as in Section 5.3. Such templates would also be subject to applicability concerns.

The definition of the design space and any changes to the parameterisation of the model is perceived to be comparatively simple, mainly because such input is received through user interfaces in many existing engineering applications. Besides, the design space definition files are structured Yaml files; thus they can be easily generated from input in a user interface, and their generality is demonstrated by the fact that they have been used for all the cases in Chapter 5.

When the analysis procedures, for the cases previously presented, was developed, the first step was always to simulate the mechanism in some specific position. With the result from FEDEM loaded into data frames, the analysis procedure was created by iteratively composing the data frames in the interactive programming environment Jupyter. In Jupyter, which was introduced in Section 4.2.1, it is easy to visualise each partial result [1] by plotting the values. To some extent this simplifies the process of defining the analysis procedure, both since it makes the definition less abstract, as there is actual data to work with, and since it opens up the internals of the procedure, including the possibility of introspecting it. The use of actual visualisable data would, for these reasons, be useful in the user interface that is used for specifying the analysis procedure.

An alternative to simplifying the definition of the analysis procedure is to bypass it entirely. By building surrogate models for the all the responses of interest, and visualising them in a user interface, no formulation of the objective and constraints is necessary. The engineer could investigate the behaviour of the mechanism both by evaluating the SMs at specific points in the design space, and by looking at larger correlations, e.g. by plotting a design variable against a response. This would, when the SMs have been created, allow the engineer to test a new design in seconds, rather than minutes and hours. In some sense, this approach keeps the design evaluation procedure and optimisation algorithms in the engineer's mind, and the optimisation algorithm is connected to the model with the help of a user interface.

Second to the lack of a user interface, the usability of the system, in its current state, is limited by the time required to perform the simulations. Each of the samples in Figure 5.17 uses about 1.5 minutes, for all samples that constitute a total of about 6.5 hours, which makes time requirements of the system excessive. Since the sampling of the design space is embarrassingly parallelisable [24, p. 48], it is expected that the run time could be reduced by a factor equal to the number of parallel executions, even across multiple machines. Since the parallelisation is simple, this work has focused on reducing the uncertainty relating to other concerns, the only steps that are necessary to parallelise the execution is to ensure that the files, which are used to communicate with the external tools, are written to different locations on the file system. OpenMDAO even has some built-in support for this.

[1] A partial result would be e.g. a variable in the textual code in Section 4.8, or the output of a block in the graphical code.

CHAPTER

SEVEN

CONCLUSION

The prototype system developed in this project successfully integrates the AML based KBE system with Python based optimisation tools. This system has reliably performed thousands of mechanism design evaluations, and the evaluations have been used to improve designs.

Such improvements are obtained by using a selection of optimisation techniques. While both finite difference and a selection of optimisation algorithms were successfully applied, the use of surrogate models and design of experiments were found to be the preferred way of working. The use of SMs and DoE was not used to reduce the execution time, but selected since it alleviates the need for specifying numerical constants, and since it allows the specification of the optimisation problem to be iterative and interactive. Reductions in execution time are nevertheless expected to be easily obtainable, as the problem is embarrassingly parallelisable.

Reductions in the execution time would make the system more user-friendly. Likewise, the addition of a graphical user interface would increase the usability of the system. To this end, the prototype uses a serialisation format to define the design space. Since this format was used in all the cases, it is seen as general to many mechanisms. Furthermore, the structure makes it easy to generate from a graphical user interface.

For the specification of objectives and constraints a technique that treats the simulation responses as signals were successfully used for all cases. The signals were composed using a textual syntax, and a corresponding graphical syntax was shown to be similar to the syntax used to define control systems.

BIBLIOGRAPHY

[1] Reinier Van Dijk et al. "Multidisciplinary design and optimization of a plastic injection mold using an integrated design and engineering environment". In: *NAFEMS World Congress 2011*. NAFEMS ltd, 2011. URL: `http://resolver.tudelft.nl/uuid:26d5c229-c770-4080-82af-b12ae97a2a0b`.

[2] *Docker - Build, Ship, and Run Any App, Anywhere*. Docker Inc. URL: `https://www.docker.com/` (visited on 2017-05-15).

[3] Clark C. Evans. *The Official YAML Web Site*. URL: `http://yaml.org/` (visited on 2017-05-15).

[4] *Fedem*. Fedem AS. URL: `http://www.fedem.com/` (visited on 2016-10-31).

[5] *Fedem Release 7.1 Theory Guide*. Included with Fedem. Fedem AS, 2014.

[6] *Fedem User's Guide*. Included with Fedem. Fedem AS, 2014.

[7] Carlos Felippa. *NFEM Chapter 3: Residual Force Equations*. URL: `http://www.colorado.edu/engineering/CAS/courses.d/NFEM.d/NFEM.Ch03.d/NFEM.Ch03.pdf` (visited on 2016-10-03).

[8] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN: 0-201-63361-2.

[9] Philip E. Gill, Walter Murray, and Michael A. Saunders. "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization". In: *SIAM Review* 47.1 (2005-01), pp. 99–131. ISSN: 0036-1445. DOI: `10.1137/S0036144504446096`. URL: `http://epubs.siam.org/doi/abs/10.1137/S0036144504446096`.

[10] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

[11] John T Hwang. "A modular approach to large-scale design optimization of aerospace systems". PhD thesis. URL: `http://mdolab.engin.umich.edu/sites/default/files/Hwang_dissertation.pdf`.

[12] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001. URL: `http://www.scipy.org/` (visited on 2017-05-15).

[13] V. Roshan Joseph and Ying Hung. “Orthogonal-Maximin Latin Hypercube Designs”. In: *Statistica Sinica* 18.1 (2008), pp. 171–186. ISSN: 10170405. URL: http://www.jstor.org/stable/24308251.

[14] Dr. Gaetan Kenway. *pyOptSparse*. URL: https://bitbucket.org/mdolab/pyoptsparse (visited on 2016-10-31).

[15] Anders K Kristiansen and Eivind Kristoffersen. “Automating tasks in the design loop for mechanism design”. MA thesis. Norwegian University of Science and Technology, 2016.

[16] Arnt Lima. “TMM4540:Industrial ICT, Engineering Design and Materials, Specialization Project: Project Report, Theory and Algorithms for Structural Optimization”. Unpublished final year project report. 2016.

[17] Z. Lyu, Z. Xu, and J.R.R.A. Martins. “Benchmarking Optimization Algorithms for Wing Aerodynamic Design Optimization”. In: *The Eighth International Conference on Computational Fluid Dynamics* (2014).

[18] *Matplotlib: Python plotting*. Matplotlib development team. URL: http://matplotlib.org/index.html (visited on 2017-05-15).

[19] Arvid Myklebust. “Synthesis of Multi-Link Mechanisms for Dynamic Specifications”. PhD thesis. University of Florida, 1974.

[20] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. 2nd. Springer Science+Business Media, LLC, 2006. ISBN: 0-387-30303-0.

[21] *NumPy*. NumPy developers. URL: http://www.numpy.org/ (visited on 2017-05-15).

[22] Travis E. Oliphant. “Python for Scientific Computing”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20. DOI: 10.1109/MCSE.2007.58. eprint: http://aip.scitation.org/doi/pdf/10.1109/MCSE.2007.58. URL: http://aip.scitation.org/doi/abs/10.1109/MCSE.2007.58.

[23] *OpenMDAO*. URL: http://openmdao.org (visited on 2016-08-25).

[24] Peter Pacheco. *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011. ISBN: 978-0-12-374260-5.

[25] Fernando Pérez and Brian E. Granger. “[IP]ython: a System for Interactive Scientific Computing”. In: *Computing in Science and Engineering* 9.3 (2007-05), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: http://ipython.org.

[26] Ruben E. Perez, Peter W. Jansen, and Joaquim R. R. A. Martins. “pyOpt: a Python-based object-oriented framework for nonlinear constrained optimization”. In: *Structural and Multidisciplinary Optimization* 45.1 (2012-01), pp. 101–118. ISSN: 1615-147X. DOI: 10.1007/s00158-011-0666-3. URL: http://link.springer.com/10.1007/s00158-011-0666-3.

[27] *Project Jupyter*. Project Jupyter. URL: https://jupyter.org/ (visited on 2017-05-15).

[28] *Python for Windows Extensions*. URL: https://sourceforge.net/projects/pywin32/ (visited on 2017-05-15).

[29] Gianfranco La Rocca. “Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design”. In: *Advanced*

Engineering Informatics 26.2 (2012), pp. 159–179. DOI: `10.1016/j.aei.2012.02.002`. URL: `http://dx.doi.org/10.1016/j.aei.2012.02.002`.

[30] Susana Rojas-Labanda and Mathias Stolpe. "Benchmarking optimization solvers for structural topology optimization". In: *Struct Multidisc Optim* 52 (2015), pp. 527–547. DOI: `10.1007/s00158-015-1250-z`.

[31] *RPyC - Transparent, Symmetric Distributed Computing*. URL: `http://rpyc.readthedocs.io/en/latest/` (visited on 2017-05-15).

[32] Ole Ivar Sivertsen. *Virual Testing of Mechancal Systems*. Ed. by Fai Ma and Edwin Kreuzer. Lisse: Swets & Zeitlinger, 2001. ISBN: 90-265-1811-0.

[33] Jaroslaw Sobieszczanski-Sobieski, Alan Morris, and Michel van Tooren. *Multidisciplinary Design Optimization supported by Knowlage Based Engineering*. John Wiley & Sons Ltd., 2015. ISBN: 978-1-118-49212-3.

[34] Ian Sommerville. *Software Engineering*. 9th ed. Harlow, England: Addison-Wesley, 2010. ISBN: 978-0-13-703515-1.

[35] Sigurd D Trier. "Design Optimization of Flexible Multibody Systems". PhD thesis. Norwegian University of Science and Technology, 2001.

[36] Andreas Wächter and Lorenz T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical Programming* 106.1 (2006), pp. 25–57. ISSN: 1436-4646. DOI: `10.1007/s10107-004-0559-y`. URL: `http://dx.doi.org/10.1007/s10107-004-0559-y`.

[37] Michael Waskom. *Seaborn: statistical data visualization*. URL: `https://seaborn.pydata.org/` (visited on 2017-05-15).

[38] Michael Waskom et al. *seaborn: v0.5.0 (November 2014)*. 2014-11. DOI: `10.5281/zenodo.12710`. URL: `https://doi.org/10.5281/zenodo.12710`.

[39] *Welcome to Bokeh*. Continuum Analytics. URL: `http://bokeh.pydata.org/en/latest/` (visited on 2017-05-15).

[40] Eric Xu. *A Python connector to IPOPT*. Washington Univerisity. URL: `https://github.com/xuy/pyipopt` (visited on 2017-05-15).

Appendices

APPENDIX

A

CASE 1 FILES

The listings in this appendix contains both the Yaml used to defied the design space for the mechanism used in Section 5.1 and Section 5.2 and the corresponding analysis procedure.

A.1 Design Space

```
points:
    - &p0
      name: "Crank-bearing"
      pos: [0.0, 0.0, 0.0]

    - &p1
      name: "Crank-top"
      pos: [0.0, 0.075, 0.0]

    - &p2
      name: "Rocker-bearing"
      pos:
          default: [0.15, 0.0, 0.0]
          min: [0.05, -0.10, _]
          max: [0.25, 0.10, _]

    - &p3
      name: "Rocker-top"
    pos:
        [0.15, 0.1875, 0.0]

  - &p4
    name: "Coupler-end"
    pos: [0.3, 0.3, 0.0]


links:
  - &l0
    members:
      - name: Crank
        cross-section: rectangular
        dimensions: [0.03, 0.03]

  - &l1
    members:
      - name: Coupler lower
        cross-section: rectangular
        dimensions: [0.01, 0.01]

      - name: Coupler invis
        cross-section: Abstract
        dimensions: [0.01, 0.01]

      - name: Coupler upper
        cross-section: rectangular
        dimensions: [0.013, 0.013]

  - &l2
    members:
      - name: Rocker
        cross-section: rectangular
        dimensions: [0.01, 0.01]



constraints:
  - type: revolute
    point: *p0
    links: [null, *l0]
    joint-direction: [0.0, 0.0, 1.0]

  - type: revolute
    point: *p1
```

```
        links: [*l1, *l0]
        joint-direction: [0.0, 0.0, 1.0]

      - type: revolute
        point: *p2
        links: [null, *l2]
        joint-direction: [0.0, 0.0, 1.0]

      - type: revolute
        point: *p3
        links: [*l1, *l2]
        joint-direction: [0.0, 0.0, 1.0]

      - type: free
        point: *p4
        links: [null, *l1]
        joint-direction: [0.0, 0.0, 1.0]
        fixed-dofs: []

loads:
  - type: Torque
    point: *p0
    direction: [0.0, 0.0, -1.0]
    magnitude: -5
    loaded-link: *l0

node-coupling:
  - type: damper
    points: [*p1, *p2]
    constant: 1500.0
    links: [*l1, *l2]
```

A.2 Analysis

```
y_result = fedem.get_res_on_bid(9, 'Position␣matrix:TMAT34:
    ↪ Position␣Y', 'y')
x_result = fedem.get_res_on_bid(9, 'Position␣matrix:TMAT34:
    ↪ Position␣X', 'x')
result = pd.concat([x_result, y_result], axis=1)
result.columns = ['x', 'y']
result = result.loc[:5.0]

start = np.array([0.3, 0.3])
```

```
end = np.array([0.0, 0.3])

time_at_point = lambda path, point: path.apply(lambda df:
    ↪ np.linalg.norm(point - df), axis=1).idxmin()

t_start = time_at_point(result, start)
t_end = time_at_point(result, end)
path_of_interest = result.loc[t_start: t_end]

# reparametrize the path in tau=0...1
step_ends = path_of_interest.iloc[1:]
step_starts = path_of_interest.iloc[:-1]
step_starts.index = step_ends.index
step_lengths_comps = step_ends - step_starts

step_lengths = step_lengths_comps.apply(lambda df: np.
    ↪ linalg.norm(df), axis=1)
path_length = step_lengths.sum()
relative_length = (step_lengths.cumsum() / path_length).
    ↪ to_frame()

relative_length = pd.DataFrame([0.0], index=[0.0]).append(
    ↪ relative_length)
relative_length.columns = ['tau']
relative_length.reset_index(inplace=True)
relative_length.set_index('tau', inplace=True)
relative_length.columns = ['time']

psudo_time_step_sizes = relative_length

reference_path = lambda tau: start + (end - start) * tau

deviation = pd.merge(psudo_time_step_sizes.reset_index(),
    ↪ result.reset_index(), on='time')
deviation['ref_path_x'] = deviation.apply(lambda df:
    ↪ reference_path(df['tau'])[0], axis=1)
deviation['ref_path_y'] = deviation.apply(lambda df:
    ↪ reference_path(df['tau'])[1], axis=1)
deviation['distance'] = deviation.apply(lambda df: np.
    ↪ linalg.norm(df[['x', 'y']] - reference_path(df['tau'
    ↪ ])),
                                        axis=1)

deviation.set_index('time', inplace=True)
```

```
deviation['violation'] = deviation.apply(lambda df: min(0,
    ↪ df['y'] - reference_path(df['tau'])[1]), axis=1)
violation_constraint = np.trapz(deviation['violation'], x=
    ↪ deviation['tau'])

obj = np.trapz(deviation['distance'], x=deviation['tau'])

unknowns['res'] = obj
unknowns['violation'] = violation_constraint
```

APPENDIX

B

CASE 3 FILES

The listings in this appendix contains both the Yaml used to define the design space for the mechanism used in Section 5.3.3, and the code for the analysis. The Yaml file forms the base model which was used after the new parameters were calculated in the latter parts of Section 5.3.

B.1 Design Space

Note that this Yaml does not contain any load, or other means to drive the mechanism, as the KBE system used does not currently support the technique used to drive the mechanism. Instead the tools described in Section 4.3.6 are used. Specifically torsion spring is connected to the crank, and this spring drives the mechanism, by means of a enforced angular speed.

```
points:
    - &p0
      name: "P0"
      pos: [0.2578, 0.01, 0.0]

    - &p1
      name: "P1"
      pos: [0.2661, 0.6337, -0.06]

    - &p2
      name: "P2"
      pos: [0.11, 0.6864, 0.0 ]
    - &p3
      name: "P3"
      pos: [0.1085, 0.4935, −0.06]


    - &p4
      name: "P4"
      pos:
        min: [0.4, 0.05, _]
        default: [0.4849, 0.1204, −0.06]
        max: [0.6, 0.2, _]

    - &p5
      name: "P5"

      pos: [0.3959, −0.1262, 0.0]

    - &p6
      name: "P6"

      pos: [0.7925, 0.2743, 0.0]


defaults:
    - circ_member: &member_defaults
        cross−section: rectangular
        dimensions: [0.04, 0.04]


links:
    #FORURBAR members:
    - &l_crank
      members:
        - name: Crank
          <<: *member_defaults
    - &l_coupler
      members:
        - name: Coupler lower
          <<: *member_defaults

        - name: Coupler invis
          <<: *member_defaults
        - name: Coupler upper
          <<: *member_defaults
          cross-section: Abstract

    - &l_rocker
      members:
        - name: Rocker
          <<: *member_defaults

    #ADDITIONAL members
    - &l_diad0
      members:
        - name: m1
          <<: *member_defaults


    - &l_diad1
      members:
        - name: m2
          <<: *member_defaults

constraints:
    - type: revolute
      point: *p0
      links: [null, *l_cranck]
      joint-direction: [0.0, 0.0, 1.0]

    - type: revolute
      point: *p1
      links: [null, *l_diad0]
      joint-direction: [0.0, 0.0, 1.0]

    - type: revolute
      point: *p2
      links: [*l_diad0, *l_diad1]
      joint-direction: [0.0, 0.0, 1.0]

    - type: revolute
      point: *p3
      links: [*l_rocker, null]
      joint-direction: [0.0, 0.0, 1.0]

    - type: revolute
      point: *p4
      links: [*l_rocker, *l_coupler]
```

```
        joint-direction: [0.0, 0.0, 1.0]

      - type: revolute
        point: *p5
        links: [*l_coupler, *l_cranck]
        joint-direction: [0.0, 0.0, 1.0]

      - type: revolute
        point: *p6
        links: [*l_coupler, *l_diad1]
        joint-direction: [0.0, 0.0, 1.0]


loads: []


node-coupling: []
```

B.2 Analysis

```
sim_offset, sim_length = 3.1, 1.2
x_end, y_end = 0.7925, 0.274
x_start, y_start = 0.457, -0.305
phi_offset = np.pi/4
coupler_path = pd.concat([fedem.get_res_on_bid(9, 'Position
    ↪ ␣matrix:TMAT34:Position␣X', 'coupler'),
                           fedem.get_res_on_bid(9, 'Position
                               ↪ ␣matrix:TMAT34:Position␣Y',
                               ↪   'coupler')],
                          axis=1)
coupler_path = coupler_path.loc[sim_offset:sim_offset +
    ↪ sim_length]
coupler_path.columns = ['x', 'y']
time_at_point = lambda path, point: path.apply(lambda df:
    ↪ np.linalg.norm(point - df), axis=1).idxmin()
t_start = time_at_point(coupler_path[['x', 'y']], np.array
    ↪ ([x_start, y_start]))
t_end = time_at_point(coupler_path[['x', 'y']], np.array([
    ↪ x_end, y_end]))

coupler_path_high_order = [
    fedem.get_res_on_bid(9, 'Velocity:VEC3:X', 'coupler'),
    fedem.get_res_on_bid(9, 'Velocity:VEC3:Y', 'coupler'),
    fedem.get_res_on_bid(9, 'Acceleration:VEC3:X', 'coupler
        ↪ '),
    fedem.get_res_on_bid(9, 'Acceleration:VEC3:Y', 'coupler
        ↪ '),
    fedem.get_res_on_bid(9, 'Position␣matrix:TMAT34:Euler␣
        ↪ Angle␣ZYX␣Z', 'coupler'),
    fedem.get_res_on_bid(18, 'Angular␣velocity:ROT3:Z', '
        ↪ coupler')
]
path_derivs = pd.concat(coupler_path_high_order, axis=1)
path_derivs.columns = ['v_x', 'v_y', 'a_x', 'a_y', 'phi', '
    ↪ phi_dot']

return {'t_p': t_start,
        'x_p': coupler_path.loc[[t_start]]['x'],
        'y_p': coupler_path.loc[[t_start]]['y'],
        'v_x_p': path_derivs.loc[[t_start]]['v_x'],
        'v_y_p': path_derivs.loc[[t_start]]['v_y'],
        't_d': t_end,
        'x_d': coupler_path.loc[[t_end]]['x'],
        'y_d': coupler_path.loc[[t_end]]['y'],
        'v_x_d': path_derivs.loc[[t_end]]['v_x'],
        'v_y_d': path_derivs.loc[[t_end]]['v_y'],
        'a_x_d': path_derivs.loc[[t_end]]['a_x'],
        'a_y_d': path_derivs.loc[[t_end]]['a_y'],
        'phi_p': path_derivs.loc[[t_start]]['phi'],
        'phi_d': path_derivs.loc[[t_start]]['phi'],
        'phi_dot_p':  path_derivs.loc[[t_start]]['phi_dot'],
        'phi_dot_d':  path_derivs.loc[[t_end]]['phi_dot']
         }
```