Reducing the environmental cost of concrete constructions

Master Thesis Mathematical Sciences

Nynke Brouwer

First reader:
Dr. I. Kryven

Second reader:
Dr. T. van Leeuwen

Daily supervisors:
W. Sponselee
Dr. F. Roumen

Faculty of Science
Utrecht University
October, 2021

Abstract

Environmental sustainability is seldomly considered at the design phase of concrete structures, which mainly concerns optimising the structural mechanics properties. In this project we propose and implement a hybrid optimization protocol that solves the structural mechanics optimisation along with minimization of environmental costs. These hybrid method consists of two optimization phases: the first gives a fast and crude estimation of the concrete resistance based on a data assimilation technique, the second refines the results with more expensive predictive modelling. After comparing our protocol against various existing optimization methods we conclude that our hybrid optimization leads to more sustainable designs without a compromise on mechanical properties. Surprisingly, benchmarking our technique on several real world datasets revealed that having data-assimilated preconditioner not only improves environmental costs but also leads to designs with better mechanical properties in less time.

Acknowledgements

This research was part of an internship at the Dutch engineering company Movares Nederland B.V.. I would like to thank Wilco Sponselee and Frank Roumen for the opportunity to combine this thesis with an internship, for their support and their helpful feedback during the project. Moreover, I thank the whole UC1 team for making me feel part of the team and for their useful comments on my coding. Last but not least, I want to thank Ivan Kryven from the Mathematics department at Utrecht University for the guidance and his encouraging words.

Contents

Chapter 1

Introduction

When designing a concrete structure, it is important that the structure will not collapse under the load. Civil engineers are doing calculations to make sure of that. Doing these calculations can be a time-consuming activity, especially when one has to design tens or hundreds of constructions. Thus it is valuable to automate the design process. The Dutch company Movares developed UC1-Concrete [1], software that supports the design of concrete structures.

To ensure safety of the design we must consider the loading and the strength of the structure. Therefore we compute the expected load on the structure and the resistance of the design. If too heavy load is put on a weak design, the structure will fail. Apart from safety, there is another aspect a designer should keep in mind, one that is getting more and more important: the environmental impact. With climate change creeping up on us, it is becoming an urgent matter that we keep our (damaging) impact on nature as small as possible.

In this project, we are interested in finding the best design possible. One might wish for a design that minimizes material usage, a design that minimizes the cost or one that minimizes the environmental impact. This brings rise to the following question: can we find the optimal design for a cross-section of reinforced concrete? Our goal is to optimize the environmental cost of a concrete design. The project is divided into two parts. One part is a search for the best optimization method. The other part is implementing the optimization functionality in UC1. In the (re)search for a good optimization method we limit ourselves to the design of surfaces of reinforced concrete with a rectangular cross-section. In UC1 it is also possible to optimize the design for members and for other polygon shaped cross-sections.

[1] `https://movares.nl/diensten/optimaal-betonontwerp-met-uc1-concrete/`

The optimization methods are applied to two distinct optimization problems:

(i) The first attempts to mimic the procedure that an engineer follows when trying to find a good design by hand. To this end we use the design margins, which are a measurement of the difference between the load and the resistance.

(ii) The second problem is the minimization of the environmental cost indicator (ECI). The ECI is the environmental impact of a construction translated into a *shadow price* in euros [19].

As a starting point, the optimization functionality is implemented in UC1 using the MIDACO solver [2], from that we try to find a better method to ultimately use in UC1.

1.0.1 Literature review

Before we start diving into this project, we give a short summary of some of the studies done in mathematical optimization of concrete structures. There exist lots of theoretical knowledge on structural optimization, but there seems to be an imbalance with its practical applications. Most engineers still search for the best design in a trial-and-error manner where they tune the design variables by using their experience and knowledge. This will lead to a sufficient design, but optimality cannot be guaranteed. A review of the application of optimization methods in structural engineering is given by Cohn and Dinovitzer in 1994 [9], the authors demonstrate the practical benefits of applying mathematical optimization in civil engineering.

Most of the researches in structural optimization concern the optimization of the financial cost. The optimization of environmental impact of a reinforced concrete structure is done by Paya-Zaforteza et al [29], minimizing the CO2 emissions by simulated annealing. Other studies concerning the optimization of environmental sustainability can be found in [10, 28, 38]. A practical concern is that the design parameters cannot be any real-valued number. Often this is solved by optimizing with continuous values and then rounding the obtained solution. The implementation of sequential squared programming algorithm to optimize the cost with continuous valued design variables is given in [16]. In 1998, Rajeev and Krishnamoorthy introduced a genetic algorithm that generates practically feasible solutions [33]. Optimization of reinforced concrete with discrete-valued design variables by genetic algorithms are given in [8, 23]

1.0.2 Reading guide

The next part of the introduction covers the basic principles of structural mechanics in concrete that are necessary for the thesis. Readers with prior knowledge of this can move

[2] `http://www.midaco-solver.com/`

on to chapter 2. Chapter 2 introduces the optimization methods that are used in chapter 3 and 4 of this project. The first section is a review of several existing methods. Section 2 of the chapter is a description of the hybrid methods, starting with how to generate a test set and then the explanation of various data assimilation techniques for the estimation of the resistance. In the third chapter we discuss the optimization based on the design margins and apply some of the methods from chapter 2 to this problem. In chapter 4 we go to the ECI optimization and look at the performance of the optimization methods on this problem. In the end we state our conclusions, try to translate the results into practical conclusions and state some points that are still open for future research.

1.1 Computations in concrete

To test the safety of a design, we compare the expected load and the resistance of the design. The resistance of the design can be described with the Ultimate Limit State (ULS) diagram. In this section we explain the derivation of this ULS diagram and the measures that we use for optimization: the design margins and environmental cost indicator.

Effect of action on a concrete surface can cause the surface to bend, this is shown in figure 1.1. Concrete can handle compression very well, but only a little tension. For simplicity, following the safety standards in the Eurocode [3], it is assumed that concrete can handle no tension at all. Therefore the concrete is reinforced by bars of steel. This combination of concrete and steel we call reinforced concrete. A cross-section of a reinforced concrete surface is shown in figure 1.2.

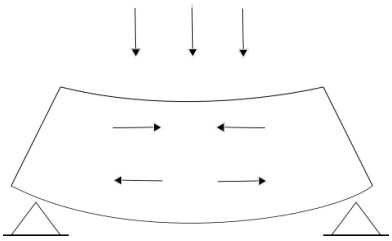


Figure 1.1: Load on a concrete structure that causes the surface to bend.

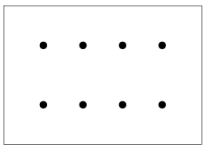


Figure 1.2: The cross-section of a reinforced concrete surface.

[3] the European guidelines: NEN-EN 1992-1-1+C2:2011 and the dutch national annex: NEN-EN 1992-1-1+C2:2011/NB+A1:2020

1.1.1 Resistance

How much a surface can bend until failure depends on its resistance against cracking. Here the point of failure of a surface is a fixed theoretical value rather than a physical state in practice. During the bending, strains occur in the cross-section of the design. This strain across the cross-section is represented in a strain distribution, which is assumed to be linear. Strain ε is measured per mille (‰), negative strain is compression and positive strain is tension. The curvature κ describes the slope of the strain and is typically measured in m^{-1}. An example of a strain distribution is given in Figure 1.3, to the right of the height axis is negative strain, to the left is positive strain. For both steel and concrete an ultimate value of strain that can be resisted is given. One ULS is described by a strain distribution that reaches one of these ultimate values. Together this results in an infinite amount of strain distributions that describe the ultimate limit states. To compare this with the load, this is derived into a combination of normal force and bending moment.

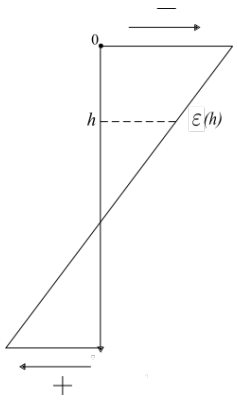


Figure 1.3: Strain distribution for a concrete cross-section.

To compute the forces in the cross section, we need to know the stresses, and how they relate to the strains. Stress (or pressure) σ is measured in MPa and every material has a given stress-strain relation. Figure 1.4 shows three possible stress-strain curves for concrete: the parabola-rectangle diagram, the bilinear diagram and the parabolic diagram respectively, the latter we will not treat in this project. Since concrete can only handle compression, the strain values on the horizontal axis are negative. In figure 1.5 we see two possible stress-strain diagrams for steel, both bilinear diagrams with different slopes. By combining the strain distribution and the stress-strain diagrams, we can calculate the stress across the cross-section of the surface for both concrete and steel. Stress on concrete with positive strain is equal to zero. A stress distribution for concrete and steel and how they are obtained are shown in Figure 1.6. Steel can handle both tension and compression and therefore has both positive and negative stress. One computes the strain at the height of a reinforcement layer and computes its corresponding stress value.

We can calculate the normal force on the cross-section with the stress distribution. The total normal force on the cross-section of the reinforced concrete is the normal force on concrete plus the normal force on steel. For material with uniformly distributed strain, the normal force is given by $N = \sigma(\varepsilon)A$, where A is the surface area. The stress on one bar of steel is uniformly distributed, so we can use the former formula for the separate layers.

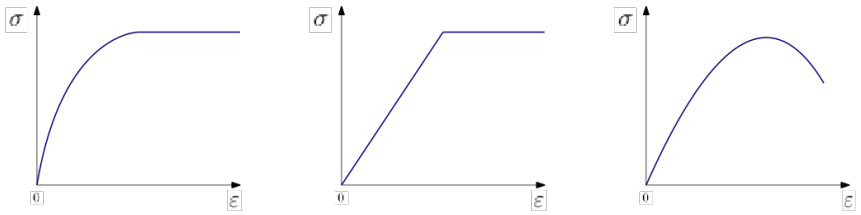


Figure 1.4: Possible stress-strain diagrams for concrete.

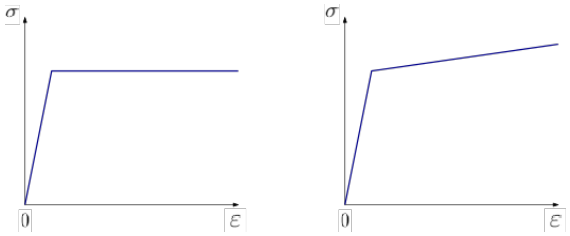


Figure 1.5: Possible stress-strain diagrams for steel.

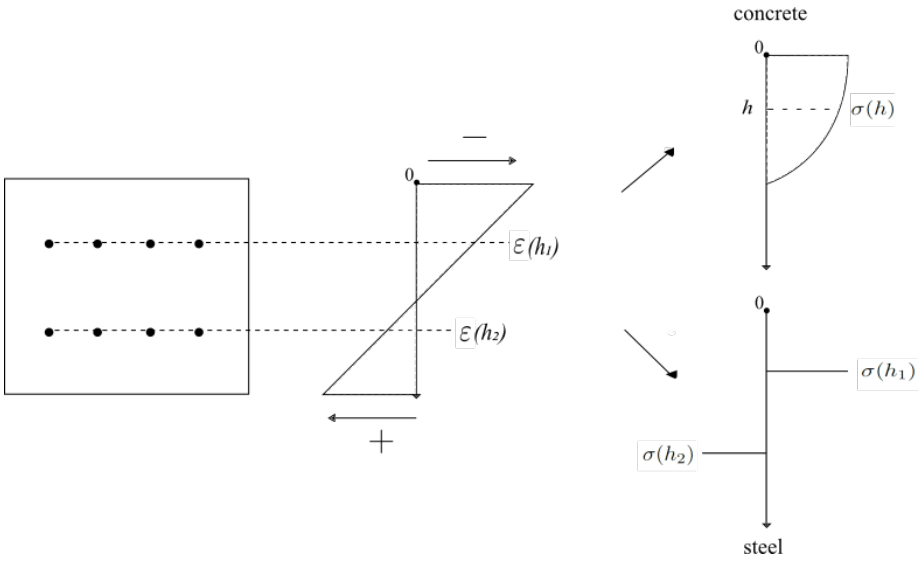


Figure 1.6: Stress distribution for concrete and steel.

Stress on concrete is not uniformly distributed as we can see in its stress distribution in figure 1.6, so we have to integrate over the cross-section. The last thing to compute is the bending moment on the cross-section. This is the sum of Nz for every material in the cross-section with uniform strain with an integral over the cross-section for the material without uniform strain, here z is the lever arm.

To summarize, we want to solve the following equations:

$$\begin{aligned} N_{\mathrm{s}} &= \frac{w}{s_1} \cdot \pi \left(\frac{d_1}{2}\right)^2 \sigma(h_1) + \frac{w}{s_2} \cdot \pi \left(\frac{d_2}{2}\right)^2 \sigma(h_2) \\ N_{\mathrm{c}} &= \int_0^{h_{\mathrm{bottom}}} w(h)\sigma(h)dh \\ N_{\mathrm{Rd}} &= N_{\mathrm{s}} + N_{\mathrm{c}} \\ M_{\mathrm{Rd}} &= M_{\mathrm{s}} + M_{\mathrm{c}} = \sum_b N_{b,\mathrm{s}} z_{b,\mathrm{s}} + \int_0^{h_{\mathrm{bottom}}} w(z)\sigma(z)(z - z_0)dz, \end{aligned} \tag{1.1}$$

where s_1 and s_2 are the spacing of the top layer and the spacing of the bottom layer, z_0 is the median, $w(h)$ is the width at height h and w is the width of the cross-section, so $\frac{w}{s_1}$ is the number of bars in the first layer. The bending moment on steel (M_{s}) is calculated by summing over all bars b. The subscript Rd stands for resistance of the design. Note that the height at the top is equal to zero and the height increases when going downwards, as is shown in the stress and strain distributions in figure 1.6.

Ultimate Limit State

How do we compute the ultimate limit state (ULS)? The ULS diagram gives the resistance of the design, in bending moment and normal force. The resistance is denoted as $M_{\mathrm{Rd}}, N_{\mathrm{Rd}}$. We can make a diagram in the M, N-parametric space of the set of all points of M, N for which the structure does not fail, the boundary of this set is called the *onion*. To compute this, we consider the failure limits in strain. Every type of material has its own failure limit, so the failure limits for the surface depend on the type of concrete and the type of steel that we are using. The failure limit of concrete is determined by the concrete strength f_{ck} and the failure limit of steel by the steel strength f_{yk}. Given the failure limits, we can find the strain distribution for every possible ultimate limit state and calculate the corresponding normal force and bending moment. Plotting these $M_{\mathrm{Rd}}, N_{\mathrm{Rd}}$ values gives us the onion shaped polygon as in figure 1.7, which we also call the M, N-diagram.

Natural uncertainty in the material strength and the load makes it hard to ensure a 100% safeness with the given strengths, so it is important to find a design that has a very small risk of failure. To minimize the effect of the uncertainty, partial factors are used to get the characteristic strength/resistance of a material and the characteristic load/effect. Given

some percentage, the characteristic strength is the value of resistance for which not more than the percentage of test results of the material is expected to fall below it [24]. Usually this percentage is 5%. This yields the characteristic design resistance $R_d = R/\gamma_R$, with γ_R being the partial factor for the resistance. This partial factor is different for both materials, but it is always larger than 1. Giving us the characteristic resistance for concrete $f_{cd} = f_{ck}/\gamma_c$ and steel $f_{yd} = f_{yk}/\gamma_s$. This happens in the calculations of the onion.

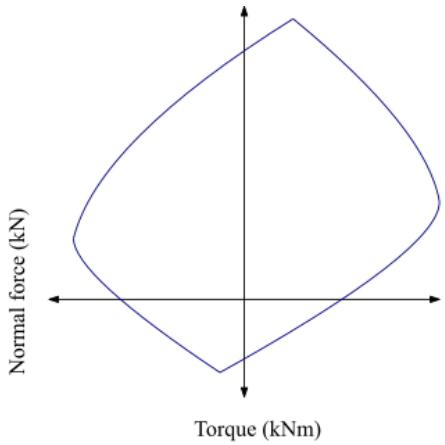


Figure 1.7: ULS figure for a given reinforced concrete cross-section.

1.1.2 Load

The ultimate limit state will be compared to the expected occurring forces. These internal forces are given as a list of points of $M_{\text{Ed}}, N_{\text{Ed}}$, called FEM points, where Ed stands for effect on the design. In short, the FEM points are obtained by putting the expected load on the structure, represented by PDEs, and approximating this load with discrete FEM points [18]. The expected forces are subject to the geometry of the construction, which means that if we adjust the width or height of the surface, the expected load will change. For that reason we fix the geometry and thus the size of the cross-section in this project. Otherwise the FEM points could differ during the optimization, making the problem much harder. The FEM points are the internal forces on a finite number of locations on the surface. As mentioned above, natural uncertainties occur also in the load. Therefore we apply a partial factor on the effects of action. The opposite of the above holds for the effect, namely the characteristic effect is the value of load for which not more than a percentage of test results is expected to fall *above* it, resulting in the effect $E_d = E * \gamma_E$, where γ_E is the partial factor for the effect.

If the forces are stronger than the resistance of the cross-section, the structure will fail, thus we want the FEM points to be smaller than the resistance and hence inside the onion.

This can be checked with design margins, which measure the distance between the points and the onion.

1.1.3 Design margins

The computations of the design margins are based on the safety standards in the Eurocode. The design margin of a point is a positive real number and is

- smaller than 1 if the point lies inside the M, N-diagram,
- equal to 1 if the point lies exactly on the M, N-diagram and
- larger than 1 if the point lies outside the M, N-diagram.

Hence it is important to have every design margin smaller than or equal to 1. This protocol is called the *Unity Check* and we denote a design margin by *uc*.

The computation of the design margin of a FEM point is similar to the L^∞-norm [40], in the way that we first compute the uc_M and uc_N separately and then take either the maximum or the minimum. Lets start with the computation of uc_N. We compare the effect on the design in normal force with the resistance of the design by drawing a vertical line through the FEM point and taking either the lower or upper intersection point of the M, N-diagram and this line, this is the resistance in normal force N_{Rd}. When N_{Ed} is above the horizontal axis we take the upper intersection, if it is below the axis we take the lower intersection. An example is shown in figure 1.8. Then we divide the normal force of the FEM point by the normal force of the resistance:

$$uc_N = \frac{N_{\text{Ed}}}{N_{\text{Rd}}}, \tag{1.2}$$

observe that this value is strictly larger than 1 if the normal force of the FEM point is larger than the normal force of the resistance and smaller or equal otherwise.

For the calculation of uc_M we use not the vertical axis as a base but the *zero-curvature line*. This line consists of all the points of M, N for which the curvature of the strain diagram is equal to zero. Draw a horizontal line through the FEM point, the intersection of the zero-curvature line and this line is M_{κ_0} and the closest of the two intersections of the onion and this line is M_{Ed}. Then the design margin for the bending moment is given by

$$uc_M = \frac{M_{\text{Ed}} - M_{\kappa_0}}{M_{\text{Rd}} - M_{\kappa_0}}. \tag{1.3}$$

If both uc_M and uc_N are more than 1, then $uc = \min(uc_M, uc_N)$, otherwise $uc = \max(uc_M, uc_N)$.

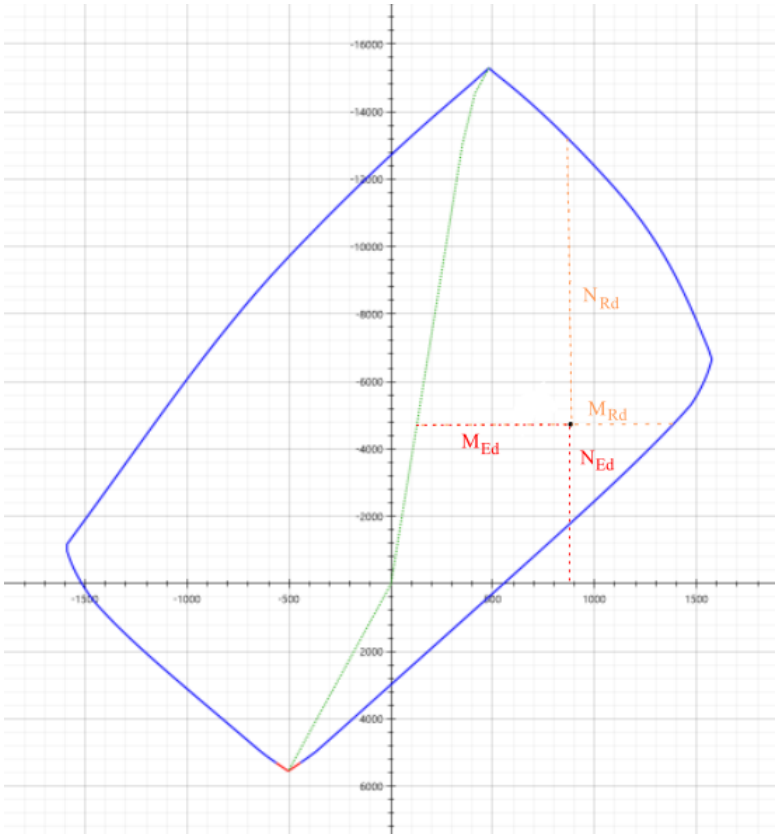


Figure 1.8: A FEM point and the corresponding resistance forces.

1.1.4 Environmental cost indicator

The last ingredient we need is the computation of the ECI. Recall that the ECI is a representation of the environmental impact of a product (or project) in euros. This is a rather new measure to define the quality of a product and is only recently getting used in the engineering industry. When measuring the environmental impact of a product the total supply chain plays a role. For example, a few things that affect the sustainability are the material usage, the transportation and the waste disposal. To determine the total cost indicator, a life cycle assessment can be applied, this analysis is also called *cradle-to-grave*. In such an analysis, every step in the life cycle of a product is being analysed. The computation of the cost is done with weighting factors per category (for example: the depletion of the ozone layer), which indicate what the cost is for a certain level of impact.

For steel we are given an ECI in euros per tonne. For concrete it is measured in euros per cubic meter and there is a different ECI for every value of concrete strength. A

higher concrete strength results in a higher ECI. It follows that if we know the concrete strength, the total size of the reinforced concrete and the volume of the reinforcement, we can calculate the total ECI for the design.

Chapter 2

Optimization methods

The goal of this chapter is to give an overview of the methods that are used in this project. At first we apply multiple existing optimizers to our problems, so we start this chapter with a review of existing optimization methods. After that, we show the methods that we developed, these are all hybrid optimization methods that use a fast estimation of the M, N-diagram. These sections mostly focus on the data assimilation techniques in the first phase of the hybrid procedure. We explain the methods, tune their parameters and compare them.

2.1 Review of existing methods

To find the best method for our optimization problem, we study different sorts of existing optimization methods and in this section we briefly go over some of them. The MIDACO solver that is currently used is shown to be a good solver for mixed integer non-linear programming (MINLP) problems in a numerical study by Schlüter et al [36]. We will later see that our problem does not require variables to be integer, so this functionality is unnecessary.

A popular class of methods are the Newtonian methods, which thank their popularity to their fast convergence. The disadvantage is that Newtonian methods need the gradient upfront, we will see later that our problem lacks any information on the gradient. A summary of Newtons method and its convergence results is given by Polyak [30]. Direct search or pattern search approaches do not require any derivatives and can even work with black-box functions. These methods exist since the 1950s, but they disappeared from the literature because of a lack of mathematical analysis. They are now widely used mainly because of their applicability. Kolda et al [21] provide a review on direct search methods for optimization, in which the authors describe direct search methods and proof convergence for one broad class of direct search methods on continuously differentiable functions.

Powell [32] provides a view on optimization without derivatives, in particular on simplex methods, pattern search methods and quadratic models. Some tricks to ensure convergence of pattern search methods are briefly mentioned, but in the article of Kolda et al [21] this is explained more extensively. Powell also describes a quadratic model that uses interpolation and some numerical results shows its efficiency. The three methods are interesting because of their applicability, but they seem to lack a convergence guarantee. Yuan [41] gives a review on trust region methods, which are useful for ill-conditioned optimization problems. The author gives convergence properties that hold when the objective is differentiable, shows that these methods have linear convergence and gives a way to obtain superlinear convergence. An example of a trust region method is the simplex method COBYLA that is invented by Powell [31].

Another type of algorithm that can be applied when we miss information on the gradient are quasi-Newton methods. Such algorithms can be particularly convenient when the behaviour of the loss function leads us to expect that the Hessian does exist. The quasi-Newton methods that are probably best known are those of Broyden [5]. His method is based on Newton's method, but it approximates the Hessian and can therefore be used for problems that lack a proper definition of the Hessian. These quasi methods can be even faster than real Newtonian methods, since Newtonian methods require the calculation of the inverse of the hessian, where quasi-Newton methods directly approximate the inverse. Quasi-Newton methods have superlinear convergence. An overview of the earlier quasi-Newton methods is given by Dennis and Moré [11], as well as motivation and convergence results. In a recent paper of Berahas et al [3], the authors give a finite difference quasi-Newton method for the optimization without derivatives that converges to the neighborhood of the solution.

2.2 Hybrid optimization

In this section we explain the new methods that we try for optimizing the design of concrete constructions. Our algorithms are hybrid algorithms, which means that we divide the optimization process into two phases. The motivation behind this is that having a good initial guess benefits the search. The initial guess can have great influence on the quality of the solution, e.g. which local minimum we end up in, and the number of function evaluations required.

Every time the loss function is evaluated, we have to construct the onion. The onion is built up by tabulating many points of a parametric curve. The resolution of the onion is 400 by default, which means that we have to solve the equations (1.1) for M_{Rd} and N_{Rd} a 400 times. This operation is defined as $O : \mathbb{R}^5 \to \mathbb{R}^{400\times 2}$. We can use a surrogate model

$\tilde{O}$ in the first phase, a model that is inexpensive to compute and is a good approximation of the onion. For this we will use some interpolation techniques, a comparative study of such interpolation methods can be found in [12].

Now we can describe the hybrid method: the first phase is the optimization on the surrogate model $\tilde{O}$, then we use the solution from the first or initialization phase as initial guess for the second phase that is the optimization with the true and more expensive model O. The construction of this estimation of $\tilde{O}$ is done with a data assimilation technique. This goes as follows:

(i) generate a test set L,

(ii) construct the ULS diagrams for each point in L with map O,

(iii) estimate the diagram of a new point x based on the diagrams of the points in L with a map $\tilde{O}$.

In total we distinguish three approximation methods to use in step (iii): neighbor interpolation, regression and radial basis function (RBF) interpolation.

First we explain how we generate the test set. After that we show the rather intuitive neighborhood interpolation method. Then a short section on how to describe a known diagram in less data with the Fourier series, reducing the required number of data points in our test set. Followed by various methods to interpolate or estimate these Fourier coefficients. After that we analyze the parameters of all methods and we end with a summary and comparison.

2.2.1 The test set

Low-discrepancy sequence

A test set is needed for the construction of the surrogate model. We want to generate a test set that is representative for the entire solution space. In a uniformly distributed random sequence, values can be unevenly spread and with a large test set it is likely that multiple values are (almost) equal. Instead we can use a low-discrepancy sequence, also called quasi-random sequences. These sequences have the property that each subsequence has a low discrepancy. Discrepancy is a measurement of the density of points in a sequence, a high discrepancy means that the points are unevenly spread, either there is a large empty space or there is a place where the density is high. The definition of discrepancy, following Niederreiter's notation [26], is

$$D_N(P) = \sup_{B \in J} \big| A(B; P) - \lambda_k(B) N \big|, \tag{2.1}$$

where λ_k is the k-dimensional Lebesgue measure that can be seen as the k-dimensional volume, $A(B;P)$ is the number of points of P in B, N is the total number of points in P and J is a set of k-dimensional intervals of the form

$$\prod_{i=1}^{k}[a_i, b_i) = \{x \in \mathbb{R}^k : a_i \leq x_i < b_i\},$$

with $0 \leq a_i < b_i \leq 1$, for $i = 1, ..., k$. This discrepancy is bounded by the star discrepancy D_N^*, which is the same as D_N, but its the family of intervals J^* is

$$\prod_{i=1}^{k}[0, u_i),$$

with $0 < u_i \leq 1$, for $i = 1, ..., k$. Then we have that $D_N^* \leq D_N \leq 2^k D_N^*$. For $k > 2$, the best known lower bound for the star discrepancy is from Bilyk, Lacey and Vagharshakyan [4]: there exists a $t > 0$ such that

$$D_N^*(x_1, ..., x_N) \geq t(\log N)^{(\frac{k-1}{2}+t)}, \tag{2.2}$$

for a sequence $\{x_1, ..., x_N\}$.

Halton sequence

The Halton sequence [17] is a generalization of the 1-dimensional van der Corput sequence. The van der Corput sequence works as follows. Let n be an integer and $a_l a_{l-1} ... a_0$ be the binary expansion of n. Define

$$C_2(n) = \frac{a_0}{2} + \frac{a_1}{2^2} + ... + + \frac{a_l}{2^{l+1}}, \tag{2.3}$$

then $0 \leq C_p(n) \leq 1$ for all n. Suppose we need a sequence of N numbers, the van der Corput sequence is given by $\{C_2(n)\}_{n=1}^{N}$.

For the Halton sequence of N points in $\mathbb{R}^k$, we take k coprime numbers, usually the first k prime numbers. The nth point in the sequence is given by $(C_{p_1}(n), C_{p_2}(n), ..., C_{p_k}(n))$. In two dimensions, a p, q-Halton sequence is a sequence of points with base p for the first element and base q for the second element.

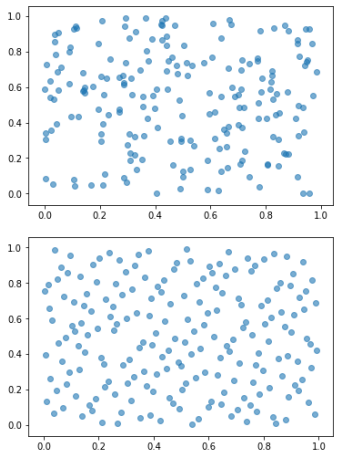


Figure 2.1: In the picture above are 200 uniformly random distributed points in $\mathbb{R}^2$. Below are 200 points that generated following a 2,3-Halton sequence.

A proven upper bound on the star discrepancy for such
a sequence is

$$D_N^* \leq c_k(\ln N)^k,$$

where c_k is a constant. This is shown in [17]. Hence the sequence attains asymptotically optimal discrepancy. The downside is that the constant increases more than exponentially in the dimension k [27]. Luckily, the dimension of our problem will not be very large in practice.

2.2.2 Neighbor interpolation

This first approximation method applies an intuitive interpolation between the data points from the test set that are close to the unknown point. In the sequel this method will be referred to as NI. This method is built on the idea that not every onion is equally important in the estimation of one onion. For example, in the approximation of a symmetrical solution a very asymmetrical onion is irrelevant and we use only the solutions that are not *too different*.

A test set is generated with the Halton sequence beforehand and for every point in the test set the corresponding ULS diagram is computed by the map O, this is the pre-processing step. In phase 1 of the optimization, we interpolate the onions of the neighboring points in the test set with this map $\tilde{O} : \mathbb{R}^5 \to \mathbb{R}^{400\times 2}$, instead of calculating the whole onion in every function evaluation with O. We do this by calculating the weighted sum of all the points, where closer points have higher weights than points that are far away.

Suppose we want to estimate the M, N-diagram of point x. The set D contains all neighbors of x and $\text{dist}(x, s)$ is the distance between point x and a point $s \in D$, we compute point $(M_x(t), N_x(t))$ on the curve by calculating the weighted mean of the points $(M_s(t), N_s(t))$ for every neighbor s:

$$M_x(t) = \begin{cases} \sum\limits_{s\in D} c_x(s)M_s(t), & \text{if } \text{dist}(x,s) \neq 0 \text{ for all } s \in D, \\ M_s(t), & \text{if } \text{dist}(x,s) = 0 \text{ for some } s \in D \end{cases}$$

and

$$N_x(t) = \begin{cases} \sum\limits_{s\in D} c_x(s)N_s(t), & \text{if } \text{dist}(x,s) \neq 0 \text{ for all } s \in D, \\ N_s(t), & \text{if } \text{dist}(x,s) = 0 \text{ for some } s \in D, \end{cases}$$

where

$$c_x(s) = \frac{w_x(s)}{\sum\limits_{s\in D} w_x(s)}.$$

The weights $w_x(s)$ are following the inverse distance weighting (IDW) strategy by Shepard [37]:

$$w_x(s) = \frac{1}{(\text{dist}(x,s))^p},$$

where p is the power parameter. We have to establish a good value for p. This parameter determines the effect that further away points have to the interpolated point. A high parameter assigns more power to the closest points and the plot of the interpolant would look like a Voronoi diagram, with similar values around the interpolation nodes and a high gradient in a very small interval between the nodes. A small power parameter assigns more influence to the further away points resulting in a more gradual change between interpolated points.

2.2.3 Fourier basis interpolation

In this section we will use Fourier series for the approximation of a M, N-diagram and capture the diagram in less data. Fourier analysis plays a great role in fields as image or audio compression and signal processing. It can be used to decompose a function in a sum of trigonometric functions that have different frequencies. The trigonometric series of a real valued periodic function $f(x)$ with period P is given by

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \big(a_n \cos(\frac{2\pi}{P} nx) + b_n \sin(\frac{2\pi}{P} nx)\big), \tag{2.4}$$

where $a_0, a_1, b_1, a_2, b_2, ...$ are coefficients that are independent of x [42]. The Fourier transform switches from time domain to frequency domain, in the sense that signal f(x) is a function of time and the coefficients $a_0, a_1, b_1, a_2, b_2, ...$ are a function of frequency. The function $f(x)$ is decomposed in a sum of sinusoids with different frequencies and the coefficients indicate how much of that frequency is present in the signal. Our goal is to find these coefficients of the series. The continuous Fourier transform for a frequency F, from which we can calculate the coefficients, is given by

$$X(F) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi Fx} dx \tag{2.5}$$

[25]. However, it is impossible to compute the integrals for the M, N-diagram, since we only have the values of the onion at c discrete sample points $f_0, f_1, ..., f_{c-1}$. Therefore we compute the discrete Fourier transform for the kth frequency

$$X_k = \sum_{n=0}^{c-1} f_n e^{-\frac{i2\pi kn}{c}}, \tag{2.6}$$

for $k = 0, 1, ..., c-1$. The solution of (2.6) is a complex number of the form $a + i \cdot b$. The onion consists of only real valued numbers. By the fact that $e^{ix} = \cos(x) + i\sin(x)$ we can translate X_k to real coefficients. The coefficients are given by

$$\begin{aligned} a_0 &= X_0, \\ a_k &= \mathrm{Re}(X_k), && k = 1, 2, ..., c-1, \\ b_k &= \mathrm{Im}(X_k), && k = 1, 2, ..., c-1, \end{aligned}$$

where if $z = a + i \cdot b$, then $\mathrm{Re}(z) = a$ and $\mathrm{Im}(z) = b$.

Now we have an approximation of $f(x)$:

$$s(x) = \frac{a_0}{2} + \sum_{n=1}^{c} \left(a_n \cos\left(\frac{2\pi}{P} nx\right) + b_n \sin\left(\frac{2\pi}{P} nx\right), \tag{2.7}$$

here we call c the number of basis functions. For larger c, the approximation gets better, but the construction of the surrogate model becomes more expensive, so we have to look for a good trade-off between those two.

Approximating the Fourier coefficients with for example neighbor interpolation (FFT+NI) hopefully outperforms the NI from previous section. Then we are approximating a map to $\mathbb{R}^{4c+2}$ instead of a map to $\mathbb{R}^{400\times 2}$, hence we probably need less data points in our test set to get a good approximation.

2.2.4 Linear model

For each point $(d_1, d_2, s_1, s_2, f_{\mathrm{ck}})$ in the test set L, we compute the onion with O and find its Fourier transform coefficients with the Fast Fourier transform (FFT) algorithm that is the map $F : \mathbb{R}^{400\times 2} \to \mathbb{R}^{4c+2}$. Based on the coefficients of solutions in the test set, we use linear regression to build a linear map $A : \mathbb{R}^5 \to \mathbb{R}^{4c+2}$ that computes the coefficients for any point x. How this linear map is built is explained in appendix A.2.1. With the approximated FFT coefficients we can calculate the interpolant with equation (2.7). The linear map A and the interpolant $s : \mathbb{R}^{4c+2} \to \mathbb{R}^{400\times 2}$ together replace the computationally heavy map O.

The motivation for this method is that in one function evaluation we can solve a linear system Ax to get the coefficients of the approximation instead of solving the equations for M_{Rd} and N_{Rd} a 400 times. The size of matrix A is $(4c+2) \times 5$ and, as we will see later, small c is sufficient. Which makes one function evaluation very cheap, especially compared to map O.

In the method FFT+LIN we apply map F to all nodes in the test set and obtain A by doing linear regression on all data points. Then use this linear map A to estimate the

coefficients of the guesses during the optimization. The Fourier transform for one onion is a very good approximation and the FFT-algorithm is fast, but it seems like the linear model does not fit properly, especially in boundary cases, see for example figure 2.5. What if we use a different model for the approximation of the Fourier coefficients? We mentioned earlier that not all interpolation nodes are equally important for the approximation of a new point. However, the linear model incorporates them all. Therefore we replace this by a model that operates more locally, considering only the nearest neighbors when constructing map A. This method applies regression only on the nearest neighbors of the current point x. Hopefully this is a good compromise between FFT+LIN and FFT+NI. We denote this method by FFT+NNLIN.

There are two ways of defining the neighborhood of x. The first is a predefined $d_{\max}$ that gives the radius of the neighborhood. The alternative way is taking the k nearest neighbors, k points in the test set that are closest by. The essence of this method is that we take a subset of the test set to build the linear model on, hoping that the map F is close to linear for this subset.

2.2.5 Radial basis function interpolation

The next approximation method that we consider is the radial basis function (RBF) interpolation [14]. This is a widely used approximation method that performs well in higher dimensions [6] and is some generalization of the inverse distance weighting. The RBF interpolant is a weighted sum of radial basis functions, functions that depend only on the distance between two data points. Looking at the procedure of previous section, this RBF interpolation replaces map A, which means that the RBF interpolant estimates the FFT coefficients. In our experiments we use Gaussians as RBF, given by $\phi_\varepsilon(r) = \exp(-(\varepsilon r)^2)$, where ε is the shape parameter. The smaller the shape parameter, the flatter the curve $\phi_\varepsilon(r)$. In figure 2.2 are two 1-dimensional Gaussians shown with different shape parameters.

In RBF interpolation the goal is to find an approximation of a function $f(x)$, which we call $s(x)$. Now, the function $s(x)$ is the weighted sum of Gaussians, that is

$$s(x) = \sum_{k=0}^{K-1} w_k \phi_\varepsilon(||x - x_k||). \tag{2.8}$$

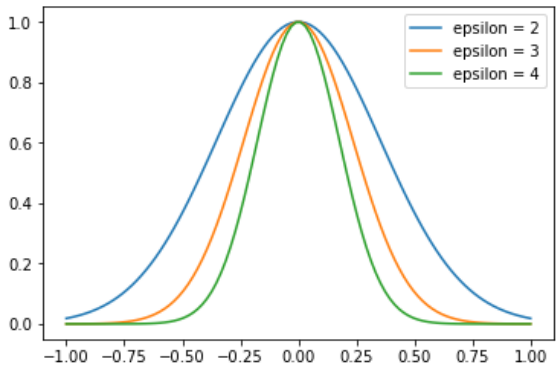


Figure 2.2: Gaussian functions with the shape parameters 2, 3 and 4.

We take K equally spaced points $x_0, x_1, ..., x_{K-1}$ at which s interpolates f, i.e. $s(x_k) = f(x_k)$ for $k = 0, 1, ..., K-1$. This yields the following linear system

$$\begin{bmatrix} \phi_\varepsilon(||x_0 - x_0||) & \phi_\varepsilon(||x_1 - x_0||) & \dots & \phi_\varepsilon(||x_{K-1} - x_0||) \\ \phi_\varepsilon(||x_0 - x_1||) & \phi_\varepsilon(||x_1 - x_1||) & \dots & \phi_\varepsilon(||x_{K-1} - x_1||) \\ \vdots & \vdots & \ddots & \dots \\ \phi_\varepsilon(||x_0 - x_{K-1}||) & \phi_\varepsilon(||x_1 - x_{K-1}||) & \dots & \phi_\varepsilon(||x_{K-1} - x_{K-1}||) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{K-1} \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{K-1}) \end{bmatrix}$$

or in short: $Iw = y$. Since ϕ_ε is strictly positive definite, the matrix I is nonsingular and there exists one unique solution w [7].

The to be approximated function is the map from x to its FFT coefficients: $\mathbb{R}^5 \to \mathbb{R}^{4c+2}$. We can solve this system beforehand, returning the weights for the interpolant (2.8) and with this we can compute the interpolated FFT coefficients for each point x.

Shape parameter

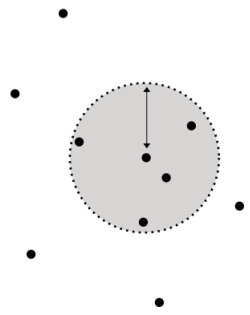
Choosing a good shape parameter for RBF interpolation has been a subject of research for over 20 years, see for example [34, 39, 35, 13]. Since the true function f is known, it is possible to compute the interpolation error of the RBF interpolant s. This we will use to find the best shape parameter. Intuitively, if the interpolation points are further away from each other, the shape parameter has to be smaller. Regarding stability, the interpolation matrix I goes to the identity matrix as ε grows to infinity. This yields stability in solving system (2.2.5), but the error between interpolation nodes will be large. When ε goes to 0, the condition number [2] of matrix I grows to infinity, which makes (2.2.5) an ill-conditioned problem. Typically the best ε is as small as possible but not yet ill-conditioned.

The RBF interpolation is all about the distances. Instead of using one constant shape

parameter for every approximation, we could alter it according to the distance from the (to be approximated) point x to the interpolation nodes. Recall how the shape parameter influences the Gaussian function and note that with relatively large distances, the shape parameter will probably have to be relatively small. Hence it makes sense to let the shape parameter be inversely proportional to the mean distance of point x to the interpolation points, i.e. given the mean distance D_x and a constant a the shape parameter is given by

$$D_x \cdot \varepsilon = a. \tag{2.9}$$

2.2.6 How to define the neighborhood



The methods NI, FFT+NI and FFT+NNLIN compute the surrogate model based on a set of neighbors. In this section we look more closely to how we can define the neighborhood. If we define the neighborhood of point x by a sphere around x with radius $d_{\max}$, then we move an equal amount in each direction, as in the figure on the right. The domain space relies on the unit in which we choose to measure the design variables, the diameters are measured in millimeters and are therefore taken from the interval $[6, 40]$. However, if the units were centimeters they would come from the interval $[0.6, 4]$. This is just a choice of scaling. Due to the fact that the scaling is arbitrary, we could choose to scale the parameter spaces by some weight. It is possible that we want more information in one direction than the other. To test this, we consider the change to the FFT coefficients in the direction of each design variable. If there is a lot of change in a certain direction, then the coefficients of points that lie far from x in this direction are most likely different from the coefficients of x and we want to include less points in this direction than in a direction with less change.

If we had the Jacobian matrix of map F, we would know all partial derivatives which describe the change in each parameter. The Jacobian matrix of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is defined as

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{2.10}$$

Column i describes the change in function f caused by x_i. For this reason, the L^2-norm of column i is a good way to quantify the influence of x_i on function f.

Recall the linear map in FFT+LIN and FFT+NNLIN, where we approximate map F with a linear model. In 2.3 is 2-dimensional linear regression shown. We can see that the regression hyperplane tells us something about the change in variables in the true function. On that account, one can interpret the entries in matrix A as a numerical estimation of the entries of the Jacobian matrix.

Now we want determine which points in the test set are relevant for the approximation of the FFT coefficients of a point x. Suppose that matrix A indicates that there is a lot of change in direction i, this means that the FFT coefficients of points that lie far away in this direction are probably far from similar to the coefficients of point x. While in a direction with small amount of change the coefficients of points that lie equally far away may say more about the coefficients of point x. Hence in dimensions with less change we want to look further. Consequently, we assign a weight to each design variable that describes the change. The weights for the three types of design variables are given in table 2.1, as expected the weights are roughly the same for all test set sizes.

We scale the parameter spaces to the size of the corresponding weights. In the weighted intervals, we can either look for points that lie inside a sphere of radius $d_{\max}$ around the current point or we can take the k closest points.

$\|L\|$	Weight diameter	Weight spacing	Weight concrete strength
100	142.42	9.82	201.12
200	141.63	10.01	201.24
300	140.27	10.05	201.66
800	139.94	9.99	201.45
1300	139.89	10.02	201.54

Table 2.1: Weights of the design variables computed for different test set sizes $|L|$.

2.3 Parameter selection

In this section we study the parameters that influence the construction of the surrogate models and are not part of the data of the concrete design. Such parameters are called the hyperparameters, they affect the approximation process and therefore must be determined beforehand. Different strategies can be used to find good hyperparameters, usually it is impossible to find the absolute best. Our approaches will mostly be based on grid search, where we exhaustively generate a set of hyperparameters in a certain interval and use the best performing value. The test sets used for the hyperparameter tuning are generated by a Halton sequence, later we will compare the results with a random sequence.

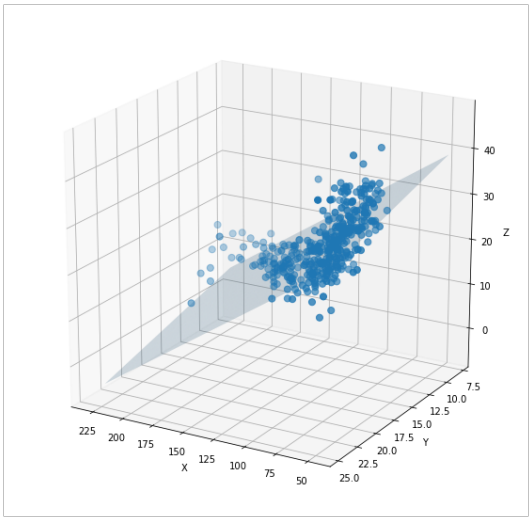


Figure 2.3: 2-dimensional regression.

To find suiting hyperparameters and to compare the estimation methods, we use leave-one-out cross-validation [1]. Cross-validation is a technique to test the quality of an estimator. This works as follows:

(i) generate a set L of n data points and compute the M, N-diagram for each point in the set,

(ii) take the first node x_1 and

(iii) estimate its diagram by interpolating on all onions except the first one, let y_1 be the true onion for x_1 and $\hat{y}_1$ be the approximation,

(iv) repeat steps (ii) and (iii) for every point in L.

We can compute the total cross-validation error by

$$\frac{\sum_{i \in L} ||y_i - \hat{y}_i||}{n}, \tag{2.11}$$

or the cross-validation mean squared error (MSE) and maximum error:

$$\frac{\sum_{i \in L} ||y_i - \hat{y}_i||^2}{n} \quad \text{and} \quad \max_{i \in L} ||y_i - \hat{y}_i||. \tag{2.12}$$

In this section we go over each of the previous explained methods and find good hyperparameters.

Neighbor interpolation

We have to determine two hyperparameters for the neighbor interpolation: power parameter p and neighborhood radius $d_{\max}$. The best power parameter depends on the density of the interpolation nodes, therefore we compute the cross-validation error (2.11) for different test set sizes and neighborhood radii. The reason to incorporate $d_{\max}$ is to limit the computation of the interpolation. We expect that too small $d_{\max}$ will give bad approximations.

Plan of action to find the best p and $d_{\max}$:

1. find best p for a fixed large $d_{\max}$,
2. then find the best $d_{\max}$ with that power parameter.

The reason for this procedure is we noticed that from a certain value of $d_{\max}$, the best value of p and the cross-validation error are (almost) the same. We will show it here for $|L| = 300$, one can read the results for other test set sizes in appendix A.1, together with the cross-validation MSE and maximum error. In figure 2.7 we see the cross-validation error as a function of p, the neighborhood radius is equal to 150. It is obvious that a minimum is reached with $p^* = 2.5$. Then with p^* the cross-validation error is calculated for different values of $d_{\max}$, the results are shown in figure 2.8. The minimum is at $d_{\max} = 41$. Thereupon, the hyperparameters used for NI with a test set consisting 300 data points are $d_{\max} = 41$ and $p = 2.5$.

Scaled dimensions

In the neighbor interpolation, we gain some by scaling the parameter spaces according to the weights in table 2.1. The neighborhood radius and power parameter are obtained in the way described above. The cross-validation errors of NI with scaled dimensions can be found in table A.2. For a test set of 200 points, the MSE is reduced by one million.

FFT+NI

Finding the best hyperparameters for this method is easy, we can just copy the best $d_{\max}$ and p of the stand-alone neighbor interpolation method. To see this, remember that the neighborhood radius told us how close a node has to be to the current point x to be a neighbor. Only the neighbors of x influence the approximation. We iterate over the interpolation nodes to see which may influence the approximation of point x and p describes how much this influence is. In the new method it is no different, the only difference is that the former outcome approximated an onion and the new outcome approximates the FFT coefficients of the onion.

Looking at the approximations, there is one thing that is remarkable. At the boundary cases, where the FFT+LIN algorithm failed, the FFT+NI works really well. On the other hand, there are also cases where FFT+LIN performed good and FFT+NI is bad. See figure 2.6 for two examples. In appendix A.3 in table A.5 are the MSE and maximum error shown for different test set sizes.

Scaled dimensions

Similar to method NI, this method can be improved by scaling the parameter spaces with the weights. Results are given in table A.6. For a test set of 200 points, the mean squared error with scaled dimensions is around 40 thousand, while the MSE with original parameter spaces was 151 thousand.

FFT+RBF

The accuracy of the RBF interpolant depends on the choice of number of test set points $|L|$ and the shape parameter ε. Per value of $|L|$, we can find the best (constant) shape parameter. Due to the effect of ε to the Gaussian function, smaller $|L|$ require smaller shape parameters. This is a consequence of the points laying further apart, so the functions have to be flatter. The density of the test set in the whole solution space is small, so the distance between points in the test set will be large. Consequently, the shape parameter ε will have to be small. To find the best shape parameter, we consider the cross-validation MSE. This we do for different values of $|L|$. Table A.7 in the appendix shows the best constant shape parameter for $|L| = 100, 200, 300$.

For the variable shape parameter we have to find a good constant a for the computation of the shape parameter ε (2.9). Figure 2.12 shows the cross-validation MSE and maximum error for different values of a, the minimum in both plots is reached at $a = 19.000$. Cross-validation errors can be found in appendix A.4.

FFT+LIN

The performance of the hybrid optimization with Fourier basis interpolation depends on two things, the number of basis functions c and the size of the test set $|L|$. The larger these parameters, the better the approximation. Unfortunately at the same time the running time will increase. As a result, the goal is to find a small c and $|L|$ such that the interpolant is still quite good. We have computed the cross-validation MSE and max error for a few combinations of c and $|L|$, the results are given in the tables in appendix A.5.

FFT+NNLIN

This method is actually two algorithms. To distinct them, lets call one the neighborhood method and the other the k-neighbors method. For the neighborhood we need to determine neighborhood radius $d_{\max}$. For the k-neighbors we need to choose k. To compare them, we compute the mean number of neighbors for every point in the test set for different $d_{\max}$. This is shown in figure 2.9. As before, we compute the cross-validation MSE and maximum error for different values of $d_{\max}$ and p, the results for $|L| = 200$ are shown in table A.9.A few things can be seen in the table, which are summed up below.

(i) For building the linear model on (on average) 10 or 20 nearest neighbors, the k-neighbor method performs better than the neighborhood method.

(ii) When building the linear model on (on average) 30 nearest neighbors, the neighborhood method performs best.

(iii) From some point on, FFT+NNLIN is better with neighborhood radius $d_{\max}$ than with the k-nearest neighbors, this is probably because neighborhood is centered around x and if we just take the first k neighbors, we might take one that is pretty far away and irrelevant for the current point.

(iv) The cross-validation errors of FFT+NNLIN with $d_{\max} = 60$ and with $d_{\max} = 67$ are lower than the errors of FFT+LIN, where we build the linear map on all points in the test set. Moreover, in table A.9 the MSE decreases when $d_{\max}$ increases, which means that there must be a radius after which the MSE goes up again. Figure 2.10 shows the cross-validation errors for different values of the neighborhood radius. There indeed exists a minimum for both the MSE and maximum error.

With a test set of 200 data points, the minimum error is obtained with $d_{\max} = 62$ and the MSE is somewhere above 8 thousand.

Scaled dimensions

The above can be applied to the scaled parameter spaces. Figure 2.11 shows the cross-validation errors in this case. The neighborhood radius that achieves both the lowest MSE and the lowest maximum error is 61. The MSE is equal to 7692.57 and the maximum error is 447.72, a significant improvement!

2.4 Summary

Here we summarize the results of the surrogate methods and compare them. We also look at the difference between generating the test set according to a random sequence and the Halton sequence.

(i) The first approximation method that we discussed is the neighbor interpolation. As we can see in table A.1, the cross-validation MSEs are around 4 million and the maximum errors are around 4 thousand for test sets that have 200 up to 600 points. With random test set we need a larger radius, as the points are not evenly spread there will be larger empty areas. Also the errors with random generated test set are higher.

(ii) The next method we consider is the FFT+LIN approximation method, results are given in tables A.3 and A.4. Using a Halton test set of 100 and 200 points, for 30 or more basis functions the MSEs are around 20 thousand and maximum errors around 800. With a randomly generated test set the results are a bit better, but here a higher number of basis functions does not mean a lower error. Clearly, FFT+LIN outperforms NI.

(iii) For a Halton test set with $|L| = 600$ the MSE of method FFT+NI is 67 thousand, a test set of 600 points is too expensive to prepare and the MSE is more than triple the MSE of FFT+LIN with a test set of 100. Therefore, FFT+NI is not an improvement on the previous methods. In some cases a randomly generated test set is better, but this still does not compete with FFT+LIN. If we compare the results of FFT+NI with NI, we see that FFT+NI does perform better, this is because NI approximates a map to $\mathbb{R}^{400\times 2}$ and FFT+NI approximates the map to $\mathbb{R}^{4c+2}$.

(iv) In figure 2.12 we can see that the cross-validation MSE with a test set of 200 with method FFT+RBF with a constant shape parameter is between 10 and 20 thousand, so this method is competitive with FFT+LIN. A random test set obtains smaller errors. Errors with both a Halton and random test set are given in table A.7.

(v) Then we consider the combination FFT+NNLIN, with the original units for the design variables the MSE is a bit larger than 8 thousand, with weighted dimensions the MSE is 7692. This is the best so far. A Halton generated test set outperformed the random test set here.

To summarize, we could immediately discard the method NI. Then the methods that remain are the methods that approximate the FFT coefficients. The best MSE and maximum error of these methods with a test set of 200 points and 15 Fourier basis functions are given in the table below.

Method	**MSE**	**Maximum error**
FFT+LIN	21700.05	792.05
FFT+NI	151727.61	2215.90
FFT+RBF (variable ε)	11081.90	539.94
FFT+NNLIN	7692.57	447.72

Table 2.2: Cross-validation errors of approximation methods on the FFT coefficients with $|L| = 200$ and $c = 15$.

2.4.1 Conclusion & discussion

The method FFT+NNLIN yields the best approximations, scaling the dimensions with weights from table 2.1 improves the method even more. Some approximations are shown in figure 2.4, the quality is quite high. The method FFT+RBF with a variable shape parameter ε is almost as good in cross-validation errors, but due to the recalculation of the shape parameter for each new point this method is quite expensive. FFT+RBF is 87 times slower than FFT+NNLIN for one node, this is tested by timing each approximation step during cross-validation and calculating the average.

The weights in table 2.1 describe the sensitivity of the onion. It is remarkable that the change in FFT coefficients is so small when changing the spacing compared to the other two variable types. It makes sense if the sensitivity to the diameter is squared the sensitivity to the spacing, but the weight for diameter is even larger than that.

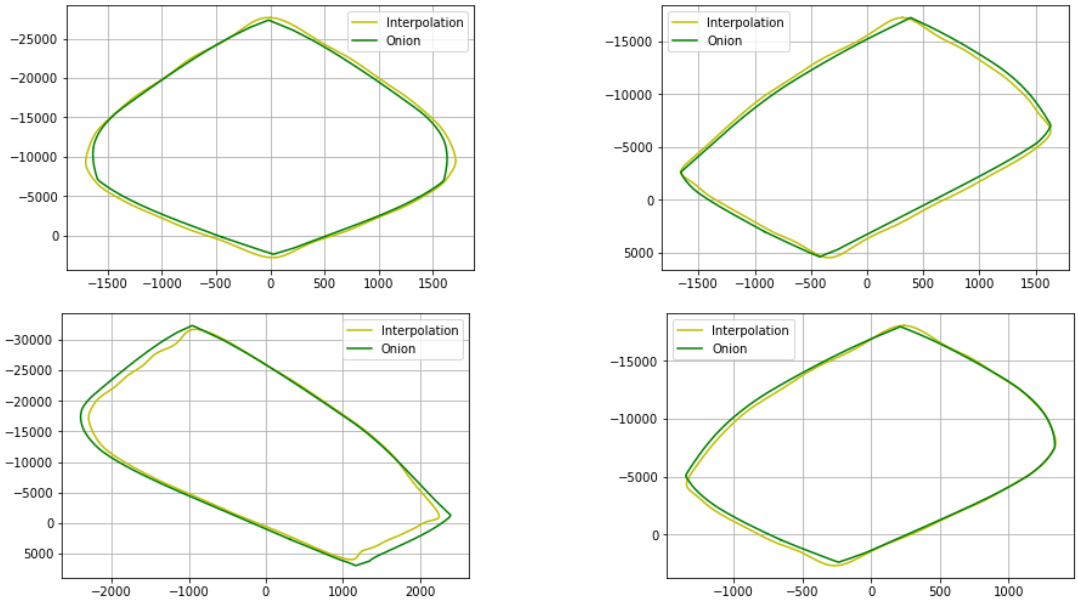


Figure 2.4: Approximations done with FFT+NNLIN in weighted dimensions.

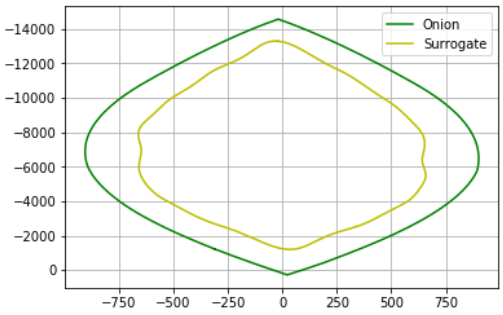


Figure 2.5: Surrogate model of point $x = (7.59, 12.71, 247.8, 295.40, 42.94)$ with a linear model that is built on the Fourier transform coefficients with $c = 15$ of points in a test set of size $|L| = 300$.

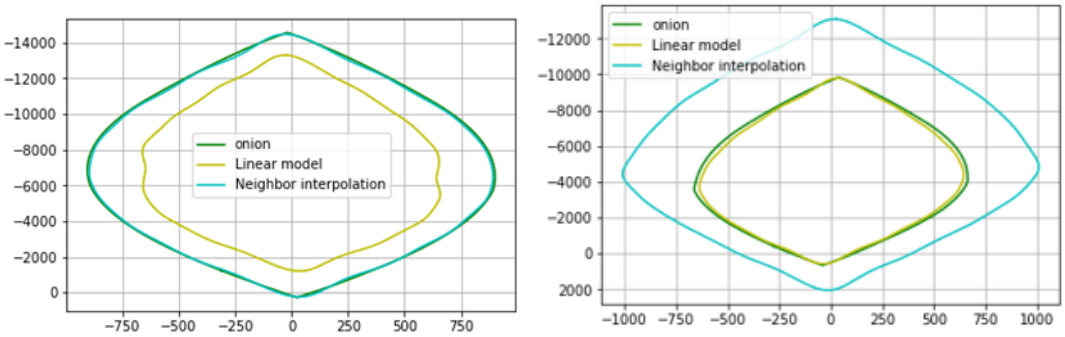


Figure 2.6: Two approximations with method FFT+NI and method FFT+LIN with $|L| = 300, p = 2.5$ and $d_{\max} = 41$.

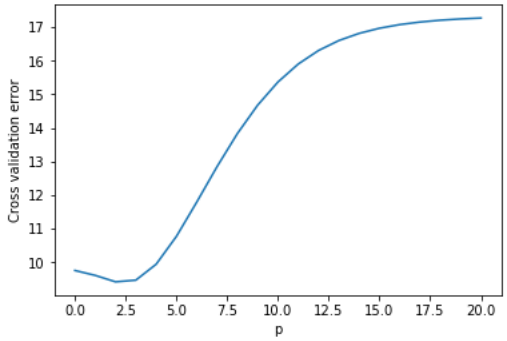


Figure 2.7: Cross-validation error as a function of power parameter p with $d_{\max} = 150$.

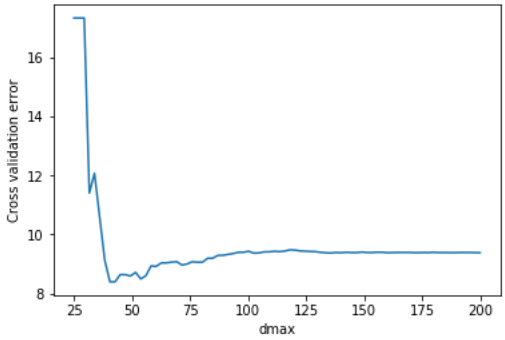


Figure 2.8: Cross-validation error as a function of neighborhood radius $d_{\max}$ with $p = 2.5$.

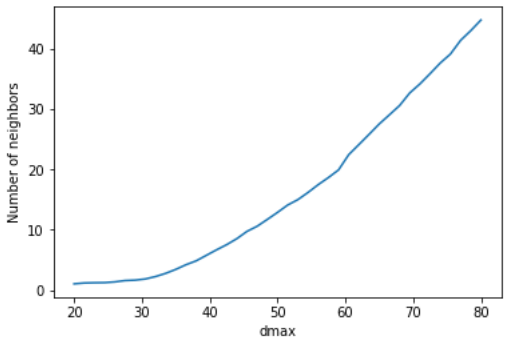


Figure 2.9: Mean number of neighbors in cross-validation for a test set of size 200.

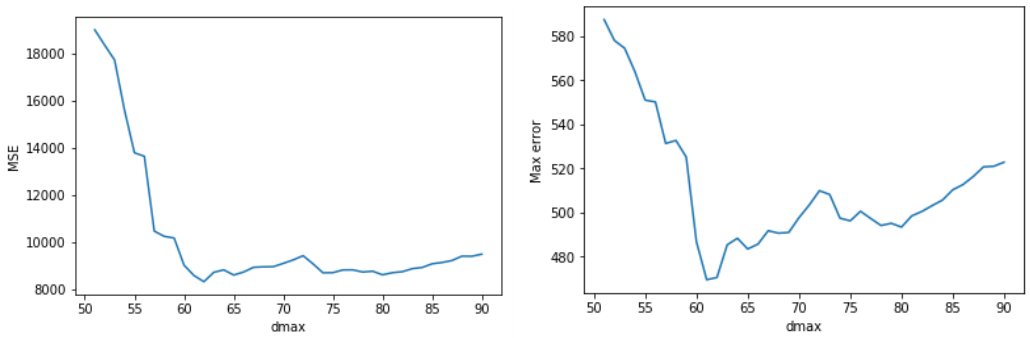


Figure 2.10: Cross-validation errors for FFT+NNLIN as a function of the neighborhood radius $d_{\max}$.

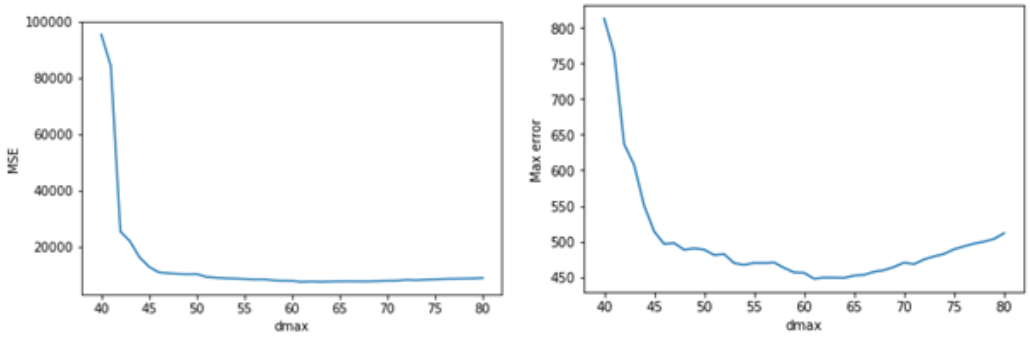


Figure 2.11: Cross-validation errors for FFT+NNLIN as a function of the neighborhood radius $d_{\max}$ with weighted dimensions.

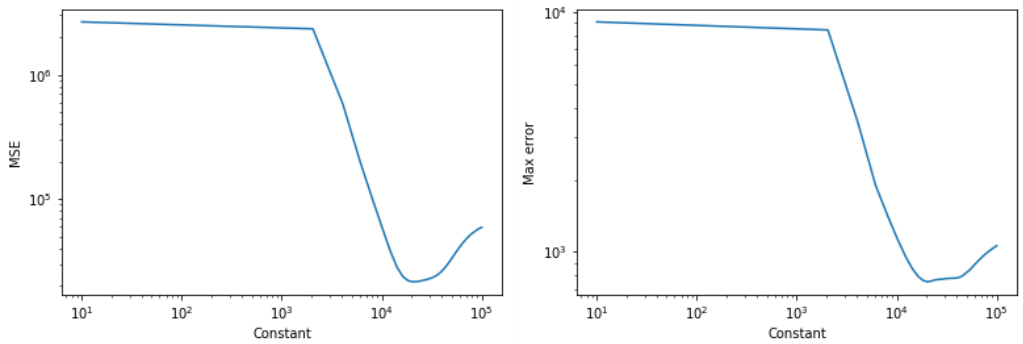


Figure 2.12: Cross-validation mean squared error and maximum error vs constant in the calculation of the variable shape parameter for FFT+RBF with $|L| = 100$ and $c = 15$.

Chapter 3

Optimization based on design margins

In this chapter we discuss the optimization based on the design margins. The idea is that we want to fit the onion as tight as possible around the FEM points, minimizing the excess capacity. This is based on the intuitive way the engineers would try to improve their design by hand. We first define the objective, then the numerical results are given of optimization with methods from previous chapter. In the end we state the conclusions and the discussions.

3.1 Defining the problem

The onion is a representation of the resistance of the design and we want to optimize the material usage such that each FEM point lies inside the onion. Intuitively, if we have a design with minimum material usage, the ULS diagram fits tight around the points, i.e. there must be two or more FEM points lying on the boundary of the diagram. Therefore we use the distance from the FEM points to the onion as our objective. The distance can be measured with the design margins. Recall that the design margin of a FEM point is equal to one when it is exactly on the boundary of the diagram, hence we want to get out design margins as close as possible to one, while the design is feasible.

The height and width of the cross-section are fixed as well as the position of the layers of reinforcing bars. For now we consider a surface with two layers of reinforcement. The optimization parameters are the diameters of the bars in the top layer d_1 and the bottom layer d_2 in millimeters, the spacing between the bars in the top layer s_1 and the spacing between the bars in the bottom layer s_2 in millimeters and the concrete strength f_{ck} in megapascal. All parameters are bounded. Our interval for the diameters are the most common sizes for reinforcing bars. The lower bound for spacing is due to the minimum

amount of concrete that is required in between the steel, to keep the structure together. The upper bound for spacing is based on the practical experience of the supervisors. The bounds for the concrete strength is given in the Eurocode.

We consider the computation of the design margins to be a black box. Let MN be the set of given internal forces, then the optimization problem is as follows:

$$\begin{array}{lrl} \min & \sum_{i \in MN} |uc(i) - 1| & \\ \text{s.t.} & uc(i) \leq 1 & \text{for all } i \in MN \\ & 6 \leq d_i \leq 40 & \text{for } i = 1, 2 \\ & 75 \leq s_i \leq 300 & \text{for } i = 1, 2 \\ & 12 \leq f_{\text{ck}} \leq 90 & \end{array} \tag{3.1}$$

where $uc(i)$ is the design margin of point i, uc stands for unity check. Since the function $uc(i)$ is a black box, we do not have any information on the gradient of the objective function. The problem is a constrained optimization problem, with a constraint space that does not hold any known useful properties such as convexity or differentiability. We can transform it into an unconstrained (but bounded) problem by applying the penalty method [15], to simplify the solution space. In this method we add a penalty to the objective value for a constraint that is being violated. This is only applied to the first constraint of problem (3.1), the bounds on the variables stay. The new loss function is

$$l(uc) = \sum_{i \in MN} (|uc(i) - 1| + \lambda * \max\{0, uc(i) - 1\}). \tag{3.2}$$

We take $\lambda > 0$ large enough to make sure that it is always preferred to have all points inside the diagram. Consider one FEM point i, if its design margin is smaller than one, this point adds $|uc(i) - 1|$ to the loss function, if its design margin is larger (or equal) than one the point adds $(\lambda + 1)|uc(i) - 1|$ to the loss function. With this in mind we expect that a small λ will be sufficient.

Note that the absolute value function in the objective of problem (3.1) causes that the objective function does not have a derivative when $uc(i) = 1$ for some $i \in MN$, which is the case at a minimum. We can replace the function $f(x) = |x|$ with a smoother function. Especially gradient-based optimization methods will probably benefit from this. We will use a smooth approximation for the absolute function, namely $g_\alpha(x) = \sqrt{x^2 + \alpha}$, where α is a small number. Observe that this function $g_\alpha(x)$ contains the same minimum as $f(x)$ and is only different around the minimum, see figure 3.1. The smaller the α, the sharper the curve at the minimum and $\lim_{\alpha \to 0} g_\alpha(x) = |x| = f(x)$.

Taking the maximum in our new loss function (3.2) also produces a kink at a minimum.

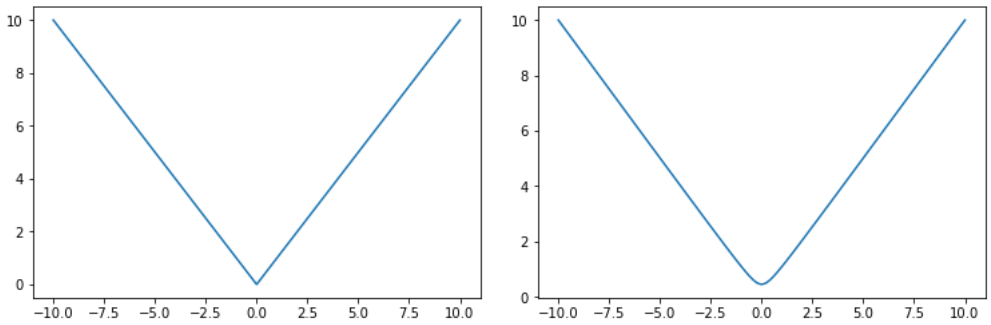


Figure 3.1: Left the function $f(x) = |x|$ and right the smoothed function with $\alpha = 0.2$: $g_{0.2}(x) = \sqrt{x^2 + 0.2}$.

We rewrite $\max\{x, y\} = \frac{1}{2}(x + y + |x - y|)$ and then replace the absolute function with its smooth approximation. The smoother loss function is given by

$$l_{\text{smooth}}(uc) = \sum_{i \in MN} \left(\sqrt{(uc(i) - 1)^2 + \alpha_1} + \frac{1}{2}\lambda(uc(i) - 1 + \sqrt{(uc(i) - 1)^2 + \alpha_2})\right). \quad (3.3)$$

The solution of our optimization problem is a vector with five elements: $x = (d_1, d_2, s_1, s_2, f_{\text{ck}})$.

3.2 Results

The algorithms are tested on four different data sets, such a set is a list of FEM points. All data is coming from UC1. The first input is called *Almere* and it is data from a real project, the FEM points in this input are centered and has the shape of a cloud. The second input is called *All Situations*, this is artificial data made for testing and it is called All Situations because it has points in each quadrant. The FEM points in this list lay more outside the onion of our initial guess than the points of Almere and there are points in all four quadrants. The other two are Test Model 1 and Test Model 2, both being artificial data and having different helpful properties.

Before running the algorithms, we compute the convex hull of the internal forces from the input. Remember that we want all the FEM points to lie inside the onion. Furthermore, the onion is a convex polygon. Hence when the outermost points are inside the onion, all the points will lie inside the onion. Thus it is only necessary to check for the points on the convex hull of the FEM points.

First we investigate the hyperparameters. The two (or three) hyperparameters that need tuning are the constraint penalty λ in loss function (3.2) and the smoothing factors α_1 and α_2 in (3.3). Then with the best values for the hyperparameters, we do a comparison of optimization methods.

3.2.1 Hyperparameters

It is required for the penalty λ in (3.2) that a feasible solution of problem (3.1) is always preferred over an infeasible solution, i.e. has a lower loss value. In the search for a good penalty we use the SciPy package to minimize function (3.2) for different λ. We compare the optimal values, the number of function evaluations until termination and the running time. We also check if any solution violates the constraint. All of the solutions are feasible. The optimization method that we use is COBYLA and the comparison is done for both input files. The results are shown in appendix B.1.1.

One decision to make is whether we want the penalty that is best in optimal solution or the penalty that is best in running time, or a combination of both. For that we dive a bit deeper in our objective function. We minimize the sum of $uc(i) - 1$ for each FEM point i and add an extra $\lambda(uc(i) - 1)$ if point i is outside the diagram. This means that we want *every* point as close as possible to the onion, while initially our goal was to have the onion lie tightly around the points, so not every point is important here. When looking at the solutions from different optimizations, we see that most solutions are good enough, i.e. two or more points on the boundary of the diagram. Therefore we focus on a short running time or a small number of function evaluations for both input files. We take $\lambda = 17$.

The other hyperparameters are the smoothing factors α_1 and α_2 in loss function (3.3). Since it is two times the approximation of the same function, namely $|x|$, we take $\alpha_1 = \alpha_2 = \alpha$. The loss function then (3.3) becomes

$$L_{\text{smooth}}(uc) = \sum_{i \in MN} \left((1 + \frac{1}{2}\lambda) \sqrt{(uc(i) - 1)^2 + \alpha} + \frac{1}{2}\lambda(uc(i) - 1) \right) \tag{3.4}$$

Recall that a larger α yields a flatter bottom. Due to the fact that the optimization stops when the reduction in loss is smaller than a certain factor, we expect that the α cannot be too large. When the differences are too small close to the minimum, the optimization will probably not end up in the true minimum but in some other solution in the neighborhood. The rounder/flatter the bottom, the bigger this neighborhood. The results are shown in appendix B.1.2. We compare the α's in optimal value after optimization, the number of function evaluations and the running time. The performances are different for every input file, which makes it hard to to point out the best. As we can see in figure B.7a, the optimal solution has lowest loss value for a small α and increases exponentially until one point, then it stabilizes. This stabilization matches our expectations in previous paragraph. Large

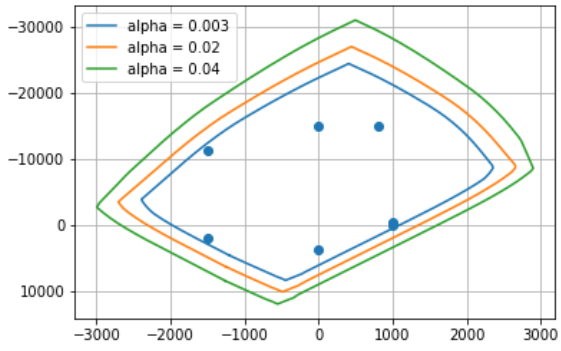


Figure 3.2: Solutions of minimization of (3.4) with SLSQP and different α's.

smoothing factors yield fast termination but the solutions are not good enough. Therefore we look for α that is as large as possible while still attaining a solution that has at least two points on the boundary. This turns out to be $\alpha = 0.003$. Figure 3.2 shows the solutions of optimization of objective (3.4) with SLSQP for different smoothing factors, observe that the two largest factors do not attain optimal solutions.

3.2.2 Method comparison

In this section we compare the different optimization methods. First we consider the available methods in the SciPy minimize function, the ones that do not need a gradient beforehand and can handle bounds. These methods can be divided in two groups, methods that are gradient-based, which try to approximate the gradient, and methods that do not consider the gradient at all. As we mentioned in previous chapter, gradient-based quasi-Newton method have a fast convergence rate, SLSQP and L-BFGS-B are such methods [11]. A gradient-free method that we will see frequently is COBYLA: Constrained Optimization BY Linear Approximation. This is a simplex method, that fits a linear model on vertices of the trust region [32]. After the results of these existing methods, we will show results of the hybrid optimization. At last we compare all the above methods with one-another.

Existing methods

We test the methods on the non-smooth loss function with penalty (3.2) and the smoother version (3.4). For all solutions is checked whether it is *good enough*, i.e. has at least two points on the M, N-diagram. In table 3.1 are the number of function evaluations, running time and loss value of the obtained solution x^* shown for the optimization on objective (3.2) with the optimization methods in the SciPy library, for both Almere and All Situations. As we expected, the gradient-based methods [1] do not perform well on this objective

[1] SLSQP, Truncated Newton method, L-BFGS-B

with a sharp kink at the minima. Here, COBYLA is by far the best method. The ECI value of the solution from this method on Almere is 14.448 and the ECI value of the result on All Situations is 16.256.

	Almere			All Situations		
Method	**Evaluations**	**Time**	$l(x^*)$	**Evaluations**	**Time**	$l(x^*)$
COBYLA	73	22.26	6.99	96	29.05	1.79
SLSQP	248	75.79	5.70	532	171.56	1.26
Powell	500	164.37	5.69	592	187.88	1.90
Truncated Newton	258	99.75	7.16	522	170.64	2.22
L-BFGS-B	600	189.42	7.12	1836	577.39	1.06

Table 3.1: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. Both on the Almere input and the All Situations input with objective (3.2) and lambda = 17.

	Test Model 1			Test Model 2		
Method	**Evaluations**	**Time**	$l(x^*)$	**Evaluations**	**Time**	$l(x^*)$
COBYLA	77	29.65	5.15	85	32.34	4.50
SLSQP	323	256.72	4.73	427	170.42	3.80
Powell	311	130.27	4.73	300	112.35	4.48
Truncated Newton	312	116.96	5.41	606	232.26	4.00
L-BFGS-B	522	211.89	5.43	1116	601.64	4.08

Table 3.2: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. On the Test Models input with objective (3.2) and $\lambda = 17$.

Due to the smoothing of the kinks in the function, we expect that gradient-based methods will perform better on objective (3.4). As we can see in table 3.4, the last four methods are still pretty slow, but SLSQP is much faster than on (3.2). COBYLA reaches its maximum number of function evaluations on this objective, which means that the linear approximation is harder on a function with a round bottom. Observe that the shortest running time (43.09 seconds for Almere) is nowhere near the shortest running time of objective (3.2) (22.26 seconds for Almere). So for now, COBYLA with objective (3.2) is the number one in ranking.

	Almere			All Situations		
Method	**Evaluations**	**Time**	$l(x^*)$	**Evaluations**	**Time**	$l(x^*)$
COBYLA	1000	352.85	6.43	1000	320.18	1.43
SLSQP	135	43.09	6.28	173	52.33	1.44
Powell	481	165.72	6.28	848	283.58	1.44
Truncated Newton	600	178.31	6.25	564	174.03	1.43
L-BFGS-B	570	186.90	6.15	606	191.21	1.42

Table 3.3: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. Both on the Almere input and the All Situations input with objective (3.4), $\lambda = 17$ and $\alpha = 0.003$.

	Test Model 1			Test Model 2		
Method	**Evaluations**	**Time**	$l(x^*)$	**Evaluations**	**Time**	$l(x^*)$
COBYLA	1000	365.04	6.06	1000	360.28	4.93
SLSQP	108	38.25	6.06	200	73.41	4.41
Powell	226	90.80	6.05	397	133.71	4.41
Truncated Newton	246	82.53	6.06	594	205.68	4.41
L-BFGS-B	258	88.37	6.04	228	83.55	4.41

Table 3.4: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. On data sets of Test Model 1 and 2 with objective (3.4), $\lambda = 17$ and $\alpha = 0.003$.

Hybrid optimization

Now we examine the performance of the hybrid method. One of the surrogate models is applied: FFT+NNLIN, this was the approximation method with the lowest errors in cross-validation. The optimization is performed on the loss function (3.2) and the method that is being used in the separate phases is COBYLA. The results are being compared with the performance of the COBYLA without initialization phase.

At the beginning, before optimization of the surrogate model, we need a test set for the construction of the model. This test set is generated according to a 5-dimensional Halton sequence. This test set must represent the whole solution space of the optimization problem and we must be able to interpolate solutions right in the middle of this space as well as on the edges.

Table 3.6 shows the number of function evaluations and running times of phase 1 and

phase 2 of FFT+NNLIN with weighted and unweighted parameter spaces and $c = 15$. The solutions and its ECI value are given in table 3.9. Combining the two tables, we do the following observations:

(i) The errors of the solutions are lower with a test set of 100 points than with 200 points. Moreover, these losses are better than the ones that the non-hybrid optimization attained and the ones from FFT+LIN in appendix B.2.1.

(ii) The optimization in phase 1 is done in less time than the optimization with COBYLA in table 3.1, but the true optimization (phase 2) did not gain much from the first phase in time. Only with $|L| = 100$ and data set Almere there is a significant reduction in number of function evaluations.

(iii) The function evaluations in phase 2 are for All Situations much less than for COBYLA (113), bur for Almere they are approximately the same.

(iv) Although the solutions of this hybrid optimization algorithm on a test set of 100 data points are lower in function value (3.2) than the result from 'normal' COBYLA, the ECI values are higher.

	Almere				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	$l(x^*)$
100	97	12.17	65	30.04	5.02
200	63	11.65	100	35.44	6.12

	All Situations				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	$l(x^*)$
100	76	17.01	100	28.51	1.62
200	203	21.06	98	33.48	1.75

Table 3.5: Number of function evaluations and optimization time in phase 1 and phase 2 of the hybrid optimization method FFT+NNLIN in seconds and the function value of the result x^*, for test sets both from the weighted parameter spaces.

Data set	x^*	**ECI**
Almere:	(19.42, 19.64, 100.71, 101.31, 29.75)	14.45
All Situations:	(38.92, 32.38, 97.70, 97.30, 41.53)	16.26
Test Model 1:	(15.57, 19.65, 101.86, 101.24, 29.38)	14.40
Test Model 2:	(19.06, 8.72 , 100.95, 97.96, 31.97)	14.42

Table 3.7: Solutions and corresponding ECIs of COBYLA on loss function (3.2).

	Test Model 1				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	$l(x^*)$
100	69	10.86	74	34.58	5.19
200	83	11.86	71	24.89	5.36

	Test Model 2				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	$l(x^*)$
100	81	10.19	78	37.26	4.48
200	88	11.57	72	24.99	4.46

Table 3.6: Number of function evaluations and optimization time in phase 1 and phase 2 of the hybrid optimization method FFT+NNLIN in seconds and the function value of the result x^*, for test sets both from the weighted parameter spaces.

3.3 Conclusion

The hybrid optimization method results in lower function values, but there is no huge improvement in the number of iterations. Since the solutions of COBYLA on loss function (3.2) were already tolerable, a lower function value is not necessary. The simplex method is by far the fastest and while other methods result in lower function values, it returns a sufficient design. Consequently, COBYLA without any tricks for initialization is the best method for this particular problem.

In next chapter we will compare the solutions of this optimization problem with the optimization of the environmental impact. The solutions of COBYLA on objective (3.2) for all four data sets are given in table 3.7. The solutions of the hybrid optimization are given in table 3.9.

	Almere		All Situations	
$\|L\|$	x^*	**ECI**	x^*	**ECI**
100	(17.39, 19.51, 102.42 , 100.29, 31.16)	14.49	(37.54 , 33.57, 98.47 , 101.18, 40.77)	16.57
200	(19.34, 19.02, 100.74, 100.85, 29.73)	14.43	(37.27, 32.08, 98.37, 98.59, 42.44)	15.92

Table 3.8: Result x^* and its environmental cost indicator of the hybrid optimization with FFT+NNLIN surrogate model, for test sets both from the weighted parameter spaces and the original.

	Test Model 1		Test Model 2	
$\|L\|$	x^*	**ECI**	x^*	**ECI**
100	(15.62, 16.63, 102.28, 101.40, 29.36)	14.36	(16.65, 8.93, 102.55, 99.96, 31.09)	14.34
200	(15.40, 18.07, 99.93, 102.36, 31.09)	14.34	(18.82, 9.08, 101.93, 101.86, 26.63)	14.29

Table 3.9: Result x^* and its environmental cost indicator of the hybrid optimization with FFT+NNLIN surrogate model, for test sets both from the weighted parameter spaces and the original.

3.4 Discussion

Surprisingly, the solutions of the hybrid optimization with a test set of 200 points have a higher ECI value than with a smaller test set, as can be seen in table 3.6. Although when reducing the number of data points further, the results get worse again. It is still unclear what causes this, as it is not what one would expect. From previous chapter we know that a larger test set does lead to better approximation of the diagrams with FFT+NNLIN.

While this objective produces safe designs, it is not the best fitness measurement. As we mentioned earlier, not *all* points have to be as close as possible. For example, sometimes all points end up on one side of a very large onion, see figure 3.3. The initial guess has great influence on which solution the method converges to. A local optimum is good enough or even a solution in some neighborhood of a minimum but it is quite vague when a solution is good and when not. Therefore we add something to the objective, namely the environmental impact, measured in ECI.

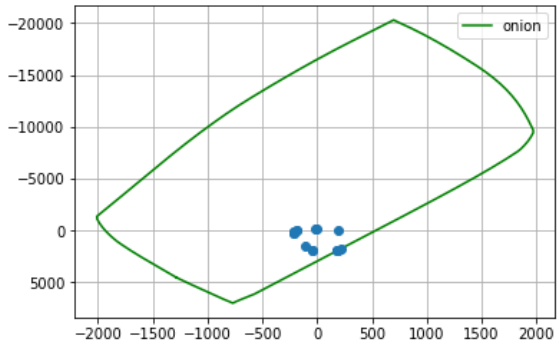


Figure 3.3: Local optimum of optimization on objective (3.2).

Chapter 4

Optimization based on ECI

Minimizing our environmental impact is an important task these days. In practice, environmental sustainability is seldomly considered during the design phase of concrete structures. Which is why having the ECI optimization functionality in UC1 would be a great asset to the product. In the sequel we discuss the optimization of the reinforcement and concrete strength when concerning the environmental cost indicator (ECI). We first define the objective, then the methods that we use are explained and the numerical results are given. The results are compared to the outcomes of the optimization on design margins, to see if the intuitive way of searching for a good solution performs well concerning the ECI cost. In the end we state the conclusion and the discussions.

4.1 Defining the problem

The constraints in this problem are equivalent to the constraints in the optimization based on the design margins, namely all our FEM points have to lie inside the onion and we still have the same bounds on the design variables. What changes is the loss function. In the previous chapter all design margins had to be as close as possible to the diagram, which resulted in some problems in the analysis. In this chapter we minimize a different measurement, the environmental cost indicator. This ECI is a shadow price of the environmental impact and thus a lower cost is always better. This brings up the following optimization problem

$$\begin{array}{lcl} \min & \mathrm{ECI}(x) & \\ \text{s.t.} & uc(i) \leq 1 & \text{for all } i \in MN \\ & 6 \leq d_i \leq 40 & \text{for } i = 1, 2 \\ & 75 \leq s_i \leq 300 & \text{for } i = 1, 2 \\ & 12 \leq f_{\mathrm{ck}} \leq 90 & \end{array} \tag{4.1}$$

and its solutions are of the form $x = (d_1, d_2, s_1, s_2, f_{\mathrm{ck}})$.

One might wonder if it is still necessary to compute each design margin. The only thing that needs to be checked is whether each point falls inside the M, N-diagram. On that account, a simple check might suffice. The issue is that if the constraint function only returns a binary value (*true* and *false* or 0 and 1) and the initial guess is infeasible, the algorithm does not know in which direction it can move to improve and eventually become feasible, as each infeasible solution is equally bad. Better is to measure the distance from onion to a point as constraint measure, then the constraint function value decreases when the onion approaches an outlying FEM point. Distance function $d(i, \text{onion})$ is equal to the distance from point i to the onion if i lies outside the onion and is minus one times the distance if i is inside the onion.

As in chapter 2, we can apply the penalty method to the first constraint, the loss function becomes

$$l(x) = \text{ECI}(x) + \lambda \sum_{i \in MN} \max\{0, d(i, \text{onion})\}). \tag{4.2}$$

To smoothen the kink in the function that is produced by taking the maximum, we again replace the absolute function by its smooth approximation $g_\alpha(x) = \sqrt{x^2 + \alpha}$, this yields the following smoother loss function:

$$l_{\text{smooth}}(x) = \text{ECI}(x) + \lambda \sum_{i \in MN} \frac{1}{2}(d(i, \text{onion}) + \sqrt{(d(i, \text{onion}))^2 + \alpha}). \tag{4.3}$$

4.2 Results

We follow the same route as in section 3.2, first selecting good hyperparameters for the two loss functions, then applying the minimization methods from the SciPy library and in the end the hybrid optimization method. This time it is more important to find the true global minimum compared to the problem in previous chapter.

4.2.1 Hyperparameters

First we look at the penalty for constraint violation λ. Again we do optimization with multiple values of λ, the results are shown in appendix C.1.1. For the Almere input we prefer $\lambda > 25$, since the constraint violation is negligible for these values for λ. For All Situations there is no clear best value for λ, as there is no point from which the violation always is zero. This is checked up to $\lambda = 1e9$. Hence we take $\lambda = 30$.

Now we want to determine a good smoothing parameter α for loss function (4.3). The

results are given in appendix C.1.2, the constraint violation is 0.0 for all α's. It seems that values that lead to a low ECI need a high number of function evaluations and vice versa. Hence it is important to look at a good trade-off between the optimal function value and number of function evaluations. Somewhere in the middle we choose $\alpha = 0.01$.

4.2.2 Method comparison

In previous chapter a lower function value did not always mean a better design, since we are now minimizing the environmental cost a lower cost value is important. This shifts the goal from only speed to a best possible solution in preferably a short amount of time. In this subsection the different optimizers are compared.

Existing methods

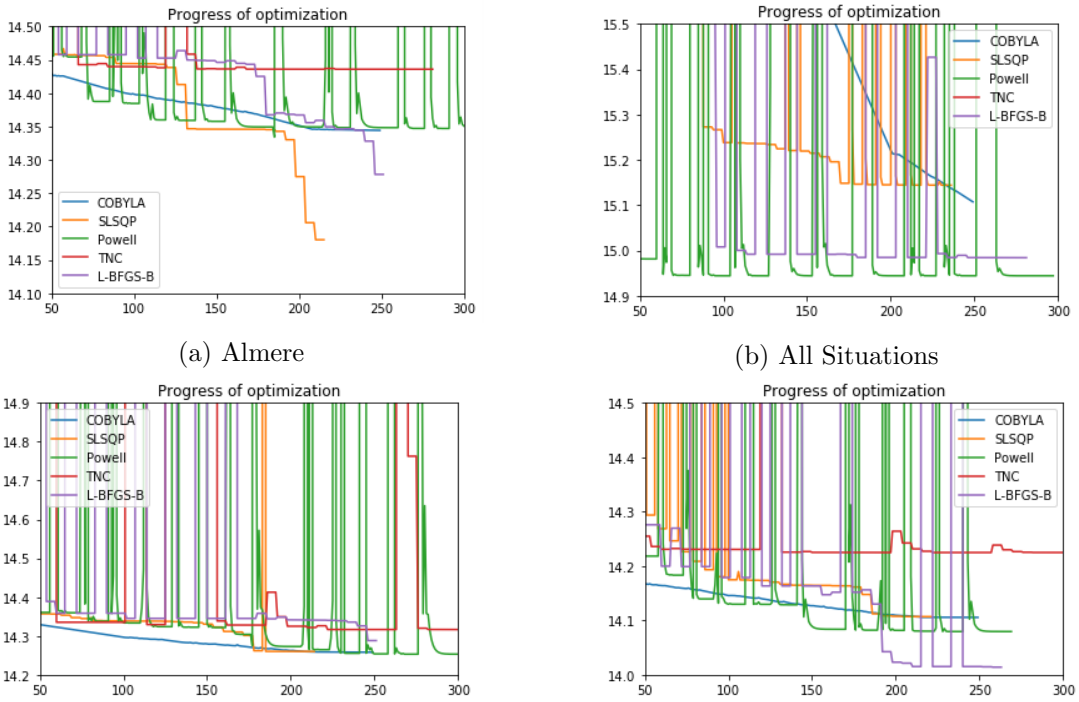
Tables 4.1 and 4.2 show the performance of SciPy minimization methods on the four data sets. It is obvious that COBYLA is the winner for Almere, Test Model 1 and 2 as it attains the (second) best solution and it requires the lowest number of function evaluations. For All Situations this is not entirely the case, COBYLA is fast, but it ends up in a bad solution. Powell would be the best choice for truly minimizing the ECI, but a waiting time of more than two minutes is not desirable. Overall, we would recommend COBYLA on this objective, the difference on Test Model 1 between the best and second best solution is minimal and we view All Situations here as being an outsider.

Now we look at the performance of these methods on the smoothed version of the objective (4.3), this is shown in table 4.3 and 4.4. We observe the following things:

(i) for most of the methods, the ECI values of the solutions are much lower than with loss function (4.2). Especially with data set All Situations.

(ii) The methods require a lot of function evaluations.

Both of these can possibly be explained by the fact that the kink at the minima in (4.2) is smoothed, which makes it easier to find the minimum. The kink might make the method "overestimate" the minimum and jump too far to the other side. Because of the change in shape around minimum it makes sense to change the stopping criteria. Observation (i) is positive, but (ii) is a problem. Therefore we fix a time limit. In agreement with the people at Movares we set this at 60 seconds, this is approximately 250 function evaluations. The question is: which method can find the best solution in 250 function evaluations? A plot of the progress on all four data sets is given in figure 4.1. On average, L-BFGS-B performs best.



(a) Almere

(b) All Situations

(c) Test Model 1

(d) Test Model 2

Figure 4.1: Progress of SciPy optimization methods on smooth objective (4.3).

	Almere			All Situations		
Method	**Evaluations**	**Time**	ECI(x^*)	**Evaluations**	**Time**	ECI(x^*)
COBYLA	88	23.31	14.43	113	28.01	16.32
SLSQP	139	39.03	14.45	355	86.84	15.23
Powell	302	81.75	14.43	579	140.24	14.97
Truncated Newton	606	171.85	14.45	606	153.90	15.91
L-BFGS-B	522	140.55	14.45	456	112.20	15.00

Table 4.1: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. Both on the Almere input and the All Situations input with objective (4.2) and $\lambda = 30$.

	Test Model 1			Test Model 2		
Method	**Evaluations**	**Time**	ECI(x^*)	**Evaluations**	**Time**	ECI(x^*)
COBYLA	96	27.46	14.31	106	27.59	14.14
SLSQP	213	52.14	14.33	279	71.24	14.15
Powell	305	81.46	14.41	302	82.01	14.31
Truncated Newton	606	170.94	14.29	222	58.45	14.15
L-BFGS-B	774	184.65	14.33	528	141.54	14.16

Table 4.2: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. On the Test Models input with objective (4.2) and $\lambda = 30$.

Hybrid optimization

The hybrid optimization applies method FFT+NNLIN, as this attained the lowest cross-validation errors. During the neighbor search the dimensions are scaled with the weights in table 2.1. This time, due to the good results in the previous subsection, we also tried the combination with COBYLA in the first phase and SLSQP in the second phase. COBYLA has a fast convergence, which is ideal for our initialization phase, SLSQP takes more time but reaches a low ECI. Hopefully a good initial guess will improve the performance of this quasi-Newtonian method.

The performance in function evaluations, time and ECI of the solution with the two-phased optimization using COBYLA in both of the phases (COBYLA+COBYLA) are given in table 4.7, results of COBYLA+SLSQP can be found in table 4.8. As in chapter 2, the smaller test set attains better results (except on input All Situations), so in the comparison we only consider the performance when $|L| = 100$. We can see the following two things:

- When optimizing with COBYLA+COBYLA, second phase COBYLA achieves better

	Almere			All Situations		
Method	**Evaluations**	**Time**	ECI(x^*)	**Evaluations**	**Time**	ECI(x^*)
COBYLA	1000	243.49	14.30	413	106.11	14.97
SLSQP	323	75.25	14.14	641	148.17	14.94
Powell	527	132.41	14.32	298	70.84	14.93
Truncated Newton	156	37.27	14.43	600	141.69	15.08
L-BFGS-B	522	125.37	14.14	642	157.85	14.98

Table 4.3: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. Both on the Almere input and the All Situations input with objective (4.3), $\lambda = 30$ and $\alpha = 0.01$.

	Test Model 1			Test Model 2		
Method	**Evaluations**	**Time**	ECI(x^*)	**Evaluations**	**Time**	ECI(x^*)
COBYLA	1000	274.31	14.25	1000	257.84	14.10
SLSQP	402	106.52	14.10	368	89.82	14.01
Powell	659	176.38	14.25	582	151.96	14.01
Truncated Newton	384	96.36	14.32	606	152.72	14.22
L-BFGS-B	732	201.68	14.10	576	147.72	14.01

Table 4.4: Performance in number of function evaluations, running time and the best found loss function value with optimization methods from the SciPy library, with the default settings. On the Test Models input with objective (4.3), $\lambda = 30$ and $\alpha = 0.01$.

solution in less function evaluations than without the pre-conditioner for most data sets.

- COBYLA+SLSQP does not always yield a better solution than straight SLSQP in the same number of function evaluations. In the cases it does attain a better solution, this does not differ much from the direct SLSQP and there is no significant speed-up.

4.2.3 Solutions

Here we look at some of the solutions and do a comparison with the solutions of the optimization based on the design margins. One can find the solutions of COBYLA on loss function (4.2) in table 4.5 and the solutions of SLSQP on loss function (4.3) in table 4.6. Solutions of the hybrid optimization can be found in appendix C.2.2. Recall that SLSQP on the smoothed objective obtained the best results among these. Comparing all results we observe that solutions with lower ECI have

(i) concrete strength as small as possible,

(ii) large diameter for the bars of steel.

This is due to the fact that a lower concrete strength means a lower ECI for concrete, but having a really low concrete strength can lead to an insufficient design, so we then need more reinforcement.

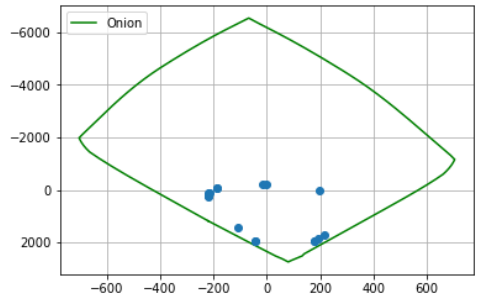
In most cases, the ECI values of soltuions in this chapter are lower than the ECIs in the optimization based on the design margins. Figure 4.2 shows the diagrams corresponding to the solutions in table 4.6. The diagrams look pretty similar to a good result if we really try to fit the onion as tight as possible around the points, with the extra constraint to have concrete strength as small as possible.

Data set	x^*	**ECI**
Almere:	(19.00, 19.65, 101.39, 101.33, 29.01)	14.43
All Situations:	(37.69, 32.03, 98.16, 97.16, 42.05)	16.32
Test Model 1:	(15.70, 14.20, 101.69 , 101.87, 27.03)	14.38
Test Model 2:	(8.78, 9.92, 102.87, 102.95, 22.65)	14.16

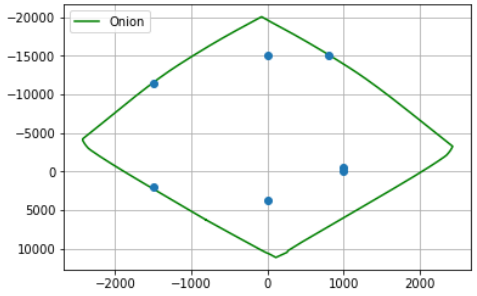
Table 4.5: Solutions and corresponding ECIs of COBYLA on loss function (4.2).

Data set	x^*	**ECI**
Almere:	(29.50, 36.24, 272.02, 300.00, 12.00)	14.15
All Situations:	(40.00, 40.00, 108.61, 99.63, 30.01)	14.94
Test Model 1:	(29.55, 25.62, 300.00, 283.42, 12.00)	14.10
Test Model 2:	(15.80, 17.88, 287.99, 300.00, 12.00)	14.01

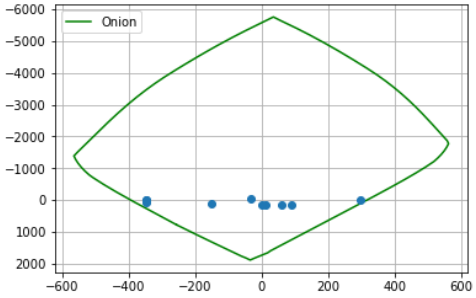
Table 4.6: Solutions and corresponding ECIs of SLSQP on loss function (4.3) without extra stopping criteria.



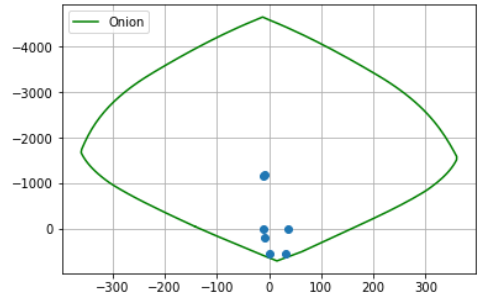
(a) Almere



(b) All Situations



(c) Test Model 1



(d) Test Model 2

Figure 4.2: Solution of SLSQP on smooth objective (4.3).

	Almere				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	111	10.01	69	19.17	14.400
200	74	8.15	81	22.68	14.399

	All Situations				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	92	4.91	97	20.83	16.17
200	123	6.62	87	20.54	15.67

	Test Model 1				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	81	5.32	82	21.57	14.34
200	89	6.53	83	20.07	14.31

	Test Model 2				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	85	3.78	91	21.73	14.14
200	101	4.75	83	19.13	14.15

Table 4.7: Number of function evaluations and optimization time in phase 1 COBYLA and phase 2 COBYLA of the hybrid optimization method FFT+NNLIN in seconds and the function value of the result x^*, for test sets from the weighted parameter spaces.

	Almere				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	96	8.20	158	42.83	14.40
200	74	6.15	156	42.68	14.34

	All Situations				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	149	8.20	170	41.83	14.98
200	123	6.62	179	49.54	14.98

	Test Model 1				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	81	5.46	159	40.08	14.21
200	89	6.50	153	40.47	14.27

	Test Model 2				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	ECI(x^*)
100	321	13.74	76	17.66	14.11
200	101	4.73	89	20.679	14.14

Table 4.8: Number of function evaluations and optimization time in phase 1 COBYLA and phase 2 SLSQP (max iter = 25) of the hybrid optimization method FFT+NNLIN in seconds and the function value of the result x^*, for test sets from the weighted parameter spaces.

4.3 Conclusion

After mimicking the manual way an engineer would optimize the design of a reinforced concrete structure, we added the environmental cost indicator to the objective. Adding the ECI to the objective definitely gave more sustainable designs. The initialization phase in the hybrid method helped COBYLA to find a better solution in less time. However, the quasi-Newton method did not gain anything from the preconditioned initial guess.

Note that when minimizing the smoothed objective (4.3), the quasi-Newton methods obtained outstanding results. The rest of the methods terminated because of a maximum number of iterations, which means that they did not manage to converge to a minimum. Also the solutions with lowest error had as small concrete strength as possible. Which makes us believe there is possibly one true minimum.

4.3.1 Recommendations

Benchmarking our hybrid protocol on the four data sets, we conclude the following:

(i) If the main goal is speed and the ECI not so much: hybrid optimization with FFT+NNLIN and COBYLA in both phases on non-smooth function with $|L| = 100$ is the way to go.

(ii) If we aim to find the best ECI: quasi-Newton methods on the smoothed function perform best. One can possibly speed-up by setting maximum number of evaluations, as there is little change in the last few iterations.

Since looking after the environment is such an important task, we would recommend to take option (ii).

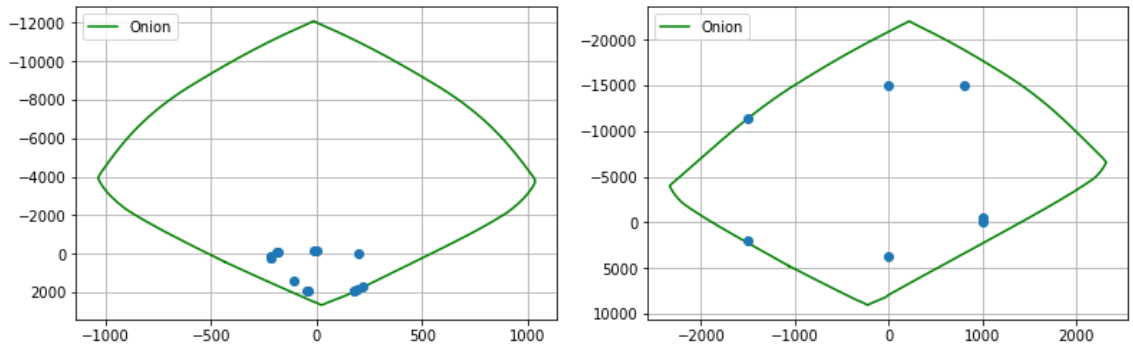


Figure 4.3: Solution loss function (4.2) with COBYLA for Almere and All Situations.

4.4 Discussion

We saw that when optimizing the environmental sustainability, the best solutions had a low concrete strength. This could be exploited in some heuristic that minimizes the concrete strength. This changes when there are extra requirements on the concrete strength. Sometimes a higher concrete strength is necessary for durability or one party demands a fixed value. Then we do not find a rule for a good solution.

Chapter 5

Concluding remarks and future work

5.1 Conclusions

The main contributions of this thesis is the hybrid optimization technique using a fast estimation of the ULS diagram of the concrete design. We showed that the FFT+NNLIN approximation method produces high quality estimations.

The problem of fitting the onion as tight as possible around the points can be solved in a relatively short amount of time. Good initial guess influences which local minimum we end up in, which is a good motivation to use the hybrid technique. Eventually this objective was too vague and we added another measure to the problem, namely the environmental cost.

Benchmarking our method on several real world datasets revealed that having data-assimilated preconditioner not only improves environmental costs but also leads to designs with better mechanical properties in less time when using the simplex method COBYLA. Quasi-Newton methods were able to achieve better solutions, although they require more time. When looking for the minimum in problem (4.1), a quasi-Newton method is recommended. We discovered that when concrete strength is one of the design variables and we minimize the ECI, we could also try to minimize the concrete strength.

5.2 Outlook

We close off with some problems still open for future research. There are at least three subjects that are not considered in this thesis, while the optimization functionality in UC1 is already able to handle these. Since it has not been a part of the research we are unable

to claim anything about the performance of the recommended optimization method. This holds for the following three:

(i) optimization with respect to the Serviceability Limit State (SLS) diagram instead of the ULS diagram,

(ii) optimizing the design of a concrete element with a non-rectangular cross-section,

(iii) having more or less than two layers of reinforcement.

Possibly clause (i) is similar to the current problem, it is just another diagram that we can try to approximate using these data assimilation techniques.

Another thing that is interesting to look at is testing the sensitivity of the solutions. Not every continuous value is admissible for the design variables. For example there can be a number of standard sizes of the bars. As we mentioned in the introduction, this is a subject that has been studied earlier, but we did not consider it in this project.

As mentioned in the discussion of chapter 3, it seems that in our particular ECI optimization problem there is one minimum. Fixing the concrete strength changes the problem and might create multiple local minimum. It would be interesting to see how the methods in this project would behave.

5.2.1 Further improvements of surrogate model

One of our tasks was to construct a good approximation of a ULS diagram. We found that there is still room to improve the FFT+NNLIN method by viewing it as a problem of neighbor selection. How this works and how we discovered this is explained in this last subsection.

Neighbor selection

Recall the FFT+NNLIN method, where we build a linear model on the k nearest neighbors. This made us wonder whether there exists a better subset of points in the test set to build a linear model on. Yielding the optimization problem of selecting a good subset of neighbors. To test this, we let a genetic algorithm GA run on this problem. A review on genetic algorithms is given in [20]. The idea behind a genetic algorithm is evolution. Initially a first population is randomly generated. A population consists of n_{pop} individuals that are binary numbers of length n_{b} corresponding to a subset of neighbors. If the ith bit is 1, then the ith neighbor is used for regression, if 0 then not. Based on some fitness function, good individuals are used for reproduction to create the new population. The procedure of finding individuals for reproduction is called *selection*. Reproduction is done with *cross-over* and *mutation*.

Our fitness function is defined as the MSE between the true FFT coefficients and the approximation from the linear model when building it on the given subset of neighbors. Selection is done by tournament selection, we take a small number of individuals from the current population and select the one with best function value for reproduction. Pros of tournament selection compared to other selection protocols is that is preserves diversity in order to prevent premature convergence. When we have the desired size of the population, we take the first two individuals and with probability $r_c \leq 1$ cross-over is applied, this rate is usually pretty high like 0.8 or 0.9. Let the first two individuals be b_1 and b_2, cross-over does the following:

(i) get a random number R between 1 and the number of bits, this is our cross-over point,

(ii) the first new individual b_1' is given by

$$b_1'[1, ..., R] = b_1[1, ..., R],$$

$$b_1'[R+1, ..., n_b] = b_2[R+1, ..., n_b],$$

(iii) the second new individual b_2' is given by

$$b_2'[1, ..., R] = b_2[1, ..., R],$$

$$b_2'[R+1, ..., n_b] = b_1[R+1, ..., n_b].$$

Then for every bit we flip its value from 0 to 1 or from 1 to 0 with a small probability r_m (typically $\frac{1}{n_b}$).

We did cross-validation with the GA, where we use the GA to find a good subset of its n_b nearest neighbors for FFT+NNLIN for each point in the test set. For $c = 30, |L| = 200, n_b = 30, n_{pop} = 50, n_{iter} = 10, r_c = 0.9$ and $r_m = 1/30$ the cross-validation MSE is 430.74 and maximum error is 81.90. When doing FFT+NNLIN and taking all neighbors in a sphere around the points in scaled dimensions the cross-validation MSE was equal to 7692.57 and the maximum error was 447.72. Note that the difference here is huge. We made an attempt to find out why the GA finds such a good subset of neighbors or find a pattern in when a subset is either good or bad. Unfortunately, we did not manage to do this, but the results of the GA are a proof that it is possible to obtain very good estimations.

Bibliography

[1] David M Allen. The relationship between variable selection and data agumentation and a method for prediction. *technometrics*, 16(1):125–127, 1974.

[2] David A Belsley, Edwin Kuh, and Roy E Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*, volume 571. John Wiley & Sons, 2005.

[3] Albert S Berahas, Richard H Byrd, and Jorge Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM Journal on Optimization*, 29(2):965–993, 2019.

[4] Dmitriy Bilyk, Michael T Lacey, and Armen Vagharshakyan. On the small ball inequality in all dimensions. *Journal of Functional Analysis*, 254(9):2470–2502, 2008.

[5] Charles G Broyden. Quasi-newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

[6] Martin Buhmann and Nira Dyn. Spectral convergence of multiquadric interpolation. *Proceedings of the Edinburgh Mathematical Society*, 36(2):319–333, 1993.

[7] Martin D Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge university press, 2003.

[8] Charles V Camp, Shahram Pezeshk, and Håkan Hansson. Flexural design of reinforced concrete frames using a genetic algorithm. *Journal of structural engineering*, 129(1):105–115, 2003.

[9] MZ Cohn and AS Dinovitzer. Application of structural optimization. *Journal of Structural Engineering*, 120(2):617–650, 1994.

[10] Guilherme Fleith de Medeiros and Moacir Kripka. Optimization of reinforced concrete columns according to different environmental impact assessment parameters. *Engineering Structures*, 59:185–194, 2014.

[11] John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[12] Paul Daniel Dumitru, Marin Plopeanu, and Dragos Badea. Comparative study regarding the methods of interpolation. *Recent advances in geodesy and Geomatics engineering*, 1:45–52, 2013.

[13] Gregory E Fasshauer and Jack G Zhang. On choosing "optimal" shape parameters for rbf approximation. *Numerical Algorithms*, 45(1):345–368, 2007.

[14] Richard Franke. Scattered data interpolation: tests of some methods. *Mathematics of computation*, 38(157):181–200, 1982.

[15] Robert M Freund. Penalty and barrier methods for constrained optimization. *Lecture Notes, Massachusetts Institute of Technology*, 2004.

[16] Andres Guerra and Panos D Kiousis. *Design optimization of reinforced concrete structures*. PhD thesis, Colorado School of Mines, 2004.

[17] John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[18] Ajay Harish. Finite Element Method – What Is It? FEM and FEA Explained. `https://www.simscale.com/blog/2016/10/what-is-finite-element-method/`, 2020. [Online; accessed 22-July-2021].

[19] Luc Hillege. Milieukostenindicator (MKI) – Overzicht. `https://ecochain.com/nl/knowledge-nl/milieukosten-indicator-mki/`, 2019. [Online; accessed 23-July-2021].

[20] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, 2021.

[21] Tamara G Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[22] Charles L Lawson and Richard J Hanson. *Solving least squares problems*. SIAM, 1995.

[23] C Lee and J Ahn. Flexural design of reinforced concrete frames by genetic algorithm. *Journal of structural engineering*, 129(6):762–774, 2003.

[24] Gopal Mishra. Limit states of steel design. `https://theconstructor.org/structural-engg/limit-states-of-steel-design/6879/`. [Online; accessed 26-July-2021].

[25] Joseph Muscat. *Functional analysis: an introduction to metric spaces, Hilbert spaces, and Banach algebras*. Springer, 2014.

[26] Harald Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of number theory*, 30(1):51–70, 1988.

[27] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.

[28] Ignacio Paya, Victor Yepes, Fernando González-Vidosa, and Antonio Hospitaler. Multiobjective optimization of concrete frames by simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, 23(8):596–610, 2008.

[29] Ignacio Paya-Zaforteza, Victor Yepes, Antonio Hospitaler, and Fernando Gonzalez-Vidosa. Co2-optimization of reinforced concrete frames by simulated annealing. *Engineering Structures*, 31(7):1501–1508, 2009.

[30] Boris T Polyak. Newton's method and its use in optimization. *European Journal of Operational Research*, 181(3):1086–1096, 2007.

[31] Michael JD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

[32] Michael JD Powell. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications*, 43(5):170–174, 2007.

[33] S Rajeev and CS Krishnamoorthy. Genetic algorithm–based methodology for design optimization of reinforced concrete frames. *Computer-Aided Civil and Infrastructure Engineering*, 13(1):63–74, 1998.

[34] Shmuel Rippa. An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advances in Computational Mathematics*, 11(2):193–210, 1999.

[35] Scott A Sarra and Derek Sturgill. A random variable shape parameter strategy for radial basis function approximation methods. *Engineering Analysis with Boundary Elements*, 33(11):1239–1245, 2009.

[36] Martin Schlüter, Matthias Gerdts, and Jan-J Rückmann. A numerical study of midaco on 100 minlp benchmarks. *Optimization*, 61(7):873–900, 2012.

[37] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, 1968.

[38] Andreia Fatima Tormen, Zacarias Martin Chamberlain Pravia, Fernando Busato Ramires, and Moacir Kripka. Optimization of steel-concrete composite beams considering cost and environmental impact. *Steel and Composite Structures*, 34(3):409–421, 2020.

[39] JG Wang and GRs Liu. On the optimal shape parameters of radial basis functions used for 2-d meshless methods. *Computer methods in applied mechanics and engineering*, 191(23-24):2611–2630, 2002.

[40] Eric W Weisstein. L^ infty-norm. `https://mathworld.wolfram.com/L-Infinity-Norm.html`, 2000. [Online; accessed 24-August-2021].

[41] Ya-xiang Yuan. A review of trust region algorithms for optimization. In *Iciam*, volume 99, pages 271–282, 2000.

[42] Antoni Zygmund. *Trigonometric series*, volume 1. Cambridge university press, 2002.

Appendix A

Tuning the optimization methods

This appendix mainly shows the results of the estimation methods that are not included in chapter 2. Also there is one subsection about the construction of the linear model in FFT+LIN.

A.1 Neighbor interpolation

	Halton sequence				Random sequence			
$\|L\|$	$d_{\max}$	p	**MSE**	**Maximum error**	$d_{\max}$	p	**MSE**	**Maximum error**
200	53	1	4724862.93	4362.72	-	-	-	-
300	41	2.5	4239120.37	3623.42	54	4	6320638.94	5064.09
400	41	3	4088917.20	3510.49	53	3	5920205.32	5049.23
600	41	3	3808484.22	3405.36	46	3.5	5422098.19	4714.03

Table A.1: The neighborhood radius $d_{\max}$ and p with the lowest cross-validation error of the neighbor interpolation for several test set sizes $|L|$ and the resulting cross-validation MSE and max error.

$\|L\|$	$d_{\max}$	p	**MSE**	**Maximum error**
200	87	2.9	3761140.90	2910
300	94	2.9	3732677.85	2999
400	94	2.9	3748428.46	2998

Table A.2: The neighborhood radius $d_{\max}$ and p with the lowest cross-validation error of the neighbor interpolation with Halton generated test set, scaled to the weights, for several test set sizes $|\tilde{L}|$ and the resulting cross-validation MSE and max error.

A.2 FFT+LIN

A.2.1 Linear model

The linear model is built by regression to approximate the FFT coefficients for a given point x. Suppose we are given a test set L of n data points

$$L = \begin{bmatrix} x_1^\intercal \\ x_2^\intercal \\ \vdots \\ x_n^\intercal \end{bmatrix}$$

and the matrix Z of size $n \times (4c+2)$ that consists of the corresponding FFT coefficients

$$Z = \begin{bmatrix} f_1^\intercal \\ f_2^\intercal \\ \vdots \\ f_n^\intercal \end{bmatrix}.$$

Ideally, we wish to find a matrix A such that $Ax_k = f_k$ for $k = 1, ..., n$. As it is unlikely that such a matrix A exists, we use regression to get a linear estimation. Therefore, take the kth column of Z

$$z_k = \begin{bmatrix} f_{1,k} \\ f_{2,k} \\ \vdots \\ f_{n,k} \end{bmatrix}$$

and then we try to solve $Lr_k = z_k$ for r_k. If the map F that we try to approximate is indeed linear, than this will give an exact outcome. Otherwise we can try to approximate r_k with a method like least squares [22], where the sum of squared difference between the true data z_k and Lr_k is minimized. The linear map is given by

$$A = \begin{bmatrix} r_1^\intercal \\ r_2^\intercal \\ \vdots \\ r_{4c+2}^\intercal \end{bmatrix}.$$

For a new point x, we can approximate its FFT coefficients by solving the system Ax.

A.2.2 Results

	Halton sequence		Random sequence	
c	**MSE**	**Maximum error**	**MSE**	**Maximum error**
10	34493.81	834.53	23391.25	717.65
15	23237.65	830.92	15674.80	711.32
20	17617.85	831.14	12075.63	707.78
25	14095.24	831.80	9552.82	714.76

Table A.3: Cross-validation MSE and maximum error for different number of basis functions with a test set of 100 data points.

	Halton sequence		Random sequence	
c	**MSE**	**Maximum error**	**MSE**	**Maximum error**
10	32304.73	794.37	26904.79	759.72
15	21700.05	792.05	18047.79	750.70
20	16582.69	777.16	13696.53	738.93
25	13192.09	792.60	10985.44	755.82

Table A.4: Cross-validation MSE and maximum error for different number of basis functions with a test set of 200 data points.

A.3 FFT+NI

	Halton sequence		Random sequence	
$\|L\|$	**MSE**	**Maximum error**	**MSE**	**Maximum error**
200	151727.61	2215.90	-	-
300	92736.31	1718.66	76321.73	1536.91
400	75371.67	1568.73	80149.39	1610.70
600	67779.37	1483.05	60020.59	1381.88

Table A.5: Cross-validation MSE and max error of FFT+NI for different test set sizes with the corresponding best found values $d_{\max}$ and p given in table A.1

$\|L\|$	**MSE**	**Maximum error**
200	40648.02	1050.75
300	38641.47	1022.89
400	36304.76	993.96

Table A.6: Cross-validation MSE and max error of FFT+NI for different test set sizes with scaled parameter spaces, the test set is randomly generated and $d_{\max}$ and p are the best found values given in table A.2.

A.4 FFT+RBF

		Halton sequence		Random sequence	
$\|L\|$	ε	**MSE**	**Maximum error**	**MSE**	**Maximum error**
100	0.015	65047.21	1216.42	62772.28	1122.20
200	0.023	105411.30	1482.15	63859.65	1163.72
300	0.026	88241.26	1275.61	76222.15	1173.33

Table A.7: The constant shape parameter ε with lowest maximum error in cross validation for different test set sizes $|L|$ with 15 FFT basis functions ($c = 15$).

A.5 FFT+NNLIN

Neighborhood method			k-Neighbor method		
$d_{\max}$	**MSE**	**Maximum error**	k	**MSE**	**Maximum error**
45	70642.09	796.88	10	8736.98	475.22
60	17844.15	551.64	20	7503.41	445.44
67	7712.56	469.32	30	7936.32	465.63

Table A.8: Cross-validation mean squared error and maximum error for the FFT+NNLIN methods with $|L| = 200$ generated by a random sequence.

Neighborhood method			k-Neighbor method		
$d_{\max}$	**MSE**	**Maximum error**	k	**MSE**	**Maximum error**
45	259242.29	1030.35	10	11381.42	554.54
60	9012.92	486.70	20	8577.22	484.55
67	8921.59	491.70	30	9209.78	493.85

Table A.9: Cross-validation mean squared error and maximum error for the FFT+NNLIN methods with $|L| = 200$ generated by Halton sequence.

A.6 Neighbors in different directions

Method	**MSE**	**Maximum error**
NI, det	616397.35	4132.03
NI, sum	10793.52	536.86
Gaussian, det	609897.94	4119.31
Gaussian, sum	615140.82	4115.81
Linear, det	189327928.79	21055.33
Linear, sum	987705078.33	25155.96

Table A.10: Cross-validation errors of approximation methods on the FFT coefficients with $|L| = 400$ and $c = 15$.

Appendix B

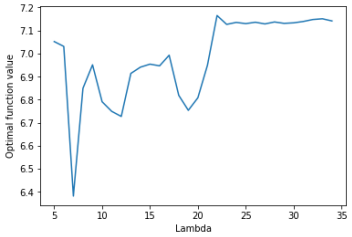
Results optimization on design margins

This chapter shows some of the results of the optimization based on design margins from chapter 3. First the results of the hyperparameters. After that some extra results of hybrid optimization methods with another estimation scheme than FFT+NNLIN, namely FFT+LIN. It is clear that these methods do not outperform the hybrid method that uses FFT+NNLIN.

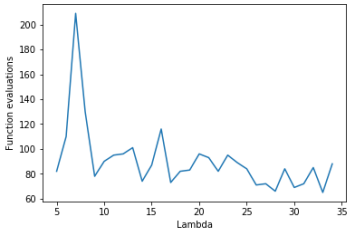
B.1 Hyperparameters in loss functions

B.1.1 Constraint violation penalty λ

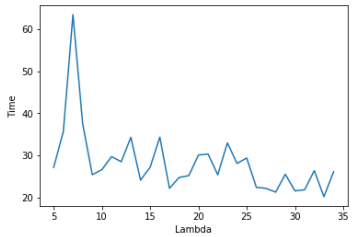
In this section we show the performance of different constraint violation penalties λ in the loss function (3.2). For the optimization is used COBYLA from the SciPy library. In figures B.1 are shown the performances of optimization on the Almere input and in figure B.3 on the All Situations input. Figure B.2 and figure B.4 show the solutions from the best penalties for Almere and All Situations respectively. We distinguish two types of optimal penalties, one that achieves the lowest minimum λ^*_{loss} and one that had the lowest running time λ^*_{time}.



(a) Optimal function value per λ.

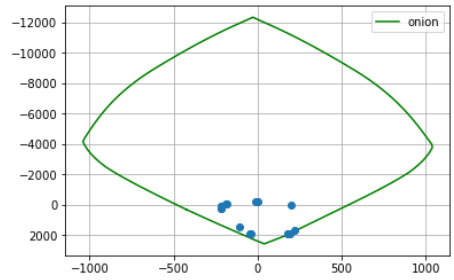


(b) Number of function evaluations per λ.

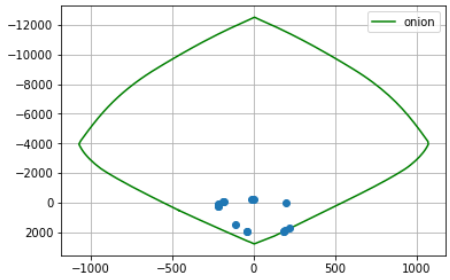


(c) Running time for the optimization per λ.

Figure B.1: Optimization with COBYLA on Almere input file for different constraint penalties λ in loss function (3.2).

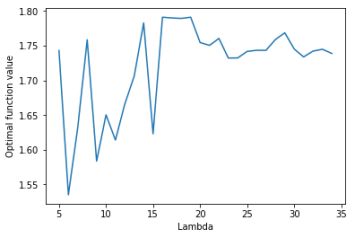


(a) Solution for penalty λ^*_{loss} with the lowest loss.

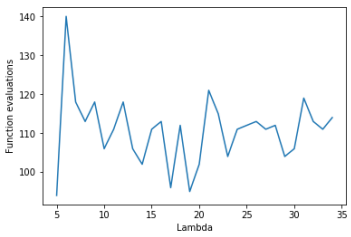


(b) Solution for penalty λ^*_{time} with the best time.

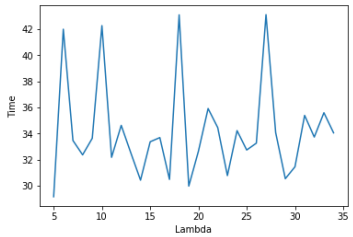
Figure B.2: Solutions with respectively the constraint penalty with the best solution in loss and the constraint penalty with the best running time in optimization with COBYLA on the Almere input.



(a) Optimal function value per λ.

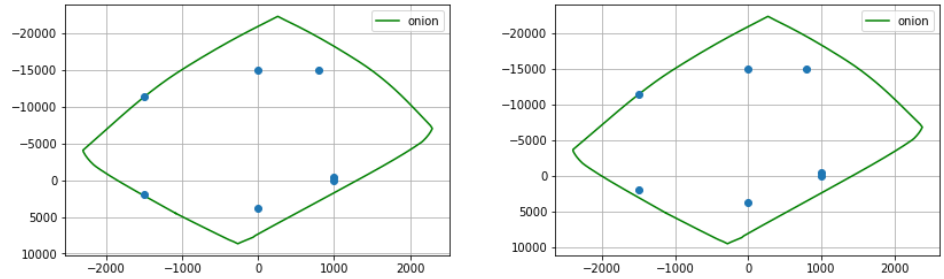


(b) Number of function evaluations per λ.



(c) Running time for the optimization per λ.

Figure B.3: Optimization with COBYLA on All Situations input file for different constraint penalties λ in loss function (3.2).

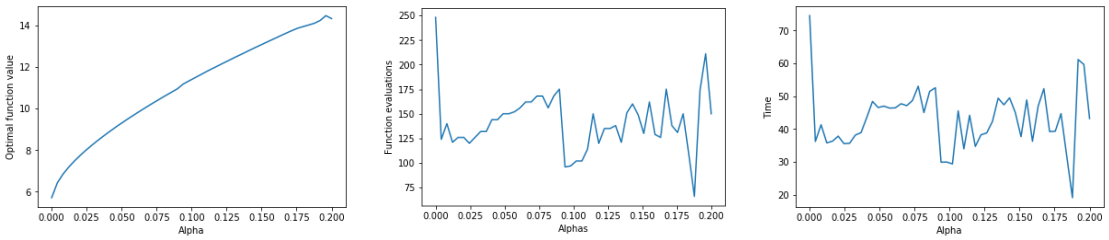


(a) Solution for penalty λ^*_{loss} with the lowest loss.

(b) Solution for penalty λ^*_{time} with the best time.

Figure B.4: Solutions with respectively the constraint penalty with the best solution in loss and the constraint penalty with the best running time in optimization with COBYLA on the All Situations input.

B.1.2 Smoothing factor α

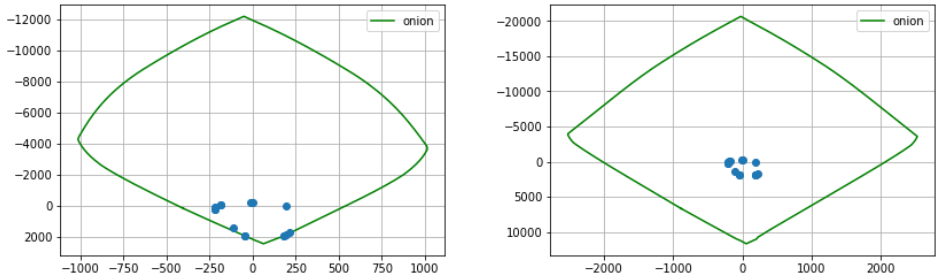


(a) Optimal function value per α.

(b) Number of function evaluations per α.

(c) Running time for the optimization per α.

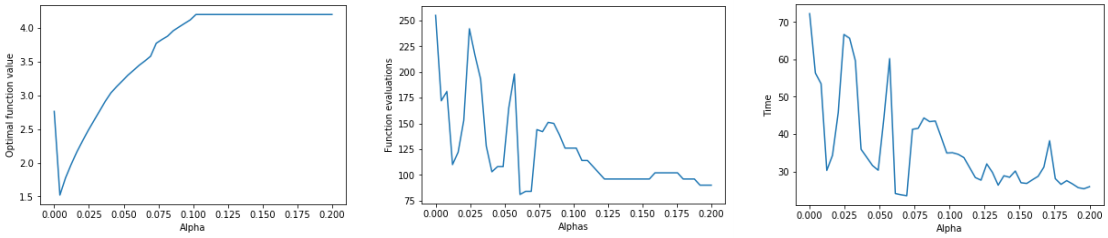
Figure B.5: Optimization with SLSQP on Almere input file for different smoothing factors α in loss function (3.3)



(a) Solution for penalty α^*_{loss} with the lowest loss.

(b) Solution for penalty α^*_{time} with the best time.

Figure B.6: Solutions with respectively the smoothing factor with the best solution in loss and the constraint penalty with the best running time in optimization with SLSQP on the Almere input.

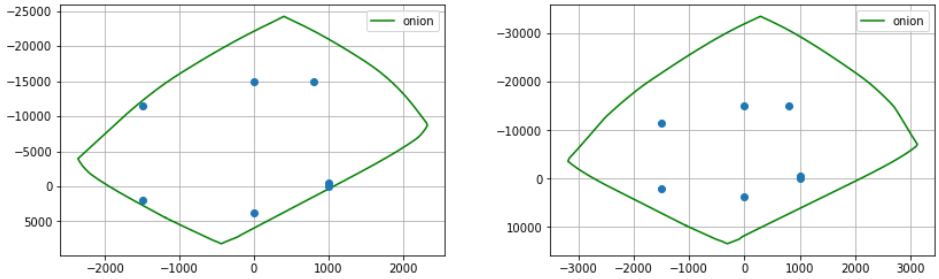


(a) Optimal function value per α.

(b) Number of function evaluations per α.

(c) Running time for the optimization per α.

Figure B.7: Optimization with SLSQP on All Situations input file for different smoothing factors α in loss function (3.3)



(a) Solution for penalty α^*_{loss} with the lowest loss.

(b) Solution for penalty α^*_{time} with the best time.

Figure B.8: Solutions with respectively the smoothing factor with the best solution in loss and the constraint penalty with the best running time in optimization with SLSQP on the All Situations input.

B.2 Optimization

B.2.1 FFT+LIN

In terms of cross-validation errors, the FFT+LIN was not a winner, but it is the quickest estimation. Therefore we do consider this in the design margin optimization. The computation time is divided in three parts: constructing the linear model, optimizing on the interpolation and optimizing on the true model. The first can be done at the startup of the program or initialization of the optimization. Thus, for this method to be competitive with COBYLA in speed, the latter two have to outperform the running time of COBYLA.

The performance of the hybrid optimization with FFT basis and a test set size of 100 is shown in table B.2 for Almere and in table B.6 for All Situations. The second, third and fourth column give the running time of constructing the surrogate model, optimization on the surrogate model and optimization on the true model respectively. The last column shows the value of the best solution found. Note that the running time of phase 1 and phase 2 summed up is larger than the running time of COBYLA in table 3.1 for both input files. This happens to be the case for all test set sizes. Although the pre-calculations in phase 1 do not give a speed up in total, the running times of phase 2 on All Situations are shorter than the straight away application of COBYLA. The optimal function values of this hybrid method are lower than before, but since we concluded that the solution of the faster method was ‘optimal’ enough and lower error did not always mean better solution, for now this is not important. Disregarding this particular problem, it does show that our hybrid method performs better as a minimization method.

c	Construct A	Phase 1	Phase 2	$l(x^*)$
10	8.31	10.25	29.92	6.52
15	9.43	8.85	28.81	5.82
20	8.46	9.65	23.09	5.70
25	8.66	8.53	25.95	6.12

Table B.1: Performance of hybrid optimization with Fourier interpolation with a test set of 50 points on Almere.

c	Construct A	Phase 1	Phase 2	$l(x*)$
10	16.87	8.625	27.78	6.31
15	16.72	7.42	20.00	6.06
20	16.64	11.48	31.11	5.88
25	16.76	8.46	23.28	6.17

Table B.2: Time to construct the linear model, time to optimize on the interpolated FFT model and time to optimize on the true model for test set size $|L| = 100$ and different values of c on Almere.

c	Construct A	Phase 1	Phase 2	$l(x^*)$
10	34.58	7.19	20.17	6.37
15	44.20	9.37	26.71	6.36
20	42.51	9.24	29.07	6.05
25	36.46	12.25	22.17	6.25

Table B.3: Performance of hybrid optimization with Fourier interpolation with a test set of 200 points on Almere.

c	Construct A	Phase 1	Phase 2	$l(x^*)$
10	51.93	10.29	22.30	6.28
15	50.40	9.13	33.13	5.76
20	53.47	7.82	27.39	5.94
25	54.49	7.48	24.97	5.84

Table B.4: Performance of hybrid optimization with Fourier interpolation with a test set of 300 points on Almere.

c	Construct A	Phase 1	Phase 2	$l(x^*)$
10	9.19	29.85	22.20	1.64
15	8.37	12.48	23.65	1.75
20	8.29	17.82	26.71	1.84
25	8.25	13.51	21.31	1.73

Table B.5: Performance of hybrid optimization with Fourier interpolation with a test set of 50 points on All Situations.

c	Construct A	Phase 1	Phase 2	$l(x*)$
10	17.26	17.37	21.60	1.67
15	20.25	17.84	22.09	1.65
20	16.65	27.98	17.40	2.11
25	16.84	15.78	31.78	1.43

Table B.6: Time to construct the linear model, time to optimize on the interpolated FFT model and time to optimize on the true model for test set size $|L| = 100$ and different values of c on All Situations.

c	Construct A	Phase 1	Phase 2	$l(x^*)$
10	34.38	18.46	21.23	1.73
15	34.53	19.01	23.01	1.80
20	33.92	17.03	21.46	1.89
25	34.00	15.92	23.00	1.57

Table B.7: Performance of hybrid optimization with Fourier interpolation with a test set of 200 points on All Situations.

c	Construct A	Phase 1	Phase 2	$l(x^*)$
10	58.28	19.31	20.43	1.79
15	51.14	19.60	22.29	1.78
20	50.22	13.42	47.62	2.12
25	50.93	13.29	27.87	1.46

Table B.8: Performance of hybrid optimization with Fourier interpolation with a test set of 300 points on All Situations.

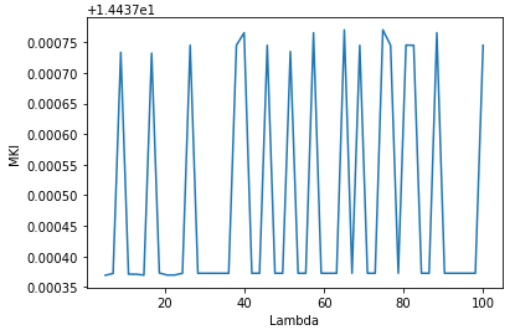
Appendix C

Results optimization on ECI

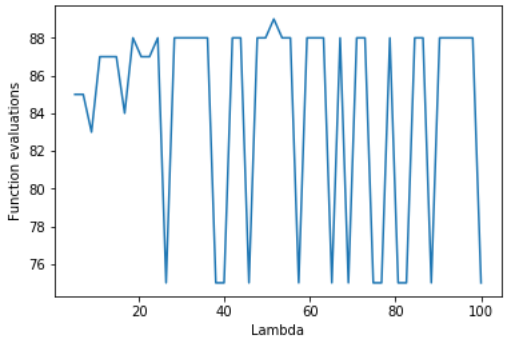
This chapter shows some of the results of the optimization based on design margins from chapter 3. First the results of the hyperparameters. After that some extra results of hybrid optimization methods with another estimation scheme than FFT+NNLIN, namely FFT+LIN. It is clear that these methods do not outperform the hybrid method that uses FFT+NNLIN. After that are the results of hybrid optimization with FFT+NNLIN with and without weighted dimensions shown, to show that the scaling does indeed benefit the search. In the end are some of the solutions that are left out of chapter 4.

C.1 Hyperparameters in loss functions

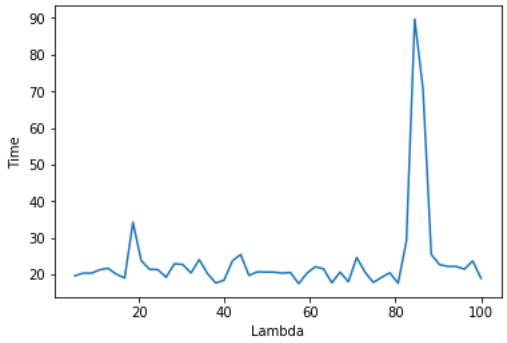
C.1.1 Constraint violation penalty λ



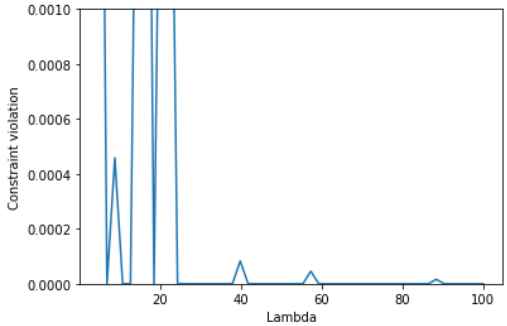
(a) Environmental cost indicator of obtained solution per λ.



(b) Number of function evaluations per λ.

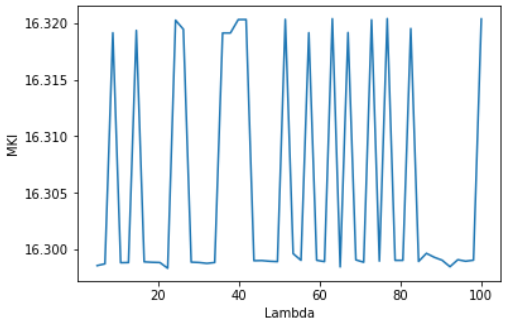


(c) Running time for the optimization per λ.

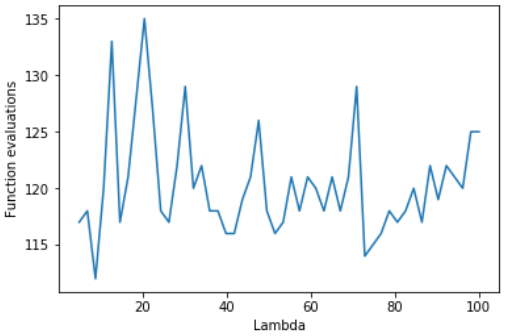


(d) Constraint violation per λ.

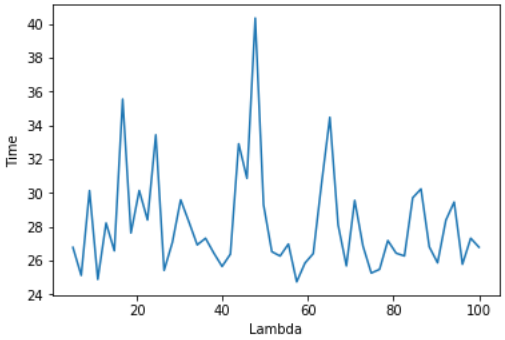
Figure C.1: Optimization with COBYLA on Almere input file for different constraint penalties λ in loss function (4.2).



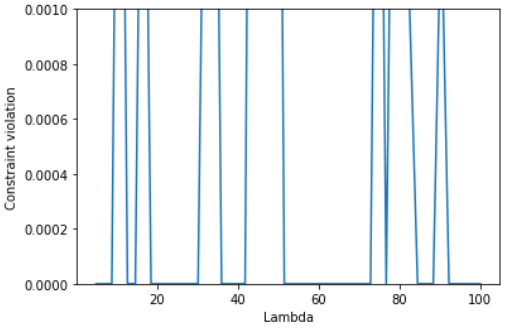
(a) Environmental cost indicator of obtained solution per λ.



(b) Number of function evaluations per λ.



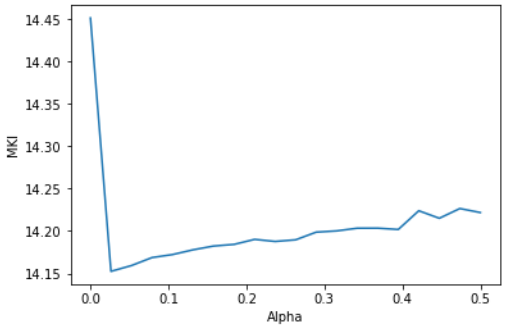
(c) Running time for the optimization per λ.



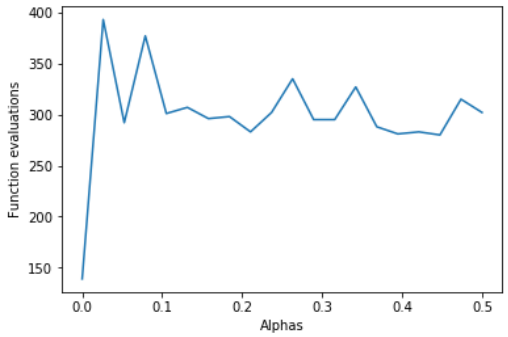
(d) Constraint violation per λ.

Figure C.2: Optimization with COBYLA on All Situations input file for different constraint penalties λ in loss function (4.2).

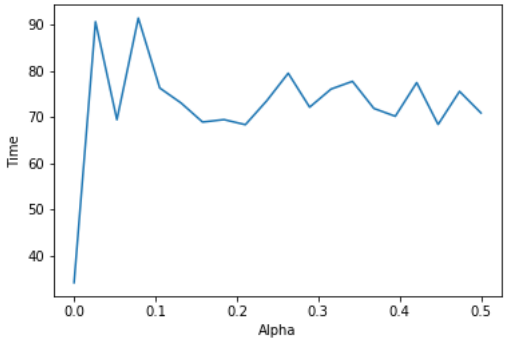
C.1.2 Smoothing parameter α



(a) Environmental cost indicator of obtained solution per α.

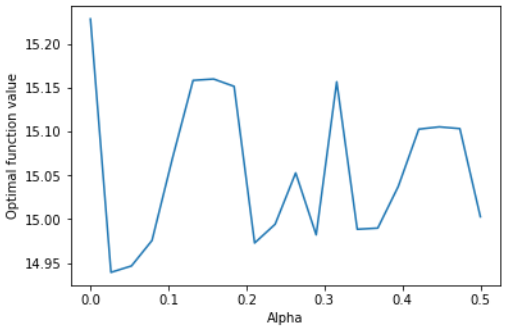


(b) Number of function evaluations per α.

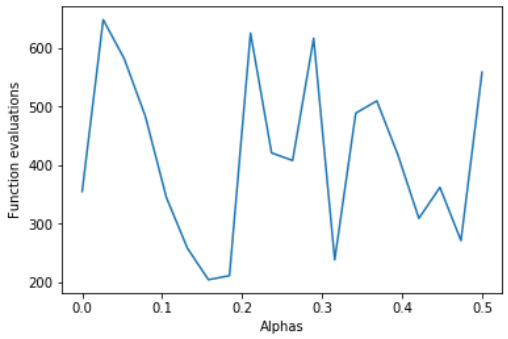


(c) Running time for the optimization per α.

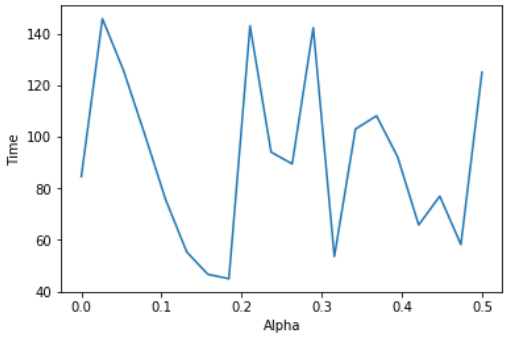
Figure C.3: Optimization with SLSQP on Almere input file for different smoothing parameters α in loss function (4.3)



(a) Environmental cost indicator of obtained solution per α.



(b) Number of function evaluations per α.



(c) Running time for the optimization per α.

Figure C.4: Optimization with SLSQP on All Situations input file for different smoothing parameters α in loss function (4.3)

C.2 Optimization

C.2.1 FFT+LIN

c	**Phase 1**	**Phase 2**	**ECI**(x^*)	$x*$
10	4.75	25.98	14.41	(17.52, 19.6, 103.48, 101.41, 28.45)
15	4.51	20.31	14.4255	(18.55, 19.63, 102.50, 101.14, 28.41)
20	5.21	66.73	14.4150	(17.42, 19.93, 102.88, 100.83, 28.38)
25	5.18	65.19	14.40832	(17.44, 19.63, 102.84, 101.16, 27.97)

Table C.1: FFT+LIN on Almere with $|L| = 50$, $\lambda = 30$, constructing the linear model takes around 10 seconds.

c	Phase 1	Phase 2	**ECI**(x*)	x*
10	5.43	23.89	14.41	(17.52, 19.64, 103.48, 101.41, 28.45)
15	5.04	21.43	14.42556	(18.55, 19.64, 102.51, 101.17, 28.43)
20	4.95	21.70	14.41441	(17.77, 19.59, 102.62, 100.79, 28.17)
25	5.45	21.54	14.40835	(17.44, 19.62, 102.85, 101.18, 27.99)

Table C.2: FFT+LIN on Almere with $|L| = 100$, $\lambda = 30$, constructing the linear model takes around 20 seconds.

c	Phase 1	Phase 2	**ECI**(x*)	x*
10	4.51	22.17	14.42642	(18.46, 19.64, 102.73, 101.35, 28.89)
15	4.79	19.37	14.4182	(17.38, 20.19, 102.56, 101.12, 28.47)
20	4.96	22.79	14.41033	(17.44, 19.63, 102.96, 100.99, 28.27)
25	4.70	20.40	14.42521	(18.53, 19.65, 102.45, 101.30, 28.40)

Table C.3: FFT+LIN on Almere with $|L| = 200$, $\lambda = 30$, constructing the linear model takes around 45 seconds.

c	Phase 1	Phase 2	**ECI**(x*)	x*
10	25.08	23.78	15.538	(38.37, 38.88, 76.86, 115.86, 36.44)
15	8.67	40.95	26.540	(32.60, 39.99, 75.03, 223.10, 80.02)
20	25.67	31.31	15.536	(37.33, 38.83, 78.34, 116.14, 36.61)
25	9.28	50.37	28.450	(39.40, 39.99, 76.11, 224.03, 85.56)

Table C.4: FFT+LIN on All Situations with $|L| = 50$, $\lambda = 30$.

c	Phase 1	Phase 2	**ECI**(x*)	x*
10	25.16	26.04	17.443	(38.82, 38.53, 76.34, 195.23, 48.84)
15	9.37	47.21	28.943	(36.85, 40.00, 75.23, 224.35, 87.17)
20	25.76	28.51	16.532	(37.63, 37.98, 77.61, 141.85, 43.02)
25	9.68	76.06	27.036	(38.02, 39.97, 76.51, 223.01, 81.3)

Table C.5: FFT+LIN on All Situations with $|L| = 100$, $\lambda = 30$.

c	Phase 1	Phase 2	**ECI**(x*)	x*
10	25.01	24.97	16.638	(37.47, 38.91, 76.62, 153.41, 43.74)
15	9.39	44.36	27.500	(39.21, 39.99, 76.09, 223.52, 82.69)
20	25.53	36.31	15.694	(37.18, 38.88, 78.39, 120.73, 37.67)
25	9.48	54.23	29.411	(37.38, 39.99, 76.13, 224.60, 88.60)

Table C.6: FFT+LIN on All Situations with $|L| = 200$, $\lambda = 30$.

C.2.2 FFT+NNLIN

In the tables C.7 is shown that the scaled parameter spaces do indeed improve the results.

	Almere				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	$l(x^*)$
100, weighted	96	10.01	92	25.17	14.407
100, original	79	6.47	86	22.13	14.411
200, weighted	74	8.15	81	22.68	14.446
200, original	76	7.40	99	28.10	14.450

	All Situations				
	Phase 1		Phase 2		
$\|L\|$	**Eval.**	**Time**	**Eval.**	**Time**	$l(x^*)$
100, weighted	92	4.91	97	20.83	16.173
100, original	125	6.06	77	17.75	15.978
200, weighted	123	6.62	87	20.54	15.670
200, original	127	8.01	84	21.25	15.951

Table C.7: Number of function evaluations and optimization time in phase 1 COBYLA and phase 2 COBYLA of the hybrid optimization method FFT+NNLIN in seconds and the function value of the result x^*, for test sets both from the weighted parameter spaces and the original.

Solutions

	Almere		All Situations	
$\|L\|$	x^*	**ECI**	x^*	**ECI**
100	(17.48, 19.81, 103.25, 102.90, 26.96)	14.40	(38.32, 32.92, 100.53, 97.51, 40.80)	16.17
200	(17.64, 19.93, 104.85, 103.80, 25.22)	14.40	(39.58, 35.22, 97.11, 99.83, 37.80)	15.67

Table C.8: Result x^* and its environmental cost indicator of the hybrid optimization with FFT+NNLIN surrogate model, COBYLA and COBYLA, for test sets both from the weighted parameter spaces and the original.

	Test Model 1		Test Model 2	
$\|L\|$	x^*	**ECI**	x^*	**ECI**
100	(15.71 , 15.31, 102.81, 102.95 , 26.39)	14.34	(8.86, 9.99 , 104.59, 104.76, 21.88)	14.14
200	(15.83, 14.28, 102.89 , 102.85, 26.61)	14.31	(9.97, 9.82, 104.31, 103.72, 21.99)	14.15

Table C.9: Result x^* and its environmental cost indicator of the hybrid optimization with FFT+NNLIN surrogate model, COBYLA and COBYLA, for test sets both from the weighted parameter spaces and the original.

	Almere		All Situations	
$\|L\|$	x^*	**ECI**	x^*	**ECI**
100	(18.06, 26.31, 104.22, 105.24, 12)	14.34	(38.16, 39.70, 99.23, 95.41, 29.99)	14.98
200	(18.30, 23.15, 106.37 , 115.06, 12)	14.34	(37.71, 39.89, 96.92, 98.24, 29.99)	14.98

Table C.10: Result x^* and its environmental cost indicator of the hybrid optimization with FFT+NNLIN surrogate model, COBYLA in phase 1 and SLSQP in phase 2, for test sets both from the weighted parameter spaces and the original.

	Test Model 1		Test Model 2	
$\|L\|$	x^*	**ECI**	x^*	**ECI**
100	(19.52, 17.29 , 139.15, 131.19, 12)	14.21	(8.96, 10.04, 106.01, 105.89, 17.08)	14.11
200	(17.87, 15.47, 104.49 , 104.42, 12)	14.27	(8.81, 10.00, 103.37, 104.65, 21.00)	144.13

Table C.11: Result x^* and its environmental cost indicator of the hybrid optimization with FFT+NNLIN surrogate model, COBYLA in phase 1 and SLSQP in phase 2, for test sets both from the weighted parameter spaces and the original.