CHALMERS

Optimization of the Dynamic Positioning problem under unchanging weather conditions

How to remain stationary in stormy seas

PETER LAURITZSON

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2015

Abstract

This thesis concerns dynamic positioning, which is important in the shipping industry. Dynamic positioning uses thrusters to stabilize marine vessels horizontally, in contrast to the more traditionally used anchors. This is important because better dynamic positioning solutions enable the use of vessels in rougher ocean conditions. We were provided with model data to review, develop, and implement optimization algorithms suited for the problem.

There are several different results, each important in its own right. Solutions found by different algorithms are contrasted to what is currently used, and it is shown that many standard optimization algorithms find sub par solutions. The discontinuous nature of the model data can be well approximated with a smooth function, using a Gaussian filter, on which the Augmented Lagrangian method can be used. The Augmented Lagrangian method is one of the algorithms with the highest convergence rates. Lastly, there is no guarantee that any solution found is close to a local optimum.

The thesis concludes with a recommendation of algorithms that improve on the current solution. The main recommendation is to use the Augmented Lagrangian method on the smooth approximation of the problem, starting from the solution found by currently used algorithms. Another recommendation is to further look into the branch and bound algorithm. The implementation of the branch and bound algorithm given in this thesis is intractable, but possible improvements are suggested.

Acknowledgements

I wish to thank to my examiner, Michael Patriksson. He always had good advice when I needed it. I also want to thank my supervisor at GVA, Johan Lennblad. He was able to steer me in good directions and kept me productive even when things looked dark. Lastly I thank GVA for proposing this thesis and giving me data to work with.

Peter Lauritzson, Göteborg 9/6/2015

Contents

1

Introduction

This thesis focuses on the optimization aspect of dynamic positioning, henceforth called DP. DP is essentially stabilizing the horizontal motions of a floating structure (see Figure 1.1 for an overview), which we call floater for short. The movements of the floater will be influenced by the weather. Wind, waves and currents will all exert force and moment[1] on the vessel. Instead of using anchors to stabilize the vessel, DP uses thrusters (basically propellers that can be rotated to any azimuth) to counteract the weather's influence. The aim of DP is to stabilize a marine vessel horizontally, either to keep it at a certain location or with respect to a moving object.

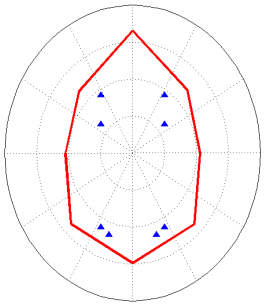


Figure 1.1: *A basic top-down view of a general floater with thrusters marked. The red line outlines the hull of the floater, and the blue triangles are the positions of the thrusters.*

This thesis discusses optimization of DP in unchanging weather conditions, which is referred to as the static DP optimization problem. Assuming that the forces acting on the vessel are static, we seek the best way to direct the thrusters and at what output. In reality weather and wind are always changing, or the vessel is changing heading, or something else imposes a change. The optimization done here can later be used as a foundation for solving DP in changing conditions. The conditions in the DP optimization problem should be such that the vessel is stationary, meaning no net moment (the floater should not spin) and no net side force (the floater is only moving against the wind, not perpendicular to the wind direction). The description given in [1] states, "A DP system

[1] The term used instead of torque in this field.

is to be designed to have a certain level of station keeping capability, reliability and redundancy.". This means that our optimization should be able to handle fail cases, which are cases where one or several of the thrusters fail to produce force. Redundancy is often measured by how much force can be produced against the wind (with no net moment and side force). This is because force against the wind is not used to negate moment or side forces. Hence, thrusters directed against the wind may be better used to counteract moment and side forces if necessary. One solution is thus more desirable than another if it produces more force opposite the wind direction and fulfills the conditions outlined above. That is to say, the objective function (the function our optimization problem maximizes) is the force against the wind.

The main reason that this optimization problem is complex and is of theoretical value, is the discontinuous nature of the functions in the problem. The function describing the maximum force generated for each degree azimuth will be called *eff* , which is sometimes referred to as the thruster efficiency. Each thruster has a maximum possible force output, but hull and stream-to-stream interactions will lower the force produced. The thruster efficiency of any degree azimuth yields the fraction of force that is produced, compared to the maximum possible force output of that thruster. Due to structural effects *eff* is discontinuous and is not guaranteed to be differentiable even where it is continuous. Figure 1.2 illustrates how the direction of the thrust impacts its efficiency. The existence of so called forbidden zones is clear in Figure 1.2, that is directions where the thruster is turned off to avoid compromising the structural integrity of the floater.

1.1 Background

Dynamic positioning involves using thrusters and propellers to automatically stabilize a floating structure in the horizontal plane, in contrast to the traditional use of anchors. Up until the 1960s [2] drilling rigs used were submersible and fixed. At that time it was discovered that semi-submersible rigs were more stable than submersibles, and better able to maneuver in confined waters. This, combined with moving drilling rigs over deeper parts of oceans, increased the interest in semi-submersible rigs that used DP. Jack-up rigs have a large depth of around 200 m, while anchoring vessels have become less economical to operate over increasingly deeper waters. In contrast, Cuss 1 (the first drilling ship to utilize DP, depicted in Figure 1.3) was stationed in waters that

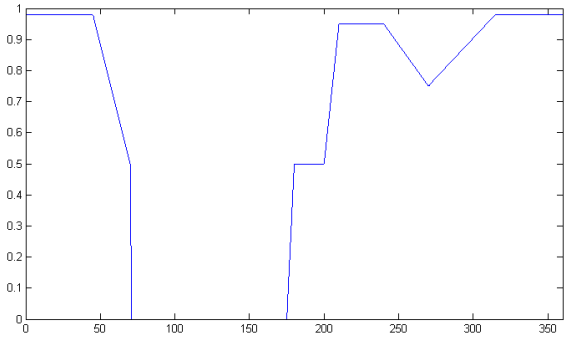


Figure 1.2: *Thruster efficiency for different directions. The y-axis is the fraction of maximum possible force that can be produced and the x-axis is azimuth degree. We can see the existence of a forbidden zone approximately between 75° and 175°.*

were over 3000 m deep.

DP increases the ocean area over which a floater can operate, which increases the potential earning of each vessel, but there are also other ways to improve the earning potential. The potential profit from a vessel can be improved by increasing the time it can stay operational. This can be done by improving the design as well as the placement of the thrusters on the vessel. Another way to improve the earning potential is to improve the steering of the thrusters. Better steering translates into less fuel used, or an increased ability to endure harsher ocean conditions.

A small improvement in steering can lead to a big increase in earning. A simple example illustrates this point nicely. Assume an oil platform needs to be operational 290 days a year to be break even. Increasing the average operational time from 300 to 303 days (a 1% increase) will increase the profits by 30%.

1.2 Short introduction to the problem

Figure 1.3: *The first floater to utilize DP was Cuss 1 in 1961, where it was able to be stationed over a depth of approximately 3500 m and drilled over 180 m beneath the sea floor (NSF photograph).*

The static DP optimization problem can be described as maximizing the pull of a vessel against the wind, while negating the moment and side forces that occur[2]. Figure 1.4 illustrates how moment and push created by wind vary with the direction of the wind. The dips at 0 and 180 degrees of the push and moment in Figure 1.4 is due to the vessel having bigger surface area when seen from the side (which can be gleaned from Figure 1.1)

The lack of a guarantee that *eff* is continuous with respect to the azimuth, combined with big[3] changes of the environmentally induced moment (with respect to wind direction) creates some issues. The discontinuous nature of the function means that it is hard to give any mathematical guarantees of optimality. It also means that using a solution to the static DP problem for one wind direction as a starting point when searching a nearby direction could be worse than starting randomly, even though it is intuitive to expect similar directions to have similar solutions.

[2]See Figure 1.4 for how the moment as well as the push changes with the direction of the wind, that is, for the same weather condition oriented differently.

[3]Several orders of magnitude

1.3 Report overview

The current method of solving the static DP problem is rule based. The aim of this thesis is to analyze the static DP problem mathematically, and hopefully supply a more theoretically robust optimization algorithm for the problem. Figure 1.4 shows how the forces acting on a vessel changes with wind directions, showing that the difficulty of solving this problem is most pronounced when the wind is perpendicular to the floating structure. Due to this we hope to see the most improvement compared to the current algorithm in wind directions perpendicular to the heading of the ship.

Chapter 2 contains information about the dynamical positioning problem and some important properties thereof. Chapters 3 and 4 delve into mathematical optimization and how it is applicable to the DP problem. The current rule based algorithm used for solving the static DP problem is described in Chapter 5. Other possible algorithms are reviewed in Chapter 6 wherein we see how some standard algorithms fare in the DP problem. Chapters 7 and 9 describe algorithms better suited for the problem, as well as specific algorithms implemented, respectively.

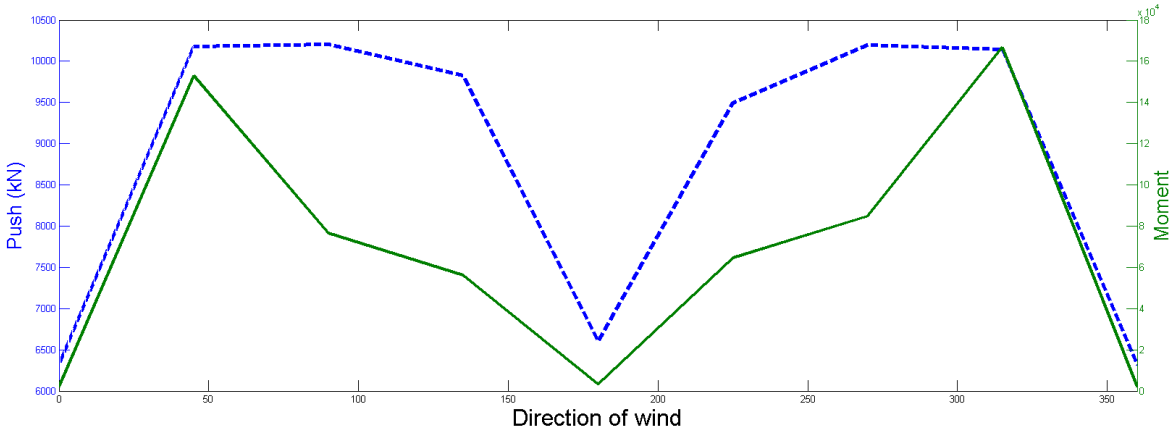


Figure 1.4: *The moment (green line) and force (blue dotted line) created by a specific weather condition, as it is rotated around the ship (the x-axis is the direction of the wind compared to the vessel heading, in degrees).*

1.4 Overview

The purpose of this thesis is to improve the computational aspect of DP. The objective of this thesis is to analyze the static DP problem mathematically, and using this analysis to implement one or two optimization algorithms. A review of a number of standard optimization algorithms and their performance on the problem will also be performed. We hope that this will help companies in the Marine sector make better informed decisions about what algorithms to use, even for floaters with different properties than the ones

considered in this thesis. Many types of marine vessels have DP, which means that the potential benefit of good results at this point in time is large.

There are some interesting questions that are outside the scope of the thesis. An expansion to the static DP problem is to simultaneously optimize the number of thrusters as well as their placements also. Another logical extension is to use knowledge about the static DP problem to solve a dynamic DP problem[4].

[4]The dynamic DP problem allows changing weather conditions, which trivially allows changing the heading of the ship.

2

Problem description

This chapter is gives a cursory introduction to the DP optimization problem, with focus on the purpose of DP optimization.

2.1 Real world description

Work done on a marine platform, or vessel, could be sensitive to horizontal movement[1], which means that it is important to keep the floater horizontally stationary. Weather (wind, waves and currents) exerts forces on a marine vessel—forces that need to be negated for the vessel to be stationary. One common way to counteract the weather's influence is to use anchors, another is to use thrusters designed for that purpose. The latter is called Dynamic Positioning (for short, in what follows: DP). An important question to answer when using thrusters is how they should be directed and what the optimal output is, in order to counteract the effects of the weather. Please note that even if the vessel is not moving horizontally there might be room for improvements; for example, lower total output translates to lower fuel consumption[2]. Often what is used for grading different solutions to the DP problem is not fuel consumption, but rather how much force can be produced against the wind direction. The reason that this metric is used is because it both positively correlates with the ability of using less fuel to counteract the weather forces as well as gives more leeway to handle unforeseen circumstances. A condensed practical description of the DP problem follows:

> We have a number of thrusters in given positions below the vessel. They may have different maximum thrust. The efficiency of each thruster depends

[1] An example would be an oil rig that should be stationary above the drilling hole

[2] An easy way to see this is that two thrusters yielding forces with opposite direction and same magnitude exert the same sum of forces as not using thrusters at all, the latter being a more fuel efficient way of maintaining that equilibrium.

on its azimuthal direction[3], with the orientation of the vessel being zero. At some angles the thruster might interact with the hull or other thrusters, which decreases the force produced considerably.

Our goal is to obtain a maximum force against the wind direction, with low (within a predefined tolerance) moment and low (within a predefined tolerance) side force. This is done by changing the thrusters' angle and power. The tolerances are specific to the vessel considered. The moment mainly produced by wind, and the side force (the force perpendicular to the wind direction) appears due to the thrusters counteracting the moment. In addition, the algorithm used to solve the example above should be able to handle fail cases, that is conditions where some of the thrusters are not used. This will add complexity since the thrusters are no longer symmetric on the ship.

While the description above regards optimization using only thruster angle and output[4] as variables, it might be of interest to expand the problem to include the placement of the thrusters on the vessel as variables also, instead of having them as parameters.

2.2 Weather conditions and model data

In our model the weather is described by three effects on the vessel: the magnitude and direction of the force by which the weather acts on the vessel, as well as the generated moment. Force magnitude and generated moment for each direction is provided by model testing. We studied 16 different weather conditions (These are described in Table 2.1), ranging from mild to hurricane force, which span the most common weather conditions. The effects of the weather conditions on the vessel is given for 8 directions uniformly spaced around the vessel, starting at 0 degrees (the heading of the vessel).

Weather condition:	1	2	3	4	5	6	7	8
Wind speed (m/s)	2.5	5	7.5	10	12.5	15	17.5	20
Weather condition:	9	10	11	12	13	14	15	16
Wind speed (m/s)	22.5	25	27.5	30	32.5	35	37.5	40

Table 2.1: Wind speed for the weather conditions used

We also have a model of the floating structure. The model of the floater is given by the following properties: thruster placement, depicted in Figure 1.1, maximum thruster force, and a function describing the efficiency of each thruster in different directions. This information suffices for a full mathematical description (detailed in Chapter 4) and is heavily influenced by the shape and structure of the floater.

[3]All directions in this thesis refers to the azimuthal direction, and will henceforth be known solely as direction.

[4]Power output, output, power, and force will be used synonymously in this thesis, meaning the power output of the thruster.

2.3 Measure of success

The vessel needs to remain stable in all weather conditions as well as in all wind directions. This implies that an algorithm which finds good solutions in headwind but cannot find a feasible point (i.e. a point such that all constraints are fulfilled) for side winds is worse than one which solves all wind directions but with low force against the wind. One solution is said to be better than another for the same weather conditions and direction if the magnitude of the constraint violation is less than the other, and if they are similar (or if there is no constraint violation) then objective function values are compared. The reason for comparing the magnitude of the constraint violations is that the algorithms should be robust[5].

An important image to understand is Figure 2.1, which represents a solution given by an optimization algorithm for a sample weather condition. It is instrumental to first study this image, since similar images are used to present results in this report. The idea behind this graphical representation of results is that it is easy to see whether there may be room for improvements, as well as unwanted behavior. The way the figure is structured implies that the closer the green line is to the red line, the less room for improvement exists, which means that the solutions depicted are close to optimal for head- and tailwinds. Also, if the green line dips below the purple line at any point (which almost happens at 90 and 270 degrees), the objective function is negative (i.e. the wind pushes the vessel backwards). The reason for the behavior of the wind force is the larger area exposed to the wind from the side compared to the front and back. This is also the cause for the dips of the green line at the same spots—we require a lot of force from the thrusters to equalize the moment created, which means less force that can be used to maximize the objective function.

[5] A difference of 1% in the constraint violation does not imply much difference in robustness, while a constraint violation one magnitude or more less implies that there is some leeway.

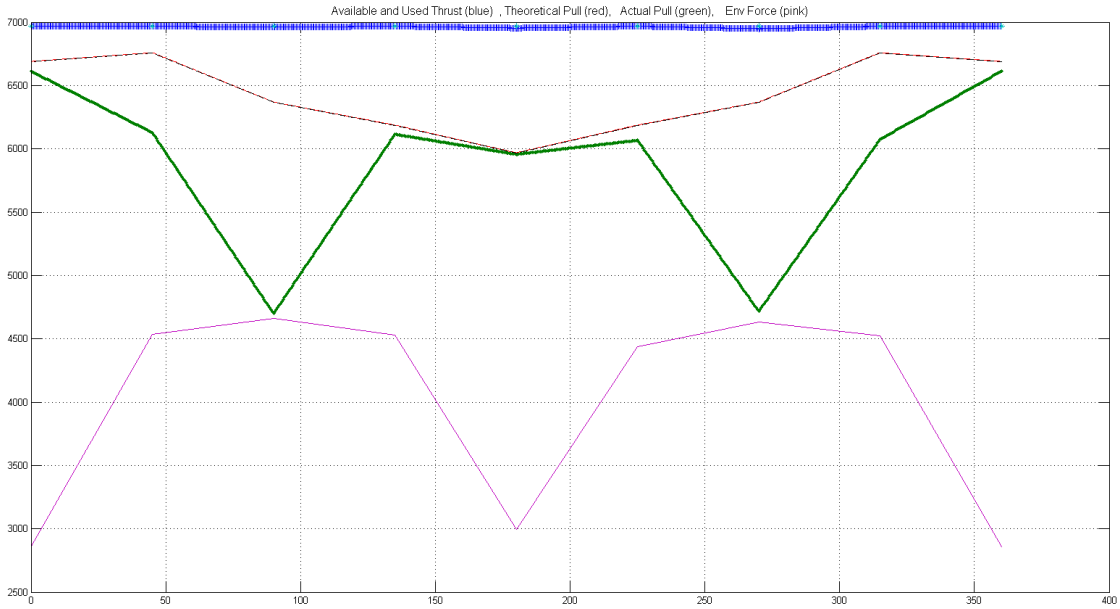


Figure 2.1: *This image depicts the performance of the solutions found to the static DP problem. The horizontal axis is the direction of the wind, compared to the heading of the vessel, the vertical axis is the force generated, and the purple (bottom) line is the force the wind exerts on the vessel. The green (middle) line is the force generated from the thrusters, and the red line is the relaxed maximum. The blue line is the maximum of the sum of magnitudes of thruster forces, that is, regardless of direction.*

3

Optimization background

The main purpose of this chapter is to provide a strong enough mathematical foundation to create a mathematical description of the static DP problem (described in Chapter 2). This chapter also provides a cursory glance at the theoretical concepts required to understand this thesis[1]. The mathematical models used will be explained more in depth in related chapters. The reader is assumed to have knowledge corresponding to a bachelor degree in mathematics.

3.1 What is mathematical optimization

Optimization is the mathematical discipline concerned with searching for the best value of a function. Usually the best value is taken to be the minimum value possible, but there are other metrics that can be used. Letting f be a function of variable x with domain $\mathcal{D}$, a formulation of an unconstrained optimization problem can be mathematically described by

$$\min_{x \in \mathcal{D}} f(x).$$

The function f to be optimized is called an objective function.
There may be constraints that have to be fulfilled for a solution to an optimization problem to be valid. Denoting equality constraints[2] by h and inequality constraints[3] by g, and assuming that the domains of the functions f,g and h are supersets of $\mathcal{D}$, the constrained optimization problem over the set $\mathcal{D}$ is described mathematically by

[1] If any concepts remain unclear the reader is recommended to research these on their own (see [3] for a good introduction to optimization).

[2] Constraints such that a function h has to be equal to zero.

[3] A function g such that it needs to be less than or equal to zero.

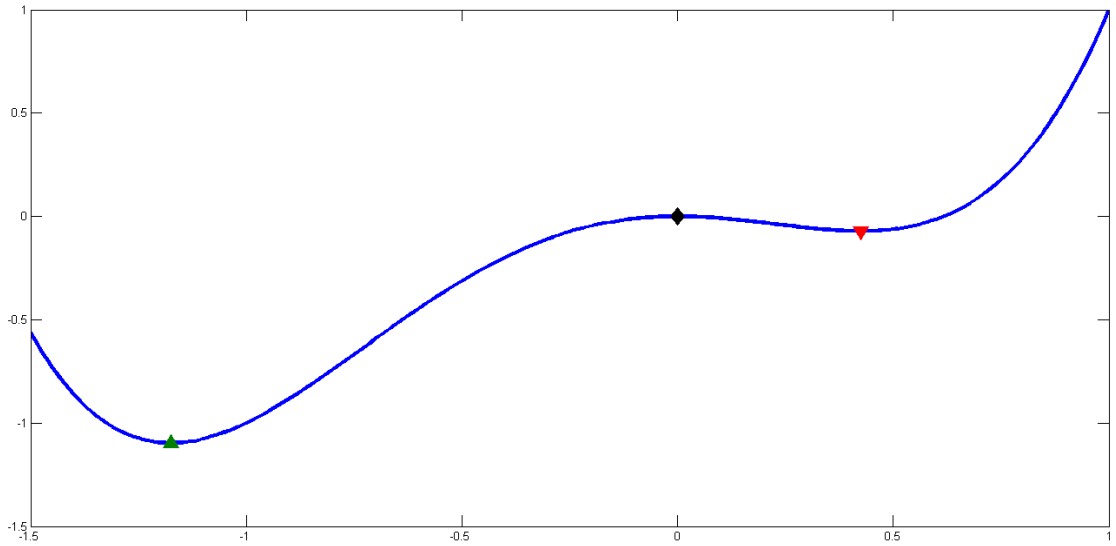


Figure 3.1: *A local minimum (the red downwards triangle), a local maximum (the black diamond) and the global minimum (the green upwards triangle) of the function $x^4+x^3-x^2$. The horizontal axis is the variable values, and the vertical axis the function values.*

$$\min_{x \in \mathcal{D}} f(x) | h(x) = 0 \wedge g(x) \leq 0.$$

Which is often stated as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h(x) = 0 \\ & g(x) \leq 0 \end{aligned}$$

Defining the set $\mathcal{S} := \mathcal{D} \cap \{x : h(x) = 0\} \cap \{x : g(x) \leq 0\}$, we can equivalently state the above problem as

$$\min_{x \in \mathcal{S}} f(x).$$

The set S is called the *feasible set*.

3.2 Local and global optimization

There may be many points in the set S that optimize the function in some way. Points that are the best that can be found in a neighborhood are called local optima. A point

$x^* \in \mathcal{S}$ is called a local minimum of a function f if

$$(\exists \delta > 0)(\forall x \in \mathcal{S} \cap (\|x - x^*\| < \delta))(f(x^*) \leq f(x)).$$

The idea of the definition is that there exists a neighborhood of the local minimum such that all function values in this neighborhood are greater than, or equal to, the objective value at the minimum. Points in S that yield the best value possible of f are called global optima[4]. A point $x^* \in \mathcal{S}$ is called a global minimum of f if

$$(\forall x \in \mathcal{S})(f(x^*) \leq f(x))$$

This implies that any global optimum is also a local optimum. Conversely, if the point x' is not a local optimum it can not be a global optimum.

3.3 How optimization is performed

In Section 3.2 it is shown that any global optima needs to be a local optima, and many optimization algorithms search for local optima, hoping that the found solution performs sufficiently well. Please observe, as is stated later in this paragraph, that trying to find a local optimum is not the same as guaranteeing that one is found. An exhaustive search is rarely computationally efficient, and often there is no easy way to find all the points which fulfill some necessary condition for global optimality. The idea behind many optimization algorithms is to move around in the function landscape trying to find the best point, and terminating when they cannot improve further. The terminating condition is often a necessary condition of local optimality[5]. Please note that a necessary condition is often not sufficient, which means that a solution found might not even be a local optimum. What is certain is that we can not reject the possibility that the solution is a local optimum. This is generally the best possible result, with local or global optimality guarantees only for specific class of problems.

[4]There may be more than one; take for instance the constant function $f(x) = 1$ where all points are global optima.

[5]A local optimum is a point such that a small step in any direction will always be worse than staying put, so this is not surprising.

4

Mathematical formulation of the DP problem

A mathematical formulation of the DP problem can be created by combining the description given in Chapter 2 with the mathematical foundation in Chapter 3. Starting with a description of the parameters and variables used in the mathematical formulation, this chapter then continues by defining and motivating the objective function. Please observe that a structural optimization is also possible, where the positions of the thrusters as well as the number of thrusters are variables instead of parameters. The structural optimization was not a focus of this thesis, but could be interesting for future work.
Note that we could also do a structural optimization, adding the positions of the thrusters (and the number of thrusters) to the variables instead of setting them as parameters.

Parameters

N,	the number of thrusters.
$\{x_i,y_i\}_{1\leq i\leq N}$,	the positions of the thrusters.
$\{e_{i,\phi}\}_{1\leq i\leq N,0\leq \phi<360}$,	the efficiency of thruster i at angle ϕ, in fraction of maximum thrust.
$\{t_i\}_{1\leq i\leq N}$,	the maximum thrust possible for thruster i.
w_d,	wind direction.
w_t,	moment created by the wind.
w_l,	wind load, i.e., the force produced by the wind.
tol_t,	moment tolerance.
tol_{sf},	side force tolerance.

Variables

$\boldsymbol{d_i}$, direction of thruster i, $0 \leq d_i < 360$, $1 \leq i \leq N$.

$\boldsymbol{f_i}$, force of thruster i, in fraction of maximum possible, $0 \leq f_i \leq 1$, $1 \leq i \leq N$.

Def:
Let τ_i be the demanded force from thruster i,

$$\tau_i := f_i \cdot t_i \cdot e_{i,d_i}$$

Henceforth, index i denotes an index in the index set $\{i : 1 \leq i \leq N\}$. The objective to maximize is the force against the wind, as measured by the sum of the forces provided by the thrusters, while neutralizing moment and side forces. The moment and side forces are said to be neutralized when their magnitudes are below a predetermined threshold, called tolerance. The objective function is thus defined as the sum of the force from each thruster in the wind direction.[1] The side force is described analogously. The moment is the force created perpendicular to the lever arm[2], which can be decomposed into x-axis and y-axis components.

Objective function

$$\text{maximize} \qquad -w_l + \sum_i \tau_i \cdot \cos(\tfrac{d_i - w_d}{180/\pi})$$

$$\begin{aligned} \text{subject to} \qquad |\sum_i \tau_i \sin(\tfrac{d_i - w_d}{180/\pi})| &\leq \text{tol}_{sf} \\ |-w_t + \sum_i \tau_i (cos(\tfrac{d_i}{180/\pi}) y_i - \sin(\tfrac{d_i}{180/\pi}) x_i)| &\leq \text{tol}_t \end{aligned}$$

Note that we may state the problem equivalently with a change of variables from a percentage of the maximum power (variable f_i) to an absolute power amount, by letting the new force variable be $f_i' := f_i \cdot t_i \cdot e_{i,d_i} =: \tau_i'$ (note that we can obviously discard either f_i' or τ_i', the main reason to define both equally is as a call back to our problem statement above), and having the constraints $0 \leq f_i' \leq t_i \cdot e_{i,d_i}$.

4.1 Problem properties

The mathematical formulation given above has many advantages over a descriptive formulation. The main advantage of a mathematical formulation is that it is easier to analyze the properties of the optimization problem. Finding these properties are advantageous for determining which optimization algorithm to employ (further discussed in

[1] Mathematically this is done as the sum over τ_i (the force from each thruster) times cosine the angle between the power output and the wind direction.

[2] Defined as the cross product between the displacement vector and the force

Chapter 7, but also in Chapters 6 and 9). Another important advantage of the mathematical formulation is the possibility of finding convergence guarantees on solutions.

4.1.1 Complexity of the problem

There are several sources leading to the high complexity of the DP optimization problem. The first is the dependence between the two constraints[3]. The second is the discontinuous efficiency function (depicted in Figure 4.1). Note the appearance of forbidden zones, which are areas where the thruster must be turned off since the jet would otherwise interact strongly with the hull[4]. The discontinuous nature of the efficiency function (henceforth known as *eff*) makes the optimization problem discontinuous[5], and discontinuous optimization problems are harder to solve than continuous ones. The discontinuities are a consequence of, among other things, the existence of forbidden zones. These forbidden zones raise the complexity of the problem since most algorithms will have trouble exploring the feasible set efficiently[6].

4.1.2 Properties of the problem

This section will present mathematical properties of the functions involved in the DP optimization problem.

Periodicity

A glance at the objective and constraint functions might lead to the conclusion that they are periodic (since they contain trigonometric functions), but the angle is required to be in [0,360) which means that only one period of the trigonometric functions is used. The trigonometric functions allow a trivial extension of the domain of the direction variables from [0,360) to $(-\infty,\infty)$. There is no analogous way to remove the bounds on the force variables.

Number of local optima

The number of local optima is an important characteristic of the problem. If the function has few local optima, a randomly selected local optimum has high probability of being the global optimum. Hence, an algorithm that converges to a local optimum might perform well regardless of starting point. Conversely, if the function has many local optima it is important to try to ensure that the algorithm does not get stuck in a bad performing local optimum. The number of local optima of the objective function is investigated by

[3]When any thruster is changed to counteract moment, it will impact side forces, and vice versa.

[4]There is a forbidden zone of around 100 degrees for thrusters 3 and 6, and a smaller one for thrusters 4 and 5, in the model data depicted.

[5]When *eff* is discontinuous it will make both the objective function and the constraint functions discontinuous.

[6]A thruster directed into its forbidden zone means that the thruster can not produce any force, most likely leading to a worse solution.

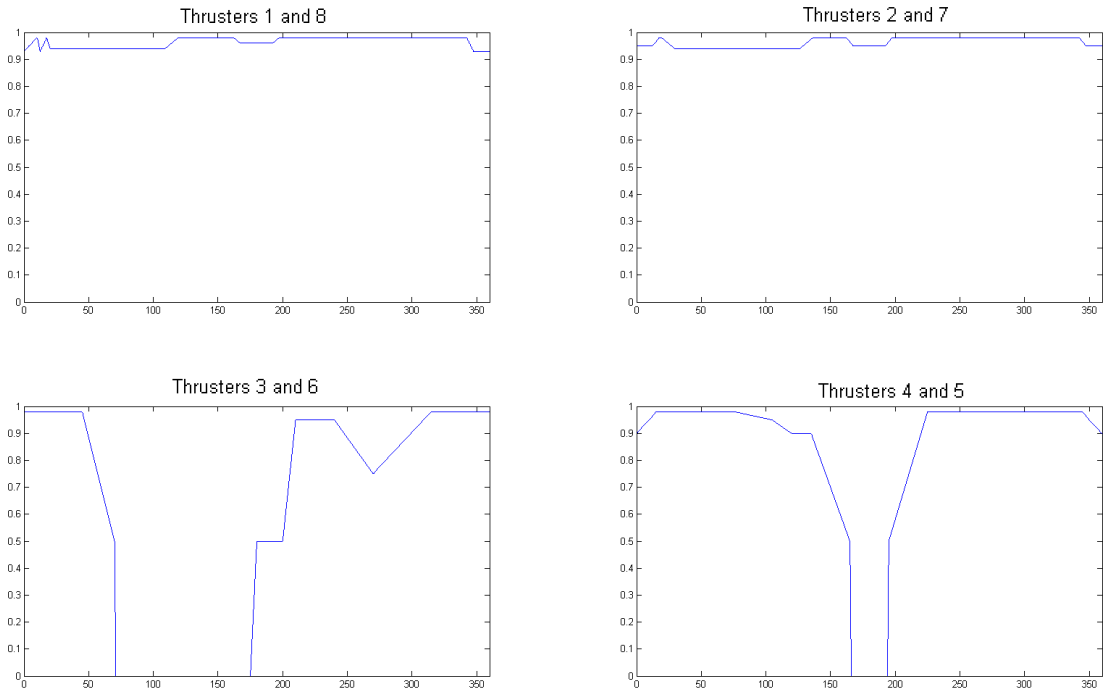


Figure 4.1: *The efficiency (depicted as the blue line), as a function of the degree azimuth, for the thrusters on a sample model. Note in particular the forbidden zones, where the efficiency is zero (the thrusters are turned off), in the two bottom images.*

taking a random weather condition (9 was selected) and a random direction (90 degrees was selected), and letting a coordinate descent (refer to Section 7.1.1 for a description of the algorithm) start from random starting points 9 times. Out of 21 generated starting points, 20 solutions found were unique (defined as $\|x_i^* - x_j^*\|_{\max} \leq 0.01$ of domain of variable). This result correlates with a high number of local optima of the objective function. Please note that there is no guarantee that coordinate descent reaches a local optimum.

Difference between optima

The number of local optima of a function does not always have a big impact on the optimization. There may be many local optima whose function values do not differ significantly from the global optimum. Hence, the question becomes what the probability is of reaching an optimum that performs similarly to the global optimum, starting randomly in the domain. This is an important question because if there is a procedure for finding local optima quickly, it may be iterated many times to find a good performing local optimum. The difference between different local optima is investigated in a similar

manner to the number of local optima above. Taking the solutions found in Section 4.1.2, the difference between the best function value and the third best (100 percentile vs. 90 percentile) is 13% in thrust generated. This difference in thrust generated is more than desired, and hence the idea presented above (execute a descent algorithm many times and return the best value) is not recommended.

Differentiability

Differentiability is an important aspect of the optimization problem. The number of times the functions can be differentiated impacts both the mathematical guarantees on solutions found, as well as the performance of different optimization algorithms. Generally speaking, increasing the number of derivatives that exists improves both the guarantees and the performances of algorithms. The objective function as well as the constraint functions are sums of smooth functions conjoined with *eff* . Hence both the objective function and the constraint functions are as many times differentiable as *eff* . Looking at Figure 4.1 we see that *eff* is discontinuous, particularly near the forbidden zones. In conclusion, the static DP problem is not continuous, and hence not once differentiable either. Please note that it is always possible to find a numerical gradient given a small step length, but there exists no limit when the step length tends towards zero. That is, for any given displacement δ the numerical gradient in that direction can be set as $\frac{f(x+\delta)-f(x)}{|\delta|}$, but this expression does not converge as $\delta \to 0$.

Convexity

The domain is convex, and both sine and cosine are convex for half their period, but *eff* is not. This implies that it is easy to conclude whether a problem, constrained to a sub domain in which *eff* is convex, is convex. There is no general procedure for finding the largest sub domains in which *eff* is convex, which means that splitting the static DP problem into convex and non-convex subproblems is not possible.

Other

Some other interesting properties are the computational costs of evaluating the objective and constraint functions. If the objective or constraint functions are expensive (either in time or resources required) to evaluate, it might be a reason to use response surfaces or surrogate functions (see for example [4] and [5] for more information about black box optimization). Analysis shows no substantial cost of evaluating the function, 10 000 calls took 8.73 seconds, and thus there is no immediate reason for utilizing black box optimization. The type of constraints impacts the problem type, and consequently what algorithms to use as well as their convergence theory. The main constraints are that the domain of the functions (i.e. not the feasible set) is bounded and closed, and the constraints are non-linear, discontinuous inequality constraints.

4.2 Guarantees for the solution to the static DP problem

The sections above show that many difficulties may arise when solving the DP optimization problem. The discontinuous nature of the objective and constraint functions means that no guarantees of convergence is possible for most algorithms, nor any guarantee that the solution found is a local optimum. Even so, it is nonetheless interesting to analyse how well algorithms that require differentiability perform when supplied with numerical derivatives. The existence of forbidden zones means that there is a high risk of being caught in a section of the domain containing no good feasible points. These difficulties make us cautious about giving any guarantees preceding the research in this thesis, i.e. we can not make any immediate mathematical guarantees on any solution found. We see no properties of the problem that allow a clear line of attack, and thus we feel that the probability of finding an algorithm outperforming the current rule based one is low.

5

The current algorithm

Two rule-based optimization algorithms were provided by GVA. These algorithms were built specifically for solving the static DP problem (described in Chapter 4). In essence the algorithms aim to solve the problem in the same way a human might, by first setting the thrusters directly against the wind, and then making small adjustments to try to fulfill the constraints. Computing the solution takes little time due to the rule-based nature of the algorithms[1].

5.1 Description of the algorithms

Both algorithms follow a similar list of rules when determining the solution. Given model and weather data, there are four distinct steps taken:

1. Find the relaxed maximum, i.e. the maximum of the objective function under no constraints.
2. Reduce the thruster with most impact on moment, until zero net moment is achieved (This is done for several thrusters iteratively, if needed).
3. Assume a new wind direction, and pull towards it, until the original side forces are equalized.
4. Use the reduced thrusters from step 2 to negate any left-over moment and side forces.

The only difference between the two algorithms lies in step 2. The main principle of one of the algorithms to achieve moment balance is to reduce the power output from one or a few thrusters, while the other changes the thruster direction instead. The latter has the complication that *eff* is dependent on direction.

[1]The algorithms both run 160 cases in about 4 seconds, including cache time

5.2 Issues and strengths

There are many good properties of these algorithms, but also one bad. The main strengths of the algorithms are the speed by which they provide and answer to the problem, that they explore the domain efficiently (i.e. initialize subroutines on all sides of the forbidden zones, which means that the forbidden zones matter less to these algorithms), that the results seem to be close to optimal for easy weather conditions, and that the combination of both of them manage to find feasible points under some tougher conditions. The main issue is that there is no real theoretical justification for the algorithms (which means that they may be provably suboptimal), which is due to the rule-based nature of the algorithms.

Similar to human based optimization

A human solving the static DP problem might follow a similar procedure to these algorithms. It is thus easy to follow what they do, and improve on the rules followed, but it also implies that they do not utilize information about the objective and constraint functions fully.

Rule-based

A rule-based algorithm is often created from a particular set of circumstances. Hence, a potential problem with a rule-based algorithm is that it may be too specific. That is, it may work well in situations similar to the one it was created for, but not in situations that differ in certain ways—and it can be hard to figure out what differences are important. As an example, both algorithms described in Section 5.1 solve the static DP problem for a specific vessel model in easy weather conditions very well, but break down in some of the harsher conditions. This is most likely due to the rules being set beforehand, with standard conditions in mind, which means that they do not react appropriately when the weather forces are larger.

Explores the domain

The algorithms converge quickly. The quick convergence allows them to explore many different starting points (which is also part of their implementation). The algorithms explore both sides of any forbidden zone, and are fast enough that trying tens of different starting points still yields a fast execution speed.

Lack of robustness

They yield good results quickly for easier conditions (but lack in harsher ones). It may be that the solutions found (even those close to the relaxed maximum) are not locally optimal.

6

Algorithms described in modern text books: any that work?

This chapter details the performance of a few algorithms, recommended as good candidates in modern literature, on the static DP problem. The static DP problem is formulated in Chapter 4. Where derivatives are required, a numerical approximation was supplied.

6.1 Possible algorithms

These methods are mainly taken from [3] and [6], with the particle swarm optimization found in [7]. Some convergence theory will be supplied. For further reading and theory see the previously mentioned books.

6.1.1 Penalty method with line search

The constrained DP problem is converted to an unconstrained problem using the penalty method (also known as exterior penalty method). The penalty method converts a problem of the form

$$\begin{aligned} &\min_x f(x) \\ \text{s.t. } &g(x) \leq 0 \end{aligned}$$

to the form

$$\min_x f(x) + \phi(g(x))_k$$

where ϕ is called the penalty function. The idea behind the penalty method is that by letting ϕ tend to infinity for positive values of g, when $k \to \infty$, the minimum of

the penalty problem will tend towards the constrained minimum. Convergence of the penalty method is seen (with usual notation) if f,g,$\phi \in C^1$, as well as $\phi'(x) \geq 0 \forall x \geq 0$. Then if the point x_k^* is stationary for the problem containing ϕ_k, and $x_k^* \to x^*$ as $k \to \infty$, as well as x^* being feasible and linear independence constraint qualifications hold at x^*, x^* is stationary for the constrained problem. In our problem f,$g \notin C^1$, so this result does not apply, but even without proven convergence, the idea is interesting. There is still a need to solve every subproblem (that is, solve the penalty problem for specific k), which is done here with line search algorithms. The different line searches implemented are coordinate descent, gradient descent, conjugate gradient descent, quasi-Newton (BFGS) and Newtons method.

Overview of the line search algorithms

A more thorough discussion of deterministic line search algorithms is presented in section 7.1, this section presents the basic idea behind a few of the line search algorithms mentioned in section 6.1.1, as well as some convergence theory.
Gradient descent is a line search method, based on the idea that following the steepest descent direction might improve the function value most. The steepest descent direction is the negative gradient, hence the name gradient descent. After finding the steepest descent direction, a line search is executed to find the approximate minimum of the function along that direction. The next iterate is then started from that minimum. This continues until the gradient is zero (or less than a predetermined threshold). The idea behind gradient descent seems like it should reach a stationary point, which is not true. A common convergence result is as follows: If we have that f is once differentiable, ∇f Lipschitz continuous, $c_1\|\nabla f(x_k)\| \leq -\nabla f(x_k)^T \rho_k, c_1 > 0$, $\|\rho_k\| \leq c_2\|\nabla f(x_k)\|, c_2 > 0$ and the step length α_k satisfies $\alpha_k \to 0$ and $\lim_{k\to\infty} \sum_{i=1}^{k} \alpha_i = \infty$, then for gradient descent we have that either $f(x_k) \to \infty$ or $\nabla f(x_k) \to 0$.
The quasi-Newton method approximates a Hessian for the function, and solves a quadratic approximation to find the line search direction. The advantage of the quasi-Newton over pure gradient descent is that the approximated Hessian will allow the algorithm to use information about the curvature of the function. Also, since the Hessian is approximated, the function does not need to be in C^2. There is a similar convergence result for the quasi-Newton method to the gradient descent method. Suppose that $f \in (C)^3$, and assume that $\{x_k\}$ converges to a point x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then if and only if

$$\lim_{k\to\infty} \frac{\|(B_k - \nabla^2 f(x^*)\rho_k)\|}{\|\rho_k\|} = 0$$

holds, $\{x_k\}$ converges super linearly.
Newtons method uses the true Hessian for the quadratic approximation, instead of an approximate Hessian as is the case in the quasi-Newton methods. Using the true Hessian implies that the quadratic model is a better fit to the original function, but this also presupposes that the true Hessian is known. The convergence theory only differs slightly from the quasi-Newton above. For Newtons method, suppose instead that $f \in (C)^2$ and

that $\nabla^2 f$ is Lipschitz continuous in a neighborhood of a solution x^*. Then if the starting point is sufficiently close to x^*, the iterates converge quadratically to x^*, as well that $\{\|\nabla f_k\|\}$ converges quadratically to zero.

6.1.2 Algorithms for constrained optimization

There are other ways to handle constrained problems than using the penalty method. An interior point method (also known as barrier or interior penalty method) using BFGS approximation of the Hessian was executed on the problem. The interior point method assumes that the initial point is feasible, and sets up a barrier function so that the minimization never crossed to infeasible points. This barrier function is smooth, and grows without bound as the constraints get closer to being unfulfilled. The logarithmic function is commonly used as a barrier function. The interior point method has convergence theory quite similar to the penalty method. Assuming same properties as for the penalty method, we have that if x_k is stationary, $x_k \to x^*$ and linear independence constraint qualifications hold at x^*, x^* is stationary for the original problem. Other possibilities include the active set method (disregarding inactive constraints to optimize more efficiently), sequential quadratic programming (approximating the function by quadratic functions to find the next iterate) and trust-region reflective (using a simpler function as a model of the true function, which is then solved to find the next iterate).

6.1.3 Particle swarm optimization

Particle swarm optimization (further known as PSO) is an interesting optimization algorithm. A priori it is a strong candidate, since it requires no continuity, and has specific steps against getting caught in a local optimum (the inertia of each particle). The generic PSO algorithm works on unconstrained problems, so the trials run in the thesis are on two different penalty functions. The first is a simple quadratic penalty function, and the second utilizes the interior point idea—any infeasible point will be graded as worse than any feasible point, two feasible points will be graded on the objective function value of those points, and two infeasible points will be graded on the size of the constraint violation. Please read [8] for more in depth information about constraint handling in PSO.

6.1.4 Iterated hill climbing

The idea here is to utilize a quick local optimization algorithm (coordinate descent) with random starting points a large number of times, and then returning the best found. The same caveats apply as in section 6.1.1, since this is still a coordinate descent algorithm. The trials run here tested 1000 different starting points, taken with uniform probability over the domain.

6.2 Results

The results (seen in appendix B) clearly show that most of the algorithms tested in this chapter perform worse than the currently implemented rule based algorithm. Algorithms that fail to find a feasible point when the environmental conditions are mild (wind speeds below 10 m/s) can be discarded. The most probable reason that the algorithms fail is that they can not deal with the discontinuities, not improving where there may exist descent directions.
Some interesting results to be noted are as follows. Firstly, the bad performance of the PSO algorithm. A priori it seemed like a good candidate (does not require continuity as well as avoidance staying in local optimum). We believe that the bad performance is mainly due to badly selected parameters. If bad parameter values is the reason, it highlights a problem with PSO—it requires extensive parameter tweaking. One set of parameters that perform well for some enviromental conditions is not guaranteed to work for others. Also note that there are no guarantees on the solution. The second interesting result is that there are some algorithms that perform OK on the static DP problem, main the active set method and the penalty method with line searches (for the warm start scenarios). It seems that these algorithms handle a numerical approximation to the gradient better than others, being able to find improvements from the supplied starting point. Neither the active set nor the penalty method performs well with a cold start, and as such they seem to explore the domain only locally.

6.3 Conclusions

The current algorithm outperforms all tested algorithms when using a cold start. The tested algorithms neither have proof of convergence with on the static DP problem (there is, for example, no Lipschitz continuity), nor perform better than the current algorithm. This clearly shows that other ideas are needed.

7

Reviewed optimization algorithms

This chapter details some possible ideas for algorithms, while noting that the standard algorithms tested in Chapter 6 all performed worse than the current rule-based algorithm. Chapter 9 gives a more in detail look at the algorithms selected for implementation. That is, the methods in this chapter that seem most promising.

7.1 Deterministic, line search

A line search method for optimizing a function consists of two steps. First, a direction is selected. Second, an optimum is found along this direction. There are several ways to select a search direction, four of which are described below.

Wolfe and Goldstein conditions

Finding the exact solution along the search direction is more computationally expensive than finding an approximate solution, with no real advantage. The Wolfe as well as the Goldstein conditions[1] are conditions that try to determine when an approximate solution is 'good enough' to terminate our line search. There are other ideas beside finding the exact solution and using either set of conditions described above. One of these ideas is deciding a longest allowed step length, then making a grid between the start and end points, lastly selecting the best function values in the grid as the approximate solution.

7.1.1 Derivative free descent method

Coordinate descent is, as the name implies, testing the coordinate directions to see whether any might be a descent direction. A basic implementation would be to select a coordinate direction (along both the positive and negative axis), and see if a small step

[1] See [6] for detailed description

along this direction yields a decrease in the function value. If a decrease was found, a line search is commenced along this direction. The solution to the line search is taken as the next iterate, and the steps are repeated. If the direction selected is not a descent direction, select another coordinate direction and repeat the steps above. Terminate when no coordinate directions yield improvement. The main advantage of this method is that it does not require continuity. Coordinate descent will also converge to the global optimum for convex, smooth function. If the function is not smooth (as with the static DP problem), no such guarantee exists.

7.1.2 [Conjugate] gradient descent

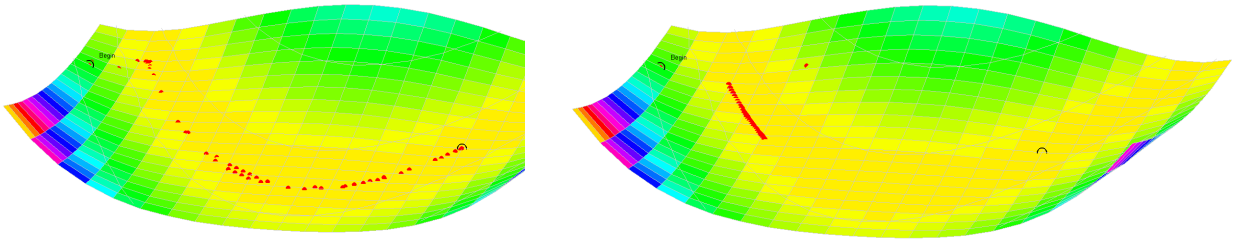
Gradient descent as well as conjugate gradient descent utilizes local information about the function around a point. Gradient descent finds the quickest descending direction in that point, by linearizing the function at that point and selecting the best direction. This is possible because $f \in \mathcal{C}^k$, $f(x+\Delta x)-(f(x)+\Delta x^T \nabla f(x)) \to 0$ as Δx gets smaller. That is, in a small enough neighborhood around a point the gradient is the best predictor of nearby function values. The search direction chosen by gradient descent is $-\nabla f$ when minimizing, which explains the name. A variant called conjugate gradient descent uses information about the previous iterates to better select the next search direction. The difference between conjugate and basic gradient descent is that conjugate gradient descent requires the descent direction in iterate k to be conjugate[2] to the previous descent direction. Both of these algorithms require $f \in \mathcal{C}^1$, and for continuous functions they generally outperform coordinate descent. Conjugate gradient descent ensures that the descent directions selected do not repeat the mistakes of previous iterations, which leads to better convergence at a higher computational cost. When these algorithm terminate the gradient is less than some predetermined tolerance, and hence the solution found will approximately be a stationary point.

7.1.3 [Quasi-]Newton method

Newtons method utilizes not only the slope of the function at the current point, but also the curvature. Newtons method will select a descent direction using information about how the line selected will behave a bit further out (i.e. the curvature). Contrast this to the gradient descent method, which takes the steepest descending direction at the point as the search direction, with no regard to the curvature. This usage of curvature is done by implementing a search direction dependent on the Hessian of the function at the given point.
The main difference between the Newton and the quasi-Newton method is that the quasi-Newton method does not require an explicit Hessian, rather approximating the Hessian by other means. Because of this approximating procedure, the quasi-Newton method does not require the function to be twice differentiable. Other than using an approximation of the Hessian, the idea behind both methods is the same. Most standard

[2]Two vectors u and v are conjugate with respect to A if $< u,v >_A = 0$

approximations of the Hessian yield similar convergence properties for the quasi-Newton method as for the Newton method. Both methods have quadratic convergence given start in a neighborhood around a local optimum where the Hessian is both non-singular and Lipschitz continuous. This quadratic convergence is possible due to the extra properties required of the problem for these methods, in contrast with gradient descent or coordinate descent.



(a) Rosenbrocks banana function solved with quasi-Newton, BFGS approximation. The circle to the right is the global minimum, and the leftward circle is the starting point.

(b) Rosenbrocks banana function solved with gradient descent, terminated after 250 function calls.

Figure 7.1: Difference in convergence rate between gradient descent and quasi-Newton (BFGS) showcased, using the Rosenbrock banana function. It is easy to see that the usage of curvature information in the Newton methods allow better convergence.

7.2 Deterministic, model based

Model based optimization algorithms work by utilizing a simpler model function, either to find the next iterate or to execute a global optimization on. The model based algorithms can be split into two camps, the black box algorithms which aim to approximate the true function over its entire domain, and the methods which use a local model to find the next iterate.

7.2.1 Trust region

A trust region algorithm is an example of an algorithm which uses a local model to find the next iterate. This usage of a local model is in a sense opposite to a line search. Instead of deciding a search direction and then finding the step length, trust region decides a biggest step length and then finds where to go. More specifically, to the problem $\min_{x \in S} f(x)$, at iteration k with current best solution x_k, the next iterate, x_{k+1}, is found as follows:

- Create a model m of f
- Set a largest step Δ

- Find x^* s.t. $x^* = \underset{\|x-x_k\| \leq \Delta}{\operatorname{argmin}} \; m(x)$
- If $f(x_k) - f(x^*)$ is large enough, increase Δ and repeat. This difference is often compared to the predicted difference (i.e. $m(x_k) - m(x^*)$), and a large value means that the model approximates the true function well, and hence it can be possible to search further away.
- Else if $f(x_k) - f(x^*)$ is too small, decrease Δ and repeat. This means that the model function is a bad approximation, and shorter steps are needed to make progress.
- Else set $x_{k+1} = x^*$

The idea is to create a simple model of how the function f behaves near our current iterate, then optimizing this model to find the next iterate. The advantage of the trust region method is that the model is often easier to solve than the real function.

7.2.2 Surrogate function / Simulation based methods

Simulation based methods aim to create a model of the function over its entire domain, which is then optimized. The model function is called a surrogate function. Surrogate functions are often used if the problem is expensive to evaluate (either high in cost or time), or if there is no explicit model of the function—so called black box optimization. The main advantage of simulation based methods is that they can be low cost, regardless of the cost of the real function, as well as the surrogate function can be chosen to be more easily optimized. The main disadvantage is the need to find a surrogate function that describes the problem well.

7.3 Stochastic

Stochastic optimization algorithms refer to methods that are stochastic, not optimizing stochastic functions. The main idea behind stochastic optimization algorithms is to use randomness to influence the exploration of a function, and thus better search for an optimum.

7.3.1 Particle swarm optimization

Particle swarm optimization (henceforth abbreviated PSO) is a stochastic optimization algorithm inspired by the movement of herds in nature. PSO emulates a flock of birds exploring a landscape, where each bird is affected by the others. More specifically, each iteration of a PSO method contains several (hence the term 'swarm') function values at different points (called particles, corresponding to individuals in the herd), that each is trying to maximize its fitness. Due to this, PSO requires only function evaluations, not continuity or differentiability.

The idea behind PSO is that each particle is influenced by its best previous value, as well as the best previous value for the swarm. These previous values will influence the path taken by each particle. The particles also have velocities associated with them, just like a flock of birds can have faster and slower moving members. This velocity will change randomly in proportion to the distance to the previous best values. Lastly, as with herds of animals in nature, the particles have inertia, helping them avoid getting stuck in local optima. Due to the fact that PSO has no requirements on the function, it is versatile and can tackle many different problems. The stochasticity of the method, combined with the lack of extra conditions on the function, mean that there is no guarantee on the found solution.

7.3.2 Simulated annealing

Simulated annealing is in some ways the other end of the stochastic optimization algorithm spectrum, compared to PSO. Whereas PSO uses a swarm to explore the function landscape, simulated annealing uses one estimate. The idea is to have the exploration and exploitation behavior dependent on the time. In the beginning, simulated annealing will explore the function landscape with a big step size, and the further along the algorithm has come, the more it will search near the best values already found (exploit the information it has). The same advantages and disadvantages as with PSO apply here.

7.4 Branch and bound

There is in particular one algorithm that does not fall into any of the previous categories, and that is the Branch and bound algorithm[3]. Branch and bound is a deterministic method that is neither a line search method nor model based. Branch and bound (further referred to as BnB) in its most basic form is a deterministic algorithm for unconstrained optimization, that requires no conditions on the objective function, and finds a solution that is approximately globally optimal[4].
BnB requires a way to find an upper and a lower bound for the objective function on a set. The upper bound function is often denoted U, and the lower bound function L. These functions need to have the following characteristics, with x_d^* denoting the $\text{argmin}_x f(x)_{x \in d}$, $U : U(d) \geq f(x_d^*) : f(x_d^*) \leq f(x) \forall x \in d$ and $L : L(d) \leq f(x), \forall x \in d$. An additional requirement is that these bounding function should converge to the same value as the set shrinks, $|U(d) - L(d)| \to 0$ as $\|d\| \to 0$. In practice, strict convergence is not needed, rather $|U(d) - L(d)| \to \epsilon$ as $\|d\| \to 0$ is sufficient to find an ϵ-approximate global solution. If the function is Lipschitz continuous, an upper bound can always be found as the function value of any point in the set d. The main idea of the BnB method is to start by finding the upper and lower bound of the entire domain. Then every iteration, partitioning the domain further and computing the upper and lower bounds

[3] A well known implementation of a partitioning algorithm similar to the branch and bound algorithm is the DIRECT (DIvide RECTangle) algorithm.

[4] An ϵ-approximate global minimum is here defined as a point x^* s.t. $f(x^*) \leq f(x) + \epsilon, \forall x \in D$

of each new subset. This continuous until the difference between the best lower bound and the best upper bound is within a predetermined tolerance ϵ. This guarantees an ϵ-approximate solution. Explicitly this is done as follows:

- Start by setting $\mathcal{D} = \{D\}$, $U_{global} = \infty$, $L_{global} = -\infty$, and $\epsilon > 0$.
  1. Select a member d of $\mathcal{D}$ to partition into subsets d_i
  2. Update $\mathcal{D} \leftarrow \mathcal{D} \cup d_i - d$
  3. Update $U_{global} = \min_{d \in \mathcal{D}} U(d)$ and $L_{global} = \min_{d \in \mathcal{D}} L(d)$
  4. Remove $d \in \mathcal{D} : L(d) > U_{global}$ from $\mathcal{D}$
  5. Repeat steps 1-4 until $U_{global} - L_{global} < \epsilon$

The steps above is a generic description of the algorithm. Firstly, the lower and upper bound functions impact the performance of the algorithm. A common implementation is to have the lower bound as an relaxed solution. The choice of a solving a relaxed problem can make it easier to find a solution, which improves the time complexity of the BnB method. There is no generic way to find the upper bound though. In addition to different possibilities of the lower and upper bound functions, there are several options on implementation of some of the steps above. There are several ways to select which leaf to branch (step 1 above). One standard method is to select the leaf with smallest lower bound, with the idea to push the lower bound upwards (in other words, selecting the smallest lower bound leaves best room for improvement). The partitioning of the selected leaf can also be done in different ways, and the choice impacts the computational complexity of the algorithm. Two standard implementations for the partitioning of the selected leaf are to split the longest dimension in half, and to partition the leaf into equally sized balls.

(a) A schematic view over the domain after each branch

(b) The union of the branches on each level equals their parents, $D_1 \cup D_2 = D$, $D_{21} \cup D_{22} = D_2$.

Figure 7.2: A sample branch and bound graph

There are some serious advantages as well as disadvantages to this method. A big advantage is that the found solution is within ϵ from the global optimum. The main issues

are that the BnB method can be computationally intensive, depending on how many leaves needs examining, trivially becoming exponential worst case time complexity[5]. The other main issue is the problem of finding good lower and upper bounding functions, that are quick to evaluate and still converge.

7.5 Unconstrained model of constrained problem

Many optimization problems are constrained optimization problems. The optimization algorithms presented above, in Sections 7.1, 7.2, 7.3, 7.4, solve unconstrained optimization problems. The methods presented in this section are ways of converting an constrained optimization problem to an unconstrained one.

7.5.1 Penalty method

A penalty method puts a penalty on constraint violations, with the idea that when the penalty is made larger, the solutions to the penalty function will converge to a solution to the constrained problem. See Section 6.1.1 for more detail. Suppose the problem is

$$\begin{aligned} &\min_{x \in D} f(x) \\ \text{s.t. } &g(x) \leq 0 \end{aligned}$$

The quadratic penalty problem is then defined as

$$\min_{x \in D} f(x) + \mu \cdot \max(0, g(x))^2$$

The solutions to the quadratic penalty problem (with $\mu \in \mathbb{R}_+$) will, under suitable conditions, tend towards the constrained optimum when $\mu \to \infty$. The convergence toward the constrained optimum means that the constrained problem may be solved with algorithms suited for unconstrained optimization, using the penalty method approach. There are some problem specific parameters that can impact the solution. The penalty parameter μ needs to increase fast enough (to reduce the number of subproblems to solve), but not too fast (where the warm start of the next subproblem is too far removed from the new local optimum). There is also no guarantee of finding a good approximate solution unless the penalty parameters gets arbitrarily big.

7.5.2 Augmented Lagrangian

The augmented Lagrangian method is a mix between the penalty method and the Lagrangian of the problem. Note that the Lagrangian $\mathcal{L}$ for an optimization problem

[5]The exponential worst case time complexity is trivial, since all leaves smaller than or equal in size to δ might need investigating

$\min_{x \in D} f$ with equality[6] constraints $g = 0$ is

$$\mathcal{L}(x,\lambda) = f(x) - \lambda \cdot g(x)$$

Where the global minimum of the original constrained optimization problem is a stationary point of the lagrangian.
The augmented Lagrangian, denoted as Φ below, is a method that utilizes a hybrid between a penalty function and a lagrangian:

$$\Phi(x,\lambda,\mu) = f(x) - \lambda \cdot g(x) + \frac{\mu}{2} \cdot g(x)^2$$

Where each iteration updates the multipliers λ_k, converging to the true λ. This means that the penalty parameter μ can stay small. A big advantage is that it often avoids ill-conditioning, as well as a good solution might be found in fewer iterations than with the penalty method.

7.5.3 Exact penalty and exact augmented Lagrangian methods

A short note on the existence of exact penalty and exact augmented Lagrangian methods. The solutions to the penalty method and the Lagrangian methods above will tend towards a solution to the constrained problem, as the penalty parameters increase. An exact penalty (or exact augmented Lagrangian) method is such that if $\mu > M$, for some constant M, only one optimization is required to find the constrained optimum. Please see [9] for further information.

[6] Note that inequality constraints can be converted to equality constraints by the introduction of slack variables.

8

Implemented algorithms

As seen in Chapter 6, many common optimization algorithms fail with this particular problem. The most interesting methods explained in Chapter 7 were implemented instead, as the most likely candidates to succeed on the static DP problem. The implementation of these candidates are explained in this chapter.

8.1 Branch and bound

Branch and bound is one of the implemented algorithms. This implementation uses a lower bound function that solves the relaxed problem, i.e. the problem without constraints, and the upper bound function is the value of any feasible point in the set. If no feasible point is found, the leaf is discarded. Please see Chapter 4 for the mathematical formulation of the function and constraints of the optimization problem. We select the leaf to branch as the leaf with currently smallest lower bound (note that there are other ways of selecting the leaf). Each leaf can be seen as a hyper rectangle (in this instance) in $\mathbb{R}^n$, i.e., denoting the leaf by l, $l = I_1 \times \cdots \times I_n$, where I_k is an interval. The selected leaf is then branched along the middle of the largest dimension.

8.2 Simulation based algorithm

Another algorithm that was implemented is one that exchanges the objective and constraint functions for surrogate functions. The reason behind implementing surrogate functions is that the surrogate function can be chosen to be smooth, and not that the evaluations become less expensive (the function is already cheap to evaluate). This smoothening avoids issues with the discontinuous *eff* function, and hopefully avoids the problems plaguing the algorithms in Chapter 6. The original functions are not costly to evaluate, hence smoothening the problem without replacing the functions completely is a viable option—and the one we took. We only need to apply a filter on *eff* to make the

objective and constraint functions smooth.
The transformation chosen is *eff* correlated with a Gaussian (more specifically, normal distribution), which, due to symmetry, is equivalent to setting the new *eff* function *eff* *:

$$eff^* := eff * \mathcal{N}(\mu,\sigma^2)$$

where we let $\mu = 0$. This has two big consequences:

1. *eff* * is smooth, since $(f * g)' = (f * g')$
2. *eff* $^*(x) \to$ const., $\forall x, \sigma \to \infty$, and *eff* $^* \to$ *eff* , $\sigma \to 0$

That is, using *eff* * in our objective and constraint functions means that they are smooth. Hence we may use algorithms that require gradients, second derivatives and more to solve this subproblem. Secondly, solving the subproblem for a small enough σ should let us get close to a solution to the original problem. Lastly, note that starting with a large σ means that we will have 'smudged out' small variances and follow the main features of the function. Thus the idea is to implement this algorithm, starting with a large σ, solve the subproblem, decrease σ and use the previous solution as a warm start. Repeat until σ is small enough. Our hope is that this will tackle both the discontinuous nature of the static DP problem, as well as the forbidden zones.

8.2.1 Penalty method with line search

Two different ways of solving the subproblem for each σ were implemented. The first of which is a penalty method with line search, where the next step is found by selecting the step which minimizes the penalty function the most of the following line search methods; coordinate descent, gradient descent, conjugate gradient descent, quasi-newton and newton. Wolfe conditions (described in Section 7.1) were implemented to know when to terminate the line search.

8.2.2 Augmented Lagrangian with trust region optimization

The second implemented method for solving the subproblem is an augmented Lagrangian method, using a trust region approach. The trust region (see Section 7.2.1 for more details) is taken as a quadratic approximation to the goal function, ie $m(x) = f(x_0) + \nabla f(x_0)^T x + x^T H(x_0) x$.

8.3 Global search (specific to DP)

The idea here is that smoothening *eff* might not be sufficient to help the algorithms traverse the forbidden zones, it might be a good idea to implement something that specifically counters them. Specifically, the main problem is that the efficiency table has big dips that make any solution moving into these dips worse than previous iterates. This means that it is likely that an algorithm will stop at the edge of a big dip instead

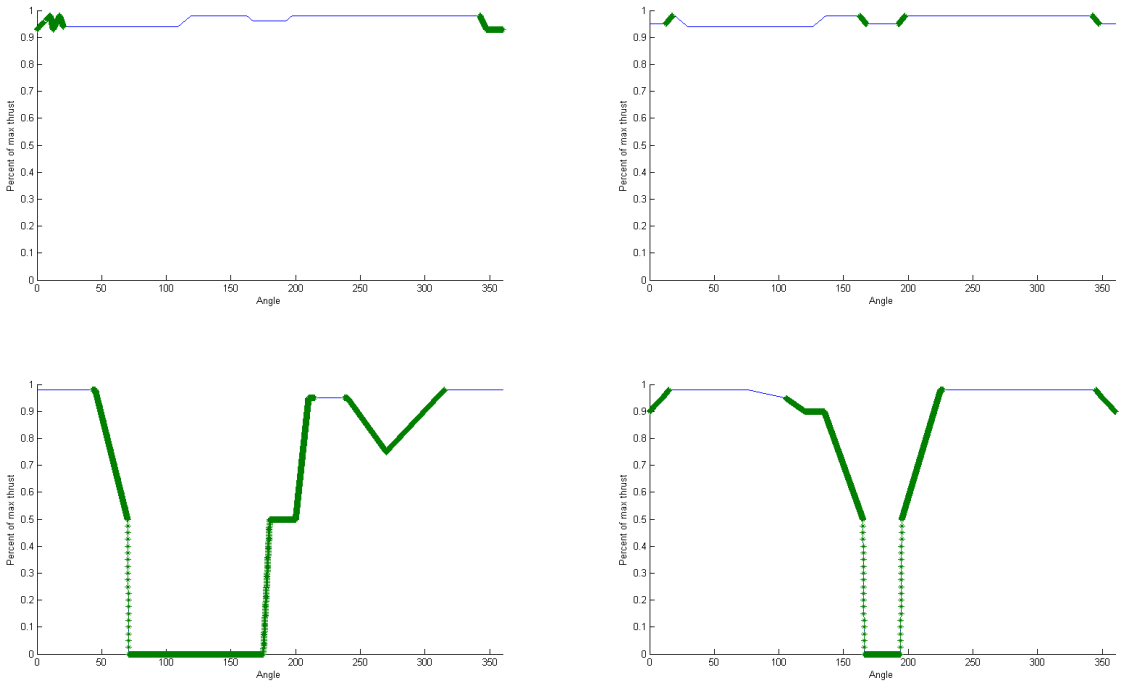


Figure 8.1: *Thruster efficiency functions, with markings where a 'gap' is defined to be. This definition is the core idea in the global search method, where if any iterate reaches a green point the algorithm initilizes a new instance on the other side of the gap.*

of moving to the other side, even if the other side is preferable. A function was defined to give the value 'true' if the algorithm is in the vicinity of such a dip (denoted as green dots in Figure 8.1). If a solution is selected that returns 'true' of this function, a new instance of the algorithm is initiated, starting from the other side of the dip. The point is that both sides of the forbidden zones will be explored, with added computational cost. This was implemented for an augmented lagrangian method, with a smoothed *eff* , using a small σ.

9

Computational results

The compiler used is MATLAB 2012b. The OS used is windows vista, and the system running the computations has 3 GB memory with an i3 M350 CPU at 2.27 GHz. We have set a time limit of 5 hours for solving the problem for 8 wind directions of the same wind conditions (i.e. using the same environmental field, but rotating it). If the algorithm takes longer, we deem it unsuccessful. Any algorithm that takes less time will have the solution compared to other solutions, time not taken into account.

9.1 Branch and bound

Branch and bound (described in Section 8.1) does not terminate within the time limit. It seems that the upper bound functions takes most of the execution time. We note that the upper bound function does not always return a feasible point, even when one exists, but rather gets stuck in a non-feasible local optimum. We also note that the number of leaves grows quickly to the tens of thousands.

9.2 Simulation based method

The simulation based method (described in Section 8.2) was implemented with two different subroutines, the results of which is described in the coming subsections. We mainly observe that, as seen in Figure 9.1, the smoothed function approximates the true function quite well. The difference between the objective function value of the filtered function and the true function at the solution found for $\sigma = 0.25$ is 0.47%.

9.2.1 Line search and penalty method

Using a single line search method (i.e. one of the following; coordinate descent, gradient descent, conjugate gradient descent, quasi-newton or newtons method) was worse than

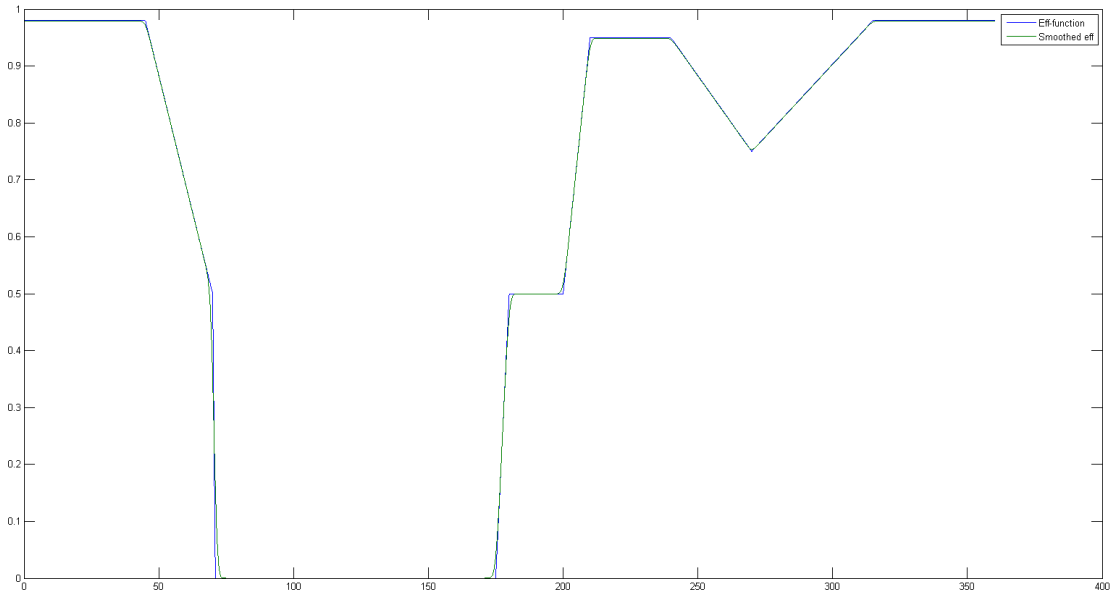


Figure 9.1: *The* eff *function, in blue, compared to the smoothed* eff *values, in green, for* $\sigma = 0.25$

trying all of them and selecting the best next step, with little improvement in running time. Regardless, the results from penalty method with line search was highly dependent on the starting point. The simulated based method with a penalty method, line search subroutine, does not find a global optimum. This is readily seen at 180°. This failure to converge to the global optimum is likely due to not being able to cross the forbidden zones, regardless of the smoothing trick.

9.2.2 Trust region and augmented Lagrangian

The augmented Lagrangian, with a trust region subroutine, produces the best results of the implemented algorithms. When cold starting the results are robust and often feasible, but rarely beating the current algorithm. When warm starting, using the solution found by the current algorithm (described in Chapter 5), the augmented Lagrangian method greatly improves the solution—particularly in cases where the current algorithm does not yield a feasible solution. We believe that the success of warm starting is due to the current algorithm handling the forbidden zones well, which means that the starting point for the augmented Lagrangian is in a neighbourhood of a good solution. This algorithm is a good fit for the current problem, and merits further research.

9.3 Global search (specific to DP)

The global search algorithm (described in Section 8.3) outperforms the other implemented algorithms when all are cold starting, except the current algorithm. Often 30 or more instances of the algorithm are created due to the forbidden zones, which means that this algorithm takes long to terminate. Comparing the global search (cold starting) to the augmented Lagrangian (warm starting), we see that the augmented Lagrangian generally finds a better solution, faster. In instances where the augmented Lagrangian does not find a feasible point, the global search method is a reasonable try before giving up. Further research might improve running times and performance, since the current implementation is naive.

10

Conclusion and future work

10.1 Conclusion

The static DP problem has properties which mean that most standard algorithms perform poorly. There are two algorithms detailed in this report, the global search (specific to DP, described in Section 8.3) and the augmented Lagrangian (described in Sections 7.5.2 and 8.2.2) that we recommend further research into. The augmented Lagrangian with a warm start improves on the current algorithm, and the global search might be a more general way of exploring the domain than the current rule based algorithm.
In practice, we recommend using the already existing algorithm to provide a warm start for the augmented Lagrangian with trust region solver, for a small σ (a value of 0.25 has sufficed in our computations) as the smoothing parameter. A big benefit of this approach is the optimality guarantees for the subproblem, which seems to have some transferability to the true problem. The second recommendation would be cold starting in the augmented Lagrangian method, and go through a decreasing sequence of σ. Although this leads to a great increase in computing time, this is more theoretically justifiable and hence we can make better guarantees for the found solutions. We recommend using this when warm starting fails (that is, when the existing algorithm does not yield a good starting point) to avoid a high computational cost.
Honorable mention goes to the branch and bound algorithm as well as the global search augmented Lagrangian. The branch and bound method is currently intractable, but would have the advantages of not using a subproblem, as well as netting an approximate global optimum, hence further research to try to make it tractable is warranted. Currently, the main issues seem to be finding a feasible point quickly and guaranteed (if one exists) in the sub domains created, as well as updating the leaves when new bounds are found. The main issue with the global search augmented Lagrangian is the time taken. The main ideas to continue researching is ranking the different sub domains (divided due to forbidden zones) with some heuristic and stop when satisfied with the solution—or

discard those sub domains that seem to yield bad solutions after few iterations.

10.2 Future work

There are two main extensions of the work done in this thesis. The first is to analyze a dynamic DP problem, when the whether conditions can change. How to optimize performance under changing environment. It might be possible to infer a decent solution from the solutions to the static DP problem, which is why the dynamic DP problem is a natural extension. The second natural extension is to allow the thruster placement to change. This is hence a multi-objective optimization, where different thruster placements impact the cost and other properties of the floating structure, but the placements also impact the capabilities of the floater with regard to neutralizing environmental forces.

Bibliography

[1] American Bureau of Shipping, Guide for dynamic positioning systems.

[2] J. Dang, H. Lahehij, Hydrodynamic Aspects of Steerable Thrusters.

[3] N. Andréasson, A. Evgrafov, M. Patriksson, An Introduction to Continuous Optimization.

[4] T. Hangelbroek, A. Ron, Nonlinear approximation using gaussian kernels.

[5] K. Holmström, N.-H. Quttineh, M. Edvall, An Adaptive Radial Basis Algorithm for Expensive Black-Box Mixed-Integer Constrained Global Optimization.

[6] J. Nocedal, S. Wright, Numerical Optimization.

[7] M. Wahde, Biologically Inspired Optimization Methods: An Introduction.

[8] C. A. Coello Coello, A Survey of Constraint Handling Techniques used with Evolutionary Algorithms, Computer Methods in Applied Mechanics and Engineering 191 (2002) 1245–1287.

[9] G. D. Pillo, Exact penalty methods, in: In I. Ciocco (Ed.), Algorithms for Continuous Optimization, Kluwer Acad. Publ, 1994, pp. 1–45.

A

Tables

	1		2		3		4		5		6		7		8	
Direction	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment
0	865.1	-12	1079	59.4	1473.3	180.9	2076.7	354.9	2860.3	583.1	3675.9	865.3	4491.3	1208	5375.4	1618
45	2018.8	-18390	2318.4	-23540	2824.7	-32590	3552.1	-44930	4534.4	-59020	5757	-75530	7141.8	-96450	8625.8	-122200
90	2409	3441	2671.1	564	3095.7	-4028	3721.7	-10540	4663.1	-19030	5916.8	-29770	7332.1	-42890	8781	-58440
135	2006.8	36450	2308.7	33360	2820.3	28900	3553.5	22230	4527.1	11690	5700.9	-2601	7002.3	-18820	8383.6	-36600
180	928.4	-630.2	1152.1	-468.9	1562.1	-201.5	2186.1	175.2	2997.3	652.1	3845.9	1212	4695.7	1844	5617.1	2540
225	2017.3	-34340	2306.8	-31020	2797.9	-26150	3502.5	-18920	4438.9	-7692	5567	7448	6812.7	24670	8130.2	43610
270	2379.1	-1810	2641.3	1418	3065.4	6566	3690.1	13820	4631	23220	5885.4	34990	7303.6	49170	8758.4	65770
315	1991.3	17430	2292.5	23140	2801.9	33170	3534.2	46900	4523.3	62820	5755.6	81690	7132.8	104800	8607.3	133000
360	865.1	-12	1079	59.4	1473.3	180.9	2076.7	354.9	2860.3	583.1	3675.9	865.3	4491.3	1208	5375.4	1618

	9		10		11		12		13		14		15		16	
Direction	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment	Force	Moment
0	6317	2103	7313.8	2675	8369.8	3344	9499.9	4127	10705.9	5044	12004.5	6114	13396.1	7358	14886.5	8796
45	10179.2	-152900	11817.7	-188700	13570.9	-229400	15448.8	-275300	17449.3	-326700	19490.1	-381500	21534.7	-438000	23679	-498200
90	10205.4	-76420	11643.2	-96910	13150.3	-120000	14744	-145600	16423.2	-174000	18199.4	-205000	20071.6	-238600	22054.4	-275000
135	9819.8	-56020	11330	-77310	12941.4	-100500	14665.9	-125400	16506.2	-152000	18474	-180300	20570.1	-210400	22802.8	-242400
180	6599.1	3292	7640.3	4090	8743.3	4924	9922.2	5789	11178.8	6682	12529.5	7599	13976.5	8542	15523.5	9505
225	9494.2	64390	10924.4	87180	12447.6	112000	14076.4	138700	15813.2	167200	17670.5	197700	19649.2	230000	21757.4	264400
270	10192.5	84730	11644.2	106100	13169.4	129800	14785.5	156000	16491.2	184600	18297.7	215600	20203.5	249000	22223	285000
315	10141.1	166700	11747.9	205500	13456	249600	15273.9	299300	17169.4	354300	19024.9	413000	20951.9	475700	22958.6	542800
360	6317	2103	7313.8	2675	8369.8	3344	9499.9	4127	10705.9	5044	12004.5	6114	13396.1	7358	14886.5	8796

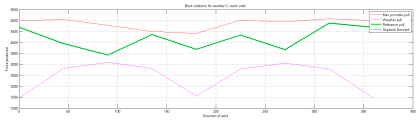
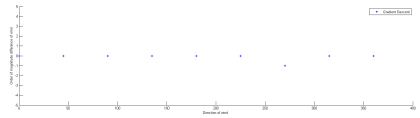
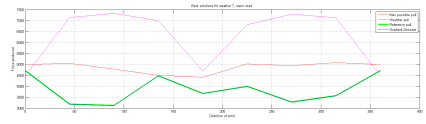
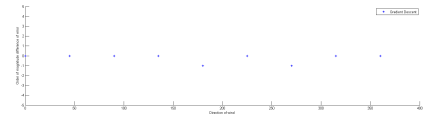
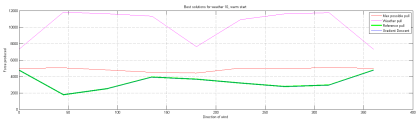
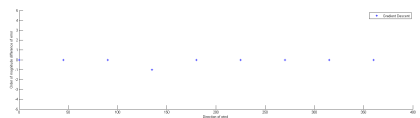
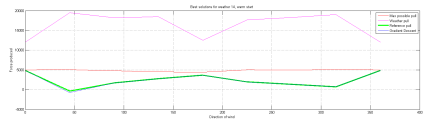
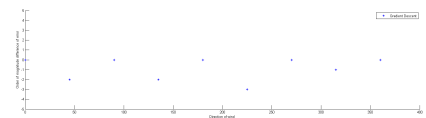
B

Complete results

B.1 First implemented algorithms

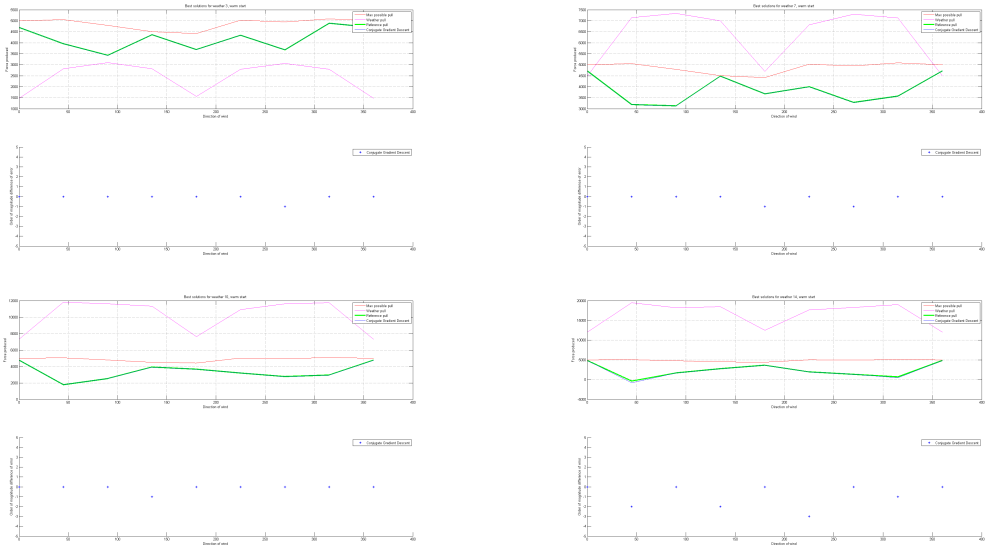
B.1.1 Using dpopt as warm start

Gradient descent

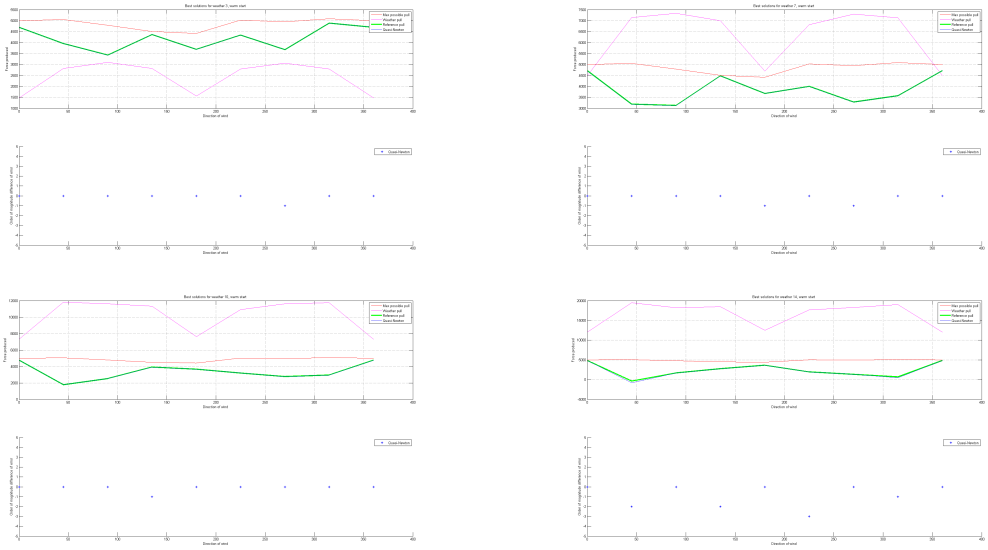
Conjugate Gradient descent

The algorithm tries both the next conjugate gradient descent step, as well as the pure gradient descent step, and selects the one with most decrease.



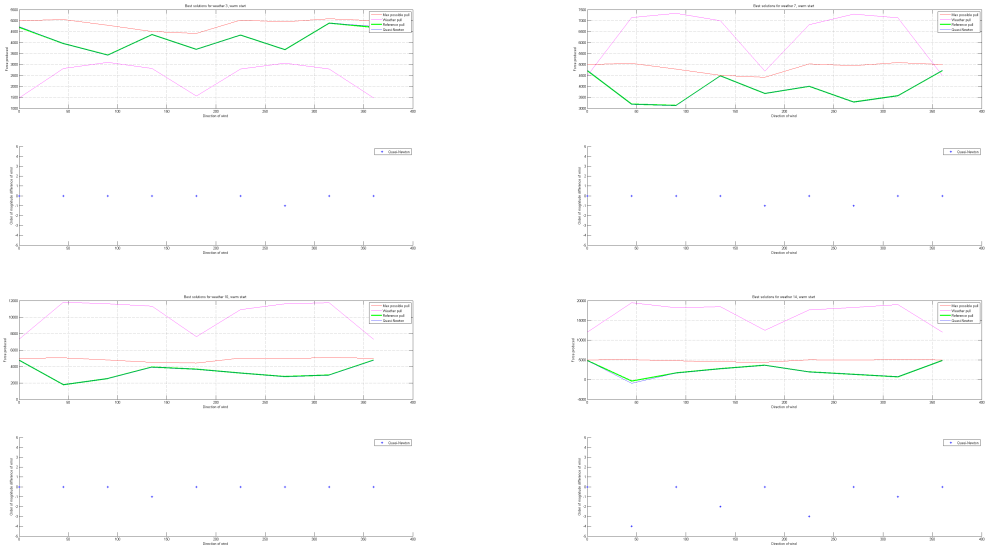
Quasi-Newton

As above, we enable the quasi-Newton step while retaining the possibility of conjugate gradient descent and gradient descent.

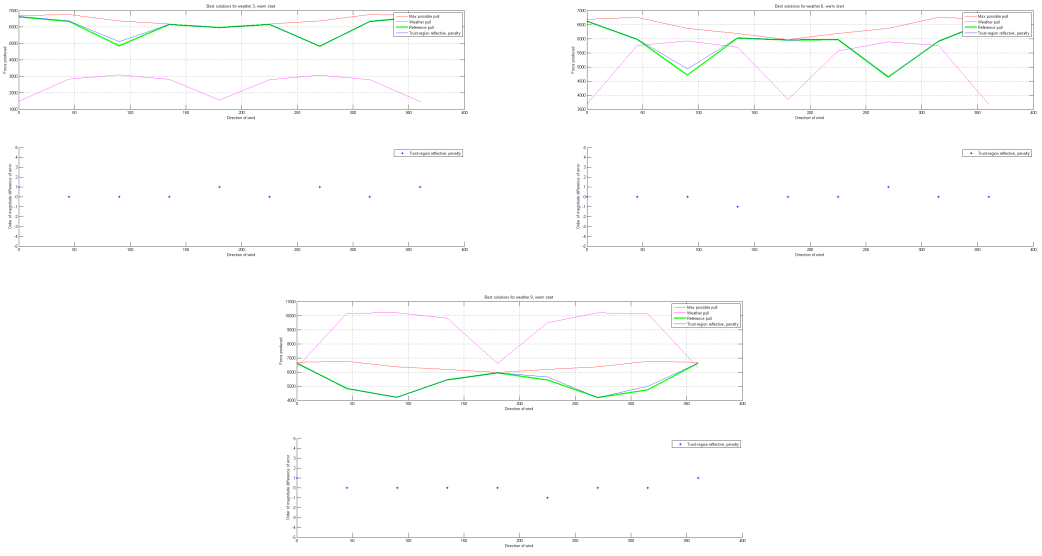


Newton

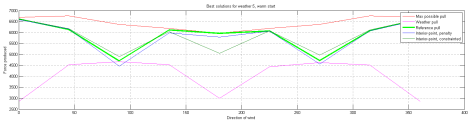
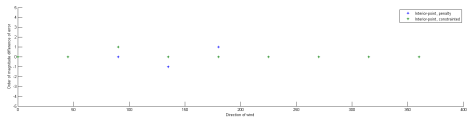
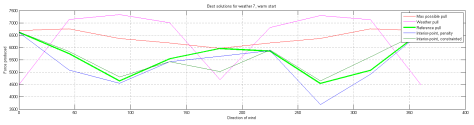
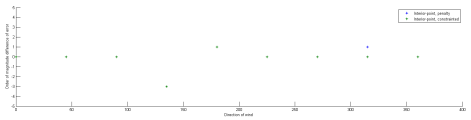
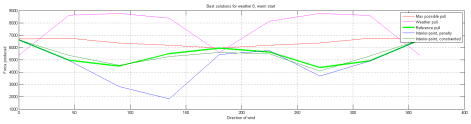
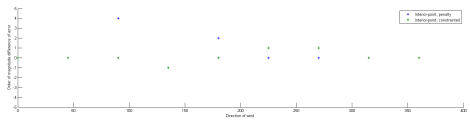
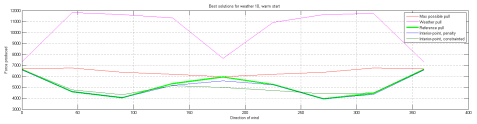
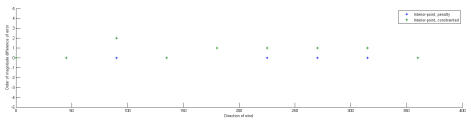
Here the algorithm tries all four possibilities above, that is; Newton, Quasi-Newton, Conjugate gradient descent and Gradient descent.



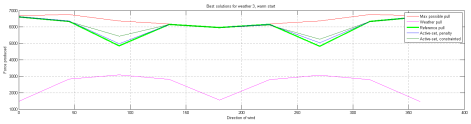
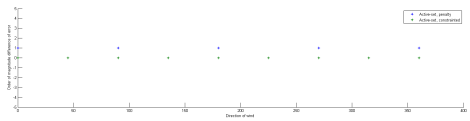
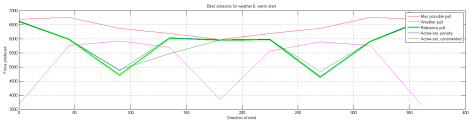
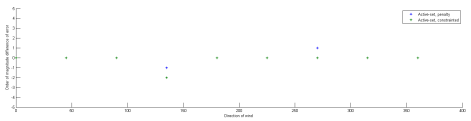
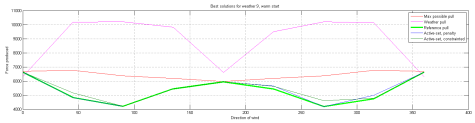
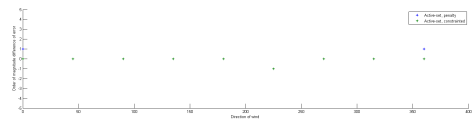
Trust-region reflective



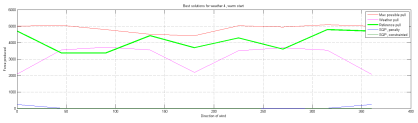
Interior-Point

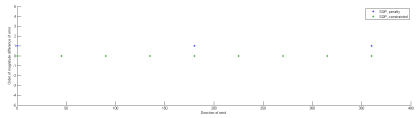









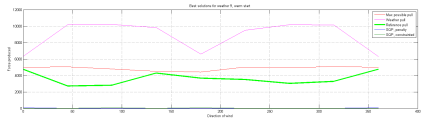
Active-set

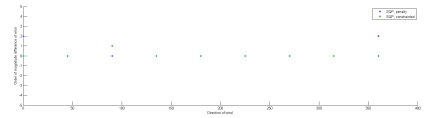







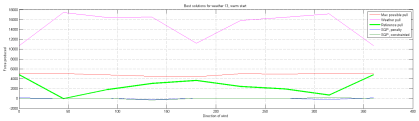
Sequential quadratic programming

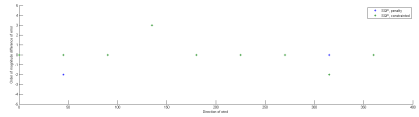






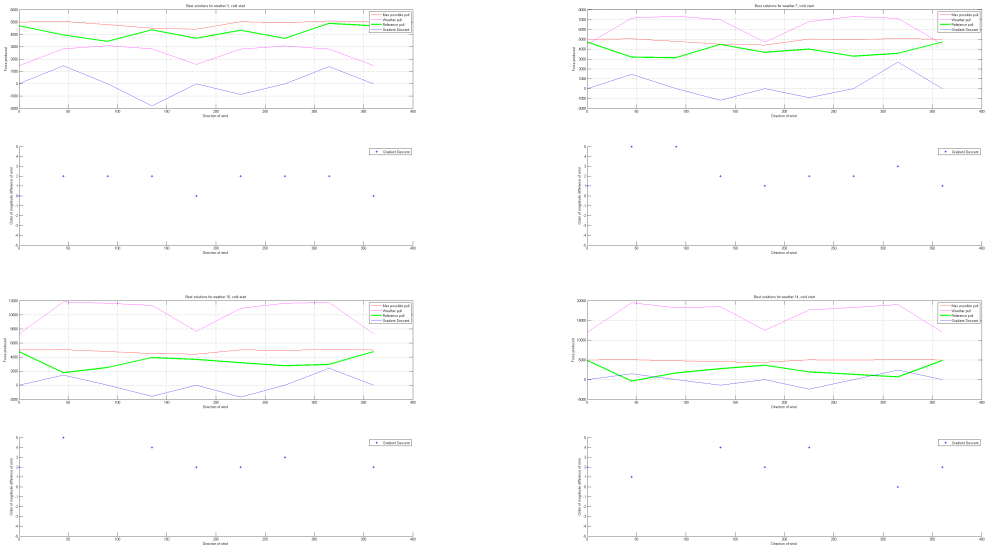






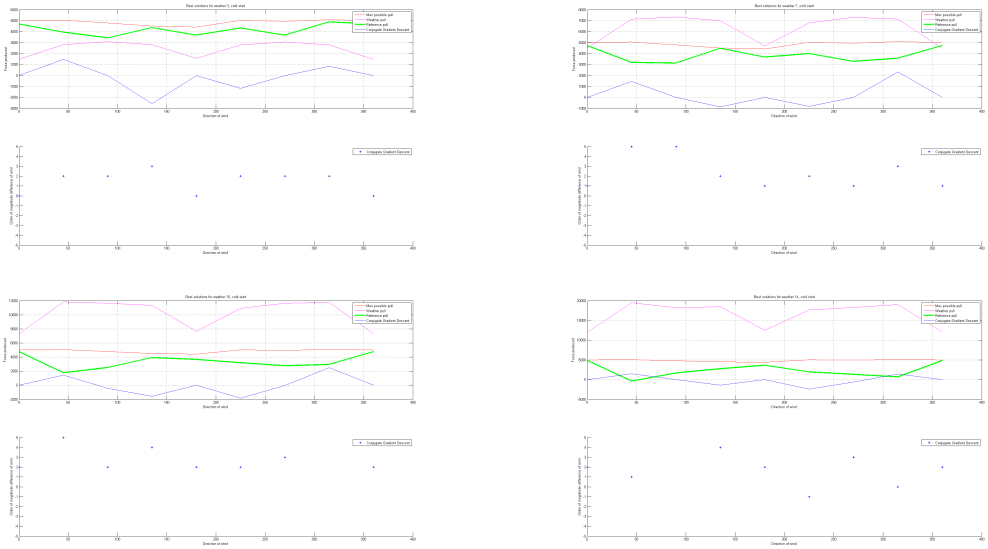
B.1.2 Cold start

Gradient descent



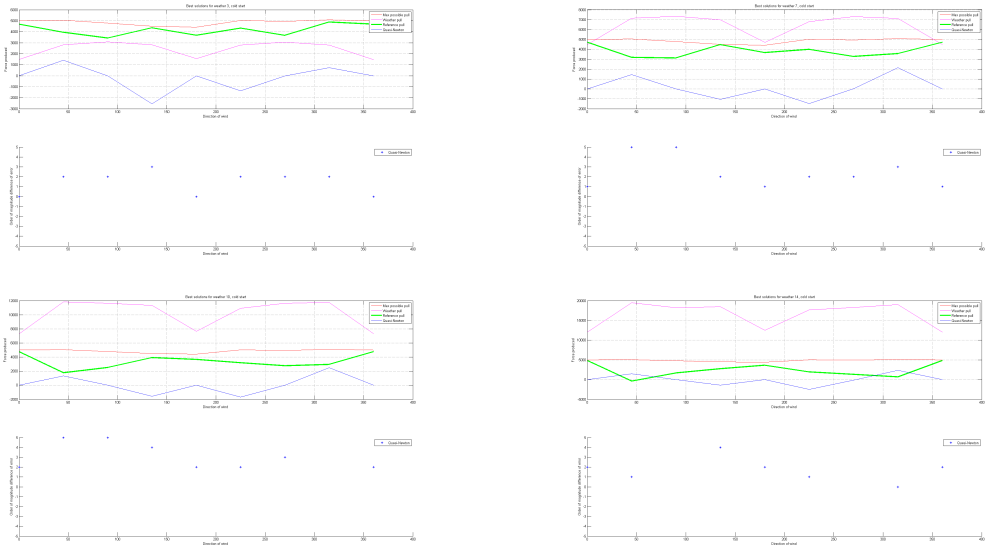
Conjugate Gradient descent

The algorithm tries both the next conjugate gradient descent step, as well as the pure gradient descent step, and selects the one with most decrease.



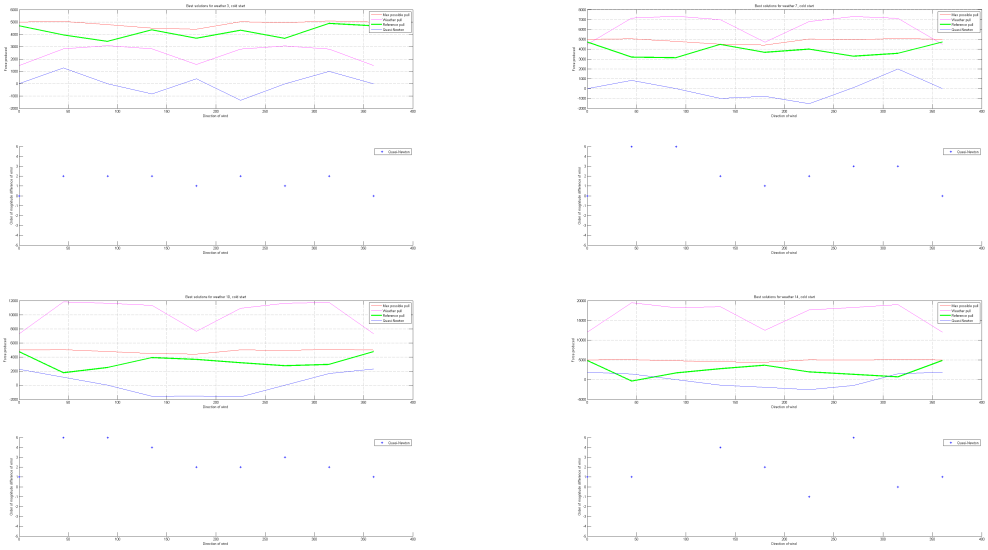
Quasi-Newton

As above, we enable the quasi-Newton step while retaining the possibility of conjugate gradient descent and gradient descent.

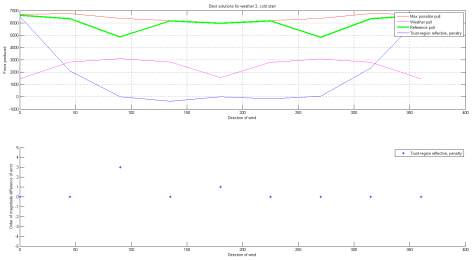


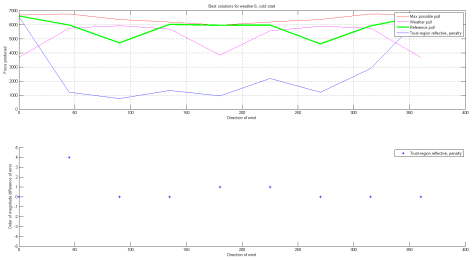
Newton

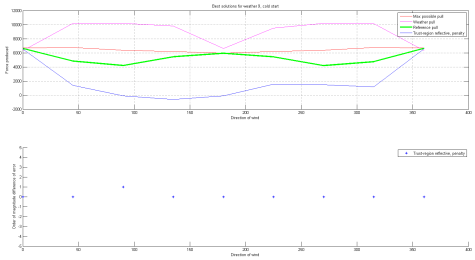
Here the algorithm tries all four possibilities above, that is; Newton, Quasi-Newton, Conjugate gradient descent and Gradient descent.



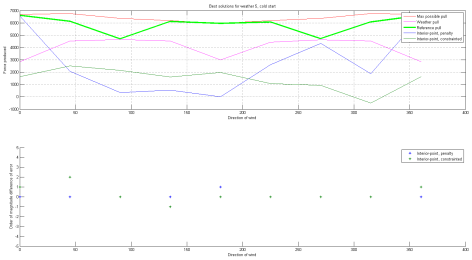
Trust-region reflective

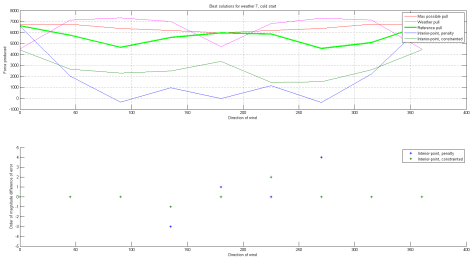


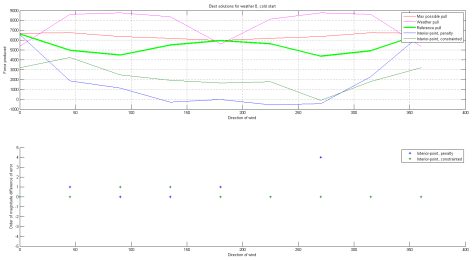


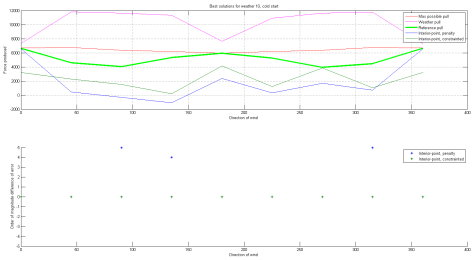


Interior-Point

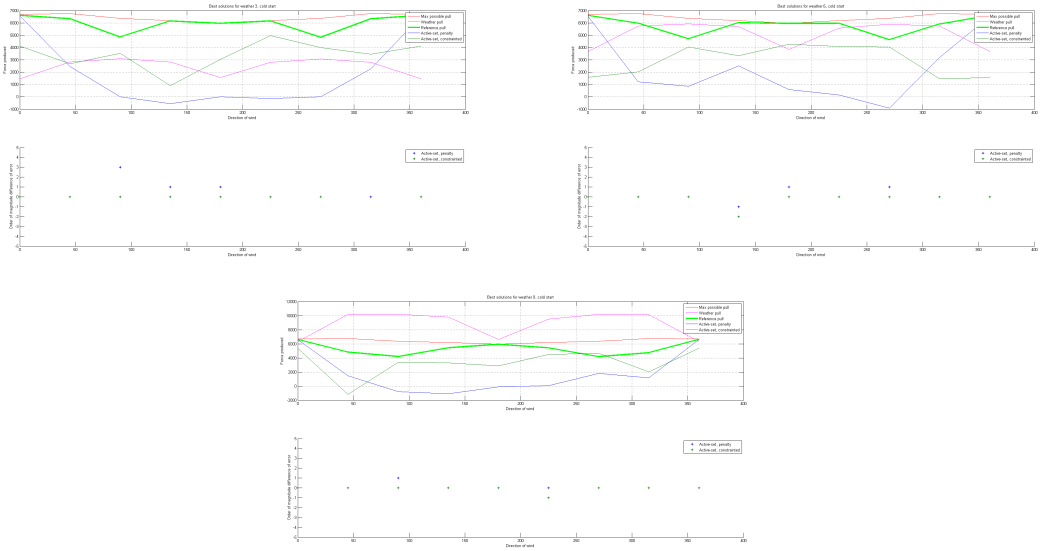




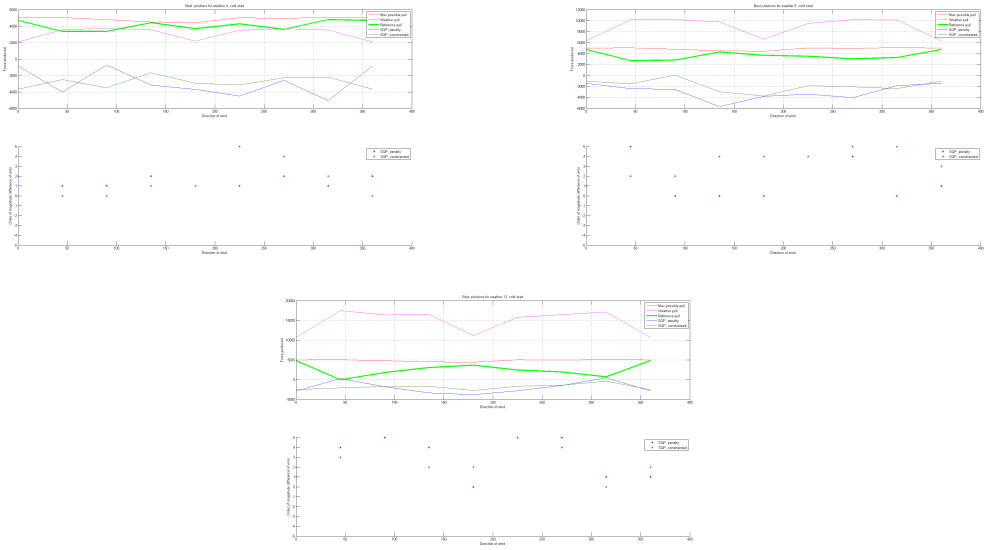




Active-set



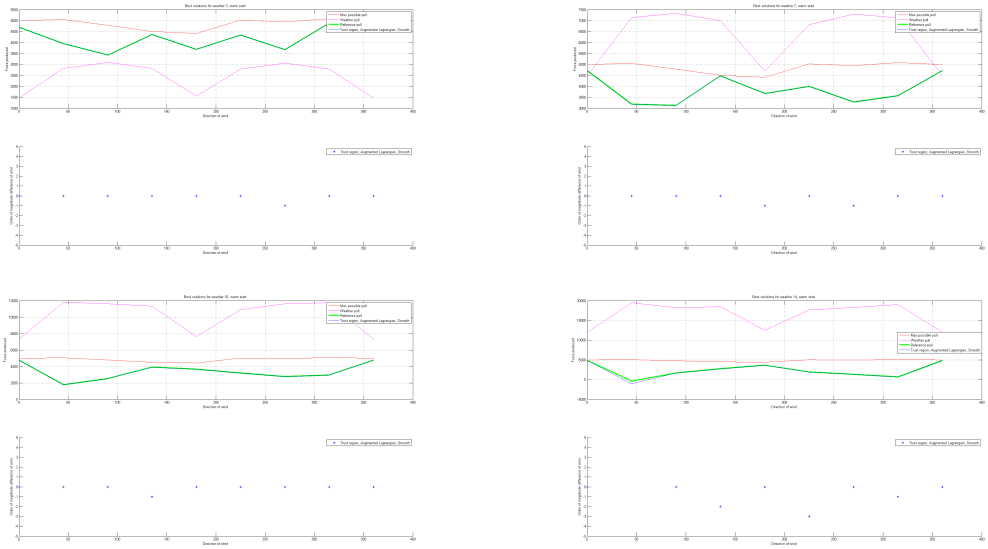
Sequential quadratic programming



B.2 Final results

B.2.1 Augmented Lagrangian, Smoothed *eff*, Trust region solver

Warm start



Cold start