NTNU – Trondheim
Norwegian University of
Science and Technology

Nonlinear Model Predictive Control using Derivative-Free Optimization

Jon Dæhlen

Master of Science in Engineering Cybernetics (2)
Submission date: June 2013
Supervisor: Tor Arne Johansen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

Today it is common to solve the Non-Linear Programming (NLP) problem arising in NMPC by the use of gradient-based optimization. However, these techniques may not be suited if the prediction model, and thus the optimization problem, is not differentiable. Such cases can arise when the prediction model contains logic operators, lookup-tables or a variable-step ODE-solver is used to simulate the prediction model.

Although the model may be made differentiable by alterations or using another ODE-solver, this can compromise the accuracy of the prediction, and thus the performance of the NMPC can suffer. Therefore it is desirable to investigate techniques that can solve a NLP without requiring the gradient of the objective function or its constraints.

Derivative-Free Optimization(DFO) has been subject to substantial research, as it is common for gradient information to be unavailable in optimization. This class of algorithms have been frequently applied to simulation-based optimization, as well as to some extent to NMPC. However, for the latter the studies have mainly been limited to smaller and simpler systems.

This thesis will investigate some theoretical fundamentals of DFO, and present some common practical algorithms. Then a case-study is performed which develops a NMPC for a particular industrial crude-oil separator. This controller is simulated using the presented algorithms, as well as a gradient-based SQP. The experience gained is then used to modify an existing algorithm to improve real-time performance.

The findings of the thesis are that DFO is significantly more robust against the numerical issues which the model of the case-study inhabits compared to the SQP tested. Further, one of the DFO-algorithms is comparable with the particular SQP, both with respect to computational consumption and accuracy of the solutions. Another algorithm is also very robust against numerical issues. Distribution of the computational load over several processing cores can improve real-time performance significantly, as well as minimizing the computational consumption by reducing the number of predictions required at each time step. This makes DFO a promising alternative to gradient-based optimization in NMPC.

Sammendrag

I dag er det vanlig å løse det ikke-lineære programmerings problemet (NLP) som oppstår i NMPC ved bruk av gradient-basert optimalisering. Disse teknikkene kan være lite egnet hvis prediksjonsmodellen, og således optimaliseringsproblemet, ikke er deriverbar. Slike tilfeller kan oppstå når predisjonsmodellen inneholder logiske operatører, oppslagstabeller eller hvis en variabel steglengde ODE-løser brukes til å simulere prediksjonsmodellen.

Selv om modellen kan gjøres deriverbar ved å endre denne, eller bruke en annen ODE-løser, kan dette gå ut over nøyaktigheten av prediksjonen, og ytelsen til NMPC kan dermed lide. Derfor er det ønskelig å undersøke teknikker som kan løse en NLP uten at det kreves gradient av objektfunksjonen eller dens bibetingelser.

Deriverings-Fri Optimalisering (DFO) har vært gjenstand for mye forskning, da optimalisering hvor gradient informasjon er utilgjengelig er et vanlig problem. Denne klassen av algoritmer har vært hyppig brukt i simulering-basert optimalisering, og til en viss grad også i NMPC. Men for sistnevnte har studiene i hovedsak vært begrenset til mindre og enklere systemer.

Denne oppgaven vil studere det teoretiske grunnlaget for DFO, og presentere noen vanlige praktiske algoritmer. Deretter er en case-studie utført som utvikler en NMPC for en bestemt industriell råolje-separator. Denne regulatoren er simulert ved hjelp av de presenterte algoritmene, samt en gradient-basert SQP. Erfaringene fra disse simularingene er så brukt til å modifisere en eksisterende algoritme, for å forbedre sanntidsytelse.

Funnene i oppgaven er at DFO er betydelig mer robust enn SQP-algoritmen som er testet mot de numeriske problemstillingene som prediksjonsmodellen i case-studiet innehar. En av DFO-algoritmene er definitivt sammenlignbar med den testede SQP algoritmen, både med hensyn til krav til beregningskraft og nøyaktigheten av løsningene, mens en annen er svært robust mot numeriske problemer. Videre kan distribusjon av beregningene over flere prosessorer gi betydelig forbedring i sanntidsytelse, samt å minimere nødvendig datakraft ved å redusere antall prediksjoner nødvendig i hvert tidssteg. Dette gjør DFO til et lovende alternativ til gradient-basert optimalisering i NMPC.

Preface

This report documents the work on my MSc thesis at the Department of Engineering Cybernetics, during the final semester at the Norwegian University of Science and Technology(NTNU).
I would like to thank my supervisor, Prof. Tor Arne Johansen at the Centre for Autonomous Marine Operations and Systems at NTNU for his good advice and guidance during my MSc thesis.
Gisle Otto Eikrem at Statoil provided the dynamic model of the crude-oil separator used for the case-study in the thesis. It was invaluable to have a model of an industrial application for the testing of the resulting controller, and made both the work and the final result significantly more interesting.
A thank also goes to D. Kwame Kufalor (Giorgio) for important input during the implementation, and highly interesting discussions on the topic of MPC and control engineering.

Trondheim, Norway
June 2013

Jon S. Dæhlen

Contents

Chapter 1

Introduction

Model Predictive Control (MPC) technology uses a model of the process to be controlled to predict the future behaviour of this and manipulates its control inputs to achieve a desired control objective. This is accomplished by solving an open-loop optimal-control problem at each time instant[38], by applying a numerical optimization algorithm. It is known for its general handling of constraints in the states and inputs and inherit capability of Multiple-input multiple-output (MIMO) systems. The method has been widely used in the process industry where the plant works around a relatively constant set-point and the dynamics are slow. It has been and still is controllers based on models that is linear approximations to the real plant and limitation to linear constraints on the states and inputs that is the most common. This is referred to as linear MPC and gives some numerical advantages that greatly simplifies the optimization within the controller[5]. However in the later years Nonlinear Model Predictive Control (NMPC) has started to attract much attention[10], as the use of non-linear models improves the controller performance on highly non-linear systems and allows for operation over wider range of set points[5]. Also NMPC allows for non-linear constraints which gives a great deal of flexibility.

Today the probably most common method for solving a NMPC problem is by linearising the model around the previous optimal trajectory, referred to as the nominal trajectory. This is often done by applying a small perturbation to each input and build a step-response model. This can be optimized as in linear MPC, and a new linearisation is performed in the next time step around the updated nominal trajectory[23]. This however makes the model only valid around small perturbations from its current state, and e.g. if the set-point is changed, prediction of larger transients may be desirable. Today state-of-the-art to overcome this seems to be Sequential-quadratic programming (SQP) and Interior-Point (IP)-methods[15]. Common for both is that they recursively linearise the model and the constraints until convergence is achieved, which in turn requires to somehow retrieve the gradient of these.

If the model is known explicitly, an off-line differentiation can be performed and the gradient and possibly the Hessian implemented. For simple systems this process can be done manually, however as models usually are complex and to avoid calculation error the use of symbolic mathematical computation software is preferable. Examples of such software can be Maxima[3], Maple[4] and a toolbox is available for Matlab[2]. Given the expression for the model, this software will perform an automatic differentiation according to calculus rules and give an analytical answer. Some software also has the ability to return the solution in source code in a selection of programming languages, for easy implementation. These methods will however require the expression of the model to be known. Another somewhat similar software is ADIFOR. This is a Fortran-subroutine that given a model implemented as a subroutine in this language, ADIFOR will return source code that gives the partial derivatives of this model[1].

However all of the above techniques will require that the model is described and available explicitly. This may not always be the case, as it may not be available in an appropriate programming language, can be made up of a mix of subroutines from different programming languages or the source can simply by unavailable. This will render the symbolic software in the best case hard to apply[31]. The model may even not be continuous, as logic operators and lookup-tables are common, and the model may not be explicitly available at all as it may be a numerical simulation[42]. In these cases it is likely that the most common and intuitive is to retrieve the gradient from finite-differences. This method is however known to be sensitive towards numerical issues[46], and when performing the simulation it can be desirable to use a variable-step Ordinary Differential Equation (ODE)-solver. This can speed up the simulation time significantly, however it is known that this tends to induce numerical noise, non-smoothness and discontinuities, which possibly can be amplified through finite-differences and thus compromise the performance of the NMPC.
This may in some cases be avoided using a fixed-step size in the ODE-solver. However choosing a small step-size will make the simulation more time-consuming, and large step size will make it less accurate, both directly inflicting the performance of the controller. Further, the model may be simplified to be able to be expressed as an explicit ODE. Again, this will necessarily make the model less accurate. Also, developing a new model may be a time-consuming process as model development and validation is considering one of the most time consuming task in MPC development[23]. This motivates for investigating optimization methods not requiring derivatives, namely Derivative-free optimization (DFO).

DFO has been a subject for much research, as optimization problems where the gradient is unavailable or unreliable always has been present[12]. It is well-known that much information is contained in the derivatives[12], and this is reflected in the fact that DFO usually has a slower convergence rate than its gradient-based counterpart. DFO tends to be characterized by requiring a larger number of evaluations of the objective to compensate for this lack of information. DFO has been applied in NMPC in several occasions, by use of Genetic Algorithms (GA)[10][43][5][55][9]. However in these cases the goal has been to overcome extreme non-linearities and improve computational consumption by finding sub-optimal solutions, and although GA is known to be well-suited towards noise and discontinuities it also tends to have slow convergence rate. [57] used the Nelder-Mead simplex method for solving a highly non-linear NMPC problem and reported that this were 10 times faster than GA, and [54] used parallel computing on a Graphics Processing Unit (GPU) to obtain even faster computation with the algorithm. These did however suggest to try other optimization methods and more practical problems. [32] reported that a Derivative-free Trust-region method (DFTRM) out-performed the Nelder-Mead method in an optimal-control problem, however the exact nature of the optimization problem was not stated. DFTRM is also known to require less function evaluations than algorithms as Nelder-Mead and GA[42], thus investigating the use of these methods in NMPC aimed at controlling a realistic process seems appropriate.

DFTRM is much used in simulation-based optimization[35][22][18][26], which essentially is the same optimization problem as in NMPC, only that the time available for optimization is significantly longer and performed only once. The reason for this lack of coverage is probably that as DFTRM is neither suited for extreme non-linearities nor can be expected to be faster than gradient-based optimization and will only have an advantage in the special case when models with numerical issues as described are applied. However as they should be expected more effective than GA on approximately quadratic problems[42], which often is the case in NMPC, and that the ability to use such challenging models in NMPC definitely is attractive, this encourages investigation of DFTRM in NMPC. Also several new DFO algorithms has emerged in the later years[52][7][36][42], and together with the increased availability of computational power it seems appropriate to investigate the suitability for DFO in NMPC. The starting point for the work in this report is [13]. This is a investigation done preliminary for this report to get a overview of the field of DFO.

1.1 Outline

The rest of the report will be structured as follows:

- Chapter 2 will present some background in the field of optimization and control, and introduce some notation used throughout the report.
- Chapter 3 will present some fundamental mathematical concepts used in state-of-the-art DFTRM, and focus on interpolating a polynomial model and the concept of Trust-Region (TR) that these methods are based on.
- Chapter 4 applies this theory were first an introduction to typical DFO algorithms are presented, then a practical genetic algorithm and the basic structure of a DFTRM is investigated. The important issue of constraint handling is briefly discussed, and three well-known algorithms are reviewed in detail.
- Chapter 5 presents and analyses a non-linear system with the characteristics which generates a challenging NMPC problem with respect to differentiation. Then a NMPC controller is developed to control this system.
- Chapter 6 simulates this controller in closed-loop, testing the three different algorithms investigated in chapter 4 for solving the optimization problem. A SQP algorithm is also applied for comparison, and to show the difficulties that gradient-based algorithms may run into.
- Chapter 7 will use the insight gained during theory and simulations to propose what can be done to make an improved algorithm that exploits the properties of the NMPC problem to make DFO applicable in a real-time application. Such a modification is done to one of the algorithms from chapter 4, and the algorithm is simulated on the same problem as in chapter 6.
- Chapter 8 concludes the findings of the report and suggests further work on the subject DFO in NMPC.

Chapter 2

Optimal control preliminaries

2.1 Optimization

Optimization is choosing the best among a set of choices, given the current circumstances. The choice to be taken can be of both discrete nature e.g. being simply to choose "yes" or "no" to a dilemma by weighting the implications of the choices, or continuous e.g. to choose the setting of the gas pedal of a car, trying to choose the best compromise between fuel consumption and travel time. In mathematical terms continuous optimization is that the domain of decision variables is continuous, like a subset of an euclidean space.
Continuous optimization is commonly used in several fields, from optimizing income and minimizing risk in economics, optimizing production efficiency at a plant[40] to engineering application when optimizing design parameters. An example of the latter is mentioned in [6], as it briefly describes how optimization is applied to find the optimal values for 31 variables describing the rotor blades of a compressor. Optimization are being applied to an increasingly number of fields, which generates more challenging optimization problems and demands for methods to solve these.

To solve a practical optimization problem, the case at hand must be described in terms of a mathematical set of equations, referred to as the objective. As all cases are different, there exist no general procedure for accomplishing this, however the goal is to end up with a function $f(x)$ that is to be minimized or maximized. The argument x is the set of variables that are to be chosen, referred to as Manipulated variables (MV). A minimization problem is usually set up on the form

$$\min_{x \epsilon \mathbb{R}^N} f(x) \tag{2.1a}$$

$$\text{s.t}$$

$$c_i(x) \leq 0, i \in \mathcal{I} \tag{2.1b}$$

$$c_i(x) = 0, i \in \mathcal{E} \tag{2.1c}$$

where $f(x) : \mathcal{R}^n \mapsto \mathcal{R}^1$ is the function to be minimized, referred to as the objective function. This value is often referred to as "cost", to illustrate that it is desirable to minimize the value of $f(x)$. If the only goal of the optimization is to minimize the cost, without any regard of the values of x, i.e. $(\mathcal{I} = \mathcal{E} = \emptyset)$, the problem is referred to as an unconstrained minimization problem. On the other hand if it in addition is introduced constraints on the MV as illustrated by equation (2.1b) and (2.1c) it is referred to as a constrained problem. The constraints are again divided into inequality(2.1b) and equality constraints(2.1c). Simple examples of such can be for the former that the power setting of a engine must lie between zero and 100%, and for the latter that the flow into a tank must be equal to that out.

Constrained problems are again divided into classes after how hard they are to solve. The simplest and frequently used are the Linear Programming (LP)class[40]. This class has a linear objective function and only equality constraints that are linear. The class is commonly solved with the well-known simplex algorithm, or the more modern IP method[40] which will be explained later in this section. Further a more complex class is the Quadratic Programming (QP) class. No standard definition of this class exist[40], however convexity plays an important role. If the objective $f(x)$ is a quadratic function with a positive-definite Hessian matrix, then it is a convex one, i.e. by following the direction of the gradient the minima will always be found. Introducing constraints the problem is still convex if c_i are linear for $i \in \mathcal{E}$ and concave for $i \in \mathcal{I}$[40]. If these conditions are fulfilled it is common to refer to the problem as a QP and it can be solved with well-known methods such as active-set and interior-point[40]. Also because the QP is convex, it inherits no local minimum points. Thus the solver can guarantee to find the global solution[39].

The last class to be discussed is the Non-Linear Programming (NLP), which will be the subject for this report. This problem typically arises when the criteria to the QP are not fulfilled. Often the objective $f(x)$ is not a purely quadratic

function, or a non-linear equality function arises, as will be shown to be the case for NMPC applications.
No general method exist that can solve this class of problem and guarantee a global minima in a finite time[39], because the optimization problem are no longer necessarily convex[27]. A common method to find at least a local minima is SQP. This method approximates the objective and constraints with quadratic and linear models respectively for each iteration to create a QP, and solves this with a QP solver such as active-set to find a good search direction. Then a line-search is performed along this direction, which is terminated when certain conditions of sufficient decrease are fulfilled[40].

Just as common is the IP or barrier method[15]. Two main approaches exist for this method. One being to introduce a slack variable into the inequality constraints, and minimize this in a log-barrier function with a variable penalty, together with the objective function. This way the inequality constraints are eliminated, and the problem is solved repeatability using Newtons direct method with decreasing penalty on the barrier function. When the penalty reaches zero, a solution is found.
The second approach is also based on eliminating the equality constraints by the use of log-barrier functions, however it applies the Karush-Kuhn-Tucker (KKT)-conditions. By creating a equality-constrained minimization problem of these and solving as in the approach described above, a point that satisfies the conditions and thus by definition is a minima is approximately found. For further reading [40] is recommended reading.

Other techniques exist which are considered semi-global, meaning that they have features to avoid local minimum points and often find the global solution to a non-linear, non-convex problem, however not always. Examples of this is GA[10], DIviding RECTangles (DIRECT) and DFTRM also can be categorized as semi-global[26]. These are typically methods that combines several techniques with randomness to obtain a larger degree of "exploration" of the objective than is typical for gradient-based algorithms as QP solvers and SQP. However still no technique can guarantee a global optimum to general a non-convex problem in a finite time.

2.2 Finite-horizon optimal-control and LQR

Before introducing MPC, the finite-horizon optimal-control problem is presented. This is the problem that a MPC solves at each time step, and is concerned about optimizing the trajectory of a dynamic system over a given time-horizon[17].
The goal of the controller is to find an optimal set of inputs $u(k)$ that controls the real system, from now on referred to as the plant, in a desirable(optimal) way[37]. To accomplish this the controller uses a model that approximates the behaviour of the plant, called the prediction model[27]. Such a model is usually developed to be a continuous differential equation in, i.e. $\dot{x} = l(x, u)$. However to be implementable it must be discretised. This leads to the model on the form of a difference equation, which will be denoted $x(k+1) = g(x(0), x(k), u(k))$[38]. Note that this model might as well be time-variant[34], however it is kept time-invariant for convenient notation. Then a typical finite-horizon optimal-control problem can be

$$\min_{x\epsilon\mathbb{R}^N, u\epsilon\mathbb{R}^n} \quad f(x(1), \ldots, x(K), u(0) \ldots u(K-1), x(0)) =$$

$$\sum_{k=0}^{K-1} [x_e(k+1)^T Q x_e(k+1) + u_e(k)^T R u_e(k) + \Delta u(k)^T U \Delta u(k)] \tag{2.2a}$$

$$+x_e(K)^T T x_e(K) \tag{2.2b}$$

s.t

$$x(k+1) = g(x(k), u(k)), x(0) \text{ given} \tag{2.2c}$$

$$x_{min} \leq x(k) \leq x_{max} \tag{2.2d}$$

$$u_{min} \leq u(k) \leq u_{max} \tag{2.2e}$$

$$\Delta u_{min} \leq \Delta u(k) \leq \Delta u_{max} \tag{2.2f}$$

Here $\Delta u(k) = u(k+1) - u(k)$, and K is referred to as the prediction-horizon and is the number of time steps into the future that the behaviour will be considered[27]. If the number of steps that the inputs are optimized had been shorter, this would be referred to as the control-horizon, however in this case these two are equal. Further $x(0)$ is given, and is the initial state of the system. $x_e(k) = x(k) - x_{ref}$ is the predicted control error and $u_e(k) = u(k) - u_{ref}$. The reference vectors can be time-dependent if this is desirable, but are here kept constant for simplicity of notation. In addition, if the time between the samples are not equal, e.g. by the use of variable-step solver in (2.2c), a discrete integration method as the midpoint-rule must be applied to weight each sample equal with respect to its time duration. The above formulation uses the L_2-norm to minimize the control error. This seems to be the most common, however both L_1 and L_∞-norm have been seen in early research, leading to a LP-problem[19]. Inspecting the objective function (2.2a), a choice of the Q, R, U and T as positive semi-definite diagonal matrices

$$Q_{ij}, R_{ij}, U_{ij}, T_{ij} \begin{cases} \geq 0 & \text{if } i = j \\ = 0 & \text{if } i \neq j \end{cases} \tag{2.3}$$

will lead to a objective that is convex[17]. The three terms within the sum of (2.2a) represents three different objectives that may be desirable in a control system[53]. The first will seek to minimize the error between the states and their references, and is the typical overall goal of the controller. However the last term will pull in the direction of accomplishing this with as little change in the input as possible. In a practical application this is an important term, as rapid change in the actuators can cause wear and tear on these as well as on the system. The second term will try to get the input close to some ideal value. This can e.g. be zero, if a high cost is associated with using the input, or can be a steady-state setting it is desirable to tend towards[53]. Together this is referred to as the stage cost[27]. The last term is called terminal cost. This is to approximate the infinite-horizon tail of the cost function in order to improve the stability and performance properties of the controller[38], by punishing solutions that ends with an error.

Considering the constraint (2.1c), this is the constraint that binds the prediction model to the optimization. All states and inputs calculated must "fit" the model, and as the model starts at the current state, a minimization will lead to a input sequence $u(k), k = 0 \ldots K-1$ where a compromise of the criteria of the objective function are taken into account to take the system from its initial state $x(0)$ and to their respective references. How much each term is prioritised depends on the values set in the Q, R and U, thus the term "weight matrices". A high value on a given position of the diagonal in the matrix, will result in a high effort on keeping this variable close to its desired value.

The remaining constraints (2.2d)-(2.2f) are simply to keep the variables within acceptable/possible limits, referred to as bounds. In a real-world application, disturbances(i.e. model mismatch) can drive the plant to a state which is infeasible. This may then cause the optimization algorithm and the controller to fail[39]. Therefore (2.2c) and (2.2f) are often the only constraint that never are allowed to be violated(hard-constrained)[23]. The rest of the constraints are often extended with slack-variables which are minimized in the objective to ensure that a feasible solution always is obtainable(soft-constraints)[39]. Note also that these constraints may not only be bounds, but can be what ever linear or non-linear constraint that might be desirable.

If using a linear approximation to the plant to be controlled, this implies that the equality constraint (2.2c) is linear. By also removing the constraints (2.2d)-(2.2f) and setting $K = \infty$, this problem can be solved using the Ricatti-equation[34]. This results in a feedback-law, known as a infinite-horizon Linear quadratic requlator (LQR)[17]. Because of the simplicity and robustness of the feedback controller, this is a common solution used in the industry.

On the other hand if the constraints (2.2d)-(2.2f) are present, there exist no pure feedback solution with constant gain to the problem. Therefore the infinite-horizon problem is approximated by setting K to a finite value[21], creating a finite-horizon optimal-control problem, and applying an optimization algorithm to solve (2.2). It should however be noted that by assuming that no inequality-constraints are active after a finite time and apply LQR for the remaining infinite horizon, a solution can be found for a linear, constrained infinite horizon problem[25]. Still considering $g(x(k), u(k))$ to be linear, according to the discussion about convexity in section 2.1 the whole problem is a QP as the inequality constraints clearly are linear. This is a very common case in MPC, as the use of step and impulse-response models has been and still are frequently used in MPC[25][19], although in recent research the linear model is usually described as a state-space model[39]. Further for systems that are to operate close to a set point, and little change is expected, linear dynamic models may replicate the behaviour of the plant well. This is often the case for the process-industry, where the plant operates close to a set point and transients are mostly present in start-up and shut-down operations. The problem can then be solved using e.g. an Active-set[53] or IP algorithm which is attractive, as these algorithms are known to be fast, well proven, and a global solution can always be found in a convex QP.

In some cases it may be desirable to use a non-linear model in the constraint (2.2c), and/or use non-linear constraints on the inputs and states for reasons stated initially in the thesis. The problem is then no longer a QP, and a non-linear solver such as SQP or IP must be applied. This is the kind of problem that will be the topic for this report.

2.3 Model-predictive Control

If the model $g(x(k), u(k), x_0)$ were a perfect replication of the real plant and no unknown disturbances were present, the solution to (2.2) could be calculated off-line. When applied to the plant this would have followed the optimal trajectory perfectly. Unfortunately this is never the case, and something must be applied to correct for these unknown factors. This is exactly what MPC does, by acquiring new measurements of the states and implementing these as $x(0)$ in (2.2), and solves the new optimization problem. Then the first instance in the minimizing sequence, $u(0)$, is implemented, the rest discarded[39]. This process is repeated with a period significantly smaller than the control horizon, thus MPC gets a feedback-effect as the input clearly is dependent on the current state[37]. The technique has proven very robust, it is relatively simple to understand when the finite-horizon optimal-control technique is understood, it naturally handles MIMO systems and maybe most importantly it handles constraints in a general manner without resorting to ad-hoc solutions[19]. When it also has the ability to handle non-linear systems, it is a very attractive approach for control engineering.

Not considering the development of the prediction model $g(x(k), u(k), x_0)$, the clearly biggest challenge of the MPC approach is to solve the problem (2.2) sufficiently fast. This operation must be performed in real-time, with a period depending on the time constants of the process.

It is common to apply so-called input-blocking to the MPC problem[53]. This is a quite simple way of reducing the number of MV in the problem to be optimized. Instead of dividing the input sequence into K number of MV, it is taken into account that the input should, and will, change somewhat slowly. Thus the input can be held constant in some intervals without much change in the result. The intervals with constant amplitude are called input blocks, and one block is treated as one manipulated variable. The number of blocks to be chosen depends greatly on the system at hand, however between five to ten blocks seems a common choice[23]. Also it is reasonable to choose the blocks early in the horizon shorter than those later. This way the controller gains freedom to manoeuvre in the start of the horizon, then settling towards the end when the change in the input is being forced to be smaller.

As mentioned in section 2.2, a non-linear optimization problem arises if non-linear constraints, non-convex objective or maybe most commonly a non-linear prediction model is introduced. This is referred to as NMPC, and is today state-of-the-art within process control engineering. Two main approaches exist when defining a NMPC problem. The first is the way that the prediction model is introduced as an explicit equality constraint in (2.2). This way of defining the system in the MPC problem is called the multiple shooting approach[25]. This approach has some advantages, such as suited for parallelisation[14]. However it requires in addition to a initial guess of the optimal input sequence, a corresponding trajectory to this guess. Further the gradient calculation requires the prediction model to be continuous and differentiable at the whole feasible set. As described initially in the report, this may not always be the case, as the model may be implemented in a simulation environment such as Modelica or Simulink. This means that given a initial state $x(0)$ and a discrete series of control inputs $u(k)$, the solution to the initial-value problem from step 0 to K can be denoted

$$[x(1), \dots, x(\hat{K})] = \ell(u(0), \dots, u(K-1), x(0), K) \tag{2.4}$$

where $x(k)$ is the resulting states at time index k. Because ℓ may be a variable-step ODE-solver, $\hat{K}$ is introduced because the resulting states is likely to have different sampling interval than the input sequence. In the fixed-step case $\hat{K} = K$. This new form of the model can simply replace $x(k)$ in equation (2.2a). This will then always make sure that constraint (2.2c) is satisfied, and equation (2.2c) and the variables $x(0), \dots, x(K)$ can then be removed from the problem. This approach is called single-shooting[27].

Re-stating the complete NMPC problem to be solved, re-using the objective function (2.2a) using the single-shooting approach

$$\begin{aligned}
\min_{u \epsilon \mathbb{R}^n} \; & f(\ell(u(0), \dots, u(K-1), x(0), K), u(0), \dots, u(K-1)) && \text{(2.5a)}\\
& := f(u) && \text{(2.5b)}\\
& \text{s.t} && \\
& \quad x_{min} \leq x(k) \leq x_{max}, \quad k = 1 \dots \hat{K} && \text{(2.5c)}\\
& \quad u_{min} \leq u(k) \leq u_{max}, \quad k = 0 \dots K-1 && \text{(2.5d)}\\
& \quad \Delta u_{min} \leq \Delta u(k) \leq \Delta u_{max}, \; k = 0 \dots K-2 && \text{(2.5e)}
\end{aligned}$$

where the equality constraint (2.2c) is removed from the list of explicit constraints. The possibility of $K \neq \hat{K}$ also requires a modification of (2.2a), and typically the sum over the state trajectory is replaced with a discrete integral. The definition to $f(u)$ is to illustrate that there are only the input blocks that are MV. Note that because of input blocking, the number of MV will be less than K, however this is not considered in (2.5). This notation for the objective function will be used throughout the thesis.

Clearly although the constraints are linear, the objective function may no longer be a convex one, thus no longer a QP. Applying SQP or IP methods these will somehow need to do a linearisation of the objective function to approximate the problem with a QP, thus requiring the gradient.

Chapter 3

DFO preliminaries

This chapter is mainly a summary of the chapters concerning quadratic interpolation, regression and building quadratic models from under-determined sample sets in [11]. It will present the parts that are important to understand the behaviour of DFO, with special focus on solving a NMPC optimization problem. The objective function is denoted $f(x)$ where $x \in \mathcal{R}^n$. Most derivative-free optimization algorithms uses some sort of a linear or quadratic model to locally estimate $f(x)$ and performs the minimization on this instead. Especially the TR-methods are heavily based on this approach, which seeks to create a model, $q(x)$, on the form

$$q(x) = q(\Theta, \Phi, x) \approx f(x) \tag{3.1}$$

where Θ is the vector of unknown coefficients and determining these can be done by retrieving a set of m samples in the domain of $f(x)$, namely $Y \subset \mathcal{R}^n$, and solve a set of equations. Φ is called the polynomial basis and is a set of functions in x. When the model is built from a sample set Y, the set of equations are solved such that equation (3.1) holds exactly for the sample points, i.e.

$$q(y^i) = f(y^i) \; \forall \; i \in [1, m] \tag{3.2}$$

$q(x)$ is then said to interpolate these points. Three questions now rises: How to choose the basis Φ, where to sample, and how many samples should be retrieved from $f(x)$? To answer these questions, some basic concepts in interpolation and regression theory is needed.

Choosing a basis and a suitable set of sample points Y for use in $q(\Theta, \Phi, x)$ is important to make sure the resulting model captures the properties of $f(x)$ well within the area of interest. The main bases used in DFO will be presented, and the term "poisedness" will be a key concept in this chapter. This is a measure of how well the samples in Y covers, i.e. spans, all directions in the sampling space $\mathbb{R}^n$ to retrieve as much information of the curvature of $f(x)$ as possible[56]. This has a direct connection to how well the model $q(x)$ describes the true function $f(x)$ within the region of interest. Theories and some practical algorithms to retrieve a set Y which have such desirable qualities will be presented. A discussion of how to solve for Θ when the basis Φ and sample set Y is present and techniques for deciding $q(x)$ when there is less or more knowns than unknowns will be reviewed.

3.1 Monomial basis and the Horner Scheme

Φ from equation (3.1) is referred to as the polynomial basis. This is how the variables are combined into polynomial terms before multiplied with the vector of unknowns Θ, referred to as the regressor. The probably most common basis used in mathematics are the monomial, also called natural basis. Expanding the system of equations for a second-order interpolation problem $\Theta\Phi(y^i) = f(y^i), i \in [1, m]$

$$\theta_1(y_1^i)^2 + \theta_2 y_1^i y_2^i + \theta_3 y_1^i + \theta_4 (y_2^i)^2 + \theta_5 y_2^i + \theta_6 = f(y^i),\ i = [1, m] \tag{3.3}$$

which obviously requires $m = 6$ in the two-dimensional case to be able to determine the coefficients in Θ. This way of arranging the monomial is called the Inverse Lexical Order. This is an advantageous basis as it is easy to evaluate numerically. Consider generalizing equation (3.3) into n variables and factorizing it as

$$\begin{aligned} (y_1^i + y_2^i + y_3^i + \cdots + y_n^i + 1)y_1^i \\ +(y_2^i + y_3^i + \cdots + y_n^i + 1)y_2^i \\ +(y_3^i + \cdots + y_n^i + 1)y_3^i \\ \vdots \\ +(y_n^i + 1)y_n^i + 1 \end{aligned}$$

It is easy to see how this can be evaluated using two for-loops. In this thesis algorithm 1 is used in the optimization in chapter 7 to evaluate polynomials and retrieve the Hessian matrices and gradient vectors associated with Θ. Further, if only the function value at y is required and not the Hessian and gradient, this can be accomplished in $\mathcal{O}(n)$ by the use of the Horner scheme[7][44]. This scheme sees the inverse lexical order as a sum of multiplications. This way the algorithm also reduces the number of multiplications, which is a costly operation. Algorithm 2 shows the single variable Horner scheme for evaluating a d-dimensional polynomial. Extension to the multi-variable case is not covered

Algorithm 1 Evaluate polynomials arranged in inverse lexical order

```
1. Initialize: Given vector $\Theta \in \mathbb{R}^m$ of coefficients to be evaluated at point $y \in \mathbb{R}^n$.
2. Instantiate the matrix that will hold the final Hessian and gradient of the polynomial $B \in \mathbb{R}^{n \times n+1}$ and a counter variable $l$.
3. for $i = 1$ to $i = n$
     for $j = i$ to $i = n + 1$
       $G_{ij} = \Theta_l$
       $l = l + 1$
     end for
   end for
4. Extract Hessian: $H_{ij} = G_{ij},\ i, j \in [1, n]$
   Extract gradient: $G_i = G_{i,n+1},\ i \in [1, n]$
5. Evaluate the polynomial at $y$: $z = y^T Hy + Gy + \Theta_m$
6. Return $H$,$G$ and $z$
```

Algorithm 2 Single-variable Horner scheme

```
1. Initialization: Given $\Theta \in \mathbb{R}^d + 1$ to be evaluated at $y \in \mathbb{R}^1$. Instantiate summation variable $l = \Theta_{d+1}$.
2. for i=d+1 to 1
     $l = l + \Theta_i y$
   end for
3. return l
```

in this thesis, but can be found in [44]. Compared to algorithm 1 the Horner scheme only requires $\mathcal{O}(n)$ flops compared to $\mathcal{O}(n^2)$ for the former, and in a real-time implementation the Horner scheme should be used when applicable. The system of equations to be solved when interpolating becomes

$$q(\Theta, \Phi(Y)) = \Theta^T \Phi(Y) = \begin{bmatrix} f(y^1) \\ f(y^2) \\ \vdots \\ f(y^m) \end{bmatrix} \tag{3.4}$$

and the interpolation model $q(x)$ can be found. This model is built on the monomial basis, and solving such a system can be done by inverting $\Phi(Y)$. However this method is both computational costly and sensitive for the condition of $\Phi(Y)$. Therefore other techniques are more common when performing fully determined interpolation numerically, and Lagrange and Newton polynomials are often encountered.

3.2 Lagrange polynomial

A commonly used basis in DFO are Lagrange polynomials, which forms an interpolation basis. The definition of Lagrange Polynomials are given in **Definition 3.3** in [12]:

Given a set of interpolation points $Y = \{y^1, y^2 \dots y^m\}$, a basis of m polynomials $\ell_j(x)$, j=1,...,m, in $\mathcal{P}_n^d$ is called a basis of Lagrange polynomials if

$$\ell_j(y^i) = \delta_{ij} = \left\{ \begin{array}{ll} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{array} \right.$$

Here $\mathcal{P}_n^d$ denotes that the degree of the polynomial basis is d and the number of free variables is n. In other words the Lagrangian polynomial is equal to zero for all points in the sample set Y except the point used to create the polynomial where it is equal to one. Note how these polynomials are created purely out of the position of the sample points, with no connection to the actual value of the objective function $f(y^i)$. Combining equation (3.2) and definition 3.3, it is easy to verify that the interpolation model $q(x)$ can be found from [8]

$$q(x) = \sum_{i=1}^{m} f(y^i)\ell_i(x) \tag{3.5}$$

A common formula in mathematical literature for computing the Lagrange polynomial belonging to sample y^i in the one-dimensional case is

$$\ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{m} \frac{(x - y^j)}{(y^i - y^j)} \tag{3.6}$$

Given that no two sample points are the same(i.e. poised). Algorithm 3, which is algorithm 6.1 in [12], is for computing the Lagrange polynomials for the multivariate case, which is the algorithm applied in chapter 7. It is quite intuitive, using a pivot-like technique to begin with the polynomial-point pair having the biggest absolute value, normalizing this to one and remove the component of all other pairs. The implementation used in chapter 7 uses a random set of numbers to produce the initial polynomial instead of the monomial basis, and uses a minimum value of 10 times the floating-point precision of *Matlab* as the minimum allowed limit of $|\ell_i(y^{j_i})|$ to avoid degeneracy.

Algorithm 3 Compute Lagrange basis

```
1. Initialization: Given poised sample set $Y = \{y^1, y^2 \dots y^m \in \mathbb{R}^n\}$ and an
   estimate of the Lagrange basis $\ell_i(x), i \in [1, m]$ e.g. the monomial basis.

2. for i = 1 to m+1
   Choose $j_i = \underset{i \leq j \leq m}{argmax} |\ell_i(y^j)|$.
   If $\ell_i(y^{j_i}) = 0$ then stop, $Y$ is not poised.
   Else swap point $y^{j_i}$ and $y^i$ in $Y$.
   End if
   Normalize $\ell_i(x)$ to one for $y^i$:
   $\ell_i(x) \leftarrow \frac{\ell_i(x)}{\ell_i(y^i)}$
   Make $\ell_i$ equal to zero for all other points:
   for j = 1 to m+1, $j \neq i$
   $\ell_j(x) \leftarrow \ell_j(x) - \ell_j(y^i)\ell_i(x)$
   End for
   End for

3. Return the Lagrange basis $\mathcal{L}(x) = \{\ell_1(x), \ell_2(x), \dots, \ell_m(x)\}$
```

Lagrange polynomials are frequently used within DFO algorithms to ensure good quality of $q(x)$, however as algorithm 3 shows, all points must be available before the polynomials can be computed. This makes the procedure of creating Lagrange polynomials computationally expensive. Therefore a closely related polynomial also seen frequently in DFO, is the Newton Polynomial.

3.3 Newton polynomial

Also called Newton Fundamental Polynomials (NFP), Newton Interpolation Polynomials and Newtons Divided Difference Method results in a polynomial basis as the Monomial basis and the Lagrange polynomial. However NFP has a numerical advantage as a given NFP, η_j say, can be computed only knowing the sample points $y^i, i \in [1, j]$, in contrast to Lagrange polynomials were all sample points must be known for all polynomials[7] before starting the computation. This property encourages use of parallelisation while building the interpolation model.
In the one-dimensional case, given a sample set containing three samples $Y \in \mathbb{R}^3$, the NFP for a quadratic interpolation is given by

$$q(x) = \sum_{i=1}^{3} \alpha_i \eta_i \tag{3.7}$$

where

$$\begin{aligned} \eta_1 &= \quad 1 \\ \eta_2 &= \quad (x - y^1) \\ \eta_3 &= \quad (x - y^1)(x - y^2) \end{aligned} \tag{3.8}$$

Now using the requirement that $q(y^1) = f(y^1)$, $q(y^2) = f(y^2)$ and $q(y^3) = f(y^3)$, this gives an explicit equation to solve the interpolation polynomial

$$\begin{aligned} \alpha_1 &= \quad f(y^1) \\ \alpha_2 &= \quad \tfrac{(f(y^2)-f(y^1))}{y^2-y^1} \\ \alpha_3 &= \quad \tfrac{1}{y^3-y^1}\left(\tfrac{f(y^3)-f(y^2)}{y^3-y^2} - \tfrac{f(y^2)-f(y^1)}{y^2-y^1}\right) \end{aligned} \tag{3.9}$$

Here α_i is called the i'th order divided difference. This may resemble finite-differences, however this requires the sample points to lie relatively close. Note the similarity of η_i to ℓ_i. Inserting y^i into η_j will always make the polynomial zero. The multi-variate case of NFP will not be covered here, but can be found in [20].

3.4 Poisedness and quality of the model

Now it is known how to perform an interpolation to retrieve a model $q(x)$ of the objective function $f(x)$. However the question of where to sample $f(x)$ to create the sample set Y is not discussed. This is in fact a key element especially in DFTRM, as the position of the points relative to each other determines how well the model $q(x)$ resembles the objective function $f(x)$. Consider solving the interpolation problem (3.4) simply by inverting $\Phi(Y)$

$$\theta = \Phi(\bar{Y})^{-1} \begin{pmatrix} f(Y_0) \\ f(Y_1) \\ \vdots \\ f(Y_m) \end{pmatrix} \tag{3.10}$$

This will require Φ to be square and non-singular, and for certain choices of Y this may not be the case[12]. This means that one or more of the degrees of freedom in $f(x)$ is not captured by Φ[29]. If this is the case, Y is singular, thus said to be non-poised for interpolation. A formal definition of a poised set is given in
Lemma 3.2 in [12]:

Given a function $f : \mathbb{R}^n \to \mathbb{R}$ and a poised set $Y \in \mathbb{R}^n$, the interpolating polynomials exist and is unique.

Further, if $\Phi(Y)$ is close to singular with the current sample set, the resulting model may not replicate the approximated model adequately, and a small error in the objective function may induce a significant error in the model. In a practical application it would be useful to be able to measure how close $\Phi(Y)$ is to become singular, and thereby be able to ensure a certain quality and robustness of the resulting model[12]. This is called well-poisedness, and is commonly measured in a scalar Λ, thus referred to as Λ-poisedness. This is an useful tool in DFO, as it makes it possible to ensure that the sample set is a certain distance from making $\Phi(Y)$ non-invertible[12]. Note also that if a sample set Y is poised for a certain polynomial basis Φ, then it also poised for all other bases. In fact, it does not matter which basis is used to interpolate, as the resulting polynomial will be the same if the sample set is poised[12].
The Λ-poisedness measure of well-poisedness is defined in
Definition 3.6 in [12]:

Let $\Lambda > 0$ and a set $B \in \mathbb{R}^n$ be given. Let $\Phi = \phi_1(x), \phi_2(x), ..., \phi_m(x)$ be a basis in $\mathcal{P}_n^d$. A poised set $Y = y^1, y^2, ..., y^m$ is said to be Λ-poised in B (in the interpolation sense) if and only if
1. for the basis of Lagrange polynomials associated with Y

$$\max_{1 \le i \le m} \max_{x \in B} |\ell_i(x)| \le \Lambda \tag{3.11}$$

or, equivalently,

2. for any $x \in B$ there exist $\lambda(x) \in \mathbb{R}^m$ such that

$$\sum_{i=1}^{m} \lambda_i(x) \phi(y^i) = \phi(x) \tag{3.12}$$

with

$$\|\lambda(x)\|_\infty \le \Lambda \tag{3.13}$$

or, equivalently,

3. replacing any point in Y by any $x \in B$ can increase the volume of the set $\{\phi(y^1), \phi(y^2), ..., \phi(y^m)\}$ at most by a factor Λ

Here B is the set that $f(x)$ is to be approximated within. When introducing DFO algorithms, this can be seen as the set(Trust region) where the approximation $q(x)$ is considered to be a valid approximation to $f(x)$. As 3.11 of definition 3.6 shows, a upper limit of the Λ-poisedness can be measured by finding the Lagrange polynomial with the biggest absolute value in B. Inspecting (3.6), this makes sense as the coefficients of the polynomials are found from dividing on the difference between the sample points. If the points lie close, thus not well poised, the coefficients will become large. Further more if the points y^i and y^j, $i \neq j$ lies close, the coefficients of these two corresponding Lagrange-polynomials will be large to be able to change between one and zero on a small change in the variables.
3.12 of the definition gives a intuitive understanding of what Λ-poisedness is. y^i is sampling point i, while x represents points on the set B and ϕ is the monomial basis. The smaller the scalar Λ is, the better poised the sample set is. If Λ is equal to one, then the sample set spans $\mathbb{R}^n$ perfectly within B. The last statement of the definition is a geometric interpretation of 3.12[12].

Some important properties of Λ poisedness which will be used to obtain well-poised sample-sets is stated in
Lemma 3.7[12]:

1. If B contains a point in Y and Y is Λ-poised in B, then $\Lambda \geq 1$
2. If Y is Λ-poised in a given set B, then it is Λ-poised (with the same constant) in any subset of B
3. If Y is Λ-poised in B for a given constant Λ, then it is $\tilde{\Lambda}$-poised in B for any $\tilde{\Lambda} \geq \Lambda$

Where the fourth point is omitted as the purpose here is a working understanding of the poisedness concept. Especially point 2 is important, as it will be used in DFTRM. The poisedness will never decrease by shrinking the set B[12]. This is quite intuitive when working with approximations, as shrinking the region that $q(x)$ is considered to be a good approximation within, will not make $q(x)$ a less accurate approximation. Actually, as will be used in chapter 4, a decrease of B can, and usually will, make the approximation better.

3.5 Obtaining a well-poised sample set

When applying interpolation in an optimization algorithm, the sample set will continuously change and therefore the poisedness of the set has to be checked and, if necessary, improved in some way. This can either be done by finding the point that contributes the worst to the Λ-poisedness and find a good substitute for this, or remove a set of points(possibly all) and find new points that gives a well-poised set. Before proceeding to the algorithms that ensures sampling sets that are sufficiently well-poised, a class of models called *Fully Quadratic Models* is presented. Models belonging to this class meets certain requirements that will be interesting when looking into algorithms that creates poised sample sets. [12] gives the definition of this class in
Definition 6.2 in [12]:

1. If a model $q(x)$ belongs to the class of fully quadratic models, then $\|q(x) - f(x)\|$, $\|\nabla q(x) - \nabla f(x)\|$ and $\|\nabla^2 q(x) - \nabla^2 f(x)\|$ is bounded from above by finite constants on a set B, where ∇ denotes the gradient with respect to x.
2. There exist an algorithm that in a finite, bounded number of steps can either verify that a model is fully quadratic or create a quadratic model that is fully quadratic on a set.

A quadratic model built from a Λ-poised set, will always belong to the class of fully quadratic models[12], and the constants bounding the model error in the first point of Definition 6.2 will make it possible to ensure a certain quality of the model. The second point states that it is possible to create such a model in a finite number of iterations. The relationship between poisedness and Fully Quadratic can be understood intuitively from equation (3.12). If $\Lambda \leq \infty$ then $f(x)$ and $q(x)$ have certain similarities on the set that $q(x)$ is trusted to be a good approximation, because the difference between the two are bounded by a scalar. The bound on the first and second order gradients then follows from this, given that the gradient is defined for $f(x)$ within the set B.

Algorithm 4 uses Lagrange polynomials to improve the poisedness of the sample set(this is Algorithm 6.3 in [12]). Each iteration will simply find the Lagrangian polynomial that gives the most contribution to Λ, following the definition of Λ-poisedness of equation (3.11). Because each polynomial is created with a certain sample point, the sample point belonging to this polynomial is the point that increases Λ the most. If the value of Λ to this point is within the tolerance, the sample set is considered well-poised and the Lagrange polynomials are updated. If not, a point which do contribute to bad poisedness is found. This does not have to be the same point as found in step 2 in the algorithm, however if it is the algorithm can be used to increasingly improve the sample set one sample at a time(by simply setting the Λ-threshold very low)[12]. This point is then exchanged with the point that gave the "bad" Lagrange polynomial, and the Lagrange polynomials is updated with the new sample set.

The most difficult operation in algorithm 4 is the maximization of the Lagrange polynomials, as the absolute value can make the surface non-differentiable and several local maximum points may exist. This requires either some global optimization procedure, or the solution can be approximated.

Algorithm 4 is computationally expensive, and a more effective version using LU-factorization is presented in [12] as algorithm 6.5 , however the Lagrange-version is here presented as it gives the most direct understanding of the relation towards Λ-poisedness.

In chapter 7 the approach of completing the sample set is applied, and algorithm 5 is used. This is algorithm 6.4 from [12], and uses a pivot basis to replace any points that makes the set non-poised. Further, if the given sample set is smaller than the desired size(incomplete), it will generate new points, that gives good poisedness, until the set is completed. This algorithm uses a pivotal basis instead of the Lagrange basis, and builds this with Gaussian elimination. The

Algorithm 4 Improving the poisedness of a sample set using Lagrange polynomials

1. Choose $\Lambda_{max} \geq 1$ which will be the maximum acceptable value of the Λ-poisedness. Given a sample set Y with m points, compute the Lagrange polynomials associated with Y if not already done.

2. Find the Lagrange polynomial which gives the largest contribution to the poisedness by estimating

   $$\Lambda = \max_{0 \geq i \geq p} \max_{x \in B} |\ell_i(x)| \tag{3.14}$$

   where B is the area (set) where the model is trusted to be a good enough approximation to $f(x)$(see Fully Quadratic Model).

3. **If** $\Lambda \leq \Lambda_{max}$
   Y is Λ-poised, exit algorithm.
   Else
   let i be an index such that

   $$\max_{x \in B} |\ell_i| > \Lambda \tag{3.15}$$

   and let $y_*^i \in B$ be a point that minimizes $|\ell_i|$ in B. Update Y by replacing the worst point y^i with y_*^i.
   End if

4. Update all Lagrange polynomials.

5. Go to point 2

implementation used in chapter 7 applies the same technique for approximating the maximum of the Lagrange polynomials as the UOBYQA and Condor optimization algorithms, which are described in chapter 4.5.

Algorithm 5 Completing a non-poised sample set using LU-factorization

```
1. Given a sample set $Y$ with $d$ points, compute an approximation to the
   pivot polynomials $u_i, i \in [1, m_{ini}]$ associated with $Y$, e.g. the monomial
   basis. Choose the desired size of the sample set $m$.

2. for $i = 1$ to $m$
   Choose $j_i = \underset{i \leq j \leq d}{argmax} |u_i(y^j)|$.
   Choose the pivotal element:
   If $|u_i(y^{j_i})| > 0$ and $i < m_{ini}$
   Swap points $y^i$ and $y^{j_i}$.
   Else
   Make point well-poised:
   Compute or estimate $y^i = \underset{x \in B}{max} |u_i(x)|$
   End If
   For $j = i + 1$ to $m$
   Gaussian elimination:
   $u_j(x) = u_j(x) - \frac{u_j(y^i)}{u_i(y^i)} u_i(x)$
   End For
   End For

3. Return poised set $Y$
```

3.6 Over-determined models

Until now the matrix Φ of the interpolation model has been square. This gives a determined set of equations, which is then solved exactly. However if the number of sampling points increases, Φ will get more rows than columns and the set of equations will become over-determined. If the function to be approximated is an analytical function without noise and the sample set is poised, these equations can be solved simply by neglecting some of the equations such that Φ becomes square. However, when noise is present, the set will not necessarily have an exact solution as in the interpolation case. In this case the set of equations can be solved in a least-squares manner, called regression, by minimizing the approximation error[12].

$$\min_{\alpha} \|\Phi(\mathbf{Y})\theta - f(Y)\|^2 \tag{3.16}$$

Equation (3.16) has an unique minimum if $\Phi(Y)$ has full column rank[12], which is obtained if the sample set Y is poised. Regression is also quite similar to interpolation when looking at bases and poisedness. E.g. the Lagrange polynomials are given as in interpolation, however now solved in a least-square fashion. From **Definition 4.4** in [12]

$$\ell_j(y^i) \overset{l.s.}{=} \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

Further, the Λ-poisedness is still measured as the maximum absolute value of the Lagrange polynomial on the set B[12]. Because of the bigger sample set and the least-square approach, regression will have a smoothing-effect on $f(x)$, thus suppressing noise. On the other hand, to have any noticeable effect of this smoothing effect in all degrees of freedom, the sample set will probably need to be quite much bigger than in the interpolation case. This will require a significant increase in the number of function evaluations, which is unattractive as this is an expensive operation in NMPC. To be able to justify such an approach, the improvement in accuracy of $q(x)$ and the convergence rate of the optimization algorithm must increase accordingly.

3.7 Under-determined models

The last type of model that will be mentioned is under-determined models. This occurs when the number of sample points is too small for a fully determined interpolation to be possible. The matrix Φ in equation (3.1) gets less rows than columns. To solve such a system, the non-determined Degrees of freedom (DOF) must be filled in some way. One possible procedure can be to remove columns from $\Phi(Y)$ such that the matrix becomes square[12]. This will in essence be to neglect some degrees of freedom in the interpolation. However, which columns to be removed is not trivial. If the "wrong" column is removed, then $\Phi(Y)$ may become singular.[12] As seen in section 3.5, singularity of $\Phi(Y)$ and the poisedness of Y has a direct connection. It is then necessary to choose between several possible models, corresponding to which columns are removed from Φ. The goal then essentially becomes to choose a model, which is a subset of Φ, that Y is well-poised in. This can be done through a minimum-norm solution. To find the unknown parameters Θ in the model (3.1), minimum norm may be applied as defined in [12]

$$\Theta^{mn} = \Phi(Y)^T(\Phi(Y)\Phi(Y))^T f(Y) \tag{3.17}$$

As in regression, (3.17), the Lagrange polynomials are given as in interpolation, however now solved using the minimum norm. From [12]:

$$\Phi(Y)\theta_j \overset{m.n.}{=} e_j,\ j = 1 \dots m$$

Where m.n. stands for minimum norm, and e_j is the j'th column of the identity matrix of size m. The interpolation polynomial is given as in equation (3.5). As there will be fewer Lagrange polynomials than columns in Φ, the Lagrange polynomials will not be a basis. However, as in interpolation and regression, if the Lagrange polynomials exist and is unique, the sample set Y is poised[12].
As a matter of fact, the Minimum Norm Solution is mainly used for linear under-determined systems. However because of the nature of the MPC problem, it is desirable to approximate quadratic functions. The most common way to create quadratic models from under-determined sample sets are through the Minimum Frobenius norm. This is somewhat related to the Minimum norm, but splits the problem into a linear and a quadratic term, then solves this as an optimization problem.[11] This method has been successfully implemented in several algorithms, e.g. NEWUOA[50] and BOBYQA[52]. In these algorithms undetermined space of the Hessian is absorbed by minimizing the Frobenius norm of the Hessian matrix of the previous iteration and the current, subject to

that the Hessian must be symmetric and to the interpolation condition (3.2). This is referred to as "Minimum change-updating" in [40].
Typical for under-determined models are the difficulty of determining a bound on the error of the model, as described in definition 6.2 in [12]. How well an under-determined implementation will work on the MPC problem is probably most easily verified by experiments, but the advantages of requiring a low number of function evaluations at each iteration point compared to fully determined methods is encouraging. However these advantages may be compromised if the quality of the model becomes bad, and the algorithms gets a slow convergence rate.

Chapter 4

DFO algorithms

This chapter will present how the theory of chapter 3 is applied in DFO algorithms. The focus will be on DFTRM, however also the GA is presented as it is experimented with using this to improve the robustness of the NMPC. Lastly some well-known and proven algorithms are presented in detail, to give basis for evaluating the results to come in chapter 6.

DFO algorithms comes in a vast number of types and varieties. The maybe simplest that can be though of is the "brute-force" method, which simply tries a large number of random solutions and chooses the best one. This however quickly becomes impractical when the number of MV, n, grows, and more sophisticated methods have been developed.
A common characteristic of these methods are that they start with an initial set of m samples, $Y \subset \mathbb{R}^n$, of the objective function $f(x) : \mathbb{R}^n \mapsto \mathbb{R}^1$, and creates new points by trying to figure out where the optimum of $f(x)$ is by looking at the positions of $y^i, i \in [1, m]$ and the resulting functions values $f(y^i), i \in [1, m]$. Several DFO methods are discussed in [12] and [40], and some of these are presented and tested in [13]. However in this thesis only methods that proved promising during initial testing on the more complex problem yet to be presented are investigated, these being DFTRM which uses quadratic interpolation models. In addition the GA is presented, which proved to be very robust for finding a feasible starting point on numerically challenging NMPC problems.
In [13] the linear DFTRM Cobyla[28] gave very promising results as it used remarkably few iterations. However as suspected this algorithm runs into problems when the number of MV increases as the complexity of the problem grew, usually not being able to come up with a reasonable solution at all.

4.1 Genetic algorithm

GA is inspired from the natural selection of the strongest individuals in nature. The method uses the terminology chromosomes and genes for describing a possible solution and the individual MV for the given chromosome respectively. Thus there exist m chromosomes, each built up of n genes. Further population refers to the current set of chromosomes at the current iteration[12].
For each iteration the cost of each chromosome is evaluated and the current population is sorted. A predetermined survival percentage is used to remove the chromosomes with the highest cost. New chromosomes are made from combinations of the surviving ones, called mating. Several ways of performing this combination have been proposed, e.g. interpolation, extrapolation or some heuristic crossover[24]. This promotes the best solutions, and makes the algorithm converge towards an optimal solution.
A percentage of the genes are exchanged with a random number, called mutation. This randomness gives the exploration effect that the GA is mostly known for, and the GA has proved to be effective when searching for a global optimum in a non-convex problem, even if several similar solutions exists and the cost landscape is very complex[24].

The GA used in this thesis is based on the algorithm presented in [24], however it uses a modified method of combining chromosomes which proved effective by experiments. This method repeats the following until all the discarded chromosomes are replaced: Choose two surviving chromosomes c_i and c_j, where i starting at 1 is increased for each repetition and restarted if $i > n_{sel}$, while c_j is a random surviving chromosome. c_i and c_j is exchanged if $f(c_j) < f(c_i)$. Then a random number p between 0.5 and 1.5 is chosen, and the new chromosome is $c_{new} = pc_i + (1-p)c_j$.
This method is inspired from the observation that the objective function in a NMPC optimization problem is at least usually locally convex around the optimum solution. Then the line from c_j to c_i is drawn. The optimal solution may as well lie in between these two as along the extended line beyond c_i, however never in a negative direction along this line from c_j. A choice between the midpoint between these points and 0.5 beyond c_i in positive direction therefore seems reasonable, and experiments proved these values to give good results. The algorithm used is stated in algorithm 6.

The algorithm uses so-called elitism, as it never changes the best chromosome. This is an important feature in GA, as the mutation can actually change the best solution to a worse one if this issue is not specially handled.

Algorithm 6 Genetic algorithm

1. Given mutation rate $r_{mut} \in [0,1]$, selection rate $r_{sel} \in [0,1]$, $b_l, b_u \in \mathbb{R}^n$, the desired size of the population n_{pop} and maximum number of iterations k_{max}. If not supplied, generate a random initial population p_0.

2. Calculate how many chromosomes which will survive between iterations $n_{sel} = r_{sel} n_{pop}$ and how many will be discarded $n_{disc} = n_{sel} - n_{pop}$. Also find the total number of genes to be mutated $n_{mut} = r_{mut} n_{gen} n_{pop}$. Set $k = 1$

3. Evaluate the cost of each chromosome, and sort the population in increasing order.

4. If $k > k_{max}$ then return with solution set to the first chromosome in the list.

5. Replace the n_{disc} chromosomes at the bottom om the list by combining the n_{sel} chromosomes from the top, using the method described above.

6. For all genes in the population $g_i, i \in [1, n_{pop} n]$ set $g_i = b_u$ if $g_i > b_u$ or $g_i = b_l$ if $g_i < b_l$.

7. Choose n_{mut} random genes from the population which is not in the best chromosome. Replace each of these with random number between b_l and b_u.

8. Go to step 3

4.2 Trust-region methods

TR methods are based on making a model $q(x)$ of the true objective function $f(x)$ around the current best solution x_k, called the current iterate, k being the iteration index. The minimization is performed on this model instead of the true objective function.

Both derivative and derivative-free variants exist, common for both types are that they rely on Lemma 3.7 from [11], also described in chapter 3. The Lemma is applied through the trust-region radius Δ, which is a scalar defining a hypersphere around x_k. The optimization of $q(x)$, which gives the Trust-region step s^{TR}, is performed under the constraint that the solution must be within this TR. A typical trust-region sub problem is

$$\min_{s \epsilon \mathbb{R}^n} q(x_k + s) \tag{4.1a}$$

$$\text{s.t}$$

$$\|s\| \leq \Delta \tag{4.1b}$$

Where s defines the step length from the current iterate. This is common in TR-algorithms, and it is appropriate to define a new sample set D that is Y shifted around x_k and is denoted $D = \{y^1 - x_k, y^2 - x_k \dots y^m - x_k\}$, denoting element i in D as s^i.

If the original problem were an unconstrained minimization of $f(x)$, problem (4.1) may actually be harder to solve than the original one[7]. However typical for applications of DFO and especially single-shooting NMPC problems is that $f(x)$ is non-quadratic and non-convex, and evaluations of it is very expensive and contaminated with numerical noise. Also the function may not be differentiable, or the derivatives may come at a high cost. By making e.g. the quadratic approximation $q(x_k + s) = s^T Hs + Gs + f(x_k) \approx f(x_k + s)$, this model can evaluated repeatedly at a low cost and both the gradient and Hessian are easily available. Thus a variety of techniques exist for solving (4.1) to retrieve the trust-region step s^{TR}.

Now the model predicts a reduction, $pred = q(x_k + s^{TR}) - q(x_k)$ and the actual reduction $ared = f(x_k + s^{TR}) - f(x_k)$ is computed and compared with $pred$. What is done next separates the different algorithms, however it is common that if the difference between the two is large a new TR step is computed with a smaller "local" TR radius[7]. This is the point where Lemma 3.7 comes into place, as a smaller step will(hopefully) not lead to such a big model error.

Clearly the main concern is to actually retrieve $q(x)$, and make sure it is of adequate similarity to $f(x)$. In DFTRM this is done without calculating or evaluating any derivatives of $f(x)$. $q(x)$ is constructed by the interpolation techniques described in chapter 3. This is the reason why "poisedness" play such an important role in DFTRM, as the position of the points in Y is essential to the accuracy of $q(x)$ with respect to $f(x)$. Therefore it is usual for DFTRM to solve a second sub problem, which is the maximization of the polynomial basis seen in chapter 3, which purpose is to check the poisedness of the sample set after the solution from the TR sub problem is included in Y. If the poisedness is still adequate, nothing needs to be done. Else several different approaches exist, however the most intuitive may be to run a geometry improving algorithm such as algorithm 4.

Note that it is usual for a TR algorithm to use two separate trust region radii, one global Δ and one local ρ[7]. The global is the radius that $q(x)$ is currently trusted within. The second is used "locally" in the algorithm. If the difference between *pred* and *ared* is to large, ρ is reduced and the sub problem (4.1) is solved again. Δ is on the other hand the hypersphere that the second sub problem is solved within, thus new poisedness-improving sample points will typically lie on this sphere.

Algorithm 7 describes a simplified DFTRM. Although several aspects is left out, it incorporates the most important steps. Especially important is step 6. If the calculated TR-step is small, this indicates that the algorithm is getting close to a solution. It is therefore necessary to reduce the spread of the samples in Y by reducing Δ, making it increasingly more accurate around the exact solution of $f(x)$. Simplified it can be said that the local TR-radius ρ is used to correct for model mismatch, ensuring that the steps is small enough that the model is sufficiently accurate. The global TR makes the algorithm converge towards the solution.
Common stopping criteria is minimum global TR-radius Δ_{end}, maximum number of function evaluations, and/or algorithm iterations.

For simplicity of notation only x_k is denoted with iteration index to identify it from the general vector x. For other variables changing from an iteration to the next this should be clear from the context. Further it should be noted that BOBYQA, UOBYQA and Condor uses two iterates, x_k and x_{base}. x_k holds the current best point while x_{base} is set equal to x_k only when the Δ is updated. The sample set is always shifted to lie around x_{base} in these algorithms, as it makes the interpolation more accurate[7] due to less numerical round-off

Algorithm 7 Simplified general trust-region algorithm

1. Initialization: Given a poised sample set Y_0, a feasible starting point x_0 and an initial trust-region radius Δ_0.

2. Model building: Construct $q(x)$ from the interpolation condition $q(y^i) = f(y^i), i \in [1, m]$.

3. TR-sub problem: Find the solution $x^+ = x_k + s^{TR}$ to TR-sub problem, and calculate model agreement $r = \frac{ared}{pred}$.

4. If r is adequate, replace a leaving point y^l with x^+ in Y and continue. Else reduce ρ and go to step 3.

5. Second sub-problem: Check the poisedness of Y.
If adequate continue.
Else initiate poisedness-improvement mechanism.

6. Model correctness check: If the step length $\|s^{TR}\|$ is below a threshold, reduce Δ for the next iteration such that the model starts to be focused around a smaller area thus improving correctness.

7. Stopping criteria: If allowed, go to step 2.

errors[52]. x_{base} is not considered in this report, as it is only important for practical implementation and not for a general understanding of the algorithms.
To ensure that the TR restricts all DOF the same, the MV of the problem are scaled within the algorithm. This is an important concept as the order of magnitude of the MV can be vary largely. Although important for the success of the algorithm, this aspect is left out of the descriptions in this report. However this scaling is often to be pre-supplied by the user, or the algorithm takes basis in the constraints provided to decide a typical span of a MV.

4.3 Constraint handling

A common limitation in DFO is often the lack of constraint handling. In Derivative-based optimization (DBO), constraints is usually handled through Lagrange multipliers. This gives the gradient of the constraints, and gives these algorithms the ability to "slide" along a constraint in a direction that reduces $f(x)$[7]. For DFO these multipliers are never available directly, because of the need for differentiation. Some TR algorithm handles constraints by applying them in the TR-sub problem[52][6]. This will work well for explicit constraints, however implicit constraints which requires evaluation of the true objective to be decided if violated gives problems. The only two general approaches found by the author is the use of penalty-functions and hidden constraints[30]. The latter is the only one that will guarantee that the constraint is never violated, as an error is returned from the objective function if a solution is not feasible. The algorithm will then simply "retreat" the solution a little against a known feasible solution(preferably the iterate) until no error is returned. This technique is mainly meant as an "emergency" mechanism, if the objective function of some reason is not defined at certain points. Most developers of DFTRM recommends to avoid using this features for constraint handling, and rather rely on using an penalty function[30].

4.4 Wedge

The Wedge algorithm is unique by introducing a constraint when solving the sub-problem that ensures that the new point can be added to the sample set without compromising the poisedness of the set. The algorithm has both a linear and a quadratic version, and the versions are similar enough that both will be presented here as the linear one gives an intuitive understanding of the poisedness-ensuring mechanism of the algorithm. This section is mainly based on [36], with notation to fit the rest of this thesis.

As model-improving steps requires evaluations of the objective and does not directly contribute to a better point, such steps can be considered very costly[56]. To guarantee that the TR-step fulfils this condition, an additional constraint is introduced in the sub problem. The constraint is designed such that the new point has a certain distance from making the sample set degenerate, which can be recognized as a poisedness condition. The set where the constraint is active will be defined later and is denoted $\mathcal{W}$. This will change as Y changes, thus usually each iteration. γ is a constant deciding the size of $\mathcal{W}$, the bigger γ the more well-poised the sample set will be kept. However if the solution to the sub problem lies inside $\mathcal{W}$, a compromise must be made of how much the value of the objective is allowed to increase by moving the solution outside this region. Therefore the algorithm incorporates an update-mechanism for γ, reducing it if the function value increases by more than a factor called $fracOptRed$, in this thesis denoted μ.
The Wedge sub-problem to be solved is

$$\min_{s\epsilon\mathbb{R}^n} q(s+x_k) = s^T Hs + Gs + f(x_k) \tag{4.2a}$$

s.t

$$\|s\| \leq \Delta_k \tag{4.2b}$$

$$s \notin \mathcal{W}_k \tag{4.2c}$$

When the algorithm has solved the sub problem and found the TR-step s^{TR}, this forms the candidate to a new iterate $x_k^+ = x_k + s^{TR}$. Whether it indeed makes this replacement, enters the sample set or is rejected depends on the model validation step. In the two former cases it will replace a point y^l, which is the point that currently has the longest distance to x_k.
The generation of the initial sample set is the same as used by the BOBYQA-algorithm which will be presented in section 4.6. BOBYQA chooses the side of

x_k that the samples in Y lies on from the distance to the bounds, while Wedge makes this choice at random. For further details about the technique the reader is referred to [36].

Algorithm 8 shows how simple and straight-forward the DFTRM becomes when using the Wedge-constraint. However this comes at a cost, as solving the sub-problem clearly becomes more of a challenge. How the Wedge-constraint is formed and solved remains to be explained, as well as the model-building step.

The algorithm has three optional rules for changing the trust region. The one explained in algorithm 8 is the default, and takes into account that there is only a need to expand the trust region if s^{TR} actually is constrained by it, as an expansion can lead to a less accurate model.

The algorithm uses QR-factorization to build linear models, and this procedure is quite straight-forward. However the quadratic model is built in an interesting way. The model (4.2a) is factorized as

$$q(s + x_k) = f(x_k) + Gs + \sum_{i<j} H_{ij} s_i s_j + \frac{1}{2} + \sum_i H_{ii} s_i^2 \tag{4.3}$$

$$= f(x_k) + \hat{g}^T \hat{s} \tag{4.4}$$

where

$$\hat{g} = \begin{bmatrix} G^T & \{H_{ij}\}, i < j & \{\frac{1}{\sqrt{2}} H_{ii}\} \end{bmatrix}^T \tag{4.5}$$

$$\hat{s} = \begin{bmatrix} s^T & \{s_i s_j\}, i < j & \{\frac{1}{\sqrt{2}} s_i^2\} \end{bmatrix}^T \tag{4.6}$$

Then LU-factorization is applied to solve the system

$$\hat{g}^T \begin{bmatrix} \hat{s}^1 & \hat{s}^2 & \dots & \hat{s}^m \end{bmatrix} = \begin{bmatrix} f(y^1) & f(y^2) & f(y^m) \end{bmatrix}^T - f(x_k) \tag{4.7}$$

Where $\hat{s}^i$ denotes (4.6) inserted point i from D.

4.4.1 Defining and solving the Wedge sub-problem

For equation 4.7 to be solvable, $\begin{bmatrix} \hat{s}^1 & \hat{s}^2 & \dots & \hat{s}^m \end{bmatrix}$ must be non-singular when the leaving point y^l is replaced with the trial-point s^{TR}. This means that this point must not lie in the $(n-1)$-dimensional subspace spanned by $D \setminus s^l$[36]. The set of points that will make D degenerate is called the taboo-region, and is described using the null-space vector B of the set D, which is an orthogonal

Algorithm 8 The Wedge algorithm

```
1. Initialization: Create a initial sample set $Y$, choose constants for TR-update $\gamma_1$, $\gamma_2$, objective increase-tolerance $\mu$, initial value for TR radius $\Delta_0$ and initial angle on the Wedge constraint $\gamma_0$. Choose starting point for optimization $x_0$.

2. Choose the sample point $y^{out}$ that is farthest from the current iterate $x_k$, by finding $\underset{i \in \{1,2..m\}}{max} \|y^i - x_k\|$

3. Create the model $q(s + x_k) = s^T Hs + Gs + f(x_k)$ from the sample set $Y \cup x_k$ using equation (4.7), and define the wedge $\mathcal{W}_k$ by use of technique in section 4.4.1.

4. Solve the sub problem 4.2 to compute the step $s_k^{TR}$ from the current iterate $x_k$ to the new candidate point using technique described in section 4.4.1 This step also updated the angle of the Wedge-constraint $\gamma$.

5. Retrieve the actual reduction of the objective $ared = f(x_k) - f(x_k + s_k)$ and the predicted reduction $pred = q(x_k) - q(x_k + s_k)$, and calculate $r = \frac{ared}{pred}$.

6. Trust-region update:
   IF$(ared \leq 0)$
   THEN $\Delta^+ = \gamma_1 \|s_k\|$
   ELSE IF $(ared > 0$ AND $\|s_k\| \leq \Delta)$
   THEN$\Delta^+ = \gamma_2 \Delta$ Expand the trust-region only if the trial point is constrained by it.
   ENDIF

7. Update the sample set Y:
   IF ared $< 0$
   THEN Set $x^+ = x_k + s_k$ and $Y^+ = Y \cup (x_k) \setminus y^{out}$ Successful iteration.
   ELSE
   IF $(\|s_k\| \leq \|x_k - y^{out}\|)$
   THEN Set $Y^+ = Y \cup (x_k + s_k) \setminus y^{out}$
   ELSE Set $Y^+ = Y$
   ENDIF
   Set $x_k = x^+$
   Set $\Delta = \Delta^+$
   Set $Y = Y^+$
   ENDIF

8. Stopping criteria: If allowed, go to step 2.
```

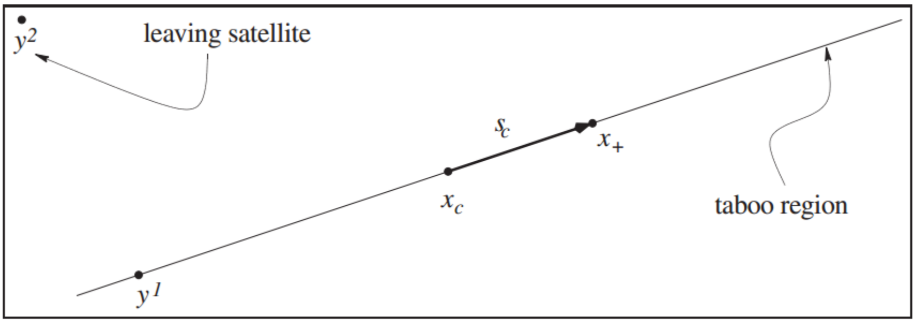


Figure 4.1: The taboo region illustrated in a two-dimensional space for the linear case. Figure taken from [36].

vector to the displacement vectors in $D \setminus s^l$. In the linear case the taboo region is defined as

$$\mathcal{T} = \{b^T s^i = 0, i = 1 \ldots m, i \neq l\} \tag{4.8}$$

Figure 4.1 shows a two-dimensional linear space, where two points is needed to make a exact interpolation. When the leaving sample point is not considered, the new trial point cannot lie on a extended line between the current iterate and the one sample point left in the sample set, as this will only give information about the planes slope in the direction of this line.
This condition will ensure that the sample set is not degenerate, however the trial point may lie infinitely close to the taboo region. Therefore, a requirement for the trial point to lie a certain distance from the taboo region is introduced, by demanding a certain angle between b and s in (4.8). The Wedge constraint for the linear case is then defined by

$$|b^T s| \geq \gamma \|b\| \|s\| \tag{4.9}$$

For the quadratic case the form of the model (4.3) is exploited, as this model is on the same form as the linear model. In essence, the same technique is applied, using a null-space vector $\hat{b}$ such that the taboo-region is defined as

$$\mathcal{T} = \{\hat{b}^T s^i = 0, i = 1 \ldots m, i \neq l\} \tag{4.10}$$

and demanding an angle between $\hat{b}$ and $\hat{s}$ to keep a certain degree of well-

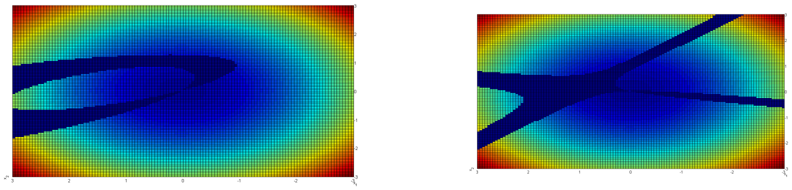


Figure 4.2: Plot showing the resulting sub-problem. The darkest region shows the region disallowed by the Wedge-constraint when the point (1,0) is chosen as the leaving point(left), and (1,1) as the leaving point(right).

poisedness, forming the Wedge constraint

$$|\hat{b}^T\hat{s}| \geq \gamma\|\hat{b}\|\|\hat{s}\| \tag{4.11}$$

As the sub problem (4.2) is solved in the form with $s \in \mathcal{R}^n$, the constraint (4.11) must also be expressed in this form. [36] addresses this issue, however these details are not important for a working understanding of the algorithm. In a similar way as $\hat{g}$ in 4.3, $\hat{b}$ is a vector extended from b as

$$\hat{b} = (b^T, \{B_{ij}\}, \{\frac{1}{\sqrt{2}}B_{ii}\}) \tag{4.12}$$

The final Wedge-constraint for the quadratic case in $\mathcal{R}^n$ can be expressed as

$$|b^Ts + \frac{1}{2}s^TBs| \geq \gamma\|s\|\sqrt{1 + \frac{1}{2}\|s\|_2} \tag{4.13}$$

In the same manner as (4.9) this constraint is non-linear, and both resulting sub problems are no longer convex.

Figure 4.2 shows the contour plot of the sub problem when the Wedge-constraint is introduced for the quadratic function $f(x_1, x_2) = x_1^2 + x_2^2$. The darkest region shows the regions not allowed for the trial point to enter. The trust-region constraint (4.2b) is not considered. Not only does the optimization problem become non-convex, but the allowed set is divided into disconnected regions. This problem has no general solution[40], however the Wedge-algorithm solves it using a combination of some techniques and heuristics.

Both in the linear and in the quadratic case the sub-problem (4.2) is first solved regardless of the Wedge-constraint (4.2c), retrieving a candidate to the trust-region step s^+. In the linear case this is accomplished by the formula $s^+ = -\frac{\Delta}{\|g\|}g$, and in the quadratic case by the use of techniques described by Mòre and Sorensen[36]. These are both common techniques in DFTRM. If the Wedge constraint (4.2c) is satisfied for the step s^+, the sub-problem solving mechanism can exit with solution $s^{TR} = s^+$. If not, the approaches are somewhat different for the quadratic and the linear case.

In the linear case, if s^+ lies in the Wedge-region, it can be shown that the solution of the full sub-problem (4.2) is on the edge of the Wedge-region, i.e. the Wedge constraint (4.2c) used as an equality constraint. s^+ is thus rotated in the plane that b and s^+ spans[36], until the Wedge constraint is just fulfilled.

For the quadratic case, the wedge constraint is used as a penalty-function. By computing the gradient of the left-hand side of the Wedge-constraint (4.13)

$$\nabla\phi(s) = sign(b^T s \frac{1}{2} s^T Bs)(Bs + b) \tag{4.14}$$

and generating trial-steps s^{trial} which satisfies $\|s^{trial}\| = \|s^+\|$, s^{trial} can be chosen such that decrease in the violation of the wedge-constraint occurs, while not changing the right-hand side of this. As in the linear case, these trial-steps are computed by rotating s^+ in the plane spanned by s^+ and $(B^T s + b)$[36].

A last action the algorithm takes is to make sure that the width of the Wedge-constraint, i.e. γ, is not to large. As the trial steps progresses and reducing the violation of the Wedge-constraint (4.13), the function value of the sub problem (4.2a) will possibly increase(because the trial steps moves away from the minimum point of $q(s)$). If the trial step s^{trial} gives an increase in $q(s)$ such that $\mu(q(x_k) - q(s^+)) \leq q(x_k) - q(s^{trial})$, this is seen as an indication that the width of the Wedge-constraint is to large. γ is then chosen such that the Wedge constraint is just satisfied, by solving (4.13) as an equality with respect to γ by inserting the trial step s^{trial}. This new γ is then used in the next iteration of the algorithm.

The sub-problem is now solved with $s^{TR} = s^{trial}$.

4.5 UOBYQA and CONDOR

UOBYQA is a well-known DFO algorithm, its name being a acronym for Unconstrained Optimization By Quadratic approximations[46]. It is claimed to be robust against noise in the objective function[45], and suited for problems with less than $n = 20$ free variables[46].

Condor is an extension of UOBYQA, exploiting parallel computing(actually possibility of distributing over several computers), added tolerance towards noisy objective functions and, maybe most interesting for the purpose of NMPC, handling of both bound, linear and non-linear constraints[7].

This section will present the relevant details of these two algorithms, taking basis in UOBYQA and comments will be made where additions/changes are made within Condor from this. The material presented is based on [45], [46], [7] and [8].

The UOBYQA algorithm uses a fully determined interpolation model, i.e. requiring $m = \frac{1}{2}(n+1)(n+2)$ interpolation point. The interpolation is performed using Lagrange polynomials, which also are used to maintain and control the quality of the quadratic model. Condor is here using parallel(distributed) computing over a network connection, by starting the optimization as UOBYQA, but starting a second process in parallel where the model is verified and improved on other processing cores if the user chooses to enable this feature. The optimization progresses as UOBYQA, and when finished the two processes joins and a better model is available for the next iteration. A good model is essential for fast convergence rate, however expensive to retrieve. Distributing this process clearly gives an advantage for optimization problems that have long running time, however for a real-time MPC problem which is solved typically from every tenth of a second to each second, this feature does not seem directly applicable as the communication between different processing cores over a network connection has a significant time consumption in it self and may be undesirable.

UOBYQA solves two trust-region sub-problems similar to the simplified algorithm 7. The first is the usual minimization of $q(x)$ within the trust-region. Because CONDOR handles both linear and non-linear constraints, these are introduced in the first sub problem as seen in (4.22) for this algorithm.

The second sub-problem is for the purpose of maintaining the poisedness of the sample set Y as described in section 3.5. However UOBYQA uses a different technique in finding the point to be replaced, as the quality of the model is monitored using a bound M on the difference between the third derivative of the model $q(x)$ and of the objective function $f(x)$. As neither of the function are differentiated, the bound is on a theoretical plane, however it is interesting to see ways of controlling the accuracy of the model and not only the poisedness of the sample set Y.

Algorithm 9 shows the steps of the UOBYQA algorithm.

The update of the local TR-radius ρ according to

$$\rho = \begin{cases} \max[\rho, \frac{5}{4}\|s^{TR}\|, \Delta + \|s\|] & \text{if } 0.7 \leq r \\ \max[\frac{1}{2}\rho, \|s^{TR}\|] & \text{if } 0.1 < r < 0.7 \\ \frac{1}{2}\|s^{TR}\| & \text{if } r < 0.1 \end{cases} \tag{4.15}$$

and the global

$$\Delta = \begin{cases} \Delta_{end} & \text{if } \Delta_{end} < \Delta \leq 16\Delta_{end} \\ \sqrt{\Delta_{end}\Delta} & \text{if } 16\Delta_{end} < \Delta \leq 250\Delta_{end} \\ 0.1\Delta & \text{if } 250\Delta_{end} < \Delta \end{cases} \tag{4.16}$$

In addition to the steps of algorithm 9, Condor has a parallel extension that is started between step 1 and 2, which continuously improves the model. The progress of this is then retrieved to increase the quality of $q(x)$ before performing the first sub problem, and before evaluating the quality of the model in step 10.

Condor also has a way of handling hidden constraints(see virtual constraints in[6]). This mechanism reduced the trust region when a function evaluation returns an error. This feature is not meant to be used to handle known, explicit constraints, but rather make the algorithm tolerant against failures in evaluating the objective function as discussed in section 4.3. As the simulation of the prediction model in the objective may fail for certain combinations of states and inputs, this is a feature that clearly improves the robustness of the controller.

4.5.1 Finding and replacing the "worst" point in the sample set

As the theory behind interpolation in chapter 3 reveals, maintaining the sample set Y with respect to poisedness is a non-trivial task. Chapter 3 presents some algorithms to generate Lagrange polynomials and use this to maintain the poisedness of the sample set. However an issues left unaddressed is the maximization of the Lagrange polynomials. This together with how the UOBYQA

Algorithm 9 The UOBYQA algorithm

1. Initialization: Given Δ_0, set $\rho = \Delta$ and generate a poised sample set Y using techniques described in [45]. Also set the bound on the model error $M = 0$.

2. Solve the first sub problem using the Mòre and Sorensen algorithm, to retrieve the trust region step s^{TR}.
   If $\|s^{TR}\| < \frac{\rho}{2}$ go to step 13.

3. Calculate the predicted reduction of the objective function $pred = q(x_k) - m(x_k + s^T R)$ and update M from equation (4.18).

4. Calculate the actual reduction in the objective, $ared = f(x_k) - f(x_k + s^{TR})$, and find the model agreement $r = \frac{ared}{pred}$.

5. Update the local trust region radius ρ by the use of the rules (4.15).

6. Find the point y^l that contributes the worst to the model $q(x)$, by the use of the technique described in section 4.5.1, and replace this with $x_k + s^{TR}$. Choose the new current iterate to be $x_k = min[x_k, x_k + s^{TR}]$.

7. Update the Lagrange polynomials to interpolate the new point.

8. Update M from equation 4.18.

9. If $f(x_k + s^{TR}) < f(x_k)$ or $\|s^{TR}\| > 2\rho$ then go to step 2.

10. Check the validity of the model $q(x)$ using equation (4.19).
    If model in invalid, use techniques described in section 4.5.1 to find the point which contributes the most to the model error and replace it. Update M from equation (4.18). Go to step 3.
    Else if $\|s^{TR} > \rho\|$, go to step 3.
    Else continue.

11. **If** Stopping criteria fulfilled exit algorithm with solution $x_k + s^{TR}$.

12. Update the global trust-region radius Δ using the rules (4.16). and go to step 2.

and Condor algorithms chooses a point y^l to replace in the sample set Y and finds a new point y^{new} are addressed in this section.

UOBYQA and Condor is based on calculating the error between the quadratic approximation $q(x)$ and the actual objective function $f(x)$. The theory takes basis in that comparing the Taylor expansions of $q(x)$ and $f(x)$, it is the third and higher order derivatives that will be different because $q(x)$ is a quadratic approximation to $f(x)$. Note that this value is purely theoretical, as $f(x)$ is never actually differentiated. Only the bound on the error will be stated here, as the proof can be found in [7] and [45]. The interpolation error is given by

$$InterpolationError = |q(x) - f(x)| < \frac{1}{6}M\sum_{j=1}^{m}|\ell_j(x)|\|x - y^j\|^3 \tag{4.17}$$

M is the bound on the third derivative of the Taylor expansion of $f(x)$ and is updated from

$$M = max[\frac{1}{6}M, \frac{|q(x_k + s^{TR}) - f(x_k + s^{TR})|}{\sum_{j=1}^{m}|\ell_j(x_k + s^{TR})|\|x_k + s^{TR} - y^j\|}] \tag{4.18}$$

This bound on the interpolation error is the driving factor of how the poisedness of the sample set is kept in UOBYQA and Condor.

At step 10 of algorithm 9, the validity of the model is checked. If all the points in Y are less than 2ρ away from the current iterate, the model is considered valid. If not, all the points which are further away than 2ρ are inspected, starting from the furthest. Sample point y^i is considered valid if

$$M = \|y^i - x_k\|^3 \underset{s}{max}\{|\ell_i(x_k + s)| : \|s\| \leq \rho\} < \epsilon \tag{4.19}$$

where ϵ is a bound for the interpolation error. This is an approximation of the contribution sample point y^i makes to the interpolation error[45]. When comparing this to the procedure of finding a sample point to replace in chapter 3, the difference is multiplication with the points distance from the current iterate. This is because the techniques used by UOBYQA and Condor are concerned with interpolation error, and not purely poisedness as described in chapter 3 and [12]. UOBYQA and Condor will therefore rather favour points which are close to the current iterate because of its implication to the model quality described in definition 3.6 in [12] and chapter 3.

If this test fails for subscript j, the sample point y^j is replaced with $x_k + d$, where d the solution to

$$\underset{s}{max}\{|\ell_j(x_k + s)| : \|s\| \leq \Delta\} \tag{4.20}$$

Equation (4.20) is then recognized as the second sub-problem, as seen in algorithm 4 and 5 and discussed in section 4.2.

A last interesting case described in [7] is if given a point y^{new}, and it is desirable to include this in the sample set Y by dropping a point in this, even though Y has passed the validation of equation (4.19). Which point in Y to replace to exploit the properties of y^{new} best? From chapter 3 it is known that it is desirable to choose a point y^l to be replaced such that $\ell_l(y^{new})$ becomes large, and as always dropping a point far from the current iterate is desirable. This leads to choosing y^l as the point that maximizes

$$\begin{cases} |\ell_i(y^{new})|\underset{i}{max}[1, \frac{\|y^i - y^{new}\|}{\rho^3}] & \textbf{if} f(y^{new}) < f(x_k) \\ |\ell_i(y^{new})|\underset{i}{max}[1, \frac{\|y^i - x_k\|}{\rho^3}] & \textbf{if} f(y^{new}) \geq f(x_k) \end{cases} \tag{4.21}$$

where $i \in [1, m]$ for the second equation, and x_k is considered a part of the sample set in the first one, as y^{new} in this case will become the new iterate.

4.5.2 Solving the first sub-problem

In UOBYQA and the unrestricted, open-source version of Condor, the first sub problem is solved using versions of the Mòre and Sorensen algorithm, which solves quadratic problems on the form of 4.1. The constrained version of Condor implements its constraint handling at this point, by modifying the constraints to accommodate the trust region step such that the TR sub problem becomes

$$\min_{s \epsilon \mathbb{R}^n} q(x_k + s) = s^T H s + G s + f(x_k) \tag{4.22a}$$

$$\text{s.t}$$

$$\|s\| \leq \Delta_k \tag{4.22b}$$

$$b_l \leq x_k + s \leq b_u, b_l, b_u \in \mathbb{R}^n \tag{4.22c}$$

$$A(x_k + s) \geq b, A \in \mathbb{R}^{n_L \times n}, B \in \mathbb{R}^r \tag{4.22d}$$

$$c_i(x_k + s) \geq 0, i \in [1, n_{NL}] \tag{4.22e}$$

where n_L is the number of linear constraints, and n_{NL} is the number of non-linear constraints. This problem is almost as hard to solve as the original problem, only that the objective (4.22a) now is a quadratic function, and as it is

an analytical one it is significantly easier to evaluate than evaluating the true objective which in NMPC can (and in this thesis will) involve a simulation.
Condor checks if the non-linear constraint (4.22e) is active for each iteration. If active, a SQP-algorithm is applied to find a search-direction and a line-search is performed. If not a combination of the Mòre and Sorensen algorithm, the Active-set algorithm and the null-space method is used to solve the problem. This latter solution makes it possible to extend the Active-set method to also handle the non-linear constraint associated with the trust-region constraint (4.22b). More details of this solution can be found in [7], and concerning the active-set method, SQP and null-space method [40] can be recommended reading.

4.5.3 Solving the second sub-problem

As described in chapter 3, the cross-terms in the Lagrange polynomials and the absolute value makes the second sub-problem non-convex, and to find an exact solution some sort of global optimization algorithm must be applied. However UOBYQA and Condor uses an approximation to solve the problem.
Starting from optimization problem (4.20), this is rewritten to the form

$$\max_{s \in R^n} \ell_i(s) = \frac{1}{2} s^T (H_\ell) s + (G_\ell) s \tag{4.23a}$$

$$\text{s.t}$$

$$\|s\| \leq \Delta \tag{4.23b}$$

$$\tag{4.23c}$$

where only one Lagrange polynomial is considered, therefore the subscript i denoting the specific polynomial is omitted for simple notation. H_ℓ and G_ℓ are the coefficient matrices for the first and second-order terms of the Lagrange polynomial.
The technique used by UOBYQA and Condor exploits that because of the shape of the constraint (4.23b), s can be exchanged with $-s$ and still satisfy this constraint. This makes it possible to consider (4.23) as

$$\max_{s \in R^n} \frac{1}{2} |s^T (H_\ell) s| + \max_{s \in R^n} |(G_\ell) s| \tag{4.24a}$$

$$\text{s.t}$$

$$\|s\| \leq \Delta \tag{4.24b}$$

If the solution $\hat{s}$ to the linear term of (4.24a) and $\tilde{s}$ to the quadratic term could be found, making the choice between $\hat{s}$,$-\hat{s}$, $\tilde{s}$ and $-\tilde{s}$ as the one that gives the largest value to (4.23) can be shown to be no less than half the size of the exact solution[45].

As in the Wedge algorithm(see section 4.4 and [36]), the solution for the linear term can be calculated from $\hat{s} = \pm\rho\frac{G_\ell}{\|G_\ell\|}$. The solution to the quadratic is $\tilde{s} = \pm\rho\frac{v_{max}}{\|v_{max}\|}$, where v_{max} is the eigenvector corresponding to the eigenvalue of H_ℓ with largest absolute value. This approach is used in chapter 7, however computing the eigenvectors are expensive, so [45] suggests a method to approximate the direction of these. Algorithm 10 shows the steps performed to solve the quadratic part of the second sub-problem. When both $\hat{s}$ and $\tilde{s}$ are retrieved, the algorithm sets the final solution to a linear combination to these to improve it further. This combination goes beyond just $\tilde{s}$, $-\tilde{s}$, $\hat{s}$ and $-\hat{s}$. Therefore the reader is referred to equation (2.18) in [45] and the above section in this, keeping in mind that G_j and h_j corresponds to H_ℓ and G_ℓ respectively, and d in [45] corresponds to s.

Algorithm 10 Solve the quadratic part of the second sub-problem

1. Decide the vector ω which is the column of H_ℓ with the largest euclidean norm.
2. The direction $\tilde{d}$ of $\tilde{s}$ is picked from the span of ω and $H_\ell\omega$, namely $\tilde{d} = \alpha\omega + \beta H_\ell\omega$
3. Choose α and β by maximizing the expression

$$\{\alpha, \beta\} = \underset{\alpha,\beta\in\mathbb{R}^1}{max} \frac{|(\alpha\omega + \beta H_\ell\omega)^T H_\ell(\alpha\omega + \beta H_\ell\omega)|}{\|\alpha\omega\beta H_\ell\omega\|} \tag{4.25}$$

4. Choose $\tilde{s}$ on the edge of the trust-region, $\tilde{s} = \rho\frac{\tilde{d}}{\|\tilde{d}\|}$

4.6 BOBYQA

The BOBYQA algorithm is the successor of COBYLA, UOBYQA, and NEWUOA[28]. The algorithms main structure is quite similar to the that of algorithm 7 and UOBYQA as of solving two sub problems with purposes of minimizing a quadratic approximation and ensure poisedness. Its main difference from the UOBYQA algorithm described previously is that it builds the model from an under-determined sample set, and support bound constrains on the MV. However the technique used to update H from the previous Hessian is numerically sensitive because the KKT-matrix must be inverted, therefore much attention is given to avoid numerical round-off errors in this calculations[50], leading to this being the focus of the second sub problem, called Alternative Move (ALTMOV). This leads to the details of the algorithm to be more mathematically complex than the algorithms previously described in detail, however it has both in this thesis and other experiments shown very promising results[52][28]. This section is an excerpt of the parts in [52] concerning the initialization, model update technique and solving the sub-problems, as the rest of the algorithm is mostly similar to UOBYQA(see section 4.5).

4.6.1 Initialization

The initialization procedure of BOBYQA gives a good working understanding of how the under-determined model is solved in this algorithm. The choice of the starting point for the optimization also proved to play an important role when the algorithm was applied to the NMPC problem of section 6.5.

Given an initial starting point $x_0 \in \mathcal{R}^n$, trust region radius $\Delta_0 \in \mathcal{R}^1$ and vectors of upper and lower bounds for the MV, $b_u, b_l \in \mathcal{R}^n$ respectively. Also the user specifies the number of interpolation points $m \in [n+2, \frac{(n+1)(n+2)}{2}]$ for the interpolation model. When creating the sample set, BOBYQA makes step with magnitude Δ_0 along coordinate directions. These steps are chosen as

$$\begin{cases} y^{i+1} = x_0 + \Delta_0 e_i \text{ and } y^{n+i+1} = x_0 - \Delta_0 e_i & \textbf{if } b_l < x_0 < b_u \\ y^{i+1} = x_0 + \Delta_0 e_i \text{ and } y^{n+i+1} = x_0 + 2\Delta_0 e_i & \textbf{if } x_0 = b_l \\ y^{i+1} = x_0 - \Delta_0 e_i \text{ and } y^{n+i+1} = x_0 - 2\Delta_0 e_i & \textbf{if } x_0 = b_u \end{cases} \tag{4.26}$$

where $i \in [1, n]$ and $e_i = \begin{bmatrix} 0, \dots, 0, & 1, & 0, \dots, 0 \end{bmatrix}^T$ with the i'th element equal to one. BOBYQA automatically changes the components of the initial iterate to lie on the bound, if this component is outside the feasible region. Also, as the first line of (4.26) implies, all components of x_0 must have a distance of at least Δ_0 from the bounds. x_0 is therefore changed according to the following rules

$$x_{0i} = \begin{cases} b_{li} & \textbf{if } x_{0i} < b_{li} \\ b_{ui} & \textbf{if } x_{0i} > b_{ui} \\ x_{0i} + \Delta_0 & \textbf{if } b_l i < x_0 i < b_l i + \Delta_0 \\ x_{0i} - \Delta_0 & \textbf{if } b_u i > x_0 i > b_u i - \Delta_0 \end{cases} \tag{4.27}$$

Further, if $b_{ui} - b_{li} < 2\Delta_0$ there is no room to make the sample points of (4.26), and an error is returned.

If $m \leq 2n + 1$ then the initial Y is now complete. If not, more points are generated by combining the already defined points, i.e. for $i \in [2n + 2, m]$

$$y_i = y^{p(i)} + y^{v(i)} - x_0 \tag{4.28}$$

where $p(i)$ si defined as

$$p(i) = \begin{cases} i - 2n - 1 & \textbf{if } 2n + 2 \leq i \leq 3n + 1 \\ p(i - n) & \textbf{if } 3n + 2 \leq i \leq m \end{cases} \tag{4.29}$$

and $v(i)$ as

$$v(i) = \begin{cases} p(i) + i & \textbf{if } p(i) + k \leq n \\ p(i) + i - n & \textbf{if } p(i) + k > n \end{cases} \tag{4.30}$$

Next BOBYQA builds the first quadratic model

$$q(x_0 + s) = f(x_0) + G_0 s + s^T H_0 s \tag{4.31}$$

by giving priority to the bias $q(x_0)$ and the gradient G_0, which requires a total of $n + 1$ points. Then the coefficients of the diagonal of the Hessian $H_{ii}, i \in [1, m - n - 1]$ is interpolated from the points $y^i, i \in [n + 1, m]$. Further, if $m > 2n + 1$, the remaining coefficients $H_{q(i)p(i)}$ is calculated from the points $y^i, i \in [2n + 2, m]$. The remaining elements of H is set to zero, which can be interpreted as minimizing the Frobenious norm of H[52].

Note that [51] recommends $2n + 1$ as the number of interpolation points for BOBYQA, which will start the optimization with a diagonal approximation to the Hessian[50].

Also an initial inverse of the KKT-matrix is calculated, so that the inverse for iteration 1 has a previous matrix to be calculated from. However the details is omitted here, and can be found in [52].

4.6.2 Model update technique

The BOBYQA algorithm uses under-determined interpolation, using from $n+2$ to $\frac{(n+1)(n+2)}{2}$ sample points. The model is written on the same form as 4.31,

$$q(x_k + s) = s^T H s + G s + f(x_k) \tag{4.32}$$

As mentioned earlier, this model is determined with $n + 1$ sample points by neglecting the Hessian H. However to guarantee that H will have a change from one iteration to the next the size of m must be chosen to $n + 2$ or larger [40],[50]. The remaining DOF in H are absorbed by minimizing the Frobenious norm of the difference of the Hessian from one step (k) to the next $(k + 1)$.

$$\|H(x)_{k+1} - H(x)_k\|_F \tag{4.33}$$

thus taking up the undetermined space in $q(x)$ by making it close to the previous Hessian. The update of $q(x)$ from one iteration to the next is performed by requiring that H is symmetric and the interpolation condition to hold, and solving the equality-constrained, convex optimization problem

$$\begin{aligned}
&\min_{s \in \mathbb{R}^n} \ \|H(s)_{k+1} - H(s)_k\|_F && \text{(4.34a)}\\
&\text{s.t}\\
&\quad H = H^T && \text{(4.34b)}\\
&\quad q(y^l) = f(y^l) \ \forall \ l \in [1, m] && \text{(4.34c)}
\end{aligned}$$

By using the KKT-matrix[40] for (4.34) and inserting the interpolation points from Y this problem can be solved as a set of linear equations to decide the coefficients of H_{k+1}.

When solving this system of equations the KKT-matrix is inverted by calculating the inverse of this matrix from the previous inverse, by only updating the row and column that actually is changed due to the one point that has changed in the last iteration[48]. This way the work of updating the Hessian requires $\mathcal{O}(n^4)$ operations if the sample set is fully determined, and $\mathcal{O}(n^2)$ operations when $m = 2n + 1$[52], thus encouraging a small sample set. Details of how the model update is performed is described in [48] and [49].

4.6.3 Solving the first sub-problem(TRSBOX)

The BOBYQA sub problem

$$\underset{s\in R^n}{min}\ q(x_k+s) \tag{4.35a}$$

$$\text{s.t}$$

$$\|s\| \leq \Delta \tag{4.35b}$$

$$b_l \leq x_k+s \leq b_u, b_l, b_u \in \mathbb{R}^n \tag{4.35c}$$

is solved using an active-set algorithm called Trust Region Step in the Box (TRSBOX). This algorithm stores the current active bounds in the set $\mathcal{I}$, and uses a function $\mathcal{P}_{\mathcal{I}}$ to calculate a Cauchy step which moves along the active constraints. $\mathcal{P}_{\mathcal{I}}(x)$ is such that given a vector $v \in \mathbb{R}^n$ and $\hat{v} = \mathcal{P}_{\mathcal{I}}(v) \in \mathbb{R}^n$, component i of $\hat{v}$ is

$$\hat{v}_i = \begin{cases} 0 & \mathbf{if} i \in \mathcal{I} \\ v_i & \mathbf{if} i \notin \mathcal{I} \end{cases} \tag{4.36}$$

and the Cauchy-step d is calculated as

$$d = -\mathcal{P}_{\mathcal{I}}(G(s)) \tag{4.37}$$

Then a line search is performed that is terminated with solution s if the model is no longer decreasing along the line, or if further progress is restricted by the constraint (4.35b). Further, if the line-search hits a bound(4.35c) a termination check is performed.

$$\|\mathcal{P}_{\mathcal{I}}(G(x_k+s))\|\Delta \leq 0.01[m(x_k) - m(x_k+s)] \tag{4.38}$$

If the expression (4.38) returns true, little progress is likely to be gained from further iterations, and termination occurs with solution s. Otherwise the new active constraints are added to $\mathcal{I}$ and TRSBOX is restarted. Additionally, s may be moved around on the trust region boundary if a sufficient improvement in $q(x)$ justifies this. The details of this procedure is omitted here, but can be found in [52].

4.6.4 Solving the second sub-problem(ALTMOV)

If the solution from TRSBOX has a norm of less than $\|\frac{1}{2}\Delta_k\|$, the poisedness-improving mechanism ALTMOV is invoked, which is the BOBYQA-equivalent to the second sub-problem. The goal of this algorithm is to find a replacement for a leaving point y^l, that is the point furthest away from x_k, such that the inverse of the KKT-matrix is kept valid. ALTMOV generates two different steps, d and c which accomplishes this. The first is chosen to lie on a line between the current iterate x_k and one of the interpolation points $y^i, i \in [1, m]$. The absolute-value of the Lagrange polynomial corresponding to the leaving point is evaluated along each of these lines(except the line corresponding to the leaving point itself). The point along all of these lines that gives the maximum value without violating the constraints (4.35b) and (4.35c) is chosen as d.

Also the alternative to d is generated, namely c. This step is found by calculating the Cauchy step of the linearised Lagrange-polynomial corresponding to the outgoing sample, ℓ_l subject to the bounds (4.35c) and the trust region constraint (4.35b).

$$\underset{s \in R^n}{max} \; |\ell_l(x_k) + s^T \nabla \ell_l(x_k)| \tag{4.39a}$$

$$\text{s.t}$$

$$\|s\| \leq \Delta_k \tag{4.39b}$$

$$b_l \leq s + x_k \leq b_u, b_l, b_u \in \mathbb{R}^n \tag{4.39c}$$

Problem (4.39) is both minimized and maximized, to take into the account the goal of maximizing the absolute value. By choosing $s_i = (b_l)_i - (x_k)_i$ if $\nabla \ell_l > 0$, $s_i = (b_u)_i - (x_k)_i$ if $\nabla \ell_l < 0$ and zero otherwise, problem (4.39) is solved if the trust region constraint is satisfied. If not, ALTMOV iteratively retreats some of the components of c against x_k until it is satisfied.
Whether to use c or d to replace y^t is decided when updating the model $q(x)$ later in the iteration, and for these details the reader is referred to [52].

In addition to ALTMOV, BOBYQA has a feature called RESCUE. This procedure is initiated when rounding errors occurs in the updating of the inverse KKT-matrix, causing this to become close to singular. RESCUE handles this by moving some point in Y. According to [52] this method is rarely invoked, and is omitted here as this sections purpose is to give a understanding of the main mechanisms of BOBYQA.

Chapter 5

A NMPC case study

To test the performance of the optimization algorithms, a NMPC scheme is developed for a system with a prediction model where DFO becomes beneficial. Such characteristics can be high stiffness and usage of a variable-step solver inducing numerical noise, logic operators, lookup tables and sets where the model is not defined. These characteristics typically occurs in fluid-flow models, and such a scenario were described in the fluid-tank model in [13], where the square-root characteristics of a fluid-valve induced highly varying stiffness, which in turn generated a rugged cost-surface when a variable-step ODE solver were used to simulate the prediction model. However this scenario proved to me rather simple, as it contained only a single input and one differentiated state. In this report a more complex MIMO system is tested, to differentiate the algorithms more clearly, as suggested in [13]. This chapter will present such a system, and how the NMPC problem is defined. Also the system will be analysed in a similar manner as the one from [13], to show what characteristics it inhabits.

5.1 The crude-oil separator

The system that will be used to test the DFO algorithms is a model of a sub-sea crude-oil separator, developed by Statoil. The model is a simplified model of a laboratory setup as the system is still under design, and figure 5.1 shows its structure. The following description of the separation process is an excerpt from [41], where also the details of the simplifications can be found.

The unit separates the input crude oil flow q_{in} into its gas q_{out1} and liquid q_{out2} component flows. The input flow rate may be time varying, as well as the fraction of gas in the crude oil, called Gas Volume Fraction (GVF), which varies from zero(only liquid) to one(only gas).

The unit is split into two main control volumes, each containing a liquid level. The Gas-Liquid Cylindrical Cyclone (GLCC) does the first separation of the crude oil, however at high flow rates the GLCC does a very rough separation[41].

Therefore both the gas output flow q_1 and the liquid output flow q_2 from GLCC is fed through a second separation stage, called the De-Liquidizer (DL) and Phase Splitter (PS) respectively. Both these stages are co-axial cyclones, which rotates the input mixture to create a centrifugal force which separates the gas and liquid phases. The gas is then extracted from the centre of the cyclones, while liquid will be along the walls.

Gas-flow from the GLCC to the DL, q_1, is controlled by the valve u_1. The gas outlet from the PS, q_3, is fed into the mixture inlet of the DL, while the liquid outlet of the DL, q_5, is fed right into the liquid outlet of the separator. Both of these flows are controlled by valves u_3 and u_4 respectively. At outlet q_{out1} there is a compressor to move the gas to the surface, and at q_{out2} there is a pump for the same purpose for the liquid. The feed to these are controlled by the valves $u_{compressor}$ and u_7 respectively. Note that the former will be referred to as u_c from now.

The model also contains the liquid levels of the GLCC and the DL. These are assigned notation h_s and h_{dl} respectively, and measured in meters. These levels are not necessarily desirable to keep at a set-point, as allowing them to move freely within certain bounds can make the whole system act more smoothly. However the cyclones has a physical maximum liquid level where they will overflow. Both these limits are at 2 meters, and these therefore naturally becomes constraints in the NMPC. The model is not valid for liquid levels below zero, thus it makes sense to also constrain the levels to positive values.

Also important variables for the NMPC is the pressures of Control volume 1 (CV1), p_1, of Control volume 2 (CV2), p_2, and the difference between these pressures Δ_p. From [41], p_1 should be kept between 9 and 15 bar, p_2 between 8 and 12 bar while Δ_p between 0.5 and 4 bar. Simulations done in earlier studies has shown that these constraints are relatively easy to satisfy [41] [33], however the consequence of a violation is not accounted for.

The overall goal of the separation process is to keep the GVF of the gas outlet, GVF_{out1}, as close as possible to one, and the GVF of the liquid outlet, GVF_{out2}, as close as possible to zero. This goal is rarely feasible to obtain perfectly, however two physical constraints are present on these variables. The compressor and the pump receiving q_{out1} and q_{out2} has limits on the GVF of their input flow. Violating these limits can inflict damage to the units. This results in a lower constraint in the GVF of q_{out1} of 0.97 and the GVF of q_{out2} should be constrained to below 0.03. However it should be pointed out that these constraints can be allowed violated for short periods of time. Although satisfying these constraints under violent disturbances is feasible as shown in

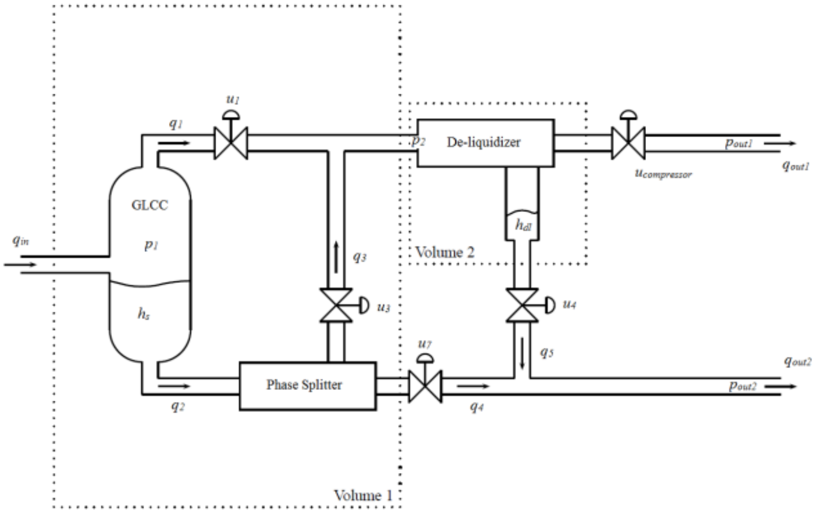


Figure 5.1: The separator unit used in testing the NMPC

[33], this results in violent input controls. Also the system has a big spread of time constants, from the rapidly changing GVF and pressures to the slow-varying liquid levels. To control the quick modes, the input blocks of the MPC horizon should be chosen short, while controlling the slow modes requires a long horizon. Together these two criteria results in a large amount of MV. Therefore [41] and [33] pre-stabilises the plant using PID-controllers for controlling the liquid levels by feeding the measurements of h_{dl} and h_s back to the valves u_4 and u_7 respectively. This both reduces the number of variables as the MPC only has to control three valves instead of five, and as the slow dynamics is stabilized a much shorter horizon can be chosen. Both the approach of controlling all valves with MPC and using pre-stabilization is tested in this thesis.

A summary of the variables of the model with their notation are shown in table A.1. Note that the model distinguishes between parameters and states, where the latter is variables which must be integrated and fed back into the model, whereas parameters do not need to be integrated before fed back. To simulate the model in open loop for deciding the structure and initial settings for the tuning variables of the NMPC optimization problem, initial values for the model is found from experiments and simulations. Table B.1 shows values

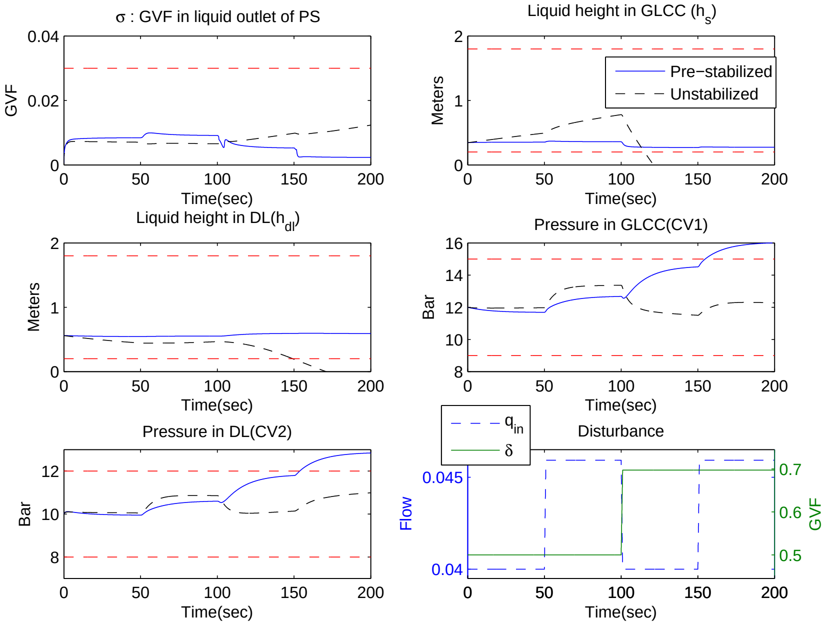


Figure 5.2: The separator model with steady-state inputs and steps in the disturbance.

that will lead to a steady-state behaviour of the model. The analysis is done by applying steps in the disturbance, and keeping the valve openings constant. First a step in the input GVF occurs, then this step is set back to its steady-state value. The input flow is then perturbed, and towards the end both disturbances are perturbed simultaneously.

Figure 5.2 shows the result of the simulation. In the un-stabilised case, the liquid levels are highly sensitive towards the disturbance. As the flow increases, so does the liquid levels as more liquid is introduced in the unit. However when the input GVF δ increases a smaller fraction of the input flow is liquid, thus the liquid levels decreases and actually reaches zero and continue in negative direction relatively quickly. As this clearly is not realistic the trajectory after this point is not representative in respect to the real-world unit, however it should be noted that all other states remains within their constraints. Note that the GVF of the DL, μ, and the difference pressure Δ_p is not plotted to save space, as these variables are generally very close to their optimal values and are thus not considered very interesting.

By applying the PID-controllers for pre-stabilizing the liquid levels, figure 5.2 shows that the liquid levels now become locally stable, and are kept close to their set-points. However this comes at the cost that the pressures in the control volumes increases. It seems that when only one of the disturbances are perturbed, the pressures will converge close to their constraints, however when both q_{in} and δ are perturbed from their steady-state values, constraints are violated. Also the output GVF of the PS σ is more excited from the active usage of u_4 and u_7, which both controls the flow where σ is measured.

These results suggests that controlling the model leads to contradicting goals, and there must be compromises between the performance of the different outputs. MPC is thus well suited, as it is intuitive to adjust the priority of the different objectives as discussed in chapter 2.

5.2 Defining the NMPC problem

It is reported from Statoil and [41] that the system sets great demands for the choice of the ODE-solver, and this may not be available for the multiple-shooting approach. Also the model is implemented in the C++ programming language, and as parameters and states are distinguished, it is intuitive to do the simulations in Simulink. This makes it convenient to call the simulation from within the cost function which is a Matlab-function, thus becoming a single-shooting NMPC problem.

From section 5.1 it is known that the most important purpose of the controller is to keep the output GVF σ and μ from the separator sufficiently pure to protect the compressor and pump, keep the liquid levels h_{dl} and h_s between zero and their maximum bounds to keep the model within its valid limits and ensure that the pressures is within their limits. Also the valve inputs are scalars between zero and one which introduces bounds on these. [41] and [33] used ideal values to punish valve openings that deviate from the steady-state settings. During the simulations in this thesis it is experimented with both punishing change in the input sequence and using penalty for deviation from ideal values. The former gave a smoother trajectory of the inputs, however penalty for deviation from the ideal values(the steady-state values of table B.1) proved important to make

the valves converge quickly when the dynamics of the system settles.
In [41] it was used set-points for the pressures p_1, p_2 and Δ_p, and this is continued in this thesis. Further it is introduced set-points for the output GVF, σ and μ, of zero and one respectively. This is in an attempt to make it easier for the controller to minimize variations in these variables, by applying a small weight in errors for these variables "pulling" them away from the constraint even if this is not violated. The same idea is used when tuning the other constrained states and parameters. Although the constraints shall ensure that the given variable is never outside its bound, by applying a set-point with a small priority as far as possible away from the constraints, the variable is not as likely to encounter the constraint. In some cases though it may be desirable to work as closely as possible to constraints, then this approach is not ideal. In this case study however it does not matter where between the constraints a variable operates, and the approach proved to work well.
Two different optimization problems are defined, one for the plant with stabilized liquid levels and one without pre-stabilization. The latter also introduces so-called terminal-cost[38] for the liquid levels, as this improves the performance of the controller. Both problems are defined in the same way, although in the pre-stabilized case all penalties associated with u_4, u_7, h_{dl} and h_s are set to zero, as well as the terminal cost for the liquid levels.
The resulting objective of the NMPC is summarized in table 5.1, and (5.2) shows the resulting optimization problem, when the notation from section 2.3 is used.

Note that to simplify the tuning of the optimization problem, scaling to the weights are applied. This scaling is done according to the formula presented in [23]:

$$\tilde{q}_i = \frac{q_i}{(i_{max} - i_{min})^2} \tag{5.1}$$

where q_i is the weight for variable i from table 5.1, and $\tilde{q}_i$ is the scaled weight applied in the objective function. i_{max} and i_{min} are typical maximum and minimum values for variable i. This can e.g. be constraints, or physical maximum and minimum limits, or a combination. This way if two weights are chosen equal, the corresponding variables will become equally punished for a relatively equal deviation.

Variable	Set-point	Upper bound	Lower bound	Priority	Terminal
σ	0	0.03	-	1	-
μ	1	-	0.97	10	-
p_1	12	15	9	40	-
p_2	10	12	8	40	-
Δ_p	2	4	0.5	60	-
h_s	0.7	2	0	40	40
h_{dl}	0.5	2	0	20	60
u_c	0.38	1	0	2	-
Δu_c	0	-	-	4	-
u_1	0.24	1	0	1	-
Δu_1	0	-	-	2	-
u_3	0.05	1	0	0.24	-
Δu_3	0	-	-	0.24	-
u_4	0.40	1	0	0.6	-
Δu_4	0	-	-	0.6	-
u_7	0.15	1	0	0.6	-
Δu_7	0	-	-	0.6	-

Table 5.1: Objective of the NMPC. Priority are the weights applied to the stage-cost, before scaled according to (5.1) and applied in (5.3b)-(5.3f). Terminal is the terminal-weights, before they are scaled and applied in (5.3g). If pre-stabilization is used, the priority to h_{dl}, h_s, u_4 and u_7 set to zero, as well as the terminal cost to h_{dl} and h_s.

.

$$\min_{u\epsilon\mathbb{R}^n} \; f(\ell(u(0),\ldots,u(K-1),x(0),K),u(0),\ldots,u(K-1)) \tag{5.2a}$$

s.t

$$\sigma(k) \leq \sigma_{max},\; k=1,2\ldots\hat{K} \tag{5.2b}$$

$$\mu_{min} \leq \mu(k),\; k=1,2\ldots\hat{K} \tag{5.2c}$$

$$p_{1_{min}} \leq p_1(k) \leq p_{1_{max}},\; k=1,2\ldots\hat{K} \tag{5.2d}$$

$$p_{2_{min}} \leq p_2(k) \leq p_{2_{max}},\; k=1,2\ldots\hat{K} \tag{5.2e}$$

$$\Delta_{p_{min}} \leq \Delta_p(k) \leq \Delta_{p_{min}},\; k=1,2\ldots\hat{K} \tag{5.2f}$$

$$h_{s_{min}} \leq h_s(k) \leq h_{s_{max}},\; k=1,2\ldots\hat{K} \tag{5.2g}$$

$$h_{dl_{min}} \leq h_{dl}(k) \leq h_{dl_{max}},\; k=1,2\ldots\hat{K} \tag{5.2h}$$

$$u_{min} \leq u_i(k) \leq u_{max},\; k=0,1\ldots K-1 \tag{5.2i}$$

Where the objective function is defined as

$$f(x(1),\ldots,x(\hat{K}),u(0)\ldots u(K-1)) = \tag{5.3a}$$

$$\sum_{k=1}^{\hat{K}} \frac{1}{\Delta t(k)}[q_{sigma}\|\sigma(k)-\sigma_{ref}\|_2 + q_{\mu}\|\mu(k)-\mu_{ref}\|_2 + \tag{5.3b}$$

$$q_{h_s}\|h_s(k)-h_{s_{ref}}\|_2 + q_{h_{dl}}\|h_{dl}(k)-h_{dl_{ref}}\|_2 \tag{5.3c}$$

$$+q_{p_1}\|p_1(k)-p_{1_{ref}}\|_2 + q_{p_2}\|p_2(k)-p_{2_{ref}}\|_2 \tag{5.3d}$$

$$+q_{\Delta_p}\|\Delta_p(k)-\Delta_{p_{ref}}\|_2] \tag{5.3e}$$

$$+\sum_{k=1}^{K}\sum_{i=1}^{5} r_i\|u_i(k)-u_{i_{ref}}\|_2 \tag{5.3f}$$

$$+e_{h_s}\|h_s(\hat{K})-h_{s_{ref}}\|_2 + e_{h_{dl}}\|h_{dl}(\hat{K})-h_{dl_{ref}}\|_2 \tag{5.3g}$$

Where e_{h_s} and $e_{h_{dl}}$ are the terminal weights of table 5.1, after scaling according to (5.1). Note the introduction of the differentiated time vector Δt, which holds the interval between each time step of the states. This is assumed to be available from the ODE-solver, and is introduced be able to do a discrete integration. This is important when applying a variable-step solver, as the number of sample points returned by $\ell(u(0),\ldots,u(K-1),x(0),K)$ will depend on the dynamics of the system, which will change according to the state it is in. Failing to taking this into account will typically favour solutions where the solver chooses long

steps, because this will tend to make the sum smaller.
For the choice of the control, and prediction horizon, figure 5.2 suggests that the slowest dynamics has a time constant of approximately 40 seconds. This choice proved to work well during simulations in the "Full MPC" case. For the pre-stabilized case, pressures seems to be surprisingly slow when inspecting figure 5.2. However as the pressure in the DL shows, they are closely correlated with the liquid levels, and it is likely that the dynamics of the pressures becomes faster when the liquid levels becomes high thus less space is available for the gas in the cyclones. Simulations proves that a horizon down to 5 seconds stabilizes the model, however larger choices give smoother trajectories of the valves. Therefore a horizon of 20 seconds is chosen. The input sequence is divided into 5 blocks. The length of each input block is $\{10\%, 10\%, 20\%, 20\%, 40\%\}$ of the total length of the control horizon. Increasing the number of blocks does not give an significant improvement in closed-loop performance.

5.3 Developing a penalty-function for constraint handling

Because DFO algorithms often does not handle constraints, and because of the single-shooting approach to NMPC used in this thesis, the most common and general way of handling constraints on the system states is the use of penalty-functions[30]. This expansion of the objective function introduces penalty(i.e. increased cost) for constraint violations. The penalty function will here be denoted $\psi : \mathcal{R}^j \mapsto \mathcal{R}^1$, j being the number of states, and $x(k) \in \mathcal{R}^j$ is the trajectory of the states to be constrained.
Several different approaches for designing $\psi(x_k)$ has been proposed in literature[40]. Given that it is desirable to constrain $x(k)$ such that $x(k) \leq x_c$, $k = 1 \dots \hat{K}$, $x_c \in \mathcal{R}^j$, a common choice for ψ is the L_1 or L_2-norm

$$\psi(x(k)) = q_c \|x_c - x(k)\|_i^- \tag{5.4}$$

where the minus-sign denotes that it is only the elements in $x(k)$ that violets the constraint that contributes to the norm. In the objective function the penalty for the whole horizon is summarized(or integrated if variable-step solver is used), and added to the original objective function. q_c is a weighting constant, which must be chosen such that the complete problem becomes well-scaled, and large enough that the algorithm will prioritize to minimize the penalty function before minimizing the rest of the objective function. At this point the L_1 separates from the L_2-approach, as the former will ensure that the penalty function always is reduced to zero if possible[23]. However it has the disadvantage of being non-differentiable when ψ crosses zero, which is likely to happen regularly when operating close to constraints. During some simulations DFTRM tends to struggle with this penalty function, as the trust region quickly is being reduced due to the difference between the smooth model and the non-smooth objective function.

Other common choices of ψ is the logarithmic

$$\psi(x(k)) = \begin{cases} -\log(x_c - x(k)) & \text{if } x(k) < x_c, \\ \infty & \text{if } x(k) \geq x_c \end{cases} \tag{5.5}$$

and inverse functions

$$\psi(x(k)) = \begin{cases} \frac{1}{x_c - x(k)} & \text{if } x(k) < x_c, \\ \infty & \text{if } x(k) \geq x_c \end{cases} \tag{5.6}$$

however these are not very suitable for model-based derivative-free optimization such as the DFTRM used in this thesis. These algorithms samples the objective function around the current iterate, and as figure 5.3 illustrates, if this iterate is close to the constraint, samples may well lie outside the feasible area.

Using penalty functions that returns ∞ in the infeasible set clearly will introduce problems when trying to build a model from the sample set, as the infeasible samples will not contain any information other than that they are indeed infeasible.

Other proposals are adding a fixed value to the resulting cost if constraints are violated[30]. This however makes the problem highly non-smooth, and DFTRM proved to have severe difficulties with this approach as the gradient becomes undefined when crossing between the infeasible and the feasible set. If using other algorithms which are developed to handle non-smoothness and discontinuities, this approach may however be effective.

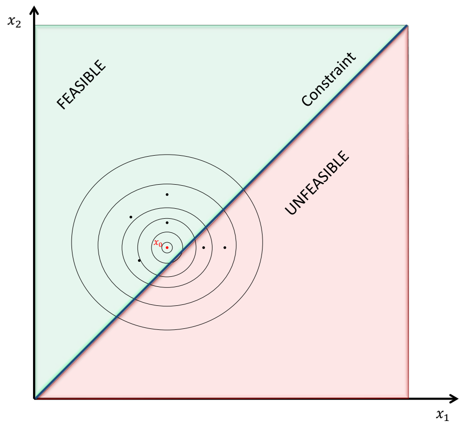


Figure 5.3: Illustration of how the sample points may lie in the infeasible set, and therefore this area should return a valid function value whenever possible.

Based on this the L_2 norm is chosen as the most promising candidate, and penalties are implemented in $\psi(x(k))$ for deviation in the constraints (5.2b)-(5.2h). The new objective function consisting of the old objective and the penalty function can be written

$$\begin{aligned} &\bar{f}(x(1),\dots,x(\hat{K}),u(0)\dots u(K-1)) \\ &\qquad = f + \sum_{k=1}^{\hat{K}} \frac{1}{\Delta t(k)} \psi(x(k)) \end{aligned} \tag{5.7a}$$

where the argument for the old objective function (5.3) is not denoted for simplicity of notation. As this new objective takes into account the constraints, the new optimization problem can be written

$$\min_{u \in \mathbb{R}^n} \; \bar{f}(\ell(u(0),\dots,u(K-1),x(0),K), \tag{5.8a}$$

$$u(0)\dots u(K-1)) \tag{5.8b}$$

$$\tag{5.8c}$$

$$\text{s.t}$$

$$u_{min} \leq u_i(k) \leq u_{max} \tag{5.8d}$$

$$k = 0, 1 \dots, K-1$$

The weights on the penalties are chosen two times that of the respective weight presented in table 5.1, and scaled according to (5.1) before applied in ψ. Note that the constraints on the inputs are left as explicit constraints. This is because these are bounds directly on the MV, and is therefore easy to check if are satisfied, and also easy to correct if violated. For convenient notation, f is defined as the complete objective function, i.e. $f := \bar{f}$.

5.4 Implementing the model and NMPC in Matlab/Simulink

To test the performance of the NMPC, the separator model is implemented in Simulink in closed loop with the NMPC. Figure 5.4 shows the Simulink diagram of the model that is to be controlled, referred to as the plant. This is simulated with Matlabs fixed-step solver *ode1*, with a sampling period of 10^{-2} seconds. This is the same sample period as used in [33], and should be able to capture all the dynamics of the system well. The NMPC controller block is down-sampled from 10^{-2} to $2 \cdot 10^{-1}$ seconds to emulate a realistic sampling frequency of the controller. Although higher sample time succeeds in controlling the plant, this gives a relatively large variance in the trajectories of the valves. This is probably because the pressure and GVF states have fast dynamics, and together with mismatch between the plant and the prediction model this makes

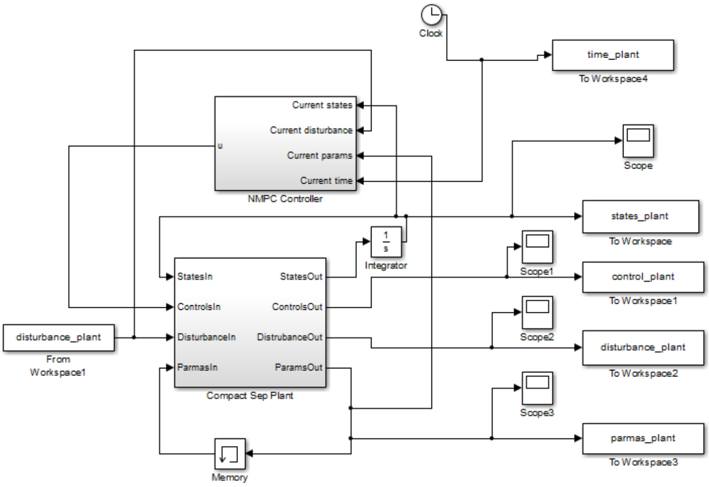


Figure 5.4: The Simulink diagram for the simulator of the Separator(Compact Sep Plant), and the NMPC in feedback(NMPC Controller).

the plant change to much from one step of the controller to the next. The chosen frequency is then a compromise between controller performance and computational consumption.

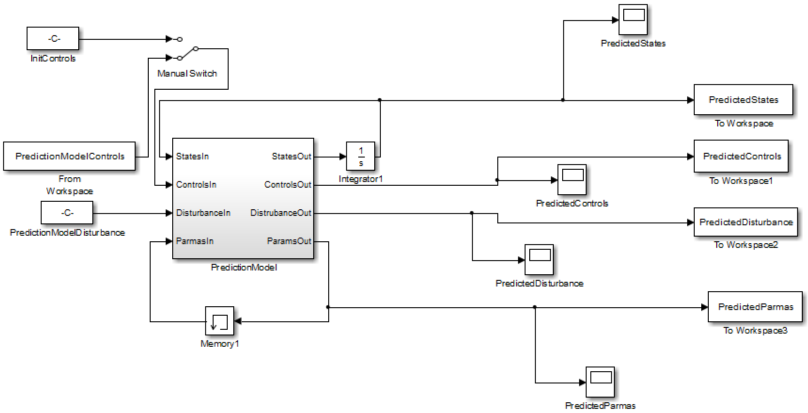


Figure 5.5: The Simulink diagram for the prediction model used by the NMPC.

Figure 5.5 shows the Simulink implementation of the prediction model used by the NMPC. Two different ODE-solvers, both from Matlab, are used to simulate this. The variable-step solver *ode23s* and the fixed-step *ode1*. The variable-step solver uses a relative and absolute tolerance of 10^{-2}, and a minimum and maximum step length of 10^{-2} and $2{\cdot}10^{-1}$ respectively. The upper limit is chosen such that the solver will be computationally faster than the fixed-step version when there is little disturbance and the system are in steady-state, while the lower limit is the same as that of the plant simulator. The solver tends towards the minimum step length during the fastest transients of the system, typically occurring during quick changes in the disturbance. In these circumstances the prediction model will be simulated at the same precision as the plant itself, thus giving an almost perfect prediction at high stiffness. It does however not become completely perfect as $ode1$ and $ode23s$ doesn't give the exact same solution at the same step length, as *ode1* is the Euler-rule, while *ode23s* is built on the Runge-Kutta methods of order 2 and 3. That this prediction becomes this good when the systems fastest dynamics are excited is not necessary realistic, however this does not matter as it is the algorithms ability to handle numerical noise in the objective function that is the purpose of the simulation. Also the tolerances of the variable-step solver are chosen rather large. This reduces the computational load during simulation, thus decreases the time consumption of the NMPC computations significantly. However because this also decreases the accuracy of the predictions, it also increases the numerical noise in the objective function, and increases the model mismatch between the plant and the prediction model.

The fixed-step solver uses a step-length of 10^{-1} seconds, to be computationally fast. As the low tolerances of the variable-step solver, this introduces a model-mismatch, which increases the realism of the simulation as model-mismatch always will be present in a real-world application.

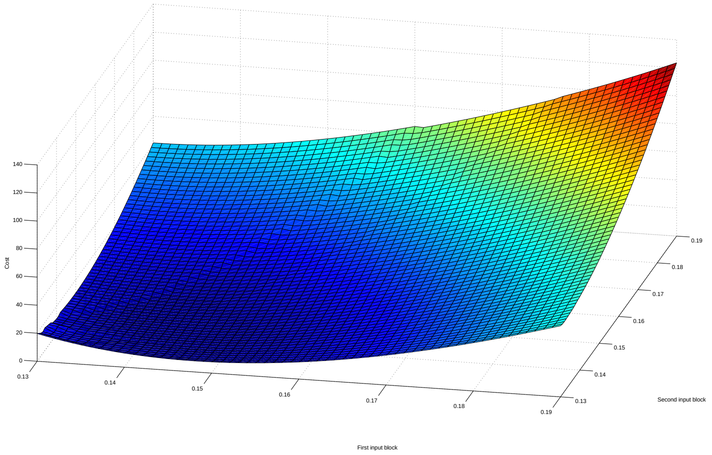
The implementation of the prediction model is done under the assumption that the current disturbance is estimated, and the most recent estimate is used in the prediction. Because the NMPC uses the whole sample period (0.2 seconds) to solve the optimization problem, the input implemented at time step t, is calculated using the disturbance at $t - 0.2$. However the real-world plant will always have a error in this estimate, which is not considered in the simulation.

5.5 Analysing the optimization problem

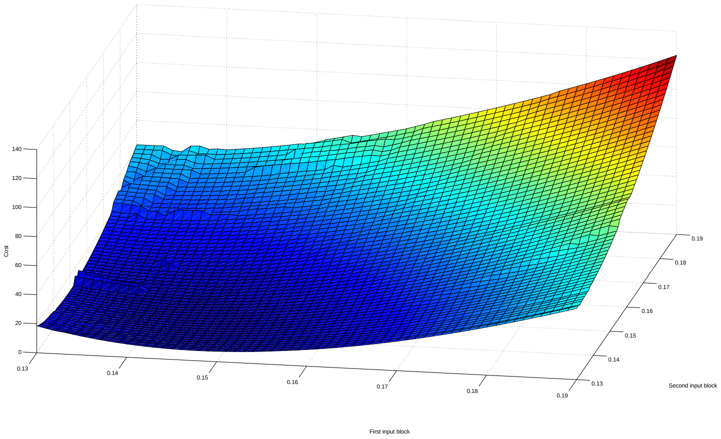
Before applying optimization algorithms to solve the NMPC problem, it is useful to investigate the properties of the problem. Two sets of inputs and measured states of the plant $x(0)$ are chosen from a simulation, one set where the plant is in a stiff condition, and one where this is not the case. Then the cost is evaluated with different inputs by holding the three first and the two last input blocks equal, varying one valve. It is the "Full MPC" approach that is investigated, as this is the numerically most challenging.

Figure 5.6(a) and 5.6(b) shows the resulting cost when applying fixed and variable-step ODE-solver respectively, in a non-stiff case. In the fixed-step case the cost surface is relatively smooth, although some minor discontinuities can be seen traversing the cost surface. Experiments show that the tested gradient-based SQP algorithm handles this problem well. When applying a variable-step solver noise appears. Local minimums seems to exist in the top left corner of figure 5.6(b), which is challenging for all algorithms.

Figure 5.7(a) and 5.7(b) is the same experiment as figure 5.6(a) and 5.6(b), however the system is in a stiff setting. It is important to notice that even when using a fixed-step solver, the problem may still become discontinuous, non-smooth and numerically challenging. Notice that at least two minimum points exist, one at the bottom of the curve, and one on the edge next to this.

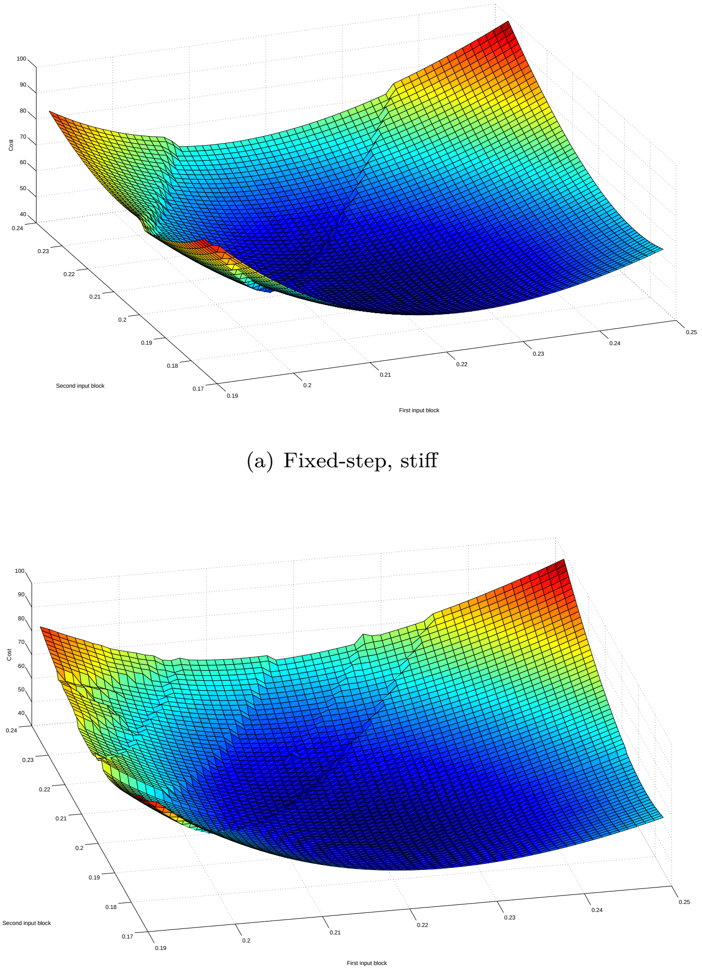


(a) Fixed-step, non-stiff



(b) Variable-step, non-stiff

Figure 5.6: Cost surface of the NMPC optimization problem when applying fixed-step and variable-step solver when the plant is in a non-stiff configuration, and varying u_7 between 0.13 to 0.19.



(a) Fixed-step, stiff

(b) Variable-step, stiff

Figure 5.7: Cost surface of the NMPC optimization problem when applying fixed-step and variable-step solver when the plant is in a stiff configuration. The first block of u_7 is varied between 0.19 and 0.25, the second between 0.17 and 0.23

These two points seem to have quite a similar function value, thus there can be argued that it does not matter which one is chosen(from an optimization point-of-view), however it is desirable that the same point is chosen at each time step and not switching between them, as this will create unnecessary variance in the inputs to the plant.
When using a variable-step solver, as in the non-stiff case, noise is introduced leading to a function that is likely to be even harder to differentiate. Also more local minimums are introduced. Applying finite-differences to this is likely to introduce large errors in the gradient.

Chapter 6

Using Wedge, Condor and BOBYQA for NMPC

This chapter will apply DFO algorithms to solve the optimization problem of section 5.2. This requires the optimization problem formulation (5.8), as non of these algorithms has the ability to handle constraints in the way they appear in the single-shooting approach to NMPC, as described in chapter 5.2. Condor and BOBYQA handles bounds on MV, however Wedge does not. Therefore these bounds are handled through penalty function for this algorithm.

Two main categories of simulations are performed on the case study, namely using NMPC to control all states("Full MPC") and where the two slowest states are pre-stabilized with PID-controllers as described in section 5.1 and 5.2. The plant is simulated as described in section 5.4, and both variable and fixed-step solver is used to simulate the prediction model, using the same settings as in section 5.4. Warm-start is also applied by always starting the optimization at the previous solution to exploit that the solution is similar from one time step to the next.

A time series of disturbances is used to test the system. This is shown in figure 6.1. It takes base in a disturbance series that [41] used for simulations, and represents the worst disturbance the real plant can be expected to undergo. However to test the performance of the algorithms under challenging dynamic circumstances, the applied series is made "worse". As can be seen from figure 6.1 the system in closed-loop with the NMPC controller will be tested both for constant, mild and severe change in the disturbance. After the last steps the controllers ability to bring the system back to steady-state is tested.

The model, both in the prediction configuration and the plant, proved to be very numerically challenging. Especially a negative rate of change in δ makes the ODE-solver occasionally return infinite states or NaN(Not a number)(i.e. undefined points), in the case of variable-step solver and "Full MPC" in combination with fixed-step solver. Investigations shows that in these cases the set of

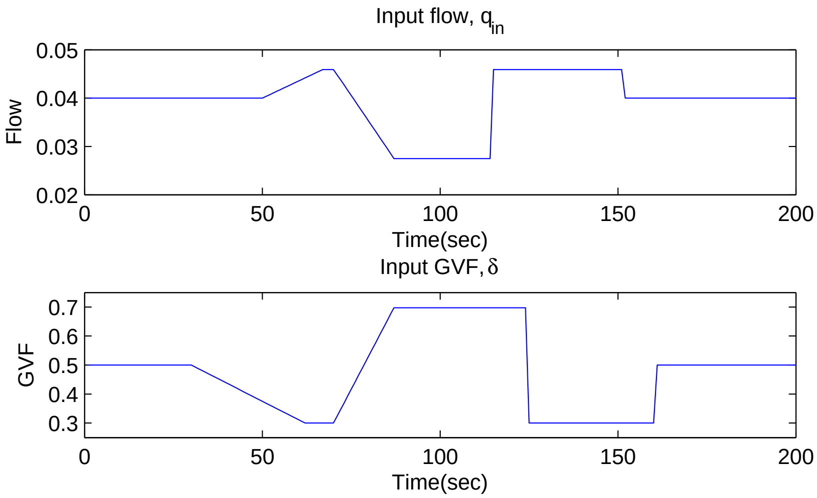


Figure 6.1: The disturbance sequence used during simulations.

inputs where the model is defined, becomes very small, particularly when using the variable-step solver. The objective function also becomes very non-smooth and quickly changing from one time step to the next, and an optimal solution in one time step may become undefined in the next. None of the algorithms handles undefined starting points. In these cases a last-resort solution must be used, by implementing the solution from the previous time step and hope that this eventually becomes defined again as the simulation progresses.

It should however be mentioned that an experiment is performed where the optimization is divided into two stages. The first stage uses the GA described in section 4.1 to find a defined starting point x_0, which is a sub-optimal solution. Then a second stage applies a DFTRM with this solution as starting point, to improve the optimality of this. This two-stage approach proves very robust, and by allocating say 40 function evaluations to the GA at each time step it not only finds a feasible starting point, but also does some good steps towards the solution such that the DFTRM converge quicker. However because of the randomness in the GA each simulation tends to result in a slightly different solution, and it is therefore hard to directly compare these. If e.g. it is desirable to use the variable-step solver with parameters that tends to result in infeasibility or simply increase the robustness of the controller, such a approach can definitely have a positive effect.

6.1 Choice of parameters for the algorithms

Tuning of the algorithms proves to be decisive on the results, and it seems that there always exist room for improvement. The maximum allowed number of function evaluations ,k_{max}, is chosen such that all algorithms gets a comparable amount of computation time. This is done under the assumption that evaluation of the objective function is significantly more computationally costly than then rest of the operations.
Further all algorithms must be supplied with the initial TR-radius Δ_0. Inspecting the algorithms presented in chapter 4, choosing this large should give a good "filtering-effect" against noise[46], as the initial sample set will be widely spread. On the other hand, when the system operates relatively steady, it will be more desirable to start with a small radius. This will make the algorithm already "zoomed" in on the solution, thus requiring fewer function evaluations to zoom on to the solution and stop at a better solution when k_{max} is reached. Thus a good choice of the TR-radius would be the distance from the current starting point to the current solution, the latter obviously unknown. Simulations proves that the change of a input is usually in the magnitude of 10^{-2} during relatively large disturbances, and usually smaller. Taking basis in this and experimenting with different choices of the initial TR radius, $\Delta_0 = 0.05$ proved to be a reasonable compromise between robustness and accuracy. This may seem large taken into account that it is very rare for a variable to change this much, however it is chosen to also ensure robustness. If the algorithm of some reason, e.g. noise, gets stuck in a local minima or is forced to revert to a last-resort solution far from the optimum, this may give an "exploration"-effect that ensures steps in the correct direction and may overcome local minimums.
During the simulations all DFO algorithms uses a minimum TR-radius, Δ_{end}, as stopping criteria in addition to a maximum number of function evaluations. This is chosen purely from simulations, to find values for each algorithm that stops the algorithm before the noise level becomes significant.

6.2 Remarks about MPC tuning

To keep the focus on the algorithms during the following results, some remarks concerning the tuning of the MPC will be stated here.
Inspecting the best found solution of figures 6.4 and 6.5, it is definitely desirable to make changes to the weight matrices if it is to be used in a real-world application. For the "Full-MPC" approach u_5 should be less aggressive, as the variations may be limited by actuator dynamics. It is experimented with larger weights in the corresponding indices in the weight matrices, however this did not improve the result. It is likely that some underlying dynamics should also be controlled, or one or more of the other controlled states should be less weighted to relax the demand on u_5.
Further more, u_c seems to get some minor oscillations when δ has a negative rate of change using the pre-stabilized plant. This is likely to have its origin in the controlled state σ and α, as these clearly has "spikes" at these points in the time series. As these two states are known to have relatively fast dynamics, the "correct" solution to this problem is probably to increase the number of input blocks, however this will increase computational load. Increasing the respective indices in the weight matrices did not give any improvement, however it seems reasonable that less weight on the error in σ and α may make the controller less aggressive. It should also be mentioned that this tendency greatly improves as the control horizon is extended, and choosing a horizon of 5 seconds makes the oscillations severe. Also more active use of terminal weights, or even using a barrier-function to create terminal constraints may create less aggressive behaviour[39]. Note that as discussed initially the objective function is known to be numerically difficult in the case of negative rate of change in δ, thus the issue may not be easy to solve only by tuning the controller.
Because the focus of this report is numerical optimization rather than tuning of MPC, limited time is spent on this issue. It may also be argued that is interesting to see how the algorithms performs on a sub-optimally tuned NMPC problem.

Algorithm	MPC	Fs/Vs	W/C sol.	Av. sol.	Av. evals.	S.S. evals.
SQP	P.S.	Fs	176.7	5.9	178.6	93
BOBYQA	P.S.	Fs	10.2	1.6	117	80
Condor	P.S.	Fs	25.5	2.2	247	252
Wedge	P.S.	Fs	16.9	2.5	168	143
BOBYQA	P.S.	Vs	28.7	2.7	80	71
Condor	P.S.	Vs	11.6	1.2	245	249
Wedge	P.S.	Vs	89.5	3.9	164	151
Wedge*	P.S.	Vs	53.7	1.1	197	150
SQP	Full	Fs	194	38.6	101	74
BOBYQA	Full	Fs	119.6	3.4	213	212
Condor	Full	Fs	42.9	7.2	463	456
Wedge	Full	Fs	140.3	8.7	389	382

Table 6.1: Summary of the results. "W/C sol." is the highest cost the respective algorithm finds during simulation, while "Av. sol." is the average cost. "Av. evals. and "S.S. evals." is the average number of function evaluations used during the whole simulations, and during the first 30 seconds while the disturbance is constant, respectively. Wedge* $\gamma_1 = 0.75$ instead of 0.5.

6.3 Summary of the results

Table 6.1 summarize the results for the four algorithms that is tested. "Fs/Vs" refers to the method used to simulate the prediction model, fixed-step or variable step respectively. "MPC" is pre-stabilized(P.S.) or Full. "W/C sol." and "Av. sol" refers to the highest and the average cost of the solutions found, while "Av. evals" is the average number of function evaluations used. The highest number of function evaluations is the same as the maximum allowed number for the respective algorithm, thus stated in the section for the algorithm. "S.S. evals" is the average number of function evaluations used from the start of the time series under steady-state conditions until the disturbance starts to change at 30 seconds. In the figures showing the resulting costs when the NMPC is simulated, it is important to note that the pre-stabilized plant does not weight error on the liquid levels and deviation from ideal values on two of the inputs, as these are not controlled by the NMPC. Therefore the resulting cost of the pre-stabilized and "Full MPC" is not directly comparable. These are plotted in the same plot merely to save space.

6.4 Gradient-based SQP

For comparison, the finite-differences, gradient-based SQP algorithm from the "fmincon()" method in the optimization-toolbox in Matlab, is applied to both the pre-stabilized and the "Full MPC" approach. The algorithm is allocated $k_{max} = 400$ function evaluations, and stopped if the step size is less than a magnitude of 10^{-5}. It is however only applied using fixed-step solver for simulation of the prediction model, as variable-step solvers makes the algorithm do unrecoverable errors. It should be noted that this SQP-implementation is known to not be state of the art, however it gives an indication of how gradient-based algorithms handles this NMPC problem.
Several times the algorithm seem to suffer from discontinuities and non-smoothness in the objective function, returning costs in the magnitude of several thousand. For the algorithm to be applicable for the whole time series, the algorithm is restarted with x_0 chosen to the steady-state values of the inputs presented in table B.1 if the solution found using the solution from the previous time step is greater than 500. If still no good solution is found, the solution from the previous time step is used as the control input. This procedures are used quite often, indicated by the peaks in cost in figure 6.2. Note that the last-resort procedure described initially in this chapter to handle undefined initial points is also implemented, thus the algorithm has two different sort of "failures" to be handled; wrong solution due to errors in the differentiation and lack of solution due to undefined point in the model.
Figure 6.2 and 6.3 shows the resulting control inputs and states respectively. Clearly SQP suffers greatly from the discontinuities. This is reflected in table 6.1, as this algorithm has the highest costs of all tested. Just as important however is the number of function evaluations. The average number of function evaluations used is actually higher for the pre-stabilized case, may suggest that the "Full MPC" approach has some numerical difficulties. If the number of evaluations are reasonably high, this may suggest that the algorithm has succeeded. The low number of evaluations for the "Full MPC" can be interpreted as local minimums trapping the algorithm, or discontinuities simply making the algorithm fail.
When inspecting the figures this is partly confirmed, as the cost for the "Full MPC" is extremely high over long periods of time as the fall-back solution is applied. For the pre-stabilized case it is, at least while the disturbance is moderate, similar to that found by the other algorithms. This shows how the pre-stabilized case is easier to optimize than the "Full MPC", as it is less contaminated with discontinuities and non-smoothness, as well as undefined points.

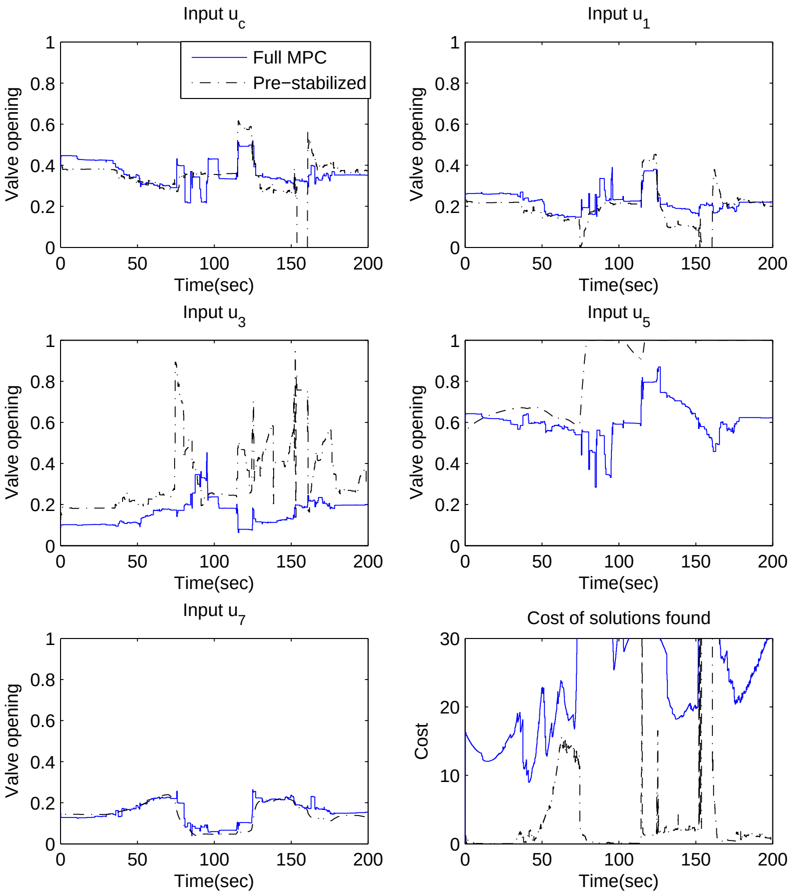


Figure 6.2: Control inputs using SQP, and prediction model simulated with fixed-step solver.

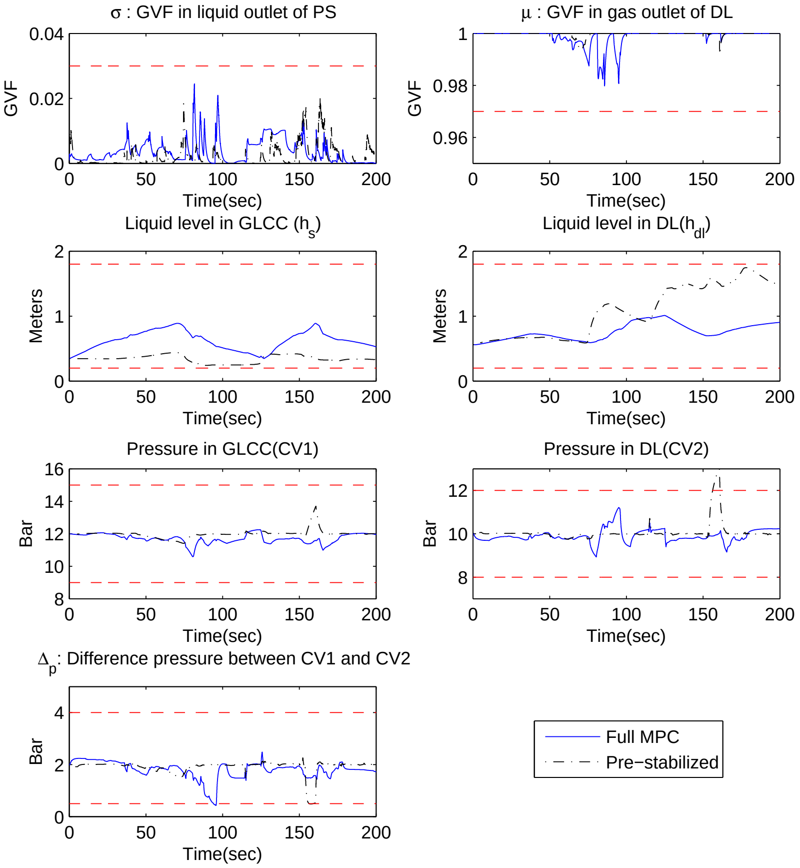


Figure 6.3: Controlled states using SQP, and prediction model simulated with fixed-step solver.

6.5 BOBYQA

The stop criteria used for BOBYQA are $\Delta_{end} = 10^{-3}$ and $k_{max} = 216$ for the pre-stabilized plant and $k_{max} = 600$ for the "Full MPC".
Note that it is not the minimum TR-radius that is set in BOBYQA, but the minimum step-size. This is due to the specific implementation of BOBYQA used, namely in the NLOpt package[28]. The two types of stopping criteria are closely related, however the one used in BOBYQA will stop when a solution such that the change in one of the variables from the current iterate is less than Δ_{end}. This however seems to work very well, as the algorithm adjusts nicely the number of iterations necessary.
These values are chosen partly from experiments, and from the suggestion of [52] of optimizing a problem in $\frac{1}{2}n^2$ function evaluations. However it turns out that this limit makes the algorithm perform poorly when the change in the objective is severe. Thus the stated limit is chosen, and a relatively high Δ_{end} such that the algorithm terminates after fewer iterations if convergence is achieved quickly due to small change in the objective function from the previous time step.
Some unexpected behaviour is observed with BOBYQA. Now and then it returns a solution that is worse than its starting point x_0. This seems counter-intuitive, however inspecting the initialization-procedure of section 4.6, it turns out that if the given starting point is close to the bounds, it will automatically be moved. To avoid this, the cost of the starting point is evaluated before BOBYQA is started, and compared with the returned solution when the algorithm exits, then choosing the best solution.

Figure 6.4 and 6.5 shows the control inputs and states using BOBYQA both for pre-stabilized and "Full MPC".

The most interesting result from using the fixed-step ODE-solver with BOBYQA is that it has the lowest number of function evaluations used, and still the solutions found are satisfactory. For the pre-stabilized plant with fixed-step solver it is both the best average and worst-case solution. Table 6.1 shows that as long as the disturbance is small, thus the objective function relatively smooth and similar from one time step to the next, the number of function evaluations increases slowly.
Inspection of the figures and comparison with other algorithms also indicates that the solution found by BOBYQA in the "Full MPC" case is good as long as the disturbance is moderate, thus the objective function is relatively smooth. This is also backed up by table 6.1, where BOBYQA has the lowest average cost, however the worst-case solution suffers, as this case occurs during the most se-

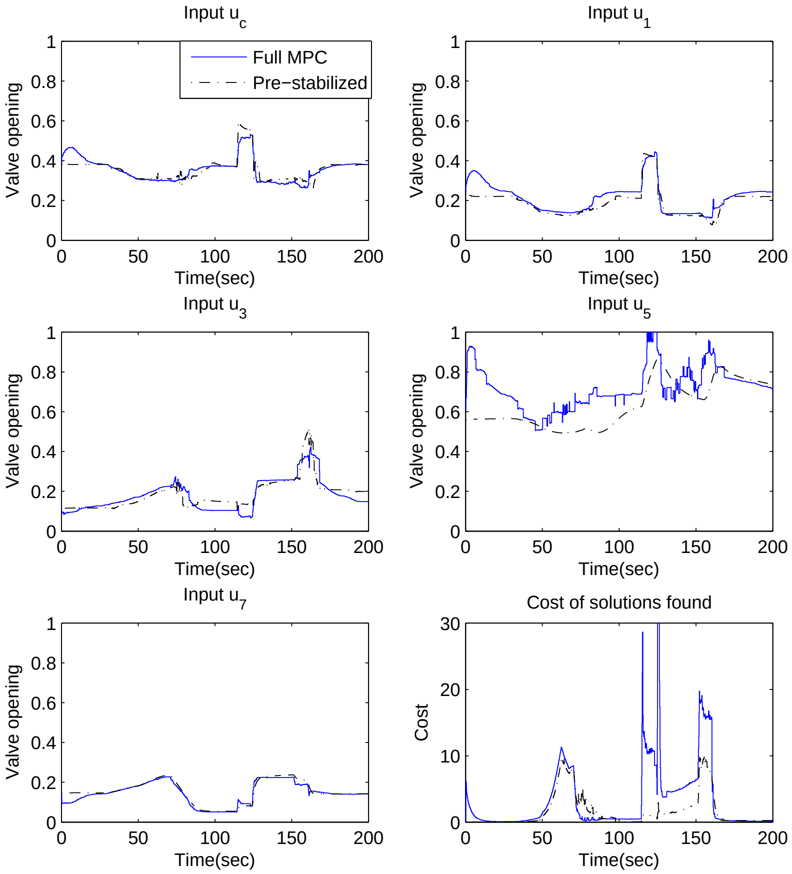


Figure 6.4: Control inputs using BOBYQA, and prediction model simulated with fixed-step solver.

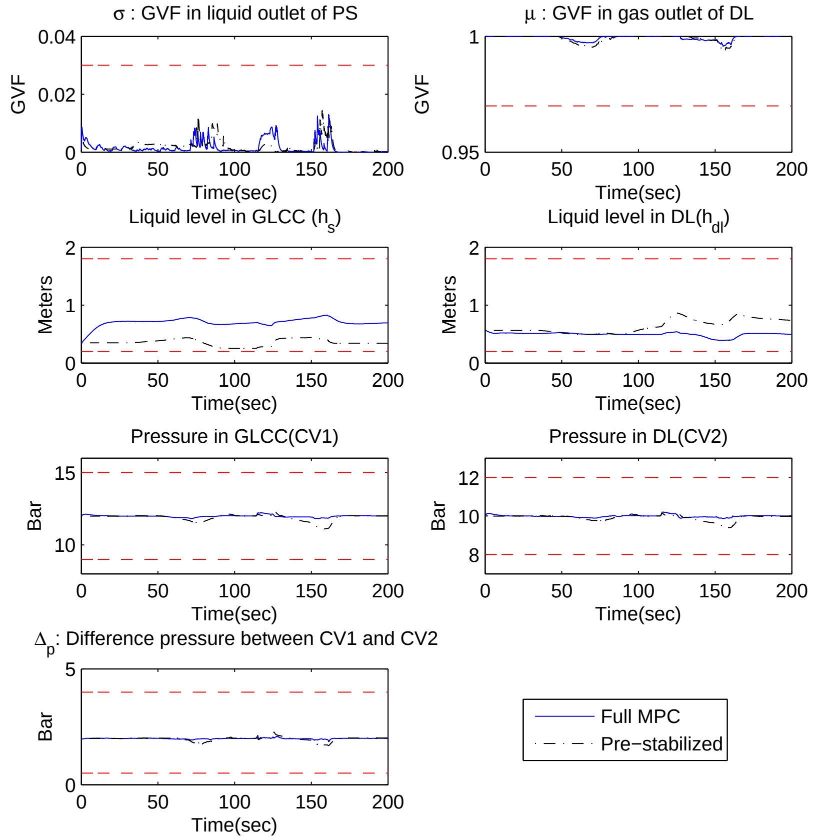


Figure 6.5: Controlled states using BOBYQA, and prediction model simulated with fixed-step solver.

vere disturbance. Is is plausible that the cause of this can be traced back to the underdetermined interpolation in the algorithm. It is known from [52] and [50] that it is common for the difference between the Hessian of $f(x)$ and that of $q(x)$ to be relatively large. The success of the algorithm during little numerical issues may then be interpreted as although the mismatch between $f(x)$ and $q(x)$ can be substantial, the minimum point is similar. It is known that $f(x)$ is at least to some extent convex close to the solution, and this is usually quite close to the starting point for the optimization, x_0. As this point is used as a point in the interpolation in the initial model, it is reasonable to believe that $q(x)$ is accurate around the actual solution. Also because Δ_0 is chosen relatively large, all other points in the initial sample set will have a larger function value due to the convex tendencies of $f(x)$. As the Hessian of $q(x)$ is a diagonal matrix initially, this will then become positive definite if these assumption holds, and the algorithm will initially make steps towards the actual solution even though the model mismatch is large. When more numerical noise and discontinuities are introduced, as is the case when the disturbance is severe in the "Full MPC" approach, these assumptions may not hold, and the algorithm may take steps in bad directions, e.g. due to a negative-definite Hessian.
Further, as the initial model is built up from fewer samples than is the case with fully determined interpolation, less information is also stored. As the algorithm progresses, fewer iterations will be needed before the model is renewed by "fresh" sample points. This can give a better ability for the model to adapt to local non-linearities as the trust-region moves.
These are only speculations, however it would definitely be interesting to monitor the accuracy of $q(x)$ and the TR-steps made by the algorithm to try to determine what is the origin of the effectiveness of BOBYQA. Such findings may then be exploited to make the algorithm even better for NMPC, or may give good guidelines in how a NMPC problem should be designed to make BOBYQA and possibly other DFO algorithms effective on such problems.

Inspecting the result using the variable-step solver in figure 6.6 and 6.7, it is clear that this has a big impact on BOBYQAs performance, supporting the previous claims. Considering the worst-case cost, this peaks at an amplitude of 28.7 instead of 10.2 in the case of fixed-step solver. The origin of these are from the NLOpt-implementation of BOBYQA. If the objective at any point returns an error, which occasionally happens due to the model, it will immediately exit, returning the current best cost. In these simulations the algorithm did no progress at these points, only returning the start-point. However as the simulation progressed using the last-resort solution, the state of the plant altered

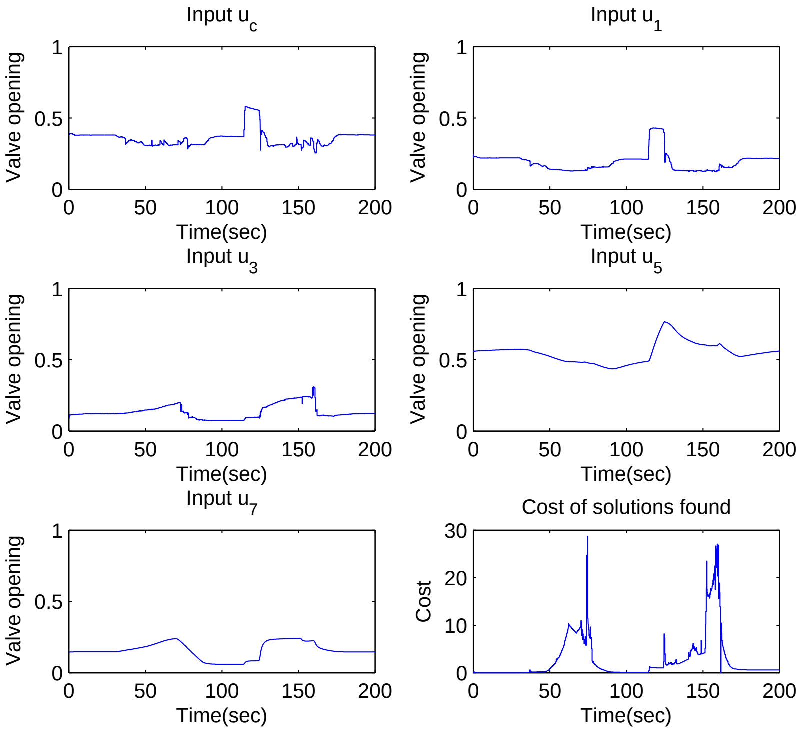


Figure 6.6: Control inputs using BOBYQA, and prediction model simulated with variable-step solver.

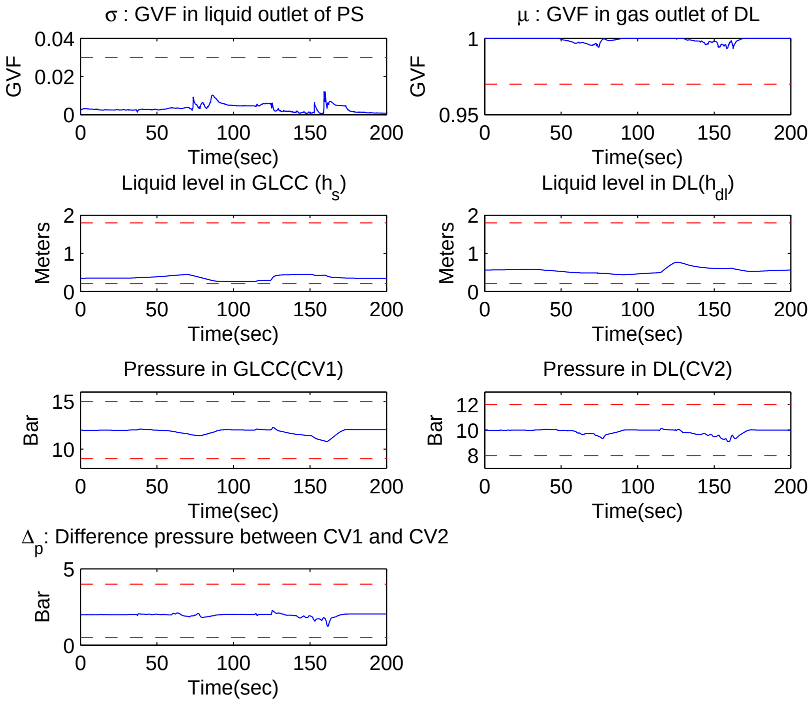


Figure 6.7: Controlled states using BOBYQA, and prediction model simulated with variable-step solver.

and the starting point became defined again.
Also noteworthy is that the average number of function evaluations is equal for the steady-state as the whole simulation in this case. This suggests that the variable-step solver generates numerical noise that makes the algorithm quickly reduce its local TR, ρ, because non-smoothness in $f(x)$ will induce model mismatch between this and the smooth $q(x)$. This makes the algorithm do small steps, and the minimum allowed step size may be reached quickly.

6.6 Condor

Condor is constrained to 100 iterations after the initial model building, and $\Delta_{end} = 10^{-3}$. This typically results in a total of approximately 250 function evaluations for the pre-stabilized plant and approximately 450 evaluations for the "Full-MPC", as model improvement may consume more than one function evaluations for one iteration. As BOBYQA is given $k_{max} = 600$ for the "Full MPC", this may seem unfair. However Condor rarely stops due to Δ_{end} is reached, and usually uses all of its allowed iterations. This is likely because of extensive model improvement, to be robust against noise. As the algorithm uses a full sample set, it is intuitive that it would perform better than BOBYQA as Condor may have a better model of $f(x)$ initially, especially if significant off-diagonal elements exist in $\nabla^2 f(x)$.

Figure 6.8 and 6.9 shows the control input and controlled states when Condor is applied to the fixed-step cases. These figures suggests that for the pre-stabilized plant, the solutions with Condor and BOBYQA are similar. Condor tends to have slightly higher spikes during the negative transients of δ. However during the "Full-MPC" approach Condor tends to not be able to minimize the objective function quite as good as BOBYQA, especially during moderate disturbances. Around 150 seconds, where the algorithms seems to struggle the most due to discontinuities, Condor performs better.

Condors ability to handle noise is confirmed by simulations with a variable-step ODE-solver, shown in figure 6.10 and 6.11. Inspecting the cost of the solutions found, Condor finds solution that is at least of a factor of two smaller than that of BOBYQA under similar circumstances. Also note from table 6.1 that the cost in the fixed-step and variable-step cases are almost similar, suggesting that Condor is little affected by numerical noise.
Due to the implementation of Condor, the algorithm will in contrast to BOBYQA exit and stop the simulation if the initial point is undefined, or the sample set around the initial point cannot be formed within a reasonable radius due to

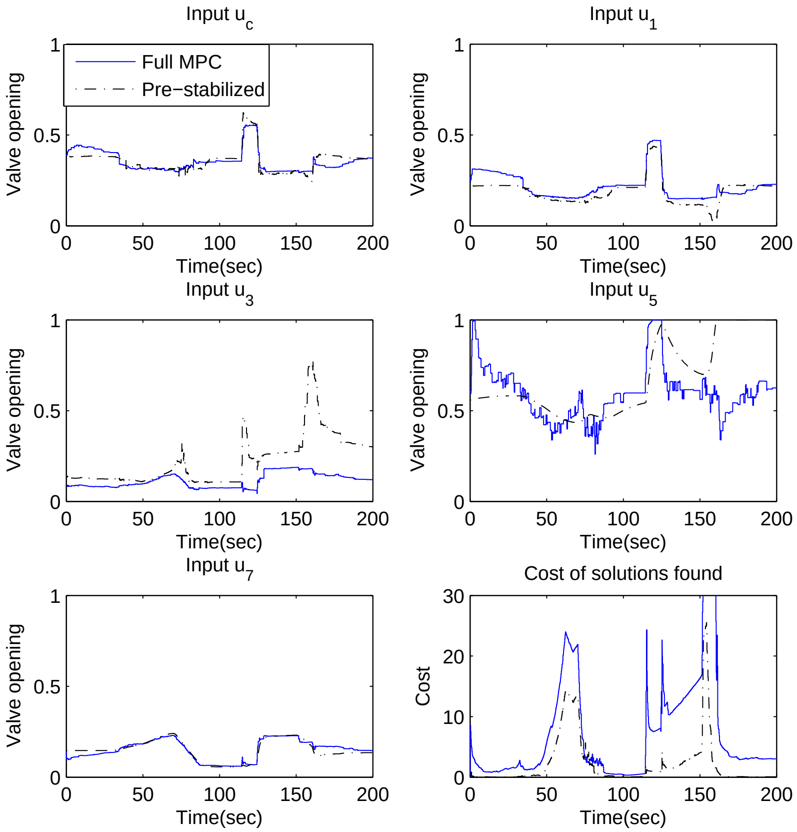


Figure 6.8: Control inputs using Condor, and prediction model simulated with fixed-step solver.

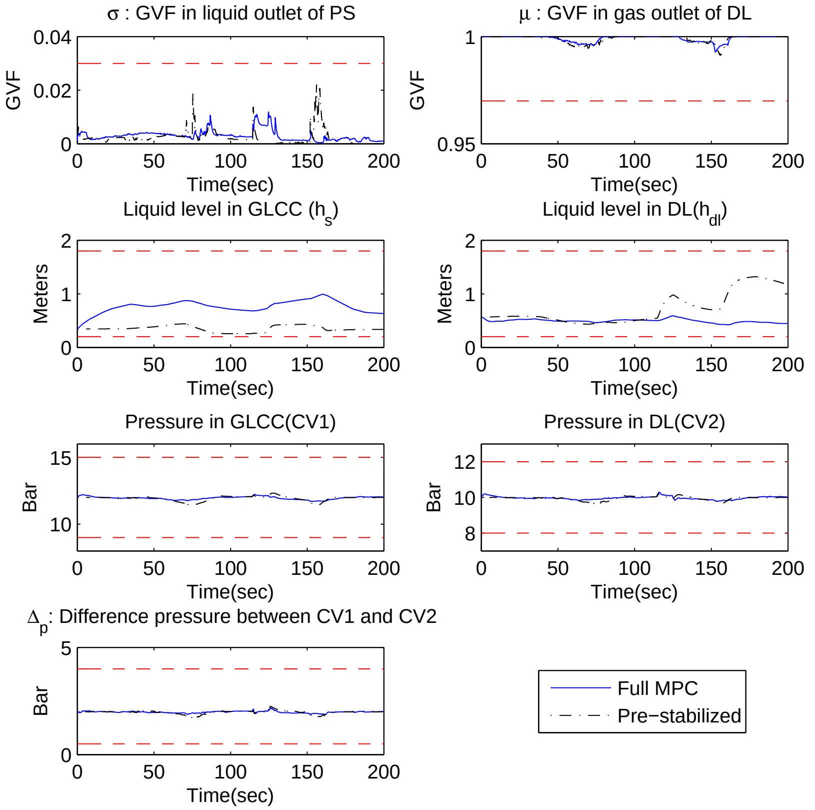


Figure 6.9: Controlled states using Condor, and prediction model simulated with fixed-step solver.

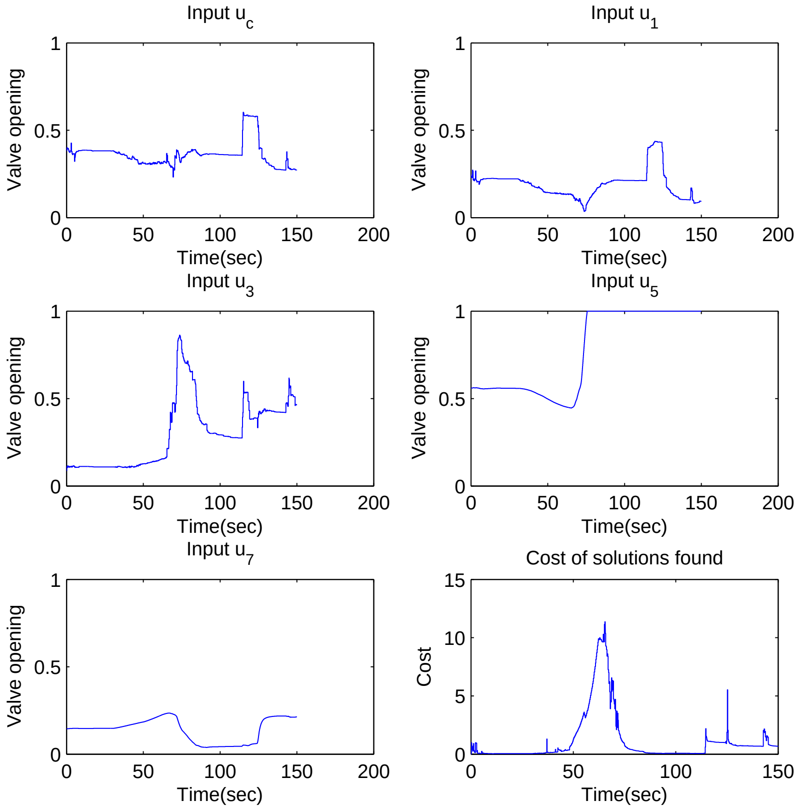


Figure 6.10: Control inputs using Condor, and prediction model simulated with variable-step solver.

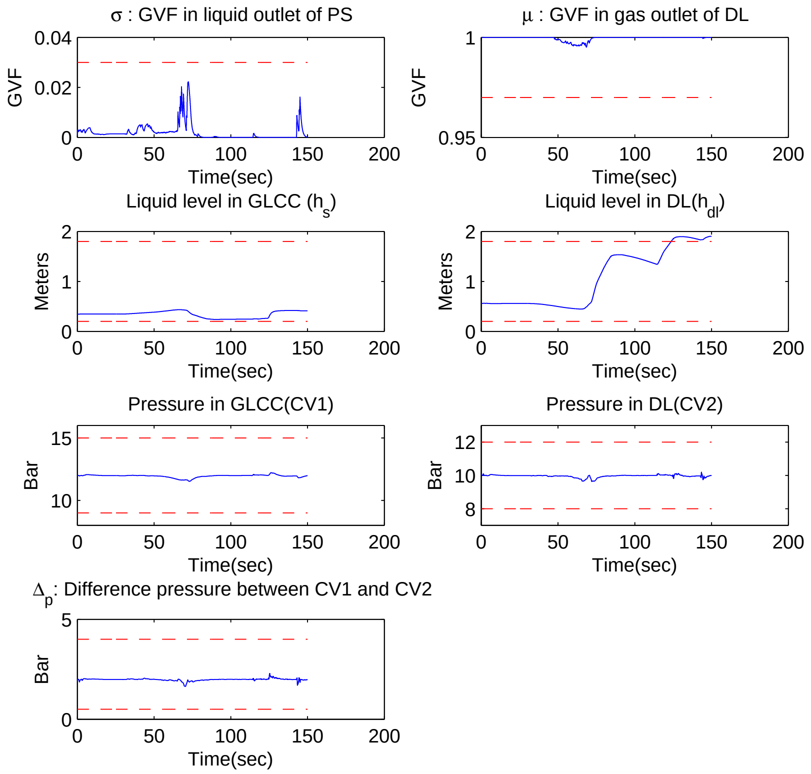


Figure 6.11: Controlled states using Condor, and prediction model simulated with variable-step solver.

hidden constraints. Therefore the simulations did not get past the step in δ, in contrast to BOBYQA that just exits and implements the last solution in these cases.

Compared to the case with the fixed-step solver, it can be noted that h_{dl} exceeds its limit, thus u_5 saturating. This state and input are controlled by other internal controllers, however the difference in the behaviour is noteworthy.

6.7 Wedge

As seen in section 4.4, Wedge has an extensive number of tuning parameters. This makes it possible to tweak the algorithm to fit the problem at hand in a best possible manner, however it also makes the task to find the optimal set of parameters significantly more difficult. The parameters used are stated in table 6.2. In addition the stopping criteria are set to $\Delta_{end} = 10^{-3}$, $k_{max} = 461$ for the "Full MPC" approach and $k_{max} = 246$ for the pre-stabilized, to be comparable to Condor. This is in contrast to Condor constrained on the number of function evaluations, and not iterations.

From the paper presenting Wedge[36] it is clear that the purpose of the algorithm is not to optimize noisy functions. On the other hand, all DFO algorithms tends to be more suited to noisy optimization than gradient-based algorithms. However Wedge should not be expected to handle noise as well as Condor that has an explicit handling of noise implemented and is developed for noisy optimization[6].

Variable	Value
Δ_0	0.05
γ_0	0.4
γ_1	0.5
γ_2	2
μ	0.5

Table 6.2: Parameters used in the simulations of Wedge.

Figure 6.12 and 6.13 shows the resulting control inputs and states when the fixed-step solver is applied. Compared to Condor, Wedge seems to perform equally or better when the disturbance is moderate. Inspecting the cost from 0 to 50 seconds, it seems to be lower or equal, especially for the "Full MPC" approach. It is also important to note that the algorithm uses significantly fewer evaluations than Condor, as it tends to reach Δ_{end} quicker. Also note that the variance in u_5 seems to be smaller with Wedge than Condor. Although desirable, this "filtering" is actually a side-effect when the algorithm does not find the exact solution. The solution will become somewhat more similar to the previous which is the current start point, thus less variance will occur.

As expected the performance of Wedge suffers when the disturbance and thus

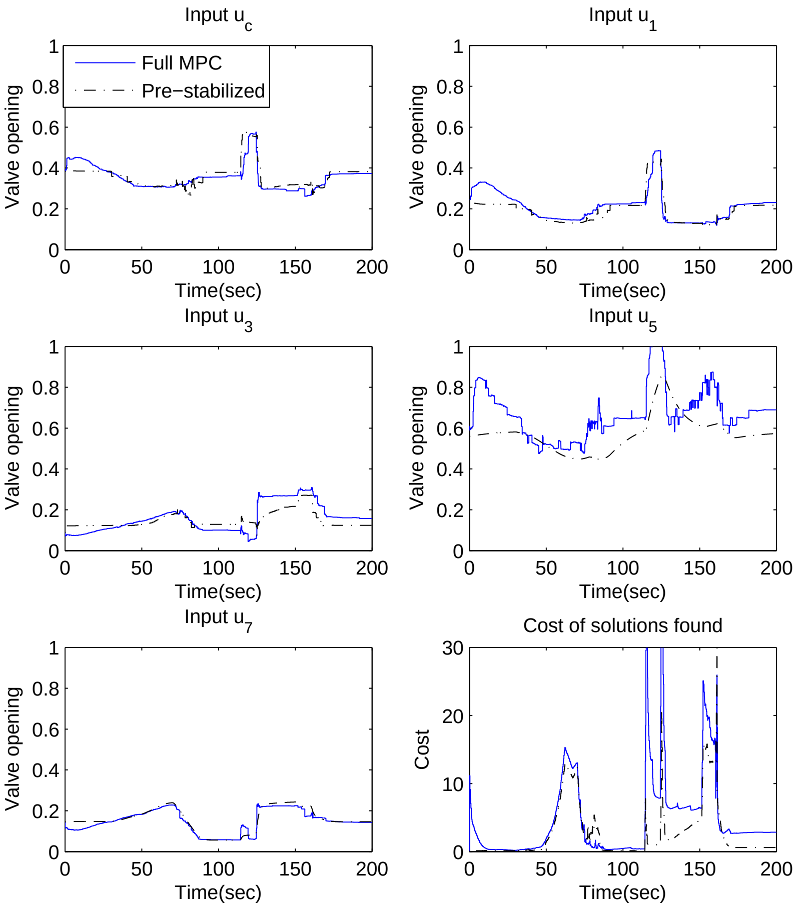


Figure 6.12: Control inputs using Wedge, and prediction model simulated with fixed-step solver.

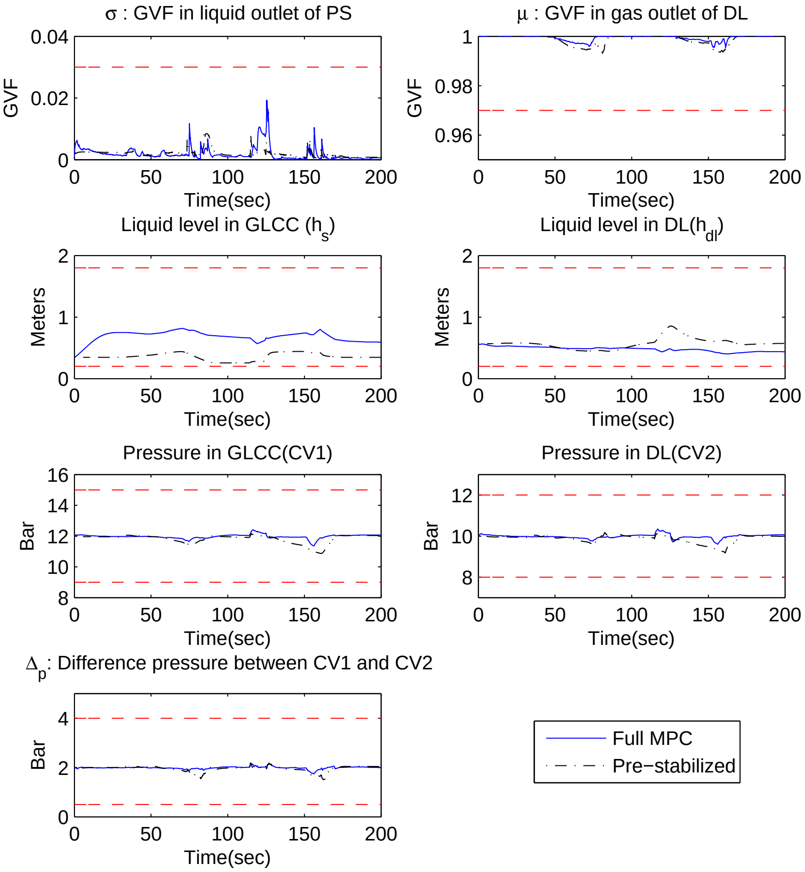


Figure 6.13: Controlled states using Wedge, and prediction model simulated with fixed-step solver.

the non-smoothness becomes severe, especially clear from the worst-case solution in the "Full MPC" approach. This is likely because of the simple TR update, i.e. point 6 of algorithm 8 which incorporates only one TR. When the disturbance is severe, discontinuities can be introduced, as the system becomes stiff due to exciting fast dynamics. The difference between $f(x)$ and $q(x)$ is thus likely to be relatively large. The TR will typically be reduced to compensate for this. In the case of Condor and BOBYQA this may only reduce the local trust region ρ, and allow a larger step in the next iteration if a model improvement has occurred. However in Wedge it is the global TR that is reduced, and in the next iteration the TR-step will be constrained by the reduced TR. The algorithm also recovers quickly after the peaks in cost during the largest steps in the disturbance, as the algorithm restarts with Δ_0.

Because Wedge is a Matlab-implementation, it is easier to investigate the behaviour of the algorithm. Firstly, the poisedness-improving mechanism is rarely invoked, especially when good progress is achieved such that trial-points are relatively far from each other. When the algorithm starts close to the solution, trial-points are closer thus the wedge-constraint becomes active more often. This is intuitive and according to [16].
Further it seems that the Hessian of $q(x)$ is rarely positive-definite, the reason for this is unknown as this would be expected at least under steady-state operation(see figure 5.6(a)). Also the absolute value of the gradient of $q(x)$ does not decrease as the algorithm progresses. This may suggest that the model is not very accurate or the function evaluations are noisy, as the gradient would be expected to decrease when approaching the solution. It would definitely be interesting to inspect the approximation models of BOBYQA and Condor as well, to see if the tendencies also occur in these.

Figure 6.14 and 6.15 shows the control inputs and states when Wedge is applied when simulating the prediction model using the variable-step solver. Its performance up to around 110 seconds is largely the same as BOBYQA, continuing the tendency that Wedge handles moderate disturbances the best. As the algorithm progresses, it often tends to terminate early after very few iterations due to the Δ_{end} stopping criteria. This can be seen clearly in table 6.1, as the algorithm does not have a good average solution as long as $\gamma_1 = 0.5$. The worst case-solution is also high, however this is because the algorithm encounters undefined points when the peaks in the cost occur in figure 6.14. If better handling of this is implemented, the worst-case is likely to be lower.
The choice of γ_1 and Δ_0 proved to be of great significance. Choosing the latter larger clearly gives an improvement in worst-case performance during transients,

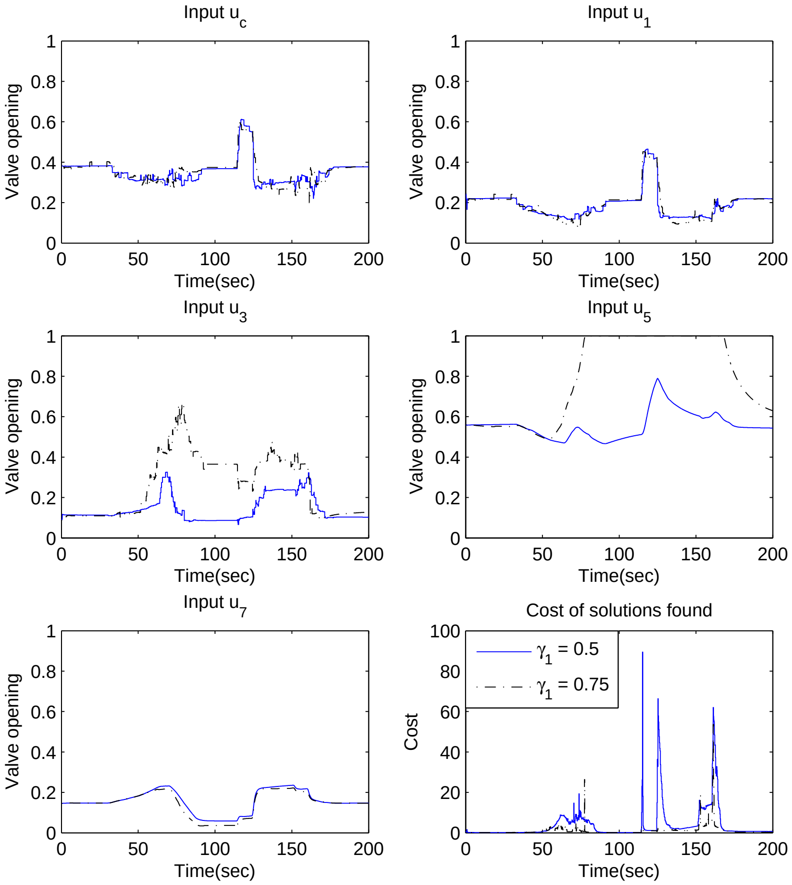


Figure 6.14: Control inputs using Wedge, and prediction model simulated with variable-step solver. Note that $\Delta_0 = 0.075$.

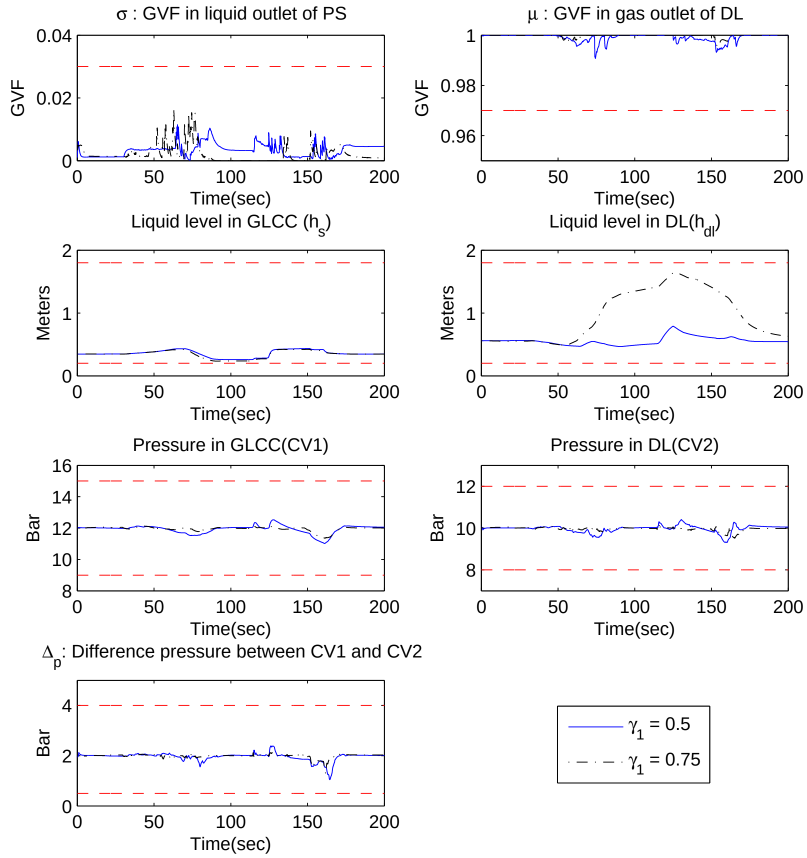


Figure 6.15: Controlled states using Wedge, and prediction model simulated with variable-step solver. Note that $\Delta_0 = 0.075$.

however compromises steady-state performance. On the other hand choosing Δ_0 small gives a significantly larger amplitude in the worst-case cost, however the performance up to 50 seconds is improved.

This is likely because the algorithm becomes more robust towards change in the objective function as Δ_0 increases, as the initial sample set will have a larger radius. Thus $q(x)$ is valid over a larger set, more likely to include the actual solution. In addition it seems intuitive that a large Δ_0 also will give a smoothing effect against noise in the initial steps. However because the maximum number of function evaluations is reached before the algorithm has converged a less exact solution will be found. On the other hand, choosing Δ_0 small works well as long as the objective function does not change significantly from one time step to the next, as the initial starting point will be close to the solution and the algorithm can "zoom" in on the exact solution.

γ_1 seems to have a similar effect, and proves to be closely related with the algorithms robustness against noise. Increasing this gives the best average cost on the variable-step case of all the algorithms, as well as an significant improvement in the worst-case. However as is clear from figure 6.14, the variance in the input sequence increases. This is likely because an increase in γ_1 allows for more iterations before reaching k_{max}, thus better "exploration" when noise is present and the optimum is far from x_0. However when x_0 is close to the solution and the objective function is relatively smooth, more iterations are needed for the sample set to be zoomed in on the solution. As the algorithm still stops due to Δ_{end} stopping criteria, it is not intuitive that variance in the inputs occurs as the sample set should be zoomed in on the solution. An explanation may be that local minimums exist, and because the TR is reduced more slowly, the algorithm of some reason is more likely to get stuck in these.

6.8 Remarks on the result

There is no clear winner among the algorithms, as all seem to have their strong and weak properties. An important observation is that there are large differences in their ability to find an optimum, or even near-optimal solution within the tolerances and iteration limits. It is interesting that the gradient-based algorithm do fail due to non-smoothness and makes large steps in bad directions, both when fixed and variable-step ODE solvers are applied. All DFO algorithms handles this well, and although some are more sensitive due to numerical noise and discontinuities than others, all of these algorithms are significantly more tolerant than the tested SQP against such issues. The simulations shows that the

choices of tuning parameters of the algorithms is of great significance, as shown with Wedge where the worst-case performance greatly improves by increasing the reduction factor of the TR, however this resulted in increased variances of the control inputs and worse performance during steady-state. Most surprising is probably the performance of BOBYQA, which is able to come up with good solutions with a very limiting number of function evaluations. Comparing to the gradient-based SQP algorithm, BOBYQA actually seems directly comparable in performance. Condor stands out as the best suited against noise and discontinuities, induced when using the variable-step solver, and the fast dynamics are excited by large disturbances. The algorithm seems almost unaffected in these cases.
This may be traced back to the model-improvement steps in Condor. While BOBYQA only uses poisedness-based techniques and the rules for reduction of the TR, (4.15)-(4.16), to ensure model accuracy, Condor also weights the difference between the model and the actual objective function using (4.17)-(4.19). If the accuracy is poor, Condor may spend more evaluations on model-improvement than BOBYQA. It should also be noted that during these transients, Condor never exceeds 461 function evaluations, as BOBYQA uses all of its 600 allowed iterations. This suggests that Condor may be more well suited for handling problems where noise and discontinues typically arises, as when stiff dynamics are excited and the ODE solver makes larger errors. On the other hand, when the disturbances are moderate and the objective is relatively smooth, Condor may seem to spend unnecessary many evaluations on model improvement. In these cases BOBYQA and Wedge are more suited, as these algorithms has a better ability to terminate after fewer iteration when a sufficiently good solution is found. Further, in the case of a real-time application such as MPC, the worst-case performance is of primary significance as computational resources must be reserved for this case. Considering this, Condor will have an advantage, as a smaller number of evaluations must be reserved for this algorithm compared to BOBYQA.
Wedge seem to handle moderate noise well, as it has the best average solution when choosing $\gamma_1 = 0.75$. However the algorithm does not handle severe noise well, as the highest worst-case cost of all the DFO algorithms occurs using Wedge. It is in this case important to note that Wedge uses fewer evaluations than Condor, and using stopping criteria allowing for more function evaluations, or another TR-update rule may improve the worst-case performance of Wedge.

On the issue of model accuracy in Wedge, discussed in section 6.7, it can be questioned if this is due to the lack of a second sub-problem to improve poisedness of the sample set in Wedge. However [16] has suggested that DFTRM is quite robust against non-poised sample sets, and good results is presented without any poisedness-ensuring mechanism. Further the Wedge-constraint (4.13) is rarely active during simulations, and BOBYQA is known to often work with a Hessian of $q(x)$ that has a large difference to that of $f(x)$, and still perform well[52]. This together with the observed change in the performance of Wedge when changing the factor used in the TR update, γ_1, it is more likely that it is the Δ-update rules of step 6 in algorithm 8 that is not very robust against large discontinuities and non-smoothness. Mismatch between $f(x)$ and $q(x)$ due to numerical issues will reduce the single TR, and the new sample points will become close.

The computational consumption of the simulations in this chapter is much more than what can be accepted in a real-time for the case-study. This is because of the required amount of function evaluations, and that these requires a relatively large amount of resources. Especially the use of the algorithms directly on the optimization problem requires that a new sample-set will be generated each time the controller is run(i.e. each 0.2 seconds in the case-study). In the case of Condor and Wedge, this clearly is a challenge, as the work needed to build the initial model becomes $\mathcal{O}(n^4)$[47]. With the choice of number of input blocks being 5 and for "Full MPC", this gives a sample set of 351 samples in the case study.

In this thesis, all simulations are performed on one CPU/core. This raises an significant question, as although BOBYQA is the clear winner with respect to the number of function evaluations used, few of this are evaluated in the initial sample set. If evaluations is to be spread over several CPUs, it is this initial work that is the most suited for distributed processing. If using, say 16 CPUs which may be accomplished in a Digital Signal Processor (DSP), the algorithms using fully determined sample sets may become the fastest with respect to computation time. Considering that Wedge and Condor requires approximately 110 evaluations after the initial sample set building, all the initial 351 evaluations may be spread across 16 CPUs. This gives a processing time consumption similar to $\frac{351}{16} + 110 = 132$ evaluations, while BOBYQA using an initial sample set of 51 samples and a total of 212 evaluations requires a time similar to $\frac{51}{16} + 161 = 202$ evaluations giving Wedge and Condor a clear advantage. When also considering the remark above that computational resources must be reserved for the worst-case, this will give Condor and Wedge an even larger advantage, as BOBYQA in this case requires 600 function evaluations. By also considering using a multi-core GPU for parallel computing as shown in

[54], the computational time required to evaluate the initial sample set can be even more reduced.
Further reduction of this can be accomplished by using other simulation software, as using a C++-based simulation utility, and compile this into a mex-file. Because the prediction model currently is implemented in Simulink, the model is compiled each time it is executed, i.e. each function evaluation, which is unnecessary time consuming.

The NLOpt-package containing BOBYQA is a C++-implementation, as is also the Condor algorithm. As many industrial control systems is based on this programming language, the path to using BOBYQA and Condor in such a system is shorter than Wedge, which will have to be converted from Matlab to a more appropriate language.

Based on this, it currently seems that BOBYQA and Condor are two algorithms that may be interesting for use in NMPC. For the case where the numerical noise and discontinuities are not severe, however enough to make gradient-based algorithms fail, BOBYQA seems like a very promising candidate. It requires a low number of function evaluations that is comparable with the gradient-based SQP tested, thus is especially attractive for use where only a single core is available. If the numerical issues are severe, Condor is attractive. It usually requires more function evaluations, but are significantly more robust against such numerical difficulties. As it is more suited for parallelism, and already have a parallelisation mechanism implemented, it is also interesting for use on multi-core systems. The clearly biggest obstacle for using DFO in NMPC, and for NMPC in general, is real-time performance. A general characteristic of a MPC problem is that the objective function is relatively similar from one time step to the next. This is really unexploited information in the way the algorithms are used above, other than the use of warm start by starting the optimization at the solution from the previous time step. To take this warm-start philosophy further, it seems useful to somehow recycle the model from the previous iteration to the next, only re-evaluating a fraction of the sample points. A discussion of how such an algorithm can be achieved, and an experimental simulation, will be presented in the next chapter.

Chapter 7

Improved warm-start

One of the biggest obstacles in NMPC is to solve the optimization problem fast enough for real-time performance. This is especially true for DFO which is known to require a large number of function evaluations[12]. Although BOBYQA scores well on this by using a under-determined model as seen in section 6.5, this algorithm seems to require an increased number of iterations after the initial model building, which cannot be parallelised directly. This is probably because a smaller number of sample points initially will give less information to use in the initial model $q(x)$. Also the simulations show that the two algorithms using a fully determined sample set tends to be more robust against noise and discontinuities than BOBYQA. As the reason for using DFO is to overcome just such difficulties in the prediction model, these algorithms are attractive. However they require $\frac{(n+1)(n+2)}{2}$ sample points for the initial model building, and this is the biggest obstacle when considering to use DFO in a real-time application. If the number of points required to be evaluated initially for each time step can be reduced significantly without an extensive increase in the algorithm iterations, this would at least be a step in the right direction to make DFO more applicable in a real-time NMPC setting. As the objective function is known to typically be similar in time step k compared to time step $k-1$, this information may be exploited to build an accurate initial model and still use few function evaluations for this. This chapter will propose a modification of the Wedge-algorithm that exploits the nature of the NMPC problem to reduce the number of function evaluations in the initial model building. The reason for using the Wedge-algorithm is purely because this is a Matlab-implementation and therefore easy to modify. The concept can be transferred e.g. to Condor, or even to BOBYQA as will be discussed later.

7.1 Initial model based on old data

The idea behind the modification is to take the warm-start a step further, by also warm-starting the approximation model $q(x)$. The original idea is to start the algorithm at time step k with a model built from the final sample set in the previous time step. However because this sample set is "zoomed" in on the previous solution, the distance between these samples are usually very small, slightly larger than Δ_{end}, and only valid within the hypersphere of this radius.

Therefore a sample set purely used for building of the initial model at each time step is introduced, Y_{Init}. This set, with its corresponding function values f_{Init}, are saved from one time step to the next. Then, only a fraction of these points are moved and function values updated at each time step. After the initial model building is finished this sample set is copied into a "local" sample set used in the iterations of the algorithm, Y_{Iter} and f_{Iter}. This way the points Y_{Iter} will be "zoomed" in on the solution, Y_{Init} still having a relatively large distance between its points. When the algorithm is finished for the current time step, Y_{Iter} and f_{Iter} are discarded. When some of the points in Y_{Init} are updated in the next time step, the current iterate(i.e. the solution from the previous time step) is chosen as the centre for these new points. This way, as the time steps progresses the points in Y_{Init} will slowly "follow" the positions of the current iterate.

To ensure that all points are updated within a given time frame, Y_{Init} is divided into n_s subsets $\mathcal{S}_i, i \in 1 \dots n_s$. The samples belonging to these subsets are fixed, i.e. a sample will never change the subset it belongs to, even though it may(and will) change its geometric position. Then one sub-set is updated at each time step, cycling such that no set has samples older than n_s time steps. The initial model is then built from Y_{Init}, resulting in a model fulfilling the interpolation condition

$$q_k(y^i) = \begin{cases} f_k(y^i) & \forall\ i \in \mathcal{S}_k \\ f_{k-1}(y^i) & \forall\ i \in \mathcal{S}_{k-1} \\ f_{k-2}(y^i) & \forall\ i \in \mathcal{S}_{k-2} \\ \vdots & \\ f_{k-n_s}(y^i) & \forall\ i \in \mathcal{S}_{k-n_s} \end{cases} \tag{7.1}$$

and k denotes the current time step. For the fully determined algorithm Wedge this implies that only $\hat{m} = \frac{(n+1)(n+2)}{2n_s}$ samples needs to be evaluated initially for each time step. This will come at the cost that the initial model may be less accurate, possibly requiring more function evaluations in the proceeding iterations. However the motivation for this approach is the observation that the

difference between $q(x)$ and $f(x)$ is often rather large both in the initial model building and iterations of Wedge. Especially the magnitude of the gradient does usually not decrease at all as the algorithm progresses trough iterations, which would be expected if the accuracy of $q(x)$ is good. Still Wedge seems to perform well in section 6.7, and this suggests that DFTRM is quite robust against model error.
Also the findings in [16] suggests this, as this article experimented with a DFTRM without any poisedness-ensuring mechanism. This caused the sample set to become badly poised inducing large error in the model, still the algorithm was reported to perform well. Further [52] has reported that BOBYQA works with large error in the Hessian of $q(x)$ compared to $\nabla^2 f(x)$, but the algorithm still seemed to work well. This encourages the approach of re-using old sample points and function values of the objective function, although this results in an known model error.

7.2 Updating the sample set Y_{Init}

Because it is known from chapter 3 that the accuracy of $q(x)$ is closely related with poisedness, this is used in the modification to Wedge to ensure that the algorithm gets a good start as possible for each time step. Further, at the fist time step the algorithm uses the same initialisation procedure as the original Wedge algorithm. Thus for the first time step the algorithms are similar. The update procedure for Y_{Init} of the modified Wedge-algorithm is stated in algorithm 11.

- After the initialisation, step 2 shifts Y_{Init} such that its origin is x_0, because algorithm 5 requires this.
- Step 3 chooses which sub set is to be updated. This is done by the use of modulus, e.g. in the simulation where $n_s = 3$ and $t_s = 0.2$, l will cycle through $l = 1, 2, 3, 1, 2, 3, \ldots$.
- Step 4 creates a vector which contains the current order of the points(which currently is in increasing order). The indices corresponding to the points to be updated, i.e. points belonging to $\mathcal{S}_l$, are removed.
- Step 5 removes the sample points in Y_{Init} to be updated. This makes algorithm 5 find new points that makes the set well-poised.

Algorithm 11 Update procedure for Y_{Init}

```
1. Given current time instance $t$, sample period $t_s$, current iterate $x_0$, number
   of update sets $n_s$ and initial TR $\Delta_0$. Load $Y_{Init}$ from the previous time
   step and corresponding objective values $f_{Init}$.
2. $Y_{Init} \leftarrow Y_{Init} - x_0$
3. $l \leftarrow \mod(\frac{t}{t_s}, n_s) + 1$
4. $o \leftarrow [i = 1, 2, \ldots, m] \setminus \mathcal{S}_l$
5. $Y_{Init} \leftarrow Y_{Init}[o]$
6. $Y_{Init}, o \leftarrow$ CompleteNon-PoisedSet$(Y_{Init}, \Delta_0, o)$
7. $(Y_{Init})_{o(j)} \leftarrow (Y_{Init})_j,\ j = 1 \ldots m$
8. $Y_{Init} \leftarrow Y_{Init} + x_0$
9. $(f_{Init})_j \leftarrow f((Y_{Init})_j) \forall j \in \mathcal{S}_l$
10. $j \leftarrow \underset{i \in \mathcal{S}_l}{argmin} (f_{Init})_i$
11. **If** $(f_{Init})_j < f(x_0)$
        Swap $x_0$ and $(Y_{Init})_j$
    **End if**
12. $Y_{Iter} \leftarrow Y_{Init}$
    $f_{Iter} \leftarrow f_{Init}$
    Store $Y_{Init}$ and $f_{Init}$ for use in the next time step.
13. Return $Y_{Iter}$ and $f_{Iter}$
```

- Step 6 executes a slightly modified version of algorithm 5, with Y_{Init}, Δ_0 and o as arguments. The modification of algorithm 5 is for the use of the list o that keeps the order of the sample points. The indices in o are moved when the algorithm swaps points such that index i in o tells which sample currently is on this index in Y_{Iter}. o is returned from the algorithm, thus Y_{Init} can be re-organised to its original order.

- Step 7 does this re-organisation.

- Step 8 shifts the sample set back to a global origin, to prepare for the evaluations of the points in the objective function.

- Step 9 evaluates the function value of the new sample points in Y_{Init} found in step 6, and replaces the corresponding entries in f_{Init}.

- Step 10 finds the index among the new points that has the least function value.

- Step 11 If the point found in step 9 is less than the function value of the current iterate, these two points are swapped. Because the objective function is likely to have changed since the previous time step and the current iterate is the solution of this, this swap occurs quite often as the new samples in Y_{Init} lies in the neighbourhood of x_0.

- Step 12 copies Y_{Init} into Y_{Iter} and f_{Init} into f_{Iter}, such that the iterating procedure of the optimization algorithm works on these. The initial sets are no longer needed in this time step, and is stored such that the next time step can retrieve it in its initialization step.

- Step 13 returns Y_{Iter} and f_{Iter} to the optimization algorithm. Then the first model will be built from these sets, which incorporated old function values. As the optimization progresses, iterates will retrieve new sample points and function values and replace with points in Y_{Iter} and f_{Iter}.

Note the shifts to the sample set back and forth between the global origin and x_0 as origin. This should be modified such that it is only shifted from the initial x_0 of the previous time step to that of the current. This may avoid some numerical round-off errors[50], as shifts are sensitive towards numerical precision. However the simplified presentation is made to keep a good overview in the implementation.

An important aspect is step 10 and 11. When checking if f_{Init} contains any better solutions than x_0, only the newly evaluated sample points can be checked. If

Algorithm	MPC	Fs/Vs	W/C sol.	Av. sol.	Av. evals.	S.S. evals.
Wedge	P.S.	Fs	16.9	2.5	168	143
ModWedge	P.S.	Fs	11.7	1.5	89	76
Wedge	P.S.	Vs	89.5	3.9	164	151
Wedge*	P.S.	Vs	53.7	1.1	197	150
ModWedge	P.S.	Vs	66.1	4.7	90	76
Wedge	Full	Fs	140.3	8.7	389	382
ModWedge	Full	Fs	73.8	11.1	165	162

Table 7.1: Wedge compared with(ModWedge) and without (Wedge) modified update procedure. Wedge* uses $\gamma_1 = 0.75$ instead of 0.5.

all points in f_{Init} are checked, x_0 may be switched with a point that had a lower value in an earlier time step, which may be worse in this step. This way the function values are only used to build the initial model, not in any way directly in optimization.

It is only implemented two stopping criteria, namely minimum TR and maximum number of function evaluations k_{max}. This is simply because these proved the most significant in the simulation in chapter 6.

7.3 Simulations

During the simulations it turns out that the modified Wedge algorithm can handle a larger value on γ_1 without inducing great variance in the optimal solution, and 0.85 is used in these simulations. This is likely to be because it builds the model also on previous data, thus creating a filtering effect. The rest of the parameters are $\Delta_0 = 0.05$, $k_{max} = 110$, $\mu = 0.5$, $\gamma_2 = 2$ and $\gamma_0 = 0.4$ i.e. the same as the simulation in section 6.7.

Table 7.1 compares the modified algorithm with the original. It is observed that for the pre-stabilized fixed-step case, the solutions are better using approximately half the function evaluations. The other results are somewhat as expected, as the use of substantially fewer evaluations and an initial model with known error the result sometimes becomes worse. However the solution is not much worse than that of the original algorithm, in particular in the worst case.

Figure 7.1 and 7.2 shows the resulting input sequences and states when the modified algorithm is applied to the NMPC case-study, using fixed-step solver. Inspecting the resulting cost in figure 7.1 and comparing with that of the original

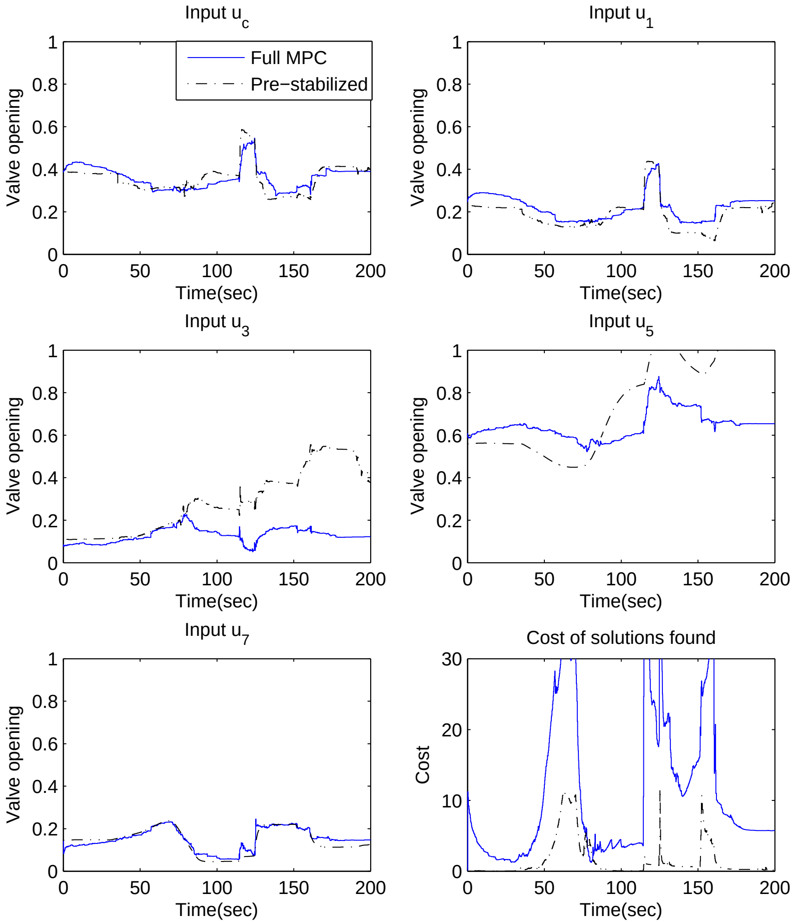


Figure 7.1: Control inputs using modified Wedge, and prediction model simulated with fixed-step solver.

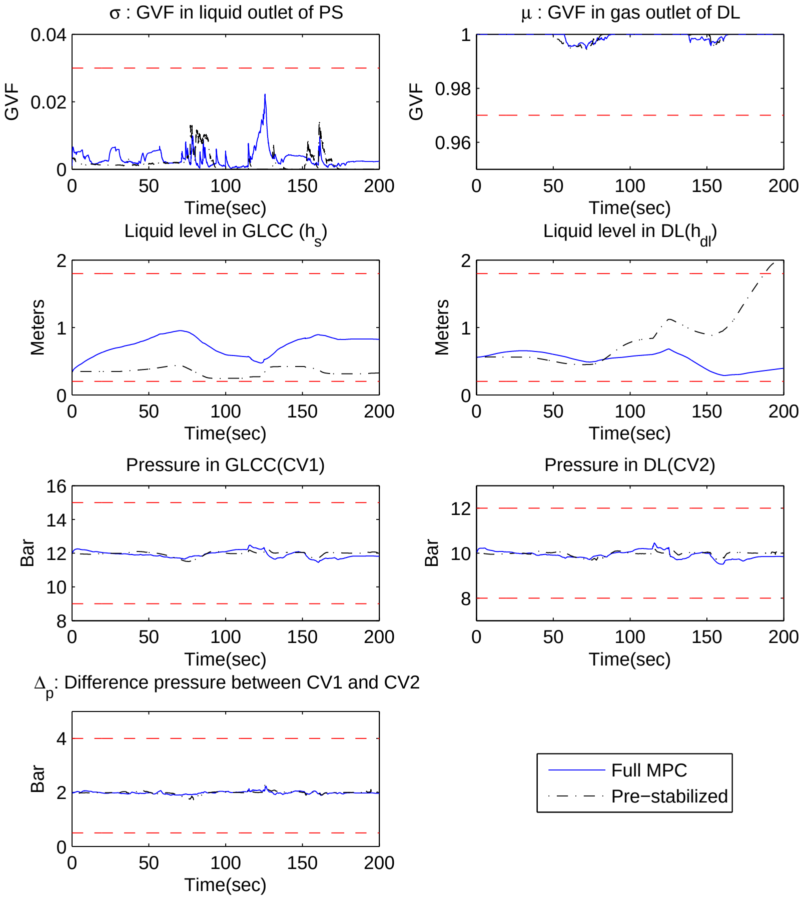


Figure 7.2: Controlled states using modified Wedge, and prediction model simulated with fixed-step solver.

algorithm in figure 6.12, the modified version seems to succeed better in keeping the worst-case costs low for the pre-stabilized plant, as also indicated by table 7.1. However for the "Full MPC" case the modified version starts to struggle as the average solution is worse. From the analysis of the optimization problem it is known that the cost surface becomes less smooth for the "Full MPC" than for the pre-stabilized case, and this may induce a change in the objective function that makes $q(x)$ inaccurate. What such changes may be will be discussed in section 7.5. Still the worst-case solution is better than the original algorithm, which may be the most important case in NMPC as discussed in section 6.8. The re-use of old data is also reflected in the trajectory of the cost, as it seems that the algorithm is less aggressive i.e. more time steps are needed before the cost is minimized. Especially interesting in this case is the trajectory of input u_5 for the "Full MPC", which is significantly smoother for the modified algorithm. Although this is probably because it uses more time steps to find the optimal solution thus really are an "error", it has an positive effect in this NMPC setting, as smooth trajectories are desirable.

Inspecting the results using variable-step solver of figure 7.3 and 7.4, the fist thing to notice is that even thought the modified version uses $\gamma_1 = 0.85$, the result is similar to that of figure 6.14 and 6.15 when using $\gamma_1 = 0.5$. The conclusion from this and the figures is that $\gamma_1 = 0.5$ gives the most precise results during steady-state and $\gamma_1 = 0.75$ gives the best worst-case performance, due to more exploration, however the latter gives more variance in the input trajectories. For the modified algorithm it seems that $\gamma_1 = 0.85$ gives advantages with respect to worst-case performance but still seem to keep the variance low(at least better than $\gamma_1 = 0.75$ for the original version). Compared to the original algorithm using $\gamma_1 = 0.5$, the accuracy of the solutions found seems to be of the same order of magnitude. Because the modified version has a filtering effect, it is slower to converge, the average solution is somewhat worse, however the ability to use high γ_1 gives an improved worst-case performance as both table 7.1 and the figures shows.

7.4 Discussion of the results

Generally, the simulations gives promising results for the approach of re-using "old" function evaluations. Compared to the original Wedge-algorithm it has an smoothing effect on the inputs due to the use of old data, which actually in this case results in better controller performance even though the cost of the solution is slightly higher. This reflects that limited time is spent on designing

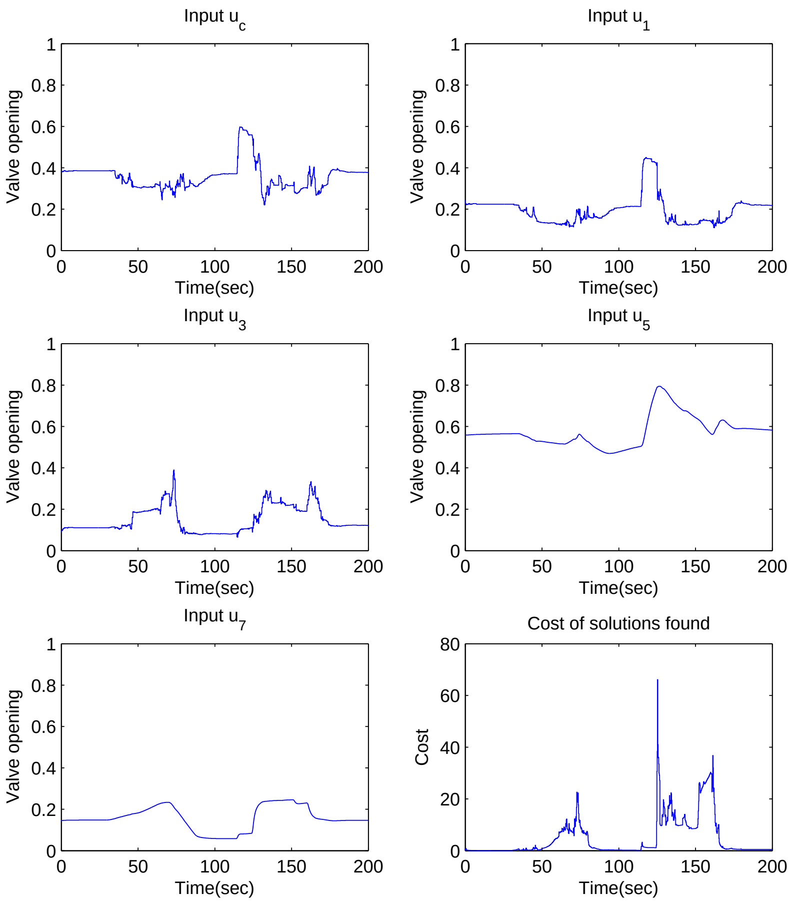
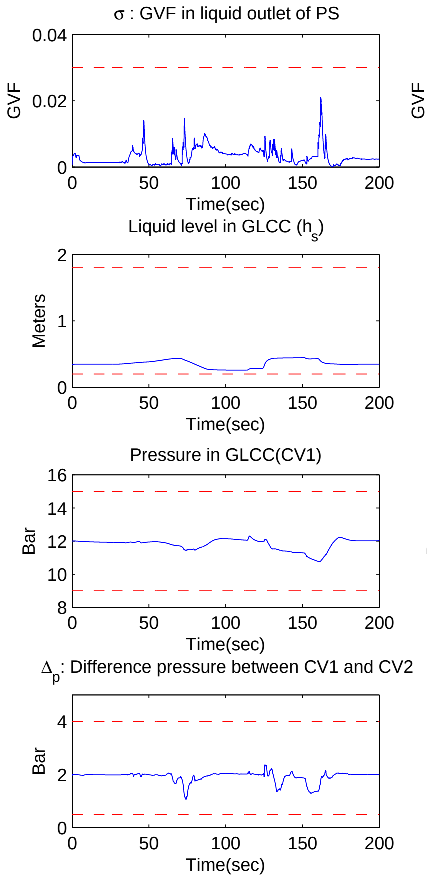


Figure 7.3: Control inputs using modified Wedge, and prediction model simulated with variable-step solver.



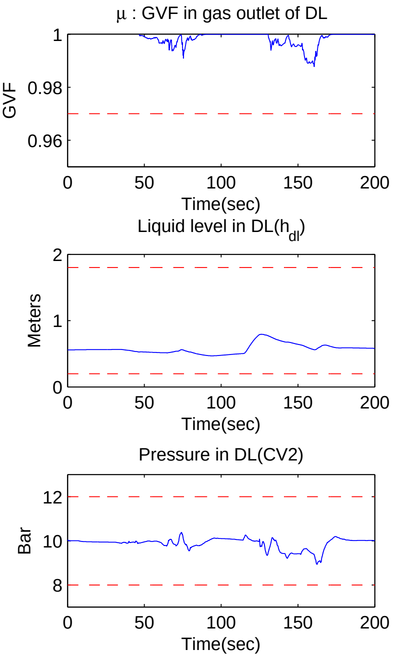


Figure 7.4: Controlled states using modified Wedge, and prediction model simulated with variable-step solver.

and tuning the optimization problem, as discussed in section 6.2. It must also be taken into account that the method used here is simple, and is mainly to investigate if improved warm-start is a possible way to go.

An important aspect in the context of recycling function evaluations, is the topic of parallelisation. It seems that although the total number of function evaluations is reduced, the algorithm tends to require more evaluations proceeding the initial model building. For the fixed-step case the modified algorithm uses approximately ten evaluations more during the iterations. Distributing the initial function evaluations over 16 cores as in section 6.8, this results in a an increase in relative processing time consumption of 6 evaluation and a reduction in 5 evaluations for the pre-stabilized and "Full MPC", respectively, using the fixed-step solver. For the case of variable-step solver the modified algorithm either uses 22 evaluations less than Wedge in the case of $\gamma_1 = 0.5$, or 11 evaluations more if $\gamma_1 = 0.75$. These differences are so small that they should be seen as similar, as they may vary from case to case.

7.5 Further work

Intuitively as long as the change in the objective function is small as is the case during moderate disturbances, the model error is likely to be small as discussed in section 5.5. However, consider figure 7.5: The objective function is convex with positive-definite Hessian both at $t = 1$ and $t = 2$. From $t = 1$ to $t = 2$ one sample in the sample set is to be updated, and the choice falls on the sample marked three. This will in fact lead to a negative-definite Hessian, and may lead to a solution to the TR sub-problem which leads in the wrong direction.

Note that given the performance of the modified algorithm, it does not seem like the algorithm suffer greatly from this issue. DFTRM seems to be either very robust against large errors in $q(x)$, or the problem does not arise very often. An observation from figure 7.5 is that updating two very close samples is more likely to generate this issue than when updating two points far from each other. Then the use of a well-poised Y_{Init} is likely to have a positive effect on this issue as a poised set has samples with large distance between each other. Considering this, and under the weak assumption that the change in the objective is small enough, the described issue may not be relevant for a practical problem. Even so, for a real-world NMPC application, this problem should be investigated further to ensure robustness in the controller.

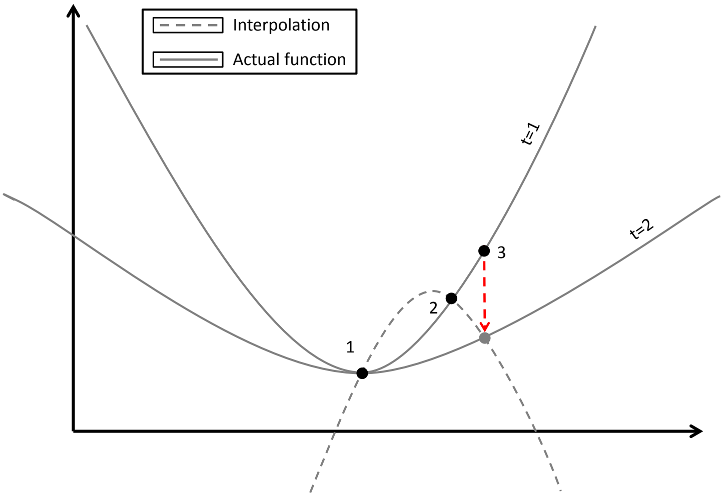


Figure 7.5: Illustration of how an unfortunate choice of sample update can lead to a negative-definite Hessian.

A solution may be to dynamically choose which sample points to be updated, and not use the fixed sets as described in section 7.2. The issue may occur if some and not all of the samples spanning an axis is updated. If either all or none of the samples spanning a given axis is chosen to be updated, the issue should not arise. Consider figure 7.6. If the horizontal axis are chosen for update, the samples points x^3, x^4 and x^5 must be re-evaluated. For the vertical axis x^1, x^2 and x^5 is chosen. It also requires that each sample spans few axes, which is not a very ideal case seen from a poisedness point-of-view. By running a poisedness-ensuring mechanism as described in chapter 3, samples tend to span a substantial number of axes, thus the described technique does not become applicable as almost all samples will have to be updated at each time step.

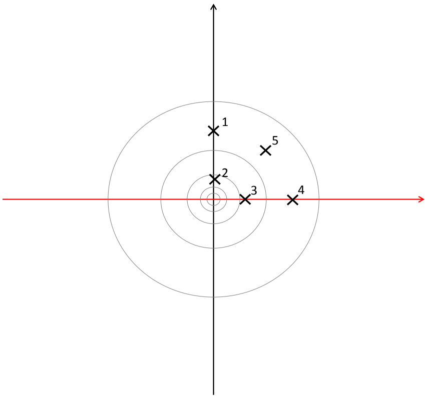


Figure 7.6: Illustration of how one sample, x^5, spans several axes.

A more promising approach for solving the problem may be to apply the techniques used for model building by BOBYQA. A possible method may be one that does not use two sample sets, but stores the Hessian of the initial model building for use in the next time step rather than storing the sample points. Then sampling say $2n + 1$ points, as with BOBYQA in section 6.5, and using the update technique (4.34), the information of the objective function in previous steps will be transferred through the Hessian to the current time step. Although not necessarily solving the problem described in figure 7.5, the method may "weight" earlier Hessians and not allowing such a sudden big change to it as figure 7.5 illustrates. As BOBYQA proved effective using this update technique, this seems like a promising way to go.

On the topic of parallelisation discussed in section 7.4, it is clear that a large initial sample set can be accepted if this results in a significant reduction in function evaluations in the proceeding iterations. Although the ideal case is to reduce both the required initial sampling set as well as the number of proceeding evaluations, it is clear that when a relatively large number of processing cores can be used for parallelisation, as is the case when applying a GPU for parallelisation[54], a large initial sample set can be used. Therefore it would be interesting to investigate the possibility to apply over-determined interpolation in the DFTRM as presented in section 3.6. This will then require that the number of iterations is significantly reduced, and if such an approach has this effect is unknown. Although the initial model should become more accurate and this intuitively should make the converge of the algorithms faster, experiments both in this thesis and previously[16] has shown that DFTRM does not need an accurate model to perform well. It is therefore unknown if these methods will exploit the information in an over-determined model.

It would also be interesting to investigate to combine the update-technique of section 7.2 with the over-determined scheme described above. This can be accomplished by re-cycling all of the sample points from a given number of time steps back in time, and use these to form an over-determined interpolation. Considering a small change in the objective function, this may combine the noise-suppressing abilities of an over-determined model with the reduction in function evaluations of the approach of section 7.2.

Continuing the discussion on computational consumption and real-time performance from section 6.8, the modified algorithm still is to computationally expensive to be applicable in real-time for the case study. However because this is a simple implementation, it is likely that further research and development of the algorithm in combination with parallelisation and computationally faster simulation software for the prediction model can make it come closer to this goal. Further other measures can also be though of, as altering the NMPC problem. This can be to experiment with fewer input blocks , or maybe less number of blocks on the pressures as these seem to be quite simple to stabilize. Also there can be considered increasing the sample interval of the controller, which will lower the demand for computation speed for the controller.

Chapter 8

Conclusion

The goal of the thesis were to investigate the suitability of DFO in a NMPC optimization problem, to overcome issues such as numerical noise and discontinuities in the objective function. Some fundamental theory behind DFO and survey of some existing DFO algorithms have been presented, namely Wedge, UOBYQA, Condor and BOBYQA. A case-study of an industrial, numerically difficult NMPC problem using the single-shooting approach was presented, and both a gradient-based SQP algorithm and DFO were used to simulate the closed-loop system. Lastly, a warm-start modification to one of the algorithms for reducing the computational consumption was presented and simulated.

Analysis of the NMPC problem used in the simulations uncovered that discontinuities can arise in the objective function both when using a fixed and variable-step ODE-solver. Simulations using a gradient-based SQP-algorithm in the NMPC, proved that these types of algorithms are highly vulnerable to such numerical difficulties, and the algorithm experienced issues with differentiation for both types of ODE solvers. It is encouraging that all the DFO algorithms tested performed better with respect to worst-case performance during the simulations than the SQP. Although the issue is more severe when applying a variable-step solver, this stresses the importance of optimization algorithms that is robust against such numerical difficulties.

As in NMPC in general, real-time performance is a key aspect when applying DFO. Simulations proved that the DFO algorithm BOBYQA performed well with respect to the number of evaluations of the objective function, however it tends to have trouble when severe numerical noise, non-smoothness and discontinuities are introduced in the case study. As long as the objective function is relatively smooth, BOBYQA proved to be comparable with the SQP algorithm used. Generally, if only one processing core is available for computation, BOBYQA is a computationally efficient DFO algorithm. It is, however, not as suited for parallelisation as the other DFO algorithms.

Considering the ability to handle severe numerical noise, non-smoothness and discontinuities, Condor stands out as the best suited in the case study. Although having the largest demand of function evaluations on average, it handles severe numerical difficulties well without a significant increase in the required number of function evaluations, compared to the smooth case. This makes this algorithm attractive in NMPC, as computational resources usually must be reserved for worst-case anyway.

Wedge has a large number of tuning parameters, giving possibilities for fine-tuning the algorithm for the problem at hand. This was illustrated as a change in one tuning parameter gave the algorithm the best average cost of the solutions, and simultaneously improved worst-case performance significantly using less computational resources. However, it also illustrated how the cost of the solutions should not be the only criterion to evaluate, as variance in the optimized variables increased in the simulations. It is likely that a better formulated NMPC problem would have improved on this matter, but this remains unknown.

Parallelisation of the function evaluations is an important aspect, and it was discussed how this can make the Wedge and Condor algorithms faster with respect to computational time consumption than BOBYQA, even though these requires more evaluations of the objective function than the latter. Under these circumstances Condor seems very attractive, because of its worst-case performance and that parallelisation is already implemented.

A novel modification to the Wedge algorithm, which recycles function evaluations from earlier time steps to reduce computational time consumption, is presented and simulated. This approach seems promising, considering the simplicity of the implementation. It is likely that further research and development can result in a DFO algorithm that is able to optimize in real-time, and be robust against numerical issues, and suggestions are made that may improve the algorithm further.

8.1 Further work and final remarks

The simulations performed required large computation time, and to try to improve the real-time performance of the simulations will give the ability to do a larger number of simulations in a shorter period of time. This will make it easier to evaluate the quality of the NMPC formulation, and also the performance of the algorithms themselves under different conditions. This may be accomplished by using a less computationally demanding software to simulate the prediction model, and avoid using Simulink as this does not seem suited for the purpose. Also, applying parallelisation will decrease the computational time consumption greatly, and as the C++ version of Condor already has such a feature this would be interesting to exploit.
Investigation and possibly improvements of the NMPC formulation is needed for the case-study presented in this thesis, as the simulations has suggested that this is not optimal. Altering the NMPC horizon and input blocking may also lower the computational demand for the case-study. Simulations with other models are needed to verify that the results are transferable to other cases.

The simulations has shown that DFO has robustness that is desirable in certain NMPC problems. The algorithms is known to have a slower converge rate and require a large number of function evaluations, especially for large-scale problems. However, with the increased availability of computational resources, especially parallelisation, they have the ability to be fast enough for real-time performance. If the optimization problem is smooth and differentiable, gradient-based optimization is still preferable, but DFO is a good alternative when this is not the case.

Bibliography

[1] Adifor home page. `http://www.mcs.anl.gov/research/projects/adifor/`. Accessed: 12/12/2012.

[2] Matlab symbolic toolbox. Accessed: 31/04/2013.

[3] Maxima home page. `http://maxima.sourceforge.net/`. Accessed: 12/12/2012.

[4] Maple home page. `http://www.maplesoft.com/products/maple/`, 2012. Accessed: 12/12/2012.

[5] H. Al-Duwaish and W. Naeem. Nonlinear model predictive control of hammerstein and wiener models using genetic algorithms. In *Control Applications, 2001. (CCA '01). Proceedings of the 2001 IEEE International Conference on*, pages 465–469, 2001.

[6] F.V. Berghen. Condor user's guide. `http://www.applied-mathematics.net/CONDORManual/CONDORManual_1.0.pdf`. document v. 1.05.

[7] F.V. Berghen. *CONDOR: A constrained, non-linear, derivative-free parallel optimizer for continous, high computing load, noisy objective functions.* PhD thesis, Facultè des Sciences Appliquèes, Universitè Libre de Bruxelles, 2004.

[8] F.V. Berghen and H. Bersini. CONDOR, a new parallel, constrained extension of powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. *Journal of Computational and Applied Mathematics*, 181:157–175, September 2005.

[9] L.G. Bleris, J. Garcia, M.V. Kothare, and M.G. Arnold. Towards embedded model predictive control for system-on-a-chip applications. *Journal of Process Control*, 16(3):255–264, 2006.

[10] W. Chen, X. Li, and M. Chen. Suboptimal nonlinear model predictive control based on genetic algorithm. In *Intelligent Information Technology Application Workshops, 2009. IITAW '09. Third International Symposium on*, pages 119–124, 2009.

[11] A.R. Conn, K. Scheinberg, and L.N. Vicente. Geometry of interpolation sets in derivative free optimization. *Mathematical programming*, 111:141–172, 2008.

[12] A.R. Conn, K. Scheinberg, and L.N. Vicente. *Introduction to Derivative-Free Optimization.* Society for Industrial and Applied Mathematics and the Mathematical Programming Society, 2009.

[13] J.S. Dæhlen. Paralell-implementation of derivative-free model-predictive control. Project report, NTNU, 2012.

[14] M. Diehl, H.G. Bock, J.P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577 – 585, 2002.

[15] M. Diehl, H.J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In Lalo Magni, Davide-Martino Raimondo, and Frank Allgöwer, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 391–417. Springer Berlin Heidelberg, 2009.

[16] G. Fasano, J.L. Morales, and J. Nocedal. On the geometry phase in model-based algorithms for derivative-free optimization. *Optimization Methods and Software*, 24(1):145–154, 2009.

[17] B. Foss. Linear quadratic control. Note from the subject TTK4135 Optimization and Control at NTNU, March 2004.

[18] K.R. Fowler, J.P. Reese, C.E. Kees, J.E. Dennis Jr., C.T. Kelley, C.T. Miller, C. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray, and T.G. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743 – 757, 2008.

[19] C.E. García, D.M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335 – 348, 1989.

[20] M. Gasca and T.B. Sauer. Polynomial interpolation in several variables. *Advances in Computational Mathematics*, 12:377–410, 2000.

[21] L. Grüne and J. Pannek. Practical NMPC suboptimality estimates along trajectories. *Systems and Control Letters*, 58(3):161 – 168, 2009.

[22] V. Gunnerud, A. Conn, and B. Foss. Embedding structural information in simulation-based optimization. *Computers and Chemical Engineering*, 53(0):35 – 43, 2013.

[23] S.O. Hauger. Lectures in course TK16: Model predictive control. Department of Engeneering Cybernetics, NTNU, September 2012.

[24] R.L. Haupt and S.E. Haupt. *Practical Genetic Algorithms, Second Edition.* John Wiley & Sons, Inc, 2004.

[25] L. Imsland. Introduction to model predictive control. *Lecture notes from the course Optimization and Control, Department of Engineering Cybernetics, NTNU*, 2007.

[26] S. Jakobsson, M. Patriksson, J. Rudholm, and A. Wojciechowski. A method for simulation based optimization using radial basis functions. *Optimization and Engineering*, 11(4):501–532, 2010.

[27] T.A. Johansen. Introduction to nonlinear model predictive control and moving horizon estimation. `www.irk.ntnu.no/ansatte/Johanse_Tor.Arne/nonlinear.pdf`, 2011.

[28] S.G. Johnson. The NLopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`, July 2012. Accessed: 12/12/2012.

[29] B. Karasözen. Survey of trust-region derivative-free optimization methods. `http://144.122.137.13/iam/images/2/29/Preprint58.pdf`, 2000. Accessed: 01/11/2012.

[30] C.T. Kelley. Implicit filtering. `http://www4.ncsu.edu/~ctk/imfil.html`, February 2011. Accessed: 12/12/2012.

[31] T.G. Kolda, R.M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):pp. 385–482, 2003.

[32] D. Koller and S. Ulbrich. Optimal control of hydroforming processes. *Proceedings in Applied Mathematics and Mechanics*, 11(1):795–796, 2011.

[33] D.K. Kufoalor. Nonlinear model predictive control of a subsea separation process: Real-time optimization and numerical methods. Presentation, December 2012.

[34] W.H. Kwon, A.M. Bruckstein, and T. Kailath. Stabilizing state-feedback design via the moving horizon method†. *International Journal of Control*, 37(3):631–643, 1983.

[35] A.M. Law and M.G. McComas. Simulation-based optimization. In *Simulation Conference, 2000. Proceedings. Winter*, volume 1, pages 46–49 vol.1, 2000.

[36] M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming*, 91:289–305, 2002. 10.1007/s101070100264.

[37] D.Q. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *Automatic Control, IEEE Transactions on*, 35(7):814–824, 1990.

[38] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Staility and optimality. *Automatica*, 36:789–814, 2000.

[39] M. Morari and J.H. Lee. Model predictive control: past, present and future. *Computers and Chemical Engineering*, 23(4–5):667 – 682, 1999.

[40] J. Nocedal and S.J. Wright. *Numerical Optimization, Second Edition*. Springer, 2006.

[41] P. Norgren. Compact subsea separation unit: Nonlinear model predictive control and nonlinear observers. Master's thesis, NTNU, 2011.

[42] R. Oeuvray and M. Bierlaire. Boosters: a derivative-free algorithm based on radial basis functions. *International Journal of Modelling and Simulation*, 29(1):26, 2009.

[43] C. Onnen, R. Babuška, U. Kaymak, J.M. Sousa, H.B. Verbruggen, and R. Isermann. Genetic algorithms for optimization in predictive control. *Control Engineering Practice*, 5(10):1363 – 1372, 1997.

[44] J.M. Pena and T. Sauer. On the multivariate horner scheme. *SIAM Journal on Numerical Analysis*, 37(4):pp. 1186–1197, 2000.

[45] M.J.D. Powell. UOBYQA: Unconstrained optimization by quadratic approximations. DAMTP 2000/NA14, December 2000.

[46] M.J.D. Powell. UOBYQA: Unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[47] M.J.D. Powell. On trust region methods for unconstrained minimization without derivatives. *Mathematical Programming*, 97(3):605–623, 2003.

[48] M.J.D. Powell. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming*, 100(1):183–215, 2004.

[49] M.J.D. Powell. On the use of quadratic models in unconstrained minimization without derivatives. *Optimization Methods and Software*, 19(3-4):399–411, 2004.

[50] M.J.D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In G. Pillo, M. Roma, and Panos Pardalos, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 255–297. Springer US, 2006. 10.1007/0-387-30065-1_16.

[51] M.J.D. Powell. Developments of NEWUOA for unconstrained minimization without derivatives. *Dept. Appl. Math. Theoretical Phys., Univ. Cambridge, Cambridge, UK, Tech. Rep. DAMTP*, 2007. DAMTP 2007/NA05.

[52] M.J.D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 2009.

[53] S.J. Qin and T.A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733 – 764, 2003.

[54] A. Sadrieh and P. A. Bahri. Application of graphic processing unit in model predictive control. In M.C. Georgiadis E.N. Pistikopoulos and A.C. Kokossis, editors, *21st European Symposium on Computer Aided Process Engineering*, volume 29 of *Computer Aided Chemical Engineering*, pages 492 – 496. Elsevier, 2011.

[55] H. Sarimveis and G. Bafas. Fuzzy model predictive control of non-linear processes using genetic algorithms. *Fuzzy Sets and Systems*, 139(1):59 – 80, 2003.

[56] K. Scheinberg and P. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM Journal on Optimization*, 20(6):3512–3532, 2010.

[57] B. Tlili, F. Bouani, and M. Ksouri. A derivative-free constrained predictive controller. In *Proceedings of the 10th WSEAS international conference on Systems*, pages 360–365. World Scientific and Engineering Academy and Society (WSEAS), 2006.

Appendix A

List of variables in the separator model

Disturbance variables		Input variables	
δ	GVF separator input	u_1	Valve constraining q_1
q_{in}	Flow rate separator input	u_3	Valve constraining q_3
p_{in}	Pressure GLCC input	u_4	Valve constraining q_5
p_{out1}	Pressure gas outlet of separator	u_7	Valve constraining q_4
p_{out2}	Pressure liquid outlet of separator	u_c	Valve constraining q_{out1}
State variables		Parameter variables	
α	GVF of gas outlet of GLCC	p_1	Pressure in CV1
β	GVF of liguid outlet of GLCC	p_2	Pressure in CV2
σ	GVF of liquid outlet of PS	Δ_p	Difference pressure $\lvert p_1 - p_2 \rvert$
ν	GVF of gas outlet of PS	q_1	Gas flow rate from GLCC
μ	GVF of gas outlet of DL	q_2	Liquid flow rate from GLCC
η	GVF of liquid outlet of DL	q_3	Gas flow rate from PS
m_{1l}	Liquid mass in CV1	q_4	Liquid flow rate from PS
m_{1g}	Gas mass in CV1	q_5	Liquid flow rate from DL
m_{2l}	Liquid mass in CV2	q_{out1}	Gas flow rate from separator
m_{2g}	Gas mass in CV2	q_{out2}	Liquid flow rate from separator
		h_s	Liquid level in GLCC
		h_{dl}	Liquid level in DL

Table A.1: The disturbances, states and parameters relevant for the separator model used in simulations.

Appendix B

Steady-state setting of the separator

Disturbance variables		Input variables	
Name	Value	Name	Value
δ	$5 \cdot 10^{-1}$	u_1	$3.81 \cdot 10^{-1}$
q_{in}	$4 \cdot 10^{-2}$	u_3	$0.24 \cdot 10^{-1}$
p_{in}	9	u_4	$8.78 \cdot 10^{-2}$
p_{out1}	7	u_7	$6.09 \cdot 10^{-1}$
p_{out2}	7	u_c	$1.44 \cdot 10^{-1}$
State variables		Parameter variables	
Name	Value	Name	Value
α	$9.34 \cdot 10^{-1}$	p_1	12.00
β	$7.18 \cdot 10^{-2}$	p_2	10.00
σ	$7 \cdot 10^{-4}$	Δ_p	2.00
ν	$6.84 \cdot 10^{-1}$	q_1	$1.67 \cdot 10^{-2}$
μ	1	q_2	$2.03 \cdot 10^{-2}$
η	$3.35 \cdot 10^{-2}$	q_3	$2.10 \cdot 10^{-3}$
m_{1l}	51.72	q_4	$1.82 \cdot 10^{-2}$
m_{1g}	3.53	q_5	$1.50 \cdot 10^{-3}$
m_{2l}	28.07	q_{out1}	$2.07 \cdot 10^{-2}$
m_{2g}	$9.22 \cdot 10^{-1}$	q_{out2}	0.00
		h_s	$3.45 \cdot 10^{-1}$
		h_{dl}	$5.56 \cdot 10^{-1}$

Table B.1: Steady-state values for the variables of the separator model.

Appendix C

Article for publication in journal

Nonlinear Model Predictive Control using Trust-Region Derivative-Free Optimization with Application to a Subsea Oil-Gas Separation Process

Jon S. Dæhlen, Gisle Otto Eikrem, Tor Arne Johansen

Abstract

Gradient-based optimization may not be suited if the objective and constraint functions in a Nonlinear Model Predictive Control(NMPC) optimization problem are not differentiable. Derivative-Free Optimization(DFO) have been frequently used in Simulation-based optimization, as well as to some extent also in NMPC. However the NMPC studies have mainly been limited to smaller and simpler systems. The findings are that DFO is significantly more robust against the numerical issues, compared to a gradient-based SQP tested. A novel warm-start modification to the Wedge algorithm to improve computational consumption is proposed and simulated with promising results.

1 Introduction

In the later years Nonlinear Model Predictive Control (NMPC) has attracted much attention[4], as the use of non-linear models improves the controller performance on highly non-linear systems and allows for operation over wider range[1].

A common method for solving an NMPC problem is by linearising the model around the previous optimal trajectory, referred to as the nominal trajectory. This can be optimized as in linear Model Predictive Control (MPC), and a new linearisation is performed in the next time step around the updated nominal trajectory. This however makes the model only valid around small perturbations from its current state, and e.g. if the set-point is changed, accurate prediction of larger transients may be desirable. Today state-of-the-art to overcome this seems to be Sequential-quadratic programming (SQP) and Interior-Point (IP)-methods[7]. Common for both is that they recursively linearise the model and the constraints until convergence is achieved, which in turn requires to somehow retrieve the gradient of these. If the model is known explicitly, an off-line symbolic differentiation can be performed and the gradient and possibly the Hessian implemented.

However this may not always be the case, as the model may not be available in an appropriate programming language, it can be made up of a mix of subroutines from different programming languages or the source can simply be unavailable. This will render the symbolic software hard or impossible to apply. The model may even not be continuous, as logic operators are common, and the model may not be explicitly available at all as it may be embedded in a numerical simulation software[13]. In these cases it is likely that the most common and intuitive is to retrieve the gradient from finite-differences. This method is however known to be sensitive towards numerical issues[16]. When performing the simulation it can be desirable to use a variable-step Ordinary Differential Equation (ODE)-solver. This can speed up the simulation time significantly, however it is known that this tends to induce numerical noise and discontinuities, which possibly can be amplified through finite-differences and thus compromise the performance of a gradient-based NMPC optimization. This motivates for investigating optimization methods not requiring derivatives, namely Derivative-free optimization (DFO).

DFO has been applied in NMPC in several occasions, by use of Genetic Algorithms (GA)[4][14][1][21][3]. However in these cases the goal has been to overcome extreme non-linearities. Although GA is known to be well-suited towards noise and discontinuities it also tends to have slow convergence rate. [23] used the Nelder-Mead simplex method for solving a highly non-linear NMPC problem and reported that this were 10 times faster than GA, and [20] used parallel computing on a Graphics Processing Unit (GPU) to obtain even faster computation. These did however suggest to try other optimization methods and more practical problems. [10] reported that a Derivative-free Trust-region method (DFTRM) out-performed the Nelder-Mead method in an optimal-control problem, however the exact nature of the optimization problem was not stated. DFTRM is also known to require less function evaluations than algorithms as Nelder-Mead and GA[13], thus investigating the use of these methods in NMPC aimed at controlling a realistic process seems appropriate.

The main contribution of the paper is an evaluation of DFTRM on a challenging sub-sea oil and gas separation process, including a novel warm-start procedure.

The rest of the paper will be structured as follows:

> Section 2 will briefly present the single-shooting formulation of the NMPC problem.
>
> The case study of an industrial sub-sea crude-oil separation unit will be presented in section 3. The objectives of the controller is also presented, and the resulting NMPC problem is analysed.
>
> Section 4 gives a short description of DFTRM, and presents some existing DFO algorithms, and a novel warm-start procedure.

The closed-loop system is simulated in section 5. The optimization is done both with the DFO algorithms and a gradient-based SQP-algorithm.

Section 6 summarises and concludes the findings.

For more details on the implementation and the simulations, the reader is referred to [6].

2 Nonlinear Model Predictive Control Formulation

The single-shooting approach to NMPC is based on parametrizing the objective function of the optimization problem in the input sequence(i.e. the Manipulated variables (MV)) and the current state of the system. This approach is well suited when the prediction model of the plant is a simulation, i.e. the solution to an initial value problem using an ODE-solver. Denoting this solution

$$[x(1), \dots, x(\hat{K})] = \ell(u(0), \dots, u(K-1), x(0), K) \tag{1}$$

Where K is the discrete length of the control horizon, and $x(0)$ is the current state of the plant. The left-hand side of (1) is then the resulting trajectory of the states of the model from time step 1 to $\hat{K}$. Note that if a variable-step solver is used, K is likely to be different from $\hat{K}$. Considering an objective function on the form $f(x(1), \dots, x(\hat{K}), u(0), \dots, u(K-1))$, which is common in NMPC, the complete single-shooting NMPC problem can be stated

$$\begin{aligned}
\min_{u\in\mathbb{R}^n} \; & f(\ell(u(0), \dots, u(K-1), x(0), K), u(0), \dots, u(K-1), x(0)) && \text{(2a)}\\
& := f(u) && \text{(2b)}\\
& \text{s.t} && \\
& \quad x_{min} \leq x(k) \leq x_{max} && \text{(2c)}\\
& \quad u_{min} \leq u(k) \leq u_{max} && \text{(2d)}\\
& \quad \Delta u_{min} \leq \Delta u(k) \leq \Delta u_{max} && \text{(2e)}
\end{aligned}$$

The definition of $f(u)$ is to illustrate that there are only the input blocks that are free variables, (MV). Note that because of input blocking, the number of MV will be less than K, however this is not considered in (2) for simplicity of notation. Clearly although the constraints are linear, the objective function may not be convex. Applying SQP or IP methods will somehow need to make a linearisation of the objective function to approximate (2) with a Quadratic Programming (QP) problem, thus requiring the gradient of $f(u)$. As mentioned initially, the prediction model may not be differentiable, thus this process may become troublesome.

3 Subsea Separation Process Case Study

The system that will be used to test the DFO algorithms is a model of a sub-sea crude-oil separator, developed by Statoil. The model is a simplified model of a laboratory setup as the system is still under design, and figure 1 shows its structure. The following description of the separation process is an excerpt from [12], where also the details of the simplifications can be found.

The unit separates the input crude oil flow q_{in} into its gas q_{out1} and liquid q_{out2} component flows. The input flow rate may be time varying, as well as the fraction of gas in the crude oil, called Gas Volume Fraction (GVF), which is between zero(liquid only) to one(gas only). These variations are here considered as disturbances which can be measured but without any knowledge of their future behaviour.

The unit is split into two main control volumes, each containing a liquid level. The Gas-Liquid Cylindrical Cyclone (GLCC) does the first separation of the crude oil, however at high flow rates the GLCC does a very rough separation[12]. Therefore both the gas output flow q_1 and the liquid output flow q_2 from GLCC is fed through second separation stages, called the De-Liquidizer (DL) and Phase Splitter (PS) respectively. Both these stages are co-axial cyclones, which rotates the input mixture to create a centrifugal force which separates the gas and liquid phases. The gas is then extracted from the centre of the cyclones, while liquid will move along the walls.

Gas-flow from the GLCC to the DL is controlled by the valve u_1. The gas outlet from the PS, q_3, is fed into the mixture inlet of the DL, while the liquid outlet of the DL, q_5, is fed right into the liquid outlet of the separator. These flows are controlled by valves u_3 and u_4 respectively. At outlet q_{out1} there is a compressor to move the gas to the surface, and at q_{out2} there is a pump for the same purpose for the liquid. The feed to these are controlled by the valves $u_{compressor}$ and u_7 respectively. Note that the former will be referred to as u_c from now.

The model also contains the liquid levels of the GLCC and the DL, measured in meters, and assigned notation h_s and h_{dl} respectively. These levels are not necessarily desirable to keep at a set-point, as allowing them to move freely within certain bounds can make the whole system act smoothly. However the cyclones has a physical maximum liquid level where they will overflow. Both these limits are at 2 meters, and these therefore naturally become constraints in the NMPC.

Also important variables for the NMPC is the pressures of Control volume 1 (CV1), p_1, of Control volume 2 (CV2), p_2, and the difference between these pressures Δ_p. From [12], p_1 should be kept between 9 and 15 bar, p_2 between 8 and 12 bar while Δ_p between 0.5 and 4 bar. Simulations done in earlier studies has shown that these constraints are relatively easy to satisfy [12], however the consequence of a violation is not accounted for.

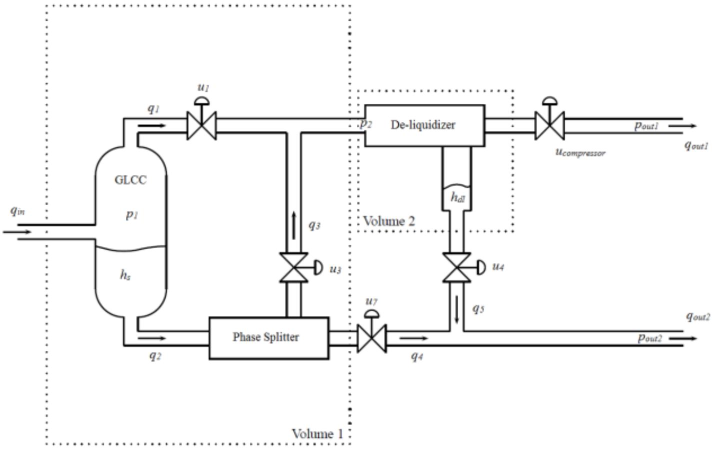


Figure 1: The separator unit used in testing the NMPC

The overall goal of the separation process is to keep the GVF of the gas outlet, GVF_{out1}, as close as possible to one, and the GVF of the liquid outlet, GVF_{out2}, as close as possible to zero. This goal is rarely feasible to obtain perfectly, however two physical constraints are present on these variables. The compressor and the pump receiving q_{out1} and q_{out2} has limits on the GVF of their input flow. Violating these limits can inflict damage to the units. This results in a lower constraint in the GVF of q_{out1} of 0.97 and the GVF of q_{out2} should be constrained to below 0.03. However it should be pointed out that these constraints can be allowed to be violated for short periods of time. Although satisfying these constraints during large disturbances is feasible, this results in large input controls. Also the system has a big spread of time constants, from the rapidly changing GVF and pressures to the slow-varying liquid levels. To control the quick modes, the input blocks of the MPC horizon should be chosen short, while controlling the slow modes requires a long horizon. Together these two criteria results in a significant amount of optimization variables. Therefore [12] pre-stabilizes the plant using PID-controllers for controlling the liquid heights by feeding the measurements of h_{dl} and h_s back to the valves u_4 and u_7 respectively. This both reduces the number of variables as the MPC only has to control three valves instead of five, and as the slow dynamics is stabilized a shorter horizon can be chosen. Both the approach of controlling all valves with MPC and using pre-stabilization are simulated in closed-loop.
For analysis of the dynamics of the model, a simulation without NMPC is made both for the pre-stabilized and the unstabilized model. This simulation is made by applying steps in the disturbances at different points in

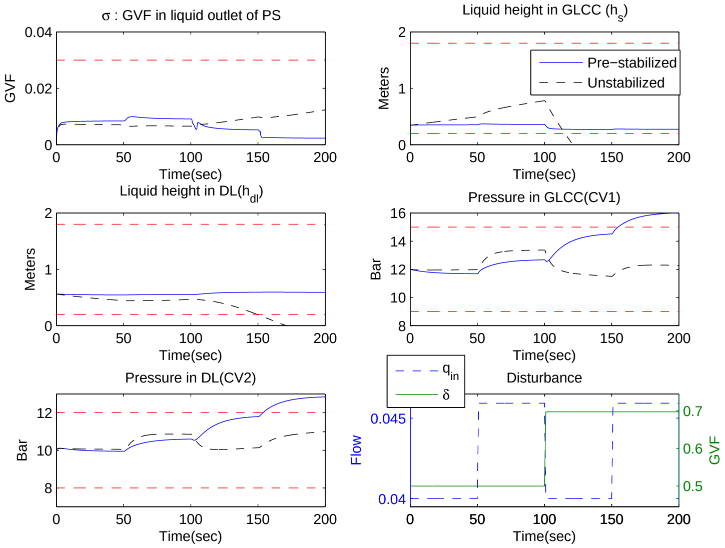


Figure 2: The separator model with steady-state inputs and steps in the disturbance.

time. Figure 2 shows the result of the simulation. The applied disturbance sequences are shown in the lower right plot. In the un-stabilized case, the liquid levels are highly sensitive towards the disturbance. As the flow increases, so does the liquid levels as more liquid is introduced in the unit. However when the input GVF δ increases a smaller fraction of the input flow is liquid, thus the liquid levels decreases and actually reaches zero and continues in negative direction relatively quickly. As this clearly is not realistic the trajectory after this point is not representative with respect to the real-world unit, however it should be noted that all other states remains within their constraints. Note that the GVF of the DL, μ, and the difference pressure Δ_p is not plotted to save space, as these variables are generally very close to their optimal values and are thus not considered very interesting. By applying the PID-controllers for pre-stabilizing the liquid levels, figure 2 shows that the liquid levels now become locally stable, and are kept close to their set-points. However this comes at the cost that the pressures in the control volumes increases. It seems that when only one of the disturbances are perturbed, the pressures will converge close to their constraints, however when both q_{in} and δ are perturbed from their steady-state values the

Variable	Set-point	Upper bound	Lower bound	Priority	Terminal
σ	0	0.03	-	1	-
μ	1	-	0.97	10	-
p_1	12	15	9	40	-
p_2	10	12	8	40	-
Δ_p	2	4	0.5	60	-
h_s	0.7	2	0	40	40
h_{dl}	0.5	2	0	20	60
u_c	0.38	1	0	2	-
Δu_c	0	-	-	4	-
u_1	0.24	1	0	1	-
Δu_1	0	-	-	2	-
u_3	0.05	1	0	0.24	-
Δu_3	0	-	-	0.24	-
u_4	0.40	1	0	0.6	-
Δu_4	0	-	-	0.6	-
u_7	0.15	1	0	0.6	-
Δu_7	0	-	-	0.6	-

Table 1: Objective of the NMPC. Priority is the corresponding weight to the variable in (4b)-(4f), and terminal is the corresponding terminal weight of (4g). Note that these values are relative according to the magnitude of the respective variable, and are scaled before applied in the optimization problem. If pre-stabilization is used, the priority to h_{dl}, h_s, u_4 and u_7 set to zero, as well as the terminal cost to h_{dl} and h_s.

constraints are violated. Also σ is more excited from the active usage of u_4 and u_7, which both effects the flow where σ is measured.

3.1 Control structure and objectives

The resulting specification of the NMPC is summarized in table 1, and (3) shows the resulting optimization problem.

$$\min_{u \in \mathbb{R}^n} \; f(\ell(u(0),\ldots,u(K-1),x(0),K),u(0),\ldots,u(K-1)) \tag{3a}$$

$$\text{s.t}$$

$$u_{min} \leq u_i(k) \leq u_{max}, \; i = 1,3,4,7,c \qquad k = 0,1\ldots K-1 \tag{3b}$$

where the constraints on the states are handled through penalty-functions. The objective function becomes

$$f(x(1),\ldots,x(\hat{K}),u(0)\ldots u(K-1)) = \tag{4a}$$

$$\sum_{k=1}^{\hat{K}} \frac{1}{\Delta t(k)}[q_{sigma}\|\sigma(k)-\sigma_{ref}\|_2 + q_{\mu}\|\mu(k)-\mu_{ref}\|_2 + \tag{4b}$$

$$q_{h_s}\|h_s(k)-h_{s_{ref}}\|_2 + q_{h_{dl}}\|h_{dl}(k)-h_{dl_{ref}}\|_2 \tag{4c}$$

$$+q_{p_1}\|p_1(k)-p_{1_{ref}}\|_2 + q_{p_2}\|p_2(k)-p_{2_{ref}}\|_2 \tag{4d}$$

$$+q_{\Delta_p}\|\Delta_p(k)-\Delta_{p_{ref}}\|_2] \tag{4e}$$

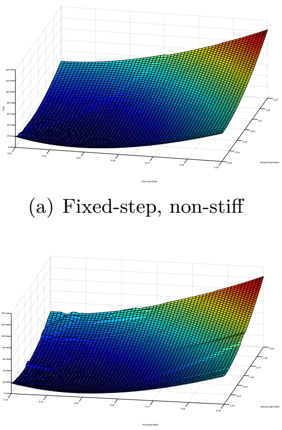
$$+\sum_{k=1}^{K}\sum_{i} r_i\|u_i(k)-u_{i_{ref}}\|_2,\ i \in \{1,3,4,7,c\} \tag{4f}$$

$$+e_{h_s}\|h_s(\hat{K})-h_{s_{ref}}\|_2 + e_{h_{dl}}\|h_{dl}(\hat{K})-h_{dl_{ref}}\|_2 \tag{4g}$$

Where e_{h_s} and $e_{h_{dl}}$ are the terminal weights of table 1. Note the introduction of the differentiated time vector, which holds the time interval between each time step. This is to be able to do a discrete integration. This is important when applying a variable-step solver, as the number of sample points returned by $\ell(u(0),\ldots,u(K-1),x(0),K)$ will depend on the dynamics of the system, which will change according to the state this is in.
For the choice of the control, and prediction horizon, figure 2 suggests that the slowest dynamics has a time constant of approximately 40 seconds. This choice proved to work well during simulations in the "Full MPC" case. For the pre-stabilized case pressures seems to be surprisingly slow. However as the pressure in the DL shows, they are closely correlated with the liquid levels, and it is likely that the dynamics of the pressures becomes faster when the liquid levels becomes high thus less space is available for the gas in the cyclones. Simulations proves that horizons down to 5 seconds stabilizes the model, however larger choices gives smoother trajectories of the valves. Therefore a horizon of 20 seconds is chosen. The number input sequences are divided into(input blocking) 5 blocks. The length of each input block is $\{10\%, 10\%, 20\%, 20\%, 40\%\}$ of the total length of the control horizon. Increasing the number of blocks does not give an significant improvement in closed-loop performance.

3.2 Analysing the optimization problem

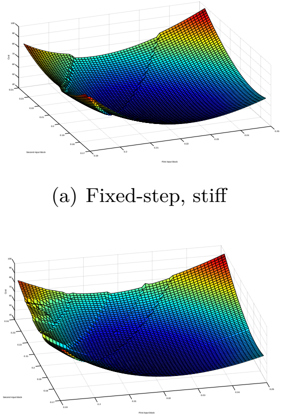
Before applying optimization algorithms to solve the NMPC problem, it is useful to investigate the properties of the problem. Two sets of inputs and measured states of the plant $x(0)$ are chosen from a simulation, one set where the plant is in a stiff condition, and one where this is not the case. Then the cost is evaluated at different points in this setting by holding the three first and the two last input blocks equal, varying one valve. It is the



(a) Fixed-step, non-stiff

(b) Variable-step, non-stiff

Figure 3: Cost surface of the NMPC optimization problem when applying fixed-step and variable-step solver when the plant is in a non-stiff configuration, and varying u_7 between 0.13 to 0.19.



(a) Fixed-step, stiff

(b) Variable-step, stiff

Figure 4: Cost surface of the NMPC optimization problem when applying fixed-step and variable-step solver when the plant is in a stiff configuration. The first block of u_7 is varied between 0.19 and 0.25, the second between 0.17 and 0.23

"Full MPC" approach that is investigated, as this is the numerically most challenging. The experiments are done using Matlabs *ode1* solver with a sampling period of 0.1 seconds in the fixed-step case, and Matlabs *ode23s* is used for variable-step simulations. The latter uses an absolute and relative tolerance of 10^{-2}, and a minimum and maximum step size of 0.01 and 0.2 seconds respectively. Figure 3(a) and 3(b) shows the resulting cost when applying fixed and variable-step ODE-solver respectively, in the non-stiff case. In the fixed-step case the cost surface is relatively smooth, although some minor discontinuities can be seen traversing the cost surface. Experiments show that the tested gradient-based SQP algorithm handles this problem well. When applying a variable-step solver significant numerical noise appears. Local minimums seems to exist in the top left corner of figure 3(b), which is challenging for all algorithms.

Figure 4(a) and 4(b) is the same experiment as figure 3(a) and 3(b), however the system is in a stiff setting. It is important to notice that even when using a fixed-step solver, the problem may still become discontinuous and non-smooth. This is because the model is highly non-linear due to phase transitions, logic operators is used in the implementation and the dynamics are stiff. Note that at least two minimum points exist, one at the bottom of the curve, and one on the edge next to this.

When using a variable-step solver, as in the non-stiff case, noise is introduced leading to a objective function that is even harder to differentiate. Also more local minima are introduced. Applying gradient-based SQP to this proves to make the algorithm fail, usually either getting stuck in a local minima or resulting in a extremely slow convergence. This is likely because large values of the gradient can occur when differentiating the discontinuities, making the algorithm try large steps in the wrong direction.

4 Derivative-free trust-region methods

DFO algorithms comes in a vast number of types and varieties. A common characteristic of these methods are that they start with an initial set of samples $Y \subset \mathbb{R}^n$ of the objective function $f : \mathbb{R}^n \mapsto \mathbb{R}^1$, and creates new points by trying to figure out where the optimum of $f(x)$ is by looking at the positions of $y^i, i \in [1, m]$ and the resulting functions values $f(y^i), i \in [1, m]$[5].

DFTRM are based on making a polynomial model $q(x)$ of the true objective function $f(x)$ around the current iterate x_k, k being the iteration index. This model is made from Y and its corresponding function values. The minimization is then performed on this model instead of the true objective function. A quadratic model $q(x)$ is chosen, since optimality conditions are simple, and inspection of figure 3(a) suggests that the objective resembles a quadratic function. Some important aspects are introduced in these algorithms, one of the most important is the concept of well-poisedness in Y. This refers to how far the model construction is from becoming degenerate. This has a direct implication on the accuracy of $q(x)$ compared to $f(x)$, and is usually handled in DFTRM by solving a second optimization problem which is to generate new points that ensures well-poisedness in Y, called the second sub-problem. Because this requires evaluation of the objective function and does not contribute directly to the minimization, poisedness becomes a necessary but costly issue[22].

To compensate for lack of accuracy in $q(x)$, a Trust-Region (TR), denoted by a radius Δ, is applied. This is the hypersphere around x_k where $q(x)$ is trusted to be accurate. The reduction in $q(x)$ found in the first sub-problem is compared with the actual reduction of $f(x)$. If the difference is large, the radius of the TR is shrunk to improve the accuracy of $q(x)$ within the TR. It is also common to use a minimum TR-radius as a stopping criterion, such that the optimization is stopped if $\Delta \leq \Delta_{end}$.

Further, the algorithms need enough samples initially to build the initial model $q(x)$. Because this uses an interpolation method, the number of samples required for a fully determined model is $m = \frac{1}{2}(n + 1)(n + 2)$, where n in the number of MV. Because of the large computational resources required for this, it is definitely a challenge to handle this in many real-time applications.

4.1 Wedge, UOBYQA, Condor and BOBYQA

The BOBYQA algorithm is the successor of COBYLA, UOBYQA, and NEWUOA[19]. The algorithms main structure is quite similar to that of UOBYQA, as of solving two sub-problems. Its main difference from UOBYQA is that it builds the model from an under-determined sample set, and support bound constrains on the MV. $q(x)$ is built using the minimum Frobenius-norm update[17], which takes up space in the undetermined degrees of freedom by minimizing the norm between the current $\nabla^2 q(x)$ with that from the previous time step. It is optional to use between $m = n + 2$ to $\frac{1}{2}(n + 1)(n + 2)$ sample points, however[19] recommends $2n + 1$, which is used during the simulations. This choice makes $\nabla^2 q(x)$ a diagonal matrix initially[19]. The implementation used in the simulation, is that of the NLOpt-package[9].

UOBYQA is a well-known DFO algorithm for unconstrained optimization. It is claimed to be robust against noise in the objective function[15], and best suited for problems with less than $n = 20$ free variables[16]. $q(x)$ is built from a fully determined interpolation, and to ensure that Y does not become degenerate a second sub-problem is solved to ensure well-poisedness.

The simulations are performed using Condor, which is an extension of UOBYQA[2], exploiting parallel computing, added tolerance towards noisy objective functions and, maybe most interesting for the purpose of NMPC, handling of both bound, linear and non-linear constraints. As parallelism is not applied in the simulations, the main difference from UOBYQA concerning the simulations is that another method for solving the first sub-problem is used in Condor to handle constraints.

The Wedge algorithm builds the interpolation model from a fully-determined sample set using LU-factorisation[11]. The algorithms approach to poisedness is that it guarantees that the point found as the solution of the first sub-problem can replace a pre-determined point in the sample set Y, without making this set degenerate. To guarantee that the TR-step fulfils this condition, an additional constraint is introduced in the sub problem. The so-called constraint is designed such that the new point will ensure a certain distance from degeneracy, and this distance is specified with the parameter γ. From [11] it is clear that the purpose of the algorithm is not specifically to optimize noisy functions. On the other hand, DFO algorithms tends in general to be more suited to noisy optimization than gradient-based algorithms.

4.2 Improved warm-start

One of the biggest obstacles in NMPC is to solve the optimization problem fast enough for real-time performance. This is especially true for DFO which is known to require a large number of function evaluations, where the initial evaluations are often dominating. Simulations shows that algorithms using a fully determined sample set tends to perform better when numerical noise and discontinuities are present. If the number of points required to be evaluated initially for each time step can be reduced significantly without an extensive increase in the number of algorithm iterations, this would be a step in the right direction to make DFO more applicable in a real-time NMPC setting. As the NMPC objective function is known to typically be similar in time step k compared to time step $k-1$, this information may be exploited to build an accurate initial model and still use few function evaluations for this. Therefore a modification to the Wedge-algorithm is proposed that exploits the nature of the NMPC problem to reduce the number of function evaluations in the initial model building made at the beginning of each time step k.

The idea behind the modification is to take warm-start a step further than just using the previous optimal solution as initial guess, by also warm-starting the approximation model $q(x)$. Because the sample set becomes "zoomed" in on the solution as the algorithm iterates, a sample set purely used for building of the initial model at each time step is introduced, Y_{Init}. This set, with its corresponding function values f_{Init}, are saved from one time step to the next. Then, only a fraction of these points are replaced and the corresponding function values are updated at each time step. After the model building is finished this sample set is copied into a "local" sample set used in the iterations of the algorithm, Y_{Iter} and f_{Iter}. This way the points Y_{Iter} will be "zoomed" in on the solution, Y_{Init} still having a relatively large distance between its points. When the algorithm is finished for the current time step, Y_{Iter} and f_{Iter} are discarded. When some of the points in Y_{Init} are updated in the next time step, the start point x_0(i.e. the solution from the previous time step) is chosen as the centre for these new points. This way, as the time steps progresses the points in Y_{Init} will slowly track the positions of the current iterate.

To ensure that all points are updated at regular intervals, the indices of Y_{Init} are divided into n_s sub-sets, $\mathcal{S}_i, i \in 1 \ldots n_s$, and a sample points affiliation to a given index is constant for the scope of the controller. The points belonging to the indexes to a given sample set is updated at each time step, cycling such that no set is associated with samples older than n_s time steps. The initial model is then built from Y_{Init}, resulting in a model fulfilling the interpolation condition

$$q_k(y_i) = \begin{cases} f_k(y_i) & \forall\ i \in \mathcal{S}_k \\ f_{k-1}(y_i) & \forall\ i \in \mathcal{S}_{k-1} \\ f_{k-2}(y_i) & \forall\ i \in \mathcal{S}_{k-2} \\ \vdots & \\ f_{k-n_s}(y_i) & \forall\ i \in \mathcal{S}_{k-n_s} \end{cases} \tag{5}$$

and k denotes the current time step. For a fully determined algorithm this implies that only $\hat{m} = \frac{(n+1)(n+2)}{2n_s}$ samples needs to be evaluated initially at each time step. This will come at the cost that the initial model may be less accurate, possibly requiring more function evaluations in the proceeding iterations. However the motivation for this approach is the observation that in the case-study the difference between $q(x)$ and $f(x)$ is often rather large both in the initial model building and iterations of Wedge. Especially the magnitude of the gradient does usually not decrease at all as the algorithm progresses trough iterations, which would be expected if the accuracy of $q(x)$ is good. Still Wedge seems to perform well during simulations, and this suggests that DFTRM is quite robust against model error.
Also the findings in [8] supports this observation, as this article experimented with a DFTRM without any poisedness-ensuring mechanism. This caused a large error in $q(x)$, still the algorithm was reported to perform well. Further [19] has reported that BOBYQA works with large difference in the Hessian of $q(x)$ compared to that of $f(x)$, but the algorithm still seemed to work well. This encourages the approach of re-using old sample points and function values of the objective function, although this results in an known model error.
Because it is known that the accuracy of $q(x)$ is closely related with poisedness, this is used in the modification to Wedge to ensure that the algorithm gets a good start at each time step. This is accomplished by the use of an algorithm that adds sample points to an existing sample set, choosing the new points such that the set becomes well-poised. This algorithm can be found in [5], where it is known as algorithm 6.5. By removing the points to be updated from Y and feeding the set to this algorithm, the algorithm ensures that the set is filled up and is well-poised. Further, at the first time step the algorithm uses the same initialisation procedure as the original Wedge algorithm.

5 Subsea oil-gas separation process simulation results

Both the pre-stabilized case and the non-stabilized, referred to as "Full MPC" approach, are simulated in closed-loop in this section. The disturbance sequence applied can be seen in figure 5. Warm-start is also applied by always starting the optimization at the previous solution to exploit that the solution is similar from one time step to another. The plant model that is to be controlled is simulated with Matlabs *ode1* solver at a sampling period of 0.01 seconds, and the sampling frequency of the controller is 0.2 seconds, i.e. a new input sequence is calculated with this interval. The prediction model is tested with the same fixed and variable-step solvers as used in section 3.2. That the plant and prediction model is simulated differently, will induce a model-mismatch between the predictions and the plant. Such mismatches are always present in real-world applications, and should therefore make the simulations more realistic. Further the Δ_{end} stopping criterion is set to 10^{-3} for all the DFO algorithms.

The model, both in the prediction configuration and the plant, proved to be very numerically challenging. Especially a negative rate of change in δ seems to make the ODE-solver tend to return infinite states or "Not A Number"(referred to as undefined points). Investigations shows that in these cases the set of inputs where the model is defined, becomes very small, especially when using the variable-step solver. The objective function also becomes very non-smooth and quickly changing from one time step to the next, and an optimal solution in one time step may become undefined in the next. None of the algorithms handles undefined starting points. In these cases a last-resort solution must be used, by implementing the solution from the previous time step and hope that this eventually becomes defined again as the simulation progresses. For the "Full MPC" case with variable-step solver the problem becomes extremely numerically difficult, and non of the algorithms performs well, thus this case is omitted.

Table 2 summarize the results for the five algorithms that is tested.

The SQP-algorithm tested, is the gradient-based implementation *fmincon()* in Matlabs optimization toolbox, using finite-differences gradient approximations. This algorithm experiences severe difficulties during the simulations, as the algorithm takes large steps in bad directions, and returns solutions with a cost in the magnitude of several thousand. This is likely because finite-differences is applied to non-smoothness and discontinuities, and significant errors occurs. This is handled by restarting the algorithm at different starting points in these cases, implementing the solution from the previous time step if this does not succeed. However when using variable-step solver the algorithm did never recover, and the simulation failed, thus this case is omitted in table 2. In addition the previous described handling of undefined starting points is implemented.

5.1 BOBYQA

BOBYQA is limited to use a maximum of $k_{max} = 600$ function evaluations for the "Full MPC" case, and $k_{max} = 216$ for the pre-stabilized case during the simulations. The most interesting result from using the fixed-step ODE-

Algorithm	MPC	Fs/Vs	W/C sol.	Av. sol.	Av. evals.	S.S. evals.
SQP	P.S.	Fs	176.7	5.9	178.6	93
BOBYQA	P.S.	Fs	10.2	1.6	117	80
BOBYQA	P.S.	Vs	28.7	2.7	80	71
Condor	P.S.	Fs	25.5	2.2	247	252
Condor	P.S.	Vs	11.6	1.2	245	249
Wedge	P.S.	Fs	16.9	2.5	168	143
Wedge	P.S.	Vs	89.5	3.9	164	151
Wedge*	P.S.	Vs	53.7	1.1	197	150
ModWedge	P.S.	Fs	11.7	1.5	89	76
ModWedge	P.S.	Vs	66.1	4.7	90	76
SQP	Full	Fs	194	38.6	101	74
BOBYQA	Full	Fs	119.6	3.4	213	212
Condor	Full	Fs	42.9	7.2	463	456
Wedge	Full	Fs	140.3	8.7	389	382
ModWedge	Full	Fs	73.8	11.1	165	162

Table 2: Summary of the results. "W/C sol." is the highest cost the respective algorithm finds during simulation, while "Av. sol." is the average cost. "Av. evals. and "S.S. evals." are the average number of function evaluations used during the whole simulations, and during the first 30 seconds while the disturbance is constant, respectively. The largest number of function evaluations is equal to k_{max} of the respective algorithm. Wedge* $\gamma_1 = 0.75$ instead of 0.5.

solver with the algorithm is the number of function evaluations. Table 2 shows that as long as the disturbance is small, thus the objective function being relatively smooth and similar from one time step to the next, the number of function evaluations increases slowly with the number of MV.

The simulations also indicates that the solutions found by BOBYQA in the "Full MPC" case is not any worse than that of the other algorithms, at least not while the disturbance is moderate. It is hard to tell why this is the case, however it is known from [19] and [18] that the difference between the Hessian of $f(x)$ and that of $q(x)$ often is relatively large. The success of the algorithm may then be interpreted as although the mismatch between $f(x)$ and $q(x)$ can be substantial, the minimum point is similar. It is known that $f(x)$ is at least to some extent convex close to the solution and that this is usually quite close to the starting point for the optimization x_0. As this point is used as a point in the interpolation in the initial model, is it reasonable to believe that $q(x)$ is accurate around the actual solution. Because Δ_0 is chosen relatively large, all other points in the initial sample set will have a larger function value due to the convex tendencies of $f(x)$. As the Hessian of $q(x)$ is a diagonal matrix initially, this will then become positive definite if these assumption holds, and the algorithm will make steps towards the actual solution even though the model mismatch is large.

Further, as the initial model is built up from fewer samples than is the case with fully determined interpolation, less information is also stored. As the algorithm progresses, fewer iterations will be needed before the model is renewed by "fresh" sample points. This can give a better ability for the model to adapt to local non-linearities as the trust-region moves.

The simulations using variable-step solver shows that this has a great impact on BOBYQAs performance. Considering the worst-case cost of 28.7, instead of 10.2 in the case of fixed-step solver. The cause of this is that the implementation used will exit and return the current best solution if an undefined point is encountered.

Noteworthy is also the reduction in the number of function evaluations when the variable-step solver is applied. This is likely because the TR is reduced quickly when numerical noise is present, thus reaching Δ_{end} after fewer evaluations than when the objective is smooth and quadratic, i.e. the algorithm terminates prematurely.

5.2 Condor

Condor is set to stop after 100 iterations, and rarely stops due to reaching Δ_{end}. This causes the algorithm to not reduce the number of function evaluations when x_0 is close to the actual solution, unlike the case with BOBYQA. Therefore the number of function evaluations are almost as large during steady-state as during transients.

The results using Condor for the pre-stabilized MPC are better compared to BOBYQA when using the variable-step solver, and worse when using fixed-step, although using significantly more function evaluations. This reflects that Condor is designed for noisy optimization. In the "Full MPC" case, the algorithm has the best worst-case performance, however the average solution is second best. Again, Condor tends to handle the discontinuities caused by severe disturbances better than BOBYQA, it is however out-performed in smooth cases.

5.3 Wedge

Wedge is limited to $k_{max} = 461$ and $k_{max} = 246$ function evaluations for the "Full MPC" and the pre-stabilized cases respectively. It seems to perform equally or better compared to Condor, when the disturbance is moderate, which can be seen in [6]. It is also important to note that the algorithm uses significantly fewer evaluations than Condor, as it tends to reach Δ_{end} quicker. Also the variance in the inputs tends to be smaller with Wedge than with Condor. Although desirable, this "filtering" is actually a side-effect when the algorithm does not find the exact solution. The solution will become more similar to the previous which is the current start point, thus less variations will occur. As expected the performance of Wedge suffers when the disturbance and thus the non-smoothness becomes severe, especially clear from the worst-case solution in the "Full MPC" approach. Also the algorithm performs better on the pre-stabilized case, which is known to be less numerically difficult.

Because Wedge is a Matlab-implementation, it is easier to investigate the behaviour of the algorithm. It is observed that the poisedness-improving mechanism is rarely invoked, especially when good progress is achieved such that trial-points are relatively far from each other. When the algorithm starts close to the solution, trial-points are closer thus the wedge-constraint becomes active more often. This is intuitive and according to [8].

Further it seems that the Hessian of $q(x)$ is rarely positive-definite in the case-study. The reason for this is unknown, as this would be expected at least under steady-state operation(see figure 3(a)). Also the absolute value of the gradient of $q(x)$ does not decrease as the algorithm progresses. This may suggest that the model is not very accurate, as the gradient would be expected to decrease when approaching the solution.

When applying the variable-step solver, the performance up to around 110 seconds(see [6]) is largely the same as BOBYQA, continuing the tendency that Wedge handles moderate disturbances well. As the algorithm progresses, it often tends to terminate early after very few iterations due to the Δ_{end} stopping criteria.

γ_1 is a parameter deciding how fast the TR is to be shrunk if the accuracy of $q(x)$ is poor. Choosing this large, will make the TR shrink less, and this proves to make the algorithm more robust against noise than choosing it smaller, and resulted in the best solutions with respect to optimality. However much more variance is introduced in the resulting input sequences. The initial TR Δ_0 has a similar effect. This is likely because the algorithm becomes more robust towards change in the objective as Δ_0 increases, as the optimum is more likely to be included within the initial TR. However because the maximum number of iterations are reached before the algorithm has converged, a less exact solution will be found. On the other hand, choosing Δ_0 small works well as long as the objective function does not change significantly from one time step to the next, as the initial starting point will be close to the solution and the algorithm can "zoom" in on the exact solution.

5.4 Wedge with improved warm start

Figure 5 and 6 shows the optimized input sequence of the valves, and trajectory of the states respectively when the modified Wedge-algorithm is applied to solve the NMPC problem.

The modified version of Wedge is simulated with the same parameters as the original version is, except a larger value on γ_1. This is increased to 0.85 without inducing more variance in the optimal solution, as is the case in the original algorithm. This is likely to be because it builds the model also on previous data, thus creating a filtering effect.

For the pre-stabilized, fixed-step case, the modified algorithm finds solutions with costs in the same magnitude as the other algorithms. For the variable-step case the average solutions are between the two simulations using the original algorithm. In the "Full MPC" case the average solution is the worst of the DFO algorithms, although the worst-case solution is only superseded by Condor. However in all cases the number of function evaluations are very few, especially in the "Full MPC" case, where the reduction is substantial compared to the original algorithm. This suggests that the concept of recycling old function evaluations is indeed a promising way to go.

5.5 Summary and discussion

There is no clear winner among the algorithms, as all seem to have their strong and weak properties. An important observation is that there are large differences in their ability to find an optimum, or even near-optimal solution within the tolerances and iteration limits. It is interesting that the gradient-based SQP did fail in all cases, probably due to finite differences applied to non-smoothness, and therefore had to rely on last-resort solutions. All DFO algorithm did handle at least all fixed-step cases, proving that they have a part to play in NMPC. The simulations shows that the choices of tuning parameters of the algorithms is of great significance, as shown with Wedge

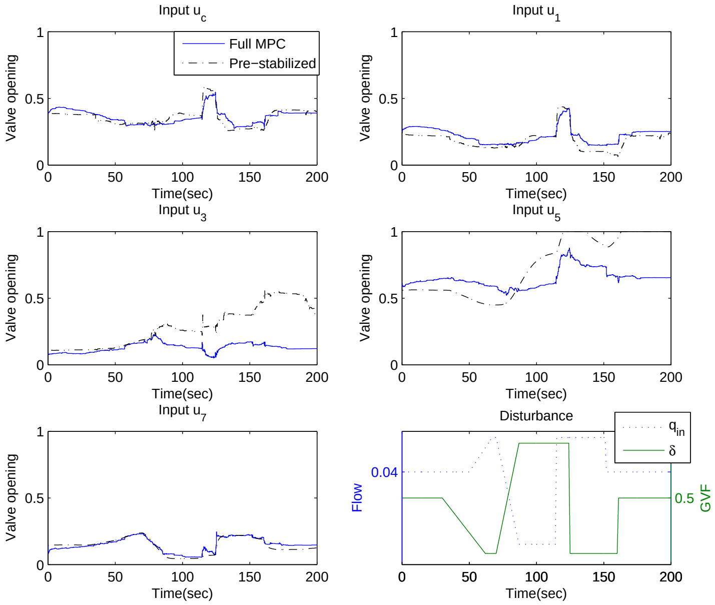


Figure 5: Control inputs using modified Wedge, and prediction model simulated with fixed-step solver. In the pre-stabilized case u_3 and u_4 is controlled by internal feedback loops.

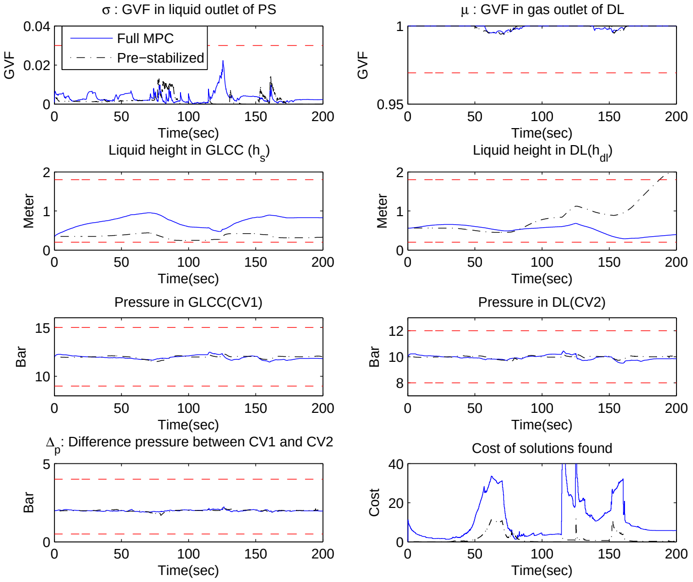


Figure 6: Controlled states using modified Wedge, and prediction model simulated with fixed-step solver. In the pre-stabilized case, h_{dl} and h_s is controlled by internal feedback loops.

where the performance greatly improves by increasing the reduction factor of the TR, however this results in increased variances of the control inputs and worse performance during steady-state. Comparing BOBYQA to the gradient-based SQP algorithm, BOBYQA actually seems directly comparable, both in computational consumption and accuracy. Condor stands out as the best suited against noise and discontinuities induced when using the variable-step solver, and when the fast dynamics are excited by large disturbances. Although using relatively many function evaluations, it does not seem to require more even if numerical noise and discontinuities is severe. In the case of a real-time application such as MPC, the worst-case performance is of primary significance as computational resources must be reserved for this case. Although not clear from table 2, BOBYQA and both versions of Wedge uses k_{max} function evaluations in the worst case(details in [6]), thus Condor is attractive.

Regarding real-time performance, all experiments are performed on one CPU/core. This raises a significant question, as although BOBYQA and the modified Wedge algorithm is the clear winner with respect to the number of function evaluations used, fewer of this are evaluated in the initial sample set than in Condor and Wedge. If evaluations is to be spread over several CPUs, it is this initial work that is the most suited for distributed processing. If using, say 16 CPUs which may be accomplished in a Digital Signal Processor (DSP), the algorithms using fully determined sample sets may become the fastest with respect to time.

Considering that Condor requires approximately 110 evaluations after the initial sample set building in the "Full MPC" case, all the initial 351 evaluations may be spread across 16 CPUs. Considering perfect parallelisation, this gives a processing time consumption similar to $\frac{351}{16} + 110 = 132$ evaluations, while BOBYQA using an initial sample set of 51 samples and a total of 212 evaluations requires a time similar to $\frac{51}{16} + 161 = 202$ evaluations. This gives Wedge and Condor a clear advantage. When also considering the remark above that computational resources must be reserved for the worst-case, this will give Condor and Wedge an even larger advantage, as BOBYQA in this case requires 600 function evaluations. By also considering using a multi-core GPU for parallel computing as shown in [20], the computational time required to evaluate the initial sample set can be reduced even more.

Further, more sophisticated techniques for recycling previous function evaluations in the modified Wedge algorithm is likely to improve the performance significantly. Such techniques can be to apply the minimum-norm solution used in BOBYQA to make $\nabla^2 q(x)$ similar to that of the previous model, and that way carry information between time steps. [5] describes over-determined interpolation, and this may be applied in NMPC by re-cycling old sample sets and form an over-determined interpolation in the current time step.

6 Conclusions

The simulations has shown that numerical issues where the tested finite-differences gradient-based SQP has difficulties can occur both when using variable and fixed step ODE-solver. All DFO-algorithms tested handles these issues better, which emphasises that these algorithms have robustness that is desirable in NMPC.

Although all DFO algorithms have their strengths and weaknesses, it currently seems that BOBYQA and Condor are two algorithms that may be interesting for use in NMPC. For the case where the noise and discontinuities are not severe, however enough to make gradient-based algorithms fail, BOBYQA seems like a very promising candidate. It requires a low number of function evaluations that is comparable with finite-difference gradient-based SQP, thus is especially attractive for use where only a single core is available.

If the numerical noise and discontinuities are severe, Condor is attractive. It requires more function evaluations, but is significantly more robust against such numerical difficulties. As it is more suited for parallelism, and already has a parallelisation mechanism implemented, it is also interesting for use on multi-core systems.

A novel modification to the Wedge-algorithm shows that exploiting the nature of the NMPC problem definitely can give advantages with respect to computational time consumption, by recycling function evaluations from previous time steps.

If the optimization problem is smooth and differentiable, gradient-based algorithms are still preferable. However as this is not always the case, DFO is a good supplement for ensuring robustness in NMPC.

References

[1] H. Al-Duwaish and W. Naeem. Nonlinear model predictive control of hammerstein and wiener models using genetic algorithms. In *Control Applications, 2001. (CCA '01). Proceedings of the 2001 IEEE International Conference on*, pages 465–469, 2001.

[2] F.V. Berghen and H. Bersini. CONDOR, a new parallel, constrained extension of powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. *Journal of Computational and Applied Mathematics*, 181:157–175, September 2005.

[3] L.G. Bleris, J. Garcia, M.V. Kothare, and M.G. Arnold. Towards embedded model predictive control for system-on-a-chip applications. *Journal of Process Control*, 16(3):255–264, 2006.

[4] W. Chen, X. Li, and M. Chen. Suboptimal nonlinear model predictive control based on genetic algorithm. In *Intelligent Information Technology Application Workshops, 2009. IITAW '09. Third International Symposium on*, pages 119–124, 2009.

[5] A.R. Conn, K. Scheinberg, and L.N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics and the Mathematical Programming Society, 2009.

[6] J. S. Dæhlen. Derivative-free optimization in NMPC. Master's thesis, NTNU, 2013.

[7] M. Diehl, H.J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In Lalo Magni, DavideMartino Raimondo, and Frank Allgöwer, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 391–417. Springer Berlin Heidelberg, 2009.

[8] G. Fasano, J.L. Morales, and J. Nocedal. On the geometry phase in model-based algorithms for derivative-free optimization. *Optimization Methods and Software*, 24(1):145–154, 2009.

[9] S.G. Johnson. The NLopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`, July 2012. Accessed: 12/12/2012.

[10] D. Koller and S. Ulbrich. Optimal control of hydroforming processes. *Proceedings in Applied Mathematics and Mechanics*, 11(1):795–796, 2011.

[11] M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming*, 91:289–305, 2002. 10.1007/s101070100264.

[12] P. Norgren. Compact subsea separation unit: Nonlinear model predictive control and nonlinear observers. Master's thesis, NTNU, 2011.

[13] R. Oeuvray and M. Bierlaire. Boosters: a derivative-free algorithm based on radial basis functions. *International Journal of Modelling and Simulation*, 29(1):26, 2009.

[14] C. Onnen, R. Babuška, U. Kaymak, J.M. Sousa, H.B. Verbruggen, and R. Isermann. Genetic algorithms for optimization in predictive control. *Control Engineering Practice*, 5(10):1363 – 1372, 1997.

[15] M.J.D. Powell. UOBYQA: Unconstrained optimization by quadratic approximations. DAMTP 2000/NA14, December 2000.

[16] M.J.D. Powell. UOBYQA: Unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[17] M.J.D. Powell. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming*, 100(1):183–215, 2004.

[18] M.J.D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In G. Pillo, M. Roma, and Panos Pardalos, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 255–297. Springer US, 2006. 10.1007/0-387-30065-1_16.

[19] M.J.D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 2009.

[20] A. Sadrieh and P. A. Bahri. Application of graphic processing unit in model predictive control. In M.C. Georgiadis E.N. Pistikopoulos and A.C. Kokossis, editors, *21st European Symposium on Computer Aided Process Engineering*, volume 29 of *Computer Aided Chemical Engineering*, pages 492 – 496. Elsevier, 2011.

[21] H. Sarimveis and G. Bafas. Fuzzy model predictive control of non-linear processes using genetic algorithms. *Fuzzy Sets and Systems*, 139(1):59 – 80, 2003.

[22] K. Scheinberg and P. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM Journal on Optimization*, 20(6):3512–3532, 2010.

[23] B. Tlili, F. Bouani, and M. Ksouri. A derivative-free constrained predictive controller. In *Proceedings of the 10th WSEAS international conference on Systems*, pages 360–365. World Scientific and Engineering Academy and Society (WSEAS), 2006.