Study of early termination of MPC Algorithms

Gustav Henriks

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6023
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2016

Abstract

With a steady development of technology, the use of Model Predictive Control (MPC) has become more and more popular since the computation time has gone down. With this increase, a need for determining which MPC algorithm is good for solving a certain type of MPC problem has occurred which would facilitate the choice of algorithm and also could increase the performance. One of the determining aspects is if a solver has the possibility to terminate early (stop the algorithm before it has performed all of its iterations), for example if it is placed on an embedded system with strict real-time bounds. With the use of an MPC Benchmarking suite available at ABB Corporate Research Switzerland, the early termination of MPC algorithms has been investigated. With the usage of 19 benchmarks and 3 different solvers that uses Interior point method, Gradient descent and Active-set method a large number of results have been looked through with the help of different Machine Learning methods. The result has been classified as good or bad performance when terminated early and different models have been fitted to predict this data. From this a group of key-features have been attempted to get extracted to see if there is a possibility on beforehand to tell if a control problem and a certain solver can be early terminated. Important features that were found were mostly concerning whether the control input u was under constraint or not. Good results were especially achieved for a machine learning model based on the Active-set solver qpOASES which could give good indications on whether a certain problem could get early terminated or not.

Acknowledgements

I would like to thank my supervisors Joachim Ferreau at ABB Switzerland and Pontus Giselsson at the Department of Autmatic Control at LTH, Lund University, for their support and guidance throughout this thesis.

Contents

1

Introduction

1.1 Background

The use of Model Predictive Control has been under steady increase since its first application in the 70s. Along with its increasing popularity, the speed of the controller has become quicker and quicker which makes it more attractive to put in an embedded system. These systems can be placed in environments where strict real-time constraints are to be upheld which could be of a problem for the MPC.
Various optimization algorithms exist that can solve a linear MPC problem and the process of how these work is something that differs from algorithm to algorithm. This also make them better or worse suited to solve a certain type of problem. When this combination of solver and problem is not a good match there is a risk that the problem might not be solved within the real-time constraints, or in other words, the solver does not converge within given time limits. However, if the result given from the solver at a not fully converged stadium does not differ or barely differ from the fully converged solution, a bad match of algorithm and solver could still be a doable combination. By looking at the problem that is to be solved, it would be of interest to determine if the problem together with a solver is feasible on beforehand, for example by noting a certain set of key-features. Therefore a feature-selecting machine learning approach will be taken upon this thesis to see whether this is possible or not.

1.2 Objective

The purpose of this thesis is to investigate early terminated MPC algorithms; both from a closed-loop performance perspective as well as how good the algorithm itself behaves. The methods to evaluate the performance will be to see how easy it is to create a machine learning model and predict if the algorithm will behave well or badly depending on what problem it is solving.

1.3 Outline

In Section 2 the theory behind the algorithms used as well as for Model Predictive control will be presented. Further in Section 3 the ABB Benchmarking Suite will be described briefly, then the methods of processing the simulation results including the Machine Learning methods used for this. In Section 4 the results from the machine learning part is presented and discussed upon, and lastly the findings of the thesis are concluded and put into perspective to see if any future research can be done in Section 5.

2

Optimization and MPC

2.1 Quadratic programming

The function that most of the context in this thesis will be based upon is the function in (2.1) which will be minimized by a number of chosen algorithms.

$$\begin{aligned} \text{Min } q(x) = & \frac{1}{2}x^T Hx + x^T c + q_0 \\ \text{subject to: } & Ax \geq b, \\ & A_{eq}x = b_{eq}. \end{aligned} \tag{2.1}$$

This is the convex quadratic objective function , where H is a positive semi-definite symmetric matrix, c, x are vectors in $\mathbb{R}^n$ and A, A_{eq} are fixed matrices in $\mathbb{R}^{m \times n}$. x is the targeted variable that is to be minimized and q_0 a constant [12].

2.1.1 KKT-conditions

Conditions that are necessary for a local minimum to (2.1) are the KKT-conditions (Karush-Kuhn-Tucker Conditions). These are related to the Lagrangian function of the quadratic problem. The Lagrangian function is defined as in (2.2) [12]

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i=\mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) \tag{2.2}$$

where the scalar λ is the Lagrange multiplier, $f(x)$ the function to be minimized (which for our case makes $f(x) = q(x)$ from (2.1)) and $c_i(x)$ its constraints with $i \in \mathcal{E}$ referring to the equality constrains and $i \in \mathcal{I}$ to the inequality constraints. Using this function, a set of conditions can be formed, see (2.6).

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \tag{2.3}$$
$$c_i(x^*) = 0, \quad \text{for all i} \in \mathcal{E} \tag{2.4}$$
$$c_i(x^*) \geq 0, \quad \text{for all i} \in \mathcal{I} \tag{2.5}$$
$$\lambda_i^* \geq 0, \quad \text{for all i} \in \mathcal{I} \tag{2.6}$$
$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all i} \in \mathcal{E} \cup \mathcal{I} \tag{2.7}$$

where x^* is the local solution to (2.1). In (2.7) either constraint i is active or $\lambda_i^* = 0$. These conditions will be used later on in the thesis to check if a solution is accurate enough and this is done by seeing if the conditions in (2.3)–(2.7) are fulfilled or if they are within a certain tolerance value.

2.2 Algorithms

A lot of different algorithms exist that solve the quadratic programming problem that the MPC handles. The way they solve the problem is also different and therefore there is a need to distinguish them.

2.2.1 Gradient methods

The Gradient method, or maybe more known as the Steepest Descent method is a very simple optimization algorithm that many other algorithms are based upon. In brief it can be seen in (2.8) where the function is updated until convergence.

$$\theta_{i+1} = \theta_i - \alpha \nabla_{\theta_i} \mathrm{J}(\theta_i) \tag{2.8}$$

where J is the function that is to be minimized, α is the step size and θ is the targeted variable [9].

Emerging from this comes a type of algorithm that has been investigated in this thesis: Proximal gradient descent. This algorithm works as following, given

$$\text{minimize } l(x) + \psi(x) \tag{2.9}$$

a function

$$\text{prox}_\psi(y) = \underset{x}{\text{argmin}} \{ \psi(x) + \frac{1}{2} ||x - y||_2^2 \} \tag{2.10}$$

is defined where $\psi(x)$ is a convex function and $l(x)$ a smooth function. Function (2.10) is then used in (2.11) to create the proximal gradient method.

$$x^{k+1} = \text{prox}_{t\psi}(x^k - t \nabla l(x^k)) \tag{2.11}$$

where t is the step size which is relating to ∇l.

From (2.11) the Fast Dual Proximal Gradient Method can be derived. This algorithm solves the Dual problem to functions of the same form as (2.1), but it can only handle the equality constraint in (2.1) and therefore A refers to A_{eq} in this section. The Dual problem is, simply put, a reformation of (2.1) which is maximized instead of minimized. Firstly, the dual variable μ is introduced for the equality constraint in (2.1), $A_{eq}x = b_{eq}$. The dual problem then becomes (2.12)

$$\text{minimize } d(\mu) + g^*(\mu) \tag{2.12}$$

where $d(\mu)$ is the dual function

$$d(\mu) := f^*(-A^T\mu). \tag{2.13}$$

and f^*, g^* are conjugate functions determined as

$$f^*(y) = \sup_x \{y^T x - f(x)\} \tag{2.14}$$

Functions (2.10)–(2.13) are then used in the Fast Dual Proximal Gradient Method seen below.

Algorithm 1 Fast Dual Proximal Gradient Method

Set: $\mu^0 = \mu^{-1}, \beta^0 = 0$
for $k \geq 0$ **do**
$v^k = \mu^k + \beta^k(\mu^k - \mu^{k-1})$
$x^k = \underset{x}{\operatorname{argmin}}\{f(x) + (v^k)^T Ax\}$
$\mu^{k+1} = \operatorname{prox}_{\frac{1}{L}g^*}\left(v^k + \frac{1}{L}Ax^k\right)$
end for

β^k is picked to achieve a fast conversion and $L = ||AH^{-1}A^T||_2^2$ is the Lipschitz constant. For a more detailed explanation how the algorithm works, see [4].

QPgen: A publicly available algorithm that implements Algorithm 1 is QPgen which has been used in the simulation part in this thesis. The mayor part of the algorithm is the use of preconditioning to transform ill-conditioned optimization problems. This is done by adding an invertible matrix to the inequality constraints in (2.1) on both sides which makes it possible to alter the constraint to make it more efficient to handle [4]. This algorithm will be used in the simulations in both Dual and Primal mode.

2.2.2 Interior point methods

Interior point methods are in general performed by removing the inequality constraints from (2.1) and instead penalizing constraints in the objective function. The resulting equality-constrained problem is then solved by the help of the Newton method [7]. The Newton method is a simple optimization method that can be seen in (2.15) where the process of finding the next step for a function f is shown.

$$x_{k+1} = x_k - \frac{\nabla f(x_k)}{\nabla^2 f(x_k)} \tag{2.15}$$

In this thesis, the algorithm used is called the Primal-Dual Interior Point method. The process of how this one works will be described hereby. Starting from (2.1) the KKT conditions are computed and become as in (2.16)–(2.19),

$$Hx - A^T\lambda + c = 0, \tag{2.16}$$
$$Ax - b \geq 0, \tag{2.17}$$
$$(Ax-b)_i\lambda_i = 0, \qquad i = 1,2,...,m \tag{2.18}$$
$$\lambda \geq 0. \tag{2.19}$$

A slack vector $y \geq 0$ is introduced which turns (2.16)–(2.19) into

$$Hx - A^T\lambda + c = 0, \tag{2.20}$$
$$Ax - y - b = 0 \tag{2.21}$$
$$y_i\lambda_i = 0, \qquad i = 1,2,...,m, \tag{2.22}$$
$$\{y,\lambda\} \geq 0. \tag{2.23}$$

Furthermore, a complementary measure μ is introduced,

$$\mu = \frac{y^T\lambda}{m} \tag{2.24}$$

Next up, a reformation of (2.20)–(2.23) is necessary to construct the primal-dual method:

$$F(x,y,\lambda;\sigma\mu) = \begin{bmatrix} Hx - A^T\lambda + c \\ Ax - y - b \\ \mathcal{Y}\Lambda e - \sigma\mu e \end{bmatrix} = 0, \tag{2.25}$$
$$\{\sigma,\mu\} \geq 0. \tag{2.26}$$

with

$$\mathcal{Y} = diag(y_1, y_2, ..., y_m), \qquad \Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_m), \qquad e = (1, 1, ..., 1)^T$$

here, $\sigma \in [0, 1]$. By fixing μ and using Newton's method on (2.25), a linear system is obtained:

$$\begin{bmatrix} H & 0 & -A^T \\ A & -I & 0 \\ 0 & \Lambda & \mathcal{Y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} Hx - A^T\lambda + c \\ Ax - y - b \\ \mathcal{Y}\Lambda e - \sigma\mu e \end{bmatrix} \tag{2.27}$$

Now the Primal-Dual Interior Point Algorithm can be presented, see Algorithm 2.

Algorithm 2 Primal-Dual Interior Point Algorithm

Given (x^0, λ^0, y^0) with $(x^0, y^0) \geq 0$;
set $k \leftarrow 0$ and $\mu_0 = (y^0)^T \lambda^0 / m$
repeat
  Choose σ_k
  Solve (2.27) with $(x, \lambda, y) = (x^k, \lambda^k, y^k)$ and $(\mu, \sigma) = (\mu_k, \sigma_k)$
    to obtain $(\Delta x^k, \Delta\lambda^k, \Delta y^k)$;
  Choose step length $\alpha_k \in (0, 1]$ and set
    $(x^{k+1}, \lambda^{k+1}, y^{k+1}) \leftarrow (x^k, \lambda^k, y^k) + \alpha_k(\Delta x^k, \Delta\lambda^k, \Delta y^k)$
    $\mu_{k+1} \leftarrow (y^{k+1})^T \lambda^{k+1} / m$; $k \leftarrow k + 1$;
until termination threshold is achieved [13]

Forces PRO: The selected algorithm that was used for this thesis was Forces PRO. Forces PRO is a commercial solver which leads to that the blueprint for how the solver works exactly is not publicly available, but one of the method it can be set to use is the Primal-Dual Interior Point Algorithm [1].

2.2.3 Active set methods

Active-set methods work in the manner that they come up with an estimate of the active set as the solution by using a two-phase iterative method. The first phase finds a feasible point for the constraints in (2.1) while the second phase tries to maintain the feasibility whilst the objective is minimized. By active set, the set of constraints that are active at a current point is targeted, which for inequality constrains means that they are active if both sides of the constraint are equal to each-other. [17]. An example of the Active-set method can be seen in Algorithm 3 [5].

qpOASES: qpOASES is an online Active-set algorithm which makes an assumption that a QP does not change a lot in comparison to its previous QP. By doing this, previous solution information from the earlier QP is used to solve the next one which gives a faster solution time and makes it well adapted for real-time

Algorithm 3 Active-set method

Let x^0 be a starting point that is fulfilling the constraints in (2.1)
Let W be the current working set containing a subset of the active inequality constraints at x^0.
With equation (2.1):
for k=0..N **do**
  Find the optimal Δ considering the constraints in W as equality constraints and disregard the constraints not in W.
  Solve the quadratic function in (2.1) substituting x for $x^k + \Delta$ with the equality constraint $A_{eq}\Delta = 0$
  if $x^k + \Delta$ is feasible **then**
    $x^{k+1} = x^k + \Delta$
    Compute Lagrange multipliers λ for equality constraints and μ for active inequality constraints.
    if all $\lambda \geq 0$ **then**
      x^{k+1} is the optimal solution to (2.1)
    else
      Remove constraints that belong to the most negative λ from W.
    end if
  else
    Find the maximum step length α such that $x^{k+1} = x^k + \alpha\Delta$ is feasible.
    Add the primary constraint to W
  end if
end for

Algorithm	Type	Mode
Forces	Interior-Point Method	-
qpOASES	Active Set	Warm start
qpOASES	Active Set	Cold start
QPgen	Gradient Descent	Dual
QPgen	Gradient Descent	Primal

Table 2.1 Summary over investigated algorithms

application. Using this method is referred to as *Warm Starting*, and by not using this assumption *Cold Starting* which handles each QP separately. Both warm starting and cold starting have been examined in this thesis [2].

2.2.4 Chosen Algorithms and their Convergence Bounds

In Table 2.1 the following algorithms can be seen that were picked for the simulation test in this thesis.

For the 3 different methods described in this chapter, a different amount of iterations are in general needed for the method to converge. On the other hand, these limits are not something that is available on beforehand for all the solvers.

For the Interior point method, a theoretical lower iteration bound exist, but according to [16], this bound has a margin compared to its actual convergence bound that is too big to be practically applicable. Also for the Active set method, a convergence rate analysis is not available, and hence no a priori convergence limit can be determined. Depending on the choice of warm starting or cold starting the Active-set method, the average performance can be less computationally demanding while the worst-case performance is almost always the same [7].

The only method that has a somewhat reliable convergence rate is the Gradient method. This is depended on the step size t seen in (2.11) and the sequence of iterates x_i generated by the method (for more guidance how these are computed the reader is referred to [16]).

2.3 MPC: Model Predictive Control

Model Predictive Control (MPC) is an advanced control theory method based on optimal control that targets the behavior of a predicted process. With the use of optimization, it is a strong method to handle multiple-input multiple-output systems and constraints [15].

The MPC formulation, which will be used for the simulations in this thesis, can be seen in (2.28) [7]. What the MPC does is that it solves a series of QP:s that are created for a control problem of the form in (2.28) where only the first indices of x, u are applied and the rest are discarded.

$$\begin{aligned}
&\min_{x,u} \sum_{k=0}^{N-1} \begin{pmatrix} y_k - y_k^r \\ u_k - u_k^r \end{pmatrix} \begin{pmatrix} Q_k & S_k \\ S_k^T & R_k \end{pmatrix} \begin{pmatrix} y_k - y_k^r \\ u_k - u_k^r \end{pmatrix} + \begin{pmatrix} g_k^y \\ g_k^u \end{pmatrix}^T \begin{pmatrix} y_k - y_k^r \\ u_k - u_k^r \end{pmatrix} \\
&\quad + (x_N - x_N^r) P (x_N - x_N^r) \\
&\text{with } x_0 \text{ given,} \\
&\quad x_{k+1} = A_k x_k + B_k u_k \\
&\quad y_k = C_k x_k + D_k u_k \\
&\quad u_k^{min} \le u_k \le u_k^{max} \\
&\quad y_k^{min} \le y_k \le y_k^{max} \\
&\quad d_k^{min} \le M_k y_k + N_k u_k \le d_k^{max} \\
&\quad d_N^{min} \le T x_N \le d_N^{max}
\end{aligned} \tag{2.28}$$

Here, for the discrete time step k and the control horizon N, u_k is the control input, x_k the states and y_k the process output with y_k^r and u_k^r as the references to be tracked. $Q_k \in \mathbb{R}^{n_y \times n_y}$ and $R_k \in \mathbb{R}^{n_u \times n_u}$ are weighting matrices and $P \in \mathbb{R}^{n_x \times n_x}$ a penalty

matrix for the terminal state x_N, $g_k^y \in \mathbb{R}^{1\times n_y}$ and $g_k^u \in \mathbb{R}^{1\times n_u}$ are linear penalties for reference tracking. Setting S_k to zero makes (2.28) convex which turns it into the same for as (2.1).

2.3.1 Open- and Closed-Loop Scenarios

Two main scenarios for evaluating the different algorithms in 2.1 that were used were open-loop and closed-loop. They differ a bit in how they are constructed and can be seen in the next two paragraphs.

Open-Loop: When simulations are done in the open-loop case, the evaluation of the results comes simply from the solution to (2.1), the variable x which contains the optimization variables in (2.28) (x,u). This is also the output of the solvers. To be able to compare the results from different examples, a normalization of the output was done according to (2.29).

$$x = \frac{||x^{opt} - x||}{||x||}, \qquad ||x|| > 0 \tag{2.29}$$

Here the Euclidean norm is used.

Closed-Loop: The objective function used for evaluating the closed-loop performance is almost the same as (2.28), it can be seen in (2.30). Here, the output of the solvers are not used directly but are instead computed after the solver is done to be able to handle penalization of constraint violations.

$$\begin{aligned}&\sum_{k=0}^{N-1} \begin{pmatrix} y_k - y_k^r \\ u_k - u_k^r \end{pmatrix} \begin{pmatrix} Q_k & S_k \\ S_k^T & R_k \end{pmatrix} \begin{pmatrix} y_k - y_k^r \\ u_k - u_k^r \end{pmatrix} + \begin{pmatrix} g_k^y \\ g_k^u \end{pmatrix}^T \begin{pmatrix} y_k - y_k^r \\ u_k - u_k^r \end{pmatrix} \\ &+ constraintViolation(u_k, y_k)\end{aligned} \tag{2.30}$$

Furthermore, *constraintViolation* is referring to a function that returns a value depending on how much the constraints of the u and y vectors are violated. The function is calculated as in (2.31) and is only activated if the trajectory violates the given constraints

$$\frac{1}{w}\left(\frac{x_k - x^{max}}{x^{max} - x^{min}}\right)^2 + \frac{1}{w}\left(\frac{x^{min} - x_k}{x^{max} - x^{min}}\right)^2 \tag{2.31}$$

Here, w is a weight chosen to give higher penalization on constraint violation than on deviation from the reference trajectories. The trajectories are also weighted by its feasible area (its maximum subtracted by its minimum), and then squared to make small violations less penalized then large violations.

3

Simulation and Data Analysis

3.1 Simulation Setup

3.1.1 ABB's MPC Benchmarking Suite

Currently being developed in ABB Corporate Research in Baden, Switzerland, is an MPC Benchmarking Suite for Matlab that enables a user to test and compare various MPC Algorithms. The structure of performing a simulation with the Suite can be seen in Figure 3.1.

In more detail the simulation done in the Suite works as followed:
The user selects the following

- Solver
  - Solver options
    - Formulation
    - Number of maximum iterations
    - Enable warmstart
- Benchmark
  - Benchmark case
- Global properties
  - Open-loop or closed-Loop
  - Termination criteria for MPC algorithm

and this information is send through the suite. Firstly the problem structure of the selected benchmark is checked for consistency, which can be if fields are missing

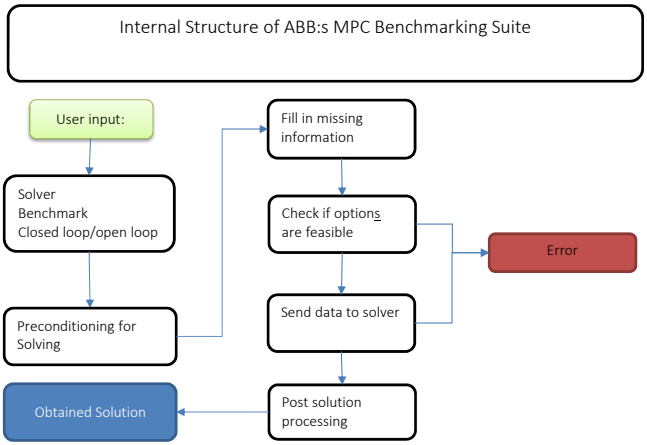

Figure 3.1 A flow chart illustrating a simulation done with the MPC Benchmarking Suite

in the setup. These are then set to a default setting. Secondly, the series of QPs that are to be solved are created and passed to the solver (the chosen algorithm) as optimization problems. The solver returns the solution which is then post processed to retrieve the important information. The solver options a user can select are seen are described in the following paragraphs.

Open-Loop: Setting the global properties to open-loop lets the MPC algorithm run the simulation and for each new quadratic problem solve it without any information regarding the solution of the previous QP.

Closed-Loop: Closed-loop does the opposite compared to open-loop and provides the algorithm with the previous solution. Furthermore, a processing step was added to the suite for this thesis: The control signal given from the solution of the MPC algorithm is always assumed to be within the control signal constraints, which in practice is to map the control signal to its closest constraint if the constraint is violated.

Other parameters: To be able to make a fair comparison of the different solvers that were picked, some parameters were needed to be set as general. When using the Benchmarking Suite, there is an option of setting the formulation of the quadratic programming matrices. Either they are sparse or condensed, with the difference that

the matrices in (2.1) (H, A, A_{eq}) are either structured in a sparse or dense manner. What changes is that the variable to be minimized, x, either contains the control variables (x, u) for sparse formulation or only u for condensed formulation which in other words is a state elimination of the quadratic programming. Which out of these methods is best depends on factors such as if the number of states is small and the control horizon is long (sparse formulation) or the opposite (condensed formulation). To simplify and make the comparison easier for this thesis, the sparse formulation was picked for all examples and solvers. That way, an extra transformation (sparse $\rightarrow$ condensed) of the problem was avoided [7].

Furthermore, to avoid having singular matrices, a small weight was added to all hessians (H in (2.1)).

3.2 Benchmarks

In the benchmarking suite, a total of 22 benchmarks were available for testing algorithms on. Out of these, 19 benchmarks were found to be working for at least one of the chosen algorithm and were therefore also picked for simulation. A table of the used benchmarks with a short summary of their properties can be seen in Table 3.1. The benchmarks are originally picked from both the academia as well as internal ABB scenarios.

Each Benchmark has a certain amount of 'Cases', which represent a small change of the control problem like different form of u and y references. If you summarize all the cases over the 19 benchmarks, there is a total of 66 control problems available. In Table 3.2 a summary of number of problems available per solver can be seen.

3.3 Running Simulations

The simulations that were done were proceeded as follows. Firstly all five solvers were allowed to run each problem to obtain the number of iterations each solver needed for converging according to their internal converging limit. To investigate the effects of early termination, this iteration limit was changed and passed as a parameter for the next simulation run. When evaluating the first run of simulations it occurred that qpOASES *cold started* and qpOASES *warm started* a lot of times only demanded a few numbers of iteration to converge. With this information, a step of $\frac{1}{6}$ was picked as the fixed step size for all further simulation. What this meant was that the next simulations were run for steps of $\frac{1}{6}$ of the converging iteration limit, so each problem for each solvers were run for $17\%, 33\%, 50\%, 67\%, 83\%$ and 100%. The results were saved for further evaluation later on.

Benchmark name	nx	nu	ny	Open-Loop stable
Aircraft	4	2	2	no
Ballonplate	2	1	2	yes
BinaryDistillationColumn	11	3	3	yes
Compressor	6	2	6	yes
DcMotor	4	1	4	yes
DoubleInvertedPendulum	4	2	2	no
FiordosExample	2	1	2	no
ForcesExample	2	1	2	no
Helicopter	6	2	6	no
NonlinearCstr	4	2	4	yes
Pendulum	3	1	3	no
PolytopicTerminal	2	1	1	yes
Quadcopter	12	4	12	yes
RandomStableInputBounds	10	3	10	yes
RobotArm	4	2	4	yes
Shell	9	3	3	yes
Spacecraft	7	4	7	no
ToyExample	2	1	2	yes
TripleInvertedPendulum	6	3	3	no

Table 3.1 List of all the benchmarks used in the simulations.

Solvers	Open-loop cases	Closed-loop cases
Forces	43	44
qpOASES ws	64	66
qpOASES cs	64	66
QPgen Dual	50	52
QPgen Primal	16	16

Table 3.2 Summary of the number of cases used for simulation for Open-loop as well as Closed-loop scenario.

3.4 Dataset formulation

One of the more prominent things about the Benchmarking suite is the possibility to make extensive simulations and obtain large amount of data. For this subject, a dataset of features containing information about the benchmark that was being solved, the solver that solves it and what termination point that is currently being handled was created. From the earlier stages of the projected where the simulations were run, each solution for each scenario was saved for further use in setting up the dataset and other evaluations. The idea was then rather straight-forward; 1. create a script that for each scenario retrieves the solution and the properties of the

scenario, 2. perform necessary action to be able to use these properties as features in a dataset, 3. establish the dataset matrix.

The total number of features used in the dataset are 159 and are related to Control analysis, Numerical analysis, Which solver(s) and Output related (used as output for the model).

The features related to numerical analysis are Euclidean norms and conditioning number of the matrices in the benchmarking processes. The thought behind these features is that larger norms would imply that the processing would be tougher and same thing for conditioning numbers. A case where the conditioning number goes towards infinity would indicate the matrices as singular and thus have a need of processing which could increase the number of iterations needed. Condition number is calculated as in (3.1).

$$||\mathbf{A}^{-1}|| \cdot ||\mathbf{A}|| \tag{3.1}$$

where $\mathbf{A}$ is a square matrix and $||\cdot||$ is the Euclidean norm. Furthermore, the matrices in (2.28) are checked if they are positive definite/semi-definite and added as categorical features.

The control theory related features are for example related to the controllability, observability, step change analysis, poles and zeros of the dynamic system and also MPC specific features as control horizon length, initial and terminal state and also whether look ahead for reference changes is on or not.

The output related features are not really used for predicting the output but are available to use as alternative output for the model creation. They were also found to be useful in checking the validation of the models created later on in this thesis. In Table 3.3 there is a short summary of the features used where they are counted and grouped into different categories.

Feature category	**Number of features**
Which Solver	10
Benchmark Properties	142
- Control related	58
- Numerical analysis	81
Output related	7
Categorical	59
Numerical	100

Table 3.3 Summary of different feature groups used in the simulation. Here Categorical refers to features that are either 0 or 1 and numerical to features that can have any real value.

3.4.1 Feature scaling

To be able to use the features for simulation evaluation the features have to be scaled. If they are not, the comparison between the different features and their importance would be hard to do because of their varied range. The scaling was performed by mapping the features to a range of 0 to 1 according to (3.2) for a certain feature j and for all samples $i = 1 : N$.

$$X_{i,j} = \frac{X_{i,j} - X_{i,j}^{min}}{X_{i,j}^{max} - X_{i,j}^{min}} \tag{3.2}$$

Inevitability, features with an infinite value were present which were taken care of by simply removing the values which had infinite value and adding related categorical features which would be set to one if an infinite value is present and zero if it is not.

Several features were created based on if a problem or a solver contained a certain property. These features are referred to as categorical features as they would either be of the nature 'Yes' or 'No'. However, since all the other data are of numerical origin, it is wanted to change these into numerical values as well. This was done by simply setting the categorical features to either 1 or 0 referring to 'Yes' or 'No'.

A succeeding problem with this is that it was not always certain that a presence of a particular property would increase the wanted outcome that the data would be used for, therefore each categorical feature would need a counterpart which is just the inverted version of itself. This means that a new feature which would have 1 or 0 referring to 'No' respectively 'Yes' was added for each categorical feature.

3.5 Coupling data to results

After computing the dataset, which became a matrix of the size $1284 \cdot 159$ for the open-loop case and $1254 \cdot 159$ for the closed-loop case, with entries from 5 different solvers and 6 different iteration limits. Each row is a sample for a certain iteration limit, solver and problem and each column contains one of the 159 features. Now the process of finding a relation between the dataset and its output began.

3.5.1 Regression analysis

A commonly used statistical approach to examine your data is to create a Regression model. This model is based on the different features and the output was the maximum deviation from the fully converged objective function value. The linear regression fit is a simple relation that can be seen in Equation (3.3)

$$\mathbf{y}_i = \beta_0 + \beta_1 \mathbf{x}_i \tag{3.3}$$

Here β_0 is an interception-constant and β_1 is a vector of predictor constants of the same length as $\mathbf{x}_i$. The regression fit applied on QPgen *Dual* can be seen in Figures 3.2 and 3.3.

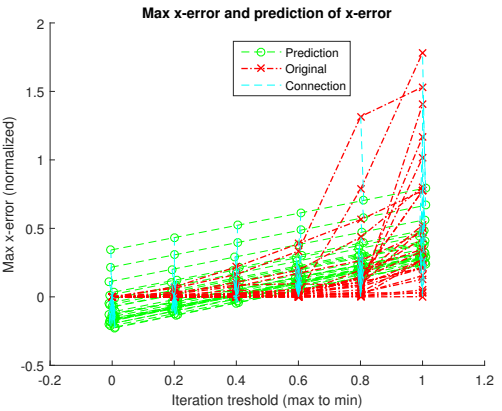

Figure 3.2 Plot showing a linear regression (green points) between the maximal deviation from the solution for the solver QPgen *Dual*. The x-axis is set from fully converged ($x = 0$) to $\frac{1}{6}$ ($x = 1$) of the fully converged iteration limit. The *Prediction* data points have a slight offset to make it easier to establish their representing *Original* data point.

As visible in the plots, the deviation is of a higher order than linear which makes the linear regression fit badly. A perfect fit would overlap the green dots with their corresponding red dots. In the Table 3.4 the Normalized Root Mean Squared Deviation (NRMSD) can be seen for each solver. It is calculated according to (3.4)

$$\frac{1}{y^{max} - y^{min}} \sqrt{\frac{\sum_{t=1}^{n} (\hat{y}_t - y_t)^2}{n}} \tag{3.4}$$

where $\hat{y}$ is the predicted output and y is the original output.

The lesson learned from using regression analysis is that we get a fit that is not well adapted for the type of data that is present. Using a higher order of regression would most certainly lead to a better fit but a problem consists with that the output is never below zero; the deviation is always positive. This calls for a more adaptive model fitting.

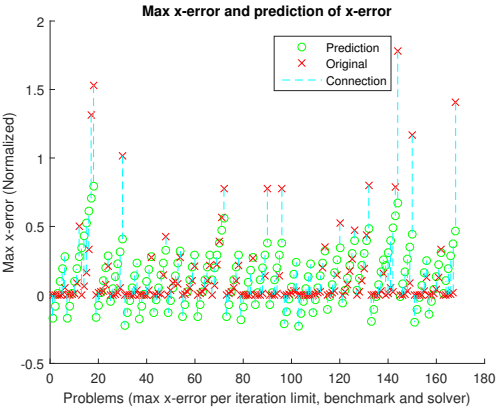

Figure 3.3 Plot showing a linear regression between the maximal deviation from the solution for the solver QPgen *Dual* in the open-loop case. Here each set of points(*Prediction*, *Original* and their connection) is aligned in the x-direction where each problem comes after one of another. It can be seen as the 'side view' of Figure 3.2.

Solver	NRMSD
All	2.82e-02
Forces	5.61e-02
qpOASES	6.11e-02
qpOASES ws	6.25e-02
qpOASES cs	7.11e-02
QPgen	1.02e-01
QPgen Dual	1.19e-01
QPgen Primal	1.68e-01

Table 3.4 NRMSD for linear regression model for each Solver and sub groups

3.5.2 Generalized Linear Models

A Generalized Linear model (GLM) is a build-on onto the linear regression seen in (3.3) with the difference that assumptions can be made about the output data. For this case, the output data is always larger than zero. How the GLM handles this is by having a link function between the input data and the output. This link function can be of the linear case as in (3.3) and it can also be of other types, for example exponential. For this case, a matching link function would be a log-link function [11]. This gives the relation seen in (3.5) which is the same as (3.6).

$$log(\mathbf{y}_i) = \beta_0 + \beta_1 \mathbf{x}_i \tag{3.5}$$

$$\mathbf{y}_i = e^{\beta_0 + \beta_1 \mathbf{x}_i} \tag{3.6}$$

Here, β_0 is an interception-constant and β_1 is a vector of predictor constants of the same length as $\mathbf{x}_i$. In Figures 3.4 and 3.5 the GLM fit can be seen.

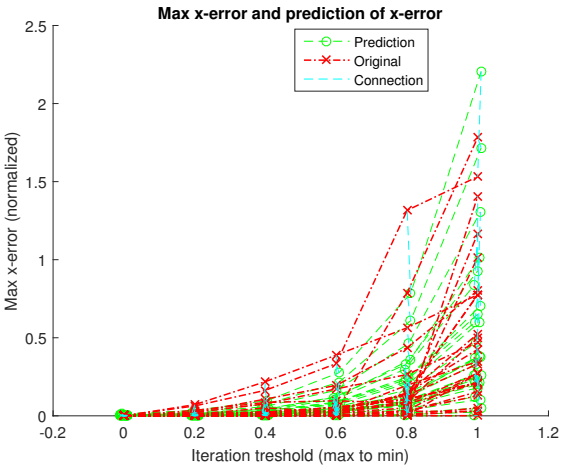

Figure 3.4 Plot showing a Generalized linear model regression (green) between the maximal deviation from the solution for the solver QPgen *Dual*. The x-axis is set from fully converged ($x = 0$) to $\frac{1}{6}$ ($x = 1$) of the fully converged iteration limit. The *Prediction* data points have a slight offset to make it easier to establish their representing *Original* data point.

A direct comparison with Figures 3.2 and 3.3 tells that the model is much more adapted for the data that is present. In Table 3.5 the NRSMD are presented. Not only does the model visually seem more logical, overall the error gets lower by almost 9 times. For qpOASES the result is just slightly higher. Still, the results are not adequate enough.

3.5.3 Turning numerical output into categorical output

To make the problem of predicting the performance of a solver or a group of solvers given certain benchmark properties into a simpler problem, the numerical output would have to be turned into a categorical output. This categorical output would be divided into either 0, which for this case would refer to a good performance given a certain iteration step limit, or into 1 which would refer to a bad performance.

To enable this, a threshold would be needed to be set. Setting a general threshold that simply determines if the performance is good or bad is not an easy task since there are many different solvers and benchmarks, and the threshold would most

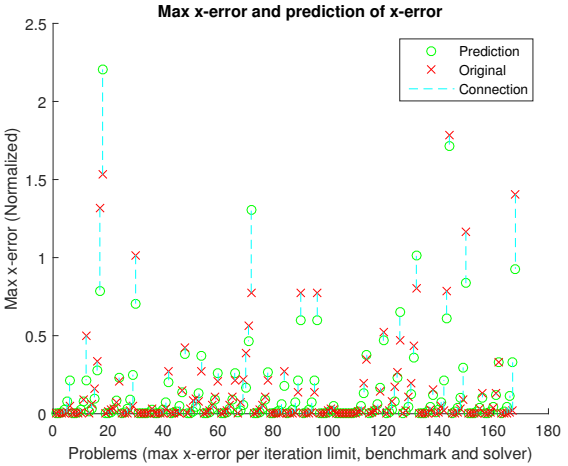

Figure 3.5 Plot showing a Generalized linear model regression between the maximal deviations from the solution for the solver QPgen *Dual* in the open-loop case. Here each set of points (*Prediction*, *Original* and their connection) is aligned in the x-direction where each problem comes after one of another. It can be seen as the 'side view' of Figure 3.4.

Solver	**NRMSD**
All	3.17e-03
Forces	7.34e-03
qpOASES	5.50e-02
qpOASES ws	4.98e-02
qpOASES cs	6.28e-02
QPgen	5.98e-02
QPgen Dual	6.42e-02
QPgen Primal	4.58e-02

Table 3.5 NRMSD for a Generalized linear model for each Solver and sub groups

certainly fluctuate with this variation. Therefore the connection with a good and bad performance cannot be directly asserted, instead the threshold should be seen as a guideline towards quality of performance.

Comparing the datasets created after converting numerical output into categorical output can be seen in Table 3.6

	1	0	% 1:s
Open-Loop	928	356	72.3
Closed-Loop	300	954	24,0

Table 3.6 Comparison of the open-loop and the closed-loop datasets.

Solvers	Maximum correlation	Feature
All	0.308	Not QPgen Primal
Forces	0.16	Step change: Peak time
qpOASES	0.182	if condition number of y_{ref} is inf
qpOASES ws	0.161	if condition number of y_{ref} is inf
qpOASES cs	0.203	if condition number of y_{ref} is inf
QPgen	0.343	if condition number of u_{ref} is inf
QPgen Dual	0.301	has Full state
QPgen Primal	0.48	if condition number of u_{ref} is inf

Table 3.7 Table showing the highest correlating feature for the Open-Loop case

3.6 Feature selection

Feature Selection is something that is done out of two reason. The first is to break down the problem into a smaller problem; a dimensional reduction. This can example be used for compression of data. The second reason is to get a better understanding of the problem that your data was produced from: getting rid of superfluous features and only keeping the ones that are relevant. Depending on what the reason is for making a reduction of features, there are different approaches. The second reason is what this section will be about.

3.6.1 Correlation based methods

A first step of finding relation between dataset with its features and an output would be to look at the correlation between them. In this way it is possible to select the single feature of the dataset that has the largest correlation. When it comes to obtain more features, correlation between the output and the dataset is not a good method. This is because selecting features that have the 2nd best correlation and onward does not guarantee that the prediction will be better, the 2nd best feature could be completely correlated to the best feature and hence it would not add any extra information to the prediction. On the other hand, the least correlated feature could be the one that improves the prediction. Therefore, simply using correlation is only good for creating a model with one single feature, which is not complex enough for the task in this thesis. In Table 3.8 and Table 3.7 the highest correlating features are shown. This time, the fully converged iteration points are not accounted for since the deviation here always is zero.

Solvers	Maximum correlation	Feature
All	0.196	Has output constraints
Forces	0.234	u_{min} constraints present
qpOASES	0.315	$u_{max}^{max} - u_{min}^{max}$
qpOASES ws	0,342	$u_{max}^{min} - u_{min}^{min}$
qpOASES cs	0.379	Has output constraints
QPgen	0.317	Lookahead is on
QPgen Dual	0.346	Lookahead is on
QPgen Primal	0.326	condition number of P

Table 3.8 Table showing the highest correlating feature for the Closed-Loop case

3.6.2 Machine learning based methods

A more and more widespread method of examining data to find and draw conclusions and assumptions is to use the help of machine learning theory.

3.6.2.1 Regularization Regularization is a method where you penalize the number of features used for creating your output.
An example of a regularization method that has been used in this thesis is called Lasso [3]. This method solves a problem of the form as seen in (3.7)

$$\min_{\beta_0,\beta}\Big(\frac{1}{N}\text{Deviance}(\beta_0,\beta)+\lambda\sum_{j=1}^{p}|\beta_j|\Big) \tag{3.7}$$

where Deviance refers to how well the created model fits to the data with the variables β_0 which is the intercept, and β which are the predictor coefficients. N is the number of observations and p is the number of features. λ is picked by the user and a high value increases the penalization of the L^1 norm of the predictor coefficients which means that a big number of the predictors will be zero.

Adding this method to the regression method you get Regularized Regression. Now it is possible to create a model with fewer features and more easily draw conclusions of which properties are important for a good or a bad fit.

For the open-loop case, the way the deviation from the fully converged objective function is computed makes the deviation to always be positive. Therefore, creating a model with Regularized Regression that is able to predict a negative deviation makes no sense. This called for finding a more specialized method.

3.6.2.2 Logistic Regression As the name suggested, Logistic Regression is a form of regression like Linear Regression , but instead of giving a numerical output, Logistic regression is a classifier that gives an categorical output of 1:s or 0:s. In short, this is done my having a link function that maps the data to a span between 0 and 1. Then, a decision boundary is placed in between which groups the outputs

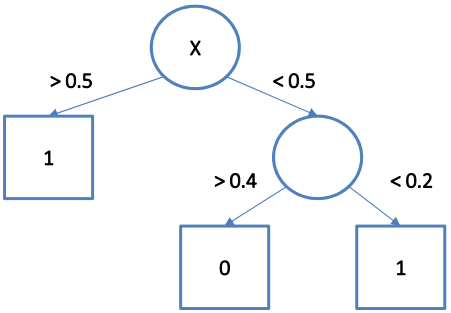

Figure 3.6 A simple decision tree performing a classification for the variable X. First input is called the root, the second round boxes are referred to as nodes and the square boxes as leaves.

on either side to the categories 0 or 1. This method can be used together with Regularization and also GML to create a powerful features selector that is possible to adapt to the input data [10].

3.6.2.3 Random Forest Random Forest is a Machine Learning method that classifies an output into an arbitrary number of classes. It is based on Decision Trees which is illustrated in Figure 3.6 and Bagging. In a Decision Tree, each node is split with the help of the best available variable in the dataset. In Random Forest, each node is split with the best variable among a subset of predictors that are randomly chosen at each node. Bagging is a method of averaging several unbiased and noisy models to create a model with low variance. The Random Forest algorithm is described in Algorithm 4 [8].

Algorithm 4 Random Forest

Generate N randomly picked subset of dataset X
for each subset **do**
    Grow one Decision Tree with each node containing the best split among variables resulting from a randomly picked number of the total number of predictors.
end for
New data is predicted as a majority vote of all grown Decision Trees.

How well the Random Forest performs is calculated by something called Out

-of-bag (OOB) error. The OOB error, which can be compared to cross-validation to obtain an unbiased performance estimation (described later in the thesis), is calculated by testing each decision tree on the data (Out-of-bag data) that was not included when creating it.

With the help of the OOB error, it is possible to obtain the importance of each feature in the dataset. The way this works is by looking at how much the prediction is increased for OOB data when a certain feature variable is changed and all others are kept the same. This measurement is referred to as the OOB Permuted Predictor Delta Error [8].

3.6.2.4 Cross-validation Cross-validation is a method to assert that the model or prediction you create is not a too optimistic approach of your data. This means that the model fits the data too good and will not be a good predictor of new unseen data. What it does is that it creates subsets of the dataset that you have, creates a model on that subset and then tests that model on another unseen subsets. In Algorithm 5 a cross-validation scheme can be seen [6].

Algorithm 5 Cross-validation

```
Split dataset X into k folds
for each fold do
    Current fold becomes test set and the rest of the folds becomes training set
    Fit model/parameters onto the training set
    Test the model on the "unseen" test set
end for
Best performance on the test set is picked as final model parameters
```

3.6.2.5 F1-Score When facing a classification problem, it might not be straightforward to assess how to evaluate the performance of a created model. There will be cases where data points from a class is classified as another class and this is often a relation that is a bit skewed. To handle this, there is a good scoring method called the F1-score that can be used. Before introducing it, there are some terminologies that need to be clarified. In a classification problem with two classes, as in this case, there are 4 different types of outcomes: True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN). In Table 3.9 the relations with these terms can be seen. These terms are then used to create the Precision value ((3.8)) and Recall (3.9). Together they create the F1-score (3.10). For a good F1-score, both Precision and Recall need to be high which means that the amount of False Positives and False Negatives need to be low (A F1-score of 1 would indicate a 'perfect' classifier) [14].

		Actual	
		0	**1**
Predicted	**0**	TN	FN
	1	FP	TP

Table 3.9 A table showing how the different classification outcomes for two classes (**0** and **1**) relate to each other

$$\text{Precision} = \frac{TP}{TP+FP} \tag{3.8}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3.9}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.10}$$

4

Results and Discussion

4.1 Preprocessing of results

As mentioned earlier, some solvers make up a bad match with certain problems and hence the solutions that is produced has a KKT-violation that is much higher when comparing to other problem for the same solver. This results should not be taken into consideration when it comes down to evaluating the results and are therefore removed. A limit of a KKT-violation of 0.01 was set as a threshold for removing result that has a higher KKT-violation than that. The choice of 0.01 comes from that when running processes in real world simulation, there will always be a numerical error related to disturbances and model uncertainty, hence a value of 0.01 will allow the results that have a good KKT-violation value, as well as some results that are on the edge of being bad results but still removing the results that are clearly not feasible.

In General when analyzing the results in Section 4, a high score for the GLM together with the combination of a low OOB error and high OOB Permuted Predictor Delta Error is something to look for.

4.2 Open-Loop

4.2.1 Feature exploration

GLM: Using a GLM with Lasso regularized logistic regression, the following features were picked as seen in Table 4.1. The feature indicating at what iteration percentage the simulation is terminated at was picked for all solvers and is not included in the table. For this case, the F1-score shows that the eight different models build are able to make predictions that are better than just guessing (F1-score higher than 0.5), with the different qpOASES subsets has the highest score and the QPgen subsets has the lowest. For qpOASES the features used are either related to if there is a constraint present on u_{max}, the variation of u_{ref} or the conditioning or magnitude of the problem's matrices. Furthermore, Forces has a relatively high F1-score as well

where a terminal state $\neq 0$ as the first picked feature.

Solvers	Features	F1-score
ALL	Qpgen Dual	0.901
	QPgen Primal	
	u_{max} constraints present	
Forces	Terminal state $\neq 0$	0.922
	Condition Number of P	
	Step change: % of undershoot	
qpOASES	u_{max} constraints present	0.973
	Standard deviation of u_{ref}	
	Condition Number of M	
qpOASES ws	u_{max} constraints present	0.972
	Standard deviation of u_{ref}	
	Condition Number of Q	
qpOASES cs	u_{max} constraints present	0.972
	Standard deviation of u_{ref}	
	Norm of C	
QPgen	Condition number of P	0.755
	Has Full State	
	Number of QPs	
QPgen Dual	Condition number of P	0.779
	Norm of the the process zeros	
	$x_{0,max} - x_{0,min}$	
QPgen Primal	Condition number of y_{ref}	0.767
	Largest gain of the LTI system	
	Horizon length	

Table 4.1 Most important features disregarding the iteration percentage indicating feature, when creating a GLM with Lasso regularized logistic regression for the open-loop case. For each solver group, the features are ordered according to their importance in descending order.

Random Forest: Looking at the OOB Permuted Predictor Delta Error the following features which highest importance for each solver can be seen in Table 4.2. For all solvers, the feature indicating at what iteration percentage the simulation is terminated at had the highest Delta Error value (ranging between 2.50 and 0.77) and is not included in the table. Comparing with the results achieved with the GLM, qpOASES has the smallest error together with the model for the whole dataset (all solvers), and also here QPgen has a higher OOB error. Since qpOASES has both

Solvers	Features	Delta Error	OOB Error
All	QPgen Dual	0.699	0.079
	QPgen Primal	0.686	
	qpOASES ws	0.639	
Forces	norm of u_{ref}	0.152	0.214
	Lookahead is on	0.143	
	Pos-semi def.(Q,P)	0.134	
qpOASES	u_{max} constraints present	0.391	0.048
	NbrOf active constraints	0.369	
	% of states under constraint	0.303	
qpOASES ws	% of states under constraint	0.441	0.055
	u_{max}-u_{min}	0.322	
	u_{min} constraints present	0.301	
qpOASES cs	NbrOf active constrains	0.280	0.125
	$\sum \lvert u_{min} \rvert$	0.280	
	u_{min} constraints present	0.270	
QPgen	norm of u_{ref}	0.318	0.231
	Sampling time	0.379	
	Min of the natural freq of each pole	0.299	
QPgen Dual	Norm of u_{ref}	0.327	0.262
	Horizon length	0.321	
	$\sum \lvert y_{max} \rvert$	0.312	
QPgen Primal	condition number of y_{ref}	0.249	0.219
	Min of the natural freq of each pole	0.219	
	$u_{max} - u_{min}$	0.206	

Table 4.2 Most important features disregarding the iteration percentage indicating feature, when creating a random forest according to the OOB Permuted Predictor Delta Error. Also visible is the OOB error for 100 different decision trees where all features are used.

the best performance in both machine learning methods, the results are a bit more interesting since, for the GLM case, the chosen features actually can predict how the convergence looks like. For the Random forest case, the constraints of u and its reference are also mentioned as well as "% of states under constraints" which also includes u.

The feature with the highest Delta Error, for the five different solvers, which is related to the importance of the feature, is found for qpOASES ws with "% of states under constraints".

The most important features from Table 4.1 and 4.2 are summarized and illus-

trated as histograms in Figures 4.1–4.3. In Figure 4.1 it is possible to see that more than half of the picked features have control related origin. More specifically when looking at Figure 4.2, the control input u seem to be involved where around 35% of the features where related to it. In Figure 4.3 the most common features are summarized where the feature of "u_{max} constraint present" is the one that occurs most times. Furthermore, two more u-related features are visible in this plot.

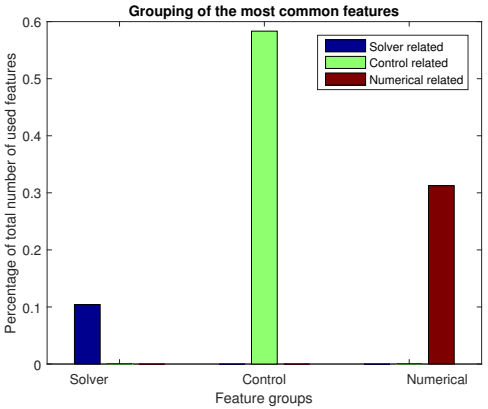

Figure 4.1 Histogram showing the most used feature groups for the different models created for open-loop

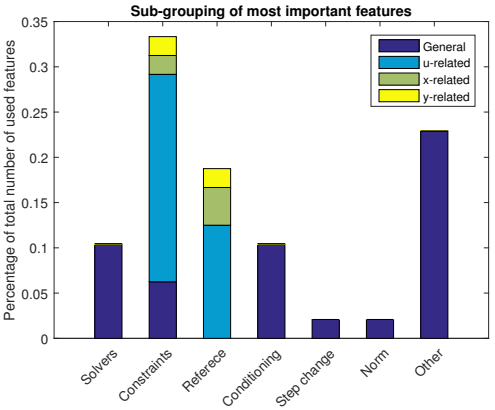

Figure 4.2 Histogram showing the most used features in new sub-groups for the different models created for open-loop

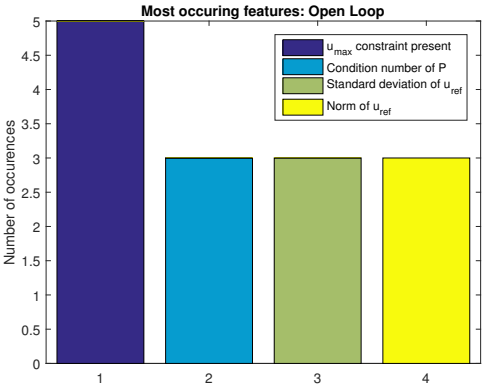

Figure 4.3 Histogram showing the most commonly used features for the different models created for open-loop

4.3 Closed-Loop

4.3.1 Feature exploration

GLM: For the GLM case, performing a cross-validation for a Lasso regularized logistic regression was found to not be obtainable.

Random Forest: The most important features when creating a Random Forest can be seen in Table 4.3.

For the closed-loop case, the GLM Lasso failed to create a model for the data. This is surprising but not completely unexpected since the closed-loop scenario adds an extra level of complexity compared to the open-loop scenario. With the Random Forest for the QPgen dual subset, the predictors for features that were all zero for this particular subset were active in the model creation, which is a sign that something went wrong and could also be a sign that some training sets were only containing 0:s. Still, the Random Forest managed to create models for the rest of the 7 sets. The downside is that Random Forest uses not only the most important features, and because of that the same conclusions can not be made as for the open-loop case.

Smallest OOB error was seen for QPgen and QPgen primal. The most important features are all control theory related where the peak time for a step change both occurs in both subsets.

The feature with the highest Delta Error for the five solvers is the range of the initial states in x_0. This is a higher delta error than for the open-loop case (0.548 vs 0.441), but the OOB Error is, when put into perspective with the other scenario,

much higher.

Solvers	Features	Delta Error	OOB Error
All	qpOASES cs	1.043	0.122
	Not Forces	0.920	
	qpOASES ws	0.900	
Forces	norm of u_{min}	0.367	0.147
	u_{min} constraints are present	0.346	
	u_{max} constraints are present	0.321	
qpOASES	Not qpOASES ws	0.828	0.135
	qpOASES ws	0.689	
	qpOASES cs	0.677	
qpOASES ws	Largest pole	0.457	0.152
	Step change: peak value	0.395	
	$\sum \lvert y_{min} \rvert$	0.386	
qpOASES cs	$x_{0,max} - x_{0,min}$	0.548	0.116
	$\sum \lvert x_0 \rvert$	0.546	
	Number of QP:s	0.471	
QPgen	Horizon length	0.303	0.080
	Number of QPs	0.298	
	Step change: peak value	0.288	
QPgen Dual	-	-	-
	-	-	
	-	-	
QPgen Primal	Step change: peak time	0.143	0.063
	Norm of the process zeros	0.143	
	Sampling time	0.143	

Table 4.3 Most important features disregarding the iteration percentage indicating feature, when creating a random forest with 100 decision trees according to the OOB Permuted Predictor Delta Error. Noting that QPgen Dual is missing results, the reason is that the Random Forest method failed to make a model for this subset.

The most important features from Table 4.3 are summarized and illustrated as histograms in Figures 4.4–4.6. As for the open-loop case, the control related features are again mostly used (figure 4.4) but with the u-related features not being as prominent as in the open-loop case (seen in 4.2). Looking at the most used features in Figure 4.6, the feature giving the peak time after a step change is mostly used, but noting here is that it only occurs 3 times which is a total occurrence of 12.5%.

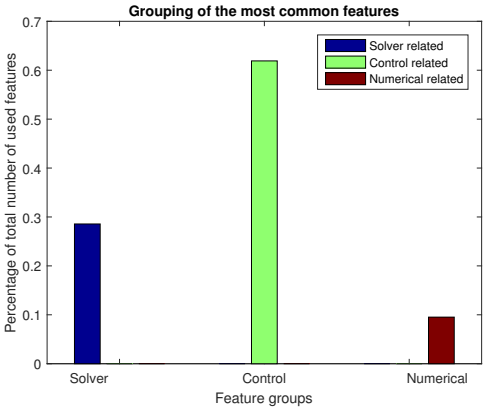

Figure 4.4 Histogram showing the most used feature groups for the different models created for closed-loop

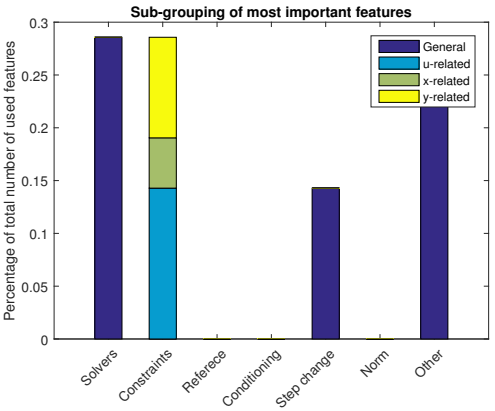

Figure 4.5 Histogram showing the most used features in new sub-groups for the different models created for closed-loop

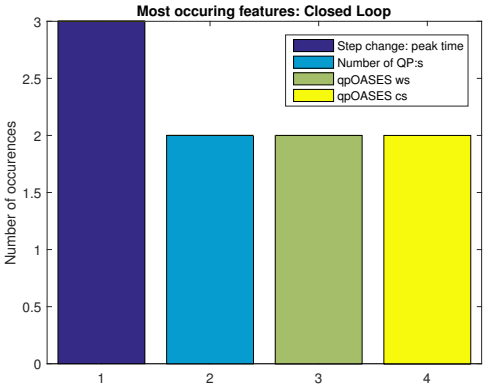

Figure 4.6 Histogram showing the most commonly used features for the different models created for closed-loop

4.4 Compiled

In Figures 4.7–4.9 a compiled version, where the results from both the open-loop and the closed-loop scenario is considered, of histograms can be seen summarizing the most important features from Tables 4.1–4.3. Since control related features where the most used ones for both the open- and closed-loop case it is also the most used ones when compiling both the cases together. This is seen in Figure 4.7. In Figure 4.8, the most used sub-group overall is the features realted to the constraints of the control variables, where the u-related are again most dominant. The most interesting plot is Figure 4.9 where the most commonly used features are shown, here the feature signaling if u_{max} constraint is present or not is by far the most used feature.

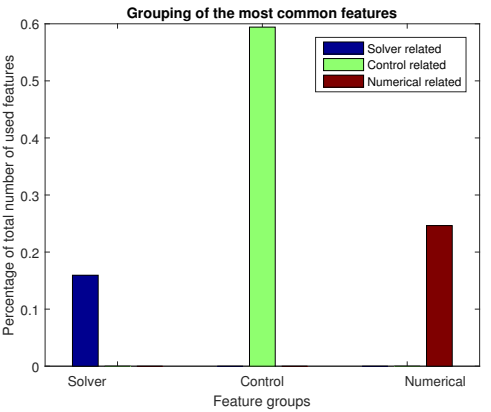

Figure 4.7 Histogram showing the most used feature groups for the different models created both for closed-loop and open-loop

4.5 Discussion of Sections 2 and 3

Given the different proportions of the datasets could be a reason for why the modeling was much more troublesome for the closed-loop case than for the open-loop. As seen in Table 3.6, the number of 1:s and 0:s for the two dataset is almost inverted. Why this can create a problem is that for a certain subset, there is a risk that a training set created with cross-validation or with Random Forest will only contain zeros.

In Tables 3.7 and 3.8, the features with the highest correlation with the output can be seen. Interestingly, in Table 3.7 for the open-loop case, the Condition number of y_{ref} and u_{ref} are mentioned several times. Since neither y_{ref} nor u_{ref} are square

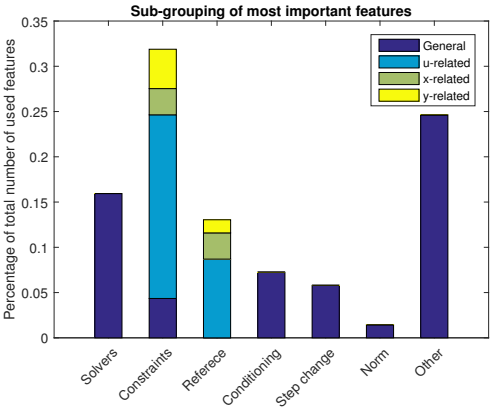

Figure 4.8 Histogram showing the most used features in new sub-groups for the different models created both for closed-loop and open-loop

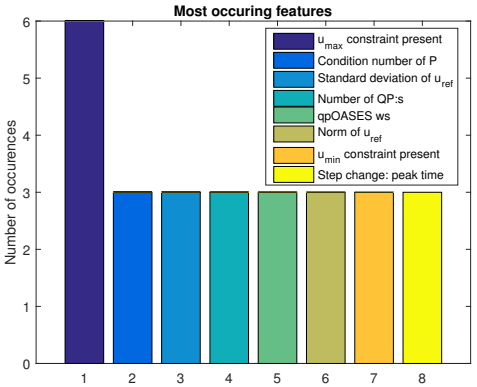

Figure 4.9 Histogram showing the most commonly used features for the different models created both for closed-loop and open-loop

matrices, the condition number is calculated in another way than in (3.1) (more precisely by the help of Singular Value Decomposition). These features in Tables 3.7 and 3.8 are hard to interpret, one case could be that they are "hiding" more important features that are of numerical origin, while the "if condition number is infinite" related features are either 0 or 1 with 1 as the highest value a feature can have. Therefore a numerical feature with a lower value than 1 would have a less correlation.

5

Conclusions and Future research

5.1 Conclusions

The findings in this thesis show that it is possible to predict for some solvers whether or not they will come up with a good enough solution when terminated early, when only using a few numbers of features. The Machine learning model-based predictions done in ABB's Benchmarking Suite work best for an open-loop scenario whereas for closed-loop, less results were achieved.

The features that were most used in general are related to the control input u. Good results were achieved for especially qpOASES in the open-loop case, with the control input u here as well as the most relevant feature. The results for the closed-loop case were not as well defined, with QPgen Primal as the best solver but with low-valued feature importance.

5.2 Future Research

The research regarding this topic can be extended in several directions to see if the results will improve, for example adding more iteration steps or coming up with more benchmarking cases (time and processing demanding), or trying out different machine learning methods than the ones used in this thesis. Relating the work more further to how the algorithms work in very detail is also something that can be looked upon.

Bibliography

[1] Alexander Domahidi. *FORCES: Fast Optimization for Real-time Control on Embedded Systems*. http://forces.ethz.ch/doku.php?id=start. Oct. 2012.

[2] Hans Joachim Ferreau et al. "qpOASES: a parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6.4 (Dec. 2014), pp. 327–363. ISSN: 1867-2949, 1867-2957. DOI: 10.1007/s12532-014-0071-1. URL: http://link.springer.com/10.1007/s12532-014-0071-1 (visited on 07/23/2016).

[3] Wenjiang J. Fu. "Penalized Regressions: The Bridge versus the Lasso". In: *Journal of Computational and Graphical Statistics* 7.3 (Sept. 1998), p. 397. ISSN: 10618600. DOI: 10.2307/1390712. URL: http://www.jstor.org/stable/1390712?origin=crossref (visited on 10/18/2016).

[4] Pontus Giselsson and Stephen Boyd. "Preconditioning in fast dual gradient methods". In: IEEE, Dec. 2014, pp. 5040–5045. ISBN: 978-1-4673-6090-6 978-1-4799-7746-8 978-1-4799-7745-1. DOI: 10.1109/CDC.2014.7040176. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7040176 (visited on 03/24/2016).

[5] Ola Harkegard. "Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation". In: *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*. Vol. 2. IEEE, 2002, pp. 1295–1300. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1184694 (visited on 07/23/2016).

[6] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *International Joint Conference on Artificial Intelligence*. Vol. 14. 1995, pp. 1137–1145. URL: https://pdfs.semanticscholar.org/0be0/d781305750b37acb35fa187febd8db67bfcc.pdf (visited on 07/16/2016).

[7] D. Kouzoupis et al. “Towards proper assessment of QP algorithms for embedded model predictive control”. In: *Control Conference (ECC), 2015 European*. Control Conference (ECC), 2015 European. July 2015, pp. 2609–2616. DOI: 10.1109/ECC.2015.7330931.

[8] Andy Liaw and Matthew Wiener. “Classification and regression by randomForest”. In: *R news* 2.3 (2002), pp. 18–22. URL: ftp://131.252.97.79/Transfer/Treg/WFRE_Articles/Liaw_02_Classification%20and%20regression%20by%20randomForest.pdf (visited on 07/16/2016).

[9] David G. Luenberger and Yinyu Ye. *Linear and nonlinear programming*. 3rd ed. International series in operations research and management science. New York, NY: Springer, 2008. 546 pp. ISBN: 978-0-387-74502-2.

[10] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. URL: http://books.google.com/books?hl=en&lr=&id=AKuMj4PN_EMC&oi=fnd&pg=PR7&dq=%22requires+some+of+the%22+%22are+towards+the+end.+All+chapters+of+Part+III+are+optional+on+a%22+%22Parts+I,+II,+IV,+and+V+of+this+book,+chapters+on+advanced+or%22+&ots=ELnrc75yAf&sig=bt2ju-jexLaetgk-pV3LrnUcoI4 (visited on 07/19/2016).

[11] Peter McCullagh and John A. Nelder. *Generalized linear models*. Monographs on statistics and applied probability: 37. London : Chapman & Hall, 1989, 1989. ISBN: 978-0-412-31760-6.

[12] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. 2nd ed. Springer series in operations research. New York: Springer, 2006. 664 pp. ISBN: 978-0-387-30303-1.

[13] Florian A. Potra and Stephen J. Wright. “Interior-Point Methods”. In: *Journal of Computational and Applied Mathematics* 124 (Nov. 1999), pp. 281–302. URL: http://pages.cs.wisc.edu/~swright/papers/potra-wright.pdf (visited on 07/29/2016).

[14] David Martin Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”. In: (2011). URL: http://dspace2.flinders.edu.au/xmlui/handle/2328/27165 (visited on 07/16/2016).

[15] James Blake Rawlings and David Q. Mayne. *Model predictive control: theory and design*. OCLC: ocn430536884. Madison, Wis: Nob Hill Pub, 2009. 533 pp. ISBN: 978-0-9759377-0-9.

[16] Stefan Richter. “Computational complexity certification of gradient methods for real-time model predictive control”. PhD thesis. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20718, 2012, 2012. URL: https://e-collection.library.ethz.ch/view/eth:6362 (visited on 03/21/2016).

[17] Elizabeth Lai Sum Wong. “Active-set methods for quadratic programming”. In: (2011). URL: http://escholarship.org/uc/item/2sp3173p.pdf (visited on 07/23/2016).

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
November 2016

Document Number
ISRN LUTFD2/TFRT--6023--SE

Author(s)
Gustav Henriks

Supervisor
Joachim Ferreau, Corporate Research ABB
Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Study of early termination of MPC Algorithms

Abstract

With a steady development of technology, the use of Model Predictive Control (MPC) has become more and more popular since the computation time has gone down. With this increase, a need for determining which MPC algorithm is good for solving a certain type of MPC problem has occurred which would facilitate the choice of algorithm and also could increase the performance. One of the determining aspects is if a solver has the possibility to terminate early (stop the algorithm before it has performed all of its iterations), for example if it is placed on an embedded system with strict real-time bounds. With the use of an MPC Benchmarking suite available at ABB Corporate Research Switzerland, the early termination of MPC algorithms has been investigated. With the usage of 19 benchmarks and 3 different solvers that uses Interior point method, Gradient descent and Active-set method a large number of results have been looked through with the help of different Machine Learning methods. The result has been classified as good or bad performance when terminated early and different models have been fitted to predict this data. From this a group of key-features have been attempted to get extracted to see if there is a possibility on beforehand to tell if a control problem and a certain solver can be early terminated. Important features that were found were mostly concerning whether the control input u was under constraint or not. Good results were especially achieved for a machine learning model based on the Active-set solver qpOASES which could give good indications on whether a certain problem could get early terminated or not.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-47

Recipient's notes

Security classification

http://www.control.lth.se/publications/