Efficient solving of Quadratic Programming with ramp functions for QP-MPC in Python

Author:
Ingvild Rustad Haugen

Supervisor:
Morten Hovd

https://www.overleaf.com/project/633199e98d14052883dd6ba1
Specialization project
Department of Engineering Cybernetics
Norwegian University of Science and Technology

November 24, 2022

Preface

This thesis is written as a part of the requirements for the 5th year at Cybernetics and Robotics at Norwegian University of Science and Technology, Trondheim, Norway. The project was developed trough the autumn semester 2022 and delivered in December 2022.

The thesis is a continuation on the research paper published by professor Morten Hovd (NTNU) and professor Giorgio Valmorbida (Université Paris-Sacalay) *Quadratic programming with ramp functions and fast QP-MPC solutions* [6], submitted to the research jounal Automatica.

Executive summary

This rapport contains an implementation of a numerical algorithm for efficiently solving QP problems from MPC. A comparison of Matlab code and Python code is first presented, then an attempt of improvement is done in the Python script. BLA BLA MERE OM HVA RESULTATET BLE.

Abbreviations

Abbreviation	Description
MPC	Model Predicitve Control
QP	Quadratic programming
NumPy	Numerical Python (Python Package)

1

Introduction

1.1 Background and motivation

Model Predictive Control (MPC) is one of the most popular type of controllers for robust control in the process industry [**?**]. MPC introduces the advantage of including constraints on input and output variables and the possibility of having non-linearity in both the model and the constraints. However the MPC algorithm is computationally demanding since an optimization problem has to be solved at each sampling interval and have traditionally only been applied to slower processes (e.g., chemical processes). [8].

Since the beginning of MPC controllers, it has been desired to increase the computational performance through more efficient algorithms so that the application field can be expanded. This project aims to contribute to this development with a new method for solving the MPC-QP problem.

1.2 Main Objectives of the project

The presented project is a preparatory study for the Master thesis project in the first semester of 2023. The objectives of the ;aster thesis project will be to code and document a computationally efficient QP solver based on a newly developed algorithm based on ramp functions. The objectives of this preparatory study can be summarized in the following points:

1. Getting a deeper understanding of the algorithm that solves Quadratic programming with ramp functions presented in [6]

2. Optimize the numerical algorithm proposed in Quadratic programming with ramp functions and fast QP-MPC solutions in Python

3. Learn to use tools to analyze the complexity of a program trough Profiling

Although Matlab is seen as a more efficient and convenient programming language for numerical algorithms, the decision was to implement the algorithm in Python in this project. [11] This was with the objective of learning about optimization of numerical algorithms in a language well known for the student so that when implementing the code in a more efficient language later, preferable C, bugs could be avoided. Due to the complexity and lack of experience in C, it was considered that this order of the tasks would be most efficient and lead to a better outcome.

1.3 Delimitations

The task have been limited to optimizing the code in Python in regards of run time. The memory usage was not intended optimized, but it is analyzed. The and the concept of sparse matrices, used in the original Matlab code to save memory space was neglected in the Python translation. This will be considered later, when implementing the algorithm in C in the Master thesis. The Python version was run on two different models which was decided to be enough to do the analysis objected in this project. The code is written with a goal of maintainability, so one could easily change the models if desired. The code is written with a goal of efficiency, the code quality where not given a high priority.

1.4 Structure of the report

SKRIV DETTE SENERE

2 Theory

In this chapter, the theory of MPC, KKT, and .... is presented. The reader should know the theory to better understand the following project.

2.1 Mathematical optimization theory

2.1.1 The optimization problem and feasible set

Mathematical optimization problems consists of an objective function, decision variables and constraints [3]. Constraints are divided into two types, equality constraints and inequality constraints. This allows the definition of an optimization problem

$$\min_{z \in \mathbb{R}^n} f(z)$$

subject to

$$\begin{aligned} c_i(z) &= 0, \quad i \in \mathcal{E} \\ c_i(z) &\geq 0, \quad i \in \mathcal{I} \end{aligned}$$

The objective function f takes an n-dimensional vector, $z \in \mathbb{R}^n$, and projects it onto the real axis. $\mathcal{E}$ and $\mathcal{I}$ are disjunct index sets for the constraints. The solution for z may be selected from a subset of $\mathbb{R}^n$ called the feasible region. The feasible region consists of all solutions satisfying the constrains c_i.

$$\Omega = \{z \in \mathbb{R}^n \mid (c_i(z) = 0, i \in \mathcal{E}) \wedge (c_i(z) \geq 0, i \in \mathcal{I})\} \tag{2.1}$$

2.1.2 Quadratic programming

Optimization problems can be categorized in different groups by their objective function and constraints. This project will focus on solving Quadratic programming problems, where the objective function is quadratic and the constraints are linear [5]. A QP standard problem is formulated by

$$\min_{v} 0.5v^T \widetilde{H} v + c^T v \tag{2.2}$$

subject to the constraints

$$Lv \leq b \tag{2.3}$$

Where v is a vector of free variables in the optimization, $\widetilde{H}$ is the Hessian matrix, c describes the linear part of the objective function, and L and b describe the linear constraints. To ensure that there exists a unique optimal solution to the optimization problem, the Hessian matrix has to be positive definite.

2.1.3 Active set

An active constraint is defined as a constraint for which $c_i(z) = 0$, which implies that all equality constraints are active at a feasible point. An inequality constraint may be active of inactive at a feasible point. The active set is defined as the set of active constraints at the solution, that is

$$A(x) = \mathcal{E} \cup \{i \in \mathcal{I} || c_i(x) = 0\} \tag{2.4}$$

2.1.4 First order necessary conditions

The Karush-Kuhn-Tucker (KKT) conditions define necessary conditions for optimal solutions to minimization problems. A Lagrange function is defined as

$$\mathcal{L}(z, \lambda) = f(z) - \sum_{i \in \mathcal{E} \cup I} \lambda_i c_i(z) \tag{2.5}$$

where the λ_i are the Lagrange multipliers. The KKT conditions is a key result in constrained optimization.

Theorem 1. *Assume that z is a local solution of* (2.5) *and that f and all c_i are differentiable and their derivatives are continuous. Further, assume that all the active constraint gradients are linearly independent at z Then there exists Lagrange multipliers λ_i^* for $iinE \cup I$ such that the following conditions (called the KKT conditions) hold at (x^*, λ^*)*

$$\begin{aligned} \nabla_z \mathcal{L}(z^*, \lambda^*) &= 0 \\ c_i(z^*) &= 0, \quad i \in \mathcal{E} \\ c_i(z^*) &\geq 0, \quad i \in \mathcal{I} \\ \lambda_i^* &\geq 0, \quad i \in \mathcal{I} \\ \lambda_i c_i(z^*) &= 0, \quad i \in \mathcal{I} \end{aligned} \tag{2.6}$$

The

2.2 Model Predictive Control

Model Predictive Control, MPC, is a method for process control used to control a process while satisfying a set of constraints. [5]. It is a multi variable control algorithm that uses an internal dynamic model of the process, a cost function to minimize over the moving horizon and an optimization algorithm that minimize the cost function with respect to the control input u. The cost function is typically formulated as Linear programming (LP) or Quadratic programming (QP). LP problems are more efficiently solved than QP problems, but QP problems leads to a smoother control action and is also what is used in the algorithm in [6]. This theory section will for this reason descibe the formulation of a QP-MPC problem.

2.2.1 Objective functions for discrete time systems

The algorithm in [6] aims to optimize a dynamic model sampled at discrete points in time. The objective functioon of dynamic optimization problems can be defined as

$$f\left(x_1,\ldots,x_N,u_0\ldots,u_N\right)=\sum_{t=0}^{N-1} f_t\left(x_{t+1},u_t\right) \tag{2.7}$$

The objective function is defined from $t=0$ to $t=N$ where t is the time index. The time from 0 to N is called the prediction horizon. The objective function is given as the sum of the contribution from all time steps in the prediction horizon. The formulation of a MPC optimization problem begins with a linear, discrete-time state-space model.

$$x_{k+1}=Ax_k+Bu_k+Ed_k \tag{2.8}$$

$$y_k=Cx_k+Fd_k \tag{2.9}$$

The subscripts describes the sampling instants. Like in control literature, the state x, input u, disturbance d and measurement y should be interpreted as deviation variables, that represents the deviation of a set of variables from the obtained model.

2.2.2 MPC-QP optimization problem

A typical optimization problem in MPC is be given by

$$\begin{aligned}\min_u f(x,u)=&\sum_{i=0}^{n-1}\Big\{\left(x_i-x_{ref,i}\right)^T Q\left(x_i-x_{ref,i}\right)\\&+\left(u_i-u_{ref,i}\right)^T P\left(u_i-u_{ref,i}\right)^T\Big\}\\&+\left(x_n-x_{ref,n}\right)^T S\left(x_n-x_{ref,n}\right)\end{aligned} \tag{2.10}$$

subject to constraints

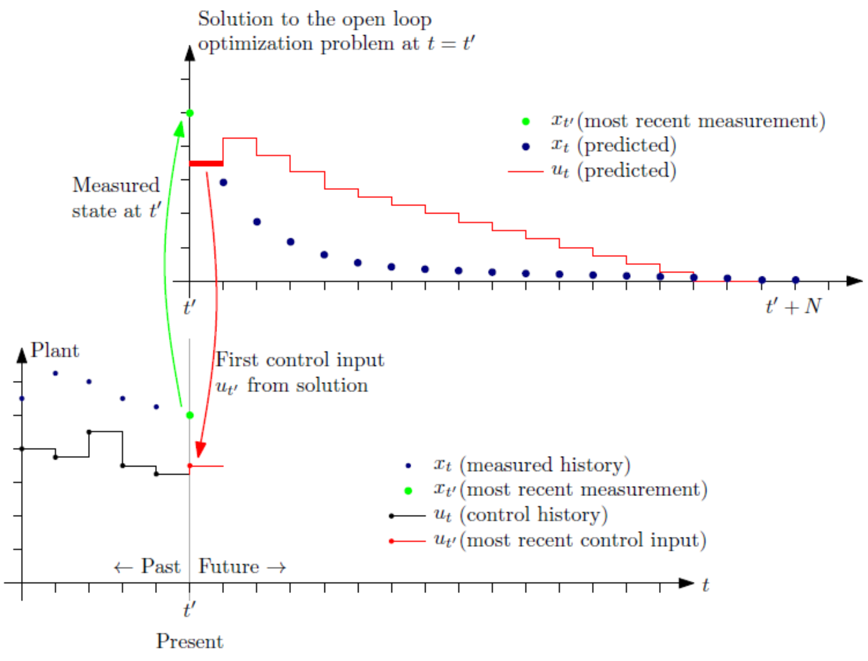
$$\begin{aligned} x_0 = & \text{ given} \\ M_i x_i + N_i u_i \leq G_i & \quad \text{for } 0 \leq i \leq n-1 \\ M_n x_n \leq G_n & \end{aligned} \tag{2.11}$$

The objective function penalize deviation of states and inputs from their desired reference trajectories.

2.2.3 MPC concept

In MPC, an optimal solution is computed at every time step t to add feedback control. [3] At every sampling instant, a finite horizon open-loop optimal control problem is solved. The current state of the plant is the initial state of the optimization problem and the first control in the calculated control sequence is applied to the plant. The MPC concept is illustrated in subsection **??**. The optimization problem is solved at each time step for a finite horizon, but only the first control input u_t is applied to the plant. Then, in the next step, MPC recalculates the whole optimization problem again for the horizon from t to t + N with the updated state prediction x_t as the initial value. MPC uses a *moving horizion*, which means that the MPC horizon changes at each time step, including the current step t and the N next steps, where N is the prediction horizon.

Figure 2.1 Illustation of MPC [3]



Solution to the open loop optimization problem at $t = t'$
$x_{t'}$ (most recent measurement)
x_t (predicted)
u_t (predicted)
Measured state at t'
t'
$t' + N$
Plant
First control input $u_{t'}$ from solution
x_t (measured history)
$x_{t'}$ (most recent measurement)
u_t (control history)
$u_{t'}$ (most recent control input)
← Past
Future →
t
t'
Present

2.3 The algorithm

The algorithm described in [6] proposes a new method for solving MPC-QP problems more efficiently. The MPC-QP problem to be solved by the algorithm given in subsection 2.2.2 can be reformulated as in [1] to

$$\begin{aligned} &\underset{\mathbf{u}}{\text{minimize}} \mathbf{u}^\top H\mathbf{u} + x(k)^\top F^\top \mathbf{u} \\ &\text{subject to} \\ &\quad G\mathbf{u} \leq S_u x(k) + w. \end{aligned} \tag{2.12}$$

By application of the variable change $z = u + H^- Fx(k)$ the formulation of the QP-MPC problem becomes

$$\begin{aligned} &\underset{\mathbf{z}}{\text{minimize}} \mathbf{z}^\top H\mathbf{z} \\ &\text{subject to} \\ &\quad G\mathbf{z} \leq Sx(k) + w \end{aligned} \tag{2.13}$$

where $S = S_u + GH^{-1}F$.

2.3.1 Ramp functions to define the active set

As explained in subsection 2.2.3, the MPC algorithm aims to solve a QP problem for the MPC horizon at each time step. The algorithm solves the optimization problem in (2.13) by the use of ramp functions.

Definition 1. *The ramp function r(y) is given by*

$$r(y) = \begin{cases} 0 & \text{if } y < 0 \\ y, & \text{if } y \geq 0 \end{cases} \tag{2.14}$$

Definition 2. *For $y \in \mathbb{R}^m$ we define the vector-valued ramp function $\phi : \mathbb{R}^m \longrightarrow \mathbb{R}^m$ by*

$$\phi(y) = \begin{bmatrix} r(y_1) \\ r(y_2) \\ \vdots \\ r(y_m) \end{bmatrix} \tag{2.15}$$

Theorem 2. *The solution to the QP-MPC problem given in section 2.2 is given by*

$$\mathbf{u}(x) = -H^{-1}F^T x - H^{-1}G_T\phi(y) \tag{2.16}$$

$$y = -Sx + (I - GH^{-1}G_T\phi(y) - \omega \tag{2.17}$$

The proof of Theorem 2 is given in [6] and is not repeated here.
To solve the QP-MPC problem in section 2.2, (2.17) needs to be solved. For this, we take advantage from that the ramp function $\phi(y)$ can be expressed as $\phi(y) = I_A y$ where

$I_A \in \mathbb{D}^m$ with its diagonals verifying $I_{Ai,i} \in \{0{,}1\}$. The binary diagonal matrix I_A is defined by

$$I_{A(i,i)} = \begin{cases} 1 & \text{if } i \in A \\ 0 & \text{if } i \notin A \end{cases} \tag{2.18}$$

The set A is the set of active constraints, i.e. the set for which the values of the multipliers λ are not zero. Using this definition of the ramp function and re-arranging (2.17) results in:

$$y - (I - GH^{-1}G^T)I_A y = -Sx - \omega \tag{2.19}$$

In the next section, the algorithm for solving to solve the MPC-QP problem will be presented.

2.3.2 Solution to (2.19) by ramp functions

The algorithm exploits the fact that the multipliers, λ, is given by the ramp function:

$$\lambda = \phi(y) \tag{2.20}$$

With the ramp function defining the multipliers, it yields that if $y_i < 0$, then the corresponding multiplier $\lambda_i = 0$, and in this case the corresponding diagonal elements of I_A is zero, $I_{A(i,i)} = 0$. In the opposite case, $y_i \geq 0$, then $I_{A(i,i)} = 1$. This can be summed up in a definition defining the pairs (y, I_A).

Definition 3. *For a given x, the pair (y, I_A) is said to be compatible if (y, I_A) satisfy (2.19) and $I_{A(i,i)} = 1$ if $y_i \geq 0$ and $I_{A(i,i)} = 0$ if $y_i < 0$.*

Thus the solution to (2.17) is found whenever (2.19) is fulfilled with a compatible pair (y, I_A). The algorithm to solve (2.17) searches for the set of active constraints by adding or removing elements to the active set A, i.e. modifying the matrix diagonal of I_A and then update y.

The derivation of a mathematical algorithm to solve (2.19) starts with defining

$$Q(\mathcal{A}) = \left(I_{\mathcal{A}^c} + GH^{-1}G^\top I_{\mathcal{A}}\right) \tag{2.21}$$

and use this to write the solution to (2.19) by

$$\begin{aligned} y - \left(I - GH^{-1}G^\top\right) I_{\mathcal{A}} y &= -Sx - w \\ Q(\mathcal{A})y &= -Sx - w \\ y &= Q(\mathcal{A})^{-1}(-Sx - w). \end{aligned} \tag{2.22}$$

Consider the case where *i enters the active set* A i.e $i \notin A$ and define $A \cup \{i\}$. With

$$I_{\mathcal{A}_{+i}} = I_{\mathcal{A}} + e_i e_i^\top \tag{2.23}$$

and

$$I_{\mathcal{A}^c_{+i}} = I_{\mathcal{A}^c} - I_{\mathcal{A}^c(\cdot,i)} e_i^\top \tag{2.24}$$

we get

$$Q\left(\mathcal{A}_{+i}\right)=Q(\mathcal{A})-\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot i)}e_i^\top . \tag{2.25}$$

The matrix $Q\left(\mathcal{A}_{+i}\right)$ presents a sum of $Q(\mathcal{A})$ and to get a solution for y as in (2.22) the inverse of $Q\left(\mathcal{A}_{+i}\right)$ is needed. The matrix inversion lemma is given by

Lemma 1. *Let $Q \in \mathbb{R}^{NxN}$ be an invertible matrix*

$$\left(Q+q_u q_c q_v\right)^{-1}=Q^{-1}-\left(q_c^{-1}+q_v Q^{-1} q_u\right)^{-1}\left(Q^{-1} q_u q_v Q^{-1}\right) \tag{2.26}$$

where $q_c \in \mathbb{R} \setminus \{0\}$ is a scalar, $q_u \in \mathbb{R}^N$ and $q_v^T \in \mathbb{R}^N$. that is q_u is a column vector and q_v is a row vector.

With the use of Lemma 1 with $q_c=-1, q_v=e_i^T$ and $q_u=\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}$, the inverse of $Q\left(\mathcal{A}_{+i}\right)$ become

$$\begin{aligned}Q\left(\mathcal{A}_{+i}\right)^{-1}=&Q(\mathcal{A})^{-1}\\&-\left(-1+e_i^\top Q(\mathcal{A})^{-1}\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}\right)^{-1}\times\\&\left(Q(\mathcal{A})^{-1}\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}e_i^\top Q(\mathcal{A})^{-1}\right)\end{aligned} \tag{2.27}$$

By defining

$$v(\mathcal{A},i)=Q^{-1}(\mathcal{A})\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}, \tag{2.28}$$

One can rewrite (2.27) as

$$\begin{aligned}Q\left(\mathcal{A}_{+i}\right)^{-1}=&Q(\mathcal{A})^{-1}\\&-\left(-1+v(\mathcal{A},i)_i\right)^{-1}v(\mathcal{A},i)\left(Q(\mathcal{A})^{-1}\right)_{(i,\cdot)}\end{aligned} \tag{2.29}$$

The update of $y_+[i]$ is given by

$$\begin{aligned}y_+[i]=&Q\left(\mathcal{A}_{+i}\right)^{-1}(-Sx-w)\\=&Q(\mathcal{A})^{-1}(-Sx-w)\\&-\left(-1+e_i^\top Q(\mathcal{A})^{-1}\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}\right)^{-1}\times\\&\left(Q(\mathcal{A})^{-1}\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}e_i^\top Q(\mathcal{A})^{-1}\right)(-Sx-w)\\=&y-\left(-1+e_i^\top Q(\mathcal{A})^{-1}\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}\right)^{-1}\times\\&\left(Q(\mathcal{A})^{-1}\left(I_{\mathcal{A}^c}-GH^{-1}G^\top\right)_{(\cdot,i)}\right)y_i,\end{aligned} \tag{2.30}$$

that is simplified with (2.28) to,

$$y_+[i]=y-y_i\left(-1+v(\mathcal{A},i)_i\right)^{-1}v(\mathcal{A},i) \tag{2.31}$$

which describes the update to y when adding a constraint to the active set, A. Consider now the case where $i \in A$ and define $A_{-i} = A \setminus \{i\}$, when i is removed from the set A,

$$Q\left(\mathcal{A}_{-i}\right) = Q(\mathcal{A}) + e_i^\top \left(I_{\mathcal{A}^c} - GH^{-1}G^\top\right)_{(,,i)} \tag{2.32}$$

The matrix $Q\left(\mathcal{A}_{-i}\right)$ presents a sum of $Q(A)$ and with the use of Lemma 1 with $q_c = 1, q_v = q_v = e_i^T$ and $q_u = \left(I_{\mathcal{A}^c} - GH^{-1}G^\top\right)_{(\cdot,i)}$, the inverse of $Q\left(\mathcal{A}_{-i}\right)$ is given by

$$\begin{aligned} Q\left(\mathcal{A}_{-i}\right)^{-1} &= Q(\mathcal{A})^{-1} \\ &\quad - \left(1 + v(\mathcal{A}, i)_i\right)^{-1} v(\mathcal{A}, i) \left(Q(\mathcal{A})^{-1}\right)_{(i,\cdot)} \end{aligned} \tag{2.33}$$

which gives the update of y the following expression

$$y_{-}[i] = y - y_i \left(1 + v(\mathcal{A}, i)_i\right)^{-1} v(\mathcal{A}, i) \tag{2.34}$$

The removing and adding of constraints from and to A given in (2.31) and (2.34) can be summed up with the following points

- Remove first from A constraints corresponding to negative y_i. If there are more than one such constraint, remove first the one corresponding to the most negative y_i.
- Then add to A a constraint that is not a member of and corresponds to a positive value of y_i. If there is more than one such constraint, add first the one corresponding to the largest y_i.

The steps above are implemented in Algorithm 1 below. Algorithm 1 is called inside Algorithm 2 which returns the input $u(x_k)$ to be applied to the plant.

Algorithm 1 Solution to the algebraic equation (2.17)

```
Require: $GH^{-1}G, A, invQ, y$ satisfying $y = invQ(-Sx - \omega)$
 1: while $(A, y)$ not compatible do
 2:     if $ind(sign(y, -1)) \cap A \neq \emptyset$ then
 3:         $L \leftarrow ind(sign(y, -1)) \cap A$
 4:         $i \leftarrow ind(min(y, L))$
 5:         $v \leftarrow invQ(I_{A^C} - GH^{-1}G^T))_{(\cdot,i)}$
 6:         $A \leftarrow A \setminus i$
 7:         $q_0 \leftarrow 1$
 8:     else if $ind(sign(y, 1)) \neq A$ then
 9:         $L \leftarrow ind(sign(y, 1)) \setminus A$
10:         $i \leftarrow ind(max(y, L))$
11:         $v \leftarrow invQ(I_{A^C} - GH^{-1}G^T))_{(\cdot,i)}$
12:         $A \leftarrow A \cup i$
13:         $q_0 \leftarrow -1$
14:     end if
15:     $invQ \leftarrow invQ - (q_0 + v_i)^{-1} v(invQ)_{(i,\cdot)}$
16:     $y \leftarrow y - y_i(q_0 + v_i)^{-1} v$
17: end while
18: return $A, invQ, y$
```

Algorithm 2 Solution o the QP-MPC in section 2.2

```
Require: $GH^{-1}G, S, \omega, F$
1: while MPC running do
   Obtain $x(k)$
2:     $Qinv \leftarrow I$
3:     $A \leftarrow \emptyset$
4:     $y \leftarrow (-Sx(k) - \omega)$
5:     $y \leftarrow Algorithm1(GH^{-1}G, A, invQ, y)$
6:     $u(x_k) \leftarrow -H^{-1}F^T x_k - H^{-1}G^T \phi(y)$
   Apply $u(x_k)$ to the plant
7: end while
```

2.4 Python as a programming language

Python is a language with the benefit of easily read syntax bringing comfort and efficiency to the development process [7]. However, when it comes to run-time performance, a good performance is easier to obtain in low-level programming languages like for example C. In difference from Python, C is a compiled language, meaning the whole code is converted directly to machine code and executed by the processor. Python is a interpreted language, running through the code line by line and executing each command. The execution speed is what makes the difference between this two categories, which is why many Python packages are written in the compiled language C.

The solution to the trade-off between development efficiency Python provides and the run-time efficiency from a compiled language is to use a multi-language model, where a high-level language is used with libraries as an interface to software packages written in low-level languages. Python can be seen as a part of a system of software package solutions that provides a good environment for computational work. It is also significant that the scientific computing libraries are free and open source, that makes it possible for the developer to have insight in how the packages are implemented and with what methods.

2.4.1 NumPy

NumPy, Numerical Python, is a fundamental Python package for scientific computing. The core of NumPy is implemented in C, and thus provides high efficiency for array manipulation and processing. NumPy provides vectorized computations, which eliminates the need for many explicit loops over the array elements. When code is vectorized, it means that the vecotirzed operations are executed in optimized, pre-compiled C code [10]. In general, vectorized code comes with the advantages of more concise code, fewer lines and thus fewer bugs.

In NumPy, the arithmetic operations between arrays are done by applying batch operations which enables rapid computations. When avoiding loops, the code obtains higher performance. the power of vectorization is demonstrated with simple program examples given below.

```
from timeit import default_timer as timer
import numpy as np
arr = np.ones((100,1))
arr2 = np.zeros((100,1))
arr4 = np.zeros((100,1))
start = timer()
for i in range(100):
   arr2[i] = arr[i][0]*2
   arr4[i] = arr[i][0]*4
end = timer()
timetot = end-start
print(timetot)
```

This program is written without the advantage of vectorization and takes 1.27e-04.

```
from timeit import default_timer as timer
import numpy as np
   arr = np.ones((100,1))
   arr2 = np.zeros((100,1))
   arr4 = np.zeros((100,1))
   start = timer()
   arr2 = arr*2
   arr4 = arr*4
   end = timer()
   timetot = end-start
   print(timetot)
```

This program is written by the use of vectorization and it takes 1.25e-05 seconds to execute. The non vectorized program is 10 times slower. This simple example demonstrates how avoiding loops can affect the run time and in programs with a big amount of computations the vectorization is essential for effeciency.

2.5 Analyzing and evaluating Python code

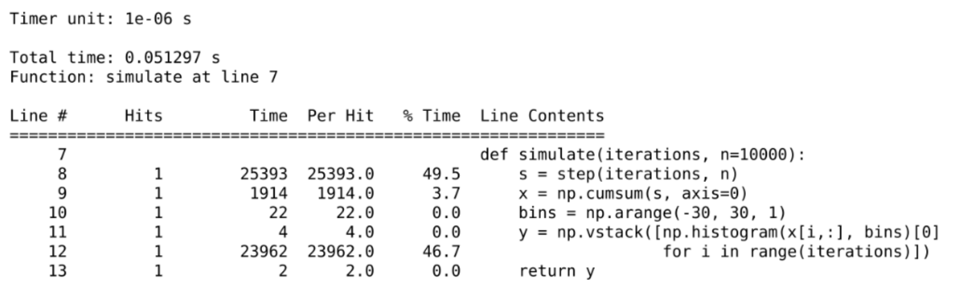
To be able to optimize code, it is necessary to have knowledge about what parts of a program that is requiring the most resources i.e. time or memory. For this, one can benefit from using some analyzing tools called Profilers [9]. A profiler creates a report containing details about functions in a program. In this section, the tool *Line Profiler* and *Memory Profiler* will be described.

2.5.1 *Line Profiler*

Line Profiler The *Line Profiler* analysis tool from the LineProfiler package is a tool to analyze the run time of code. This tool make it possible to discover the run time bottle neck and thus optimize this parts. The *Line Profiler* returns data for each line in the code in terms of Hits, Time, Per Hit, %Time and the Line Content [9].

The Hits gives the number of times this line was called while the Per Hit column gives the time of each call. The %Time represents the percentage of the total time taken by that specific line and the Line content gives the code of the line. The tool gave an overview of which lines one should attempt to optimize. An example of a terminal output after running *Line Profiler* can bee seen in Figure 2.2. The unit is μs.

Figure 2.2 The result of running *Line Profiler* on a function [9]



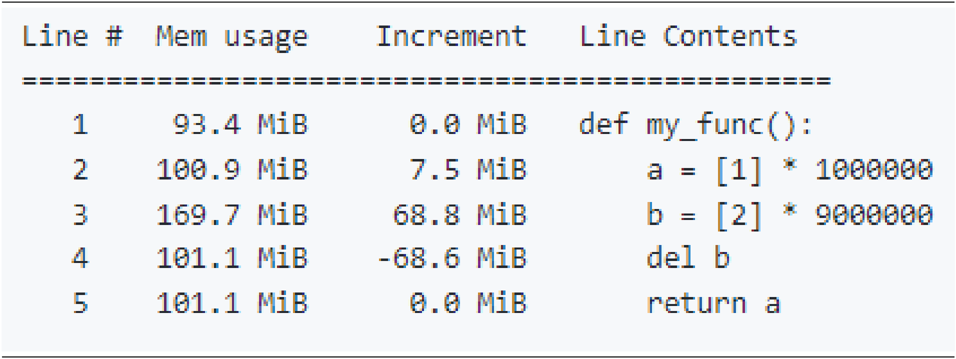
```
Timer unit: 1e-06 s

Total time: 0.051297 s
Function: simulate at line 7

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     7                                           def simulate(iterations, n=10000):
     8         1        25393  25393.0     49.5      s = step(iterations, n)
     9         1         1914   1914.0      3.7      x = np.cumsum(s, axis=0)
    10         1           22     22.0      0.0      bins = np.arange(-30, 30, 1)
    11         1            4      4.0      0.0      y = np.vstack([np.histogram(x[i,:], bins)[0]
    12         1        23962  23962.0     46.7                     for i in range(iterations)])
    13         1            2      2.0      0.0      return y
```

2.5.2 *Memory Profiler*

In similarities with the *Line Profiler* described in the previous section, the memory can also be profiled and analyzed using *Memory Profiler* from the memory_profiler package.

When running *Memory Profiler* data is returned to the terminal with Line, Mem usage, Increment, Occurrences and Line Contents. The Line column represents the line number, the Mem usage represents the memory usage of the Python interpreter after the line has been executed and the increment returns the difference in memory usage after the line is executed. An example of what is showing in the terminal after running *Memory Profiler* is shown if Figure 2.3. The unit is MiB, mebibyte, which represents 1024*1024 bytes, i.e. 2^{20} bytes.

Figure 2.3 The result of running *Memory Profiler* on a function [9]



```
Line #   Mem usage    Increment   Line Contents
================================================
     1     93.4 MiB      0.0 MiB   def my_func():
     2    100.9 MiB      7.5 MiB       a = [1] * 1000000
     3    169.7 MiB     68.8 MiB       b = [2] * 9000000
     4    101.1 MiB    -68.6 MiB       del b
     5    101.1 MiB      0.0 MiB       return a
```

Memory usage might also be a factor in writing more efficient code, as avoiding unnecessary array creations and copies can speed up the program.

2.6 Matlab as a programming language

Matlab is an interpreted language, which is easy to use [2] for scientific computations as it comes with various extension libraries with solutions to technical tasks. In other programming languages, more functions need to be made by the developer, but Matlab provides hundreds of built in functions. There are also many special case toolboxes to solve complex problems in specific fields. This make it an ideal language to do rapid prototyping of new programs. Matlab have to possibility of vectorization, which demonstrated in section 2.4.1 contribute to efficiency. Matlab is faster than Python with Numpy in most arithmetic operations [11]. However, Matlab, is not that widely used in the industry due to the high cost.

3

Methods and tools used

In this chapter, the models, methods and tools used in the project will be presented. The installation process on the computer of the different tools have been omitted. When investigating how to improve the code, the code was running only with model 2 and the results where presented. This is done with the assumption that improving the run time with model 2 will also improve the run time with model 1. This assumption was tested at the end of the improvement process and the results can be found in the next chapter.

3.1 Models used

Two different models where used to run the code and compare with Matlab. It was considered to be enough with two models, one with 2 states and one with 4 states. This will satisfy the ability to do the desired analysis of run time and comparison with the Matlab version. For this reason, the two models of different size was considered sufficient in this project.

Model 1 is the double integrator example given by the system dynamics

$$x(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 1 \\ 0.3 \end{bmatrix} u(k) \tag{3.1}$$

with constraints

$$-1 \leq u(k) \leq 1, \quad \begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x(k) \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix} \tag{3.2}$$

The prediction horizon $N = 10$ is used, the initial state is given by

$$x_0 = \begin{bmatrix} 5 \\ -2 \end{bmatrix} \tag{3.3}$$

Model 2 is a four-state system with discrete-time dynamics given by

$$\begin{aligned} x(k+1) = &\begin{bmatrix} 0.928 & 0.002 & -0.003 & -0.004 \\ 0.041 & 0.954 & 0.012 & 0.006 \\ -0.052 & -0.046 & 0.893 & -0.003 \\ -0.069 & 0.051 & 0.032 & 0.935 \end{bmatrix} x(k) \\ &+ \begin{bmatrix} 0 & 0.336 \\ 0.183 & 0.007 \\ 0.090 & -0.009 \\ 0.042 & 0.012 \end{bmatrix} u(k) \end{aligned} \tag{3.4}$$

with constraints

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq u(k) \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \end{bmatrix} \leq Cx(k) \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{3.5}$$

The prediction horizon is $N = 30$ and the initial state is given by

$$x_0 = \begin{bmatrix} 25.5724 \\ 25.3546 \\ 9.7892 \\ 0.2448 \end{bmatrix} \tag{3.6}$$

3.2 NumPy

The Numerical Python library, NumPy, was chosen to use for the implementation of the code in Python as it enables a wide range of mathematical operations on multidimensional arrays [4]. As NumPy provides the computational performance of C and the comfort and readability of Python, it was concluded a suitable library for this project, where efficient computations is desirable.

3.3 Code translation

The code was translated from Matlab to Python by the use of NumPy for the array manipulation and the arithmetic operations. The code was attempt translated as directly as possible, so that it was possible to do a comparison of the two versions. This made impact on the code quality, but the objective of the project was efficient QP solving, that is readability was neglected. The source code editor used was Visual Studio code from Microsoft and the interpreter used was Python 3.10.8 64-bit.

3.3.1 Code structure

The code where divided into separate files calling each other. In Matlab this files needs to be executed in a specific order, to set the variables needed by the other files. In Python, however, when importing a file dependency at the beginning of the file, it is only necessary to run one file, as this instruction execute the dependent files automatically. All implemented Python files can be seen in [APPENDIX].

MPC_setup.py sets all needed matrices for the chosen model, which is also set in this file. For simplicity, some of the constant matrices where read from the setup file in Matlab, so that the need to implement the function *termset* was eliminated. The file *MPC_run.py* runs the MPC algorithm and solves the QP problem for each time step. The QP problem was for each time step solved in a while loop, adding and eliminating constraints from the active set. The MPC algorithm returned an x array with all the states calculated in the MPC and a u array with all calculated inputs to the model. These arrays was filled with one element at each iteration of the for loop, i.e. at each time step.

3.3.2 The use of debugger

While implementing the code, the Python debugger in VS code was utilized. The debugger provided possibility to set breakpoints, step-through the code and observe variables. The tool was utilized to compare the code with the Matlab code, using the debugger and the *Workspace* in Matlab to compare variables.

3.3.3 Verification of the Python version

To verify that the Python algorithm gave the same results for input and output of the models as in the Matlab algorithm some plots where made. In Python, the package *matplotlib* was used, and the output trajectories *y_m* and input trajectories *u* where plotted. In Matlab, the same trajectories where plotted with the built in function *plot* and compared to the Python plots.

3.4 Measurements of run time

Both versions was decided to solve the MPC problem for 100 time-steps, thus the variable tend was set to 100. The measurements where taken in the exact same lines in each version of the code.

In Python the $default\_timer$ function from the timeit-package was used to get the current time at the beginning and end of the desired code, then the difference between the timestamps where computed.

In Matlab, the function-pair tic and toc where used to masure the elapsed time between their calls.

Both timing methods rely on wall-time, that is, they measure the time spent on the execution, in opposite to f. eks CPU time who measures the time the CPU was busy.

To ensure comparable results, the code is always run on the same computer so that the hardware does not cause differences for the software performance. Two different timers for measuring computation time was implemented in each of the versions.

1. One timer was implemented to measure the time of each step, that is, the time used to solve the QP problem and compute the next input, $u(x)$, to the system.

2. One timer was implemented to measure the run-time of the complete algorithm for all 100 steps.

These timers where elected so that it is possible to see how the solution to the QP problem evolves for each time step and although it was possible to calculate the whole run time by multiplying with the number of time steps, it was considered better to measure the whole algorithm run time. That was, to be able to analyze data more directly, without needing calculations. The results from the Python code was saved to a .txt-file and the data was read by the Matlab function $readmatrix$. This gave the possibility to plot all data with Matlab and compare results.

3.5 Run time analysis with the use of *Line Profiler*

To investigate what parts of the implemented code that consumed most of the computation time the tool $LineProfiler$ from the $line\_profiler$ Python Module was utilized.

There are several profiling tools available in Python, but after reading of the different options, the *Line Profiler* was chosen due to the data it generated and also the simplicity of use. In advantage of *cProfiler* the data generated by *Line Profiler* was considered to be more straightforward and easier to interpret.

The *LineProfiler* required a function to analyze which lead to some restructuring of the code. The code in *MPC_run.py* was put inside a **main** function which was called at the end of the file. An additional file. *lineproftest.py*, was written to call the *Line Profiler* on the **main** function. The script called to use *Line Profiler* is given below.

Algorithm 3.1: linproftest.py

```
from line_profiler import LineProfiler
from MPC_run import main

lp = LineProfiler()

main = lp(main)
main()
lp.print_stats()
```

When using LineProfiler to run the MPC_run script, the total run-time of the script increases in comparison to running the script directly. However, calling MPC_run from a different script with *LineProfiler* is still a useful tool for run-time analysis.

The strategy for optimizing the code used in this project can be summarized as the following approach in 6 steps:

1. Run *MPC_run.py* with *LineProfiler*

2. Interpret the results and search for lines with high impact on total time i.e. with a high percentage in the **% Time column**

3. Intend to restructure and improve the selected line

4. Run *MPC_run.py* with *LineProfiler* and keep the adjustment if the performance is better

5. Verify that the results are the same as expected i. e. run *plots.py* and verify the plots
6. Repeat the algorithm from step 2

The process provides a method to approach the problem, but contains no instruction on how to execute step 2 as this needs to be customized to the line content of the specific line. There is also not a limit on how many times this algorithm could be executed when attempting to improve a program. It will be a reasonable choice to prioritize to optimize the lines with a high **% Time**, but the **Hits** should also affect the prioritization. Eventually, when the run time has increased considerably it will be reasonable to move on to other optimization techniques.

3.6 Memory usage analysis with *Memory Profiler*

To analyze the memory requirements in the Python script, a *Memory Profiler* was utilized. The usage and results generated is similar to the *Line Profiler* described in the previous section which made it a natural choice after working with *Line Profiler*. In addition, as explained in 2.5.2, a memory-optimized program can contribute to better efficiency.

In *MPC_run.py*, the line @profile, was added above the defined **main** function described in the previous section. Then the file was called from the terminal with the *Memory Profiler* with **python -m memory_profiler MPC_run.py**.

4

Results and improvements

This chapter will present the results from using the methods described in the previous chapter and what actions that have been done to improve the code.

4.1 Validation of the versions

To validate that the Python translated code worked as expected, the input to the system, $u(x_k$, and the systems state trajectories x was plotted in both Matlab and Python. The Python module $matplotlib$ was used for the Python version, while the Matlab build-in function $plot$ was used for the Matlab version.

Figure 4.1 Input and output trajectories for the implemented Python translation, model 2

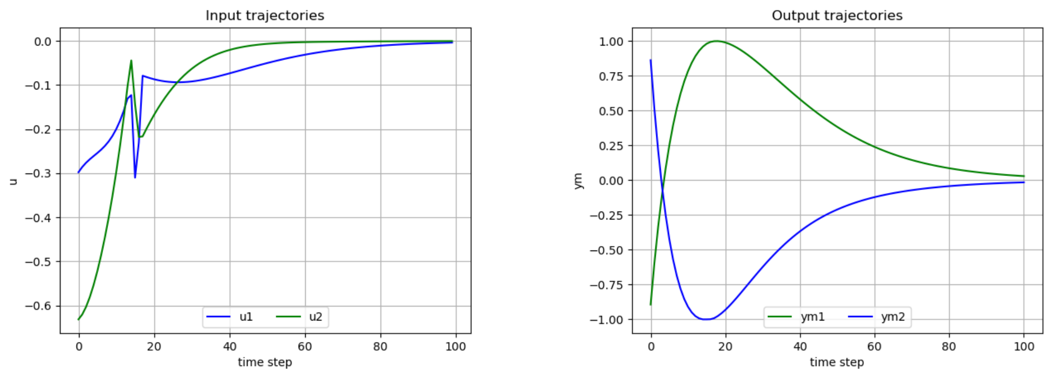
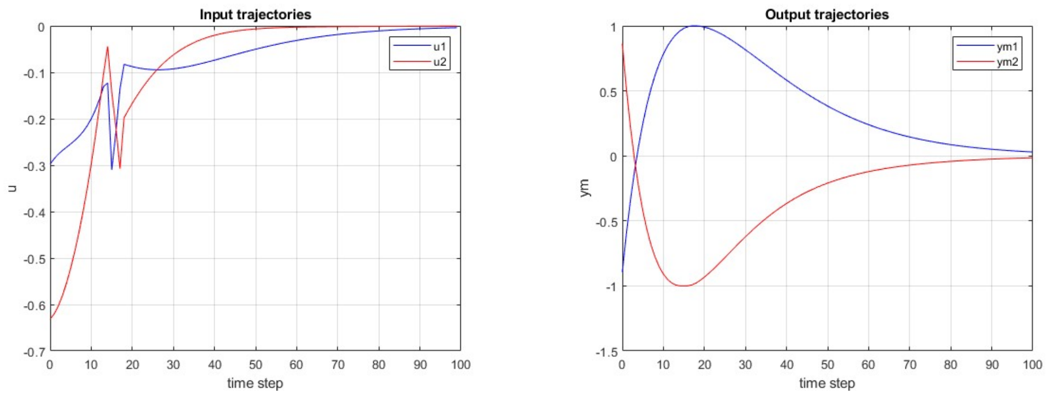


Figure 4.2 Input and output trajectories for Matlab original, model 2



As seen in the plots, the two versions plots coincides on a satisfying level, making it possible to use these results for further investigation and measurements.

The time spent by the while-loop for each step in the for-loop for the two models are shown in Figure 4.3 and in Figure 4.4. The plots where made logarithmic for better comparison between the two programming languages.

Figure 4.3 Run-time at each step in Python translation and Matlab original for model 1

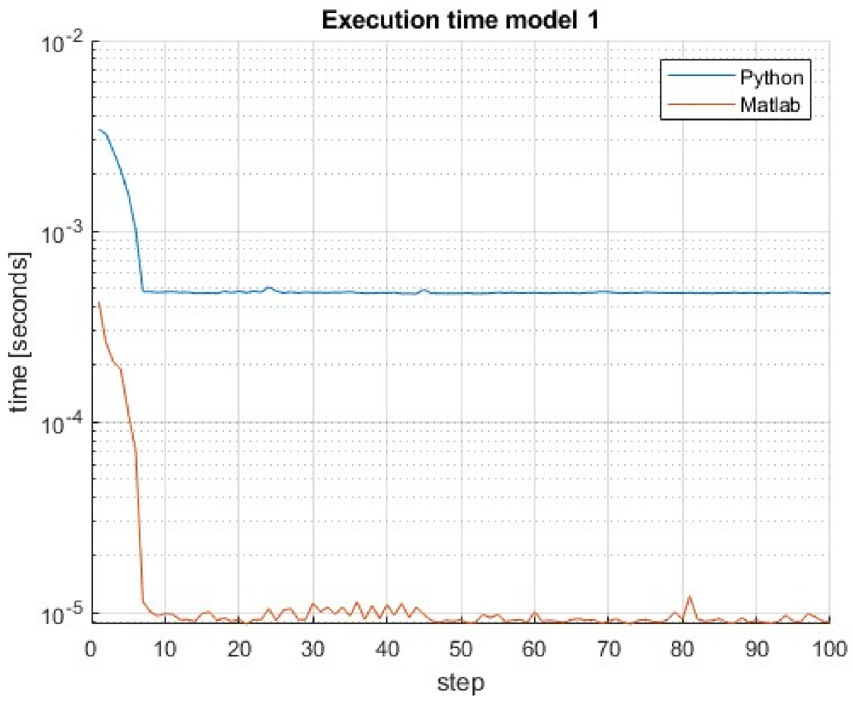
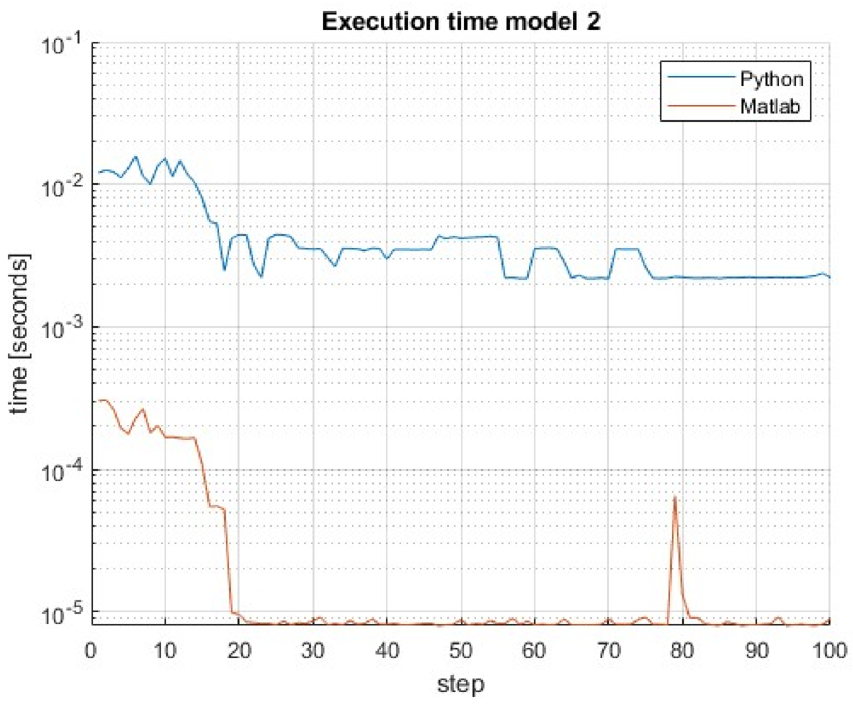
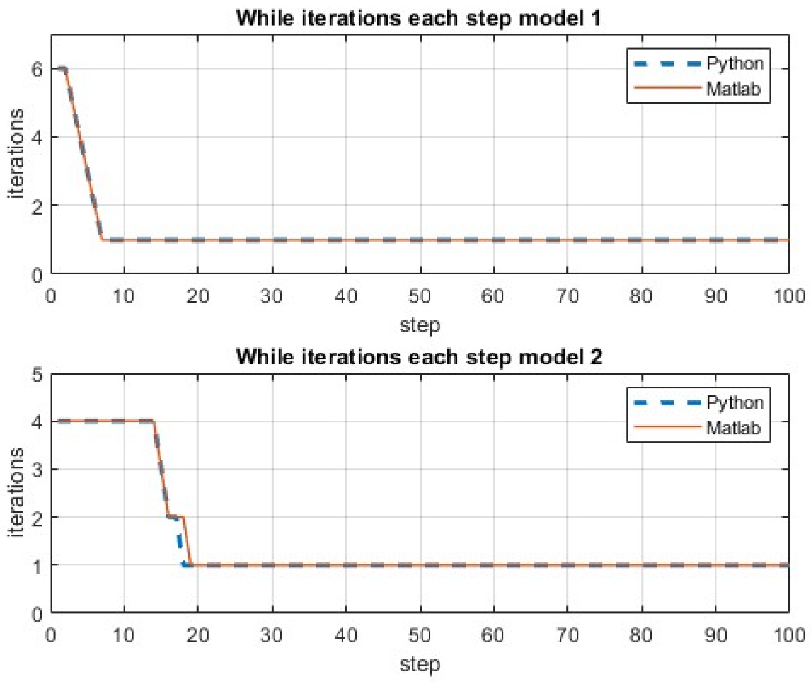


Figure 4.4 Run-time at each step in Python translation and Matlab original for model 2



In addition, the while-loops spent to compute the solution at each time step was saved for each version and plotted, this can be seen in Figure 4.5.

Figure 4.5 Number of while iterations at each time-step for model 1 and model 2



Model	n	N	max. # of iterations	steps with maximum iterations	avg. of iterations pr time step	avg. runtime for 10 runs [s]
1	2	10	6	2	1.2	0.0608775
2	4	30	4	14	1.46	0.3733584

Table 4.1: Performance of Python translation

Model	n	N	max. # of iterations	steps with maximum iterations	avg. of iterations pr time step	avg. runtime for 10 runs [s]
1	2	10	6	2	1.2	0.0019658
2	4	30	4	14	1.47	0.0045570

Table 4.2: Performance of Matlab original

4.1.1 Analysis with the use of $LineProfiler$

To investigate what parts of the implemented code that consumes most of the computation time the tool $LineProfiler$ from the $line\_profiler$ Python Module was utilized. A selection of some lines are given below in Table 4.3.

Line #	Hits	Time [μs]	Per Hit [μs]	% Time	Line Contents
38	1	1127.0	1127.0	0.0	HGz = -Hinv@np.transpose(Gz)
57	100	261.0	2.6	0.0	solved = False
72	46	18505.0	402.3	0.2	y = np.subtract((y0),(y0[iz].item())*vAd)
76	146	4894693.0	33525.3	46.7	i1 = min(lam)
87	146	4951030.0	33911.2	47.2	i2 = max(y-lam)

Table 4.3: Table with LineProfiling results

The algorithm described in section 3.5 was used to improve the Python code with the help from *Line Profiler*. The results and improvements are given in below.

4.1.2 Improvement 1

The *LineProfiler* result from the directly translated Python version showed that line 76 and 87 where very time consuming. This is where the element to remove or add to the active set, A, is determined. These lines are in addition to being slow, called with a higher frequency than the rest of the lines, causing them to have big impact on the total run-time.

After some investigation, a replacement for line 76 was found, the replacement can be seen in Table 4.4. This replacement improved line 76 by a factor of 221 and line 88 by a factor of 170. The change of these two lines decreased the average run-time of the whole script with a significant factor as seen in **??**. Another line in the code performing a division of a matrix by a constant was improved by changing *np.mulitply()* with *np.divide()* and doing some mathematical manipulation. The improvement of this was not of big impact on total time, but the **% Time** went from 1.3 to 0.9.

[flytt til diskusjonsdelen] This big improvement was possible due to how NumPy lets you combine the benefits of Python being a comfortable coding language while offering the run-time efficiency benefits of C.

Line #	Hits	Time [μs]	Per Hit [μs]	% Time	Line Contents
76	146	21832.0	149.5	0.4	i1=lam.min()
87	146	29073.0	199.1	4.1	i2 = (y-lam).max()

Table 4.4: Code improvement nr. 1

4.1.3 Improvement 2

The improvement changed the results from the *Line Profiler* as the decrease of impact on the total run time leads to other lines increasing their impact. Another line that got high impact was the line where the update of Q is calculated. The data for this line can be seen in Table 4.5.

Line #	Hits	Time [μs]	Per Hit [μs]	% Time	Line Contents
108	46	191553.0	4164.2	34.9	Qmat1i = np.subtract(Qmat0i, vAd@np.matrix(Qmat0i[iz,:]))

Table 4.5: Table with extracted LineProfiling results before improvement 1

The improvement suggested was to split this calculation in two parts, one for calculating the subtrahend and one for calculating the difference. This adjustment gave the results shown in Table 4.6.

Line #	Hits	Time [μs]	Per Hit [μs]	% Time	Line Contents
106	46	111791.0	2430.2	19.7	vAdQmat0i = vAd@np.matrix(Qmat0i[iz,:])
107	46	43618.0	948.2	7.7	Qmat1i = np.subtract(Qmat0i, vAdQmat0i)

Table 4.6: Table with extracted LineProfiling results after improvement 2

This gave a new average run time shown in Table 4.7. After this result of dividing arithmetic operations in smaller parts, some other lines where attempted divided, but with no increased performance, thus the original code was kept.

4.1.4 Improvement 3

Working with the method described in section section 3.5, another improvement was intended and succeed. The reset of the variable *Qmat0i* was done inside the for-loop with the NumPy call *np.identity(nc)* consuming 10% of the total run time. As a solution, the variable *Qreset* was mase outside the loop with the call *np.identity(nc)* which led to this line being called 1 time instead of 100. This led to a small improvement in average run time, as seen in Table 4.7

Python translation	After improvement nr. 1	After improvement nr. 2	After improvement nr. 3
0.3733584	0.0369917	0.0246870	0.0203659

Table 4.7: Average run-time of 10 runs after each improvement

5 Discussion

5.1 Limitations

5.2 Further work

6 Conclusion

Bibliography

[1] A. Bemporad, M. Morari, V.D., Pistikopoulos, E.N., 2002. The explicit linear quadratic regulator for constrained systems. Automatica.

[2] Chapman, S.J., 2015. Matlab programming for engineers. Cengage Learning.

[3] Foss, B., Heirung, T.A.N., 2016. Merging Optimization and Control. Department of Engineering Cybernetics Norwegian University of Science and Technology — NTNU.

[4] Harris, C.R., M.K.v.d.W.S.e.a., 2020. Array programming with NumPy. Springer Nature. doi:https://doi.org/10.1038/s41586-020-2649-2.

[5] Hovd, M., 2022. John Wiley Sons, Inc.

[6] Hovd, M., Valmorbida, G., 2022. Quadratic programming with ramp functions and fast QP-MPC solutions.

[7] Johansson, R., 2015. Numerical Python: A Practical Techniques Approach for Industry. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-0553-2_19, doi:10.1007/978-1-4842-0553-2_19.

[8] M. Rubagotti, P.P., Bemporad, A., 2013. Stabilizing embedded MPC with computational complexity guarantees. IEEE.

[9] Rossant, C., 2018. IPython Interactive Computing and Visualization Cookbook. Packt Publishing.

[10] unknown, 2008-2022. What is NumPy? doi:https://numpy.org/doc/stable/user/whatisnumpy.html.

[11] Weiss, A., Elsherbeni, A., 2020. Computational Performance of MATLAB and Python for Electromagnetic Applications. doi:10.23919/ACES49320.2020.9196078.

Appendix

A Python code

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut rhoncus, elit a egestas volutpat, nisi orci volutpat sem, eget eleifend lacus urna sit amet nunc. Duis ultrices, urna eu laoreet efficitur, metus lectus congue metus, aliquam suscipit ante est sed diam. Suspendisse pellentesque commodo diam, at pretium tellus consectetur commodo. Vestibulum gravida nisl augue, vel mattis risus auctor quis. Suspendisse sed aliquam nisl, maximus dapibus dolor. Maecenas vitae tortor at orci sollicitudin convallis. Etiam maximus mi ipsum, ac interdum ex semper eu. Proin condimentum finibus elit eget interdum. Praesent ut porta dolor.

Sed at erat fringilla, ornare mi ac, tincidunt velit. Curabitur ac eleifend enim. Duis justo justo, semper sed blandit et, consectetur non lacus. Suspendisse lobortis elit eros. Vestibulum eget tellus volutpat, bibendum ligula quis, venenatis orci. Nullam accumsan volutpat nibh, a commodo risus semper id. Phasellus efficitur eros vel sapien tristique dictum. Maecenas condimentum metus justo, mollis pellentesque ex elementum tincidunt. Vivamus id vehicula dui.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras a orci porttitor, luctus odio ut, viverra ex. Suspendisse erat sapien, pulvinar id est ut, lobortis condimentum quam. Mauris cursus tempus maximus. Nunc at felis id elit interdum malesuada eget sed diam. Donec lobortis nulla sed tortor accumsan, eget placerat felis tristique. Donec fringilla leo ex. Sed pulvinar congue sapien, vitae convallis turpis laoreet a. Sed vel ex ipsum. Phasellus eu magna fringilla, interdum nunc ut, imperdiet magna. Vestibulum consequat ligula ut vestibulum congue.

Vestibulum nec sapien ut turpis rhoncus lacinia a eget nisl. Nunc sodales pharetra sem a lobortis. Etiam iaculis aliquet ligula, at elementum neque accumsan sed. Aenean in dui lobortis, ornare lorem non, rutrum eros. Etiam ac velit vel orci tempor facilisis sit amet eu velit. Vivamus sed aliquet urna. Morbi varius, sapien id dictum vehicula, ante odio egestas dui, eget interdum tortor nisi sit amet sapien. Ut id vehicula nisl. Sed egestas, leo ultricies porttitor semper, neque lectus ultrices arcu, quis dignissim ex nisl placerat ipsum. Phasellus purus magna, venenatis eget enim id, eleifend facilisis lacus.

Integer maximus, nulla quis lobortis dapibus, nunc risus scelerisque leo, quis tempor justo felis ut tellus. Vivamus nec urna eu nisi rutrum rutrum ut in est. Vestibulum lorem ante, rhoncus non odio elementum, interdum facilisis enim. Praesent sed mollis magna, gravida condimentum risus. Nam placerat, sapien volutpat consectetur maximus, elit mi laoreet odio, eu euismod nunc arcu vel leo. Mauris tristique lacus vitae tellus blandit tristique. In luctus congue orci non bibendum.

Donec vehicula metus ac sem finibus tempus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Praesent ut justo rhoncus, maximus ipsum et, viverra nunc. Etiam sed molestie metus, vitae auctor magna. Nunc varius dapibus lorem, vitae aliquet nisl cursus ac. Aliquam et posuere lectus. In hac habitasse platea dictumst. Suspendisse venenatis ante sit amet tellus elementum, at molestie neque convallis. Vestibulum sodales nisi ut magna semper vestibulum. Donec fringilla et nisi pulvinar ultricies. Sed tincidunt quis leo sit amet mollis. Sed quis turpis sagittis, sagittis neque quis, auctor massa.

Praesent accumsan, purus at dictum imperdiet, augue dolor elementum ipsum, a mollis massa justo ut quam. Pellentesque cursus sagittis justo. Nullam accumsan pulvinar justo non hendrerit. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam a quam sed lectus tincidunt tempor. Donec est sem, auctor vel condimentum vel, ultrices at tellus. Quisque lacus magna, iaculis nec dui eget, vestibulum rutrum elit. Aliquam euismod massa eros, id molestie ligula rhoncus sed. Quisque nec massa vel nulla viverra rutrum. Aliquam nec ullamcorper lectus, quis egestas justo. Suspendisse ut porta tellus. Cras tortor justo, maximus nec mauris quis, congue euismod orci. Mauris id lectus laoreet, lobortis lectus at, tristique nisi. Nulla non arcu maximus nunc mollis scelerisque vel nec turpis. Nam varius dictum eros, ac iaculis nunc auctor at.

Fusce interdum ligula vitae vulputate sollicitudin. Mauris malesuada diam lacus, nec hendrerit nisl tempor eget. Etiam ullamcorper sit amet ligula vitae viverra. Nulla elit justo, molestie ac orci ac, auctor tincidunt massa. Morbi felis augue, tempor eget tellus ultricies, sollicitudin rutrum est. Proin diam lorem, sagittis sed risus nec, facilisis accumsan ante. Aliquam non efficitur sem. Nam in nisi tempor elit feugiat tincidunt ut vel sem.

Her skal jeg putte inn all koden i Python sånn den blir til slutt :)
article listings xcolor